Optimization Bootcamp

With Applications in Machine Learning, Control, and Inverse Problems

Steven L. Brunton
Department of Mechanical Engineering
University of Washington

DRAFT

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Contents

1 Optimization Overview **1**
- 1.1 Anatomy of an Optimization Problem . . . 2
- 1.2 Convexity in Optimization . . . 7
- 1.3 Gradients and the Geometry of Optimization . . . 21
- 1.4 Constraints and Lagrange Multipliers . . . 25
- 1.5 Applications: Inverse Problems, Control, and Machine Learning . . . 28

2 Gradient Based Optimization **40**
- 2.1 Gradient Descent . . . 41
- 2.2 Momentum Methods . . . 50
- 2.3 Stochastic Gradient Descent and ADAM . . . 54
- 2.4 Newton's Method . . . 63
- 2.5 Quasi-Newton Methods and BFGS . . . 67
- 2.6 Gradient Vector Calculus Identities . . . 71

3 Linear Programming **79**
- 3.1 Problem Statement and Examples . . . 80
- 3.2 Polytopes and High Dimensional Geometry . . . 84
- 3.3 Dantzig's Simplex Method . . . 87
- 3.4 Simplex Tableau Method . . . 94
- 3.5 Duality in Linear Programming . . . 98
- 3.6 Integer Programming and NP-Hard Problems . . . 100

4 Least-Squares Regression **114**
- 4.1 Least-Squares Regression . . . 116
- 4.2 Singular Value Decomposition . . . 123
- 4.3 Minimum Norm Regression . . . 127
- 4.4 Condition Number and Regularization . . . 130
- 4.5 Constrained Least-Squares . . . 135
- 4.6 Quadratic Programming . . . 136
- 4.7 Nonlinear Least-Squares: Gauss-Newton Methods . . . 141
- 4.8 The Conjugate Gradient Method . . . 146

5 Constraints and Duality **157**
- 5.1 Lagrange Multipliers . . . 159
- 5.2 Karush–Kuhn–Tucker (KKT) Conditions . . . 166

DRAFT

5.3 Duality . . . 172
5.4 Interior Point Methods . . . 185
5.5 Calculus of Variations and the Euler-Lagrange Equations . . . 190

6 Global and Nonconvex Optimization 214
6.1 Nelder-Mead Downhill Simplex . . . 215
6.2 Simulated Annealing . . . 221
6.3 Evolutionary Algorithms . . . 227
6.4 Distributed and Swarm Optimization . . . 232
6.5 Regularization, Relaxation, and Splitting . . . 236

7 Bayesian Modeling and Estimation 253
7.1 Parameter Estimation in Statistics . . . 254
7.2 Bayesian Inference . . . 261
7.3 Density Estimation and Empirical Priors . . . 269
7.4 Expectation Maximization and Gaussian Mixture Models . . . 276
7.5 Least-Squares Regression as a Statistical Estimation Problem . . . 278

8 Inverse Problems 286
8.1 Mathematical Formulation . . . 287
8.2 Classic Inverse Problems . . . 288
8.3 Inverse Problems in Medical Imaging . . . 291
8.4 Modern Computational Image Inverse Problems . . . 297
8.5 Image-Based Inversion: With Physics . . . 301
8.6 PDE-Constrained Inversion . . . 305
8.7 System Identification . . . 309

9 Optimization in Control 324
9.1 Linear Optimal Control . . . 327
9.2 Nonlinear Optimal Control . . . 335
9.3 Reinforcement Learning . . . 338
9.4 Dynamic Programming . . . 342
9.5 Model Predictive Control . . . 345
9.6 Linear Matrix Inequalities . . . 350

10 Optimization in Machine Learning 364
10.0.1 Automatic Differentiation . . . 364
10.0.2 Backpropagation . . . 364
10.0.3 Neural Network Training . . . 364
10.0.4 Deep Reinforcement Learning . . . 364
10.0.5 Physics Constrained Machine Learning . . . 364

References 366

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 1

Optimization Overview

This chapter provides an introductory overview of key concepts in optimization, as well as several motivating applications. Although this chapter is not meant to be entirely standalone, it will serve as the bulk of an initial *bootcamp* course, augmented with selected topics from later chapters. Many of the sections here will be explored in much more detail in later chapters, and so this is mostly meant to provide a rapid overview and categorization of the major topics in optimization, important algorithms, and how the various ingredients fit together.

We begin this chapter with a brief categorization of different optimization problems, in the so-called *anatomy* of an optimization problem. We introduce the notions of an objective function, to be maximized or minimized, and constraint equations that limit the set of feasible solutions.

Next we introduce two of the main concepts in optimization: convexity and the gradient. Convexity provides the most fundamental dichotomy in optimization, where convex problems are essentially solved, at least numerically, and where local optimal solutions are also globally optimal. Chapters 3 and 4 on linear programming and quadratic programming (e.g., least-squares regression) cover two broad classes of convex optimization problems. Non-convex problems are much more challenging, and do not generically have solutions that scale computationally. Chapter 6 will explore non-convex optimization problems in more detail, especially modern approaches that relax non-convex problems to nearby convex optimization problems.

The gradient is another fundamental concept in optimization, as it provides critical information about the local geometry of the optimization landscape, and which direction (locally) improves the solution. This topic will be explored in much greater depth in Chapter 2, including gradient descent, Newton's method, and modern extensions for high-dimensional systems, such as stochastic gradient descent in machine learning.

This chapter will also provide simple examples of unconstrained and constrained optimization, to remind the reader of basic properties and approaches. These will be simple low-dimensional optimization problems, where it is possible to solve for the optimum solution using basic calculus. We will also briefly introduce the notion of Lagrange multipliers, which are an intuitive and physical way of handling constraint equations. Lagrange multipliers are closely related to the notion of a dual optimization problem, and both concepts will be developed further in Chapter 5.

We conclude this chapter with a brief discussion about three major application areas

DRAFT

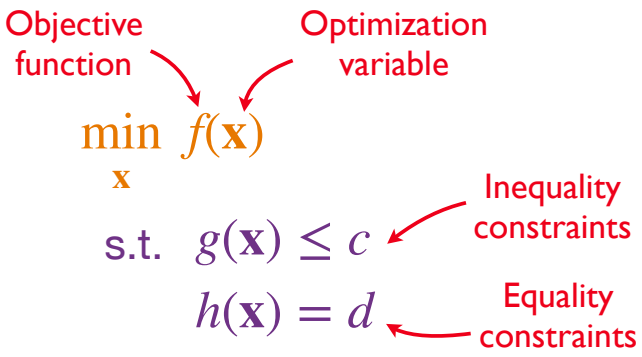


Figure 1.1: The anatomy of an optimization problem. The two main ingredients are the objective function f and the constraint equations g and h. The decision variable $\mathbf{x}$ represent the free variables to optimize over.

of modern optimization: inverse problems, control theory, and machine learning. Each of these topics will be their own chapter in the later half of this book, with an additional chapter on Bayesian optimization to complement the topics of inverse problems and machine learning.

1.1 Anatomy of an Optimization Problem

In the simplest form, an optimization problem involves minimizing or maximizing an objective function $f(\mathbf{x})$ with respect to a set of decision variables $\mathbf{x} \in \mathbb{R}^n$:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}). \tag{1.1}$$

This is an *unconstrained* optimization problem, as it is assumed that the decision variable may take any value. For scalar functions of a scalar variable, we have experience solving these optimization problems by finding points where the derivative of f with respect to x is equal to zero that also have a positive second derivative.

However, it is often the case that we have constraints on $\mathbf{x}$, which are written either as inequality or equality constraint equations. In general, a *constrained* optimization problem over a variable $\mathbf{x} \in \mathbb{R}^n$ may be written in the following *standard* form:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}), && && \text{(1.2a)} \\ \text{Subject to} \quad & g_i(\mathbf{x}) \leq 0, && i = 1, \ldots, m, && \text{(1.2b)} \\ & h_j(\mathbf{x}) = 0, && j = 1, \ldots, p, && \text{(1.2c)} \end{aligned}$$

where $f(\mathbf{x})$ is the objective function to be minimized, and $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$ are inequality and equality constraint equations that restrict the set of feasible $\mathbf{x}$ to search over. Typically the objective function f is a scalar function over a vector input $\mathbf{x}$, although in multi-objective optimization the objective function will have multiple components and will be written as a vector function $\mathbf{f}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We will often write the constraint equations in vector form as:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}), \tag{1.3a}$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{x}) \preccurlyeq \mathbf{0} \tag{1.3b}$$

$$\mathbf{h}(\mathbf{x}) = \mathbf{0} \tag{1.3c}$$

where the symbol $\preccurlyeq$ indicates that each row of the equation must be an inequality. In this text, we will often simply use the $\leq$ symbol, as it should be clear that this holds row-wise for matrix inequalities. Figure 1.1 shows this schematically with labels on the variables and functions, color coded with the colors used throughout this chapter.

The inequality and equality constraints determine a feasible set $\boldsymbol{\mathcal{S}}$ of admissible values of $\mathbf{x}$ over which to optimize the objective function f.

When the objective function f and the feasible set $\boldsymbol{\mathcal{S}}$ are both convex, then Eq. (1.2) is a *convex* optimization problem, which comes with strong theoretical guarantees and efficient computational algorithms. Linear programming and quadratic programming (e.g., least-squares regression) are two of the largest and most important classes of convex optimization problems. The most important result is that a local minimum of a convex optimization problem is also a global minimum. If the problem is strictly convex, then this minimum is unique. Convex sets and functions will be discussed later in this chapter, with several examples.

We generally consider minimization problems in standard form, as there are strong theoretical and computational guarantees for minimization problems where the objective function f and the feasible set $\boldsymbol{\mathcal{S}}$ are both convex. However, maximization is a trivial extension with similar theoretical guarantees when f is concave and the feasible set is convex. This is still considered a *convex* optimization problem, as maximizing a concave function f is equivalent to minimizing the convex function $-f$. For the special case of a linear program, where linear objective function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ is both convex and concave, we may pose minimization or maximization problems.

The Objective Function. We will often solve unconstrained optimization problems, in which case we are only concerned with the properties of the objective function f. In either the constrained or unconstrained cases, there are several important properties of f that determine which optimization algorithm to employ:

- Is f known or unknown?

  In many modern applications, there is not a mathematical model for f, and it must be computed by interacting with a simulation or experiment. Increasingly, we may be able to construct data-driven surrogate models for f that can accelerate the optimization process.

- Can we compute the gradient ∇f?

  Sometimes we have an analytic expression for the gradient of f, as in least-squares regression (see Chapter 4). In other cases, we must approximate the gradient numerically, for example using finite difference derivatives or stochastic gradient descent.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- Is f high dimensional?

  Many modern problems involve very high-dimensional decision variables $\mathbf{x}$, so that gradients and Hessians of f are high-dimensional. For example, in deep learning, it is not uncommon to train a model with millions or billions of free parameters to be optimized over. Again, this motivates a certain class of algorithm, such as stochastic gradient descent.

- Is f convex or nonconvex?

  This is the biggest dichotomy of optimization problems

- Is f linear or nonlinear?

  If f is linear, as in linear programming (see Chapter 3) then the objective function is both convex and concave. It is possible to compute the gradient of a linear objective function analytically, and the maximum or minimum of the function will depend entirely on the geometry of the feasible set given by the constraint equations.

- Is f expensive to compute or evaluate?

  In many engineering applications, such as aerodynamic shape optimization of a wing, it is expensive to evaluate the objective function for a single $\mathbf{x}$. This may involve a computationally intensive, high-fidelity simulation, or actually fabricating and testing the wing in a wind tunnel. When function evaluation is expensive, we require algorithms that converge in very few iterations, which often motivate surrogate models.

- Does f encode dynamics in time?

  When the system being optimized involves time dynamics, this often changes the optimization algorithm and computational cost involved. Control theory (see Chapter 8) is a good example where we are optimizing a controller to modify the performance of a dynamical system, which inherently varies in time. Other optimization problems, such as the aerodynamic shape optimization problem, may involve time averaging dynamic effects, such as turbulent fluctuations.

The Constraint Equations. Constraint equations arise naturally in optimization problems. Any time a decision variable involves a constrained resource, such as time, space, money, materials, labor, etc., these may be expressed in terms of constraint equations. This is common in linear programming, which involves minimizing or maximizing an objective function under linearly constrained resources.

We will see many other types of constraints throughout this book. For example, we may want to solve a least-squares regression problem where we know that some variables must be non-negative, or within some radius (see Fig. 1.2). Control theory is another example, where the constraints are governed by the dynamics of the system.

Figure 1.2 shows two examples of feasible sets given by inequality constraints. The first example is a linear inequality constraint, and the second example is a nonlinear inequality constraint. Any linear inequality constraint, given by $g(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \leq 0$ determines a feasible half-plane in the ambient n-dimensional space. The complementary inequality $\mathbf{c}^T\mathbf{x} > 0$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

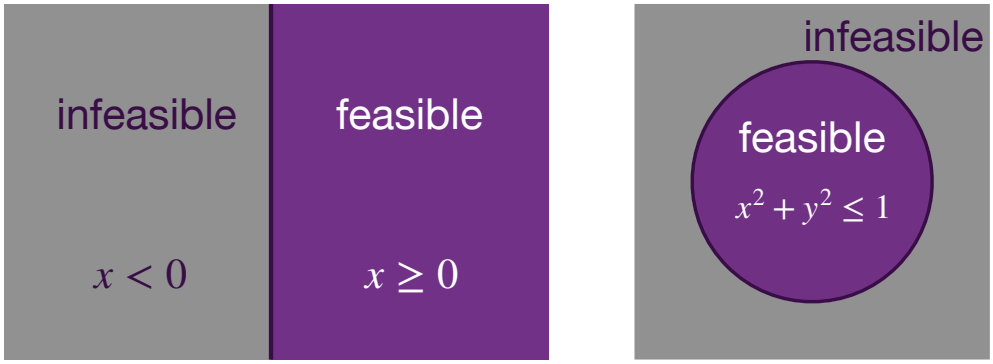


Figure 1.2: Examples of feasible and infeasible sets determined by inequalities. For a simple linear inequality, the regions will define convex half-spaces. For nonlinear inequalities, as shown on the right, the regions may be more geometrically interesting. In this case, the feasible region is still convex.

determines an infeasible half plane. The nonlinear constraint $g(\mathbf{x}) = x^2 + y^2 - 1$ determines a feasible circular disk of radius equal to 1, with an infeasible region exterior to this disk. In both cases, the feasible region is convex, as we will see defined formally soon, because both constraint functions are convex functions.

The Feasibility Problem. In nearly every domain using optimization—engineering design with physical constraints, finance with regulatory constraints, or machine learning with fairness or interpretability constraints—maximizing or minimizing an objective is meaningless if we cannot locate any feasible solutions. Thus, verifying feasibility is a foundational step before refining or optimizing a solution.

For high-dimensional problems with complicated constraint equations, even finding a feasible $\mathbf{x}$ might be a challenge. The *feasibility problem* is

$$\begin{aligned} \text{find} \quad & \mathbf{x}, && \text{(1.4a)} \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} && \text{(1.4b)} \\ & \mathbf{h}(\mathbf{x}) = \mathbf{0}. && \text{(1.4c)} \end{aligned}$$

Ensuring feasibility (i.e., satisfying all constraints) can be at least as challenging as optimizing an objective. One might have to solve large sets of equations or inequalities to find a point that meets all physical, geometric, or statistical constraints, before even addressing optimality. We will see an example of a feasible set obtained via the intersection of several inequality constraints later, in Fig. 1.6, in the context of linear programming.

Feasibility is challenging because complex or nonlinear constraints define potentially complicated solution sets, and high-dimensional constraints can intersect in complex ways, causing narrow or fragmented feasible sets, leading to degeneracy. Sometimes it is unknown if a feasible region exists at all. In some inverse or PDE-constrained setups, even minor data mismatch can dramatically change the nature of the feasible set, making it ill posed. For example, in inverse problems one must find parameters consistent with observed data, enforcing model constraints (e.g., PDEs or boundary conditions). A naive

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

guess for the parameters might not satisfy the forward model, rendering the solution infeasible. Similarly, in control theory, a controller must often require satisfying closed-loop stability constraints, performance bounds, or input saturations. Ensuring a feasible controller even exists can be nontrivial, especially if the system is near the edge of controllability.

There are several strategies to approach the feasibility problem. For linear or convex constraints, there are specialized approaches, such as linear matrix inequality solvers, that can systematically search for feasible solutions. For more complex constraints, we may start with a simpler version of the constraints and continuously transition to the more challenging constraints, using homotopy or continuation to track the solutions through this transition. Other approaches involve penalty or barrier methods that transform constraints into large penalty terms in the objective, promoting feasible solutions. Projection and iterative methods, such as Douglas–Rachford methods [121], may also be used to project guesses onto a feasible set; these are often used in PDE constrained or geometric contexts.

Linear Programming (LP). Linear programming involves optimizing a linear objective function subject to linear equality and inequality constraints.

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x}, && \text{(1.5a)} \\ \text{Subject to:} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, && \text{(1.5b)} \\ & \mathbf{x} \geq \mathbf{0}, && \text{(1.5c)} \end{aligned}$$

where $\mathbf{c} \in \mathbb{R}^n$ is the cost vector, $\mathbf{A} \in \mathbb{R}^{m \times n}$, and $\mathbf{b} \in \mathbb{R}^m$ define the constraints.

The inequality constraints determine a convex feasible region $\boldsymbol{\mathcal{S}}$, called a polytope, that generalizes the notion of a polygon to higher dimensions. The objective function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ is also convex, making this a convex optimization problem.

Slack Variable Form. It is possible to write the linear inequality constraints from linear programming in slack variable notation by introducing a set of slack variable $\mathbf{s} \in \mathbb{R}^m$, with as many dimensions as there are constraint equations:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x}, && \text{(1.6a)} \\ \text{Subject to:} \quad & \mathbf{x}, \mathbf{s} \geq \mathbf{0}, && \text{(1.6b)} \\ & \mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b}. && \text{(1.6c)} \end{aligned}$$

It is easy to see that this is equivalent to Eq. (1.5), since $\mathbf{s} = \mathbf{b} - \mathbf{A}\mathbf{x} \geq 0$. Each of the components s_i of $\mathbf{s}$ quantify how close $\mathbf{x}$ is to the ith inequality boundary. This will be very useful in the simplex algorithm, which optimizes a linear program by walking from vertex to vertex of the feasible polytope, until a local optimum value is found.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Quadratic Programming (QP). Quadratic programming optimizes a quadratic objective function subject to linear constraints:

$$\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x}, \tag{1.7a}$$
$$\text{Subject to:} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \tag{1.7b}$$
$$\mathbf{x} \geq \mathbf{0}. \tag{1.7c}$$

Quadratic programs are widely used in finance and control theory, and they are the basis of least-squares regression in data analysis and machine learning. When the matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is a symmetric positive semi-definite matrix, then the quadratic program is *convex*.

Box Constraints. It is common to have constraints where the components of $\mathbf{x}$ must be between a lower and an upper bound, $\mathbf{l}$ and $\mathbf{u}$, respectively. These are known as *box* constraints:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}), \tag{1.8a}$$
$$\text{s.t.} \quad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{1.8b}$$

It is possible to rewrite these constraints in standard form as:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}), \tag{1.9a}$$
$$\text{s.t.} \quad \mathbf{l} - \mathbf{x} \leq \mathbf{0} \tag{1.9b}$$
$$\mathbf{x} - \mathbf{u} \leq \mathbf{0}. \tag{1.9c}$$

Equality Constraints. Equality constraints $\mathbf{h}(\mathbf{x}) = 0$ can be rewritten as two inequality constraints:

$$\mathbf{h}(\mathbf{x}) \leq 0 \tag{1.10a}$$
$$\mathbf{h}(\mathbf{x}) \geq 0 \tag{1.10b}$$

or equivalently as

$$\mathbf{h}(\mathbf{x}) \leq 0 \tag{1.11a}$$
$$-\mathbf{h}(\mathbf{x}) \leq 0. \tag{1.11b}$$

Although this is a useful concept, algorithmically this is poorly conditioned and it is not often used in practice. Note that an equality constraint can only be convex if both $\mathbf{h}(\mathbf{x})$ and $-\mathbf{h}(\mathbf{x})$ are convex, which is only true if $\mathbf{h}$ is an affine function of $\mathbf{x}$: $\mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$.

1.2 Convexity in Optimization

Convexity is one of the most important concepts in modern optimization [48]. The categorization of optimization problems into convex and non-convex is as fundamental as the categorization of differential equations into linear and nonlinear, with convex optimization

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

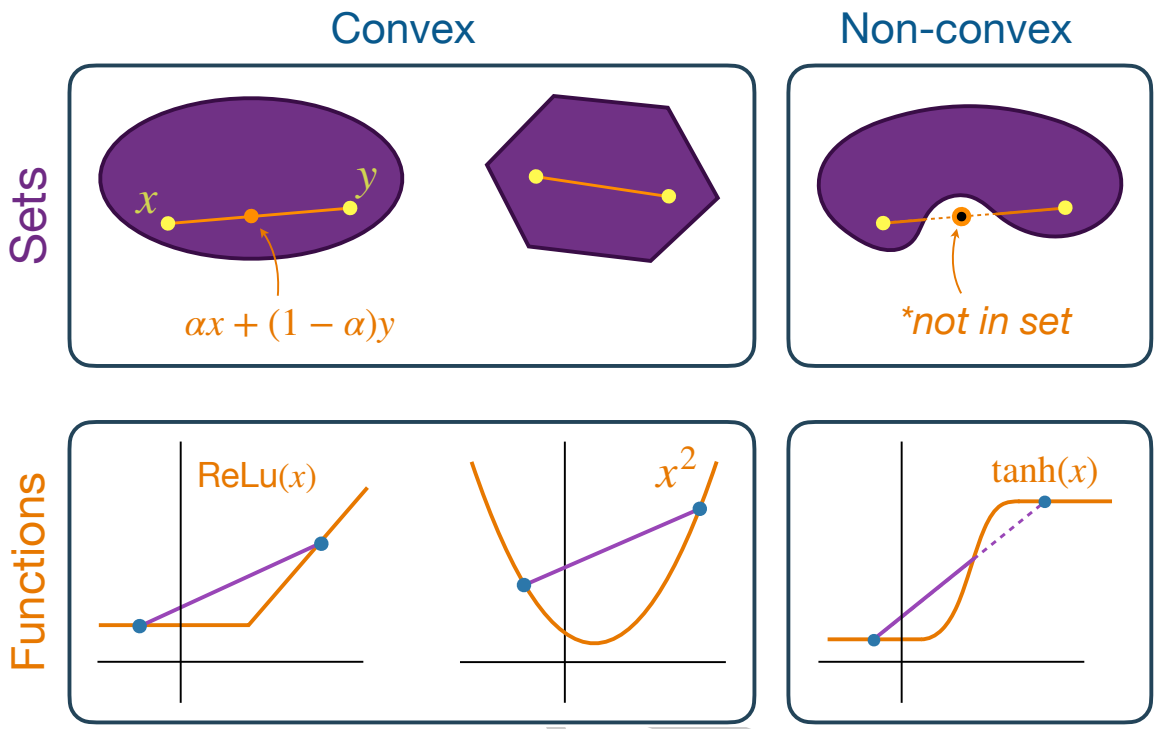


Figure 1.3: Examples of convex and non-convex sets (top) and convex and non-convex functions (bottom). Points

problems being largely solved, at least algorithmically, with robust and scalable numerical solutions.

Convex optimization problems are problems where the objective function is convex and the feasible set determined by the constraint equations are also convex [48, 264, 265]. Generically, there are fast, scalable approaches to solving convex optimization problems. Many of these problems, such as classic least-squares regression (see Chapter 4), have closed form, analytic solutions. Other problems, such as linear programming (see Chapter 3), have mature algorithms with favorable computational scaling properties.

Moreover, in a convex optimization problem, any local optimal solution will also be a globally optimal solution. This is a particularly important property in high-dimensional optimization problems, where searching through the optimization parameter space for global solutions may be intractable. In contrast, in convex optimization, if an algorithm converges to a local minima, we have confidence that this is the global minima as well.

In this section, we will introduce key notions of convex sets and functions, along with some fundamental theoretical results. Examples of convex sets and function, as well as non-convex sets and functions, are shown in Fig. 1.3.

DRA

Copyright © July 15, 2025 Brunton. All Rights Reserved.

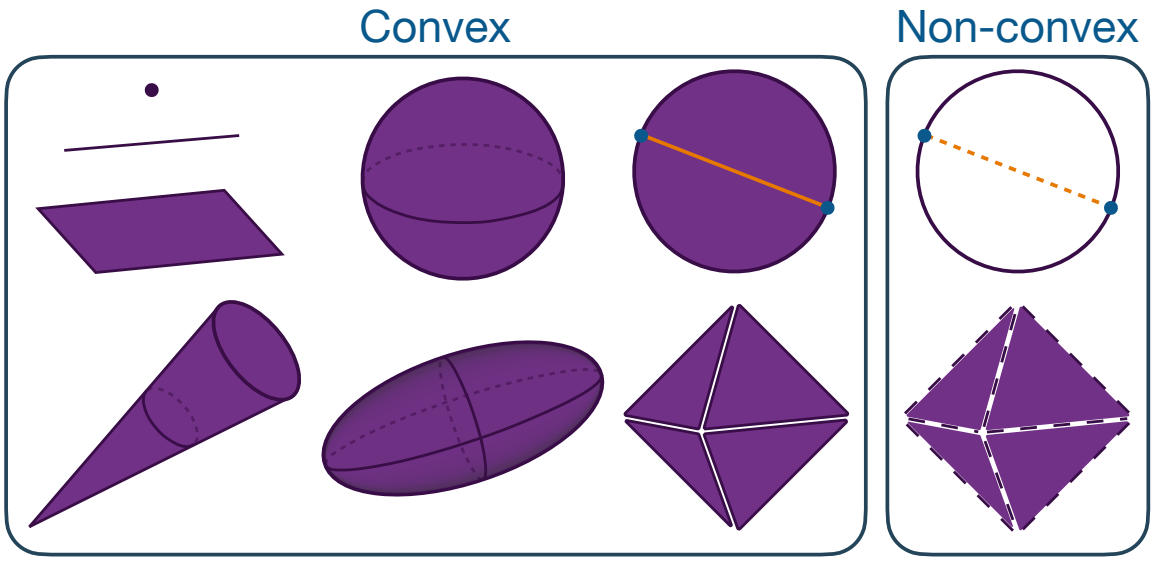


Figure 1.4: Several examples of convex and non-convex sets. Points, lines and line segments, planes and half-planes, spaces and half-spaces (called *slabs*) are all convex. A filled disk or higher-dimensional ball (e.g., a filled sphere), and filled ellipsoids are convex. Polytopes defined by the intersection of multiple inequality constraints generalize polygons to higher dimensions and are also convex. A hollow sphere or a circle without its interior is not a convex set, because the lines connecting points on the boundary include interior points that are not in the set; likewise, a geometric object that has part but not all of its boundary may be non-convex.

Convex Sets

A set $\mathcal{C} \subseteq \mathbb{R}^n$ is convex if and only if the line connecting any two points $\mathbf{x}, \mathbf{y} \in \mathcal{C}$ is also entirely in $\mathcal{C}$:

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in \mathcal{C} \quad \text{for} \quad \alpha \in [0,1]. \tag{1.12}$$

The parameter $\alpha \in [0,1]$ defines the line segment connecting the two points $\mathbf{x}$ and $\mathbf{y}$ in the convex set $\mathcal{C}$; $\alpha = 0$ corresponds to the point $\mathbf{y}$, $\alpha = 1$ corresponds to the point $\mathbf{x}$, and $\alpha \in [0,1]$ sweeps out the line segment between $\mathbf{x}$ and $\mathbf{y}$. This is shown schematically in the top row of Fig. 1.3. If any of these points $\alpha\mathbf{x} + (1-\alpha)\mathbf{y}$ are not also in $\mathcal{C}$, for any $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, then the set is not convex. In this book, we will assume that convex sets are defined in the context of a vector space, although there are more general notions.

Intuitively, a set is convex if there is a continuous *line of site* between any two points in the set, so that there are no disconnected regions, holes, or fingers. The presence of multiple disconnected regions[1] or fingers would become extremely difficult to search over, especially in high-dimensional spaces, where the number of such edge cases may grow exponentially.

Additional examples of convex and non-convex sets are provided in Fig. 1.4. There are broad classes of convex sets, including points, lines and line segments, planes and

[1] Disconnected regions and holes are a subject of *topology*.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

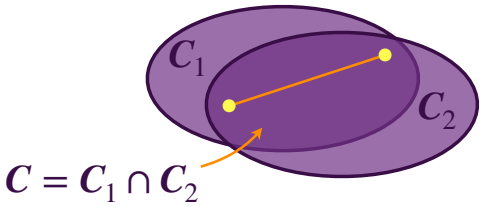


Figure 1.5: The intersection of convex sets is also convex.

half-planes, spaces and half-spaces (sometimes referred to as *slabs*). In addition, several geometric shapes also define convex sets. For example, a disk (i.e., a filled in circle with its boundary) and a higher-dimensional ball

$$B_r = \{\mathbf{x} \,|\, \|\mathbf{x}\|^2 \leq r\}$$

(i.e., a filled in sphere) also define convex sets. A hollow sphere or a circle without its interior is not a convex set, because the lines connecting points on the boundary include interior points that are not in the set. Similarly, filled ellipsoids are convex sets. Filled cones, where the cross-section is convex, are also convex sets.

Intersection of Convex Sets. The intersection of two convex sets is also convex, as illustrated in Fig. 1.5. This is simple to prove mathematically. Although we won't typically prove results in this book, it is illustrative to see this simple proof:

Proof: Denote the two convex sets $\mathcal{C}_1$ and $\mathcal{C}_2$, and let their intersection be

$$\mathcal{C} = \mathcal{C}_1 \cap \mathcal{C}_2.$$

If $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, then $\mathbf{x}, \mathbf{y} \in \mathcal{C}_1$ and $\mathbf{x}, \mathbf{y} \in \mathcal{C}_2$. Since $\mathcal{C}_1$ is convex, then

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in \mathcal{C}_1$$

for $\alpha \in [0, 1]$. The same is true for $\mathcal{C}_2$. Thus,

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in \mathcal{C}_1 \cap \mathcal{C}_2 = \mathcal{C}$$

for $\alpha \in [0, 1]$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, which proves the result. ■

The intersection of several linear inequality constraints, $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, define a convex polytope, which generalize polyhedra to higher-dimensional spaces. An example of such a convex polytope is visualized in Fig. 1.6. These sets are *pointy*, with the bounding surfaces, edges, and vertices being characterized by subsets of the inequality constraints. These convex polytopes will be explored extensively in the context of linear programming in Chapter 3.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

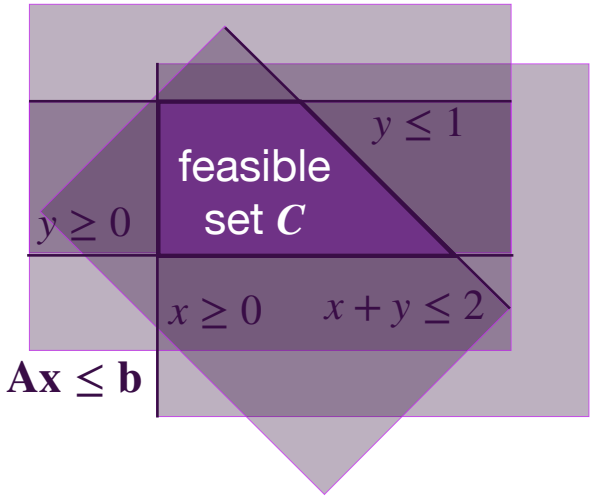


Figure 1.6: Convex polytopes are obtained by the intersection of several linear inequality constraints. Convex polytopes play an important role in several classes of optimization problem, including linear programming (see Chapter 3).

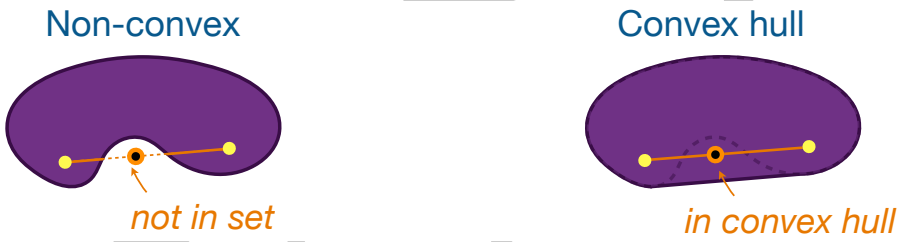


Figure 1.7: Illustration of the convex hull, $hull(\boldsymbol{\mathcal{S}})$, of a non-convex set $\boldsymbol{\mathcal{S}}$.

Convex Hull. It is possible to *convexify* a non-convex set $\boldsymbol{\mathcal{C}}$ by systematically adding the points that violate the convexity condition. The smallest convex set containing a set $\boldsymbol{\mathcal{C}}$ is known as the *convex hull*, and is visualized in Fig. 1.7. Mathematically, this is written as $hull(\boldsymbol{\mathcal{C}})$.

The convex hull of a convex set $\boldsymbol{\mathcal{C}}$ is trivially itself. Further, the convex hull of any set $\boldsymbol{\mathcal{C}}$ is the intersection of every convex set containing $\boldsymbol{\mathcal{C}}$, hence making it the smallest convex set containing $\boldsymbol{\mathcal{C}}$.

Affine Sets and Transformations

Affine sets satisfy an even more restrictive condition than convex sets. An affine set is any set in which every line passing through two points of the set lies *completely* within that set. Formally, a set $\mathcal{A}$ is affine if, for any $\mathbf{x}, \mathbf{y} \in \mathcal{A}$ the point $\alpha\mathbf{x} + (1-\alpha)\mathbf{y}$ lies in $\mathcal{A}$ for any real α. In contrast, a *convex* set requires $\alpha\mathbf{x} + (1-\alpha)\mathbf{y}$ to remain in the set *only* for $\alpha \in [0, 1]$. Thus, for a convex set the line segment connecting any two points $\mathbf{x}, \mathbf{y}$ in the set must also be in the set, while for an affine set, the entire infinite line connecting any two points $\mathbf{x}, \mathbf{y}$ in the set must also be in the set.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

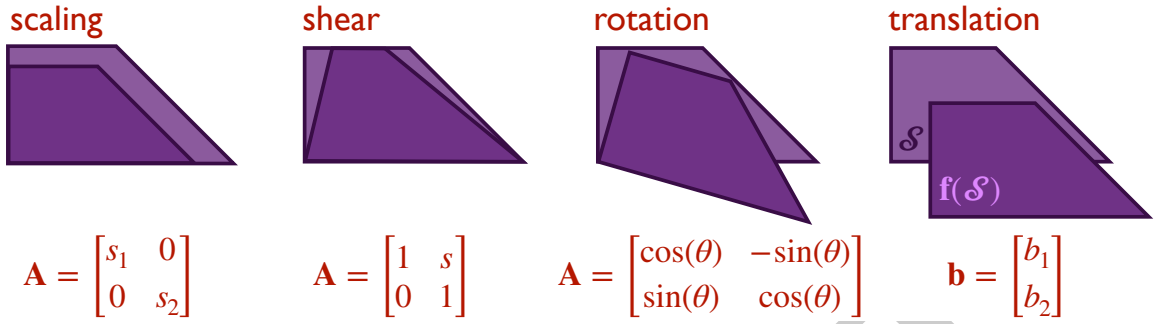


Figure 1.8: Examples of various affine transformations $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$. Scaling, shear, rotation, translation, and various combinations thereof, are all affine transformations. Affine transformations preserve convexity.

Affine sets are automatically convex, because if a set is closed under *all* real combinations $\alpha\mathbf{x} + (1-\alpha)\mathbf{y}$ for any scalar α then in particular it is closed under such combinations for $\alpha \in [0, 1]$. A simple example of an affine set is a line $\{\mathbf{x}_0 + t\mathbf{v} : t \in \mathbb{R}\}$. More generally, a plane or hyperplane is affine.

An affine set can be viewed as a "translated" version of a vector subspace. This condition means that $\mathcal{A}$ behaves like a vector space *except* it need not contain the origin, although it may. Thus, an affine set is essentially a vector space that may be shifted or translated by some fixed offset. Affine sets generalize linear vector subspaces by allowing for an arbitrary translation in space. All vector spaces are affine sets, but the converse is not true.

Affine sets arise naturally in optimization when translating or shifting feasible regions. For instance, solving linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ defines an affine subspace. Many linear and affine constraints preserve affine structure, and problems that reduce to finding a point in an affine subspace are simpler than those restricted to convex but curved sets. Hence, in optimization, one often solves feasibility in an affine set, such as $\mathbf{A}\mathbf{x} = \mathbf{b}$, and then superimposes convex inequalities to further restrict that region. Convex sets remain convex under affine transformations, as shown in Fig. 1.8. This is an exercise for the reader.

Dual Spaces and Supporting Hyperplanes

Another important concept in optimization is the notion of dual spaces and supporting hyperplanes. A supporting hyperplane to a set $\mathcal{C} \subset \mathbb{R}^n$ at a boundary point $\mathbf{x}_0$ is a hyperplane

$$\{\mathbf{x} : \mathbf{y}^\top(\mathbf{x} - \mathbf{x}_0) = 0\}$$

such that $\mathcal{C}$ lies entirely in one closed half-space determined by that hyperplane. Equivalently, the hyperplane touches $\mathcal{C}$ at $\mathbf{x}_0$ but does not cut through its interior.

If $\mathcal{C} \subset \mathbb{R}^n$ is a convex set, one may define its *dual* set in a dual vector space. Concretely, for a bounded convex body containing the origin, the dual set $\mathcal{C}^*$ is given by

$$\mathcal{C}^* = \left\{ \mathbf{y} \in \mathbb{R}^n : \mathbf{y}^\top \mathbf{x} \leq 1 \ \text{ for all } \mathbf{x} \in \mathcal{C} \right\}.$$

Geometrically, each supporting hyperplane to $\mathcal{C}$ is described by an equation $\mathbf{y}^\top\mathbf{x} = \alpha$, where $\mathbf{y}$ is a normal. Hence, specifying all such $\mathbf{y}$ and offsets α that touch $\mathcal{C}$ at least at one

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

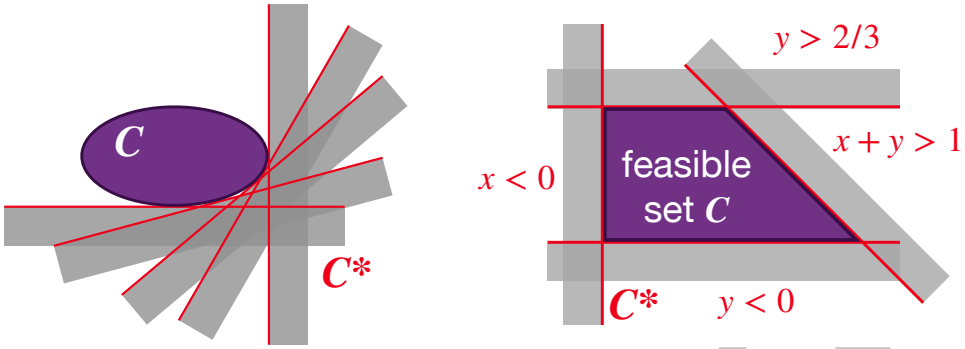


Figure 1.9: For every convex set $\mathcal{C}$, there is a dual space $\mathcal{C}^*$ that is defined by the union of all *supporting hyperplanes*, namely the tangent planes of infeasible regions that bound the convex set.

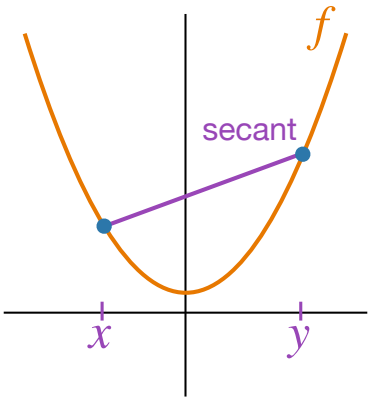


Figure 1.10: A convex function has the property that every secant line connecting any two points $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ on the function will not be below the function between $\mathbf{x}$ and $\mathbf{y}$. A strictly convex function will have the secant line strictly above the function.

point effectively characterizes its boundary. One can think of the *dual space* as collecting the normal directions $\mathbf{y}$ that define half-spaces containing $\mathcal{C}$. While it is not simply the union of all such hyperplanes, the dual set or polar set captures precisely those $\mathbf{y}$ whose supporting hyperplanes do not exclude $\mathcal{C}$. Thus, the dual space perspective encodes the convex set's boundary geometry via its family of supporting planes and their associated normal vectors. This is visualized geometrically in Fig. 1.9.

Convex Functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if the line connecting any two points on the function lies above the function:

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}). \tag{1.13}$$

This concept is illustrated graphically in Fig. 1.10. The line connecting the two points $\mathbf{x}$ and $\mathbf{y}$ is called the *secant*. Several examples of other convex and non-convex functions are

Copyright © July 15, 2025 Brunton. All Rights Reserved.

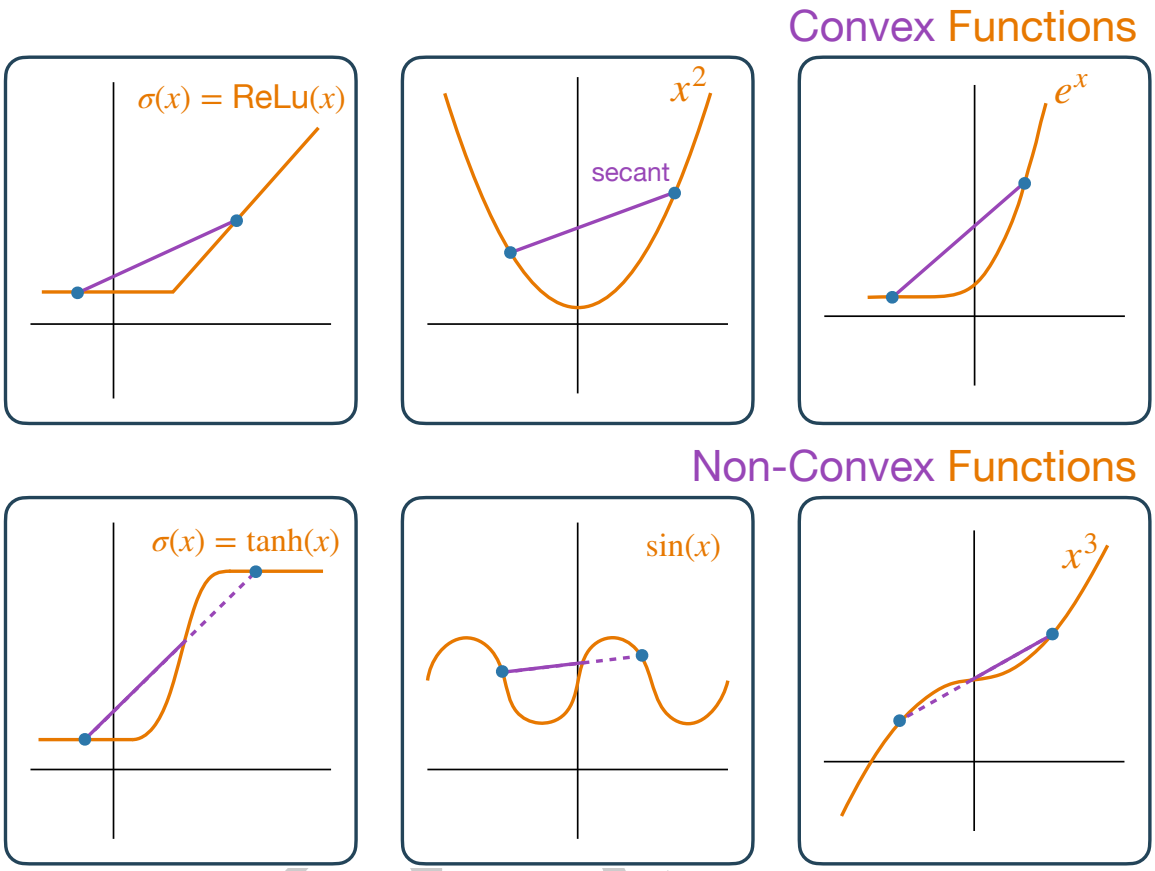


Figure 1.11: Examples convex and non-convex functions. The left two functions are commonly used activation functions in the nodes of neural networks.

shown in Fig. 1.11.

Technically we will often define f on a domain $\mathcal{D}$, in which case f may be considered convex restricted to $\mathcal{D}$ for any $\mathbf{x}, \mathbf{y} \in \mathcal{D}$.

Properties. There are several important properties of convex functions that we will use throughout this book:

- If $f(x)$ and $g(x)$ are convex, then $f(x)+g(x)$ is convex. This is useful in the context of machine learning, where we often construct sophisticated loss functions by adding several convex loss terms, as in:

$$L = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 + \alpha\|\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1.$$

- If $f(\mathbf{x})$ is convex, then $-f(\mathbf{x})$ is concave. The minimum of $f(\mathbf{x})$ becomes the maximum of $-f(\mathbf{x})$

- If $f(\mathbf{x})$ is convex, then $f(\mathbf{A}\mathbf{x} + \mathbf{b})$ is convex.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

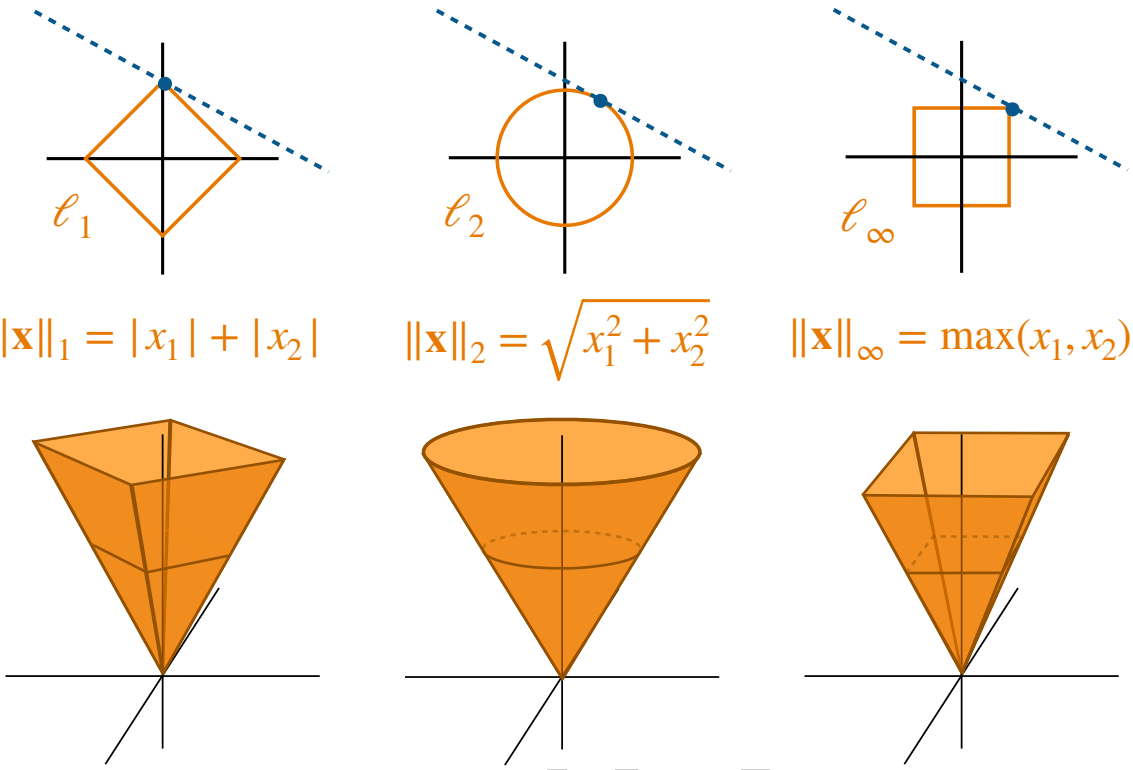


Figure 1.12: Illustration of several norms, all of which are convex functions. This is particularly important in machine learning, where we often minimize the sum-squared error in some appropriate norm. The ℓ_1 norm promotes sparsity, the ℓ_2 norm quantifies the sum-squared error, and the ℓ_∞ norm is used to penalize worst-case performance.

- If $f(\mathbf{x})$ and $g(\mathbf{x})$ are convex, then the pointwise maximum $h(\mathbf{x})$ of f and g is convex:

$$h(\mathbf{x}) = \max\{f(\mathbf{x}), g(\mathbf{x})\}.$$

- The composition of two functions $h(\mathbf{x}) = f(g(\mathbf{x}))$ is convex if:
  - f is convex and non-decreasing and g is convex;
  - f is convex and non-increasing and g is concave.

  Some examples include:
  - If g is convex then $\exp(g(x))$ is convex;
  - If g is concave then $-\log(-g(x))$ is convex;
  - if g is concave and positive, then $1/g$ is convex.

Norms are Convex

One very important class of convex functions are *norms*, so that $f(\mathbf{x}) = \|\mathbf{x}\|$. Norms widely used in optimization, as we are often trying to minimize the error of a function fit, where

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

error is quantified in an appropriate norm, such as the ℓ_2 norm. Several norms are shown in Fig. 1.12.

A norm is used to quantify the length of a vector, and it satisfies the following three properties:

1. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ for all scalar $\alpha \in \mathbb{R}$ (homogeneity)
2. $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (triangle inequality)
3. $\|\mathbf{x}\| \geq 0$ and $\|\mathbf{x}\| = 0$ only if $\mathbf{x} = \mathbf{0}$. (non negativity)

To prove that all norms are convex, we may write the convexity condition from Eq. (1.13) when $f(\mathbf{x}) = \|\mathbf{x}\|$ as:

$$\begin{aligned} \|\alpha\mathbf{x} + (1-\alpha)\mathbf{y}\| &\leq \|\alpha\mathbf{x}\| + \|(1-\alpha)\mathbf{y}\| && \text{(1.14a)} \\ &= \alpha\|\mathbf{x}\| + (1-\alpha)\|\mathbf{y}\| && \text{(1.14b)} \\ &= \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}). && \text{(1.14c)} \end{aligned}$$

The gradient of the L_2 norm in two-dimensions is given by

$$\nabla\|\mathbf{x}\|_2^2 = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} = 2\mathbf{x}.$$

This is useful in the context of least-squares regression (see Chapter 4), where we must minimize the sum of square errors

$$\nabla\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = 0.$$

The notion of a convex norm extends to infinite-dimensional L^p Hilbert spaces, where the norm of a function f is now given by

$$\|f\|_p = \left(\int |f|^p \, dx \right)^{\frac{1}{p}}.$$

This allows us to extend our optimization framework to function spaces and optimize systems governed by partial differential equations. We now use the Minkowsky inequality:

$$\|f+g\|_p \leq \|f\|_p + \|g\|_p.$$

There are several *pseudo* norms that are not actually norms, but can be useful anyhow. Two pseudo norms are shown in Fig. 1.13. The ℓ_0 norm quantifies how many non-zero entries are in a vector $\mathbf{x}$, which is a measure of the sparsity of the vector. Although ℓ_0 is neither a norm, nor is it convex, it is often possible to minimize the ℓ_0 norm in practice by *relaxing* the problem and minimizing the ℓ_1 norm instead. Relaxation is a fundamental technique in non-convex optimization, and will be discussed more throughout this book.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

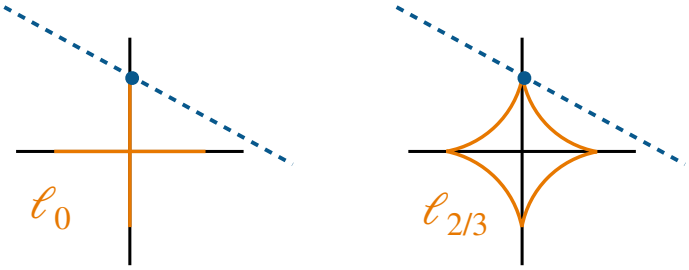


Figure 1.13: Examples of non-convex *pseudo* norms. These are not technically norms.

Relaxation and Regularization

Two of the most important topics in modern optimization are those of relaxation and regularization. These are so important that they will appear throughout the book, starting in Chapter 4.

Often, it is possible to relax non-convex terms in the objective function to related convex approximations, and there is a large body of analysis showing how and when the optimal solutions to the relaxed problems will converge to the optimal solutions of the original non-convex problems. Recall that often the addition of a non-convex term, such as $\|\cdot\|_0$ lead to NP-hard optimization problems, where it would require a brute force combinatorial search to find the global optimum. Thus, relaxation provides a powerful, and often enabling, strategy that is increasingly used in data analysis and machine learning.

Regularization is another critically important topic, where we add terms to the objective function to make an ill-posed problem well posed. Regularization allows one to incorporate prior knowledge and to guide the optimization to more favorable solutions. For example, if we want a sparse solution $\mathbf{x}$ we may add the $\|\cdot\|_1$ norm, which relaxes the $\|\cdot\|_0$ pseudo norm. If we want a low-rank matrix solution $\mathbf{X}$ we may add the nuclear norm $\|\mathbf{X}\|_*$, which relaxes $\text{rank}(\mathbf{X})$. Many of these regularizing terms arise as Bayesian priors, as discussed in Chapter 7.

Epigraph of a Function

The *graph* of a function is the set of all points on the function:

$$\text{graph}(f) = \{(\mathbf{x}, f(\mathbf{x})) \,|\, \mathbf{x} \in \boldsymbol{\mathcal{D}}\}. \tag{1.15}$$

This is a useful representation of the function as a set of points in the $n+1$-dimensional space $\mathbb{R}^n \times \mathbb{R}$ given by $(\mathbf{x}, f(\mathbf{x}))$.

The *epigraph* of a function is the set of all points above the function:

$$\text{epi}(f) = \{(\mathbf{x}, y) \,|\, \mathbf{x} \in \boldsymbol{\mathcal{D}} \text{ and } y \geq f(\mathbf{x})\}. \tag{1.16}$$

The epigraph of a convex function f is shown in Fig. 1.14. A function is convex if and only if its epigraph is a convex set, which is left as an exercise.

The *hypograph* of a function is the set of all points below the function:

$$\text{hyp}(f) = \{(\mathbf{x}, y) \,|\, \mathbf{x} \in \boldsymbol{\mathcal{D}} \text{ and } y \leq f(\mathbf{x})\}. \tag{1.17}$$

A function is concave if and only if its hypograph is a concave set.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

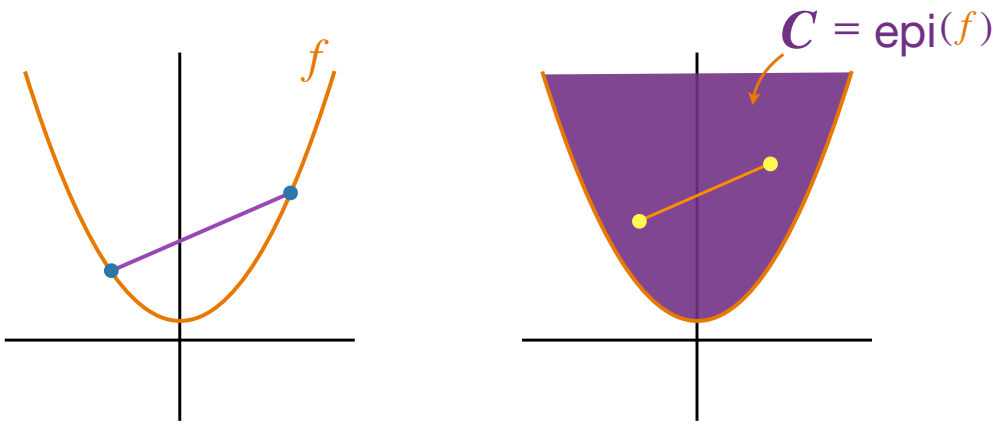


Figure 1.14: The epigraph of a function f is the set of points x, y above the graph. The function f is convex if and only if the epigraph is a convex set.

Jensen's Inequality

Jensen's inequality is one of the most powerful and intuitive results in the theory of convex functions, and it is used throughout various fields of mathematics and statistics. In essence, it generalizes the convexity condition for a convex function.

Mathematically, Jensen's inequality may be stated as

$$\alpha_1 f(\mathbf{x}_1) + \alpha_2 f(\mathbf{x}_2) + \cdots + \alpha_m f(\mathbf{x}_m) \geq f(\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_m \mathbf{x}_m) \tag{1.18}$$

for any $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ and $\alpha_j \geq 0$ such that $\alpha_1 + \alpha_2 + \cdots + \alpha_m = 1$.

This inequality has an intuitive geometric interpretation. The standard condition for convexity means that the line segment connecting any two points on the function will be above, or at least not below, the function. Jensen's inequality generalizes this idea to more than two points on the function. Specifically, if we have three points (i.e., $m = 3$) on a convex function, then the triangle formed with these three points as the vertices also must not lie below the function. This is shown schematically in Fig. 1.15.

This geometric interpretation also generalizes to more points m and to higher-dimensional spaces $\mathbf{x} \in \mathbb{R}^n$. Given a set of m points $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ then the linear combination $\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_m \mathbf{x}_m$ forms a convex polytope in $\mathbb{R}^n$. Some subset of the points will be the vertices of this polytope. Other points may be interior to the polytope. Similarly, the points $(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)), \cdots, (\mathbf{x}_m, f(\mathbf{x}_m)$ form a convex polytope that is entirely a subset of the epigraph of f. In other words, this convex polytope lies entirely above the function f, similar to the secant line. This is shown in Fig. 1.16.

Probability Example. One particularly useful application of Jensen's inequality is in the field of probability. We may use Jensen's inequality to prove a fundamental property about the expected value of a function of a random variable x. If $p(x) \geq 0$ and $\int_S p(x)\, dx = 1$,

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

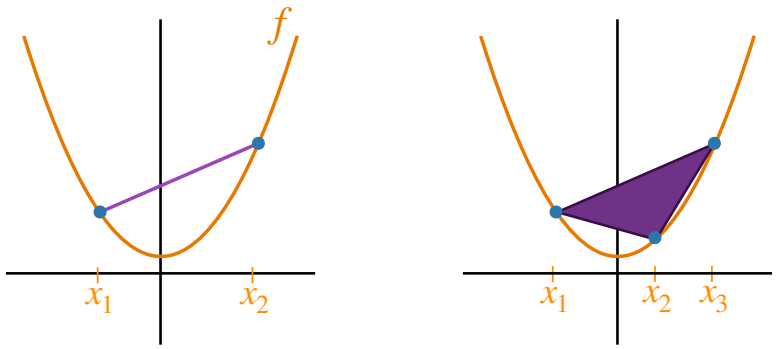


Figure 1.15: Illustration of Jensen's inequality in two-dimensions. Jensen's inequality generalizes the convexity condition that all secant lines lie above the function to state that all convex polytopes formed by m points on the function will also lie above the function.

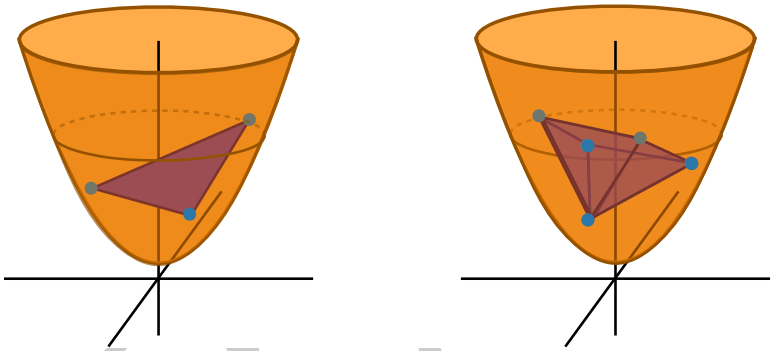


Figure 1.16: Illustration of Jensen's inequality in three-dimensions. In higher dimensions, Jensen's inequality states that any convex polytope formed from m points on the function will lie entirely above the function, similar to the secant lines connecting any two points on the function.

then

$$f\underbrace{\left(\int_S p(x)x\,dx\right)}_{E(x)} \leq \underbrace{\int_S p(x)f(x)\,dx}_{E[f(x)]}. \tag{1.19}$$

Thus

$$f\left(E(x)\right) \leq E\left(f(x)\right). \tag{1.20}$$

A concrete example of this inequality is the following:

$$\left(E(x)\right)^2 \leq E\left(x^2\right). \tag{1.21}$$

To prove this using Jensen's inequality, we must extend the finite sum in Eq. (1.18) to a continuous integral. This will be left as a homework exercise for the reader.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

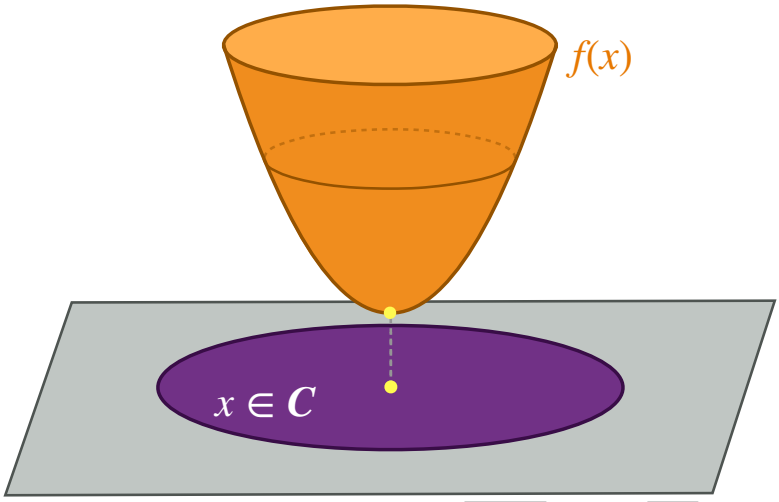


Figure 1.17: Illustration of a convex optimization problem, defined by a convex objective function over a convex feasible set. This example is strictly convex, with a unique global minima.

Convex Optimization Problems

An optimization problem with a convex function f defined over a convex feasible set $\mathcal{C}$ is a *convex* optimization problem, as shown in Fig. 1.17. If either the function f, or the feasible set $\mathcal{C}$ are non-convex, then the optimization problem is non-convex as shown in Fig. 1.18. Non-convex problems may have multiple local minima, maxima, and saddle points, and the complexity of the optimization landscape typically becomes more challenging in higher dimensions.

We are now ready to prove the central result in convex optimization: that for a convex function f over a convex set $\mathcal{C}$, a local minimum is also a global minimum. We will prove this by contradiction.

Proof. We assume that $\mathbf{x}$ is a local minimum, so that there is an ϵ-ball around $\mathbf{x}$

$$B_\epsilon(\mathbf{x}) = \{\mathbf{z} | \|\mathbf{z} - \mathbf{x}\| < \epsilon\}$$

so that any feasible $\mathbf{z} \in B_\epsilon(\mathbf{x})$ satisfies

$$f(\mathbf{z}) \geq f(\mathbf{x}).$$

We now assume that there exists a $\mathbf{y}$ that is a *better* minima

$$f(\mathbf{y}) < f(\mathbf{x}).$$

This is the condition that contradicts the main theorem, and so we will prove that this violates the assumption above.

Because $\mathbf{x}$ and $\mathbf{y}$ are both feasible, in the convex set $\mathcal{C}$, then the entire line segment connecting them is also feasible, meaning that it is also in the convex set $\mathcal{C}$. These points

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

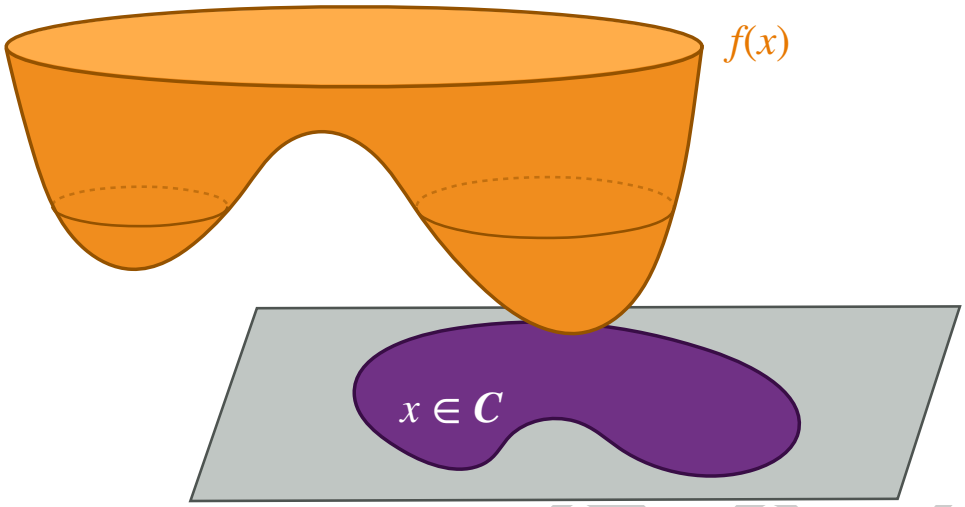


Figure 1.18: Illustration of a non-convex optimization problem, where the objective function and feasible set are both non-convex. This example has two local minima and a saddle point.

are parameterized by $\alpha \in [0,1]$ so that $\alpha = 0$ corresponds to the point $\mathbf{y}$ and $\alpha = 1$ to the point $\mathbf{x}$.

Some of these points are inside the ϵ-Ball. Specifically, we can choose an α so that the point is exactly on the boundary of the ϵ-Ball:

$$\alpha_1 = \frac{\epsilon}{\|\mathbf{y} - \mathbf{x}\|_2}$$

and thus we may choose another α that is halfway between $\mathbf{x}$ and this boundary:

$$\alpha_2 = \frac{\epsilon}{2\|\mathbf{y} - \mathbf{x}\|_2}.$$

We denote this point by $\mathbf{z} = (1 - \alpha_2)\mathbf{x} + \alpha_2\mathbf{y}$.

However, by the convexity of f, and given that $\mathbf{y}$ is a lower minima with $f(\mathbf{y}) < f(\mathbf{x})$, then for every $\alpha \in [0,1]$, we have

$$\alpha f(\mathbf{y}) + (1 - \alpha) f(\mathbf{x}) < f(\mathbf{x})$$

which implies that

$$f(\mathbf{z}) < f(\mathbf{x}).$$

However, since $\mathbf{z}$ is feasible and $\mathbf{z} \in B_\epsilon(\mathbf{x})$, this contradicts the assumption that $\mathbf{x}$ is a local minimum. ■

1.3 Gradients and the Geometry of Optimization

The gradient operator from vector calculus generalizes the derivative of a function of a single variable to the derivative of a function of multiple variables. The gradient has a deep

Copyright © July 15, 2025 Brunton. All Rights Reserved.

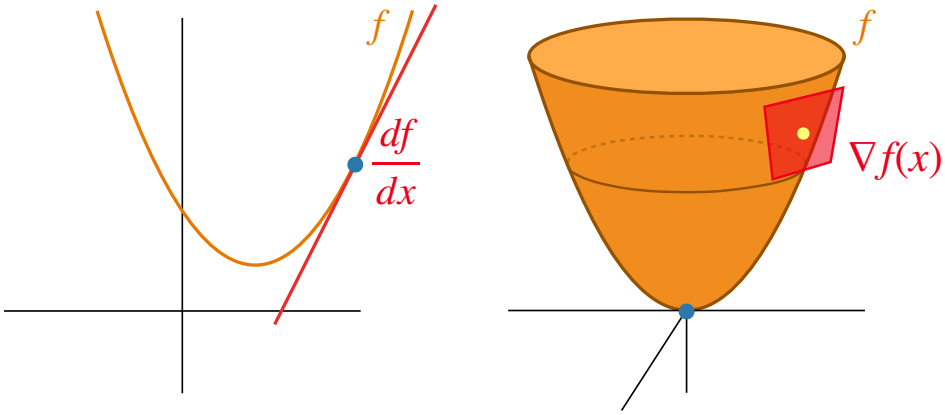


Figure 1.19: Examples of how the derivative and gradient define the tangent line and tangent space, respectively, for a one-dimensional function and a higher-dimensional function.

geometric interpretation, quantifying the local direction in which the function increases most rapidly. Thus, we may use this gradient direction to iteratively walk towards a local minimum or maximum of our objective function. This idea is the basis of all of gradient descent methods, including the stochastic gradient descent from machine learning, which will be discussed in Chapters 2 and 10. The gradient is visualized for a scalar function and a two-dimensional function in Fig. 1.19.

The gradient of a scalar function in two dimensions, $f(x, y)$ is defined as

$$\nabla f = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \end{bmatrix}. \tag{1.22}$$

The components of this vector are partial derivatives that quantify how much the function f changes when x and y are varied infinitesimally. Thus, if the function increases more rapidly in the y direction, the second component will be larger. The gradient of a function $f(\mathbf{x})$ of an n-dimensional vector $\mathbf{x} \in \mathbb{R}^n$ is given by

$$\nabla f = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_n \\ \vdots \\ \partial f/\partial x_n \end{bmatrix}. \tag{1.23}$$

Each component of ∇f measures how much f will change when the component x_j is varied.

Just as the derivative df/dx of a scalar function $f(x)$ defines a tangent line to the function at the point x, the gradient $\nabla f(\mathbf{x})$ of a function $f(\mathbf{x})$ defines a tangent plane to the function at $\mathbf{x}$. The scalar derivative df/dx is the slope of the tangent line, and $\nabla f(\mathbf{x})$ defines the slope of the tangent plane. Specifically, the gradient $\nabla f(\mathbf{x}_0)$ defines a linear approximation to $f(\mathbf{x})$ near $f(\mathbf{x}_0)$ via a first-order Taylor series approximation:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \underbrace{(\mathbf{x} - \mathbf{x}_0)}_{\Delta \mathbf{x}}. \tag{1.24}$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

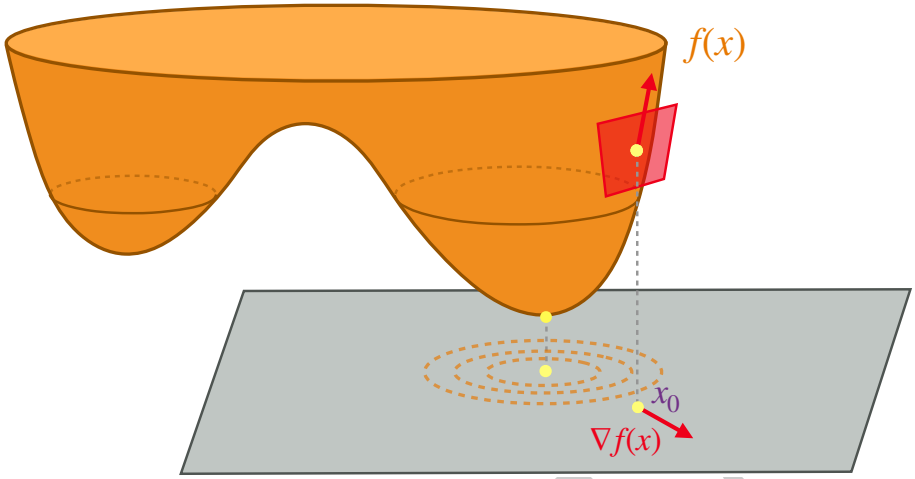


Figure 1.20: Illustration of a non-convex function, with two local minima. The gradient ∇f is also visualized, both as a tangent plane to the objective function f and as a vector direction over the set of feasible points.

The gradient tells how the function f will locally change, given a small change in $\mathbf{x}$. If we denote the small change by $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$, then the change in the function is approximated by $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x}$. In two dimensions, where $\Delta\mathbf{x} = \begin{bmatrix}\Delta x_1 & \Delta x_2\end{bmatrix}^T$, this can be written approximately as:

$$\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} = \frac{\partial f}{\partial x_1}\Delta x_1 + \frac{\partial f}{\partial x_2}\Delta x_2. \tag{1.25}$$

One of the most important properties of the gradient in the context of optimization is that it defines the direction in which $\Delta\mathbf{x}$ will maximally increase the function f locally. This is easy to see from Eq. (1.24), since $\nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0)$ is the inner product of the vector $\nabla f(\mathbf{x}_0)$ and $\Delta\mathbf{x}$, and this inner product will be largest when the two vectors point in the same direction.

Similarly, if we choose $\Delta\mathbf{x}$ in the negative gradient direction, $-\nabla f(\mathbf{x}_0)$ will cause the function to locally decrease the fastest, providing the basis for gradient descent. The gradient of a non-convex function is visualized in Fig. 1.20, where it is clear that the negative gradient may be used to locally minimize the function.

Gradients, Hessians, and Convexity

There are deep connections between the gradient of a function, $\nabla f(\mathbf{x})$, and convexity. In words, a function is convex if and only if all tangent planes defined by the gradient at every point on the function lie entirely below the function. Thus, the tangent planes are never above the function, and only bound it from below. In fact, they define the supporting hyperplanes for the convex epigraph, as discussed below.

We may write this connection mathematically as follows. A function $f(\mathbf{x})$ is convex if and only if it lives above its first order Taylor series approximation, given by

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \cdot (\mathbf{x} - \mathbf{x}_0). \tag{1.26}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The point $\mathbf{x}_0$ is the base point of the Taylor series expansion, and $\mathbf{x}$ is any other point in the domain of f. This statement is true for all $\mathbf{x}_0$ and $\mathbf{x}$. If there is an $\mathbf{x}^*$ so that $\nabla f(\mathbf{x}^*) = \mathbf{0}$, then $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for all $\mathbf{x}$. Therefore, $\mathbf{x}^*$ is a global minimum of the function $f(\mathbf{x})$ and $f(\mathbf{x})$ is a convex function. Proving this for a vector $\mathbf{x}$ is not trivial, but it is simpler and can be illustrative to show this for a scalar x.

Second Order Hessian Condition. Just as the gradient condition in Eq. (1.26) provides a first-order condition for convexity, there is a second-order condition in terms of the Hessian $\mathbf{H} = \nabla^2 f(\mathbf{x})$:

$$\mathbf{H} = \nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{1.27}$$

A function $f(\mathbf{x})$ is convex if and only if $\nabla^2 f(\mathbf{x}) \geq \mathbf{0}$ everywhere. This can be seen by expanding $f(\mathbf{x})$ in a second-order Taylor series about $\mathbf{x}_0$:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0). \tag{1.28}$$

It is clear that Eq. (1.26) will always be true if $\nabla^2 f(\mathbf{x}_0)$ is positive semidefinite[2]. When $\mathbf{H} > 0$ is positive definite, then the function is *strictly* convex, meaning that it has a single global minimum; this prevents a possible line or region of equivalent local minima which is possible when $\mathbf{H} = 0$ and we have a *flat* portion of the objective function. In words, if $\mathbf{H} > 0$ then the function has positive curvature everywhere, meaning it is shaped like a bowl.

Tangent Planes are Supporting Hyperplanes. Earlier we defined the graph of $f(\mathbf{x})$ as the pair of points $(\mathbf{x}, f(\mathbf{x}))$. Then the tangent plane may be defined by its normal vector, which is designed to be perpendicular to the gradient of this point on the graph:

$$\vec{T} = \nabla (\mathbf{x}, f(\mathbf{x})) = \begin{bmatrix} \mathbf{1} \\ \nabla_{\mathbf{x}} f(\mathbf{x}) \end{bmatrix},$$

where $\mathbf{1}$ is a vector of ones, obtained by taking the gradient of $\mathbf{x}$: $\nabla \mathbf{x} = \mathbf{1}$. The normal vector that is perpendicular to this tangent vector is given by:

$$\vec{n} = \begin{bmatrix} \nabla f(\mathbf{x}) \\ -\mathbf{1} \end{bmatrix}.$$

Interestingly, for a convex function, the set of all tangent planes defines a set of *supporting hyperplanes* that tightly surround the convex set above the function (i.e., the epigraph of f), as shown in Fig. 1.21.

[2] A matrix $\mathbf{Q}$ is positive semidefinite if $\mathbf{x}^T \mathbf{Q} \mathbf{x} \geq 0$ for all $\mathbf{x}$. We often denote this as $\mathbf{Q} \geq 0$. If $\mathbf{x}^T \mathbf{Q} \mathbf{x} = 0$ only for $\mathbf{x} = 0$ then the matrix is positive definite, denoted as $\mathbf{Q} > 0$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

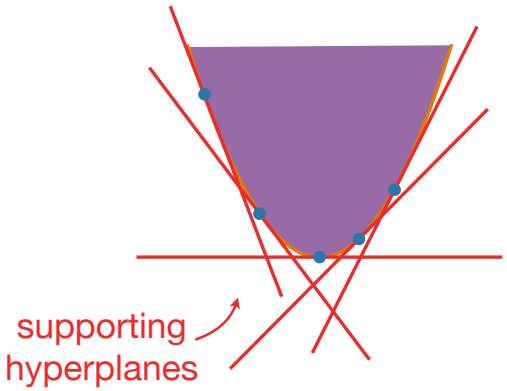


Figure 1.21: The tangent vectors of a convex function define the boundaries of its supporting hyperplanes. The normal direction to each of these hyperplanes is given by $\vec{n} = \begin{bmatrix} \nabla f(\mathbf{x}) \\ -\mathbf{1} \end{bmatrix}$ as described below.

1.4 Constraints and Lagrange Multipliers

Constraints are a fundamental ingredient of many optimization problems, and there are several approaches to handle them. The method of Lagrange multipliers provides an elegant way to enforce constraints by embedding them into the objective function. This approach not only offers practical techniques for solving constrained problems but also introduces key ideas in duality, which are central to optimization theory. Lagrange multipliers are widely used in control theory and other engineering applications. Here we will provide a very simple and intuitive overview of Lagrange multipliers, which will be explored more in Chapter 5 and throughout the book.

Consider the simple constrained optimization problem with equality constraints:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}), \tag{1.29a}$$

$$\text{s.t.} \quad h(\mathbf{x}) = 0. \tag{1.29b}$$

Using Lagrange multipliers, we can reformulate this problem as:

$$\min_{\mathbf{x}} \quad \underbrace{f(\mathbf{x}) + \lambda h(\mathbf{x})}_{\mathcal{L}}, \tag{1.30}$$

where $\mathcal{L}$ is referred to as the *Lagrangian*. Here, λ is a Lagrange multiplier that weights the constraint violation. In essence, this may be viewed as a regularization term that promotes the constraint being satisfied. Regularization is a key concept in optimization that we will use throughout the book to enforce constraints and promote other prior physical knowledge.

We will show in Chapter 5 that a condition for optimality of these types of constrained problem is that the gradients of the objective function and constraint equation must align:

$$\nabla_{\mathbf{x}} \mathcal{L} = \nabla f(\mathbf{x}^*) + \lambda \nabla h(\mathbf{x}^*) = 0, \quad \nabla_{\lambda} \mathcal{L} = h(\mathbf{x}^*) = 0. \tag{1.31}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

If there are several equality constraints, so $\mathbf{h}(\mathbf{x}) = 0$, then this may be written as

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}). \tag{1.32}$$

Here $\boldsymbol{\lambda}$ is a vector of the Lagrange multipliers.

For inequality constraints, $g(\mathbf{x}) \leq 0$, we modify the notion of the Lagrange multiplier λ to satisfy additional conditions:

$$\lambda \geq 0, \quad \lambda g(\mathbf{x}) = 0. \tag{1.33}$$

This ensures that λ enforces the constraint only when it is active, meaning that the optimal solution is at the boundary of this constraint equation, and not in the interior. This generalized theory of Lagrange multipliers is know as KKT theory and will be discussed more in Chapter 5.

One way to see how Lagrange multipliers work is through the use of a penalty function $P(g(\mathbf{x}))$ which is 0 for $g(\mathbf{x}) \leq 0$ and ∞ for $g(\mathbf{x}) > 0$:

$$P(g(\mathbf{x})) = \begin{cases} 0, & g(\mathbf{x}) \leq 0 \\ \infty, & g(\mathbf{x}) > 0. \end{cases}$$

If we add this penalty function to the objective function

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) + P(g(\mathbf{x}))$$

then clearly the optimal solution is equivalent to the constrained optimization problem. However, the penalty function is no longer smooth or easy to optimize over. So instead we add $\lambda g(\mathbf{x})$ and as $\lambda \to \infty$ this approaches $P(g(\mathbf{x}))$. Technically we don't need to take the limit as $\lambda \to \infty$; at some finite λ, this term will become large enough to make the constraint active, after which the optimal solution will be the same. Finally, the dual problem is given by[3]:

$$\max_{\lambda \geq 0} \min_{\mathbf{x}} \quad f(\mathbf{x}) + \lambda g(\mathbf{x}). \tag{1.34}$$

The dual provides a lower bound on the primal problem and has practical and theoretical significance, particularly in large-scale optimization and sensitivity analysis. It is not obvious that these formulations are equivalent, but under some conditions, the optimal solutions of each will in fact be equivalent. This notion of duality in optimization will be revisited several times in this book.

Simple Examples of Lagrange Multipliers

Here we will explore two simple examples with actual functions to see how the mechanics of Lagrange multipliers work. The first example will involve an active constraint, so that the optimal solution is at the boundary of the constraint, and the second example will show how Lagrange multipliers work when there is a trivial constraint that doesn't change the optimum solution.

[3]We will discuss duality in much greater detail in later chapters, such as Chapter 5. *Minimax* and *maximin* problems often arise in the context of dual problems, where we minimize over one variable, such as the primal variable $\mathbf{x}$, and maximize over another variable, such as the dual variable $\boldsymbol{\lambda}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Example with Active Constraint. Consider the optimization problem

$$\begin{aligned} \min_x \quad & f(x), \\ \text{s.t.} \quad & g(x) \leq 0 \end{aligned}$$

with $f(x) = (x-1)^2 + 1$ and $g(x) = x$. The minimum of the unconstrained optimization problem occurs at $x^* = 1$, but the constraint forces $x \leq 0$. It is possible to plot this objective function and constraint and verify the intuition that the constrained optimal solution is on the boundary of the constraint, so that $x^* = 0$.

Reformulating this problem using Lagrange multipliers yields the dual formulation:

$$\max_{\lambda \geq 0} \min_x \ (x-1)^2 + 1 + \lambda x.$$

We first compute the inner minimization, which results in

$$\min_x \ f + \lambda g \quad \Longrightarrow \quad \nabla f + \lambda \nabla g = 2x - 2 + \lambda = 0 \quad \Longrightarrow \quad x^* = \frac{1}{2}(2 - \lambda).$$

Next we substitute this into $f(x) + \lambda g(x)$ and maximize with respect to $\lambda \geq 0$:

$$\begin{aligned} \max_{\lambda \geq 0} f(x^*) + \lambda g(x^*) &= \max_{\lambda \geq 0} \left(1 - \frac{\lambda}{2} - 1\right)^2 + 1 + \lambda \left(1 - \frac{\lambda}{2}\right) \\ &= \max_{\lambda \geq 0} \frac{\lambda^2}{4} + 1 + \lambda - \frac{\lambda^2}{2} = \max_{\lambda \geq 0} -\frac{\lambda^2}{4} + \lambda + 1. \end{aligned}$$

Thus the optimum λ is obtained by taking the derivative with respect to λ and setting equal to zero:

$$-\frac{\lambda}{2} + 1 = 0 \quad \Longrightarrow \quad \lambda^* = 2.$$

Plugging this back into our expression for x^* yields $x^* = 0$.

A fundamental question arises: why is $\lambda = 2$ special? Remember, we are not really trying to minimize $f(x) + \lambda g(x)$ but rather minimize $f(x)$ such that $g(x) \leq 0$. If $\lambda < 2$ then the constraint is not satisfied. If $\lambda > 2$ then the constraint is satisfied but $f(x)$ is not minimized. To visualize this, plot the augmented objective function for several values of λ, along with the constraint region. This will be explored more in Exercise 1-11.

Example with Trivial Constraint. Let's now explore a case where the constraint is not active in the minimization problem. We will consider the same constrained optimization problem but with $f(x) = (x+1)^2 + 1$ and $g(x) = x$. The minimum of the unconstrained problem occurs at $x^* = -1$ which also satisfies the constraint.

Again, we reformulate this problem using the augmented Lagrangian form:

$$\max_{\lambda \geq 0} \min_x \ (x+1)^2 + 1 + \lambda x \tag{1.35}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We solve the inner minimization over x:

$$\frac{\partial \mathcal{L}}{\partial x} = 2x + 2 + \lambda = 0 \quad \Longrightarrow \quad x = -1 - \frac{\lambda}{2}. \tag{1.36}$$

Substituting this into the Lagrangian yields

$$-\frac{\lambda^2}{4} - \lambda + 1. \tag{1.37}$$

Maximizing with respect to yields

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{\lambda}{2} - 2 = 0 \quad \Longrightarrow \quad \lambda = -2. \tag{1.38}$$

Since this is negative, the maximum of the function for $\lambda \geq 0$ occurs when $\lambda^* = 0$, at which value $x^* = -1$. Thus, when the constraint is not active, then $\lambda^* = 0$.

1.5 Applications: Inverse Problems, Control, and Machine Learning

The power of optimization is particularly evident in its many modern applications. Entire books could be, and have been, written about individual application areas, for example in industrial optimization or network flow optimization.

In this text, we will focus on three major application areas to motivate and develop theory: inverse problems, control theory, and machine learning. These applications are of particular modern relevance, and represent three frontiers of optimization research. In addition, each of these application domains often involve interacting with or modeling the real, physical world, making them highly relevant for science and engineering.

Modern applications of optimization are typically high-dimensional and nonlinear, leading to non-convex problems. There are several approaches to either relax these into convex problems, or to tackle the full non-convex optimization, which we will develop in later chapters.

Inverse Problems

Inverse problems seek to infer unknown parameters or inputs that best match observed data, which is typically represented by an equation or system where the solution minimizes the discrepancy between observed and modeled data. Thus the goal is to find the unknown parameters $\mathbf{x}$ of a forward model $\mathbf{F}(\mathbf{x})$ that best matched observed data $\mathbf{y}_{\text{obs}}$:

$$\mathbf{F}(\mathbf{x}) \approx \mathbf{y}_{\text{obs}}. \tag{1.39}$$

We typically employ optimization to reframe the problem:

$$\min_{\mathbf{x}} \|\mathbf{F}(\mathbf{x}) - \mathbf{y}_{\text{obs}}\|^2. \tag{1.40}$$

The objective function is the model mismatch $f(\mathbf{x}) = \|\mathbf{F}(\mathbf{x}) - \mathbf{y}_{\text{obs}}\|^2$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

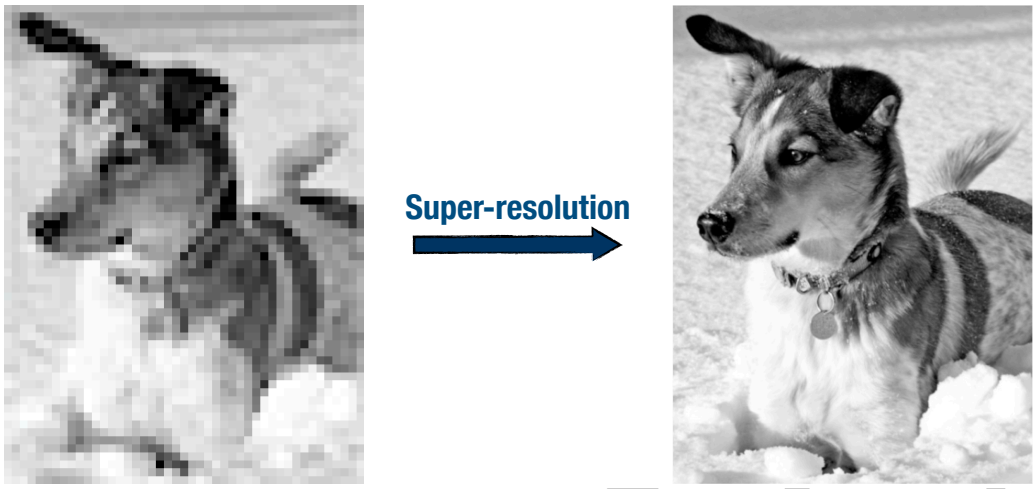


Figure 1.22: Inverse problems, such as super-resolution inversion, are optimization problems, as we seek an inverse solution that minimizes some appropriate loss function. The super-resolution task of inverting a low-resolution image to uncover the true high-resolution image is ill-posed, meaning that there are many high-resolution images that yield the same low-resolution input. Thus regularization and other prior information must be added to the optimization problem to promote a unique and reasonable solution.

Often, the observations are partial and noisy, and thus don't contain enough information to uniquely determine the unknown parameters, making the inverse problem *ill-posed* or *ill-conditioned*. Thus, we employ relaxation, regularization, and other prior information to make the problem well posed.

Chapter 8 will provide a deep dive into applied optimization for inverse problems. Some of the main examples and case studies include:

- **Classical inverse problems.** There is a rich history of classical inverse problems, such as finding the roots of a polynomial, solving a (potentially ill-posed) matrix system of equations, or estimating the derivative of noisy data. These all provide historical context for fundamental algorithms that are ubiquitous in machine learning and data science.

- **Image-based inverse problems.** Much of the modern work in inverse optimization has centered around inverse problems in imaging sciences, especially in medical imaging and astronomy. Both fields are characterized by partial and indirect measurements that must be used to reconstruct an image of interest. Figure 1.22 provides one such example of an image-based inverse problem: the task of super-resolution, or reconstructing a high-resolution image from a low-resolution image. Other examples will include de-noising, de-blurring, filling in damage or occlusions, etc.

- **Image-based inverse problems with physics.** If an image is obtained from a physical system, it is often possible to use physical priors to regularize the optimization

Copyright © July 15, 2025 Brunton. All Rights Reserved.

problem. For example, modern computational imaging of black holes involves inverse optimization blended with physics.

- **PDE-constrained inversion.** One of the most challenging classes of inverse problems are those that involve a physical system governed by complex partial differential equation dynamics. For example, designing the shape of a wing to optimize aerodynamic performance, is one of the computational grand challenge problems in engineering. These problems are so challenging in part because of the computational overhead associated with simulating the multiscale dynamics of these systems. Other relevant problems include seismic inversion, or localizing earthquakes and estimating geophysical parameters from seismic data.
- **System identification.** Modeling time series data with ordinary differential equations is a broad topic with a rich history in inverse optimization. This field is closely related to control theory and has dramatically advanced in recent years with advances in applied optimization.

Control Theory

The goal of a control system is to design a control law

$$\mathbf{u} = \mathbf{k}(\mathbf{x}) \tag{1.41}$$

that minimizes a cost function

$$J \triangleq J(\mathbf{x}, \mathbf{u}). \tag{1.42}$$

subject to the constraints of a dynamical system that governs the evolution of the system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{1.43}$$

The cost function J will often be a positive-semidefinite quadratic form, and for linear dynamics, there is an analytic closed form solution, derived using Lagrange multipliers.

In the more general case, we will only have access to partial, noisy measurements $\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{n}$, the dynamics model will be nonlinear and have uncertainty, and there will be disturbances to the system. All of these make the optimization problem more challenging.

Much of the technological progress in the past century can be attributed to improvements in our ability to control complex dynamical systems. There are still many challenges in control theory that will undoubtedly benefit from advanced techniques in applied optimization. Chapter 9 will explore several topics in optimization for control theory, including:

- **Linear optimal control.** The most complete theory of control has been developed for linear systems, and this topic provides an intuitive mathematical foundation for nearly every other field of control. Linear optimal control includes the linear quadratic regulator (LQR) full-state feedback control (shown in Fig. 1.23) and the Kalman filter, both of which have analytic solutions. The dynamics enters these problems as a constraint, providing a concrete and intuitive connection to Lagrange multipliers.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

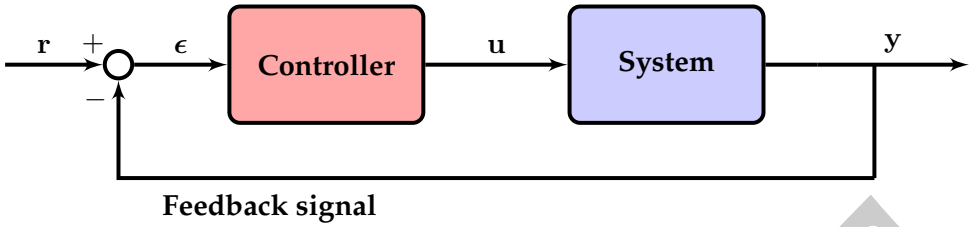


Figure 1.23: Closed-loop feedback control diagram. The sensor signal $\mathbf{y}$ is fed back and subtracted from the reference signal $\mathbf{r}$, providing information about how the system is responding to actuation and external disturbances. The controller uses the resulting error ϵ to determine the correct actuation signal $\mathbf{u}$ for the desired response. Feedback is often able to stabilize unstable dynamics while also effectively rejecting disturbances and attenuating sensor noise.

- **Nonlinear optimal control.** Full nonlinear control problems are characterized by the Hamilton-Jacobi-Bellman equations, a set of partial differential equations that must be satisfied for an optimal control solution. The HJB equations are based on variation optimal conditions that generalize the classic ideas of variational principles in Lagrangian and Hamiltonian mechanics.

- **Reinforcement learning.** Reinforcement learning is an important class of optimization problems at the intersection of control theory and machine learning, where the optimal control actions of an agent are learned through a process of exploration and exploitation. RL is a deep topic, and a more modern treatment involving deep learning will be provided in the chapter on machine learning.

- **Dynamic programming.** Dynamic programming may be thought of as the optimization engine used to solve nonlinear optimal control problems. DP is also used extensively in reinforcement learning, although modern approachees are increasingly incorporating deep learning.

- **Model predictive control.** Model predictive control is perhaps the most widely-used industrial control optimization technique. MPC uses real-time optimization to determine a control action over a receding, finite-time horizon. The performance and applicability of MPC is directly related to our ability to solve challenging optimization problems in real time, and is thus one of the main applications of modern optimization.

- **Linear matrix inequalities.** Linear matrix inequalities are how we solve many linear control problems in practice using modern techniques in optimization. LMIs are often not included in a course on control theory or in a course on optimization. However, they are the workhorse of many control techniques.

- **Path planning.** Closely related to reinforcement learning and nonlinear optimal control, path planning is a rich topic in optimization that involves navigating a complex

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

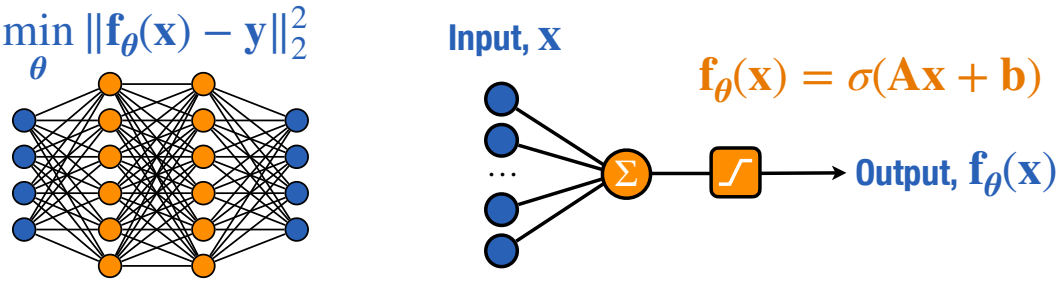


Figure 1.24: Training a neural network is inherently an optimization problem, involving tuning the weights of a neural network to minimize a loss function averaged over the training data.

environment subject to constraints and heterogeneous costs. Path planning is closely related to several classical variational problems in mechanics, such as the brachistochrone.

- **Differentiable control.** If the system model and controller are differentiable, for example represented as neural networks, then powerful gradient-based optimization techniques become available, even for high-dimensional control problems.

Machine Learning

Machine learning is the most ubiquitous and compelling modern application of applied optimization. Machine learning involves using optimization to tune the parameters of a model to minimize a lost function averaged over a set of training data; a schematic of a neural network is shown in Fig. 1.24. The machine learning model is a function

$$\mathbf{y} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) \tag{1.44}$$

where $\mathbf{x}$ are the inputs to the model, $\mathbf{y}$ are the outputs, and $\boldsymbol{\theta}$ are the parameters to be optimized over. The structure of the machine learning model $\mathbf{f}_{\boldsymbol{\theta}}$ defines a space of possible functions that might describe the data, parameterized by $\boldsymbol{\theta}$.

The loss function $\mathcal{L}$ will typically include several terms, with the mean squared error (MSE)

$$\mathcal{L}_1 = \sum_{j=1}^{N} \|\mathbf{y}_j - \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_j)\|^2 \tag{1.45}$$

summed over the number of data points N.

Chapter 10 provides an in-depth discussion of how optimization is used in various aspects of machine learning, including:

- **Deep learning.** Deep learning is the fastest growing field of machine learning, where a deep neural network is used for function approximation. This field is fueled by an abundance of data, modern high-performance computing architectures, and advanced optimization techniques, such as stochastic gradient descent and the ADAM optimizer.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- **Stochastic gradient descent.** SGD is the optimization engine underlying deep learning. It scales to very large systems and incorporates momentum methods, such as the ADAM optimizer, to improve convergence.
- **Hyperparameter optimization.** Most machine learning algorithms involve several hyperparameters, such as the size of the neural network, weights of regularizers in the loss function, etc. Tuning hyperparameters is a challenging meta-optimization problem.
- **Deep reinforcement learning.** We will revisit reinforcement learning from the modern perspective of deep learning. Deep learning has massively improved the scope and applicability of reinforcement learning, for example to improve protein folding, nuclear fusion, to dominate humans in nearly all games, and for other advanced control applications.
- **Generative methods.** Generative methods in machine learning, such as large language models and diffusion models for image generation, are becoming increasingly powerful and involve advanced optimization methods to formulate and train these models.
- **Physics-constrained machine learning.** Incorporating prior knowledge, such as known physics, is one of the open frontiers of machine learning research. There are two main approaches that will be discussed here: 1) incorporating physical priors as regularizing terms in the loss function (e.g., physics informed neural networks), and 2) directly enforcing physics as constraints in a constrained optimization problem.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) Convex optimization, by S. Boyd and L. Vandenberghe, 2009 [48].

(2) Numerical optimization, by J. Nocedal and S. J. Wright, 1999 [268].

(3) Practical methods of optimization, by R. Fletcher, 2000 [131].

(4) Introductory lectures on convex optimization: A basic course, by Y. Nesterov, 2013 [264].

(5) Constrained optimization and Lagrange multiplier methods, by D. P. Bertsekas, 2014 [34].

(6) Lectures on convex optimization, by Y. Nesterov, 2018 [265].

(7) Engineering optimization: theory and practice, by S. S. Rao, 2019 [288].

(8) Practical optimization, by P. E. Gill, M. Walter, and M. H. Wright, 2019 [145].

(9) Introduction to applied optimization, by U. M. Diwekar, 2020 [101].

Papers and reviews

(1) CVXPY: A Python-embedded modeling language for convex optimization, by S. Diamond and S. Boyd, *Journal of Machine Learning Research*, 2016 [98].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

Exercise 1–1: An aircraft must fly 2000 miles. It costs $1000\$$ per hour to fly. In addition, fuel costs $5\$$ per gallon, and the rate of fuel consumption is $100 + .001V^2$, where V is the cruising speed, measured in miles per hour.

(a) What is the minimum-cost cruising speed V, time of flight t, and trip cost C?

(b) Suppose the cost of fuel doubles. What is the minimum-cost cruising speed, time of flight, and trip cost?

Exercise 1–2: Determine if the following functions are convex or not. What is the largest domain of convexity around the origin?

(a) $f(x) = x^4$

(b) $f(x) = x^4 - .01x^2$

(c) $f(x, y) = -e^{x^2+y^2}$

(d) $f(\mathbf{x}) = -e^{\|\mathbf{x}\|}$

(e) $f(\mathbf{x}) = \sum_j |x_j|$

(f) $f(x, y) = 4\cos(x)^2 + 2\sin(y)^2$

(g) $f(x, y) = 4\cos(x)^2 - 2\sin(y)^2$

(h) $f(x, y) = xy$

Exercise 1–3: Sketch the feasible sets for the following constraint equations:

(a) $S_1 = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1\}$

(b) $S_2 = \{(x, y) \in \mathbb{R}^2 : x^2 - y^2 \leq 1\}$

(c) $S_3 = \{(x, y) \in \mathbb{R}^2 : |x| + |y| \leq 1\}$

(d) $S_4 = \{(x, y) \in \mathbb{R}^2 : x \geq 0, y \geq 0\}$

(e) $S_5 = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$

(f) $S_6 = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \geq 1\}$

(g) $S_7 = \{(x, y) \in \mathbb{R}^2 : y \geq x^2\}$

(h) $S_8 = \{(x, y) \in \mathbb{R}^2 : y^2 \leq x\}$

(i) $S_9 = \{(x, y) \in \mathbb{R}^2 : y = 2x + 3\}$

(j) $S_{10} = \{(x, y) \in \mathbb{R}^2 : 1 \leq x^2 + y^2 \leq 4\}$

Exercise 1–4: Compute the gradient ∇f and Hessian $\nabla^2 f$ of the following functions:

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(a) $f(x,y) = x^2 + y^2$

(b) $f(x,y) = x^4 - x^2y^2 + y^4$

(c) $f(x,y) = x^2y + y^2x$

(d) $f(x,y) = e^{x^2+y^2}$

(e) $f(x,y) = \sin(xy)$

(f) $f(\mathbf{x}) = \|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$

(g) $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} + b$, where $\mathbf{a} \in \mathbb{R}^n, b \in \mathbb{R}$

(h) $f(\mathbf{x}) = \ln(1 + \|\mathbf{x}\|^2)$

(i) $f(\mathbf{x}) = e^{\mathbf{a}^T\mathbf{x}}$, where $\mathbf{a} \in \mathbb{R}^n$

Exercise 1–5: Determine the extrema of the following functions, and which of these are local/global minima:

(a) $f(x) = \cos(x)$

(b) $f(x) = -e^{(x-5)^2}$

(c) $f(x,y) = x^2 - y^2$ (hyperbolic paraboloid)

(d) $f(x,y) = x^4 + y^4 - x^2y^2$

(e) $f(x,y) = x^2 + 2y^2 - 4x$

(f) $f(x) = x^6 - 3x^4 + 2x^2$

(g) $f(x) = e^{-x^2}$

(h) $f(\mathbf{x}) = \|\mathbf{x}\|^2 + 2\mathbf{a}^T\mathbf{x} + b$, where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{a}, b$ are constants

(i) $f(\mathbf{x}) = \|\mathbf{x}\|^3$, where $\mathbf{x} \in \mathbb{R}^n$

Exercise 1–6: Identify the following optimization problems as linear, convex, or nonconvex, and justify your answer:

1. Minimize $f(\mathbf{x}) = x_1^2 + x_2^2$ subject to $x_1 + x_2 = 1$.
2. Minimize $f(\mathbf{x}) = e^{x_1} + e^{x_2}$ subject to $\|\mathbf{x}\| \leq 2$.
3. Maximize $f(\mathbf{x}) = x_1x_2$ subject to $x_1, x_2 \geq 0$ and $x_1 + x_2 = 5$.

Exercise 1–7: Consider the following general optimization problem domains. In each, determine what are the optimization variables $\mathbf{x}$, what is the objective function $f(\mathbf{x})$ and what are the constraints on the system. Try to add as much detail as possible to put these systems into a standard form.

1. Portfolio management in a hedge fund;

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

2. Optimizing the shape of a wing for aerodynamic performance;
3. Optimizing the pricing of concert tickets in the Sphere.

Exercise 1–8: Show that the L_0 "norm"

$$\|\mathbf{x}\|_0 = \#\text{nonzero entries of } \mathbf{x}$$

is not actually a norm. What property does it violate?

Exercise 1–9: Explain why the ℓ_1 norm is a suitable proxy for the ℓ_0 pseudo norm. Does this relaxation become more or less effective for higher dimensional $\mathbf{x} \in \mathbb{R}^n$?

Exercise 1–10: Using Lagrange multipliers, solve the problem

$$\min_{x,y\in\mathbb{R}} \quad x^2 + y^2 \quad \text{subject to} \quad (x-1)^2 + (y-1)^2 = 2.$$

Plot the constraint region and identify the minimizing point(s).

Exercise 1–11: In this problem, we will revisit the Lagrange multiplier example from Section 1.4. Plot the augmented Lagrangian form of the objective function

$$(x-1)^2 + 1 + \lambda x$$

for several values of $\lambda > 2$, $\lambda < 2$, and $\lambda = 2$. Include the constraint $x \leq 0$ on the plot. Which value of $\lambda \geq 0$ maximizes the augmented function?

Exercise 1–12: Prove that a function is convex if and only if its epigraph is convex.

Exercise 1–13: Prove that for 3 points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ then $\alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \alpha_3\mathbf{x}_3$ forms a convex triangle in $\mathbb{R}^n$ if $\alpha_1 + \alpha_2 + \alpha_3 = 1$.

Exercise 1–14: Prove that for a set of m points $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ then $\alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \cdots + \alpha_m\mathbf{x}_m$ forms a convex polytope in $\mathbb{R}^n$ if $\alpha_1 + \alpha_2 + \cdots + \alpha_m = 1$. Give an example where every point $\mathbf{x}_j$ is a vertex of the polytope, and give an example where some of the points are not vertices of the resulting polytope.

Exercise 1–15: Prove Jensen's inequality.

Exercise 1–16: Generalize Jensen's inequality to continuous integrals, as in Eq. (1.19).

Exercise 1–17: Show that if a set $\boldsymbol{\mathcal{S}} \subseteq \mathbb{R}^n$ is convex, then its projection onto any subset of of the coordinates of $\mathbb{R}^n$ is also convex.

Exercise 1–18: Show that if a set $\boldsymbol{\mathcal{S}} \subseteq \mathbb{R}^n$ is convex, then its affine projection onto any plane will also be convex.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 1–19: Show that if a set $\boldsymbol{\mathcal{S}} \subseteq \mathbb{R}^n$ is convex, then its projection onto any affine space is convex.

Exercise 1–20: Given two points $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, show that the set of points closer to $\mathbf{a}$ than $\mathbf{b}$ is a convex half-plane:

$$\boldsymbol{\mathcal{S}} = \{\mathbf{x} \,|\, \|\mathbf{x} - \mathbf{a}\| \leq \|\mathbf{x} - \mathbf{b}\|\}.$$

Exercise 1–21: [BV 3.1] Given a convex function $f(x) : \mathbb{R} \to \mathbb{R}$, show that

$$\frac{f(x) - f(a)}{x - a} \leq \frac{f(b) - f(a)}{b - a} \leq \frac{f(b) - f(x)}{b - x}$$

for $x \in (a, b)$. Draw a geometric sketch to demonstrate this.

Exercise 1–22: This exercise will explore the convex properties of matrices.

1. Show that the maximum eigenvalue $\lambda_{\max}(\mathbf{X})$ of a matrix $\mathbf{X}$ is a convex function.
2. Show that the log of the determinant of a matrix, $\log\det\mathbf{X} = \sum_i \log\lambda_i$ is a convex function for symmetric positive definite matrices $\mathbf{X}$.
3. Show that the trace, $\text{Tr}(\mathbf{X}) = \sum_i \lambda_i$ is a linear function, i.e. that $\text{Tr}(\mathbf{X} + \mathbf{Y}) = \text{Tr}(\mathbf{X}) + \text{Tr}(\mathbf{Y})$, and hence is a convex function.

Exercise 1–23: Prove that the Cartesian product of two convex sets is a convex set.

Exercise 1–24: Let C_1 and C_2 be convex. Then their Minkowski sum $C_1 + C_2 = \{x + y | x \in C_1, y \in C_2\}$ is also convex. Prove this and give some examples.

Exercise 1–25: Prove the following:

1. If $f(\mathbf{x})$ is convex, then $f(\mathbf{x}) + k$ is convex for any constant $k \in \mathbb{R}$.
2. If $f(\mathbf{x})$ is convex, then $\alpha f(\mathbf{x})$ is convex for any constant $alpha \in \mathbb{R}$.
3. If $f(\mathbf{x})$ is convex, then $f(\mathbf{A}\mathbf{x} + \mathbf{b})$ is convex for any affine transformation $\mathbf{A}\mathbf{x} + \mathbf{b}$. If the original domain is $\boldsymbol{\mathcal{D}}$, what is the new domain?

Exercise 1–26: Show that the triangle inequality holds for the following four norms:

1. $f(\mathbf{x}) = \|\mathbf{x}\|_1$
2. $f(\mathbf{x}) = \|\mathbf{x}\|_2$
3. $f(\mathbf{x}) = \|\mathbf{x}\|_2^2$
4. $f(\mathbf{x}) = \|\mathbf{x}\|_\infty$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 1–27: Compute the gradient of the four norms in the previous problem in two-dimensions and visualize in the x_1-x_2 plane.

Exercise 1–28: For each of the affine transformations $\mathbf{T}$ (scaling, shear, rotation, and translation), show that they preserve the convexity of a set $\boldsymbol{\mathcal{C}}$. In particular

$$\mathbf{T}(\boldsymbol{\mathcal{C}}) = \{\mathbf{T}\mathbf{x} \,|\, \mathbf{x} \in \boldsymbol{\mathcal{C}}\}$$

is convex.

Exercise 1–29: Consider two convex sets in 2D: a closed unit disk and a closed unit square, both centered at the origin. Show by hand how the affine transformations (scaling, shear, rotation, and translation) change these sets. Then write a code to apply these transformations and plot the resulting set.

Exercise 1–30: Show that all of the affine transformations (scaling, shear, rotation, and translation) are invertible.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 2

Gradient Based Optimization

Gradient descent is one of the most popular and versatile techniques in all of optimization, and stochastic gradient descent is the backbone of modern machine learning training. In this chapter we will introduce leading gradient-based methods and variants, and discuss scalability, stability, and performance on several numerical examples.

A local minimum of a smooth, unconstrained objective function $f(\mathbf{x})$ will be a point $\mathbf{x}^*$ with zero gradient, $\nabla f(\mathbf{x}^*) = \mathbf{0}$, and positive semi-definite Hessian, $\nabla^2 f(\mathbf{x}^*) \geq \mathbf{0}$. Thus, finding points $\mathbf{x}$ where $\nabla f(\mathbf{x}) = \mathbf{0}$ is a central challenge in optimization. Gradient-based approaches use information about the gradient of the objective function, $\nabla f(\mathbf{x})$, to iteratively search for local minima $\mathbf{x}^*$. For example, the function f may be the loss function of a neural network, as in Chapter 10, and we will use stochastic gradient descent to find parameters $\mathbf{x}$ that locally minimize this function.

In some cases, it is possible to directly solve for points where $\nabla f(\mathbf{x}) = \mathbf{0}$, as in the least squares problem where $f(\mathbf{x})$ is a quadratic function; this will be explored thoroughly in Chapter 4. However, the function $\nabla f(\mathbf{x})$ is generally too complex to solve for its roots explicitly. Instead, we often use *iterative methods*, such as gradient descent and Newton's method, to obtain a sequence of points $\mathbf{x}_k$ that eventually converge towards the local minimum $\mathbf{x}^*$. Specifically, we seek an algorithm

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k) \tag{2.1}$$

so that starting with an initial guess $\mathbf{x}_0$, we may iterate through $\mathbf{F}$:

$$\mathbf{x}_0 \xrightarrow{\mathbf{F}} \mathbf{x}_1 \xrightarrow{\mathbf{F}} \mathbf{x}_2 \xrightarrow{\mathbf{F}} \cdots \xrightarrow{\mathbf{F}} \mathbf{x}_k \xrightarrow{\mathbf{F}} \cdots$$

The goal is to iteratively decrease the objective function towards a local minimum $\mathbf{x}^*$:

$$f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq f(\mathbf{x}_2) \geq \cdots f(\mathbf{x}_k) \geq \cdots f(\mathbf{x}^*). \tag{2.2}$$

We would like for this algorithm to converge quickly, ideally monotonically, and with minimal computational complexity.

This chapter will explore several of the leading iterative, gradient-based algorithms in optimization. In each case, we will discuss convergence rate, stability, computational complexity, and scaling to higher-dimensional problems. In all iterative methods, the resulting solution is only locally optimal, unless the problem is convex. We also stress the importance of the initial guess $\mathbf{x}_0$ for non-convex problems, as this will determine which local

DRAFT

minima the algorithm converges to and how fast it converges. First-order methods use first derivative information, e.g., the gradient ∇f, to build locally linear approximations to f, which are then used to develop iterative algorithms. Gradient descent, and its extension to high-dimensional optimization problems in neural network training, stochastic gradient descent, are two popular first-order methods we will explore.

Here we will mostly consider unconstrained problems, although gradient-based approaches may also be quite useful in constrained optimization problems. Similarly, one benefit of gradient-based optimization is that very little structure is assumed, other than the existence of derivatives.

Throughout this chapter, it will help to appreciate algorithms that use the gradient by imagining what we might do if we don't have access to the gradient; usually this involves some intelligent form of *perturb and observe*, which can be quite slow to converge, especially in higher dimensions. In contrast, gradient descent scales to very large systems, and increasingly we are able to approximate derivatives of very complex functions, such as a neural network training objective function, using modern automatic differentiation and stochastic gradient descent, which will be discussed more in Chapter 10.

We will also see that having access to higher derivatives of the objective function will allow us to develop more accurate iterative algorithms, such as Newton's method. This is analogous to other numerical methods, where using higher-order Taylor approximations yield more accurate schemes. Second-order methods use the second derivative, e.g., the Hessian $\nabla^2 f$, to develop more accurate iteration schemes that converge in fewer iterations, but at the cost of more computational complexity per iteration. Newton's method, and its related Quasi-Newton methods are second-order algorithms discussed here. Many of these algorithms are adapted for nonlinear least squares problems, discussed in Chapter 4, and for machine learning training, discussed in Chapter 10.

2.1 Gradient Descent

Gradient descent seeks to find a local minima of an objective function $f(\mathbf{x})$ by taking iterative steps in the negative gradient direction. Recall that the gradient of a function $\nabla f(\mathbf{x})$ determines the direction of locally steepest ascent, and so the negative gradient $-\nabla f(\mathbf{x})$ is the direction of locally steepest descent. The gradient descent algorithm is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k), \tag{2.3}$$

where γ is the step size, also referred to as the learning rate in machine learning. This is illustrated schematically in Fig. 2.1.

Gradient descent is based on a first-order Taylor series approximation of $f(\mathbf{x})$:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \underbrace{(\mathbf{x} - \mathbf{x}_0)}_{\Delta \mathbf{x}}. \tag{2.4}$$

This perspective will be useful when determining the stability and convergence of gradient descent algorithms.

The most immediate question is how to choose a good step size γ. If γ is too large, the iteration will be unstable or barely stable (see Fig. 2.2 for an illustration). Specifically

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

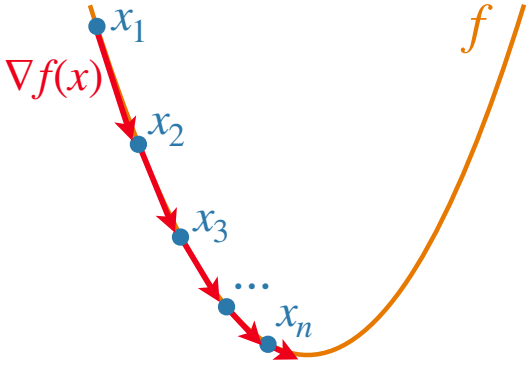


Figure 2.1: Schematic illustrating gradient descent for a simple one-dimensional, quadratic objective function. As the iterate $\mathbf{x}_k$ gets closer to the minimum of f, the gradient becomes smaller (i.e., the slope of the gradient becomes closer to 0), so the gradient descent steps also become smaller. Thus, the algorithm takes larger steps when further from the minimum.

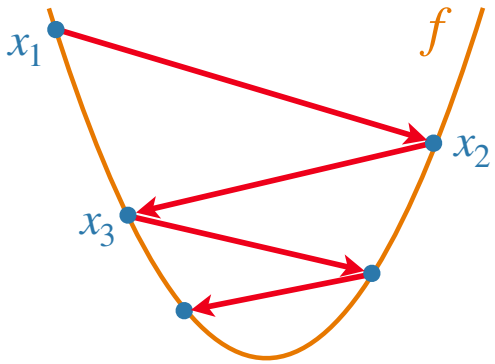


Figure 2.2: Gradient descent with too-large of a step size will oscillate around the minima, or worse, it will diverge. Here we are not plotting the actual gradient, which is always tangent to the function, but instead the actual steps $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k)$. If γ is too large, then the step will overshoot the minimum, causing oscillations, or potentially instability.

the gradient step may overshoot the minimum, causing oscillations, or even divergence. Similarly, if γ is too small, the iteration will converge very slowly, making it ineffective for problems where it is expensive to evaluate f. If we have information about the second derivative $\nabla^2 f$, we can use this to determine a better γ, as will be discussed in the context of Newton's method in Section 2.4.

There are several approaches to determine the step size γ:

- Fixed γ: A fixed stepsize γ may be good for a convex objective function f with a well-conditioned Hessian $\nabla^2 f$. Later we will explore a simple example with quadratic objective to see the effect of the condition number of $\nabla^2 f$ on a fixed-step gradient

Copyright © July 15, 2025 Brunton. All Rights Reserved.

descent algorithm. Fixed step schemes are simple, but may require tuning for adequate convergence.

- Decaying γ: It may be helpful to have a step size γ that slowly decays with the iteration k, so that it converges quickly at first and then more slowly. A simple decaying step size formula is

$$\gamma_k = \frac{\gamma_0}{1 + \beta k}, \quad \beta > 0. \tag{2.5}$$

  This may seem counter intuitive, that a step size should be smaller for a large gradient (i.e., earlier in the iteration where f is steeper) to avoid overshooting, and larger for a small gradient (i.e., later in the iteration, where f is flatter) to accelerate convergence. However, for stochastic gradient descent, a decaying γ acts to *anneal* the algorithm, letting it take large exploratory steps early, and cooling to smaller exploitative steps later in the iteration. Annealing will be discussed in Chapter 6.

- Line search: A line search for γ is effective for smooth, deterministic objective functions f. However, it may be computationally costly, especially when it is expensive to evaluate f. A line search uses the gradient direction and then runs a second optimization over the step size γ in that direction to find the best step size according to some criteria. This secondary line-search optimization has the form:

$$\gamma_k = \underset{\gamma}{\operatorname{argmin}} f(\mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k)). \tag{2.6}$$

  We often use a line search in Newton's method, since each iteration is more expensive, justifying the additional cost of a line search to improve convergence.

- Adaptive γ: There are several adaptive methods, and this is where we will spend most of our emphasis because of their favorable properties.
  - Momentum methods: It is possible to use physical intuition to imbue the step size with *momentum*, so that it is less prone to getting stuck in saddle points and locally flat regions. This is such an important concept, used widely in machine learning, that it will be discussed further in Section 2.2.
  - RMSProp: This algorithm scales the step size with a moving average of past squared gradients, regularizing the step. This is particularly effective for ill-conditioned problems.
  - Adaptive moment estimation (ADAM): This approach combines momentum methods and RMSProp and is the workhorse algorithm to train neural networks with stochastic gradient descent due to its robustness. This will be discussed in more detail in Section 2.3.

- Learning rate schedules: Various other approaches exist to modify the learning rate in different stages of the optimization. This is again closely related to annealing methods in non-convex optimization, including exponential decay.

It is important to emphasize that gradient descent will only converge to the nearest local minima (see Fig. 2.3), so for non-convex objective functions with multiple local minima, it is important to start with a good initialization.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

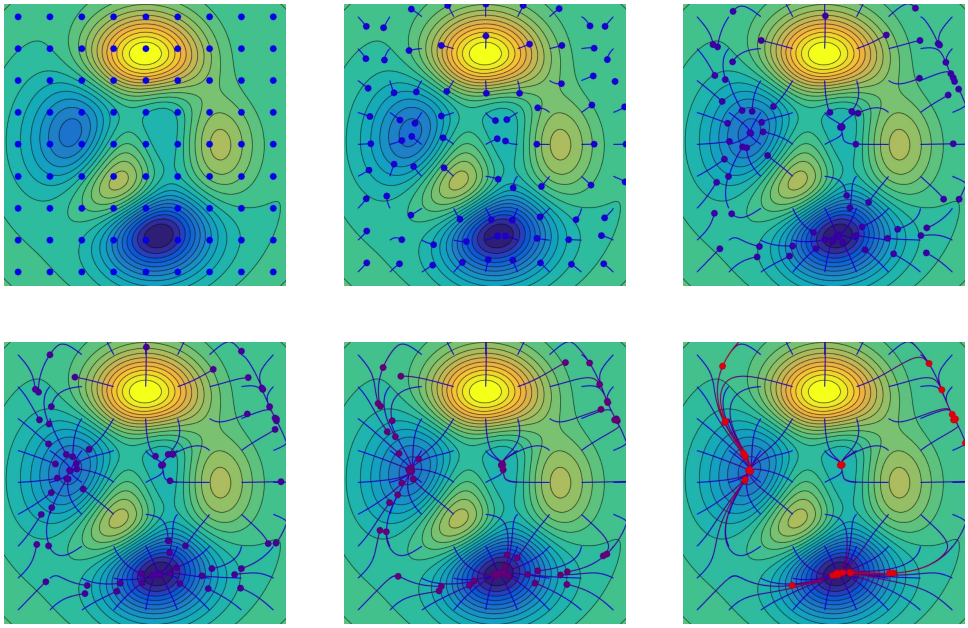


Figure 2.3: Illustration of gradient descent on a non-convex objective function with multiple local minima. Points descend to their nearest local minimima. Reproduced from Wikipedia (https://en.wikipedia.org/wiki/File:Gradient_Descent_in_2D.webm); author Gabriel Peyré.

DRAFT

Numerical Examples

In our first example, we will apply gradient descent to find the minimum of a simple quadratic cost function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} \qquad \text{where} \qquad \mathbf{Q} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \tag{2.7}$$

with positive semidefinite $\mathbf{Q}$. This problem actually has a closed-form solution, which will be discussed more in Chapter 4. The minimum value is at the origin, $\mathbf{x}^* = \mathbf{0}$. This will provide a simple example to test the algorithm, where we know the answer.

Figure 2.4 shows the gradient descent iteration for three step sizes: $\gamma = \{0.01, 0.1, 0, 543\}$. We can clearly see the effects of γ being too small or too large, namely either slow convergence or nearly unstable oscillating convergence, respectively. A simple starter gradient descent code is provided in Code 2.1.

Code 2.1: Gradient descent code for a quadratic objective function.

```
# Define the quadratic objective function with its gradient
def quadraticObjective(X, A, b, c):
    return 0.5 * X.T @ A @ X + b.T @ X + c

def gradientDescent(A, b, c, x0, step_size, max_iter):
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

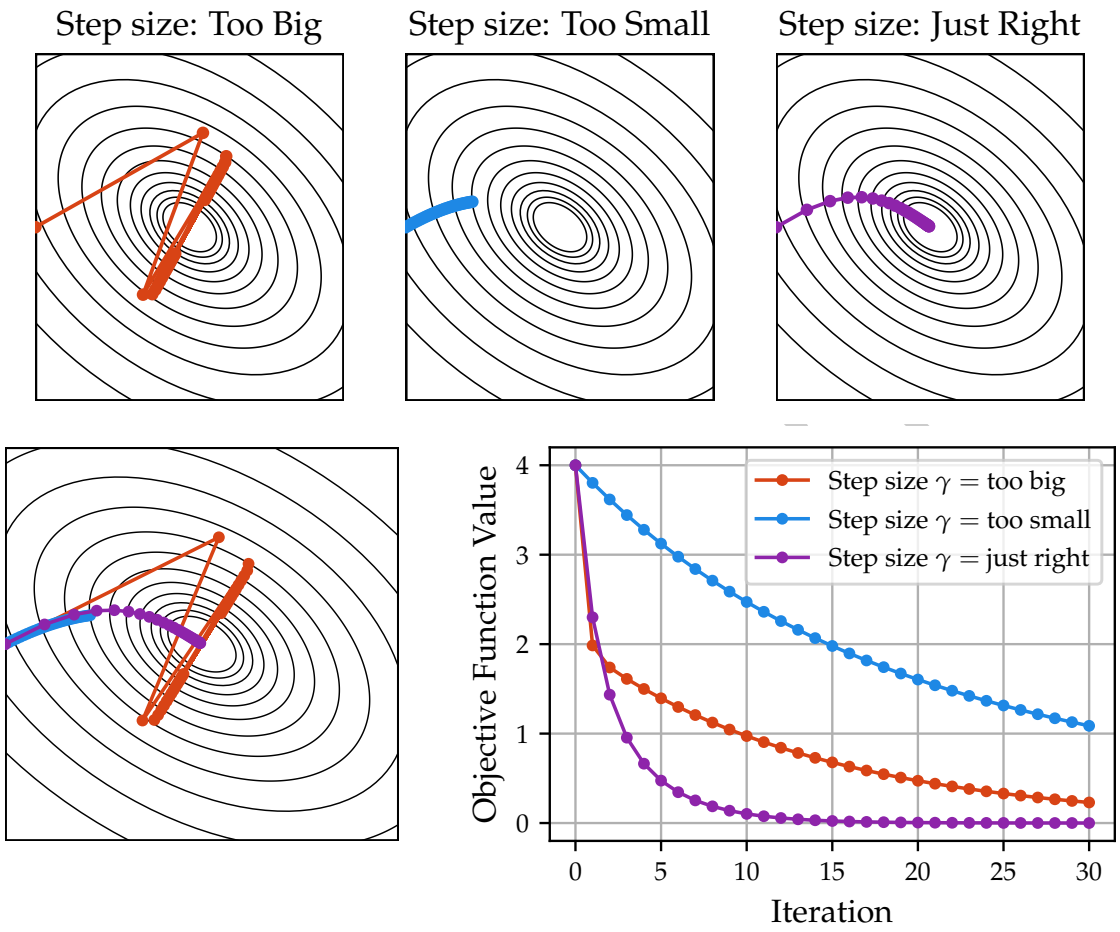


Figure 2.4: Gradient descent on a simple quadratic objective function with three step sizes: $\gamma \in \{0.01, 0.1, 0.543\}$. The step size $\gamma = 0.1$ converges rapidly and without oscillations. A smaller size $\gamma = 0.01$ converges too slowly, and a larger step size $\gamma = 0.543$ is nearly unstable, oscillating as it slowly converges. The top plot shows each trajectory individually, superimposed on the level sets of the objective function. The bottom left plot shows all step sizes together for comparison. The bottom right plot shows the objective function versus iteration number, which is how we typically visualize convergence of an algorithm.

```
    xvals = [np.array(x0)] # Initialize with x0, f(x0)
    fvals = [quadraticObjective(np.array(x0), A, b, c)]
    x = np.array(x0, dtype=float)

    for _ in range(max_iter):
        grad = A @ x + b  # Gradient of objective function
        x = x - step_size * grad
        xvals.append(x.copy())
        fvals.append(quadraticObjective(x, A, b, c))
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

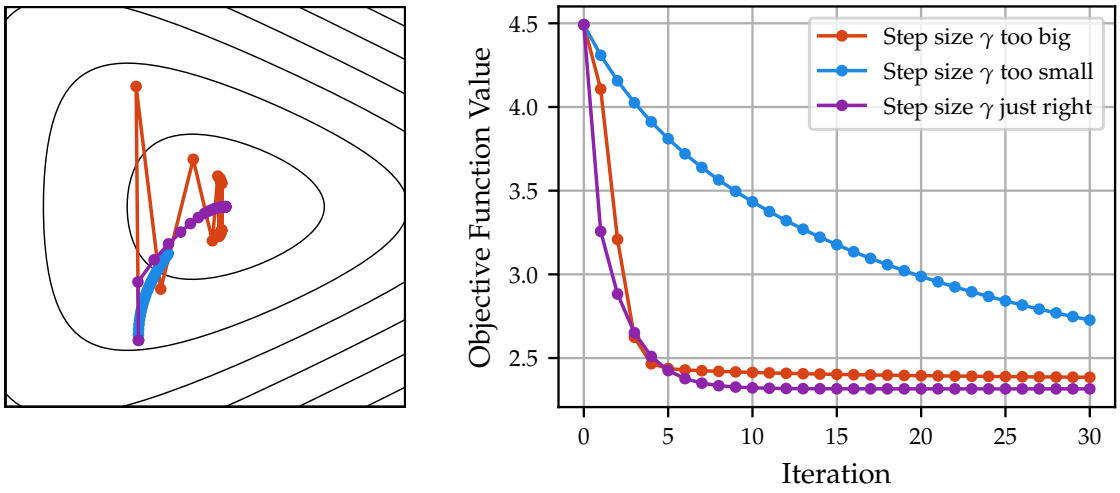


Figure 2.5: Gradient descent on a non-quadratic objective function with three step sizes.

```
    return np.array(xvals), np.array(fvals)
```

It is possible to initialize the problem and run the gradient descent optimization using the following:

```
A = np.array([[2, 1], [1, 3]]) # Positive-definite matrix
b = np.array([0, 0])           # Linear term
c = 0                          # Constant term
x0 = np.array([-2.0, 0.0])     # Starting point
step_size = 0.1                # Learning rate
N = 30                         # Maximum iterations

xvals, fvals = gradientDescent(A, b, c, x0, step_size, N)
```

As a more challenging example, inspired by Boyd and Vanderberghe [48], we explore gradient descent on a non-quadratic objective function:

$$f(\mathbf{x}) = e^{x_1+2x_2-.2} + e^{x_1-2x_2-.2} + e^{-x_1-.2}. \tag{2.8}$$

The gradient of this function is

$$\nabla f(\mathbf{x}) = \begin{bmatrix} e^{x_1+2x_2-.2} + e^{x_1-2x_2-.2} - e^{-x_1-.2} \\ 2e^{x_1+2x_2-.2} - 2e^{x_1-2x_2-.2} \end{bmatrix}.$$

The performance of gradient descent with various step sizes is shown in Fig. 2.5. Similar behavior is observed.

Gradient Flows

The gradient descent algorithm in Eq. (2.3) may be viewed as a simple forward Euler integration scheme with timestep $\Delta t = \gamma$ for the ordinary differential equation

$$\dot{\mathbf{x}}(t) = -\nabla f(\mathbf{x}(t)). \tag{2.9}$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

This vector field $-\nabla f(\mathbf{x})$ is known as a *gradient flow*. It is a conservative, irrotational vector field, and it is incompressible if $\nabla^2 f = 0$.

Recall that the forward Euler integration scheme for a vector field $\dot{\mathbf{x}} = \mathbf{v}(\mathbf{x})$ is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \mathbf{v}(\mathbf{x}_k).$$

It will be left as an exercise to the reader to show that the simple gradient descent algorithm is a forward Euler integration of the gradient flow equation with $\Delta t = \gamma$. A natural question is whether or not it is possible to devise a gradient descent iteration based on a better numerical integration scheme for this gradient flow, such as a higher-order Runge-Kutta integrator; this is also an exercise for the reader.

Convergence of Gradient Descent

If f is convex and Lipschitz continuous with Lipschitz constant L[1], then gradient descent with a step size $\gamma = 1/L$ will have the following convergence

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{L}{2k}\|\mathbf{x}_0 - \mathbf{x}^*\|^2. \tag{2.10}$$

Thus the convergence rate is $\mathcal{O}(1/k)$, i.e., sublinear in k.

If f is strongly convex, this convergence rate can be improved. A function is called μ-strongly convex if

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|^2 \tag{2.11}$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\mu > 0$. In this case, the convergence rate is:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^k \|\mathbf{x}_0 - \mathbf{x}^*\|^2. \tag{2.12}$$

Stability of Gradient Descent

Here we will explore gradient descent on a simple quadratic objective function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c. \tag{2.13}$$

The gradient is given by

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{2.14}$$

and the Hessian is

$$\mathbf{H} = \nabla^2 f(\mathbf{x}) = \mathbf{A}. \tag{2.15}$$

In many cases, we can solve for the optimum solution $\mathbf{x}^* = -\mathbf{A}^{-1}\mathbf{b}$ directly (see Chapter 4), so that $\nabla f(\mathbf{x}^*) = \mathbf{0}$. However, there are several classes of quadratic problems where we will still use gradient descent:

[1]Lipschitz continuity of a function f is a stronger condition than f being continuous, but weaker than f being differentiable. A function f is Lipschitz continuous with Lipschitz constant L if $|f(\mathbf{x}_1) - f(\mathbf{x}_2)| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|$ for all $\mathbf{x}_1, \mathbf{x}_2$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- Large $\mathbf{A}$: When the dimension of $\mathbf{x} \in \mathbb{R}^n$ is large, for example n is in the millions or even billions, then $\mathbf{A}^{-1} \in \mathbb{R}^{n \times n}$ is intractable to compute and it may not even be possible to store $\mathbf{A}$. Even in this case, variants of gradient descent, such as the conjugate gradient method, will often apply.
- Sparse $\mathbf{A}$: When $\mathbf{A}$ is a large, sparse matrix, it may be faster to use iterative gradient-based methods than to directly invert $\mathbf{A}$. These matrices occur frequently in physics applications, such as fluid mechanics, where interactions are local, so terms like diffusion and convection typically lead to very sparse $\mathbf{A}$ matrices.
- Ill-conditioned $\mathbf{A}$: In this case, it is often difficult to use standard inversion techniques, and instead we use preconditioners followed by iterative descent.

In any case, even when the problem is small and we could solve it directly, we learn by exploring gradient descent on a simple problem where we know the answer.

We will now examine the stability properties of gradient descent and how they depend on the step size γ and the eigenvalues of $\mathbf{A}$. This will be closely related to the stability of numerical integrators, such as forward Euler integration for ordinary differential equations $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$. Again, even though we know the answer analytically as $\mathbf{x}(t) = \exp(\mathbf{A}t)\mathbf{x}(0)$, we learn a great deal by analyzing the stability properties of various integrators on simple systems where we know the answer.

For a quadratic objective function, the gradient descent formula

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k)$$

becomes

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \left(\mathbf{A}\mathbf{x}_k - \mathbf{b}\right) \tag{2.16a}$$

$$= \left(\mathbf{I} - \gamma \mathbf{A}\right) \mathbf{x}_k - \gamma \mathbf{b}. \tag{2.16b}$$

Again, $\mathbf{A}$ is the Hessian of $f(\mathbf{x})$. Let λ_j be the eigenvalues of $\mathbf{A}$, which represent the curvature of f in different directions. The maximum curvature is given by $\lambda_{\max}$ and the minimum curvature is given by $\lambda_{\min}$.

The gradient descent algorithm is stable if all eigenvalue of $\mathbf{I} - \gamma \mathbf{A}$ are stable. For the discrete-time system in Eq. (2.16b), stability means that all eigenvalues of $\mathbf{I} - \gamma \mathbf{A}$ must have magnitude less than one. This guarantees stability of the iteration:

$$\mathbf{x}_k \to \mathbf{x}^* = -\mathbf{A}^{-1}\mathbf{b}.$$

For $\mathbf{A}$ symmetric, then the eigenvalues of of $\mathbf{I} - \gamma \mathbf{A}$ are $1 - \gamma \lambda_j$ where $\lambda_j \in \text{eigs}(\mathbf{A})$. $\mathbf{A}$ is often symmetric for many real quadratic optimization problems, which we will see in least squares in Chapter 4.

More generally, if $\mathbf{A}$ is diagonalizable, there exists an invertible matrix of eigenvectors $\mathbf{T}$ so that $\mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \mathbf{D}$, where $\mathbf{D}$ is a diagonal matrix. In this case

$$\mathbf{T}^{-1} \left(\mathbf{I} - \gamma \mathbf{A}\right) \mathbf{T} = \mathbf{I} - \gamma \mathbf{D}$$

which is diagonal with eigenvalues $1 - \gamma \lambda_j$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

For stability we require

$$|1 - \gamma\lambda_j| < 1 \quad \forall j.$$

This may be rewritten as

$$-1 < 1 - \gamma\lambda_j < 1 \quad \implies \quad -2 < -\gamma\lambda_j < 0 \quad \implies \quad 0 < \gamma < \frac{2}{\lambda_{\max}}.$$

Thus, this establishes a range of stable step sizes γ.

Optimal Step Size

Now we turn to the problem of determining an optimal step size γ for fastest convergence of the gradient descent algorithm.

The optimal step size for gradient descent with a quadratic objective function is given by

$$\gamma_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}},$$

which arises from the spectral properties of the Hessian matrix $\mathbf{A}$ of a quadratic objective function. This choice minimizes the convergence rate by balancing the contributions from the smallest and largest eigenvalues of $\mathbf{A}$.

The convergence rate is determined by the *spectral radius* of $(\mathbf{I} - \gamma\mathbf{A})$, which is the largest absolute eigenvalue:

$$\rho(\mathbf{I} - \gamma\mathbf{A}) = \max_i |1 - \gamma\lambda_i|.$$

To minimize the convergence rate, we choose γ such that $\rho(\mathbf{I} - \gamma\mathbf{A})$ is as small as possible.

The eigenvalues of $(\mathbf{I} - \gamma\mathbf{A})$ are symmetric about the point $1 - \gamma\lambda_{\text{mid}}$, where:

$$\lambda_{\text{mid}} = \frac{\lambda_{\min} + \lambda_{\max}}{2}. \tag{2.17}$$

To minimize the spectral radius, the eigenvalues are distributed evenly around zero:

$$\mu_{\max} = -\mu_{\min}.$$

Thus we set $|1 - \gamma\lambda_{\min}| = |1 - \gamma\lambda_{\max}|$ for an optimal γ^*:

$$1 - \gamma^*\lambda_{\min} = -(1 - \gamma^*\lambda_{\max})$$

and solve for γ^*:

$$\gamma^* = \frac{2}{\lambda_{\min} + \lambda_{\max}}.$$

This optimum step size γ^* minimizes the spectral radius of $(\mathbf{I} - \gamma\mathbf{A})$, leading to the fastest convergence for the quadratic problem. The best possible performance is limited by the ratio of eigenvalues $\lambda_{\max}/\lambda_{\min}$, which is closely related to the condition number, which will be discussed more in Chapter 4. A larger ratio of eigenvalues will lead to slower convergence.

As a simple exercise, compute the maximum step size $\gamma_{\max}$ and optimal step size γ^* for the quadratic example in Fig. 2.4. You can then test gradient descent with a γ just above and below the maximum stability boundary and also the optimal γ^*.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

2.2 Momentum Methods

Gradient descent struggles near saddle points of non-convex objective functions, where the gradient may be negligibly small, causing the convergence to grind to a halt. This issue is especially acute when using stochastic gradient descent for high-dimensional problems, such as training a neural network. The small gradients near a saddle, compounded with noisy estimates of the gradient, can cause the iteration algorithm to get stuck.

Momentum methods, and their variants, seek to address this issue by giving the optimization algorithm *inertia* to carry it through these flat saddle regions of small gradient. The standard gradient descent algorithm only uses the current instantaneous gradient for the next step in the iteration. Momentum methods integrate past gradient information to smooth out noisy gradient estimates and to provide inertia. This is a rather physical idea, similar to rolling a ball down the objective function landscape. If the ball gains momentum in areas of high gradient, it will have inertia to roll through flat saddle regions.

There are several approaches to momentum that are used in gradient descent, especially in the context of machine learning, that we will discuss here.

Heavy Ball

The classic momentum method [304] is the simplest modification to gradient descent to include momentum:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}). \tag{2.18}$$

The additional term $\beta(\mathbf{x}_k - \mathbf{x}_{k-1})$ introduces a history dependence, where $\mathbf{x}_k - \mathbf{x}_{k-1}$ may be interpreted as the *velocity* of the last iterative step.

We may formally introduce this velocity vector as

$$\mathbf{v}_{k+1} = \beta \mathbf{v}_k - \gamma \nabla f(\mathbf{x}_k)$$

so that the gradient descent update now takes the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k) + \beta_k \mathbf{v}_k \tag{2.19a}$$

$$\mathbf{v}_{k+1} = \beta \mathbf{v}_k - \gamma \nabla f(\mathbf{x}_k). \tag{2.19b}$$

This introduces a new hyper-parameter β, the decay rate for the velocity update. The velocity $\mathbf{v}$ is an exponentially discounted sum of past gradients.

The momentum method was introduced in machine learning in the 1980s but is based on earlier work by Polyak from the 1960s [279].

Differential Equation Interpretation of Momentum Method

Recall that it is possible to recast gradient descent as a discrete forward Euler approximation to the gradient flow ODE in Eq. (2.9) from Sec. 2.1. The heavy ball momentum method may be similarly derived from a second-order ODE:

$$m\ddot{\mathbf{x}}(t) = -\nabla f(\mathbf{x}(t)) - \delta \dot{\mathbf{x}}, \tag{2.20}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

where m is the mass of the particle and δ is the friction coefficient. Taking a simple central difference scheme approximation for $\ddot{\mathbf{x}}$ and an Euler approximation for $\dot{\mathbf{x}}$ for a time step of Δt yields

$$m\frac{\mathbf{x}_{k+1}-2\mathbf{x}_k+\mathbf{x}_{k-1}}{\Delta t^2} \approx -\nabla f(\mathbf{x}_k) - \delta\frac{\mathbf{x}_{k+1}-\mathbf{x}_k}{\Delta t}.$$

Multiplying both sides by Δt^2 and grouping terms, this simplifies as

$$\begin{aligned} m\left(\mathbf{x}_{k+1}-2\mathbf{x}_k+\mathbf{x}_{k-1}\right) &= -\Delta t^2\nabla f(\mathbf{x}_k) - d\Delta t\left(\mathbf{x}_{k+1}-\mathbf{x}_k\right) \\ \implies \quad (m+d\Delta t)\mathbf{x}_{k+1} &= -\Delta t^2\nabla f(\mathbf{x}_k) + (2m+\delta\Delta t)\mathbf{x}_k - m\mathbf{x}_{k-1} \\ &= -\Delta t^2\nabla f(\mathbf{x}_k) + (m+\delta\Delta t))\mathbf{x}_k + m\left(\mathbf{x}_k-\mathbf{x}_{k-1}\right) \\ \implies \quad \mathbf{x}_{k+1} &= \mathbf{x}_k - \gamma\nabla f(\mathbf{x}_k) + \beta\left(\mathbf{x}_k-\mathbf{x}_{k-1}\right) \end{aligned}$$

where γ and β are given by

$$\gamma = \frac{\Delta t^2}{m+\delta\Delta t} \qquad \beta = \frac{m}{m+\delta\Delta t}.$$

Nesterov's Accelerated Gradient Algorithm

Nesterov's accelerated gradient (NAG) algorithm [263, 363] is closely related to the classic momentum method, and has become an industry standard algorithm in machine learning. Nesterov's formulation is a relatively simple variation on Eq. (2.18):

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k - \gamma\nabla f(\mathbf{x}_k+\beta(\mathbf{x}_k-\mathbf{x}_{k-1})) + \beta(\mathbf{x}_k-\mathbf{x}_{k-1}) && \text{(2.21a)} \\ &= \mathbf{x}_k - \gamma\nabla f(\mathbf{x}_k+\beta\mathbf{v}_k) + \beta\mathbf{v}_k, && \text{(2.21b)} \end{aligned}$$

where $\mathbf{v}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$. The only difference is that the gradient is now evaluated at the point $\mathbf{y}_k = \mathbf{x}_k + \beta\mathbf{v}_k$ instead of $\mathbf{x}_k$. This is called a *look ahead*, and is very reminiscent of classic Runge-Kutta integration schemes in the numerical integration of ordinary differential equations.

We may formalize this into a two-step procedure, as in other momentum methods, in terms of the look ahead vector $\mathbf{y} = \mathbf{x} + \beta\mathbf{v}$:

$$\begin{aligned} \mathbf{y}_k &= \mathbf{x}_k + \beta(\mathbf{x}_k-\mathbf{x}_{k-1}) && \text{(2.22a)} \\ \mathbf{x}_{k+1} &= \mathbf{y}_k - \gamma\nabla f(\mathbf{y}_k). && \text{(2.22b)} \end{aligned}$$

Sutskever demonstrated how to relate NAG to classic momentum, in the following way. It is possible to modify the index on the look ahead variable and rewrite NAG as

$$\begin{aligned} \mathbf{y}_{k+1} &= \mathbf{x}_k + \beta_k\left(\mathbf{x}_k-\mathbf{x}_{k-1}\right) && \text{(2.23a)} \\ \mathbf{x}_{k+1} &= \mathbf{y}_{k+1} - \gamma\nabla f(\mathbf{y}_{k+1}) && \text{(2.23b)} \end{aligned}$$

which can be rearranged as

$$\begin{aligned} \mathbf{y}_{k+1} &= \mathbf{x}_k + \beta_k\mathbf{v}_k && \text{(2.24a)} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \beta_k\mathbf{v}_k - \gamma\nabla f(\mathbf{x}_k+\beta_k\mathbf{v}_k). && \text{(2.24b)} \end{aligned}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

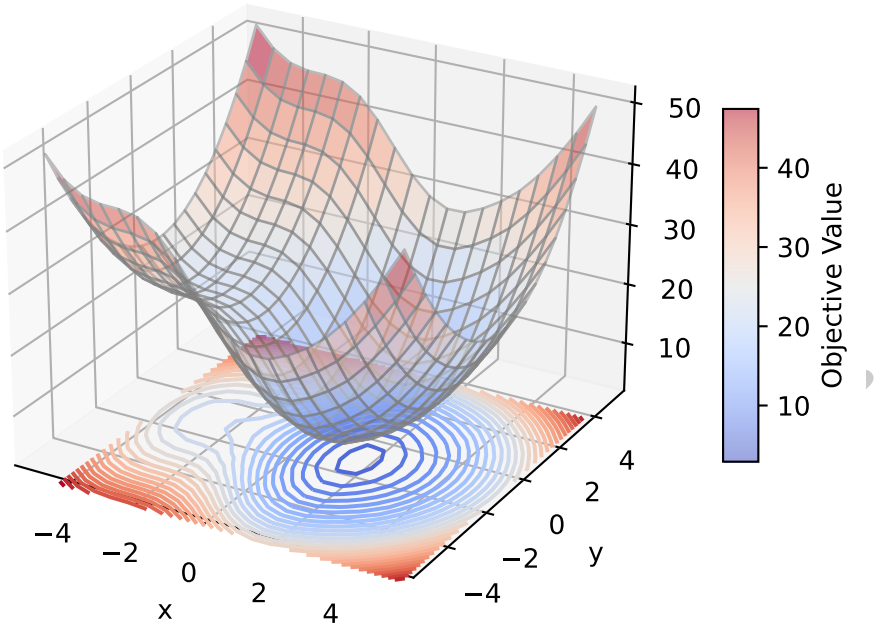


Figure 2.6: Surface plot and contour lines of objective function with saddle point and minima, given by Eq. (2.25).

Thus, NAG essentially reverses the order of classical momentum, so that we perform the momentum update first and then proceed to gradient descent.

Numerical Example

Gradient descent performs very poorly near saddle points, as the gradient nearly vanishes, causing painfully slow convergence. This is particularly acute in machine learning, where there may be many local minima separated by saddle points.

Momentum methods, and Nesterov's accelerated gradient (NAG) method in particular, address this challenge by adding momentum to the optimization point $\mathbf{x}_k$, allowing it to move through saddle regions with non-zero velocity.

This concept is illustrated for a simple objective function with a saddle, given by

$$f(\mathbf{x}) = 15e^{-.25(x_2)^2} + .5e^{-2(x-2)^2} + x^2 + y^2. \tag{2.25}$$

A three-dimensional surface plot of this objective over its contour map is shown in Fig. 2.6. Figure 2.7 shows a comparison of gradient descent with and without momentum on the objective function in Eq. (2.25) with a saddle point. It is clear that the method with momentum converges much more rapidly.

```
def nesterov_accelerated_gradient(start, grad_func, step_size,
    momentum, max_iter):
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

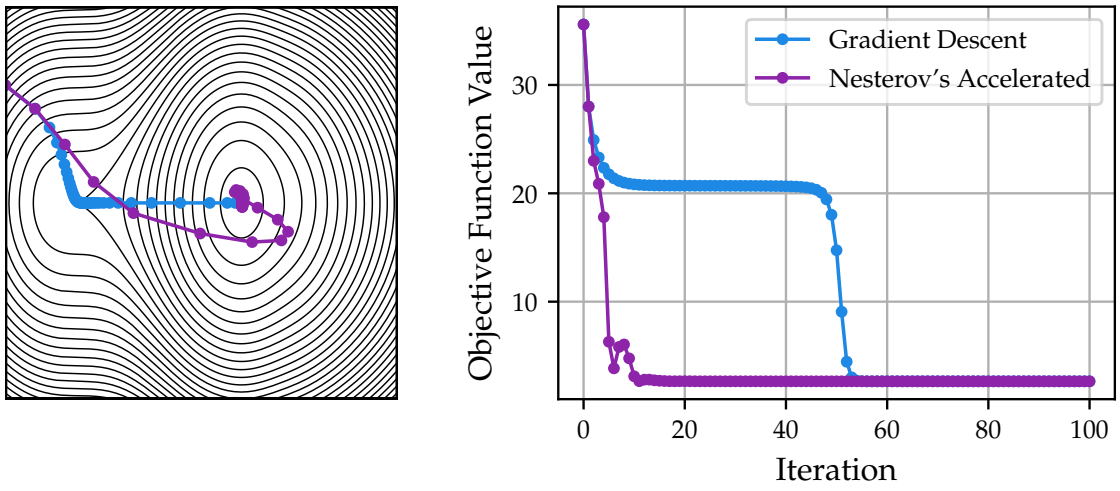


Figure 2.7: Comparison of simple gradient descent and Nesterov's accelerated method on an objective function with a saddle point.

```
    for _ in range(max_iter):
        y = x + momentum * velocity
        grad = grad_func(y[0], y[1])
        velocity = momentum * velocity - step_size * grad
        x = x + velocity
        trajectory.append(x.copy())
        values.append(double_well_potential(x[0], x[1]))

    return np.array(trajectory), np.array(values)
```

This is then initialized with the initial condition $(-5, 3)$ and run with a step size of $\gamma = 0.1$, a momentum coefficient $\beta = 0.9$ and a maximum number of iterations of 100.

Convergence of Nesterov's Method

Nesterov's achieves faster convergence than standard gradient descent. If f is convex and f is Lipschitz continuous with constant L, then the convergence of NAG is:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{2L}{(k+1)^2}\|\mathbf{x}_0 - \mathbf{x}^*\|^2. \tag{2.26}$$

Thus, the convergence rate is $\mathcal{O}(1/k^2)$, which is faster than the $\mathcal{O}(1/k)$ rate of standard gradient descent. This is a major improvement over the convergence of gradient descent.

If f is μ-strongly convex, then Nesterov's method achieves an accelerated linear convergence rate:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq 2\left(1 - \sqrt{\frac{\mu}{L}}\right)^k (f(\mathbf{x}_0) - f(\mathbf{x}^*)). \tag{2.27}$$

This shows a geometric decay rate with a significantly better contraction factor compared to gradient descent.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Differential Equation Interpretation of Nesterov's Method

Just as there is an intuitive physical interpretation of the simple momentum method, there is an interpretation of Nesterov's method as the limit of a continuous second-order ordinary differential equation [332]:

$$\ddot{\mathbf{x}} + \frac{3}{t}\dot{k} + \nabla f(\mathbf{x}) = 0 \tag{2.28}$$

with initial conditions $\mathbf{x}(0) = \mathbf{x}_0$ and $\dot{\mathbf{x}}(0) = \mathbf{0}$.

The time parameter is related to the step size as $t \approx k\sqrt{\gamma}$, where $\gamma \leq 1/L$ is the step size and L is the Lipschitz constant of ∇f.

AdaGrad and RMSProp

AdaGrad [114] is related to the classic momentum method, but instead of using the discounted sum of past gradients for the velocity $\mathbf{v}$, we use the sum of past gradients squared

$$\mathbf{v}_k = \mathbf{v}_{k-1} + (\nabla f(\mathbf{x}_k))^2 \tag{2.29}$$

resulting in the following iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\gamma}{\sqrt{\mathbf{v}_k} + \boldsymbol{\epsilon}} \nabla f(\mathbf{x}_k). \tag{2.30}$$

The small factor $\boldsymbol{\epsilon}$ is added to the denominator to avoid division by zero.

The notation in Eqs. (2.30) and (2.29) is quite sloppy, although this is how it is often presented. We will be more precise here. Technically the expression $(\nabla f(\mathbf{x}))^2$ doesn't make dimensional sense, since ∇f is a vector. Moreover, it doesn't make sense to divide by the square root of a vector. Instead we consider $(\nabla f(\mathbf{x}))^2$ to be a vector of the components of the gradient squared, so that $(\nabla f(\mathbf{x}))^2 = \begin{bmatrix}(\partial f/\partial x_1)^2 & (\partial f/\partial x_2)^2 & \cdots & (\partial f/\partial x_n)^2\end{bmatrix}^T$. Similarly, when we divide by $\sqrt{\mathbf{v}}$, we are actually multiplying by a vector of elementwise inverses: $\mathbf{w} = \begin{bmatrix}1/\sqrt{v_1} & 1/\sqrt{v_2} & \cdots & 1/\sqrt{v_n}\end{bmatrix}^T$, where v_j is the j-th component of $\mathbf{v}$.

The running sum $\mathbf{v}_k$ has no discount, and so it only increases, causing the iterations to slow over time. RMSProp [169] addresses this issue by adding a decay factor to $\mathbf{v}_k$, similar to other momentum methods:

$$\mathbf{v}_{k+1} = \beta \mathbf{v}_k + (1-\beta)\left(\nabla f(\mathbf{x}_k)\right)^2. \tag{2.31}$$

These algorithms have been studied extensively, and there are many practical theoretical results, such as sufficient conditions for convergence [386].

2.3 Stochastic Gradient Descent and ADAM

Stochastic gradient descent (SGD) is a variant of gradient descent that scales to very high dimensional optimization problems, making it suitable for large-scale neural network training. ADAM (adaptive momentum estimation) is an optimization algorithm that combines SGD with momentum and adaptive learning rates, via AdaGrad and RMSProp, making

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

updates smoother and more responsive. Together, these provide the engine for training modern machine learning models, such as deep neural networks.

SGD makes the most sense in the context of machine learning, where we are optimizing the parameters $\boldsymbol{\theta}$ of a function $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$, such as a neural network, to best fit observed data. Specifically, given pairs of input–output data $\{\mathbf{x}_j, \mathbf{y}_j\}_{i=1}^N$, we seek to find the parameters $\boldsymbol{\theta}$ so that $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_j)$ best approximates $\mathbf{y}_j$, averaged over the data:

$$\min_{\boldsymbol{\theta}} \sum_{j=1}^{N} \|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_j) - \mathbf{y}_j\|^2. \tag{2.32}$$

This is a *nonlinear least squares* problem, which may be approached via iterative descent methods. Nonlinear least squares will be discussed in much more detail in Chapter 4 on least squares and Chapter 10 on machine learning.

It is possible to compute the gradient of the loss function $L(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}\|^2$ with respect to $\boldsymbol{\theta}$:

$$\nabla L = 2\nabla \mathbf{f}_{\boldsymbol{\theta}}^T \cdot (\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}). \tag{2.33}$$

The gradient of $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$ may be computed analytically, or approximated using the backpropagation algorithm in the case that it is a neural network. Backpropagation is essentially the chain rule applied to layers of a neural network using automatic differentiation [273, 24]. Automatic differentiation and backpropagation are the backbone of machine learning training and will be discussed more in Chapter 10.

Stochastic gradient descent approximates this gradient on a subset of the data, called a batch, instead of the entire dataset. Evaluating the loss function and gradients on random batches of data adds randomness to the process. This has several key benefits, including computational speed-ups, robustness to getting stuck in local minima, better exploration of the optimization landscape, and improved generalization of resulting models. Thus, stochastic gradient descent introduces a new hyper-parameter, namely the batch size.

The resulting iteration may be written as:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \nabla_{\boldsymbol{\theta}} L. \tag{2.34}$$

In principle, nothing looks to have changed from gradient descent, and all of the stochastic computations and backpropagation are hidden within the gradient $\nabla_{\boldsymbol{\theta}} L$. This is a feature, not a bug, as it means we can seamlessly extend SGD to include other gradient-based innovations like momentum and adaptive learning rates, such as ADAM.

ADAM: Adaptive Momentum Estimation

The adaptive momentum estimation (ADAM) [200] approach combines the best features of momentum methods and RMSProp, and it has become the workhorse optimization algorithm for deep learning. The original ADAM paper was the most cited paper in the world, in any field, for a decade, demonstrating its profound impact on the community.

ADAM uses a discounted running sum of squared gradients $\mathbf{v}$, as in RMSProp, and a momentum term $\mathbf{m}$ as in the heavy ball momentum methods. The gradient update is

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \frac{\hat{\mathbf{m}}_{k+1}}{\sqrt{\hat{\mathbf{v}}_{k+1}} + \boldsymbol{\epsilon}} \tag{2.35}$$

where the momentum and squared gradient terms are updated according to

$$\mathbf{m}_{k+1} = \beta_1 \mathbf{m}_k + (1 - \beta_1) \nabla f(\mathbf{x}_k) \tag{2.36a}$$

$$\mathbf{v}_{k+1} = \beta_2 \mathbf{v}_k + (1 - \beta_2) \left(\nabla f(\mathbf{x}_k)\right)^2 . \tag{2.36b}$$

These terms are normalized before their use in the gradient descent update:

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1 - \beta_1^{k-1}} \tag{2.37a}$$

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1 - \beta_2^{k-1}}. \tag{2.37b}$$

SGD is inherently used to compute the gradients in the update steps in ADAM. In principle, it is possible to combine other momentum and adaptive learning rate variants with SGD, such as a Nesterov accelerated SGD. However, ADAM has become the default industry standard for machine learning training because of its robustness and performance.

Example: SGD on a Linear Regression Problem

First we add noise to data from a linear model $y = \mathbf{A}\mathbf{x} + b$ with a two-dimensional state $\mathbf{x}$.

```
N = 2000  # number of samples
A = np.random.randn(N, 2)  # design matrix
true_x = np.array([2.0, -3.0])  # true slopes
true_b = 5.0

# True y = A * x + b plus some noise
noise_std = 0.5
noise = noise_std * np.random.randn(N)
y = A.dot(true_x) + true_b + noise
```

We need codes to compute the prediction $\hat{y}$ and the mean-squared error over the data set:

```
def predict(A, x, b):
    # A: shape (batch_size, 2)
    # x: shape (2,)
    # b: scalar
    return A.dot(x) + b # shape (batch_size,)

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred)**2)
```

The following code computes the approximate gradient on a batch of data

```
def batch_gradients(A_batch, y_batch, x, b):
    # A_batch, x, b: same shape
    # y_batch: shape (batch_size,)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
    batch_size = A_batch.shape[0]
    y_pred = predict(A_batch, x, b)
    residuals = (y_pred - y_batch)

    grad_x = (2.0 / batch_size) * A_batch.T.dot(residuals)
    grad_b = (2.0 / batch_size) * np.sum(residuals)
    return grad_x, grad_b
```

Now we define the three methods under comparison, standard (full-batch) gradient descent, mini-batch stochastic gradient descent, and stochastic gradient descent with $n = 1$. The learning rate is γ. The standard gradient descent is

```
def full_batch_gd(A, y, gamma=0.01, n_epochs=50):
    x = np.zeros(2)  # unknown parameter vector
    b = 0.0
    mse_history = []

    for epoch in range(n_epochs):
        grad_x, grad_b = batch_gradients(A, y, x, b)
        x -= gamma * grad_x
        b -= gamma * grad_b

        y_pred = predict(A, x, b)
        mse = mean_squared_error(y, y_pred)
        mse_history.append(mse)
    return x, b, mse_history
```

the mini-batch SGD gradient descent is

```
def mini_batch_sgd(A, y, batch_size=50, gamma=0.01, n_epochs=50):
    x = np.zeros(2)
    b = 0.0
    N = A.shape[0]
    mse_history = []

    for epoch in range(n_epochs):
        indices = np.random.permutation(N)     # Shuffle data
        for start in range(0, N, batch_size):
            end = start + batch_size
            batch_idx = indices[start:end]
            A_batch = A[batch_idx]
            y_batch = y[batch_idx]

            grad_x, grad_b = batch_gradients(A_batch, y_batch, x, b)
            x -= gamma * grad_x
            b -= gamma * grad_b

        # Check MSE after each epoch
        y_pred = predict(A, x, b)
        mse = mean_squared_error(y, y_pred)
        mse_history.append(mse)
    return x, b, mse_history
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

and the SGD with batch size 1 is

```
def sgd_batch1(A, y, gamma=0.001, n_epochs=50):
    x = np.zeros(2)
    b = 0.0
    N = A.shape[0]
    mse_history = []

    for epoch in range(n_epochs):
        indices = np.random.permutation(N)
        for i in indices:
            A_i = A[i:i+1]        # shape (1,2)
            y_i = y[i:i+1]        # shape (1,)
            grad_x, grad_b = batch_gradients(A_i, y_i, x, b)
            x -= gamma * grad_x
            b -= gamma * grad_b

        # MSE after epoch
        y_pred = predict(A, x, b)
        mse = mean_squared_error(y, y_pred)
        mse_history.append(mse)

    return x, b, mse_history
```

Historically "true" SGD used a batch size of 1, although now in practice we almost always use mini-batch SGD. In the SGD with batch size 1, we lower gamma to reduce the sensitivity to noise and risk of divergence.

Running all three methods, we compare the results in Fig. 2.8. The mini-batch and SGD algorithms converge in many fewer epochs than full-batch gradient descent. However, an epoch for each method implies a single pass through all data, so an epoch of SGD with batch size 1 involves 2000 single steps with a single data point.

The SGD and full-batch gradient descent may be viewed as the extremes of mini-batch stochastic gradient descent. Figures 2.9 and 2.10 compare computation time and scaling for different mini-batch sizes. It appears that a minibatch of 10 gives a balance of convergence and computation time.

Example: SGD on a Nonlinear Regression Problem

We now apply SGD to a nonlinear regression problem of fitting a polynomial of high-order to noisy data. SGD will be used in Chapter 10 for machine learning, which may be thought of as high-dimensional nonlinear regression. The following code generates a small amount of training and test data, $n = 50$, with relatively large noise for a polynomial $-10x^2 + 2x^4$. We will fit this polynomial in a space of 10 degree polynomials.

```
def true_poly(x):
    return (-10.0 * x**2) + (2.0 * x**4)

num_train = 50 # Small training set makes problem challenging
num_test  = 50
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

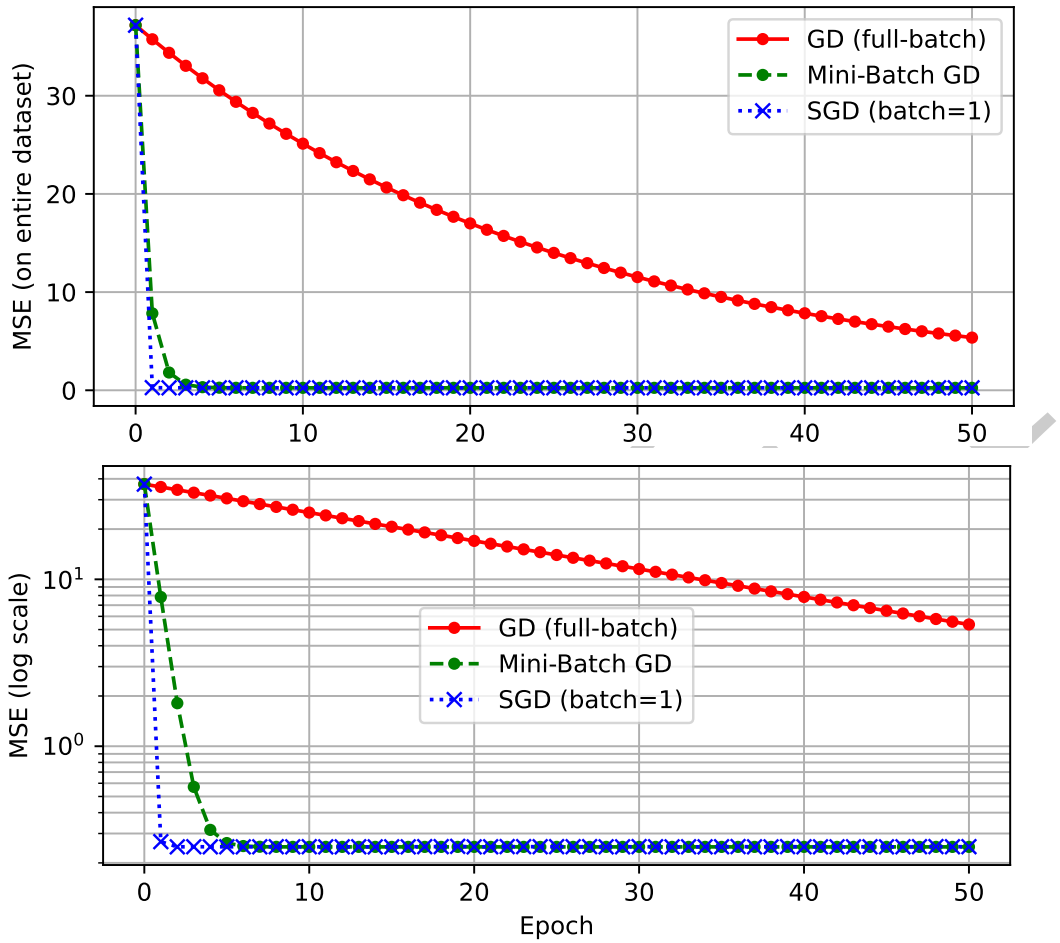


Figure 2.8: Convergence of various gradient descent methods for the linear least-squares regression problem. Methods include standard gradient descent using all data (red), mini-batch stochastic gradient descent with $n = 50$ data points (green), and stochastic gradient descent with $n = 1$ (blue). (top) Standard mean squared error on entire dataset and (bottom) MSE on log scale.

```
x_train = np.linspace(-2, 2, num_train) # data in range [-2,2]
y_train_true = true_poly(x_train)

noise_std = 2.0 # Add relatively large noise
noise_train = noise_std * np.random.randn(num_train)
y_train = y_train_true + noise_train

np.random.seed(2023) # Test data (different seed)
x_test = np.linspace(-2, 2, num_test)
y_test_true = true_poly(x_test)
noise_test = noise_std * np.random.randn(num_test)
y_test = y_test_true + noise_test
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

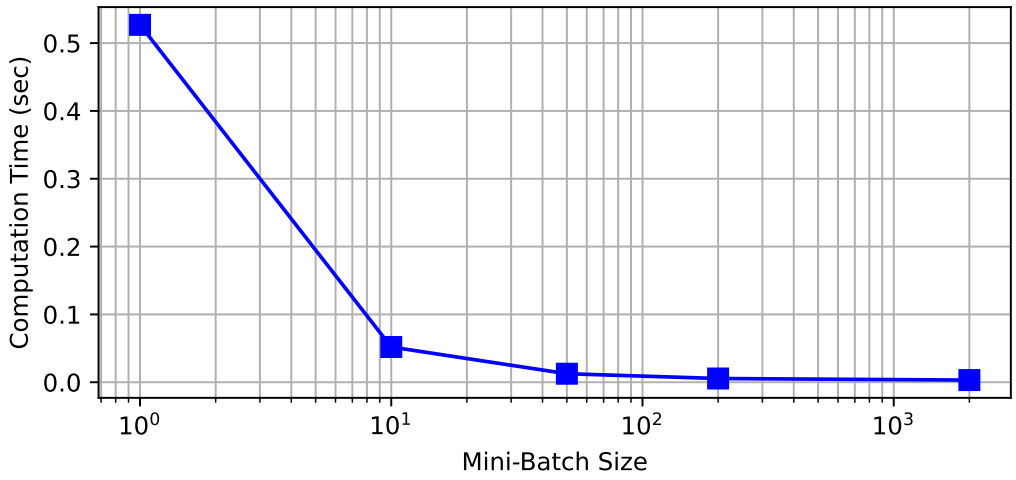


Figure 2.9: Computation time in seconds for various mini-batch sizes.

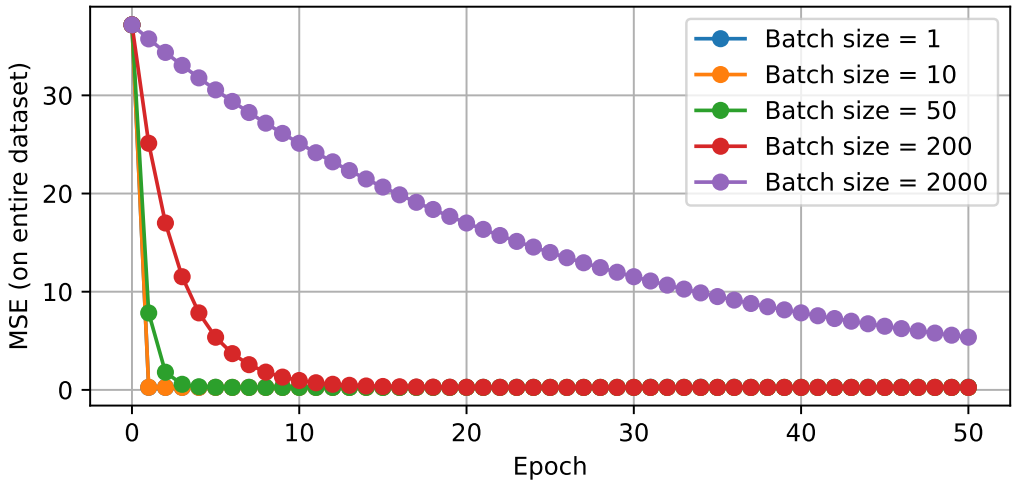


Figure 2.10: Convergence of batch gradient descent with different batch size for the linear least-squares regression problem.

We compute the 10 degree polynomial library

```
def create_poly_features(x, degree=10):
    # Return matrix with columns [x^0, x^1, ..., x^degree]
    return np.column_stack([x**i for i in range(degree+1)])

X_train = create_poly_features(x_train, degree=10)
X_test  = create_poly_features(x_test,  degree=10)
```

The gradient is computed on a mini-batch of data given by

```
def batch_gradient(X, y, w):
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
    # Compute gradient of MSE w.r.t. w: -2/N * X^T (y - Xw)
    N = X.shape[0]
    residuals = y - X.dot(w)
    grad = -(2.0 / N) * X.T.dot(residuals)
    return grad
```

The following code implements ADAM in PyTorch. First we import torch:

```
import torch
import torch.nn as nn
import torch.optim as optim
```

Next we convert our matrices and vectors to PyTorch tensors and define a linear model:

```
X_train_t = torch.tensor(X_train, dtype=torch.float32)
y_train_t = torch.tensor(y_train.reshape(-1, 1), dtype=torch.float32)
X_test_t  = torch.tensor(X_test, dtype=torch.float32)
y_test_t  = torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)

# Define a linear model (# features = # polynomial terms)
model = nn.Linear(X_train.shape[1], 1)

with torch.no_grad(): # Optional: initialize weights/bias to zero
    model.weight.zero_()
    model.bias.zero_()
```

Finally we set up the ADAM optimizer and training loop:

```
optimizer = optim.Adam(model.parameters(), gamma=1e-2)
criterion = nn.MSELoss()

n_epochs = 5000
adam_loss_history = []
for epoch in range(n_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_t)
    loss = criterion(outputs, y_train_t)
    loss.backward()
    optimizer.step()
    adam_loss_history.append(loss.item())
    if (epoch+1) % 200 == 0:
        print(f"Epoch {epoch+1}/{n_epochs}, Loss: {loss.item():.4f}")

model.eval()      # Evaluate ADAM model
with torch.no_grad():
    train_loss_adam = criterion(model(X_train_t), y_train_t).item()
    test_loss_adam  = criterion(model(X_test_t), y_test_t).item()

y_adam_train = model(X_train_t).detach().numpy().flatten()
```

Figure 2.12 shows the convergence of the methods vs. epoch, again with ADAM achieving the lowest MSE. ADAM outperforms full-batch gradient descent and SGD, as expected. Figure 2.11 shows the training and test results for the various methods.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

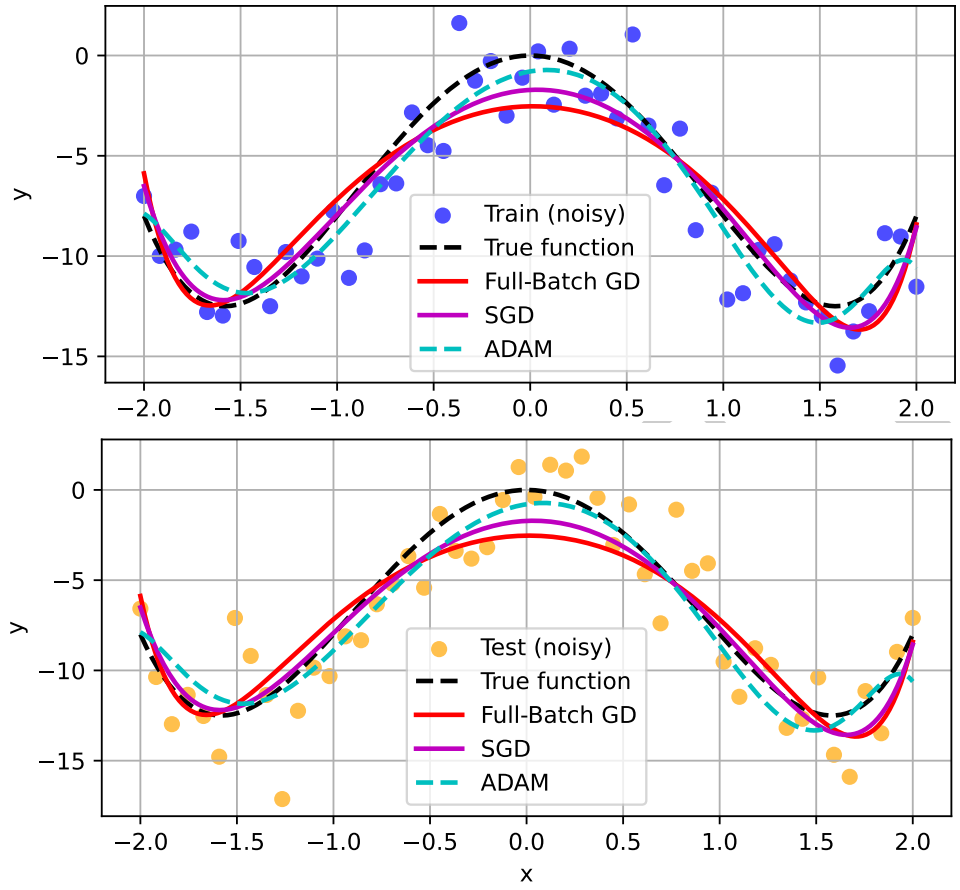


Figure 2.11: Comparison of standard gradient descent on all data, stochastic gradient descent, and SGD with ADAM on the training data (top) and testing data (bottom) for the nonlinear least-squares problem of fitting a polynomial.

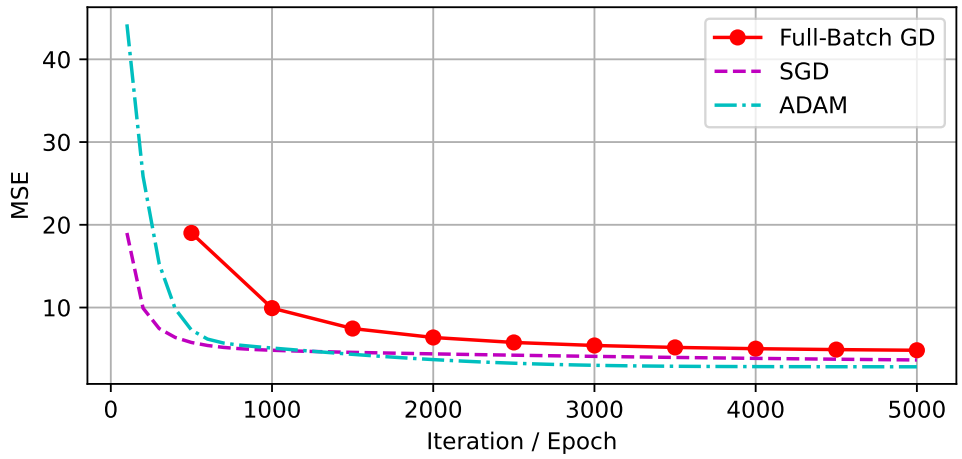


Figure 2.12: Convergence of mean square error for various methods on the nonlinear least-squares problem of fitting a polynomial.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

2.4 Newton's Method

Originally developed to find the roots of polynomials $f(\mathbf{x})$, Newton's method may also be used to find roots of $\nabla f(\mathbf{x}) = \mathbf{0}$ in optimization. Newton's method is an effective second-order iterative descent scheme that uses the second derivative information to improve gradient descent.

Instead of using a first-order Taylor series approximation, as in gradient descent, we will use a second-order Taylor series approximation. For a scalar variable x:

$$\tilde{f}(x) \approx f(x_0) + f'(x_0)\underbrace{(x - x_0)}_{\Delta x} + \frac{1}{2}f''(x_0)\underbrace{(x - x_0)}_{\Delta x}^2. \tag{2.38}$$

It is possible to solve for the minimum of this approximate function by setting $\nabla\tilde{f} = 0$:

$$\nabla\tilde{f} = f'(x_0) + f''(x_0)(x - x_0) = 0 \quad \implies \quad x - x_0 = \Delta x = -\frac{f'(x_0)}{f''(x_0)}.$$

Turning this into an iterative algorithm yields Newton's method in one-dimension:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{2.39}$$

This is very much like gradient descent in Eq. (2.3), with a step size $\gamma = 1/f''(x_k)$.

Extending to a multivariate $\mathbf{x}$, our second-order Taylor series approximation becomes

$$\tilde{f}(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T\nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0). \tag{2.40}$$

The Hessian $\mathbf{H} = \nabla^2 f(\mathbf{x})$ is a matrix of second partial derivatives:

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{2.41}$$

By the commutative property of partial derivatives, this is a symmetric matrix.

Again, it is possible to solve for the minimum of the approximate function in Eq. (2.40) by computing the gradient[2] $\nabla\tilde{f}$ and setting equal to zero:

$$\begin{aligned} \nabla\tilde{f} &= \nabla f(\mathbf{x}_0) + \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) = \mathbf{0} \\ \implies \quad \mathbf{x} - \mathbf{x}_0 &= \Delta\mathbf{x} = -\left(\nabla^2 f(\mathbf{x}_0)\right)^{-1} \cdot \nabla f(\mathbf{x}_0). \end{aligned}$$

Thus, this establishes the most basic Newton's method iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k^{-1}\nabla f(\mathbf{x}_k) \tag{2.42}$$

[2]Computing the gradient $\nabla\tilde{f}$ required several vector calculus identities, such as $\nabla\mathbf{c}^T\mathbf{x} = \mathbf{c}$ and $\nabla\mathbf{x}^T\mathbf{Q}\mathbf{x} = 2\mathbf{Q}\mathbf{x}$ for symmetric $\mathbf{Q}$. These identities are provided in Sec. 2.6.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

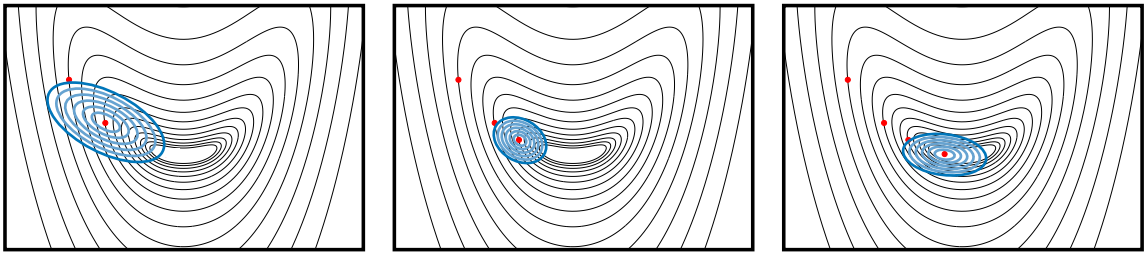


Figure 2.13: Newton's method fits a locally quadratic model at the current iteration point, and then steps to the minima of this quadratic fit for the next iteration step.

where $\mathbf{H}_k = \nabla^2 f(\mathbf{x}_k)$ is the Hessian at point $\mathbf{x}_k$. Newton's method has much faster convergence than gradient descent, but requires an expensive evaluation of the Hessian $\nabla^2 f$.

Newton's method is exact if $f(\mathbf{x})$ is a second-order polynomial, for example in the least squares regression problem. Figure 2.13 illustrates this procedure for a non-convex objective function landscape.

Convergence and Scaling

Newton's method has quadratic error convergence under mild assumptions. Specifically, if f is twice continuously differentiable, $\nabla^2 f$ is Lipschitz continuous near the minimizer $\mathbf{x}^*$, and $\nabla^2 f(\mathbf{x}^*)$ is positive definite, then the error $\boldsymbol{\epsilon}_k = \mathbf{x}_k - \mathbf{x}^*$ at iteration k satisfies

$$\|\boldsymbol{\epsilon}_{k+1}\| \leq \frac{1}{2}\|\boldsymbol{\epsilon}_k\|^2. \tag{2.43}$$

This quadratic convergence of the error is *much* stronger than the $\mathcal{O}(1/k^2)$ convergence of Nesterov's momentum method and ADAM in the iteration number k. Quadratic convergence of the error means that the number of correct digits in the estimate roughly doubles at each step near the optimum. These momentum-based methods improve over standard gradient descent, but they are still much slower than Newton's method near a solution.

Lipschitz continuity of the Hessian $\nabla^2 f$ means that there exists a $L \geq 0$ such that

$$\|\nabla^2 f(\mathbf{x}) - \nabla^2 f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\| \tag{2.44}$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. If $\nabla^2 f$ is continuously differentiable and its third derivatives are bounded, then this condition is satisfied. This is a relatively mild assumption for most smooth functions.

Of course, Newton's method comes at a significantly higher per-iteration cost. The Hessian $\mathbf{H} = \nabla^2 f$ is an $n \times n$ matrix, so its memory footprint scales as n^2 and computing its inverse scales as n^3. This leads to two key issues with Newton's method:

1. The Hessian may be unavailable or too expensive to compute;
2. The Hessian may not be positive definite, so that the function f is locally a maximum or a saddle point.

Both of these challenges motivate quasi-Newton methods, such as the BFGS and L-BFGS algorithms that will be discussed in the next section.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

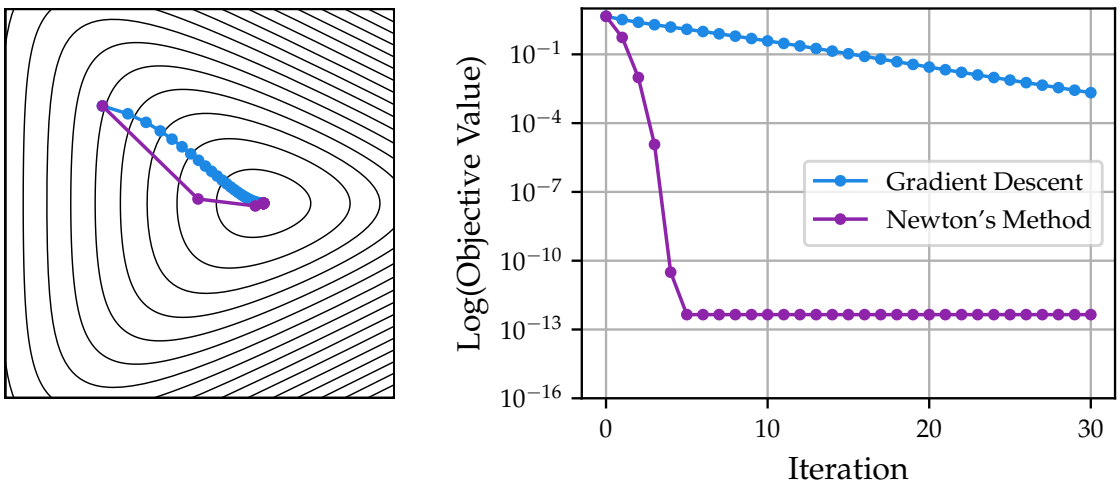


Figure 2.14: Comparison of gradient descent and Newton's method on a non-quadratic objective function, plotted on a semilog error scale to better visualize the convergence rate. It is clear that the second-order Newton's method is dramatically faster than the first-order convergence of simple gradient descent.

Numerical Examples

We now compare gradient descent and Newton's method on the non-quadratic objective function in Eq. (2.8). Newton's method is exact for quadratic objective functions, and so it would converge to the minimum of the quadratic function in Eq. (2.7). Therefore we compare these algorithms on a more challenging non-convex objective function.

The following short code executes Newton's method:

```
def newtons_method(start, max_iter):
    trajectory = [np.array(start)]
    values = [objective_function(*start)]
    point = np.array(start, dtype=float)

    for _ in range(max_iter):
        grad = gradient(point[0], point[1])
        hess = hessian(point[0], point[1])
        hess_inv = np.linalg.inv(hess)
        point -= hess_inv @ grad
        trajectory.append(point.copy())
        values.append(objective_function(point[0], point[1]))

    return np.array(trajectory), np.array(values)

# Run updated methods
nm_trajectory, nm_values = newtons_method(start_point, max_iter)
```

Figure 2.14 shows the trajectory and objective function value versus iteration for both algorithms. We use an initial condition of $[-2, 1]$ and a max number of iterations of 30;

Copyright © July 15, 2025 Brunton. All Rights Reserved.

for gradient descent, we use step size $\gamma = 0.05$. Newton's method clearly converges more rapidly than gradient descent. Note that the objective function is plotted on a logarithmic scale. This plot shows the log-linear scaling of gradient descent and the incredible convergence of Newton's method. Within five steps, the Newton's method solver converges to within near machine precision of the global minimum.

Of course, this incredible performance of Newton's method comes at the cost of requiring Hessian information of the objective function f:

$$\mathbf{H} = \nabla^2 f(\mathbf{x}) = \begin{bmatrix} e^{x_1+2x_2-0.2} + e^{x_1-2x_2-.2} + e^{-x_1-0.2} & 2e^{x_1+2x_2-0.2} - 2e^{x_1-2x_2-0.2} \\ 2e^{x_1+2x_2-0.2} - 2e^{x_1-2x_2-0.2} & 4e^{x_1+2x_2-0.2} + 4e^{x_1-2x_2-0.2} \end{bmatrix}.$$

In code:

```
# Define the objective function and its derivatives
def objective_function(x, y):
    return np.exp(x+2*y-0.2) + np.exp(x-2*y-0.2) + np.exp(-x-0.2)

def gradient(x, y):
    grad_x = np.exp(x+2*y-0.2) + np.exp(x-2*y-0.2) - np.exp(-x-0.2)
    grad_y = 2 * np.exp(x+2*y-0.2) - 2 * np.exp(x-2*y-0.2)
    return np.array([grad_x, grad_y])

def hessian(x, y):
    h_xx = np.exp(x+2*y-0.2) + np.exp(x-2*y-0.2) + np.exp(-x-0.2)
    h_xy = 2 * np.exp(x+2*y-0.2) - 2 * np.exp(x-2*y-0.2)
    h_yy = 4 * np.exp(x+2*y-0.2) + 4 * np.exp(x-2*y-0.2)
    return np.array([[h_xx, h_xy], [h_xy, h_yy]])
```

Comparison of All Methods Now we compare all three methods, gradient descent, Nesterov's accelerated gradient descent, and Newton's method, on a custom nonlinear objective function with *banana* shaped level sets. The objective function, its gradient, and its Hessian are given by:

$$\begin{aligned} f(\mathbf{x}) &= (x_2 - x_1^2)^2 + x_1^4 \\ \nabla f(\mathbf{x}) &= \begin{bmatrix} 4x_1^3 - 4x_1(x_2 - x_1^2) \\ 2(x_2 - x_1^2) \end{bmatrix} \\ \mathbf{H} = \nabla^2 f(\mathbf{x}) &= \begin{bmatrix} 12x_1^2 - 4(x_2 - 3x_1^2) & -4x_1 \\ -4x_1 & 2 \end{bmatrix}. \end{aligned}$$

Figure 2.15 shows the comparison of all three methods. As expected, the second-order Newton's method dramatically outperforms the first-order gradient descent methods, although at a larger cost of evaluation.

Line Search Algorithm

The second-order, quadratic approximation in Eq. (2.40) is better than the simple first-order gradient approximation. However, away from the base point $\mathbf{x}_0$, this approximation may degrade, especially for more complex objective function landscapes. Thus, instead

Copyright © July 15, 2025 Brunton. All Rights Reserved.

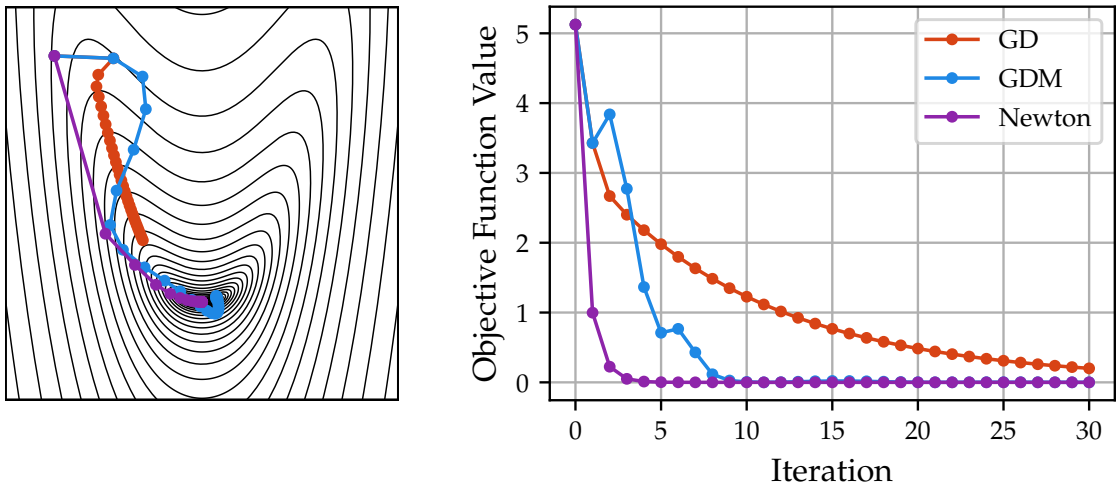


Figure 2.15: Comparison of gradient descent (GD), gradient descent with momentum (GDM), and Newton's method on a nonlinear objective function.

of using Eq. (2.42) to update $\mathbf{x}_{k+1}$, we may use it to determine a better gradient direction $\mathbf{d} = \mathbf{H}^{-1}\nabla f(\mathbf{x})$, and then do a line search in this direction. The cost of each iteration in the second-order Newton's method is expensive, so the additional cost of a line search is justified. It is common to use the Wolfe conditions for the line search, which moves in the new gradient direction $\mathbf{d}$ until $\nabla_{\mathbf{d}} f < \tau$, where τ is a threshold that indicates when a sufficiently flat region has been found. This typically involves two stages:

1. We take steps of size Δx and size $2\Delta x$ until we find a valid point where either: i) $f_2 > f_1$ or ii) $\nabla f_2 > 0$. In the first case, the objective function increased in the second step, so it must have reached a local minimum between. In the second case, the slope of the second point is positive, which also indicates a local minimum between. If these conditions are not met, then we double the step size and repeat.

2. Once a region containing a minimum has been identified, we *zoom* in to the minimum with a bisection method. This amounts to reducing the interval until a point is found. There are several fancier tricks we can play, like using spline interpolation if the are multiple points in the bisection search.

2.5 Quasi-Newton Methods and BFGS

Quasi-Newton methods are a broad class of approaches to approximate the Hessian used in Newton's method [52, 96, 269]. This topic has been developed in the numerical applied mathematics community for over half a century. Here we will provide an introduction to this deep topic, including motivation, basic derivations, and key results. However, there is much more depth that can be found in reviews and textbooks dedicated to the subject.

When $\mathbf{H}$ is too expensive to compute, or when $\mathbf{H}$ is not positive definite, we may approximate this Hessian with a matrix $\mathbf{A}$. The matrix $\mathbf{A}$ provides a quadratic approximation

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

$\tilde{f}$ to our objective function $f(\mathbf{y})$ at some point $\mathbf{y}$ near our current base point $\mathbf{x}$:

$$\tilde{f}_{\mathbf{A}}(\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{1}{2}(\mathbf{y}-\mathbf{x})^T\mathbf{A}(\mathbf{y}-\mathbf{x}). \tag{2.45}$$

To design an effective approximation $\mathbf{A}$ to $\mathbf{H}$, we need to put some constraints on this new matrix:

- $\mathbf{A}$ should be symmetric, as $\mathbf{H}$ is symmetric.
- $\mathbf{A}$ should be positive definite, since we will invert it in Newton's method.
- The quadratic approximation $\tilde{f}_{\mathbf{A}}$ should have the same gradient at the current point $\mathbf{x}_k$ and last point $\mathbf{x}_{k-1}$:

$$\nabla\tilde{f}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k) \tag{2.46a}$$
$$\nabla\tilde{f}(\mathbf{x}_{k-1}) = \nabla f(\mathbf{x}_{k-1}). \tag{2.46b}$$

- Since $\mathbf{A}$ will be updated at each iteration step, we add a subscript $\mathbf{A}_k$. We want $\mathbf{A}_k$ to be "close" to $\mathbf{A}_{k-1}$. There are several notions of closeness we will investigate below.

Based on these conditions, there are several approaches to approximate $\mathbf{A}$, which we will discuss further below:

- SR1: This approach seeks a rank-1 update to $\mathbf{A}$, so that

$$\text{rank}\,(\mathbf{A}_k - \mathbf{A}_{k-1}) = 1. \tag{2.47}$$

  Specifically, a rank-1 update will have the form $\mathbf{A}_k = \mathbf{A}_{k-1} + \mathbf{u}\mathbf{v}^T$, where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ are vectors the same size as $\mathbf{x}$. This is the simplest update approach, and is often used in trust region methods. However, for general optimization, there might not be a rank-1 update that satisfies the various conditions above.

- PSB: Another approach is to find an update that has minimum Frobenius norm distance to the previous approximation:

$$\min \|\mathbf{A}_k - \mathbf{A}_{k-1}\|_F \tag{2.48}$$

  where the Frobenius norm, $\|\cdot\|_F^2$, is the sum of the squares of every element of the matrix argument. However, this approach does not guarantee that $\mathbf{A}_k$ is positive definite. We will discuss matrix norms and optimizing over matrix arguments more in Chapters 4 and 6.

- DFP: The DFP approach is similar to PSB, but incorporates a special weight matrix $\mathbf{W}$ into the Frobenius norm:

$$\min \|\mathbf{W}\,(\mathbf{A}_k - \mathbf{A}_{k-1})\,\mathbf{W}\|_F. \tag{2.49}$$

  Depending on the choice of $\mathbf{W}$, this approach will guarantee that $\mathbf{A}_k$ is positive definite if $\mathbf{A}_{k-1}$ is positive definite.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- BFGS: This is the most stable and widely used quasi-Newton method, and it is closely related to DFP. Since we will eventually be using the inverse of the Hessian approximation, $\mathbf{B} = \mathbf{A}^{-1}$, in the Newton iteration, BFGS directly updates $\mathbf{B}$ instead of $\mathbf{A}$:

$$\min \|\mathbf{W}\left(\mathbf{B}_k - \mathbf{B}_{k-1}\right)\mathbf{W}\|_F. \tag{2.50}$$

It is important to note that all of these methods have analytic solutions, resulting in explicit update formulas that are efficient, in terms of a few matrix-vector products. To initialize these updates, we typically set $\mathbf{A}_0 = \mathbf{B}_0 = \mathbf{I}$, the identity matrix.

Now we will provide basic derivations and formulas for how to update $\mathbf{A}$ and $\mathbf{B}$ in the various schemes above. We introduce a few important points of notation:

$$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \tag{2.51a}$$
$$\mathbf{q}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k). \tag{2.51b}$$

These vectors quantify the change in $\mathbf{x}$ and $\nabla f(\mathbf{x})$ at each step of the iteration: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k \implies \mathbf{p}_k = \Delta\mathbf{x}_k$, and similarly $\mathbf{q} = \Delta\mathbf{g}_k$, where $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$. We will update these vectors at every iteration, and so we drop the subscript k for ease of notation.

There is an important geometric property of the Hessian, namely that $\mathbf{H}\Delta\mathbf{x} \approx \Delta\mathbf{g}$, where $\mathbf{g} = \nabla f(\mathbf{x})$ is again the gradient of f at $\mathbf{x}$. Therefore, we should expect this to also hold for our approximate Hessian $\mathbf{A}$:

$$\mathbf{A}\mathbf{p} = \mathbf{q}. \tag{2.52}$$

This is known as the secant equation. The inverse equation also holds for $\mathbf{B}$:

$$\mathbf{B}\mathbf{q} = \mathbf{p}. \tag{2.53}$$

We are now ready to derive the various methods introduced briefly above. Each of these quasi-Newton methods rely on properties of the inverses of rank-1 and rank-k updates, known as the Sherman Morrison formula and Woodbury matrix identities, respectively. These results are provided at the end of this section.

Rank-1 Update to the Hessian

Although this approach is not used as commonly as BFGS, it is illustrative to derive the rank-1 matrix update

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \mathbf{u}\mathbf{v}^T. \tag{2.54}$$

Using the secant equation and rearranging, we have

$$\left(\mathbf{A}_k + \mathbf{u}\mathbf{v}^T\right)\mathbf{p} = \mathbf{q} \tag{2.55a}$$
$$\implies \mathbf{u}\mathbf{v}^T\mathbf{p} = \mathbf{q} - \mathbf{A}_k\mathbf{p} \tag{2.55b}$$
$$\implies \mathbf{u} = \frac{1}{\mathbf{v}^T\mathbf{p}}\left(\mathbf{q} - \mathbf{A}_k\mathbf{p}\right). \tag{2.55c}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We choose $\mathbf{v} \propto \mathbf{u}$ so that $\mathbf{u}\mathbf{v}^T$ is a symmetric rank-1 matrix. In this case, the update equation becomes

$$\mathbf{v} = \mathbf{q} - \mathbf{A}_k\mathbf{p} \tag{2.56a}$$

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \frac{1}{\mathbf{v}^T\mathbf{p}}\mathbf{v}\mathbf{v}^T. \tag{2.56b}$$

Technically, we want $\mathbf{B} = \mathbf{A}^{-1}$ for Newton's iteration, which results in the following *inverse Hessian* update:

$$\mathbf{v} = \mathbf{p} - \mathbf{B}_k\mathbf{q} \tag{2.57a}$$

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{1}{\mathbf{v}^T\mathbf{q}}\mathbf{v}\mathbf{v}^T. \tag{2.57b}$$

Broyden's Update, BFGS, and DFP

There is a so-called "Broyden's family" of quasi-Newton methods, so we begin by presenting the symplist Broyden update:

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \frac{1}{\mathbf{p}^T\mathbf{A}_k\mathbf{q}}\left(\mathbf{p} - \mathbf{A}_k\mathbf{q}\right)^T\mathbf{p}^T\mathbf{A}_k \tag{2.58a}$$

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{1}{\mathbf{p}^T\mathbf{p}}\left(\mathbf{q} - \mathbf{B}_k\mathbf{p}\right)\mathbf{p}^T. \tag{2.58b}$$

Note that we may either choose to update $\mathbf{A}$ or $\mathbf{B}$, and don't necessarily need to update both.

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) [148, 130, 318, 53] algorithm is a rank-2 update, which provides more degrees of freedom to guarantee that $\mathbf{A}_{k+1}$ is symmetric, positive definite, and *close* to the previous $\mathbf{A}_k$. The updates for $\mathbf{A}$ and $\mathbf{B}$ are given by

$$\mathbf{A}_{k+1} = \left(\mathbf{I} - \frac{\mathbf{p}\mathbf{q}^T}{\mathbf{q}^T\mathbf{p}}\right)\mathbf{A}_k\left(\mathbf{I} - \frac{\mathbf{q}\mathbf{p}^T}{\mathbf{q}^T\mathbf{p}}\right) + \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{q}^T\mathbf{p}} \tag{2.59a}$$

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{q}\mathbf{q}^T}{\mathbf{q}^T\mathbf{p}} - \frac{\mathbf{B}_k\mathbf{p}\left(\mathbf{B}_k\mathbf{p}\right)^T}{\mathbf{p}^T\mathbf{B}_k\mathbf{p}}. \tag{2.59b}$$

It is more clear from the update of $\mathbf{B}_{k+1}$ that this is a rank-2 update, as each term after $\mathbf{B}_k$ is a rank-1 matrix.

The Davidon-Fletcher-Powell (DFP) [130, 90, 281, 131] algorithm is closely related to BFGS. In fact, DFP is the dual of BFGS; however, it was quickly supplanted with BFGS, which has improved stability. For completeness, we present the DFP algorithm here:

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{q}} - \frac{\mathbf{A}_k\mathbf{q}\mathbf{q}^T\mathbf{A}_k}{\mathbf{q}^T\mathbf{A}_k\mathbf{q}} \tag{2.60a}$$

$$\mathbf{B}_{k+1} = \left(\mathbf{I} - \frac{\mathbf{q}\mathbf{p}^T}{\mathbf{q}^T\mathbf{p}}\right)\mathbf{B}_k\left(\mathbf{I} - \frac{\mathbf{p}\mathbf{q}^T}{\mathbf{q}^T\mathbf{p}}\right) + \frac{\mathbf{q}\mathbf{q}^T}{\mathbf{q}^T\mathbf{p}}. \tag{2.60b}$$

Thus it is clear that DFP is the dual of BFGS, with the $\mathbf{p}$ and $\mathbf{q}$ terms swapped. However, the update for $\mathbf{B}$, which is what we want for the Newton iteration, is much simpler in BFGS than in DFP.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The Broyden family of updates is obtained by interpolating between BFGS and DFP:

$$\mathbf{B}_{k+1} = (1-\alpha)\mathbf{B}_{k+1}^{\text{BFGS}} + \alpha\mathbf{B}_{k+1}^{\text{DFP}} \quad \text{where} \quad \alpha \in [0,1]. \tag{2.61}$$

However, we typically set $\alpha = 0$ and use BFGS.

There are other prominent variants, such as the low-memory BFGS (LBFGS) [245, 267, 228, 63], which is widely used in maximum entropy learning [236, 8] and has a stochastic extension for large-scale optimization [62].

Inverse Formulas for Rank-k Matrix Updates

Here we provide the Sherman-Morrison formula for the inverse of a rank-1 matrix update and the Woodbury matrix identity for the inverse of a rank-k matrix update. In both cases, we assume that we have access to a matrix $\mathbf{A}$ and its inverse $\mathbf{A}^{-1}$. We then seek a simple formula for the inverse of the rank-k update of $\mathbf{A}$.

The Sherman-Morrison formula [319] for the inverse of a rank-1 update is

$$\left(\mathbf{A} + \mathbf{u}\mathbf{v}^T\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}. \tag{2.62}$$

Here the rank-1 matrix $\mathbf{u}\mathbf{v}^T$ is added to $\mathbf{A}$. The matrix $\mathbf{A} + \mathbf{u}\mathbf{v}^T$ will only have an inverse if $1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u} \neq 0$.

Although this expression seems rather involved, it is worth noting that it may be computed simply with just a few matrix-vector products and vector-vector inner products. More history and details are presented in the SIAM review of Hager [160]. The version presented here is from Bartlett [22].

The Woodbury matrix identity [373] is a more general result that works for a rank-k matrix update. In this case we add $\mathbf{U}\mathbf{v}^T$ to $\mathbf{A}$, where $\mathbf{U}, \mathbf{v} \in \mathbb{R}^{n\times k}$ are both matrices with k linearly independent columns. In this case, Woodbury's identity is

$$\left(\mathbf{A} + \mathbf{U}\mathbf{v}^T\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I} + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{v}^T\mathbf{A}^{-1}. \tag{2.63}$$

Here $\mathbf{I}$ is a $k \times k$ identity matrix. As with the Sherman-Morrison formula, this rank-k updated matrix will only have an inverse if $\mathbf{I} + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{U}$ is invertible. This result is commonly attributed to Woodbury [373], although it was technically presented earlier in [159], as reviewed in [160].

2.6 Gradient Vector Calculus Identities

There are several important vector calculus identities we will use repeatedly when computing gradients. Here we state and derive the most common identities that we will use.

It is often the case that we must take the gradient of a scalar objective function obtained by taking an inner product of our decision vector $\mathbf{x}$ with a constant vector $\mathbf{c}$, such as the objective function $\mathbf{c}^T\mathbf{x}$ in linear programming in Chapter 3. This gradient is given by

$$\nabla\left(\mathbf{c}^T\mathbf{x}\right) = \nabla\left(\mathbf{x}^T\mathbf{c}\right) = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \end{bmatrix} = \mathbf{c}. \tag{2.64}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

This is easy to derive by expanding $\mathbf{c}^T\mathbf{x}$ and $\mathbf{x}^T\mathbf{c}$:

$$\mathbf{c}^T\mathbf{x} = \begin{bmatrix} c_1 & c_2 & \cdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} = c_1x_1 + c_2x_2 + \cdots = \begin{bmatrix} x_1 & x_2 & \cdots \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \end{bmatrix} = \mathbf{x}^T\mathbf{c}. \tag{2.65}$$

Thus, the gradient of the expanded sum in the middle, $c_1x_1 + c_2x_2 + \cdots$ is the vector $\mathbf{c}$.

It is also common to take the gradient of a matrix-vector product, such as $\mathbf{A}\mathbf{x}$ or $\mathbf{x}^T\mathbf{A}$. To derive these expressions, it helps to carefully expand out the product first:

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1j}x_j \\ a_{2j}x_j \\ \vdots \\ a_{nj}x_j \end{bmatrix} \tag{2.66a}$$

$$\mathbf{x}^T\mathbf{A}^T = \begin{bmatrix} x_1 & x_2 & \cdots x_n \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots a_{1n} \\ a_{21} & a_{22} & \cdots a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} x_ja_{1j} & x_ja_{2j} & \cdots x_ja_{nj} \end{bmatrix} \tag{2.66b}$$

where the sum over j implied. Using these expansions, we may then compute the Jacobian $\mathbf{D}(\mathbf{A}\mathbf{x})$ and the gradient $\nabla\left(\mathbf{x}^T\mathbf{A}^T\right)$:

$$\nabla(\mathbf{A}\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}(a_{1j}x_j) & \frac{\partial}{\partial x_2}(a_{1j}x_j) & \cdots \\ \frac{\partial}{\partial x_1}(a_{2j}x_j) & \frac{\partial}{\partial x_2}(a_{2j}x_j) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots a_{1n} \\ a_{21} & a_{22} & \cdots a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \mathbf{A} \tag{2.67}$$

and

$$\nabla\left(\mathbf{x}^T\mathbf{A}^T\right) = \begin{bmatrix} \nabla x_ja_{1j} & \nabla x_ja_{2j} & \cdots \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots a_{1n} \\ a_{21} & a_{22} & \cdots a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \mathbf{A}^T. \tag{2.68}$$

This implies the following

$$\nabla\left(\mathbf{x}^T\mathbf{A}\right) = \mathbf{A}. \tag{2.69}$$

Both of these properties are actually true even for nonsquare $\mathbf{A}$ matrices, which is left as an exercise for the reader. Sometimes we will use the gradient notation $\nabla(\mathbf{A}\mathbf{x})$ for the Jacobian $\mathbf{D}(\mathbf{A}\mathbf{x})$.

Finally, we often must take the gradient of a quadratic form $\mathbf{x}^T\mathbf{Q}\mathbf{x}$, which arises in the least-squares regression problem. This gradient is given by

$$\nabla\left(\mathbf{x}^T\mathbf{A}\mathbf{x}\right) = \left(\mathbf{A} + \mathbf{A}^T\right)\mathbf{x} \tag{2.70}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

which is equal to $2\mathbf{Ax}$ if $\mathbf{A}$ is symmetric. To derive this expression we expand the quadratic form using the formulas above

$$\mathbf{x}^T\mathbf{Ax} = \begin{bmatrix} x_1 & x_2 & \cdots & c_n \end{bmatrix} \begin{bmatrix} a_{1j}x_j \\ a_{2j}x_j \\ \vdots \\ a_{nj}x_j \end{bmatrix} \tag{2.71a}$$

$$= x_1 \sum_j a_{1j}x_j + x_2 \sum_j a_{2j}x_j + \cdots + x_n \sum_j a_{nj}x_j \tag{2.71b}$$

It is then possible to take the partial with respect to x_i of this expression:

$$\frac{\partial \mathbf{x}^T\mathbf{Ax}}{\partial x_1} = \sum_j a_{1j}x_j + [x_1a_{11} + x_2a_{21} + \cdots + x_na_{n1}] \tag{2.72a}$$

$$\frac{\partial \mathbf{x}^T\mathbf{Ax}}{\partial x_2} = \sum_j a_{2j}x_j + [x_1a_{12} + x_2a_{22} + \cdots + x_na_{n2}] \tag{2.72b}$$

$$\vdots$$

$$\frac{\partial \mathbf{x}^T\mathbf{Ax}}{\partial x_i} = \underbrace{\sum_j a_{ij}x_j}_{\mathbf{Ax}} + \underbrace{[x_1a_{1i} + x_2a_{2i} + \cdots + x_na_{ni}]}_{\mathbf{A}^T\mathbf{x}} \tag{2.72c}$$

which finally yields $\nabla \mathbf{x}^T\mathbf{Ax} = \mathbf{Ax} + \mathbf{A}^T\mathbf{x}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested reading

Papers and reviews

(1) Adam: A method for stochastic optimization, by D. P. Kingma and J. Ba, *arXiv:1412.6980*, 2014 [200].

(2) Learning representations by back- propagating errors, by D. E. Rumelhart, G. E. Hinton, an R. J. Williams, *Nature*, 1986 [304].

(3) Some methods of speeding up the convergence of iteration methods, by B. T. Polya, *USSR computational mathematics and mathematical physics*, 1964 [279].

(4) A method for unconstrained convex minimization problem with the rate of convergence O ($1/k^2$**)**, by Y. Nesterov, *Dokl. Akad. Nauk. SSSR*, 1983 [263].

(5) Nesterov's Method for Convex Optimization, by N. J. Walkington, *SIAM Review*, 2023 [363].

(6) Quasi-Newton methods, motivation and theory, by J. E. Dennis Jr., and J. J. Moré, *SIAM Review*, 1977 [96].

(7) A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights, by W. Su, S. Boyd, and E. J. Candes, *Journal of Machine Learning Research*, 2016 [332].

(8) On the limited memory BFGS method for large scale optimization, by D. C. Liu and J. Nocedal, *Mathematical Programming*, 1989 [228].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

Exercise 2–1: In this exercise, we will explore the effect of step size on convergence and stability of gradient descent. Consider the quadratic objective function in (2.7). Compute the step size γ where the iteration goes unstable. Simulate gradient descent with several different step sizes and plot the results. Is the stability boundary as predicted?

Exercise 2–2: Implement a simple gradient descent for the function

$$f(x) = x^2 + 2x + 5.$$

1. Show the iteration of x_k for $k = 0, \ldots, 10$ starting from $x_0 = 5$.
2. What happens if you choose a step size $\alpha = 1$? How about $\alpha = 0.01$?

Exercise 2–3: Consider the ODE $\dot{x} = -\nabla f(x)$ for $f(x) = \frac{1}{2}x^2$.

1. Solve the ODE analytically.
2. Compare the continuous gradient flow solution to the discrete updates from gradient descent with step size $\alpha = 0.1$.

Exercise 2–4: Implement both Heavy Ball *and* Nesterov's Accelerated Gradient (NAG) to minimize

$$f(x) = \sin^2(x) + 0.1x^2.$$

1. Compare convergence rates and plot the objective $f(x_k)$ vs. iteration k.
2. Discuss any overshoot or oscillations for each method.

Exercise 2–5: Use Newton's method to find the minimum of

$$f(x, y) = x^4 + x^2y^2 + y^2.$$

1. Compute the Hessian.
2. Run the method starting from $(x_0, y_0) = (2, 1)$ and use line search.
3. Compare with gradient descent for the same starting point.

Exercise 2–6: Consider the following constrained optimization problem

$$\begin{aligned} \min_{\mathbf{x}} \ & x_1^2 + x_2^2 \\ \text{s.t.} \ & x_1 \geq 2. \end{aligned}$$

Use the method of Lagrange multipliers to show that this problem has a minimum at the point $(x_1, x_2) = (2, 0)$. Compute the gradient of the objective function and the gradient of the constraint equation. What do you notice?

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Now consider the modified problem

$$\begin{aligned} \min_{\mathbf{x}} \ & x_1^2 + x_2^2 \\ \text{s.t.} \ & x_1 + x_2 \geq 2. \end{aligned}$$

Show that the minimum of this problem occurs at $(1, 1)$. Compute the gradient of the objective function and the gradient of the constraint equation. What is the pattern here?

Exercise 2–7: Show that the forward Euler integration of the gradient flow ODE in Eq. (2.9) is equivalent to the gradient descent iteration in Eq. (2.3) with $\Delta t = \gamma$.

Exercise 2–8: Derive a new gradient descent algorithm using a more sophisticated Runge-Kutta integration scheme for the gradient flow ODE in Eq. (2.9). For example, recall that the 2nd order Runge Kutta integration scheme is given by the following:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \mathbf{v}_2$$

where

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{v}(\mathbf{x}_k) \\ \mathbf{v}_2 &= \mathbf{v}(\mathbf{x}_k + \Delta t \mathbf{v}_1/2). \end{aligned}$$

Code this and compare with gradient descent, specifically convergence and stability.

Exercise 2–9: As a simple exercise, compute the maximum step size $\gamma_{\max}$ and optimal step size γ^* for the quadratic example in Fig. 2.4. You can then test gradient descent with a γ just above and below the maximum stability boundary and also the optimal γ^*.

Exercise 2–10: Show that the eigenvalues of $(\mathbf{I} - \gamma\mathbf{A})$ are symmetric about the point $1 - \gamma\lambda_{\text{mid}}$ from Eq. (2.17).

Exercise 2–11: Plot the convergence rate vs iteration k for Nesterov's accelerated gradient method for a Lipschitz continuous function and for a μ-strongly convex function, using Eqs. (2.26) and (2.27). Try different K and μ and discuss the trends.

Exercise 2–12: We will now explore how gradient descent scales with the dimension of an optimization problem. To make the problem more amenable to analysis, we will consider a quadratic objective function, where we know the optimal step size γ.

First, we will need to be able to generate positive semidefinite $\mathbf{Q}$ matrices of various dimensions $n \times n$. We have several choices:

1. Random: We may generate a quadratic positive semidefinite matrix $\mathbf{Q} = \mathbf{A}^T\mathbf{A}$ by first generating a random $\mathbf{A} \in \mathbb{R}^{m \times n}$.

2. Custom Eigenvalues: It is possible to rescale the eigenvalues of $\mathbf{Q}$ as follows:

   (a) Compute the SVD of $\mathbf{A}$: $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(b) Assuming that the rank of $\mathbf{A}$ is n, then the eigenvalues λ_j of $\mathbf{Q}$ will be the squares of the non-zero singular values σ_j^2.

(c) Thus, to design custom eigenvalues $\hat{\lambda}_j > 0$, we must simply rescale (e.g., replace) the corresponding singular value with $\sqrt{\hat{\lambda}_j}$, resulting in $\hat{\mathbf{\Sigma}}$. Then we construct a new custom $\hat{\mathbf{A}}$ and $\hat{\mathbf{Q}}$

$$\hat{\mathbf{A}} = \mathbf{U}\hat{\mathbf{\Sigma}}\mathbf{V}^T \quad \Longrightarrow \quad \hat{\mathbf{Q}} = \hat{\mathbf{A}}^T\hat{\mathbf{A}}$$

with the custom spectral behavior.

3. Structured: Generate a matrix with diagonal dominant structure

$$\mathbf{Q} = \begin{bmatrix} 2 & 0 & 0 & \cdots & 1 \\ 0 & 3 & 0 & \cdots & 1 \\ 0 & 0 & 4 & \cdots & 1 \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & n+1 \end{bmatrix}.$$

There are several exercises to explore the behavior of gradient descent for different $\mathbf{Q}$:

1. Begin with the structured, diagonally-dominant $\mathbf{Q}$ from above, and a fixed step $\gamma = 0.01$. Write a code to iterate through the gradient descent algorithm, starting from an initial condition of all ones (i.e., $\mathbf{x}_0 = [1, 1, \cdots, 1]$). Plot the convergence versus iteration step for all n from $n = 1$ to $n = 1000$. Put all values of n on the same plot, and change the color of each line from blue ($n = 1$) to red ($n = 1000$). Plot the convergence in both regular and logarithmic values. Discuss the observed trends.

2. Now, repeat the exact same exercise above, but for each value of n, compute the optimal step size γ_n based on the quadratic matrix $\mathbf{Q}$.

3. Now, we will consider the case of a $\mathbf{Q}$ matrix generated from random $\mathbf{A}^T\mathbf{A}$. Write a code to iterate through the gradient descent algorithm and compute how many steps are required before we are within 1% of the optimal value. Recall for this example the optimal $\mathbf{x}^*$ is at the origin, $\mathbf{x}^* = \mathbf{0}$. Run this code for several randomly generated $\mathbf{Q}$ with various n, say 100 random $\mathbf{Q}$ for each dimension. Generate a violin plot of the number of iteration steps as a function of dimension n.

4. Repeat for $\gamma = 0.01$ and for the optimal γ^* for each individual $\mathbf{Q}$ matrix. Also make a violin plot for the optimal γ^* as a function of n.

5. Repeat the exercise above, but now with the eigenvalues of $\mathbf{Q}$ designed to be randomly, uniformly distributed in the range $\lambda_{\min} = 1$ to $\lambda_{\max} = 2$.

6. Repeat the exercise above, but with the eigenvalues of $\mathbf{Q}$ designed to be logarithmically spaced in the range .001 to 1000.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 2–13: Run the same exact progression of exercises as above, but using SGD and mini-batch GD to see how they scale with dimension. Compare and discuss the results.

Exercise 2–14: We will now explore the method of alternating descent, which will discussed more in Chapter 6. When gradient information is unavailable, it is still possible to design a descent algorithm as follows. We simply cycle through each coordinate x_j of the variable $\mathbf{x}$, performing a line search to minimize the objective function in that variable alone. By iterating through each coordinate several times, we will descend towards the optimal value $\mathbf{x}^*$. This method is also known as the method of coordinate descent.

Write a code to perform coordinate descent on the simple quadratic example with $\mathbf{Q} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$ and plot the results. Repeat with $\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$. In both cases, how many times do you need to alternate through each coordinate before reaching the optimal value to within 1%? Keep track of how many function evaluations $f(\mathbf{x})$ were needed for coordinate descent, and plot convergence versus function evaluations. Compare with gradient descent with $\gamma = 0.1$.

You will have several choices for your coordinate descent algorithm, including the step size, how to perform the line search, convergence criteria, etc. A good starting guideline is to use a step size $\gamma = 0.1$ and use bisection to either half or double the step at each step in the line search, based on some simple criteria. Then, end the line search in a given coordinate direction after the change in the decision variable $\mathbf{x}$ is less than some cutoff $\tau \approx 0.001$.

Exercise 2–15: Repeat the coordinate descent exercise above for larger problems with $n = 2$ to $n = 10$. Specifically, explore the following

1. Begin with the structured, diagonally-dominant $\mathbf{Q}$ from above, and a fixed step $\gamma = 0.01$. Write a code to iterate through the coordinate descent algorithm, starting from an initial condition of all ones (i.e., $\mathbf{x}_0 = [1, 1, \cdots, 1]$). Plot the convergence versus iteration step for all n from $n = 2$ to $n = 10$.

2. Now, consider a $\mathbf{Q}$ matrix generated from random $\mathbf{A}^T\mathbf{A}$ with the eigenvalues of $\mathbf{Q}$ designed to be randomly, uniformly distributed in the range $\lambda_{\min} = 1$ to $\lambda_{\max} = 2$. Plot the convergence versus coordinate iterations, and convergence versus function evaluations, for each $n = 2$ to $n = 10$, but with a single random $\mathbf{Q}$ for each dimension.

3. Repeat the exercise above, but with the eigenvalues of $\mathbf{Q}$ designed to be logarithmically spaced in the range $.001$ to 1000.

Exercise 2–16: Verify that the inverse Hessian update in Eq. (2.57) is in fact the inverse of the rank-1 update in Eq. (2.56).

Exercise 2–17: Verify that the BFGS update in Eq. (2.59) satisfies the secant condition that the updated matrix remains positive definite. Compute the normed error of $\mathbf{A}_{k+1} - \mathbf{A}_k$.

Exercise 2–18: Verify that the identities $\nabla\left(\mathbf{x}^T\mathbf{A}\right) = \mathbf{A}$ and $\mathbf{D}(\mathbf{A}\mathbf{x}) = \mathbf{A}$ hold for non-square matrices $\mathbf{A}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 3

Linear Programming

Linear programming is one of the most widely used and well-studied classes of convex optimization problems. Up until now, we have mainly explored unconstrained optimization, choosing to focus on the objective function. In linear programming, the objective function is linear, and so the problem is almost entirely determined by the structure of the linear inequality constraints.

Inequality constraints capture the tradeoffs associated with constrained resources, such as time, space, money, materials, labor, etc. In fact, much of the theoretical and algorithmic development of linear programming occurred around World War II, due to increasingly complex logistical and supply chain optimization problems. Most real-world problems involve constrained resources, and so linear programs are ubiquitous in modern industrial optimization. Some of the many applications include:

- Resource allocation and workforce scheduling;
- Factory layout, space usage, production scheduling, and inventory management;
- Project management and task scheduling;
- Crop planning and irrigation;
- Portfolio optimization, risk management, and pricing;
- Network flow: routing, bandwidth allocation, etc.;
- Power generation: scheduling, renewable mix, etc.;
- Supply chain logistics, fleet routing and delivery scheduling;
- Airline flight, crew, and maintenance scheduling;
- Healthcare: resource and staff allocation;
- Drug dosage, radiation therapy, etc.

Although there is no closed-form solution to a generic linear programming task, there are several accurate and efficient algorithms that scale well to large problems. In this chapter, we will define the mathematical ingredients of a linear program, provide several examples to illustrate various applications, and discuss solution techniques and extensions.

DRAFT

3.1 Problem Statement and Examples

At its heart, linear programming involves a linear objective function and a set of linear constraint equations:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x}, && \text{(3.1a)} \\ \text{Subject to:} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, && \text{(3.1b)} \\ & \mathbf{x} \geq \mathbf{0}, && \text{(3.1c)} \end{aligned}$$

where $\mathbf{c} \in \mathbb{R}^n$ is the vector of linear weights for the objective function, and $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ define the constraint equations. Technically we don't always require $\mathbf{x} \geq \mathbf{0}$, but it is a common constraint that variables are non-negative.

The feasible set is given by a convex polytope, which is the intersection of the half-planes defined by each of the constraint equations:

$$\mathbf{a}_j^T\mathbf{x} \leq b_j, \quad \text{for } j = 1, \cdots, m \tag{3.2}$$

and the positive quadrant defined by

$$x_j \geq 0, \quad \text{for } j = 1, \cdots, n. \tag{3.3}$$

The intersection of a convex set is also a convex set, and so the feasible polytope is convex. A polytope is generally a *pointy* geometric object, generalizing the notion of a polygon, like a triangle or a quadrilateral, to higher dimensions. The individual constraints in Eqs. (3.2) and (3.3) capture specific properties of the constrained resources, such as total space usage, cost of goods, non-negative prices, etc. Technically, the feasible set might not be bounded, if there are not enough inequality constraints; however, we will mostly focus on the case of bounded polytopes.

The linear objective function is both convex and concave, as it is a flat plane. Therefore, both minimization and maximization tasks are *convex* in linear programming. Many tasks will involve maximizing the objective function, rather than minimizing it, for example when the objective function represents profit or productivity. In this case we write the linear program as

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x}, && \text{(3.4a)} \\ \text{Subject to:} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, && \text{(3.4b)} \\ & \mathbf{x} \geq \mathbf{0}. && \text{(3.4c)} \end{aligned}$$

The gradient of the linear objective function is a constant[1]

$$\nabla f(\mathbf{x}) = \mathbf{c} \tag{3.5}$$

so that level sets of the objective function are straight lines increasing perpendicular to the gradient vector. This is visualized in Fig. 3.1.

[1] See the derivation of this in Section 2.6.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

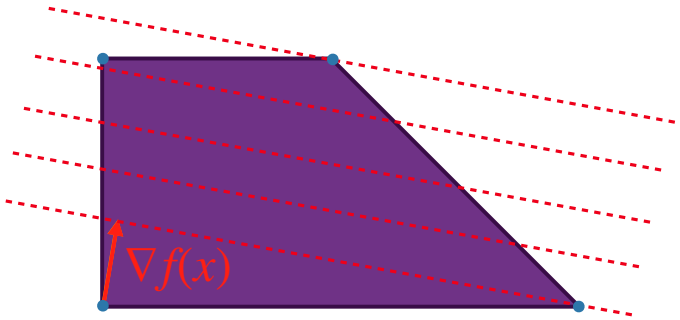


Figure 3.1: Visualization of level sets of the objective function overlaid on top of the convex feasible polytope. The maximum and minimum of the objective function will always intersect one of the vertices of the polytope.

The geometry of the linear objective function and bounded feasible polytope mean that the maximum and minimum will always occur on a vertex of the feasible set. In degenerate cases, there may be a line of local maxima or minima, connecting two vertices, when the gradient is exactly perpendicular to this line. However, even in this case, the local minima will still occur on at least one vertex. See Figs. 3.1 and 3.2 for an illustration of these concepts.

Linear programs are convex optimization problems, so a local maximum (minimum) is also a global maximum (minimum). Therefore, a common strategy for optimizing a linear program is to search across vertices until a local extremum is found, which will then also be a global extremum. However, the number of vertices may grow rapidly in higher dimensions[2] so an efficient search strategy is required. Exhaustively searching through all vertices is intractable.

We will first illustrate how to solve a linear program by hand on a simple low-dimensional example. We will use Dantzig's simplex method [87, 88], which is a method to walk from vertex to vertex in directions that increase the objective function until no neighboring vertex increases the objective function. At this point a local maximum is attained, and by convexity this is also a global maximum. This method is relatively straightforward and scales well to large numerical problems.

For even larger linear programs, we will use interior point methods, which are the other large class of solution techniques; these will be discussed more in Chapter 5. However, Dantzig's simplex method is simple and robust, and it is still widely used in industry.

Simple Example

Consider the following simple example of a linear program with two decision variables and two constraint equations. Say there is a factory that manufactures two types of microscopes: inexpensive microscopes for classrooms, and expensive microscopes for research labs. The profit for each teaching microscope is $\$100$ per unit and the profit for each re-

[2]The number of vertices on an n dimensional hypercube is 2^n, so that a 10 dimensional cube has over a thousand vertices and a 20 dimensional cube has over a million vertices.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

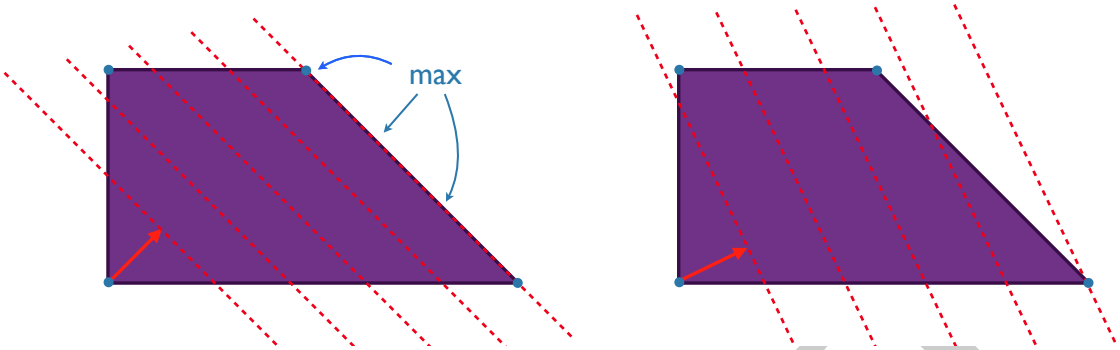


Figure 3.2: If the gradient of the objective function, given by the vector $\nabla f = \mathbf{c}$, rotates, then the maximum may occur on a different vertex. There are also cases where the gradient vector will be perpendicular to one of the edges of the polytope, in which case the entire edge, including the two bounding vertices, will be a local maximum.

search microscope is $\$400$ per unit. The factory can make a maximum of 50 microscopes total per day, and there is a maximum demand for research microscopes of 30 per day. How many of each type of microscope should be made each day to maximize profit?

This linear program can be formulated and solved simply, by hand. In fact, we do not need to resort to formal mathematical arguments to know that the best strategy is to make 30 of the more profitable research microscopes, up to the limit of demand, and then make the remaining 20 inexpensive teaching microscopes to fill the factory's capacity. However, stating and solving this problem formally will help illustrate how to solve more complex linear programs in a systematic way using Dantzig's simplex method.

If we let the number of teaching microscopes be x and the number of research microscopes be y, then the linear program may be stated formally as:

$$\max_{x,y} \; 100x + 400y \tag{3.6a}$$

$$\text{s.t.} \; x, y \geq 0 \tag{3.6b}$$

$$y \leq 30 \tag{3.6c}$$

$$x + y \leq 50. \tag{3.6d}$$

The inequality constraints and resulting convex feasible polytope are shown in Fig. 3.3. The maximum will occur on one of the vertices, so the simple brute force approach would be to simply list all of the vertices and compute the objective function on each vertex:

$$\begin{aligned} v_1 &: (0,0) & f &= 0 \\ v_2 &: (50,0) & f &= 5000 \\ v_3 &: (0,30) & f &= 12000 \\ v_4 &: (20,30) & f &= 14000. \end{aligned}$$

The maximum occurs at the vertex $(20, 30)$, which is consistent with our intuition. However, we need a more robust algorithm for solving these problems that scales to higher dimensions. To this end, we will introduce Dantzig's simplex method in a few sections.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

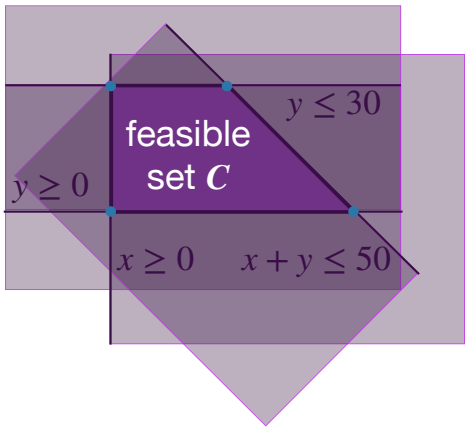


Figure 3.3: The convex feasible polytope given by the constraint equations in Eq. (3.6). Feasible solutions are limited to the positive quadrant ($x \geq 0$ and $y \geq 0$). The additional constraints $y \leq 30$ and $x + y \leq 50$ form a bounded, convex polytope with four vertices.

It is also straightforward to solve linear programs in Python, for example using the `pulp` package with the following code:

```
import pulp

# Create a linear programming problem instance
lp_problem = pulp.LpProblem("Maximize_Profit", pulp.LpMaximize)

# Define the variables
x = pulp.LpVariable('x', lowBound=0, cat='Continuous')
y = pulp.LpVariable('y', lowBound=0, cat='Continuous')

# Set the objective function
lp_problem += 100 * x + 400 * y, "Total Profit"

# Add constraints
lp_problem += y <= 30, "Constraint_y_upper_bound"
lp_problem += x + y <= 50, "Constraint_combined"

lp_problem.solve()  # Solve the problem

print("Optimal value of x:", pulp.value(x))
print("Optimal value of y:", pulp.value(y))
print("Maximum Profit:", pulp.value(lp_problem.objective))
```

The output of this code is

```
Optimal value of x: 20.0
Optimal value of y: 30.0
Maximum Profit: 14000.0
```

which agrees with our earlier calculations.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

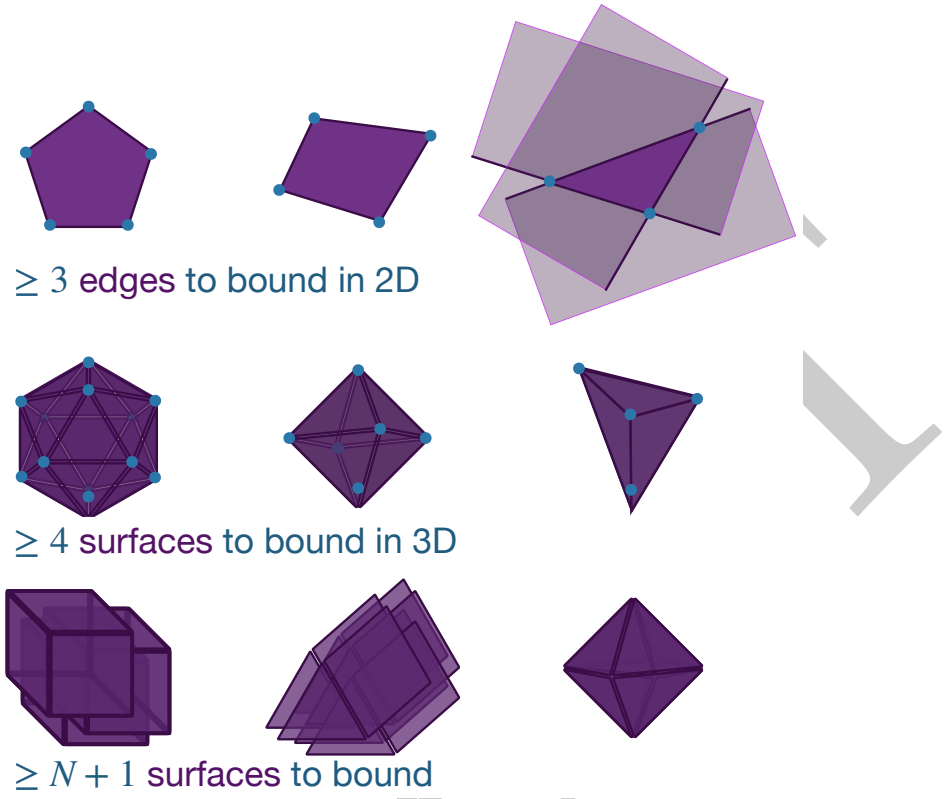


Figure 3.4: Examples of polytopes and their geometries in various dimensions.

3.2 Polytopes and High Dimensional Geometry

Before we proceed to Dantzig's simplex method, it is helpful to understand the geometry of the convex polytopes that define the feasible regions in linear programs. The essence of the simplex method involves searching from vertex to vertex until a local maximum, and thus a global maximum, is found. Thus, we must understand how the vertices relate to the inequality constraints, and how we may efficiently move from vertex to vertex.

Fundamentally, each boundary of the polytope is determined by a particular inequality constraint being *active* or *tight*; this means that they become equality constraints on the boundaries. Each vertex is the intersection of several boundaries, meaning that some subset of the inequality constraints must be *active* or *tight*. We will introduce the slack variable form, which allows us to flexibly relax some inequality constraints while making others active, which will be helpful to walk from vertex to vertex.

It is impossible to fully visualize geometric objects in higher-dimensional spaces, so it is helpful to build intuition in two-dimensions and three-dimensions, and then extrapolate rules and intuition. Figure 3.4 describes the basic notion of polytopes in two-dimensions, in three-dimensions, and in n-dimensions. There are several basic properties that build our intuition:

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- Vertices are always connected by one-dimensional edges.
- The intersection of 2 edges is a vertex.
- The intersection of 2 two-dimensional surfaces is a one-dimensional edge.
- In two-dimensions, polytopes are bounded by at least 3 one-dimensional edges; picture $x, y \geq 0$ and $x + y \leq 1$.
- In three-dimensions, polytopes are bounded by at least 4 two-dimensional surfaces; picture $x, y, z \geq 0$ and $x + y + z \leq 1$.
- In n-dimensions, polytopes are bounded by at least $n + 1$ distinct $n - 1$-dimensional surfaces; picture $\mathbf{x} \geq 0$ and $\sum_j x_j \leq 1$.
- In n-dimensions, a single inequality constraint defines a half-space bounded by an $n - 1$-dimensional surface; so in two-dimensions an inequality constraint defines a half-space bounded by an edge, and in three-dimensions an inequality constraint defines a half-space bounded by a two-dimensional surface.
- Thus, the intersection of $n + 1$ inequality constraints will bound a polytope in n-dimensions; confirm that this matches the specific cases in two-dimensions and three-dimensions above.
- In n-dimensions, the vertices of a polytope are given by the exact equality of n inequality constraints.

The basis of the simplex method in linear programming, which is explored fully in the next section, involves starting with a vertex and then walking from vertex to vertex in directions that locally increase the objective function, until a local optimum is found. Identifying vertices is challenging in higher dimensions, as we will see in a moment.

Slack Variables and Polytopes

We introduce slack variables $\mathbf{s} \in \mathbb{R}^m$ to Eq. (3.4) to make it easier to determine which inequalities are active, which will in turn enable us to and move from vertex to vertex. The slack variables, $\mathbf{s}$, have as many dimensions as there are constraint equations in $\mathbf{Ax} \leq \mathbf{b}$. Introducing slack variables, the linear program becomes

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x}, && \text{(3.7a)} \\ \text{Subject to:} \quad & \mathbf{x}, \mathbf{s} \geq \mathbf{0}, && \text{(3.7b)} \\ & \mathbf{Ax} + \mathbf{s} = \mathbf{b}. && \text{(3.7c)} \end{aligned}$$

This is equivalent to Eq. (3.4), since $\mathbf{s} = \mathbf{b} - \mathbf{Ax} \geq 0$. We typically move everything except for $\mathbf{s}$ to the right of the equation, which results in

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x}, && \text{(3.8a)} \\ \text{Subject to:} \quad & \mathbf{x}, \mathbf{s} \geq \mathbf{0}, && \text{(3.8b)} \\ & \mathbf{s} = \mathbf{b} - \mathbf{Ax}. && \text{(3.8c)} \end{aligned}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Each of the components s_i of $\mathbf{s}$ quantify how close $\mathbf{x}$ is to the i-th inequality boundary. When $s_i = 0$, then the i-th inequality is *active*, and we are on the boundary defined by this equality.

Recall that for a feasible polytope in n-dimensions, each vertex is defined by n inequality constraints being *active* or *tight*. A constraint being active means that it becomes an equality constraint, so that the corresponding slack variable is equal to zero and the decision variable is on the boundary of the constraint; i.e., it is *tight* up against the constraint boundary. The remaining constraints are called *inactive* or *loose*, and there is some *slack* between the current decision variable and those inequality boundaries[3]. For example, at the origin, the constraints $\mathbf{x} \geq 0$ are tight, and the variables $\mathbf{s}$ are all strictly greater than zero, indicating that they are *slack* or *loose* variables.

In slack variable form, the variables on the left of the equality in Eq. (3.8c) are the *loose* or *slack* variables, and those to the right are the tight variables. If we move to another vertex, where the j-th row of $\mathbf{Ax} \leq \mathbf{b}$ becomes an active constraint, then the variable $\mathbf{s}_j$ becomes tight and moves to the right hand side of the equation, and whichever variable is now loose moves to the left hand of the equation. Writing the inequality constraints in slack variable notation makes it simple to see which constraints are tight, and hence which vertex we are currently centered about. An interior point will be away from any boundary, in which case all of the variables will be loose.

In higher dimensions, finding a new feasible vertex, given by n active constraint equations, can be very computationally challenging. The challenge is that solving for the vertex, given a set of n active constraints, involves a large matrix inversion. For example, consider a problem with n decision variables $\mathbf{x} \in \mathbb{R}^n$ and m constraint equations, so that $\mathbf{A} \in \mathbb{R}^{m \times n}$. Each vertex is determined by n of the constraints being *active* or *tight*, so vertices will be given by a subset of n of the $n + m$ constraints being active; remember that there are also generally n positivity constraints $\mathbf{x} \geq 0$ in addition to the m constraints in $\mathbf{Ax} \leq \mathbf{b}$, resulting in $n + m$ total constraints. For each of these n active constraints, we must solve a linear equation to move the active variables to the right hand side of the equations. For the sake of simplicity, lets say that the n active constraints are selected from the m equations in $\mathbf{Ax} \leq \mathbf{b}$. Then these constraints, in slack variable notation, will become

$$\mathbf{s} = -\mathbf{Bx} + \mathbf{c} \tag{3.9}$$

where $\mathbf{B}$ and $\mathbf{c}$ are obtained by selecting the given n rows of $\mathbf{A}$ and $\mathbf{b}$. Then we must solve this equation for $\mathbf{x}$ on the left and $\mathbf{s}$ on the right, which involves an $\mathcal{O}(n^3)$ matrix inversion. Thus, determining and solving for vertices becomes prohibitively expensive in higher dimensions.

If instead, we start with a vertex and walk to a new, nearby vertex that is connected via an edge, this is a much simpler set of matrix computations. If we move to a new vertex from an existing vertex, we need only choose one slack variables to tighten and one tight variable to loosen. This corresponds to choosing one inactive constraint to make active, and one active constraint to make inactive. This procedure of only swapping one pair of variables is much more computationally efficient than solving for an entirely new vertex.

[3] In some literature, the active/tight variables are called non-basic variables, and the loose/slack variables are called basic variables. We do not use this notation, here, but it is useful for the reader to be aware of all of the various synonyms in the linear programming literature.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

3.3 Dantzig's Simplex Method

Dantzig's simplex method relies on two major properties of linear programs:

- A local extremum will occur on a vertex, assuming that the feasible set is a bounded polytope;
- The local extremum is a global extremum because linear programs are convex optimization problems.

Solving a linear program then amounts to finding a local maximum by searching across vertices. The simplest approach would be to enumerate all vertices and evaluate the objective function. However, as discussed earlier, the number of vertices scales poorly with the dimensions of the problem, and even identifying and enumerating individual vertices becomes a non-trivial linear algebra problem in higher dimensions.

The second option is to start with a vertex, and intelligently walk from vertex to vertex until we find a local minimum or maximum, which will be globally optimal by convexity. The process of going from vertex to vertex is called *pivoting*, and there are several strategies. The most commonly used pivoting technique is called Dantzig's pivot rule. In general, this is much more computationally efficient than solving for an entirely new vertex from a set of n active constraints, as we simply pick one active constraint to relax, and a new inactive constraint to tighten, walking us to the next vertex.

The idea is to walk from vertex to vertex(e.g., pivot to pivot) until the objective function cannot be improved any more in any free direction, at which point local optimality, and hence global optimality, will be achieved. The objective function must monotonically increase, or at least not decrease, from pivot to pivot, otherwise the function is not convex.

We use slack variables to allow us to walk from vertex to vertex. Any given vertex is defined by some of the inequalities being tight and some being loose. Recall that a tight inequality means that a variable is against a boundary, and increasing it any further will violate the inequality constraint. A loose variable is one that may increase a nonzero, positive amount without violating any inequality constraint.

Dantzig's pivot rule proceeds as follows:

1. Pick one of the active constraints to relax (i.e., to loosen). This constraint corresponds to one of the inequalities that is currently *active* or *tight*, meaning that it is one of the boundaries defining the current vertex. The inequality we choose to loosen will define the boundary we walk away from, towards a new vertex.

   Dantzig's pivot rule is simple: pick the direction where the objective function increases the most, and if this variable is currently tight, then loosen it. The direction of fastest increase of the objective function is given by the gradient ∇f.

   It is convenient to define the objective function f in terms of tight variables only, which is possible using slack variable form. Then the gradient is computed with respect to the tight variables, so we only look for directions of opportunity that can be loosened.

2. Once we select a tight variable to loosen (i.e., an active constraint to relax), we must choose another loose variable to tighten (i.e., an inactive constraint to make active).

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

This will define a new vertex—the next vertex—that we walk to. Generally, we pick among the slack variables, finding the inequality that will be violated first when increasing the newly loosened variable.

To find which inequality will be violated first, we take the constant b_j in each candidate inequality, divided by the coefficient a_j on the newly loosened variable in that row. The inequality that will be violated first will have the smallest ratio b_j/a_j. If the newly loosened variable doesn't appear in a given row equation, then this row isn't considered for selecting a tight variable.

3. Finally, we need to recenter our problem at the new vertex. We accomplish this by taking the inequality that will be violated first, and swapping the position of the loosened and tightened variables, so that the newly loosened variable is on the left of the equation, and the newly tightened variable is on the right of the equation. Then, we replace every instance of this newly loosened variable in the objective function and every other inequality. As we will see in the simplex tableau formulation in the next section, this is nearly identical to row reduction and pivoting in Gaussian elimination, when computing the inverse of a matrix.

Simple Example

Let's now revisit the simple example from Sec. 3.1. The linear program may be written as:

$$\max_{\mathbf{x}} \; 100x_1 + 400x_2 \tag{3.10a}$$
$$\text{s.t.} \; x_1, x_2 \geq 0 \tag{3.10b}$$
$$x_1 + x_2 \leq 50 \tag{3.10c}$$
$$x_2 \leq 30. \tag{3.10d}$$

Notice that we have renamed the variables (x, y) with (x_1, x_2), since this will help us generalize to higher-dimensional problems. It is possible to write this in slack variable notation:

$$x_1, x_2, s_1, s_2 \geq 0 \tag{3.11a}$$
$$x_1 + x_2 + s_1 = 50 \tag{3.11b}$$
$$x_2 + s_2 = 30. \tag{3.11c}$$

Moving the slack variables to the left side of the equality constraints yields:

$$x_1, x_2, s_1, s_2 \geq 0 \tag{3.12a}$$
$$s_1 = 50 - x_1 - x_2 \tag{3.12b}$$
$$s_2 = 30 - x_2. \tag{3.12c}$$

In slack variable form, we can read off the values of the variables on the left of the equality by setting the variables on the right equal to zero. This means that at the current vertex we have $x_1 = x_2 = 0$ and $s_1 = 50, s_2 = 30$. Thus, this equation is written in a form that is centered at the vertex $(0, 0)$. At each step of Dantzig's simplex algorithm we

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

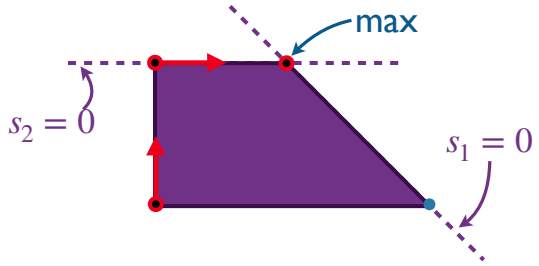


Figure 3.5: Illustration of how slack variables allow one to walk from vertex to vertex. To go from the first to the second vertex, we tighten the s_2 slack variable until $s_2 = 0$, at which point we are tight against the boundary of this inequality. Similarly, to go from the second to the third vertex, we tighten the s_1 slack variable until $s_1 = 0$, at which point we are tight against the boundary of this inequality.

will walk to another vertex, choosing one variable on the right to tighten and one variable on the left to loosen, so that we will move from one inequality constraint boundary to another. The convex feasible polytope is visualized in Fig. 3.5, with the inequality boundaries defined by the slack variables in red.

From here on, we won't keep writing $x_1, x_2, s_1, s_2 \geq 0$, as this will always be true.

- Step 1, vertex $(0, 0)$: Currently, the variables x_1, x_2 are tight and s_1, s_2 are loose. The gradient is $\nabla f = \begin{bmatrix} 100 \\ 400 \end{bmatrix}$, which is larger in the x_2 direction, so we choose to loosen the x_2 variable. Since the $s_2 = 0$ constraint will be violated first when increasing x_2, we choose to tighten s_2. This choice determines along which edge we will progress and which inequality constraint will specify the next vertex in the iteration.

  Loosening x_2 and tightening s_2 changes the second slack constraint equation to

  $$x_2 = 30 - s_2.$$

  Substituting this into the cost function and other constraint equations yields the optimization problem, centered around the new vertex $(0, 30)$.

  The optimization problem now reads:

  $$\max_{x_1, s_2} \; 100x_1 - 400s_2 + 12000 \tag{3.13a}$$
  $$\text{s.t.} \;\; s_1 = 20 - x_1 + s_2 \tag{3.13b}$$
  $$x_2 = 30 - s_2. \tag{3.13c}$$

  The objective function increased by 12000 at this vertex. This may also be evaluated easily by setting $x_1 = s_2 = 0$, as these are our tight variables at this new vertex.

- Step 2, vertex $(0, 30)$. Now the variables x_1, s_2 are tight and s_1, x_2 are loose. The objective function now has a gradient $\nabla f = \begin{bmatrix} 100 \\ -400 \end{bmatrix}$, which only increases in the

Copyright © July 15, 2025 Brunton. All Rights Reserved.

positive x_1 direction. Thus, this is the only direction we can move our vertex and increase the objective function.

We now choose x_1 to loosen, and s_1 to tighten, as the $s_1 = 0$ constraint will be the first to be violated when increasing x_1. The first slack constraint equation now becomes

$$x_1 = 20 - s_1 + s_2.$$

Substituting this into our objective function and other equations now yields a new optimization problem, centered at the vertex $(20, 30)$:

$$\begin{aligned} \max_{\mathbf{s}} \quad & -100s_1 - 400s_2 + 14000 && \text{(3.14a)} \\ \text{s.t.} \quad & x_1 = 20 - s_1 + s_2 && \text{(3.14b)} \\ & x_2 = 30 - s_2. && \text{(3.14c)} \end{aligned}$$

Now the objective function value has increased to 14000 at the vertex $(x_1, x_2) = (20, 30)$, with $s_1 = s_2 = 0$. Since the gradient of the objective function decreases in both the positive s_1 and s_2 directions, this is vertex a local maximum and cannot be increased by moving in the s_1 or s_2 directions. And by convexity, this local maximum is also a global maximum.

Thus, the global optimum point occurs at $(x_1, x_2) = (20, 30)$ with a value of 14000.

Another Example in Two Dimensions

Let's consider another simple two-dimensional example:

$$\begin{aligned} \max_{\mathbf{x}} \quad & x_1 + 2x_2 && \text{(3.15a)} \\ \text{s.t.} \quad & x_1, x_2 \geq 0 && \text{(3.15b)} \\ & 2x_2 - x_1 \leq 2 && \text{(3.15c)} \\ & 2x_1 - x_2 \leq 2. && \text{(3.15d)} \end{aligned}$$

The feasible set is visualized in Fig. 3.5. In this example, the path from vertex to vertex does not always strictly follow a coordinate direction, as in the example above, and we will see more clearly the role of the slack variable in Dantzig's simplex method.

We begin by putting this problem in slack-variable notation:

$$\begin{aligned} \max_{\mathbf{x}} \quad & x_1 + 2x_2 && \text{(3.16a)} \\ \text{s.t.} \quad & x_1, x_2, s_1, s_2 \geq 0 && \text{(3.16b)} \\ & s_1 = 2 + x_1 - 2x_2 && \text{(3.16c)} \\ & s_2 = 2 + x_2 - 2x_1. && \text{(3.16d)} \end{aligned}$$

- Step 1, vertex $(0, 0)$: Currently, the variables x_1, x_2 are tight and s_1, s_2 are loose. The gradient is $\nabla f = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, which is larger in the x_2 direction, so we choose to loosen the

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

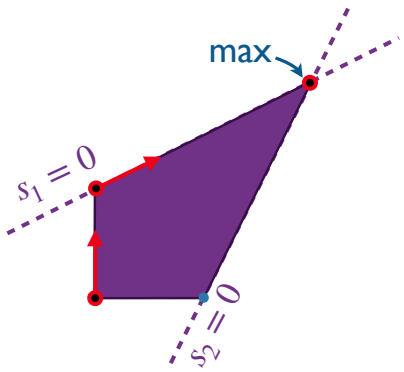


Figure 3.6: Illustration of convex feasible polytope for the linear program in Eq. (3.15). We also include the slack variable boundaries to illustrate how slack variables allow one to walk from vertex to vertex.

x_2 variable. Since the $s_1 = 0$ constraint will be violated first when increasing x_2, we choose to tighten s_1. This choice determines which edge we walk along and which inequality constraint ($s_1 = 0$ will specify the next vertex we walk to).

Loosening x_2 and tightening s_1 changes the first slack constraint equation to

$$x_2 = 1 - s_1/2 + x_1/2. \tag{3.17}$$

Substituting this into the cost function and other constraint equations yields the optimization problem, centered around the new vertex $(0, 1)$.

The optimization problem now reads:

$$\max_{x_1, s_1} \; 2 + 2x_1 - s_1 \tag{3.18a}$$

$$\text{s.t.} \; x_2 = 1 - s_1/2 + x_1/2 \tag{3.18b}$$

$$s_2 = 3 - 3x_1/2 - s_1/2. \tag{3.18c}$$

Note that the objective function increased by 2 at this vertex.

- Step 2, vertex $(0, 1)$: Now the variables x_1, s_1 are tight and x_2, s_2 are loose. The objective function has a gradient $\nabla f = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$, which only increases in the positive x_1 direction. Thus, we choose x_1 to loosen and s_2 to tighten, as this is the first inequality condition that will be violated when increasing x_1. The s_2 constraint becomes

$$x_1 = 2 - s_1/3 - 2s_2/3. \tag{3.19}$$

Substituting this into our objective function and other equations now yields a new optimization problem, centered at the vertex $(2, 3)$

$$\max_{s_1, s_2} \; 6 - 5s_1/3 - 4s_2/3 \tag{3.20a}$$

$$\text{s.t.} \; x_1 = 2 - s_1/3 - 2s_2/3 \tag{3.20b}$$

$$x_2 = 2 - 2s_1/3 - s_2/3. \tag{3.20c}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

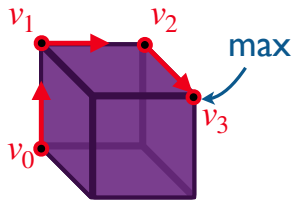


Figure 3.7: Illustration of convex feasible polytope for the linear program in Eq. (3.21). This polytope is a simple cube.

At this point the objective function is decreasing in both the positive s_1 and s_2 directions (the tight variables), so we are at a local maximum, and hence a global maximum. The maximum value is 6 at the point $(x_1, x_2) = (2, 2)$.

Example in Three Dimensions

We will now consider an example in three dimensions:

$$\begin{aligned} \max_{\mathbf{x}} \quad & x_1 + 2x_2 + 3x_3 && \text{(3.21a)} \\ \text{s.t.} \quad & \mathbf{x}, \mathbf{s} \geq 0 && \text{(3.21b)} \\ & s_1 = 1 - x_1 && \text{(3.21c)} \\ & s_2 = 1 - x_2 && \text{(3.21d)} \\ & s_3 = 1 - x_3. && \text{(3.21e)} \end{aligned}$$

The feasible set is visualized in Fig. 3.5.

- Step 1, vertex $(0, 0, 0)$: Currently, the variables x_1, x_2, x_3 are tight and s_1, s_2, s_3 are loose. The gradient is $\nabla f = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T$, which is larger in the x_3 direction, so we choose to loosen the x_3 variable. Since the $s_3 = 0$ constraint will be violated first when increasing x_3, we choose to tighten s_3. This choice determines along which edge we will walk and which inequality constraint will specify the next vertex.

  Loosening x_3 and tightening s_3 changes the second slack constraint equation to

  $$x_3 = 1 - s_3.$$

  Substituting this into the cost function and other constraint equations yields the optimization problem, centered around the new vertex $(0, 0, 1)$.

  The optimization problem now reads:

  $$\begin{aligned} \max_{x_1, x_2, s_3} \quad & x_1 + 2x_2 - 3s_3 + 3 && \text{(3.22a)} \\ \text{s.t.} \quad & s_1 = 1 - x_1 && \text{(3.22b)} \\ & s_2 = 1 - x_2 && \text{(3.22c)} \\ & x_3 = 1 - s_3. && \text{(3.22d)} \end{aligned}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The objective function increased by 3 at this vertex. This may also be evaluated easily by setting $x_1 = x_2 = s_3 = 0$, as these are our tight variables at this new vertex.

- Step 2, vertex $(0, 0, 1)$. Now the variables x_1, x_2, s_3 are tight and s_1, s_2, x_3 are loose. The objective function now has a gradient $\nabla f = \begin{bmatrix} 1 & 2 & -3 \end{bmatrix}^T$, which increases most rapidly the positive x_2 direction. Thus, we now choose x_2 to loosen, and s_2 to tighten, as the $s_2 = 0$ constraint will be the first to be violated when increasing x_2. The second slack constraint equation now becomes

$$x_2 = 1 - s_2$$

Substituting this into our objective function and other equations now yields a new optimization problem, centered at the vertex $(0, 1, 1)$:

$$\begin{aligned} \max_{x_1, s_2, s_3} \quad & x_1 - 2s_2 - 3s_3 + 5 && \text{(3.23a)} \\ \text{s.t.} \quad & s_1 = 1 - x_1 && \text{(3.23b)} \\ & x_2 = 1 - s_2 && \text{(3.23c)} \\ & x_3 = 1 - s_3. && \text{(3.23d)} \end{aligned}$$

Now the objective function value has increased to 5 at the vertex $(x_1, x_2, x_3) = (0, 1, 1)$.

- Step 3, vertex $(0, 1, 1)$. Now the variables x_1, s_2, s_3 are tight and s_1, x_2, x_3 are loose. The objective function now has a gradient $\nabla f = \begin{bmatrix} 1 & -2 & -3 \end{bmatrix}^T$, which only increases in the positive x_1 direction. Thus, we now choose x_1 to loosen, and s_1 to tighten, as the $s_1 = 0$ constraint will be the first to be violated when increasing x_1. The first slack constraint equation now becomes

$$x_1 = 1 - s_1$$

Substituting this into our objective function and other equations now yields a new optimization problem, centered at the vertex $(1, 1, 1)$:

$$\begin{aligned} \max_{s_1, s_2, s_3} \quad & -s_1 - 2s_2 - 3s_3 + 6 && \text{(3.24a)} \\ \text{s.t.} \quad & x_1 = 1 - s_1 && \text{(3.24b)} \\ & x_2 = 1 - s_2 && \text{(3.24c)} \\ & x_3 = 1 - s_3. && \text{(3.24d)} \end{aligned}$$

Since the gradient of the objective function decreases in all directions, this vertex is a local maximum and cannot be increased. And by convexity, this local maximum is also a global maximum.

Thus, the global optimum point occurs at $(x_1, x_2, x_3) = (1, 1, 1)$ with a value of 6.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

$$\max_{\mathbf{x}} \; f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \quad \text{s.t.} \quad \mathbf{x}, \mathbf{s} \geq 0, \quad \mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b}$$

$$\begin{array}{c} \\ f \\ \mathbf{x} \end{array} \begin{array}{c} \begin{array}{cccc} f & \mathbf{x} & \mathbf{s} & \end{array} \\ \left[\begin{array}{cccc} 1 & -\mathbf{c}^T & \mathbf{0} & 0 \\ \hline \mathbf{0} & \mathbf{A} & \mathbf{I} & \mathbf{b} \end{array}\right] \end{array}$$

Figure 3.8: Illustration of the mapping of a linear program into simplex tableau form.

3.4 Simplex Tableau Method

We saw that Dantzig's simplex algorithm conceptually operates by moving from one vertex of the feasible polytope to another, testing and improving the objective until no further improvement is possible. However, rewriting the entire linear program at each pivot step quickly becomes cumbersome in high dimensions.

The *simplex tableau method* provides a systematic matrix-based procedure for these pivot steps, making it possible to program a computer to perform a sequence of row operations similar to Gaussian elimination. The basic idea is to rewrite the linear program in a matrix format, known as the *tableau*, where it is simpler to keep track of the operations in each iteration. In each iteration, we pivot on a selected row and column entry, updating the tableau to reflect how we move from one feasible solution (vertex) to the next. The transcription of a linear program into the simplex tableau is shown in Fig. 3.8.

The operations involved in the simplex tableau method will be closely related to the row operations and pivoting that we are familiar with from classic matrix inversion. Recall that to invert a matrix $\mathbf{A}$, we may form the augmented system

$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{I} \end{array}\right]$$

and repeat row operations and pivoting until the matrix on the left becomes the identity $\mathbf{I}$ and the matrix on the right becomes the inverse of $\mathbf{A}$:

$$\left[\begin{array}{c|c} \mathbf{I} & \mathbf{A}^{-1} \end{array}\right]. \tag{3.25}$$

In the simplex tableau, we follow a similar pattern of pivoting to maintain an identity-like block that represents the current slack variables at each vertex, with the remaining columns representing the active variables. This means each pivot step modifies one row (corresponding to the variable to make active) and one column (corresponding to the variable to make slack), just as we do in a single Gaussian elimination pivot.

Constructing the Initial Tableau

Initializing the tableau essentially amounts to transcribing the problem into a matrix notation, given by the following:

$$\begin{array}{c} \\ f \\ \mathbf{s} \end{array} \begin{array}{c} \begin{array}{cccc} f & \mathbf{x} & \mathbf{s} & \end{array} \\ \left[\begin{array}{cccc} 1 & -\mathbf{c}^T & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{A} & \mathbf{I} & \mathbf{b} \end{array}\right] \end{array}. \tag{3.26}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

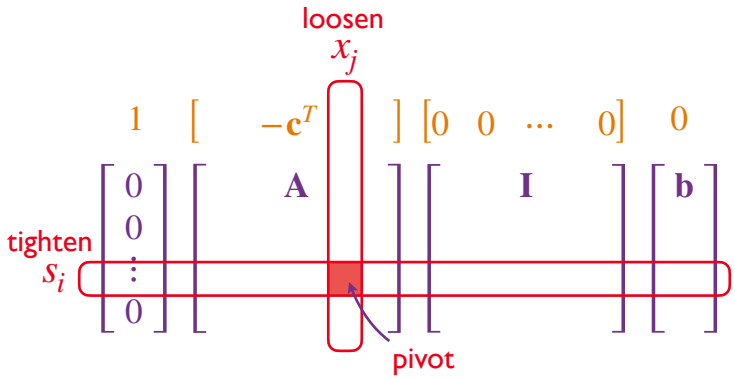


Figure 3.9: Illustration of Simplex Tableau method. Similar to row-reduction when computing the inverse of a matrix, we iteratively choose pivot locations and eliminate all terms above and below the pivot with row operations.

The objective function and constraint equations may be read easily from this tableau. The objective function is the first row, written in terms of the active and inactive variables; as we move from vertex to vertex, different subsets of these variables will take turns being active and inactive. The constraint equations similarly appear in matrix form that can be read as

$$\mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b}$$

The first column corresponds to the objective function f, the next n columns to the active variables (initially the variables $\mathbf{x}$), the next m columns to the inactive slack variables (initially the variables $\mathbf{s}$), and the last column to the right hand side, which will include information about the current value of f at the current vertex. The first row encodes the objective function equation and the second row encodes the m constraint equations. We typically label the left of the second row with the names of the slack variables for that row equation (initially $\mathbf{s}$).

Pivot Step and Iteration

We now define an iterative procedure that is equivalent to how we walked from vertex to vertex in the simplex method above. This is shown schematically in Fig. 3.9. Each iteration picks:

1. *Tight Variable to Loosen (column):* Among the active variables, choose one whose coefficient in the objective row suggests an increase in objective if we increase it in the positive direction. A simple rule is Dantzig's pivot rule: pick the variable with the most negative coefficient (in a maximization problem) in the objective row. This variable will be selected to be loosened.

2. *Loose Variable to Tighten (row):* Among the slack variables, choose the one that becomes zero first if we increase the newly loosened variable, to avoid violating constraints. This is determined by the minimum ratio test: pick the row with the smallest ratio RHS/pivot coefficient.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

3. *Row Reduction about Pivot:* Perform row operations so that the pivot entry becomes 1, and all other entries in that pivot column become 0. This is exactly like a Gaussian elimination pivot. The newly chosen column variable becomes loose and the row variable becomes tight. Sometimes we say that the row variable leaves the basis.

After the pivot, we obtain an updated tableau reflecting a new feasible vertex with a higher objective. We repeat until no negative coefficients remain in the objective row, or until we meet a stopping criterion in other pivot rules. At that point, we have found a local optimum, which is also a global optimum by convexity.

The simplex tableau method is the matrix-level realization of Dantzig's simplex algorithm. Each pivot corresponds to row operations akin to Gaussian elimination, systematically replacing one loose variable with another and updating the objective row. Despite alternative polynomial-time interior-point methods, the tableau approach remains a cornerstone of linear programming and is still used in many modern solvers, thanks to its interpretability and strong performance on typical industrial LPs.

Simple Example, Revisited

We now revisit the example from Eq. (3.10), which we solved by hand using Dantzig's simplex method. Here we will solve this using the simplex tableau method. Comparing each step side-by-side, it will become clear that these are equivalent methods.

We begin by initializing the tableau, as described in Eq. (3.26) above:

$$\begin{array}{c} \\ f \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{cccccc} f & x_1 & x_2 & s_1 & s_2 & \end{array} \\ \left[\begin{array}{cccccc} 1 & -100 & -400 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 50 \\ 0 & 0 & 1 & 0 & 1 & 30 \end{array}\right] \end{array}. \tag{3.27}$$

Now we proceed by identifying a pivot and performing row operations to eliminate entries in the same column. The pivot in this step will correspond to the x_2 variable, as the gradient has the largest magnitude in this direction. After row operations, the simplex tableau becomes:

$$\begin{array}{c} \\ f \\ s_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccccccc} f & x_1 & x_2 & s_1 & s_2 & \end{array} \\ \left[\begin{array}{ccccccc} 1 & -100 & 0 & 0 & 400 & 12000 \\ 0 & 1 & 0 & 1 & -1 & 20 \\ 0 & 0 & 1 & 0 & 1 & 30 \end{array}\right] \end{array}. \tag{3.28}$$

As in Eq. (3.13), we can read off key information from the tableau at this step. For example, the objective function value is 12000. Similarly, based on the pivot locations and the values in the right column, we see that we are at the vertex given by $(s_1, x_2) = (20, 30)$, or by $(x_1, x_2) = (0, 30)$.

We now identify the next pivot, x_1, as there is still room to optimize in the objective function by loosening this variable. Performing the given row operations in this column

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

yields

$$\begin{array}{c} \\ f \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cccccc} f & x_1 & x_2 & s_1 & s_2 & \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 100 & 300 & 14000 \\ 0 & 1 & 0 & 1 & -1 & 20 \\ 0 & 0 & 1 & 0 & 1 & 30 \end{array}\right] \end{array}. \tag{3.29}$$

Now we see that there is no more room to optimize, so we have found a local, and hence global optimum solution. The objective function at this pivot is 14000. Similarly, based on the pivot locations and the values in the right column, we see that we are at the vertex $(x_1, x_2) = (20, 30)$.

Exceptions and Different Problem Types

The simplex tableau method is a workhorse algorithm for linear programming that is robust and flexible. There are some exceptions and edge cases that require special attention.

Unbounded Problems. If during the minimum ratio test all pivot coefficients are less than zero, or no feasible pivot is found, the objective can increase indefinitely, indicating an unbounded linear program.

Degeneracy. Sometimes multiple constraints become active at once, or the ratio test yields a tie. This can cause *cycling*, where we pivot among the same vertices. Solvers use anti-cycling rules, like Bland's rule, to break these ties.

Phase I and Artificial Variables. Sometimes an LP does not naturally fit into slack form with $\mathbf{x} \geq 0$ and $\mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b}$, for example if we have equality constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$, or constraints that cannot directly yield a feasible basis of slack variables. In such cases, it may not be obvious how to initialize the tableau. We introduce *artificial variables* to find a feasible solution about which to initialize. We run a "Phase I" simplex to find a feasible solution, then remove the artificial variables and proceed with a standard tableau method in "Phase II."

We begin by adding artificial variables to each constraint that lacks a straightforward slack variable. For instance, in an equality constraint $\mathbf{a}_i^\top \mathbf{x} = b_i$, we insert an *artificial* variable $r_i \geq 0$ so the row becomes $\mathbf{a}_i^\top \mathbf{x} + r_i = b_i$. The variable r_i must be driven to zero in any feasible solution, since it is purely artificial to "bridge" the gap if we cannot otherwise pivot into feasibility.

Phase I involves solving an auxiliary LP where the objective is to minimize the sum of all artificial variables:

$$\min \sum_i r_i$$

subject to the same constraints. If the Phase I optimum is zero, a feasible solution to the original problem exists, and the artificial variables can be made inactive. If the minimum is greater than zero, then no feasible solution exists.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We then proceed to Phase II, which involves removing the artificial variables from the model (they should all be zero if feasibility is possible) and proceeding with the original objective. Now we have a valid feasible basis from Phase I, we can run the usual simplex steps to optimize the real cost function.

Hence, *Phase I* is purely about *finding feasibility*, while *Phase II* solves the original LP from that feasible starting point. Once $r_i = 0$ for each artificial variable, they no longer appear in the constraints or objective, effectively removing them from the model.

Minimization, Mixed Constraints, or Upper Bounds. Minimization can be turned into maximization by negating the objective. Upper bounds on variables can be introduced or turned into slack variables. In principle, the simplex tableau is flexible for all these variations and the same tableau steps apply.

Large-Scale or Interior-Point Methods. For extremely large problems, the tableau method can become memory-intensive or slow. Modern LP solvers often switch to interior-point methods, which will be discussed more in Chapter 5. Still, the simplex tableau remains conceptually simple, robust, and widely used for mid-sized linear programs.

3.5 Duality in Linear Programming

One of the most elegant and useful results in optimization is the concept of *duality*. We will discuss duality in much more detail in Chapter 5, and it will be the foundation of interior point methods. We introduce duality briefly here because it is particularly intuitive and straightforward in the case of a linear program. For every linear program (called the *primal*), there is an associated *dual* linear program whose objective and constraints reflect the primal's constraint structure and objective variables, but swapped in a systematic way.

Consider the following primal problem with decision variables $x_1, x_2 \geq 0$:

$$\begin{aligned} \max \quad & c_1 x_1 + c_2 x_2, && \text{(3.30a)} \\ \text{subject to:} \quad & -x_1 + x_2 \leq b_1, && \text{(3.30b)} \\ & 2x_1 - x_2 \leq b_2. && \text{(3.30c)} \end{aligned}$$

To build its dual, we introduce one dual variable for each primal constraint. Let y_1 correspond to the first constraint ($-x_1 + x_2 \leq b_1$) and y_2 correspond to the second ($2x_1 - x_2 \leq b_2$). The *dual problem* then becomes:

$$\begin{aligned} \min \quad & b_1 y_1 + b_2 y_2, && \text{(3.31a)} \\ \text{subject to:} \quad & y_1,\, y_2 \geq 0, && \text{(3.31b)} \\ & -y_1 + 2y_2 \geq c_1, && \text{(3.31c)} \\ & y_1 - y_2 \geq c_2. && \text{(3.31d)} \end{aligned}$$

Observe that each *primal constraint* has become a *dual variable*, and each *primal variable* x_j is constrained by a corresponding inequality in the dual problem. In short, each constraint in the primal becomes a variable in the dual, and each variable in the primal becomes a constraint in the dual.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

General Form: Primal and Dual

More generally, for the primal LP

$$\begin{aligned} \max \quad & \mathbf{c}^\top \mathbf{x}, && \text{(3.32a)} \\ \text{subject to:} \quad & \mathbf{x} \geq 0, && \text{(3.32b)} \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b}, && \text{(3.32c)} \end{aligned}$$

the dual is given by

$$\begin{aligned} \min \quad & \mathbf{b}^\top \mathbf{y}, && \text{(3.33a)} \\ \text{subject to:} \quad & \mathbf{y} \geq 0, && \text{(3.33b)} \\ & \mathbf{A}^\top \mathbf{y} \geq \mathbf{c}. && \text{(3.33c)} \end{aligned}$$

This pairing reveals several key properties:

- *Weak duality*: For any feasible primal $\mathbf{x}$ and any feasible dual $\mathbf{y}$, we have
$$\mathbf{c}^\top \mathbf{x} \leq \mathbf{b}^\top \mathbf{y}.$$
- *Strong duality*: Under mild conditions, such as the existence of an optimal basic feasible solution in the primal, the maximum primal objective and minimum dual objective coincide at optimality.

The form of duality above, with $\mathbf{c}^\top \mathbf{x} \leq \mathbf{b}^\top \mathbf{y}$ is a statement of pointwise duality for any feasible pair. This is a direct inequality holding for all pairs $(\mathbf{x}, \mathbf{y})$ satisfying their respective primal and dual constraints. It doesn't explicitly involve $\max$ or $\min$, but simply says that no feasible primal solution can exceed in value the cost of any feasible dual solution.

It is also possible to write weak duality in terms of the optimal values. If we take the maximum (or supremum, if it doesn't exist) over all feasible $\mathbf{x}$ and the minimum (or infimum, if it doesn't exist) over all feasible $\mathbf{y}$, we obtain the well-known *weak duality* statement:

$$\max_{\mathbf{x}} \left\{ \mathbf{c}^\top \mathbf{x} \right\} \leq \min_{\mathbf{y}} \left\{ \mathbf{b}^\top \mathbf{y} \right\}.$$

Here, $\max$ refers to the optimal primal objective, if it is finite and attained, and $\min$ refers to the optimal dual objective, again if it is finite and attained. This is just the aggregated version of the pointwise statement: if $\mathbf{c}^\top \mathbf{x} \leq \mathbf{b}^\top \mathbf{y}$ holds for every feasible pair, then certainly the best (maximum) $\mathbf{x}$ cannot exceed the worst (minimum) $\mathbf{y}$.

If we produce a primal feasible $\mathbf{x}$, we can guess or compute a dual feasible $\mathbf{y}$. If $\mathbf{c}^\top \mathbf{x} = \mathbf{b}^\top \mathbf{y}$, we know we have found an optimal solution by strong duality. If $\mathbf{c}^\top \mathbf{x} < \mathbf{b}^\top \mathbf{y}$, we have a gap and can keep searching in the primal or adjust the dual until these meet. Many algorithms, including interior-point methods, solve primal and dual problems simultaneously. Sometimes the dual is simpler to handle or interpret than the primal, especially in network flow or resource allocation examples.

Duality is a *cornerstone* of linear programming theory, providing both geometric and computational insights that strongly link the primal constraints and objective to those of

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

the dual, ensuring that if one problem is solved optimally, so is the other. There are several important roles of the dual formulation in providing optimality certificates, bounding error, and estimating sensitivity of solutions to parameters. Once both primal and dual solutions meet strong duality, each feasible solution in one problem "certifies" that the corresponding solution in the other problem is optimal. If we only have a near-feasible primal solution, the dual can supply an upper bound on the best objective. This aids in bounding errors and performing sensitivity analysis on how changes in $\mathbf{b}$ or $\mathbf{c}$ shift optimal solutions.

Duality, Supporting Hyperplanes, and Lagrange Multipliers. One way to understand duality is via *supporting hyperplanes*: each dual variable y_i can be thought of as a Lagrange multiplier enforcing the i-th constraint in the primal. The dual objective $\mathbf{b}^\top \mathbf{y}$ measures the total "cost" of satisfying these constraints. Meanwhile, the condition $\mathbf{A}^\top \mathbf{y} \geq \mathbf{c}$ ensures that any attempted linear combination of constraints is sufficient to bound the primal objective $\mathbf{c}^\top \mathbf{x}$ from above.

In more advanced settings like PDE-constrained optimization, robust control, or machine learning, dual variables (Lagrange multipliers) measure how "tight" constraints are. The primal-dual viewpoint underlies the KKT conditions, bridging linear programs to more general optimization problems. These connections will be explored more in Chapter 5.

3.6 Integer Programming and NP-Hard Problems

An important extension of linear programming is the case where some or all of the decision variables must be integer-valued, often $\{0, 1\}$ in the case of a yes/no decision. This transforms a linear program into an integer linear program (ILP), which is typically much harder to solve. In fact, many ILPs turn out to be *NP-hard* problems, meaning that no known polynomial-time algorithm can solve them in the worst case.

An ILP is typically formulated as

$$\begin{aligned} \max \quad & \mathbf{c}^\top \mathbf{x} && \text{(3.34a)} \\ \text{subject to} \quad & \mathbf{A}\,\mathbf{x} \leq \mathbf{b}, && \text{(3.34b)} \\ & \mathbf{x} \in \{0,1\}^n \text{ (or } \mathbb{Z}^n). && \text{(3.34c)} \end{aligned}$$

It is possible to relax this to a linear program by allowing each x_j to be continuous, $x_j \geq 0$. That relaxation is a simpler convex problem, solvable in polynomial time by standard LP methods. However, the solution to the relaxed problem may not satisfy the integrality constraints. Nevertheless, the LP relaxation often provides a bound on the best possible integer solution, since any feasible integer solution is also feasible for the relaxation, the relaxed optimum is an upper bound for $\max$ problems. It also helps guide a branching strategy for exact integer methods, such as branch-and-bound or cutting-plane approaches. Thus, integer programming can be viewed as linear programming plus integrality constraints. These typically require more sophisticated algorithms beyond simplex or interior point. We often rely on these standard LP subroutines *inside* a branch-and-bound or cutting-plane framework to systematically enforce integrality.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Integer programming is an essential extension of linear programming whenever the decision variables must be integral (e.g. selecting flights, discrete schedules, or capacity expansions). These problems typically become NP-hard, though LP-based relaxations remain crucial for bounding and guiding ILP solution algorithms. We will now discuss several integer linear programming problems below, such as the knapsack problem.

The Knapsack Problem

In the knapsack problem, the task is to select a number of items to fill a knapsack in order to maximize the value of the accumulated items while not exceeding the knapsack's weight capacity W. Consider a set of n items, each with a value (profit) p_i and a weight w_i. If item i is selected then $x_i = 1$, and $x_i = 0$ otherwise. The optimization problem of maximizing the total profit without exceeding capacity can be written as:

$$\max \quad \sum_{i=1}^{n} p_i \, x_i \tag{3.35a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i \, x_i \;\leq\; W, \tag{3.35b}$$

$$x_i \in \{0, 1\}. \tag{3.35c}$$

The objective function may be written in vector form as $\mathbf{p}^T\mathbf{x}$ and the constraints may be written at $\mathbf{w}^T\mathbf{x} \leq W$. This integer linear program is famously NP-hard: enumerating all subsets of items grows factorially in n; selecting j objects from n is given by $\binom{n}{m} = \frac{n!}{k!(n-k)!}$. Nonetheless, it is central to many industrial and scheduling applications where a resource limit, like weight or volume, bounds the number of items or tasks that can be selected.

Although this is, in general, combinatorially hard, there are simple relaxations and numerical algorithms to solve. The following code solves the knapsack problem very efficiently using `pulp`:

```
from pulp import *

n = 10
weights = [4,2,5,4,8,5,8,5,10,4]
prices = [15,11,20,12,26,21,29,15,30,12]
C = 20

# initialize problem
model = LpProblem(sense=LpMaximize)
variables = [LpVariable(name=f"x_{i}", cat=LpBinary) for i in range(n
    )]
model += lpDot(weights, variables) <= C
model += lpDot(prices, variables)

# solve
status = model.solve(PULP_CBC_CMD(msg=False))
print("print:", model.objective.value())
print("b_j:", *[variables[i].value() for i in range(n)])
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The knapsack formulation is important as a *prototype* for many resource allocation problems. Whether we are packing boxes into a truck, scheduling tasks on a processor with finite time slots, or allocating some limited budget among competing investments, the structure of "items with value and resource consumption" is the same. Hence, the knapsack problem, and its variants, has driven much research in branch-and-bound methods and heuristics for large-scale combinatorial optimization.

Below are a few notable ILP applications that naturally arise in engineering or industrial contexts:

- **Airline Flight Scheduling and Crew Assignment.** Airlines must decide how many planes to assign to each route, how to sequence flight legs, and which crews operate which flights, often subject to maintenance windows, regulated rest times, gate limits, etc. Each route or crew assignment is often modeled via a $0-1$ decision variable; for example, we might introduce a variable x_r that is 1 if route r is flown, and 0 otherwise. Other sets of variables track crew usage, maintenance constraints, or time windows. The objective might be to maximize total profit or minimize total operational costs or delays. Integrality is essential, here, since a flight either runs or does not run, and a plane can only be in one place at once.

- **Job Scheduling.** When scheduling tasks or jobs on a finite set of machines, we must decide integer allocations in both time (job start/end times in discrete slots) and capacity (which machine does a job run on?). If m machines and n jobs exist, one might define x_{ij} to be 1 if job j is assigned to machine i, or x_{jk} to be 1 if job j starts in time slot k. The constraints ensure no two overlapping jobs share the same machine, and each job is done exactly once. The objective might be to maximize throughput.

- **Extended Logistic and Supply Chain Problems.** Beyond the simple knapsack scenario, many supply chain or transportation problems require binary decisions for opening a warehouse location, running a shipping route, or investing in capacity expansions. Each of these decisions appears as a $0-1$ variable (e.g. open vs. closed). Meanwhile, one might have continuous flow variables representing how much product is shipped. The result is a mixed-integer linear program (MILP).

These are all NP-hard problems. The essential difficulty is the discrete nature of certain choices. If the dimension of the integrality constraints grows, enumerating all possibilities can become exponential in size. Traditional polynomial-time LP solvers cannot trivially enforce integrality, so more elaborate tree search (branch-and-bound) or cutting-plane methods are used.

Even so, modern ILP solvers can handle quite large instances efficiently in many real-world cases, combining heuristics, branching strategies, and LP relaxations. But from a theoretical standpoint, ILPs remain hard problems, unlike polynomially solvable LPs.

Auction and Bidding Approaches for Airline Takeoff Scheduling

A further illustration arises in certain airport takeoff or landing slot scheduling scenarios, where each airline holds a set of daily departure times (slots) but may wish to exchange or "jump the line" on specific flights. One real-world scheme grants each airline a limited

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

number of "exception cards" (or priority tokens) per day, letting them move certain high-value flights ahead of others in the queue. This prioritization process, if formulated as a direct combinatorial optimization problem, is typically NP-hard.

To circumvent an exhaustive search, one strategy uses *bidding or auction* mechanisms. Each airline may bid on, or buy from another airline, the right to prioritize one flight using an exception card on a given day, subject to a maximum count of allowed priority uses. Each airline reveals partial valuations for accelerating specific flights (e.g. high-revenue international routes, reducing costly downstream delays and re-bookings, etc). The final allocation emerges from a decentralized, market-based exchange of priority slots, often producing near-optimal solutions in a much more efficient scheme. While not guaranteeing the absolute global optimum, this bidding approach is far more computationally tractable, still captures crucial economic and operational priorities, and yields a practically efficient assignment of departure slots. Thus, leveraging an auction-based relaxation can yield a *good enough* solution.

Traveling Salesman Problem (TSP)

The *traveling salesman problem (TSP)* is another classic NP-hard problem that is closely related to these airline scheduling problems. Given a list of cities or nodes and the pairwise distances or costs of traveling between them, the TSP requires finding a single route that visits each city exactly once, returning to the start, while minimizing total travel distance.

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} c_{ij}\, x_{ij} && \text{(3.36a)} \\ \text{subject to} \quad & x_{ij} \in \{0,1\}, && \text{(3.36b)} \\ & \sum_{j} x_{ij} = 1 \;\; \forall\, i, && \text{(3.36c)} \\ & \sum_{i} x_{ij} = 1 \;\; \forall\, j. && \text{(3.36d)} \end{aligned}$$

Here, $x_{ij} = 1$ if the path goes directly from city i to city j, and $x_{ij} = 0$ otherwise; E denotes the set of edges connecting cities. Ensuring we do not create smaller subtours (i.e., cycles not visiting all nodes) requires additional *subtour elimination constraints*, making the TSP a complicated integer linear program.

Much like airline scheduling, where we route planes among airports, TSP variants appear in supply-chain networks to optimize the schedule of a fleet of trucks. Whenever we want a single vehicle or agent to visit a set of locations exactly once, or with minimal repeated visits, the TSP arises.

Because the TSP is NP-hard, exact algorithms often employ branch-and-cut or specialized heuristics. One can also define a *linear programming relaxation*, for instance ignoring the integer constraints on x_{ij} or relaxing the subtour constraints. Though solving the LP relaxation yields only a fractional solution, it can still give a valuable *lower bound* on the route cost, guiding branch-and-bound methods in searching for an optimal integer solution.

While the TSP can be formulated as a straightforward MILP, real-world instances are frequently too large for naive exact methods. Instead, practitioners combine strong LP re-

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

laxations, such as branch-and-cut, specialized cutting planes, such as subtour cuts, and other heuristics, such as Lin–Kernighan, genetic algorithms, simulated annealing, etc. Even though TSP is among the most famous NP-hard problems, its structure is well studied, and modern heuristics can handle extremely large instances effectively in practice.

Geometric Packing and Floor Arrangement

Beyond canonical problems like the knapsack or TSP, there are also *geometric* packing and layout tasks that combine integer decisions with geometric constraints. For instance, arranging equipment on a factory floor may demand discrete variables indicating whether or not to include certain machines, akin to a knapsack choice, while also incorporating geometric feasibility constraints: no overlapping pieces, no exceeding the factory boundary, and possibly minimum aisle spaces for safety.

In such a scenario, each piece of equipment i might have a binary variable $x_i \in \{0, 1\}$ that represents whether we decide to place that item at all. There will also be continuous variables (X_i, Y_i) for the position of item i on the floor plan. We thus obtain a *mixed-integer* geometric model:

$$\max \quad \sum_{i=1}^{n} v_i \, x_i \tag{3.37a}$$

$$\text{subject to} \quad x_i \in \{0, 1\}, \quad (X_i, Y_i) \in \mathbb{R}^2 \tag{3.37b}$$

plus geometric packing constraints such as no objects overlapping:

$$\begin{aligned} &\|(X_i - X_j\| \geq L_i + L_j + \delta \\ &\|(Y_i - Y_j\| \geq H_i + H_j + \delta \quad \text{for any } i \neq j \text{ with } x_i = x_j = 1, \end{aligned}$$

and boundary constraints ensuring that each item stays within the floor edges. In the above, L_i is the length of object i and H_i is the height, assuming it fits inside of a simple rectangle. In reality, the geometric constraints may be considerably more sophisticated.

This mixture of integer and geometric constraints tends to be *even more complex* than standard integer linear programs, because overlap constraints are often nonlinear. One approach is to approximate or discretize the floor into a grid, leading to a large 0–1 layout model. These geometric ILPs or nonlinear IPs are used in several contexts, such as: factory floor planning to decide on which machines to include (integer decisions) and where to place them (continuous geometry); in warehouse layout to determine item placements on a 2D or 3D grid subject to collision constraints; and VLSI chip design which involves a 2D or 3D packing of logic blocks with integrality and adjacency constraints. In all cases, the geometry combined with integrality can make the problem *extremely challenging*, and practitioners rely on approximation schemes, heuristics, or specialized constraint solvers that handle both the discrete choices and the continuous layout constraints. Thus, while these are arguably "packing" variants of integer programming, they require more sophisticated geometric reasoning and typically are not purely linear.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) Linear Programming and Extensions, by G. B. Dantzig, 1963 [88].

(2) Introduction to Linear Optimization, by D. Bertsimas and J. N. Tsitsiklis, 1997 [36].

(3) Integer and Combinatorial Optimization, by G. L. Nemhauser and L. A. Wolsey, 2014 [372].

(4) Theory of Linear and Integer Programming, by A. Schrijver, 1998 [315].

(5) Introduction to Operations Research, by F. S. Hillier, and G. J. Lieberman, 2021 [168].

(6) Combinatorial Optimization: Theory and Algorithms, by B. Korte, and J. Vygen, 2018 [206].

(7) Knapsack Problems: Algorithms and Computer Implementations, by S. Martello and P. Toth, 1990 [242].

Papers and reviews

(1) Maximization of a Linear Function of Variables Subject to Linear Inequalities, by G. B. Dantzig, *in T.C. Koopmans (Ed.) Activity Analysis of Production and Allocation*, 1951 [87].

(2) A New Polynomial-Time Algorithm for Linear Programming, by N. Karmarkar, *Combinatorica*, 1984 [197].

(3) Outline of an Algorithm for Integer Solutions to Linear Programs, by R. E. Gomory, *Bulletin of the American Mathematical Society*, 1960 [154].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

For the next several problems, consider the following linear program:

$$\begin{aligned} \max_{\mathbf{x}} \; & x_1 + 2x_2 \\ \text{s.t.} \; & x_1, x_2 \geq 0 \\ & x_1 \leq 300 \\ & x_2 \leq 400 \\ & x_1 + x_2 \leq 500. \end{aligned}$$

Exercise 3–1: Draw the feasible region and identify all of the vertices. Which vertex has the maximum value?

Exercise 3–2: For each vertex, write down the inequality relationships that are active at that vertex. Also write them in slack variable notation.

Exercise 3–3: Solve this problem using Dantzig's simplex method.

Exercise 3–4: Solve this problem using the simplex tableau method, and compare with the answer from above.

Exercise 3–5: Write this linear program in dual form, interpret the dual variables, and solve using any method you'd like. Do we have weak or strong duality?

Exercise 3–6: Write the dual of the dual form of this problem from the last exercise and show that it is the same as the original linear program.

Exercise 3–7: Repeat all of the problems above for the following linear program:

$$\begin{aligned} \max_{\mathbf{x}} \; & 7x_1 + 8x_2 + 10x_3 \\ \text{s.t.} \; & x_1, x_2, x_3 \geq 0 \\ & 2x_1 + 3x_2 + 2x_3 \leq 1000 \\ & x_1 + x_2 + 2x_3 \leq 800. \end{aligned}$$

Exercise 3–8: Suppose you are a farmer choosing how to allocate land to farm corn and agave. You have a total of 50 acres. Corn yields a profit of $\$100$ per acre, while agave yields a profit of $\$400$ per acre. However, there is a maximum demand of 30 acres worth of agave. How much of each crop should you produce?

Formulate this as a linear program and solve using any method you like. Draw a picture of the feasible region with the vertices.

Exercise 3–9: Go through several of the examples from the first page of this chapter and formulate a linear program in standard form: what is the objective function to maximize or minimize, what are the constrained resources, and what are the optimization variables?

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 3–10: Consider the following linear program with 10 constraint equations and 3 variables:

$$\max_{\mathbf{x}} \; x_1 + x_2 + x_3 \tag{3.38a}$$
$$\text{s.t. } \mathbf{x} \geq 0 \tag{3.38b}$$
$$x_1 \leq 3 \tag{3.38c}$$
$$x_2 \leq 4 \tag{3.38d}$$
$$x_3 \leq 5 \tag{3.38e}$$
$$x_1 + x_2 \leq 5 \tag{3.38f}$$
$$x_1 + x_3 \leq 5 \tag{3.38g}$$
$$x_2 + x_3 \leq 5 \tag{3.38h}$$
$$x_1 + x_2 + x_3 \leq 6. \tag{3.38i}$$

Write down all vertices, and please indicate which constraint equations are active for each vertex.

Exercise 3–11: Construct a 2D polytope in the (x_1, x_2)-plane by intersecting 4 or more half-planes.

1. Sketch this polytope.
2. Formulate a linear objective you wish to minimize or maximize.
3. Identify all corner (extreme) points and check which one(s) optimize your objective.

Exercise 3–12: Graphical Solution. Solve the following LP graphically:

$$\min \; z = 3x + 4y$$

subject to

$$\begin{cases} x + 2y \geq 6, \\ x \geq 0, \; y \geq 0. \end{cases}$$

(a) Sketch the feasible region. (b) Identify all extreme points. (c) Determine which point(s) minimize z. (d) Verify if the solution is bounded and feasible.

Exercise 3–13: Simplex Tableaux. Solve by the *Simplex Method*:

$$\max \; z = 2x_1 + 3x_2$$

subject to

$$\begin{cases} x_1 + x_2 \leq 8, \\ x_1 + 3x_2 \leq 15, \\ x_1, x_2 \geq 0. \end{cases}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(a) Convert to standard form. (b) Set up the initial simplex tableau. (c) Perform pivot steps until optimal. (d) Report the optimal basis and final solution (x_1^*, x_2^*) and z^*.

Exercise 3–14: Economic Meaning of Dual Prices. Suppose you have the primal:

$$\max\ z = 5x_1 + 3x_2$$

subject to

$$\begin{cases} 2x_1 + x_2 \le 20, \\ 3x_1 + 4x_2 \le 48, \\ x_1, x_2 \ge 0. \end{cases}$$

(a) Write the dual. (b) Solve the primal and dual. (c) Interpret the dual variables as *shadow prices* or *resource valuations*. (d) Suppose the right-hand side in the first constraint changes from 20 to 21. Predict the new optimal value z^* using dual prices.

Exercise 3–15: Multiple Solutions and Degeneracy. Solve the LP:

$$\max\ z = x + 2y$$

subject to

$$\begin{cases} x + y \le 8, \\ 2x + 2y \le 16, \\ x, y \ge 0. \end{cases}$$

(a) Show that the feasible region has lines of solutions. (b) Find the entire set of optimal solutions (since it's multiple). (c) Discuss degeneracy in the simplex method for this example. (d) Solve with standard simplex steps and see how the method handles ties.

Exercise 3–16: Dimensional Arguments. Suppose you have an LP in $n = 5$ decision variables and $m = 3$ constraints (plus nonnegativity).

1. How many dimensions is the feasible region?
2. What is the maximum possible number of extreme points? (Trick question—explain carefully if it is infinite, finite, etc.)
3. Provide a geometric interpretation of the statement "An LP's optimum, if it exists, is attained at a vertex (extreme point)."

Exercise 3–17: Matrix Notation for Simplex. Write the standard form LP as:

$$\max\ \mathbf{c}^\top \mathbf{x} \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \ge 0.$$

(a) Show how pivoting corresponds to *replacing one basis column with a non-basis column*. (b) Illustrate the dictionary approach vs. the matrix approach. (c) Solve a small system using matrix invertibility steps rather than tabular expansions.

Exercise 3–18: Resource Allocation. Formulate a linear program for a small real-world scenario:

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- You have two production lines, each with limited hours.
- You produce three products, each requiring different processing times on each line.
- You must satisfy demand for at least 500 units of product A, 300 of B, 400 of C.
- Your objective is to minimize total operating cost or maximize profit.

(a) Write all decision variables, constraints, and objective in a coherent LP. (b) State assumptions or linear approximations you used. (c) Suggest how you'd solve it and whether integer constraints might be needed.

Exercise 3–19: Solve the following linear program using the **Simplex Method** by hand:

$$\min\ z = 3x_1 + 2x_2 \quad \text{subject to} \quad \begin{cases} x_1 + x_2 \geq 4, \\ 2x_1 + x_2 \geq 6, \\ x_1,\ x_2 \geq 0. \end{cases}$$

Show each step of your tableau.

Exercise 3–20: (Duality) Write the dual of the above linear program. Interpret the dual variables in terms of a real-life scenario (e.g., resource allocation or blending).

Exercise 3–21: Solve by the *Simplex Method*:

$$\max\ z = 2x_1 + 3x_2$$

subject to

$$\begin{cases} x_1 + x_2 \leq 8, \\ x_1 + 3x_2 \leq 15, \\ x_1, x_2 \geq 0. \end{cases}$$

(a) Convert to standard form. (b) Set up the initial simplex tableau. (c) Perform pivot steps until optimal. (d) Report the optimal basis and final solution (x_1^*, x_2^*) and z^*.

Exercise 3–22: Given the primal problem:

$$\max\ 4x_1 + x_2$$

subject to

$$\begin{cases} 2x_1 + 3x_2 \leq 12, \\ x_1 + x_2 \geq 5, \\ x_1, x_2 \geq 0, \end{cases}$$

(a) Convert to a standard form suitable for taking the dual. (b) Write the dual problem explicitly (be mindful of $\leq, \geq$ constraints and signs). (c) Identify which variables in the dual are unrestricted vs. nonnegative if any. (d) State the dual objectives and constraints

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

clearly.

Exercise 3–23: Suppose you have the primal:

$$\max\ z = 5x_1 + 3x_2$$

subject to

$$\begin{cases} 2x_1 + x_2 \le 20, \\ 3x_1 + 4x_2 \le 48, \\ x_1, x_2 \ge 0. \end{cases}$$

(a) Write the dual. (b) Solve the primal and dual. (c) Interpret the dual variables as *shadow prices* or *resource valuations*. (d) Suppose the right-hand side in the first constraint changes from 20 to 21. Predict the new optimal value z^* using dual prices.

Exercise 3–24: Consider the LP:

$$\max\ z = 2x + 5y$$

subject to

$$\begin{cases} x + 2y \le b_1, \\ 3x + 4y \le b_2, \\ x, y \ge 0. \end{cases}$$

(a) Solve it for $(b_1, b_2) = (8, 20)$. (b) Perform sensitivity analysis to find allowable ranges of b_1, b_2 for which the current optimal basis remains optimal. (c) Illustrate how the optimal solution or basis might change if $b_1 > 10$, or $b_2 < 16$.

Exercise 3–25: Now we will investigate the scaling of a linear program as the number of decision variables n and the number of constraint equations m increases.

Exercise 3–26: We will use linear programming for an integer programming task.

Exercise 3–27: Space allocation. A factory has a total floor area of 100,000 square feet that can be allocated among different production lines (e.g., producing *Product A* vs. *Product B*) to maximize total profit. Each product requires a certain minimum area (or at most some maximum area) and has a certain profit (per square foot or per unit produced). Specifically, suppose product A yields \$50 per 100 sq. ft. (i.e. \$0.50 per sq. ft.) and product B yields \$1.00 per sq. ft. Moreover, we require at least 20,000 sq. ft. for product A to keep it viable.

Write this as a linear program and solve using any method you like.

Exercise 3–28: Portfolio management. An investor chooses how much capital to allocate among different assets (e.g., stocks, bonds, etc.) to maximize expected return. Each asset has an expected return (per dollar invested), but also a maximum allowable fraction or total risk limit.

Suppose the investor has \$100,000 total to distribute among three assets.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- Asset 1 has expected return 8% (0.08) per year, max \$60,000,
- Asset 2 has expected return 6% per year, no explicit max,
- Asset 3 has expected return 11% but at most \$20,000 can be placed to keep risk moderate.

Formulate an LP to maximize total annual return subject to $\sum x_i \leq 100000$ and the given max constraints on x_1 and x_3.

Exercise 3–29: Network flow bandwidth allocation. A communications network has edges (links) each with a certain bandwidth capacity. We want to push as much data flow from a main source to a main sink as possible (or from multiple sources to multiple sinks), subject to capacities. The objective is to maximize total flow.

Suppose a simple network:

- Node set: {source s, node A, node B, sink t}}.
- Edges:
  - (s, A) capacity 10,
  - (s, B) capacity 5,
  - (A, B) capacity 3,
  - (A, t) capacity 6,
  - (B, t) capacity 7.

Formulate the max-flow LP, then solve for flow on each edge to find total capacity from s to t.

Exercise 3–30: Fleet scheduling. An airline must decide how to allocate planes across different routes or time blocks from a single airport. Each route has a certain revenue per plane, and there's a limited number of planes available. The objective is to maximize total revenue, subject to demand or flight constraints.

Let the airline have 10 planes total. There are 3 routes:

- Route 1: revenue \$3000 per plane, max 6 planes,
- Route 2: revenue \$4000 per plane, max 4 planes,
- Route 3: revenue \$2500 per plane, no explicit max but the total is 10.

Formulate an LP (or ILP, for integer planes) to assign planes to each route for maximum total revenue.

Exercise 3–31: Solve the following linear program using Dantzig's simplex method.

Exercise 3–32: Solve the same linear program using the simplex tableau method.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 3–33: Artificial Variables. Solve the following LP using either the *Two-Phase Method* or *Big-M Method*:

$$\min\ z = x_1 + 2x_2 + x_3$$

subject to

$$\begin{cases} x_1 + x_2 + x_3 = 10, \\ 2x_1 + 3x_2 + x_3 \geq 15, \\ x_1, x_2, x_3 \geq 0. \end{cases}$$

Make sure to show each tableau or step clearly.

Exercise 3–34: Parametric Sensitivity. Consider the LP:

$$\max\ z = 2x + 5y$$

subject to

$$\begin{cases} x + 2y \leq b_1, \\ 3x + 4y \leq b_2, \\ x, y \geq 0. \end{cases}$$

(a) Solve it for $(b_1, b_2) = (8, 20)$. (b) Perform sensitivity analysis to find allowable ranges of b_1, b_2 for which the current optimal basis remains optimal. (c) Illustrate how the optimal solution or basis might change if $b_1 > 10$, or $b_2 < 16$.

Exercise 3–35: Knapsack Problem. Show that the 0–1 knapsack problem can be formulated as an integer LP:

$$\max \sum_{i=1}^{n} v_i x_i$$

subject to

$$\sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}.$$

(a) State the real-life meaning of each variable, constraint, and objective. (b) Solve a small instance ($n = 4$ or 5) by enumeration or a solver. (c) Is it a polynomial-time solvable problem or NP-hard? Briefly justify.

Exercise 3–36: ILP to LP Relaxation. Consider the integer program:

$$\min\ 4x_1 + 5x_2 \quad \text{subject to} \quad \begin{cases} 2x_1 + x_2 \geq 8, \\ x_1 + 3x_2 \geq 12, \\ x_1, x_2 \in \{0, 1, 2, 3, \dots\}. \end{cases}$$

(a) Formulate the LP relaxation by allowing $x_1, x_2 \geq 0$ real. (b) Solve the relaxation. (c) Show how a naive rounding might yield an infeasible or suboptimal integer solution. (d) Find an approach that tries to fix this via a better *rounding* or *branch-and-bound* approach.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 3–37: Branch-and-Bound Basics. Outline how you would apply a branch-and-bound method to solve:

$$\max\ z = 3x_1 + x_2 \quad \text{subject to} \quad \begin{cases} 2x_1 + 5x_2 \le 20, \\ x_1, x_2 \in \mathbb{Z}_{\ge 0}. \end{cases}$$

(a) Solve the LP relaxation. (b) Branch on a variable that is fractional in the solution. (c) Use bounding to prune subproblems if possible. (d) Compare final integer solution to the relaxed optimum.

Exercise 3–38: Cycling Example. Provide an LP example known to cause *cycling* in the standard simplex method (e.g., a small 2D or 3D example). (a) Show how pivoting in a naive rule enters a cycle. (b) Implement Bland's rule (choose the entering variable with smallest index, etc.) and show that it resolves the cycle. (c) Summarize why cycling is rare but can occur.

Exercise 3–39: Gomory Cuts for an Integer LP. Let us revisit an ILP:

$$\max\ x_1 + 3x_2 \quad \text{subject to} \quad \begin{cases} 2x_1 + x_2 \le 4, \\ x_1, x_2 \in \mathbb{Z}_{\ge 0}. \end{cases}$$

(a) Solve the LP relaxation. If fractional, use a Gomory fractional cut. (b) Show how the cut is derived from the final tableau. (c) Add it to the system, re-solve, and see if an integer solution emerges.

Exercise 3–40: Solve the LP:

$$\max\ z = x + 2y$$

subject to

$$\begin{cases} x + y \le 8, \\ 2x + 2y \le 16, \\ x, y \ge 0. \end{cases}$$

(a) Show that the feasible region has lines of solutions. (b) Find the entire set of optimal solutions (since it's multiple). (c) Discuss degeneracy in the simplex method for this example. (d) Solve with standard simplex steps and see how the method handles ties.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 4

Least-Squares Regression

Least-squares regression is one of the most foundational and ubiquitous concepts in applied mathematics and statistics, with applications in data analysis, inverse problems, control theory, and machine learning. The least-squares regression problem involves solving for the unknown parameters $\mathbf{x}$ of a model that is written as a linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{4.1}$$

For example, each row of this equation may represent demographic information and health outcomes for individuals in a medical trial. In this case, the columns of $\mathbf{A}$ include demographic and behavioral information (age, sex, height, weight, tobacco use, etc.), and the entries in $\mathbf{b}$ represent a health outcome that we want to predict, such as probability of heart disease. The coefficients $\mathbf{x}$ tell us how to weight the entries of $\mathbf{A}$ to predict $\mathbf{b}$.

Linear least-squares regression is a convex optimization problem, where the model parameters $\mathbf{x}$ are chosen to minimize the sum-squared error of the fit over the data

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2. \tag{4.2}$$

For an invertible matrix $\mathbf{A}$ this amounts for a matrix inversion problem (see Fig. 4.1, but when $\mathbf{A}$ is non-square, this requires an optimization solution to find the *best* fit (see Fig. 4.2). Typically the problem will either be *over determined*, meaning that no $\mathbf{x}$ will exactly solve this equation, or *under determined*, meaning that infinitely many $\mathbf{x}$ vectors solve the equation. In both cases, we will formulate an optimization problem, with appropriate regularization, to solve the least-squares regression problem.

Linear least-squares regression is a rare example of an optimization problem that has an analytic, closed-form solution. It is possible to compute the gradient of the objective function and set it equal to zero, even for non-square matrices $\mathbf{A}$, resulting in the solution

$$\mathbf{x} = \mathbf{A}^\dagger \mathbf{b} \tag{4.3}$$

where $\mathbf{A}^\dagger$ is the Moore-Penrose pseudo-inverse. In practice, we will compute $\mathbf{A}^\dagger$ for large systems of equations using the singular value decomposition (SVD), which is itself an optimal matrix decomposition.

Nonlinear least squares generalizes this problem, where we now optimize over the parameters $\boldsymbol{\theta}$ of a nonlinear function fit $\mathcal{N}_{\boldsymbol{\theta}}(\mathbf{x})$

$$\min_{\boldsymbol{\theta}} \ \|\mathcal{N}_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}\|_2^2. \tag{4.4}$$

DRAFT

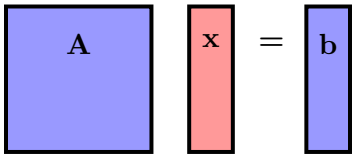


Figure 4.1: Solving a matrix system of equations, $\mathbf{A}\mathbf{x} = \mathbf{b}$, with the same number of equations as unknowns involves matrix inversion.

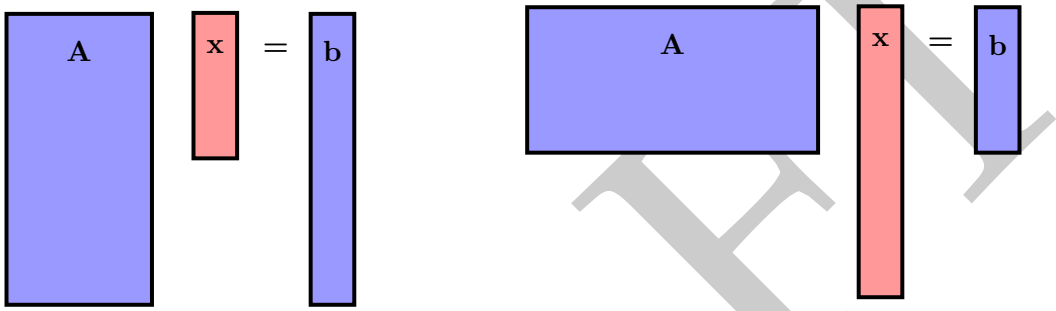


Figure 4.2: The least-squares regression problem may be over-determined (left) so that there are more unknowns than equations, or under-determined (left) so that there are more equations than unknowns.

For example, a neural network is such a function $\mathcal{N}_{\boldsymbol{\theta}}$. In general this is a non-convex optimization problem that requires iterative methods, such as stochastic gradient descent from Chapter 2 or Gauss-Newton methods that will be discussed later in this chapter.

In this chapter, we will begin with the simple linear least-squares regression problem in Eq. (4.2) for the over determined case, where there are more equations than unknowns and we must minimize the sum squared errors. Next we introduce the singular value decomposition, which is how we solve least-squares problems computationally in practice. The SVD is particularly useful for large, low-rank matrices.

At this point we will revisit the under-determined regression case (see Fig. 4.2, right). Since there are typically infinitely many $\mathbf{x}$ that solve a system equations with more unknowns than equations, we will solve for the $\mathbf{x}$ with the minimum norm $\|\mathbf{x}\|_2^2$ in the so-called *minimum-norm* regression problem. This will be formulated as a constrained least-squares problem using Lagrange multipliers, which we will revisit in more depth in Chapter 5. We will also briefly introduce a special class of inverse problems known as *compressed sensing*, where a high-dimensional sparse signal is reconstructed from surprisingly few measurements. This may be viewed as a minimum-norm regression problem where we minimize the $\|\mathbf{x}\|_1$ norm to promote sparsity. This example will be revisited and developed in homework problems in Chapters 6 and 8.

The SVD naturally leads to a discussion on the condition number of the matrix $\mathbf{A}$, which determines how much small errors on the vector $\mathbf{b}$ get amplified when (pseduo)inverting $\mathbf{A}$ to solve for $\mathbf{x}$. Once we have introduced the notion of ill-conditioned matrix inverse problems, we will discuss how to add regularization to improve the condition number.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We will discuss Tikhonov regularization, also known as ridge regression, and the LASSO regression, which add either $\|\mathbf{x}\|_2^2$ or $\|\mathbf{x}\|_1$ to the objective function to promote regularity or sparsity in the solution $\mathbf{x}$, respectively. These regularization terms decrease the variance in the solution $\mathbf{x}$ given a noisy $\mathbf{b}$, but introduce a bias, in one of our first examples of the *bias-variance tradeoff*. Regularization is a key topic in optimization and these norms will be discussed more in Chapters 6 and 7.

The next section of this chapter involves how to add constraints to the least-squares problem, using Lagrange multipliers and the KKT conditions. This is a topic of central importance, as we often have constraints on the vector $\mathbf{x}$ that must be enforced. We will discuss constraints in much more detail in Chapter 5. In linear control theory (Chapter 9) the dynamics enter as constraints to a quadratic optimization problem, and in machine learning (Chapter 10) we often use KKT least-squares to enforce symmetries and other prior information on the coefficients of a model to make it more physical.

The least-squares problems introduced so far fall into a broad class of optimization called *quadratic programming*. We discuss this large category of optimization problems, how least squares fits in, and some of the leading computational approaches to solving these problems.

Finally, we explore nonlinear least-squares problems introduced in Eq. (4.4), which encompasses a broad class of nonlinear inverse optimization problems. Most of machine learning, such as training a neural network, falls into this class of nonlinear least-squares regression, since we almost always use the sum-squared error (or mean-squared error) as the first and primary term in our loss function. In this chapter, we will introduce more classical approaches to solve these problems, including Gauss-Newton methods and the conjugate gradient method, which is used to solve very large scale least-squares problems. More modern approaches, such as stochastic gradient descent, were introduced in Chapter 2 and will be discussed more in Chapter 10 on machine learning.

Throughout this chapter, we assume that the reader is comfortable with how to compute the gradients of linear and quadratic functions over a vector $\mathbf{x}$. These relationships and derivations are provided in Section 2.6 if a refresher is needed.

4.1 Least-Squares Regression

Given a number of observations of the predictors and outcomes of a system, arranged as rows of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{R}^m$, respectively, regression seeks to find the relationship between the columns of $\mathbf{A}$ that is most consistent with the outcomes in $\mathbf{b}$. This relationship is quantified by the linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

which indicates that the vector $\mathbf{b}$ may be approximated as a linear combination of the columns of $\mathbf{A}$. This linear combination is given by the vector $\mathbf{x} \in \mathbb{R}^n$, which is to be determined. The least-squares objective function in Eq. (4.2) is visualized in two-dimensions in Fig. 4.3. The vector of outcomes, $\mathbf{b}$, is often contaminated with measurement noise, which we typically model as the addition of a vector $\boldsymbol{\epsilon}$ of independent, identically distributed Gaussian white noise: $\mathbf{b} = \mathbf{b}_{\text{true}} + \boldsymbol{\epsilon}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

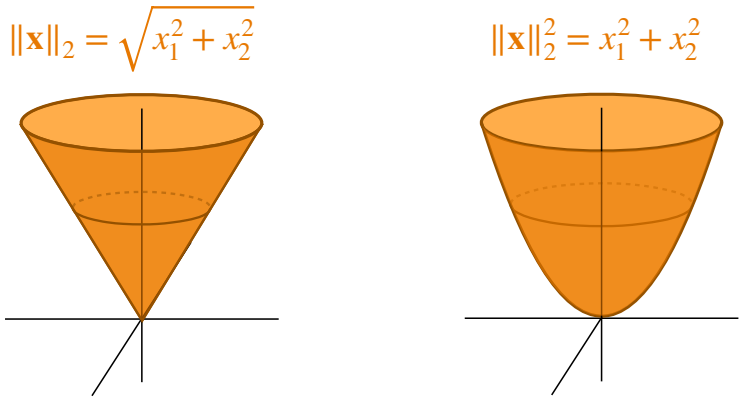


Figure 4.3: The root-mean-square error norm $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2}$ (left) and the sum-squared error norm $\|\mathbf{x}\|_2^2 = x_1^2 + x_2^2$ (right) visualized in two dimensions.

It is possible to solve the linear least-squares problem in Eq. (4.2) analytically by computing the gradient of the objective function and setting it equal to zero. We may expand the objective function as

$$\begin{aligned} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 &= (\mathbf{A}\mathbf{x} - \mathbf{b})^T (\mathbf{A}\mathbf{x} - \mathbf{b}) && \text{(4.5a)} \\ &= \left(\mathbf{x}^T \mathbf{A}^T - \mathbf{b}^T\right) (\mathbf{A}\mathbf{x} - \mathbf{b}) && \text{(4.5b)} \\ &= \mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x} - \mathbf{x}^T \mathbf{A}^T \mathbf{b} - \mathbf{b}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{b}. && \text{(4.5c)} \end{aligned}$$

Computing the gradient and setting equal to zero yields

$$\begin{aligned} \nabla \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 &= 2\mathbf{A}^T \mathbf{A}\mathbf{x} - \mathbf{A}^T \mathbf{b} - \mathbf{A}^T \mathbf{b} = 0 && \text{(4.6a)} \\ \implies \quad \mathbf{A}^T \mathbf{A}\mathbf{x} &= \mathbf{A}^T \mathbf{b} && \text{(4.6b)} \\ \implies \quad \mathbf{x} &= \left(\mathbf{A}^T \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{b}. && \text{(4.6c)} \end{aligned}$$

The matrix $\mathbf{A}^\dagger = \left(\mathbf{A}^T \mathbf{A}\right)^{-1} \mathbf{A}^T$ is known as the Moore-Penrose pseudo-inverse, and it is defined even for non-square matrices $\mathbf{A}$. However, this is only valid when $\left(\mathbf{A}^T \mathbf{A}\right)$ is invertible. The matrix $\mathbf{A}^T \mathbf{A}$ is invertible when $\mathbf{A}$ has n linearly independent columns and $m \geq n$. Thus, this formula is only applicable for the over-determined case in Fig. 4.2. A more technical discussion of exactly when a matrix is over-determined or under-determined is presented at the end of this section.

In practice, we compute this pseudo-inverse for large matrices using the singular value decomposition, which is discussed extensively in the next section. In brief, given the SVD of a real matrix $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

then the Moore-Penrose pseudo-inverse may be written as

$$\mathbf{A}^\dagger = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T \quad \implies \quad \mathbf{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T \mathbf{b}. \tag{4.7}$$

The matrices $\mathbf{U}$ and $\mathbf{V}$ are unitary, so it is simple and efficient to invert them by taking their transpose: $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. Similarly, $\mathbf{\Sigma}$ is a diagonal matrix, which is also easy to invert.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

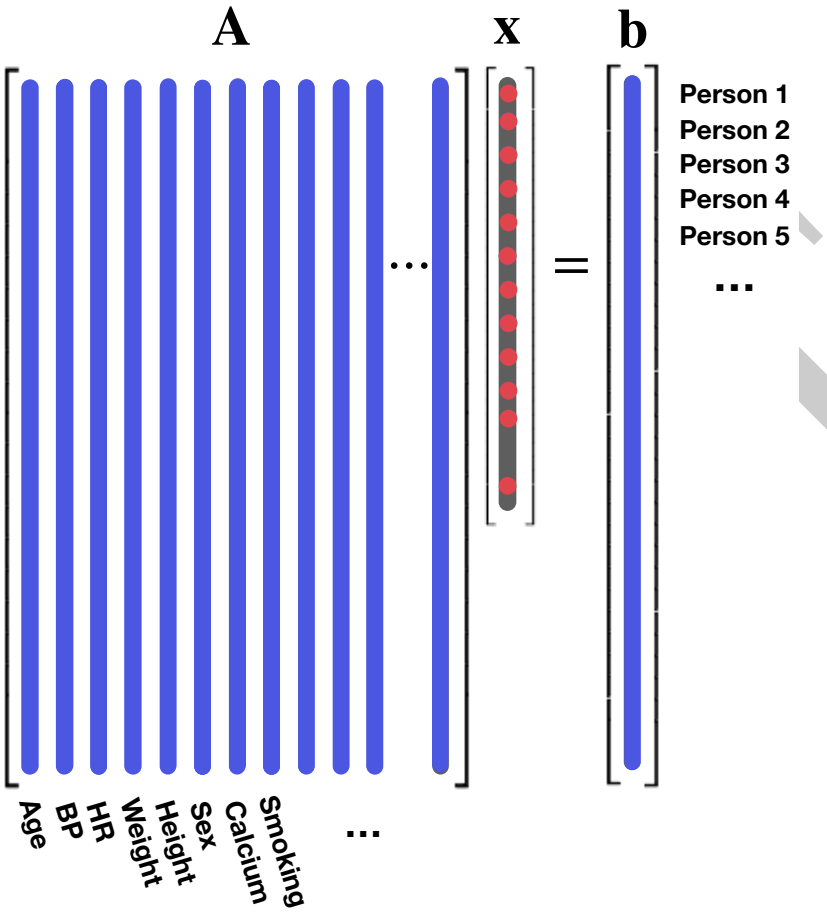


Figure 4.4: Illustrative schematic of how data from a regression problem is formed into matrices and vectors. In this example, alluded to in the introduction to this chapter, we have health data ($\mathbf{A}$) and outcomes ($\mathbf{b}$) for several individual patients. The unknown vector $\mathbf{x}$ represents the model coefficients for how we predict $\mathbf{b}$ given the information in $\mathbf{A}$. In later sections, we will use sparse regularization to find the fewest columns of $\mathbf{A}$ that best correlate with $\mathbf{b}$.

Examples of Least-Squares Regression

Regression is an important statistical tool to relate variables to one another based on data [237]. At the beginning of this chapter we already discussed a motivating example where the matrix $\mathbf{A}$ represents demosgraphics and behavioral data for a number of individual patients, the $\mathbf{b}$ represents health outcomes, and $\mathbf{x}$ represents the prognostic model. This is visualized in Fig. 4.4.

We will now explore several simple computational regression examples in python. All of these examples are unconstrained. We will revisit more complex examples with constraints in Section 4.6.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Example: Random Data. As a first example, we will generate random data $\mathbf{A}$ and $\mathbf{b}$ and will solve for $\mathbf{x}$ using the pseudo-inverse and using the SVD.

```
import numpy as np

# Initialize A (m x n) and b (m x 1).
np.random.seed(42)
m, n = 20, 5
A = np.random.randn(m,n)
b = np.random.randn(m)

# Solve Least Squares Ax=b using SVD
U, S, VT = np.linalg.svd(A,full_matrices=0)
x = VT.T @ np.linalg.inv(np.diag(S)) @ U.T @ b

x = np.linalg.pinv(A)*b    # Alternative
```

Example: Cement Heat Generation. Now, we explore a simple engineering data set that describes the heat generation for various cement mixtures that comprise four basic ingredients. In this problem, we are solving $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{13\times 4}$, since there are four ingredients and heat measurements for 13 unique mixtures. The goal is to determine the weighting $\mathbf{x}$ that relates the proportions of the four ingredients to the heat generation. It is possible to find the minimum error solution using the SVD, as shown in the code below.

```
A = np.loadtxt(os.path.join('..','DATA','hald_ingredients.csv'),
    delimiter=',')
b = np.loadtxt(os.path.join('..','DATA','hald_heat.csv'),delimiter=',
    ')

# Solve Ax=b using SVD
U, S, VT = np.linalg.svd(A,full_matrices=0)
x = VT.T @ np.linalg.inv(np.diag(S)) @ U.T @ b

x = np.linalg.pinv(A)*b    # Alternative
```

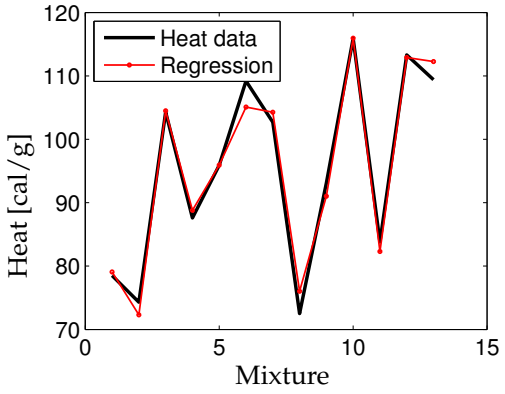
The results of the regression fit are show in Fig. 4.5.

Example: Boston Housing Data. In this example, we explore a larger data set to determine which factors best predict prices in the Boston housing market [162]. This data is available from the UCI Machine Learning Repository [15].

There are 13 attributes that are correlated with house price, such as per-capita crime rate and property-tax rate. These features are regressed onto the price data, the best-fit price prediction is plotted against the true house value in Fig. 4.6. Although the house value is not perfectly predicted, the trend agrees quite well. It is often the case that the highest value outliers are not well captured by simple linear fits, as in this example.

This data contains prices and attributes for 506 homes, so the attribute matrix is of size 506×13. It is important to pad this matrix with an additional column of ones, to take into account the possibility of a non-zero constant offset in the regression formula. This corresponds to the "y intercept" in a simple one-dimensional linear regression. The code

Copyright © July 15, 2025 Brunton. All Rights Reserved.



	Ingredient	Regression
1	Tricalcium aluminate	2.1930
2	Tricalcium silicate	1.1533
3	Tetracalcium alumiferrite	0.7585
4	Beta-dicalcium silicate	0.4863

Figure 4.5: Heat data for cement mixtures containing four basic ingredients.

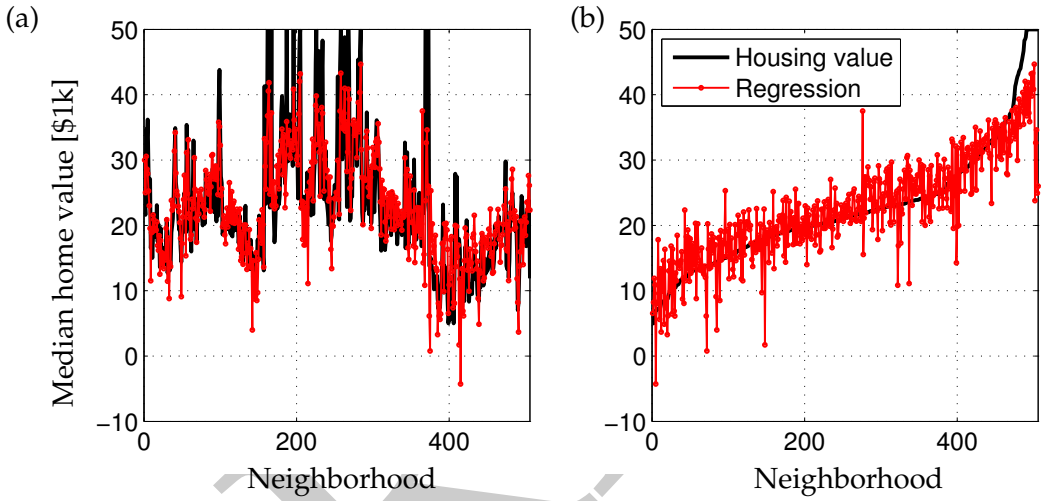


Figure 4.6: Multi-linear regression of home prices using various factors: (a) unsorted data and (b) data sorted by home value.

for this example is nearly identical to the example above, and is available on the book's GitHub.

Outliers and Other Norms

Least-squares regression is sensitive to large outliers in the data, which will skew the entire regression fit. This is because the error associated with outliers are squared, which weights them disproportionally compared with data points having small error. This skewing of the regression fit is shown schematically in Fig. 4.7.

To address the robustness of least squares to outliers, it may be possible to instead minimize the sum of absolute values of the errors,

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_1 \tag{4.8}$$

known as the least absolute deviation (LAD) regression. Because errors are not squared, large outliers do not disproportionally skew the regression. The robust $\|\cdot\|_1$ norm regres-

Copyright © July 15, 2025 Brunton. All Rights Reserved.

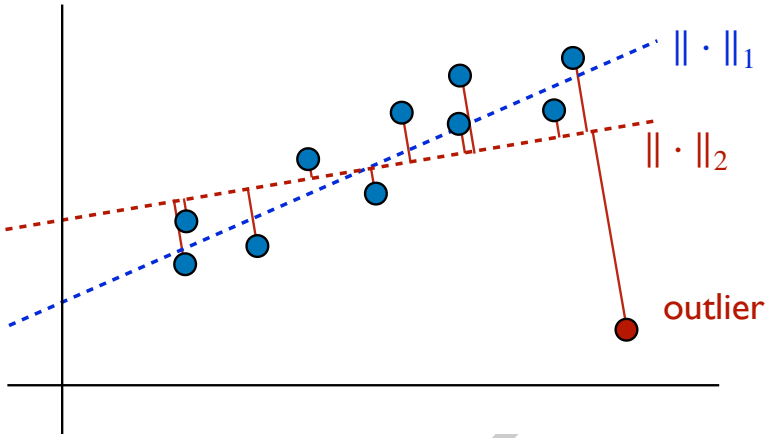


Figure 4.7: Least-squares regression, i.e. minimizing the $\|\cdot\|_2^2$ norm, is sensitive to outliers, as the square of the distance to the outlier skews the entire regression fit. Least absolute deviation regression, i.e., minimizing the $\|\cdot\|_1$ norm, is more robust to outliers.

sion is shown in Fig. 4.7 and Code 4.1. This is a theme in optimization, that typically the $\|\cdot\|_1$ norm is more robust to outliers.

Code 4.1: Use of ℓ_1 norm for robust statistical regression.

```
x = np.sort(4*(np.random.rand(25,1)-0.5),axis=0) # data
b = 0.9*x + 0.1*np.random.randn(len(x),1) # Noisy line y=ax
atrue = np.linalg.lstsq(x,b,rcond=None)[0] # Least-squares a

b[-1] = -5.5  # Introduce outlier
acorrupt = np.linalg.lstsq(x,b,rcond=None)[0] # New slope

## L1 optimization to reject outlier
def L1_norm(a):
    return np.linalg.norm(a*x-b,ord=1)

a0 = acorrupt   # initialize to L2 solution
res = minimize(L1_norm, a0)
aL1 = res.x[0]  # aL1 is robust
```

Interestingly, there has been a major debate about the use of the $\|\cdot\|_2$ norm versus the $\|\cdot\|_1$ norm in regression for over two hundred years. Gauss and Legendre independently introduced the least-squares regression in the early 1800s. However, fifty years earlier, in 1757, Boscovich introduced the LAD regression to solve problems related to estimating the shape of the earth. There have been ongoing debates for the past two centuries about which norm is appropriate for which applications.

Now we have firm mathematical foundations for what prior information these norms quantify, related to Bayesian statistics and inverse problems, which will be discussed more in Chapters 7 and 8, respectively. Roughly speaking, the least-squares regression is an estimator for the mean of a distribution, while the least-absolute-deviation regression es-

Copyright © July 15, 2025 Brunton. All Rights Reserved.

timates the median; the median is known to be more robust to outliers[1]. In fact, the least-squares regression is an *unbiased* estimator for the mean of a distribution assuming Gaussian noise on the measurement vector $\mathbf{b}$. This will be discussed more in Chapter 7.

When is a Matrix Over-determined or Under-determined?

A non-square matrix $\mathbf{A}$ may either be *under-determined* or *over-determined*:

- **Under-determined:** In this case $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $m \ll n$ (i.e., $\mathbf{A}$ is a short-fat matrix), so that there are fewer equations than unknowns. This type of system is likely to have columns that span all of $\mathbb{R}^m$, since it has many more columns than are required for a linearly independent basis.[2] Generically, if a short-fat $\mathbf{A}$ has m linearly independent columns (i.e., its column space spans $\mathbb{R}^m$), then there are infinitely many solutions $\mathbf{x}$ for every $\mathbf{b}$. The system is called *under-determined* because there are not enough values in $\mathbf{b}$ to uniquely determine the higher-dimensional $\mathbf{x}$.
- **Over-determined:** Similarly, consider the *over-determined system*, In this case $\mathbf{A}$ is a tall-skinny matrix with $m \gg n$, so that there are more equations than unknowns. This matrix cannot have m linearly independent columns, and so it is guaranteed that there are vectors $\mathbf{b}$ that have no solution $\mathbf{x}$. In fact, there will only be a solution $\mathbf{x}$ if $\mathbf{b}$ is in the column space of $\mathbf{A}$, i.e., $\mathbf{b} \in \text{col}(\mathbf{A})$.

Technically, there may be some choices of $\mathbf{b}$ that admit infinitely many solutions $\mathbf{x}$ for a tall-skinny matrix $\mathbf{A}$ and other choices of $\mathbf{b}$ that admit zero solutions even for a short-fat matrix. The solution space to the system in (8.5) is determined by the following four fundamental subspaces of $\mathbf{A}$:

- The column space, $\text{col}(\mathbf{A})$, is the span of the columns of $\mathbf{A}$, also known as the *range*.
- The orthogonal complement to $\text{col}(\mathbf{A})$ is $\ker(\mathbf{A}^T)$.
- The row space, $\text{row}(\mathbf{A})$, is the span of the rows of $\mathbf{A}$. The row space of $\mathbf{A}$ is equal to $\text{row}(\mathbf{A}) = \text{col}(\mathbf{A}^T)$.
- The kernel space, $\ker(\mathbf{A})$, is the orthogonal complement to $\text{row}(\mathbf{A})$, and is also known as the *null space*. The null space is the subspace of vectors that map through $\mathbf{A}$ to zero, i.e., $\mathbf{A}\mathbf{x} = \mathbf{0}$.

More precisely, if $\mathbf{b} \in \text{col}(\mathbf{A})$ and if $\dim(\ker(\mathbf{A})) \neq 0$, then there are infinitely many solutions $\mathbf{x}$. Note that the condition $\dim(\ker(\mathbf{A})) \neq 0$ is guaranteed for a short-fat matrix. Similarly, if $\mathbf{b} \notin \text{col}(\mathbf{A})$, then there are no solutions, and the system of equations in (8.5) is called *inconsistent*.

[1] As an example, consider the mean versus median wealth in the United States of America, as of September 2024. The median wealth is $\sim \$200k$, while the mean wealth is $\sim \$1,000k$ because of a relatively small number of extremely wealthy individuals, aka outliers.

[2] It is easy to construct degenerate examples where the columns of a short-fat matrix do not form a full basis for $\mathbb{R}^m$, such as

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

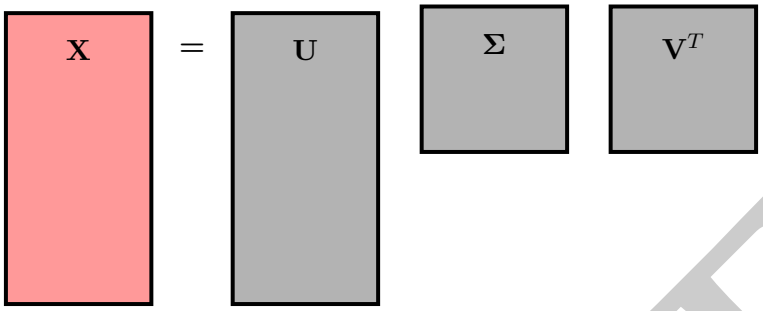


Figure 4.8: The singular value decomposition of a matrix $\mathbf{X}$ provides a hierarchical decomposition into a set of orthogonal rows and columns that best capture the variance in the data.

The fundamental subspaces above satisfy the following properties:

$$\text{col}(\mathbf{A}) \oplus \text{ker}(\mathbf{A}^T) = \mathbb{R}^n, \tag{4.9a}$$

$$\text{col}(\mathbf{A}^T) \oplus \text{ker}(\mathbf{A}) = \mathbb{R}^n. \tag{4.9b}$$

Remark 1 *There is an extensive literature on random matrix theory, where the above stereotypes are almost certainly true, meaning that they are true with high probability. For example, a system* $\mathbf{Ax} = \mathbf{b}$ *is extremely unlikely to have a solution for a random matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and random vector* $\mathbf{b} \in \mathbb{R}^m$ *with* $m \gg n$*, since there is little chance that* $\mathbf{b}$ *is in the column space of* $\mathbf{A}$*. These properties of random matrices play an important role in sparse inverse optimization problems.*

In the over-determined case when no solution exists, we would often like to find the solution $\mathbf{x}$ that minimizes the sum-squared error $\|\mathbf{Ax} - \mathbf{b}\|_2^2$, the so-called *least-squares* solution. Note that the least-squares solution also minimizes $\|\mathbf{Ax} - \mathbf{b}\|_2$. In the under-determined case when infinitely many solutions exist, we may like to find the solution $\mathbf{x}$ with minimum norm $\|\mathbf{x}\|_2$ so that $\mathbf{Ax} = \mathbf{b}$, the so-called *minimum-norm* solution.

4.2 Singular Value Decomposition

The singular value decomposition (SVD) is the technique of choice for large matrix inverse problems. The SVD provides a numerically stable matrix decomposition that can be used for a variety of purposes and is guaranteed to exist. We will use the SVD to obtain optimal low-rank approximations to matrices and to perform pseudo-inverses of non-square matrices to find the solution of a system of equations $\mathbf{Ax} = \mathbf{b}$. The SVD is also the algorithm underlying principal component analysis (PCA), where high-dimensional data is decomposed into its most statistically descriptive factors.

We denote the full SVD as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{4.10}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

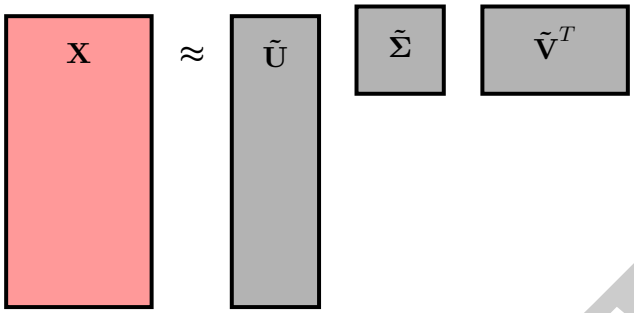


Figure 4.9: The truncated singular value decomposition approximates a matrix $\mathbf{X}$ using the first r singular values and singular vectors.

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are unitary matrices[3] with orthonormal columns, and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a matrix with real, non-negative entries on the diagonal and zeros off the diagonal. The columns of $\mathbf{U}$ are called *left singular vectors* of $\mathbf{A}$ and the columns of $\mathbf{V}$ are *right singular vectors*. The diagonal elements of $\mathbf{\Sigma}$ are called *singular values* and they are ordered from largest to smallest: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. The SVD is visualized in Fig. 4.8.

The above can easily be generalized for complex valued matrices $\mathbf{A} \in \mathbb{C}^{m \times n}$, in which case we use the complex conjugate transpose * instead of the real transpose ^{T}. For real-valued matrices, the complex conjugate transpose is the same as the regular transpose $\mathbf{A}^* = \mathbf{A}^T$.

When $m \geq n$, the matrix $\mathbf{\Sigma}$ has at most n non-zero elements on the diagonal. If there are linearly dependent columns of $\mathbf{A}$, then there will be even less non-zero singular values. If we denote the diagonal matrix of non-zero singular values as $\hat{\mathbf{\Sigma}}$, then it is possible to *exactly* represent $\mathbf{A}$ using the *economy* SVD:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} \hat{\mathbf{U}} & \hat{\mathbf{U}}^\perp \end{bmatrix} \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix} \mathbf{V}^T = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^T. \tag{4.11}$$

The rank of $\mathbf{A}$ is equal to the number of non-zero singular values. The SVD can also be used to obtain an optimal rank-r approximation of $\mathbf{A} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T$, where the rank $r < n$ is chosen to retain the largest singular values. In this case, the matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ contain the first r columns of $\mathbf{U}$ and $\mathbf{V}$ and $\tilde{\mathbf{\Sigma}}$ contains the first $r \times r$ block of $\mathbf{\Sigma}$. This is shown schematically in Fig. (4.9).

The four fundamental subspaces of $\mathbf{A}$ may be related to the $\mathbf{U}$ and $\mathbf{V}$ matrices in the SVD as follows:

- The column space of $\mathbf{A}$ is the same as the column space of $\hat{\mathbf{U}}$.
- The orthogonal complement to the column space of $\mathbf{A}$ is given by the column space of $\hat{\mathbf{U}}^\perp$.
- The row space of $\mathbf{A}$ is spanned by the columns of $\hat{\mathbf{V}}$.
- The kernel space of $\mathbf{A}$ is given by $\text{col}(\hat{\mathbf{V}}^\perp)$.

[3] A square matrix $\mathbf{U}$ is unitary if $\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Least-Squares Regression with the SVD

First, if we substitute an exact truncated SVD $\mathbf{A} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$ in for $\mathbf{A}$, we can "invert" each of the matrices $\hat{\mathbf{U}}$, $\hat{\mathbf{\Sigma}}$, and $\hat{\mathbf{V}}^T$ in turn, resulting in the Moore–Penrose *left pseudo-inverse* [276, 277, 300, 385] $\mathbf{A}^\dagger$ of $\mathbf{A}$:

$$\mathbf{A}^\dagger \triangleq \hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{U}}^T \quad \Longrightarrow \quad \mathbf{A}^\dagger\mathbf{A} = \hat{\mathbf{V}}\hat{\mathbf{V}}^T. \tag{4.12}$$

Note that $\mathbf{A}^\dagger\mathbf{A}$ will only equal the identity $\mathbf{I}_{n\times n}$ if the truncated SVD captures all non-zero singular values; otherwise if we use a rank$-r$ approximation, then $\tilde{\mathbf{V}}\tilde{\mathbf{V}}^T \neq \mathbf{I}_{n\times n}$, and it will only approximate the identity. This may be used to find both the minimum-norm and least-squares solutions to (8.5):

$$\mathbf{A}^\dagger\mathbf{A}\mathbf{x}^* = \mathbf{A}^\dagger\mathbf{b} \quad \Longrightarrow \quad \mathbf{x}^* = \hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{U}}^T\mathbf{b}. \tag{4.13}$$

Plugging the solution $\mathbf{x}^*$ back in to (8.5) results in

$$\mathbf{A}\mathbf{x}^* = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{U}}^T\mathbf{b} \tag{4.14a}$$

$$= \hat{\mathbf{U}}\hat{\mathbf{U}}^T\mathbf{b}. \tag{4.14b}$$

This is a projection of the vector $\mathbf{b}$ onto the space spanned by the columns of $\hat{\mathbf{U}}$. If $\mathbf{b}$ is not in the span of $\mathbf{A}$, and hence in the span of $\hat{\mathbf{U}}$, then $\mathbf{x}^*$ will only approximately solve this equation and there will be some residual error from the components of $\mathbf{b}$ that are not in the span of $\mathbf{A}$.

Computing the SVD

The SVD is computed by first reducing the matrix $\mathbf{A}$ to a bidiagonal matrix and then using an iterative algorithm to compute the SVD of the bidiagonal matrix. For matrices with high aspect ratio (i.e., $m \gg n$), then the first step may be achieved by first computing a QR factorization to reduce $\mathbf{A}$ to an upper triangular matrix, followed by Householder reflections to reduce this upper triangular matrix into a bidiagonal form. The second step may be performed using a modified QR algorithm developed by Golub & Kahan [150]. There are numerous variations, alternatives, and important results on the computation of the SVD; for a more thorough discussion, see [150, 61, 152].

The SVD is computed in Python using the following code:

```
>>> import numpy as np
>>> X = np.random.rand(5, 3)  # create random data matrix
>>> U, S, VT = np.linalg.svd(X,full_matrices=True) #full SVD
>>> Uhat, Shat, VThat = np.linalg.svd(X,full_matrices=False)  #
    economy SVD
```

Computing the pseudo-inverse $\mathbf{A}^\dagger$ is computationally efficient, after the expensive up-front cost of computing the SVD. Inverting the unitary matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ involves matrix multiplications by the transpose matrices, which are $\mathcal{O}(m^2)$ operations. Inverting $\tilde{\mathbf{\Sigma}}$ is even more efficient, since it is a diagonal matrix, requiring $\mathcal{O}(m)$ operations. In contrast, inverting a dense square matrix would require an $\mathcal{O}(m^3)$ operation.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Optimal approximation and error bounds

The SVD is itself the globally optimal solution of an optimization problem. For a given rank r, there is no better approximation for $\mathbf{X}$, in the ℓ_2 sense, than the truncated SVD approximation $\tilde{\mathbf{X}}$. The Eckart-Young theorem below will state this precisely and provide expressions fo the error of the truncated approximation. Thus, high-dimensional data may be well described by a few dominant patterns given by the columns of $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$.

Schmidt (of Gram-Schmidt) generalized the SVD to function spaces and developed an approximation theorem, establishing the truncated SVD $\tilde{\mathbf{X}}$ as the optimal low-rank approximation of the underlying matrix $\mathbf{X}$ [312]. Schmidt's approximation theorem was rediscovered by Eckart and Young [119], and is sometimes referred to as the Eckart-Young theorem.

Theorem 1 (Eckart-Young [119]) *The optimal rank-r approximation to $\mathbf{X}$, in a least-squares sense, is given by the rank-r SVD truncation $\tilde{\mathbf{X}}$:*

$$\underset{\tilde{\mathbf{X}},\, s.t.\ \mathrm{rank}(\tilde{\mathbf{X}})=r}{\operatorname{argmin}} \|\mathbf{X}-\tilde{\mathbf{X}}\|_F = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T. \tag{4.15}$$

Again, $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ denote the first r leading columns of $\mathbf{U}$ and $\mathbf{V}$, and $\tilde{\mathbf{\Sigma}}$ contains the leading $r \times r$ sub-block of $\mathbf{\Sigma}$. The Frobenius norm above is defined as $\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|X_{ij}|^2}$, which is equivalent to the two norm of the vectorized matrix $\mathbf{X}(:)$.

Thus, the Eckart-Young theorem guarantees that the truncated SVD provides the best matrix approximation of a given rank in the Frobenius norm. It is also possible to exactly quantify the error of the rank-r SVD approximation:

$$\|\mathbf{X}-\tilde{\mathbf{X}}\|_F^2 = \sum_{k=r+1}^{m}\sigma_k^2. \tag{4.16}$$

Thus, all other rank-r matrices $\tilde{\mathbf{X}}$ will have at least this much error. Because the error scales with the size and magnitude of $\mathbf{X}$, it is often more useful to consider the relative error

$$\frac{\|\mathbf{X}-\tilde{\mathbf{X}}\|_F^2}{\|\mathbf{X}\|_F^2}. \tag{4.17}$$

This expression for the relative error in the Frobenius norm has two intuitive interpretations. The squared Frobenius norm error of mean-subtracted data has the interpretation of the amount of missing variance in the approximation $\tilde{\mathbf{X}}$, a property that is used extensively in the principal component analysis (PCA).

Remarkably, the SVD also provides an optimal rank-r approximation in the matrix 2-norm, also known as the spectral norm:

$$\underset{\tilde{\mathbf{X}},\, \text{s.t. } rank(\tilde{\mathbf{X}})=r}{\operatorname{argmin}} \|\mathbf{X}-\tilde{\mathbf{X}}\|_2 = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T. \tag{4.18}$$

The 2-norm of a matrix $\mathbf{X}$ given by $\|\mathbf{X}\|_2 = \max_{\mathbf{v}\neq\mathbf{0}}\frac{\|\mathbf{X}\mathbf{v}\|_2}{\|\mathbf{v}\|_2}$. The error expression for the rank-r SVD approximation is even simpler in the 2-norm:

$$\|\mathbf{X}-\tilde{\mathbf{X}}\|_2 = \sigma_{r+1}. \tag{4.19}$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

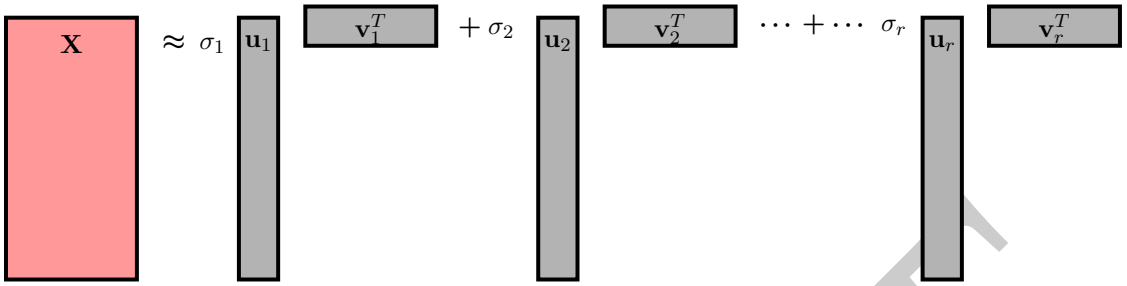


Figure 4.10: The SVD of a matrix $\mathbf{X}$ may be thought of as a sum of rank-1 matrices given by $\sigma_j \mathbf{u}_j \mathbf{v}_j^T$.

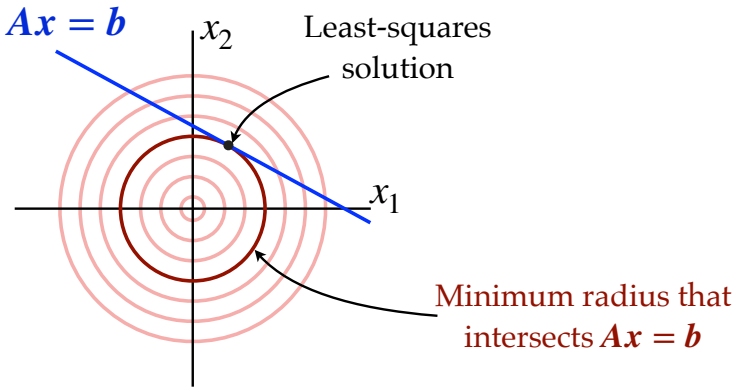


Figure 4.11: Level sets of the ℓ_2 norm are circles, and the least-squares solution gives the point $\mathbf{x}$ on $\mathbf{Ax} = \mathbf{b}$ that intersects the minimum radius level set of this norm.

This error expression is rather simple to derive by expanding

$$\mathbf{X} - \tilde{\mathbf{X}} = \sum_{k=r+1}^{m} \sigma_k \mathbf{u}_k \mathbf{v}_k^T. \tag{4.20}$$

Since each of the $\mathbf{v}_k$ vectors are orthonormal, the maximum $\|(\mathbf{X} - \tilde{\mathbf{X}})\mathbf{v}\|_2 = \sigma_{r+1}$ is achieved for $\mathbf{v} = \mathbf{v}_{r+1}$. In essence, because $\mathbf{\Sigma}$ is diagonal, the SVD may be thought of as a sum of rank-1 matrices, as shown in Fig. (4.10).

4.3 Minimum Norm Regression

The minimum-norm regression problem is similar to the least-squares problem above, except for the case of an under-determined system of equations

$$\mathbf{Ax} = \mathbf{b} \tag{4.21}$$

where $\mathbf{A}$ is a short-fat matrix, so there will be less equations than unknowns. In this case, under mild conditions, there will be infinitely many vectors $\mathbf{x}$ that solve this system of equations.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

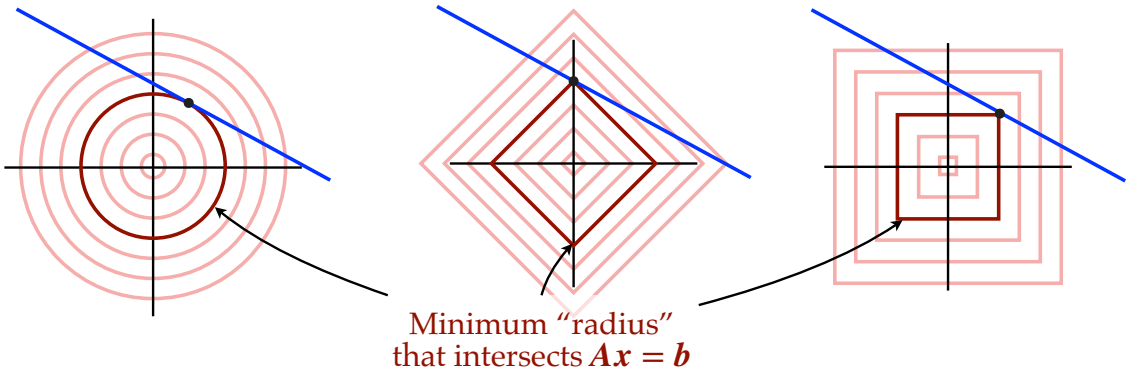


Figure 4.12: The ℓ_2, ℓ_1, and ℓ_∞ norms visualized with the line $\mathbf{Ax} = \mathbf{b}$ in two dimensions.

The minimum-norm regression problem seeks the $\mathbf{x}$ that satisfies the system of equations and has the minimum ℓ_2 norm:

$$\min_{\mathbf{x}} \quad \|\mathbf{x}\|_2^2 \tag{4.22a}$$

$$\text{s.t.} \quad \mathbf{Ax} = \mathbf{b}. \tag{4.22b}$$

Notice here that the system of equations now enters the optimization problem as a constraint. The minimum-norm solution is visualized in Fig. 4.11 for the ℓ_2 norm. This idea may be generalized to other norms, as shown in Fig. 4.12

We may use the augmented Lagrange-multiplier form and expand the objective function $\|\mathbf{x}\|_2^2 = \mathbf{x}^T\mathbf{x}$, resulting in:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{x}^T\mathbf{x} + \boldsymbol{\lambda}^T(\mathbf{Ax} - \mathbf{b}). \tag{4.23}$$

The gradient condition on $\mathbf{x}$ becomes

$$\nabla_{\mathbf{x}}\mathcal{L} = 2\mathbf{x} + \mathbf{A}^T\boldsymbol{\lambda} = 0 \quad \Longrightarrow \quad \mathbf{x} - \frac{1}{2}\mathbf{A}^T\boldsymbol{\lambda} \tag{4.24a}$$

$$\mathbf{Ax} - \frac{1}{2}\mathbf{AA}^T\boldsymbol{\lambda} \tag{4.24b}$$

and the gradient condition on $\boldsymbol{\lambda}$ becomes

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \mathbf{Ax} - \mathbf{b} = 0 \quad \Longrightarrow \quad -\frac{1}{2}\mathbf{AA}^T\boldsymbol{\lambda} = \mathbf{b} \tag{4.25a}$$

$$\Longrightarrow \quad \boldsymbol{\lambda} = -2\left(\mathbf{AA}^T\right)^{-1}\mathbf{b} \tag{4.25b}$$

which leads to

$$\mathbf{x} = -\frac{1}{2}\mathbf{A}^T\boldsymbol{\lambda} = \mathbf{A}^T\left(\mathbf{AA}^T\right)^{-1}\mathbf{b}. \tag{4.26}$$

This is the Moore-Penrose *right* pseudo-inverse:

$$\mathbf{x} = \mathbf{A}^T\left(\mathbf{AA}^T\right)^{-1}\mathbf{b}. \tag{4.27}$$

This may be computed using the singular value decomposition, as in the least-squares regression. If $\mathbf{A} = \mathbf{U\Sigma V}^T$ then $\mathbf{x} = \mathbf{V\Sigma}^{-1}\mathbf{U}^T\mathbf{b}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Compressed Sensing

Most natural signals, such as images and audio, are highly compressible, for example in a Fourier or wavelet basis [56]. This compressibility is the basis of modern image and audio compression, such as JPEG and MP3. When the signal is written in an appropriate basis (e.g., Fourier or wavelet) only a few modes are active, reducing the number of coefficients that must be stored and transmitted for an accurate representation.

However, until recently, images and audio signals were typically collected in full resolution, only to be subsequently compressed for storage and transfer. Recently, within the past twenty years, this paradigm has been turned upside down in what is now known as *compressed sensing*. In compressed sensing, the idea is to collect a small but random subsampling of the original high-dimensional data and then infer which Fourier or wavelet coefficients are large and active in the signal. For example, instead of collecting all million pixels in a megapixel image, it may be possible to collect 10% of the pixels randomly and infer the large Fourier modes in the compressed JPEG image.

Compressed sensing is a very challenging underdetermined regression problem, where instead of solving for the minimum $\|\cdot\|_2$ norm solution, we search for the *sparsest* solution to a system of equations. Here we will set up the basic compressed sensing regression problem and state the associated optimization problem. Then we will have several homework exercises to explore various aspects of this problem in more detail.

A compressible signal $\mathbf{x} \in \mathbb{R}^n$ may be written as a sparse vector $\mathbf{s} \in \mathbb{R}^n$ (containing mostly zeros) in a transform basis $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$:

$$\mathbf{x} = \mathbf{\Psi}\mathbf{s}. \tag{4.28}$$

Specifically, the vector $\mathbf{s}$ is called K-sparse in $\mathbf{\Psi}$ if there are exactly K nonzero elements. If the basis $\mathbf{\Psi}$ is generic, such as the Fourier or wavelet basis, then only the few active terms in $\mathbf{s}$ are required to reconstruct the original signal $\mathbf{x}$, reducing the data required to store or transmit the signal. Images and audio signals are both compressible in Fourier or wavelet bases, so that after taking the Fourier or wavelet transform, most coefficients are small and may be set exactly equal to zero with negligible loss of quality. With knowledge of the sparse vector $\mathbf{s}$ it is possible to reconstruct the signal $\mathbf{x}$ from (4.28).

The goal of compressed sensing is to measure a small subset of the original signal in a measurement vector $\mathbf{y} \in \mathbb{R}^p$ and infer the sparsest vector $\mathbf{s}$ that is consistent with these measurements. The measurements are related to the full state $\mathbf{x}$ via $\mathbf{y} = \mathbf{C}\mathbf{x}$, where $\mathbf{C} \in \mathbb{R}^{p \times n}$ is a measurement matrix with $p \ll n$:

$$\mathbf{y} = \mathbf{C}\mathbf{x} = \mathbf{C}\mathbf{\Psi}\mathbf{s} = \mathbf{\Theta}\mathbf{s}. \tag{4.29}$$

The system of equations in (4.29) is underdetermined since there are infinitely many consistent solutions $\mathbf{s}$. The *sparsest* solution $\hat{\mathbf{s}}$ satisfies the following optimization problem:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_0 \text{ subject to } \mathbf{y} = \mathbf{C}\mathbf{\Psi}\mathbf{s}, \tag{4.30}$$

where $\|\cdot\|_0$ denotes the ℓ_0 pseudo-norm, given by the number of nonzero entries; this is also referred to as the cardinality of $\mathbf{s}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The optimization in (4.30) is non-convex, and in general the solution can only be found with a brute-force search that is combinatorial in n and K. In particular, all possible K-sparse vectors in $\mathbb{R}^n$ must be checked; if the exact level of sparsity K is unknown, the search is even broader. Because this search is combinatorial, solving (4.30) is intractable for even moderately large n and K, and the prospect of solving larger problems does not improve with Moore's law of exponentially increasing computational power.

Fortunately, under certain conditions on the measurement matrix $\mathbf{C}$, it is possible to relax the optimization in (4.30) to a convex ℓ_1-minimization [69, 105]:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_1 \text{ subject to } \mathbf{y} = \mathbf{C}\boldsymbol{\Psi}\mathbf{s}. \tag{4.31}$$

The ℓ_1 norm is also known as the taxicab or Manhattan norm because it represents the distance a taxi would take between two points on a rectangular grid. The ℓ_1 minimum-norm solution is sparse, while the ℓ_2 minimum-norm solution is not. We will discuss convex relaxation in much more detail in Chapter 6.

Alternative formulations

In addition to the ℓ_1-minimization in (4.31), there are alternative optimization approaches based on *greedy algorithms* [343, 344, 346, 345, 348, 167, 347, 142, 349, 140, 260, 141] that determine the sparse solution of (4.29) through an iterative matching pursuit problem. For instance, the compressed sensing matching pursuit (CoSaMP) [260] is computationally efficient, easy to implement, and freely available.

When the measurements $\mathbf{y}$ have additive noise, say white noise of magnitude ε, there are variants of (4.31) that are more robust:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_1, \text{ subject to } \|\mathbf{C}\boldsymbol{\Psi}\mathbf{s} - \mathbf{y}\|_2 < \varepsilon. \tag{4.32}$$

A related convex optimization is the following:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{C}\boldsymbol{\Psi}\mathbf{s} - \mathbf{y}\|_2 + \lambda\|\mathbf{s}\|_1, \tag{4.33}$$

where $\lambda \geq 0$ is a parameter that weights the importance of sparsity. Eqs. (4.32) and (4.33) are closely related [346].

4.4 Condition Number and Regularization

Very often the least-squares regression problem is ill-conditioned, meaning that the columns of $\mathbf{A}$ are nearly linearly dependent. In this case, the pseudo-inverse amplifies noise on the vector $\mathbf{b}$, introducing high variance and a large degree of uncertainty into the estimate $\mathbf{x}$.

In this section, we will formally define the condition number, which is a fundamental concept in regression and optimization that will be used throughout the remainder of this book. We then introduce the Tikhonov regularization, also known as ridge regression, that improves the conditioning of ill-conditioned least-squares problems, decreasing variance but introducing bias. We also introduce the related LASSO regression, which promotes sparsity of the solution vector $\mathbf{x}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Condition Number

The condition number of a matrix $\mathbf{A}$ is a measure of how sensitive matrix multiplication and inversion are to errors in the input. Larger condition number indicates higher sensitivity and worse performance. The condition number $\kappa(\mathbf{A})$ is directly related to the singular values of the matrix:

$$\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})}. \tag{4.34}$$

The condition number is a central concept in all of numerical linear algebra and applied computation. It is easiest to understand the effect of a large condition number when considering the linear system of equations $\mathbf{Ax} = \mathbf{b}$. If the vector $\mathbf{b}$ is not specified perfectly, but instead has some error $\boldsymbol{\epsilon}_\mathbf{b}$, then the system becomes

$$\mathbf{A}(\mathbf{x} + \boldsymbol{\epsilon}_\mathbf{x}) = \mathbf{b} + \boldsymbol{\epsilon}_\mathbf{b}, \tag{4.35}$$

where $\boldsymbol{\epsilon}_\mathbf{x}$ is the corresponding error in $\mathbf{x}$.

If we assume a worst-case scenario, where $\boldsymbol{\epsilon}_\mathbf{b}$ is aligned with the singular vector corresponding to the minimum singular value $\sigma_{\min}$, and the vector $\mathbf{b}$ is aligned with the singular vector corresponding to the maximum singular value $\sigma_{\max}$, then the estimated solution for $\mathbf{x} + \boldsymbol{\epsilon}_\mathbf{x}$ is

$$\mathbf{x} + \boldsymbol{\epsilon}_\mathbf{x} \approx \mathbf{A}^\dagger(\mathbf{b} + \boldsymbol{\epsilon}_\mathbf{b}) = \frac{1}{\sigma_{\max}}\mathbf{b} + \frac{1}{\sigma_{\min}}\boldsymbol{\epsilon}_\mathbf{b}. \tag{4.36}$$

The signal-to-noise of the estimated $\mathbf{x}$ is given by

$$\|\mathbf{x}\|/\|\boldsymbol{\epsilon}_\mathbf{x}\| = \frac{\sigma_{\min}}{\sigma_{\max}}\|\mathbf{b}\|/\|\boldsymbol{\epsilon}_\mathbf{b}\|,$$

which is the signal-to-noise- of $\|\mathbf{b}\|/\|\boldsymbol{\epsilon}_\mathbf{b}\|$ degraded by a factor equal to the condition number $\kappa(\mathbf{A})$. Thus, noise on $\mathbf{b}$ is amplified by the condition number in the inversion process.

Even if the vectors $\mathbf{b}$ and $\boldsymbol{\epsilon}_\mathbf{b}$ are not perfectly aligned with the worst-case directions, they are likely to have some component of all of the singular vector directions. In this case, components of the error will still experience large amplification relative to other components of the desired output. One approach to mitigate a large condition number is to truncate the SVD rank more aggressively, essentially increasing the effective minimum singular value $\sigma_{\min}$. However, this comes at the cost of decreasing the size of the subspace $\tilde{\mathbf{U}}$ used to approximate the output. The following code explores how to truncate the SVD based on a desired condition number:

```
>> kappanew = 1e-5 # Desired condition number
>> U, S, Vt = np.linalg.svd(A, full_matrices=False)
>> r = np.sum(S > np.max(S) * kappanew)
>> invA = Vt[:r,:].T @ np.linalg.inv(np.diag(S[:r])) @ U[:,:r].T #
   Rank-r approximation
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

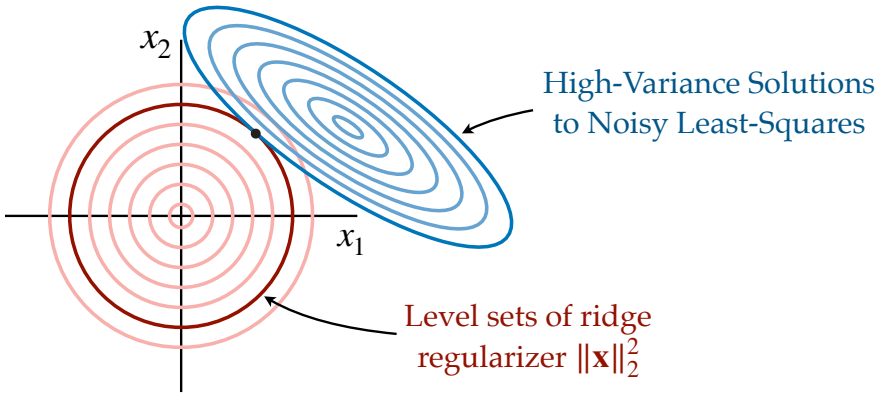


Figure 4.13: Schematic of Tikhonov regularization, aka ridge regression. The original ill-conditioned least-squares problem has high variance, as visualized by the eccentric blue ellipses, which represent the solutions given Gaussian noise on the vector $\mathbf{b}$. Tikhonov regularization pulls these solutions towards level sets of $\|\mathbf{x}\|_2^2$, shown as red circles.

Tikhonov Regularization

Tikhonov regularization, also known as ridge regression, addresses the poor condition number of the least-squares regression when the columns of $\mathbf{A}$ are very nearly parallel and the condition number of the pseudo-inverse is large.

Recall that when $\mathbf{A}$ is non-square we cannot invert it to solve for $\mathbf{x}$ and so instead we compute the pseudo-inverse via the singular value decomposition:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad \Longrightarrow \quad \mathbf{A}^\dagger = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T = \cdots = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T. \tag{4.37}$$

When the columns of $\mathbf{A}$ are nearly parallel, then the condition number of $\mathbf{A}$ is large, and hence the condition number of $\mathbf{A}^T\mathbf{A}$ is even larger. This manifests as the diagonal matrix $\mathbf{\Sigma}$ having nearly degenerate singular values, so that inverting $\mathbf{\Sigma}^{-1}$ involves dividing by very small numbers.

Any noise on $\mathbf{b}$ will be amplified by this ill-conditioned pseudo-inverse. This has the effect of some entries of $\mathbf{x}$ being large and opposite, leading to high variance in the resulting solution. Said another way, if we sample from many instances of small noise on $\mathbf{b}$, we will get a large, highly eccentric ellipsoid of solutions $\mathbf{x}$ centered around the true solution, as shown in Fig. 4.13.

The solution is to regularize the least-squares problem with an additional loss term $\|\mathbf{x}\|_2^2$:

$$\mathbf{x} = \underset{\mathbf{x}'}{\operatorname{argmin}} \|\mathbf{A}\mathbf{x}' - \mathbf{b}\|_2^2 + \alpha\|\mathbf{x}\|_2^2. \tag{4.38}$$

where α is a hyper-parameter that weights this regularizing term.

It is again possible to analytically compute the gradient of this augmented objective function and set it equal to zero, resulting in the analytic solution to the ridge regression problem

$$\mathbf{x} = \left(\mathbf{A}^T\mathbf{A} + \alpha\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{b}. \tag{4.39}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

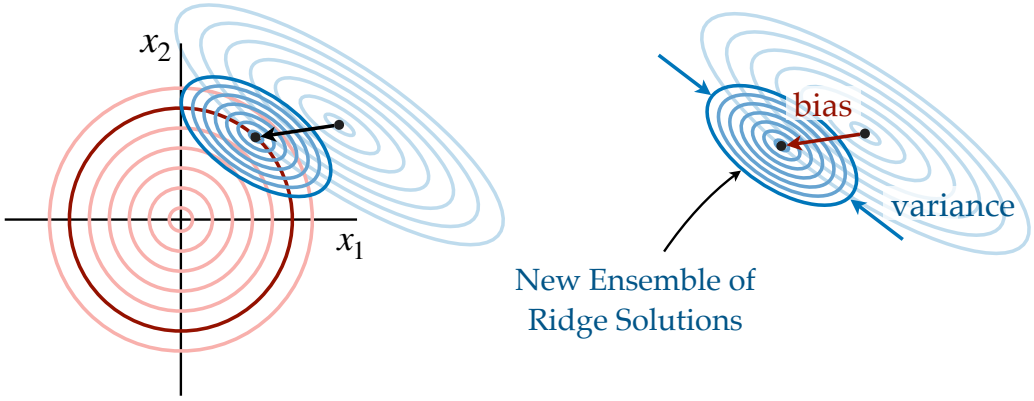


Figure 4.14: Schematic of bias-variance tradeoff in Tikhonov regularization. The regularized solution is represented by the dark blue ellipses, which have lower variance than the original solution, given by the light blue ellipses. However, Tikhonov regularization introduces a bias, shifting the mean of the new distribution.

We may view the additional term $\alpha\|\mathbf{x}\|_2^2$ as arising from a Lagrange multiplier enforcing the condition that $\|\mathbf{x}\|_2^2 = c$, where c is a constant (i.e., $\alpha(\|\mathbf{x}\|_2^2 - c)$. This will work for any constant c, as it essentially promotes the individual entries of $\mathbf{x}$ being similar in magnitude, as opposed to having very large and opposite entries canceling each other out, which is the case when $\mathbf{A}$ is ill conditioned.

This leads to a fundamental property in many regularized optimization problems: the bias-variance tradeoff. The original problem has high variance, and so we add regularization. However, this regularization term introduces a bias towards a solution where the entries of $\mathbf{x}$ are on the unit circle, as in Fig. 4.14.

There is also a Bayesian perspective to ridge regression, where the ridge regularizer arises from a Bayesian prior that the entries of $\mathbf{x}$ are Gaussian random variables with zero mean and the same variance. We will discuss this more in Chapter 7.

Ridge Regression via SVD. The formula in Eq. (4.39) can be computed without directly inverting the square matrix $(\mathbf{A}^T\mathbf{A} + \alpha\mathbf{I})$ by using the SVD. If we substitute $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, then we have

$$\begin{aligned}(\mathbf{A}^T\mathbf{A} + \alpha\mathbf{I})^{-1} &= (\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T + \alpha\mathbf{I})^{-1} \\ &= (\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T + \alpha\mathbf{V}\mathbf{I}\mathbf{V}^T)^{-1} \\ &= (\mathbf{V}(\mathbf{\Sigma}^2 + \alpha\mathbf{I})\mathbf{V}^T)^{-1}\end{aligned}$$

This may be inverted simply, as $\mathbf{V}$ is unitary and $\mathbf{\Sigma}^2 + \alpha\mathbf{I}$ is diagonal. So essentially, we add α times the identity matrix to the squared singular values in the matrix inversion step.

Sparse Regression

The least absolute shrinkage and selection operator (LASSO) is an ℓ_1 penalized regression technique that balances model complexity with descriptive capability [339]. Other related

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

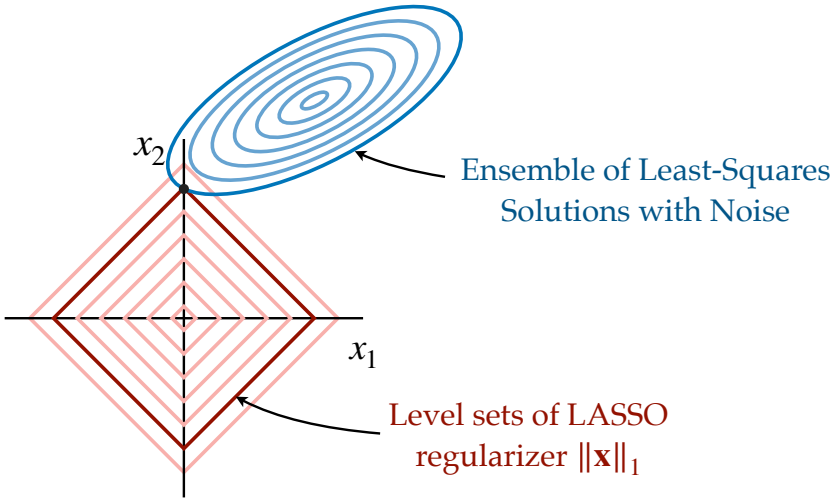


Figure 4.15: Schematic of LASSO regularization, to promote sparsity

sparsity-promoting techniques have subsequently been explored [14, 71, 299, 18, 378].

This principle of *parsimony* in a model is also a reflection of Occam's razor, stating that among all possible descriptions, the simplest correct model is probably the true one. Since its inception by Tibshirani in 1996 [339], the LASSO has become a cornerstone of statistical modeling, with many modern variants and related techniques [163, 371, 179]. The LASSO is closely related to the earlier nonnegative garrote of Breimen [50], and is also related to earlier work on soft-thresholding by Donoho and Johnstone [106, 107].

Least-squares regression will tend to result in a vector $\mathbf{x}$ that has nonzero coefficients for all entries, indicating that *all* columns of $\mathbf{A}$ must be used to predict $\mathbf{b}$. However, we often believe that the statistical model should be *simpler*, indicating that $\mathbf{x}$ may be sparse. The LASSO adds an ℓ_1 penalty term to *regularize* the least-squares regression problem; i.e., to prevent overfitting:

$$\mathbf{x} = \underset{\mathbf{x}'}{\operatorname{argmin}} \|\mathbf{A}\mathbf{x}' - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1. \tag{4.40}$$

Typically, the parameter λ is varied through a range of values and the fit is *validated* against a test set of holdout data. If there is not enough data to have a sufficiently large training and test set, it is common to repeatedly train and test the model on random selection of the data (often $80\,\%$ for training and $20\,\%$ for testing), resulting in a *cross-validated* performance. This cross-validation procedure enables the selection of a parsimonious model that has relatively few terms and avoids overfitting.

LASSO regression is frequently used to build statistical models for disease, such as cancer and heart failure, since there are many possible predictors, including demographics, lifestyle, biometrics and genetic information. Interpretability is important in statistical models, as these models are often communicated to a non-technical audience, including business leaders and policy makers. Generally, a regression model is more interpretable if it has fewer terms that bear on the outcome, motivating yet another perspective on sparsity.

Many statistical systems are overdetermined, as there are more observations than can-

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

didate predictors. However, the LASSO regression works well with overdetermined problems. Note that an early version of the geometric picture in Fig. 4.15 to explain the sparsity-promoting nature of the ℓ_1 norm was presented in Tibshirani's 1996 paper [339].

Elastic Net. The elastic net [387] combines the ridge and LASSO regularization terms:

$$\mathbf{x} = \underset{\mathbf{x}'}{\operatorname{argmin}} \|\mathbf{A}\mathbf{x}' - \mathbf{b}\|_2 + \lambda\|\mathbf{x}\|_1 + \alpha\|\mathbf{x}\|_2^2. \tag{4.41}$$

Elastic net regression is widely used in various machine learning applications.

4.5 Constrained Least-Squares

Up until now we have only considered unconstrained least-squares regression. However, many systems have constraints on the admissible values of $\mathbf{x}$. For example, we might know that some values of $\mathbf{x}$ must be non-negative, or we may have other prior knowledge, such as a certain symmetry in the coefficients of $\mathbf{x}$ or that the entries of $\mathbf{x}$ must be within some lower and upper bounds (i.e., box constraints). Other common constraints include some equality constraints on the coefficients $\mathbf{C}\mathbf{x} = \mathbf{d}$. We will discuss constraints in much more generality and detail in the next chapter. Here we briefly introduce the simplest equality constraints in least-squares.

We may derive the constrained least-squares solution for linear equality constraints:

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \tag{4.42a}$$

$$\text{s.t.} \quad \mathbf{C}\mathbf{x} = \mathbf{d}. \tag{4.42b}$$

Expanding the objective function, we may write this as a quadratic program with linear equality constraints as

$$\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} \tag{4.43a}$$

$$\text{s.t.} \quad \mathbf{C}\mathbf{x} = \mathbf{d} \tag{4.43b}$$

with the following matrices:

$$\mathbf{Q} = 2\mathbf{A}^T\mathbf{A} \qquad \mathbf{c}^T = -2\mathbf{b}^T\mathbf{A}. \tag{4.44}$$

We may then use Lagrange multipliers to augment the objective function with the constraint equation, resulting in the following augmented Lagrangian:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} + \boldsymbol{\lambda}^T(\mathbf{C}\mathbf{x} - \mathbf{d}). \tag{4.45}$$

The conditions for optimality are that $\nabla_{\mathbf{x}}\mathcal{L} = 0$ and $\nabla_{\boldsymbol{\lambda}}\mathcal{L} = 0$, which result in the stationarity condition and equality constraint condition, respectively:

$$\nabla_{\mathbf{x}}\mathcal{L} = \mathbf{Q}\mathbf{x} + \mathbf{c} + \mathbf{C}^T\boldsymbol{\lambda} = 0 \tag{4.46a}$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \mathbf{C}\mathbf{x} - \mathbf{d} = 0. \tag{4.46b}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Rewriting these equations, we may write them in matrix form as:

$$\left.\begin{array}{rl} \mathbf{Q}\mathbf{x}+\mathbf{C}^T\boldsymbol{\lambda} & -\mathbf{c} \\ \mathbf{C}\mathbf{x} & =\mathbf{d} \end{array}\right\} \implies \begin{bmatrix} \mathbf{Q} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{c} \\ \mathbf{d} \end{bmatrix}. \tag{4.47}$$

Writing this in terms of the original matrices and vectors from the constrained least-squares problem yields:

$$\begin{bmatrix} 2\mathbf{A}^T\mathbf{A} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 2\mathbf{A}^T\mathbf{b} \\ \mathbf{d} \end{bmatrix}. \tag{4.48}$$

It is possible to solve for $\mathbf{x}$ and $\boldsymbol{\lambda}$ by inverting this matrix system of equations, again using SVD or QR factorization. It is clear in the above that if we have no constraints, this is equivalent to the standard least-squares problem.

Minimum Norm as a Constrained Problem

If $\mathbf{A} = \mathbf{I}$ and $\mathbf{b} = \mathbf{0}$, then the constrained least squares reduces to the minimum-norm problem above.

Specifically, if we solve

$$\min_{\mathbf{x}} \quad \|\mathbf{x}\|_2^2 \tag{4.49a}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.49b}$$

then the constrained problem becomes

$$\begin{bmatrix} \mathbf{I} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix} \tag{4.50}$$

which has optimal solution $\mathbf{x}^* = \mathbf{A}^T\left(\mathbf{A}\mathbf{A}^T\right)^{-1}\mathbf{b}$. Working out the details of this derivation is left as an exercise for the reader.

4.6 Quadratic Programming

Linear least squares may be written as a *quadratic programming* problem, which generalizes linear programming to quadratic objective functions.

A quadratic program is written in standard form as:

$$\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x}, \tag{4.51a}$$

$$\text{Subject to:} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{4.51b}$$

where $\mathbf{Q}$ is a symmetric matrix.

If $\mathbf{Q}$ is positive semi-definite, so that $\mathbf{x}^T\mathbf{Q}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$, then this quadratic program is convex and any local minimum is a global minimum. If $\mathbf{Q}$ is indefinite, then the quadratic program is non-convex, and the solution may be much more involved.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

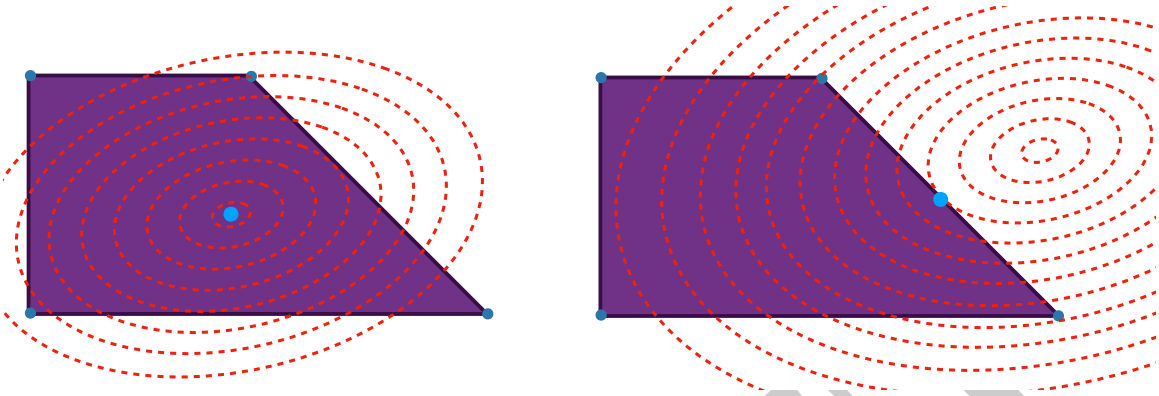


Figure 4.16: A quadratic program may have *inactive* constraints (left) so that the minimum lies in an interior point of the feasible region. Alternatively, a quadratic program may have *active* constraints (right) so that the minimum lies on the boundary of the feasible region. Here we visualize the convex feasible polytope in purple with level sets of the quadratic objective function in red.

In the case that $\mathbf{Q}$ is positive semidefinite, there are several cases based on whether or not the constraints are *active* or not. If the constraints are inactive, then the optimum value may occur at an interior point of the feasible set. If the constraints are active, then the optimum may live anywhere on the boundary of the feasible set. In linear programming, the solution occurred on vertices, but for quadratic programming with active constraints the optimum value can occur on an edge, a face, or a vertex. Thus, it may be harder to search for solutions because there is no obvious *simplex* search method. This is illustrated schematically in Fig. 4.16.

There are several approaches to solve quadratic programs:

- Active set methods: The idea behind active set methods is to guess which constraints are active and then solve a smaller problem with only those active constraints. Roughly speaking, if $\mathbf{a}_j^T\mathbf{x} \leq \mathbf{b}_j$ is active, then $\mathbf{a}_j^T\mathbf{x} = \mathbf{b}_j$, and we may solve the constrained least-squares problem with this active constraint using KKT methods.

- Interior point methods: IPMs are among the leading approaches to solve linear programming problems in high-dimensions. They scale well to large systems when we have active linear constraints like $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.

- Sequential quadratic programming (SQP): This is good for medium-scale system with possibly nonlinear constraints.

These will be discussed in much more detail in Chapter 5.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Least Squares as a Quadratic Program

It is simple to see that the linear least-squares problem is a quadratic program:

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2 = (\mathbf{A}\mathbf{x}-\mathbf{b})^T(\mathbf{A}\mathbf{x}-\mathbf{b}) \tag{4.52a}$$

$$= \mathbf{x}^T\mathbf{A}\mathbf{A}^T\mathbf{x} - 2\mathbf{b}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{b}. \tag{4.52b}$$

Thus, this loss function is a quadratic form with

$$\mathbf{Q} = \mathbf{A}^T\mathbf{A} \qquad \mathbf{c}^T = -2\mathbf{b}^T\mathbf{A}. \tag{4.53}$$

We see that $\mathbf{Q}$ is symmetric and positive semi-definite for any $\mathbf{A}$ matrix in the original least-squares problem, so the resulting quadratic program is convex.

Quadratic Programming Examples in Python

In Python, one can solve many QPs conveniently with `cvxpy`, similar to how linear programs are solved. Other libraries include `qpsolvers` and `OSQP`, and there are several other libraries with specialized solvers. The package `cvxpy` automatically analyzes whether your problem is a QP, even with additional constraints. If the problem is a QP, a specialized solver (e.g. `OSQP`) is chosen if available. For large QPs, direct factorization or warm starts with `OSQP` or `GUROBI` might be beneficial. The user can also fine-tune performance, for example by setting `prob.solve(solver=cp.OSQP, warm_start=True)`.

Any least-squares regression can be solved as a quadratic program: we minimize a quadratic objective (i.e., the squared residuals) subject to linear or affine constraints on the parameters. We will explore constrained least-squares more in Chapter 5 on constraints and duality.

Below are some code examples illustrating different constraint types.

Python Example: Least-Squares as a Quadratic Program. Here we use `cvxpy` to solve a least-squares regression problem, first *without* any constraints. Again, `cvxpy` recognizes squared norms automatically, so we can write $\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2$ as a quadratic objective.

```
import cvxpy as cp
import numpy as np

# Initialize A (m x n) and b (m x 1).
np.random.seed(0)
m, n = 20, 5
A = np.random.randn(m,n)
b = np.random.randn(m)

# Unconstrained Least Squares
x = cp.Variable(n)  # Regression coefficients
objective = cp.Minimize(cp.sum_squares(A@x - b))

prob = cp.Problem(objective)
result = prob.solve()
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
print("Optimal value:", result)
print("Optimal x:", x.value)
```

We now include the *linear constraint* $x_j \geq 0.5$ and $\sum_j x_j \leq 3$:

```
# Constrained Least Squares
x_constrained = cp.Variable(n)
constraints = [ x_constrained >= 0.5,
                cp.sum(x_constrained) <= 3.0 ]

obj_constrained = cp.Minimize(cp.sum_squares(A@x_constrained - b))
prob_constrained = cp.Problem(obj_constrained, constraints)
result_constrained = prob_constrained.solve()
```

Portfolio Optimization (Markowitz Model). A canonical QP arises in *portfolio optimization*: we choose $\mathbf{x}$, the fraction of total budget in each asset, to minimize risk subject to a target return and other constraints:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \tfrac{1}{2}\mathbf{x}^T \boldsymbol{\Sigma}\, \mathbf{x} \\ \text{s.t.} \quad & \mathbf{r}^T \mathbf{x} \;\geq R_0, \\ & \mathbf{1}^T \mathbf{x} = 1, \\ & \mathbf{x} \geq 0. \end{aligned}$$

Here, $\boldsymbol{\Sigma}$ is the covariance matrix of returns, indicating some spread of risk, $\mathbf{r}$ is the vector of expected returns per asset, R_0 is a minimal acceptable return, and $\mathbf{x} \geq 0$ ensures no short-selling.

The following code solves this portfolio optimization problem:

```
n = 5 # Example data has n=5 points
Sigma = np.array([[0.05,0.01,0.02,0.0,0.0],
                  [0.01,0.06,0.0,0.01,0.0],
                  [0.02,0.0,0.07,0.01,0.0],
                  [0.0,0.01,0.01,0.05,0.01],
                  [0.0,0.0,0.0,0.01,0.04]]) # 5x5 covariance
r = np.array([0.10,0.12,0.14,0.09,0.11])  # expected returns
R0 = 0.11                                 # target return

x = cp.Variable(n, nonneg=True)  # decision variable

# Objective: minimize x^T Sigma x
objective = cp.Minimize(0.5 * cp.quad_form(x, Sigma))

constraints = [
    cp.sum(x) == 1,      # sum of allocations = 1
    r @ x >= R0          # minimum return
]

prob = cp.Problem(objective, constraints)
result = prob.solve(solver=cp.SCS)
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
print("Optimal risk:", result)
print("Optimal x:", x.value)
print("Return of portfolio:", r @ x.value)
```

Here, each constraint $x_i \leq x_{i+1}$ is linear, so the feasible set is a polyhedron defined by the entries of the vector $\mathbf{x}$ monotonically increasing. The sum of squares objective is strictly convex. The solver can be a direct QP solver like `OSQP`, or the default `SCS`.

Regularized Least-Squares with Box Constraints. Here we will implement a Tikhonov-regularized least-squares with additional box constraints on the state: $l \leq x_i \leq u$. We will revisit box constraints in Section 5.2 using Lagrange multipliers and the KKT conditions.

```
np.random.seed(42)
m, n = 8, 4
A = np.random.randn(m,n)
b = np.random.randn(m)
alpha = 0.2

x = cp.Variable(n)  # Decision variable x in R^n

# Objective: (1/2)*||A x - b||^2 + alpha/2 * ||x||^2
objective = cp.Minimize( 0.5*cp.sum_squares(A@x - b) + 0.5*alpha*cp.
    sum_squares(x) )

# Example box constraints: -1 <= x_i <= 2
l, u = -1.0, 2.0
constraints = [ x >= l, x <= u ]

prob = cp.Problem(objective, constraints)
res = prob.solve(solver=cp.OSQP)
print("Optimal value:", res)
print("x =", x.value)
```

This problem is a quadratic objective with linear constraints, solvable by standard QP solvers like `OSQP`, `ECOS`, `SCS`, or others in `cvxpy`.

A Quadratic Cost with Additional Linear Coupling Constraints. Suppose we have two variables $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and we want to minimize

$$\tfrac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \tfrac{1}{2}\|\mathbf{C}\mathbf{y} - \mathbf{d}\|^2$$

subject to a coupling constraint $\mathbf{D}\mathbf{x} + \mathbf{E}\mathbf{y} \leq \mathbf{f}$. This is a typical multi-block scenario.

```
np.random.seed(0)
n = 4
A = np.random.randn(6,n)
b = np.random.randn(6)
C = np.random.randn(6,n)
d = np.random.randn(6)

D = np.random.randn(3,n)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
E = np.random.randn(3,n)
f = np.array([1,2,3])

x = cp.Variable(n)
y = cp.Variable(n)

obj = 0.5*cp.sum_squares(A@x - b) + 0.5*cp.sum_squares(C@y - d)

constraints = [D@x + E@y <= f]

prob = cp.Problem(cp.Minimize(obj), constraints)
res = prob.solve()
print("Objective val:", res)
print("x =", x.value)
print("y =", y.value)
```

Here, the objective is the sum of two separate least-squares terms, and we tie $\mathbf{x}, \mathbf{y}$ together with the linear constraint $\mathbf{D}\mathbf{x}+\mathbf{E}\mathbf{y} \leq \mathbf{f}$, turning it into a single QP. `cvxpy` handles the combination of quadratic terms and linear constraints automatically.

4.7 Nonlinear Least-Squares: Gauss-Newton Methods

When we move from simple least-squares regression in Eq. (4.2) to the nonlinear least squares in Eq. (4.4), the problem is no longer convex and there is no closed-form solution. Gauss-Newton methods are a powerful class of iterative techniques to solve these nonlinear least-squares problems. We will use the following notation for this nonlinear least-squares problem:

$$\mathcal{L} = \frac{1}{2}\sum_{j=1}^{m} \|\mathbf{y}_j - \mathcal{N}(\mathbf{x}_j;\boldsymbol{\theta})\|_2^2 = \frac{1}{2}\sum_{j=1}^{m} f_j(\boldsymbol{\theta})^2 = \frac{1}{2}\|\mathbf{f}(\boldsymbol{\theta})\|_2^2. \tag{4.54}$$

The sum from $j = 1$ to m is typically a sum over m data points $\{(\mathbf{x}_j, \mathbf{y}_j\}_{j=1}^m$, so that we are minimizing the sum-squared error of a function fit over all data. The function $\mathbf{f}(\boldsymbol{\theta})$ is a vector of normed *residuals*, so that each $f_j(\boldsymbol{\theta}) = \|\mathbf{r}_j\|_2$ where the residual error is $\mathbf{r}_j = \mathbf{y}_j - \mathcal{N}(\mathbf{x}_j)$ evaluated at the j-th data point. The variables $\boldsymbol{\theta}$ are the parameters of the nonlinear regression model and are the variables we will optimize over. This nonlinear least-squares formulation is very general, and encompasses much of neural network training among other nonlinear regression problems. In the case of a neural network, discussed more in Chapter 10, $\mathcal{N}$ represents the neural network and $\boldsymbol{\theta}$ is the vector of weights and biases that we train to minimize the loss function.

Solving for $\min_{\boldsymbol{\theta}} \|\mathbf{f}(\boldsymbol{\theta})\|^2$ amounts to finding points where $\nabla_{\boldsymbol{\theta}}\mathcal{L} = \mathbf{0}$ with positive curvature. This is a nonlinear system of equations, leading to a non-convex optimization problem, so we solve with iterative methods, such as gradient descent or Newton's method.

Similar to other Newton methods for large-scale problems, here we will approximate the Hessian. In this way, Gauss-Newton may be viewed as a special case of a quasi-Newton method from Chapter 2, where the quadratic-residual structure is used to approximate the Hessian. However, they are often considered distinct classes of methods, as they have diverged in implementations and use cases.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Here we compute the gradient and Hessian of the loss function $\mathcal{L}$, with respect to the optimization variable $\boldsymbol{\theta}$. The gradient is

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{j=1}^{m} f_j(\boldsymbol{\theta}) \nabla f_j(\boldsymbol{\theta}) = \mathbf{J}^T \mathbf{f} \tag{4.55}$$

where $\mathbf{J}$ is the Jacobian matrix of derivatives

$$\mathbf{J} = \mathbf{D}\mathbf{f} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \cdots \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} — & \nabla f_1^T & — \\ — & \nabla f_2^T & — \\ & \vdots & \end{bmatrix} \tag{4.56}$$

and its transpose is given by

$$\mathbf{J}^T = \begin{bmatrix} | & | & \\ \nabla f_1 & \nabla f_2 & \cdots \\ | & | & \end{bmatrix}. \tag{4.57}$$

Thus $\mathbf{J}^T \mathbf{f}$ yields the sum in Eq. (4.55).

We now write an expression for the Hessian $\mathbf{H} = \nabla^2_{\boldsymbol{\theta}} \mathcal{L}$, which we will derive in the next subsection below:

$$\mathbf{H} = \nabla^2_{\boldsymbol{\theta}} \mathcal{L} = \mathbf{J}^T \mathbf{J} + \sum_{j=1}^{m} f_j(\boldsymbol{\theta}) \nabla^2 f_j(\boldsymbol{\theta}) = \mathbf{J}^T \mathbf{J} + \mathbf{Q}. \tag{4.58}$$

The first term only includes first derivatives, and the second term contains second derivatives.

For many problems, it is too expensive to compute $\mathbf{Q}$. Fortunately, because $f_j(\boldsymbol{\theta})$ are the residual errors, it is common to assume that these are small, and approximate the Hessian just using the first order terms:

$$\mathbf{H} \approx \mathbf{J}^T \mathbf{J}. \tag{4.59}$$

This is the primary assumption used in Gauss-Newton methods.

Newton's method then becomes:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}. \tag{4.60}$$

However, instead of storing and computing $\mathbf{H}^{-1}$, which would involve $\mathcal{O}(n^3)$ operations, we instead solve the linear system of equations

$$\mathbf{H}\mathbf{s}_k = -\nabla_{\boldsymbol{\theta}} \mathcal{L} \tag{4.61}$$

and directly use $\mathbf{s}_k$, which should equal $\mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}$, for the update in Newton's method:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{s}_k. \tag{4.62}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Using the Gauss-Newton approximation to the Hessian in Eq. (4.59), this becomes

$$\mathbf{J}_k^T \mathbf{J}_k \mathbf{s}_k \approx -\mathbf{J}_k^T \mathbf{f}(\boldsymbol{\theta}_k) \tag{4.63}$$

where $\mathbf{J}_k = \mathbf{J}(\boldsymbol{\theta}_k) = \mathbf{D}\mathbf{f}(\boldsymbol{\theta}_k)$.

Therefore, the full Gauss-Newton iteration is

$$\text{Solve} \quad \mathbf{J}_k^T \mathbf{J}_k \mathbf{s}_k = -\mathbf{J}_k^T \mathbf{f}(\boldsymbol{\theta}_k) \tag{4.64a}$$

$$\text{Iterate} \quad \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{s}_k. \tag{4.64b}$$

Gauss-Newton methods reduce the nonlinear least-squares problem to a sequence of linearized regular least-squares problems[4]. This can be seen several ways. In each iteration step, solving

$$\mathbf{J}^T \mathbf{J}\mathbf{s} = -\mathbf{J}^T \mathbf{f} \tag{4.65}$$

is equivalent to a least-squares problem

$$\min_{\mathbf{s}} \frac{1}{2} \|\mathbf{J}\mathbf{s} + \mathbf{f}\|^2. \tag{4.66}$$

Specifically, we may expand this function as

$$\frac{1}{2}\left[(\mathbf{J}\mathbf{s} + \mathbf{f})^T (\mathbf{J}\mathbf{s} + \mathbf{f}) \right] \tag{4.67a}$$

$$\frac{1}{2}\left[\mathbf{s}^T \mathbf{J}^T \mathbf{J}\mathbf{s} + 2\mathbf{s}^T \mathbf{J}^T \mathbf{f} + \mathbf{f}^T \mathbf{f} \right]. \tag{4.67b}$$

Minimizing this with respect to $\mathbf{s}$ amounts to computing the gradient with respect to $\mathbf{s}$ and setting equal to $\mathbf{0}$, which yields:

$$\mathbf{J}^T \mathbf{J}\mathbf{s} = \mathbf{J}^T \mathbf{f}. \tag{4.68}$$

Deriving the Hessian

To derive the Hessian expression in Eq. (4.58), we first introduce a helpful bit of notation. If $\nabla_{\boldsymbol{\theta}} \mathcal{L} = \mathbf{J}^T \mathbf{f}(\boldsymbol{\theta})$, then we denote the j-th component as:

$$\nabla_{\theta_j} \mathcal{L} = \sum_{i=1}^{m} f_i(\boldsymbol{\theta}) \frac{\partial f_i}{\partial \theta_j}. \tag{4.69}$$

Similarly, we denote the (j,k)-th component of the Hessian as $H_{jk} = \nabla^2_{\theta_j \theta_k} \mathcal{L}$, which is given by:

$$H_{jk} = \nabla^2_{\theta_j \theta_k} \mathcal{L} = \sum_i \left(\frac{\partial f_i}{\partial \theta_k} \frac{\partial f_i}{\partial \theta_j} + f_i \frac{\partial^2 f_i}{\partial \theta_j \partial \theta_k} \right) = \mathbf{J}^T \mathbf{J} + \mathbf{Q}. \tag{4.70}$$

[4]Solving these linearized regular least-squares problems may be challenging for large-scale problems, which motivates the conjugate gradient method in the next section.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

To see that $\mathbf{J}^T\mathbf{J} = \sum_i \frac{\partial f_i}{\partial \theta_k}\frac{\partial f_i}{\partial \theta_j}$, we expand as

$$\mathbf{J}^T\mathbf{J} = \begin{bmatrix} — & \nabla f_1^T & — \\ — & \nabla f_2^T & — \\ & \vdots & \end{bmatrix} \begin{bmatrix} | & | & \\ \nabla f_1 & \nabla f_2 & \cdots \\ | & | & \end{bmatrix} \tag{4.71}$$

so the (j,k)-th element of $\mathbf{J}^T\mathbf{J}$ is given by the j-th row of $\mathbf{J}^T$ times the k-th column of $\mathbf{J}$:

$$\begin{bmatrix} \partial f_1/\partial \theta_j & \partial f_2/\partial \theta_j & \cdots \end{bmatrix} \begin{bmatrix} \partial f_1/\partial \theta_k \\ \partial f_2/\partial \theta_k \\ \vdots \end{bmatrix} = \sum_i \frac{\partial f_i}{\partial \theta_k}\frac{\partial f_i}{\partial \theta_j} = \nabla^2_{\theta_j\theta_k}\mathcal{L}. \tag{4.72}$$

Issues with Gauss-Newton

There are a number of well known issues with Gauss-Newton methods, and there is a vast literature on how to address these challenges. The Gauss-Newton approximation to the Hessian matrix as the Jacobian of the residuals is efficient, but it assumes the residuals are small and ignores second-order terms. This can lead to divergence of the iteration when residuals are large, so that Gauss-Newton performs poorly in regions far from the solution. Moreover, there are several failure modes when $\mathbf{J}^T\mathbf{J}$ is singular or nearly singular.

If $\mathbf{J}^T\mathbf{J}$ is nearly singular, then the update equation $\mathbf{J}^T\mathbf{J}\Delta\mathbf{x} = -\nabla\mathcal{L}(\mathbf{x})$, may result in a large update step $\Delta\mathbf{x}$, which can cause the iteration to overshoot the solution, leading to divergence.

When $\mathbf{J}^T\mathbf{J}$ is nearly singular, then the linear system in Eq. (**??**) is ill conditioned, and solving it becomes numerically sensitive to errors. Small errors in the residuals or the Jacobian can cause large changes in the solution $\Delta\mathbf{x}$. This may lead to erratic or unstable behavior of the iteration.

Finally, if $\mathbf{J}^T\mathbf{J}$ is nearly singular, the update $\Delta\mathbf{x}$ might point in a direction that does not improve the objective function. This can cause the optimization to stall.

The Levenberg-Marquardt algorithm improves the robustness of Gauss-Newton methods for ill-conditioned nonlinear least-squares problems, and is discussed in the next subsection.

Levenberg Marquardt Algorithm

Gauss-Newton methods are efficient for well-conditioned problems, but have known failure modes when the Hessian approximation is ill-conditioned. The Levenberg-Marquardt (LM) algorithm [221, 240, 138] combines Gauss-Newton with gradient descent to improve the robustness for poorly conditioned or ill-posed problems. LM also helps when the initial guess is far from the solution and residuals are large, making the Hessian approximation in Gauss-Newton inaccurate. This hybrid approach is highly effective and is used in a wide range of application domains that involve solving nonlinear systems, including computer vision, curve fitting, machine learning, parameter estimation in control, and robotics.

The fundamental idea behind Levenberg-Marquardt is to regularize the Hessian approximation with a weighted identity matrix, so that

$$\mathbf{H} \approx \left(\mathbf{J}^T\mathbf{J} + \alpha\mathbf{I}\right). \tag{4.73}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Thus the update equation becomes

$$\left(\mathbf{J}^T\mathbf{J} + \alpha\mathbf{I}\right)\Delta\boldsymbol{\theta} = -\mathbf{J}^T\mathbf{f}. \tag{4.74}$$

The damping parameter α blends between Gauss-Newton ($\alpha = 0$) and gradient descent ($\alpha \to \infty$). The transition to gradient descent as $\alpha \to \infty$ is simple to see, as the update becomes

$$\Delta\boldsymbol{\theta} = -\mathbf{J}^T\mathbf{f} \tag{4.75}$$

when the $\alpha\mathbf{I}$ term dominates the Hessian approximation.

The LM modification to the Hessian approximation in Eqs. (4.73) and (4.74) may be viewed as adding Tikhonov regularization (see Sec. 4.4), which is known to improve the robustness of ill-conditioned least-squares regression problems.

Typically, the parameter α is adapted dynamically, based on the residuals. When the initial guess is far from the solution, a large α is used, and the gradient-descent behavior stabilizes the updates. As the iteration approaches the optimal solution and residuals decrease, LM automatically shifts to the faster Gauss-Newton method. Thus, LM achieves fast quadratic convergence near the solution. In a sense, LM may be viewed as transitioning between exploration (i.e., gradient descent) and exploitation (i.e., Gauss-Newton) in different phases of the iteration.

There are several approaches for modifying α dynamically. The simplest approach is to simply increase α if the loss function does not decrease sufficiently, and to decrease α if the change in the loss function is sufficient. This is achieved by multiplying or dividing α by a constant factor $\nu > 1$. Since the loss function naturally decreases toward the solution, the changes to the loss function become smaller with successful iterations, and so we use a relative change in the loss function with respect to the quadratic approximation to determine if the change in loss is sufficient. This is quantified by the relative ratio ρ:

$$\rho = \frac{\Delta\mathcal{L}_{\text{actual}}}{\Delta\mathcal{L}_{\text{predicted}}}, \tag{4.76}$$

where

$$\Delta\mathcal{L}_{\text{actual}} = \mathcal{L}(\boldsymbol{\theta}_k) - \mathcal{L}(\boldsymbol{\theta}_{k+1}) \tag{4.77}$$

is the actual reduction, and

$$\Delta\mathcal{L}_{\text{predicted}} = -\mathbf{J}^T\mathbf{f}\Delta\boldsymbol{\theta} - \frac{1}{2}\Delta\boldsymbol{\theta}^T(\mathbf{J}^T\mathbf{J} + \alpha\mathbf{I})\Delta\boldsymbol{\theta} \tag{4.78}$$

is the predicted reduction from the quadratic model.

Even if $\Delta\mathcal{L}_{\text{actual}}$ becomes small near the minimum, ρ ensures that the reduction is evaluated relative to the quadratic Hessian prediction. This approach avoids issues with absolute magnitudes of reductions. The parameter α is usually initialized with a small positive value (e.g., $1.e-3$).

This simple approach of increasing or decreasing α based on the relative ratio ρ has a *trust region* interpretation [282, 254, 97, 268, 82], where α controls the size of the region in which the quadratic model is trusted. For example, we may use a condition such as

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- If $\rho > 0.75$, reduce α (the model is trustworthy).
- If $\rho < 0.25$, increase α (the model is unreliable).

Marquardt, of LM, explicitly adjusts α based on the norm of the step $\Delta\boldsymbol{\theta}$, so that

$$\alpha = \max(\alpha\nu, \epsilon\|\Delta\boldsymbol{\theta}\|), \tag{4.79}$$

where $\epsilon > 0$ ensures sufficient regularization.

It is also possible to introduce more sophisticated regularizations, beyond Tikhonov regularization. For example, instead of adding $\alpha\mathbf{I}$ to the Hessian approximation, it is possible to adaptively scale each diagonal entry individually:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} + \alpha\,\mathrm{diag}(\mathbf{J}^T\mathbf{J}). \tag{4.80}$$

This approach dynamically scales damping based on the local curvature.

4.8 The Conjugate Gradient Method

The conjugate gradient (CG) method is an iterative algorithm for solving large, sparse systems of linear equations [342, 9],

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{4.81}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is symmetric positive definite, $\mathbf{x} \in \mathbb{R}^n$ is the unknown vector, and $\mathbf{b} \in \mathbb{R}^n$ is the right-hand side. CG is particularly efficient for large, sparse systems because it avoids explicitly storing or factorizing $\mathbf{A}$. Instead, it builds an approximate solution iteratively using Krylov subspaces and ensures convergence in at most n iterations. CG is often used in scientific computing, for example to solve the Poisson equation in computational fluid dynamics. The conjugate gradient algorithm is also frequently used to accelerate Gauss-Newton methods for very large, sparse systems. It is a cornerstone of modern optimization.

In the context of the Gauss-Newton algorithm above, the conjugate gradient method is used to solve the linear system[5]:

$$\mathbf{J}^T\mathbf{J}\Delta\mathbf{x} = -\mathbf{J}^T\mathbf{f}, \tag{4.82}$$

Instead of directly inverting the approximate Hessian $\mathbf{J}^T\mathbf{J} \approx \mathbf{H}$, the conjugate gradient method solves the system iteratively, avoiding the explicit computation of the Hessian matrix. Instead, it requires only matrix-vector products, which dramatically improves the memory efficiency of the algorithm for large-scale problems, where it might be prohibitive to construct and store the Hessian approximation. The improvement in memory and speed is even more pronounced when the Hessian is sparse.

In addition to improving the scalability of Gauss-Newton methods to large-scale problems, the conjugate gradient method also improves numerical stability by avoiding explicit

[5]Note that here we have returned to using $\mathbf{x}$ for the optimization variable, while in the discussion on Gauss-Newton we used $\boldsymbol{\theta}$.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

inversion of the Hessian, which may be ill-conditioned. Similarly, it is possible to add regularization and preconditioners flexibly in the conjugate gradient formulation. However, convergence still depends on the condition number of $\mathbf{J}^T\mathbf{J}$ and preconditioning is often necessary.

The idea behind the conjugate gradient method is to expand the solution $\mathbf{x}$ in Eq. (4.81) in a basis of conjugate vectors $\mathbf{p}_k$, that are orthogonal with respect to the inner product defined by $\mathbf{A}$:

$$\mathbf{p}_i^T\mathbf{A}\mathbf{p}_j = 0 \quad \forall i \neq j. \tag{4.83}$$

If $\mathbf{A}$ is a symmetric, positive definite matrix, the set of $\{\mathbf{p}_j\}_{j=1}^n$ form a basis for $\mathbb{R}^n$, and any vector $\mathbf{x}$ may be expanded in this basis. If these vectors are chosen judiciously, then we may be able to approximate $\mathbf{x}$ quite accurately without all n of these vectors.

Given a set of conjugate vectors $\{\mathbf{p}_j\}_{j=1}^n$, we approximate the solution $\mathbf{x}$ as

$$\mathbf{x} = \sum_{j=1}^{n} \alpha_j \mathbf{p}_j. \tag{4.84}$$

To approximate the k-th coefficient α_k, we use $\mathbf{A}\mathbf{x} = \mathbf{b}$

$$\mathbf{A}\mathbf{x} = \sum_{j=1}^{n} \alpha_j \mathbf{A}\mathbf{p}_j = \mathbf{b} \tag{4.85}$$

and take the inner product of both sides with $\mathbf{p}_k$:

$$\mathbf{p}_k^T\mathbf{b} = \sum_{j=1}^{n} \alpha_j \mathbf{p}_k^T\mathbf{A}\mathbf{p}_j. \tag{4.86}$$

Since $\mathbf{p}$ are conjugate vectors, the only nonzero term in the sum is $\alpha_k\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k$:

$$\mathbf{p}_k^T\mathbf{b} = \alpha_k\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k \tag{4.87}$$

so that

$$\alpha_k = \frac{\mathbf{p}_k^T\mathbf{b}}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}. \tag{4.88}$$

The conjugate gradient method iteratively builds up this basis $\mathbf{p}_k$, and the approximation to $\mathbf{x}$, i.e., the coefficients α_k in the basis $\mathbf{p}_k$.

Gradient Descent Formulation. The CG method can be viewed as an enhancement of the classical gradient descent algorithm for minimizing the quadratic objective function:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}. \tag{4.89}$$

The gradient of $f(\mathbf{x})$ is:

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}. \tag{4.90}$$

Since we are trying to set $\nabla f(\mathbf{x}) = \mathbf{0}$, we use the negative gradient as our first vector $\mathbf{p}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}$. Thus, all future vectors will be conjugate to the gradient direction, hence the name.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Iterative Construction of Conjugate Vectors. The following description is the simplest way to understand the construction of the conjugate vectors $\mathbf{p}$, but it turns out that there is a more efficient algorithm. We will present the intuitive construction first here, followed by the streamlined algorithm.

The residual at each step is $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$, so the initial conjugate vector $\mathbf{p}_1$ is also the first residual. Classic gradient descent would take the next step in this residual direction, but the conjugate gradient method requires that the next step is in a direction that is conjugate to the previous $\mathbf{p}$ vectors; again, conjugate just means orthogonal with respect to the inner product $\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{A}} = \mathbf{u}^T \mathbf{A} \mathbf{v}$. Thus, we take the next step in a direction that is obtained by taking the residual $\mathbf{r}_k$ and removing all components in each of the previous $\mathbf{p}_j$ directions. This amounts to a Gram-Schmidt orthonormalization in the weighted inner product space determined by $\mathbf{A}$:

$$\mathbf{p}_k = \mathbf{r}_k - \sum_{j=1}^{k-1} \frac{\mathbf{p}_j^T \mathbf{A} \mathbf{r}_k}{\mathbf{p}_j^T \mathbf{A} \mathbf{p}_j} \mathbf{p}_j. \tag{4.91}$$

Each term in the sum can be understood in the following way: the numerator is the projection of $\mathbf{r}_k$ in the $\mathbf{p}_j$ direction, the denominator is a normalization factor, and the $\mathbf{p}_j$ to the right of the fraction indicates that this vector is being subtracted from $\mathbf{r}_k$ in exactly the amount that is contained in $\mathbf{r}_k$. Thus, this sum removes all components from all previous $\mathbf{p}_j$ conjugate vectors, so that $\mathbf{p}_k$ is guaranteed to be conjugate to all prior vectors.

Once the new vector $\mathbf{p}_k$ is constructed, we may update $\mathbf{x}$ as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k = \mathbf{x}_k + \frac{\mathbf{p}_k^T \mathbf{b}}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} \mathbf{p}_k \tag{4.92}$$

where α_k was previously derived in Eq. (4.88).

In actuality, there are ways of compressing and streamlining several of these steps, which we will explore below.

Krylov Subspaces and Conjugate Directions. At each iteration, CG minimizes $f(\mathbf{x})$ within the k-dimensional Krylov subspace:

$$\mathcal{K}_k = \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\}, \tag{4.93}$$

where $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ is the initial residual. Unlike gradient descent, CG constructs a sequence of search directions $\mathbf{p}_k$ that are $\mathbf{A}$-*conjugate*, as defined before. Conjugate directions ensure that the minimization along each direction does not interfere with progress along previously chosen directions, leading to faster convergence.

The CG method iteratively updates $\mathbf{x}_k$, the residual $\mathbf{r}_k$, and the search direction $\mathbf{p}_k$ as follows.

We begin with an initial guess $\mathbf{x}_0$:

$$\mathbf{x}_0, \quad \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0, \quad \mathbf{p}_0 = \mathbf{r}_0. \tag{4.94}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Then, for each iteration k we compute:

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}, \tag{4.95a}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{4.95b}$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k \tag{4.95c}$$

and update the next conjugate direction:

$$\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}, \tag{4.96a}$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k. \tag{4.96b}$$

Notice here that we modified our expression for α_k to only include the current residual $\mathbf{r}_k$ and the current conjugate vector $\mathbf{p}_k$. It will also be helpful for the reader to verify that the new expression for the residual is equivalent to $\mathbf{r}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b}$.

The iterations terminate when $\|\mathbf{r}_k\|$ is below a predefined tolerance or after n iterations.

The CG method converges in at most n iterations because the Krylov subspace expands to include the entire space $\mathbb{R}^n$. However, practical performance depends on the condition number $\kappa(\mathbf{A}) = \lambda_{\text{max}}/\lambda_{\text{min}}$, where λ_{max} and λ_{min} are the largest and smallest eigenvalues of $\mathbf{A}$. The convergence rate is approximately:

$$\|\mathbf{x}_k - \mathbf{x}^*\|_{\mathbf{A}} \leq C \left(\frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1} \right)^k, \tag{4.97}$$

where $\mathbf{x}^*$ is the true solution and C is a constant depending on the initial error. Thus, the performance of CG deteriorates for poorly conditioned systems (large $\kappa(\mathbf{A})$), which slows convergence. Preconditioning is commonly used to improve the condition number of $\mathbf{A}$, transforming the system into:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}, \tag{4.98}$$

where $\mathbf{M}$ is a preconditioner chosen to approximate $\mathbf{A}$ while being easy to invert.

CG is also memory efficient, requiring storage only for a few vectors ($\mathbf{x}_k, \mathbf{r}_k, \mathbf{p}_k$), making it ideal for large-scale problems. It avoids explicit matrix factorization and is particularly suited for sparse systems.

Generalized Minimal Residual Method (GMRES)

There is a closely related method, GMRES, that also solves linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ iteratively. Both are Krylov subspace methods, but while CG minimizes the energy norm $\|\mathbf{x} - \mathbf{x}^*\|_{\mathbf{A}}$, GMRES directly minimizes the two-norm of the residual, $\|\mathbf{r}\|_2$ by building an orthonormal basis for the Krylov space. CG applies to symmetric positive definite (SPD) matrices, while GMRES applies to any square matrix. However, GMRES also has higher cost per iteration, as it requires orthogonalization at each step; similarly, the storage requirements grow with each iteration. That said, GMRES will typically be more robust and is a good alternative to direct methods like LU decomposition when they are too computationally expensive.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) **Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control**, by S. L. Brunton and J. N. Kutz, 2022 [56].

(2) **An introduction to statistical learning**, by G. James, D. Witten, T. Hastie, and R. Tibshirani, 2013 [179].

(3) **Primal-dual interior-point methods**, by S. J. Wright, 1997 [375].

Papers and reviews

(1) **Regression shrinkage and selection via the lasso**, by R. Tibshirani, *Journal of the Royal Statistical Society B*, 1996 [339].

(2) **Robust face recognition via sparse representation**, by J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009 [374].

(3) **On the stability of inverse problems**, by A. N. Tikhonov, *Dokl. akad. nauk sssr*, 1943 [340].

(4) **On the stability of inverse problems**, by G. H. Golub, P. C. Hansen and D. P. O'Leary, *SIAM Journal on Matrix Analysis and Applications*, 1999 [151].

(5) **Regularization and variable selection via the elastic net**, by H. Zou and T. Hastie, *Journal of the Royal Statistical Society: Series B* , 2005 [387].

(6) **Interior-point methods**, by F. A. Potra and S. J. Wright, *Journal of computational and applied mathematics*, 2005 [280].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

Exercise 4–1: Recall that the minimum norm solution may be formulated in the KKT framework with the following system of equations:

$$\begin{bmatrix} \mathbf{I} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix} \tag{4.99}$$

Show that the solution of this system is $\mathbf{x}^* = \mathbf{A}^T \left(\mathbf{A}\mathbf{A}^T\right)^{-1} \mathbf{b}$.

Exercise 4–2: Show that $\nabla(\mathbf{A}\mathbf{x}) = \mathbf{A}$ and $\nabla(\mathbf{x}^T\mathbf{A}) = \mathbf{A}$ even for non-square matrices $\mathbf{A}$.

Exercise 4–3: For each of the following matrices, indicate if it is positive definite, semi-definite, indefinite, etc.

1. $\mathbf{Q} = \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}$
2. $\mathbf{Q} = \begin{bmatrix} 1 & 2 \\ -2 & -1 \end{bmatrix}$
3. $\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 2 & 2 \end{bmatrix}$
4. $\mathbf{Q} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$
5. $\mathbf{Q} = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 1 \end{bmatrix}$

Exercise 4–4: Consider the following matrix

$$\mathbf{Q} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

What are conditions on a, b, c, d for this matrix to be positive definite? Positive semi-definite?

Exercise 4–5: Show that the trace of a 2×2 matrix is the sum of its eigenvalues. Show that the determinant of this matrix is the product of its eigenvalues.

Exercise 4–6: Show that the trace of a general $n \times n$ matrix is the sum of its eigenvalues.

Exercise 4–7: Show that the determinant of a general $n \times n$ matrix is the product of its eigenvalues.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 4–8: Show that a matrix $\mathbf{Q}$ is positive semidefinite if $\mathbf{Q} = \mathbf{A}^T\mathbf{A}$. Show that this is also a necessary condition for $\mathbf{Q}$ to be positive semidefinite.

Exercise 4–9: Give an example of a real-valued 2×2 matrix that is positive definite and not symmetric. How does this relate to the requirement that $\mathbf{Q} = \mathbf{A}^T\mathbf{A}$ above?

Exercise 4–10: Show that if $\mathbf{Q} = \mathbf{A}^T\mathbf{A}$ for a non-singular $\mathbf{A}$, then $\mathbf{Q}$ is positive definite.

Exercise 4–11: Consider a symmetric matrix $\mathbf{A}$ with maximum eigenvalue λ. Show that $\mathbf{Q} = \alpha\mathbf{I} - \mathbf{A}$ is positive semidefinite for $\alpha \geq \lambda$.

Exercise 4–12: A matrix is called diagonally dominant if the magnitude of each diagonal entry is larger than the sum of magnitudes of all other entries in the same row:

$$|a_{jj}| > \sum_{i \neq j} |a_{ji}|.$$

Show that a diagonally dominant matrix with positive diagonal entries is positive semi-definite.

Exercise 4–13: The matrices $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$ are *correlation* matrices for the columns and rows of $\mathbf{X}$, respectively. Specifically, the ij-th entry of $\mathbf{X}^T\mathbf{X}$ is the inner product of the i-th column with the j-th column of $\mathbf{X}$. Expand $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$ in terms of the SVD, $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, and simplify.

Exercise 4–14: Show that $\mathbf{U}$ are eigenvectors of the symmetric matrix $\mathbf{X}\mathbf{X}^T$. Similarly show that $\mathbf{V}$ are eigenvectors of $\mathbf{X}^T\mathbf{X}$. (Hint: remember that $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices, so $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$.)

Exercise 4–15: Show that if $\mathbf{X}$ is self-adjoint (i.e., $\mathbf{X} = \mathbf{X}^T$) then the singular values of $\mathbf{X}$ are equal to the absolute value of the eigenvalues of $\mathbf{X}$.

Exercise 4–16: Load a high resolution image, like the image of Mordecai the snow dog from Fig. 4.17. This image has 2000×1500 pixels.

1. Compute the SVD of this image and plot the singular values on a logarithmic plot.

2. For various values of r from 1 to 100, compute the rank-r SVD approximation and compute the the error between the original matrix image $\mathbf{X}$ and the reconstruction $\hat{\mathbf{X}}$; use the Frobenius norm. Plot this error versus the rank r.

3. Now compute the error of the rank-r approximation using Eq. (4.16) and compare with the actual error.

4. What rank r is needed to reconstruct the image to 90% accuracy? How about for 99%?

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.



Figure 4.17: High resolution image to demonstrate SVD compression.

Exercise 4–17: This exercise will test the concept of condition number. We will test the accuracy of solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ when noise is added to $\mathbf{b}$ for matrices $\mathbf{A}$ with different condition numbers.

(a) To build the two matrices, generate a random $\mathbf{U} \in \mathbb{R}^{100\times100}$ and $\mathbf{V} \in \mathbb{R}^{100\times100}$ and then create two $\mathbf{\Sigma}$ matrices: the first $\mathbf{\Sigma}$ will have singular values spaced logarithmically from 100 to 1, and the second $\mathbf{\Sigma}$ will have singular values space logarithmically from 100 to 1e-6. Use these matrices to create two $\mathbf{A}$ matrices, one with a condition number of 100 and the other with a condition number of 100 million. Now create a random $\mathbf{b}$ vector, solve for $\mathbf{x}$ using the two A matrices and compare the results. Add a small $\boldsymbol{\epsilon}$ to $\mathbf{b}$, with norm 1e-6 smaller than the norm of $\mathbf{b}$. Now solve for $\mathbf{x}$ using this new $\mathbf{b} + \boldsymbol{\epsilon}$ and compare the results.

(b) Repeat the experiment above with many different noise vectors $\boldsymbol{\epsilon}$ and compute the distribution of the error; plot this error as a histogram and explain the shape.

(c) Repeat the above experiment comparing two $\mathbf{A}$ matrices with different singular value distributions: the first $\mathbf{\Sigma}$ will have values spaced linearly from 100 to 1 and the second $\mathbf{\Sigma}$ will have value spaced logarithmically from 100 to 1. Does anything change? Please explain why yes or no.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(d) Repeat the above experiment, but now with a $\mathbf{A}$ matrix that has size 100×10. Explain any changes.

Exercise 4–18: There are very specific conditions that must be met for the ℓ_1-minimization in (4.31) to converge with high probability to the sparsest solution in (4.30) [66, 68, 20]:

(a) The measurement matrix $\mathbf{C}$ must be *incoherent* with respect to the sparsifying basis $\mathbf{\Psi}$, meaning that the rows of $\mathbf{C}$ are not correlated with the columns of $\mathbf{\Psi}$,

(b) The number of measurements p must be sufficiently large, on the order of

$$p \approx \mathcal{O}(K \log(n/K)) \approx k_1 K \log(n/K). \tag{4.100}$$

The constant multiplier k_1 depends on how incoherent $\mathbf{C}$ and $\mathbf{\Psi}$ are.

Roughly speaking, these two conditions guarantee that the matrix $\mathbf{C\Psi}$ acts as a unitary transformation on K sparse vectors $\mathbf{s}$, preserving relative distances between vectors and enabling almost certain signal reconstruction with ℓ_1 convex minimization. This is often referred to as the restricted isometry property.

(a) Construct a measurement matrix $\mathbf{C}$ with p rows and n columns. To start, use $p = 100$ and $n = 1000$. The entries of $\mathbf{C}$ should be independent, identically distributed Gaussian elements with mean 0 and standard deviation 1.

Similarly construct the basis $\mathbf{\Psi}$ given by the inverse discrete cosine transform.

(b) Compute $\mathbf{C\Psi} = \mathbf{\Theta}$. What is its condition number?

(c) Compute random pairs of vectors $\mathbf{x}$ and $\mathbf{y}$ with given amounts of sparsity in the basis $\mathbf{\Psi}$ (e.g., $K = 1, 2, \cdots 100$).

(d) Compute the normalized inner product of these vectors in the high-dimensional ambient space, $\langle \mathbf{x}, \mathbf{y} \rangle$, and compare with the normalized inner product in compressed space. Specifically $\langle \mathbf{\Theta x}, \mathbf{\Theta y} \rangle$.

(e) Plot the error between these inner products as a function of the sparsity K. To get good statistics, with error bounds, you will likely need to test several pairs of $\mathbf{x}$ and $\mathbf{y}$ for each sparsity K (maybe 100 pairs).

(f) Repeat the above error plots with more or less rows of $\mathbf{C}$ (i.e., vary p from 10 to 1000). Plot the error in a two-dimensional `pcolor` plot of K vs p. What trends do you notice?

(g) Now, for the vectors $\mathbf{x}$ generated above, solve the compressed sensing problem for $\mathbf{s}$, the sparse vector, and compute the ℓ_0 error between the estimated $\mathbf{s}$ and the true $\mathbf{s}$. Plot the average error in a `pcolor`. How does this compare to the above?

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 4–19: Consider the following one-dimensional linear regression problem, where we are simply trying to find the constant x that best scales a vector $\mathbf{a}$ into a vector $\mathbf{b}$:

$$\begin{bmatrix} | \\ \mathbf{b} \\ | \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{a} \\ | \end{bmatrix} x = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T x.$$

The solution is given by

$$x = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}.$$

Please write down $\mathbf{U}, \mathbf{\Sigma}$, and $\mathbf{V}$ in terms of $\mathbf{a}$, $\mathbf{b}$, and x.

Write down the solution x in terms of the vectors $\mathbf{a}$ and $\mathbf{b}$.

Exercise 4–20: As a simple example, construct an artificial data set consisting of 100 observations of an outcome, arranged in a vector $\mathbf{b} \in \mathbb{R}^{100}$. Make it so that each outcome in $\mathbf{b}$ is given by a combination of exactly 2 out of 10 candidate predictors, whose observations are arranged in the rows of a matrix $\mathbf{A} \in \mathbb{R}^{100\times 10}$.

Add a small amount of noise (5%) to the vector $\mathbf{b}$ and solve for $\mathbf{x}$ using ordinary least squares, Tikhonov regularized least squares, and LASSO regression. Please comment on the results.

Compare the LASSO prediction before and after de-biasing, which involves taking the sparse pattern returned by LASSO and performing a final least squares regression just onto these terms.

Exercise 4–21: Repeat the above problem with different measurement matrices.

- Try a measurement matrix $\mathbf{C}$ where each entry is uniformly distributed between 0 and 1.
- Try a measurement matrix $\mathbf{C}$ where each entry is a Bernoulli random variable with equal probability of being a 0 or a 1.
- Try a measurement matrix $\mathbf{C}$ where each entry is a Bernoulli random variable with 95% probability of being a 0.
- Try a measurement matrix $\mathbf{C}$ where each row has a single nonzero entry of 1, which corresponds to measuring a single spatial location in $\mathbf{x}$.

How do the trends on the error versus sparsity and number of measurements p change?

Exercise 4–22: Construct a $\mathbf{C}$ and $\mathbf{\Psi}$ matrix from above with $p = 100$ and $n = 1000$ and $\mathbf{C}$ a Gaussian random matrix. Construct a sparse vector $\mathbf{s}$ with $K = 10$ and $\mathbf{x} = \mathbf{\Psi}\mathbf{s}$.

Solve this underdetermined sparse least squares problem using the three methods discussed in this chapter:

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- Use the standard compressed sensing formulation from Eq. (4.31);
- Use the constrained ℓ_1 minimization with constraints $\|\mathbf{C\Psi s} - \mathbf{y}\| < \epsilon$ from Eq. (4.32);
- Use the Dantzig selector from Eq. (4.33), which is more akin to the classic LASSO.

Add noise of varying magnitude to the measurement in $\mathbf{y} = \mathbf{C\Theta s}$ and repeat the methods above. Try different ϵ and λ and discuss the various trends and findings.

Exercise 4–23: To see the sparsity promoting effects of the ℓ_1 norm, we consider a generic underdetermined system of equations. First construct a random matrix system of equations $\mathbf{y} = \mathbf{\Theta s}$ with $p = 200$ rows (measurements) and $n = 1000$ columns (unknowns). In general, there are infinitely many solutions $\mathbf{s}$ that are consistent with these equations, unless we are very unfortunate and the row equations are linearly dependent while the measurements are inconsistent in these rows. In fact, this is an excellent example of the probabilistic thinking used more generally in compressed sensing: if we generate a linear system of equations at random, that has sufficiently many more unknowns than knowns, then the resulting equations will have infinitely many solutions with *high probability*.

Solve this system of equations using both the minimum ℓ_1 norm and minimum ℓ_2 norm solutions. The minimum ℓ_2 norm solution is obtained using the pseudo-inverse, related to the SVD. The minimum ℓ_1 norm solution is obtained via the **cvx** package.

Plot the histogram of the coefficients of both solutions $\mathbf{x}$ and comment on the results.

Exercise 4–24: In this exercise, we will reproduce the Tikhonov regularization schematic in Figs. 4.13 and Fig. 4.14.

First, generate a matrix $\mathbf{A}$ with condition number κ; start with $\kappa = 100$. To do this, you can use the SVD with

$$\mathbf{U} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \quad \mathbf{\Sigma} = \begin{bmatrix} \kappa & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}.$$

Generate an ensemble of $\mathbf{b}$ vectors centered around $\mathbf{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ with additive Gaussian noise with zero mean and standard deviation of 0.1.

Solve for $\mathbf{x}$ using least squares and plot the ellipsoid of solutions.

Now, solve for $\mathbf{x}$ using Tikhonov regularized least squares and plot the ellipsoid of solutions. Try different regularization strengths α.

Estimate the bias and the decrease in variance with various α.

Exercise 4–25: Repeat the exercise above, but with the LASSO regression.

Exercise 4–26: Generate examples of matrices $\mathbf{A} \in \mathbb{R}^{3\times 2}$ that are overdetermined and underdetermined. Repeat for matrices $\mathbf{A} \in \mathbb{R}^{2\times 3}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 5

Constraints and Duality

Constraints are an essential component of many optimization problems. Most real-world problems are fundamentally constrained, either by resources, time, material properties, cost, among others. Until now we have mostly either considered unconstrained problems, such as least-squares regression and gradient descent, or problems with the simplest constraints and cost functions, such as linear programming.

Constraints play a central role in shaping the feasible set of an optimization problem. In Chapter 3 on linear programming, we saw that the constraints almost entirely determine the problem, as the optimum always lives on the vertex of the feasible polytope defined by the constraints. We also saw constraints arise in Chapter 4 in the context of constrained linear regression. The importance of constraints in optimization extend far beyond these simple linear and quadratic problems, becoming critical for modeling a wide range of real-world systems.

We will begin this chapter with a more in-depth discussion about Lagrange multipliers for enforcing equality constraints. We will derive the conditions for optimality of a constrained problem, with several examples. Next, we will formally introduce the more general KKT conditions, which hold for inequality constraints as well. Lagrange multipliers and the KKT conditions have an intuitive geometric interpretation in terms of directional derivatives of the objective and constraint functions. Specifically, locally optimal solutions will be those such that the gradient of the objective function has no direction tangent to the constraint equations. Thus, the objective normal must be in the span of the constraint normal directions, and the linear combination of these constraint normals will be the Lagrange multipliers.

The interactions between constraints and the objective function are surprisingly rich and interesting. The concept of *duality* provides crucial insight into this connection. At its core, duality provides a deeper perspective on constrained optimization, often by reformulating the original, primal problem into a complementary dual problem. Analyzing this dual problem can reveal tight bounds on the original objective and provide methods for efficient computation. In some cases, the dual formulation may be simpler or more computationally tractable.

Duality is most easily understood through linear programming (via the primal-dual pair of linear programs) and quadratic programming (via the classical use of Lagrange multipliers and KKT conditions). However, its relevance extends well beyond these specific problem classes. In mechanics and physics, for instance, dual principles connect varia-

DRAFT

tional formulations of equilibrium conditions with complementary energy or stress-based descriptions. In the calculus of variations, dual arguments can simplify or solve certain boundary value problems by re-expressing constraints or re-labelling the roles of independent and dependent variables.

Duality will also lead us to interior point methods (IPMs), which are a powerful class of workhorse algorithms for constrained optimization. There are several variants of IPMs that leverage primal and dual formulations to efficiently find optimal solutions to convex problems. These methods may also be relaxed and used for non-convex problems, as will be discussed in Chapter 6, although there are typically no guarantees in the non-convex case.

Finally, we will introduce infinite dimensional optimization problems and the calculus of variations. This is a rich field with an equally rich history. Although it may seem beyond the scope of this chapter, the concepts of constraints and duality were first introduced in these infinite-dimensional problems in the context of classical mechanics. The Euler-Lagrange equations provide optimality conditions for infinite-dimensional optimization problems. Further, the Legendre transformation is an important transformation of the Lagrangian into a dual Hamiltonian formulation. Variational optimization is extremely relevant for broad and useful classes of infinite dimensional problems, including control theory, machine learning, and shape optimization in fluid applications.

To summarize, this chapter will accomplish three major goals:

1. Describing how to handle constraints, and how they may be put into the objective function through Lagrange multipliers and KKT conditions;

2. Discuss the closely related concept of duality, which is important in advanced algorithms, such as IPMs discussed in Sec. 5.4;

3. Discuss infinite-dimensional problems where the objective is a functional of a function to be optimized.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

5.1 Lagrange Multipliers

We begin by discussing equality constraints $\mathbf{h}(\mathbf{x}) = \mathbf{0}$, which are the simplest class of constraints to handle. The constrained optimization problem is written as

$$\min_{\mathbf{x}} \quad f(\mathbf{x}), \tag{5.1a}$$

$$\text{s.t.} \quad \mathbf{h}(\mathbf{x}) = \mathbf{0}. \tag{5.1b}$$

Geometrically, for $\mathbf{x}^*$ to be an optimal solution, the gradient of $f(\mathbf{x}^*)$ can have no component tangent to $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ at $\mathbf{x}^*$. A tangent direction $\mathbf{d}$ to $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ satisfies $\nabla \mathbf{h}(\mathbf{x}) \cdot \mathbf{d} = 0$. These tangent directions are *feasible* directions in which we can move the variable $\mathbf{x}$ to improve the objective function while still maintaining the constraint equation. If ∇f improves in one of these feasible tangent directions, then the current point is not optimal. Thus, ∇f must be normal to $\mathbf{h}(\mathbf{x})$, meaning that it is in the span of the normals $\nabla h_j(\mathbf{x})$.

This is simple to visualize for a single constraint $h(\mathbf{x}) = 0$ in two-dimensions, $\mathbf{x} \in \mathbb{R}^2$, as in Fig. 5.1. If $f(\mathbf{x})$ decreases in a direction that is tangent to h, meaning that $\nabla f(\mathbf{x})$ has a component tangent to h, then we can achieve a more optimal solution by moving in this direction. Thus, a necessary condition for optimality is that $\nabla f(\mathbf{x})$ is orthogonal to the tangent space at h, which means it is in the direction of ∇h. We will revisit and generalize this geometric intuition to higher dimensions and more constraints soon.

The method of Lagrange multipliers reformulates the constrained optimization problem, moving the constraint equations into the objective function. This introduces additional optimization variables, the Lagrange multipliers $\boldsymbol{\lambda}$, that act as regularization weights to enforce the constraint:

$$\max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} \quad f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}). \tag{5.2}$$

The augmented objective function $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x})$ is known as the Lagrangian. This is closely related to the Lagrangian from classical mechanics (Sec. 5.5) and control theory (Chapter 9). In Sec. 5.3 we will see that Eq. (5.2) is related to the dual form of the original equality-constrained primal optimization problem in Eq. (5.1).

We have thus converted a lower-dimensional constrained problem into a higher-dimensional unconstrained problem, for which their are many solution techniques discussed earlier. For example, we can use gradient descent, Newton's method and quasi-Newton methods, least squares, etc., depending on the dimension, convexity, and geometry of the problem.

The conditions for optimality of a solution $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ are the following:

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}, \tag{5.3a}$$

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}. \tag{5.3b}$$

These are the *first-order* optimality conditions or the stationarity conditions, and they are a subset of the Karush-Kuhn-Tucker (KKT) conditions, which will be discussed in Sec. 5.2.

The second condition amounts to the constraint being enforced at $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$, as

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0} \quad \implies \quad \mathbf{h}(\mathbf{x}^*) = \mathbf{0}. \tag{5.4}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

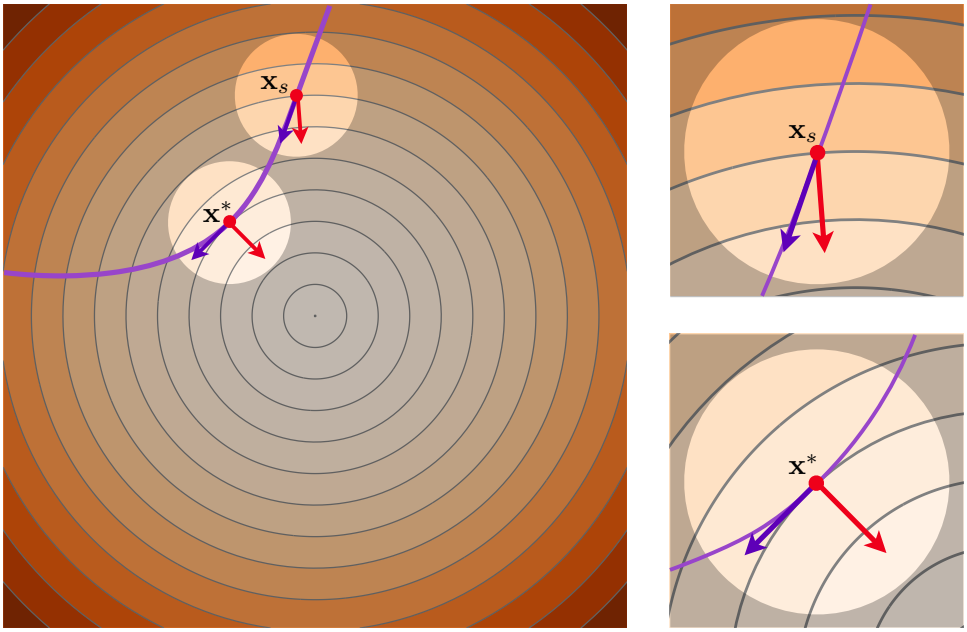


Figure 5.1: Illustration of optimality of a solution $\mathbf{x}^*$ in terms of the gradient of the objective function (red arrow) and tangent of the constraint equation (purple arrow); the constraint is shown in purple. We zoom into two points on the right. In the top, we investigate a sub-optimal point $\mathbf{x}_s$, where the constraint tangent and objective normal are not orthogonal, so we may improve the objective function by moving in the feasible constraint tangent direction. In the bottom, we see the optimal point $\mathbf{x}^*$, where the constraint tangent and objective normal directions are orthogonal. Shading illustrates regions we zoom into.

The first condition yields the following expression

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0} \quad \Longrightarrow \quad \nabla_{\mathbf{x}} f(\mathbf{x}^*) + \boldsymbol{\lambda}^{*T}\nabla_{\mathbf{x}}\mathbf{h}(\mathbf{x}^*) = \mathbf{0}. \tag{5.5}$$

This implies the following

$$\nabla_{\mathbf{x}} f(\mathbf{x}^*) = -\boldsymbol{\lambda}^{*T}\nabla_{\mathbf{x}}\mathbf{h}(\mathbf{x}^*). \tag{5.6}$$

For a single equality constraint $h(\mathbf{x}) = 0$, this simplifies to

$$\nabla_{\mathbf{x}} f(\mathbf{x}^*) = -\lambda^*\nabla_{\mathbf{x}} h(\mathbf{x}^*), \tag{5.7}$$

which is the same as the geometric interpretation that ∇f must have no component in the tangent direction to $h(\mathbf{x}) = 0$ at an optimal $\mathbf{x}^*$. Said another way, ∇f must point in a direction normal to the constraint surface defined by $h(\mathbf{x}) = 0$, given by ∇g.

In this example, *feasible directions* $\mathbf{d}$ at a point $\mathbf{x}$ satisfy

$$\nabla h(\mathbf{x}) \cdot \mathbf{d} = 0.$$

Optimality of $\mathbf{x}^*$ requires that the directional derivative of the objective function vanishes in every *feasible* direction $\mathbf{d}$:

$$\nabla f(\mathbf{x}^*) \cdot \mathbf{d} = 0 \quad \forall\, \mathbf{d} \text{ with } \nabla h(\mathbf{x}^*) \cdot \mathbf{d} = 0.$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Hence $\nabla f(\mathbf{x}^*)$ must be orthogonal to the tangent space and therefore lies in the one-dimensional normal space spanned by $\nabla h(\mathbf{x}^*)$:

$$\nabla f(\mathbf{x}^*) = \lambda \nabla h(\mathbf{x}^*)$$

The scalar λ is the *Lagrange multiplier*, providing the exact rescaling of the normals.

When we have more constraint equations, we will see that there may be several normal directions to the constraint equations, and the gradient of the objective function must be in this constraint-normal subspace. The exact linear combination of these constraint normals are the Lagrange multipliers in higher dimensions.

Simple Example

Here we introduce a concrete example to illustrate the method of Lagrange multipliers to solve a constrained optimization problem.

Consider $f(\mathbf{x})$ to be a linear objective function $\mathbf{c}^T\mathbf{x} = x + 2y$ with the constraint that our solution remains on the unit circle $x^2 + y^2 = 1$:

$$\begin{aligned} \min_{x,y} \quad & x + 2y \\ \text{s.t.} \quad & x^2 + y^2 = 1. \end{aligned}$$

We may solve this problem by hand, and also analyze the behavior of the augmented Lagrangian for various values of λ.

First, we move the constraints to the objective function via the augmented Lagrangian $\mathcal{L}(\mathbf{x}, \lambda)$:

$$\max_{\lambda} \min_{x,y} \quad x + 2y + \lambda(x^2 + y^2 - 1).$$

Notice here that we rearranged the constraint $x^2 + y^2 = 1$ to be in the standard form $h(x, y) = x^2 + y^2 - 1 = 0$. Now we minimize over (x, y) and then maximize over λ to find the optimal solution (x^*, y^*, λ^*):

$$\begin{aligned} \min_{x} \quad & x + 2y + \lambda(x^2 + y^2 - 1) \quad \implies \quad 1 + 2\lambda x = 0 \quad \implies \quad x^* = -\frac{1}{2\lambda} \\ \min_{y} \quad & x + 2y + \lambda(x^2 + y^2 - 1) \quad \implies \quad 2 + 2\lambda y = 0 \quad \implies \quad y^* = -\frac{1}{\lambda}. \end{aligned}$$

Substituting these back into the Lagrangian and optimizing over λ we have:

$$\max_{\lambda} -\frac{1}{2\lambda} - \frac{2}{\lambda} + \lambda\left(\frac{1}{4\lambda^2} + \frac{1}{\lambda^2} - 1\right) = \max_{\lambda} \frac{-5}{4\lambda} - \lambda \quad \implies \quad \lambda^* = \frac{\sqrt{5}}{2}. \tag{5.8}$$

Thus, we have the optimal point $x^* = -\frac{1}{\sqrt{5}}, y^* = -\frac{2}{\sqrt{5}}$. It is simple to verify that this point is in fact on the unit circle. As an exercise, confirm that the gradient of the objective function is normal to the constraint equation at this point.

It is interesting to visualize the augmented Lagrangian $L(x, y, \lambda)$ as a function of λ, as in Fig. 5.2. Here we plot the augmented objective function, both as a three-dimensional

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

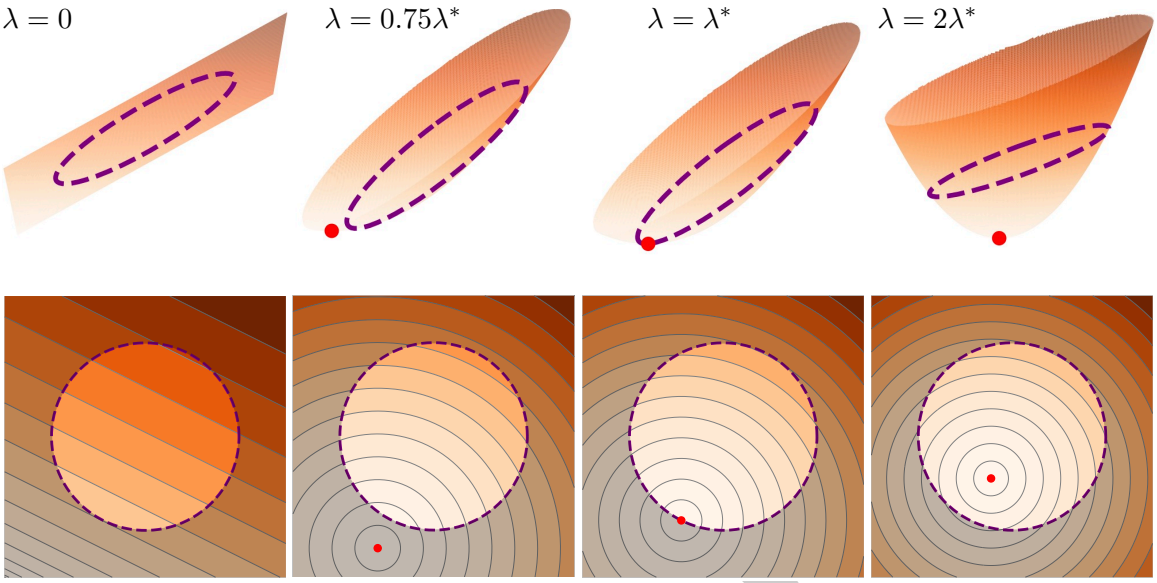


Figure 5.2: Illustration of the geometry of the augmented Lagrangian as we vary λ. The top row shows the three-dimensional surface plots of the augmented Lagrangian, and the bottom row shows the two-dimensional contour plots of the same; the purple circle denotes the constraint $x^2 + y^2 = 1$. At λ^*, the optimum value of the augmented Lagrangian is on the constraint boundary. For smaller $\lambda < \lambda^*$, the optimum is outside of the boundary, and for larger $\lambda > \lambda^*$ the optimum is inside the boundary. Therefore, the optimum λ^* is the Lagrange multiplier required so that the minimum is on the boundary of the constraint and the objective function has the same value as the augmented Lagrangian at that point.

surface, and as a two-dimensional contour plot. For $\lambda = 0$, we recover the simple linear objective function. For small $\lambda < \lambda^*$, the optimum of the augmented Lagrangian moves toward the constraint boundary. At λ^* the optimum of the Lagrangian is exactly on the constraint boundary, so $\lambda^* h(x) = 0$, indicating that we have found an optimal solution and that the value of the augmented Lagrangian is the same as that of the original objective function at this point. Further, this also implies that the constraint $h(\mathbf{x}) = 0$ is satisfied. Finally, for larger $\lambda > \lambda^*$, we see that the optimal value of the Lagrangian moves inside the constraint equation. Therefore, the Lagrange multiplier λ is weighting the constraint equation so that the optimum of the augmented objective function is *exactly* on the constraint boundary.

More Constraints

When we have more constraints, then $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ defines a *constraint manifold* in $\mathbb{R}^n$. If the constraints are convex (i.e., affine), $\mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$, then they define a constraint subspace in $\mathbb{R}^n$. In either case, a necessary condition for optimality of a point $\mathbf{x}^*$ is that the gradient of f must have no component in the tangent directions to $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ at $\mathbf{x}^*$. Otherwise, we could improve the solution by moving in this direction.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

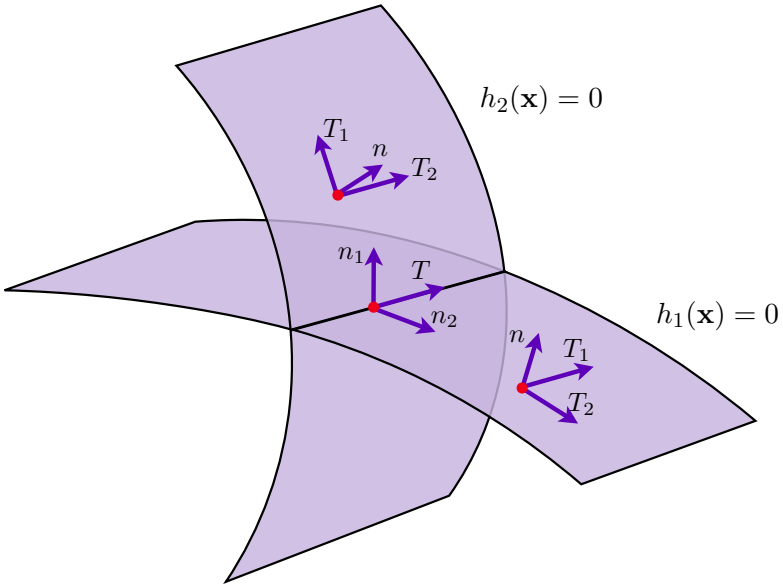


Figure 5.3: Geometric interpretation of constraint equations in higher dimensions as defining feasible submanifolds.

The tangent space $T_{\mathbf{x}}$ to the constraint manifold at a point $\mathbf{x}$ is the linear subspace:

$$T_{\mathbf{x}} = \{\mathbf{d} \,|\, \nabla h_j(\mathbf{x}) \cdot \mathbf{d} = 0 \,\forall j\}. \tag{5.9}$$

For the gradient of the objective function, ∇f, to not have any component in the tangent space, it must be a linear combination of the constraint normals $\{\nabla h_j(\mathbf{x})\}$. The necessary condition in Eq. (5.6) establishes exactly this linear combination, where the Lagrange multipliers λ_j are exactly the coordinates of $\nabla f(\mathbf{x})$ in the basis of constraint normals. If $\nabla f = -\boldsymbol{\lambda}^T \nabla \mathbf{h}$, then we may write the following:

$$\nabla f(\mathbf{x}^*) = -\lambda_j \nabla h_j(\mathbf{x}^*) \tag{5.10}$$

which implies that

$$\nabla f \cdot \nabla h_j \neq 0, \tag{5.11}$$

so that ∇f is not in the tangent space $T_{\mathbf{x}^*}$. This can be visualized geometrically in Fig. 5.3.

Tilt and Match Interpretation. There is an intuitive geometric interpretation of Lagrange multipliers, building on the intuition from Fig. 5.2. We think of the Lagrange multiplier as defining a family of tilted objective functions

$$f_{\boldsymbol{\lambda}}(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}).$$

We then vary $\boldsymbol{\lambda}$ until a level set $f_{\boldsymbol{\lambda}=\boldsymbol{\lambda}^*}$ becomes *tangent* to the constraint surface $\mathbf{h}(\mathbf{x}) = \mathbf{0}$. At that critical $\boldsymbol{\lambda}^*$, the first-order condition $\nabla f + \boldsymbol{\lambda}^T \nabla \mathbf{h} = \mathbf{0}$ is satisfied.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

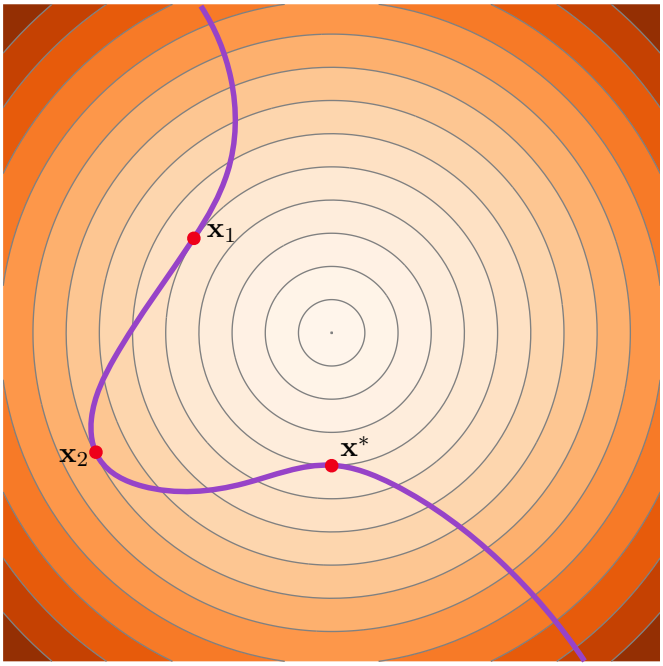


Figure 5.4: For a non-convex optimization problem, having the constraint tangent and objective normal directions be orthogonal is a necessary but not sufficient condition for the solution to be globally optimal. Here we see three points that satisfy this condition: $\mathbf{x}_1$ is a local minimum, $\mathbf{x}_2$ is a local maximum, and $\mathbf{x}^*$ is a global minimum.

Convexity and Sufficient Conditions

The optimality conditions above are *necessary* conditions for a state $\mathbf{x}^*$ to be optimal. In the case of a convex problem, then they are also *sufficient*. Again, for $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ to be convex it must be an affine constraint $\mathbf{A}\mathbf{x} = \mathbf{b}$. However, if either f or the feasible set defined by $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ are non-convex, then the optimality conditions may not be sufficient. Essentially, these conditions just indicate a stationary point, but it may be a local minimum, maximum, or saddle point, similar to the case of gradient descent. We must then introduce second-order conditions to test for optimality. Again, we let $T_{\mathbf{x}}$ be the tangent space to the constraint manifold at a point $\mathbf{x}$:

$$T_{\mathbf{x}} = \{\mathbf{d} \,|\, \nabla \mathbf{h}(\mathbf{x}) \cdot \mathbf{d} = 0\}\,.$$

We conclude that $\mathbf{x}^*$ is a strict local minimum if

$$\mathbf{d}^T \nabla^2_{\mathbf{x}\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{d} > 0 \quad \forall \mathbf{d} \in T_{\mathbf{x}}$$

as long as $\mathbf{d} \neq \mathbf{0}$. Figure 5.4 shows an example of a non-convex constraint equation that gives rise to several locally optimal points that satisfy the necessary conditions.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

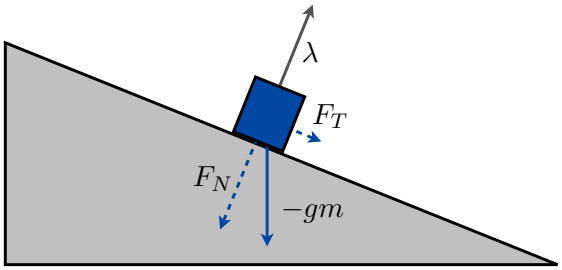


Figure 5.5: Here we have a simple illustration of a physical constraint arising from a block sliding down an incline without friction. In this case, gravity pulls down at a constant force, and the ramp applies a normal force so that the block stays on top.

Regularity Conditions

The use of Lagrange multipliers requires a mild set of regularity conditions, called constraint qualifications, to ensure the KKT conditions apply. These conditions rule out pathological cases, such as when the active constraints at a point are not linearly independent or when the feasible region has no interior.

For the case of equality constraints, a key regularity condition is that at a feasible point $\mathbf{x}^*$, the gradients of the constraint equations must be linearly independent. In other words

$$\{\nabla h_1(\mathbf{x}^*), \dots, \nabla h_m(\mathbf{x}^*)\}$$

must all be linearly independent. This ensures that the Jacobian $\nabla \mathbf{h}(\mathbf{x}^*)$ has full row rank. If the gradients of the constraints are linearly dependent, then you can't form a meaningful Lagrange multiplier condition, as there might be multiple or no solutions to the stationarity equation. Similar regularity conditions exist for the more general KKT theory of inequality constraints, and these will be discussed more in the next section.

History

Lagrange first introduced the method of multipliers to eliminate constraint forces in holonomic[1] mechanics [213, 241]. Figure 5.5 shows a simple mechanical system of a block sliding down an inclined plane, where the plane exerts a force to keep the block exactly on the boundary; this force is a *Lagrange multiplier* enforcing the constraint that the block remains exactly on the boundary of the inclined plane. We will discuss the broader connections of constraints, Lagrange multipliers, and classical mechanics in Section 5.5.

Nearly two hundred years later, this approach was generalized to handle inequality constraints by Karush [198] and popularized by Kuhn and Tucker [210], marking the birth of modern nonlinear programming. Since then, the field has developed rapidly, with the theory of duality and the primal-dual algorithms that followed.

[1] A *holonomic constraint* is a constraint on a mechanical system that can be expressed as an explicit equation relating the generalized coordinates, typically in the form $h(q_1, q_2, \dots, q_n, t) = 0$. Such constraints restrict the configuration space but not the velocities independently. In contrast, *nonholonomic* constraints involve the velocities and cannot be integrated to purely positional constraints. For example, a coin rolling without slipping is a non-holonomic system, while a block sliding down an incline is a holonomic system.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

5.2 Karush–Kuhn–Tucker (KKT) Conditions

In the previous section, we saw how Lagrange multipliers allow us to enforce equality constraints by augmenting the objective function. Here we introduce the KKT conditions for optimality [198, 210], which generalize the Lagrange multiplier approach to also handle inequality constraints of the form $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}), \tag{5.12a}$$

$$\text{s.t.} \quad \mathbf{h}(\mathbf{x}) = \mathbf{0} \tag{5.12b}$$

$$\mathbf{g}(\mathbf{x}) \leq \mathbf{0}. \tag{5.12c}$$

We assume that we have m equality constraints $h_j(\mathbf{x}) = 0$ for $j = 1, \cdots, m$, and p inequality constraints $g_j(\mathbf{x}) \leq 0$ for $j = 1, \cdots, p$.

Similar to the approach of Lagrange multipliers, the constrained minimization problem is recast into Lagrangian form, where the optimal solution is the global minimum over the domain of the decision variables and a global maximum over the Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. We define the augmented Lagrangian as

$$\begin{aligned}\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= f(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j \, h_j(\mathbf{x}) + \sum_{j=1}^{p} \mu_j \, g_j(\mathbf{x}) \\ &= f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}).\end{aligned}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^p_+$ are the Lagrange multipliers for equality and inequality constraints, respectively. Here $\mathbb{R}_+$ indicates the non-negative reals, so $\mu \geq \mathbf{0}$.

The first-order KKT conditions provide a set of necessary conditions for the point $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ to be optimal:

$$\text{(stationarity)} \quad \nabla f(\mathbf{x}^*) + \boldsymbol{\lambda}^{*T} \nabla \mathbf{h}(\mathbf{x}^*) + \boldsymbol{\mu}^{*T} \nabla \mathbf{g}(\mathbf{x}^*) = \mathbf{0} \tag{5.13a}$$

$$\text{(primal feasibility)} \quad \mathbf{h}(\mathbf{x}^*) = \mathbf{0}, \ \mathbf{g}(\mathbf{x}^*) \leq \mathbf{0} \tag{5.13b}$$

$$\text{(dual feasibility)} \quad \boldsymbol{\mu}^* \geq \mathbf{0}, \tag{5.13c}$$

$$\text{(complementary slackness)} \quad \boldsymbol{\mu}^{*T} \mathbf{g}(\mathbf{x}) = 0. \tag{5.13d}$$

If $f(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are convex and $\mathbf{h}(\mathbf{x})$ are affine, so that the optimization problem is convex, then the KKT conditions are necessary and sufficient for a point to be globally optimal[2]. For non-convex problems, the KKT system gives necessary first-order conditions.

The stationarity condition comes from $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}$, and this generalizes the unconstrained optimality condition that $\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \mathbf{0}$. We see that in the absence of inequality constraints, we recover the stationarity condition for equality constraints in Eq. (5.5). The stationarity equation also states that the gradient of the objective function is in the span of the active constraint normals. Recall that some of the inequality constraints may be inactive, with $\mu_j = 0$, so only the active constraint normals are considered.

The primal feasibility equation states that the primal point $\mathbf{x}^*$ must be in the feasible set, which obviously must be satisfied. Dual feasibility states that each μ_j must be non-negative, which is slightly less obvious. In Section 5.3 on duality, we will see that we can

[2]There are mild additional conditions that must also be satisfied, and we will discuss these soon.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

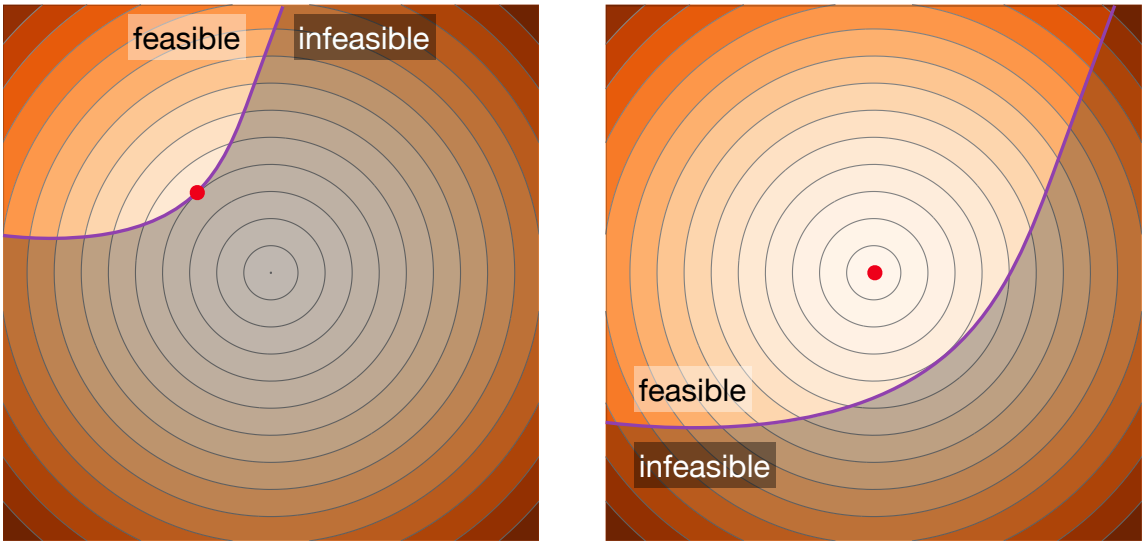


Figure 5.6: The KKT conditions generalize the Lagrange multiplier optimality conditions for equality constraints to inequality constraints. Here we see two examples of an inequality constrained optimization. In the first case on the left, the optimal point is on the edge of the feasible set, so the constraint is active. In the second case on the right, the optimal point is on the interior of the feasible set, so the constraint is inactive. In both cases, the complementary slackness condition $\boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) = 0$ is satisfied.

rewrite the primal optimization problem in Eq. (5.12) in terms of a maximization of a dual function $\phi(\boldsymbol{\lambda}, \boldsymbol{\mu})$ in terms of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. This dual function will have a term $\boldsymbol{\mu}^T \mathbf{g}(\mathbf{x})$, so if any μ_j is negative, then multiplying with any inactive negative constraints $g_j(\mathbf{x}) < 0$, we can drive this maximization to infinity. Thus, $\boldsymbol{\mu} \geq \mathbf{0}$ is required to prevent this.

The fourth complementary slackness condition, Eq. (5.13d), is at the heart of the KKT conditions for inequality constraints. It states that $\mu_j^* g_j(\mathbf{x}^*) = 0$ for each j, so either $\mu_j = 0$ or $g_j(\mathbf{x}^*) = 0$. This means that either $g_j(\mathbf{x}^*) = 0$ is *active*, in which case it acts like a standard equality constraint and $\mu_j > 0$ is a standard Lagrange multiplier, or $g_j(\mathbf{x}^*) < 0$ is *inactive*, in which case $\mu_j = 0$. When the constraint is inactive, so $\mu_j = 0$, then this constraint cannot push on the optimum solution via the gradient in the stationarity condition in Eq. (5.13a). These two cases are illustrated schematically in Fig. 5.6.

Geometry of Complementary Slackness and Feasible Directions. Consider a single inequality constraint $g_j(\mathbf{x}) \leq 0$. The boundary $g_j(\mathbf{x}) = 0$ defines a surface with outward-pointing normal $\nabla g_j(\mathbf{x})$. At a point on the boundary, motion is permitted only in directions $\mathbf{d}$ that satisfy $\nabla g_j(\mathbf{x}) \cdot \mathbf{d} \leq 0$, thereby maintaining feasibility. If the constraint is active, the KKT stationarity condition includes the term $\mu_j \nabla g_j(\mathbf{x})$, where $\mu_j \geq 0$ acts as a Lagrange multiplier *pushing* against the boundary of the constraint. If the constraint is inactive, so $g_j(\mathbf{x}) < 0$, then stationarity omits $\nabla g_j(\mathbf{x})$, since $\mu_j = 0$. Thus, the complementary slackness condition $\mu_j g_j(\mathbf{x}) = 0$ ensures that the gradient contribution $\mu_j \nabla g_j(\mathbf{x})$ appears in the stationarity condition if and only if the constraint is active.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Algorithmic Perspective

Algorithmically, the KKT system of equations underpin many of the most powerful modern algorithms for constrainted optimization, including active-set and interior-point methods, augmented-Lagrangian methods, as well as the simple projected gradient method for box-constrained least squares. These algorithms are closely related to the notion of duality, and we will discuss them more in Chapters 5 and 6. We provide a brief overview here:

(a) Active-Set Methods (ASM) and Sequential Quadratic Programming (SQP). Active set methods involve guessing which inequality constraints are active and become equality constraints, solving the unconstrained augmented Lagrangian in Eq. (5.2), for example with Newton's method, and testing which constraints enter or exit after this iteration. When $f(\mathbf{x})$ is quadratic with constant Hessian, as in least-squares regression, then this becomes a sequential quadratic program.

(b) Interior-Point Methods (IPM). IPMs attempt to iteratively solve for an optimal point $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ that satisfy the KKT conditions. We replace each $g_j(\mathbf{x}) \leq 0$ in the augmented objective function with a log-barrier function $-t\log(-g_j(\mathbf{x}))$. A Newton iteration on the barrier-augmented objective function will enforce $g_j(\mathbf{x}) \leq 0$. As $t \to 0$, the solution converges to a KKT point. We will explore IPMs in detail in Section 5.4.

(c) Augmented Lagrangian (ALM). Here we rewrite our inequality constraints $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ as equality constraints by introducing slack variables: $\mathbf{g}(\mathbf{x}) + \mathbf{s} = \mathbf{0}$ with $\mathbf{s} \geq \mathbf{0}$. We then write the augmented Lagrangian to include penalty terms $\frac{\rho}{2}\|\mathbf{g}(\mathbf{x}) + \mathbf{s}\|^2$ to enforce feasibility. An outer-loop optimization updates the multipliers $(\boldsymbol{\lambda}, \boldsymbol{\mu})$, while the inner-loop optimization minimizes the augmented Lagrangian over $(\mathbf{x}, \mathbf{s})$, often using unconstrained solvers, such as least-squares regression if f is quadratic. We will discuss augmented Lagrangian methods more in Chapter 6.

(d) Projected Gradient Method. When the feasible set is defined by simple box constraints, i.e., $\boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}$, we can enforce feasibility by projecting each step of the unconstrained gradient method onto the box $[\boldsymbol{\ell}, \mathbf{u}]$. This corresponds to the special case of the KKT conditions where each bound constraint is treated as a separate inequality, and the associated multipliers are recovered by a soft-thresholding operation on the gradient. We will explore this in the example at the end of this section.

Regularity Conditions

There are regularity conditions, known as constraint qualifications, for when a constrained optimizer $\mathbf{x}^*$ will satisfy the KKT conditions. The regularity conditions are more complicated in the case of mixed equality and inequality constraints than those for simple equality constraints in Section 6.1. In this case, several constraint qualifications can be used:

- **Linear Independence Constraint Qualification (LICQ).** This is one of the most commonly used regularity conditions. As before, the gradients of all active constraints (i.e., equality constraints $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ and inequality constraints $g_j(\mathbf{x}^*) = 0$ active at $\mathbf{x}^*$) must be linearly independent. LICQ ensures that the Lagrange multipliers $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ are uniquely defined and that the stationarity condition captures all active constraints.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- **Mangasarian–Fromovitz Constraint Qualification (MFCQ).** This is a bit weaker than LICQ and states that the equality constraint gradients $\nabla \mathbf{h}(\mathbf{x}^*)$ are linearly independent, and that there exists a direction $\mathbf{d} \in \mathbb{R}^n$ such that:

$$\nabla h_i(\mathbf{x}^*) \cdot \mathbf{d} = 0 \quad \text{for all } i, \qquad \nabla g_j(\mathbf{x}^*) \cdot \mathbf{d} < 0 \quad \text{for all active } j.$$

  This condition guarantees the existence of a feasible direction that moves away from the active inequality constraints. In other words, the feasible set is not "pointed" or degenerate at $\mathbf{x}^*$. Unlike LICQ, MFCQ allows for linear dependence among active inequalities, as long as there is a direction of descent that maintains feasibility.

- **Slater's Condition.** For convex optimization problems, Slater's condition provides a powerful and practical regularity test. It simply requires the existence of a strictly feasible point $\mathbf{x}_0$ such that:

$$\mathbf{h}(\mathbf{x}_0) = \mathbf{0} \quad \text{and} \quad \mathbf{g}(\mathbf{x}_0) < \mathbf{0}$$

  so that all inequality and equality constraints are satisfied strictly at $\mathbf{x}_0$. Slater's condition guarantees strong duality and ensures that the KKT conditions are necessary and sufficient for optimality in convex problems. Note that this only has to be true for a single point $\mathbf{x}_0$ in the feasible region, and gives a powerful global result. For linear programs, Slater's condition automatically holds if the feasible set has a non-empty interior. For non-convex problems, Slater's condition does not guarantee strong duality or the sufficiency of the KKT conditions. It may still ensure that the KKT conditions are necessary for optimality, but only if other constraint qualifications, such as MFCQ or LICQ, are also satisfied. We will revisit this in Section 5.3.

Example: Least Squares with Box Constraints

Here we will analyze the constrained least-squares regression problem with box constraints on the regression variable $\mathbf{x}$

$$\min_{\mathbf{x}} \quad \tfrac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \tag{5.14a}$$

$$\text{s.t.} \quad \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}. \tag{5.14b}$$

We may also write the box constraints on each coordinate x_j as $\ell_j \leq x_j \leq u_j$.

We briefly presented a python code to solve the least-squares regression problem with box constraints in Sec. 4.6. To solve this using the KKT conditions, we introduce dual variables $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ for the upper and lower bounds $\boldsymbol{\ell}$ and $\mathbf{u}$, respectively. The augmented Lagrangian is given by

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\nu}) = \tfrac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \boldsymbol{\mu}^T\mathbf{x} - \boldsymbol{\nu}^T\mathbf{x}.$$

The KKT conditions then become

$$\begin{aligned}
&\mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \boldsymbol{\mu} - \boldsymbol{\nu} = \mathbf{0} \\
&\boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}, \\
&\boldsymbol{\mu}, \boldsymbol{\nu} \geq \mathbf{0} \\
&\boldsymbol{\mu}^T(\mathbf{x} - \boldsymbol{\ell}) = 0, \quad \boldsymbol{\nu}^T(\mathbf{u} - \mathbf{x}) = 0.
\end{aligned}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We may solve for an optimal point that satisfies the KKT conditions using several algorithms. Because the problem is convex, the KKT conditions are both necessary and sufficient[3]. Here we will sketch the interior point method and projected-gradient algorithm approaches. We will revisit the IPM approach in Section 5.4.

Interior Point Method. The idea here is to represent each constraint using a logarithmic barrier function in the augmented Lagrangian objective function

$$\phi_t(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 - t\sum_j \left(\log(x_j - \ell_j) + \log(u_j - x_j)\right).$$

This results in an unconstrained problem that may be solved via Netwon's method. We will show in Section 5.4 that as $t \to 0$, this will converge to a solution of the KKT conditions. The associated dual Lagrange multipliers are $\mu_j = t/(x_j - \ell_j)$ and $\nu_j = t/(u_j - x_j)$.

Projected Gradient Algorithm. The basic idea of the projected gradient algorithm is to start with an initial $\mathbf{x}_0$ in the box $[\boldsymbol{\ell}, \mathbf{u}]$, take an unconstrained gradient iteration step, and project the solution back into the feasible box. We then iterate until convergence.

We denote the unconstrained least-squares objective function by $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$. At each iteration of gradient descent, we take an unconstrained step:

$$\mathbf{z}_{k+1} = \mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k)$$

and project it back into the feasible box:

$$\mathbf{x}_{k+1} = \mathcal{P}_{[\boldsymbol{\ell},\mathbf{u}]}(\mathbf{z}_{k+1}) \quad \text{where} \quad \left[\mathcal{P}_{[\boldsymbol{\ell},\mathbf{u}]}(\mathbf{z})\right]_j = \min\left\{u_j, \max\{\ell_j, z_j\}\right\}.$$

The projection step in the projected-gradient algorithm can thus be viewed as applying a *soft-thresholding* operator to enforce the KKT conditions coordinate-wise. Convergence implies the existence of Lagrange multipliers $(\boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$, even if they are not computed directly.

We may explicitly write down the KKT conditions for box constraints at a minimizer $\mathbf{x}^*$. First, we rewrite the KKT stationarity condition $\nabla f(\mathbf{x}^*) + \boldsymbol{\mu} - \boldsymbol{\nu} = \mathbf{0}$ in coordinates $\frac{\partial f}{\partial x_j}(\mathbf{x}^*) = \nu_j - \mu_j$. The KKT conditions at an optimal $\mathbf{x}^*$ then become:

- If $\ell_j < x_j^* < u_j$, then both constraints are inactive, $\mu_j = \nu_j = 0$ and $\frac{\partial f}{\partial x_j}(\mathbf{x}^*) = 0$.
- If $x_j^* = \ell_j$ then the lower bound is active, $\mu_j \geq 0,\ \nu_j = 0$ and $\frac{\partial f}{\partial x_j}(\mathbf{x}^*) = -\mu_j \leq 0$.
- If $x_j^* = u_j$ then the upper bound is active, $\mu_j = 0,\ \nu_j \geq 0$ and $\frac{\partial f}{\partial x_j}(\mathbf{x}^*) = \nu_j \geq 0$.

Thus, the signs of the gradient components tell us which bound is active and correspond directly to the Lagrange multipliers μ_j and ν_j. This is a special case of the KKT conditions for general inequality-constrained optimization.

Note that there is another, more technical way to state this. At a minimizer $\mathbf{x}^*$, the KKT conditions imply $-\nabla f(\mathbf{x}^*) \in N_{[\boldsymbol{\ell},\mathbf{u}]}(\mathbf{x}^*)$, where $N_{[\boldsymbol{\ell},\mathbf{u}]}(\mathbf{x}^*)$ is defined as the *normal cone* of the box constraint. The normal cone is a set of vectors that *push outward* from the constraint set at $\mathbf{x}^*$, meaning a set of vectors that are perpendicular to the boundary of the feasible set.

[3] We will leave it as an exercise to verify the regularity conditions are satisfied for this problem.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Saddle Point Property of KKT Optimal Point

The KKT conditions imply that the optimal point $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ will be a saddle point of the constrained Lagrangian $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ in the following sense:

$$\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \leq \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$$

for all $\mathbf{x}$ in the primal feasible set and for all $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ in the dual feasible set.

The right inequality says that $\mathbf{x}^*$ is a primal minimizer of the Lagrangian over feasible $\mathbf{x}$ for fixed $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$. Similarly, the left inequality says that $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ are dual maximizers of the Lagrangian over feasible $\boldsymbol{\lambda}, \boldsymbol{\mu}$ for a fixed $\mathbf{x}^*$. In general, for non-convex problems, this saddle point condition holds only locally, as KKT points may correspond to local but not global optima.

This saddle-point view is especially useful when proving strong duality in convex programs. It is often equivalent to the condition that the primal and dual optimal values coincide[4]:

$$\min_{\mathbf{x} \in \boldsymbol{\mathcal{S}}} f(\mathbf{x}) = \max_{\boldsymbol{\lambda}; \boldsymbol{\mu} \geq 0} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

Strong duality implies the existence of such a saddle point, and the converse is also true for convex problems. Suppose strong duality holds:

$$\min_{\mathbf{x} \in \boldsymbol{\mathcal{S}}} f(\mathbf{x}) = \max_{\boldsymbol{\lambda}; \boldsymbol{\mu} \geq 0} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

Let $\mathbf{x}^*$ be a primal minimizer, and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be a dual maximizer. Then:

$$\mathbf{x}^* \in \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*),$$

and

$$(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \in \arg\max_{\boldsymbol{\lambda}; \boldsymbol{\mu} \geq 0} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

Hence, the pair $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ satisfies the saddle point inequality:

$$\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}) \leq \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \leq \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$$

for all $\mathbf{x} \in \boldsymbol{\mathcal{S}}$, $\boldsymbol{\mu} \geq 0$, and unconstrained $\boldsymbol{\lambda} \in \mathbb{R}^m$.

For a convex optimization problem with differentiable functions f, $\mathbf{h}$, and $\mathbf{g}$, and when Slater's condition is satisfied, then the following statements are all equivalent:

1. The KKT conditions hold at $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$;
2. The point $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is a saddle point of $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$;
3. Strong duality holds:
$$\min_{\mathbf{x} \in \boldsymbol{\mathcal{S}}} f(\mathbf{x}) = \max_{\boldsymbol{\lambda}; \boldsymbol{\mu} \geq 0} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

We will explore this idea in several homework exercises at the end of this chapter.

[4] Technically the $\min_{\mathbf{x}}$ will be replaced with the *infimum* $\inf_{\mathbf{x}}$, and we will introduce this in the next section.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

5.3 Duality

Duality is one of the most profound and useful concepts in optimization. For every constrained optimization problem, there is a *dual* optimization problem that provides a lower bound to the optimum of the original *primal* problem. We have already seen a simple example of a dual optimization problem in Chapter 3 on linear programming. These ideas generalize naturally to other convex and nonconvex optimization problems. Regardless of the convexity of the primal problem, the dual problem is always concave.

The dual optimization problem is closely related to Lagrange multipliers and the KKT conditions. Specifically, the Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the dual optimization variables. We will see that in the case of simple equality constraints, the dual problem will be formulated in terms of the Lagrange multipliers $\boldsymbol{\lambda}$ with no dual constraints. In the case of equality and inequality constraints, the dual problem will be formulated in terms of the Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ with simple positivity constraints on $\boldsymbol{\mu}$.

The relationship between primal and dual problems form the basis of interior point methods, which are workhorse algorithms for modern high-dimensional constrained optimization, as discussed in Section 5.4. Duality is also important in the alternating descent method of multipliers (ADMM), discussed in Chapter 6, and branch-and-bound algorithms for combinatorial and non-convex problems. These approaches are central for many problems we have already seen, such as box-constrained least squares, LASSO, and ridge regression. It is typically simpler to solve the concave dual problem, which provides a provable lower-bound on the primal optimization. When the dual equals the primal, then no further improvement is possible, providing a *certificate* of optimality.

Given a constrained optimization problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}), && \text{(5.15a)} \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} && \text{(5.15b)} \\ \text{and} \quad & \mathbf{g}(\mathbf{x}) \leq \mathbf{0}. && \text{(5.15c)} \end{aligned}$$

then the dual function[5] is

$$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \right). \tag{5.16}$$

The dual problem is to find the best lower bound on the primal optimal value by solving

$$\begin{aligned} \max_{\boldsymbol{\lambda}, \boldsymbol{\mu}} \quad & \phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) && \text{(5.17a)} \\ \text{s.t.} \quad & \boldsymbol{\mu} \geq \mathbf{0} && \text{(5.17b)} \end{aligned}$$

The dual function is the infimum of the augmented Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}). \tag{5.18}$$

[5] The *infimum* (inf) of a function is its greatest lower bound. It generalizes the minimum when the minimum might not be attained. We write $\inf_{\mathbf{x}} f(\mathbf{x})$ to refer to the lowest value that $f(\mathbf{x})$ *approaches* over the feasible domain $\mathbf{x} \in \boldsymbol{\mathcal{S}}$, even if no $\mathbf{x}$ actually achieves it. We use *inf* to ensure well-defined expressions, even when the primal function does not attain its minimum. It can be replaced with $\min$ when it exists.

Example: Let $f(x) = x$ over the open interval $(0, 1)$. The infimum of f is equal to zero, $\inf_{x \in (0,1)} f(x) = 0$, even though $\min_{x \in (0,1)} f(x)$ does not exist because $f(x) \to 0$ as $x \to 0^+$, but $0 \notin (0, 1)$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We don't restrict the infimum to the feasible set $\mathbf{x} \in \boldsymbol{\mathcal{S}}$, and instead the infimum is taken over the entire domain of $\mathbf{x}$. The dual function is always concave, even if the primal problem is non-convex, as Eq. (5.16) is the pointwise infimum of functions that are affine in $(\boldsymbol{\lambda}, \boldsymbol{\mu})$.

We have the following fundamental relationship between the primal and dual optimization problems, called weak duality, which states that the optimum value of the dual problem is always a lower bound to the optimum value of the primal problem:

$$\max_{\boldsymbol{\lambda};\boldsymbol{\mu}\geq\mathbf{0}} \phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \min_{\mathbf{x}\in\boldsymbol{\mathcal{S}}} f(\mathbf{x}). \tag{5.19}$$

This is an extremely useful property for algorithms and optimality certificates. Specifically, the dual variables (i.e., the Lagrange multipliers $(\boldsymbol{\lambda}, \boldsymbol{\mu})$) allow us to apply pressure to the constraints, so that we may tune these dual variables to maximize the lower bound you can guarantee. The dual function measures how low you can push the objective function if you allow constraint violations but penalize them. Thus, dual solutions give safe lower bounds on the primal problem, even if you allow infeasible points.

There is an even more fundamental property of the primal-dual relationship that even holds for non-optimal points. Weak duality always holds, and states that for any feasible primal $\mathbf{x} \in \boldsymbol{\mathcal{S}}$ and any feasible dual $(\boldsymbol{\lambda}, \boldsymbol{\mu})$, then

$$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq f(\mathbf{x}). \tag{5.20}$$

Thus, the value of any dual feasible point is a lower bound on the value of any primal feasible point. This is always true, even for nonconvex problems.

Strong duality is a special case when the dual and primal optima are equal. Strong duality holds for convex problems under mild regularity conditions, such as Slater's condition, namely the existence of a strictly feasible point. Under these mild assumptions, strong duality states that the primal and dual optimum values are equal to each other:

$$\phi(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}^*). \tag{5.21}$$

For nonconvex problems strong duality might not hold and there may be a duality gap.

Dual variables are also important to quantify the sensitivity of the optimum to perturbations in the constraint right-hand sides. When the necessary conditions are met, dual variables are the partial derivatives of the optimal value with respect to constraint right-hand sides. This connection bridges optimization theory with sensitivity analysis and supports the practical use of duality in engineering, economics, and control.

Duality for Equality Constraints Only

If the primal has equality constraints and no inequality constraints, the dual becomes

$$\max_{\boldsymbol{\lambda}} \ \phi(\boldsymbol{\lambda})$$

in terms of a dual function

$$\phi(\boldsymbol{\lambda}) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) \right).$$

There are no constraints on the dual variables $\boldsymbol{\lambda}$ in the dual problem. Only inequality constraints induce constraints on the dual variables $\boldsymbol{\mu}$. We may then solve the unconstrained dual problem using simpler techniques, such as Newton's method or gradient descent.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Dual Linear Programs

We now illustrate the key concepts of duality for the linear program in Chapter 3. We will begin with a simple and straightforward derivation of the dual problem, and then will provide some intuition for what this dual linear program means.

Deriving the Dual Linear Program. We begin with the basic linear program, where we write the positivity constraint $\mathbf{x} \geq \mathbf{0}$ in the maximization statement:

$$\max_{\mathbf{x} \geq \mathbf{0}} \quad \mathbf{c}^T \mathbf{x}, \tag{5.22a}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}. \tag{5.22b}$$

To be explicit, we will rearrange the inequality constraints so that $\mathbf{b} - \mathbf{A}\mathbf{x} \geq \mathbf{0}$:

$$\max_{\mathbf{x} \geq \mathbf{0}} \quad \mathbf{c}^T \mathbf{x}, \tag{5.23a}$$

$$\text{s.t.} \quad \mathbf{b} - \mathbf{A}\mathbf{x} \geq \mathbf{0}. \tag{5.23b}$$

Now we are able to move the constraints into the augmented Lagrangian objective function using Lagrange multipliers $\mathbf{y}$

$$\min_{\mathbf{y} \geq \mathbf{0}} \max_{\mathbf{x} \geq \mathbf{0}} \quad \underbrace{\mathbf{c}^T \mathbf{x} + \mathbf{y}^T (\mathbf{b} - \mathbf{A}\mathbf{x})}_{\mathcal{L}(\mathbf{x}, \mathbf{y})}. \tag{5.24}$$

Note that $\mathbf{y}$ is non-negative because we augment with inequality constraints via the KKT conditions above; i.e., it is playing the role of $\boldsymbol{\mu}$ in Eq. (5.18).

We may now arrange the terms in the augmented Lagrangian, collecting terms with $\mathbf{x}$, which will become our Lagrange multipliers for the dual problem:

$$\min_{\mathbf{y} \geq \mathbf{0}} \max_{\mathbf{x} \geq \mathbf{0}} \quad \mathbf{y}^T \mathbf{b} + (\mathbf{c}^T - \mathbf{y}^T \mathbf{A})\mathbf{x}. \tag{5.25}$$

We now transpose terms in the objective function, because it is a scalar:

$$\min_{\mathbf{y} \geq \mathbf{0}} \max_{\mathbf{x} \geq \mathbf{0}} \quad \mathbf{b}^T \mathbf{y} + \mathbf{x}^T (\mathbf{c} - \mathbf{A}^T \mathbf{y}). \tag{5.26}$$

We may finally swap the order of the maximization over $\mathbf{x}$ and minimization over $\mathbf{y}$, because we have strong duality for linear programs[6]

The term $\mathbf{x}^T$ is now the Lagrange multiplier for the inequality constraints $\mathbf{c} - \mathbf{A}^T \mathbf{y} \leq \mathbf{0}$ in the dual problem. We move these augmented terms back to inequality constraints

$$\min_{\mathbf{y} \geq \mathbf{0}} \quad \mathbf{b}^T \mathbf{y}, \tag{5.28a}$$

$$\text{s.t.} \quad \mathbf{A}^T \mathbf{y} - \mathbf{c} \geq \mathbf{0} \tag{5.28b}$$

[6] For convex linear programs we have strong duality and the expression

$$\max_{\mathbf{x} \geq \mathbf{0}} \min_{\mathbf{y} \geq \mathbf{0}} \quad \mathbf{c}^T \mathbf{x} + \mathbf{y}^T (\mathbf{b} - \mathbf{A}\mathbf{x}) \qquad \leq \qquad \min_{\mathbf{y} \geq \mathbf{0}} \max_{\mathbf{x} \geq \mathbf{0}} \quad \mathbf{c}^T \mathbf{x} + \mathbf{y}^T (\mathbf{b} - \mathbf{A}\mathbf{x}). \tag{5.27}$$

becomes an equality. Thus we may also swap their order in Eq.(5.25) and Eq. (5.26). More generally, we may think of $\max_{\mathbf{x}}$ and $\min_{\mathbf{y}}$ as two players in a game, so swapping their order gives $\min_{\mathbf{y}}$ the advantage of going second, hence the inequality and weak duality.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

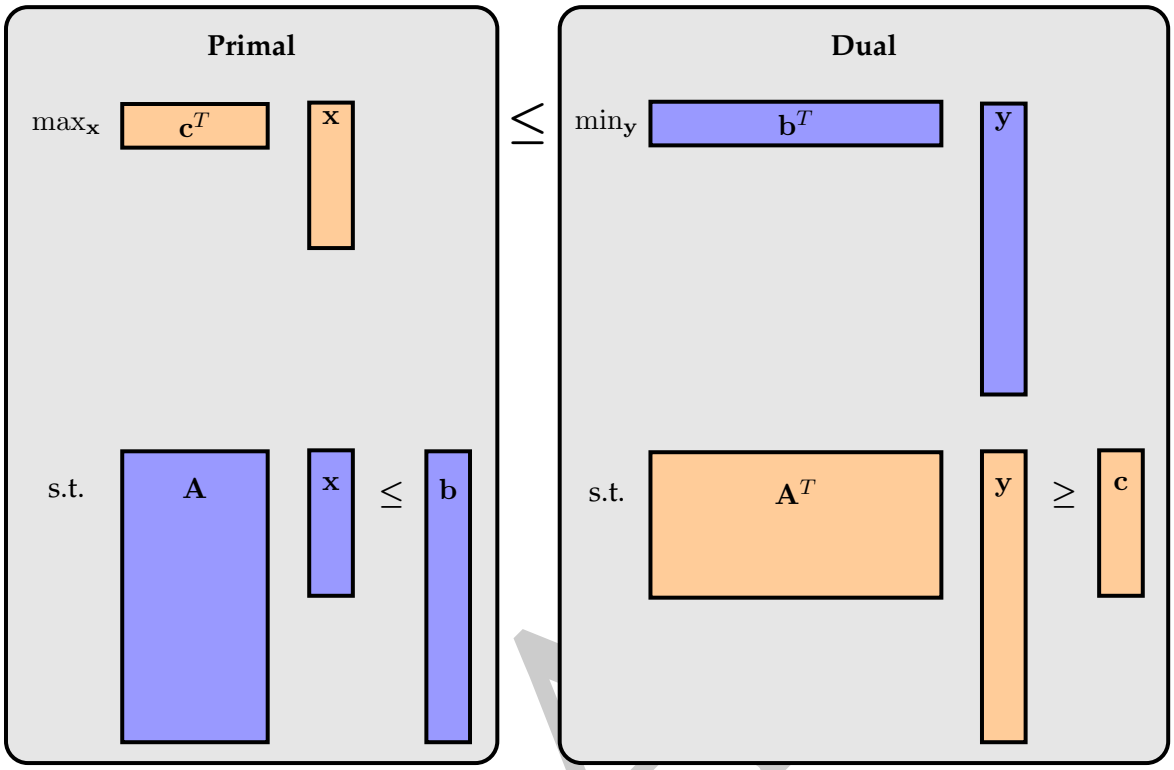


Figure 5.7: Visualization of primal (left) and dual (right) linear programs. In this case, the primal has more constraints than decision variables, while the dual has more decision variables than constraints. The dual minimization always provides an upper bound for the primal maximization. Because linear programs are convex, this duality becomes strong, and the bound is tight, so the dual solution is equal to the primal solution.

so that we arrive at the dual problem in standard form

$$\min_{\mathbf{y} \geq \mathbf{0}} \quad \mathbf{b}^T \mathbf{y}, \tag{5.29a}$$

$$\text{s.t.} \quad \mathbf{A}^T \mathbf{y} \geq \mathbf{c}. \tag{5.29b}$$

The relationship between the primal and dual problems is shown schematically in Fig. 5.7. Although we indicate weak duality with an inequality, for the special convex case of a linear program we have strong duality and equality holds.

Intuition Behind the Dual Linear Program. It is helpful to discuss the intuition behind the dual form of a linear program, as this is the simplest case to understand, and it will provide a basis for more complex primal-dual problems and algorithms.

Given a maximization $\max_{\mathbf{x} \geq \mathbf{0}} \mathbf{c}^T \mathbf{x}$ subject to the constraints $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, the idea is to try and provide the tightest upper bound on $\mathbf{c}^T \mathbf{x}$ by taking linear combinations of the constraint equations $\mathbf{a}_j \mathbf{x} \leq b_j$. The coefficients of the linear combinations of these equations

Copyright © July 15, 2025 Brunton. All Rights Reserved.

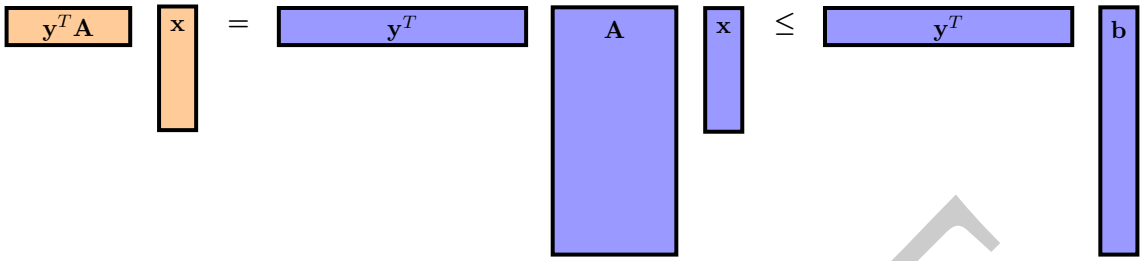


Figure 5.8: The idea behind a dual linear program is to find a linear combination of constraint equations that provide a tight bound for the primal problem. In this case, the combination of constraint equations is given by the vector $\mathbf{y}^T$, so that $\mathbf{y}^T\mathbf{A}$ are the coefficients multiplying $\mathbf{x}$, and $\mathbf{y}^T\mathbf{A}\mathbf{x}$ is bounded from above by $\mathbf{y}^T\mathbf{b}$. We will then minimize $\mathbf{y}^T\mathbf{b}$ subject to $\mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T$, where the original objective function was $\mathbf{c}^T\mathbf{x}$. Notice that this is the transpose of the typical dual problem formulation, although they are equivalent because they are both scalar functions and are therefore equal to their transposes.

will be denoted by $\mathbf{y}$, which we saw above are the dual variables and the Lagrange multipliers enforcing the constraints. We know that any feasible solution will satisfy all of these constraints, so any linear combination will also be satisfied for any feasible point. This linear combination of the constraints yields the following equation:

$$\mathbf{y}^T\mathbf{A}\mathbf{x} \leq \mathbf{y}^T\mathbf{b}. \tag{5.30}$$

Thus, the product $\mathbf{y}^T\mathbf{A}$ provides the coefficients of $\mathbf{x}$. If these coefficients are greater than or equal to $\mathbf{c}^T$, then $\mathbf{y}^T\mathbf{b}$ provides a direct upper bound on $\mathbf{c}^T\mathbf{x}$; note that this relies on $\mathbf{x} \geq \mathbf{0}$, although this condition may be relaxed.

Thus, for any $\mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T$, then $\mathbf{y}^T\mathbf{b}$ will provide an upper bound on the primal. Since we seek the tightest bound, we minimize $\mathbf{y}^T\mathbf{b}$ over $\mathbf{y} \geq \mathbf{0}$ such that $\mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T$. This is shown schematically in Fig. 5.8.

Toy example. The following example illustrates the idea of duality quite simply.

$$\begin{array}{lll} \text{(Primal)} & & \text{(Dual)} \\ \min\limits_{x_1,x_2} \quad 3x_1 + 5x_2 & \Longleftrightarrow & \max\limits_{y} \quad 6y \\ \text{s.t.} \quad 2x_1 + x_2 = 6 & & \text{s.t.} \quad 2y \leq 3 \\ \qquad x_1, x_2 \geq 0 & & \qquad y \leq 5 \end{array}$$

The dual optimum $y^* = \frac{3}{2}$ gives a lower bound $6y^* = 9$, matching the primal optimum $(x_1^*, x_2^*) = (3, 0)$, which gives a cost of 9. Strong duality holds in this case.

Why duals can be cheaper. If $m \ll n$, corresponding to more variables than constraints, then the dual has only m variables. Conversely, if $m \gg n$ the primal is smaller. Interior-point LP solvers exploit this symmetry: they always factor an $m \times m$ or $n \times n$ KKT matrix, choosing whichever is cheaper.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Dual Variables as Sensitivities

Dual variables, i.e. Lagrange multipliers, represent the sensitivity of the optimal value of the objective function to perturbations in the constraint right-hand sides. Specifically, they may be written as partial derivatives of the optimal value with respect to the constraint right-hand sides. If we write the constraints as $\mathbf{h}(\mathbf{x}) = \mathbf{b}$ and $\mathbf{g}(\mathbf{x}) \leq \mathbf{c}$, then, under appropriate conditions, we have the sensitivity interpretation:

$$\lambda_j^* = \frac{\partial f^*}{\partial b_j}, \qquad \mu_j^* = \frac{\partial f^*}{\partial c_j},$$

where f^* is the optimal value as a function of the constraint right-hand sides.

This interpretation is valid when strong duality holds, when the optimal value function $f^*(\mathbf{b}, \mathbf{c})$ is differentiable with respect to $\mathbf{b}$ and $\mathbf{c}$, when the primal solution is unique and varies smoothly with $\mathbf{b}$ or $\mathbf{c}$, and when the KKT conditions hold at the optimal solution. Additional regularity conditions may be required.

Thus, each dual variable λ_j^* quantifies how much the optimal objective value f^* will increase or decrease per unit increase in the right-hand side b_j of an equality constraint. Similarly, if an inequality constraint $g_j(\mathbf{x}) \leq c_j$ is *active* at the solution, then μ_j^* measures the marginal cost or benefit of loosening or tightening that constraint. If the inequality constraint is *inactive*, then $\mu_j^* = 0$ and has no local effect on the optimal value.

This is also known as the *envelope theorem* in parametric optimization, and is widely used in economics and operations research, where dual variables are interpreted as *shadow prices* or *marginal values of resources*.

The sensitivity interpretation may not hold when the problem is nonconvex and the optimal value function is not differentiable, when strong duality fails, when the optimal solution is non-unique or varies discontinuously with $\mathbf{b}$ or $\mathbf{c}$, or when the dual variables do not exist or are not unique.

Conjugate Functions and Fenchel Duality

To better understand duality in constrained optimization, it helps to first examine a more general and abstract form of duality: Fenchel duality, which arises naturally from the concept of convex conjugate functions. The conjugate function $f^*(\mathbf{y})$, called the Fenchel dual[7], of a function $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$f^*(\mathbf{y}) = \sup_{\mathbf{x}} \left(\mathbf{y}^T \mathbf{x} - f(\mathbf{x}) \right). \tag{5.31}$$

This transforms a function $f(\mathbf{x})$ into a function of the linear functional $\mathbf{y}^T\mathbf{x}$ minus $f(\mathbf{x})$, evaluated at the point where the difference is largest. The domain of the function f, denoted $\mathrm{dom}(f)$ is the set $\mathbf{x}$ such that $f(\mathbf{x}) < \infty$; thus, the conjugate function has no awareness of constraints, and is entirely related to the function f. Intuitively, $f^*(\mathbf{y})$ captures the tightest affine lower bound of $f(\mathbf{x})$ with slope $\mathbf{y}$.

The Fenchel conjugate, also known as the convex conjugate, will help bridge the topics of convex analysis and Lagrangian duality [204].

[7] The Fenchel conjugate is a generalization of the classical Legendre transform, which we will discuss in Sec. 5.5. Both transformations relate a function to a dual representation based on its linear supporting hyperplanes, but the Fenchel conjugate is much more general.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Fenchel–Moreau Theorem and the Biconjugate

The biconjugate of f is the conjugate of the conjugate:

$$f^{**}(\mathbf{x}) = \sup_{\mathbf{y}} \left(\mathbf{x}^T \mathbf{y} - f^*(\mathbf{y}) \right). \tag{5.32}$$

The Fenchel–Moreau Theorem states that if f is proper[8], convex, and lower semicontinuous[9], then:

$$f(\mathbf{x}) = f^{**}(\mathbf{x}), \quad \forall \mathbf{x} \in \mathrm{dom}(f). \tag{5.33}$$

Thus, the conjugate gives a full dual representation of a convex function.

Relation to the Lagrange Dual Function

Now, consider the dual function

$$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \right). \tag{5.34}$$

This is precisely a partial infimum over the Lagrangian:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}), \tag{5.35}$$

where $\phi(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is a concave function, being a pointwise infimum of affine functions in $(\boldsymbol{\lambda}, \boldsymbol{\mu})$. If f is convex and the constraints are convex, this construction fits naturally into the Fenchel duality framework.

In the special case where $\mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$ and $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, then:

$$\phi(\boldsymbol{\lambda}) = \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) \right) = -\boldsymbol{\lambda}^T \mathbf{b} + \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{A}\mathbf{x} \right), \tag{5.36}$$

[8] Proper means that $f(\mathbf{x}) < +\infty$ for at least one point $\mathbf{x}$, so that its domain is nonempty, and $f(\mathbf{x}) > -\infty$ for all $\mathbf{x}$, so that it doesn't have a degenerate minimum.

[9] In the context of convex optimization, lower semicontinuous means that the function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ satisfies:

$$\liminf_{\mathbf{x} \to \mathbf{x}_0} f(\mathbf{x}) \geq f(\mathbf{x}_0) \quad \text{for all } \mathbf{x}_0 \in \mathbb{R}^n.$$

This property guarantees that the function does not jump downward, i.e., it doesn't have discontinuities where it suddenly takes a much lower value than nearby points. A continuous function is always lower semicontinuous.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

which can be rewritten as[10]:

$$\phi(\boldsymbol{\lambda}) = -\boldsymbol{\lambda}^T\mathbf{b} - f^*(-\mathbf{A}^T\boldsymbol{\lambda}), \tag{5.37}$$

where f^* is the Fenchel conjugate of f. This gives an explicit form of the dual function in terms of the conjugate of the objective function.

Extension to Convex Inequalities: Fenchel–Rockafellar Duality. Now suppose the problem also includes convex inequality constraints $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$, where each g_j is convex. Then the dual function becomes

$$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \boldsymbol{\mu}^T\mathbf{g}(\mathbf{x}) \right), \quad \text{with } \boldsymbol{\mu} \geq \mathbf{0}. \tag{5.38}$$

We can write this as

$$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) = -\boldsymbol{\lambda}^T\mathbf{b} + \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T\mathbf{A}\mathbf{x} + \boldsymbol{\mu}^T\mathbf{g}(\mathbf{x}) \right). \tag{5.39}$$

Define the function

$$\tilde{f}(\mathbf{x}) := f(\mathbf{x}) + \boldsymbol{\mu}^T\mathbf{g}(\mathbf{x}), \tag{5.40}$$

which remains convex as a nonnegative weighted sum of convex functions. Then the dual function becomes

$$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) = -\boldsymbol{\lambda}^T\mathbf{b} - \tilde{f}^*(-\mathbf{A}^T\boldsymbol{\lambda}), \tag{5.41}$$

where $\tilde{f}^*$ is the Fenchel conjugate of the function $\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\mu}^T\mathbf{g}(\mathbf{x})$. This provides an explicit form of the dual in terms of conjugates, which is useful for analysis and computation.

Fenchel duality provides a foundational theory for convex conjugate pairs, while Lagrangian duality generalizes duality to constrained problems.

Dual of the Dual: Convex Relaxation

In convex problems, the dual of the dual often corresponds to the convex closure of the primal. When the primal problem is convex and satisfies appropriate regularity conditions (e.g. Slater's condition), then strong duality holds and the KKT conditions are both necessary and sufficient for optimality. In this case, we can show:

$$f^{**} = f \tag{5.42}$$

[10] We use the identity that $-\sup_{\mathbf{x}} f(\mathbf{x}) = \inf_{\mathbf{x}} -f(\mathbf{x})$ to rewrite

$$\inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T\mathbf{A}\mathbf{x} \right) = -\sup_{\mathbf{x}} \Big(\mathbf{x}^T \underbrace{(-\mathbf{A}^T\boldsymbol{\lambda})}_{\mathbf{y}} - f(\mathbf{x}) \Big) = -f^*(-\mathbf{A}^T\boldsymbol{\lambda}).$$

This uses the definition of the Fenchel conjugate:

$$f^*(\mathbf{y}) = \sup_{\mathbf{x}} \left(\mathbf{x}^T\mathbf{y} - f(\mathbf{x}) \right)$$

with $\mathbf{y} = -\mathbf{A}^T\boldsymbol{\lambda}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

and hence that the dual of the dual problem recovers the primal problem. This is closely related to the notion of strong duality.

Recall that the primal problem is convex when the objective function f is convex, the inequality constraints g_j are convex, and the equality constraints h_j are affine. Slater's condition states that there exists a strictly feasible point satisfying all inequality constraints strictly and equality constraints exactly. We also require the additional technical properties that f is proper, lower semicontinuous, and not pathological.

If f is not convex, then $f^{**} \neq f$. Instead, f^{**} is the *convex envelope* of f:

$$f^{**}(\mathbf{x}) \leq f(\mathbf{x}) \quad \forall\, \mathbf{x}.$$

The dual of the dual, f^{**}, is the tightest convex function below f. In this case, the dual of the dual is the convex relaxation of the primal problem.

Similarly, the dual of the dual is not equal to the primal if Slater's condition fails. If f is convex but no strictly feasible point exists, the strong duality may fail and the dual of the dual may not match the primal. There can be a *duality gap*. Finally, if f is not proper or lower semicontinuous, duality may be undefined or degenerate.

Example: Convex Envelope of a Nonconvex Function

Consider the nonconvex function:

$$f(x) = \max(-x^2,\ -|x|),$$

which is formed by taking the pointwise maximum of two concave functions. The resulting function is nonconvex, as it dips down too sharply at $x = 0$ for any single supporting hyperplane to touch from below.

The convex envelope of a function f is the largest convex function that lies below f everywhere. In this case, the convex envelope of f is:

$$f^{**}(x) = -|x|,$$

which is the tightest convex function lying entirely beneath $f(x)$.

This is precisely what arises when we compute the dual of the dual. If f is not convex, its conjugate f^* is still convex, and the biconjugate f^{**} satisfies:

$$f^{**}(x) \leq f(x),$$

with equality if and only if f is convex and lower semicontinuous. Here, since f is not convex, we have strict inequality near $x = 0$.

Figure 5.9 shows the nonconvex function $f(x)$ and its convex envelope $f^{**}(x) = -|x|$.

Geometric Interpretation of Duality

A key geometric insight behind Lagrange duality is the relationship between *supporting hyperplanes* and the construction of the dual function. In convex optimization, duality is not just an algebraic formula, but it is also a geometric process of constructing the best possible linear lower bound on a convex function, subject to constraints.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

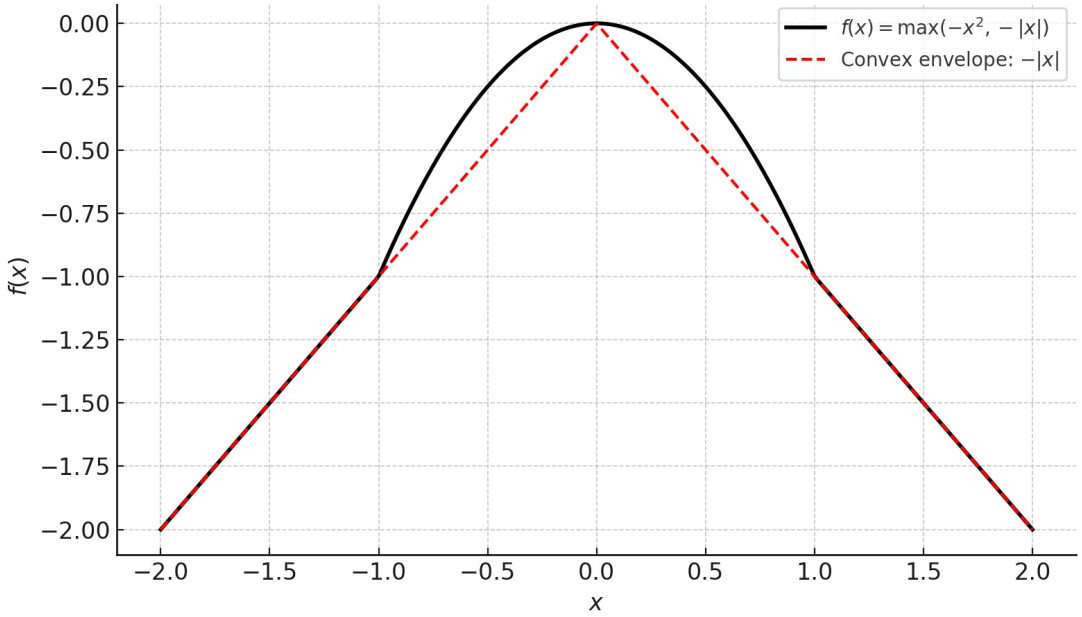


Figure 5.9: Illustration that the dual of the dual of a nonconvex function is the convex envelop of the function. In this example, the nonconvex function $f(x) = \max(-x^2, -|x|)$ relaxes to the convex envelop $-|x|$.

For now we will consider the case of equality constraints $\mathbf{h}(\mathbf{x}) = \mathbf{0}$, to simplify the discussion. In the primal problem, we search for the lowest level set of f that intersects the feasible region defined by $\mathbf{h}(\mathbf{x}) = \mathbf{0}$. In the dual, we search over $\boldsymbol{\lambda}$ for the best affine lower bound (i.e., supporting plane) on $f(\mathbf{x})$ that respects the constraint. The optimal $\boldsymbol{\lambda}^*$ corresponds to the slope of this best supporting hyperplane.

Duality and Supporting Hyperplanes

Let $\mathcal{C} \subset \mathbb{R}^n$ be a convex set, and let $\mathbf{x}_0 \in \partial\mathcal{C}$ be a boundary point. A *supporting hyperplane* at $\mathbf{x}_0$ is a hyperplane that touches $\mathcal{C}$ at $\mathbf{x}_0$ and lies entirely on one side of $\mathcal{C}$.

Mathematically, this means that there exists a vector $\mathbf{a} \in \mathbb{R}^n$ such that:

$$\mathbf{a}^T(\mathbf{x} - \mathbf{x}_0) \geq 0 \quad \text{for all } \mathbf{x} \in \mathcal{C}.$$

Recall that the dual function is

$$\phi(\boldsymbol{\lambda}) = \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) \right),$$

which is concave in $\boldsymbol{\lambda}$ and gives a family of lower bounds on the primal objective.

In the case of convex equality constraints $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ (i.e., affine constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$), then each choice of λ defines a linear perturbation of the objective function

$$f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{A}\mathbf{x}.$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

This can be interpreted as tilting the original function f in the direction determined by $\boldsymbol{\lambda}^T\mathbf{A}$. Taking its infimum provides a lower bound on the primal optimal value. The dual function is then given by:

$$\phi(\boldsymbol{\lambda}) = -\boldsymbol{\lambda}^T\mathbf{b} + \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \boldsymbol{\lambda}^T\mathbf{A}\mathbf{x} \right).$$

Geometrically, the pair $(\boldsymbol{\lambda}, \phi(\boldsymbol{\lambda}))$ defines a supporting hyperplane to the epigraph of f. Specifically, the inequality

$$f(\mathbf{x}) \geq -\boldsymbol{\lambda}^T\mathbf{A}\mathbf{x} + \left(\phi(\boldsymbol{\lambda}) + \boldsymbol{\lambda}^T\mathbf{b} \right)$$

defines an affine lower bound on f that is tight at the minimizing $\mathbf{x}$ for that value of $\boldsymbol{\lambda}$, assuming strong duality. Thus, the dual variables $\boldsymbol{\lambda}$ parameterize a family of affine underestimators of f, one for each direction of perturbation, forming the set of supporting hyperplanes to $\mathrm{epi}(f)$.

The *optimal dual variable* $\boldsymbol{\lambda}^*$ corresponds to the slope of the tightest such supporting hyperplane that lies below the graph of f, touches the graph at an optimal point $\mathbf{x}^*$, and respects the constraint $\mathbf{h}(\mathbf{x}) = \mathbf{0}$.

We will explore how to generalize this geometric picture to handle inequality constraints in the homework.

Example: Scalar Convex Problem

To build geometric intuition for Lagrangian duality, we study a scalar convex problem:

$$\begin{aligned} \min_{x} \quad & f(x) = (x-2)^2 + 1 \\ \text{s. t.} \quad & h(x) = x - 1 = 0. \end{aligned}$$

This problem is very simple, since there is a single feasible point, $x = 1$, and hence this must also be the optimal value of x. However, it is still helpful as an example.

We form the Lagrangian

$$\mathcal{L}(x, \lambda) = f(x) + \lambda h(x) = (x-2)^2 + 1 + \lambda(x-1), \tag{5.43}$$

and define the dual function as the pointwise infimum over x:

$$\phi(\lambda) = \inf_x \mathcal{L}(x, \lambda) = \inf_x \left((x-2)^2 + 1 + \lambda(x-1) \right). \tag{5.44}$$

The function $\phi(\lambda)$ is concave and gives a lower bound on the optimal value of the constrained problem for all λ. We compute the dual function explicitly by minimizing over x:

$$\phi(\lambda) = -\frac{1}{4}\lambda^2 + \lambda + 1, \tag{5.45}$$

which is concave in λ, as expected. The dual achieves its maximum at $\lambda^* = 2$, corresponding to the primal optimal point $x^* = 1$ and value $f(x^*) = 2$. This value of λ^* defines the best affine lower bound that respects the constraint.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

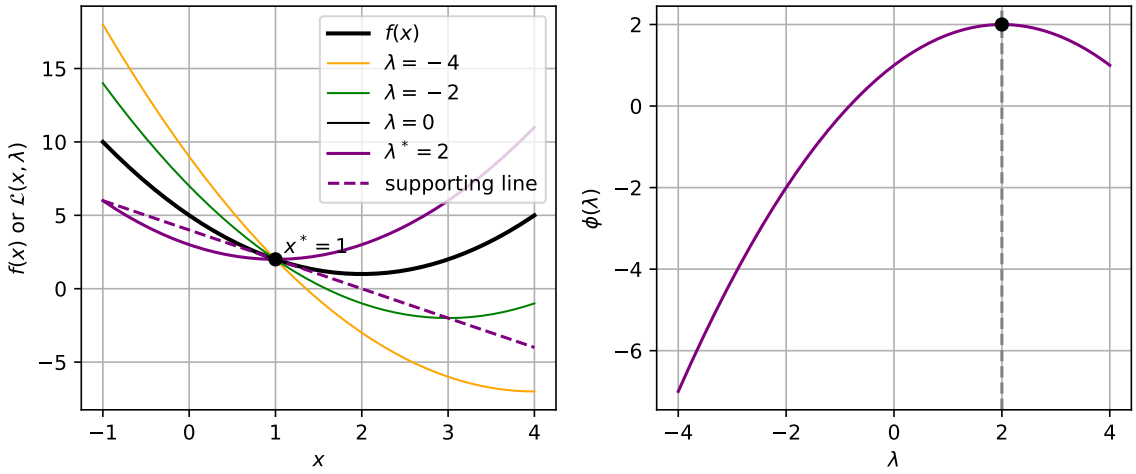


Figure 5.10: (left) Illustration of original function $f(x)$ along with the *tilted* Lagrangian $\mathcal{L}(x, \lambda)$ for several values of λ, including $\lambda^* = 2$. The optimal dual variable $\lambda^* = 2$ is the negative of the slope of the tightest supporting hyper-plane at the optimal/feasible point $x^* = 1$. (right) Illustration of the dual function $\phi(\lambda)$.

For each λ, the function $x \mapsto f(x) + \lambda(x - 1)$ is a linear perturbation of f and can be interpreted as a *tilted function*. Its graph lies below $f(x)$ and defines an affine lower bound, namely a supporting line, to $\text{epi}(f)$. These affine functions act as *supporting hyperplanes* to the graph of $f(x)$.

The optimal value λ^* corresponds to the negative of the slope[11] of the tightest such supporting line that touches the graph of f at the feasible point $x^* = 1$. The supporting line to the epigraph of f at $x^* = 1$ is

$$t = f(1) + f'(1)(x - 1) \tag{5.46}$$

which has slope $f'(x^*) = f'(1) = -2$.

Figure 5.10 plots the graph and epigraph of $f(x)$, along with several tilted functions $f(x) + \lambda(x - 1)$. We also show the supporting line corresponding to $\lambda^* = -2$ and the dual function $\phi(\lambda)$. These plots illustrate how each λ defines an affine underestimator of f and how the dual captures the best such affine bound.

Note that the dual function $\phi(\lambda)$ is less than or equal to the primal function $f(x)$ for all λ and feasible points x. Note that there is only a single feasible point x and $f(x) = 2$ at this point, and the dual function $\phi(\lambda)$ has a maximum value of 2.

[11] The KKT optimal condition states that $\partial\mathcal{L}/\partial x = f'(x^*) + \lambda^* = 0$ which implies that $\lambda^* = -f'(x^*)$. We may further understand this geometrically. We are adding a linear *tilt* term $\lambda(x - 1)$ to $f(x)$. For the total Lagrangian function to be minimized by the feasible point $x = 1$, the slope of the tilted Lagrangian function at $x = 1$ must be zero. So again, taking the partial of $\mathcal{L}$ with respect to x and evaluating at the feasible point $x = 1$, we find that $\lambda^* = -f'(1)$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Duality for Constrained Least Squares

Least squares is another canonical optimization problem that we have analyzed extensively in Chapter 4. Therefore, it is natural to revisit constrained least-squares from the perspective of duality.

Consider the constrained least squares problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & \tfrac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2 \\ \text{s.t.} \quad & \mathbf{C}\mathbf{x}=\mathbf{d}. \end{aligned}$$

We may formulate the dual problem by first forming the Lagrangian

$$\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) = \tfrac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 + \boldsymbol{\lambda}^T(\mathbf{C}\mathbf{x}-\mathbf{d}).$$

To formulate the dual function, we minimize over $\mathbf{x}$, which in this case amounts to finding a stationary point $\mathbf{x}$ that satisfies

$$(\mathbf{A}^T\mathbf{A})\mathbf{x} + \mathbf{C}^T\boldsymbol{\lambda} = \mathbf{A}^T\mathbf{b}$$

After some algebraic steps, left as an exercise, we may write the dual function as

$$\phi(\boldsymbol{\lambda}) = -\tfrac{1}{2}\|P_\perp(\mathbf{b}-\mathbf{A}^T\mathbf{C}^T\boldsymbol{\lambda})\|_2^2 - \boldsymbol{\lambda}^T\mathbf{d} \tag{5.47}$$

where $P_\perp$ projects onto $\text{null}(\mathbf{A}^T\mathbf{A})$. If $\mathbf{C}$ has far fewer rows than $\mathbf{A}$, solving the dual for $\boldsymbol{\lambda}$ (m unknowns) and back-substituting for $\mathbf{x}$ is faster than solving a $(n+m)$-by-$(n+m)$ KKT system.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

5.4 Interior Point Methods

Interior point methods (IPMs) are a family of iterative algorithms to solve constrained optimization problems by traversing the *interior* of the feasible region rather than exploring the boundary [48, 268]. We've seen the simplex method to solve linear programs in Chapter 3, which moves from boundary vertex to vertex until an optimal solution is found. IPMs provide an alternative approach that scale to very high-dimensional problems with complex boundaries and generalize to more sophisticated convex and non-convex problems.

The interior point method finds solutions that satisfy the KKT conditions, and hence are optimal. For convex problems this optimality is global, whereas for nonconvex problems the optimality may be local. There are many types of interior point methods. We present a primal-dual approach that utilizes a logarithmic barrier function and the dual variable $\boldsymbol{\mu}$ to satisfy the KKT conditions at each iteration. Convergence and scaling results exist for IPMs on convex problems, although they are still useful for non-convex problems.

The basic principle of IPMs is to replace the original constrained optimization problem with a series of unconstrained subproblems. A common example is the barrier approach, which enforces strict interior feasibility by introducing a barrier, or penalty, that becomes infinite at the boundary of the feasible region. This is the approach we take here.

It is often difficult to solve the KKT conditions directly, so we instead solve a family of simpler related problems that converge to a solution of the KKT conditions. We explain this approach for a single constraint $g(\mathbf{x}) \leq 0$, though it generalizes to more constraints.

The first three KKT conditions stay the same, and the last condition changes from $\mu g(\mathbf{x}) = 0$ to $\mu g(\mathbf{x}) = -t$. The solution of these equations is $\mathbf{x}^*(t)$. Clearly as t goes to 0, then $\mathbf{x}^*(t)$ converges to the optimal $\mathbf{x}^*$ satisfying the KKT conditions. It is simpler to solve these perturbed KKT conditions, and we may then slowly vary t to 0, using the solution $\mathbf{x}^*(t_{k-1})$ as the initial condition to solve for $\mathbf{x}^*(t_k)$. This is known as continuation, or homotopy, and is a much more broadly useful technique beyond optimization.

When $t \neq 0$, it is possible to solve for μ directly from $\mathbf{x}$ via

$$\mu = \frac{-t}{g(\mathbf{x})}. \tag{5.48}$$

We then substitute this back into the gradient equation and solve for $\mathbf{x}$:

$$\nabla f(\mathbf{x}) + \frac{-t}{g(\mathbf{x})} \nabla g(\mathbf{x}) = 0. \tag{5.49}$$

A little algebra shows that this is equivalent to the following:

$$\nabla \left(f(\mathbf{x}) - t \log(-g(\mathbf{x})) \right) = 0 \tag{5.50}$$

so we are looking for the solution to the following minimization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) - t \log(-g(\mathbf{x})). \tag{5.51}$$

A solution to this problem will satisfy the first KKT condition and will also ensure that $g(\mathbf{x}) \leq 0$, since $-g(\mathbf{x})$ must be positive for $\log(-g(\mathbf{x}))$ to be finite. Similarly, from Eq. (5.48),

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

$\mu \geq 0$. Thus, all modified KKT conditions will be satisfied. The function $-t\log(-g(\mathbf{x}))$ is known as a *barrier function*, since it is a smooth convex function, defined in the interior of the feasible set, that becomes infinite outside of the feasible set, hence providing a *barrier*. Many interior-point methods involve the use of barrier functions.

Since this is an unconstrained optimization problem, we may use simpler techniques, such as gradient descent or Newton's method. Typically, Newton's method is a good choice because we will converge very rapidly with a good choice of initial condition.

When t is small, the penalty function $t\log(-g(\mathbf{x}))$ will act nearly like the discontinuous 0-∞ penalty function, discussed earlier in Section 1.4. In this case, the objective function landscape will be quite shallow, and Newton's method will converge slowly. However, if we use the optimal solution $\mathbf{x}^*(t_{k+1})$ from the last value of t_{k+1} as the initial condition for Newton's method at t_k, then this will help the algorithm converge faster.

If we start with a large t, the objective function appears to have no $f(\mathbf{x})$ term, and instead we minimize $-\log(-g(\mathbf{x}))$, which is equivalent to minimizing $g(\mathbf{x})$. We can solve this using Newton's method to find $\mathbf{x}^*(\infty)$, called the *analytic center* of the feasible set, which will be an initialization for our first value of t. The sequence of intermediate solutions $\mathbf{x}^*(t_k)$ is called the *central path*, and it is always in the interior of the feasible set.

Interior point methods have a rich historical development, being developed several times independently. There are several key branches of IPMs, although the most important ones today are primal-dual methods. IPMs were originally discovered in the 1960s by Dikin [100] and by Fiacco and McCormick in the context of penalty-barrier methods [127, 128]. However, these approaches were considered impractical for large-scale problems until the introduction of Karmarkar's primal-dual IPM algorithm for linear programming [197] in 1984, which was competitive with Dantzig's simplex method [87].

Extensions to More Constraints. It is simple to incorporate additional constraints, $g_j(\mathbf{x}) \geq 0$ for $j = 0, \cdots, m$. In this case, we design a composite barrier function that is the sum of the various individual barrier functions corresponding to each individual constraint:

$$\Phi(x) = -\sum_{i=1}^{m} \log\left(g_i(x)\right). \tag{5.52}$$

Industrial Success of IPMs for Linear Programming. The success of interior point methods in linear programming is evident in their widespread industrial adoption. More generally IPMs offer polynomial-time complexity and robust theoretical guarantees for convex optimization. For linear programming, the convergence is provably linear in the dimension of the problem. In practice, the iteration count is often even independent of dimension because of the quadratic convergence of Newton iterations. However, each of these iterations can be costly. In practice, modern implementations for large-scale LP or convex problems often outperform the simplex method for certain classes of large or sparse problems. However, for small or medium-scale LPs, or those with special structure, an optimized simplex method can still be competitive and may outperform IPMs in practice.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Barrier Method for Linear Programming

It is possible to solve linear programming using the logarithmic barrier method, as described above. We begin with a standard linear program

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} && \text{(5.53a)} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} && \text{(5.53b)} \\ & \mathbf{x} \geq 0. && \text{(5.53c)} \end{aligned}$$

The barrier method addresses the inequality constraints by introducing logarithmic barrier functions to penalize solutions that approach the constraint boundaries. Specifically, the barrier formulation replaces the original constrained problem with a sequence of unconstrained subproblems:

$$\min_{\mathbf{x}} \; f_t(\mathbf{x}) = \min_{\mathbf{x}} \; \left(\mathbf{c}^T\mathbf{x} - t \left(\sum_{j=1}^{m} \log\left(b_j - \mathbf{a}_j^T\mathbf{x}\right) + \sum_{j=1}^{n} \log\left(x_j\right) \right) \right), \tag{5.54}$$

where $t \geq 0$ is the parameter that we decrease to zero, and $\mathbf{a}_j^T\mathbf{x} = b_j$ denotes the j^{th} row of the matrix system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. As $t \to 0$, the solution to this unconstrained problem converges to the optimal solution of the original problem.

With the barrier scaling, the gradient and Hessian become:

$$\nabla f_t(\mathbf{x}) = \mathbf{c} + t \left(\sum_{i=1}^{m} \frac{\mathbf{a}_i}{b_i - \mathbf{a}_i^T\mathbf{x}} - \sum_{j=1}^{n} \frac{\mathbf{e}_j}{x_j} \right), \tag{5.55a}$$

$$\nabla^2 f_t(\mathbf{x}) = t \left(\sum_{i=1}^{m} \frac{\mathbf{a}_i\mathbf{a}_i^T}{(b_i - \mathbf{a}_i^T\mathbf{x})^2} + \sum_{j=1}^{n} \frac{\mathbf{e}_j\mathbf{e}_j^T}{x_j^2} \right). \tag{5.55b}$$

Newton's method yields the search direction $\Delta\mathbf{x}$:

$$\nabla^2 f_t(\mathbf{x})\Delta\mathbf{x} = -\nabla f_t(\mathbf{x}).$$

Typically we would select the step size α via line search to ensure feasibility (i.e., $\mathbf{A}\mathbf{x} < \mathbf{b}$ remains satisfied), then update:

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha\Delta\mathbf{x}.$$

We then update the barrier parameter t, for example $t_{k+1} = 0.9t_k$. We then continue until the duality gap and constraint residuals fall below pre-chosen tolerances.

Linear Programming Example in Python

Modern optimization packages implement efficient and robust primal-dual interior-point methods. For example, using SciPy's built-in solver, the linear program from Section 3.1 can be solved as follows:

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
import numpy as np
from scipy.optimize import linprog

# Define constraints
A = np.array([ [-1, 0],   # -x <= 0  => x >= 0
    [0, -1],              # -y <= 0  => y >= 0
    [0, 1],               # y <= 30
    [1, 1] ])             # x + y <= 50
b = np.array([0, 0, 30, 50])
c = np.array([-100, -400]) # Max c^T x = 100x + 400y => min -c^T x

# Solve using a modern interior-point method (HiGHS-IPM)
res = linprog(c, A_ub=A, b_ub=b, method='highs-ipm')
```

Modern implementations exploit sparsity, primal-dual relationships, and predictor-corrector steps to efficiently solve large-scale linear programming problems.

It is simple to implement a barrier-method IPM ourselves. Initializing the same matrix $\mathbf{A}$ and vectors $\mathbf{c}$ and $\mathbf{b}$ as above, the code is straightforward. First, we must define the logarithmic barrier function $\Phi(\mathbf{x})$, its gradient $\nabla\Phi(\mathbf{x})$, and its Hessian $\nabla^2\Phi(\mathbf{x})$:

```
def phi(x):
    return -np.sum(np.log(b - A @ x))

def grad_phi(x):
    return A.T @ (1 / (b - A @ x))

def hess_phi(x):
    vals = 1 / (b - A @ x)
    return A.T @ np.diag(vals**2) @ A
```

We may then implement the barrier-method IPM as in Code 5.1.

Code 5.1: Interior point method for a linear program. Here we multiply the barrier function by $1/t$ instead of t and let $t \to \infty$ instead of $t \to 0$, to be consistent with the literature.

```
def barrier_method_lp(A, b, c, x0, mu=10, tol=1e-8, max_iter=50):
    m, n = A.shape
    x = x0.copy()
    t = 1.0
    path = [x.copy()]

    for _ in range(max_iter):
        for _ in range(20):  # Newton steps
            grad = t * c + grad_phi(x)
            hess = hess_phi(x)
            dx = -np.linalg.solve(hess, grad)

            s = 1.0
            while np.any(A @ (x + s * dx) >= b):
                s *= 0.5
                if s < 1e-16:
                    break
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

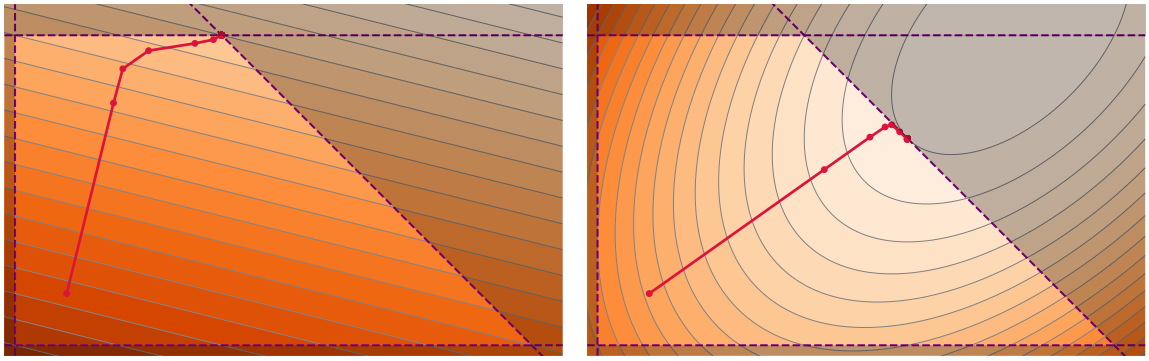


Figure 5.11: Central path of the interior point method for a convex linear program (left) and quadratic program (right). In both cases, the trajectory reaches the optimum value in very few iterations. On the right, the trajectory approaches the boundary quickly, and then slowly hugs it until the optimum. This illustrates the shallowness of the log-barrier objective landscape for some problems, making Newton's method slow to converge.

```
            x += s * dx
            path.append(x.copy())

            if np.linalg.norm(grad) < tol:
                break

        if n / t < tol:
            break
        t *= mu
    return np.array(path)
```

It is easy to call this code with the following two lines:

```
x0 = np.array([5.0, 5.0])  # Feasible initial condition
path = barrier_method_lp(A, b, c, x0)
```

The output of this code is shown in Fig. 5.11 (left). Modifying this code to solve a quadratic program with linear constraints is left as an exercise for the reader.

Barrier Method for Constrained Least Squares

Barrier-method IPMs may also be used for constrained least-squares problems and other quadratic programs with constraints such as non-negativity, upper/lower bounds, or linear constraints. The constrained-least squares problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & \tfrac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 \\ \text{s.t.} \quad & \mathbf{C}\mathbf{x} \leq \mathbf{d} \end{aligned}$$

may be solved efficiently with IPMs [48]. In this case, the Hessian is $\mathbf{A}^T\mathbf{A} + t\nabla^2\Phi(\mathbf{x})$, so each iteration involves solving a Newton system that includes both the quadratic term $\mathbf{A}^T\mathbf{A}$ and the barrier term's diagonal contributions from $\Phi(\mathbf{x})$. This is particularly powerful in high-dimensional settings when combined with sparse linear algebra techniques.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

5.5 Calculus of Variations and the Euler-Lagrange Equations

The calculus of variations is a field of mathematical analysis that deals with maximizing or minimizing functionals, which are mappings from a space of functions to the real numbers [139, 122]. For example, when optimizing the shape of a wing to maximize lift and minimize drag, the shape may be represented as a function to be optimized, given some constraints (structural, size, weight, etc.). In contrast to conventional optimization, where the unknowns are finite-dimensional vectors, variational problems involve optimizing over infinite-dimensional spaces of functions. This broader setting underlies many foundational ideas in modern optimization, including optimal control, PDE-constrained optimization, and machine learning.

Historically, the calculus of variations traces back to Johann Bernoulli's *brachistochrone problem* in the late 17th century. Euler and Lagrange developed many core ideas of the field in the 18th century, including the Euler-Lagrange equation and fundamental concepts of functional derivatives. Over time, the field evolved to accommodate more general functional spaces, such as Sobolev spaces, Banach spaces, etc., leading to powerful theoretical results such as the direct methods in the calculus of variations [255].

Today, variational calculus is vital in many areas:

- Physics and Mechanics: Action principles [1], based on variational principles, are central in classical mechanics, general relativity, and quantum field theory.
- PDE-Constrained Optimization (Chapter 8): Many engineering problems involve optimizing a geometric shape given the constraints of some physics, typically expressed as a partial differential equation. For example, designing the shape of cooling fins in a heat exchanger [214], or the shape of a wing on a modern commercial aircraft [180] are both constrained variational optimization problems.
- Optimal Control (Chapter 9): Most optimal control problems can be formulated as solving for a function to minimize or maximize a given objective function, constrained by the dynamics of the problem. The Pontryagin maximum principle, the Kalman filter, and dynamic programming are three concrete examples that have roots in variational arguments [42, 327, 27].
- Machine Learning (Chapter 10): Many problems in modern machine learning may be thought of in through a variational lens [380, 181, 201]. For example, deep neural networks often approximate functionals where the input and output functions may be discretized. Deep operator networks [232] and neural operator [223, 222] approaches formalize this idea.

A functional $\mathcal{J}$ takes a function $y(x)$ as input and returns a real number:

$$\mathcal{J}[y] = \int_a^b F\left(y(x), y'(x), x\right)\, dx, \tag{5.56}$$

where F is typically a scalar-valued function that depends on $y(x)$, its derivative $y'(x)$, and the independent variable x. A *variational problem* is to find a function $y^*(x)$ that extremizes (minimizes or maximizes) $\mathcal{J}[y]$ subject to boundary conditions or other constraints.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

In the simplest unconstrained setting, the condition for $y(x)$ to optimize $\mathcal{J}$ is given by the Euler-Lagrange equation:

$$\frac{\partial F}{\partial y} - \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial F}{\partial y'}\right) = 0. \tag{5.57}$$

This is the continuous analog of setting the gradient to zero in finite-dimensional optimization [139].

Mechanics Interpretation. In the context of classical mechanics and physics, the Euler-Lagrange equations represent the principle of least action, which is perhaps the most fundamental and unifying principle in all of physics. The Euler-Lagrange equations were originally derived to describe the motion of mechanical systems in arbitrary coordinate systems, although they provide a much more general set of optimality conditions for variational problems where we want to optimize a functional over a function. To be consistent with this literature in physics and engineering, we will adopt physical notation here.

In the context of a physical system, we replace F with the Lagrangian $L(\mathbf{q}(t), \dot{\mathbf{q}}(t), t)$, which is written in terms of generalized coordinates $\mathbf{q}(t) \in \mathbb{R}^n$ that describe the system. For example, when analyzing a simple pendulum, the generalized coordinates may be $\mathbf{q} = \begin{bmatrix}\theta & \dot{\theta}\end{bmatrix}^T$; see Fig. 5.12 for a schematic. Now the independent variable is time, t, and $\dot{\mathbf{q}}(t) = \frac{d}{dt}\mathbf{q}(t)$. We now call the functional $\mathcal{J}$ the action integral and denote it by $\mathcal{A}(\mathbf{q})$:

$$\mathcal{A}[\mathbf{q}] = \int_{t_0}^{t_1} L(\mathbf{q}(t), \dot{\mathbf{q}}(t), t)\, dt. \tag{5.58}$$

The Lagrangian is often formulated as the difference between kinetic energy $T(\mathbf{q}, \dot{\mathbf{q}}, t)$ and potential energy $V(\mathbf{q}, \dot{\mathbf{q}}, t)$: $L = T - V$. Energy transfer between kinetic and potential energy is called the *action*. The principle of least action states that among all possible trajectories a system might follow between two fixed endpoints in configuration space, the physical path taken is the one that makes the action integral stationary, meaning it extremizes the functional $\mathcal{A}$; we will formalize this notion of extremizing a functional in a moment. More informally, the physical path $\mathbf{q}(t)$ will typically minimize, or at least be a critical point, of the action integral in Eq. (5.58).

The Euler–Lagrange equations may now be written as

$$\frac{\partial L}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_j}\right) = 0 \tag{5.59}$$

where q_j is the j^{th} component of $\mathbf{q}$. The Euler-Lagrange equations are the *necessary* conditions for a solution of the variational problem, much as $\nabla_{\mathbf{x}} f(\mathbf{x}) = \mathbf{0}$ is a necessary condition for a minimum of an unconstrained objective over a discrete decision variable $\mathbf{x}$.

In the next subsections we will derive the Euler-Lagrange equations, analyze several classical examples, and explore duality in the context of infinite-dimensional optimization problems. Specifically, we will introduce the Legendre transformation and establish that the Hamiltonian is the dual form of the Lagrangian. Finally, we will discuss how constraints fit into this framework.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

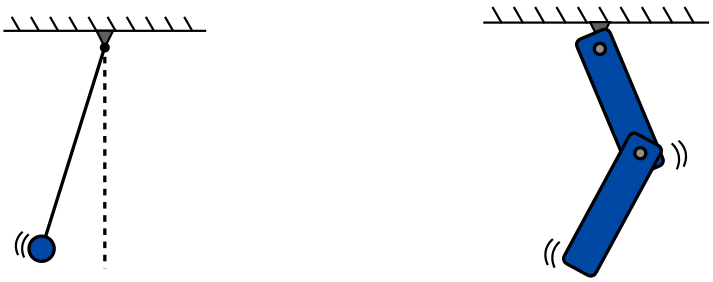


Figure 5.12: Two example mechanical pendula systems where the length of the pendulum arms are constant, so their motion is restricted to a constraint circle. The equations for the pendulum (left) are much easier to derive and solve in the intrinsic coordinates $\mathbf{q} = \begin{bmatrix}\theta & \dot{\theta}\end{bmatrix}^T$, which naturally handle the constraint, rather than in (x, y) coordinates. These benefits only increase for more complex systems, such as the double pendulum (right).

Deriving the Euler-Lagrange Equations

We will now derive the Euler-Lagrange equations as a necessary condition for optimality of variational optimization problems. The idea is that if a function $\mathbf{q}(t)$ extremizes the functional $\mathcal{A}$, then $\mathcal{A}$ must be insensitive to infinitesimal perturbations to $\mathbf{q}$, similar to how an optimal $\mathbf{x}^*$ of an objective must satisfy $\nabla f(\mathbf{x}^*) = \mathbf{0}$, meaning that infinitesimal variations in $\mathbf{x}^*$ won't change the objective. The derivative of $\mathcal{A}$ with respect to infinitesimal variations in $\mathbf{q}(t)$ is called the *first variation* $\delta\mathcal{A}$. We will vary the trajectory $\mathbf{q}(t)$ in Eq. (5.58) and require that the first variation of the action vanishes. This will result in the Euler-Lagrange equations as a necessary condition for optimality. The principle of least action says a function $\mathbf{q}(t)$ that extremizes the functional $\mathcal{A}[\mathbf{q}]$ must satisfy

$$\delta\mathcal{A} = 0, \tag{5.60}$$

where $\delta\mathcal{A}$ is the first variation of $\mathcal{A}$ with respect to variations of $\mathbf{q}(t)$ that fix the endpoints $\mathbf{q}(t_0)$ and $\mathbf{q}(t_1)$. To carry out this variation explicitly, we consider a family of curves

$$\mathbf{q}_\varepsilon(t) = \mathbf{q}(t) + \varepsilon\boldsymbol{\eta}(t), \tag{5.61}$$

where $\boldsymbol{\eta}(t)$ is a smooth variation with $\boldsymbol{\eta}(t_0) = \boldsymbol{\eta}(t_1) = \mathbf{0}$ to ensure fixed endpoints. The functional $\mathcal{A}$ may then be evaluated at $\mathbf{q}_\varepsilon$ and we may take the derivative with respect to ε

$$\delta\mathcal{A} = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\,\mathcal{A}[\mathbf{q}_\varepsilon]|_{\varepsilon=0} = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\left[\int_{t_0}^{t_1} L(\mathbf{q}_\varepsilon, \dot{\mathbf{q}}_\varepsilon, t)\,dt\right]_{\varepsilon=0}. \tag{5.62}$$

We may now compute $\frac{d}{d\varepsilon}\mathcal{A}[\mathbf{q}_\varepsilon]$ explicitly:

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\int_{t_0}^{t_1} L(\mathbf{q}+\varepsilon\boldsymbol{\eta}, \dot{\mathbf{q}}+\varepsilon\dot{\boldsymbol{\eta}}, t)\,dt = \int_{t_0}^{t_1}\left[\frac{\partial L}{\partial \mathbf{q}}\boldsymbol{\eta} + \frac{\partial L}{\partial \dot{\mathbf{q}}}\dot{\boldsymbol{\eta}}\right]\,dt \tag{5.63}$$

where $\frac{\partial L}{\partial \mathbf{q}}$ and $\frac{\partial L}{\partial \dot{\mathbf{q}}}$ are evaluated at $(\mathbf{q}+\varepsilon\boldsymbol{\eta}, \dot{\mathbf{q}}+\varepsilon\dot{\boldsymbol{\eta}}, t)$. Note that there is no term from the time dependence because $\frac{dt}{d\varepsilon} = 0$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

We now evaluate this derivative at $\varepsilon = 0$, which yields

$$\delta\mathcal{A} = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\,\mathcal{A}[\mathbf{q}_\varepsilon]|_{\varepsilon=0} = \int_{t_0}^{t_1}\left[\frac{\partial L}{\partial \mathbf{q}}(\mathbf{q},\dot{\mathbf{q}},t)\boldsymbol{\eta}(t) + \frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q},\dot{\mathbf{q}},t)\dot{\boldsymbol{\eta}}(t)\right]dt \tag{5.64}$$

This may be written in coordinate form as

$$\delta\mathcal{A} = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\,\mathcal{A}[\mathbf{q}_\varepsilon]|_{\varepsilon=0} = \int_{t_0}^{t_1}\sum_{j=1}^{n}\left[\frac{\partial L}{\partial q_j}(\mathbf{q},\dot{\mathbf{q}},t)\eta_j(t) + \frac{\partial L}{\partial \dot{q}_j}(\mathbf{q},\dot{\mathbf{q}},t)\dot{\eta}_j(t)\right]dt \tag{5.65}$$

Finally, we apply integration by parts to the second set of terms including $\dot{\eta}_j$:

$$\int_{t_0}^{t_1}\frac{\partial L}{\partial \dot{q}_j}\dot{\eta}_j\,dt = \left[\frac{\partial L}{\partial \dot{q}_j}\eta_j\right]_{t_0}^{t_1} - \int_{t_0}^{t_1}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_j}\right)\eta_j\,dt. \tag{5.66}$$

The boundary term vanishes because $\boldsymbol{\eta}(t_0) = \boldsymbol{\eta}(t_1) = \mathbf{0}$. Substituting this back into the first variation of the action from Eq. (5.65) we arrive at

$$\delta\mathcal{A} = \int_{t_0}^{t_1}\left[\frac{\partial L}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_j}\right]\eta_j\,dt. \tag{5.67}$$

The summation over j is implied. This integral must vanish for all variations $\boldsymbol{\eta}(t)$, so the integrand must be zero pointwise, resulting in the Euler–Lagrange equations:

$$\frac{\partial L}{\partial q_j} - \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_j}\right) = 0. \tag{5.68}$$

They are a necessary condition for $\mathbf{q}$ to extremize the functional $\mathcal{A}[\mathbf{q}]$, given fixed endpoints. Modifying this for non-fixed endpoints will be left as an exercise.

This viewpoint links mechanics to a broader class of variational problems in mathematical optimization. The presence of constraints is addressed either by embedding them into the Lagrangian via multipliers or by restricting the set of admissible curves, and we will discuss this more at the end of this section.

We will explore several examples that use the Euler-Lagrange equations, including the brachistochrone problem and finding geodesics on surfaces.

Brachistochrone Problem

The brachistochrone problem is one of the earliest and most classical variational problems that motivated much of the modern theory presented so far. The objective of the brachistochrone problem is to find the curve between two points so that a particle constrained to this path will reach the end point as quickly as possible. Although simple to state, this problem defied several of the greatest minds at the time[12].

[12]Bernoulli posed this problem as an open challenge to the mathematical community. As the story goes, Netwon, long retired from mathematics and arriving home after a day at work jotted down the solution that evening. Although submitted anonymously, Bernoulli, when reading the solution knew it was Newton, saying "I could tell the lion by his paw print".

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

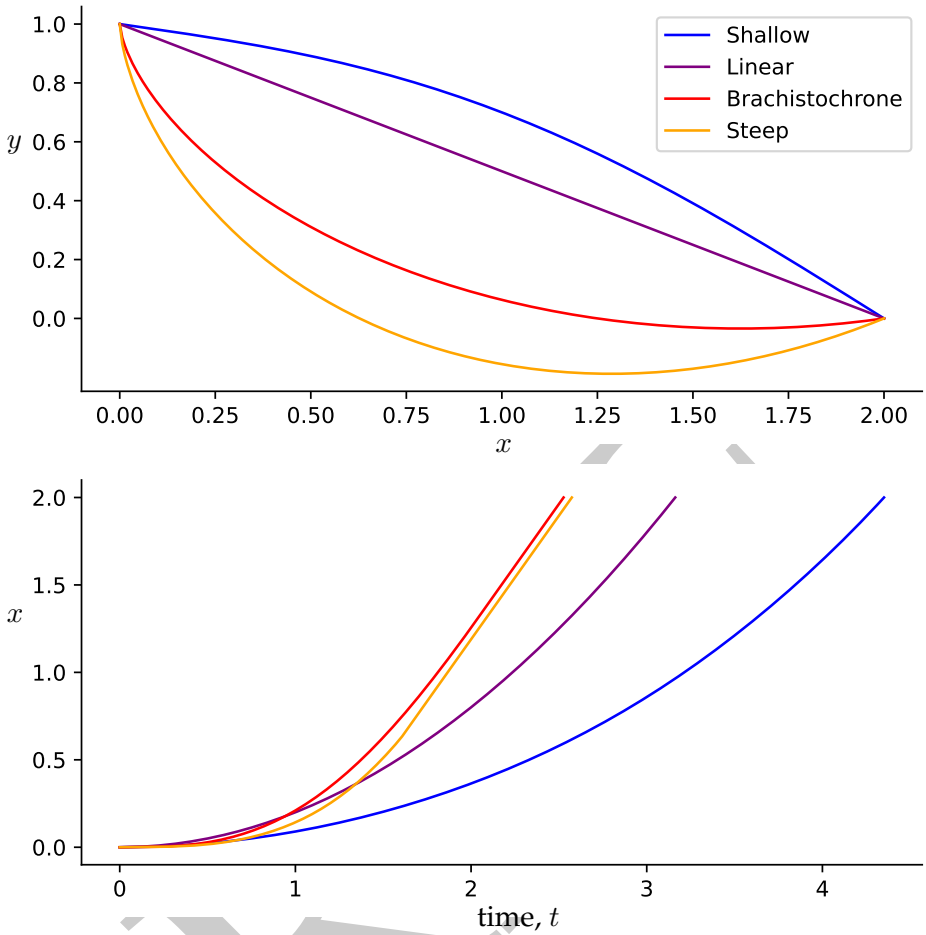


Figure 5.13: The solution to the brachistochrone problem is the cycloid (red) which would take a particle the minimum time to roll from the start to the end. (top) Several candidate curves, including the optimal cycloid, and (bottom) the x position versus t. We see that the cycloid reaches the end point $x = 2$ in the shortest time.

A simple guess for the optimal path may be a straight line, as this is the shortest path between two points. However, adding a slight curvature below the line will cause the particle to travel faster initially and gain momentum to arrive sooner. Several curves are shown in Fig. 5.13, along with their travel times. Freeze frames of the progress along these various paths are shown in Fig. 5.14. The optimal solution to the brachistochrone problem is called a cycloid, and we will derive it now.

We want a path $y(x)$, where y is measured vertically down from the top, that minimizes the total time T it takes for a frictionless bead to slide under uniform gravity from $(x = 0, y = 0)$ to $(x = x_1, y = y_1)$.

Assuming the bead starts at $y = 0$ with zero speed, then it's speed v at any point depends on the drop in height:

$$v = \sqrt{2gy}.$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

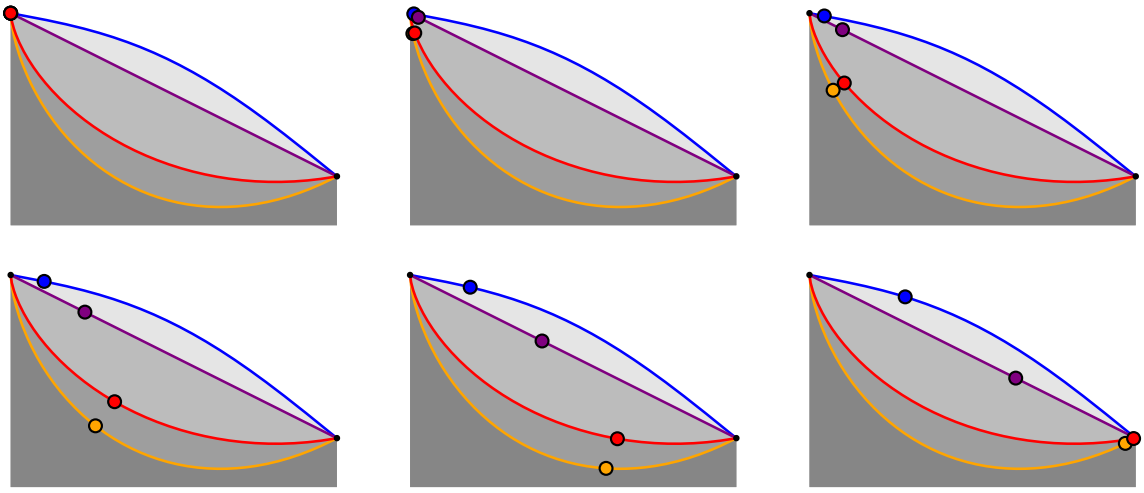


Figure 5.14: Snapshots of the various particles rolling down each of the four ramps from Fig. 5.13. The red particle on the optimal cycloid reaches the end first.

An infinitesimal path element ds along the curve satisfies

$$ds = \sqrt{1 + (y'(x))^2}\, dx,$$

since $ds^2 = dx^2 + dy^2$ and $y'(x) = dy/dx$.

The time to traverse an infinitesimal arc is

$$dT = \frac{\mathrm{d}s}{v} = \frac{\sqrt{1 + y'^2}\, dx}{\sqrt{2gy}}.$$

Hence the total time is

$$T[y(\cdot)] = \int_0^{x_1} \frac{\sqrt{1 + y'^2}}{\sqrt{2gy}}\, dx.$$

We can pull out the constant $\sqrt{1/(2g)}$, so the functional to be minimized becomes

$$T[y(\cdot)] = \sqrt{\frac{1}{2g}} \int_0^{x_1} \frac{\sqrt{1 + y'^2}}{\sqrt{y}}\, dx = \sqrt{\frac{1}{2g}} \int_0^{x_1} F(y, y')\, dx, \quad \text{where} \quad F(y, y') = \frac{\sqrt{1 + y'^2}}{\sqrt{y}}.$$

In standard variational–calculus notation, we take

$$L(x, y, y') = \frac{\sqrt{1 + y'^2}}{\sqrt{y}}.$$

A function $y(x)$ that extremizes T must then satisfy

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial L}{\partial y'}\right) - \frac{\partial L}{\partial y} = 0.$$

Notice that L does not explicitly depend on x. This lets us use a simpler form, the Beltrami identity, which we will show next.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Beltrami Identity. Because $L = L(y, y')$ has no explicit x dependence, we have

$$L(y, y') - y' \frac{\partial L}{\partial y'} = \text{constant}.$$

In our notation,

$$L(y, y') = \frac{\sqrt{1 + y'^2}}{\sqrt{y}}, \quad \frac{\partial L}{\partial y'} = \frac{1}{\sqrt{y}} \frac{y'}{\sqrt{1 + y'^2}}.$$

Hence

$$L - y' \frac{\partial L}{\partial y'} = \frac{\sqrt{1 + y'^2}}{\sqrt{y}} - y' \left(\frac{1}{\sqrt{y}} \frac{y'}{\sqrt{1 + y'^2}} \right) = \frac{1}{\sqrt{y}\sqrt{1 + y'^2}} = \text{constant}.$$

We will let the constant be $1/(2a)$, so that

$$\frac{1}{\sqrt{y}\sqrt{1 + y'^2}} = \frac{1}{2a} \implies \sqrt{1 + y'^2} = 2a\sqrt{y}.$$

Solution: A Cycloid. We may now solve for $y'(x)$

$$1 + y'^2 = 4a^2 y \implies y'^2 = 4a^2 y - 1, \qquad y' = \sqrt{4a^2 y - 1}.$$

This is a first-order differential equation that we can solve by separation of variables. A standard approach is to parametrize by the angle of a rolling circle:

$$y(t) = a(1 - \cos t), \quad x(t) = \frac{a}{2}(t - \sin t).$$

The constants a and t_{end} are fixed by the boundary conditions. The curve $(x(t), y(t))$ is a cycloid, which is the extremal, minimal-time curve, matching the classical solution.

It is possible to numerically verify that this solution is optimal by trying small variations $q_\varepsilon(t) = q(t) + \varepsilon\eta$. We visualize the ensemble of nearby trajectories and their travel times in Fig. 5.15. We plot the arrival times for the ensemble as a histogram in Fig. 5.16, where we see that the cycloid is in fact the curve with shortest arrival time.

Tautochrone Problem

The tautochrone problem, first posed by Huygens in the 17th century[13], asks for the curve along which a bead, sliding without friction under uniform gravity, reaches the lowest point in the same amount of time regardless of its starting point on the curve. As in the brachistochrone problem, we assume the particle starts from rest at height $y = 0$ and slides down a curve $y(x)$, where y is measured vertically downward. We will show that the solution is also a cycloid, visualized in Fig. 5.17.

The particle's speed at height y is determined by conservation of energy:

$$v = \sqrt{2gy}.$$

[13] Huygens even constructed a pendulum with a cycloidal constraint to ensure isochronous motion, a major step in the design of accurate timekeeping devices.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

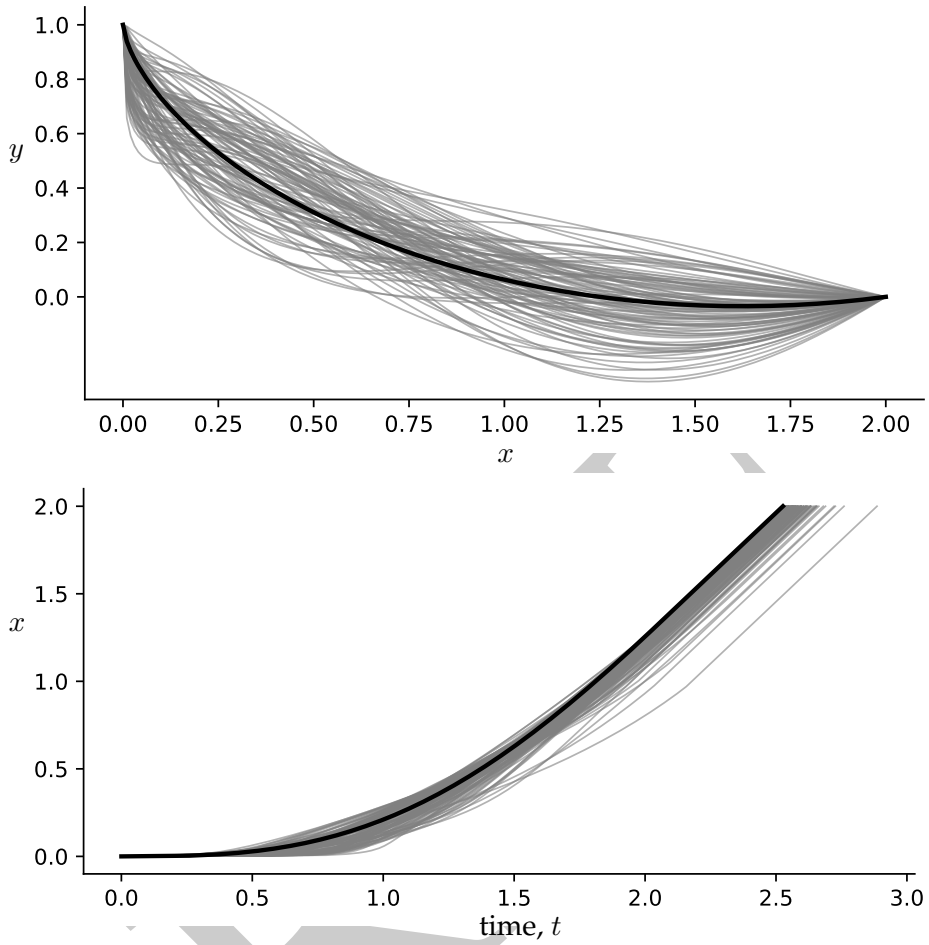


Figure 5.15: Here we compute the trajectories for several variations on the optimal cycloid. Each variation is visualized in gray. (top) The shape of the curves and (bottom) the position x versus time t. Among all variations, the cycloid has the minimum time to the end point $(2, 0)$. A histogram of the final time to reach the goal is shown in Fig. 5.16.

The arc length of the curve element is

$$ds = \sqrt{1 + y'^2}\, dx.$$

So the time to traverse an arc is

$$dT = \frac{\mathrm{d}s}{v} = \frac{\sqrt{1 + y'^2}}{\sqrt{2gy}}\, dx.$$

Let x be the horizontal coordinate corresponding to a starting point and x^* the endpoint at the bottom of the curve. The total time to reach the bottom from point x is

$$T[x] = \int_x^{x^*} \frac{\sqrt{1 + y'^2}}{\sqrt{2gy}}\, dx.$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

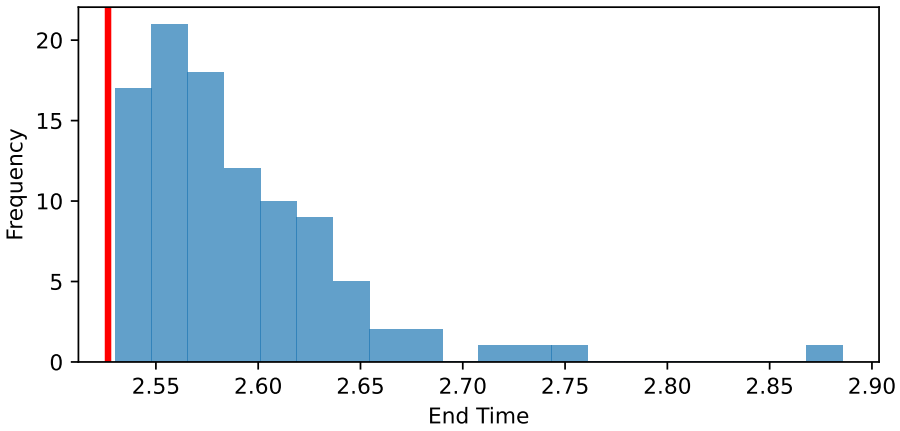


Figure 5.16: Histogram of the final time to reach the goal state for the optimal cycloid (red) and the ensemble of variations around the cycloid (blue).

Tautochrone condition. We now impose the tautochrone condition: the time $T[x]$ must be constant for all x, regardless of where the particle is released along the curve. This can only be true if the integrand is an exact differential of a function that increases linearly with arc length or with a parameter t such that $T = t^* - t$.

A classical way to enforce this is to change variables and parametrize the curve with respect to time t. Let

$$x = x(t), \quad y = y(t),$$

and assume the bead reaches the bottom at time $t = t^*$. Then, as before,

$$ds = \sqrt{\dot{x}^2 + \dot{y}^2}\, dt, \quad v = \sqrt{2gy},$$

so the total time becomes

$$T = \int_0^{t^*} dt.$$

But we can also write

$$dt = \frac{\mathrm{d}s}{v} = \frac{\sqrt{\dot{x}^2 + \dot{y}^2}}{\sqrt{2gy}}\, dt.$$

This implies that

$$\frac{\sqrt{\dot{x}^2 + \dot{y}^2}}{\sqrt{2gy}} = 1 \quad \Longrightarrow \quad \dot{x}^2 + \dot{y}^2 = 2gy.$$

We may now solve this differential equation, which describes a particle moving in a central gravitational field. Recognizing the solution as a cycloid, we may now choose a standard cycloidal parametrization:

$$x(t) = a(t - \sin t), \quad y(t) = a(1 - \cos t).$$

Then,

$$\dot{x} = a(1 - \cos t), \quad \dot{y} = a \sin t,$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

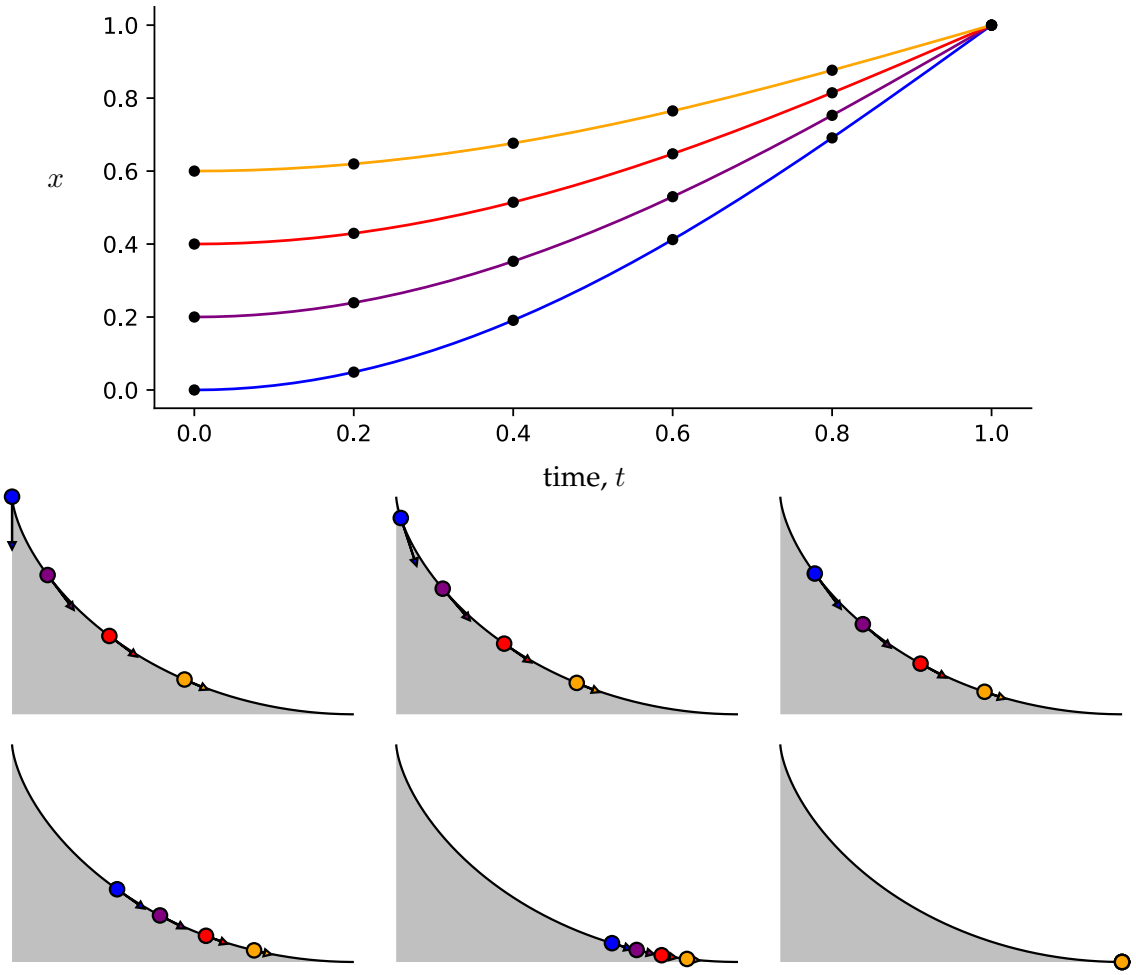


Figure 5.17: (top) Plots of four different solutions of the tautochrone, starting from different points on the curve. Each trajectory reaching the bottom ($x = 1$) at the same time ($t = 1$). (bottom) Several snapshots of the different trajectories at various times. The times are indicated in the top plot as black dots. Modified from `https://en.wikipedia.org/wiki/File:Tautochrone_curve.gif`; original code by Claudio Rocchini.

so

$$\dot{x}^2 + \dot{y}^2 = a^2(1 - \cos t)^2 + a^2 \sin^2 t = 2a^2(1 - \cos t),$$

and since

$$y(t) = a(1 - \cos t),$$

we get:

$$\dot{x}^2 + \dot{y}^2 = 2a^2 y(t).$$

Therefore, the condition $\dot{x}^2 + \dot{y}^2 = 2gy$ is satisfied if $a = \frac{g}{1}$, or more simply, $a =$ const and $g = 2a$. So the parametrization is valid for the right scaling of a.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The solution to the tautochrone problem is again a cycloid

$$x(t) = a(t - \sin t), \quad y(t) = a(1 - \cos t),$$

where a is a constant related to the acceleration due to gravity. This is exactly the same curve that solves the brachistochrone problem. Thus, the cycloid is the unique curve that both minimizes the time of descent and equalizes descent time from all starting points.

Example: Circle Maximizes Area for a Fixed Perimeter

Below is a classical Euler–Lagrange style derivation of the isoperimetric problem—finding the plane curve that maximizes the enclosed area A among all curves of fixed perimeter L. We show that the solution must be a circle.

Unlike the simpler brachistochrone problem, or other geodesic problems, here we have both a functional to be maximized (the area) and a constraint (the curve's perimeter is fixed). In the calculus of variations, one introduces a Lagrange multiplier to handle the constraint, then applies the Euler–Lagrange equation to the resulting combined functional.

First we parameterize the curve. A convenient choice is polar coordinates $[r(\theta), \theta]$, with θ going from 0 to 2π. The curve is

$$(x(\theta), y(\theta)) = (r(\theta)\cos\theta,\ r(\theta)\sin\theta), \qquad \theta \in [0, 2\pi].$$

The perimeter is given by

$$L[r] = \int_0^{2\pi} \sqrt{r(\theta)^2 + \dot{r}(\theta)^2}\, d\theta,$$

where $\dot{r} = dr/d\theta$, and the area is given by

$$A[r] = \frac{1}{2}\int_0^{2\pi} r(\theta)^2\, d\theta.$$

To enforce the constraints, we Introduce a Lagrange multiplier λ and consider

$$\Phi[r] = \int_0^{2\pi} \left[\tfrac{1}{2}\, r^2 - \lambda\, \sqrt{r^2 + \dot{r}^2}\right]\, d\theta + \lambda L_0,$$

with L_0 the prescribed perimeter. The integrand is

$$\mathcal{L}(r, \dot{r}) = \tfrac{1}{2}r^2 - \lambda\sqrt{r^2 + \dot{r}^2}.$$

The Euler-Lagrange equation is then

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(-\lambda\, \frac{\dot{r}}{\sqrt{r^2 + \dot{r}^2}}\right) - \left[r - \lambda\, \frac{r}{\sqrt{r^2 + \dot{r}^2}}\right] = 0.$$

A constant radius $r(\theta) \equiv R$ satisfies this (choose $\lambda = R$). The length constraint gives $R = L_0/(2\pi)$, and the enclosed area is πR^2, maximal among all curves of perimeter L_0. Thus, the stationary solution is a circle, which uniquely maximizes area for a given perimeter.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Legendre Transformation and the Dual Hamiltonian Form

To connect to the Hamiltonian picture, one performs the Legendre transform of the Lagrangian. For a single coordinate q_j, define the momentum

$$p_j \;=\; \frac{\partial L}{\partial \dot{q}_j}.$$

Then the Hamiltonian is

$$H(\mathbf{q}, \mathbf{p}, t) \;=\; \mathbf{p}\,\dot{\mathbf{q}} \;-\; L(\mathbf{q}, \dot{\mathbf{q}}, t),$$

with $\dot{\mathbf{q}}$ viewed as a function of $(\mathbf{q}, \mathbf{p})$ by inverting $\mathbf{p} = \frac{\partial L}{\partial \dot{\mathbf{q}}}$. In coordinates[14], $p_j = \frac{\partial L}{\partial \dot{q}_j}$, and

$$H(q_1, \ldots, q_n,\, p_1, \ldots, p_n,\, t) = \sum_{j=1}^{n} p_j\, \dot{q}_j - L(\mathbf{q}, \dot{\mathbf{q}}, t).$$

The condition that the action is stationary is equivalent to Hamilton's canonical system:

$$\dot{q}_j = \frac{\partial H}{\partial p_j}, \qquad \dot{p}_j = -\frac{\partial H}{\partial q_j}.$$

From a variational perspective, one can treat $(\mathbf{q}, \mathbf{p})$ as independent variables and impose the symplectic structure $\omega = \sum dq_j \wedge dp_j$ and read off the Hamilton equations as a statement that H is constant along solutions if H has no explicit time dependence. The Hamiltonian is thus a dual form of the Lagrangian, in the sense that the energy function arises from a Legendre transform, turning velocities into momenta.

Constraints in Hamiltonian Formalism. In the presence of constraints $\mathbf{h}(\mathbf{q}) = 0$ (and possibly momenta constraints $\mathbf{h}(\mathbf{q}, \mathbf{p}) = 0$), one must carry them forward into the Hamiltonian framework. The consistent procedure is to identify the primary constraints in phase space, then incorporate them using Dirac–Bergmann or simpler Lagrange multiplier methods. The final effect is that the Hamiltonian is augmented by $\boldsymbol{\mu}^T \mathbf{h}(\mathbf{q}, \mathbf{p})$, or we restrict the dynamics to a constraint submanifold of phase space. Either way, the principle of stationary action is preserved, but now in the space of $(\mathbf{q}, \mathbf{p})$ subject to these constraints.

Relationship to Fenchel Conjugate. The Legendre transform applies to smooth, convex functions and maps $f(x)$ to $f^*(y) = \sup_x(xy - f(x))$ where $y = \nabla f(x)$. The Fenchel conjugate extends this to any proper convex function, including those that are non-differentiable or take $+\infty$ values. Both describe the tightest linear underestimator of f at a given slope and are involutive under regularity conditions: $f^{**} = f$. In optimization, the Fenchel conjugate underlies duality theory. In physics, the Legendre transform connects dual variables (e.g., energy vs. entropy, Lagrangian vs. Hamiltonian).

[14] Often the momentum variables are denoted with a subscript p_j while the configuration coordinates are denoted with a subscript q^j. This notation is more classical in physics, representing that $\mathbf{p}$ is a dual vector to the velocity $\dot{\mathbf{q}}$, and is often used for shorthand, such as the Einstein summation convention. However, here we choose to use subscripts for all coordinates to conform to simpler engineering vector notation.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The formula for the Legendre transform is identical to that of the Fenchel conjugate, but the Legendre transform assumes differentiability and full-domain convexity. In particular, the Fenchel conjugate allows functions that take the value $+\infty$ to encode constraints, and it applies even to nondifferentiable functions such as norms or indicator functions.

Quadratic Function. Let $f(x) = \frac{1}{2}x^2$. Then

$$f^*(y) = \sup_{x \in \mathbb{R}} \left(xy - \frac{1}{2}x^2 \right). \tag{5.69}$$

Setting the derivative to zero yields the maximizer $x = y$, so

$$f^*(y) = y^2 - \frac{1}{2}y^2 = \frac{1}{2}y^2. \tag{5.70}$$

Here, the Legendre transform and the Fenchel conjugate coincide, because f is convex and differentiable everywhere.

Indicator Function. Let $f(x) = \delta_{[0,1]}(x)$, the indicator function of the interval $[0, 1]$:

$$f(x) = \begin{cases} 0 & \text{if } x \in [0,1], \\ +\infty & \text{otherwise.} \end{cases} \tag{5.71}$$

Then the Fenchel conjugate is

$$f^*(y) = \sup_{x \in [0,1]} (xy) = \begin{cases} 0 & \text{if } y = 0, \\ y & \text{if } y > 0, \\ 0 & \text{if } y < 0, \end{cases} \tag{5.72}$$

which equals the support function of the interval $[0, 1]$. The Legendre transform is not defined here due to the lack of differentiability and full domain.

KKT Conditions and Duality in Infinite Dimensions

One can state the infinite-dimensional optimization problem as:

$$\begin{aligned} &\text{minimize} && \mathcal{J}[y] \\ &\text{subject to} && y \in \mathcal{Y}, \\ &&& g_j[y] \geq 0, \quad j = 1, \ldots, m, \end{aligned}$$

where $\mathcal{Y}$ is an admissible function space, and $c_i[y]$ are (possibly integral) constraints. The KKT conditions in the infinite-dimensional setting are similar to those in the finite-dimensional case. They still involve stationarity, via a functional derivative condition, akin to Euler-Lagrange; primal feasibility, meaning that the constraints are satisfied; dual feasibility, meaning that the Lagrange multipliers $\mu_j \geq 0$; complementary slackness, $\mu_j g_j[y] = 0$ for each i; and possibly boundary or transversality conditions. These conditions are infinite-dimensional analogs of those in finite-dimensional convex optimization [170].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

In indirect methods, one formally applies the Euler-Lagrange equation, or KKT conditions, to obtain differential equations and solves them analytically or numerically. In contrast, direct methods discretize the function space, via finite elements or finite differences, and then solve the resulting large, finite-dimensional problem using standard optimization tools such as interior point methods or sequential quadratic programming [268].

A common workflow in PDE-constrained optimization involves representing $y(x)$ by a set of finite parameters (e.g., piecewise polynomials), approximating $\mathcal{J}[y]$ and any constraints by summations or quadrature, and solving the resulting finite-dimensional optimization problem using gradient or Newton iterations. This approach is popular due to existing solvers that can handle large-scale, sparse systems efficiently.

Alternatively, one may solve the variational (KKT) system at the continuous level first and then discretize. The advantage is a more precise expression of conditions such as adjoint equations. However, it can be more complex to implement and analyze.

Constraints as Additional Conditions. In many mechanical systems, one has holonomic constraints expressible as $\phi(q) = 0$. Such a constraint can be incorporated into the Lagrangian through the method of Lagrange multipliers or by reduction [40]. When one sets up $L' = L + \lambda\,\phi(q)$, the variation with respect to λ yields $\phi(q) = 0$, and variation with respect to q yields the Euler–Lagrange equations with an additional $\lambda\,\partial\phi/\partial q$ term.

An alternative is to interpret constraints as restricting the space of admissible curves $q(t)$. The variation is then taken only among curves satisfying $\phi(q(t)) \equiv 0$ for all t. The final outcome is that the constraint is satisfied at each instant, and the Euler–Lagrange equations incorporate the constraint forces, typically signaled by a Lagrange multiplier.

Types of Derivatives

Here, we outline various types of derivatives encountered in variational and infinite-dimensional optimization. These include total derivatives from finite-dimensional calculus, as well as their infinite-dimensional analogues, the Gâteaux and Fréchet derivatives. Each are used to derive and understand optimality conditions such as the Euler–Lagrange equations.

Total Derivative. The total derivative $\mathbf{D}f$ of a function $f : \mathbb{R}^n \to \mathbb{R}$, defined on a finite-dimensional $\mathbf{x}$, measures the rate of change of f in all directions in $\mathbb{R}^n$. It is defined by the gradient vector of partial derivatives:

$$\mathbf{D}f(\mathbf{x}) = \nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix},$$

and it satisfies the approximation[15]

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|).$$

This concept generalizes to Banach spaces through directional and functional derivatives.

[15]The term $o(\|\boldsymbol{\delta}\|)$ means that the remainder vanishes faster than linearly as $\|\boldsymbol{\delta}\| \to 0$, which is the defining condition for differentiability. In contrast, $\mathcal{O}(\|\boldsymbol{\delta}\|)$ only guarantees that the error is bounded by a constant multiple of $\|\boldsymbol{\delta}\|$ near zero. For precise first-order expansions, little-o is the appropriate notation.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Functional Derivative. Let $\mathcal{A}[\mathbf{q}]$ be a functional that maps a function $\mathbf{q}(t)$ to a scalar, as in Eq. (5.56). The functional derivative $\delta\mathcal{A}/\delta\mathbf{q}(t)$ generalizes the notion of a gradient to infinite-dimensional function spaces. It satisfies the identity:

$$\delta\mathcal{A}[\mathbf{q};\boldsymbol{\eta}] := \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\mathcal{A}[\mathbf{q}+\varepsilon\boldsymbol{\eta}]\bigg|_{\varepsilon=0}.$$

When $\mathcal{A}$ is an integral, as in Eq. (5.56), it is possible to show that this may be written as

$$\delta\mathcal{A}[\mathbf{q};\boldsymbol{\eta}] = \int_{t_0}^{t_1} \frac{\delta\mathcal{A}}{\delta\mathbf{q}(t)} \cdot \boldsymbol{\eta}(t)\, dt.$$

This defines $\delta\mathcal{A}/\delta\mathbf{q}(t)$ as the function, or distribution, that plays the role of the gradient in the inner product with any admissible variation $\boldsymbol{\eta}(t)$. Showing this last step will be an exercise for the reader. The Euler–Lagrange equations result from setting the functional derivative to zero under admissible variations.

Here we will discuss two particulary useful types of functional derivatives, the Gâteaux and Fréchet derivatives. The Gâteaux derivative is more common and requires less stringent conditions than the Fréchet derivative. In practice, the Gâteaux derivative is easiest to compute but weakest, and the Fréchet derivative ensures stronger properties like continuity. It is also possible to use these definitions to define directional derivatives, extending the finite-dimensional notion to infinite dimensions.

Gâteaux Derivative. Let $\mathcal{A} : \mathcal{X} \to \mathbb{R}$ be a functional defined on a normed vector space $\mathcal{X}$. The Gâteaux derivative at $\mathbf{q} \in \mathcal{X}$ in the direction $\boldsymbol{\eta} \in \mathcal{X}$ is defined as:

$$\mathbf{D}f(\mathbf{q};\boldsymbol{\eta}) := \lim_{\varepsilon\to 0} \frac{\mathcal{A}(\mathbf{q}+\varepsilon\boldsymbol{\eta}) - \mathcal{A}(\mathbf{q})}{\varepsilon}.$$

It generalizes the directional derivative and may exist even if $\mathcal{A}$ is not differentiable in the stronger Fréchet sense. If this limit exists for all $\boldsymbol{\eta}$, then $\mathcal{A}$ is Gâteaux differentiable at $\mathbf{q}$.

Fréchet Derivative. The Fréchet derivative is a stronger and more rigorous version of the Gâteaux derivative. A functional $\mathcal{A} : \mathcal{X} \to \mathbb{R}$ is Fréchet differentiable at $\mathbf{q}$ if there exists a bounded linear operator $L : \mathcal{X} \to \mathbb{R}$ such that

$$\lim_{\|\boldsymbol{\eta}\|\to 0} \frac{|\mathcal{A}(\mathbf{q}+\boldsymbol{\eta}) - \mathcal{A}(\mathbf{q}) - L(\boldsymbol{\eta})|}{\|\boldsymbol{\eta}\|} = 0.$$

We write this as

$$\mathbf{D}\mathcal{A}(\mathbf{q})[\boldsymbol{\eta}] = L(\boldsymbol{\eta}),$$

and call $\mathbf{D}\mathcal{A}(\mathbf{q})$ the Fréchet derivative. If a functional is Fréchet differentiable, then it is also Gâteaux differentiable, but the converse is not necessarily true. Fréchet differentiability is sufficient to apply powerful results from functional analysis and optimization. Formally, one often uses Fréchet derivatives in Banach spaces (e.g., $W^{1,2}$ Sobolev spaces) to handle more complex problems [122].

In the exercises, we will explore examples where the Gâteaux and Fréchet derivatives agree, and examples where they are different to solidify these concepts.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) Constrained optimization and Lagrange multiplier methods, by D. P. Bertsekas, 2014 [34].

(2) Conjugate duality and optimization, by R. T. Rockafellar, 1974 [297].

(3) Conjugate duality in convex optimization, by R. I. Bot, 2009 [44].

(4) Introduction to mechanics and symmetry, by J. E. Marsden and T. S. Ratiu, 2009 [241].

Papers and reviews

(1) Lagrange multipliers and optimality, by R. T. Rockafellar, *SIAM review*, 1993 [298].

(2) Duality in nonconvex optimization, by J. F. Toland, *Journal of Mathematical Analysis and Applications*, 1978 [341].

(3) Playing with duality: An overview of recent primal? dual approaches for solving large-scale optimization problems, by N. Komodakis, J.-Ch. Pesquet, *IEEE Signal Processing Magazine*, 2015 [204].

(4) Discrete mechanics and optimal control, by O. Junge, J. E. Marsden, and S. Ober-Bl':obaum, *IFAC Proceedings Volumes*, 2005 [188].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

Exercise 5–1: Solve the constrained problem:

$$\min_{x,y\in\mathbb{R}} (x-2)^2 + (y+1)^2 \quad \text{subject to} \quad x^2 + y^2 = 4.$$

Find the Lagrange multiplier and interpret the geometric meaning of this solution.

Exercise 5–2: Verify that Eq. (5.8) yields a solution where the gradient of the objective function is normal to the constraint equation.

Exercise 5–3: Consider

$$\min_{x\in\mathbb{R}^2} \; f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \geq 0, \; i = 1, 2.$$

List the KKT conditions in terms of the Lagrange multipliers μ_1, μ_2. Discuss complementary slackness and stationarity in your own words.

Exercise 5–4: We will now explore the *saddle point* property of the KKT conditions for the case of equality constraints. Consider the following constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) = x_1^2 + x_2^2 \\ \text{s.t.} \quad & x_1 + x_2 = 1. \end{aligned}$$

Part (a). Use the KKT conditions to solve for the optimal point.

1. Write the equality constraint in the form $h(\mathbf{x}) = 0$ and form the Lagrangian $\mathcal{L}(\mathbf{x}, \lambda)$.
2. Compute the gradient with respect to $\mathbf{x}$ and λ.
3. Set $\nabla_{\mathbf{x}}\mathcal{L} = \mathbf{0}$ and $\nabla_\lambda \mathcal{L} = 0$, and solve for the optimal point $\mathbf{x}^* = (x_1^*, x_2^*)$ and the multiplier λ^*.

Part (b). Verify the saddle point property. We have just shown that the KKT point $(\mathbf{x}^*, \lambda^*)$ is a stationary point of the Lagrangian, i.e., $\nabla_{\mathbf{x}}\mathcal{L} = \mathbf{0}$ and $\nabla_\lambda \mathcal{L} = 0$.

1. Show that $\mathcal{L}(\mathbf{x}^*, \lambda) \leq \mathcal{L}(\mathbf{x}^*, \lambda^*)$ for all λ and that $\mathcal{L}(\mathbf{x}, \lambda^*) \geq \mathcal{L}(\mathbf{x}^*, \lambda^*)$ for all $\mathbf{x}$ satisfying the constraint $x_1 + x_2 = 1$.
2. Conclude that $(\mathbf{x}^*, \lambda^*)$ is a true saddle point of the Lagrangian.

Part (c). Plot the Lagrangian.

1. Plot the Lagrangian $\mathcal{L}(\mathbf{x}, \lambda^*)$ as a function of (x_1, x_2) on the constraint line $x_1 + x_2 = 1$.
2. Plot $\mathcal{L}(\mathbf{x}^*, \lambda)$ as a function of λ.
3. Verify the saddle behavior through these plots.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 5–5: We now explore an example where the saddle point property is a bit more subtle. Consider the following constrained optimization problem:

$$\begin{aligned} \min_{x} \quad & f(x) = 3x \\ \text{s.t.} \quad & x \leq 3 \\ & x \geq -1. \end{aligned}$$

Part (a). Use the KKT conditions to solve for the optimal point.

1. Rewrite the problem in standard form for inequality constraints with $g_1(x) \leq 0$ and $g_2(x) \leq 0$. Form the Lagrangian $\mathcal{L}(x, \boldsymbol{\mu})$.

2. Compute the gradient with respect to x and $\boldsymbol{\mu}$.

3. Derive the KKT conditions and solve for the optimal point x^* and $\boldsymbol{\mu}^*$.

Part (b). Verify the saddle point property.

1. At the KKT point (x^*, μ_1^*, μ_2^*) evaluate the full gradient $\nabla_{x,\mu_1,\mu_2}\mathcal{L}$. Show that it is not equal to zero at the optimum point, and explain why this does not contradict the saddle point interpretation.

2. Explain what the saddle point condition means in this context. Specifically, show that

$$\mathcal{L}(x^*, \mu_1, \mu_2) \leq \mathcal{L}(x^*, \mu_1^*, \mu_2^*) \leq \mathcal{L}(x, \mu_1^*, \mu_2^*)$$

holds for all $\mu_1, \mu_2 \geq 0$ and feasible x. This is the saddle point property of the constrained Lagrangian.

Part (c). Plot the Lagrangian.

1. Plot the Lagrangian $\mathcal{L}(x, \mu_1^*, \mu_2^*)$ as a function of x, and describe how the minimum over x is achieved.

2. Then plot $\mathcal{L}(x^*, \mu_1, \mu_2)$ as a function of μ_1, $\mu_2 \geq 0$, and verify that the KKT point is a saddle point over the feasible region.

3. Verify the saddle behavior through these plots.

Part (d). Discussion.

1. Explain why the constrained Lagrangian does not generally have a zero gradient at the saddle point when inequality constraints are present.

2. In your own words, summarize how the saddle-point interpretation of the KKT conditions differs from the usual unconstrained interpretation of saddle points as critical points with zero gradient.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 5–6: Verify that the regularity conditions are satisfied for the box-constrained least squares problem in Section 5.2.

Exercise 5–7: Derive the Lagrangian and the dual function for

$$\min_{\mathbf{x}} \quad x_1 + x_2 \quad \text{subject to} \quad \begin{cases} x_1^2 + x_2^2 \leq 25, \\ x_1 + x_2 \geq 2, \\ x_1,\, x_2 \geq 0. \end{cases}$$

Write the dual problem explicitly and discuss strong vs. weak duality here.

Solve the primal and the dual problem explicitly and discuss the results.

Exercise 5–8: Derive the dual program for the simple linear program with equality constraints:

$$\begin{aligned} &\text{Minimize} \quad \mathbf{c}^T\mathbf{x} \\ &\text{Subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned}$$

First draw the feasible set. Is it convex?

Write down the Lagrangian, the dual function, and the dual program. Solve both the primal and dual problem for a specific example of your choice.

Solve this optimization problem using an interior point method.

Exercise 5–9: Show that

$$f(x) = \begin{cases} 0 & \text{if } x = 0, \\ 1 & \text{if } x \neq 0. \end{cases}$$

is not lower semicontinuous.

Exercise 5–10: Show that dual function in Eq. (5.16) is always concave, even if the primal problem is non-convex, because it is the pointwise infimum of functions that are affine in $(\boldsymbol{\lambda}, \boldsymbol{\mu})$.

First, explain how it is the pointwise infimum of functions that are affine in $(\boldsymbol{\lambda}, \boldsymbol{\mu})$. Specifically, show that the augmented Lagrangian is affine in $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. Then show that the pointwise infimum of affine functions is always concave.

Exercise 5–11: Fill in the missing steps to derive the Lagrangian dual function in Eq. (5.47) for the constrained least squares problem.

Exercise 5–12: Prove that the dual function $\phi(\boldsymbol{\lambda})$ in Eq. (5.16) is always concave.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 5–13: Prove that the dual function $\phi(\boldsymbol{\lambda}, \boldsymbol{\mu})$ in Eq. (5.39) is always concave.

Exercise 5–14: Here we will extend the geometric interpretation of duality to include inequality constraints $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ as well. Specifically, in Sec. 5.3, we saw that for the case of convex equality constraints $\mathbf{h}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$, the pair $(\boldsymbol{\lambda}, \phi(\boldsymbol{\lambda}))$ defined a supporting hyperplane to the epigraph of f, especially defining a tight lower bound on f. The slope of this hyperplane is $\boldsymbol{\lambda}$.

In this exercise, we extend this geometric picture to convex programs with *inequality* constraints. The supporting hyperplane now arises not from the epigraph of but from the geometry of the *feasible region*.

Consider the convex optimization problem:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{g}(\mathbf{x}) \leq \mathbf{0},$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex and lower semicontinuous, and $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ is a vector of convex constraint functions.

Let $\mathbf{x}^*$ be an optimal solution, and let $\boldsymbol{\mu}^* \geq 0$ be the associated vector of Lagrange multipliers.

1. The feasible set $\mathcal{C}$ defined by the constraints $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ is convex. Show that the set of active constraints at $\mathbf{x}^*$, defined as

   $$\mathcal{A} := \{i \mid g_i(\mathbf{x}^*) = 0\},$$

   determines a face of the boundary $\partial\mathcal{C}$.

2. Prove that for each active constraint $j \in \mathcal{A}$, the gradient $\nabla g_j(\mathbf{x}^*)$ defines an inward-pointing normal to $\mathcal{C}$ at $\mathbf{x}^*$.

3. Show that the vector

   $$\sum_{j \in \mathcal{A}} \mu_j^* \nabla g_j(\mathbf{x}^*)$$

   is a *conic combination* of the active constraint normals, meaning it is a linear combination with nonnegative coefficients. A conic combination of vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is any vector of the form

   $$\sum_{j=1}^{k} \alpha_j \mathbf{v}_j, \quad \text{where } \alpha_j \geq 0 \text{ for all } j.$$

   Geometrically, this means the sum lies in the *conic hull* (also known as the *positive span*) of the vectors.

4. (optional) Show that the conic hull of a set of vectors is always convex.

5. Show that the vector $\sum_{j \in \mathcal{A}} \mu_j^* \nabla g_j(\mathbf{x}^*)$ defines the normal to a supporting hyperplane of $\mathcal{C}$ at $\mathbf{x}^*$, i.e., a hyperplane that locally touches the feasible region and lies entirely on one side of it.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

6. The stationarity condition

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{m} \mu_i^* \nabla g_i(\mathbf{x}^*) = 0$$

means that the negative gradient $-\nabla f(\mathbf{x}^*)$ lies in the conic hull of the active constraint normals. That is, the direction of steepest descent for f is exactly blocked by the boundary of the feasible set. There is no feasible direction of descent, which characterizes local optimality.

Geometrically, this condition ensures that the level sets of f are tangent to the feasible region at $\mathbf{x}^*$, and that a supporting hyperplane exists that separates the feasible region from the sublevel set $\{\mathbf{x} \mid f(\mathbf{x}) < f(\mathbf{x}^*)\}$.

Use the KKT stationarity condition to conclude that the gradient $\nabla f(\mathbf{x}^*)$ is in the span of the constraint normals. What does this imply geometrically about feasible directions of descent?

7. Show that no feasible descent direction exists at $\mathbf{x}^*$, and that any direction $\mathbf{d}$ with $\nabla f(\mathbf{x}^*)^T \mathbf{d} < 0$ must violate at least one constraint.

Exercise 5–15: We will now explore the previous geometric interpretation of duality on a specific problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & x_1^2 + x_2^2 \\ \text{s.t.} \quad & x_1 + x_2 - 1 \leq 0, \\ & -x_1, -x_2 \leq 0. \end{aligned}$$

1. Determine and plot the feasible region.
2. Identify the optimal point $\mathbf{x}^*$ and the active constraints.
3. Show that the gradients of the active constraints form inward normals to the boundary.
4. Verify the KKT stationarity condition and visualize the balance of forces.
5. Visualize the supporting hyperplane determined by the active constraints.

Exercise 5–16: Verify that the convex envelop of

$$f(x) = \max(-x^2,\ -|x|),$$

is equal to

$$f^{**}(x) = -|x|,$$

as in Fig. 5.9.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 5–17: Show that the solution to the optimization problem in Eq. (5.51) will satisfy the first KKT condition.

Exercise 5–18: Show that the gradient and Hessian of the log-barrier-augmented objective function in Eq. (5.54) for linear programming is given by the expressions in Eq. (5.55).

Exercise 5–19: Modify the interior point method code 5.1 to solve a quadratic program with the same linear constraints as in the example. The quadratic objective is given by

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x}, \quad \text{where} \quad \mathbf{Q} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \quad \text{and} \quad \mathbf{c}^T = \mathbf{x}_0\mathbf{Q} \quad \text{for} \quad \mathbf{x}_0 = \begin{bmatrix} 40 \\ 30 \end{bmatrix}.$$

The solution path is shown in Fig. 5.11 (right).

Also show by hand that the analytic minimum of $f(\mathbf{x})$ is at $\mathbf{x} = \mathbf{x}_0$.

Exercise 5–20: Solve the box-constrained least-squares problem in Eq. (5.14) using the interior point method.

Plot the central path for different IPM parameters.

Exercise 5–21: For the box-constrained least-squares problem in Eq. (5.14), derive the following expressions for the Lagrange multipliers: $\mu_j = t/(x_j - l_j)$ and $\nu_j = t/(u_j - x_j)$.

Exercise 5–22: Solve the box-constrained least-squares problem in Eq. (5.14) using the projected gradient algorithm for various step sizes γ.

Compare with the IPM solution from above.

Exercise 5–23: A mass on a frictionless plane is constrained to move on a curve given by $h(x, y) = 0$. Write the equations of motion using Lagrange multipliers to enforce the constraint. Compare this to the standard Newton's laws for constrained motion.

Write down the specific equations of motion for a simple parabolic $h(x, y) = y - x^2 = 0$.

Exercise 5–24: Modify the derivation of the Euler-Lagrange equations to handle non-fixed endpoints.

Exercise 5–25: Derive the expression in Eq. (5.63) by expanding the Lagrangian inside the integral in powers of ε and then taking the derivative with respect to ε at 0.

Exercise 5–26: Derive the Beltrami identity, namely that if $L = L(y, y')$ has no explicit x dependence, then

$$L(y, y') - y' \frac{\partial L}{\partial y'} = \text{constant}.$$

Exercise 5–27: Plug the cycloid solution for the Brachistichrone problem back into the Euler-Lagrange equations to verify that it is locally extremal.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 5–28: Similarly, show that the cycloid solution is also a solution to the tautochrone problem by showing that it solves the Euler-Lagrange equations.

Exercise 5–29: Show that a straight line is the extremal path that is shortest between two points A and B by defining a functional and verifying that the Euler-Lagrange equations are satisfied.

Exercise 5–30: Derive the formula for the great arc as a geodesic (e.g., shortest distance path between two points on a curved surface) on the surface of a sphere.

Exercise 5–31: Here we will explore Lagrange multipliers for another simple mechanical system, the pendulum. Derive the equations of motion with the three canonical approaches:

1. Newton's laws in (x, y) coordinates with constraints as Lagrange multipliers;
2. The Euler-Lagrange equations in angular coordinates $(\theta, \dot{\theta})$;
3. Hamilton's equations in conugate coordinates (q, p), with $q = \theta$.

Compare and discuss these various equations and how the constraints are being satisfied.

Exercise 5–32: Show that

$$\delta\mathcal{A}[\mathbf{q};\boldsymbol{\eta}] := \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\mathcal{A}[\mathbf{q}+\varepsilon\boldsymbol{\eta}]\bigg|_{\varepsilon=0} = \int_{t_0}^{t_1} \frac{\delta\mathcal{A}}{\delta\mathbf{q}(t)} \cdot \boldsymbol{\eta}(t)\, dt$$

when $\mathcal{A}$ is an integral, as in Eq. (5.56).

First, expand the perturbed functional to first order:

$$\mathcal{A}[\mathbf{q}+\varepsilon\boldsymbol{\eta}] = \mathcal{A}[\mathbf{q}] + \varepsilon \int \mathcal{B}(\boldsymbol{\eta})\, dt + \mathcal{O}(\varepsilon^2),$$

and extract the integrand of the linear term as $\delta\mathcal{A}/\delta\mathbf{q}(t)$.

Exercise 5–33: Compute the functional derivative of

$$\mathcal{A}[\mathbf{q}] = \int_{t_0}^{t_1} \left(\tfrac{1}{2}\|\dot{\mathbf{q}}(t)\|^2 - V(\mathbf{q}(t))\right)\, dt.$$

Specifically, consider $\mathbf{q}_\varepsilon(t) = \mathbf{q}(t) + \varepsilon\boldsymbol{\eta}(t)$ and compute:

$$\delta\mathcal{A} = \int_{t_0}^{t_1} \left(-\nabla V(\mathbf{q}(t)) \cdot \boldsymbol{\eta}(t) + \dot{\mathbf{q}}(t) \cdot \dot{\boldsymbol{\eta}}(t)\right)\, dt.$$

Integrate the second term by parts, with fixed endpoints, and set equal to zero to yield a particularly simple form of the Euler-Lagrange equations for a particle in a potential well

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

$V(\mathbf{q})$.

Exercise 5–34: Here we will explore an example where the Gâteaux and Fréchet derivatives are the same. Let

$$F[q] = \int_0^1 q(t)^2 \, dt$$

where $q \in L^2[0,1]$. Define a perturbation $q_\varepsilon(t) = q(t) + \varepsilon\eta(t)$, where $\eta(t)$ is an admissible variation in $L^2[0,1]$.

1. Show that the Gâteaux derivative of F in the direction η is:

$$\delta F[q;\eta] = \lim_{\varepsilon\to 0} \frac{F[q+\varepsilon\eta] - F[q]}{\varepsilon} = 2\int_0^1 q(t)\eta(t)\, dt.$$

2. Verify that this directional derivative is linear and continuous in η.
3. Conclude that the Fréchet derivative exists and coincides with the Gâteaux derivative.
4. Show that the functional derivative exists in the distributional sense and satisfies:

$$\frac{\delta F}{\delta q(t)} = 2q(t),$$

meaning that for any admissible $\eta(t)$,

$$\delta F = \int_0^1 \frac{\delta F}{\delta q(t)} \eta(t)\, dt.$$

Exercise 5–35: Here we will explore an example where the Gâteaux and Fréchet derivatives are different. Now, let

$$F[q] = \left(\int_0^1 |q(t)|\, dt\right)^2$$

with $q \in L^1[0,1]$. Define $q_\varepsilon(t) = q(t) + \varepsilon\eta(t)$ for any $\eta \in L^1[0,1]$.

1. Show that F is well-defined and finite for all $q \in L^1[0,1]$.
2. Compute the Gâteaux derivative of F at $q = 0$ in the direction η.
3. Show that this derivative equals zero for all directions η, i.e.,

$$\delta F[0;\eta] = 0.$$

4. Argue why this Gâteaux derivative is not linear or continuous in η in any neighborhood of $q = 0$.
5. Conclude that the Fréchet derivative does not exist at $q = 0$.
6. Discuss whether a functional derivative exists in this case, and what it would mean.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 6

Global and Nonconvex Optimization

In this chapter we explore several methods for non-convex and gradient-free optimization. This chapter will have a bit less structure than the previous chapters, because the topic of non-convex optimization is itself less structured. This is a vast field, and we will only cover some of the major highlights.

First, we will explore two workhorse algorithms that assume very little structure in the optimization problem: the Nelder-Mead downhill simplex method, and simulated annealing. The downhill simplex method, not to be confused with Dantzig's simplex method from linear programming, uses a simplex (aka a polytope) to sample the objective function landscape and then take steps that deform the simplex towards better objectives. Simulated annealing replicates the natural process of annealing, or cooling, when creating strong metal alloys, where a specific cooling schedule is used to promote strong crystal structures in the material. In simulated annealing, the optimization algorithm begins "hot", exploring the objective function landscape, and gradually "cools" to begin exploiting favorable local minima.

Next, we introduce evolutionary algorithms, including genetic algorithms and genetic programming. Genetic programming is particularly interesting in the context of modern explainable and interpretable AI. This will lead into a discussion of swarm and particle optimization techniques, with examples.

Finally, we will cover the important combined topics of regularization, relaxation, and splitting methods, which unify much of the discussions of constraints and duality in the last chapter. This will lead to workhorse splitting algorithms, such as the alternating descent method of multipliers (ADMM).

DRAFT

6.1 Nelder-Mead Downhill Simplex

The Nelder-Mead Downhill Simplex method is a popular gradient-free, local search technique for unconstrained optimization problems. It was introduced by John Nelder and Roger Mead in 1965 [261]. Unlike gradient-based methods, which require information about the gradient or Hessian of an objective function, Nelder-Mead works purely with function value evaluations. It is particularly useful when the objective function is non-differentiable or discontinuous in certain regions, or when numerical or analytical gradients are difficult or expensive to obtain. Typically this approach works well when the dimension of the problem is relatively small, commonly up to a few dozen variables.

Despite its popularity in practice and its ease of implementation, the Nelder-Mead method does not have strong theoretical convergence guarantees for arbitrary non-convex problems. In fact, there are known pathological cases in higher dimensions where performance can degrade. Nevertheless, it remains widely used in engineering and applied mathematics for small- to medium-scale problems [329], particularly when function evaluations are expensive and derivative approximations or adjoint-based approaches are complicated or infeasible.

The idea behind the downhill simplex method is quite simple. Consider a minimization problem. Given an optimization variable $\mathbf{x} \in \mathbb{R}^n$, we initialize a simplex in $\mathbb{R}^n$ with $n+1$ vertices $\{\mathbf{x}_j\}_{j=0}^n$, evaluate the objective function at all vertices, and modify the vertex with the highest objective function value to ideally move to a lower objective function value. In this way, we iteratively move the centroid of the simplex to a lower objective function value, one vertex at a time, without needing to compute a gradient at any step. In a sense, this simplex "rolls down" the objective function landscape; see Fig. 6.1 for an illustration. There are several details and options for how to modify the vertex with the highest value, depending on the dimension of the problem, the structure of the objective function landscape, among other considerations. In addition, we may shrink or expand the size of the simplex depending on properties of the objective function landscape.

This may be mathematically formalized as follows. At each step, the current simplex is defined by the points $\{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_n\}$. We assume that the points are sorted so that

$$f(\mathbf{x}_0) \leq f(\mathbf{x}_1) \leq \cdots \leq f(\mathbf{x}_n),$$

where f is the objective function being minimized. We define the centroid of the n best points in the simplex, omitting the $n+1$-th, as

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{j=0}^{n-1} \mathbf{x}_j.$$

Then, at each step of the Nelder-Mead iteration, we modify the simplex according to the following procedure:

1. **Reflection.** We reflect the point $\mathbf{x}_n$ about the centroid $\bar{\mathbf{x}}$

$$\mathbf{x}_n' = \bar{\mathbf{x}} + \alpha(\bar{\mathbf{x}} - \mathbf{x}_n)$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

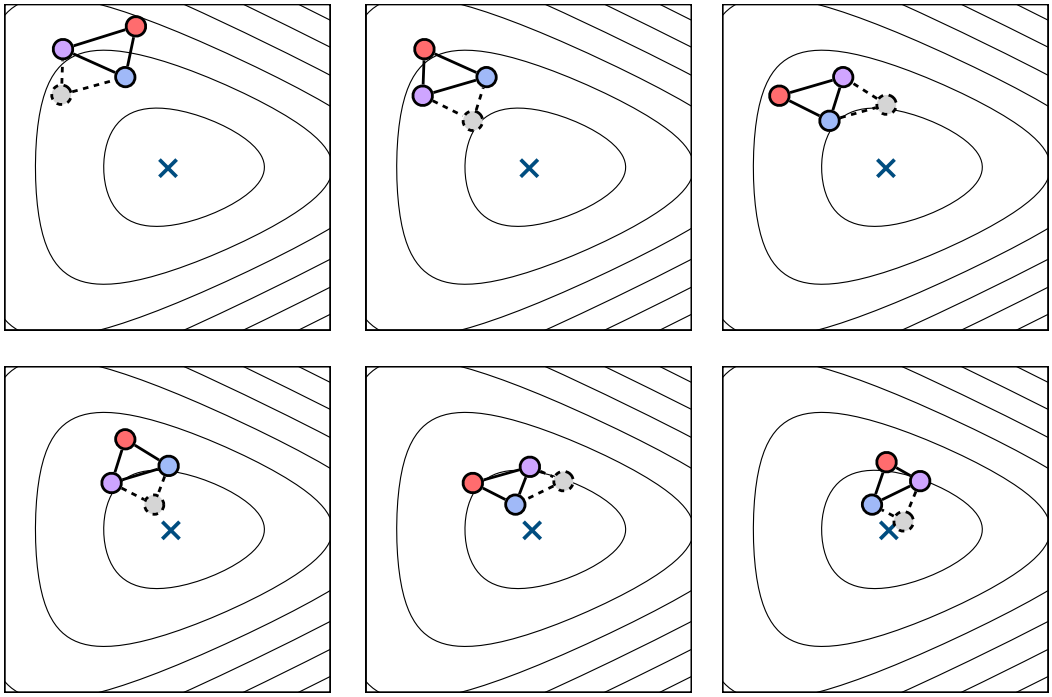


Figure 6.1: Schematic illustration of the downhill simplex method on a simple two-dimensional objective function with a single global minima. In this example, simple reflections suffice to drive the simplex towards the minima.

where α is the reflection coefficient that either increases the distance from the centroid ($\alpha > 1$), decreases it ($\alpha < 1$) or keeps it the same ($\alpha = 1$). Typically $\alpha \approx 1$ is a good initial choice.

We now compare $f(\mathbf{x}')$ against the objective value at the other vertices. If the new reflected vertex $f(\mathbf{x}')$ is better than any other vertex, but is not the global best vertex,

$$f(\mathbf{x}_0) \leq f(\mathbf{x}') < f(\mathbf{x}_{n-1})$$

then replace $\mathbf{x}_n$ with $\mathbf{x}'_n$, finish the iteration, and proceed back to step 1 for the next iteration. Otherwise, proceed to the next steps, depending on the ranking of $f(\mathbf{x}')$.

2. **Expansion.** If $\mathbf{x}'_n$ is the new best vertex, so that

$$f(\mathbf{x}') < f(\mathbf{x}_0)$$

then expand the point further in this new reflected direction:

$$\mathbf{x}'' = \bar{\mathbf{x}} + \gamma(\bar{\mathbf{x}} - \mathbf{x}_n)$$

where $\gamma \approx 2\alpha > 1$ is the expansion coefficient. If the new value is still the best, i.e. $f(\mathbf{x}'') < f(\mathbf{x}')$, then replace $\mathbf{x}_n$ with $\mathbf{x}''$, otherwise replace with $\mathbf{x}'$. The iteration ends and we proceed with step 1 in the next iteration.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

3. **Contraction.** If $\mathbf{x}'_n$ is still the worst vertex, so that

$$f(\mathbf{x}') \geq f(\mathbf{x}_{n-1})$$

then we perform a contraction

$$\mathbf{x}'' = \bar{\mathbf{x}} + \rho(\bar{\mathbf{x}} - \mathbf{x}_n)$$

where $\rho \approx 0.5\alpha < 1$ is the contraction coefficient. If the new vertex $\mathbf{x}''$ is better than $\mathbf{x}'$, i.e. $f(\mathbf{x}'') < f(\mathbf{x}')$, then replace $\mathbf{x}_n$ with $\mathbf{x}'$ and end this iteration. Otherwise, when all else fails, as a douse of cold water, we proceed to the shrinkage operation.

4. **Shrink.** Here we contract all other vertices $j = 1, \cdots n$ towards the best vertex $\mathbf{x}_0$:

$$\mathbf{x}_i \leftarrow \mathbf{x}_0 + \sigma\big(\mathbf{x}_i - \mathbf{x}_0\big),$$

where $0 < \sigma < 1$ is the *shrinkage* coefficient. Typically $\sigma \approx 0.5$ is reasonable.

These operations are shown schematically in Fig. 6.2. The above parameters are the defaults originally proposed by Nelder and Mead. Adjusting them can change the aggressiveness of exploration and the method's stability.

There are several common stopping criteria to end the iterations:

- The function values at all simplex vertices are sufficiently close to one another (e.g., $\max_j |f(\mathbf{x}_j) - f(\mathbf{x}_0)| < \epsilon$).
- The simplex size (e.g., the maximal distance between vertices) is below a tolerance.
- The number of function evaluations or iterations reaches a specified maximum.

Now that we have seen the algorithm, and before exploring a coding example, it is important to reiterate the strengths and limitations of this approach. The downhill simplex method is very simple to implement and requires no gradient or higher-order derivative information, making it a reasonable choice for many situations. It is also possible to code up from scratch and implement special or custom features. Thus, the algorithm is often effective for small-scale problems where function evaluations are costly, or gradient approximations are infeasible.

However, there are several drawbacks as well. There are relatively few convergence results for non-convex problems, and convergence to a local minima is not guaranteed to be global. However, the algorithm can often navigate several nearby local minima depending on the size of the initial simplex and optimization hyper parameters. In higher dimensions, the downhill simplex algorithm may perform poorly, often requiring many function evaluations. Finally, without careful tuning, the method might stagnate or roam slowly around an optimum.

However, in practice, the Nelder-Mead downhill simplex method remains one of the most widely used derivative-free local optimization techniques. Its simple conceptual framework and relatively straightforward hyperparameter setup make it attractive for problems in low to moderate dimensions, particularly when function evaluations are expensive or noisy but one still prefers a local method to refine near a suspected optimum.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

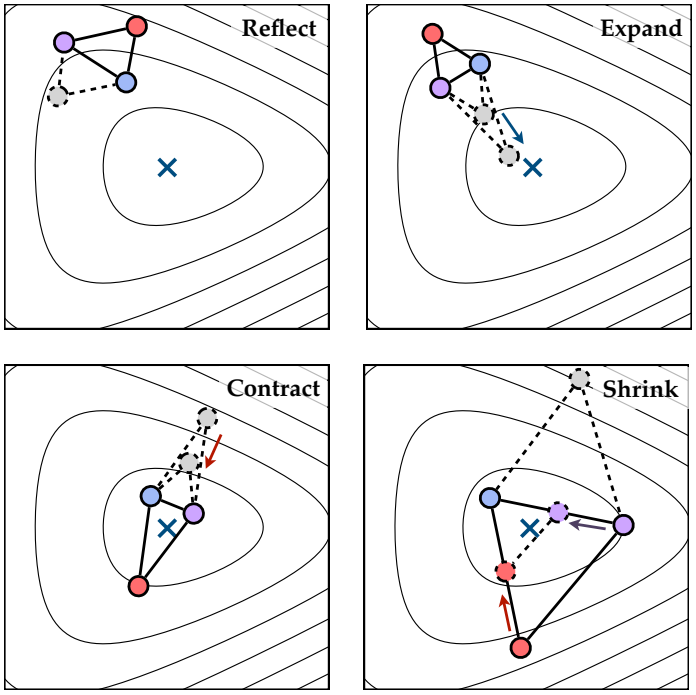


Figure 6.2: Illustration of the various downhill simplex operations. The first operation is a simple reflection. If the reflected point is the new optimum in the simplex, then we move to the expand operation, moving more in this favorable direction. If the new point is still the worst, then we contract, moving closer to the centroid. If all else fails, we move all points closer to the current optimum point.

Example in Python

Here we will demonstrate the ability of the downhill simplex method to find a global optima of a non-convex landscape with four local minima.

The objective function is given by

$$f(\mathbf{x}) = 1 + f_1(\mathbf{x}) + f_2(\mathbf{x}) + f_3(\mathbf{x}) + f_4(\mathbf{x}) \tag{6.1}$$

where each $f_j(\mathbf{x})$ is defined as

$$f_j(\mathbf{x}) = -\alpha_j e^{-\frac{1}{2}\|\mathbf{x}-\mathbf{x}_j\|^2}$$

and $\mathbf{x}_j$ is the center of the j-th minima and α_j is the depth. For this example, we use $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (3, 1.5, 1, 2)$ at centers $\mathbf{x}_1 = (2, 2), \mathbf{x}_2 = (-2, 2), \mathbf{x}_3 = (2, -2)$, and $\mathbf{x}_4 = (-2, -2)$. This function is visualized in Fig. 6.3.

The downhill simplex algorithm is simple to implement, and there are many established libraries (e.g., `scipy.optimize` with `method='Nelder-Mead'`). The following code performs a single iteration of the Nelder-Mead downhill simplex method, where α, γ, ρ, and σ are the reflection, expansion, contraction, and shrinkage coefficients.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

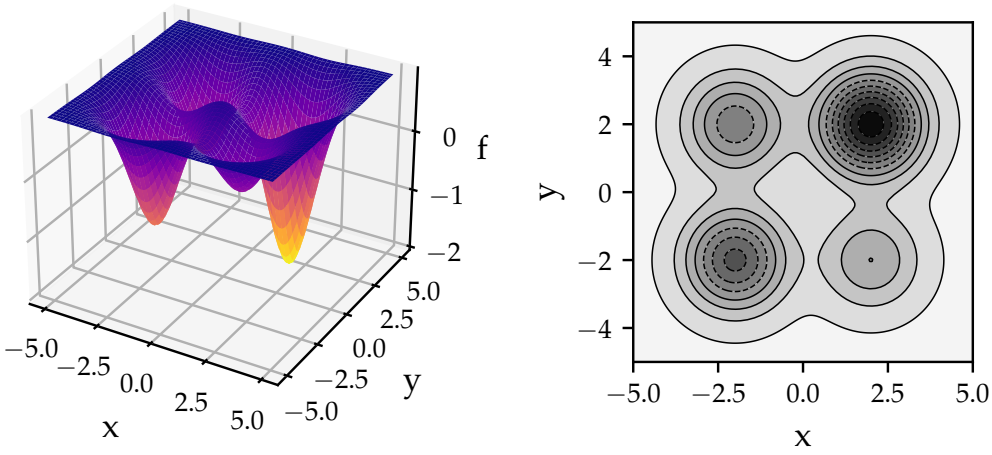


Figure 6.3: The objective function with four local minima, visualized in three-dimensions (left) and in contours (right).

```python
def nelder_mead_step(vertices, alpha, gamma, rho, sigma):
    vals = np.array([f(v) for v in vertices])
    order = np.argsort(vals)
    vertices = vertices[order]
    vals = vals[order]

    x1, x2, x3 = vertices # vertices : np.ndarray of shape (3,2)
    f1, f2, f3 = vals     # best, middle, worst

    center = (x1 + x2) / 2.0 # center of best and middle points

    # Reflection
    x_reflect = center + alpha * (center - x3)
    f_reflect = f(x_reflect)

    if f_reflect < f1:          # Expansion
        x_expand = center + gamma * (x_reflect - center)
        f_expand = f(x_expand)
        if f_expand < f_reflect:
            vertices[2] = x_expand
        else:
            vertices[2] = x_reflect
    elif f_reflect < f2:
        vertices[2] = x_reflect
    else       # Reflection is worse than mid
        if f_reflect < f3:      # Outside contraction
            x_contr = center + rho * (x_reflect - center)
            f_contr = f(x_contr)
            if f_contr <= f_reflect:
                vertices[2] = x_contr
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

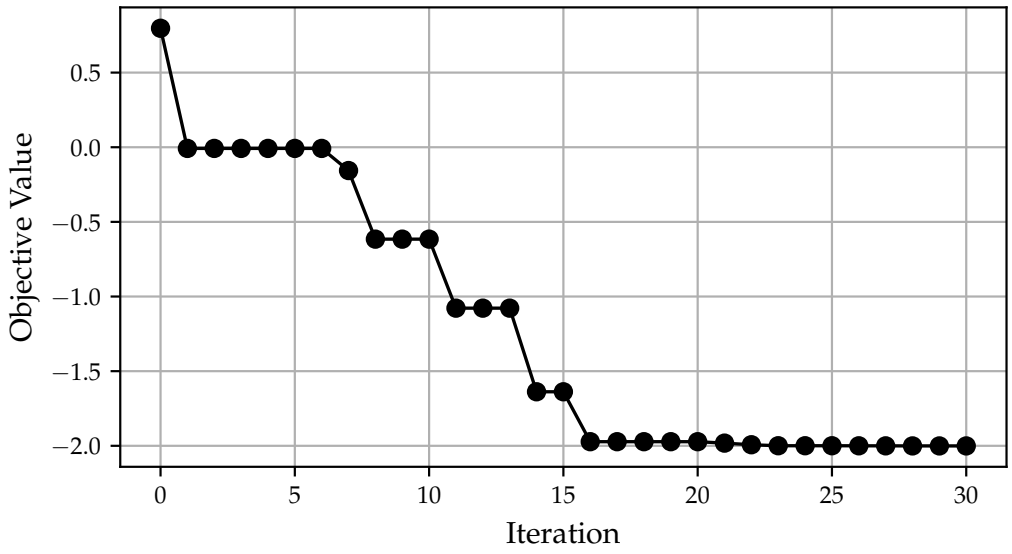


Figure 6.4: Objective function versus iteration for the downhill simplex method on the non-convex system with multiple local minima.

```
            else:                # Shrink
                vertices[1] = x1 + sigma*(x2 - x1)
                vertices[2] = x1 + sigma*(x3 - x1)
        else:                    # Inside contraction
            x_contr = center - rho * (center - x3)
            f_contr = f(x_contr)
            if f_contr < f3:
                vertices[2] = x_contr
            else:                # Shrink
                vertices[1] = x1 + sigma*(x2 - x1)
                vertices[2] = x1 + sigma*(x3 - x1)
    return vertices
```

This code would be called in the following way

```
# Initialize a large simplex in 2D
vertices = np.array([[4.0, 4.0],
    [-4.0, -3.0],
    [-2.0,  4.0],])

# Perform Nelder-Mead updates
num_iterations = 30
for it in range(1, num_iterations+1):
    vertices = nelder_mead_step(vertices)
```

The objective function versus iteration is shown in Fig. 6.4. Figure 6.5 shows the evolution of the vertex in the first sixteen iterations. After an initial search, it finds the global minima and then descends to the best value.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

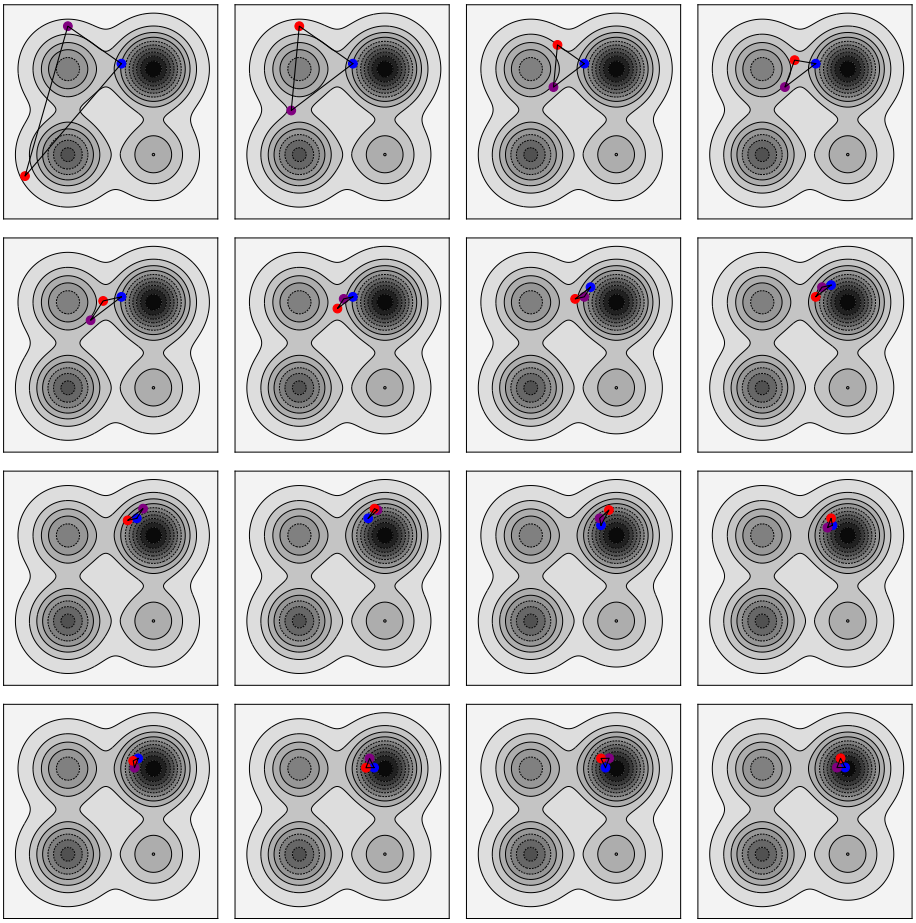


Figure 6.5: Illustration of the downhill simplex method on the non-convex system with multiple local minima. After an initial exploration phase, the simplex eventually finds the global minima and descends.

6.2 Simulated Annealing

Simulated annealing is a stochastic global optimization technique inspired by the metallurgical process of annealing, in which a material is heated and then slowly cooled to decrease defects and obtain a strong crystalline structure [203, 358]; see Fig. 6.6 for an illustration. In metallurgical annealing, the specific cooling schedule makes a tremendous difference in the final properties of the metal (see Fig. 6.7). In optimization, simulated annealing uses a similar idea, leveraging an initially higher "temperature" to explore the solution space and escape local minima, then gradually lowering the temperature to focus on refining solutions near good local minima. In simulated annealing, the temperature refers to the step size of random perturbations that are made to the current solution at each iteration step, so that a higher temperature leads to larger, more exploratory, random fluctuations.

Simulated annealing is a robust, global search algorithm that is suitable for non-convex optimization problems. Moreover, it does not require gradient information and is able to

Copyright © July 15, 2025 Brunton. All Rights Reserved.

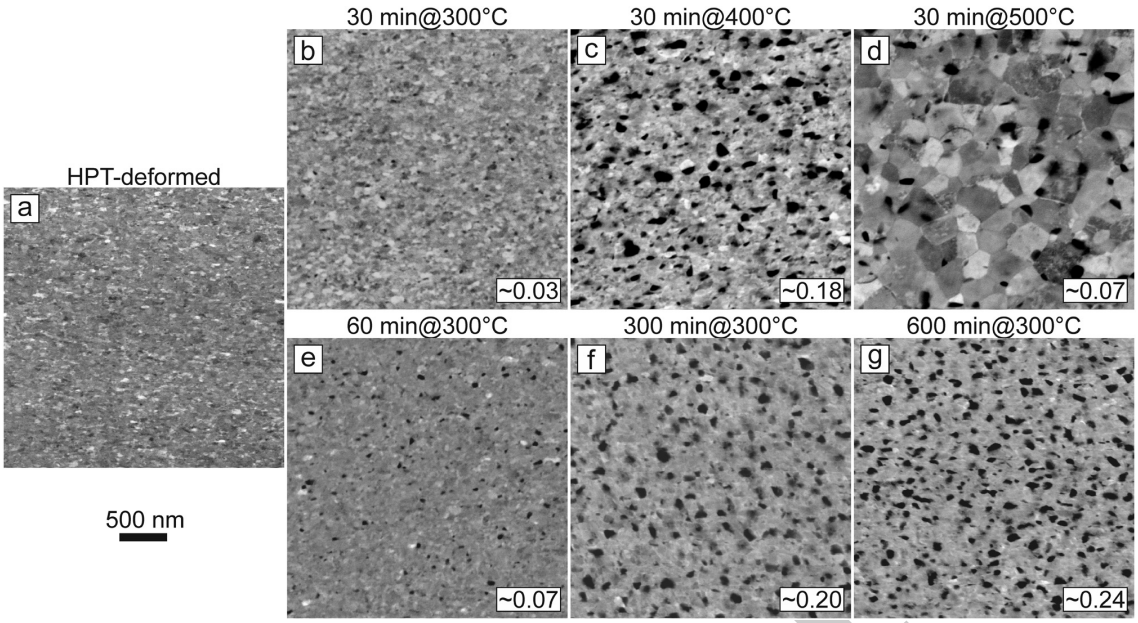


Figure 6.6: Microstructure of Titanium-Niobium allow (45% Nb by weight) formed using high pressure torsion, under various annealing schedules. Left, initial material with no annealing. Upper row: 30 minutes annealing at different temperatures. Lower row: Increasing annealing time at a single temperature. Figure taken from Völker et al. [358].

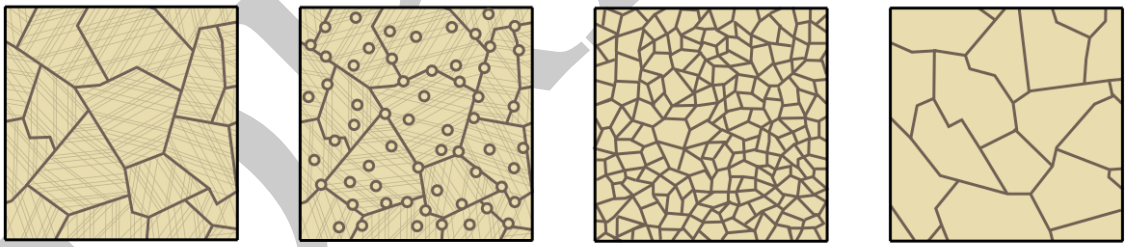


Figure 6.7: Schematic of nucleation and recrystallization of microstructure by annealing. From wikimedia by Daniele Pugliesi.

handle stochastic or noisy optimization problems, as well as problems with discrete, continuous, or mixed variables. In addition, simulated annealing is very simple to implement, as it only involves evaluations of the objective function f and some basic logic. However, the cooling schedule and other hyper-parameters play an essential role, and must be tuned carefully for good performance. As the cooling schedule is itself a function, with infinitely many degrees of freedom, we typically choose from a parameterized class of functions, such as exponential or geometric decay. A schematic of simulated annealing is shown in Fig. 6.8.

There are a number of interesting theoretical results for the convergence of simulated

Copyright © July 15, 2025 Brunton. All Rights Reserved.

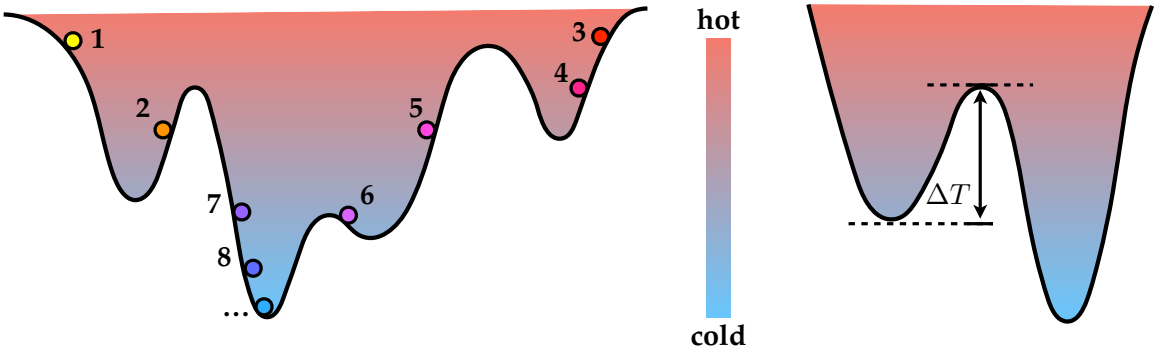


Figure 6.8: Schematic of simulated annealing. (left) We see an example path of the optimization variable in the objective function landscape as it goes from hot (yellow/red) to cold (blue). (right) The current temperature determines how likely it is to jump from one local minimum to another, which is related to the ΔT in the objective function.

annealing, for specific cooling schedules, in the infinite-time limit. However, in practice, while it is designed for global exploration, there is no mathematical guarantee that a global minima will be found in finite time. The global search can also be time-consuming, especially for large or complex problems.

Algorithm

We aim to minimize an objective function $f(\mathbf{x})$. The decision variable $\mathbf{x}$ may be continuous (i.e.., $\mathbf{x} \in \mathbb{R}^n$) or discrete. Simulated annealing iteratively samples a new candidate solution $\mathbf{x}_{k+1}$ in the neighborhood of the current solution $\mathbf{x}_k$. The difference in objective values $\Delta = f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)$ guides the acceptance of $\mathbf{x}_{k+1}$:

1. If $\Delta \leq 0$ then the new solution is better, and we accept it unconditionally.
2. If $\Delta > 0$, the new solution is worse, and we accept it with probability

$$p = \exp\left(-\frac{\Delta}{T}\right),$$

where T is the current *temperature*.

Over time, T is decreased according to the cooling schedule, reducing the acceptance of worse solutions. The algorithm then focuses more on local refinement near good solutions. Typical stopping criteria include a minimum temperature threshold, a maximum number of iterations, or detecting convergence.

Cooling Schedules, Hyperparameters, and Tuning. There are several choices for the cooling schedule. A classical geometric decay for temperature is

$$T_{k+1} = \alpha\, T_k,$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

where $0 < \alpha < 1$ is a constant (e.g., $0.9 \leq \alpha \leq 0.99$). More sophisticated schedules exist, such as logarithmic cooling or adaptive strategies that adjust temperature dynamically based on recent acceptance rates.

There are also some theoretical guarantees for cooling rates T_k that lie between the curves $1/k$ and $1/k^2$. In addition, there are several hyperparameters that must be tuned:

- Initial temperature T_0: T_0 should be high enough to allow frequent acceptance of worse moves at the start to encourage global exploration. Too large of a T_0 will increase the number of iterations required before the temperature is low enough for local exploitation of a local minima.
- Cooling factor α: For geometric decay, this determines how fast T is reduced in each iteration. Too fast, and the algorithm won't have time to explore, and too slow and the algorithm may not converge quickly or to a good minima.
- Neighborhood function or "move operator": This determines how new candidate solutions are generated from the current solution. In continuous domains, one might add small random Gaussian or uniform perturbations. In discrete domains (e.g., traveling salesman), it might be a random permutation swap.
- Stopping criteria: Several choices include reaching a minimum temperature $T_{\min}$, a maximum number of iterations, or no improvement for a number of steps.

Code Example

Here we explore the objective function in Eq. (6.1) with simulated annealing. Figure 6.9 shows the progression of the decision variable, starting from a "hot" exploratory condition in red to the final "cold" nearly optimal condition in blue. It is simple to implement a basic simulated annealing code. The following code tracks the best objective value at each iteration and returns the path of visited points.

```
def simulated_annealing(init_point, n_iter=200, T0=5.0, alpha=0.98,
    step_size=1.0, bounds=(-5,5)):
    current = np.array(init_point, dtype=float)
    current_val = f(current)
    path = [current.copy()]
    temp = T0 # Initial temperature

    for i in range(n_iter):
        # Propose random step
        candidate = current + np.random.randn(2) * step_size
        candidate = np.clip(candidate, bounds[0], bounds[1])
        candidate_val = f(candidate)
        delta = candidate_val - current_val

        # Accept if better or with probability e^(-delta/T)
        if delta < 0:
            current, current_val = candidate, candidate_val
        else:
            if np.random.rand() < np.exp(-delta / temp):
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

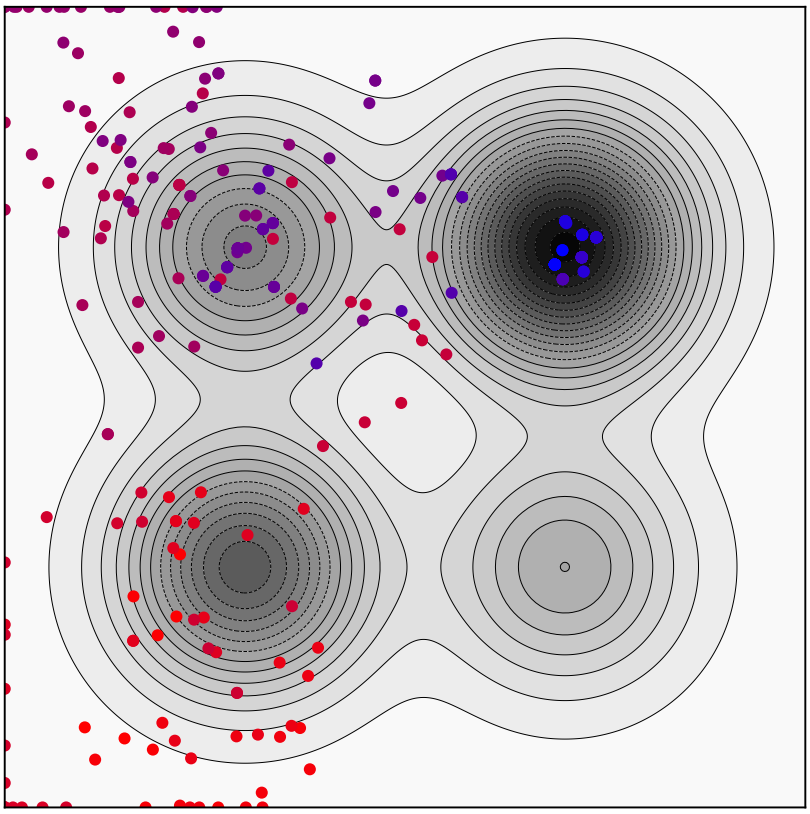


Figure 6.9: Illustration of the simulated annealing process, where initial "hot" points are red, and as they "cool" they become more blue. Eventually the global minimum is found.

```
                current, current_val = candidate, candidate_val

        temp *= alpha # Lower the temperature
        path.append(current.copy()) # Store the path
    return np.array(path)
```

We would then call this function as follows:

```
xIC = np.array([-4.0, -4.0])  # Initial guess (bad)
path = simulated_annealing(
    init_point=xIC,n_iter=200,T0=10.0,alpha=0.98,step_size=1.)
```

Alternatively, it is possible to use a standard package.

```
from scipy.optimize import dual_annealing

# Domain bounds for x,y in [-5, 5]
bounds = [(-5, 5), (-5, 5)]

result = dual_annealing(objective_2d, bounds=bounds, maxiter=1000,
    seed=0)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

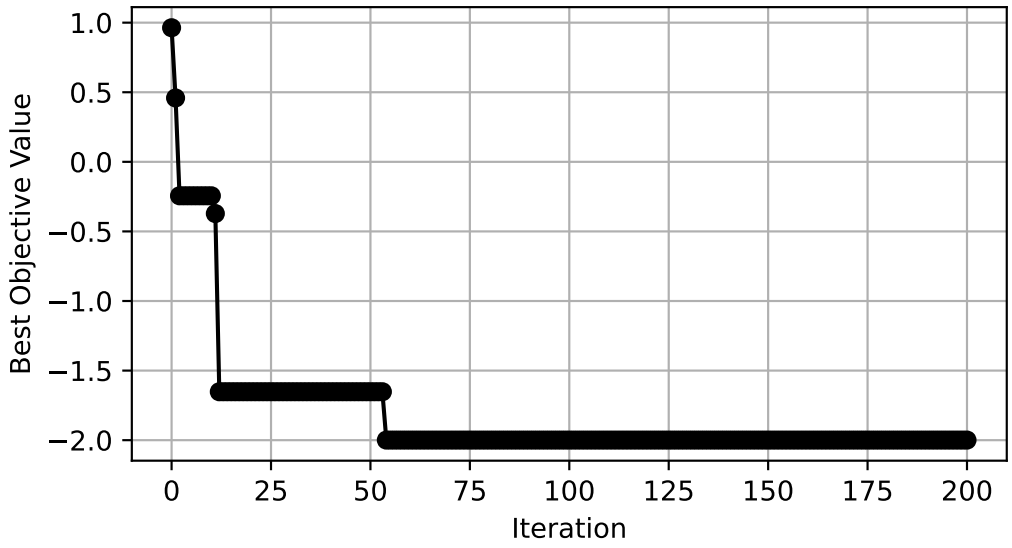


Figure 6.10: The objective function versus iteration for simulated annealing is shown.

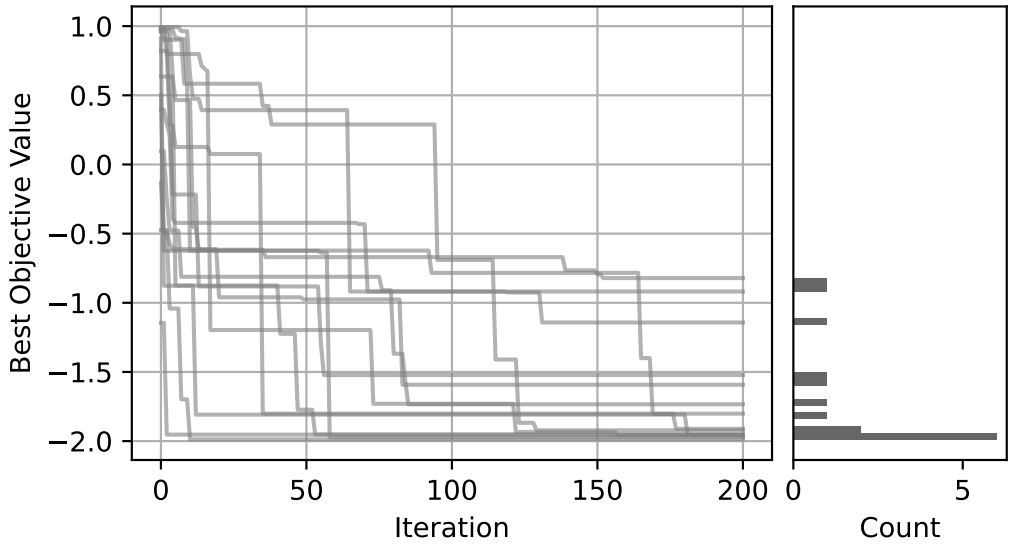


Figure 6.11: Ensemble of multiple simulated annealing paths (left), with a histogram of the final optimized value (right)

The result object has several attributes, including `x` (the best point found), `fun` (the objective function at that point), `nfev` (# of function evaluations), and `nit` (# of iterations).

To access more data internal to each iteration, for example for plotting the optimization path, we may use a callback function. See the code online for an example.

It is also possible to plot the cost per iteration of the annealing process in Fig. 6.10. It is illustrative to repeat this process for an ensemble of many annealing paths, in Fig. 6.11, to see how the distribution evolves with iteration number.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

6.3 Evolutionary Algorithms

Evolutionary algorithms form a broad class of methods for non-convex optimization, based on the principles of biological evolution and survival of the fittest. Evolutionary operations, such as replication, mutation, and exchange of genetic information, are used to iteratively evolve populations of candidate solutions. With a judicious choice of hyperparameters, the performance distribution of this population will improve and individuals in the population may approach globally optimal solutions. Evolutionary algorithms broadly comprise two major categories of methods: genetic algorithms for parameter optimization and genetic programming for function optimization.

Parameter optimization is a simpler problem, where we learn the parameters of a function with known structure. The goal is to find the optimal parameters given some objective function, for example training a neural network fitting $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$ to fit data y, or trying to learn the parameters of a control law with optimal performance.

Function optimization is more challenging, because we are learning both the structure and parameters of a function that optimizes a given objective function. Again, the task may be to fit data with a learned approximation $f(\mathbf{x})$ or learn a control law $u = f(\mathbf{x})$. However, in this case, we will also learn the structure of the function f.

We begin with genetic algorithms, because of their simplicity and broad applicability, and then proceed to the more challenging, but more flexible, class of genetic programming. Genetic programming is often referred to as symbolic regression, because it yields analytic representations of the function being optimized. This is useful for explainable AI [84].

Genetic algorithms

Genetic algorithms (GAa) are among the simplest algorithms for parameter optimization, based on the biological principle of optimization through natural selection and fitness [174, 91, 147]. In GA, a population of many realizations with different parameter values compete to minimize a cost function, and successful parameter values are propagated to future generations through a set of *genetic* rules. The parameters of a system are generally represented by a binary sequence, as shown in Fig. 6.12. As a concrete example, these parameters may represent the three control gains K_P, K_I, and K_D for a PID controller [56]. A number of realizations with different parameter values, called *individuals*, are initialized in a population and their performance is evaluated and compared on a well-defined task. Successful individuals with a lower cost have a higher probability of being selected to advance to the next generation, according to the following genetic operations:

Elitism (optional): A set number of the most fit individuals with the best performance are advanced directly to the next generation.

Replication: An individual is selected to advance to the next generation.

Crossover: Two individuals are selected to exchange a portion of their code and advance to the next generation; crossover exploits and enhances successful strategies.

Mutation: An individual is selected to have some of its code modified with new values; mutation promotes diversity and increases the exploration of parameter space.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

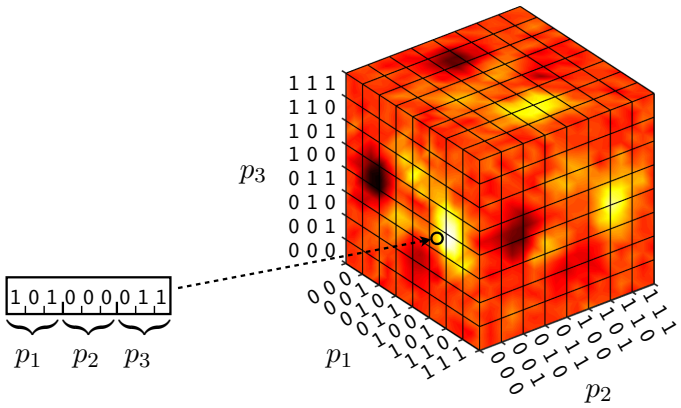


Figure 6.12: Depiction of a parameter cube with three parameters p_1, p_2, and p_3. For example, these three parameters may represent the proportional, integral, and derivative gains in a PID controller. The genetic algorithm represents a given parameter value as a *genetic sequence* that concatenates the various parameters. In this example, the parameters are expressed in binary representation that is scaled so that **000** is the minimum bound and **111** is the upper bound. Color indicates the cost associated with each parameter value.

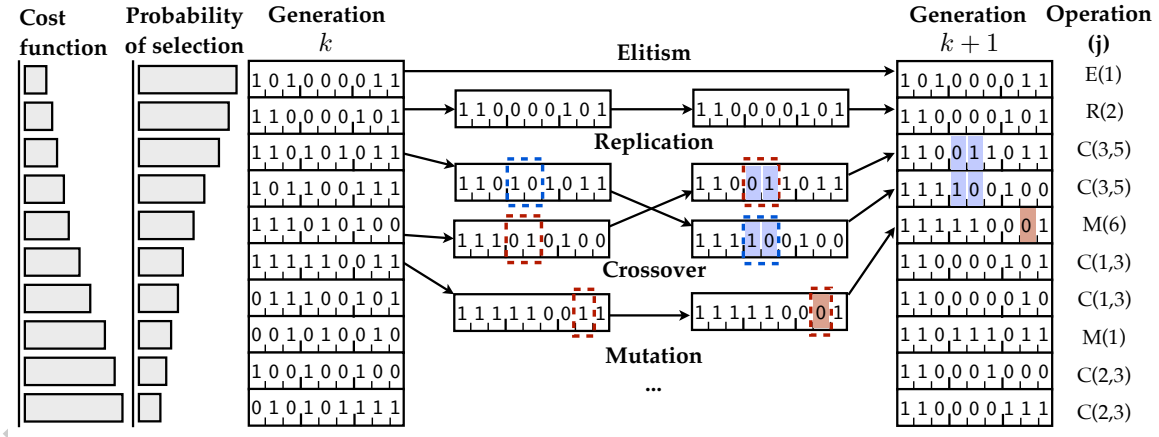


Figure 6.13: Schematic illustrating evolution in a genetic algorithm. Individuals in generation k are each evaluated and ranked based on their cost function, which is inversely proportional to their probability of selection for genetic operations. Individuals are chosen based on this weighted probability to advance to generation $k+1$ using the four operations: elitism, replication, crossover, and mutation. Generation iterations are repeated until the population statistics converges or another stopping criterion is reached.

For replication, crossover, and mutation, individuals are randomly selected to advance to the next generation with the probability of selection increasing with fitness. The genetic operations are illustrated for the PID control example in Fig. 6.13. These generations are evolved until the fitness of the top individuals converges or other stopping criteria are met.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

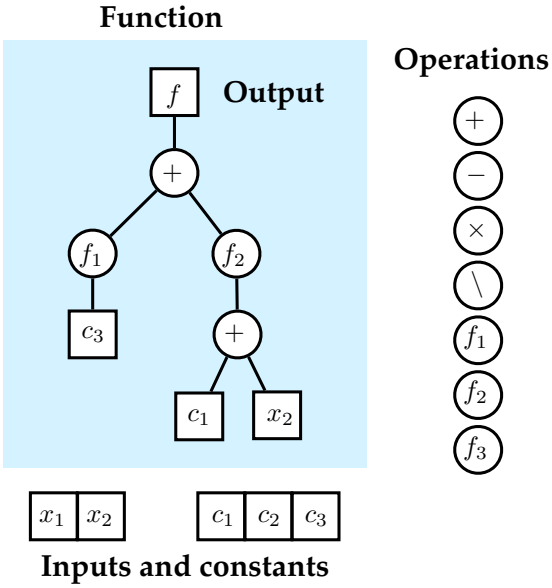


Figure 6.14: Illustration of function tree used to represent the function $f(\mathbf{x})$.

Genetic algorithms are generally used to find nearly globally optimal parameter values, as they are capable of exploring and exploiting local wells in the cost function. GA provides a middle ground between a brute-force search and a convex optimization, and is an alternative to expensive Monte Carlo sampling, which does not scale to high-dimensional parameter spaces. However, there is no guarantee that genetic algorithms will converge to a globally optimal solution. There are also a number of hyper-parameters that may affect performance, including the size of the populations, number of generations, and relative selection rates of the various genetic operations.

Genetic programming

Genetic programming (GP) [208, 207] is a powerful generalization of genetic algorithms that simultaneously optimizes both the structure and parameters of an input–output map (i.e., a function), as shown in Fig. 6.14. More recently, genetic programming has been used extensively to learn dynamical systems [43, 313, 84] or control laws [118].

The function tree representation in GP is quite flexible, enabling the encoding of complex functions of the input data $\mathbf{x}$ through a recursive tree structure. Each branch is a signal, and the merging points are mathematical operations. Inputs and constants are the leaves, and the overall function output $f(\mathbf{x})$ is the root. The genetic operations of crossover, mutation, and replication are shown schematically in Fig. 6.15.

In general, although evolutionary algorithms are extremely general and powerful, there are very few theoretical guarantees on convergence and stability. That said, they have had a resurgence of interest recently as a technique for learning human-interpretable and explainable representations of machine learning models [85].

Copyright © July 15, 2025 Brunton. All Rights Reserved.

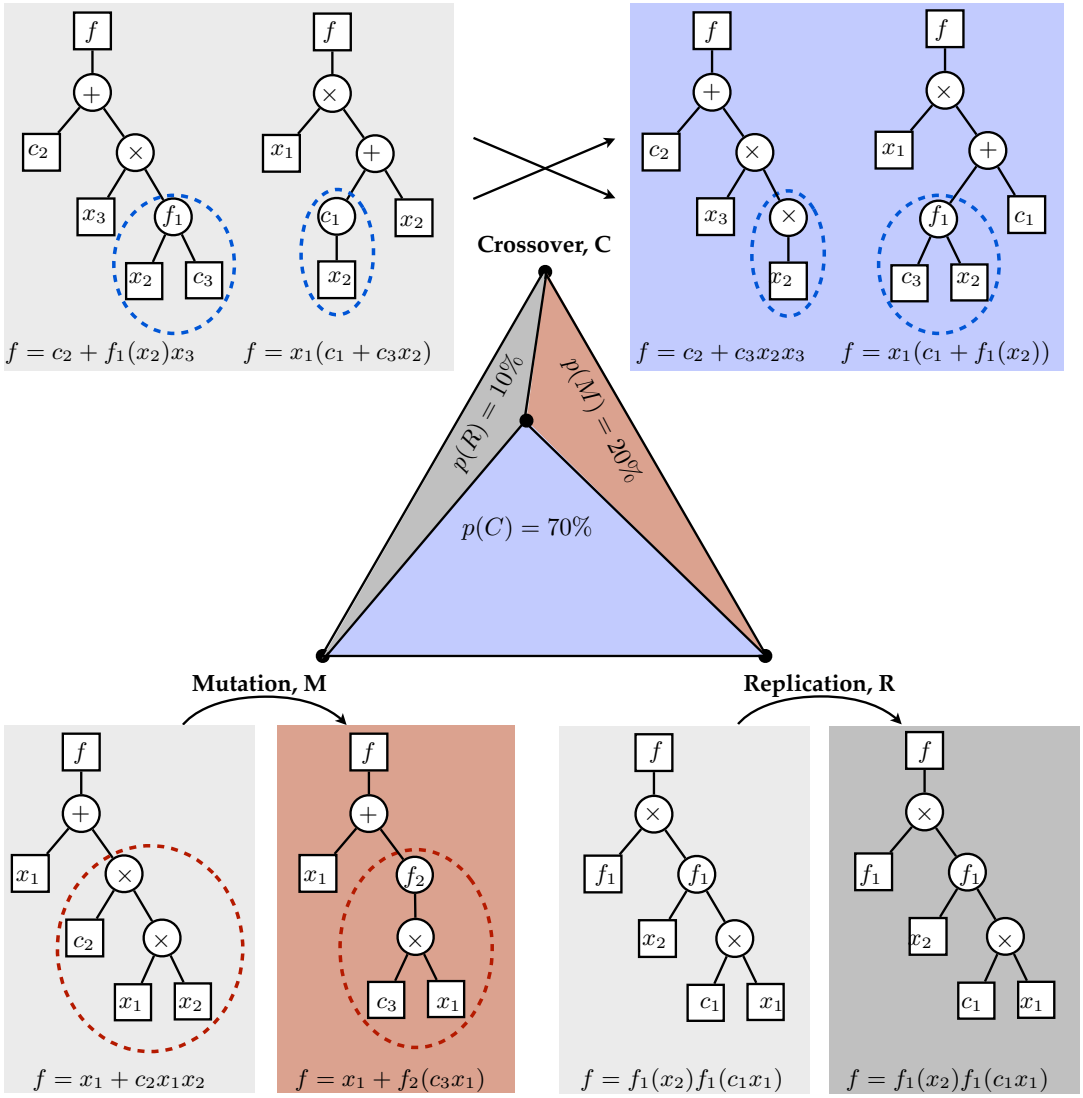


Figure 6.15: Genetic operations used to advance function trees in genetic programming.

Python Example with PySR

Here we explore a simple example of symbolic regression using the `PySR` package [84]. The following code generates data from the Lorenz system and learns a differential equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, shown in Fig. 6.16. We will revisit this in as an inverse problem in Chapter 8.

```
from pysr import PySRRegressor
from sympy import lambdify, symbols

# Generate data from Lorenz system
x0 = [5, 5, 5] # Initial state
sol_train = solve_ivp(lorenz, (0, 2), x0, t_eval=t_train)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

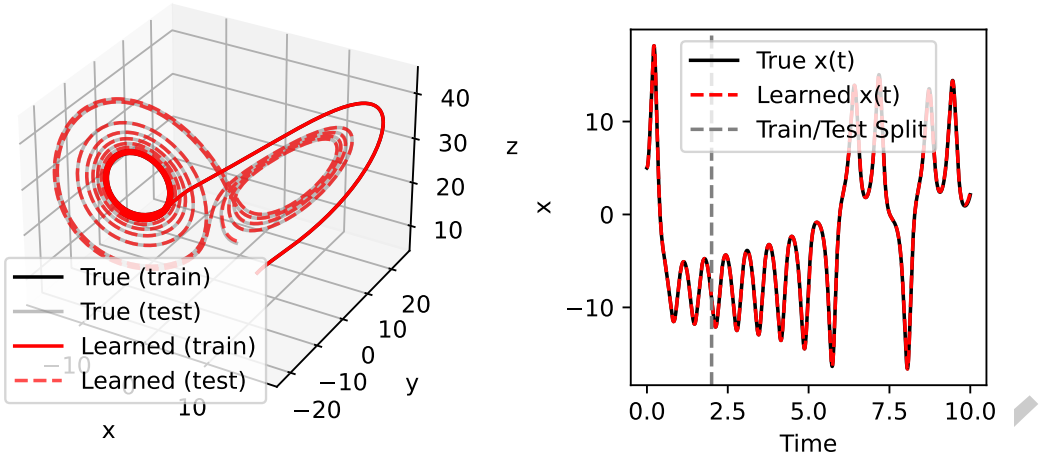


Figure 6.16: Output of PySR on Lorenz system. The model is trained on the first 2 time units and tested on the next 8. The model performance is in excellent agreement.

```
X_train = sol_train.y.T
dXdt_train = np.gradient(X_train, dt, axis=0) # compute derivatives

# --- Train PySR models for dx/dt, dy/dt, dz/dt ---
models = []
rhs_exprs = []
variables = symbols("x0 x1 x2")
for i, label in enumerate(["dx/dt", "dy/dt", "dz/dt"]):
    model = PySRRegressor(
        model_selection="best",
        binary_operators=["+", "-", "*", "/"],
        unary_operators=["sin", "cos", "exp"],
        loss="loss(x, y) = (x - y)^2",
        niterations=100, maxsize=10, populations=10)
    model.fit(X_train, dXdt_train[:, i])
    models.append(model)
    rhs_exprs.append(lambdify(variables, model.get_best().
        sympy_format, modules=["numpy"]))

# --- Define learned system using all three expressions ---
def learned_lorenz(t, state):
    x, y, z = state
    dx = rhs_exprs[0](x, y, z)
    dy = rhs_exprs[1](x, y, z)
    dz = rhs_exprs[2](x, y, z)
    return [dx, dy, dz]

# Simulate full learned model over [0, 2]
sol_learned = solve_ivp(learned_lorenz, (0, 2), x0, t_eval=t_full)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

6.4 Distributed and Swarm Optimization

With the rise of big data, parallel computing, and interconnected systems, optimization strategies that harness the power of multiple agents or sub-problems in a distributed manner have become essential. Here we will discuss two main paradigms: distributed optimization and swarm optimization. *Distributed optimization* aims to solve large-scale problems by decomposing them into subproblems that run in parallel, synchronizing partial solutions through communication protocols [47, 259]. In contrast, *swarm optimization* takes inspiration from natural phenomena, such as birds flocking, fish schooling, and ants foraging, to design optimization algorithms. Particle swarm optimization (PSO) [199], ant colony optimization (ACO) [108], and related *swarm intelligence* methods embody this approach.

Historically, distributed optimization can be traced back to early studies on decentralized control and parallel processing in the 1970s and 1980s [350]. Over time, as networking technologies matured, applications of distributed optimization proliferated in sensor networks, communication systems, and large-scale machine learning (e.g., federated learning). Meanwhile, swarm intelligence took shape in the 1990s with foundational algorithms such as *particle swarm optimization* by Kennedy and Eberhart [199] and *ant colony optimization* by Dorigo and co-authors [108]. These bio-inspired algorithms emphasize simplicity, adaptability, and the emergent behavior arising from the coordination of many simple agents.

Both distributed and swarm approaches can be viewed as leveraging the idea of *collective decision-making* to handle large, complex problems. Distributed optimization typically focuses on rigorous mathematical decomposition, while swarm methods favor iterative search processes guided by heuristics and emergent coordination.

Distributed Optimization

A common scenario in distributed optimization arises when one has a large-scale objective

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) = \sum_{j=1}^{m} f_j(\mathbf{x}),$$

where each $f_j(\mathbf{x})$ is associated with a different agent (or node) $j \in \{1, \ldots, m\}$ in a network [259]. Each agent knows only its local function $f_j(\mathbf{x})$ and is responsible for processing or storing part of the data. The agents must collaboratively find a global minimizer $\mathbf{x}^*$.

Agents communicate over a network, exchanging local information, such as gradients, subgradients, or partial solutions. A fundamental concept is *consensus*, where all agents agree on a common solution vector, or feasible point. Methods like the *consensus algorithm* iteratively average local solutions to converge to a consistent global solution.

The alternating direction method of multipliers (ADMM) is one popular choice to decompose the large problem into subproblems, each solved locally, with updates to ensure global consistency [47]. We will discuss ADMM in much more detail in Sec. **??**.

Distributed optimization has several advantages, including scalability, privacy, and efficient parallelism. Because this approach decomposes large problems across multiple

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

machines or agents, it may leverage parallel computation and reduces communication overhead if well designed. Another key advantage is that data can remain local to each node, which is important for privacy and ownership in sensitive applications (e.g., federated learning).

However, these algorithms are relatively complex to implement, requiring careful handling of asynchrony, message passing, and partial updates. Convergence may also be slower than centralized solvers if the communication or coordination overhead is large. Finally convergence depends on communication reliability and network topology.

Particle Swarm Optimization (PSO)

Swarm optimization methods view a population of agents (e.g., particles, ants, bees) each maintaining a candidate solution. The overall solution quality emerges from how these agents *interact* and *update* themselves based on shared experience or local heuristics [199, 41].

Swarm optimization, in contrast to distributed optimization, is relatively easy to implement and is versatile for many black-box or non-smooth problems. It relies on global search heuristics (e.g., swarm intelligence) to escape local minima in non-convex problems. Like distributed optimization, it is parallelizable, as swarm updates can be computed in parallel for each agent.

There are no exact guarantees for swarm optimization, as the heuristics can fail to find global optima or converge slowly. The performance is also sensitive to hyper-parameter tuning. Finally, the algorithm scaling is not always favorable, as it may require large swarms and many iterations to converge in high-dimensional space.

In PSO, each particle has a position $\mathbf{x}_j$ (candidate solution) and velocity $\mathbf{v}_j$ in the search space. The algorithm updates these based on the best known position the particle has individually found and the best known position found by any particle:

$$\begin{aligned}\mathbf{v}_j^{(k+1)} &= \omega \mathbf{v}_j^{(k)} + c_1 r_1 \left(\mathbf{x}_j^{(k)*} - \mathbf{x}_j^{(k)}\right) + c_2 r_2 \left(\mathbf{x}^{(k)*} - \mathbf{x}_j^{(k)}\right), \\ \mathbf{x}_j^{(k+1)} &= \mathbf{x}_j^{(k)} + \mathbf{v}_j^{(k+1)},\end{aligned}$$

where $\mathbf{x}_j^{(k)*}$ is particle j's best historical position, $\mathbf{x}^{(k)*}$ is the global best across all particles, and r_1, r_2 are random scalars in $[0, 1]$. The parameters ω, c_1, c_2 control the balance between exploration and exploitation.

Code Example

Here we revisit the example objective function in Eq. (6.1) with PSO. The following code will initialize a swarm of particles and iterate through the PSO algorithm. Figure 6.17 shows the progression of the swarm versus iteration, with early iterations being plotted in red and later iterations plotted in blue. The swarm converges toward the optimum value in relatively few iterations, shown in Fig. 6.18.

The code to run the PSO on this example is as follows.

```
def particle_swarm_optimization(n_particles=30, n_iter=20, w=0.5, c1
    =1.5, c2=1.5, bounds=(-5, 5)):
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

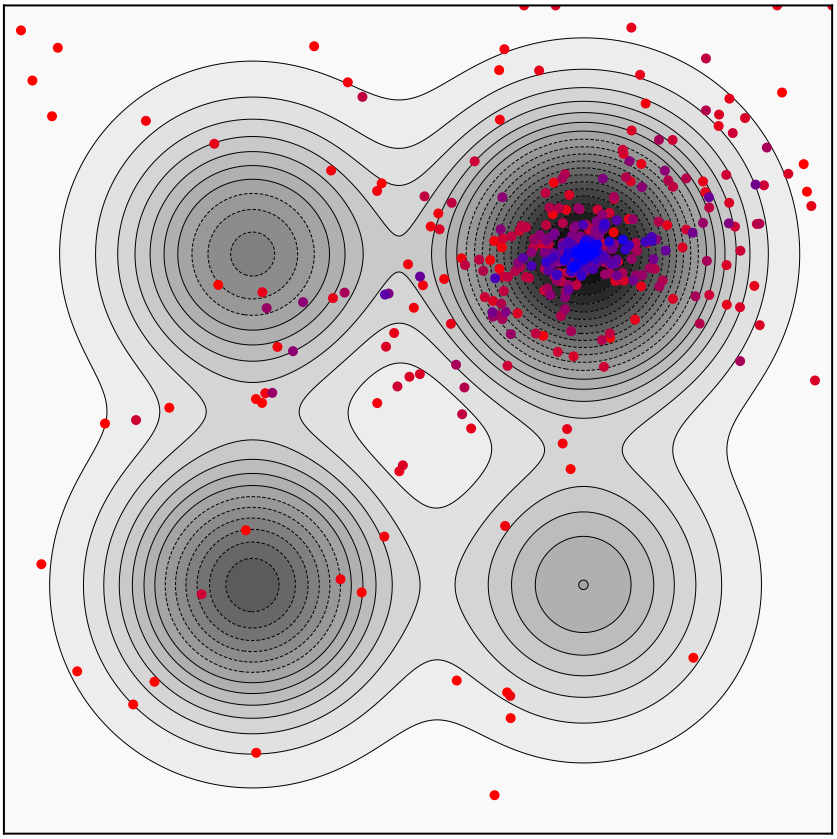


Figure 6.17: Illustration of the particle swarm optimization, where the initial swarm locations are plotted in red, and as they optimize they become more blue. Eventually the global minimum is found.

```
    positions = np.random.uniform(bounds[0], bounds[1], size=(
        n_particles, 2))
    velocities = np.zeros_like(positions)
    p_best = positions.copy()
    p_best_vals = np.apply_along_axis(lambda p: f(*p), 1, p_best)
    g_best = p_best[np.argmin(p_best_vals)]
    history = [positions.copy()]

    for _ in range(n_iter):
        r1 = np.random.rand(n_particles, 1)
        r2 = np.random.rand(n_particles, 1)
        velocities = w * velocities + c1 * r1 * (p_best - positions)
            + c2 * r2 * (g_best - positions)
        positions = np.clip(positions + velocities, bounds[0], bounds
            [1])
        vals = np.apply_along_axis(lambda p: f(*p), 1, positions)

        improved = vals < p_best_vals
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

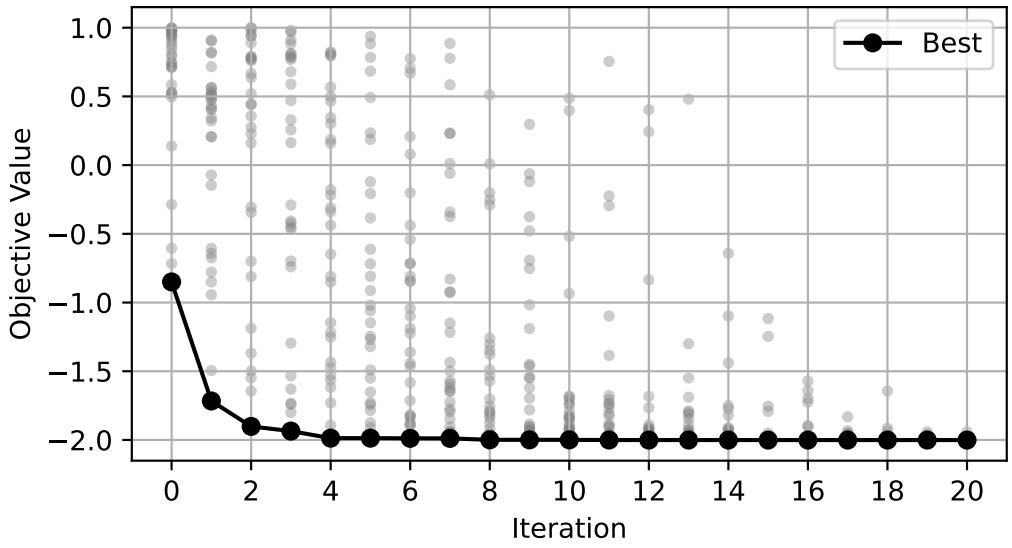


Figure 6.18: The objective function versus iteration for particle swarm optimization is shown.

```
            p_best[improved] = positions[improved]
            p_best_vals[improved] = vals[improved]
            g_best = p_best[np.argmin(p_best_vals)]
            history.append(positions.copy())

    return np.array(history), g_best
```

Ant Colony Optimization (ACO)

ACO, inspired by how ants deposit and follow pheromone trails, represents solutions as paths in a graph. Ants incrementally construct solutions by probabilistically choosing edges based on:

$$P_{i,j} = \frac{\tau_{i,j}^{\alpha} \, \eta_{i,j}^{\beta}}{\sum_{k \in \mathcal{N}_i} \tau_{i,k}^{\alpha} \, \eta_{i,k}^{\beta}},$$

where $\tau_{i,j}$ is the pheromone level on edge (i, j), $\eta_{i,j}$ is a heuristic measure (e.g., inverse of distance), and α, β tune the relative influence of pheromone vs. heuristic. After constructing solutions, ants *update* pheromones on edges that yielded better solutions, reinforcing promising paths [108].

Both PSO and ACO rely on simple local rules at the agent level. Remarkably, the collective system can converge toward high-quality solutions, even in complex, high-dimensional landscapes. Convergence proofs typically rely on assumptions about problem structure and parameter tuning.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

6.5 Regularization, Relaxation, and Splitting

Real-world optimization problems are often ill-posed or numerically unstable. Regularization and relaxation are two common strategies to mitigate these challenges. *Regularization* modifies an objective function to improve stability, interpretability, or generalization. *Relaxation* replaces non-convex terms in the constraints or the objective function with convex approximations that are amenable to computations. Splitting methods are effective ways of solving regularized, relaxed optimization problems. Instead of satisfying all constraints and objectives with the same decision variable, splitting introduces new decision variables for each term, adding a final regularization term to tightly couple these independent optimization problems. This makes stiff optimization problems more tractable.

In this section, we will first introduce common regularization techniques, many of which we have discussed before in Chapter 4, and establish a common notation that is consistent with Chapter 5. We then discuss how non-convex terms, such as the $\|\cdot\|_0$ and $\text{rank}(\cdot)$, may be *relaxed* into convex approximations that are much easier to optimize. Next we introduce proximal algorithms, which are essential when we have non-smooth terms in our objective function, such as $\|\cdot\|_1$, the nuclear norm $\|\cdot\|_*$, the total variation used in image processing in Chapter 8, and other nonsmooth regularizers used in neural network training [155] (see Chapter 10). When we have non-smooth terms, simple gradient descent is no longer reliable, and we instead use the more general proximal gradient. Splitting methods are another broad class of algorithms that allow us to more flexibly solve stiff optimization problems with multiple terms in the loss function by *splitting* these into independent optimization problems and then linking the solutions. Proximal algorithms and splitting methods culminate in two main algorithms discussed here: the sparse, regularized, relaxed regression (SR3), and the alternating direction method of multipliers (ADMM), which is closely related to Lagrange multipliers and primal-dual methods from Chapter 5. We demonstrate this approach on a challenging non-convex optimization problem in image science, the so-called robust principal component analysis (RPCA).

These concepts will be used in the later chapters on inverse problems, control theory, and machine learning, where relaxation and regularization are essential. When combined, regularization and relaxation provide a powerful framework to systematically handle large-scale, ill-conditioned, or combinatorially complex problems. Machine learning examples include sparse regression (LASSO), ridge regression, robust fitting, and training neural networks under constraints, such as derived from physics. In signal processing and image analysis, we use total variation (TV) regularization to denoise, wavelet-based sparsity constraints and sparse regularization for image reconstruction given sparse measurements, and nuclear norm regularization for matrix completion. We have already seen examples of relaxing integer constraints to a continuous linear program. Finally, in robust control and PDE-constrained optimization, small changes in model parameters can produce unstable or ill-posed results, necessitating regularization and relaxation.

Regularization Techniques

We have already seen several examples of regularization, including the ridge regression, LASSO, and elastic net. These approaches add different regularization terms to promote

Copyright © July 15, 2025 Brunton. All Rights Reserved.

particular solutions to ill-posed or under-determined least-squares regression problems. Here we revisit these ideas, generalizing our notation to handle generic regularization.

A generic form for a regularized optimization problem is given by

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) + \lambda R(\mathbf{x}) \tag{6.2}$$

where $f(\mathbf{x})$ is the original objective function and $R(\mathbf{x})$ is the regularization term[1]. Note the close relationship to the augmented Lagrangian from Chapter 5, where $R(\mathbf{x}) = 0$ is a constraint equation. Thus, regularization may be used to promote or enforce constraints.

Here we review a few key regularization terms that are commonly used.

Tikhonov (Ridge) Regularization. For a (potentially ill-posed) least-squares problem

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2,$$

adding a term $\lambda\|\mathbf{x}\|^2$ yields:

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda\|\mathbf{x}\|^2.$$

The regularization $R(\mathbf{x}) = \|\mathbf{x}\|^2$ penalizes large values of $\mathbf{x}$, stabilizing ill-conditioned regression problems. We have already discussed this in Chapter 4. Tikhonov's penalty $R(\mathbf{x}) = \|\mathbf{x}\|^2$ was developed into the ridge regression by Hoerl and Kennard [340, 172].

LASSO Regularization. The convex, non-smooth ℓ_1-norm promotes sparsity in $\mathbf{x}$:

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda\|\mathbf{x}\|_1.$$

The ℓ_1 norm was introduced as a penalty in 1969 [278], with specialized algorithms in 1973 [81], and as a robust loss in geophysics in 1973 [79]. More generally, group-sparsity, nuclear norm, and total-variation regularizers all promote specific structure in solutions.

Elastic Net. A combination of ℓ_2 and ℓ_1 terms are used to combine stability with sparsity:

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2\|\mathbf{x}\|^2.$$

This *elastic net* approach helps mitigate some limitations of purely ℓ_1-based methods (e.g., degeneracy when the number of features is very large compared to samples) [387].

Non-convex Regularizers. Non-convex regularizers, such as the ℓ_0 norm and the smoothly clipped absolute deviation (SCAD), can recover sparser or lower-rank solutions than a convex surrogate. However, the resulting optimization problems involve an NP-hard combinatorial search that doesn't scale favorably with problem size. In practice we must relax these terms with a convex proxy, such as relaxing the non-convex $R(\mathbf{x}) = \|\mathbf{x}\|_0$ to the convex ℓ_1 norm $R(\mathbf{x}) = \|\mathbf{x}\|_1$. We may also use splitting methods to make them more tractable and well-conditioned. We discuss these approaches next.

[1] We sometimes add a matrix $\mathbf{C}$ in the regularization to be slightly more general:

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) + \lambda R(\mathbf{C}\mathbf{x}). \tag{6.3}$$

This doesn't change the algorithms discussed here, and we will often omit it.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Convex Relaxation

Convex relaxation is a powerful strategy in non-convex optimization. When an original problem is difficult to solve due to discrete, combinatorial, or non-convex structure, we may construct a surrogate problem by relaxing the non-convex constraints or objective to a convex formulation. The relaxed problem can be solved efficiently using convex optimization techniques, and in some cases, the solution to the relaxation coincides with the solution of the original problem.

Many discrete or non-convex problems (e.g., integer programming, rank constraints, ℓ_0 sparsity) can be relaxed by enlarging the feasible set to a convex superset. Here we will discuss a few key relaxations that are commonly used.

Sparsity with ℓ_0. The ℓ_0 norm, which counts the number of non-zero elements of a vector, is non-convex and can be replaced by its tightest convex envelope, the ℓ_1 norm:

$$\min_{\mathbf{x}} \quad \|\mathbf{x}\|_0 \quad \Rightarrow \quad \min_{\mathbf{x}} \quad \|\mathbf{x}\|_1.$$

This relaxation can be solved efficiently, for example using splitting methods. The ℓ_0 norm is also known as the cardinality.

Rank constraints. Problems involving rank constraints (e.g., $\text{rank}(\mathbf{X}) \leq r$) are highly non-convex. In many matrix optimization problems, such as the robust principal component analysis, we will try to minimize the rank of a matrix $\mathbf{X}$, $R(\mathbf{X}) = \text{rank}(\mathbf{X})$, which counts the number of linearly independent rows or columns of $\mathbf{X}$. This non-convex objective function is challenging to even compute, let alone optimize over. However, we may relax this to the nuclear norm $R(\mathbf{X}) = \|\mathbf{X}\|_* = \sum_i \sigma_i$, which is the sum of the singular values of $\mathbf{X}$. Typically, low-rank matrices $\mathbf{X}$ will also have small nuclear norms, since linearly dependent rows or columns have zero singular values. This is a convex relaxation that is tractable with modern convex optimization algorithms such as ADMM.

Integer programming. Problems involving binary or integer variables (e.g., $x_i \in \{0, 1\}$ or $x_i \in \mathbb{Z}$) can be relaxed by allowing the variables to lie in a continuous interval (e.g., $x_i \in [0, 1]$). This yields a convex feasible set and a continuous optimization problem.

For a typical example, consider the following

$$\min_{\mathbf{x} \in \{0,1\}^n} f(\mathbf{x}) \quad \Longrightarrow \quad \min_{\mathbf{x} \in [0,1]^n} f(\mathbf{x}).$$

We have replaced the integer set $\{0, 1\}$ with the closed interval $[0, 1]$, which is its convex hull.

Polynomial optimization. Polynomial constraints are also quite challenging and often non-convex. These problems can often be approached by lifting the variables to form semidefinite constraints. For instance, terms $\mathbf{x}\mathbf{x}^T$ might be replaced by a new matrix variable $\mathbf{X} \succeq 0$ with $X_{ij} = x_i x_j$. Imposing the moment constraint $\mathbf{X} - \mathbf{x}\mathbf{x}^T \succeq 0$ results in a semidefinite constraint. More generally, an even-degree polynomial feasibility condition can be written as a sum-of-squares (SOS) certificate, which is equivalent to requiring

Copyright © July 15, 2025 Brunton. All Rights Reserved.

that an associated moment matrix be positive semidefinite. The resulting problem is a semidefinite-program (SDP) relaxation, which is convex and solvable by off-the-shelf SDP solvers. If the solution to the SDP has rank 1 or low rank, one can extract a feasible point for the original non-convex problem; this principle underlies the Lasserre SOS hierarchy and many modern polynomial-optimization techniques [215].

Duality. Convex duality offers another approach to relaxation. Rather than modifying the primal formulation, we construct a dual problem whose solution provides a lower bound on the original (primal) objective. This is particularly useful when the dual has a simpler or more structured form than the primal.

In minimizing a nonlinear or combinatorial function $f(\mathbf{x})$ subject to $\mathbf{x} \in \mathcal{F}$, we can write the Lagrangian with additional constraints that broaden $\mathcal{F}$ to a simpler set $\widetilde{\mathcal{F}} \supset \mathcal{F}$. Solving the relaxed dual problem yields a lower bound on the primal optimum [262]. If primal and dual solutions coincide, we confirm optimality. Otherwise, we refine $\widetilde{\mathcal{F}}$ or add cutting planes.

In many cases, introducing dual variables for hard constraints allows us to reformulate the problem using Lagrangian relaxation. We relax the constraints by moving them into the objective with penalties:

$$\min_{\mathbf{x}} \max_{\boldsymbol{\mu} \geq 0} \left\{ f(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}) \right\}.$$

This is the basis for methods such as dual ascent, subgradient methods, and dual decomposition. These are particularly effective for structured problems with separable constraints.

Strong duality guarantees (e.g., via Slater's condition) ensure that the optimal value of the dual problem equals the primal, so solving the dual suffices. In other cases, solving the dual yields a useful bound or approximation.

Proximal Algorithms

When optimizing problems with nonsmooth regularizers, such as the ℓ_1 norm in LASSO or total variation penalties, standard gradient descent no longer applies due to the non-differentiability of the objective. The *proximal gradient method* extends gradient descent to handle such composite optimization problems of the form

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) + R(\mathbf{x}), \tag{6.4}$$

where $f(\mathbf{x})$ is smooth and convex with Lipschitz continuous gradient, and $R(\mathbf{x})$ is convex but possibly nonsmooth.

The proximal gradient update rule is given by

$$\mathbf{x}_{k+1} = \operatorname{prox}_{R,\gamma} \left(\mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k) \right), \tag{6.5}$$

where $\gamma > 0$ is a step size, and the proximal operator of R is defined as

$$\operatorname{prox}_{R,\gamma}(\mathbf{z}) := \arg\min_{\mathbf{x}} \left(\frac{1}{2} \|\mathbf{x} - \mathbf{z}\|^2 + \gamma R(\mathbf{x}) \right). \tag{6.6}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

This algorithm alternates between a gradient step on the smooth part f and a proximal step on the nonsmooth regularizer R. In the case of $R(\mathbf{x}) = \lambda\|\mathbf{x}\|_1$, the proximal step corresponds to *soft thresholding*, and the method reduces to the classical iterative shrinkage-thresholding algorithm (ISTA). Accelerated variants, such as FISTA [25], improve convergence from $\mathcal{O}(1/k)$ to $\mathcal{O}(1/k^2)$ for convex problems, by combining ISTA with Nesterov's accelerated gradient method from Section 2.2, making these methods effective in large-scale and high-dimensional settings. The proximal gradient method is particularly useful in convex relaxations of sparse or low-rank models, such as ℓ_1-regularized least squares (LASSO), total variation denoising, and nuclear norm minimization for low-rank matrix recovery. These will be explored throughout this section.

Splitting Methods

In splitting methods, one decomposes a problem with multiple terms in the objective function, introducing extra variables to handle each term separately, and then coupling them with an additional term.

In general, the idea is to transform a problem of the form

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) + \lambda R(\mathbf{x}) \tag{6.7}$$

into the form

$$\min_{\mathbf{x},\mathbf{z}} \quad f(\mathbf{x}) + \lambda R(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}\|^2. \tag{6.8}$$

Again, the regularization term $\lambda R(\mathbf{x})$ could be used to promote or enforce constraints, as discussed in Chapter 5. Often we have a smooth loss $f(\mathbf{x})$ and regularize with a, possibly nonsmooth, term $R(\mathbf{x})$ (e.g., ℓ_1, nuclear norm, etc.). We will often include a matrix $\mathbf{A}$ in the regularizer, $R(\mathbf{A}\mathbf{x})$, to encode structural transformations in the problem. For example, $\mathbf{A}$ might represent a discrete gradient operator in imaging problems, a selection or masking matrix in sparse recovery, or a basis transformation, such as a wavelet or Fourier transform. This structure allows splitting methods, such as the ADMM framework, to efficiently decouple and handle complex composite terms like $R(\mathbf{A}\mathbf{x})$.

Many problems can be written in this form. For example, the least-squares regression problem with objective function $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$ can be written

$$\min_{\mathbf{x},\mathbf{z}} \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda R(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}\|^2. \tag{6.9}$$

Occasionally, algorithms will add an additional matrix $\mathbf{C}$ in the last term, $\frac{\rho}{2}\|\mathbf{C}\mathbf{x} - \mathbf{z}\|^2$, but for simplicity we will keep $\mathbf{C} = \mathbf{I}$. The hyperparameter ρ controls both the strength of the improvements to the geometry/regularity of the relaxed problem relative to the original and the fidelity of the relaxed problem to the original.

The LASSO regression uses $R(\mathbf{z}) = \|\mathbf{z}\|_1$, and ridge regression uses $R(\mathbf{z}) = \|\mathbf{z}\|_2^2$. It is possible to incorporate non-smooth and non-convex norms, such as ℓ_p norms with $p < 1$, via proximal gradients and relaxation, respectively.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

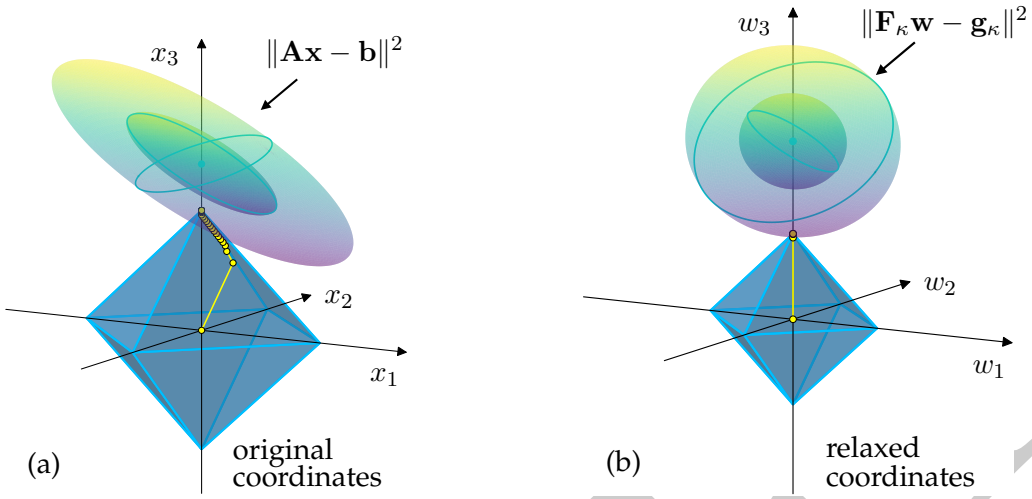


Figure 6.19: (a) Level sets (green ellipses) of the quadratic part of LASSO and corresponding path of prox-gradient to the solution (40 iterations) in $\mathbf{x}$-coordinates. (b) Level sets (green spheres) of the quadratic part of the SR3 loss and corresponding SR3 solution path (2 iterations) in relaxed coordinates $\mathbf{w}$. Blue octahedra show the ℓ_1 ball in each set of coordinates. SR3 decreases the singular values of $\mathbf{F}_\kappa$ relative to those of $\mathbf{A}$ with a weaker effect on the small ones, 'squashing' the level sets into approximate spheres, accelerating convergence, and improving performance. Figure reproduced from Zheng et al. [383].

Sparse, relaxed, regularized regression. Sparse, relaxed, regularized regression (SR3) [383] is a powerful framework for handling non-smooth and non-convex sparse optimization problems in a fast and accurate way. Sparse optimization is ubiquitous, as we will see later, and SR3 splits the sparsity and accuracy requirements between the $\mathbf{z}$ and $\mathbf{x}$ variables. The overarching algorithm iterates between solving the subproblems in $\mathbf{x}$ and $\mathbf{z}$ individually, and then updating the dual variable $\mathbf{u}$ that penalize or enforce $\mathbf{x}-\mathbf{z}=0$. It is possible to partially minimize $\mathbf{x}$ to obtain a function entirely in $\mathbf{z}$, which has been shown to have nearly spherical level sets, as shown schematically in Fig. 6.19 in Zheng et al. [383]. This is in contrast to the highly eccentric elliptical level sets of $\|\mathbf{Ax}-\mathbf{b}\|^2$.

For convex problems, the SR3 formulation is part of a larger class of problems [23], where it is possible to use alternating minimization algorithms [2]. The original alternating projection methods go back to the 1950s by Von Neumann [360] and Cheney and Goldstein [77].

As a concrete example of the improvement using SR3, we consider the LASSO problem, investigated in Zheng et al. [383]. It was shown in Su et a. [331] that when performing LASSO, false discoveries occur early in the optimization path that cannot be reversed, leading to many false positives compared to true positives. This is largely attributed to the eccentricity of the geometry of the optimization problem, which is largely mitigated with SR3. In SR3, there are nearly no false positives, leading to a more accurate and more efficient algorithm. This is shown visually in Fig. 6.20.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

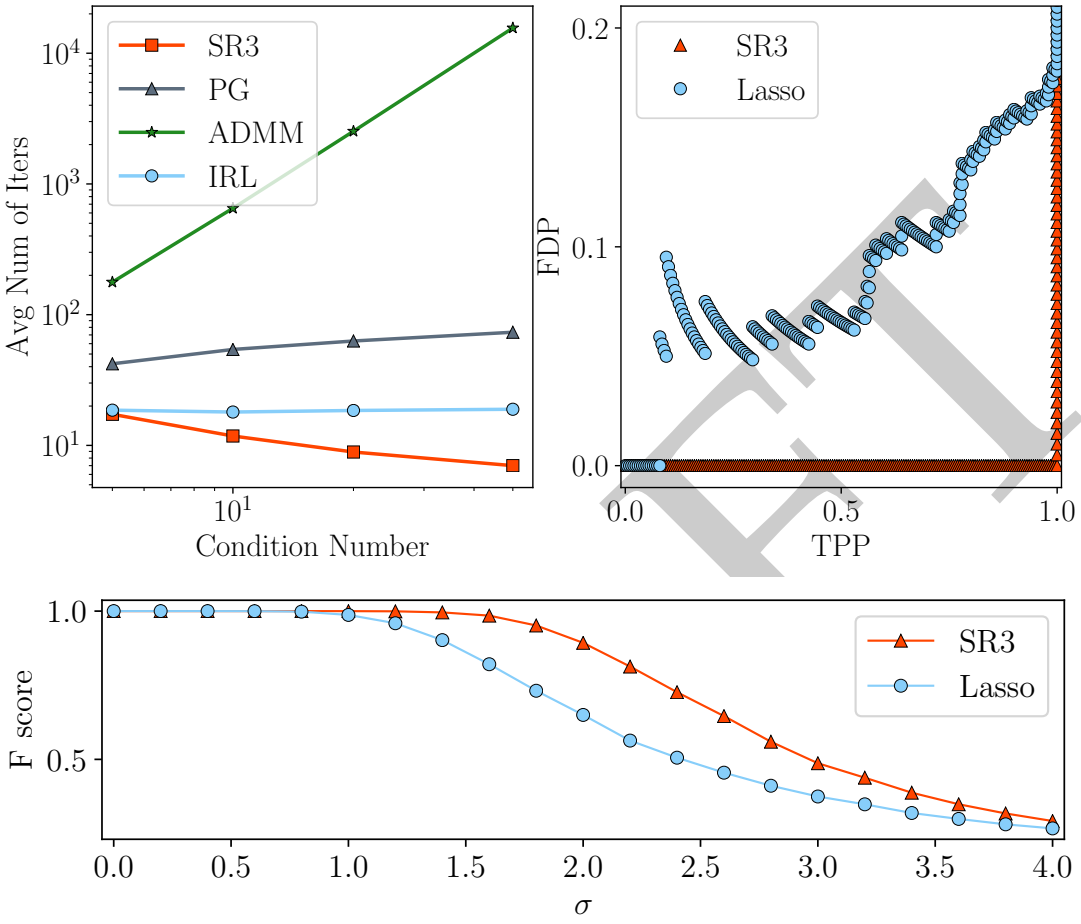


Figure 6.20: Top Left: SR3 approach (red) is orders of magnitude faster than ADMM (green) or other first-order methods such as prox-gradient (gray). While IRL (blue) requires a comparable number of iterations, its cost per iteration is more expensive than SR3. Top Right: True Positives vs. False Positives along the LASSO path (blue) and along the SR3 path (red). Bottom: F_1 score of SR3 (red) and LASSO formulation (blue) with respect to different noise levels. Figure reproduced from Zheng et al. [383].

Alternating Direction Method of Multipliers (ADMM)

The alternating direction method of multipliers (ADMM) [47, 272] is a splitting method, related to the method of Lagrange multipliers. It allows us to solve convex optimization problems by decomposing them into smaller, more manageable sub-problems, especially those that can be written as the sum of two functions that are individually easier to minimize. It combines the decomposability of dual ascent methods with the robustness of the method of multipliers. ADMM is widely used in applications, including signal processing, machine learning, and control. Several specific applications include robust PCA [67], MAP inference [243, 248], LP decoding [21], graphical models [86], and sparse feedback control [224]. We will explore robust PCA in the next subsection. ADMM is related to

Copyright © July 15, 2025 Brunton. All Rights Reserved.

earlier operator splitting methods and penalty-duality schemes [109, 146, 136, 227, 121]. Because of its widespread applicability, ADMM has inspired many efficient software implementations [47, 270]. There are also a number of works to develop theoretical results concerning guarantees and rates of convergence [164, 149, 120].

There are two common formulations, the consensus form and the composite form. These are equivalent, and will both be discussed here.

Consensus form of ADMM. A standard form of ADMM may be written as

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & f(\mathbf{x}) + g(\mathbf{z}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}. \end{aligned}$$

This form of ADMM is suitable for splitting across $\mathbf{x}$ and $\mathbf{z}$. It allows for distributed computing when f and g are separable and makes use of the augmented Lagrangian:

$$\mathcal{L}_\rho(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}) = f(\mathbf{x}) + g(\mathbf{z}) + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}\|^2.$$

Composite form of ADMM. A special form of consensus ADMM may be written as

$$\min_{\mathbf{x}} \; f(\mathbf{x}) + g(\mathbf{A}\mathbf{x}).$$

using a variable substitution:

$$\mathbf{z} = \mathbf{A}\mathbf{x}.$$

Then the problem becomes:

$$\min_{\mathbf{x},\mathbf{z}} \quad f(\mathbf{x}) + g(\mathbf{z}) \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} - \mathbf{z} = 0.$$

This matches the consensus form with:

$$\mathbf{B} = -\mathbf{I}, \quad \mathbf{c} = \mathbf{0}.$$

The two forms are equivalent; the second is a more structured special case that is easier to apply to problems like LASSO, total variation denoising, or compressed sensing.

We introduce the new variables $\mathbf{z} = \mathbf{A}\mathbf{x}$ to decouple the problem, making it less stiff and more easily distributed. If g is nonsmooth or difficult to minimize in composition, then minimizing $f(\mathbf{x}) + g(\mathbf{A}\mathbf{x})$ directly might be hard. Introducing $\mathbf{z}$ allows one to alternatively minimize $f(\mathbf{x})$ with a quadratic penalty, then minimize $g(\mathbf{z})$ separately, for example using known proximal operators; this process is repeated until convergence. The constraint $A\mathbf{x} = \mathbf{z}$ is then enforced softly via the dual variable and the quadratic penalty.

Primal-Dual Interpretation. ADMM is fundamentally a primal-dual algorithm, which is related to the discussion of constraints and duality in Chapter 5. ADMM is a special case of the *augmented Lagrangian method*, which itself is a primal-dual method. In this more general approach, we form the augmented Lagrangian with respect to the constraint (e.g., $\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}$), and then alternate the updates, first minimizing with respect to the primal

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

variables ($\mathbf{x}$, $\mathbf{z}$), and then performing a gradient ascent on the dual variable ($\boldsymbol{\lambda}$). This differs from standard Lagrangian dual ascent, which may suffer from slow convergence or instability. ADMM can be viewed as a dual ascent method with proximal regularization, which dampens the dual updates with a penalty to encourage stability and convergence. This is an application of the Douglas-Rachford algorithm [109] to the dual problem [204].

Specifically, given the optimization problem

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & f(\mathbf{x}) + g(\mathbf{z}) \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{z} \end{aligned}$$

Then ADMM iterates through the following

$$\begin{aligned} \mathbf{x}_{k+1} &= \arg\min_{\mathbf{x}} \Big(f(\mathbf{x}) + \frac{\rho}{2} \|\mathbf{x} - \mathbf{z}_k + \mathbf{u}_k\|^2 \Big), \\ \mathbf{z}_{k+1} &= \arg\min_{\mathbf{z}} \Big(g(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{x}_{k+1} - \mathbf{z} + \mathbf{u}_k\|^2 \Big), \\ \mathbf{u}_{k+1} &= \mathbf{u}_k + \mathbf{x}_{k+1} - \mathbf{z}_{k+1}, \end{aligned}$$

where $\mathbf{u}$ is a dual variable (Lagrange multiplier), and $\rho > 0$ is a penalty parameter. This framework generalizes to multiple constraints. This may be written in pseudo-code as

```
initialize x, z, u        # primal variables and scaled dual variable
repeat until convergence:
    x := prox_f( -z - u , 1/rho )
    # Solves x := argmin_x [ f(x) + (rho/2) * ||x + z  + u||^2 ]
    z := prox_g( -x - u , 1/rho )
    # Solves z := argmin_z [ g(z) + (rho/2) * ||x + z  + u||^2 ]
    u := u + x + z    # dual variable update
```

The functions `prox_f` and `prox_g` are often simple to derive and execute. For example, the proximal gradient for $f(\mathbf{x}) = \|\mathbf{x}\|_1$ is given by the soft thresholding operator:

```
def prox_l1(x, lambda, rho):
    return np.sign(x) * np.maximum(np.abs(x) - lambda/rho, 0)
```

Code Example: LASSO

Here we implement ADMM to solve the LASSO regression problem

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1 \tag{6.10}$$

by introducing a new variable $\mathbf{z}$ to handle the $\lambda\|\cdot\|_1$ term, and a Lagrange multiplier $\mathbf{u}$ to enforce $\mathbf{z} = \mathbf{x}$. Specifically, we solve the new problem

$$\min_{\mathbf{x},\mathbf{z}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{z}\|_1 + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 \tag{6.11}$$

using the proximal gradient for the non-smooth term $\|\mathbf{z}\|_1$ (e.g., the soft thresholding operator). The $f(\mathbf{x})$ and $g(\mathbf{z})$ functions are

$$f(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = \tfrac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \left(\mathbf{A}^T\mathbf{b}\right)^T\mathbf{x} + \mathbf{b}^T\mathbf{b} \tag{6.12a}$$

$$g(\mathbf{z}) = \|\mathbf{z}\|_1. \tag{6.12b}$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Based on these equations, the ADMM updates are given by

$$\mathbf{x}_{k+1} = \left(\mathbf{A}^T\mathbf{A} + \rho\mathbf{I}\right)^{-1}\left(\mathbf{A}^T\mathbf{b} + \rho\left(\mathbf{z}_k - \mathbf{u}_k\right)\right) \tag{6.13a}$$

$$\mathbf{z}_{k+1} = \mathcal{S}_{\lambda/\rho}\left(\mathbf{x}_{k+1} + \mathbf{u}_k\right) \tag{6.13b}$$

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{x}_{k+1} - \mathbf{z}_{k+1}. \tag{6.13c}$$

Here $\mathcal{S}$ is a shrinkage operator $\mathcal{S}_\tau(x) = \text{sign}(x)\max(|x| - \tau, 0)$. The updates for $\mathbf{x}$ and $\mathbf{z}$ correspond to the following proximal updates:

$$\mathbf{x}_{k+1} = \arg\min_{\mathbf{x}} \quad f(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}_k + \mathbf{u}_k\|_2^2 = \text{prox}_{f,\rho}(\mathbf{z}_k - \mathbf{u}_k) \tag{6.14a}$$

$$\mathbf{z}_{k+1} = \arg\min_{\mathbf{z}} \quad \lambda g(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{x}_{k+1} - \mathbf{z} + \mathbf{u}_k\|_2^2 = \text{prox}_{\lambda g,\rho}(\mathbf{x}_{k+1} + \mathbf{u}_k). \tag{6.14b}$$

```python
def ADMM_LASSO(A, b, lam, rho=1.0, max_iter=1000, tol=1e-4):
    # min (1/2)||Ax - b||^2 + lam * ||z||_1   s.t. x = z

    m, n = A.shape
    x = np.zeros(n)        # primal variable for f
    z = np.zeros(n)        # primal variable for g
    u = np.zeros(n)        # scaled dual variable

    # Precompute static matrices
    AtA = A.T @ A
    Atb = A.T @ b
    LHS = AtA + rho * np.eye(n)
    linsolve = lambda rhs: np.linalg.solve(LHS, rhs)

    for k in range(max_iter):
        # (1) x-update: minimize f(x) + (rho/2)||x - z + u||^2
        q = Atb + rho * (z - u)
        x = linsolve(q)

        # (2) z-update: proximal operator for lambda * ||z||_1
        v = x + u
        z_old = z.copy()
        z = np.sign(v) * np.maximum(np.abs(v) - lam / rho, 0.0)

        # (3) dual update
        u += x - z

        # (4) convergence check
        r_norm = np.linalg.norm(x - z)           # primal residual
        s_norm = rho * np.linalg.norm(z - z_old) # dual residual
        if r_norm < tol and s_norm < tol:
            break

    return z  # final solution: z approx x
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

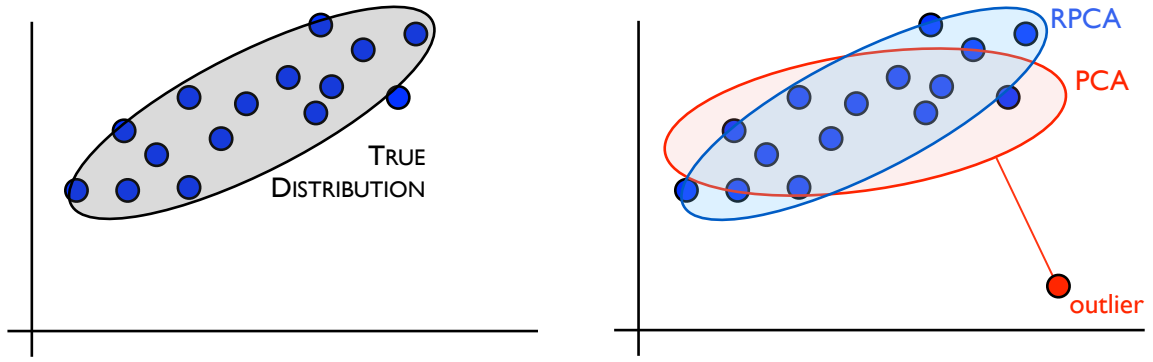


Figure 6.21: Principal component analysis seeks a Gaussian distribution that best captures the variance in a set of observation data (left). Outlier data can skew the entire approximated distribution, making standard PCA fragile to outliers (right, red). Robust PCA (right, blue) is more robust to outlier data.

Application: Robust Principal Component Analysis (RPCA)

As mentioned earlier in Chapter 4, least-squares regression is highly susceptible to outliers and corrupted data. Principal component analysis (PCA) suffers from the same weakness, making it *fragile* with respect to outliers; see Fig. 6.21. To ameliorate this sensitivity, Candès et al. [67] have developed a robust principal component analysis (RPCA) that seeks to decompose a data matrix $\mathbf{X}$ into a structured low-rank matrix $\mathbf{L}$ and a sparse matrix $\mathbf{S}$ containing outliers and corrupt data:

$$\mathbf{X} = \mathbf{L} + \mathbf{S}. \tag{6.15}$$

The principal components of $\mathbf{L}$ are *robust* to the outliers and corrupt data in $\mathbf{S}$. This decomposition has profound implications for many modern problems of interest, including video surveillance (where the background objects appear in $\mathbf{L}$ and foreground objects appear in $\mathbf{S}$), face recognition (eigenfaces are in $\mathbf{L}$ and shadows, occlusions, etc. are in $\mathbf{S}$), natural language processing and latent semantic indexing, and ranking problems[2]. Figure 6.22 shows an example in facial recognition, where a fake mustache is removed from a face, revealing the most statistically likely features underneath. This approach is also used in scientific applications, for example to de-noise experimental data [309], as shown in Fig. 6.23.

Mathematically, the goal is to find $\mathbf{L}$ and $\mathbf{S}$ that satisfy the following:

$$\min_{\mathbf{L},\mathbf{S}} \text{rank}(\mathbf{L}) + \|\mathbf{S}\|_0 \;\text{ subject to }\; \mathbf{L} + \mathbf{S} = \mathbf{X}. \tag{6.16}$$

[2]The ranking problem may be thought of in terms of the Netflix prize for matrix completion. In the Netflix prize, a large matrix of preferences is constructed, with rows corresponding to users and columns corresponding to movies. This matrix is sparse, as most users only rate a handful of movies. The Netflix prize seeks to accurately fill in the missing entries of the matrix, revealing the likely user rating for movies the user has not seen.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.



Figure 6.22: A classic example application of RPCA is in image processing, where it is possible to use a large library of human faces to build a robust statistical distribution of faces, making it possible to remove or "subtract" occlusions, such as a fake mustache, filling in details missing in the original image.

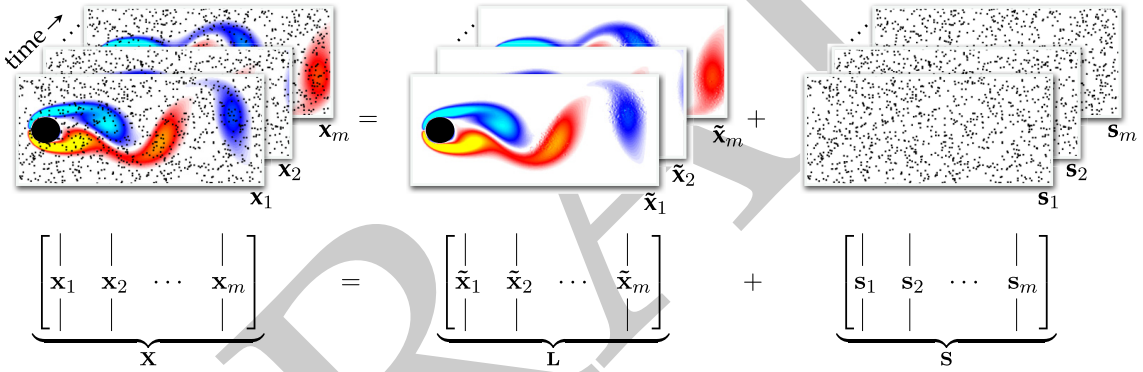


Figure 6.23: RPCA may also be applied to scientific data, such as experimentally acquired fluid flow field images, enabling robust de-noising and cleaning without removing key coherent structures [309].

However, neither the rank($\mathbf{L}$) nor the $\|\mathbf{S}\|_0$ terms are convex, and this is not a tractable optimization problem. Similar to the compressed sensing problem, it is possible to solve for the optimal $\mathbf{L}$ and $\mathbf{S}$ with *high probability* using a convex relaxation of (6.16):

$$\min_{\mathbf{L},\mathbf{S}} \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 \text{ subject to } \mathbf{L}+\mathbf{S}=\mathbf{X}. \tag{6.17}$$

Here, $\|\cdot\|_*$ denotes the nuclear norm, given by the sum of singular values, which is a proxy for rank. Remarkably, the solution to (6.17) converges to the solution of (6.16) with high probability if $\lambda = 1/\sqrt{\max(n,m)}$, where n and m are the dimensions of $\mathbf{X}$, given that $\mathbf{L}$ and $\mathbf{S}$ satisfy the following conditions:

1. $\mathbf{L}$ is not sparse
2. $\mathbf{S}$ is not low-rank; we assume that the entries are randomly distributed so that they

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

do not have low-dimensional column space.

The convex problem in (6.16) is known as *principal component pursuit* (PCP), and may be solved using the augmented Lagrange multiplier (ALM) algorithm. Specifically, an augmented Lagrangian may be constructed:

$$\mathcal{L}(\mathbf{L},\mathbf{S},\mathbf{Y}) = \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 + \langle \mathbf{Y}, \mathbf{X}-\mathbf{L}-\mathbf{S}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{L}-\mathbf{S}\|_F^2. \tag{6.18}$$

A general solution would solve for the $\mathbf{L}_k$ and $\mathbf{S}_k$ that minimize $\mathcal{L}$, update the Lagrange multipliers $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu(\mathbf{X}-\mathbf{L}_k-\mathbf{S}_k)$, and iterate until the solution converges. However, for this specific system, the alternating directions method (ADM) [225, 379] provides a simple procedure to find $\mathbf{L}$ and $\mathbf{S}$.

First, a shrinkage operator $\mathcal{S}_\tau(x) = \text{sign}(x)\max(|x|-\tau, 0)$ is constructed.

```
def shrink(X,tau):
    Y = np.abs(X)-tau
    return np.sign(X) * np.maximum(Y,np.zeros_like(Y))
```

Next, the singular value threshold (SVT) operator $\text{SVT}_\tau(\mathbf{X}) = \mathbf{U}\mathcal{S}_\tau(\mathbf{\Sigma})\mathbf{V}^*$ is constructed.

```
def SVT(X,tau):
    U,S,VT = np.linalg.svd(X,full_matrices=0)
    out = U @ np.diag(shrink(S,tau)) @ VT
    return out
```

Finally, it is possible to use $\mathcal{S}_\tau$ and SVT operators iteratively to solve for $\mathbf{L}$ and $\mathbf{S}$ in the alternating directions method of multipliers (ADMM) Code:

```
def RPCA(X):
    n1,n2 = X.shape
    mu = n1*n2/(4*np.sum(np.abs(X.reshape(-1))))
    lambd = 1/np.sqrt(np.maximum(n1,n2))
    thresh = 10**(-7) * np.linalg.norm(X)

    S = np.zeros_like(X)
    Y = np.zeros_like(X)
    L = np.zeros_like(X)
    count = 0
    while (np.linalg.norm(X-L-S) > thresh) and (count < 1000):
        L = SVT(X-S+(1/mu)*Y,1/mu)
        S = shrink(X-L+(1/mu)*Y,lambd/mu)
        Y = Y + mu*(X-L-S)
        count += 1
    return L,S
```

RPCA is demonstrated on the eigenface data with the following code

```
X = faces[:,:nfaces[0]]
L,S = RPCA(X)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) Numerical optimization, by J. Nocedal and S. J. Wright, 1999 [268].

(2) Simulated annealing, by V. Laarhove, J. M. Peter, and E. H. L. Aarts, 1987 [355].

(3) Real-Time Optimization by Extremum-Seeking Control, by K. B. Ariyur and M. Krstić, 2003 [10].

Papers and reviews

(1) Simulated annealing, by D. Bertsimas and J. Tsitsiklis, *Statistical science*, 1993 [35].

(2) Convergence properties of the Nelder–Mead simplex method in low dimensions, by J. C. Lagarias and J. A. Reeds, M. H. Wright, and P. E. Wright, *SIAM Journal on optimization*, 1998 [212].

(3) Just Relax: Convex Programming Methods for Identifying Sparse Signals in Noise, by J. A. Tropp, *IEEE Transactions on Information Theory*, 2006 [346].

(4) Robust Principal Component Analysis?, by E. J. Candès, X. Li, Y. Ma, and J. Wright, *Journal of the ACM*, 2011 [67].

(5) Compressed sensing, by D. L. Donoho, *IEEE Transactions on Information Theory*, 2006 [105].

(6) Stable signal recovery from incomplete and inaccurate measurements, by E. J. Candès, J. Romberg, and T. Tao, *Communications in Pure and Applied Mathematics*, 2006 [69].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

Exercise 6–1: Explore the downhill simplex method on the four-minima problem from Fig. 6.4. Start with several different random initial simplexes, of different sizes. Plot the ensemble of objective function values versus iteration number, as in Fig. 6.11.

Exercise 6–2: Now we will explore the downhill simplex method on a convex problem with a single global minima: a quadratic objective function $f(\mathbf{x}) = \|\mathbf{x}\|_2^2$. In this problem, you will vary the different parameters of the algorithm, and plot the ensemble of objective function values versus iterations, as above. Explore the convergence rate, success rate, etc. What are the optimal parameters?

Exercise 6–3: Try the downhill simplex method on the objective function with a saddle point in Eq. (2.25) from Chapter 2. Explore the convergence rate for different parameters.

Now, attempt to add momentum to the algorithm, similar to how we added momentum in Chapter 2. How does this change the convergence rate and stability of the algorithm?

Exercise 6–4: Extremum Seeking. When it is not possible to compute the gradient of a function directly, it is often possible to estimate the gradient by injecting an input perturbation and cross-correlating with the resulting output perturbation. The method of *extremum seeking control* (ESC) [209, 10] uses this approach to track the maximum of a function, even when the objective function varies in time according to a dynamical system.

In the simplest form, when the objective function is static, this may be viewed as a form of gradient-free nonlinear optimization. We will revisit the case of a dynamically varying objective function in Chapter 9.

Extremum-seeking control may be thought of as an advanced *perturb-and-observe* method, whereby a sinusoidal perturbation is additively injected in the actuation signal and used to estimate the gradient of an objective function J that should be maximized or minimized. The extremum-seeking control architecture is shown in Fig. 6.25. This schematic depicts ESC for a scalar input u, although the methods readily generalize for vector-valued inputs $\mathbf{u}$. A convex objective function $J(u)$, is shown in Fig. 6.24 for static plant dynamics (i.e., for $y = u$). The extremum-seeking controller uses an input perturbation to estimate the gradient of the objective function J and steer the mean actuation signal towards the optimizing value.

In extremum-seeking control, a sinusoidal perturbation is added to the estimate of the input that maximizes the objective function, $\hat{u}$:

$$u = \hat{u} + a\sin(\omega t). \tag{6.19}$$

This input perturbation passes through the system dynamics and output, resulting in an objective function J that varies sinusoidally about some mean value, as shown in Fig. 6.24. The output J is high-pass filtered to remove the mean (DC component), resulting in the

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

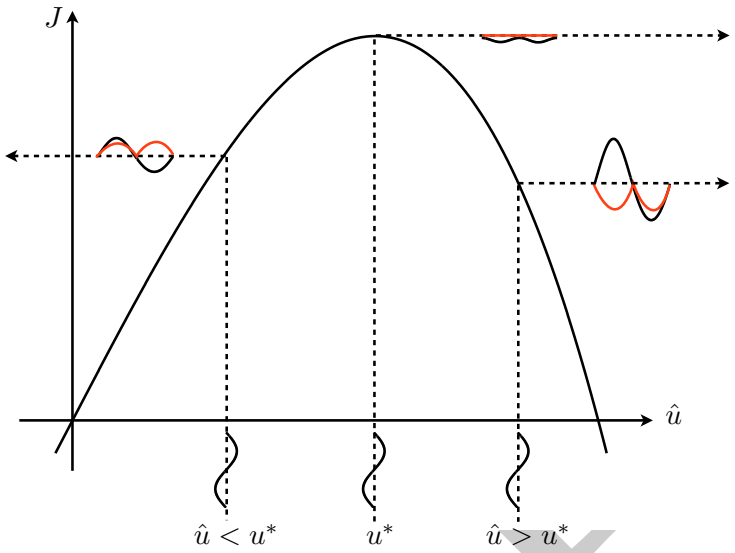


Figure 6.24: Schematic illustrating extremum-seeking control on a static objective function $J(u)$. The output perturbation (red) is in phase when the input is left of the peak value (i.e. $u < u^*$) and out of phase when the input is to the right of the peak (i.e. $u > u^*$). Thus, integrating the product of input and output sinusoids moves $\hat{u}$ towards u^*.

oscillatory signal ρ. A simple high-pass filter is represented in the frequency domain as

$$\frac{s}{s+\omega_h} \tag{6.20}$$

where s is the Laplace variable, and ω_h is the filter frequency. The high-pass filter is chosen to pass the perturbation frequency ω. The high-pass filtered output is then multiplied by the input sinusoid, possibly with a phase shift ϕ, resulting in the *demodulated* signal ξ:

$$\xi = a\sin(\omega t - \phi)\rho. \tag{6.21}$$

This signal ξ is mostly positive if the input u is to the left of the optimal value u^* and it is mostly negative if u is to the right of the optimal value u^*, shown as red curves in Fig. 6.24. Thus, the demodulated signal ξ is integrated into $\hat{u}$, the best estimate of the optimizing value

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{u} = k\,\xi, \tag{6.22}$$

so that the system estimate $\hat{u}$ is steered towards the optimal input u^*. Here, k is an integral gain, which determines how aggressively the actuation climbs gradients in J.

Roughly speaking, the demodulated signal ξ measures gradients in the objective function, so that the algorithm climbs to the optimum more rapidly when the gradient is larger. A block diagram of the extremum-seeking control schematic is shown in Fig. 6.25.

In this exercise, you will show that the average signal ξ_{avg} is proportional to the gradient of the objective function J with respect to the input u.

Consider a constant objective function, where J is simply a function of the input $J(u) = J(\hat{u} + a\sin(\omega t))$.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

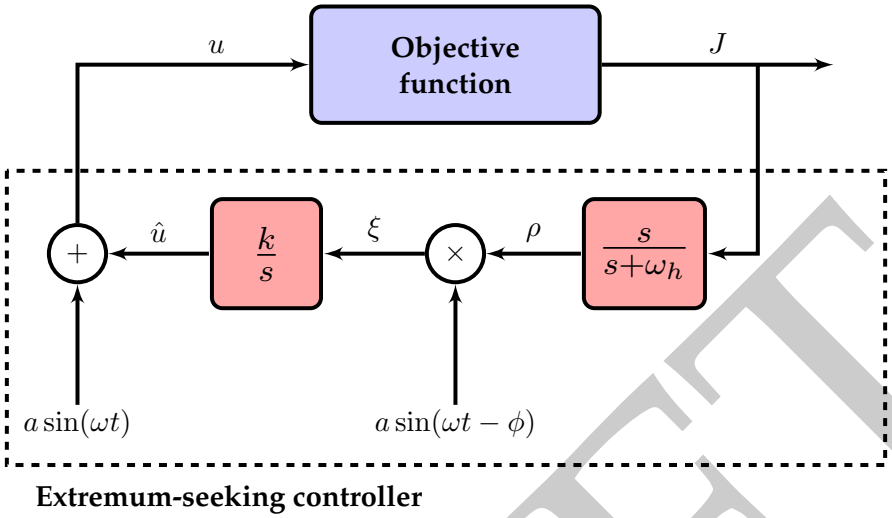


Figure 6.25: Schematic illustrating an extremum-seeking controller. A sinusoidal perturbation is added to the best guess of the input $\hat{u}$, and it passes through the plant, resulting in a sinusoidal output perturbation that may be observed in the sensor signal $\mathbf{y}$ and the cost J. The high-pass filter results in a zero-mean output perturbation, which is then multiplied (demodulated) by the same input perturbation resulting in the signal ξ. This demodulated signal is finally integrated into the best guess $\hat{u}$ for the optimizing input u.

- Expanding $J(u)$ in the perturbation amplitude a, which is assumed to be small.
- Average $\xi = a\sin(\omega t - \phi)\rho$ over one period.

Exercise 6–5: Here we consider a simple application of extremum-seeking control to find the maximum of a static quadratic cost function,

$$J(u) = 25 - (5 - u)^2. \tag{6.23}$$

Compute the optimal value by hand.

Design an extremum-seeking control with a perturbation frequency of $\omega = 10\,\text{Hz}$ and an amplitude of $a = 0.2$. Plot the controller response and the tracking of the optimal value. Hint: use a simple Butterworth high-pass filter.

Notice that when the gradient of the cost function is larger (i.e., closer to $u = 0$), the oscillations in J are larger, and the controller climbs more rapidly. When the input u gets close to the optimum value at u^*, even though the input perturbation has the same amplitude a, the output perturbation is nearly zero (on the order of a^2), since the quadratic cost function is flat near the peak. Thus we achieve fast tracking far away from the optimum value and small deviations near the peak.

Exercise 6–6: Design an extremum seeking controller to find the peak of a quartic cost function $J(u) = 16 - (2 - u)^4$, similar to the previous example with quadratic cost in (6.23). Do you need to modify the extremum seeking control parameters? Plot the response and discuss the performance.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 7

Bayesian Modeling and Estimation

Bayesian statistics provides a powerful and flexible framework for balancing prior knowledge about a system with new evidence, in the form of observation data. Many methods in optimization are either rooted in Bayesian statistics, or have a Bayesian interpretation. For example, the Tikhonov and LASSO regularized least-squares from Chapter 4 come from specific Bayesian prior distributions on the unknown parameters. Much of modern machine learning also have foundations in statistics and benefit from a Bayesian interpretation.

Prior information comes in many forms, such as the relatively mild assumption that unknown variables are Gaussian, or the strong belief in conservation laws from physics. Bayesian methods are particularly useful when we have either small data sets or complex models. They are used extensively for hyper-parameter tuning of machine learning models, for experimental design, and for applications in engineering that involve complex partial differential equations. Much of Chapter 8 on inverse problems may also benefit from Bayesian methods.

More broadly, the problem of estimating a statistical distribution from data is fundamentally an optimization problem. It is also the basis of much of modern machine learning. Density estimation is a vast topic, and the simplest approach is to assume a known probability distribution and simply estimate the parameters of the distribution using a best-fit optimization approach, such as least squares regression. If we assume that there is Gaussian noise on the observations, there is a statistical interpretation of least-squares as a *maximum likelihood estimation*. If we further include Bayesian prior information, this becomes a *maximum a posteriori* estimation problem. All of these are statistical estimation problems involving optimization.

This chapter will provide a relatively high-level and simplified view of Bayesian methods in optimization, starting with a brief recap of statistical estimation problems. Key concepts, such as density estimation, regularization, surrogate modeling, expectation maximization, and bias-variance tradeoff, will be explored here. We will provide simple examples with code, and more extensive examples in the homework.

It is assumed that the reader has a basic knowledge of probability and statistics, including the common probability distributions (e.g., Bernoulli for a single coin flip with probability of heads p, binomial for n repeated trials of a Bernoulli event, normal distribution for the large n limit, Poisson for the large n but small np limit, etc.). For a quick recap, see the *Probability Bootcamp* and the first several homework exercises.

DRAFT

7.1 Parameter Estimation in Statistics

Parameter estimation and fitting distributions is a central topic in data analysis and statistics, and it is the basis of much of modern machine learning. Typically, we will have data from a system and will try to find the probability distribution that best fits the data, resulting in an optimization problem.

The simplest approach is to assume a known probability distribution, such as a Gaussian or a Poisson distribution, and estimate the parameters of the model from data. For example, if we are modeling radioactive decay, we know from physical arguments that a Poisson distribution will be appropriate, and we need only infer the unknown parameter λ. Similarly, if we are modeling the height of a population, it is reasonable to assume a Gaussian distribution, with unknown mean μ and standard deviation σ. Statistical methods to estimate the parameters of a known distribution are known as *parametric* methods.

More complex data will likely not be described perfectly by a named probability distribution, and instead we fit a more general parameterized probability density using machine learning. Natural images are a good example of data that *does* follow some probability distribution, but that is not well modeled with classical, named distributions. Similarly, problems in physics, such as turbulence, follow probability distributions that do not yet have names. These data are often high-dimensional and have complex joint PDFs between the variables. Statistical approaches for learning distributions that do not have known parameterizations are called as *non-parametric* methods.

We will begin with the simplest parametric estimation problem, based on maximum likelihood estimation. In the absence of any prior information about the parameters of the distribution, we simply use the data to find the most likely parameters. We then extend this approach to incorporate prior information in Bayesian methods. Finally, we introduce more general, non-parametric methods in later sections.

Parametric Probability Distributions

The parametrized probability distribution may be written as

$$f(x \mid \theta) = P(X = x \mid \theta) \tag{7.1}$$

which is the probability of a random variable X taking a specific value x given the parameters θ. For example, a normal distribution is given by

$$f(x \mid \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x-\mu^2}{2\sigma^2}}$$

where the parameters are given by the mean and variance, $\theta \equiv [\mu, \sigma^2]$. If we know these parameters, then we can easily determine the probability of a particular random variable X taking the value x. Thus, we say that $P(X \mid \theta)$ is the probability of measuring the observed data X in a random trial, given the parameters θ. This is known as the *likelihood* function.

Probability is focused on constructing a model of the random process governing the variable X to arrive at the probability distribution $f(x \mid \theta)$ of observing future data X. In statistics, we are concerned with the inverse problem of determining the model parameters θ that are most consistent with the observed data X. This may be posed as an optimization

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Table 7.1: The first row is the number of clicks measured in a 10 second interval, and the second row is the number of 10 second intervals observed with that many clicks.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	6	11	28	56	105	126	146	164	161	123	101	74	53	23	15	9	5

problem by finding θ that maximizes the probability $P(X \mid \theta)$. This is the approach taken in the method of maximum likelihood estimation (MLE), which will be discussed after the following example. More generally, it would be convenient to directly have the probability of a given set of parameters θ given the data X, $P(\theta \mid X)$, which is the focus of Bayesian methods, discussed in the next section.

Example: Fitting Radioactive Decay with a Poisson Distribution

Before introducing more notation or theory, we will begin with a simple and intuitive motivating example. Consider the simple example of fitting the radioactive decay of alpha particles emitted by Americium 241, based on an example from Rice [292], which was taken from Berkson 1966 [32]. The number of alpha particle emissions per unit time should follow a Poisson[1] distribution

$$P(X = x \mid \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where λ parametrizes the distribution. The time between clicks is an exponentially distributed random variable. The precise timing of 10220 Gieger counter clicks (i.e., detected alpha particle emissions) was recorded, and the data was binned into 10-second intervals. The number of clicks per 10-second interval are given in Table 7.1.

The goal is to find the parameters of a Poisson distribution that best fit this data, which will involve an optimization problem. Figure 7.1 shows several Poisson distributions for various values of λ, along with the data. Clearly the Poisson distribution with $\lambda = 8$ is the closest to the data, but we can do better with optimization.

Figure 7.2 shows the optimal Poisson distribution, where we fit the parameter λ using the sample mean $\bar{X} = \frac{1}{N} X_j$, where N is the number of samples. For a Poisson distribution, the optimal estimate for λ is given by $\bar{X}$, as we will show using the maximum likelihood estimate. The fitting is achieved via the following code.

```
from scipy.stats import poisson

ClicksPerTenSec = np.arange(18)
ObservedCounts = [1, 6, 11, 28, 56, 105, 126, 146, 164, 161, 123,
    101, 74, 53, 23, 15, 9, 5]

# Calculate the sample mean
sample_mean = np.average(ClicksPerTenSec, weights=ObservedCounts)

# Fit a Poisson distribution with lambda = sample_mean
```

[1]The Poisson distribution is derived from the binomial distribution in the limit of small np, as it was recognized that a normal distribution will have a significant probability of negative events, which is fishy.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

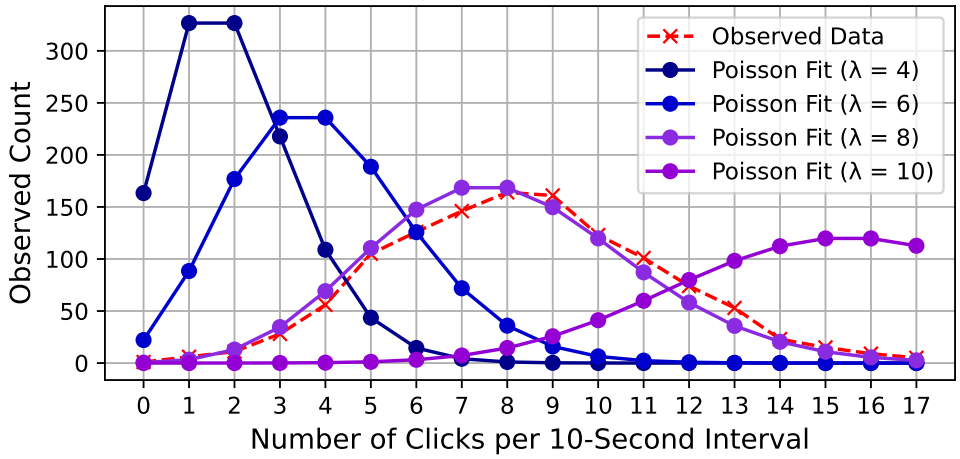


Figure 7.1: Alpha particle emission data (red) and several Poisson fits. The distribution with $\lambda = 8$ agrees best with the data, although we can find an optimal λ.

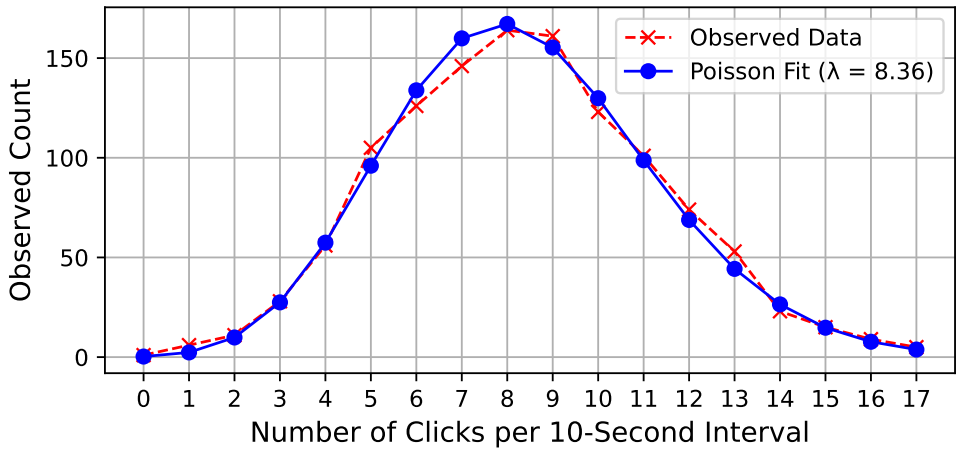


Figure 7.2: Alpha particle emission data (red) and Poisson fit (blue) using maximum likelihood estimate $\lambda = \bar{X}$.

```
poisson_dist = poisson.pmf(ClicksPerTenSec, mu=sample_mean) * np.sum(
    ObservedCounts)  # Scale to match observed data
```

It is useful to know the variance of the estimated parameter λ. Specifically, if we resample the system, how much do we expect the estimate λ to vary? We may estimate the distribution of λ^* without resampling using a *bootstrap*. This involves using the estimated λ to sample new data, and then computing the best-fit λ to this synthetic data. Figure 7.3 shows the bootstrap distribution, which is approximately normally distributed, by the central limit theorem. The following code computes the bootstrap distribution.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

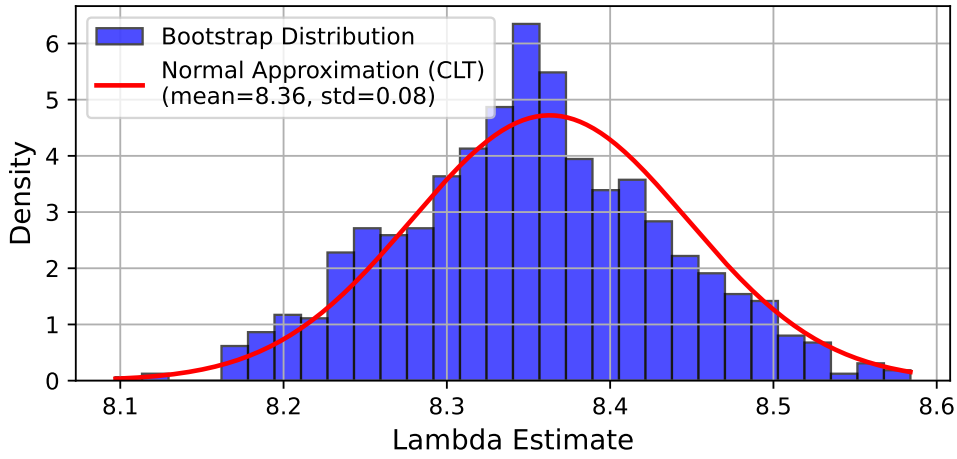


Figure 7.3: Bootstrap estimate of the distribution of the estimated λ parameter, obtained by sampling an ensemble of synthetic data from estimated Poisson distribution, and then computing the optimal λ values for each synthetic data set. The bootstrap estimate λ follows a normal distribution, via the central limit theorem, which is also computed and shown here.

```
num_bootstrap = 1000  # Number of bootstrap samples
bootstrap_lambdas = []

# Bootstrap procedure
for _ in range(num_bootstrap):
    # Generate a synthetic dataset using the estimated lambda
    synthetic_data = np.random.poisson(lam=sample_mean, size=np.sum(
        Observed))

    # Bin synthetic data (number of counts per 10-second interval)
    synthetic_hist, _ = np.histogram(synthetic_data, bins=range(19))

    # Calculate new lambda estimate from synthetic data
    new_lambda = np.average(n, weights=synthetic_hist)

    # Store new lambda estimate
    bootstrap_lambdas.append(new_lambda)

# Calculate sample standard deviation of observed data
obsv_std = np.sqrt(np.average((n-sample_mean)**2,weights=Observed))

# Calculate CLT-based standard deviation for lambda estimates
clt_std = obsv_std / np.sqrt(sum(Observed))
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Maximum Likelihood Estimation (MLE)

MLE is a method to estimate the parameters of a probability distribution from data. It frames statistical estimation as an optimization problem. This is the basis of many algorithms in machine learning, and it is the foundation for Bayesian parameter estimation.

Let $X_1, X_2, \cdots, X_n$ have a joint probability density function:

$$f(x_1, x_2, \cdots, x_n \,|\, \theta) = P(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n \,|\, \theta).$$

If the random variables X_j are independent and identically distributed (i.i.d) then

$$f(x_1, x_2, \cdots, x_n \,|\, \theta) = \prod_{j=1}^{n} P(X_j = x_j \,|\, \theta).$$

This is a common assumption for repeated trials or observations of a system that is not changing, such as sequential coin flips or measurements of radioactive decay.

We then define the likelihood of θ:

$$\text{lik}(\theta) = f(X_1, X_2, \cdots, X_n \,|\, \theta) = \prod_{j=1}^{n} f(X_j \,|\, \theta).$$

Instead of defining the PDF f in terms of the random variables x_j, we explicitly evaluate it on the data X_j. This function tells us how likely the data is given the parameters, and it is only a function of θ. Thus, we can explicitly maximize the likelihood over θ:

$$\hat{\theta} = \underset{\theta}{\text{argmax}}\ \text{lik}(\theta).$$

The optimal parameter estimate is $\hat{\theta}$. In MLE, we usually minimize the logarithm of the likelihood function $\ell(\theta) = \log\left(\text{lik}(\theta)\right)$. Because the logarithm is a smooth, monotonic function, the maximum of the log likelihood is the same as the maximum of the likelihood. The log-likelihood is easier to optimize, as the product of PDFs becomes a sum of log-PDFs:

$$\log\left(\text{lik}(\theta)\right) = \log\left(\prod_{j=1}^{n} f(X_j \,|\, \theta)\right) = \sum_{j=1}^{n} \log\left(f(X_j \,|\, \theta)\right).$$

It is generally easier to optimize these sums.

Example: Normal Distribution. Consider random variables that are distributed according to a normal distribution, $X_j \sim \text{normal}(\mu, \sigma^2)$, so that the joint PDF is given by

$$f(x, x_2, \cdots, x_n \,|\, \mu, \sigma^2) = \prod_{j=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_j - \mu}{\sigma}\right)^2}.$$

The log of the PDF is

$$\log(f) = \sum_{j=1}^{n} \left[-\log(\sigma) - \frac{1}{2}\log(2\pi) - \frac{1}{2}\left(\frac{x_j - \mu}{\sigma}\right)^2 \right]$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

so the log-likelihood function is $\log(f)$ evaluated on the data X_j:

$$\ell(\mu, \sigma^2) = \log(f) \,|_{X_j} = -n \log(\sigma) - \frac{n}{2} \log(2\pi) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (X_j - \mu)^2 .$$

To maximize the log-likelihood, we compute the partial of ℓ with respect to μ and σ and set both equal to zero:

$$\frac{\partial \ell}{\partial \mu} = \frac{1}{\sigma^2} \sum_{j=1}^{n} (X_j - \mu) = \frac{1}{\sigma^2} (n\bar{X} - n\mu)$$
$$\frac{\partial \ell}{\partial \sigma} = -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^{n} (X_j - \mu)^2,$$

where $\bar{X}$ again is the sample mean. Setting both expressions equal to zero yields

$$\frac{\partial \ell}{\partial \mu} = 0 \quad \Longrightarrow \quad \hat{\mu} = \bar{X}$$
$$\frac{\partial \ell}{\partial \sigma} = 0 \quad \Longrightarrow \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{j=1}^{n} \left(X_j - \bar{X} \right)^2 .$$

To prove that these values maximize the log-likelihood, we would need to compute the second derivative to show that the curvature is positive, which is left as an exercise.

Example: Poisson Distribution. Consider random variables that are distributed according to a Poisson distribution, $X_j \sim \text{Poisson}(\lambda)$, so that the joint PDF is given by

$$f(x_1, x_2, \cdots, x_n \,|\, \theta) = \prod_{j=1}^{n} f(x_j \,|\, \theta) = \prod_{j=1}^{n} \frac{\lambda^{x_j} e^{-\lambda}}{x_j!}.$$

The log-likelihood function for this joint distribution is given by

$$\begin{aligned} \ell(\theta) = \log(\text{lik}(\theta)) &= \log \left(f(X_1, X_2, \cdots, X_n \,|\, \theta) \right) \\ &= \log \prod_{j=1}^{n} \frac{\lambda^{X_j} e^{-\lambda}}{X_j!} \\ &= \sum_{j=1}^{n} \log \left(\frac{\lambda^{X_j} e^{-\lambda}}{X_j!} \right) \\ &= \sum_{j=1}^{n} \log \left(X_j \log(\lambda) - \lambda - \log(X_j!) \right) . \end{aligned}$$

To maximize this log-likelihood, we now compute the derivative of this with respect to $\theta = \lambda$ and set it equal to zero:

$$\frac{\mathrm{d}}{\mathrm{d}\lambda} \ell(\lambda) = \frac{1}{\lambda} \sum_{j=1}^{n} X_j - n = 0 \quad \Longrightarrow \quad \lambda = \frac{1}{n} \sum_{j=1}^{n} X_j,$$

which is the sample mean $\bar{X}$. This was how we fit λ in the previous example.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Properties of MLE. The MLE has several useful statistical properties:

- *Consistency:* The estimate $\hat{\theta}$ is consistent, meaning that $\hat{\theta} \to \theta$ in the large data limit, or as $n \to \infty$.

- *Normalcy:* Further, the estimate is a normally distributed random variable whose mean is the true θ and whose variance decreases as n increases:

$$\hat{\theta} \sim \text{normal}\left(\theta_{\text{true}}, \frac{1}{nI(\theta_{\text{true}})}\right).$$

  This is closely related to the central limit theorem, although it is subtly different. This is a very useful expression, as we can use it to construct confidence intervals on $\hat{\theta}$. The function $I(\theta)$ is given by

$$I(\theta) = \underbrace{\mathbb{E}\left[\frac{\partial}{\partial\theta}\log f(X\,|\,\theta)\right]^2}_{\mathbb{E}[\ell'(\theta)]^2} = \underbrace{-\mathbb{E}\left[\frac{\partial^2}{\partial\theta^2}\log f(X\,|\,\theta)\right]}_{-\mathbb{E}[\ell''(\theta)]}.$$

  Showing that $\hat{\theta}$ is normal is quite challenging.

  This distribution for $\hat{\theta}$ means that the MLE is asymptotically efficient, so that this estimate converges to the true value of θ as efficiently as possible, in the large n limit. This uses something important, called the Cramer-Rao inequality: Given i.i.d. data $X_1, \cdots, X_n$ and an unbiased estimate of the parameters of the distribution, $\hat{\theta} = \hat{\theta}(X_1, \cdots, X_n)$[2], then

$$\text{Var}(\hat{\theta}) \geq \frac{1}{nI(\theta)}$$

  for all smooth $f(x\,|\,\theta)$.

However, there are other approaches for distribution fitting, such as the method of moments, which matches the moments of the distribution to the moments of the data. We will also discuss related topics of hypothesis testing, for example to see if a given fit is statistically likely or not using the χ^2 test or the Kullback–Leibler (KL) divergence.

One major drawback to MLE is that it is fragile to bad data (e.g., outliers, etc.) and doesn't allow one to include prior knowledge or belief into the estimates. As a concrete example, imagine that we are estimating the probability of a coin being heads or tails. For a fair coin the probability of heads is 0.5, so $\theta = 0.5$. If we flip the coin three times and get three heads in a row, {H,H,H}, then the MLE yields $\hat{\theta} = 1$. In this case, the MLE will overfit to this bad data and suggest that the coin will always flip heads for all future flips. This is a major limitation, as we have additional prior knowledge that most coins are fair and three heads in a row is not particularly unlikely. These issues will be directly addressed in the Bayesian extension, maximum a posteriori (MAP) estimation, in the next section.

[2]Recall that the estimated parameters are in fact functions of the data $\{X_j\}_{j=1}^n$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

7.2 Bayesian Inference

In the following, we will now take the Bayesian perspective on statistical estimation. Before, we began with a parameterized probability distribution $f(X \mid \theta)$, which gives the probability of finding data X given parameters θ. However, in many cases it is more convenient to directly have the probability of a given set of parameters θ given the data X:

$$f(\theta \mid X).$$

This inverse model $P(\theta \mid X)$ is known as the *posterior* distribution, and the forward model $P(X \mid \theta)$ is known as the *likelihood*. They are related by Bayes's theorem as

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}. \tag{7.2}$$

The distribution $P(\theta)$ is called the *prior* distribution, and it encodes our prior knowledge about θ.

If we have $f(\theta \mid X)$ we can read off the optimal θ directly, but typically we only have the forward model $f(X \mid \theta)$. Thus, this is an inverse optimization problem, which is the subject of Chapter 8. In fact, the posterior distribution is called the Bayesian inverse of the likelihood function.

Bayes' Theorem. We are now ready to derive Bayes' theorem, which is a powerful way to invert conditional probability distributions of the data given parameters, $P(X \mid \theta)$, to obtain the distribution of the parameters given the data, $P(\theta \mid X)$.

First some notation. We denote the sample space Ω of all possible outcomes of a system. For example, if we have a deck of 52 cards with four suits and thirteen ranks, then there are 52 unique outcomes if we deal a single card. We denote specific events in the sample space; for example, event A may be that a heart is drawn, or event B may be that a seven is drawn. These are independent events, since the probability of a card being a heart does not change if we know that it is a seven, and vice versa.

The conditional probability of an event A occurring, given that event B is known to have occurred, is

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

where $P(A \cap B)$ is the probability that both A and B occurred. This is shown graphically in Fig. 7.4. We may multiply both sides by $P(B)$, yielding a formula for the probability $P(A \cap B)$:

$$P(A \cap B) = P(A \mid B)P(B) \tag{7.3a}$$
$$= P(B \mid A)P(A) \tag{7.3b}$$

where the second line is true since

$$P(B \mid A) = \frac{P(B \cap A)}{P(A)}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

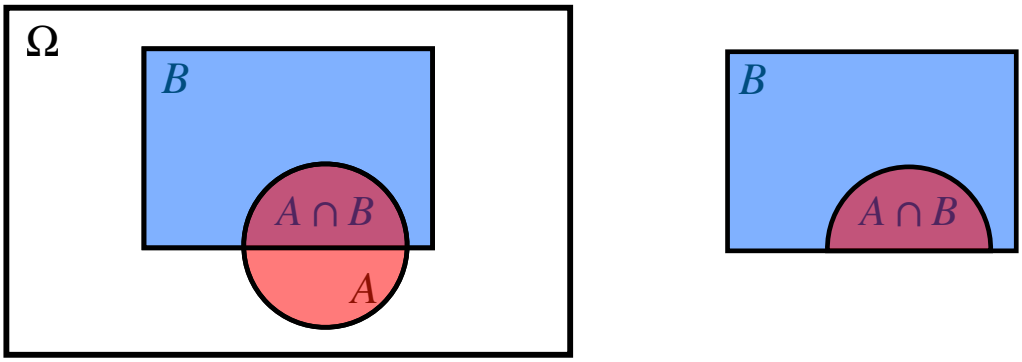


Figure 7.4: Illustration of conditional probabilities. (left) Given a sample space Ω and two events A and B, we denote $A \cap B$ as the event that both A and B occur. If we know that B occurs, then the probability of A given B, $P(A \mid B)$ is the ratio of $P(A \cap B)$ and $P(B)$, as shown in the right, where we have restricted ourselves to the set B, since we assume that B is known to have occured.

and $P(A \cap B) = P(B \cap A)$ by symmetry. If A and B are independent, then $P(A \cap B) = P(A)P(B)$; for example, a card being a heart (A) or a seven (B) are independent.

Rearranging the terms in Eq. (7.3) results in Bayes' theorem

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}, \tag{7.4}$$

which allows us to solve for $P(B \mid A)$ given $P(A \mid B)$, $P(A)$, and $P(B)$. We will use this formulation extensively in the following.

We may use Bayes' theorem to represent the probability of the unknown parameters $\boldsymbol{\theta}$ given the observed data X as

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}. \tag{7.5}$$

The distribution $P(\theta)$ is called the *prior* distribution, and it is how we incorporate prior knowledge about the parameters θ. For example, we may have prior knowledge that a coin is usually fair.

Bayes' theorem is a powerful rule for inverting conditional probabilities, based on the law of total probability. Bayes only explored the binomial distribution, and Laplace independently developed an extended and more powerful version of this theory. The modern theory was mainly developed by Laplace, and indeed maybe should be called the Bayes-Laplace theorem. Laplace may be thought of as the father of modern statistical thinking, being the first scientist to rigorously think of measurement data as a random variable with some associated distribution of measurement error.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

More General Forms of Bayes' Theorem. We might not know $P(A)$ in the denominator of Eq. (7.4), so instead we may write it in terms of B and its compliment $B^C = \Omega \setminus B$:

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A \mid B)P(B) + P(A \mid B^C)P(B^C)}.$$

This uses the law of total probability that either B or *not* B (i.e., B^C) must occur, so $P(B) + P(B^C) = 1$. Thus, the probability of A occurring, $P(A)$, must be equal to the sum of $P(A \mid B)P(B)$ and $P(A \mid B^C)P(B^C)$.

More generally, if we partition the sample space Ω into n disjoint sets

$$\Omega = \bigcup_{j=1}^{n} B_j \quad \text{with} \quad P(B_i \cap B_j) = 0 \quad \text{for } i \neq j$$

then we may write the law of total probability as

$$P(A) = \sum_j P(A \mid B_j)P(B_j).$$

This gives a more general form of Bayes' theorem

$$P(B_j \mid A) = \frac{P(A \mid B_j)P(B_j)}{\sum_{j=1}^{n} P(A \mid B_j)P(B_j)}.$$

Maximum a Posteriori (MAP) Estimation

We will now use Bayes's theorem to extend the maximum likelihood estimator to incorporate prior knowledge about θ. For example, we have strong prior knowledge that a coin we are flipping is usually fair.

In MLE we only use the likelihood $f(X \mid \theta)$. But if we have prior knowledge about θ, $f(\theta)$, then we can multiply these and get:

$$f(\theta \mid X) = \frac{f(X \mid \theta)f(\theta)}{f(X)}$$

which is a statement of Bayes's theorem. In practice, we actually want the posterior $f(\theta \mid X)$, which is the probability of the parameters θ given the data X. The factor $f(X)$ in the denominator may be thought of as a normalization constant, since it won't affect any optimization over θ. Often we choose to write

$$f(\theta \mid x) \propto f(x \mid \theta)f(\theta)$$

to remove the denominator.

Instead of computing the maximum of $\log f(X \mid \theta)$ we compute the maximum of $\log f(\theta \mid X)$, which we call the Bayesian maximum a posteriori (MAP) estimate:

$$\max \log f(\theta \mid X) = \max \log(f(X \mid \theta)f(\theta)).$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

MAP estimation is the Bayesian extension of the MLE, and it allows us to incorporate prior knowledge about the distribution on θ.

We have replaced the log-likelihood function from the MLE with a slightly more informative log-likelihood function that is informed by the prior information about θ. We will show in Section 7.5 that the LASSO and Tikhonov regularized least-squares solutions may be viewed as the MAP estimates for particular choices of priors.

Iterative Bayesian Updates

In an iterative Bayesian updating scheme, we update our prior as new information becomes available. This Bayesian update may be written as:

$$P_{k+1}(\theta \mid X) = \frac{P_k(X \mid \theta) P_k(\theta)}{P_k(X)}$$

so the posterior may be updated with new data. At each iteration, the posterior from the previous step becomes the prior distribution for the next step:

$$P_{k+1}(\theta) = P_k(\theta \mid X).$$

In this way, our prior beliefs update with new information. This scheme balances data with prior beliefs via the model $P(\theta)$. This is closely related to the notion of a Kalman filter, from Section **??**, which is also concerned with balancing measurement data and model predictions.

However, there are some well-known downsides to Bayesian statistics. There is a high sensitivity to the prior, which may be subjective or hard to quantify mathematically. A bad prior will significantly degrade a Bayesian model. These updates are also quite computationally complex, often involving Monte Carlo sampling and bootstrapping to estimate distributions from data; when the model is complex, with a high-dimensional parameter vector $\boldsymbol{\theta}$, these methods often suffer from the curse of dimensionality. Machine learning is helping to alleviate many of these scaling issues, although they are still challenging.

Figure 7.5 shows an example of a weak prior and a strong prior, both from a Gaussian distribution. Strong priors are harder to change with evidence, providing firmer belief in a given model. In many cases, we may also want *sticky* priors, for example when flipping a coin, we may assume that it is fair, despite several pieces of evidence that would cause the posterior to drift according to simple Bayesian updates.

Typically the posterior and prior distributions must be parametrized in some way, and using a known distribution, such as a Poisson or a normal distribution, is the simplest choice. However, it may be the case that the data does not follow a named distribution and a more general family of functions must be considered. Several approaches exist for parameterizing and estimating more general probability distributions, such as Gaussian mixture models (GMMs) and density estimation for unknown $P(\theta)$ and $P(X \mid \theta)$. Kernel density estimation (KDE) is a powerful technique that is widely used. KDE and GMMs will be explored in more detail in the next two subsections. In addition, it is increasingly common to parameterize the distribution with a neural network.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

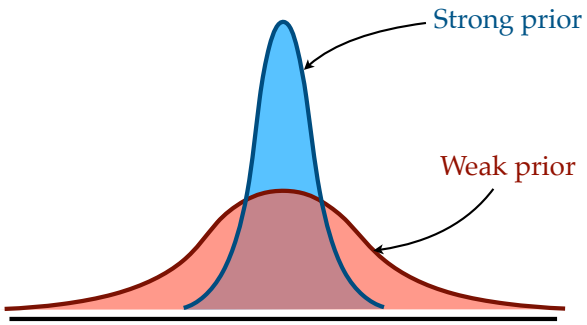


Figure 7.5: Example of a strong Gaussian prior (blue) and a weak Gaussian prior (red).

Conjugate Priors and Updates

One natural issue arises in the Bayesian update. We would like the posterior, $P(\theta \mid X)$ to follow the same distribution as the prior, $P(\theta)$, since in each iteration the updated posterior becomes the prior for the next iteration. However, multiplying the prior by the likelihood function, $P(X \mid \theta)$, gives

$$P(\theta \mid X) \propto P(X \mid \theta) P(\theta),$$

which may change the posterior distribution.

There is a clever solution. Often we know the likelihood function, since we may begin with a forward model of the probability density $P(X \mid \theta)$. In this case, we choose the prior from a special distribution so that when multiplied by the likelihood function it remains in the same distribution. These are called *conjugate* priors; where conjugate refers to its relationship to the likelihood function.

It is helpful to demonstrate this idea with a few examples. First, Gaussian distributions are conjugate with Gaussians, meaning that the product of two Gaussians remains Gaussian; this is a straightforward exercise for the reader.

Another common example is related to coin flips. Consider a random variable X that follows a binomial distribution:

$$X \sim \text{bin}(n, \theta) = \binom{n}{h} \theta^h (1-\theta)^{n-h}$$

where h is the probability of a single flip landing heads and X is the total number of heads given n coin flips. The conjugate prior distribution to a Bernoulli random variable is the beta distribution, so that $\theta \sim \text{beta}(\alpha, \beta)$:

$$P(\theta = v \mid \alpha, \beta) = k v^{\alpha-1} (1-v)^{\beta-1}$$

where k is a normalization constant related to the gamma functions[3] of α and β:

$$k = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}.$$

[3]Recall that the Γ function is related to the factorial, so that $\Gamma(n) = (n-1)!$ for positive integers n. It generalizes the factorial for non-integer n.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

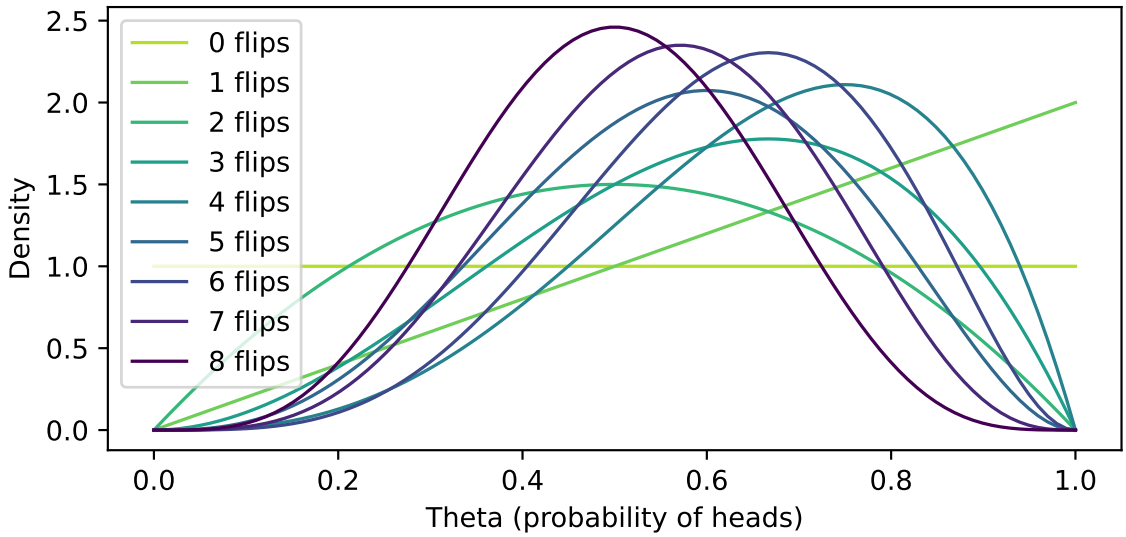


Figure 7.6: Evolution of the posterior for the coin flip example with increasing information. The posterior is approximated using the binomial distribution with the conjugate prior beta distribution.

DRAFT

Example: Coin Flips

Perhaps the simplest example of Bayesian inference is to iteratively estimate the behavior of a coin by accumulating data from sequential coin flips. We will investigate several different Bayesian approaches based on parametric and non-parametric estimation. First, we will assume that a single coin flip follows a Bernoulli distribution with an unknown probability of heads, θ. Thus, the probability of achieving n heads follows a binomial distribution. As discussed above, the conjugate prior follows a β distribution, making it is simple to accumulate information from new coin flips.

Figure 7.6 shows the posterior distribution evolving for several coin flips, starting with a uniform prior. Within a relatively small number of coin flips the posterior distribution approaches a binomial distribution centered around $\theta = 0.5$, which represents a fair coin.

As additional data is accumulated, we may follow the evolution of the posterior, along with its mean, in Fig. 7.7. We also show the ± 1 standard deviation regions about the mean in orange. We can see that there is considerable wander in the mean θ, and even by 100 coin flips, the estimated probability of heads is off of the true value by about 10%.

As mentioned earlier, Bayesian methods are sensitive to the initial prior, and a bad prior can dramatically reduce the performance of Bayesian inference. Figure 7.8 shows the evolving posterior distribution for a good and a bad starting prior. A bad initial prior (dashed) converges much more slowly to the true value of $\theta = 0.5$, whereas the good initial prior stays closer to $\theta = 0.5$.

The basic Bayesian update code with a binomial prior and a beta conjugate prior is given by the following.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

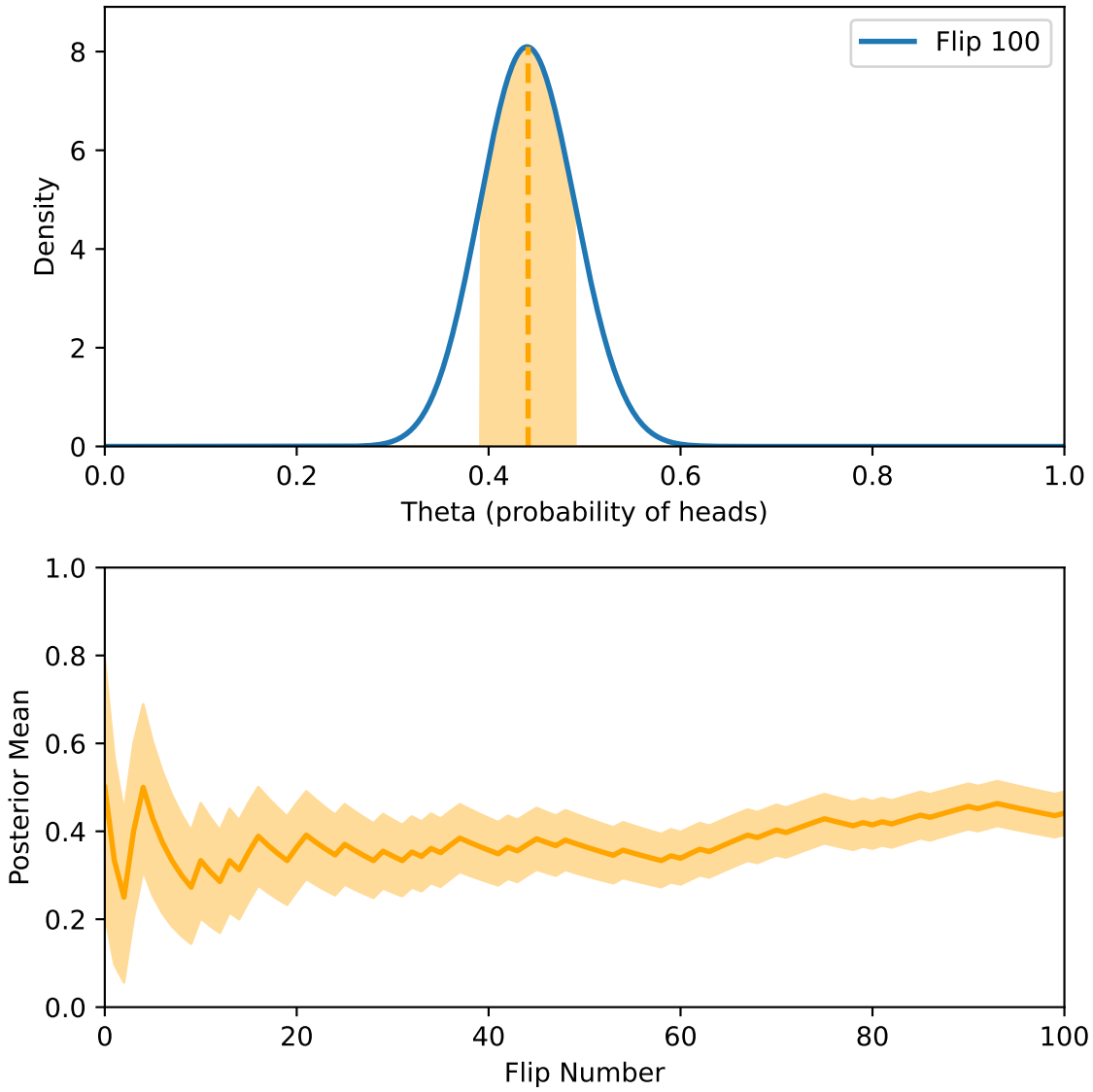


Figure 7.7: Posterior distribution (top) and posterior mean, for the standard binomial distribution with the conjugate prior beta distribution. Orange shaded regions denote ± 1 standard deviation.

```python
from scipy.stats import beta

def likelihood(theta, data):
    # Binomial likelihood for coin flips (Bernoulli trials)
    heads = np.sum(data == 'H')
    tails = np.sum(data == 'T')
    return theta**heads * (1 - theta)**tails
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

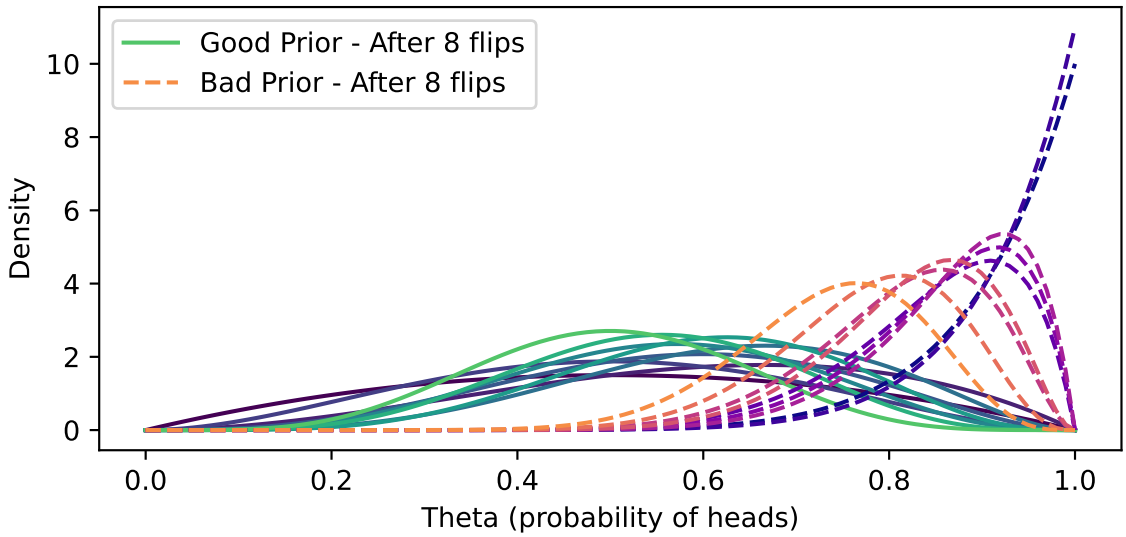


Figure 7.8: Comparison of Bayesian updates for the coin flip example with a good (solid) and a bad (dashed) prior.

```python
# Function to compute posterior distribution using Bayesian update
def bayesian_update(prior_alpha, prior_beta, data):
    # Calculate the number of heads and tails in the data
    heads = np.sum(data == 'H')
    tails = np.sum(data == 'T')

    # Update the prior (beta distribution) parameters
    posterior_alpha = prior_alpha + heads
    posterior_beta = prior_beta + tails

    return posterior_alpha, posterior_beta

# Prior: Uniform distribution (Beta(1,1))
prior_alpha = 1
prior_beta = 1

# Simulate a sequence of coin flips (H for heads, T for tails)
coin_flips = np.array(['H', 'T', 'H', 'H', 'T', 'H', 'T', 'T'])

# Perform Bayesian updates after each flip
for i, flip in enumerate(coin_flips):
    # Update the posterior with the new data (one flip)
    prior_alpha, prior_beta = bayesian_update(prior_alpha, prior_beta
        , np.array([flip]))
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exponential Family of Distributions

A major theoretical advance in statistics in the 1930s established that many common distributions belong to a large family of distributions with known conjugate prior distributions. The *exponential* family of distributions is given by

$$P(X \mid \theta) = h(x)e^{\langle \theta, \phi(X) \rangle - A(\theta)}$$

where $h(x)$ may be absorbed into the exponential and $A(\theta)$ is a normalization constant. At first, it is not obvious how a common distribution, such as the normal distribution, may be written in this form.

All exponential family members have a conjugate prior distribution give by

$$P(\theta \mid \gamma) = h_c(\theta) \exp\left[\left\langle \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}, \begin{bmatrix} \theta \\ -A(\theta) \end{bmatrix} \right\rangle - A_c(X)\right]$$

where γ_1 and θ are vectors of dimension n and γ_2 and $-A(\theta)$ are scalars.

Example: Gaussian Distribution. Here we will show how to represent a Gaussian distribution in the exponential family of distributions. For $X \sim \text{normal}(\mu, \sigma^2)$, we will have the following:

$$\boldsymbol{\theta} = \begin{bmatrix} \frac{\mu}{\sigma^2} \\ -\frac{1}{2\sigma^2} \end{bmatrix} \qquad \boldsymbol{\phi}(X) = \begin{bmatrix} X \\ X^2 \end{bmatrix}$$

so that $\boldsymbol{\theta}^T \boldsymbol{\phi}(X) = \frac{\mu X}{\sigma^2} - \frac{x^2}{2\sigma}$. Thus, if we take the exponential of this, we recover our Gaussian PDF:

$$\begin{aligned} P(X \mid \theta) \propto e^{\langle \theta, \phi(X) \rangle} &= e^{-\frac{1}{2\sigma^2}\left[X^2 - 2\mu X + \mu^2\right] + \frac{\mu^2}{2\sigma^2}} \\ &= e^{-\frac{1}{2\sigma^2}[X-\mu]^2} e^{\frac{\mu^2}{2\sigma^2}} \\ &\propto e^{-\frac{1}{2\sigma^2}(X-\mu)^2} \sim \mathcal{N}(\mu, \sigma^2). \end{aligned}$$

7.3 Density Estimation and Empirical Priors

In Bayesian inference, we want to estimate the parameters θ of a distribution from some data X, while incorporating prior beliefs about the distribution in the prior function $P(\theta)$. This is a nice way to balance partial knowledge of the system with new evidence from data.

We have investigated several examples where the likelihood function is a well known distribution, like a binomial or normal, and we can choose a good conjugate prior distribution.

However, for most real-world scenarios, the density functions will not be nice, well-behaved functions, but instead will be more complex functions. Density estimation is a powerful tool to estimate these more complex probability densities from data using a sum of simple building block distributions. Density estimation is a direct predecessor of

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

many modern machine learning methods. The learned distributions will be known as the empirical prior and posterior distributions. The simple building-block distributions will usually be Gaussians, for example in the Gaussian mixture model.

Two of the most common techniques are called kernel density estimation (KDE) and Gaussian mixture models (GMMs):

- KDE treats every data point in X like its own collocation point that will have a local Gaussian, and we estimate the PDF at other points by smoothing or interpolating across these functions.

$$P(X = x) = \frac{1}{Nh} \sum_{j=1}^{N} \mathcal{K}\left(\frac{x - X_j}{h}\right)$$

  The kernel $\mathcal{K}$ is often chosen to be a Gaussian, N is the number of data points, and h is a smoothing width, which is a hyperparameter chosen to smooth out the fit. Each X_j is a data sample. In essence, every time we see a data point X_j, this makes it more likely to see a new point near X_j in the future. In this way, KDE has a *frequentist* flavor. A schematic of kernel density estimation is shown in Fig. 7.9. Kernel density estimation is closely related to manifold learning, such as the diffusion maps embedding.

- GMMs also estimate the distribution using a sum of Gaussians. However, in KDE, there was a Gaussian for every single data point X_j, while for GMMs, we try to estimate the distribution with many fewer Gaussians. Finding the specific number of Gaussians and parameters is an optimization problem, typically using an expectation maximization (EM) algorithm. A schematic of GMMs is shown in Fig. 7.10.

Empirical distributions are *generative models*. If we have one of these empirical distributions, we can generate new data that follows these distributions. Generative models are the basis of modern methods in *generative AI*, such as large-language models and stable diffusion for image generation from text prompts.

Kernel Density Estimation

Here we demonstrate the method of kernel density estimation to learn the empirical posterior distribution for a coin. Figure 7.11 shows the evolution of the posterior as new information is accumulated. Figure 7.12 shows the distribution and the evolution of the mean and quantiles with the number of coin flips.

The following code implements the KDE method.

```
from sklearn.neighbors import KernelDensity

# Function to compute empirical likelihood of the data given samples
def likelihood(samples, data):
    # Binomial likelihood for coin flips (Bernoulli trials)
    heads = np.sum(data == 'H')
    tails = np.sum(data == 'T')
    return samples**heads * (1 - samples)**tails
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

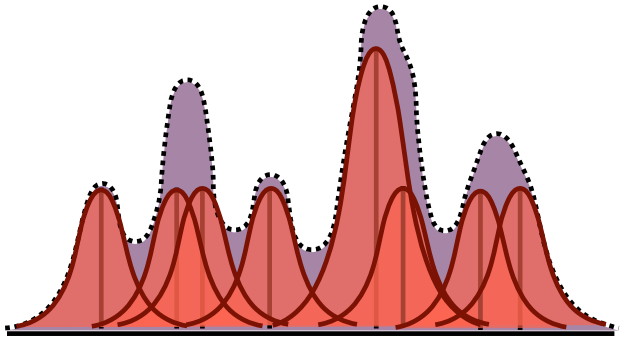


Figure 7.9: Kernel density estimation (KDE) may be thought of as a special case of a Gaussian mixture model, where there are as many Gaussians as there are data points.

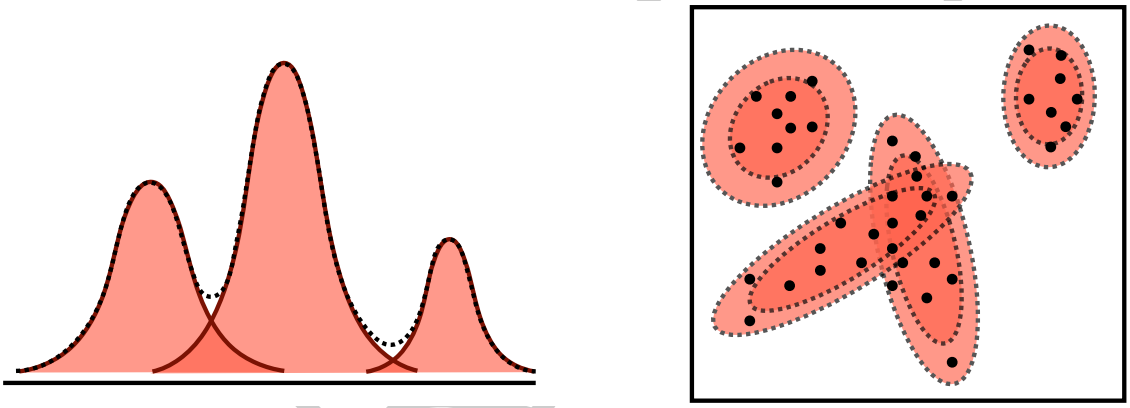


Figure 7.10: Gaussian mixture models (GMMs) visualized in one dimension (left) and two dimesnions (right). The PDF $P(\theta)$ is approximated by the sum of a relatively small number of Gaussians. The fit is typically obtained through an optimization algorithm, typically expectation maximization.

```
# Function to update the posterior distribution using KDE
def bayesian_update_kde(prior_samples, data):
    # Compute the empirical likelihood of the data
    likelihoods = likelihood(prior_samples, data)

    # Weight the prior samples by the likelihood
    posterior_weights = likelihoods / np.sum(likelihoods)

    # Resample prior based on likelihoods to form posterior samples
    posterior_samples = np.random.choice(prior_samples, size=1000, p=
        posterior_weights)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

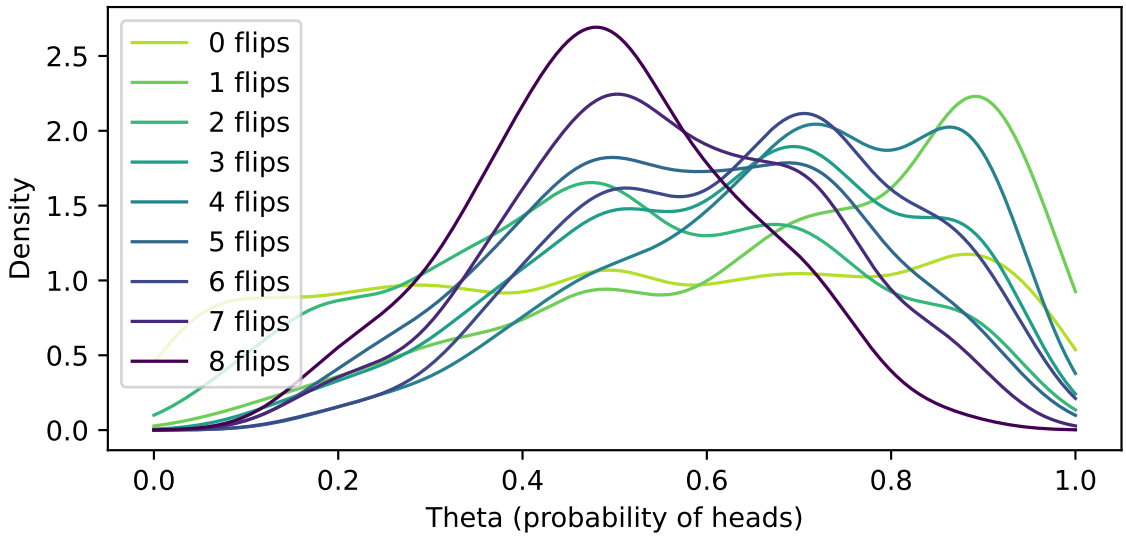


Figure 7.11: Evolution of the posterior for the coin flip example with increasing information. The posterior is approximated using kernel density estimation.

```
    # Kernel Density Estimation to estimate posterior
    kde = KernelDensity(kernel='gaussian', bandwidth=0.05).fit(
        posterior_samples[:, np.newaxis])

    return posterior_samples, kde

# Prior: Uniform distribution (between 0 and 1)
prior_samples = np.random.uniform(0, 1, 1000)

# Simulate a sequence of coin flips (H for heads, T for tails)
coin_flips = np.array(['H', 'T', 'H', 'H', 'T', 'H', 'T', 'T'])

# Store prior samples for plotting
samples_list = [prior_samples]

# Perform Bayesian updates after each flip
for i, flip in enumerate(coin_flips):
    # Update the posterior using the KDE method
    prior_samples, kde = bayesian_update_kde(prior_samples, np.array
        ([flip]))

    # Store the updated posterior samples
    samples_list.append(prior_samples)

# Create histogram (plotting omitted)
theta_range = np.linspace(0, 1, 1000)
for i, samples in enumerate(samples_list):
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

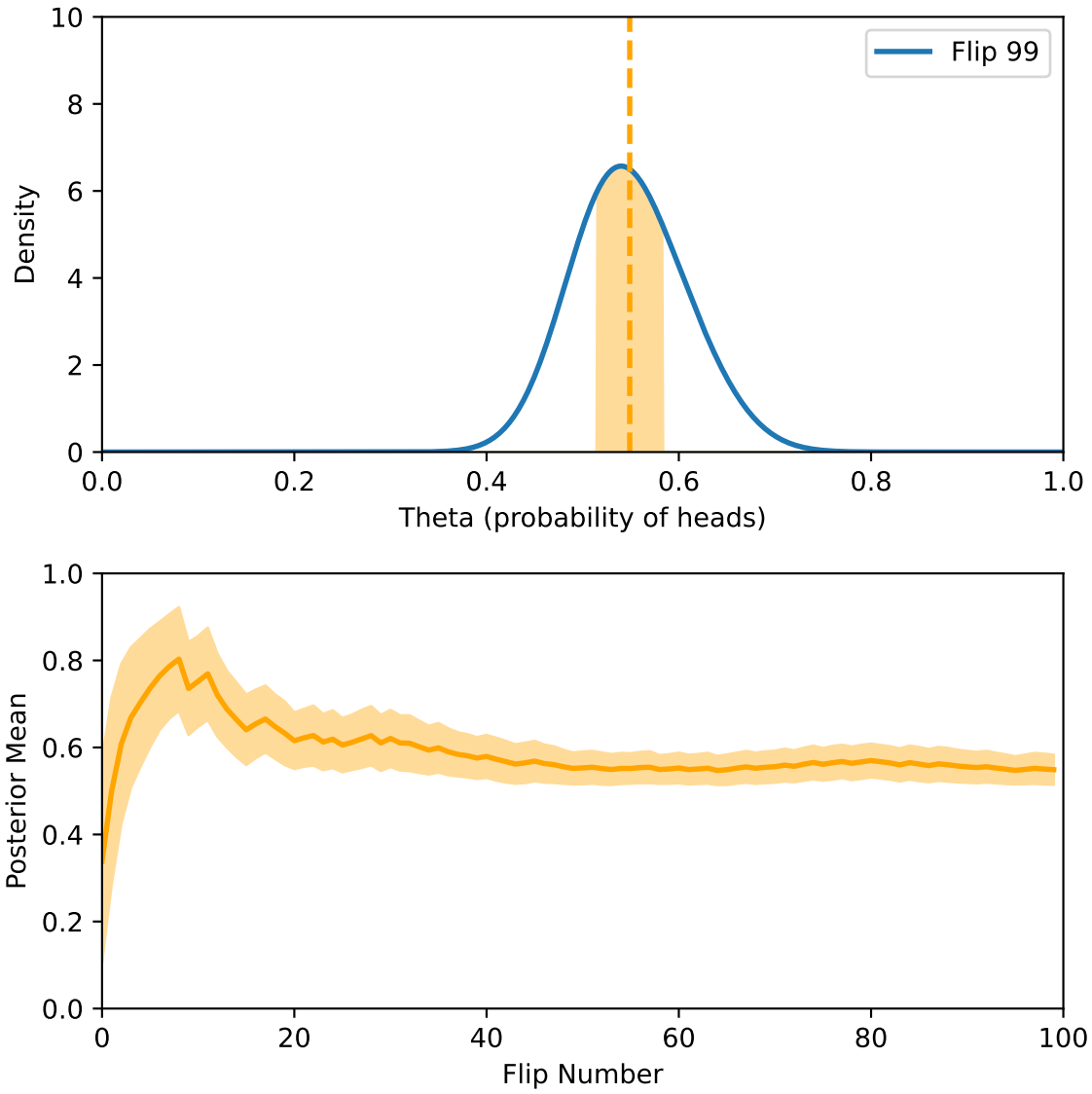


Figure 7.12: Posterior distribution (top) and posterior mean (bottom), with quantiles, for the kernel density estimation. Orange shaded regions denote ± 1 standard deviation.

```
    kde = KernelDensity(kernel='gaussian', bandwidth=0.05).fit(
        samples[:, np.newaxis])
    log_density = kde.score_samples(theta_range[:, np.newaxis])
    density = np.exp(log_density)
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Monte Carlo Sampling

When the posterior distribution is particularly complex, it may not be well described by a named distribution. In most cases when we don't know the posterior, we must resort to numerical methods, such as Monte Carlo sampling to obtain a bootstrap estimate the posterior distribution.

If the likelihood function and the prior are not conjugates, so the posterior is from a different distribution than the prior, then we may choose to use Monte Carlo sampling. Alternatively, if the parameter space is too high dimensional, we might not want to compute the posterior directly, and instead resort to sampling. There are many extensions to Monte Carlo sampling, such as Markov chain Monte Carlo (MCMC) [143] and Hamiltonian Monte Carlo [113, 258].

We used a Monte Carlo sampling method earlier to generate ensembles of data to obtain a bootstrap estimate of the unknown parameter $\hat{\theta}$ in the case of the Poisson distributed alpha particle emissions. Here we will extend this to the more powerful Bayesian inference framework.

Importantly, we assume that we do have access to the prior distribution $P(\theta)$ and the likelihood $P(X \mid \theta)$. There are several steps:

- Sample: We begin by sampling multiple θ_i from the prior distribution $P(\theta)$. This collection $\{\theta_i\}_{i=1}^n$ is an ensemble of model parameters.
- Weight: We weight each of these samples θ_i by the likelihood of the data X given this parameter: $w_i = P(X \mid \theta_i)$.
- Resampling/Bootstrapping: We are trying to approximate the posterior, which is the product of the likelihood and the prior, but with the ensemble of parameters $\{\theta_i\}$ and weights $\{w_i\}$. To do this, we resample the collection of parameters $\{\theta_i\}$ but weighted by $\{w_i\}$. This literally means creatting a new collection $\{\hat{\theta}_j\}$ by sampling, with replacement, from $\{\theta_i\}$. This will shift the distribution $P(\theta)$ based on the weights obtained from the likelihood function $P(X \mid \theta)$. The histogram of the weighted, resampled distribution $\{\hat{\theta}_j\}$ will follow the unknown posterior distribution.

We repeat this process, where the new posterior $\{\hat{\theta}_j\}$ becomes the prior sample in the next step of the iteration.

Figure 7.13 shows the posterior distribution and mean using batched Monte Carlo estimation. The following code computes this Monte Carlo sample-based estimation.

```
from scipy.stats import binom, uniform

theta_true = 0.5   # True value of theta (probability of heads)
batch_size = 10    # Number of coin flips per batch
total_flips = 250  # Total coin flips
num_samples = 5000 # Number of Monte Carlo samples

# Generate the true coin flip outcomes based on the true theta
np.random.seed(42)  # For reproducibility
coin_flips = np.random.binomial(1, theta_true, total_flips) # 1=heads
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

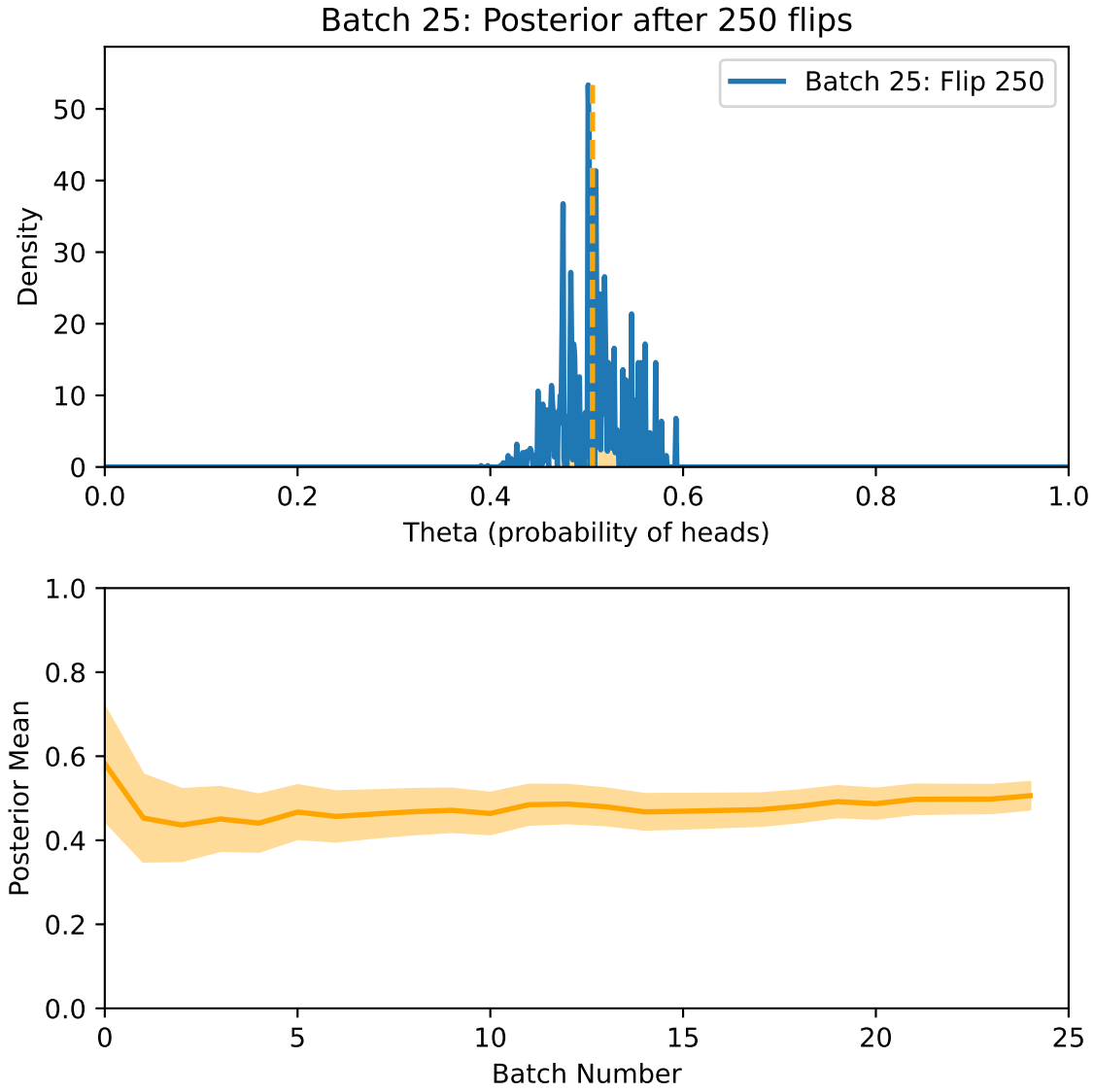


Figure 7.13: Posterior distribution (top) and posterior mean (bottom), with quantiles, for the Monte Carlo estimation. Orange shaded regions denote ± 1 standard deviation.

```
# Prior: Uniform distribution for theta (equivalent to Beta(1, 1))
prior_samples = np.random.uniform(0, 1, num_samples)

# Function to update the plots after each batch of flips
def update(batch_number):
    global prior_samples, std_shade1, std_shade2

    # Sample current batch of flips
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
    start = batch_number * batch_size
    end = start + batch_size
    batch_flips = coin_flips[start:end]

    # Calculate number of heads and tails in the current batch
    heads = np.sum(batch_flips)
    tails = batch_size - heads

    # Bionomial likelihood for each prior sample given data
    likelihoods = binom.pmf(heads, batch_size, prior_samples)

    # Reweight prior samples based on likelihood
    weights = likelihoods / np.sum(likelihoods)

    # Resample prior samples according to weights to form posterior
    posterior_samples = np.random.choice(prior_samples, size=
        num_samples, p=weights)

    # Update prior samples for next iteration (Monte Carlo approx.)
    prior_samples = posterior_samples
```

7.4 Expectation Maximization and Gaussian Mixture Models

Here we introduce expectation maximization (EM), which is a statistical learning technique. We demonstrate EM to optimize a Gaussian mixture model (GMM). EM is an approach for parameter estimation, for example using MLE or MAP, when there are hidden or latent variables, resulting in incomplete data.

In a Gaussian mixture model, we have m data points $\{X_j\}_{j=1}^m$ that are believed to belong to one of p Gaussian clusters with unknown means $\boldsymbol{\mu}_c$ and covariance matrices $\boldsymbol{\Sigma}_c$. There is also a probability α_c associated with belonging to each of the p Gaussian clusters. The subscript c indicates an index for which *cluster* the data belongs to.

If we know which cluster each data point X_j belongs to, then it is a simple statistical estimation problem to find the means and covariance matrices, corresponding to the p independent MLE or MAP optimizations. For a single Gaussian cluster, it is simple to estimate the mean and covariance matrix using the standard maximum likelihood estimate:

$$\hat{\boldsymbol{\mu}} = \frac{1}{m}\sum_{j=1}^{m} X_j$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{m}\sum_{j=1}^{m} (X_j - \hat{\boldsymbol{\mu}})^T (X_j - \hat{\boldsymbol{\mu}}) .$$

Not knowing which cluster each data point belongs to makes this a much more challenging, and generically non-convex, optimization problem to learn the model parameters. When we have multiple Gaussian clusters, there is a hidden variable corresponding to which cluster a given data point X_j belongs to. In this case, we will use the EM algorithm below.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Formally, we consider a system with variables $\mathbf{x}$ and $\mathbf{z}$, where $\mathbf{x}$ is observed and $\mathbf{z}$ is hidden. In the case of a GMM, the variable $\mathbf{z}$ determines which cluster c the observation $\mathbf{x}$ belongs to, and which Gaussian parameters $\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c$ to use. The variables follow a joint probability distribution that is governed by a parameter $\boldsymbol{\theta}$ so that $(\mathbf{x}, \mathbf{z}) \sim P(\boldsymbol{\theta})$, which may be written as

$$P(\mathbf{x}, \mathbf{z} \,|\, \boldsymbol{\theta}).$$

The goal of a statistical estimator, such as MLE or MAP, is to find the optimal parameters $\hat{\boldsymbol{\theta}}$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ P(\mathbf{x} \,|\, \boldsymbol{\theta})$$

that maximize the marginal likelihood distribution $P(\mathbf{x} \,|\, \boldsymbol{\theta})$, given by

$$P(\mathbf{x} \,|\, \boldsymbol{\theta}) = \sum_{c=1}^{p} P(\mathbf{x} \,|\, \boldsymbol{\theta}, \mathbf{z}_c) P(\mathbf{z}_c).$$

The probability of being in cluster c is just a constant $P(\mathbf{z}_c) = \alpha_c$, and the conditional probability $P(\mathbf{x} \,|\, \boldsymbol{\theta}, \mathbf{z}_c)$ is a normal distribution with parameters $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$, so the marginal distribution is

$$P(\mathbf{x} \,|\, \boldsymbol{\theta}) = \sum_{c=1}^{p} \alpha_c \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c).$$

The coefficients α_c are called the mixing coefficients. Thus, a GMM is a *generative* model. Given the model parameters $\{\alpha_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^{p}$, it is possible to generate new data that is consistent with the statistical distribution of the training data.

To learn the unknown model parameters, given that there is a hidden variable corresponding to which cluster each X_j belongs to, we will use the EM algorithm. We start with a set of clusters $\{\alpha_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^{p}$ as an initialization. There are several options to initialize a GMM, although starting with either a K-means clustering or a random initialization are common choices. After initializing, we iterate through the following two stages until convergence:

- Expectation (E) Step: For each data point X_j, we compute the probability r_{jc} that X_j is in cluster c:

$$r_{jc} = \frac{\alpha_c \mathcal{N}(X_j \,|\, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{c=1}^{p} \alpha_c \mathcal{N}(X_j \,|\, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}.$$

  This probability r_{jc} is called the *responsibility* value, and it computes the likelihood of X_j belonging in each of the p clusters. The denominator is a normalization constant. We now fix the r_{jc} values and update the model parameters to maximize the likelihood.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- Maximization (M) Step: For each cluster c, we now update the parameters $\{\alpha_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}$ to maximize the expected log-likelihood of the fit given the probabilities r_{jc} from the previous step. This amounts to recomputing each parameter using the MLE or MAP estimation, but weighted with r_{jc}. First, we compute weighting factors m_c and the probability of belonging to a cluster, α_c,

$$m_c = \sum_{j=1}^{m} r_{jc} \qquad \alpha_c = \frac{m_c}{\sum_{c=1}^{p} m_c}$$

  where the first sum is over all data points X_j. We then compute the weighted means and covariance matrices for each cluster:

$$\boldsymbol{\mu}_c = \frac{1}{m_c} \sum_{j=1}^{m} r_{jc} X_j$$

$$\boldsymbol{\Sigma}_j = \frac{1}{m_c} \sum_{j=1}^{m} r_{jc} \left(X_j - \boldsymbol{\mu}_c\right)^T \left(X_j - \boldsymbol{\mu}_c\right).$$

In the E step, we compute the expectation of the log-likelihood function given the current estimates. In the M step, we update parameters to maximize this expected log-likelihood function. This is a form of bootstrapping, where we iteratively refine an initial guess. More generally, EM works well with data following any of the exponential family of distributions, discussed earlier.

Each iteration increases the log-likelihood of the model and convergence to a local maximum is guaranteed. A good initialization is still required, as the algorithm will only converge to a local maximum.

The number of clusters is a hyper-parameter that must be carefully chosen. As with other statistical learning and machine learning algorithms with hyper-parameters, it may be possible to use information criteria, such as AIC/BIC, to penalize model complexity. Alternatively, it is possible to cross-validate the model with test data by choosing the number of clusters that maximize the log-likelihood of the model over test data.

7.5 Least-Squares Regression as a Statistical Estimation Problem

We now revisit the classic linear least-squares regression problem from a statistical estimation point of view. Specifically, we reframe least-squares regression as a maximum likelihood estimation problem, which will allow us to further include prior information in the related Bayesian maximum a posteriori estimation problem.

To recast the least-squares regression problem in a statistical framework, we assume that the errors of the data to the linear regression fit follow some statistical distribution, typically a normal distribution.

In one dimension, linear-regression finds the best-fit line

$$y = ax + b$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

that minimizes the sum-squared error of the fit. We estimate the parameters a and b. If we assume that data comes in pairs, $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, then we may formulate the data relationship as a matrix equation

$$\underbrace{\begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} a & b \end{bmatrix}}_{\boldsymbol{\theta}} \underbrace{\begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ 1 & 1 & \cdots & 1 \end{bmatrix}}_{\mathbf{X}}$$

which leads to the classic least-squares regression optimization problem

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\mathbf{Y} - \boldsymbol{\theta}\mathbf{X}\|_2^2.$$

Recall from Chapter 4 that this has a closed form solution in terms of the SVD. The least-squares solution is what is known as a point estimate, since we gain no information about the distribution of y, just its expected value.

MLE Linear Regression

If we now say that the data have noise

$$y = ax + b + \eta,$$

we introduce the error term η, which is a normally distributed random variable: $\eta \sim \mathcal{N}(0, \sigma^2)$. This is a standard assumption that may be justified by the central limit theorem. Now we will estimate $\boldsymbol{\theta} = \begin{bmatrix} a & b \end{bmatrix}$ as the MLE based on the data and the assumption of η being a Gaussian.

Our estimate for y is no longer a simple point estimate, but instead follows a distribution. This is much stronger, since we can quantify the uncertainty in our estimate, for example allowing us to provide confidence intervals.

Since y is now a random variable, we can estimate its probability density as

$$P(y \,|\, \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}\mathbf{x}, \sigma^2).$$

In other words, the expected value of y is given by the linear fit $y = \boldsymbol{\theta}\mathbf{x} = ax + b$. This estimate has a variance given by the variance of the noise, σ^2.

We now write our log-likelihood function in terms of this distribution and maximize to find the best-fit parameters $\boldsymbol{\theta}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The log-likelihood is

$$\begin{aligned}\ell(\theta) &= \log \prod_{j=1}^{n} P(y_j \,|\, x_j, \theta) \\ &= \sum_{j=1}^{n} \log P(y_j \,|\, x_j, \theta) \\ &= \sum_{j=1}^{n} \log \left[\frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(y_j - \theta x_j)^2}{2\sigma^2}} \right] \\ &= \sum_{j=1}^{n} \Bigg[\underbrace{-\frac{(y_j - \theta x_j)^2}{2\sigma^2}}_{\propto \sum (y_j - \theta x_j)^2} + \underbrace{\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)}_{\text{constant}} \Bigg].\end{aligned}$$

Thus, the likelihood function is proportional to the classic least squares regression objective function:

$$\ell(\theta) \propto \|\mathbf{Y} - \boldsymbol{\theta}\mathbf{X}\|_2^2.$$

MAP Linear Regression

We may now extend the MLE linear regression above to include prior information in the Bayesian MAP regression. Recall that the MAP estimate maximizes the log posterior, where the posterior is the Bayesian inverse, given by

$$P(\theta \,|\, x, y) \propto P(y \,|\, x, \theta) P(\theta).$$

Thus the log-posterior is given by

$$\begin{aligned}\log(P(\theta \,|\, x, y)) &= \log(P(y \,|\, x, \theta))\ell(\theta) + \log(P(\theta)) \\ &= \underbrace{\|\mathbf{Y} - \boldsymbol{\theta}\mathbf{X}\|_2^2}_{\text{MLE}} + \underbrace{\log(P(\boldsymbol{\theta}))}_{R(\boldsymbol{\theta})}.\end{aligned}$$

The second term, $\log(P(\theta))$ is the log-prior, which serves as a regularization term for the optimization to promote solutions that are consistent with the prior knowledge.

Gaussian Prior and Ridge Regression. As a simple example, we will assume that the prior distribution on the parameters θ follow a Gaussian distribution, so that

$$\theta \sim \mathcal{N}(0, \sigma^2) \quad \Longrightarrow \quad P(\theta) = e^{-\frac{\theta^2}{2\sigma^2}}.$$

In this case, the log-prior is given by

$$-\log P(\theta) = \frac{\theta^2}{2\sigma^2}$$

so the log-posteriori becomes

$$\log P(\theta \,|\, x, y) = \|\mathbf{Y} - \boldsymbol{\theta}\mathbf{X}\|_2^2 + \alpha \|\boldsymbol{\theta}\|_2^2$$

where $\alpha \propto 1/\sigma^2$. Thus, a Gaussian prior on θ appears as a Tikhonov regularization term, which we saw earlier in Sec. 4.4.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

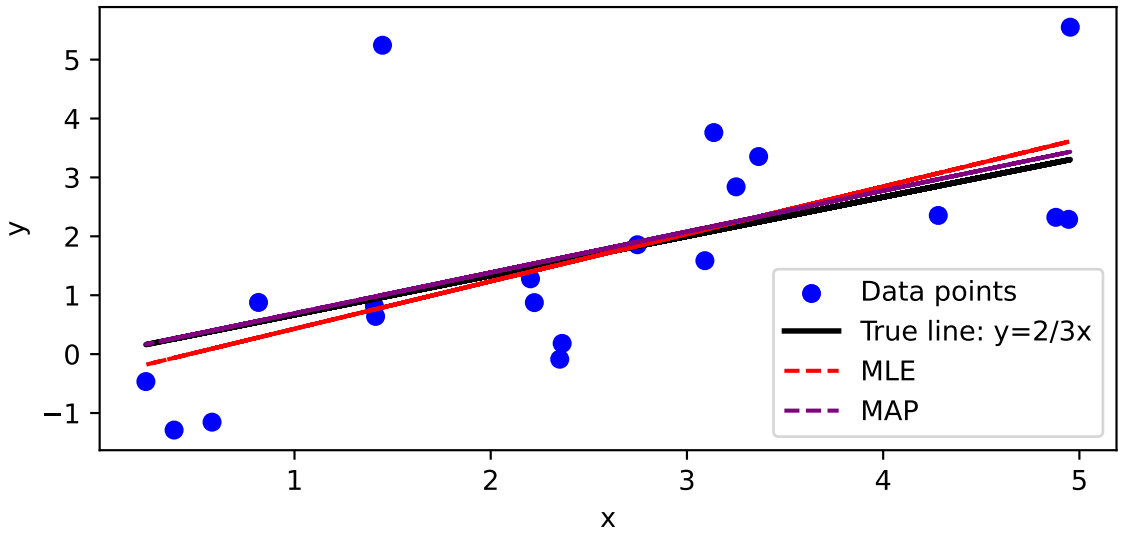


Figure 7.14: Comparison of statistical estimators in the least squares regression problem. The Bayesian MAP estimation is the most faithful. The true parameters are $a = 2/3, b = 0$. The MLE/Least-squares estimation is $a = 0.805, b = -0.374$, and the MAP estimation is $a = 0.694, b = -0.000$.

Laplace Prior and LASSO Regression. As a second example, let's assume we have a Laplace prior on $\boldsymbol{\theta}$:

$$P(\boldsymbol{\theta}) = \frac{\lambda}{2} e^{-\lambda \|\boldsymbol{\theta}\|_1}.$$

In this case, the log prior is simply $-\log P(\boldsymbol{\theta}) \propto \lambda \|\boldsymbol{\theta}\|_1$, so the log-posterior is given by

$$\log P(\boldsymbol{\theta} \,|\, x, y) = \|\mathbf{Y} - \boldsymbol{\theta}\mathbf{X}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1.$$

The second term is the sparsity-promoting regularization term in the LASSO regression problem.

Example: Linear Regression with Gaussian Prior. Figure 7.14 shows the results of an MLE and MAP estimate for the slope, a, and intercept, b, for a one-dimensional linear regression fit. If we know that the intercept is zero, $b = 0$, then this may be incorporated into the MAP estimate through a Gaussian prior $b \sim \mathcal{N}(0, \sigma^2)$, improving the regression fit.

The following code implements the MLE and MAP estimation procedures for this example.

```
from scipy.optimize import minimize

# Set seed for reproducibility
np.random.seed(45)
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
# Parameters
true_a = 2 / 3
true_b = 0
sigma_noise = 1  # Standard deviation of Gaussian noise

# Generate random x and corresponding y with Gaussian noise
n_points = 20
x = np.random.uniform(0, 5, n_points)
noise = np.random.normal(0, sigma_noise, n_points)
y = true_a * x + true_b + noise
y[19] = y[19]+3 # outlier

# Linear Least Squares Estimation
A = np.vstack([x, np.ones(len(x))]).T
theta_lsq, _, _, _ = np.linalg.lstsq(A, y, rcond=None)

# Maximum Likelihood Estimation (MLE)
def negative_log_likelihood(theta):
    a, b = theta
    y_pred = a * x + b
    residuals = y - y_pred
    return np.sum(residuals**2) / (2 * sigma_noise**2)

theta_mle = minimize(negative_log_likelihood, [1, 1]).x

# Bayesian Maximum A Posteriori (MAP) Estimation with Prior on b
# Prior: b ~ N(0, sigma_prior^2) where sigma_prior is a small value
sigma_prior_b = .01  # Strength of the prior on b
def negative_log_posterior(theta):
    a, b = theta
    log_prior_b = -0.5*(b**2/sigma_prior_b**2)  # Gaussian prior on b
    return negative_log_likelihood(theta) - log_prior_b

theta_map = minimize(negative_log_posterior, [1, 1]).x
```

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) Bayesian theory, by J. M. Bernardo and A. F. M. Smith, 2009 [33].

(2) Bayesian statistics: A review, by D. V. Lindley, 1972 [226].

(3) Bayesian inference in statistical analysis, by G. E. P. Box and G. C. Tiao, 2011 [45].

Papers and reviews

(1) On information and sufficiency, by S. Kullback and R. A. Leibler, *The Annals of Statistics*, 1951 [211].

(2) A new look at the statistical model identification, by H. Akaike, *IEEE Transactions on Automatic Control*, 1974 [4].

(3) On the mathematical foundations of theoretical statistics, by R. A. Fischer, *Philosophical Transactions of the Royal Society of London*, 1922 [129].

(4) The Bayesian approach to inverse problems, by M. Dashti and A. M. Stuart, *arXiv:1302.6989*, 2013 [89].

(5) Inverse problems: a Bayesian perspective, by A. M. Stuart, *Acta numerica*, 2010 [330].

(6) Bayesian inference in physics, by U. Von Toussaint, *Reviews of Modern Physics*, 2011 [361].

(7) Bayesian statistics and modelling, by R. van de Schoot, S. Depaoli, R. King, B. Kramer, K. Märtens, M. G. Tadesse, M. Vannucci, A. Gelman, D. Veen, J. Willemsen, *Nature Reviews Methods Primers*, 2021 [353].

(8) Optimal low-rank approximations of Bayesian linear inverse problems, by A. Spantini, A. Solonen, T. Cui, J. Martin, L. Tenorio, and Y. Marzouk, *SIAM Journal on Scientific Computing*, 2015 [324].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

Exercise 7–1: Show that if the likelihood function $p(X \mid \theta)$ is a Guassian, then the conjugate prior $p(\theta)$ is also a Gaussian.

Exercise 7–2: Consider a fair coin that is flipped $N = 10$ times. What is the probability that the coin lands heads exactly 5 times? What is the probability that the coin lands heads at least 5 times? What is the probability that the coin lands tails no more than 3 times? What is the probability that the coin lands tails no more than 3 times in the first 5 coin flips? What about the last 5 coin flips?

Exercise 7–3: How many unique 5 card hands can you draw from a deck of 52 cards if the order of the cards doesn't matter? What about if the order does matter?

Exercise 7–4: How many unique p cards can you draw from a deck of n cards if the order of the cards doesn't matter? What about if the order does matter?

Exercise 7–5: You wake up in the morning in a dark room and you need to get dressed. You randomly pull two socks out of your dresser, which contains 10 black socks and 8 red socks. What is the probability that your two socks are the same color? What is the probability that both socks are red? That both socks are black?

Exercise 7–6: Show the second equality in the following expression

$$I(\theta) = \underbrace{\mathbb{E}\left[\frac{\partial}{\partial\theta}\log f(X \mid \theta)\right]^2}_{\mathbb{E}[\ell'(\theta)]^2} = \underbrace{-\mathbb{E}\left[\frac{\partial^2}{\partial\theta^2}\log f(X \mid \theta)\right]}_{-\mathbb{E}[\ell''(\theta)]}$$

which we introduced to explain the *normalcy* property of the maximum likelihood estimator.

Exercise 7–7: Define what moments are. Derive the moments for a Normal distribution. For a Poisson distribution.

Exercise 7–8: Something about verifying the central limit theorem (add up IID from several different distributions)

Exercise 7–9: Sketch proof of CLT

Exercise 7–10: HW on bootstrap... maybe bootstrap MLE estimate and show normalcy.

Exercise 7–11: Show the limitations of MLE for a sequence of coin flips. Imagine we have a coin that flips heads three times in a row. Show the MAP distribution for various strong and weak priors.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 7–12: Emphasize that the MAP gives an entire distribution, not just a point estimate like the MLE.

Exercise 7–13: Show that the method of moments estimate for λ in the Poisson distribution is the same as the maximum likelihood estimate.

Exercise 7–14: To prove that the following values

$$\begin{aligned}\frac{\partial \ell}{\partial \mu} = 0 \quad &\implies \quad \hat{\mu} = \bar{X} \\ \frac{\partial \ell}{\partial \sigma} = 0 \quad &\implies \quad \hat{\sigma}^2 = \frac{1}{n}\sum_{j=1}^{n}\left(X_j - \bar{X}\right)^2\end{aligned}$$

maximize the log-likelihood, compute the second derivative and show that the curvature is positive.

Exercise 7–15: Suppose a coin has an unknown bias p of landing heads. We model p with a Beta prior $\mathrm{Beta}(\alpha = 2, \beta = 2)$. After observing h heads and t tails, derive the posterior distribution for p. Plot the posterior for varying h and t and interpret.

Exercise 7–16: Show that a Gaussian prior is conjugate to a Gaussian likelihood. Specifically, show that the product of these two Gaussian distributions remains a Gaussian.

Exercise 7–17: Use the fact above to design a Bayesian update scheme to iteratively estimate the parameters of an unknown Gaussian with new observed data.

Test your algorithm by generating data from a known Gaussian, with specified mean μ and variance σ^2. Initialize your prior distribution with some different values of μ and σ^2, and update these parameters with a sequence of new data. Show the convergence rate of μ and σ^2 with more data.

Repeat the process above several times, and plot the distributions of the estimated μ and σ^2 after n iterations for $n = 5, 10, 25, 50, 100$.

Exercise 7–18: Show that a binomial distribution may be written in the exponential family of distributions.

Exercise 7–19: Show that a Poisson distribution may be written in the exponential family of distributions.

Exercise 7–20: What is the conjugate prior to the Poisson distribution? Show that these distributions are prior.

Exercise 7–21: Moment generating functions (examples, properties, Gaussian, CLT, etc.)

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 8

Inverse Problems

Inverse problems are a major application domain of mathematical optimization. This topic could easily be an entire book, and so this chapter only provides a high-level overview of the field of inverse problems. This chapter could also be combined with Chapter 7 on Bayesian optimization, as the two are closely related. However, inverse problems are much more general, and Bayesian inverse problems are a special class, so they are presented as separate but related topics here. Finally, much of machine learning may be considered as an inverse problem, and these will be treated in much more detail in Chapter 10.

Inverse problems concern inferring unknown parameters and/or inputs of a system from partial observations. There are many profound examples of inverse problems in the modern world. Two of the most compelling and relatable examples are in medical imaging and computational cosmology:

- Medical imaging: Advanced medical imaging technologies, such as computed tomography (CT) scans and magnetic resonance imaging (MRI) make it possible to see inside the human body without damaging tissue with invasive surgery. This is fundamentally an inverse problem, and has revolutionized healthcare. These technologies image the body, typically via distortion of electromagnetic radiation passing through a line, or a ray, through the body. This data is then used to reconstruct a three-dimensional model, for example via the Radon transform.

  Medical imaging has saved millions of lives, enabling early detection of cancer, heart disease, and brain injuries. These tools allow doctors to diagnose conditions non-invasively, plan surgeries with precision, and monitor treatment progress.

- Astronomy: One of the oldest and most profound inverse problems has been to infer our place in the universe from imperfect and partial measurements, first from Earth with the naked eye, then with ground-based telescopes, and more recently with satellites and space probes. Much of our early mathematics and physics was discovered to improve the fit of models to observed data.

  Recently, a black hole was imaged for the first time with the Event Horizon Telescope, combining data from radio telescopes around the Earth to act as one giant telescope.

Control theory and machine learning are two additional broad classes of inverse problems that are so important as to warrant their own chapters in this book.

DRAFT

8.1 Mathematical Formulation

The fields of optimization and inverse problems are closely related, as they both involve determining solutions to often ill-posed problems under constraints. Inverse problems seek to infer unknown parameters or inputs that best match observed data, which is typically represented by an equation or system where the solution minimizes the discrepancy between observed and modeled data. Thus the goal is to find $\mathbf{x}$ such that

$$\mathbf{F}(\mathbf{x}) \approx \mathbf{y}_{\text{obs}}. \tag{8.1}$$

We typically reframe inversion as an optimization problem:

$$\min_{\mathbf{x}} \|\mathbf{F}(\mathbf{x}) - \mathbf{y}_{\text{obs}}\|^2. \tag{8.2}$$

Here, $\mathbf{F}(\mathbf{x})$ is the forward model, and $\mathbf{x}$ are the unknown parameters. When the model $\mathbf{F}$ is linear, as in linear regression from Chapter 4 or linear optimal control in Chapter 9, then the optimization problem is convex, and we may use standard regression techniques.

There are a number of challenges associated with inverse problems, including ill-posedness, model uncertainty, and complexity of the forward model $\mathbf{F}$. We will try to give a flavor for these issues, with examples, and give an idea of how they are handled in practice.

Often, the observations are partial and noisy, and thus don't contain enough information to uniquely determine the unknown parameters, making the inverse problem *ill-posed* or *ill-conditioned*. Thus, we often add constraints or regularization penalties $R(\mathbf{x})$ to ensure the solution is physical or to handle ill-posedness:

$$\min_{\mathbf{x}} \|\mathbf{F}(\mathbf{x}) - \mathbf{y}_{\text{obs}}\|^2 + \lambda R(\mathbf{x}). \tag{8.3}$$

The regularization term $R(\mathbf{x})$ may have physical meaning as encoded by a Bayesian prior (e.g., $\|\mathbf{x}\|_1$ for sparsity or $\|\mathbf{x}\|_2^2$ for smoothness). We have already examples of this in the case of least squares in Chapter 4, where we introduced the Tikhonov regularization to improve the conditioning of the matrix inversion.

There is often a mismatch between the forward model $\mathbf{F}$ and reality. For example, when optimizing the performance of an aircraft, we may rely on simulations of the aerodynamics, which only approximately match the true physics. Ill-conditioned inverse problems will be highly sensitive to these model discrepancies, necessitating careful uncertainty quantification. Aerodynamic shape optimization is also an example where the model $\mathbf{F}$ is both high-dimensional and nonlinear, which typically leads to a highly challenging, non-convex optimization problem. Typically for these problems we use adjoint methods to approximate gradients of output quantities of interest with respect to tunable inputs, and then use constrained gradient-based optimization to find locally optimal solutions. This approach requires a good initial guess, as the problem is non-convex.

These challenges are becoming tractable because of progress in machine learning and advanced optimization techniques. Machine learning provides improved forward models $\mathbf{F}$, that are both fast to evaluate and have computable gradients, from an increasing wealth of data. It is also possible to incorporate partial physics into these data-driven models, improving their generalization, and acting to regularize the problem for noisy and partial measurements. The ability to solve ill-posed and non-convex optimization problems with regularized and relaxed formulations has also improved inverse modeling.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

8.2 Classic Inverse Problems

The line between optimization and inverse problems is especially blurred for many of the classic inverse problems discussed here. For example, Newton's method is often employed specifically to find the roots of a function, which may be viewed as an inverse problem. If Newton's method is used to find the roots of $\nabla f(\mathbf{x})$, then this is an optimization problem.

Root Finding

Finding the roots of a function $f(x) = 0$ is a deceptively challenging problem and it has a rich history in mathematics. In fact, all of complex analysis and much of modern algebra was borne out of the problem of solving for the roots of a polynomial $f(x)$.

Given a polynomial function $f(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_2x^2 + c_1x + c_0$ of order d, we have well known expressions for the roots when $d \leq 4$ (e.g., the quadratic formula for $d = 2$, Cardano's formula for $d = 3$, and solutions by Ferrari and Cardano for $d = 4$) [70]. However, for $d \geq 5$ there is no general closed-form solution for the roots, making this one of the classic irreducibility results in mathematics[1].

Newton's method was originally posed specifically to solve for the roots of functions, through numerical iteration. If instead of the roots of $f(x)$ we consider the roots of the derivative $df/dx = 0$, or more generally $\nabla f(\mathbf{x}) = \mathbf{0}$, then we have the central problem of finding local minima, maxima, or saddle points in optimization. Newton's method is typically not the best numerical algorithm for finding roots of high-order polynomials, as it requires a good initial guess. For high order polynomials $f(x)$, it is common to construct a companion matrix $\mathbf{C}$, where $f(x)$ is the characteristic polynomial, and then solve for the eigenvalues of $\mathbf{C}$ using well established and numerically stable techniques from linear algebra. The companion matrix is given by

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -c_0 & -c_1 & -c_2 & \cdots & -c_{d-1} \end{bmatrix}. \tag{8.4}$$

The eigenvalues of $\mathbf{C}$ are the roots of the characteristic polynomial $f(x)$.

Numerical linear algebra techniques, such as the QR factorization [150, 153, 342] are well-studied and optimized for eigenvalue computations. They scale well to high dimensions d, are numerically well conditioned, and are included in many linear algebra packages such as LAPACK [6] and NumPy [161].

Matrix Inversion

Many physical systems may be represented as a linear system of equations,

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{8.5}$$

[1]The irrepresentability of these roots with radicals is closely related to Galois theory, which is a cornerstone of modern abstract algebra [117, 11, 328].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

where the constraint matrix $\mathbf{A}$ and vector $\mathbf{b}$ are known, and the vector $\mathbf{x}$ is unknown. Given observations $\mathbf{b}$ and a model $\mathbf{A}\mathbf{x} = \mathbf{b}$, the inverse problem is to solve for the $\mathbf{x}$ that is consistent with this observation. Matrix inversion is one of the most central themes in both optimization and inverse problems.

If $\mathbf{A}$ is a square, invertible matrix (i.e., $\mathbf{A}$ has non-zero determinant), then there is a unique solution $\mathbf{x}$ for every $\mathbf{b}$. However, when $\mathbf{A}$ is either singular or rectangular, there may be one, none, or infinitely many solutions, depending on the specific $\mathbf{b}$ and the column and row spaces of $\mathbf{A}$. Further, depending on the condition number of $\mathbf{A}$ the inversion may be ill posed. We have already seen several examples of optimization solutions to matrix inversion problems in Chapter 4, including the least squares regression in Sec. 4.1 and the Tikhonov regularized least-squares solution in Sec. 4.4 for ill-conditioned problems.

Numerical Differentiation

The problems of numerical differentiation and integration of noisy data are also classically posed as optimization problems. Numerical differentiation is particularly challenging, as it typically amplifies noise. There are several approaches to numerical differentiation involving optimization that may be more or less suitable based on the assumptions about the noise and the underlying signal.

Given a noisy signal $\mathbf{y} = [y_1, y_2, \dots, y_n]$, where each $y_j = y(x_j)$, we seek to approximate the derivative $\frac{dy}{dx}$ at each of these points $x_1, x_2, \cdots, x_n$. We will denote this derivative vector by $\mathbf{d}$, so that $d_j \approx \frac{dy}{dx}(x_j)$. For simplicity, we will assume that the data are evenly spaced, so that $x_{j+1} - x_j = \Delta x$ for all j.

Numerical differentiation is often accomplished using the method of finite-differences, for example approximating $d_j = \frac{1}{2\Delta x}(y_{j+1} - y_{j-1})$ using a central difference for $1 < j < n$ and $d_1 = \frac{1}{\Delta x}(y_2 - y_1)$ and $d_n = \frac{1}{\Delta x}(y_n - y_{n-1})$ using forward and backward differences at $j = 1$ and $j = n$, respectively. This may be written in terms of a matrix operator $\mathbf{D}$:

$$\mathbf{d} = \mathbf{D}\mathbf{y} = \frac{1}{2\Delta x}\begin{bmatrix} -2 & 2 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -2 & 2 \end{bmatrix} \mathbf{y}.$$

Because we divide by Δx, any noise or imperfection on the data $\mathbf{y}$ is amplified.

Given the derivative $\mathbf{d}$ and y_0, we may write the integration operator in matrix form as

$$\mathbf{y} = \mathbf{H}\mathbf{d} = y_0 + \Delta x \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ 1 & 1 & 1 & \cdots & 1 & 1 & 0 \end{bmatrix} \mathbf{d}.$$

This is a matrix implementation of the simple trapezoidal integration rule.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

To improve the robustness of numerical differentiation for noisy data, we pose it as an optimization problem with regularization, for example with Tikhonov regularization:

$$\mathbf{d}^* = \arg\min_{\mathbf{d}} \tfrac{1}{2}\|\mathbf{y} - \mathbf{H}\mathbf{d}\|_2^2 + \lambda\|\mathbf{d}\|_2^2. \tag{8.6}$$

Here we attempt to find the vector of derivatives $\mathbf{d}$ that best integrates to the data $\mathbf{y}$, while also having small values.

Total variation (TV) regularization [75] is another popular approach for robust numerical differentiation of noisy signals. It minimizes the trade-off between fidelity to the observed data and smoothness of the derivative, reducing the amplification of noise typically associated with differentiation. The TV-regularized derivative estimate solves:

$$\mathbf{d}^* = \arg\min_{\mathbf{d}} \tfrac{1}{2}\|\mathbf{y} - \mathbf{H}\mathbf{d}\|_2^2 + \lambda\|\mathbf{d}\|_{\text{TV}}. \tag{8.7}$$

The total variation norm is given by $\|\mathbf{d}\|_{\text{TV}} = \sum_{j=1}^{n-1} |\mathbf{d}_{j+1} - \mathbf{d}_j|$. The parameter $\lambda > 0$ controls the balance between the data fidelity term and the total variation regularization term. Note that the TV norm is closely related to the forward difference derivative operator $\mathbf{D}_{\text{FD}}$. This method is well-suited for handling piecewise-smooth signals and mitigating noise amplification in derivative estimation.

A number of numerical differentiation schemes may be found in the open source repository of Van Breugel et al. [352].

Smoothing and Filtering Noisy Data

Numerical differentiation may be viewed as a special case of smoothing or filtering noisy data. These are closely related, as smoothing is often the first step in computing a numerical derivative. There are several approaches to smoothing noisy data, and only a few methods are discussed here.

It is common to smooth noisy data by fitting it locally to a smooth spline or other polynomial fit. Depending on the amount of data, this may be accomplished in one of several ways. For large amounts of data, it is possible to take a subset of the data for fitting, where we use a least-squares regression to fit the coefficients of a polynomial model, and the remaining data is used for cross-validation. This prevents overfitting, and makes it possible to determine the correct order of the fit. If the underlying model is nonlinear in the fitting coefficients, we may use nonlinear least-squares, as discussed in Chapter 4.

Savitzky-Golay smoothing [306] is a powerful related technique, where we fit local polynomials to the data. We may then compute the analytic derivative of these polynomial fits to obtain a smoothed derivative. There are several hyper-parameters, including the polynomial order, the width of the local window, and the shape of the weighting function. It is possible to formulate this in terms of a generalized linear regression, which is also amenable to regularization with closed-form solution.

If there is a model of the underlying dynamical process generating the data, such as a linear state-space ordinary differential equation, then it is possible to use a Kalman filter [194, 327], which will be discussed more in Chapter 9.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.



Figure 8.1: Computational imaging of the human body is an important inverse problem. (left) MRI of Human Brain, image courtesy of Asnaebsa; (middle) MRI brain sagittal section, image courtesy of Chiswick Chap; (right) CT scan of normal abdomen and pelvis, image courtesy of Mikael Häggström.

8.3 Inverse Problems in Medical Imaging

One of the largest and richest classes of inverse problems involves reconstructing or improving image data from partial, noisy, or indirect measurements. The foundations of inverse problems in imaging sciences were laid well before the advent of computers and digital imaging. The field of inverse problems in imaging sciences has its roots in the 19th century, with significant developments occurring in the mid-20th century, marking the beginning of the modern era. Early contributions stem from efforts in astronomy, geophysics, and medical imaging, as researchers sought to infer hidden or indirect quantities from observable data. World War II accelerated the development of imaging technologies, such as radar and sonar, as well as related computational tools.

Two key areas that shaped the field were the mathematical work of Radon and early developments in Fourier optics. Despite their theoretical origins, both areas have become central to modern medical imaging techniques, such as X-ray computed tomography (CT) and magnetic resonance imaging (MRI); see Fig. 8.1 for examples of MRI and CT images. We will pin this discussion to problems in medical imaging, which presents some of the most impactful applications of image-based inversion. The goal is to reconstruct internal structures of the human body from indirect measurements, enabling non-invasive diagnostics. These medical applications have since driven tremendous progress in practical inversion methods.

Computed Tomography (CT) (1960s-1970s). Godfrey Hounsfield and Allan Cormack independently developed methods to reconstruct cross-sectional images from X-ray data. Their work culminated in the invention of the first CT scanner in 1972 [83, 175], which resulted in them receiving the 1979 Nobel Prize in Medicine for their contributions.

CT technology relied on the Radon transform, which was developed as a mathematical algorithm half a century earlier. At the beginning of CT, it was a major challenge to invert the Radon transform in real time on computers of the era. Techniques such as filtered

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

backprojection (FBP) were introduced to invert the Radon transform approximately. Iterative methods, such as algebraic reconstruction techniques (ART), were also developed to handle noise and incomplete data. The Kaczmarz algorithm [190] from 1937 is an iterative method for solving linear systems of equations, and decades later it became central to tomographic reconstruction. This will be a common theme in this chapter, that mathematical progress often led technological progress by decades.

Magnetic Resonance Imaging (MRI) (1970s). MRI enables the non-invasive reconstruction of high-resolution images inside the human body by measuring radio-frequency signals emitted by hydrogen nuclei excited by a strong external magnetic field. Lauterbur and Mansfield pioneered the development of MRI, earning them the 2003 Nobel Prize in Medicine [217, 239].

MRI measurements are inherently in the frequency domain, providing Fourier coefficients. Thus, the Fourier transform, developed 150 years earlier, played a critical role in reconstructing images from frequency-domain MRI data. The main early challenges included sparse sampling in k-space and noise robustness. Recent advances have included reducing the measurement time through compressed sensing techniques.

Challenges. The birth of the field was marked by a series of profound challenges, many of which remain relevant today:

- **Ill-posedness:** In 1902, Jacques Hadamard defined the three criteria for well-posed problems: 1. that a solution exists; 2. that the solution is unique; and 3. that the solution depends continuously on the data. Inverse imaging problems often violate one or more of these conditions, necessitating regularization techniques.
- **Data Sparsity:** Early imaging systems collected incomplete or noisy data, making reconstruction mathematically challenging. Even modern applications, like astronomy, involve limited and noisy measurements.
- **Computational Complexity:** The lack of computational tools in the early 20th century hindered the practical application of inverse methods. Algorithms like ART and FBP were computationally intensive before the advent of modern computers.
- **Noise Sensitivity:** Noise in the measurement data is often amplified in the reconstructed image.
- **Nonlinearity:** Many imaging problems, especially in MRI or ultrasound, are inherently nonlinear, resulting in expensive, non-convex optimization problems with complex loss landscapes.

Both CT and MRI have benefited tremendously from modern advances in computational imaging and machine learning. Deep learning, in particular, is making it possible to accelerate computation by replacing iterative optimization with fast neural network inference. Further, it is possible to learn priors so that models may be trained to encode encode anatomical priors or expected structures, improving robustness to noise and undersampling. These techniques allow for reduced radiation doses in CT and shorter scan times in MRI, improving patient outcomes and accessibility.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The Radon Transform and X-Ray Computed Tomography

In 1917, Johann Radon introduced the integral transform that bears his name, providing a mathematical basis for reconstructing functions from their integrals along lines or planes [285]. The name of his paper, translates to "*On the determination of functions from their integral values along certain manifolds*", and addressed a purely mathematical question. Radon was not motivated by practical imaging applications as we understand them today, but rather by theoretical problems in integral geometry and harmonic analysis. However, this work laid the foundation for CT and other imaging modalities, which were developed many decades later.

Radon's problem was to determine whether a function $f(x, y)$ could be uniquely reconstructed from its integrals along straight lines. In modern applications, the function $f(x, y)$ might indicate the density of a material, such as bone or tissue, that is being imaged with X-rays. The X-rays would pass through this material in straight lines, causing them to be attenuated by different amounts based on the integrated attenuation properties of the material along different paths. These attenuated, line-integrated X-rays would then be measured on the other side of the object.

A specific line may be specified by s, its perpendicular distance to the reference origin $(0, 0)$, and θ, the angle of inclination of the line with respect to the x-axis. The line will then be parameterized by a path length variable z, resulting in $(x(z), y(z)) = (z\sin(\theta) + s\cos(\theta), -z\cos(\theta) + s\sin(\theta))$. These integrals form the Radon transform:

$$\mathcal{R}(f)(s, \theta) = \int_{-\infty}^{\infty} f(x(z), y(z))\, dz = \int_{\infty}^{\infty} f(z\sin(\theta) + s\cos(\theta), -z\cos(\theta) + s\sin(\theta))\, dz.$$

Here $\mathcal{R}(f)$ is the Radon transform of f, which represents the integral of $f(x, y)$ along the line parameterized by (s, θ). This is shown in Fig. 8.2. Radon proved that a function $f(x, y)$ could be uniquely reconstructed from its line integrals under certain conditions. He also provided an explicit inversion formula, now fundamental to many imaging techniques. These transforms are closely related to the Fourier and inverse Fourier transforms.

Radon anticipated the applied challenges of ill-posedness and inversion complexity. The problem is typically ill-posed, since small errors in the measured integrals can lead to large errors in the reconstructed function. Further, Radon's explicit inversion formula, while theoretically exact, was not computationally feasible before modern computers.

Inverse Radon Transform as an Optimization Problem. There is an exact, analytic expression for the inverse Radon transform, going from $\mathcal{R}(f)(s, \theta)$ to $f(x, y)$. This is known as filtered back-projection (FBP). However, FBP is similar to numerical differentiation, making it ill-conditioned for noisy measurements or discontinuous functions $f(x, y)$. Since X-rays damage tissue, it is important to keep radiation dosages low, resulting in inherently noisy measurements. Further, interfaces between bone, tissue, and organs are all approximately discontinuous.

Thus, similar to numerical differentiation above, it is common to pose the inversion of the Radon transform as an optimization problem. Specifically, we seek the function $f(x, y)$ that minimizes the discrepancy between the observed Radon transform data $\mathcal{R}_{\text{obs}}(s, \theta)$ and the forward Radon transform $\mathcal{R}(f)(s, \theta)$ of the reconstructed function. In addition, regularization is added to improve the inversion.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

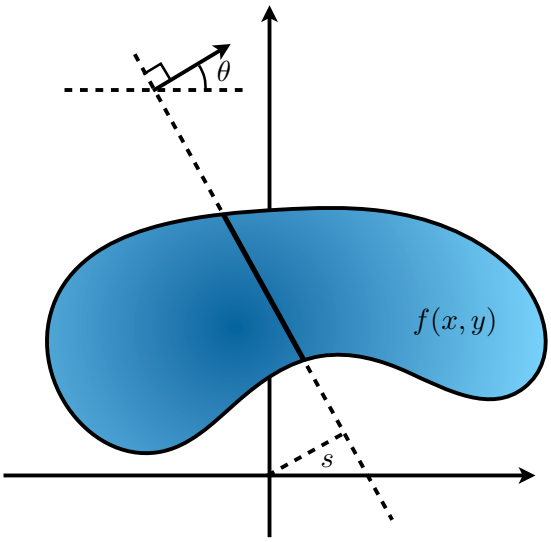


Figure 8.2: Coordinate system used to compute the Radon transform. The Radon transform is accomplished by integrating $f(x, y)$ along lines parameterized by s and θ, the shortest distance of the line to the origin, and its inclination angle from the x-axis, respectively.

The specific algorithmic approaches are somewhat involved and are not particularly intuitive or enlightening, and so we omit them here. However, the rough optimization problem may be written as

$$\min_{\mathbf{x}} \|\mathcal{R}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda R(\mathbf{x}).$$

Note that we have discretized the function $f(x, y)$ and the Radon transform $\mathcal{R}(f)(s, \theta)$ into the vectors $\mathbf{x}$ and $\mathbf{y}$, respectively. The operator $\mathcal{R}$ represents the discretized forward Radon transform, and $R(\mathbf{x})$ is the regularization term to address noise and ill-posedness.

In the inversion of the Radon transform, regularization is essential to address issues of ill-posedness, such as noise amplification and instability. Common types of regularization include Tikhonov, total variation, and sparsity. It is also possible to impose smoothness, either in a transformed domain, such as a Fourier or wavelet basis, or by penalizing the Laplacian of the function $|\nabla^2 f(x, y)|^2$. Finally, if prior knowledge about $f(x, y)$ is available (e.g., physical constraints, anatomical priors), regularization terms can encode this. The choice of regularization depends on the nature of the problem (e.g., noise type, expected smoothness) and the desired features of the reconstruction (e.g., sharp edges, sparsity). Further considerations include computational efficiency and robustness. The regularization strength λ is typically selected via cross-validation or other standard criteria.

Iterative methods, such as algebraic reconstruction techniques (ART), improve robustness but are computationally expensive. Modern deep learning techniques now incorporate priors and learned features to enhance reconstruction quality while accelerating computation.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

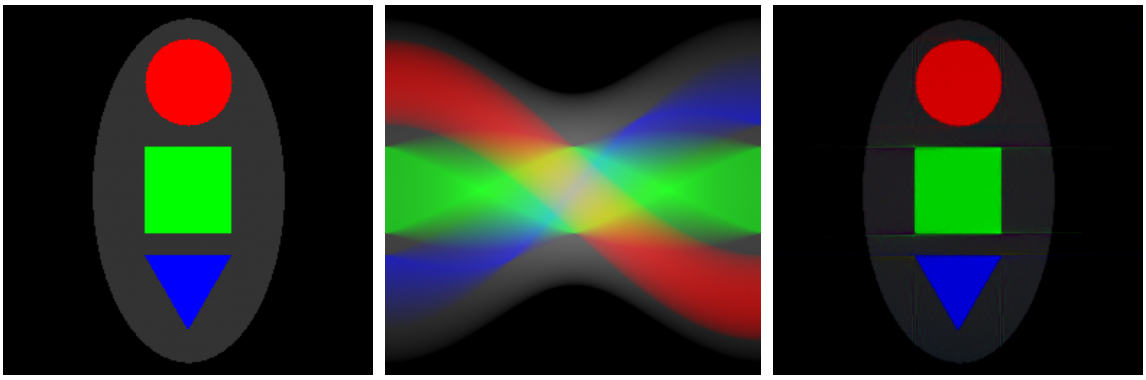


Figure 8.3: Example of Radon transform applied to simple phantom image (left). The Radon transform (middle) is inverted to achieve the reconstructed image (right). The objects are colored red, green, and blue to help illustrate which objects map to which features in the transform.

Figure 8.3 shows an example of the Radon transform, and its inverse, applied to a simple image. The input image, often referred to as a phantom, consists of a gray oval containing a circle, a square, and a triangle. Although the Radon transform yields gray-scale images corresponding to scalar intensities, here we color the circle red, the square green, and the triangle blue to make it easier for the reader to see the signature of these shapes in the transformed image.

The Fourier Transform and Magnetic Resonance Imaging

Fourier optics emerged in the 19th century as part of the broader development of wave theory, particularly the work of Joseph Fourier (1822) and Augustin-Jean Fresnel on diffraction. This field provides a framework for understanding the propagation and transformation of light as a wave, forming the basis for many imaging techniques. Inverse problems in optics are much more general than MRI, so we will briefly discuss here before focusing in on medical applications.

The central insight in Fourier optics is that a lens can act as a Fourier transform operator. When light passes through an aperture, it produces a light intensity pattern, known as a diffraction pattern, which is essentially the Fourier transform of the aperture's shape. Typically diffraction patterns are recorded after light interacts with an object or aperture. Before modern CCD cameras, light intensity patterns were captured using photographic plates or screens. The patterns recorded on a screen or photographic plate represent the Fourier transform magnitude of the object function. These patterns serve as the observable data for the inverse problem. The output is then the reconstructed object function $f(x, y)$, which could represent the structure of an aperture or an object.

The mathematical relationship between the object function f and the Fourier intensity is given by

$$I(k_x, k_y) = |\mathcal{F}[f(x, y)]|^2,$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

where $I(k_x, k_y)$ is the Fourier intensity pattern in the frequency domain, and $\mathcal{F}[f]$ is the Fourier transform of the object function $f(x, y)$.

The key challenge in Fourier optics is to reconstruct the object function $f(x, y)$ from its Fourier transform's intensity pattern. However, the intensity pattern only provides the magnitude of the Fourier transform, not the phase, making the problem nonlinear. Techniques like holography, developed in the 1940s, provided ways to recover phase information by recording interference patterns. Phase retrieval is still a challenge today, although most approaches involve computationally solving a nonlinear inverse problem. Diffraction patterns may also be noisy, requiring regularization or additional constraints for robust reconstruction.

The Fourier transform has since become a foundation for many frequency-based imaging techniques, such as MRI.

Magnetic Resonance Imaging

MRI aims to reconstruct high-resolution images of internal structures by measuring radio frequency signals emitted by hydrogen nuclei in the body under a magnetic field. Unlike X-ray CT, MRI is non-ionizing and relies on frequency-domain measurements.

The raw input data in MRI are Fourier coefficients of the spatial image, collected in k-space, where k is a spatial wavenumber. The measured signals are Fourier coefficients, because the radio-frequency emissions from spinning hydrogen nuclei in a magnetic field are detected as oscillating electromagnetic waves. The frequencies of these waves correspond to spatial variations in the magnetic field, and the inverse Fourier transform naturally maps these frequency-domain measurements in k-space to spatial-domain images. This connection arises from the physics of nuclear magnetic resonance (NMR) and the spatial encoding introduced by gradient fields.

The forward model is given by the Fourier transform, which describes the relationship between the image $\mathbf{x}$ and the measured data $\mathbf{y}$

$$\mathbf{y} = \mathbf{F}\mathbf{x} + \mathbf{n}$$

where $\mathbf{F}$ is the discrete Fourier transform operator, and $\mathbf{n}$ is noise.

The Fourier transform operator is unitary and, in general, inversion is fairly well-conditioned. However, it still helps to regularize the reconstruction problem:

$$\min_{\mathbf{x}} \|\mathbf{F}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{F}\mathbf{x}\|_1,$$

where $\|\mathbf{F}\mathbf{x}\|_1$ enforces sparsity in the Fourier transform of the image, which is a nearly universal property of natural images [56]. Unlike the more general problem of inversion in Fourier optics above, MRI measurements typically involve complex amplitude measurements, so phase retrieval from intensities is not a problem here.

The resolution of the image $\mathbf{x}$ is related to the resolution of k-space sampling in $\mathbf{y}$, and increasing the resolution of $\mathbf{y}$ increases the measurement time. Increased measurement time requires a patient to be very still for longer, and for infants this often involves anesthesia, which may have negative outcomes.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Therefore, modern MRI technologies seek to bypass measuring a full, high-resolution sampling of k-space, and instead measure a subset of frequencies k, given by the measurement vector $\mathbf{Cy}$, where $\mathbf{C} \in \mathbb{R}^{p \times n}$ is formed from $p \ll n$ rows of the identity matrix, indicating that only those p indices of $\mathbf{y}$ are sampled. In principle, this results in a highly ill-posed inverse problem to reconstruct the high-resolution image $\mathbf{x} \in \mathbb{R}^n$.

However, advances in compressed sensing [105, 68, 69] have made it possible to solve this ill-posed inverse problem with high-probability, in some instances, assuming that the image $\mathbf{x}$ is sufficiently sparse in the Fourier transform domain. In this case, it is possible to solve the following optimization problem

$$\min_{\mathbf{x}} \|\mathbf{CFx} - \mathbf{Cy}\|_2^2 + \lambda \|\mathbf{Fx}\|_1.$$

We may solve this compressed sensing problem using the methods discussed in Section 6.5. Compressed sensing MRI [233] has revolutionized our ability to obtain high-resolution MRI with considerably faster measurements.

Deep learning approaches, such as convolutional neural networks and generative adversarial networks, further enhance reconstruction speed and quality, leveraging learned priors and end-to-end optimization. Recall that the regularization terms $R(\mathbf{x})$ typically come from a prior distribution on $\mathbf{x}$, and most of the named regularization terms (e.g., Tikhonov, LASSO, etc.) come from named distributions (e.g., Gaussian, Laplace, etc.). Real-world data, like anatomical images from MRI, absolutely do follow probability distributions, but these distributions are complex and do not have names or simple parameters. Thus, learning these prior distributions from a wealth of training data, and using these priors for regularizing inverse problems, is a major advance enabled by machine learning. These learned priors and regularizers can enable super-resolution imaging from fewer measurements.

8.4 Modern Computational Image Inverse Problems

There are several modern computational image inversion techniques that may be considered as ill-posed inverse problems, where regularization is required. These techniques have rapidly become machine learning problems, with deep learning being the method of choice. We will discuss the fundamental inverse optimization problem here and provide some general approaches; a more detailed discussion of machine learning will be provided in Chapter 10.

We will describe four basic tasks here: super-resolution, which reconstructs a high-resolution image from a low-resolution image; inpainting or occlusion removal, which fills in missing or damaged portions of an image; deblurring, which recovers a crisp image from a blurred image; and change of perspective, where we reconstruct an image from another camera angle. The first three types of images are shown schematically in Fig. 8.4.

Super Resolution

The goal of super resolution is to take one or more low-resolution images $\mathbf{y}$ and recover a high-resolution image $\mathbf{x}$. Applications include satellite and biomedical imaging, consumer photography, and video surveillance.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

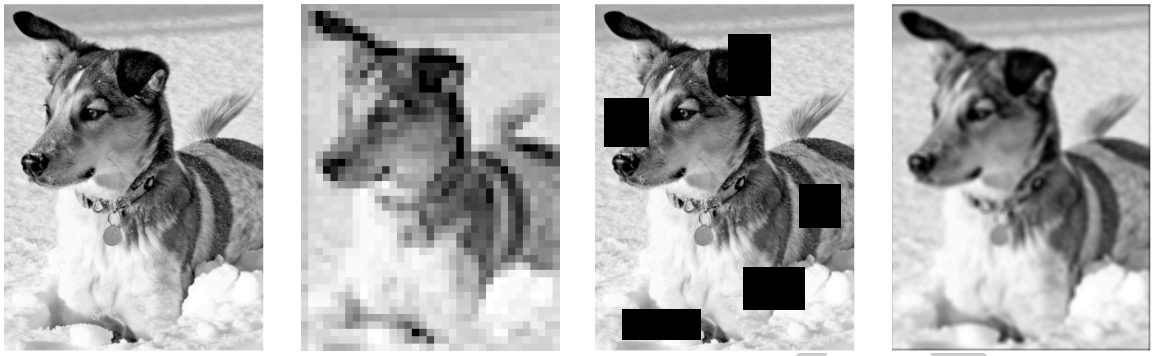


Figure 8.4: Examples of inverse problems related to image processing. From left to right: original high-resolution image; low-resolution image (goal: super-resolution); occluded image (goal: inpainting and/or repair); blurred image (goal: de-blur).

This task sounds like science fiction, as there are infinitely many high-resolution images that are consistent with a given low-resolution image. However, there is statistical structure in the space of natural images, and some high-resolution features are much more likely than others. For example, if I take off my glasses, faces appear more blurry, but I am still able to infer that eyebrows are comprised of individual hairs and that there is fine iris structure in an eye. This is because my neural network has learned these statistical features, and seamlessly fills in details. With advances in optimization and machine learning, it is becoming increasingly possible to automatically learn these statistical distributions and achieve impressive super resolution.

A common single–image super-resolution model first blurs the high-resolution image with an optical point-spread function $\mathbf{H}$, then down-samples by a factor s via a decimation operator $\mathbf{D}$, and finally adds sensor noise $\mathbf{n}$:

$$\mathbf{y} = \mathbf{DHx} + \mathbf{n}.$$

The super-resolution inverse optimization problem may then be stated as

$$\min_{\mathbf{x}} \|\mathbf{DHx} - \mathbf{y}\|_2^2 + \lambda R(\mathbf{x}).$$

Similar to the deblurring problem, the blur filter might be assumed to be unknown, in which case it may need to be learned as well, resulting in a highly non-convex blind super-resolution problem. There are also multi-frame versions that incorporate multiple low-resolution images to improve the super-resolution reconstruction.

Modern deep learning approaches [104, 365, 219, 384] now dominate super resolution performance, although there are also impressive classical approaches [133, 74, 377]. Today, CNNs, GANS, and transformers are widely used for super resolution. These techniques are also being used beyond image sciences, for example to provide sub-grid-scale models for turbulent fluid flows [134, 135].

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Repair Damage and Fill Occlusions: Image Inpainting

Image inpainting aims to reconstruct missing or damaged regions of an image. This includes applications such as restoring damaged artwork, filling in corrupted image files, and removing objects from photographs.

Consider an observed image $\mathbf{y}$ with missing or damaged pixels specified by a binary mask $\mathbf{M}$; the entries 1 correspond to good pixels, and the entries 0 correspond to bad pixels. The goal is to reconstruct a The entire image, with corrected The values of the missing pixels, represented as $\mathbf{x}$ are the unknowns.

The forward model is simply that the observed pixels match the reconstructed image:

$$\mathbf{M} \odot \mathbf{x} = \mathbf{M} \odot \mathbf{y}, \tag{8.8}$$

where the symbol $\odot$ is the Hadamard product, which is the element-wise multiplication of two matrices or vectors of the same dimensions[2].

This inversion problem may be posed as an optimization problem, where we solve for the unknown pixels $\mathbf{x}$ by minimizing:

$$\min_{\mathbf{x}} \|\mathbf{M} \odot (\mathbf{x} - \mathbf{y})\|_2^2 + \lambda R(\mathbf{x}), \tag{8.9}$$

where $R(\mathbf{x})$ enforces priors such as smoothness (e.g., total variation), sparsity, or learned image priors from machine learning. This problem is under-determined, as the missing pixels have no direct observation. Thus it is important to encode meaningful priors via regularization to recover plausible solutions.

Total variation regularization has been widely used [73], and modern deep-learning techniques such as U-Nets [301] and GAN-based approaches achieve impressive results in complex inpainting tasks. Robust principal components (RPCA) from Section 6.5 is a useful algorithm in the case that images are aligned and registered; i.e., the case where traditional SVD/PCA based image analysis is useful.

Deblurring

Another canonical imaging problem is to recover sharp images degraded by motion blur, out-of-focus optics, or atmospheric turbulence. The measurement data is a blurry image $\mathbf{y}$ and optionally a known blur kernel $\mathbf{H}$. The goal is to solve for the unknown sharp image $\mathbf{x}$, and in some cases, the blur kernel $\mathbf{H}$ if it is unknown.

The forward model convolves a sharp image with the blur kernel and adds noise $\mathbf{n}$

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}. \tag{8.10}$$

Deblurring may also be posed as an optimization problem. Given $\mathbf{y}$ and $\mathbf{H}$, we solve:

$$\min_{\mathbf{x}} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda R(\mathbf{x}), \tag{8.11}$$

where $R(\mathbf{x})$ may enforce sparsity or smoothness. Again, the problem is ill posed because there are infinitely many sharp images that give the same blurry image.

[2] If $\mathbf{A}$ and $\mathbf{B}$ are matrices of the same size, then their Hadamard product is defined as $(\mathbf{A} \odot \mathbf{B})_{ij} = \mathbf{A}_{ij} \cdot \mathbf{B}_{ij}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

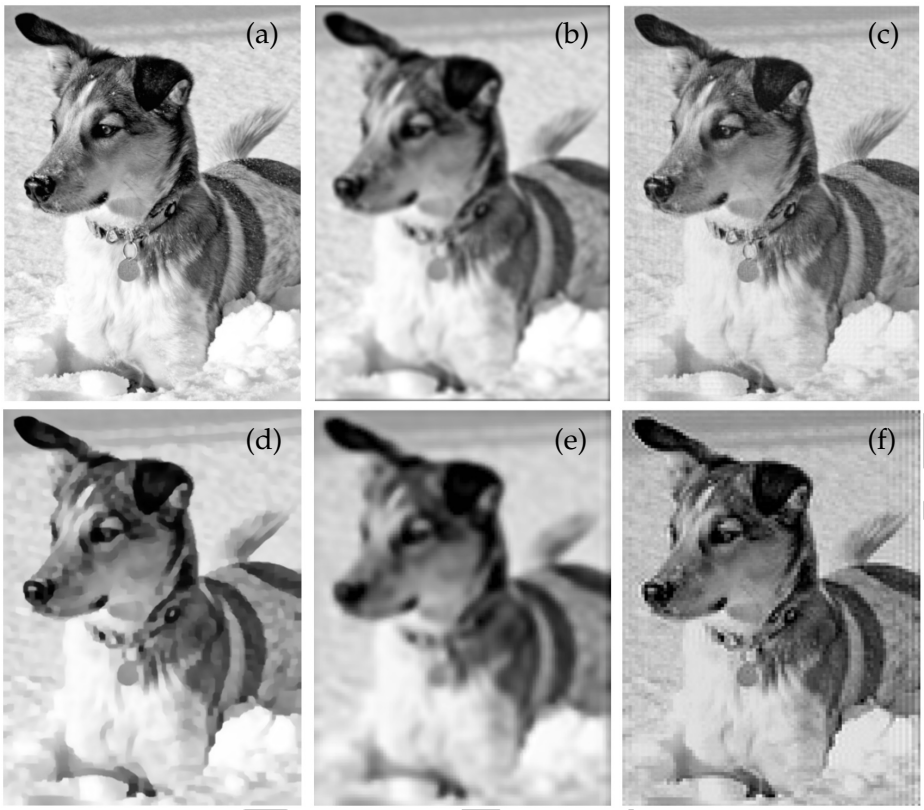


Figure 8.5: Deblurring applied to image of Mort the Snow Dog. (a) Original image, (b) blurred image, (c) least-squares reconstruction, (d) total variation regularized, (e) total variation regularized in Haar basis (f) FISTA regularization.

When the blur kernel $\mathbf{H}$ is unknown, the optimization becomes non-convex

$$\min_{\mathbf{x},\mathbf{H}} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda R(\mathbf{x}), \tag{8.12}$$

There is also the issue of computational cost, as iterative deconvolution methods can be expensive for large images. Blind deconvolution methods are widely used in photography, astronomy, and medical imaging.

Figure 8.5 shows the results of several deblurring algorithms on a blurred image of Mordecai the snow dog.

Change of Perspective and Inferring Three-Dimensional Shape

Recently, there has been incredible progress in our ability to infer a three-dimensional representation of a scene from multiple two-dimensional images, enabling fast and flexible change of perspective. The neural radiance field (NeRF) [249] method, based on deep learning, is particularly impressive.

Here the goal is to estimate the three-dimensional geometry of a scene from two-dimensional images captured from multiple viewpoints. The input data consists of multiple two-dimensional images $\mathbf{y}_j$. If the camera poses $\mathbf{P}_j$ are known, then reconstructing the

Copyright © July 15, 2025 Brunton. All Rights Reserved.



Figure 8.6: Illustration of Neural Radiance Fields (NeRFs) to reconstruct three-dimensional scene representation for rendering images from novel perspectives. Figure reproduced from Mildenhall et al. [249].

three-dimensional representation is much simpler. However, in practice, the camera poses are unknowns and must also be learned. A schematic of the NeRF approach is shown in Fig. 8.6.

Modern approaches leverage a feedforward neural network $\mathcal{N}(x, y, z)$ to predict the volume density and emitted radiance of (x, y, z) locations in a scene. From this, an image is generated using classical volume rendering techniques, for example used in video games. Both the neural network and the volume rendering are automatically differentiable (see Section **??**), so it is possible to update the neural network weights to minimize the error between the predicted and actual images using gradient descent. As this is performed over multiple viewpoints, the neural network learns a coherent model of the scene.

Mathematically, we may formalize this as follows. Let the neural network be given by the function $\mathcal{N}_{\boldsymbol{\theta}}(x, y, z)$ and the renderer by the function $\mathcal{R}$, so that $\mathcal{R}(\mathcal{N}_{\boldsymbol{\theta}}, \mathbf{P}_j)$ returns the predicted two-dimensional image from perspective $\mathbf{P}_j$. The optimization problem then becomes

$$\min_{\boldsymbol{\theta}} \|\mathcal{R}(\mathcal{N}_{\boldsymbol{\theta}}, \mathbf{P}_j) - \mathbf{y}_j\|_2^2.$$

Again, it is possible to optimize the neural network parameters $\boldsymbol{\theta}$ to minimize this loss through backpropagation. This optimization is nonlinear and high-dimensional, and solving for three-dimensional geometry from large datasets is computationally expensive. Further, a new NERF must be trained for each new scene.

8.5 Image-Based Inversion: With Physics

Image-based inversion with physics incorporates physical models and principles into the reconstruction process, leveraging domain-specific knowledge to overcome data limitations and noise. These problems involve inferring hidden properties or structures based on observable data that interact with the underlying physics of the system, such as electromagnetic waves, acoustic waves, or X-rays. This approach is fundamental in scientific applications where the physics of the imaging system is well understood, allowing for more accurate and interpretable reconstructions. Key examples include estimating the

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

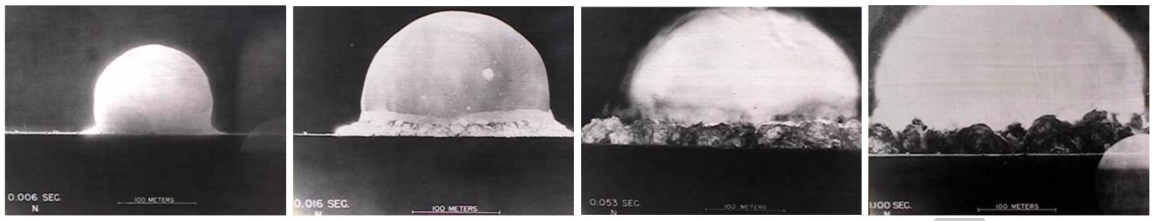


Figure 8.7: The 1945 TRINITY nuclear explosion. Rapatronic pictures taken at four times after detonation. Data on the radius over time was used by G.I. Taylor to solve the inverse problem to determine the bomb's yield, to within a relatively accurate figure. Trinity Site, Alamogordo Test Range, Jornada del Muerto desert. Courtesy of US Govt. Defense Threat Reduction Agency.

energy yield of explosions using blast wave models, reconstructing black hole images informed by relativistic effects, and determining molecular structures from X-ray diffraction patterns. These problems are characterized by their reliance on forward models rooted in physics, computational challenges in solving associated inverse problems, and the need for regularization or physical constraints to ensure meaningful results.

Yield of a Bomb from Satellite Images

The energy yield of an explosion can be estimated from satellite imagery capturing the expanding blast radius over time, a problem with significant military and scientific implications. Accurate yield estimation is critical for monitoring compliance with the comprehensive nuclear-test-ban treaty (CTBT) and understanding high-energy phenomena in controlled experiments and astrophysical contexts.

This problem dates back to the mid-20th century, when G.I. Taylor estimated the energy yield of the first nuclear tests conducted during the Manhattan Project. He was able to use a time series of images of the blast, published in a popular magazine, to back out the yield using dimensional scaling arguments. His seminal papers [336, 337] introduced the now-famous Sedov-Taylor blast wave model that connect explosion energy to the radius of the blast. See Fig. 8.7 for a sequence of images from the Trinity test.

The input data for this problem consists of time-series measurements of the blast radius, $\{R(t)\}$, derived from high-resolution satellite images taken at discrete time intervals. The unknown parameter is the explosion's energy yield, E. The forward model is based on the Sedov-Taylor blast wave theory, which relates the radius of the blast to its energy via:

$$R(t) = \alpha \left(\frac{Et^2}{\rho} \right)^{1/5}, \tag{8.13}$$

where α is a scaling constant, ρ is the ambient air density, and t is the time since detonation. For air, the constant α can be shown to be quite close to 1, and may be neglected. Using this model, the inversion problem is posed as an optimization:

$$\min_E \sum_k \left| R(t_k) - \alpha \left(\frac{Et_k^2}{\rho} \right)^{1/5} \right|^2 .$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

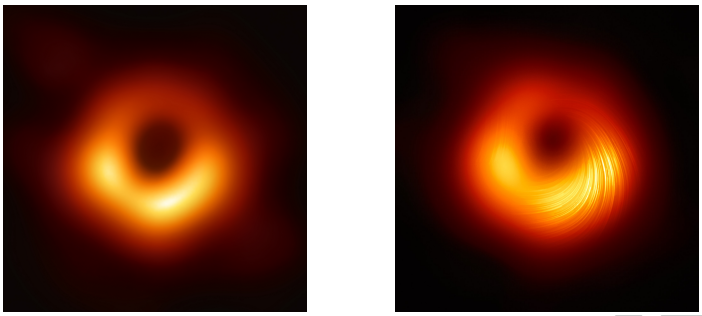


Figure 8.8: Computational images of the Messier 87 black hole, based on data collected from the Event Horizon Telescope. The image on the right is imaged using polarized light. Images courtesy of EHT Collaboration.

This problem presents several challenges. The fifth-root dependence on E makes the optimization nonlinear and sensitive to noise in the observed data, often collected under less-than-ideal conditions. Additionally, the Sedov-Taylor model assumes isotropic explosions and requires accurate ambient density, ρ, which may not be available. Despite these challenges, this approach has been applied to estimate the yields of historical nuclear tests, such as the Trinity test, and remains relevant for modern satellite-based explosion monitoring systems. The same principles extend to astrophysical applications, including the analysis of supernova explosions.

Mass of a Black Hole from Satellite Images

In 2019, the Event Horizon Telescope (EHT) collaboration achieved a groundbreaking milestone by producing the first image of a black hole, M87* [80]. This achievement confirmed key predictions of general relativity under extreme conditions and provided new insights into the dynamics of black holes and their accretion disks. The core problem involved reconstructing the brightness distribution surrounding a black hole and inferring its mass from sparse radio telescope measurements. Figure 8.8 shows the computationally reconstructed images of the M87 supermassive black hole.

The input data are sparse Fourier-domain measurements of electromagnetic emissions from the black hole, obtained using very long baseline interferometry (VLBI), which combines signals from a global array of radio telescopes. The unknowns include the brightness distribution $\mathbf{I}(M, \theta)$, and the black hole's mass, M, and spin and orientation θ. General relativistic constraints relate the mass M to features of the brightness distribution, such as the size and shape of the black hole's shadow. It is possible to construct an image $\mathbf{x}$ from the brightness distribution $\mathbf{I}(M, \theta)$ through general relativistic ray tracing. The forward model connects then connects the brightness distribution to the observed data via the Fourier transform $\mathbf{F}$:

$$\mathbf{y} = \mathbf{CFI}_{\text{GR}}(M, \theta).$$

The matrix $\mathbf{C}$ is a sparse sampling operator, as only a few Fourier frequencies are measured. The problem is framed as a joint optimization to reconstruct estimate M and θ,

Copyright © July 15, 2025 Brunton. All Rights Reserved.

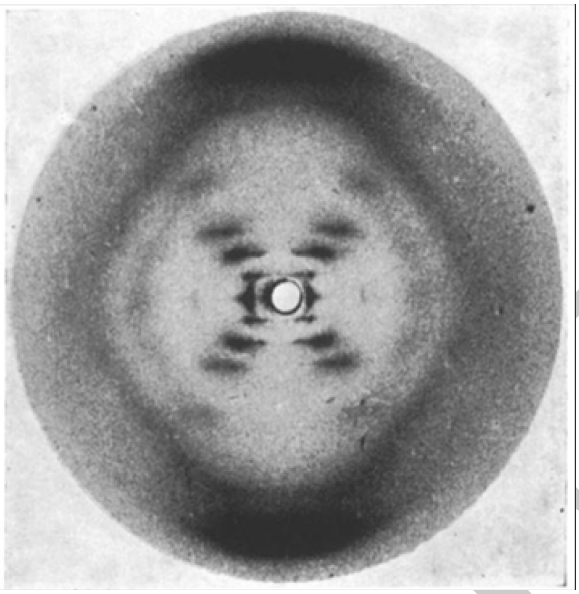


Figure 8.9: X-ray chrystallography image of DNA, first taken by Rosalind Franklin. Reproduced by Maria Evagorou, Sibel Erduran, Terhi Mäntylä. This image was used by Watson and Crick without Franklin's permission. They went on to win the Nobel Prize for Physiology or Medicine in 1962; Franklin died of cancer before the Nobel prize was awarded.

from which we can derive $\mathbf{I}(M, \theta)$ and reconstruct $\mathbf{x}$:

$$\min_{M,\theta} \|\mathbf{CFI}_{\text{GR}}(M, \theta) - \mathbf{y}\|_2^2.$$

This is known as the model-based fit, where we first derive the parameters of a model.

An alternative is to directly learn the image $\mathbf{x}$, which is regularized to be consistent with general relativistic considerations based on the mass M and other factors:

$$\min_{\mathbf{x}} \|\mathbf{CFx} - \mathbf{y}\|_2^2 + \lambda R(\mathbf{x}) + \mu R_{\text{GR}}(\mathbf{x}, M).$$

The terms $R(\mathbf{x})$ include generic image regularizers (e.g., TV, sparsity, etc.), while $R_{\text{GR}}(\mathbf{x}, M)$ include general relativistic priors, for example penalizing images who ring diameter differs from $9.6GM/(c^2D)$).

In practice, a combination of both approaches was used to image the black hole M87* and later Sagittarius A*, the Milky Way's central black hole.

X-ray Crystallography

X-ray crystallography is one of the most impactful tools in structural biology and chemistry, enabling the discovery of molecular structures such as DNA, proteins, and complex drugs. This method has revolutionized our understanding of biomolecular mechanisms, from enzyme function to drug design. The origins of X-ray crystallography date back to the early 20th century, with pioneering contributions by Max von Laue [359], who first demonstrated X-ray diffraction by crystals, and later by Bragg [49], who developed Bragg's

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

law to relate diffraction patterns to crystal structures. These efforts laid the groundwork for the Nobel prize discovery of DNA's double-helix structure by Watson and Crick, based on diffraction data by Rosalind Franklin and Maurice Wilkins [367].

The input data are diffraction intensities, $I(\mathbf{k})$, corresponding to the squared magnitudes of the Fourier transform of the electron density:

$$I(\mathbf{k}) = |\mathcal{F}[\rho(\mathbf{r})]|^2.$$

The goal is to reconstruct $\rho(\mathbf{r})$ by solving the phase retrieval problem, which arises because only the magnitudes of the Fourier coefficients are measured, not their phases:

$$\min_{\rho} \| |\mathcal{F}[\rho]|^2 - I(\mathbf{k}) \|_2^2.$$

This problem is challenging due to the phase ambiguity, making it nonlinear and underdetermined. Regularization techniques or prior knowledge, such as sparsity or physical constraints, are often required to make the problem well-posed. There are several ways that physics enters the optimization. First, knowing the chemical composition helps, as each atom contributes a known scattering pattern. Crystal symmetry further links various Fourier coefficients, greatly reducing the number of unknown phases. Further, electron density is strictly non-negative. Known molecular fragments or building blocks are also used as templates to guide the search through structures. X-ray crystallography continues to drive breakthroughs in understanding biomolecular structures, from enzymes to viral proteins, and remains an essential tool in drug discovery and materials science.

8.6 PDE-Constrained Inversion

Inverse problems constrained by partial differential equations (PDEs) lie at the intersection of physics, mathematics, and computational science. These problems involve inferring unknown parameters or properties of a system by fitting observations to models governed by PDEs. Applications span a wide range of scientific and engineering fields, including geophysics, fluid dynamics, and biomedical engineering. The key challenge is that solving such problems requires the repeated evaluation of computationally expensive PDE models, often leading to ill-posed and high-dimensional optimization tasks. Regularization techniques and advanced computational methods, such as adjoint-based gradient computation, play a crucial role in making these problems tractable. This section highlights two canonical examples: seismic inversion, where subsurface geophysical properties are inferred from seismic wave propagation and surface measurements, and the classic "Can you hear the shape of a drum?" problem, which explores the relationship between geometric shapes and their spectral signatures.

More generally, PDE-constrained design optimization can be seen as a related problem. For example, optimizing the shape of a wing to maximize lift and minimize drag is a complex design optimization, where we essentially have an expensive PDE-based forward model from wing shape to aerodynamic properties, but not the inverse model.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Seismic Inversion

Seismic inversion seeks to infer the Earth's subsurface properties, such as density and seismic velocity, from surface-based seismic wave measurements. This problem is central to geophysics, with applications in oil and gas exploration, earthquake monitoring, and carbon sequestration. By analyzing how seismic waves propagate through different subsurface materials, we can reconstruct detailed velocity models, revealing geological structures and potential resource reservoirs.

The input data consists of time-series of seismic wave amplitudes recorded at multiple receiver locations, typically from controlled sources, like explosions, or sources that may be triangulated, like earthquakes or eruptions. The unknown parameter is the spatially varying velocity model $c(\mathbf{x})$, which governs how seismic waves propagate. The forward model for this problem is based on the acoustic wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u,$$

where $u = u(\mathbf{x}, t)$ is the wavefield, $c = c(\mathbf{x})$ is the velocity, and the boundary conditions represent the Earth's surface and subsurface interfaces.

In seismic inversion, we aim to match observed data $\mathbf{y}$ with synthetic data generated by solving the forward model for a given $c(\mathbf{x})$. This is posed as an optimization problem:

$$\min_c \|\mathbf{F}(c) - \mathbf{y}\|_2^2,$$

where $\mathbf{F}(c)$ is a forward solver for the wave equation, typically implemented numerically using finite-difference, finite-element, or spectral methods.

Seismic inversion is high-dimensional, non-convex, and ill-posed, making it extremely challenging. The nature of the acoustic equation make it so that small errors in the recorded data can lead to significant errors in the estimated velocity model. Regularization methods, such as Tikhonov regularization or total variation, are often employed to stabilize the inversion. Further, the velocity model $c(\mathbf{x})$ is a three-dimensional field, and higher desired resolution leads to a extremely large optimization problems with millions or billions of parameters. Finally, the wave equation's nonlinear relationship with c results in multiple local minima, complicating gradient-based optimization. Techniques like full waveform inversion (FWI) [356] aim to mitigate these challenges by using more accurate physics and multi-scale approaches.

Traditional seismic inversion relies heavily on solving computationally expensive PDEs, such as the wave equation, which can be prohibitively time-consuming for large-scale, high-resolution models. Machine learning mitigates these challenges by learning surrogate models, which approximate the forward solver or map seismic data directly to subsurface properties. These surrogates, trained on synthetic or real datasets, dramatically reduce computational costs while maintaining accuracy [3, 364]. Further, automatic differentiation of these surrogates allow them to be flexibly tuned to data or be used to enable faster gradient based optimization.

Hybrid approaches that integrate machine learning with physics-based solvers have become particularly promising. Physics-informed neural networks (PINNs) [286] enforce

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

physical laws, such as the wave equation, as constraints during training; we will discuss PINNs more in Chapter 10. This ensures that learned models respect the underlying physics, even when trained on limited data. For example, PINNs have been used to solve seismic inversion problems by embedding the wave equation into the loss function, reducing the reliance on labeled training data [322].

Another popular hybrid approach involves combining full waveform inversion (FWI) with neural networks. In these frameworks, machine learning models act as regularizers or preconditioners for traditional optimization methods, improving convergence and reducing sensitivity to local minima [376]. In addition, HPC advancements, such as GPU-accelerated solvers and parallel processing, enable large-scale simulations and optimization in realistic 3D environments, which were previously infeasible [356].

Although full wave inversion for the velocity $c(\mathbf{x})$ is already a major challenge, it is also possible to pose an optimization to solve for the velocity $c(\mathbf{x})$ as well as the density field $\rho(\mathbf{x})$. This involves using the full elastic wave equation

$$\rho(\mathbf{x})\frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla \cdot [\mathbf{C}(\mathbf{x}) : \nabla \mathbf{u}]$$

as the forward model. However, this is often more involved than it is practical. Further, the velocity field often correlates empirically with relevant lithology, porosity, and fluid saturation. Sharp velocity drops often flag gas pockets, and high velocities can indicate carbonates or cemented sands. Further, rock physics models can help to improve models, adding physical regularization.

Can You Hear the Shape of a Drum?

The question "Can you hear the shape of a drum?" was famously posed by Mark Kac in 1966 [189]. It refers to the mathematical problem of determining the shape of a domain from its vibrational frequencies, or eigenvalues of the Laplacian. This problem, rooted in mathematical physics, has implications for quantum mechanics, shape analysis, and modern machine learning.

The membrane of our drum is give by a bounded domain Ω, which determines the geometry of the problem. The wave equation on this domain is

$$u_{tt} = c^2 \nabla^2 u \quad u_{\partial\Omega} = 0. \tag{8.14}$$

The solution is separable in time and space, so $u(\mathbf{x}, t) = \sum_j \phi_j(\mathbf{x}) \cos(\omega_j t)$, where $f_j = \frac{\omega_j}{2\pi}$ is the audible frequency associated with the j-th spatial mode ϕ_j. The spatial mode satisfies the Helmholtz equation

$$\nabla^2 \phi_j + \frac{\omega_j^2}{c^2}\phi_j = 0 \quad \Longrightarrow \quad \nabla^2 \phi_j + \lambda_j \phi_j = 0 \tag{8.15}$$

with spacial eigenfrequencies $\lambda_j = \frac{\omega_j^2}{c^2}$ and Dirichlet boundary conditions $\phi_j(\mathbf{x})|_{\partial\Omega} = 0$. Thus, the spatial eigenfrequencies λ_j of the Laplace operator on the bounded domain Ω with Dirichlet boundary conditions are directly related to the acoustic frequencies $f_j = \frac{\omega_j}{2\pi}$.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

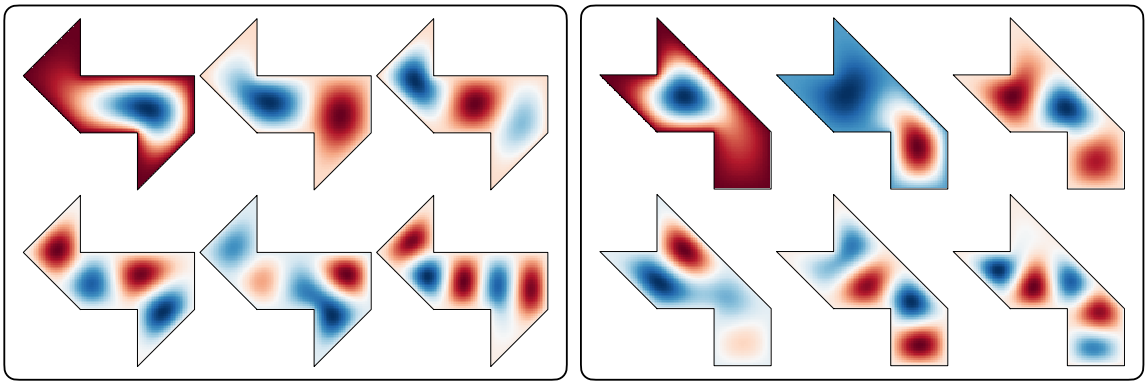


Figure 8.10: Eigenmodes of the Helmholtz equation (8.15) for two different geometries. Although the geometries are not related by simple symmetry transformations (e.g., rotations, reflections, etc.), the spectrum of the Helmholtz equation is identical for these two geometries. This means that they are *isospectral* and cannot be distinguished just by their sound. This example was discovered by Gordon, Webb, and Wolpert [156].

The input data for the optimization problem are the measured eigenfrequencies λ_j, which are computed directly from the acoustic frequencies f_j. We then solve for the unknown shape of the domain, Ω. The forward model is the Helmholtz equation (8.15) from above. The inverse problem is to match the observed eigenvalues λ_j from the model eigenvalues $\hat{\lambda}_j(\Omega)$ given a particular shape Ω.

$$\min_{\Omega} \sum_{j} |\lambda_j - \hat{\lambda}_j(\Omega)|^2.$$

This problem is inherently non-unique. In 1992, Gordon, Webb, and Wolpert [156] showed that distinct domains can have identical eigenvalues, known as iso-spectral domains. Their geometries are shown in Fig. 8.10 along with the eigenmodes of the Helmholtz equation. This introduces inherent ambiguity into the problem. However, adding physical priors or reducing the search space may help make the problem unique. It is also ill-posed, in the sense that small perturbations in the eigenvalues can correspond to vastly different geometries. Regularization techniques, such as penalizing high-frequency oscillations, are essential. Finally, representing and optimizing over domain shapes is computationally intensive, often requiring parameterization techniques like level sets or Fourier descriptors.

Despite these challenges, the problem of determining the shape of a domain from its spectral properties remains highly relevant, both mathematically and in practical applications. Advances in computational geometry, numerical optimization, and high-performance computing have all driven progress in this field. One notable area is spectral graph theory, where the eigenvalues of the graph Laplacian encode structural information about a graph, such as connectivity, community structure, and robustness. These spectral properties are used to solve problems in network analysis, such as detecting communities, ranking nodes, and designing efficient network flows [78, 325].

In machine learning, spectral embeddings derived from Laplacian eigenvalues are foundational for dimensionality reduction and clustering algorithms, such as spectral clus-

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

tering [266]. These methods leverage the intuition that the eigenvectors corresponding to the smallest nonzero eigenvalues of the graph Laplacian capture meaningful geometric relationships in high-dimensional data. The mathematical framework provided by Kac's question has inspired new techniques for analyzing and manipulating complex shapes, including shape matching, deformation, and optimization in computer graphics [51].

The interplay between domain shape and spectral properties have far reaching implications for engineering optimization and inverse problems. For example, modern shape optimization techniques incorporate spectral properties to design objects with specific vibration characteristics, such as minimizing noise in mechanical structures or maximizing efficiency in acoustic devices [29].

8.7 System Identification

Another common inverse problem is to determine a dynamical system underlying an observed time series data set. These learned dynamical systems may be used for forecasting (e.g., predicting the future), estimation, and control. There is a rich history in system identification, going back over half a century, and so here we will only present a high-level sketch of the field, with a focus on recent trends. For more detail, there are several more complete references [56, 55, 311].

Although the earliest techniques in system identification were not posed explicitly as optimization problems, it was quickly realized that optimization was essential to the underlying algorithms. There are several key dichotomies in system identification that determine which optimization technique to use:

- Linear versus nonlinear dynamics: linear system identification is considerably simpler and more structured, often leading to numerically robust, convex optimization problems based on least-squares regression.
- Partial versus full measurements: system identification based on partial measurements makes the inverse problem inherently less well posed than when full-state measurements are available.
- Controlled versus autonomous systems: when a system is being actively controlled, there is ambiguity about what are internal state dynamics versus the affect of actuation, unless the actuation signal is known.
- High-dimensional versus low-dimensional systems and/or data: the dimension of the system identification problem often determines how expensive and ill-conditioned the inverse problem is.

The field of linear system identification is much more mature [184, 311, 55], with decades of progress, and so we will begin with the linear identification problem. However, nonlinear system identification is experiencing rapid progress [43, 313, 59, 84], especially because of improved optimization algorithms and function approximation techniques from machine learning. Here we will primarily discuss the simplified case of learning linear and nonlinear dynamical systems with full measurements and without control inputs, although extensions to controlled and partially measured systems will be mentioned.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Linear System Identification

A linear dynamical system, with control inputs and output measurements, is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{8.16a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{8.16b}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state of the system, $\mathbf{u} \in \mathbb{R}^q$ is the external input (e.g., a control input), and $\mathbf{y} \in \mathbb{R}^p$ is the measurement of the system. In these equations, all variables depend on time, so $\mathbf{x} = \mathbf{x}(t)$, etc., although we typically drop the explicit dependence on time, as this is clear from the fact that we are learning a differential equation.

Linear system identification is an inverse problem of identifying the system matrices $\{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ from input–output data $\{\mathbf{u}(t), \mathbf{y}(t)\}$. More commonly, we will have data sampled discretely in time, $\{\mathbf{u}_k, \mathbf{y}_k\}$, where $\mathbf{u}_k = \mathbf{u}(k\Delta t)$ and $\mathbf{y}_k = \mathbf{y}(k\Delta t)$. We may then learn the continuous-time dynamics in Eq. (8.16), or a discrete-time dynamical system that advances the state and measurements forward by one Δt every iteration.

Typically, it is more challenging to learn the dynamical system from partial observations $\mathbf{y}$, as opposed to full-state measurements $\mathbf{x}$. However, learning linear systems from partial measurements of the inputs and outputs, $\{\mathbf{y}, \mathbf{u}\}$, has largely been a solved problem since the 1980s [171, 253, 184, 185]. The well-posedness of this inverse problem relates to whether or not the matrix pairs $(\mathbf{A}, \mathbf{B})$ and $(\mathbf{A}, \mathbf{C})$ are controllable and observable, respectively [56], and more specifically to the condition number of the controllability and observability Gramians.

If there is no control input and there are full state measurements, so that $\mathbf{y} = \mathbf{x}$, then we have a simpler inverse problem of learning the operator $\mathbf{A}$ in

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x}. \tag{8.17}$$

The dynamic mode decomposition (DMD) is a modern approach to full-state system identification of such systems, when the state $\mathbf{x}$ is high-dimensional [310, 302, 334, 311], as is the case for fluid flows, neuroscience, epidemiology, and many other spatio-temporal systems. DMD is deeply connected to Koopman operator theory [302, 55], where it is often possible to lift a nonlinear system into higher dimensions where the dynamics become linear.

All modern DMD algorithms are formed as optimization problems. The idea is to use data matrices and linear least-squares regression to find an approximate $\mathbf{A}$ that minimizes the error of the fit in some matrix norm.

DMD is inherently data-driven, and the first step is to collect a number of pairs of snapshots of the state of a system as it evolves in time. These snapshot pairs may be denoted by $\{(\mathbf{x}(t_k), \mathbf{x}(t'_k))\}_{k=1}^m$, where $t'_k = t_k + \Delta t$, and the time-step Δt is sufficiently small to resolve the highest frequencies in the dynamics. As before, a snapshot may be the state of a system, such as a three-dimensional fluid velocity field sampled at a number of discretized locations, which is reshaped into a high-dimensional column vector. These snapshots are then arranged into two data matrices, $\mathbf{X}$ and $\mathbf{X}'$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}(t_1) & \mathbf{x}(t_2) & \cdots & \mathbf{x}(t_m) \\ | & | & & | \end{bmatrix}, \quad \mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}(t'_1) & \mathbf{x}(t'_2) & \cdots & \mathbf{x}(t'_m) \\ | & | & & | \end{bmatrix}.$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The original formulations of Schmid [310] and Rowley et al. [302] assumed uniform sampling in time, so that $t_k = k\Delta t$ and $t'_k = t_k + \Delta t = t_{k+1}$. If we assume uniform sampling in time, we will adopt the notation $\mathbf{x}_k = \mathbf{x}(k\Delta t)$.

The DMD algorithm seeks the leading spectral decomposition (i.e., eigenvalues and eigenvectors) of the best-fit linear operator $\mathbf{A}$ that relates the snapshots in time:

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X}.$$

The best-fit operator $\mathbf{A}$ then establishes a linear dynamical system that best advances snapshot measurements forward in time. If we assume uniform sampling in time, this becomes

$$\mathbf{x}_{k+1} \approx \mathbf{A}\mathbf{x}_k.$$

The best-fit operator $\mathbf{A}$ is defined as the solution to the following optimization problem:

$$\mathbf{A} = \underset{\mathbf{A}}{\operatorname{argmin}} \|\mathbf{X}' - \mathbf{A}\mathbf{X}\|_F = \mathbf{X}'\mathbf{X}^\dagger, \tag{8.18}$$

where $\|\cdot\|_F$ is the Frobenius norm and † denotes the pseudo-inverse.

There are several variants of DMD, such as DMD with control (DMDc) [283], which is closely related to linear subspace identification methods, such as the eigensystem realization algorithm (ERA) [184]. The optimized DMD algorithm generalizes the optimization framework of exact DMD to perform a nonlinear regression to exponential-time dynamics, thus providing an improved computation of the DMD modes and their eigenvalues [12].

Physics Informed DMD. The physics-informed DMD has several approaches to enforce known or partially known physics, such as symmetries, in the DMD regression [17]. These physics are either enforced as constraints in the optimization, such as manifold constraints, or promoted by adding terms to the loss function. In the case of manifold-constrained DMD, the optimization problem becomes a modified Procrustes problem:

$$\min_{\mathbf{A}} \|\mathbf{X}' - \mathbf{A}\mathbf{X}\|_F \quad \text{subject to} \quad \mathbf{C}(\mathbf{A}) = 0, \tag{8.19}$$

where $\mathbf{C}(\mathbf{A})$ represents a set of constraints on $\mathbf{A}$ that encode physical properties such as conservation laws, symmetries, or invariants. Some examples of the constraints are:

- **Conservative:** The original Procrustes problem involved searching for unitary, conservative $\mathbf{A}$ satisfying $\mathbf{A}^*\mathbf{A} = \mathbf{I}$, so $\mathbf{C}(\mathbf{A}) = \mathbf{A}^*\mathbf{A} - \mathbf{I}$.
- **Translationally shift-equivariant:** Traveling wave physics, such as the wave equation or the Navier-Stokes equations are translationally invariant, even though the basic DMD solution is typically not. These systems require circulant or block-circulant $\mathbf{A}$ matrices.
- **Self-adjoint (symmetric):** Solutions to the Schrödinger equation are self-adjoint, so that $\mathbf{A}^* = \mathbf{A}$, restricting our solutions to Hermitian matrices. The constraint is then $\mathbf{C}(\mathbf{A}) = \mathbf{A}^- \mathbf{A}^*$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

- **Skew-symmetry:** Similiarly, for skew-symmetric operators, the constraint $\mathbf{C}(\mathbf{A}) = \mathbf{A} + \mathbf{A}^*$ enforces this structural property.

By enforcing these constraints, piDMD ensures that the resulting operator $\mathbf{A}$ not only minimizes the data reconstruction error but also adheres to physical principles, leading to models that are more interpretable and robust.

Nonlinear System Identification

Similarly, the general form of a nonlinear dynamical system is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{8.20a}$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}). \tag{8.20b}$$

In nonlinear system identification, the goal is still to learn the dynamics $\mathbf{f}$ and the output equations $\mathbf{g}$ from the time series data $\{\mathbf{u}(t), \mathbf{x}(t), \mathbf{y}(t)\}$. Often, we will have access to the full state $\mathbf{x}(t)$, but in the most challenging cases, we will only have input–output data $\{\mathbf{u}(t), \mathbf{y}(t)\}$.

The structure of the nonlinear functions $\mathbf{f}$ and $\mathbf{g}$ is much less constrained than in linear system identification, and so typically we must first define a parametrized family of functions, $\mathbf{f}_{\boldsymbol{\xi}}(\mathbf{x}, \mathbf{u})$ and learn the parameters $\boldsymbol{\xi}$ using optimization. For simplicity we will focus on the case with full measurements $\mathbf{y} = \mathbf{x}$ and no control input $\mathbf{u}$, as above, although extensions exist to relax each of these assumptions [60].

The sparse identification of nonlinear dynamics (SINDy) algorithm [59] leverages the fact that many dynamical systems

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}) \tag{8.21}$$

have dynamics $\mathbf{f}$ with only a few active terms, out of all possible right-hand side functions.

We then seek to approximate $\mathbf{f}$ by a generalized linear model

$$\mathbf{f}(\mathbf{x}) \approx \sum_{k=1}^{p} \theta_k(\mathbf{x})\xi_k = \boldsymbol{\Theta}(\mathbf{x})\boldsymbol{\xi},$$

with the fewest non-zero terms in $\boldsymbol{\xi}$ as possible. It is then possible to solve for the relevant terms that are active in the dynamics using sparse regression [163, 179, 339, 387] that penalizes the number of terms in the dynamics and scales well to large problems. We have already seen examples of this in the LASSO regression from Chapter 4 and the sparse optimization algorithms in Section 6.5.

Similar to DMD above, time-series data is collected from (8.21) and formed into a data matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}(t_1) & \mathbf{x}(t_2) & \cdots & \mathbf{x}(t_m) \end{bmatrix}^T.$$

A similar matrix of derivatives is formed:

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}(t_1) & \dot{\mathbf{x}}(t_2) & \cdots & \dot{\mathbf{x}}(t_m) \end{bmatrix}^T.$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

In practice, this may be computed directly from the data in $\mathbf{X}$; for noisy data, the total-variation regularized derivative tends to provide numerically robust derivatives [75]. Alternatively, it is possible to formulate the SINDy algorithm for discrete-time systems $\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k)$, as in the DMD algorithm, and avoid derivatives entirely.

A library of candidate nonlinear functions $\boldsymbol{\Theta}(\mathbf{X})$ is constructed from the data $\mathbf{X}$:

$$\boldsymbol{\Theta}(\mathbf{X}) = \begin{bmatrix} \mathbf{1} & \mathbf{X} & \mathbf{X}^2 & \cdots & \mathbf{X}^d & \cdots & \sin(\mathbf{X}) & \cdots \end{bmatrix}.$$

Here, the matrix $\mathbf{X}^d$ denotes a matrix with column vectors given by all possible time series of dth-degree polynomials in the state $\mathbf{x}$. In general, this library of candidate functions is only limited by one's imagination.

The dynamical system may now be represented in terms of the data $\mathbf{X}$ and $\dot{\mathbf{X}}$ as

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}. \tag{8.22}$$

Each column $\boldsymbol{\xi}_k$ in $\boldsymbol{\Xi}$ is a vector of coefficients determining the active terms in the k-th row in (8.21). A parsimonious model will provide an accurate model fit with as few terms as possible in $\boldsymbol{\Xi}$. This is achieved by seeking solutions that minimize the following loss function

$$\min_{\boldsymbol{\Theta}} \|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\| + \lambda\|\boldsymbol{\Xi}\|_0. \tag{8.23}$$

Here λ is a sparsity-promoting hyper parameter. The term $\|\boldsymbol{\Xi}\|_0$ counts the non-zero entries in $\boldsymbol{\Xi}$, and it is neither convex nor strictly a norm. This sparse regression problem is NP-hard and it must be relaxed, either by replacing $\|\boldsymbol{\Xi}\|_0$ with $\|\boldsymbol{\Xi}\|_1$ as in LASSO [339], or through a sequentially thresholded least-squares (STLS) procedure [59], which has been proven to converge [381]. The sequentially thresholded least-squares algorithm has since been demonstrated to be a formal relaxation of the ℓ_0 optimization problem, using proximal gradient formulation [383] discussed in Section 6.5.

There are several powerful extensions of the SINDy framework, including extensions to include control inputs [192], to disambiguate linear and nonlinear dynamics [16], to enforce symmetry constraints [230], and to promote global stability [195]. Enforcing symmetry constraints and stability both involve modifying the optimization problem, either with hard constraints, or with additional terms, and this will be discussed more in Section **??**. The STLS procedure is particularly appealing, because it involves a sequential series of least-squares regressions, so it is straightforward to add equality constraints at each stage [230].

Learning fully nonlinear dynamics from partial measurements is still a cutting-edge research challenge today, with advances in deep learning making it possible to learn nonlinear transformations from delay coordinates where sparse models may be learned [19]; these transformations are closely related to Takens's delay embedding theorem [335] and Koopman operator theory [54, 55]. The loss functions used to train these autoencoder neural networks are highly customized, and are essential to the learning process.

It is also possible to learn partial differential equations

$$\mathbf{u}_t = \mathcal{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \cdots). \tag{8.24}$$

from data, although this fits more into the realm of model discovery and is a much more recent field of research [303, 307, 57]. Typically, this may be viewed as an extension of the nonlinear system identification problem, where we learn $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ from data.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

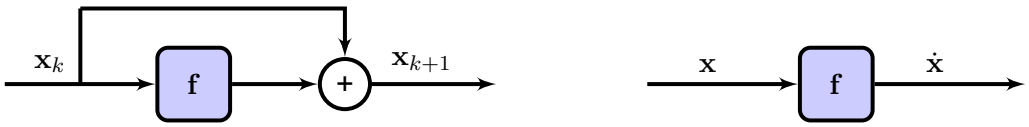


Figure 8.11: Block diagram for a residual network (left) and a neural ODE (right). The residual network mimics a forward Euler integration of the vector field $\mathbf{f}$, while a neural ODE directly learns this function, enabling more sophisticated integration schemes.

Black-Box Machine Learning Methods

More powerful, yet less interpretable methods from machine learning are increasingly being used to learn dynamical systems from data. Most of the modern techniques involve neural networks, in some form or another, although other techniques such as NARMAX [38] were developed for quite some time.

Neural ODEs are a particularly powerful class of machine learning algorithms [76] that build on the popular residual network (ResNet) architecture [165]. Both architectures are illustrated schematically in Figure 8.11.

Residual networks were introduced in 2015 [165], primarily for image classification and super-resolution tasks, and quickly became a dominant architecture. ResNets allow for much deeper neural networks, without suffering from forgetting or vanishing/exploding gradients, by introducing jump connections that directly add a copy of the input to the output. The idea is that it may be much easier to model the difference in the input and output (e.g., in super-resolution tasks, the input should be a coarse version of the output, and so the network only needs to learn the refinement operator). However, it was recognized by Chen et al. [76] that if we think of this network as modeling a differential equation $\mathbf{f}$, then the network is essentially mimicking a forward Euler integration scheme

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k) \tag{8.25}$$

with time step $\Delta t = 1$. It is well know that Euler schemes are low-order and prone to errors and instabilities. The idea behind NODE is to instead learn the differential equation $\mathbf{f}$, after which it is possible to use more sophisticated numerical integrators, such as Runge-Kutta schemes, to advance the solution.

Other prominent neural network architectures include reservoir computers [274] and long short-term memory (LSTM) networks [357]. Reservoir computers are especially simple to train, as only the last linear layer of the reservoir needs to be adjusted, which amounts to a simple least-squares regression procedure.

Open-Source Software

There are several open-source software libraries to learn dynamical systems from data:

- PyDMD [95, 177] (`https://github.com/mathLab/PyDMD`);
- PySINDy [94, 196] (`https://github.com/dynamicslab/pysindy`);
- Deeptime [173] (`https://github.com/deeptime-ml/deeptime`).

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

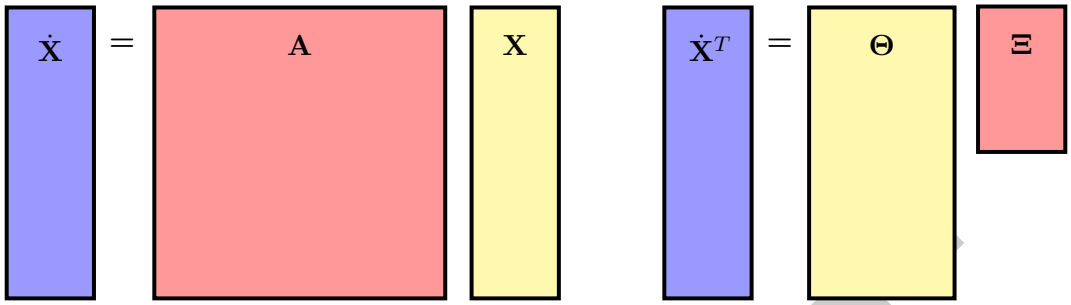


Figure 8.12: Linear system identification via DMD (left) and SINDy (right). In each case, the derivative estimates are shown in blue, the data in yellow and the unknown model parameters in red. The DMD regression is typically used for high dimensional data $\mathbf{x} \in \mathbb{R}^n$, with less snapshots $m \ll n$, so that the matrix $\mathbf{X}$ is *tall and skinny* and the matrix inversion to solve for $\mathbf{A}$ is ill-conditioned unless we add a low-rank condition. The SINDy regression is used in the opposite regime, for relatively low-dimensional $\mathbf{x} \in \mathbb{R}^n$ with more snapshots $m \gg n$. In this case, we assume a sparsity prior on Ξ. Both problems essentially amount to a linear regression $\mathbf{Ax} = \mathbf{b}$ as in Chapter 4 with regularization or constraints.

These libraries are fundamentally built on optimization. For example, in PySINDy, the optimization algorithm is explicitly defined when initializing a model:

```
import pysindy as ps
import numpy as np

# Generate synthetic data: dx/dt = -x
t = np.linspace(0, 10, 100)              # Time vector
x = np.exp(-t)                           # Data: x(t)

# Initialize and fit SINDy model using STLSQ optimizer
model = ps.SINDy(optimizer=ps.STLSQ(threshold=0.1))
model.fit(x, t=t)     # Fit the data
model.print()         # Display the discovered equation
```

It is important to note that there are several optimization techniques and regularizations that may be used for the system identification problems above, and several of them are implemented in the software packages above. While DMD uses rank to regularize the ill-conditioned inverse problem, SINDy uses sparsity. An early variant of DMD by Jovanovic et al. [182] used sparsity on the DMD mode amplitudes to downselect only the most important modes, which was useful for flows exhibiting broadband turbulence.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) Applied System Identification, by J. N. Juang, 1994 [183].

(2) System Identification: Theory for the User, by L. Ljung, 1999 [229].

Papers and reviews

(1) Can one hear the shape of a drum?, *The american mathematical monthly*, by M. Kac, 1966 [189].

(2) Estimating the dimension of a model, by G. Schwarz et al., *The Annals of Statistics*, 1978 [317].

(3) Ensemble Kalman methods for inverse problems, by M. A. Iglesias, K. J. H. Law, and A. M. Stuart, *Inverse Problems*, 2013 [178].

(4) Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation, by J. Reuther, A. Jameson, J. Farmer, L. Martinelli, and D. Saunders, *Research Institute for Advanced Computer Science, NASA Ames Research Center*, 1996 [291].

(5) Data-driven aerospace engineering: reframing the industry with machine learning, by S. L. Brunton, J. N. Kutz, K. Manohar, A. Y. Aravkin, K. Morgansen, J. Klemisch, N. Goebel, J. Buttrick, J. Poskin, A. W. Blom-Schieber, T. Hogan, and D. McDonald, *AIAA Journal*, 2021 [58].

(6) Discovering governing equations from data by sparse identification of nonlinear dynamical systems, by S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Proceedings of the National Academy of Sciences*, 2016 [59].

(7) Distilling free-form natural laws from experimental data, by M. Schmidt and H. Lipson, *Science*, 2009 [313].

(8) Compressed sensing, by D. L. Donoho, *IEEE Transactions on Information Theory*, 2006 [105].

(9) Signal recovery from random measurements via orthogonal matching pursuit, by J. A. Tropp and A. C. Gilbert, *IEEE Transactions on Information Theory*, 2007 [347].

(10) Data-driven sparse sensor placement, by K. Manohar, B. W. Brunton, J. N. Kutz, and S. L. Brunton, *IEEE Control Systems Magazine*, 2018 [238].

(11) Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information, by E. J. Candès, J. Romberg, and T. Tao, *IEEE Transactions on Automatic Control*, 2006 [68].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Homework

Exercise 8–1: Generate a noisy data signal $y = \sin(t) + \sigma w$ for $t = 0 : 0.001 : 10$, where w is normally distributed with mean 0 and standard deviation 1. The standard deviation of the added noise is σ.

Compare the finite difference derivative with the Tikhonov regularized derivative in Eq. (8.6). Compare both the forward difference and central difference schemes. Sweep through different values of σ and λ. Plot the results and compute the error between the estimated and true noise-free derivative.

Exercise 8–2: Repeat the problem above, now comparing the TV-regularized derivative, for various regularization strengths λ.

Exercise 8–3: Generate a piecewise smooth vector

$$y = \begin{cases} \text{for} \quad x \in [0,1] \\ 1 - x \\ \text{for} \quad x \in (1,2] \end{cases}$$

defined on $x \in [0, 2]$. Add noise to this vector, as above, and compare the various derivatives: finite difference, Tikhonov regularized, and TV-regularized. What do you notice?

Exercise 8–4: Bomb Inversion. Estimate the yield of the Trinity nuclear bomb, shown in Fig. 8.7 in Sec. 8.5. In particular, use the Sedov-Taylor equation, Eq. (8.13), for the radius as a function of time, given a bomb energy E. You can extract the radius at discrete instances in time using the still frames from Fig. 8.7, just as G. I. Taylor did.

How does your estimate compare to the estimate of Taylor [336, 337]?

Exercise 8–5: (1D inverse problem) Solve the simple inverse problem of identifying the parameter a in

$$y'(t) = a\, y(t), \quad y(0) = 1,$$

given noisy measurements $\{y(t_i)\}$ at times t_i.

1. Write down the least squares problem for a.
2. Solve it and compare with the true a used to generate data.
3. Discuss how noise affects your estimate.

Exercise 8–6: (Matrix inversion and SVD) Generate a random 100×100 matrix $\mathbf{A}$ (e.g., Gaussian entries). Form $\mathbf{B} = \mathbf{A}^{-1}$ (or compute the pseudoinverse if singular) and record the condition number of $\mathbf{A}$.

1. Plot the singular values of $\mathbf{A}$ and $\mathbf{B}$.
2. Discuss the effect of condition number on solving $\mathbf{Ax} = \mathbf{b}$ with noise in $\mathbf{b}$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 8–7: (Radon Transform and Tomographic Reconstruction) Create a small 64×64 phantom image (e.g., Shepp–Logan phantom or your own shape).

1. Implement a **Radon transform** for a set of projection angles θ_i. This yields a sinogram.
2. Add noise to the sinogram.
3. Try **filtered backprojection** for reconstruction. Plot the result and discuss noise artifacts.
4. Now formulate an **iterative** or **algebraic** approach:

$$\min_f \|\mathcal{R}f - (\text{sinogram})\|_2^2 + \lambda R(f).$$

   Implement a simple gradient or Kaczmarz method. Compare the reconstruction with FBP.

Exercise 8–8: (PDE-Constrained Inverse Problem in 1D) Consider the 1D Poisson equation $-u''(x) = f(x)$ on $[0, 1]$ with boundary conditions $u(0) = u(1) = 0$. Suppose you measure $u(x)$ at $m = 10$ sensor points but do not know $f(x)$.

1. Write a functional to estimate $f(x)$ from these m measurements. Discretize $f(x)$, e.g., piecewise constant or polynomials.
2. Implement an iterative approach: at each step, guess $f(x)$, solve the PDE for $u(x)$, and compare with the measured data. Update $f(x)$ to minimize the mismatch.
3. Include a regularization $\|\nabla f\|_2^2$ or $\|f\|_2^2$. Does it help reduce spurious oscillations?
4. Try with synthetic data from a known ground-truth $f(x)$. Plot your recovered $f(x)$.

Exercise 8–9: Recovering an audio signal from sparse measurements. To illustrate the use of compressed sensing to reconstruct a high-dimensional signal from a sparse set of random measurements, we consider a signal consisting of a two-tone audio signal:

$$x(t) = \cos(2\pi \times 97t) + \cos(2\pi \times 777t). \tag{8.26}$$

This signal is clearly sparse in the frequency domain, as it is defined by a sum of exactly two cosine waves. The highest frequency present is 777 Hz, so that the Nyquist sampling rate is 1554 Hz. However, leveraging the sparsity of the signal in the frequency domain, we can accurately reconstruct the signal with random samples that are spaced at an average sampling rate of 128 Hz, which is well below the Nyquist sampling rate. In this example, the full signal is generated from $t = 0$ to $t = 1$ with a resolution of $n = 4,096$ and is then randomly sampled at $p = 128$ locations in time. The sparse vector of coefficients in the discrete cosine transform (DCT) basis is solved for using matching pursuit.

Code 8.1: [Python] Compressed sensing of two-tone cosine signal.

```
from cosamp_fn import cosamp
## Generate signal, DCT of signal
```

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

```
n = 4096 # points in high resolution signal
t = np.linspace(0,1,n)
x = np.cos(2 * 97 * np.pi * t) + np.cos(2 * 777 * np.pi * t)
xt = np.fft.fft(x) # Fourier transformed signal
PSD = xt * np.conj(xt) / n # Power spectral density

## Randomly sample signal
p = 128 # num. random samples, p = n/32
perm = np.floor(np.random.rand(p) * n).astype(int)
y = x[perm]

## Solve compressed sensing problem
Psi = dct(np.identity(n)) # Build Psi
Theta = Psi[perm,:]       # Measure rows of Psi
# CS via matching pursuit
s = cosamp(Theta,y,10,epsilon=1.e-10,max_iter=10)
xrecon = idct(s) # reconstruct full signal
```

It is important to note that the $p = 128$ measurements are randomly chosen from the $4,096$ resolution signal. Thus, we know the precise timing of the sparse measurements at a much higher resolution than our sampling rate. If we chose $p = 128$ measurements uniformly in time, the compressed sensing algorithm fails. Specifically, if we compute the PSD directly from these uniform measurements, the high-frequency signal will be aliased resulting in erroneous frequency peaks.

In the compressed sensing matching pursuit (CoSaMP) code, the desired level of sparsity K must be specified, and this quantity may not be known ahead of time. The alternative ℓ_1 minimization routine does not require knowledge of the desired sparsity level *a priori*, although convergence to the sparsest solution relies on having sufficiently many measurements p, which indirectly depends on K.

Exercise 8–10: (Comparing Numerical Differentiation Techniques) Consider a noisy 1D signal $s(t) = \sin(3t) + 0.5\sin(10t)$ sampled at $t_k = k\,\Delta t$, $k = 0, \ldots, 200$, with Gaussian noise added.

1. Compute the true derivative $\dot{s}(t)$.
2. Implement three derivative approximations:
   - *Finite-differences* (central difference).
   - *Tikhonov-regularized differentiation*.
   - *Total variation (TV) regularization*, solving

     $$\min_{\dot{s}} \tfrac{1}{2}\|\mathbf{H}\dot{s} - s\|_2^2 + \lambda\|\dot{s}\|_{\mathrm{TV}},$$

     where $\mathbf{H}$ is an integration operator.
3. Plot the estimated derivatives for each method. Which is most robust to noise? How does λ affect the TV solution?

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 8–11: Generate data from a known *nonlinear* system, the Lotka–Volterra predator-prey model:

$$\dot{x}_1 = \alpha x_1 - \beta x_1 x_2, \quad \dot{x}_2 = \delta x_1 x_2 - \gamma x_2.$$

1. Collect trajectory data $\{\mathbf{x}(t_k)\}$ from an initial condition $[1, 1]$ with a $\Delta t = 0.01$. Numerically estimate $\dot{\mathbf{x}}(t_k)$, for example using central difference derivatives.
2. Build a library $\boldsymbol{\Theta}(\mathbf{x})$ with polynomials up to degree 2 (or 3).
3. Solve $\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\,\boldsymbol{\Xi}$ using **sequential thresholded least squares**.
4. Compare your discovered $\boldsymbol{\Xi}$ with the ground-truth coefficients $(\alpha, \beta, \gamma, \delta)$. Which terms are selected? How does the threshold λ affect your results?
5. Now solve for $\boldsymbol{\Xi}$ using LASSO. Compare the results with STLS. Add a small amount of noise to the trajectory $\mathbf{x}$, before computing the derivative. How does this change the results?

Exercise 8–12: This exercise will test the data requirements for identifying an accurate SINDy model for the Lorenz system, following the work of Champion et al. [72]. The equations of motion for the Lorenz system are

$$\dot{x} = \sigma(y - x) \tag{8.27}$$
$$\dot{y} = x(\rho - z) - y \tag{8.28}$$
$$\dot{z} = xy - \beta z. \tag{8.29}$$

Use parameters $(\sigma, \rho, \beta) = (10, 28, 8/3)$ and initial condition $\mathbf{x}_0 = (x(0), y(0), z(0)) = (0, 1, 20)$.

First, we will use clean data without any measurement noise.

(a) Generate a long trajectory with a fine sampling rate of $\Delta t = 0.0001$; discard the first $t = 1$ of the data. Use SINDy to generate models using an increasing length of data T. At what T is it possible to identify the correct Lorenz mode? Plot the data up until T. Do the results surprise you? (Explain why)

(b) Repeat this experiment for different sampling rates from $t = 0.0001$ to 0.1. How does the minimum data length T and the number of samples $T/\Delta t$ change?

(c) Repeat the experiment with small additive Gaussian noise on the trajectory. With noise, repeat the identification for several noise realizations to obtain an average success rate. Also try different noise magnitudes. Repeat the main figure in the Champion et al. paper.

Exercise 8–13: This example will use SINDy to learn the nonlinear Rossler system from data. The Rossler system is given by

$$\dot{x} = -y - z$$
$$\dot{y} = x + ay$$
$$\dot{z} = b + z(x - c)$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

with parameters $a = 0.2$, $b = 0.2$, and $c = 14$. Simulate this system with $\Delta t = 0.001$ from an initial condition $[1, 1, 0]$. Use this trajectory to identify a SINDy model. Explore different sparsifying thresholds and different trajectory lengths. Now add a small amount of noise to the trajectory and re-identify the model, again with different thresholds and lengths. When there is sufficiently long trajectory data, what happens when the threshold is too large? When it is too small?

Exercise 8–14: Now we will revisit the Lorenz system and explore the effect of different de-noising derivative algorithms on identification from noisy trajectory data.

(a) First, generate a trajectory with $\Delta t = 0.01$ with the standard parameters and initial condition. Discard the first 1 time units of data, and use the next 5 time units for the trajectory (i.e., from $t = 1$ to $t = 6$).

(b) Add a small amount of Gaussian noise ($\sigma = 0.001$) to the trajectory, and estimate the derivative using several techniques: forward difference, central difference, TV regularized differentiation, and polynomial smoothing.

(c) Use all of these velocity estimates to identify a SINDy model. Compare the results.

(d) Repeat for larger ans smaller noise and report on the trends.

Exercise 8–15: Now we will compare more regularization schemees for SINDy. Using the same data from the previous exercise, compare the following SINDy models using different regularization schemes:

- L_1 regularization on $\boldsymbol{\Xi}$ with LASSO;
- L_1 regularization on $\boldsymbol{\Xi}$ with STLS;
- L_2 regularization on $\boldsymbol{\Xi}$ (i.e., ridge or Tikhonov regularizatin);
- L_1 and L_2 regularization on $\boldsymbol{\Xi}$ (i.e., elastic net).

Plot the discovered coefficients vs. regularization weight. Which penalty yields more *sparse* solutions? Which is more robust to moderate noise?

Exercise 8–16: We will explore the PDE variant of SINDy, called PDE-FIND [303], to learn the viscous Burgers' equation on $x \in [0, 2\pi]$:

$$u_t + u\, u_x = \nu\, u_{xx}.$$

(a) First, simulate data $u(x, t)$ from a spectral or finite-difference code.

(b) Build a dictionary of candidate terms: $\{u, u^2, u_x, u_{xx}, u\, u_x, u^2 u_x, \dots\}$.

(c) Perform the PDE-FIND regression:

$$\min \|\mathbf{U}_t - \Theta(\mathbf{U}, \mathbf{U}_x, \dots)\, \boldsymbol{\Xi}\|_2^2 + \lambda \|\boldsymbol{\Xi}\|_1.$$

(d) Compare the discovered PDE with the ground truth. Plot discovered terms vs. ν.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(e) Discuss how noise in $u(x,t)$ affects the results and give some ideas for potentially addressing this.

Exercise 8–17: This exercise will explore identifying a PDE using PDE-FIND based on data from the Korteweg-De Vries (KdV) system

$$u_t + u_{xxx} - 6uu_x = 0.$$

This system gives rise to coherent traveling wave solutions, known as solitons, that retain their shape as they travel at a constant wave speed, despite the nonlinearity in the system. In general, a soliton solution will take the form $u(x,t) = -\frac{c}{2}\text{sech}^2\left(\frac{\sqrt{c}}{2}(x-ct)\right)$ for an arbitrary positive wave speed c. Notice that the wave speed c is linked to the amplitude and width of the wave.

(a) Generate data by initializing a single soliton solution $u(x,0) = -\frac{1}{2}\text{sech}^2\left(\frac{1}{2}(x)\right)$, with a wave speed corresponding to $c = 1$ above. Try to identify a PDE model using PDE-FIND based on this data. What is the sparsest model that supports the data?

(b) Next, use data beginning at two initial conditions. The first initial condition is the one above with $c = 1$, and the second initial condition is a soliton solution with $c = 4$. Concatenate this data and identify a PDE model using PDE-FIND. What is the sparsest model that supports the data?

(c) Discuss the two models that are identified depending on the initial data, and explain any differences.

Exercise 8–18: Now we will explore SINDy on real-world data. Collect actual measurement data from a simple mechanical system, such as a mass-spring-damper. You may either measure the relevant position or velocity variables using an accelerometer, or from video, for example recorded on a smart phone.

First, suppose that you know the equations of motion but you do not know the damping coefficient c and spring constant k.

(a) Formulate an inverse problem to estimate c and k.

(b) Fit these parameters from data.

(c) Discuss identifiability and how much data is required for a robust estimate.

Now, suppose that you don't know the equations of motion and must learn these as well.

(a) Formulate another inverse problem to estimate the model from data.

(b) Decide on an appropriate algorithm to learn the model from data.

(c) Explore the data requirements to learn a good model.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

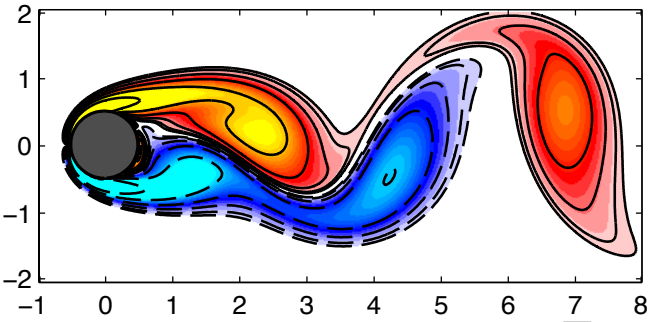


Figure 8.13: Single snapshot of the vorticity field for the fluid flow past a two-dimensional cylinder. This flow is periodic and conserves energy.

Exercise 8–19: Repeat the previous example, but for a simple pendulum, for example the arm of a grandfather clock. Again, first suppose that you know the equations of motion but don't know the damping coefficient. Then assume that you don't know the equations of motion and must learn these as well.

Exercise 8–20: In this problem, we will apply dynamic mode decomposition (DMD) to analyze a simple two-dimensional fluid flow past a circular cylinder. This flow exhibits periodic vortex shedding, and it is a canonical fluid flow that is used to understand more sophisticated flows, such as over aircraft wings.

Download the cylinder flow data from `http://dmdbook.com/DATA.zip`. There are 150 vorticity field snapshots, each of dimension 199×449. Thus, reshaping each flow field into a tall skinny vector and arranging into columns of a matrix, we obtain a data matrix with dimensions 89351×150. A single flow field snapshot is shown in Fig. 8.13.

- Compute DMD on this cylinder flow data and plot the DMD eigenvalues in the complex plane. Use $r = 20$ modes by truncating and keeping only the first 20 modes in the singular value decomposition. This means you should have 20 DMD eigenvalues. Because the flow is periodic and energy conservative, the DMD eigenvalues should come in complex conjugate pairs on the unit circle in the complex plane.

- Now add a small amount of white noise to the data, and repeat the exercise above. Try 1%, 2%, 4%, etc. and plot the new DMD eigenvalues on the same plot as the DMD eigenvalues for the clean data. What trends do you notice?

- Now use the clean data, but take a subset of the flow field snapshots so that you only sample an incomplete period of vortex shedding (i.e., only sample 75% of a vortex shedding cycle). Repeat the exercise above and plot the DMD eigenvalues.

You are welcome to use `PyDMD` for any of this assignment.

In a future homework, we will use *physics-informed DMD* (PI-DMD) to incorporate physics into the DMD regression to improve the results for noisy and partial data. What are some potential aspects of the physics that we might try to include?

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 9

Optimization in Control

Control theory is one of the major application domains of modern optimization, where we design a control law to interact with and modify the behavior of a dynamical system. For example, a cruise controller in an automobile modulates the accelerator to maintain a desired reference speed, despite unmeasured factors, such as hills, wind, and cargo. Feedback control is an increasingly essential part of the modern engineered world. The applications are nearly endless, including autopilots, the control of construction equipment, heating ventilation and cooling systems, internet traffic routing, and many more. The use of control theory in increasingly challenging settings has closely tracked progress in our ability to solve hard optimization problems, increasingly in real time.

An optimal controller balances the performance and robustness of the closed-loop system with the cost of control, as quantified by an objective function; in control, the objective function is often called the cost function. This results in a constrained optimization problem: the state of the system, which appears in the objective function, is constrained by the dynamics. Other constraints involve limitations on the available actuation power and desired limits on the system state.

Control theory is a vast topic, and a full treatment is beyond the scope of this chapter. For more details, see the *Control Bootcamp*[1]; there are also several excellent texts that provide a deeper background [116, 321, 13, 56]. This chapter presents a high-level overview of applications of optimization in control, with an emphasis on the nature of the optimization problem and solution techniques. We will almost exclusively consider closed-loop feedback control, which uses sensor measurements to inform the controller about how the system is actually responding. Closed-loop control is essential to stabilize unstable systems and to account for an imperfect model and other forms of uncertainty.

We begin with linear optimal control, as this is the most well-developed and intuitive branch of control theory, where the optimization problem often has an analytic, closed-form solution. These solutions include the classic linear quadratic regulator (LQR) and Kalman filter, which both requires solving a Riccati equation. We will derive the Riccati equation using constrained optimization and Lagrange multipliers. Linear optimal control is often also effective at regulating nonlinear systems in a neighborhood of a fixed point, where the assumption of linear dynamics is valid.

We then explore the more general nonlinear optimal control problem, which involves

[1] https://www.youtube.com/playlist?list=PLMrJAkhIeNNR20Mz-VpzgfQs5zrYi085m

DRAFT

the Hamilton-Jacobi-Bellman (HJB) equations. Reinforcement learning is introduced as a powerful approach to control complex, nonlinear systems, based on learning from experience. Reinforcement learning has been developed for a century, and it has rapidly advanced in recent years, based on modern techniques in machine learning; these will be discussed in Chapter 10. The HJB equations and reinforcement learning naturally leads to dynamic programming, which is a powerful optimization framework to solve such nonlinear optimal control problems. The topics of HJB, dynamic programming, and reinforcement learning are all closely related, and share much common development and history.

Next, we develop the model predictive control (MPC) framework, which is widely used in modern industrial control due to its flexibility, robustness, ease of implementation, and ability to respect constraints. MPC involves repeatedly solving an optimal control problem on a receding horizon, updating the state of the system and reinitializing the optimization problem as new sensor information becomes available. Thus, MPC is inherently computationally intensive. Historically, it has been used primarily for "slow" applications and for systems with simple dynamics and constraints. For example, a system with linear dynamics, quadratic objective function, and convex constraints, results in a convex optimization problem that may be solved with scaleable on-line optimization methods for real-time control. However, advances in computational hardware and optimization algorithms have made it increasingly possible to apply MPC to larger-scale systems, with faster timescales and more complex dynamics and constraints.

Finally, we investigate the framework of linear matrix inequalities (LMIs), which provides a powerful and flexible mathematical framework for formulating and solving problems in optimal control. We will show that many classical problems in optimal control, such as LQR and the Kalman filter, can be formulated as LMIs and solved efficiently with convex optimization techniques. Indeed, LMIs are a class of convex constraints, and many seemingly non-convex problems in control and optimization can be reformulated as LMIs, for which there are efficient solvers.

Mathematical Statement of the Optimal Feedback Control Problem

Figure 9.1 depicts the general feedback control framework, where sensor measurements, $\mathbf{y}$, are fed into a controller, which then computes an actuation signal, $\mathbf{u}$, to manipulate the dynamics and provide robust performance despite model uncertainty and external disturbances. The vector of external disturbances is typically decomposed as $\mathbf{w} = \begin{bmatrix}\mathbf{w}_d^T & \mathbf{w}_n^T & \mathbf{w}_r^T\end{bmatrix}^T$, where $\mathbf{w}_d$ are disturbances to the state of the system, $\mathbf{w}_n$ is measurement noise, and $\mathbf{w}_r$ is a reference trajectory that should be tracked by the closed-loop system.

The system dynamics are typically described as differential equations:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}_d) \tag{9.1a}$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{w}_n) \tag{9.1b}$$

where $\mathbf{f}$ encodes the evolution of the system state $\mathbf{x}$ under actuation $\mathbf{u}$ and disturbances $\mathbf{w}_d$, and $\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{w}_n)$ is a measurement equation.

The goal is to construct a control law

$$\mathbf{u} = \mathbf{k}(\mathbf{y}, \mathbf{w}_r) \tag{9.2}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

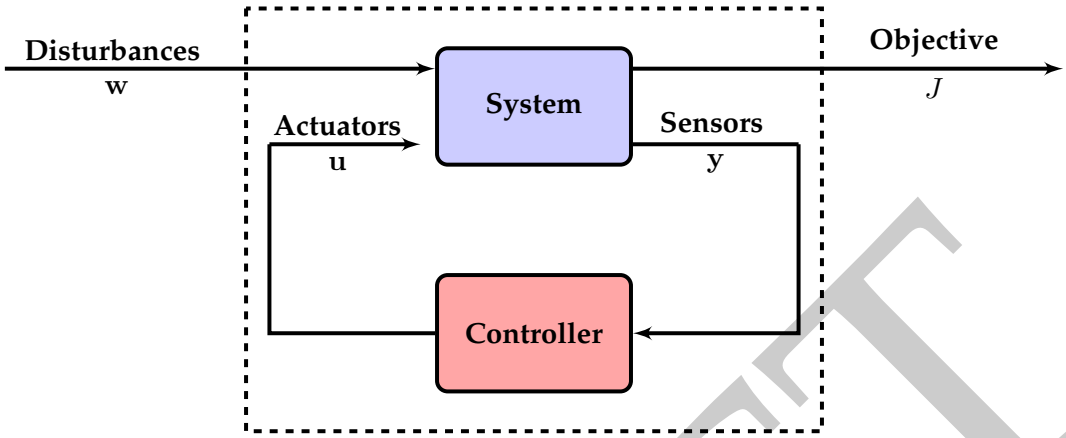


Figure 9.1: Standard framework for feedback control. Measurements, $\mathbf{y}(t)$, are fed into a controller, which decides on an actuation signal $\mathbf{u}(t)$ to control the system. The control law is designed to modify the system dynamics and provide good performance, quantified by the objective J, despite external disturbances and noise in $\mathbf{w}$. The external input $\mathbf{w}$ may also include a reference trajectory $\mathbf{w}_r$ that should be tracked.

that minimizes an objective function

$$J(\mathbf{x}, \mathbf{u}, \mathbf{w}_r). \tag{9.3}$$

The sensor measurements, $\mathbf{y}$, provide information about unmodeled dynamics and disturbances that would degrade the performance in open-loop control. Further, with feedback it is often possible to modify and stabilize the dynamics of the closed-loop system, something which is not possible with open-loop control. Thus, closed-loop feedback control is often able to maintain high-performance operation for systems with unstable dynamics, model uncertainty, and external disturbances. As the system state $\mathbf{x}(t)$ changes in time, the control signal $\mathbf{u}(t)$ changes correspondingly to stabilize the system, compensate for disturbances, and maintain robust performance.

In general, the controller in Eq. (9.2) will be a dynamical system, rather than a static function of the inputs. For example, the Kalman filter dynamically estimates the full state $\mathbf{x}$ from measurements of $\mathbf{u}$ and $\mathbf{y}$. In this case, the control law will become $\mathbf{u} = \mathbf{k}(\mathbf{y}, \hat{\mathbf{x}}, \mathbf{w}_r)$, where $\hat{\mathbf{x}}$ is an estimate of the state. Thus, optimal control typically involves optimization of a continuous function in a well-defined function space. For example, the linear quadratic regulator will involve finding the optimal control function in the space of linear functions $\mathbf{u} = \mathbf{K}\mathbf{x}$. Sometimes, the control action will be discretized in time and optimized over a finite horizon, as in MPC, resulting in a finite-dimensional optimization.

The Python Control Systems Library (`python-control`) [2] toolbox provides extensive functionality, including Python wrappers for the SLICOT optimization libraries [31] used in Matlab's control toolboxes. It is assumed that the following is added to the header:

```
from control.matlab import *
import slycot
```

[2] Available at `https://python-control.readthedocs.io/`.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

9.1 Linear Optimal Control

The most complete theory of control has been developed for linear systems [321, 116, 13]. Linear systems are generally obtained by linearizing a nonlinear system about a fixed point or a periodic orbit, resulting in the following system of equations:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{9.4a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}. \tag{9.4b}$$

Instability may quickly take a trajectory far away from the fixed point. However, an effective stabilizing controller will keep the state of the system in a small neighborhood of the fixed point where the linear approximation is valid. For example, in the case of the inverted pendulum, feedback control may stabilize the pendulum in the vertical position where the dynamics behave linearly. Note that we have neglected the disturbance and noise inputs, $\mathbf{w}_d$ and $\mathbf{w}_n$, respectively; these will be added back in the discussion on Kalman filtering.

In this section, we will formulate and solve several optimal control problems. First, we assume we have access to the full state of the system, $\mathbf{y} = \mathbf{x}$, which allows us to design the optimal full-state feedback controller, $\mathbf{u} = -\mathbf{K}\mathbf{x}$, that minimizes a quadratic objective function in terms of the state, $\mathbf{x}$, and actuation input, $\mathbf{u}$. This is known as the linear quadratic regulator (LQR), because the dynamics constraints are linear and the objective is quadratic. This problem is convex, with a single globally optimal solution. Next, we will relax the assumption of full-state feedback, and instead assume that we only have access to partial measurements, $\mathbf{y}$, from which we must estimate the full state. This will result in the Kalman filter, which is an optimal estimator, assuming Gaussian noise and disturbances; the Kalman filter may also be formulated from a Bayesian perspective, related to topics in Chapter 7. Finally, we combine these two approaches in what is known as the linear quadratic Gaussian (LQG) optimal feedback controller, given partial measurements.

Linear Quadratic Regulator and the Riccati Equation

The goal of linear, full-state feedback control is to design a control law

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \tag{9.5}$$

so that the closed-loop system

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} \tag{9.6}$$

is stable and has good performance, quantified by a specific objective function. Not all systems are controllable, meaning that some states may not be influenced by the actuation. A system is controllable if the controllability matrix

$$\mathcal{C} = \begin{bmatrix}\mathbf{B} & \mathbf{A}\mathbf{B} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix} \tag{9.7}$$

has n linearly independent columns. Then it is possible to arbitrarily manipulate the eigenvalues of the closed-loop system $(\mathbf{A} - \mathbf{B}\mathbf{K})$ through choice of $\mathbf{K}$, stabilizing the system.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

This assumes that full-state measurements are available (i.e., $\mathbf{C} = \mathbf{I}$ and $\mathbf{D} = \mathbf{0}$, so that $\mathbf{y} = \mathbf{x}$). Full-state measurements are not always available, especially for high-dimensional systems, but we will show in the next section that if the system is observable, it is possible to build a full-state estimate from the sensor measurements via a Kalman filter.

Given a controllable system, there are many stabilizing control laws $\mathbf{u} = -\mathbf{K}\mathbf{x}$. It is important to stabilize the closed-loop system, but without expending too much control effort and without making the system overly sensitive to noise and disturbances.

Choosing the best controller gain matrix $\mathbf{K}$ to stabilize the system without expending too much control effort is a classic goal of optimal control. The objective function

$$J = \int_0^\infty \mathbf{x}(\tau)^T \mathbf{Q}\mathbf{x}(\tau) + \mathbf{u}(\tau)^T \mathbf{R}\mathbf{u}(\tau)\, \mathrm{d}\tau \tag{9.8}$$

balances the cost of regulating the state with the cost of control. The matrices $\mathbf{Q}$ and $\mathbf{R}$ weight the cost of deviations of the state from zero and the cost of actuation, respectively. The matrix $\mathbf{Q} \geq \mathbf{0}$ is positive semi-definite, and $\mathbf{R} > \mathbf{0}$ is positive definite; they are often diagonal, and the diagonal elements quantify the relative importance of the control objectives.

Adding such an objective function is essential in formulating a well-posed optimization problem. The LQR control law $\mathbf{u} = -\mathbf{K}_r\mathbf{x}$ is designed to minimize the objective function subject to the dynamics constraints:

$$\min_{\mathbf{u}=-\mathbf{K}_r\mathbf{x}} \int_0^\infty \mathbf{x}(\tau)^T \mathbf{Q}\mathbf{x}(\tau) + \mathbf{u}(\tau)^T \mathbf{R}\mathbf{u}(\tau)\, \mathrm{d}\tau \tag{9.9a}$$

$$\text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}. \tag{9.9b}$$

The LQR controller is shown schematically in Fig. 9.2. Out of all possible control laws $\mathbf{u} = \mathbf{K}(\mathbf{x})$, including nonlinear controllers, the LQR controller $\mathbf{u} = -\mathbf{K}_r\mathbf{x}$ is optimal, as we will show in the next section. The name LQR refers to the fact that we are designing a linear control law to minimize a quadratic objective function, subject to linear dynamics, that regulates the state of the system to $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{0}$.

Because the objective function in Eq. (9.8) is quadratic, there is an analytical solution for the optimal controller gain $\mathbf{K}_r$, given by

$$\mathbf{K}_r = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{X}, \tag{9.10}$$

where $\mathbf{X}$ is the solution to an algebraic Riccati equation, which we will derive next:

$$\mathbf{A}^T\mathbf{X} + \mathbf{X}\mathbf{A} - \mathbf{X}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{X} + \mathbf{Q} = \mathbf{0}. \tag{9.11}$$

Solving the Riccati equation for $\mathbf{X}$, and hence for $\mathbf{K}_r$, is numerically robust and is implemented in many programming languages [216, 30]. The gain matrix $\mathbf{K}_r$ is obtained via

```
>>> Kr = lqr(A,B,Q,R)[0]
```

in python-control. However, solving the Riccati equation scales as $\mathcal{O}(n^3)$ in the state-dimension n, making it prohibitively expensive for large systems or for online computations, for example in MPC (see Sec. 9.5). This motivates LMI formulations of the LQR problem (see Sec. 9.6), which may be solved using efficient interior point methods (see Sec. 5.4). Similarly, we may develop reduced-order models that capture the dominant behavior of the system with many fewer states.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

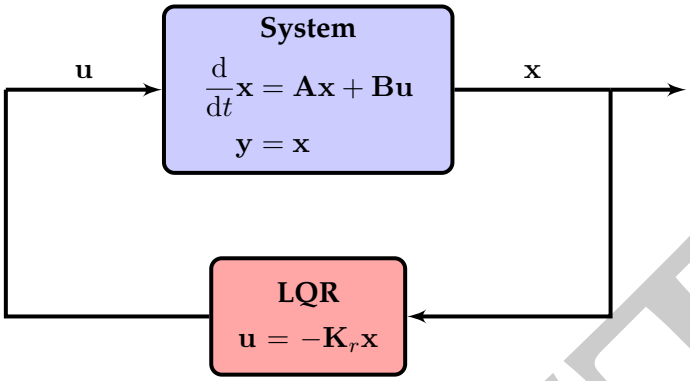


Figure 9.2: Schematic of the linear quadratic regulator (LQR) for optimal full-state feedback. The optimal controller for a linear system given measurements of the full state, $\mathbf{y} = \mathbf{x}$, is given by proportional control $\mathbf{u} = -\mathbf{K}_r\mathbf{x}$ where $\mathbf{K}_r$ is a constant gain matrix obtained by solving an algebraic Riccati equation.

Derivation of the Riccati equation for optimal control

It is worth taking a theoretical detour to derive the Riccati equation in Eq. (9.11) for the LQR optimization problem. This derivation will provide an example of how to solve convex optimization problems using the calculus of variations, and it will also provide a template for computing the optimal control solution for nonlinear systems. Because of the similarity of optimal control to the formulation of Lagrangian and Hamiltonian classical mechanics in terms of the variational principal, we adopt similar language and notation.

First, we define the LQR objective function up until a finite time t_f and add a terminal cost, introducing a factor of $1/2$ to simplify computations:

$$J = \int_0^{t_f} \underbrace{\frac{1}{2}\left(\mathbf{x}(\tau)^T\mathbf{Q}\mathbf{x}(\tau) + \mathbf{u}(\tau)^T\mathbf{R}\mathbf{u}(\tau)\right)}_{\text{Lagrangian, } \mathcal{L}} \mathrm{d}\tau + \underbrace{\frac{1}{2}\mathbf{x}(t_f)^T\mathbf{Q}_f\mathbf{x}(t_f)}_{\text{Terminal cost}}. \tag{9.12}$$

The goal is to minimize the objective J subject to the dynamics $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$. We solve this by introducing an augmented objective function

$$J_{\text{aug}} = \int_0^{t_f} \left[\frac{1}{2}\left(\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right) + \boldsymbol{\lambda}^T\left(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} - \dot{\mathbf{x}}\right)\right] \mathrm{d}\tau + \frac{1}{2}\mathbf{x}(t_f)^T\mathbf{Q}_f\mathbf{x}(t_f). \tag{9.13}$$

The Lagrange multiplier $\boldsymbol{\lambda}$ is called the *co-state* and enforces the dynamic constraints.

Taking the total variation of J_{aug} in Eq. (9.13) yields:

$$\delta J_{\text{aug}} = \int_0^{t_f} \left[\frac{\partial \mathcal{L}}{\partial \mathbf{x}}\delta\mathbf{x} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}}\delta\mathbf{u} + \boldsymbol{\lambda}^T\mathbf{A}\delta\mathbf{x} + \boldsymbol{\lambda}^T\mathbf{B}\delta\mathbf{u} - \boldsymbol{\lambda}^T\delta\dot{\mathbf{x}}\right] \mathrm{d}\tau + \mathbf{Q}_f\mathbf{x}(t_f)\delta\mathbf{x}(t_f). \tag{9.14}$$

The partial derivatives[3] of the Lagrangian are $\partial\mathcal{L}/\partial\mathbf{x} = \mathbf{x}^T\mathbf{Q}$ and $\partial\mathcal{L}/\partial\mathbf{u} = \mathbf{u}^T\mathbf{R}$. The last

[3]The derivative of $\mathbf{A}\mathbf{x}$ with respect to $\mathbf{x}$ is $\mathbf{A}$, and the derivative of $\mathbf{x}^T\mathbf{A}$ with respect to $\mathbf{x}$ is $\mathbf{A}^T$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

term in the integral may be modified using integration by parts:

$$-\int_0^{t_f} \boldsymbol{\lambda}^T \delta\dot{\mathbf{x}} \,\mathrm{d}\tau = -\boldsymbol{\lambda}^T(t_f)\delta\mathbf{x}(t_f) + \boldsymbol{\lambda}^T(0)\delta\mathbf{x}(0) + \int_0^{t_f} \dot{\boldsymbol{\lambda}}^T \delta\mathbf{x} \,\mathrm{d}\tau.$$

The term $\boldsymbol{\lambda}^T(0)\delta\mathbf{x}(0)$ is equal to zero, or else the control system would be non-causal (i.e., then future control could change the initial condition of the system).

Finally, the total variation of the augmented objective function simplifies to

$$\begin{aligned}\delta J_{\text{aug}} = &\int_0^{t_f} \left(\mathbf{x}^T\mathbf{Q} + \boldsymbol{\lambda}^T\mathbf{A} + \dot{\boldsymbol{\lambda}}^T\right)\delta\mathbf{x}\,\mathrm{d}\tau + \int_0^{t_f}\left(\mathbf{u}^T\mathbf{R} + \boldsymbol{\lambda}^T\mathbf{B}\right)\delta\mathbf{u}\,\mathrm{d}\tau \\ &+ \left(\mathbf{x}(t_f)^T\mathbf{Q}_f - \boldsymbol{\lambda}^T(t_f)\right)\delta\mathbf{x}(t_f).\end{aligned}$$

Each variation term must equal zero for an optimal control solution that minimizes J. Thus, we may break this up into three equations:

$$\mathbf{x}^T\mathbf{Q} + \boldsymbol{\lambda}^T\mathbf{A} + \dot{\boldsymbol{\lambda}}^T = \mathbf{0} \tag{9.15a}$$

$$\mathbf{u}^T\mathbf{R} + \boldsymbol{\lambda}^T\mathbf{B} = \mathbf{0} \tag{9.15b}$$

$$\mathbf{x}(t_f)^T\mathbf{Q}_f - \boldsymbol{\lambda}^T(t_f) = \mathbf{0}. \tag{9.15c}$$

The constraint in Eq. (9.15c) represents an initial condition for the reverse-time equation for $\boldsymbol{\lambda}$ starting at t_f. Thus, Eqs. (9.4a) and (9.15) with initial condition $\mathbf{x}(0) = \mathbf{x}_0$ and final condition $\boldsymbol{\lambda}(t_f) = \mathbf{Q}_f\mathbf{x}(t_f)$ form a two-point boundary value problem. This may be integrated numerically to find the optimal control solution, even for nonlinear systems.

Because the dynamics are linear, it is possible to *posit* the form $\boldsymbol{\lambda} = \mathbf{P}\mathbf{x}$, and substitute into Eq. (9.15) above. The first equation becomes:

$$\left(\dot{\mathbf{P}}\mathbf{x} + \mathbf{P}\dot{\mathbf{x}}\right)^T + \mathbf{x}^T\mathbf{Q} + \boldsymbol{\lambda}^T\mathbf{A} = \mathbf{0}.$$

Taking the transpose, and substituting Eq. (9.4a) in for $\dot{\mathbf{x}}$, yields:

$$\dot{\mathbf{P}}\mathbf{x} + \mathbf{P}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + \mathbf{Q}\mathbf{x} + \mathbf{A}^T\mathbf{P}\mathbf{x} = \mathbf{0}.$$

From Eq. (9.15b), we have

$$\mathbf{u} = -\mathbf{R}^{-1}\mathbf{B}^T\boldsymbol{\lambda} = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}\mathbf{x}.$$

Finally, combining yields:

$$\dot{\mathbf{P}}\mathbf{x} + \mathbf{P}\mathbf{A}\mathbf{x} + \mathbf{A}^T\mathbf{P}\mathbf{x} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}\mathbf{x} + \mathbf{Q}\mathbf{x} = \mathbf{0}. \tag{9.16}$$

This equation must be true for all $\mathbf{x}$, and so it may also be written as a matrix equation. Dropping the terminal cost and letting time go to infinity, the $\dot{\mathbf{P}}$ term disappears, and we recover the algebraic Riccati equation:

$$\mathbf{P}\mathbf{A} + \mathbf{A}\mathbf{P}^T - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{Q} = \mathbf{0}.$$

Although this procedure is somewhat involved, each step is relatively straightforward. In addition, the dynamics in Eq (9.4a) may be replaced with nonlinear dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, and a similar nonlinear two-point boundary value problem may be formulated with $\partial\mathbf{f}/\partial\mathbf{x}$ replacing $\mathbf{A}$ and $\partial\mathbf{f}/\partial\mathbf{u}$ replacing $\mathbf{B}$. This procedure is extremely general, and may be used to numerically obtain nonlinear optimal control trajectories.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Hamiltonian formulation. It is also possible to solve the optimization problem by introducing the following dual Hamiltonian formulation:

$$\mathcal{H} = \underbrace{\frac{1}{2}\left(\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right)}_{\mathcal{L}} + \boldsymbol{\lambda}^T\left(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}\right). \tag{9.17}$$

Then Hamilton's equations become:

$$\begin{aligned} \dot{\mathbf{x}} &= \left(\frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}\right)^T = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} & \mathbf{x}(0) &= \mathbf{x}_0 \\ -\dot{\boldsymbol{\lambda}} &= \left(\frac{\partial \mathcal{H}}{\partial \mathbf{x}}\right)^T = \mathbf{Q}\mathbf{x} + \mathbf{A}^T\boldsymbol{\lambda} & \boldsymbol{\lambda}(t_f) &= \mathbf{Q}_f\mathbf{x}(t_f). \end{aligned}$$

Again, this is a two-point boundary value problem in $\mathbf{x}$ and $\boldsymbol{\lambda}$. Plugging in the same expression $\boldsymbol{\lambda} = \mathbf{P}\mathbf{x}$ will result in the same Riccati equation as above.

Kalman Filtering and Optimal Estimation

The optimal LQR controller relies on full-state measurements of the system, which are often unavailable. Instead of measuring the full state $\mathbf{x}$, it may be possible to estimate it from limited and noisy measurements $\mathbf{y}$. Full-state estimation is mathematically possible as long as the pair $(\mathbf{A}, \mathbf{C})$ is observable, which is analogous to controllability of $(\mathbf{A}^T, \mathbf{C}^T)$. However, even if technically possible, this may be a nearly ill-posed inverse problem depending on the degree of observability, as quantified by the observability Gramian [56].

The Kalman filter [194, 368, 157] is the most common full-state estimator, as it optimally balances the competing effects of measurement noise, disturbances, and model uncertainty. It is also possible to combine a Kalman filter with the LQR full-state feedback law, as explored in the next section. To derive the Kalman filter, we re-introduce disturbances $\mathbf{w}_d$ and sensor noise $\mathbf{w}_n$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w}_d \tag{9.19a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{w}_n. \tag{9.19b}$$

The Kalman filter assumes that both the disturbance and noise are zero-mean Gaussian processes with known covariances:

$$\mathbb{E}\left(\mathbf{w}_d(t)\mathbf{w}_d(\tau)^T\right) = \mathbf{V}_d\delta(t-\tau), \tag{9.20a}$$

$$\mathbb{E}\left(\mathbf{w}_n(t)\mathbf{w}_n(\tau)^T\right) = \mathbf{V}_n\delta(t-\tau). \tag{9.20b}$$

Here $\mathbb{E}$ is the expected value and $\delta(\cdot)$ is the Dirac delta function. The matrices $\mathbf{V}_d$ and $\mathbf{V}_n$ are positive semi-definite with entries containing the covariances of the disturbance and noise terms. Extensions to the Kalman filter exist for correlated, biased, and unknown noise and disturbance terms [323, 244].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

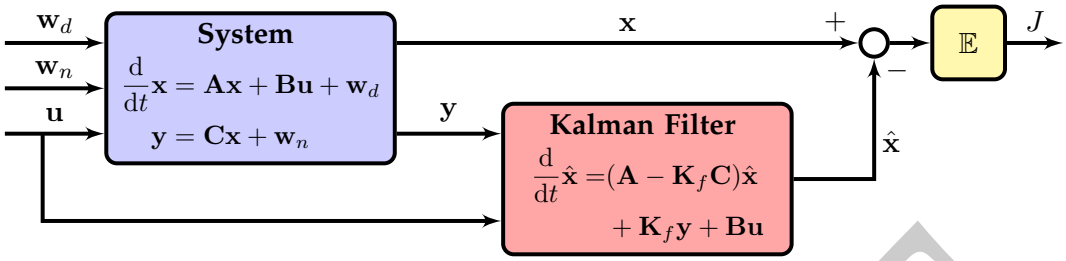


Figure 9.3: Schematic of the Kalman filter for full-state estimation from noisy measurements $\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{w}_n$ with process noise (disturbances) $\mathbf{w}_d$. This diagram does not have a feedthrough term $\mathbf{D}$, although it may be included.

It is possible to obtain an estimate $\hat{\mathbf{x}}$ of the full-state $\mathbf{x}$ from measurements of the input $\mathbf{u}$ and output $\mathbf{y}$, via the following estimator dynamical system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{K}_f\left(\mathbf{y} - \hat{\mathbf{y}}\right) \tag{9.21a}$$

$$\hat{\mathbf{y}} = \mathbf{C}\hat{\mathbf{x}} + \mathbf{D}\mathbf{u}. \tag{9.21b}$$

The matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$ are obtained from the system model, and the filter gain $\mathbf{K}_f$ is determined via a similar procedure as in LQR. $\mathbf{K}_f$ is given by

$$\mathbf{K}_f = \mathbf{Y}\mathbf{C}^T\mathbf{V}_n^{-1}, \tag{9.22}$$

where $\mathbf{Y}$ is the solution to another algebraic Riccati equation:

$$\mathbf{Y}\mathbf{A}^T + \mathbf{A}\mathbf{Y} - \mathbf{Y}\mathbf{C}^T\mathbf{V}_n^{-1}\mathbf{C}\mathbf{Y} + \mathbf{V}_d = \mathbf{0}. \tag{9.23}$$

This solution is commonly referred to as the Kalman filter, and it is the optimal full-state estimator with respect to the following objective function:

$$J = \lim_{t\to\infty} \frac{1}{t}\int_0^t \mathbb{E}\left[\left(\mathbf{x}(\tau) - \hat{\mathbf{x}}(\tau)\right)^T\left(\mathbf{x}(\tau) - \hat{\mathbf{x}}(\tau)\right)\right] \mathrm{d}\tau. \tag{9.24}$$

This objective function implicitly includes the effects of disturbance and noise, which are required to determine the optimal balance between aggressive estimation and noise attenuation. The Kalman filter is referred to as linear quadratic estimation (LQE), and has a dual formulation to the LQR optimization.

The Kalman filter gain $\mathbf{K}_f$ may be determined via

```
>> Kf = lqe(A,I,C,Vd,Vn); % design Kalman filter gain
```

where $\mathbf{I}$ is the $n \times n$ identity matrix. Optimal control and estimation are mathematical dual problems, so the Kalman filter may also be found using LQR:

```
>> Kf = (lqr(A',C',Vd,Vn))'; % LQR and LQE are dual problems
```

The Kalman filter is shown schematically in Fig. 9.3.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Substituting the output estimate $\hat{\mathbf{y}}$ from Eq. (9.21b) into Eq. (9.21a) yields:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} = (\mathbf{A} - \mathbf{K}_f\mathbf{C})\,\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y} + (\mathbf{B} - \mathbf{K}_f\mathbf{D})\,\mathbf{u}. \tag{9.25}$$

The estimator dynamical system is expressed in terms of the estimate $\hat{\mathbf{x}}$ with inputs $\mathbf{y}$ and $\mathbf{u}$. If the system is observable it is possible to place the eigenvalues of $\mathbf{A} - \mathbf{K}_f\mathbf{C}$ arbitrarily with choice of $\mathbf{K}_f$. When the eigenvalues of the estimator are stable, then the state estimate $\hat{\mathbf{x}}$ converges to the full-state $\mathbf{x}$ asymptotically, as long as the model faithfully captures the true system dynamics. To see this convergence, consider the dynamics of the estimation error $\boldsymbol{\epsilon} = \mathbf{x} - \hat{\mathbf{x}}$:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\epsilon} &= \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} - \frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} \\
&= [\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w}_d] - [(\mathbf{A} - \mathbf{K}_f\mathbf{C})\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y} + (\mathbf{B} - \mathbf{K}_f\mathbf{D})\mathbf{u}] \\
&= \mathbf{A}\boldsymbol{\epsilon} + \mathbf{w}_d + \mathbf{K}_f\mathbf{C}\hat{\mathbf{x}} - \mathbf{K}_f\underbrace{[\mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{w}_n]}_{\mathbf{y}} + \mathbf{K}_f\mathbf{D}\mathbf{u} \\
&= (\mathbf{A} - \mathbf{K}_f\mathbf{C})\boldsymbol{\epsilon} + \mathbf{w}_d - \mathbf{K}_f\mathbf{w}_n.
\end{aligned}$$

Therefore, the estimate $\hat{\mathbf{x}}$ will converge to the true full state when $\mathbf{A} - \mathbf{K}_f\mathbf{C}$ has stable eigenvalues. As with LQR, there is a tradeoff between over-stabilization of these eigenvalues and the amplification of sensor noise.

There are many variants of the Kalman filter for nonlinear systems [186, 187, 354], including the extended and unscented Kalman filters. The ensemble Kalman filter [7] works well for high-dimensional systems, as in geophysical data assimilation [290]. These methods still assume Gaussian noise. A particle filter is a more general, but more computationally intensive, alternative for arbitrary noise distributions [158, 296]. The unscented Kalman filter balances the efficiency of a Kalman filter and accuracy of a particle filter.

The Kalman Filter as a Bayesian Estimator. The Kalman filter provides a recursive algorithm for estimating the state of a linear dynamical system subject to Gaussian noise. Although often derived from control-theoretic or algebraic perspectives, the Kalman filter is fundamentally a special case of Bayesian estimation, which we discussed earlier in Chapter 7. At each time step, it performs an exact Bayesian update: combining a prior belief about the system state, based on a model, with a likelihood function derived from a new noisy measurement. This connection will be established rigorously in the homework.

Linear Quadratic Gaussian

Combining LQR with the Kalman filter yields in the linear-quadratic Gaussian (LQG) controller, resulting in optimal sensor-based feedback. Remarkably, the LQR gain $\mathbf{K}_r$ and the Kalman filter gain $\mathbf{K}_f$ may be designed separately, and the resulting sensor-based feedback will remain optimal and retain the closed-loop eigenvalues when combined.

The LQG controller consists of the LQR feedback $\mathbf{u} = -\mathbf{K}_r\hat{\mathbf{x}}$ where the state $\hat{\mathbf{x}}$ is found via the estimator dynamical system

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} = (\mathbf{A} - \mathbf{K}_f\mathbf{C} - \mathbf{B}\mathbf{K}_r)\,\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y}. \tag{9.26}$$

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

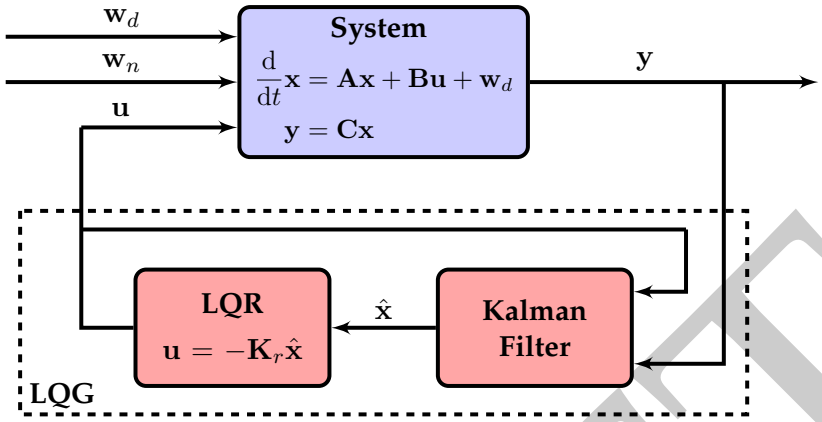


Figure 9.4: Schematic illustrating the linear quadratic Gaussian (LQG) controller for optimal closed-loop feedback based on noisy measurements $\mathbf{y}$. The LQR and Kalman filter gain matrices $\mathbf{K}_r$ and $\mathbf{K}_f$ may be designed independently. When combined, the resulting sensor-based feedback remains optimal.

The LQG controller is optimal with respect to the following ensemble-averaged version of the objective function from Eq. (9.8):

$$J(t) = \lim_{t\to\infty} \int_0^t \mathbb{E}\left[\mathbf{x}(\tau)^T\mathbf{Q}\mathbf{x}(\tau) + \mathbf{u}(\tau)^T\mathbf{R}\mathbf{u}(\tau)\right]\,\mathrm{d}\tau. \tag{9.27}$$

Applying LQR to $\hat{\mathbf{x}}$ results in the following state dynamics:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}_r\hat{\mathbf{x}} + \mathbf{w}_d \tag{9.28a}$$

$$= \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}_r\mathbf{x} + \mathbf{B}\mathbf{K}_r\left(\mathbf{x} - \hat{\mathbf{x}}\right) + \mathbf{w}_d. \tag{9.28b}$$

Again, if $\boldsymbol{\epsilon} = \mathbf{x} - \hat{\mathbf{x}}$, the closed-loop system may be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\epsilon}\end{bmatrix} = \begin{bmatrix}\mathbf{A}-\mathbf{B}\mathbf{K}_r & \mathbf{B}\mathbf{K}_r\\ \mathbf{0} & \mathbf{A}-\mathbf{K}_f\mathbf{C}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\epsilon}\end{bmatrix} + \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{I} & -\mathbf{K}_f\end{bmatrix}\begin{bmatrix}\mathbf{w}_d\\ \mathbf{w}_n\end{bmatrix}. \tag{9.29}$$

Thus, the closed-loop eigenvalues of the LQG regulated system are given by the eigenvalues of $\mathbf{A}-\mathbf{B}\mathbf{K}_r$ and $\mathbf{A}-\mathbf{K}_f\mathbf{C}$, which were optimally chosen by the LQR and Kalman filter gain matrices, respectively.

The LQG framework, shown in Fig. 9.4, requires an accurate model of the system and the magnitudes of the disturbances and measurement noise. In real-world systems, even small time delays and model uncertainty may destroy the robustness of LQG and result in instability [110]. The lack of robustness of LQG regulators to model uncertainty motivates the introduction of robust control [110]. For example, it is possible to *robustify* LQG regulators through a process known as loop-transfer recovery. However, despite robustness issues, LQG control is extremely effective for many systems, and is among the most common control paradigms.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

9.2 Nonlinear Optimal Control

Now that we have explored optimal control in the context of linear dynamics and a quadratic objective function, we are ready for the more general theory of nonlinear optimal control. The goal of this section is to provide a mathematical foundation for nonlinear optimal control, which will be useful when we discuss reinforcement learning, dynamic programming, and model predictive control. Specifically, we will derive the Hamilton-Jacobi-Bellman (HJB) equations for optimal nonlinear control, which is one of the theoretical triumphs in control theory. It relies on Bellman's principle of optimality, which states that an optimal mulit-step control policy must be locally optimal in every sub-sequence of steps.

Nonlinear optimal control problems do not generally have simple closed-form solutions, in contrast to linear optimal control. Instead, we typically leverage powerful optimization frameworks, such as reinforcement learning (Sec. 9.3) and dynamic programming (Sec. 9.4), to approximately solve the nonlinear, non-convex optimization problem. This is an enormous field, and this chapter only provides a high-level overview. For more details, see the excellent text by Stengel [327].

Mathematical Statement of Nonlinear Optimal Control Problem

The goal is to optimize the control input $\mathbf{u}(t)$ to drive a nonlinear system

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{9.30}$$

to follow a trajectory $\mathbf{x}(t)$ that minimizes an objective function

$$J(\mathbf{x}(t), \mathbf{u}(t), t_0, t_f) = \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau))\,\mathrm{d}\tau + Q(\mathbf{x}(t_f), t_f). \tag{9.31}$$

We have written $\mathbf{u}(t)$ as a function of time, rather than of the state $\mathbf{x}$, allowing us to consider the case when we optimize an open-loop control signal $\mathbf{u}(t)$ or a closed-loop controller $\mathbf{u}(t) = \mathbf{u}(\mathbf{x}(t))$. We have also dropped the output equation $\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{w}_n)$ for simplicity, although the theory below readily generalizes to handle partial measurements.

The formulation in Eq. (9.31) generalizes the LQR objective function in Eq. (9.12), so that $\mathcal{L}(\mathbf{x}, \mathbf{u})$ and $Q(\mathbf{x}(t_f), t_f)$ may be non-quadratic. There may also be constraints on the state $\mathbf{x}$ and control $\mathbf{u}$, which determine what solutions are admissible; this is especially common in model predictive control, which we will explore in Sec. 9.5.

The Value Function and the Hamilton-Jacobi-Bellman Equations

An optimal control $\mathbf{u}(t)$ will minimize the objective function J, starting at an initial state $\mathbf{x}_0 = \mathbf{x}(t_0)$. We reframe this optimization problem in terms of the *value* function $V(\mathbf{x}_0, t_0, t_f)$, which is the total integrated cost starting at $\mathbf{x}_0$ assuming the control law is optimal:

$$V(\mathbf{x}_0, t_0, t_f) = \min_{\mathbf{u}(t)} J(\mathbf{x}(t), \mathbf{u}(t), t_0, t_f). \tag{9.32}$$

Here $\mathbf{x}(t)$ is the solution to Eq. (9.30) for the optimal $\mathbf{u}(t)$. The value function is no longer a function of the control $\mathbf{u}(t)$, as this has been optimized over, and it is also not a function of

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

$\mathbf{x}(t)$, but rather of an initial state $\mathbf{x}_0$, as the remainder of the trajectory is entirely specified by the dynamics and the optimal control law. The value function is often called the *cost-to-go* in control theory, as the value function evaluated at any point $\mathbf{x}(t)$ on an optimal trajectory will give the remaining cost associated with continuing to enact the optimal policy until the final time t_f. This is a statement of *Bellman's optimality principle*, that the value function V remains optimal starting with any point on an optimal trajectory.

The Hamilton-Jacobi-Bellman[4] (HJB) equation is a partial differential equation that must be satisfied by the value function V at every intermediate time $t \in [t_0, t_f]$:

$$-\frac{\partial V}{\partial t} = \min_{\mathbf{u}(t)} \left[\left(\frac{\partial V}{\partial \mathbf{x}} \right)^T \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) + \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t)) \right]. \tag{9.33}$$

The term $\frac{\partial V}{\partial \mathbf{x}}$ is a Lagrange multiplier $\boldsymbol{\lambda}$ to enforce the dynamics constraint given by $\mathbf{f}$.

To derive the HJB equation, we take the total time derivative of the value function $V(\mathbf{x}(t), t, t_f)$ at some intermediate time t:

$$\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{x}(t), t, t_f) = \frac{\partial V}{\partial t} + \left(\frac{\partial V}{\partial \mathbf{x}} \right)^T \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \tag{9.34a}$$

$$= \min_{\mathbf{u}(t)} \frac{\mathrm{d}}{\mathrm{d}t} \left[\int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau)) \, \mathrm{d}\tau + Q(\mathbf{x}(t_f), t_f) \right] \tag{9.34b}$$

$$= \min_{\mathbf{u}(t)} \Bigg[\underbrace{\frac{\mathrm{d}}{\mathrm{d}t} \int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau) \, \mathrm{d}\tau}_{-\mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))} \Bigg] \tag{9.34c}$$

$$\implies \quad -\frac{\partial V}{\partial t} = \min_{\mathbf{u}(t)} \left[\left(\frac{\partial V}{\partial \mathbf{x}} \right)^T \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) + \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t)) \right]. \tag{9.34d}$$

The terminal cost does not vary with t, so it has zero time derivative. The derivative of the integral $\int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau)) \, \mathrm{d}\tau$ is equal to $-\mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))$ by the first fundamental theorem of calculus. Finally, the term $(\partial V / \partial \mathbf{x})^T \mathbf{f}(\mathbf{x}, \mathbf{u})$ may be brought into the minimization with respect to $\mathbf{u}(t)$, since V is already defined as the optimal cost over $\mathbf{u}$.

The derivation above relies on Bellman's principle of optimality: at any point on an optimal trajectory $\mathbf{x}(t)$, the value function V is still optimal for the rest of the trajectory:

$$V(\mathbf{x}(t), t, t_f) = \min_{\mathbf{u}(t)} \left[\int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau)) \, \mathrm{d}\tau + Q(\mathbf{x}(t_f), t_f) \right]. \tag{9.35}$$

The value function can therefore be decomposed as:

$$V(\mathbf{x}(t_0), t_0, t_f) = V(\mathbf{x}(t_0), t_0, t) + V(\mathbf{x}(t), t, t_f). \tag{9.36}$$

This makes it possible to take the total time derivative above. A more rigorous derivation is possible using the calculus of variations.

[4] Kalman recognized that the Bellman optimal control formulation was a generalization of the Hamilton-Jacobi equation from classical mechanics to handle stochastic input–output systems. These formulations all involve the calculus of variations, which we have seen in Section 5.5.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

The HJB equation is incredibly powerful, providing a PDE for the optimal solution of a general nonlinear control problem. Typically, the HJB equation is solved numerically as a two-point boundary value problem, with boundary conditions $\mathbf{x}(0) = \mathbf{x}_0$ and $V(\mathbf{x}(t_f), t_f) = Q(\mathbf{x}(t_f), t_f)$, for example using a shooting method. However, a nonlinear control problem with a three-dimensional state vector $\mathbf{x} \in \mathbb{R}^3$ will result in a three-dimensional PDE. Thus, optimal nonlinear control based on the HJB equation typically suffers from the curse of dimensionality. Phase-space clustering techniques have shown promise in reducing the effective state-space dimension for systems that evolve on a low-dimensional attractor [193]. Neural ordinary differential equations from machine learning are also making it possible to solve much larger HJB equations [284].

Discrete-time HJB and the Bellman equation

Bellman optimality is especially intuitive for discrete-time system,

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x_k}, \mathbf{u_k}), \tag{9.37}$$

where instead of optimizing over a function, we optimize over a discrete control sequence. The objective is now given by

$$J(\mathbf{x}_0, \{\mathbf{u}_k\}_{k=0}^n, n) = \sum_{k=0}^{n} \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + Q(\mathbf{x}_n, t_n). \tag{9.38}$$

Similarly, the value function is defined as the value of the cumulative objective function, starting at a point $\mathbf{x}_0$ assuming an optimal control policy $\mathbf{u}$:

$$V(\mathbf{x}_0, n) = \min_{\{\mathbf{u}_k\}_{k=0}^n} J(\mathbf{x}_0, \{\mathbf{u}_k\}_{k=0}^n, n). \tag{9.39}$$

Bellman's principle of optimality states that at any point along the optimal trajectory $\mathbf{x}(t)$, the remaining control policy is optimal with respect to this new initial state

$$V(\mathbf{x}_0, n) = V(\mathbf{x}_0, k) + V(\mathbf{x}_k, n) \quad \forall k \in (0, n). \tag{9.40}$$

Thus, the value at an intermediate time step k may be written as

$$V(\mathbf{x}_k, n) = \left[\min_{\mathbf{u}_k} \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\right] + \underbrace{V(\mathbf{x}_{k+1}, n)}_{\text{s.t. } \mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k)} \tag{9.41a}$$

$$= \min_{u_k} \left[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + V(\mathbf{F}(\mathbf{x}_k, \mathbf{u}_k), n)\right]. \tag{9.41b}$$

It is also possible, given a value function $V(\mathbf{x}_k, n)$, to determine the next optimal control action $\mathbf{u}_k$ by returning the $\mathbf{u}_k$ that minimizes the above expression. This defines an optimal policy $\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$. Dropping the functional dependence of V on the end time, we then have

$$V(\mathbf{x}) = \min_{\mathbf{u}} \left[\mathcal{L}(\mathbf{x}, \mathbf{u}) + V(\mathbf{F}(\mathbf{x}, \mathbf{u}))\right] \tag{9.42a}$$

$$\boldsymbol{\pi}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{u}} \left[\mathcal{L}(\mathbf{x}, \mathbf{u}) + V(\mathbf{F}(\mathbf{x}, \mathbf{u}))\right]. \tag{9.42b}$$

These form the Bellman equations; they are the cornerstone of reinforcement learning, which we will explore in Section 9.3.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

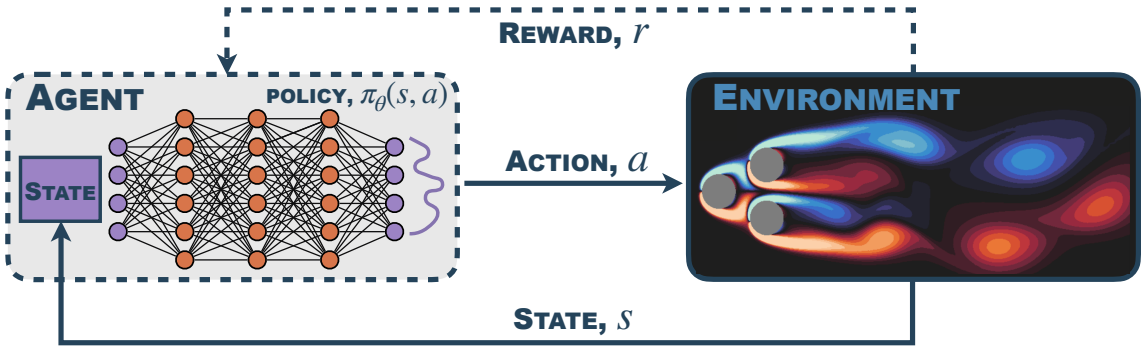


Figure 9.5: Schematic of reinforcement learning, where and agent senses its environmental state $\mathbf{s}$ and takes actions $\mathbf{a}$ according to a policy $\boldsymbol{\pi}$ that is optimized through learning to maximize future rewards $\mathbf{r}$. In this case, a deep neural network is used to represent the policy $\boldsymbol{\pi}$. This is known as a *deep policy network*.

9.3 Reinforcement Learning

Reinforcement learning (RL) is a major branch of machine learning that is concerned with how to learn control laws and policies to interact with a complex environment [333, 191]. Thus, RL is situated at the growing intersection of control theory and machine learning [289], and it is among the most promising fields of research towards generalized artificial intelligence and autonomy. Both machine learning and control theory fundamentally rely on optimization, and likewise, RL involves a set of optimization techniques within a framework for learning how to interact with the environment from experience.

In reinforcement learning[5], an agent senses the state of its environment, $\mathbf{s} \in \mathcal{S}$, and learns take appropriate actions, $\mathbf{a} \in \mathcal{A}$, to optimize future rewards $\mathbf{r}$; the sets $\mathcal{S}$ and $\mathcal{A}$ denote the sets of possible states and actions, respectively. Figure 9.5 provides a schematic overview of the reinforcement learning framework.

Reinforcement learning mimics how animals learn to interact with their environment through trial and error, through experience, through positive and negative rewards and feedback. Much of the history of reinforcement learning is linked to studies of animal behavior and the neurological basis of decisions, control, and learning [252, 316, 93, 92]. Pavlov's dog is an example demonstrating how animals learn environmental cues that will lead to a delayed reward [275]. The term *reinforcement* refers directly to the rewards, such as food, used to reinforce desirable behavior in humans and animals.

The Policy. The agent takes actions $\mathbf{a}$ through a control policy $\boldsymbol{\pi}$ that is optimized through learning to maximize future rewards $\mathbf{r}$. Rewards may be immediate or delayed, depending on the problem; in many cases, a circuitous path to the optimal solution must be learned. For example, a long term strategy is often needed to win a game of chess.

[5]Reinforcement learning uses different notation than control, although the two sets of notation can easily be mapped back and forth. The action $\mathbf{a}$ maps to the actuation $\mathbf{u}$, the state $\mathbf{s}$ maps to the full state $\mathbf{x}$ or partial measurement $\mathbf{y}$, and the policy $\pi(\mathbf{s})$ maps to the controller $\mathbf{u} = \mathbf{k}(\mathbf{x})$ or $\mathbf{u} = \mathbf{k}(\mathbf{y})$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

For deterministic systems, such as the optimal nonlinear control problem in Sec. 9.2, the policy may be written as $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$, directly analogous to the control law $\mathbf{u} = \mathbf{k}(\mathbf{x})$. However, when the environment is stochastic, we write the policy as $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$, which is the probability of taking action $\mathbf{a}$ given state $\mathbf{s}$ to maximize future rewards. This indicates that the policy is no longer deterministic, but is rather a probability density over actions $\mathbf{a}$.

In the simplest formulation, the policy may be a look-up table that is defined on the discrete state and action spaces $\mathcal{S}$ and $\mathcal{A}$, respectively. However, for most problems, this becomes prohibitively expensive, and $\boldsymbol{\pi}$ must be represented as an approximate function that is parameterized by $\boldsymbol{\theta}$, for example via a deep neural network:

$$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) \approx \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}). \tag{9.43}$$

Often, this parameterized function will be denoted $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a})$. We will revisit this function approximation in the context of deep reinforcement learning in Sec. **??**, where it is possible to represent these complex functions using deep neural networks.

The Environment. RL generalizes the optimal nonlinear control problem in Sec. 9.2. For example, the environment may be described by the deterministic, nonlinear dynamics in Eq. (9.30), or it may represent a less structured, stochastic set of causes and effects. For example, in a game of chess, the environment involves the rules of the game, along with the, sometimes, complex thought process of a human opponent. This is nearly impossible to write down as a tractable system of equations, and the set of states is almost innumerable. Thus, it is often the case that the model of the environment is not known. Further, the measured state of the system may be a partial measurement of a higher-dimensional state that evolves according to a stochastic, nonlinear process. Modern RL provides strategies for handling such high-dimensional, unstructured problems. We typically either use model-free RL strategies, or learn a data-driven model, for example using the techniques in Section 8.7, and proceed to model-based RL strategies. As is the case with human learning, RL often begins with an unstructured exploration, where trial-and-error are used to learn the environment and reward function, followed by exploitation, where a strategy is chosen and optimized.

For simplicity and pedagogical development of the theory, we typically assume that the environment is either a nonlinear dynamical system (deterministic or stochastic), or a simple Markov decision process (MDP). Here we will develop the theory for an MDP, as this is the most common approach.

In an MDP the probability of the system occurring in the current state is determined entirely by the previous state, so that it is memory-less. An MDP consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and a set of rewards $\mathcal{R}$, along with the probability of transitioning from state $\mathbf{s}_k$ at time t_k to state $\mathbf{s}_{k+1}$ at time t_{k+1} given action $\mathbf{a}_k$,

$$P(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \Pr\left(\mathbf{s}_{k+1} = \mathbf{s}' \mid \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a}\right), \tag{9.44}$$

and a reward function R

$$R(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \Pr\left(\mathbf{r}_{k+1} \mid \mathbf{s}_{k+1} = \mathbf{s}', \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a}\right). \tag{9.45}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Thus, an MDP generalizes the notion of a Markov process to include actions and rewards, making it suitable for decision making and control. Sometimes[6] the transition probability $P(\mathbf{s}', \mathbf{s}, \mathbf{a})$ will be written as $P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$.

A Markov process with a finite set of states $\mathcal{S}$, may be written in terms of the vector of probabilities of being in each of the n states $\mathbf{s} \in \mathbb{R}^n$. The Markov process $P(\mathbf{s}', \mathbf{s})$ is then given by a transition matrix $\mathbf{T}$, so that $\mathbf{s}' = \mathbf{T}\mathbf{s}$. Each column of $\mathbf{T}$ must add up to 1, which is a statement that given a particular state $\mathbf{s}$, something must happen in the transition to $\mathbf{s}'$.

Similarly, for an MDP, given a policy $\boldsymbol{\pi}$, the transition process may be written as

$$\mathbf{s}' = \sum_{\mathbf{a} \in \mathcal{A}} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) \mathbf{T}_{\mathbf{a}} \mathbf{s}. \tag{9.47}$$

Now for each action $\mathbf{a}$, $\mathbf{T}_{\mathbf{a}}$ is a Markov process with all columns summing to 1.

All Markov processes asymptotically approach a steady state $\boldsymbol{\mu}$, which is the eigenvector of $\mathbf{T}$ corresponding to eigenvalue 1. Given a policy $\boldsymbol{\pi}$, an MDP asymptotically approaches a steady state $\boldsymbol{\mu}_{\boldsymbol{\pi}} = \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) \boldsymbol{\mu}_{\mathbf{a}}$.

The Value Function. We have already seen the value function in the context of the HJB equation in Sec. 9.2. Here we define the value function in the context of RL for an MDP.

Given a policy $\boldsymbol{\pi}$, it is possible to define a value function that quantifies the desirability of being in a given state:

$$V_{\boldsymbol{\pi}}(\mathbf{s}) = \mathbb{E}\left(\sum_k \gamma^k \mathbf{r}_k \mid \mathbf{s}_0 = \mathbf{s} \right), \tag{9.48}$$

where $\mathbb{E}$ is the expected future total reward, subject to a discount rate γ. The optimal value function for the best policy is given by

$$V(\mathbf{s}) = \max_{\boldsymbol{\pi}} \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_k \mid \mathbf{s}_0 = \mathbf{s} \right). \tag{9.49}$$

Bellman's principle of optimality states that the value may be written recursively as

$$V(\mathbf{s}) = \max_{\boldsymbol{\pi}} \mathbb{E}\left(\mathbf{r}_0 + \sum_{k=1}^{\infty} \gamma^k \mathbf{r}_k \mid \mathbf{s}_1 = \mathbf{s}' \right) \tag{9.50}$$

which yields Bellman's equation:

$$V(\mathbf{s}) = \max_{\boldsymbol{\pi}} \mathbb{E}\left(\mathbf{r}_0 + \gamma V(\mathbf{s}') \right), \tag{9.51}$$

where $\mathbf{s}'$ is the next state after $\mathbf{s}$. This is true for $\mathbf{s}' = \mathbf{s}_{k+1}$ and $\mathbf{s} = \mathbf{s}_k$ for all k.

Given the value function, it is possible to extract the optimal policy as

$$\boldsymbol{\pi} = \underset{\boldsymbol{\pi}}{\operatorname{argmax}}\, \mathbb{E}\left(\mathbf{r}_0 + \gamma V(\mathbf{s}') \right), \tag{9.52}$$

[6] We might also replace the action $\mathbf{a}$ with a policy $\boldsymbol{\pi}$, which implies a sum over states:

$$P(\mathbf{s}', \mathbf{s}, \boldsymbol{\pi}) = \sum_{\mathbf{a} \in \mathcal{A}} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) P(\mathbf{s}', \mathbf{s}, \mathbf{a}). \tag{9.46}$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Optimization Techniques for Reinforcement Learning

Learning the optimal policy π, the value function V, or jointly learning both, all involve optimization. There are many approaches to learn an optimal policy π, which is the ultimate goal of RL. A major dichotomy in reinforcement learning is that of *model-based RL* versus *model-free RL*. When there is a known model for the environment, there are several strategies for learning either the optimal policy or value function through what is known as *policy iteration* or *value iteration*, which are forms of dynamic programming using the Bellman equation. We will discuss dynamic programming in the next section.

Depending on the assumed structure of π, the size and evolution dynamics of $\mathcal{S}$, and the reward landscape R, determining an optimal policy may range from a closed form optimization to a rather high-dimensional unstructured optimization. When there is no model for the environment, model-free strategies, such as Q-learning, must be employed. The reinforcement learning optimization problem may be particularly challenging for high-dimensional systems with unknown, nonlinear, stochastic dynamics and sparse and delayed rewards. All of these techniques may be combined with function approximation techniques, such as neural networks, for approximating the policy π, the value function V, or the quality function Q (discussed in Section **??**).

One of the central challenges in reinforcement learning is that rewards are often extremely rare and may be significantly delayed from a sequence of good control actions. This leads to the so-called credit assignment problem, coined by Minsky [251] to describe the challenge of knowing what action sequence was responsible for the delayed reward. These sparse and delayed rewards have been a central challenge in RL for six decades, and they are still a focus of research today. The resulting optimization problem is computationally expensive and data intensive, requiring considerable trial and error.

Today, reinforcement learning is being used to learn sophisticated control policies for complex open-world problems in autonomy and propulsion (e.g., self-driving cars, learning to swim and fly, etc.) and as a general learning environment for rule-constrained games (e.g., checkers, backgammon, chess, go, Atari, etc.). Much of the history of RL may be traced through the success on increasingly challenging games, from checkers [305] to backgammon [338] and more recently to chess and go [320]. These games serve to illustrate many of the central challenges that are still faced in RL, including the curse of dimensionality and the credit assignment problem.

The recursive nature of the value function, via Bellman's principle of optimality, is directly leveraged in reinforcement learning, where the task of learning optimal control policies is framed as estimating the value function. While classical HJB approaches rely on solving partial differential equations, RL replaces this with iterative approximations, often through simulation or data-driven methods. This connection establishes RL as a computational framework for solving Bellman-like equations in settings where models of the system dynamics may be unknown or high-dimensional. The Bellman equation reformulates the HJB's value function PDE into a recursive relationship that can be solved iteratively. This iterative structure forms the basis for RL techniques such as temporal-difference learning, which approximates the value function without explicitly solving the PDE. Modern RL techniques extend these ideas by leveraging data-driven approximations, such as neural networks.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

9.4 Dynamic Programming

Dynamic programming (DP) is a recursive optimization framework introduced by Richard E. Bellman [26, 27] to solve large multi-step optimization problems, such as those found in decision making and control, by decomposing them into simpler subproblems. In the previous sections, we derived the HJB equation as a partial differential equation that must be satisfied by the value function for optimal nonlinear control problems. While powerful and surprisingly compact, the HJB equation is generally very challenging to solve analytically. Instead, we typically solve HJB approximately using dynamic programming, which provides a constructive and recursive way to compute the value function and optimal control policy.

Policy iteration and value iteration are two examples of how we use dynamic programming in optimal nonlinear control and reinforcement learning. Specifically, dynamic programming reformulates the large optimization problem as a recursive optimization in terms of smaller sub-problems, so that local improvements to the control policy may be made sequentially with new information. This approach relies on Bellman's principle of optimality, which states that a large multi-step control policy must also be locally optimal in every sub-sequence of steps. RL generalizes these ideas by introducing function approximation techniques to handle high-dimensional state spaces and by developing model-free approaches when the dynamics are unknown. In essence, DP is the optimization backbone of classic RL, while modern RL extends DP to problems with uncertainty and complex, high-dimensional dynamics.

The Bellman equation in Eq. (9.50) indicates that the large optimization problem over an entire state-action trajectory $(\mathbf{s}_k, \mathbf{a}_k)$ may be broken into a recursive optimization at each point along the trajectory. As long as the value function is known at the next point $\mathbf{s}'$, it is possible to solve the optimization at point $\mathbf{s}$ simply by optimizing the policy $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$ at this point. Of course, this assumes that the value function is known at *all* possible next states $\mathbf{s}_{k+1}$, which is a function of the current state $\mathbf{s}_k$, the current action $\mathbf{a}_k$, and the dynamics governing the system; this becomes even more complex for non-MDP dynamics, such as the nonlinear control formulation in Section 9.2. For even moderately large problems, this suffers from the curse of dimensionality, and approximate solution methods must be employed.

When tractable, dynamic programming (i.e., the process of breaking a large problem into smaller overlapping sub-problems) provides a globally optimal solution. There are two main approaches to dynamic programming, referred to as *top down* and *bottom up*:

> **Top down:** The top-down approach involves maintaining a table of sub-problems that are referred to when solving larger problems. For a new problem, the table is checked to see if the relevant sub-problem has been solved. If so, it is used, and if not, the sub-problem is solved. This tabular storage is called *memoization*. This approach becomes combinatorially complex for many problems.
>
> **Bottom up:** The bottom-up approach involves first solving the smallest sub-problems, and then combining these to form the larger problems. This may be thought of as working backwards from every possible goal state, finding the best previous action to get there, then going back two steps, then going back three steps, etc.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Although dynamic programming still represents a brute-force search through all sub-problems, it is still more efficient than a naive brute-force search. In some cases, it reduces the computational complexity to an algorithm that scales linearly with the number of sub-problems, although this may still be combinatorially large, as in the example of the game of chess. Dynamic programming is closely related to divide-and-conquer techniques, such as quick sort, except that divide-and-conquer applies to *non-overlapping* or *non-recursive* (i.e., independent) sub-problems, while dynamic programming applies to overlapping, or recursively interdependent sub-problems.

However, the recursive strategy suggests approximate solution techniques, such as the alternating directions method, where a sub-optimal solution is initialized and the value function is iterated over. This forms the basis of policy and value iteration.

Policy iteration

Policy iteration is a two step optimization procedure to simultaneously find an optimal value function $V_{\boldsymbol{\pi}}$ and the corresponding optimal policy $\boldsymbol{\pi}$.

First, a candidate policy $\boldsymbol{\pi}$ is evaluated, resulting in the value function for this fixed policy. This typically involves a brute force calculation of the value function for this policy starting at many or all initial states. The policy may need to be run for a long time depending on the reward delay and discounting factor γ.

Next, the value function is fixed, and the policy is optimized to improve the expected rewards by taking different actions at a given state. This optimization relies on the alternative recursive formulation of the value function in Eq. (9.48) due to Bellman's Eq. (9.50):

$$V_{\boldsymbol{\pi}}(\mathbf{s}) = \mathbb{E}\left[R(\mathbf{s}', \mathbf{s}, \boldsymbol{\pi}(\mathbf{s})) + \gamma V_{\boldsymbol{\pi}}(\mathbf{s}')\right] \tag{9.53a}$$

$$= \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \boldsymbol{\pi}(\mathbf{s}))\left(R(\mathbf{s}', \mathbf{s}, \boldsymbol{\pi}(\mathbf{s})) + \gamma V_{\boldsymbol{\pi}}(\mathbf{s}')\right). \tag{9.53b}$$

Note that in this expression, we have assumed a deterministic policy $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$, otherwise, Eq. (9.53a) would involve a second summation over $\mathbf{a} \in \mathcal{A}$, with the expression multiplied by $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$.

It is then possible to hold $V_{\boldsymbol{\pi}}(\mathbf{s})$ fixed and optimize over the policy in the first term. In particular, the new deterministic optimal policy at the state $\mathbf{s}$ is given by:

$$\boldsymbol{\pi}(\mathbf{s}) = \underset{\mathbf{a}\in\mathcal{A}}{\operatorname{argmax}}\ \mathbb{E}\left[R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V_{\boldsymbol{\pi}}(\mathbf{s}')\right]. \tag{9.54}$$

Once the policy is updated, the process repeats, fixing this policy to update the value function, and then using this updated value function to improve the policy. The process is repeated until both the policy and the value function converge to within a specified tolerance. It is important to note that this procedure is both expensive and prone to local minima. It also resembles the alternating descent method that is widely used in optimization and machine learning.

The formulation in Eq. (9.53a) makes it clear that it may be possible to optimize backwards from a state known to give a reward with high probability. Additionally, this approach requires having a model for P and R to predict the next state $\mathbf{s}'$, making this a *model-based* approach.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Value iteration

Value iteration is similar to policy iteration, except that at every iteration only the value function is updated, and the optimal policy is extracted from this value function at the end. First, the value function is initialized, typically either with zeros or at random. Then, for all states $\mathbf{s} \in \mathcal{S}$, the value function is updated by returning the maximum value at that state across all actions $\mathbf{a} \in \mathcal{A}$, holding the value function fixed at all other states $\mathbf{s}' \in \mathcal{S} \backslash \mathbf{s}$:

$$V(\mathbf{s}) = \max_{\mathbf{a}} \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') \right). \tag{9.55}$$

This iteration is repeated until a convergence criterion is met.

After the value function converges, it is possible to extract the optimizing policy $\boldsymbol{\pi}$:

$$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) = \operatorname*{argmax}_{\mathbf{a}} \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') \right). \tag{9.56}$$

Although value iteration typically requires less steps per iteration, policy iteration often converges in less iterations. This may be due to the fact that the value function is often more complex than the policy function, requiring more parameters to optimize over.

Note that the value function in RL typically refers to a discounted sum of future rewards that should be maximized, while in nonlinear control it refers to an integrated cost that should be minimized. The phrase *value function* is particularly intuitive when referring to accumulated rewards in the economic sense, as it quantifies the *value* of being in a given state. However, in the case of nonlinear control theory, the *value function* is more accurately thought of as quantifying the *numerical value* of the objective function evaluated on the optimal trajectory. This notation can be confusing, and is worth careful consideration depending on the context.

Quality function

Policy iteration and value iteration are related by the quality function $Q(\mathbf{s}, \mathbf{a})$

$$\begin{aligned} Q(\mathbf{s}, \mathbf{a}) &= \mathbb{E}\left[R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') \right] && \text{(9.57a)} \\ &= \sum_{\mathbf{s}'} P\left(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}\right) \left(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') \right). && \text{(9.57b)} \end{aligned}$$

In a sense, the optimal policy $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$ and the optimal value function $V(\mathbf{s}, \mathbf{a})$ contain redundant information, as one can be determined from the other via the quality function $Q(\mathbf{s}, \mathbf{a})$:

$$\begin{aligned} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) &= \operatorname*{argmax}_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) && \text{(9.58a)} \\ V(\mathbf{s}) &= \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}). && \text{(9.58b)} \end{aligned}$$

This formulation is used for model-free Q-learning [366, 351, 123].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

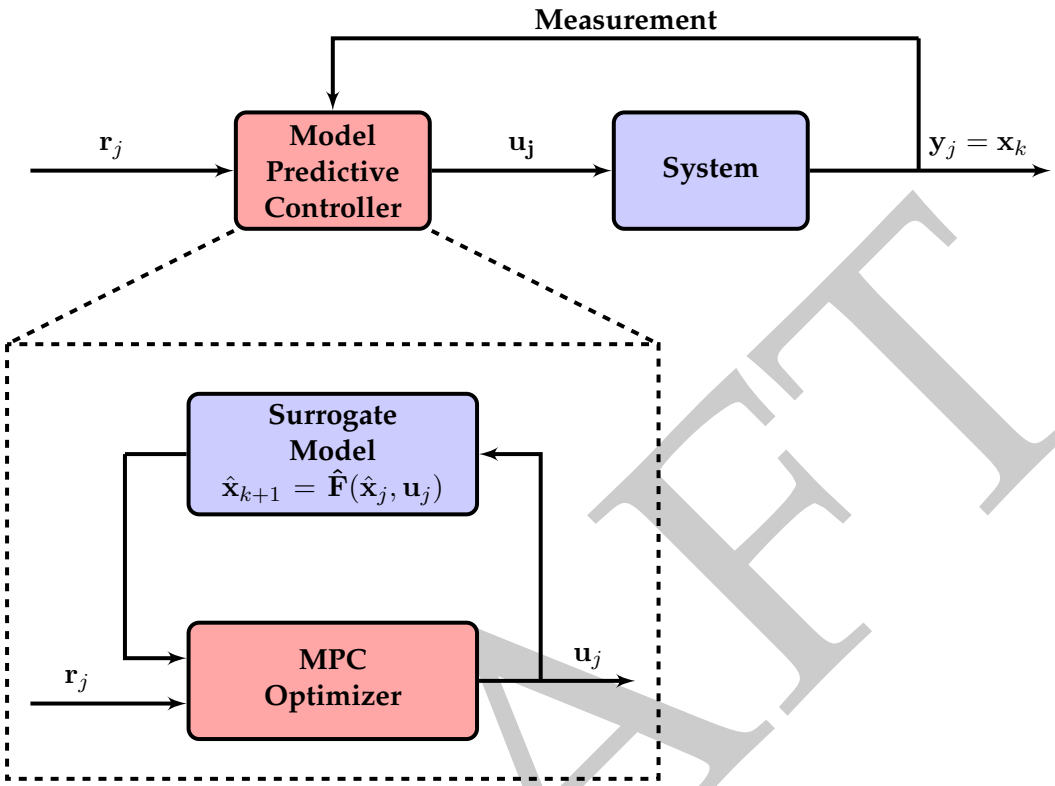


Figure 9.6: Schematic of model predictive control, where a surrogate model is used to run an optimization directly inside the control loop. This diagram assumes full state measurements $\mathbf{y} = \mathbf{x}$ for simplicity, although this is not strictly necessary for MPC.

9.5 Model Predictive Control

Model predictive control (MPC) [65, 137] is one of the most powerful, flexible, and widely used techniques in industrial control. It is also one of the most computationally intensive. MPC uses a constrained optimization loop, wrapped around a predictive model, to control dynamical systems with constraints, time delays, non-minimum phase dynamics, and instability. This optimization loop must be performed in real-time, which typically necessitates either expensive computational hardware or simplified models. Progress in MPC has directly tracked our ability to scale the complexity of this real-time optimization. Many industrial applications of MPC have slow time scales and use linear models, because they are simple to identify and fast to optimize over. However, improved computational hardware and optimization algorithms are enabling more sophisticated applications of MPC to nonlinear systems with faster time scales. For example, field-programmable gate arrays (FPGAs) are being used for embedded MPC in some applications [176, 28]. Recently, deep learning and reinforcement learning have been combined effectively with MPC [220, 382].

Model predictive control is shown schematically in Fig. 9.6. MPC determines the next immediate control action by solving a constrained optimal control problem over a reced-

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

ing horizon. The control sequence is then implemented until the system state is updated via sensor measurements, after which the receding horizon control problem is reinitialized and solved again. This iterative procedure is repeated, balancing model prediction with sensor feedback. The predictive model is used to forecast control scenarios into the future, providing fast open-loop control, while the sensor information provides a corrective feedback signal to account for instability, uncertainty, model-mismatch, and disturbances.

Mathematical Formulation of MPC

In MPC, the open-loop control input $\mathbf{u}$ is optimized on a receding time-horizon $t_c = m_c \Delta t$ to minimize a cost J over a prediction horizon $t_p = m_p \Delta t$, based on a surrogate model

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{F}}(\hat{\mathbf{x}}_j, \mathbf{u}_j). \tag{9.59}$$

The control horizon is usually less than or equal to the prediction horizon, and $\mathbf{u}$ is held constant for $t \in [t_c, t_p]$. The objective function is typically similar to an LQR problem

$$J(\mathbf{x}_j) = \sum_{k=0}^{m_p-1} \|\hat{\mathbf{x}}_{j+k} - \mathbf{r}_{j+k}\|_{\mathbf{Q}}^2 + \sum_{k=1}^{m_c-1} \left(\|\mathbf{u}_{j+k}\|_{\mathbf{R}}^2 + \|\Delta \mathbf{u}_{j+k}\|_{\mathbf{R}_\Delta}^2 \right) \tag{9.60}$$

where $\mathbf{r}_j$ is the reference trajectory to be tracked by MPC, $\|\mathbf{x}\|_{\mathbf{Q}}^2 = \mathbf{x}^T \mathbf{Q} \mathbf{x}$, and there is an additional penalty on large changes in the control, i.e. on $\Delta \mathbf{u}_j = \mathbf{u}_j - \mathbf{u}_{j-1}$. Note that the weight matrix $\mathbf{Q}$ must be positive semidefinite and $\mathbf{R}$ and $\mathbf{R}_\Delta$ must be positive semidefinite, as in LQR. It is also possible to add a terminal cost on the final state.

The objective function is then optimized over the sequence $\mathbf{U}_j = \{\mathbf{u}_{j+1}, \cdots, \mathbf{u}_{j+k}, \cdots, \mathbf{u}_{j+m_c}\}$ of control inputs, subject to the constraints of the surrogate model in Eq. (9.59), constraints on the control inputs $\mathbf{u}_j$, the rate of change of control $\Delta \mathbf{u}_j$, and the predicted state $\hat{\mathbf{x}}_j$:

$$\begin{aligned}
\min_{\mathbf{U}_j} \quad & J(\mathbf{x}_j) && \text{(9.61a)} \\
\text{s.t.} \quad & \hat{\mathbf{x}}_{k+1} = \hat{\mathbf{F}}(\hat{\mathbf{x}}_j, \mathbf{u}_j) && \text{(9.61b)} \\
& \mathbf{u}_{\min} \leq \mathbf{u}_j \leq \mathbf{u}_{\max} && \text{(9.61c)} \\
& \Delta \mathbf{u}_{\min} \leq \Delta \mathbf{u}_j \leq \Delta \mathbf{u}_{\max} && \text{(9.61d)} \\
& \mathbf{x}_{\min} \leq \hat{\mathbf{x}}_j \leq \mathbf{x}_{\max}. && \text{(9.61e)}
\end{aligned}$$

This is a large-scale nonlinear program (NLP). Efficient solvers, such as IPOPT and SNOPT, can handle inequality constraints, high-dimensional state and control vectors, and complex objective function terms. Optimization details will be discussed more below.

The optimal control is then applied for one time step, or until sensor measurements are available to update the state $\mathbf{x}$, and the procedure is repeated and the receding-horizon control re-optimized at each subsequent time step. For simplicity, we will assume that sensor measurements are available at each time step. This results in the control law:

$$\mathbf{K}(\mathbf{x}_j) = \mathbf{u}_{j+1}(\mathbf{x}_j), \tag{9.62}$$

where $\mathbf{u}_{j+1}$ is the first time step of the optimized actuation starting at $\mathbf{x}_j$. This is illustrated in Fig. 9.7. For more details, see Kaiser et al. [192] and Fonzi et al. [132].

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

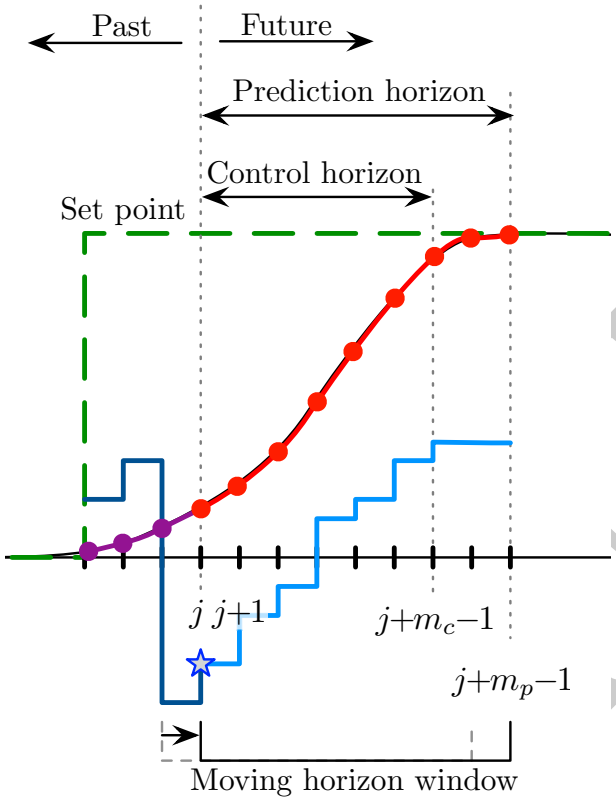


Figure 9.7: Illustration of model predictive control used to track a set point, where the actuation input $\mathbf{u}$ is iteratively optimized over a receding horizon. *Reproduced with permission from Kaiser et al. [192].*

It is possible to optimize highly customized objective functions, subject to nonlinear dynamics, with constraints on the actuation and state. However, the computational requirements of re-optimizing at each time-step are considerable, putting limits on the complexity of the model and optimization techniques. Fortunately, rapid advances in computing power and optimization are enabling MPC for real-time nonlinear control.

There have been tremendous advances in MPC, especially deep MPC, which uses deep learning for the surrogate model [220]. Deep MPC has been widely used, including for vision based driving systems [111], and controlling fluid flows [256, 37] and aeroelastic systems [132]. Deep MPC is highly useful in data-rich applications, although deep learning requires large volumes of data and may not be readily interpretable. A complementary line of research seeks to identify models for MPC based on limited data. For example, Kaiser et al. [192] showed that it is possible to rapidly identify system models from limited data using the methods discussed in Sec. 8.7, and then use these models for MPC.

Tube MPC is another robust strategy that keeps the system within a tube around the reference trajectory [231, 124], which is useful for safety critical systems with model error and disturbances. More generally, developing robust and distributed MPC algorithms with guarantees [293, 294] is important in autonomy and robotics. MPC may also be used in model-based reinforcement learning [370]. Finally, recent efforts have combined differentiable programming with MPC, resulting in differentiable predictive control [5, 112].

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Optimization Algorithms for MPC

Optimization is at the heart of MPC, and its feasibility in real-time applications depends on efficient solvers. Historically, the development of optimization techniques for MPC has closely followed advances in numerical methods and computing power. Mayne [246] provides a nice discussion of these trends and developments.

When the dynamics are linear and the objective function are quadratic and positive definite, and when the control, state and terminal constraints are convex, then the MPC problem is a strictly convex quadratic program (QP), which has a globally optimal solution that may be effectively computed. There are several established computational techniques that may be applied, many of which have been discussed earlier in this book. These approaches include active set methods [250, 125, 202, 64, 166], interior point methods [287, 99, 103], and gradient projection methods, for example using Nesterov's accelerated gradient [295, 205]. Early applications of MPC in the 1980s [137] primarily relied on these methods due to their simplicity and robustness. Active set methods were particularly popular for their ability to handle constraints directly and exploit the sparsity structure of the constraint and dynamics matrices, leading to faster and more memory-efficient solutions. As the scale of industrial problems grew, interior-point methods gained traction for their superior performance on large-scale QPs.

Nonlinear MPC (NMPC) requires the solution of a nonlinear programming (NLP) problem at each time step. It is common to use sequential quadratic programming (SQP), based on a local quadratic approximation to the optimal control problem at each iteration, which is then solved using the standard QP algorithms above. Each QP subproblem is constructed by approximating the Lagrangian with a second-order Taylor series expansion, using the Hessian of the Lagrangian to form the quadratic objective of the QP subproblem. When Hessian information is unavailable, it is possible to use quasi-Newton methods, as discussed in Sec. 2.5. Solvers like IPOPT [362] and SNOPT [144] have been widely adopted for NMPC due to their robustness and ability to handle large-scale problems with nonlinear constraints.

Recent developments have focused on first-order operator splitting methods, such as the alternating direction method of multipliers (ADMM) [47], which was discussed in Section 6.5. Splitting methods address the need for fast and scalable optimization. ADMM is particularly attractive for embedded MPC applications, where computational resources are limited. Its iterative nature and ability to exploit sparsity make it suitable for distributed and structured MPC, where the optimization problem is decomposed into smaller sub-problems. Robust operator-splitting solvers, such as OSQP [326], which is based on an ADMM framework tailored for QPs, has further advanced MPC. OSQP leverages the problem structure to achieve real-time performance on embedded systems, enabling MPC to be deployed in resource-constrained environments such as robotics and automotive control.

Despite these advances, challenges remain. Solvers for NMPC are computationally intensive and sensitive to the choice of initial guesses and regularization parameters. Robustness to model inaccuracies and disturbances is another critical area of research, especially for safety-critical applications. Future trends include the integration of machine learning for better surrogate models and leveraging hardware accelerators, such as GPUs, to further speed up computations.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Linear Time-Varying Approximations

In practice, solving the full nonlinear MPC problem in real-time is often infeasible. Instead, linear time-varying (LTV) approximations are employed to simplify the optimization. The idea is to linearize the nonlinear dynamics $\hat{\mathbf{F}}(\hat{\mathbf{x}}, \mathbf{u})$ around the current operating point $(\hat{\mathbf{x}}_j, \mathbf{u}_j)$ at each time step:

$$\hat{\mathbf{F}}(\hat{\mathbf{x}}, \mathbf{u}) \approx \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{c}, \tag{9.63}$$

where $\mathbf{A} = \frac{\partial \hat{\mathbf{F}}}{\partial \hat{\mathbf{x}}}\big|_{\hat{\mathbf{x}}_j, \mathbf{u}_j}, \mathbf{B} = \frac{\partial \hat{\mathbf{F}}}{\partial \mathbf{u}}\big|_{\hat{\mathbf{x}}_j, \mathbf{u}_j}, \mathbf{c} = \hat{\mathbf{F}}(\hat{\mathbf{x}}_j, \mathbf{u}_j) - \mathbf{A}\hat{\mathbf{x}}_j - \mathbf{B}\mathbf{u}_j$.

The resulting optimization problem is again a QP problem:

$$\begin{aligned} \min_{\mathbf{U}_j} \quad & \sum_{k=0}^{m_p-1} \|\hat{\mathbf{x}}_{j+k} - \mathbf{r}_{j+k}\|_{\mathbf{Q}}^2 + \sum_{k=1}^{m_c-1} \left(\|\mathbf{u}_{j+k}\|_{\mathbf{R}}^2 + \|\Delta\mathbf{u}_{j+k}\|_{\mathbf{R}_\Delta}^2\right) \\ \text{s.t.} \quad & \hat{\mathbf{x}}_{j+k+1} = \mathbf{A}\hat{\mathbf{x}}_{j+k} + \mathbf{B}\mathbf{u}_{j+k} + \mathbf{c}, \quad k = 0, \ldots, m_p - 1. \end{aligned}$$

By updating the linearization at each time step, LTV approximations strike a balance between computational feasibility and accuracy.

Warm-Starting and Real-Time Optimization

Warm-starting is critical to ensure that the MPC optimization can be solved in the system's sampling interval. The solution from the previous time step is used as to initialize the current optimization problem, reducing the number of iterations required for convergence. In linear MPC, warm-starting involves initializing the QP solver with the previous control sequence. This approach is particularly effective when the system operates near a steady state or follows a slowly varying trajectory.

Warm-starting dates back to the early applications of MPC in the 1980s, when computational resources were limited. Active set methods naturally benefited from warm-starting because the active set of constraints at the solution tends to change gradually over time. By initializing the active set with the previous solution, the algorithm converges faster. Similarly, interior-point methods benefit from starting near the optimal trajectory, reducing the number of Newton iterations required.

Real-time optimization solvers, such as FORCES [102] and qpOASES [126], are explicitly designed to exploit warm-starting. These solvers take advantage of problem sparsity and the structure of the QP to achieve fast convergence. In embedded systems, where computational resources are constrained, warm-starting is essential for meeting the stringent timing requirements of real-time control.

One limitation of warm-starting is its reliance on the smooth evolution of the optimization problem. Abrupt changes in the system dynamics or reference trajectory can lead to slower convergence or even failure to converge. To mitigate this, robust optimization techniques and adaptive regularization strategies are often employed. Future directions include integrating warm-starting with machine learning models to better anticipate system changes and further improve solver performance.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

9.6 Linear Matrix Inequalities

Linear matrix inequalities (LMIs) are a class of convex constraints that play a central role in modern control theory [46, 308]. They provide a unifying framework for solving a wide range of optimization and control problems, particularly those involving stability, robustness, and performance guarantees. LMIs are particularly appealing in control because they are convex, and efficient numerical solvers based on interior-point methods can solve them reliably. Many classical control problems, including stability, LQR, the Kalman filter, and robust control, can be reformulated as LMIs. LMI formulations are widely used in optimal control and MPC to encode constraints on states, inputs, and system dynamics. For instance, quadratic objectives can be cast into an LMI framework, as will be shown.

Motivating LMI: Lyapunov's Stability Problem

We follow Boyd et al. [46] and begin by motivating LMIs with their origin over a century ago. Lyapunov's direct method [234] states that a linear system

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x}$$

is stable if and only if there exists a positive definite matrix $\mathbf{P}$ such that

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} < \mathbf{0} \qquad \mathbf{P} > \mathbf{0}.$$

This is known as a Lyapunov inequality, and it is the original LMI, formulated in 1892.

Further, Lyapunov showed how this LMI can be solved by choosing any positive definite, self-adjoint matrix $\mathbf{Q} = \mathbf{Q}^T > \mathbf{0}$ and solving the linear system of equations

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} = -\mathbf{Q}$$

for $\mathbf{P}$. If $\mathbf{P}$ exists, the system is stable, and $\mathbf{V}(\mathbf{x}) = \mathbf{x}^T\mathbf{P}\mathbf{x}$ is a Lyapunov function.

LMIs were further developed in the 1960s and 1970s, resulting in many of the forms described here. Boyd et al. [46] provide a complete and interesting account of the history of LMIs in control. One notable development was by Willems in 1971 [369], who noted that LMIs can be formulated as convex optimization problems, providing computational and algorithmic benefits. The development of IPMs in the 1980s made it possible to solve LMIs efficiently at scale, and is the foundation for many modern algorithms in control.

Two Mathematical Forms of LMIs

An LMI is an inequality of the following form:

$$\mathbf{F}(\mathbf{X}) < \mathbf{Q}. \tag{9.64}$$

$\mathbf{X} \in \boldsymbol{\mathcal{X}}$ is an element of a real vector space, $\mathbf{F} : \boldsymbol{\mathcal{X}} \to \mathbb{H}^n$ is linear in $\mathbf{X}$, and $\mathbf{Q} \in \mathbb{H}^n$. $\mathbb{H}^n$ is the space of $n \times n$ Hermitian matrices, so $\mathbf{Q} \in \mathbb{H}^n$ implies $\mathbf{Q} = \mathbf{Q}^T$. $\mathbf{X}$ may be a vector, $\boldsymbol{\mathcal{X}} = \mathbb{R}^n$, or a symmetric $n \times n$ matrix, $\boldsymbol{\mathcal{X}} = \mathbb{S}^n$. An example will help make this clear:

- $\mathbf{A}^T\mathbf{X}\mathbf{A} - \mathbf{X} < \mathbf{Q}$, where $\mathbf{X} \in \mathbb{S}^n$ is a symmetric matrix.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

In general, it is possible to write an LMI in the following form

$$\mathbf{A}_0 + \sum_{i=1}^{m} x_i \mathbf{A}_i > 0, \tag{9.65}$$

where $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_m$ are symmetric matrices, $x_1, x_2, \ldots, x_m$ are decision variables, and $>$ denotes positive definiteness, meaning that all eigenvalues of the matrix are nonnegative. It is also possible to have positive-semidefinite problems, which are called non-strict [46].

A simple optimization problem involving LMIs is the feasibility problem, to compute a $\mathbf{X} \in \mathcal{X}$ such that $\mathbf{F}(\mathbf{X}) < \mathbf{Q}$. The set of such $\mathbf{X}$ is convex.

Casting Quadratic Problems as LMIs

One of the strengths of the LMI framework is that it may be possible to recast seemingly nonlinear problems into an LMI. We will illustrate this with a quadratic constraint that occurs commonly in control:

$$\mathbf{X}^T \mathbf{R}^{-1} \mathbf{X} - \mathbf{Q} < \mathbf{0} \tag{9.66}$$

This is not an LMI, since it is quadratic in the state $\mathbf{X}$. However, using the Schur complement formula, derived below, this may be recast as the following LMI:

$$\begin{bmatrix} \mathbf{Q} & \mathbf{X}^T \mathbf{A}^T \\ \mathbf{A}\mathbf{X} & \mathbf{R} \end{bmatrix} > \mathbf{0}. \tag{9.67}$$

The Schur complement formula states that the following set of matrix inequalities

$$\mathbf{R} > \mathbf{0} \tag{9.68a}$$

$$\mathbf{Q} - \mathbf{X}\mathbf{R}^{-1}\mathbf{X}^T > \mathbf{0} \tag{9.68b}$$

is equivalent to

$$\begin{bmatrix} \mathbf{Q} & \mathbf{X} \\ \mathbf{X}^T & \mathbf{R} \end{bmatrix} > \mathbf{0}. \tag{9.69}$$

To see this, we take the first set of equations, written in matrix form:

$$\begin{bmatrix} \mathbf{Q} - \mathbf{X}\mathbf{R}^{-1}\mathbf{X}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{bmatrix} > \mathbf{0} \tag{9.70}$$

and apply the following change of coordinates by multiplying by

$$\begin{bmatrix} \mathbf{I} & \mathbf{X}\mathbf{R}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{9.71}$$

on the left and by its adjoint on the right:

$$\begin{bmatrix} \mathbf{I} & \mathbf{X}\mathbf{R}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{Q} - \mathbf{X}\mathbf{R}^{-1}\mathbf{X}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{R}^{-1}\mathbf{X}^T & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{Q} & \mathbf{X} \\ \mathbf{X}^T & \mathbf{R} \end{bmatrix} > \mathbf{0}. \tag{9.72}$$

This inequality is greater than zero because the change of coordinates matrix is non-singular and positive definite.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

LQR as an LMI

Recall that the LQR optimal controller minimizes a quadratic cost:

$$J = \int_0^\infty \left(\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right)\,\mathrm{d}t, \tag{9.73}$$

where $\mathbf{Q} \geq \mathbf{0}$ and $\mathbf{R} > \mathbf{0}$ are weight matrices. The optimal control input is derived from the solution of the continuous-time algebraic Riccati equation

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{Q} = \mathbf{0}. \tag{9.74}$$

This can be reformulated as an LMI using the Schur complement formula from above:

$$\begin{bmatrix} \mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q} & \mathbf{P}\mathbf{B} \\ \mathbf{B}^T\mathbf{P} & -\mathbf{R} \end{bmatrix} < \mathbf{0}. \tag{9.75}$$

Solving this LMI for $\mathbf{P}$ provides the optimal state feedback gain $\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}$.

Kalman Filtering as an LMI

The Kalman filter gain for an optimal state estimator is likewise derived from the solution of the Riccati equation:

$$\mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^T - \mathbf{P}\mathbf{C}^T\mathbf{R}^{-1}\mathbf{C}\mathbf{P} + \mathbf{Q} = \mathbf{0}. \tag{9.76}$$

This can be recast as an LMI:

$$\begin{bmatrix} \mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^T + \mathbf{Q} & \mathbf{P}\mathbf{C}^T \\ \mathbf{C}\mathbf{P} & -\mathbf{R} \end{bmatrix} < \mathbf{0}. \tag{9.77}$$

The solution $\mathbf{P}$ provides the Kalman gain $\mathbf{K} = \mathbf{P}\mathbf{C}^T\mathbf{R}^{-1}$.

Robust H_∞ Control

Robust control designs often require ensuring stability and performance under model uncertainties. The H_∞ control problem seeks to design a controller that minimizes the worst-case gain from disturbance to output:

$$\|\mathbf{T}_{\mathbf{w}\to\mathbf{y}}\|_\infty < \gamma, \tag{9.78}$$

where γ is a scalar bound. This can be formulated as an LMI:

$$\begin{bmatrix} \mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} & \mathbf{P}\mathbf{B} & \mathbf{C}^T \\ \mathbf{B}^T\mathbf{P} & -\gamma\mathbf{I} & \mathbf{D}^T \\ \mathbf{C} & \mathbf{D} & -\gamma\mathbf{I} \end{bmatrix} < \mathbf{0}. \tag{9.79}$$

where $\mathbf{P}$ and γ are decision variables. Solving this LMI provides the controller gains that ensure robustness, as well as the performance bound γ.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Suggested Reading

Texts

(1) A course in robust control theory: A convex approach, by G. E. Dullerud and F. Paganini, 200 [116].

(2) Reinforcement learning: An introduction, by R. S. Sutton and A. G. Barto, 1998 [333].

(3) Optimal control and estimation, by R. F. Stengel, 2012 [327].

(4) Multivariable feedback control: analysis and design, by S. Skogestad and I. Postlethwaite, 2005 [321].

(5) Real-time optimization by extremum-seeking control, by K. B. Ariyur and M. Krstić, 2003 [10].

Papers and reviews

(1) A tour of reinforcement learning: The view from continuous control, by B. Recht, *Annual Review of Control, Robotics, and Autonomous Systems*, 2019 [289].

(2) Model predictive control: theory and practice—a survey, by C. E. Garcia, D. M. Prett, and M. Morari, *Automatica*, 1989 [137].

(3) Constrained model predictive control: Stability and optimality, by D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, *Automatica*, 2000 [247].

(4) Differentiable Predictive Control: An MPC Alternative for Unknown Nonlinear Systems using Constrained Deep Learning, by J. Drgona, K. Kis, A. Tuor, D. Vrabie, and M. Klauco, *arXiv*, 2020 [112].

(5) Model predictive control, by E. F. Camacho and C. B. Alba, 2013 [65].

(6) Sparse identification of nonlinear dynamics for model predictive control in the low-data limit, by E. Kaiser, J. N. Kutz, and S. L. Brunton, *Proceedings of the Royal Society of London A*, 2018 [192].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

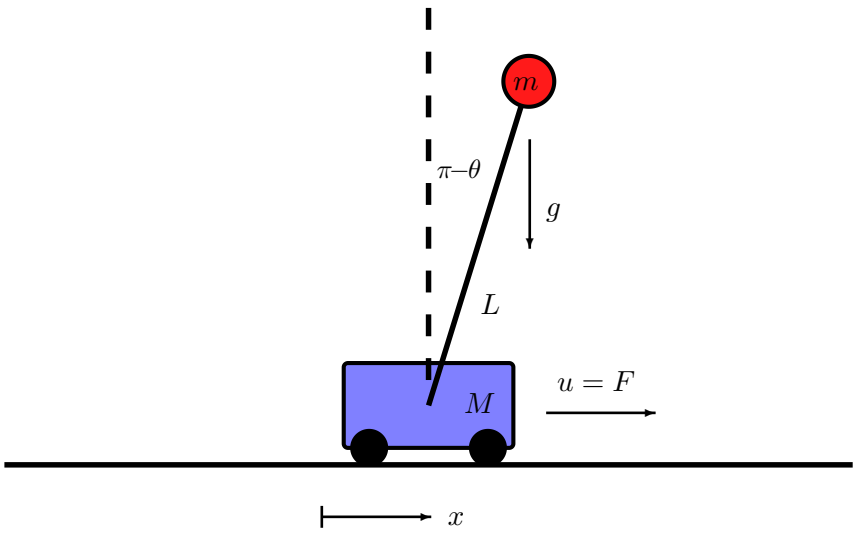


Figure 9.8: Schematic of inverted pendulum on a cart. The control forcing acts to accelerate or decelerate cart. For this example, we assume the following parameter values: pendulum mass ($m = 1$), cart mass ($M = 5$), pendulum length ($L = 2$), gravitational acceleration ($g = -10$), and cart damping ($\delta = 1$).

Homework

Exercise 9–1: LQR Design. We will explore optimal control on the inverted pendulum on a cart, shown in Fig. 9.8.

The full nonlinear dynamics are given by

$$\dot{x} = v \tag{9.80a}$$

$$\dot{v} = \frac{-m^2L^2g\cos(\theta)\sin(\theta) + mL^2(mL\omega^2\sin(\theta) - \delta v) + mL^2u}{mL^2(M + m(1 - \cos(\theta)^2))} \tag{9.80b}$$

$$\dot{\theta} = \omega \tag{9.80c}$$

$$\dot{\omega} = \frac{(m+M)mgL\sin(\theta) - mL\cos(\theta)(mL\omega^2\sin(\theta) - \delta v) - mL\cos(\theta)u}{mL^2(M + m(1 - \cos(\theta)^2))} \tag{9.80d}$$

where x is the cart position, v is the velocity, θ is the pendulum angle, ω is the angular velocity, m is the pendulum mass, M is the cart mass, L is the pendulum arm, g is the gravitational acceleration, δ is a friction damping on the dart, and u is a control force applied to the cart.

There are two fixed points, corresponding to either the pendulum down ($\theta = 0$) or pendulum up ($\theta = \pi$) configuration; in both cases, $v = \omega = 0$ for the fixed point, and the cart position x is a free variable, as the equations do not depend explicitly on x.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(a) Linearize the equations in (9.80) about the up and the down solutions to obtain $\mathbf{A}$ and $\mathbf{B}$ matrices.

(b) Confirm that the system is linearly controllable with the given $\mathbf{A}$ and $\mathbf{B}$ matrices. This is accomplished by computing the rank of the controllability matrix using `numpy.linalg.matrix_rank(ctrb(A,B))` in Python. The matrix should be full rank, which in this case is 4.

(c) Design an LQR controller to stabilize the inverted pendulum configuration ($\theta = \pi$) assuming full-state measurements, $\mathbf{y} = \mathbf{x}$. Use gain matrices $\mathbf{Q} = \mathbf{I}$ and $R = 0.0001$. What is the LQR controller gain matrix $\mathbf{K}$, and what are the closed loop eigenvalues?

K = lqr(A,B,Q,R)[0]

(d) Plot the response of the closed loop system, starting slightly off equilibrium, at $\mathbf{x}_0 = \begin{bmatrix} -1 & 0 & \pi + .1 & 0 \end{bmatrix}^T$.

(e) To visually see the optimality of the LQR controller, we will now simulate the closed-loop system response obtained by non-optimal eigenvalue placement. Compute the system response and cost function for 100 randomly generated sets of stable eigenvalues, chosen in the interval $[-3.5, -.5]$; plot these in light gray. Plot the LQR solution on top of this in dark black. You can compute the $\mathbf{K}$ matrices for the random eigenvalue placement using the `place` command:

```
K = place(A,B,neweigs)
```

Exercise 9–2: Kalman Filter Design. We now revisit the pendulum on a cart, given by Eq. (9.80), but assuming that we only have partial measurements and must estimate the state with a Kalman filter. For this example, we will develop the Kalman filter for the pendulum-down condition ($\theta = 0$), since without feedback the system in the pendulum-up condition will quickly leave the fixed point where the linear model is valid. We will use the cart position as the sensor, $y = x_1$, which corresponds to an observation matrix $\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$.

(a) Confirm that the system is linearly observable with the given $\mathbf{A}$ and $\mathbf{C}$ matrices. This is accomplished by computing the rank of the observability matrix using `numpy.linalg.matrix_rank(obsv(A,C))` in Python. The matrix should be full rank, which in this case is 4.

(b) Now design a Kalman filter with disturbance and noise covariances given by $\mathbf{V}_d = \mathbf{I}$ and $V_n = 1$. What is the Kalman filter gain matrix $\mathbf{K}_f$, and what are the eigenvalues of the estimator?

(c) Simulate the augmented system, with Kalman filter, with noise disturbance.

CONTRIBUTE CODE TO AUGMENT AND SIMULATE

(d) Now modify the $\mathbf{C}$ matrix to include some imperfection in the measurement, but don't update the Kalman filter gain. How does this change the estimate?

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Exercise 9–3: LQG Design. Combine the Kalman filter and LQR controller from the exercises above, and simulate the full system response with noise and disturbances.

For this example, we use the following LQR and LQE weighting matrices: $\mathbf{Q} = \mathbf{I}_{4\times4}, R = 0.000001, \mathbf{V}_d = 0.04\,\mathbf{I}_{4\times4}, V_n = 0.0002$.

Start the system near the vertical equilibrium, at $\mathbf{x}_0 = \begin{bmatrix}0 & 0 & 3.14 & 0\end{bmatrix}^T$, and command a step in the cart position from $x = 0$ to $x = 1$ at $t = 10$.

Compare the system response in both the linearized system and the full nonlinear system.

Exercise 9–4: This example will design an optimal controller for a rotary inverted pendulum.

(a) Derive the equations of motion for a rotary pendulum, where the base of the pendulum is mounted to a rotating arm. The control input is a torque to the rotor arm.

(b) Identify the fixed points of the system and linearize about each point. What is the stability of each fixed point? Are they linear controllable?

(c) Design an LQR controller for the pendulum up configuration assuming full state measurements.

(d) Determine the observability of the pendulum up configuration if we cannot measure the full state, but instead measure the pendulum angle and the rotor angle. Similarly, determine the observability if we only measure the pendulum angular rate and the rotor angle. Which sensor configuration is *more* observable? Pick at least one different sensor set and assess if this configuration is observable.

(e) Assuming a measurement of the pendulum angle and rotor angle, design a Kalman filter to estimate the full state. Design for disturbance magnitude 1.e-3 and sensor noise 1.e-2. Simulate the noisy system and compare the Kalman filter estimate with the true state without added noise.

(f) Design an LQG controller for the pendulum up configuration and demonstrate this controller in simulation. How does the controller perform when you introduce a small time delay? At what point does the time delay cause the system to go unstable?

Design a full-state LQR controller to track a reference position of the cart, while minimizing the pendulum swinging. Try different LQR gain matrices, and compare the step response when setting the reference state from $x = 0$ to $x = 1$.

Exercise 9–5: Derive the equations of motion for a double pendulum on a cart.

Repeat each step above for the double pendulum.

Exercise 9–6: This exercise will design a controller to move a cart with a pendulum in the down position from one point on a track to another. The goal will be to move quickly from point $x = 0$ to point $x = 1$ while minimizing the amount the pendulum swings, which

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

corresponds to the problem faced when designing a controller for a gantry crane. What will make this example more interesting is that it is a *non-minimum-phase* system.

Design a full-state LQR controller to track a reference position of the cart, while minimizing the pendulum swinging. Try different LQR gain matrices, and compare the step response when setting the reference state from $x = 0$ to $x = 1$.

Exercise 9–7: This example will explore how an ill-conditioned controllability Gramian can affect state-feedback control.

(a) Create a continuous-time, single-input state space system that has $n = 2$ states full-state measurements. Design the system to be technically controllable, yet with one of the directions being much more controllable than the other (i.e., 10^6 times more controllable). Compute the controllability Gramian for this system and compute its eigendecomposition. Explain what you find.

(b) Design an LQR controller for this system using weight matrices $\mathbf{Q} = \mathbf{I}$ and $R = 1$. Simulate the response of the closed loop system, and explain the results.

(c) Using this controller, initialize the system at thousands of points randomly sampled from a unit circle where $\|\mathbf{x}\| = 1$, and compute the LQR cost J for each of these trajectories. Plot the initial conditions on the sphere, color coded by the cost J. Reconcile this plot with the controllability Gramian you computed earlier.

Exercise 9–8: Extremum Seeking Control. The field of *adaptive* control allows the control law the flexibility to modify its action based on the changing dynamics of a system, without needing a detailed system model. Extremum-seeking control (ESC) [209, 10] is a particularly attractive form of adaptive control for complex systems because it does not rely on an underlying model and it has guaranteed convergence and stability under a set of well-defined conditions. Extremum-seeking may be used to track local maxima of an objective function, despite disturbances, varying system parameters, and nonlinearities. Adaptive control may be implemented for in-time control or used for slow tuning of parameters in a working controller.

Three distinct time-scales are relevant for extremum-seeking control:

(a) slow – external disturbances and parameter variation;

(b) medium – perturbation frequency ω;

(c) fast – system dynamics.

In many systems, the internal system dynamics evolve on a fast time-scale. For example, turbulent fluctuations may equilibrate rapidly compared to actuation time-scales.

In general, extremum-seeking control may be applied to systems with nonlinear dynamics relating the input u to the outputs $\mathbf{y}$ that act on a faster timescale than the perturbation ω. Thus, J may be time-varying, which complicates the simplistic averaging analysis above. The general case of extremum-seeking control of nonlinear systems is analyzed by Krstić and Wang in [209], where they develop powerful stability guarantees based on

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

a separation of timescales and a singular perturbation analysis. The basic algorithm may also be modified to add a phase ϕ to the sinusoidal input perturbation in (6.21). In [209], there was an additional low-pass filter $\omega_l/(s+\omega_l)$ placed before the integrator to extract the DC component of the demodulated signal ξ.

To see the ability of extremum-seeking control to handle varying system parameters, consider the time-dependent cost function given by

$$J(u) = 25 - (5 - u - \sin(t))^2. \tag{9.81}$$

The varying parameters, which oscillate at $1/2\pi$ Hz, may be considered slow compared with the perturbation frequency 10 Hz.

Design an extremum-seeking control for this slowly varying system. Plot the response and discuss how the performance varies with the various ESC parameters.

Exercise 9–9: Here we consider an example inspired by a challenging benchmark problem in Sec. 1.3 of [10]. This system has a time-varying objective function $J(t)$ and dynamics with a right-half plane zero, making it difficult to control.

In one formulation of extremum-seeking [209, 10], there are additional guidelines for designing the controller if the plant can be split into three blocks that define the input dynamics, a time-varying objective function with no internal dynamics, and the output dynamics. In this case, there are procedures to design the high-pass filter and integrator blocks.

In this example, the objective function is given by

$$J(\theta) = .05\delta(t-10) + (\theta - \theta^*(t))^2,$$

where δ is the Dirac delta function, and the optimal value $\theta^*(t)$ is given by

$$\theta^* = .01 + .001t.$$

The optimal objective is given by $J^* = .05\delta(t-10)$. The input and output dynamics are taken from the example in [10], and are given by

$$F_{\text{in}}(s) = \frac{s-1}{(s+2)(s+1)}, \qquad F_{\text{out}}(s) = \frac{1}{s+1}.$$

Using the design procedure in [10], one arrives at the high-pass filter $s/(s+5)$ and an integrator-like block given by $50(s-4)/(s-.01)$. In addition, a perturbation with $\omega = 5$ and $a = .05$ is used, and the demodulating perturbation is phase-shifted by $\phi = .7955$; this phase is obtained by evaluating the input function F_{in} at $i\omega$.

Implement this extremum seeking controller and plot the response. Discuss the performance and experiment with different levels of additive sensor noise.

Exercise 9–10: Consider the following 2D linear system:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

with

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

In this problem, we will develop the LQR optimal state-feedback controller $\mathbf{u} = -\mathbf{K}\mathbf{x}$ using several different approaches. In the next exercise we will solve the same problem using LMIs. For the remainder, we will use the following LQR gain matrices:

$$\mathbf{Q} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad R = 1.$$

(a) First, we will explicitly solve the algebraic Riccati equation, which is necessary to derive $\mathbf{K}$. The ARE is given by:

$$\mathbf{A}^\top \mathbf{P} \; + \; \mathbf{P}\,\mathbf{A} \; - \; \mathbf{P}\,\mathbf{B}\,R^{-1}\mathbf{B}^\top \mathbf{P} \; + \; \mathbf{Q} \; = \; 0.$$

For the given $\mathbf{A}, \mathbf{B}, \mathbf{Q}$ and R, this becomes:

$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \mathbf{P} \; + \; \mathbf{P} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \; - \; \mathbf{P} \begin{pmatrix} 0 \\ 1 \end{pmatrix} 1 \begin{pmatrix} 0 & 1 \end{pmatrix} \mathbf{P} \; + \; \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = 0.$$

Solve for $\mathbf{P}$ explicitly by setting $\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{pmatrix}$). This yields a small system of equations in p_{11}, p_{12}, p_{22}.

(b) Then compute $\mathbf{K} = R^{-1}\mathbf{B}^\top \mathbf{P} = \mathbf{B}^\top \mathbf{P}$ (since $R = 1$).

(c) What are the closed loop eigenvalues of $\mathbf{A} - \mathbf{B}\mathbf{K}$? What is the resulting cost?

(d) Now compute $\mathbf{K}$ using the `lqr` command, and compare.

Exercise 9–11: We will now solve the problem above using LMIs. We want to find a *feedback* $\mathbf{u} = -\mathbf{K}\mathbf{x}$ that minimizes the LQR integral. A known LMI-based approach is to search for $\mathbf{P} \succ 0$ and $\mathbf{K}$ such that the following *closed-loop Lyapunov* plus *cost term* is negative definite:

$$\text{(LMI condition)} \quad (\mathbf{A} - \mathbf{B}\mathbf{K})^\top \mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{B}\mathbf{K}) \; + \; \mathbf{Q} \; + \; \mathbf{K}^\top R\,\mathbf{K} \; < \; 0.$$

Additionally, we impose $\mathbf{P} > 0$ to ensure a valid Lyapunov function. Then one can show that among all such $(\mathbf{P}, \mathbf{K})$ feasible to this inequality, the $\mathrm{trace}(\mathbf{P})$ (or some other linear function in $\mathbf{P}, \mathbf{K}$) can serve as a surrogate for the LQR cost. Minimizing it enforces the same solution as the Riccati-based approach.

It is now possible to replicate the LQR minimization) via

$$\min \; \mathrm{trace}(\mathbf{P}) \quad \text{subject to the LMI above.}$$

In many contexts, $\mathrm{trace}(\mathbf{P})$ is a suitable stand-in for the LQR cost measure.

Any solution that *minimizes* $\mathrm{trace}(\mathbf{P})$ under the LMI constraints above will match the $\mathbf{P}$ from the standard LQR. The feedback $\mathbf{K} = (R^{-1}\mathbf{B}^\top)\mathbf{P}$ is likewise identical to the LQR gain.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(a) Solve this LMI by hand. Let $\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{pmatrix}$, $\mathbf{K} = (k_1, \ k_2)$. Write out the 2×2 matrix

$$(\mathbf{A} - \mathbf{BK})^\top \mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{BK}) + \mathbf{Q} + \mathbf{K}^\top R\mathbf{K}$$

and enforce < 0 (strict negative definiteness), plus $\mathbf{P} > 0$. See if you can solve for p_{ij}, k_i that satisfy it. (It's basically rewriting the usual Riccati equation in an LMI format.)

(b) Python `cvxpy` Approach. Use a code like:

```
import cvxpy as cp
import numpy as np

A = np.array([[0,1],[0,0]])
B = np.array([[0],[1]])
Q = np.eye(2)
R = 1.0

P = cp.Variable((2,2), PSD=True)    # P>0
K = cp.Variable((1,2))              # K in R^{1x2}

# Build the LMI:
M = (A - B@K).T@P + P@(A - B@K) + Q + K.T*(R)*K  # shape 2x2

constraints = [ P >> 0,
                0 > M ] # "M < 0" is '0 > M' in cvxpy
obj = cp.Minimize(cp.trace(P))

prob = cp.Problem(obj, constraints)
prob.solve(solver=cp.MOSEK) # or another solver

print("P =", P.value)
print("K =", K.value)
```

Compare the resulting $\mathbf{K}$ and $\mathbf{P}$ with the classical Riccati equation solution.

Exercise 9–12: We can also use LMIs to solve the simpler problem of ensuring stability (i.e. $\mathbf{A} - \mathbf{BK}$ is Hurwitz) by guaranteeing the existence of $\mathbf{P} > 0$ such that

$$(\mathbf{A} - \mathbf{B}\,\mathbf{K})^\top \mathbf{P} \ + \ \mathbf{P}\,(\mathbf{A} - \mathbf{B}\,\mathbf{K}) \ < \ 0, \quad \mathbf{P} > 0.$$

The optimization problem is then

$$\text{Find } (\mathbf{P}, \mathbf{K}) \text{ s.t.} \quad \begin{cases} \mathbf{P} > 0, \\ (\mathbf{A} - \mathbf{BK})^\top \mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{BK}) < 0. \end{cases}$$

(a) Rewrite the second inequality in a standard LMI block form if you wish (some references do a block matrix with extra variables). But in 2D, you can just write $\mathbf{M} = (\mathbf{A} - \mathbf{BK})^\top \mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{BK})$ out explicitly and ensure $\mathbf{M}$ is negative definite.

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(b) Manual approach:

- Let $\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{pmatrix} > 0$, $\mathbf{K} = (k_1,\ k_2)$.
- Then $\mathbf{A} - \mathbf{BK} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} (k_1 \ \ k_2) = \begin{pmatrix} 0 & 1 \\ -k_1 & -k_2 \end{pmatrix}$.
- Evaluate $(\mathbf{A}-\mathbf{BK})^\top \mathbf{P} + \mathbf{P}(\mathbf{A}-\mathbf{BK})$ and impose negativity. Solve for unknowns p_{ij}, k_j.

(c) Compare solutions $(\mathbf{P}, \mathbf{K})$ with the ones you got from the Riccati approach. They should coincide, or be at least consistent.

(d) The following Python `cvxpy` code defines the variables and constraints, initializes the optimization problem, and solves it.

```
P = cp.Variable((2,2), symmetric=True)} and \texttt{K = cp.
    Variable((1,2))

constraints = [P >> 0,
               (A - B@K).T @ P + P@(A - B@K) << 0]

prob = cp.Problem(cp.Minimize(0), constraints)
prob.solve()
```

If you only want feasibility, you can do `Objective = Minimize(0)`. Or if you want to incorporate a cost on `trace(P)`, you can do `Minimize(cp.trace(P))`, etc. You should get `P.value` and `K.value` that hopefully match (or are close to) your by-hand solution.

Exercise 9–13: (Reinforcement learning vs. LQR).

(a) Implement a basic RL algorithm (Q-learning or policy gradient) on the same linear system from Exercise 9-1 to try to stabilize $x = 0$.

(b) Compare the learned control law with the LQR solution.

(c) Discuss sample efficiency (how many episodes or interactions are needed).

Exercise 9–14: (Model predictive control). Use a time-discretized double integrator system $x_{k+1} = x_k + \Delta t\, v_k$ and $v_{k+1} = v_k + \Delta t\, u_k$ to formulate an MPC that moves x from 0 to 10 while keeping v under some limit.

(a) Write the MPC cost and constraints.

(b) Implement an MPC for horizon $N = 10$.

(c) Compare the closed-loop trajectory for different horizons $N = 5, 10, 20$.

Exercise 9–15: Nonlinear MPC for a pendulum swing-up. Consider $\dot\theta = \omega,\ \dot\omega = -\frac{g}{L}\sin(\theta) + u$.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(a) Linearize about $(\theta, \omega) = (\pi, 0)$ (upright) and see if it's controllable.

(b) Set up a direct-collocation-based MPC that tries to swing the pendulum from $(0, 0)$ to $(\pi, 0)$ over finite time.

(c) Compare performance with a simpler LQR that's only valid near the upright.

Exercise 9–16: Show that the LQR optimal control solution satisfies the HJB equations. First, write down the value function V for the LQR problem. Show that this function satisfies HJB. Derive an expression for the Lagrange multiplier co-state $\boldsymbol{\lambda}$ and explain.

Exercise 9–17: Repeat the steps in the derivation of the HJB equations more carefully using the calculus of variations.

Exercise 9–18: In this exercise, we will show that the Kalman filter is performing a Bayesian update, given prior information in the form of a system model, with a likelihood function derived from a new noisy measurement.

First we consider a discrete-time linear Gaussian system model with process and measurement noise:

$$\mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{w}_k, \qquad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k), \tag{9.82a}$$

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k, \qquad \mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k). \tag{9.82b}$$

Here, $\mathbf{x}_k \in \mathbb{R}^n$ is the hidden system state, $\mathbf{y}_k \in \mathbb{R}^m$ is the observed measurement, and $\mathbf{A}_k$, $\mathbf{H}_k$ are the system and observation matrices, respectively.

The Kalman filter recursively estimates the posterior distribution $p(\mathbf{x}_k \mid \mathbf{y}_{0:k})$ as a Gaussian distribution with mean $\hat{\mathbf{x}}_{k|k}$ and covariance $\mathbf{P}_{k|k}$, using the following two steps:

(a) **Prediction step (prior):**

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}_{k-1} \hat{\mathbf{x}}_{k-1|k-1}, \tag{9.83a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{A}_{k-1} \mathbf{P}_{k-1|k-1} \mathbf{A}_{k-1}^\top + \mathbf{Q}_{k-1}. \tag{9.83b}$$

(b) **Update step (posterior):**

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \left(\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k \right)^{-1}, \tag{9.84a}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{y}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \right), \tag{9.84b}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}. \tag{9.84c}$$

Show that the Kalman filter equations above arise from performing Bayesian estimation of the state $\mathbf{x}_k$ given a Gaussian prior $p(\mathbf{x}_k \mid \mathbf{y}_{0:k-1}) = \mathcal{N}(\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1})$ and a Gaussian likelihood $p(\mathbf{y}_k \mid \mathbf{x}_k) = \mathcal{N}(\mathbf{H}_k \mathbf{x}_k, \mathbf{R}_k)$.

Specifically:

(a) Derive the posterior distribution $p(\mathbf{x}_k \mid \mathbf{y}_{0:k})$ using Bayes' rule, and show that it is Gaussian.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

(b) Compute the mean and covariance of the posterior, and verify that they match the Kalman filter update equations for $\hat{\mathbf{x}}_{k|k}$ and $\mathbf{P}_{k|k}$.

(c) Briefly explain how this recursive update embodies Bayesian reasoning at each time step.

Exercise 9–19: Recall the two LMIs examples at the beginning of Sec. 9.6:

- $\mathbf{A}^T\mathbf{X}\mathbf{A} - \mathbf{X} < \mathbf{Q}$, where $\mathbf{X} \in \mathbb{S}^n$ is a symmetric matrix.
- $\mathbf{A}^T\mathbf{X}\mathbf{A} + \mathbf{B}\mathbf{Y} + \mathbf{Y}^T\mathbf{B}^T + \mathbf{T} < 0$, which is linear in $\mathbf{Z} = (\mathbf{X}, \mathbf{Y})$.

For each of these LMIs, show that the function $\mathbf{F}(\mathbf{X})$ is linear in $\mathbf{X}$ and maps a symmetric $\mathbf{X}$ to a self-adjoint matrix $\mathbf{F}(\mathbf{X})$.

Exercise 9–20: Show that the two forms of an LMI from Eq. (9.64) and Eq. (9.65) are equivalent. Specifically, show that each LMI can be transformed from one form into the other.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Chapter 10

Optimization in Machine Learning

[STILL WRITING]

10.0.1 Automatic Differentiation

10.0.2 Backpropagation

10.0.3 Neural Network Training

10.0.4 Deep Reinforcement Learning

10.0.5 Physics Constrained Machine Learning

DRAFT

Suggested Reading

Texts

(1) Deep learning, by I. Goodfellow, Y. Bengio, and A. Courville, 2016 [155].

(2) Machine learning: A probabilistic perspective, by K. P. Murphy, 2012 [257].

(3) Pattern recognition and machine learning, by C. M. Bishop, 2006 [39].

(4) Pattern classification, by R. O. Duda, P. E. Hart, and D. G. Stork, 2000 [115].

(5) Learning with kernels: Support vector machines, regularization, optimization, and beyond, by B. Schölkopf and A. J. Smola, 2002 [314].

Papers and reviews

(1) Deep learning, by Y. LeCun, Y. Bengio, and G. Hinton, *Nature*, 2015 [218].

(2) Understanding deep convolutional networks, by S. Mallat, *Philosophical Transactions of the Royal Society of London A*, 2016 [235].

(3) Constrained Sparse Galerkin Regression, by J. C. Loiseau and S. L. Brunton, *Journal of Fluid Mechanics*, 2018 [230].

(4) A unified framework to enforce, discover, and promote symmetry in machine learning, by S. E. Otto, N. Zolman, J. N. Kutz, and S. L. Brunton, *arXiv*, 2023 [271].

(5) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, by M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Journal of Computational Physics*, 2019 [286].

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

Bibliography

[1] R. Abraham, J. E. Marsden, and J. E. Marsden. *Foundations of mechanics*, volume 36. Benjamin/Cummings Publishing Company Reading, Massachusetts, 1978.

[2] F. Acker and M.-A. Prestel. Convergence d'un schéma de minimisation alternée. In *Annales de la Faculté des sciences de Toulouse: Mathématiques*, volume 2, pages 1–9. Université Paul Sabatier, 1980.

[3] A. Adler, Y. Golinval, and A. Maggi. Deep learning in seismic inverse problems: Fundamentals and applications. *Journal of Applied Geophysics*, 186:104267, 2021.

[4] H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.

[5] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control. *arXiv preprint arXiv:1810.13400*, 2018.

[6] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, et al. *LAPACK Users' guide*, volume 9. Siam, 1999.

[7] J. L. Anderson. An ensemble adjustment Kalman filter for data assimilation. *Monthly weather review*, 129(12):2884–2903, 2001.

[8] G. Andrew and J. Gao. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40, 2007.

[9] A. C. Antoulas. *Approximation of large-scale dynamical systems*. SIAM, 2005.

[10] K. B. Ariyur and M. Krstić. *Real-Time Optimization by Extremum-Seeking Control*. Wiley, Hoboken, New Jersey, 2003.

[11] E. Artin. *Galois Theory: Lectures Delivered at the University of Notre Dame by Emil Artin (Notre Dame Mathematical Lectures*. Courier Corporation, 2012.

[12] T. Askham and J. N. Kutz. Variable projection methods for an optimized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 17(1):380–416, 2018.

[13] K. J. Aström and R. M. Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[14] S. D. Babacan, R. Molina, and A. K. Katsaggelos. Bayesian compressive sensing using laplace priors. *IEEE Transactions on image processing*, 19(1):53–63, 2009.

[15] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[16] P. J. Baddoo, B. Herrmann, B. J. McKeon, and S. L. Brunton. Kernel learning for robust dynamic mode decomposition: Linear and nonlinear disambiguation optimization (lando). *Proceedings of the Royal Society A*, 478(2260):20210830, 2022.

[17] P. J. Baddoo, B. Herrmann, B. J. McKeon, J. Nathan Kutz, and S. L. Brunton. Physics-informed dynamic mode decomposition. *Proceedings of the Royal Society A*, 479(2271):20220576, 2023.

[18] R. Bai, V. Ročková, and E. I. George. Spike-and-slab meets lasso: A review of the spike-and-slab lasso. *Handbook of Bayesian variable selection*, pages 81–108, 2021.

[19] J. Bakarji, K. Champion, J. Nathan Kutz, and S. L. Brunton. Discovering governing equations from partial measurements with deep delay autoencoders. *Proceedings of the Royal Society A*, 479(2276):20230422, 2023.

[20] R. G. Baraniuk. Compressive sensing. *IEEE Sig. Proc. Mag.*, 24(4):118–120, 2007.

[21] S. Barman, X. Liu, S. Draper, and B. Recht. Decomposition methods for large scale lp decoding. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 2552–2556. IEEE, 2011.

DRAFT

[22] M. S. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, 22(1):107–111, 1951.

[23] H. H. Bauschke, P. L. Combettes, and D. Noll. Joint minimization with alternating bregman proximity operators. *Pacific Journal of Optimization*, 2(3):401–424, 2006.

[24] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43, 2018.

[25] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[26] R. Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.

[27] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[28] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace. Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations. *IEEE Transactions on Automatic Control*, 56(12):2883–2897, 2011.

[29] M. P. Bendsoe and O. Sigmund. Topology optimization: Theory, methods, and applications. *Springer*, 2009.

[30] P. Benner, J.-R. Li, and T. Penzl. Numerical solution of large-scale Lyapunov equations, Riccati equations, and linear-quadratic optimal control problems. *Numerical Linear Algebra with Applications*, 15(9):755–777, 2008.

[31] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. Slicot—a subroutine library in systems and control theory. In *Applied and computational control, signals, and circuits*, pages 499–539. Springer, 1999.

[32] J. Berkson. Examination of randomness of alpha particle emissions. *Research Methods in Statistics. Wiley, New York*, 1966.

[33] J. M. Bernardo and A. F. Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

[34] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[35] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[36] D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[37] K. Bieker, S. Peitz, S. L. Brunton, J. N. Kutz, and M. Dellnitz. Deep model predictive flow control with limited sensor data and online learning. *Theoretical and Computational Fluid Dynamics*, pages 1–15, 2020.

[38] S. A. Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[39] C. M. Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[40] A. M. Bloch, P. Krishnaprasad, J. E. Marsden, and R. M. Murray. Nonholonomic mechanical systems with symmetry. *Archive for Rational Mechanics and Analysis*, 136(1):21–99, 1996.

[41] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[42] V. Boltyanski, R. Gamkrelidze, E. Mishchenko, and L. Pontryagin. The maximum principle in the theory of optimal processes of control. *IFAC Proceedings Volumes*, 1(1):464–469, 1960.

[43] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[44] R. I. Bot. *Conjugate duality in convex optimization*, volume 637. Springer Science & Business Media, 2009.

[45] G. E. Box and G. C. Tiao. *Bayesian inference in statistical analysis*. John Wiley & Sons, 2011.

[46] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*, volume 15. Society for Industrial and Applied Mathematics, 1994.

[47] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[48] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2009.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[49] W. H. Bragg and W. L. Bragg. The reflection of x-rays by crystals. *Proceedings of the Royal Society A*, 88:428–438, 1913.

[50] L. Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373–384, 1995.

[51] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[52] C. G. Broyden. Quasi-newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

[53] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[54] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz. Chaos as an intermittently forced linear system. *Nature Communications*, 8(19):1–9, 2017.

[55] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz. Modern Koopman theory for dynamical systems. *SIAM Review*, 64(2):229–340, 2022.

[56] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2nd edition, 2022.

[57] S. L. Brunton and J. N. Kutz. Promising directions of machine learning for partial differential equations. *Nature Computational Science*, 4(7):483–494, 2024.

[58] S. L. Brunton, J. Nathan Kutz, K. Manohar, A. Y. Aravkin, K. Morgansen, J. Klemisch, N. Goebel, J. Buttrick, J. Poskin, A. W. Blom-Schieber, T. Hogan, and D. McDonald. Data-driven aerospace engineering: reframing the industry with machine learning. *AIAA Journal*, pages 1–26, 2021.

[59] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[60] S. L. Brunton, N. Zolman, J. N. Kutz, and U. Fasel. Machine learning for sparse nonlinear modeling and control. *Annual Review of Control, Robotics, and Autonomous Systems*, 8, 2025.

[61] P. A. Businger and G. H. Golub. Algorithm 358: singular value decomposition of a complex matrix [f1, 4, 5]. *Communications of the ACM*, 12(10):564–565, 1969.

[62] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[63] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1):129–156, 1994.

[64] X. Cai, M. J. Tippett, L. Xie, and J. Bao. Fast distributed mpc based on active set method. *Computers & chemical engineering*, 71:158–170, 2014.

[65] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[66] E. J. Candès. Compressive sensing. *Proceedings of the International Congress of Mathematics*, 2006.

[67] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM*, 58(3):1–37, 2011.

[68] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[69] E. J. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications in Pure and Applied Mathematics*, 8(1207–1223), 59.

[70] G. Cardano and C. Spon. Ars magna (1545). *Opera Omnia*, 4:221–302, 1968.

[71] C. M. Carvalho, N. G. Polson, and J. G. Scott. The horseshoe estimator for sparse signals. *Biometrika*, pages 465–480, 2010.

[72] K. P. Champion, S. L. Brunton, and J. N. Kutz. Discovery of nonlinear multiscale systems: Sampling strategies and embeddings. *SIAM Journal on Applied Dynamical Systems*, 18(1):312–333, 2019.

[73] T. F. Chan and S. Esedoglu. Aspects of total variation regularized l 1 function approximation. *SIAM Journal on Applied Mathematics*, 65(5):1817–1837, 2005.

[74] H. Chang, D.-Y. Yeung, and Y. Xiong. Super-resolution through neighbor embedding. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2004.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[75] R. Chartrand. Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, 2011, 2011.

[76] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

[77] W. Cheney and A. A. Goldstein. Proximity maps for convex sets. *Proceedings of the American Mathematical Society*, 10(3):448–450, 1959.

[78] F. R. K. Chung. *Spectral Graph Theory*, volume 92 of *CBMS Regional Conference Series in Mathematics*. American Mathematical Society, Providence, RI, 1997.

[79] J. F. Claerbout and F. Muir. Robust modeling with erratic data. *Geophysics*, 38(5):826–844, 1973.

[80] T. E. H. T. Collaboration. First m87 event horizon telescope results. i. the shadow of the supermassive black hole. *The Astrophysical Journal Letters*, 875:L1, 2019.

[81] A. R. Conn. Constrained optimization using a nondifferentiable penalty function. *SIAM Journal on Numerical Analysis*, 10(4):760–784, 1973.

[82] A. R. Conn, N. I. Gould, and P. L. Toint. *Trust Region Methods*. MPS/SIAM Series on Optimization. SIAM, 2000.

[83] A. Cormack. Representation of a function by its line integrals, with some radiological applications. *Journal of Applied Physics*, 34:2722–2727, 1963.

[84] M. Cranmer. Interpretable machine learning for science with pysr and symbolicregression. jl. *arXiv preprint arXiv:2305.01582*, 2023.

[85] M. D. Cranmer, R. Xu, P. Battaglia, and S. Ho. Learning symbolic physics with graph networks. *arXiv preprint arXiv:1909.05862*, 2019.

[86] P. Danaher, P. Wang, and D. M. Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(2):373–397, 2014.

[87] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[88] G. B. Dantzig. Linear programming and extensions. In *Linear programming and extensions*. Princeton university press, 2016.

[89] M. Dashti and A. M. Stuart. The bayesian approach to inverse problems. *arXiv preprint arXiv:1302.6989*, 2013.

[90] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on optimization*, 1(1):1–17, 1991.

[91] L. Davis et al. *Handbook of genetic algorithms*, volume 115. Van Nostrand Reinhold New York, 1991.

[92] N. D. Daw, J. P. O'doherty, P. Dayan, B. Seymour, and R. J. Dolan. Cortical substrates for exploratory decisions in humans. *Nature*, 441(7095):876–879, 2006.

[93] P. Dayan and L. F. Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. Computational Neuroscience Series, 2001.

[94] B. M. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton. Pysindy: a python package for the sparse identification of nonlinear dynamics from data. *Journal of Open Source Software*, 5(49):2104, 2020.

[95] N. Demo, M. Tezzele, and G. Rozza. Pydmd: Python dynamic mode decomposition. *The Journal of Open Source Software*, 3(22):530, 2018.

[96] J. E. Dennis, Jr and J. J. Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

[97] J. E. Dennis Jr and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, 1996.

[98] S. Diamond and S. Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[99] M. Diehl, H. J. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. *Nonlinear model predictive control: towards new challenging applications*, pages 391–417, 2009.

[100] I. I. Dikin. Iterative solution of problems of linear and quadratic programming. In *Soviet Math. Dokl.*,

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

volume 8, pages 674–675, 1967.

[101] U. M. Diwekar. *Introduction to applied optimization*, volume 22. Springer Nature, 2020.

[102] A. Domahidi, J. L. Jerez, N. K. Poulsen, S. Jóhannesson, A. Zanelli, and M. Diehl. Forces: Fast optimization for real-time control on embedded systems. *IFAC Proceedings Volumes*, 45(17):145–150, 2012.

[103] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones. Efficient interior point methods for multistage problems arising in receding horizon control. In *2012 IEEE 51st IEEE conference on decision and control (CDC)*, pages 668–674. IEEE, 2012.

[104] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*, pages 184–199. Springer, 2014.

[105] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[106] D. L. Donoho, I. M. Johnstone, J. C. Hoch, and A. S. Stern. Maximum entropy and the nearly black object. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 41–81, 1992.

[107] D. L. Donoho and J. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[108] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[109] J. Douglas and H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956.

[110] J. C. Doyle. Guaranteed margins for LQG regulators. *IEEE Transactions on Automatic Control*, 23(4):756–757, 1978.

[111] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg. Vision-based high-speed driving with a deep dynamic observer. *IEEE Robotics and Automation Letters*, 4(2):1564–1571, 2019.

[112] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie, and M. Klaučo. Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems. *Journal of Process Control*, 116:80–92, 2022.

[113] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

[114] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[115] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[116] G. E. Dullerud and F. Paganini. *A course in robust control theory: A convex approach*. Texts in Applied Mathematics. Springer, Berlin, Heidelberg, 2000.

[117] D. S. Dummit and R. M. Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

[118] T. Duriez, S. L. Brunton, and B. R. Noack. *Machine Learning Control: Taming Nonlinear Dynamics and Turbulence*. Springer, 2016.

[119] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[120] J. Eckstein. Augmented lagrangian and alternating direction methods for convex optimization: A tutorial and some illustrative computational results. *Rutgers Center for Operations Research (RUTCOR) Research Report*, (RRR 32-2012), 2012.

[121] J. Eckstein and D. P. Bertsekas. On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical programming*, 55:293–318, 1992.

[122] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, revised edition edition, 2010.

[123] E. Even-Dar, Y. Mansour, and P. Bartlett. Learning rates for q-learning. *Journal of machine learning Research*, 5(1), 2003.

[124] D. D. Fan, A.-a. Agha-mohammadi, and E. A. Theodorou. Deep learning tubes for tube mpc. *arXiv preprint arXiv:2002.01587*, 2020.

[125] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit mpc. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 18(8):816–830, 2008.

[126] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[127] A. V. Fiacco and G. P. McCormick. The sequential unconstrained minimization technique for nonlinear programing, a primal-dual method. *Management Science*, 10(2):360–366, 1964.

[128] A. V. Fiacco and G. P. McCormick. *Nonlinear programming: sequential unconstrained minimization techniques*. SIAM, 1990.

[129] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 222(594-604):309–368, 1922.

[130] R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[131] R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2000.

[132] N. Fonzi, S. L. Brunton, and U. Fasel. Data-driven nonlinear aeroelastic models of morphing wings for control. *Proceedings of the Royal Society A*, 476(2239):20200079, 2020.

[133] W. T. Freeman, T. R. Jones, and E. C. Pasztor. Example-based super-resolution. *IEEE Computer graphics and Applications*, 22(2):56–65, 2002.

[134] K. Fukami, K. Fukagata, and K. Taira. Super-resolution reconstruction of turbulent flows with machine learning. *Journal of Fluid Mechanics*, 870:106–120, 2019.

[135] K. Fukami, K. Fukagata, and K. Taira. Machine-learning-based spatio-temporal super resolution reconstruction of turbulent flows. *Journal of Fluid Mechanics*, 909, 2021.

[136] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximations. *Computers and Mathematics with Applications*, 2(1):17–40, 1976.

[137] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[138] H. P. Gavin. The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems. *Department of Civil and Environmental Engineering Duke University August*, 3, 2019.

[139] I. M. Gelfand and S. V. Fomin. *Calculus of Variations*. Prentice-Hall, 1963.

[140] A. C. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

[141] A. C. Gilbert, J. Y. Park, and M. B. Wakin. Sketched SVD: Recovering spectral features from compressive measurements. *ArXiv e-prints*, 2012.

[142] A. C. Gilbert, M. J. Strauss, and J. A. Tropp. A tutorial on fast Fourier sampling. *IEEE Signal Processing Magazine*, pages 57–66, 2008.

[143] W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[144] P. E. Gill, W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[145] P. E. Gill, W. Murray, and M. H. Wright. *Practical optimization*. SIAM, 2019.

[146] R. Glowinski and A. Marrocco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, 9(R2):41–76, 1975.

[147] D. E. Goldberg. *Genetic algorithms*. Pearson Education India, 2006.

[148] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[149] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk. Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623, 2014.

[150] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[151] G. H. Golub, P. C. Hansen, and D. P. O'Leary. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21(1):185–194, 1999.

[152] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerical Mathematics*, 14:403–420, 1970.

[153] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[154] R. E. Gomory. *Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed*

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

integer problem. Springer, 2010.

[155] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[156] C. Gordon, D. Webb, and S. Wolpert. Isospectral plane domains and surfaces via Riemannian orbifolds. *Inventiones mathematicae*, 110(1):1–22, 1992.

[157] M. S. Grewal. Kalman filtering. In *International Encyclopedia of Statistical Science*, pages 705–708. Springer, 2011.

[158] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund. Particle filters for positioning, navigation, and tracking. *IEEE Transactions on signal processing*, 50(2):425–437, 2002.

[159] L. Guttman. Enlargement methods for computing the inverse matrix. *The annals of mathematical statistics*, pages 336–343, 1946.

[160] W. W. Hager. Updating the inverse of a matrix. *SIAM review*, 31(2):221–239, 1989.

[161] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[162] D. Harrison Jr and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[163] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[164] B. He and X. Yuan. On the $o(1/t)$ convergence rate of the douglas-rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

[165] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[166] M. Herceg, C. Jones, and M. Morari. Dominant speed factors of active set methods for fast mpc. *Optimal Control Applications and Methods*, 36(5):608–627, 2015.

[167] K. K. Herrity, A. C. Gilbert, and J. A. Tropp. Sparse approximation via iterative thresholding. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 3, pages III–III. IEEE, 2006.

[168] F. S. Hillier and G. J. Lieberman. *Introduction to operations research*. McGraw-Hill, 2015.

[169] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[170] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. *Optimization with PDE constraints*, volume 23. Springer Science & Business Media, 2008.

[171] B. L. Ho and R. E. Kalman. Effective construction of linear state-variable models from input/output data. In *Proceedings of the 3rd Annual Allerton Conference on Circuit and System Theory*, pages 449–459, 1965.

[172] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[173] M. Hoffmann, M. Scherer, T. Hempel, A. Mardt, B. de Silva, B. E. Husic, S. Klus, H. Wu, N. Kutz, S. L. Brunton, et al. Deeptime: a python library for machine learning dynamical models from time series data. *Machine Learning: Science and Technology*, 3(1):015009, 2021.

[174] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[175] G. Hounsfield. Computerized transverse axial scanning (tomography): Part 1. description of system. *British Journal of Radiology*, 46:1016–1022, 1973.

[176] B. Houska, H. J. Ferreau, and M. Diehl. An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range. *Automatica*, 47(10):2279–2285, 2011.

[177] S. M. Ichinaga, F. Andreuzzi, N. Demo, M. Tezzele, K. Lapo, G. Rozza, S. L. Brunton, and J. N. Kutz. Pydmd: A python package for robust dynamic mode decomposition. *arXiv preprint arXiv:2402.07463*, 2024.

[178] M. A. Iglesias, K. J. Law, and A. M. Stuart. Ensemble kalman methods for inverse problems. *Inverse Problems*, 29(4):045001, 2013.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[179] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*. Springer, 2013.

[180] A. Jameson, L. Martinelli, and N. Pierce. Optimum aerodynamic design using the navier–stokes equations. *Theoretical and computational fluid dynamics*, 10(1-4):213–237, 1998.

[181] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37:183–233, 1999.

[182] M. R. Jovanović, P. J. Schmid, and J. W. Nichols. Sparsity-promoting dynamic mode decomposition. *Physics of Fluids*, 26(2):024103, 2014.

[183] J. N. Juang. *Applied System Identification*. Prentice Hall PTR, Upper Saddle River, New Jersey, 1994.

[184] J. N. Juang and R. S. Pappa. An eigensystem realization algorithm for modal parameter identification and model reduction. *Journal of Guidance, Control, and Dynamics*, 8(5):620–627, 1985.

[185] J. N. Juang, M. Phan, L. G. Horta, and R. W. Longman. Identification of observer/Kalman filter Markov parameters: Theory and experiments. Technical Memorandum 104069, NASA, 1991.

[186] S. J. Julier and J. K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. *Int. symp. aerospace/defense sensing, simul. and controls*, 3(26):182–193, 1997.

[187] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[188] O. Junge, J. E. Marsden, and S. Ober-Blöbaum. Discrete mechanics and optimal control. *IFAC Proceedings Volumes*, 38(1):538–543, 2005.

[189] M. Kac. Can one hear the shape of a drum? *The American Mathematical Monthly*, 73(4):1–23, 1966.

[190] S. Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bulletin International de l'Academie Polonaise des Sciences et des Lettres*, A35:355–357, 1937.

[191] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[192] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society of London A*, 474(2219), 2018.

[193] E. Kaiser, B. R. Noack, L. Cordier, A. Spohn, M. Segond, M. Abel, G. Daviller, J. Östh, S. Krajnović, and R. K. Niven. Cluster-based reduced-order modelling of a mixing layer. *Journal of Fluid Mechanics*, 754:365–414, 2014.

[194] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[195] A. A. Kaptanoglu, J. L. Callaham, C. J. Hansen, A. Aravkin, and S. L. Brunton. Promoting global stability in data-driven models of quadratic nonlinear dynamics. *arXiv preprint arXiv:2105.01843*, 2021.

[196] A. A. Kaptanoglu, B. M. de Silva, U. Fasel, K. Kaheman, J. L. Callaham, C. B. Delahunt, K. Champion, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton. Pysindy: A comprehensive python package for robust sparse system identification. *arXiv preprint arXiv:2111.08481*, 2021.

[197] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[198] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[199] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[200] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[201] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[202] C. Kirches, S. Sager, H. G. Bock, and J. P. Schlöder. Time-optimal control of automobile test drives with gear shifts. *Optimal Control Applications and Methods*, 31(2):137–153, 2010.

[203] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[204] N. Komodakis and J.-C. Pesquet. Playing with duality: An overview of recent primal? dual approaches for solving large-scale optimization problems. *IEEE Signal Processing Magazine*, 32(6):31–54, 2015.

[205] M. Korda and C. N. Jones. Certification of fixed computation time first-order optimization-based con-

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

trollers for a class of nonlinear dynamical systems. In *2014 American Control Conference*, pages 3602–3608. Ieee, 2014.

[206] B. H. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[207] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[208] J. R. Koza, F. H. Bennett III, and O. Stiffelman. Genetic programming as a darwinian invention machine. In *Genetic Programming*, pages 93–108. Springer, 1999.

[209] M. Krstić and H. Wang. Stability of extremum seeking feedback for general nonlinear dynamic systems. *Automatica*, 36(4):595–601, Apr. 2000.

[210] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, 1951.

[211] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[212] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on optimization*, 9(1):112–147, 1998.

[213] J. L. Lagrange. *Mechanique analitique*. La veuve Desaint, 1788.

[214] A. Larrañaga, J. Martínez, J. Míguez, and J. Porteiro. Robust optimization of heat-transfer-enhancing microtextured surfaces based on machine learning surrogate models. *International Communications in Heat and Mass Transfer*, 151:107218, 2024.

[215] J.-B. Lasserre. An explicit exact SDP relaxation for nonlinear 0–1 programs. In *Integer Programming and Combinatorial Optimization*, pages 293–303. Springer, 2001.

[216] A. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Transactions on automatic control*, 24(6):913–921, 1979.

[217] P. C. Lauterbur. Image formation by induced local interactions: Examples employing nuclear magnetic resonance. *Nature*, 242:190–191, 1973.

[218] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[219] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[220] I. Lenz, R. A. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy, 2015.

[221] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[222] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[223] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[224] F. Lin, M. Fardad, and M. R. Jovanovic. Design of optimal sparse feedback gains via the alternating direction method of multipliers. *IEEE Transactions on Automatic Control*, 58(9):2426–2431, 2013.

[225] Z. Lin, M. Chen, and Y. Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010.

[226] D. V. Lindley. *Bayesian statistics: A review*. SIAM, 1972.

[227] P. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

[228] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[229] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, 1999.

[230] J.-C. Loiseau and S. L. Brunton. Constrained sparse Galerkin regression. *Journal of Fluid Mechanics*, 838:42–67, 2018.

[231] B. T. Lopez, J.-J. E. Slotine, and J. P. How. Dynamic tube mpc for nonlinear systems. In *2019 American Control Conference (ACC)*, pages 1655–1662. IEEE, 2019.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[232] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[233] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly. Compressed sensing MRI. *IEEE signal processing magazine*, 25(2):72–82, 2008.

[234] A. M. Lyapunov. The general problem of the stability of motion. *International journal of control*, 55(3):531–534, 1992.

[235] S. Mallat. Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, 374(2065):20150203, 2016.

[236] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*, 2002.

[237] J. Mandel. Use of the singular value decomposition in regression analysis. *The American Statistician*, 36(1):15–24, 1982.

[238] K. Manohar, B. W. Brunton, J. N. Kutz, and S. L. Brunton. Data-driven sparse sensor placement: Demonstrating the benefits of exploiting known patterns. *IEEE Control Systems Magazine*, 38(3):63–86, 2018.

[239] P. Mansfield. Multi-planar image formation using nmr spin echoes. *Journal of Physics C: Solid State Physics*, 10:L55–L58, 1977.

[240] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[241] J. E. Marsden and T. S. Ratiu. *Introduction to mechanics and symmetry*. Springer-Verlag, 2nd edition, 1999.

[242] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[243] A. F. Martins, M. A. Figueiredo, P. M. Aguiar, N. A. Smith, and E. P. Xing. An augmented lagrangian approach to constrained map inference. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

[244] J. L. Maryak, J. C. Spall, and B. D. Heydon. Use of the Kalman filter for inference in state-space models with unknown noise distributions. *IEEE Transactions on Automatic Control*, 49(1):87–90, 2004.

[245] H. Matthies and G. Strang. The solution of nonlinear finite element equations. *International journal for numerical methods in engineering*, 14(11):1613–1626, 1979.

[246] D. Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

[247] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[248] O. Meshi and A. Globerson. An alternating direction method for dual map lp relaxation. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011.

[249] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[250] R. Milman and E. Davison. A fast mpc algorithm using nonfeasible active set methods. *Journal of optimization theory and applications*, 139(3):591–616, 2008.

[251] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[252] P. R. Montague, P. Dayan, and T. J. Sejnowski. A framework for mesencephalic dopamine systems based on predictive hebbian learning. *Journal of neuroscience*, 16(5):1936–1947, 1996.

[253] B. C. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, AC-26(1):17–32, 1981.

[254] J. J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In *Numerical Analysis*, pages 105–116. Springer, 1978.

[255] C. B. Morrey. *Multiple Integrals in the Calculus of Variations*. Springer, 2008.

[256] J. Morton, F. D. Witherden, A. Jameson, and M. J. Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018); arXiv preprint arXiv:1805.07472*, 2018.

[257] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[258] R. M. Neal et al. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[259] A. Nedić and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[260] D. Needell and J. A. Tropp. CoSaMP: iterative signal recovery from incomplete and inaccurate samples. *Communications of the ACM*, 53(12):93–100, 2010.

[261] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[262] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[263] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). *Dokl. Akad. Nauk. SSSR*, 269(3):543, 1983.

[264] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[265] Y. Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

[266] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems (NIPS)*, 14:849–856, 2002.

[267] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[268] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

[269] J. Nocedal and S. J. Wright. Quasi-newton methods. *Numerical optimization*, pages 135–163, 2006.

[270] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Operator splitting for conic optimization via homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[271] S. E. Otto, N. Zolman, J. N. Kutz, and S. L. Brunton. A unified framework to enforce, discover, and promote symmetry in machine learning. *arXiv preprint arXiv:2311.00212*, 2023.

[272] N. Parikh, S. Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.

[273] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NeurIPS workshop*, 2017.

[274] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott. Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. *Physical review letters*, 120(2):024102, 2018.

[275] P. I. Pavlov. *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*. Oxford University Press, 1927.

[276] R. Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge Univ Press, 1955.

[277] R. Penrose and J. A. Todd. On best approximate solutions of linear matrix equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 52, pages 17–19. Cambridge Univ Press, 1956.

[278] T. Pietrzykowski. An exact potential method for constrained maxima. *SIAM Journal on numerical analysis*, 6(2):299–304, 1969.

[279] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[280] F. A. Potra and S. J. Wright. Interior-point methods. *Journal of computational and applied mathematics*, 124(1-2):281–302, 2000.

[281] M. Powell. On the convergence of the dfp algorithm for unconstrained optimization when there are only two variables. *Mathematical programming*, 87:281–301, 2000.

[282] M. J. D. Powell. A hybrid method for nonlinear equations. *Numerical Methods for Nonlinear Algebraic Equations*, 7:87–114, 1970.

[283] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[284] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[285] J. Radon. Über die bestimmung von funktionen durch ihre integralwerte längs gewisser mannigfaltigkeiten. *Berichte Sächsische Akademie der Wissenschaften*, 69:262–277, 1917.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[286] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[287] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of interior-point methods to model predictive control. *Journal of optimization theory and applications*, 99:723–757, 1998.

[288] S. S. Rao. *Engineering optimization: theory and practice*. John Wiley & Sons, 2019.

[289] B. Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019.

[290] R. H. Reichle, D. B. McLaughlin, and D. Entekhabi. Hydrologic data assimilation with the ensemble Kalman filter. *Monthly Weather Review*, 130(1):103–114, 2002.

[291] J. Reuther, A. Jameson, J. Farmer, L. Martinelli, and D. Saunders. *Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation*. Research Institute for Advanced Computer Science, NASA Ames Research Center, 1996.

[292] J. A. Rice. *Mathematical statistics and data analysis*. Duxbury Press, 2 edition, 1994.

[293] A. Richards and J. How. A decentralized algorithm for robust constrained model predictive control. In *Proceedings of the 2004 American control conference*, volume 5, pages 4261–4266. IEEE, 2004.

[294] A. Richards and J. P. How. Robust distributed model predictive control. *International Journal of control*, 80(9):1517–1531, 2007.

[295] S. Richter, C. N. Jones, and M. Morari. Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, 2011.

[296] B. Ristic, S. Arulampalam, and N. J. Gordon. *Beyond the Kalman filter: Particle filters for tracking applications*. Artech house, 2004.

[297] R. T. Rockafellar. *Conjugate duality and optimization*. SIAM, 1974.

[298] R. T. Rockafellar. Lagrange multipliers and optimality. *SIAM review*, 35(2):183–238, 1993.

[299] V. Ročková and E. I. George. The spike-and-slab lasso. *Journal of the American Statistical Association*, 113(521):431–444, 2018.

[300] C. A. Rohde. Generalized inverses of partitioned matrices. *Journal of the Society for Industrial & Applied Mathematics*, 13(4):1033–1035, 1965.

[301] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[302] C. W. Rowley, I. Mezic, S. Bagheri, P. Schlatter, and D. Henningson. Spectral analysis of nonlinear flows. *J. Fluid Mech.*, 645:115–127, 2009.

[303] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(e1602614), 2017.

[304] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[305] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[306] A. Savitzky and M. J. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

[307] H. Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proc. R. Soc. A*, 473(2197):20160446, 2017.

[308] C. Scherer and S. Weiland. Linear matrix inequalities in control. *Lecture Notes, Dutch Institute for Systems and Control, Delft, The Netherlands*, 3(2), 2000.

[309] I. Scherl, B. Strom, J. K. Shang, O. Williams, B. L. Polagye, and S. L. Brunton. Robust principal component analysis for particle image velocimetry. *Physical Review Fluids*, 5(054401), 2020.

[310] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, Aug. 2010.

[311] P. J. Schmid. Dynamic mode decomposition and its variants. *Annual Review of Fluid Mechanics*, 54:225–

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

254, 2022.

[312] E. Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen. i teil. entwicklung willkürlichen funktionen nach system vorgeschriebener. *Math. Ann.*, 3:433–476, 1907.

[313] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[314] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[315] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[316] W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[317] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[318] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[319] J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[320] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[321] S. Skogestad and I. Postlethwaite. *Multivariable feedback control: analysis and design*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2 edition, 2005.

[322] M. Smith, J. Gao, and K. Zhang. Physics-informed deep learning for seismic wave modeling and inversion. *Geophysical Research Letters*, 47(16):e2020GL088744, 2020.

[323] J. C. Spall. The Kantorovich inequality for error analysis of the Kalman filter with unknown noise distributions. *Automatica*, 31(10):1513–1517, 1995.

[324] A. Spantini, A. Solonen, T. Cui, J. Martin, L. Tenorio, and Y. Marzouk. Optimal low-rank approximations of bayesian linear inverse problems. *SIAM Journal on Scientific Computing*, 37(6):A2451–A2487, 2015.

[325] D. A. Spielman. Spectral graph theory and its applications. *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 29–38, 2007.

[326] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[327] R. F. Stengel. *Optimal control and estimation*. Courier Corporation, 2012.

[328] I. Stewart. *Galois theory*. Chapman and Hall/CRC, 2022.

[329] B. Strom, S. L. Brunton, and B. Polagye. Intracycle angular velocity control of cross-flow turbines. *Nature Energy*, 2(17103):1–9, 2017.

[330] A. M. Stuart. Inverse problems: a bayesian perspective. *Acta numerica*, 19:451–559, 2010.

[331] W. Su, M. Bogdan, E. Candes, et al. False discoveries occur early on the lasso path. *The Annals of statistics*, 45(5):2133–2150, 2017.

[332] W. Su, S. Boyd, and E. J. Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. *Journal of Machine Learning Research*, 17(153):1–43, 2016.

[333] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[334] K. Taira, S. L. Brunton, S. Dawson, C. W. Rowley, T. Colonius, B. J. McKeon, O. T. Schmidt, S. Gordeyev, V. Theofilis, and L. S. Ukeiley. Modal analysis of fluid flows: An overview. *AIAA Journal*, 55(12):4013–4041, 2017.

[335] F. Takens. Detecting strange attractors in turbulence. *Lecture Notes in Mathematics*, 898:366–381, 1981.

[336] G. I. Taylor. The formation of a blast wave by a very intense explosion. *Proceedings of the Royal Society A*, 186(1006):273–292, 1941.

[337] G. I. Taylor. The formation of a blast wave by a very intense explosion. ii. the atomic explosion of 1945. *Proceedings of the Royal Society A*, 201(1065):175–186, 1950.

[338] G. Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.

[339] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[340] A. N. Tikhonov et al. On the stability of inverse problems. *Dokl. akad. nauk sssr*, 39(5):195–198, 1943.

[341] J. F. Toland. Duality in nonconvex optimization. *Journal of Mathematical Analysis and Applications*, 66(2):399–415, 1978.

[342] L. N. Trefethen and D. Bau. *Numerical linear algebra*. SIAM, 2022.

[343] J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[344] J. A. Tropp. Recovery of short, complex linear combinations via l_1 minimization. *IEEE Transactions on Information Theory*, 51(4):1568–1570, 2005.

[345] J. A. Tropp. Algorithms for simultaneous sparse approximation. part ii: Convex relaxation. *Signal Processing*, 86(3):589–602, 2006.

[346] J. A. Tropp. Just relax: Convex programming methods for identifying sparse signals in noise. *IEEE Transactions on Information Theory*, 52(3):1030–1051, 2006.

[347] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

[348] J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part i: Greedy pursuit. *Signal Processing*, 86(3):572–588, 2006.

[349] J. A. Tropp, J. N. Laska, M. F. Duarte, J. K. Romberg, and R. G. Baraniuk. Beyond Nyquist: Efficient sampling of sparse bandlimited signals. *IEEE Transactions on Information Theory*, 56(1):520–544, 2010.

[350] J. N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation*. PhD thesis, Massachusetts Institute of Technology, 1984.

[351] J. N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

[352] F. Van Breugel, J. N. Kutz, and B. W. Brunton. Numerical differentiation of noisy data: A unifying multi-objective optimization framework. *IEEE Access*, 8:196865–196877, 2020.

[353] R. van de Schoot, S. Depaoli, R. King, B. Kramer, K. Märtens, M. G. Tadesse, M. Vannucci, A. Gelman, D. Veen, J. Willemsen, et al. Bayesian statistics and modelling. *Nature Reviews Methods Primers*, 1(1):1, 2021.

[354] R. Van Der Merwe. *Sigma-point Kalman filters for probabilistic inference in dynamic state-space models*. Oregon Health & Science University, 2004.

[355] P. J. Van Laarhoven, E. H. Aarts, P. J. van Laarhoven, and E. H. Aarts. *Simulated annealing*. Springer, 1987.

[356] J. Virieux and S. Operto. An overview of full-waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC1–WCC26, 2009.

[357] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proc. R. Soc. A*, 474(2213):20170844, 2018.

[358] B. Völker, V. Maier-Kiener, K. Werbach, T. Müller, S. Pilz, M. Calin, J. Eckert, and A. Hohenwarter. Influence of annealing on microstructure and mechanical properties of ultrafine-grained ti45nb. *Materials & Design*, 179:107864, 2019.

[359] M. von Laue. Interference phenomena with röntgen rays. *Annalen der Physik*, 41:989–1003, 1912.

[360] J. Von Neumann. *Functional Operators, Volume 2: The Geometry of Orthogonal Spaces*, volume 2. Princeton University Press, 1950.

[361] U. Von Toussaint. Bayesian inference in physics. *Reviews of Modern Physics*, 83(3):943–999, 2011.

[362] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[363] N. J. Walkington. Nesterov's method for convex optimization. *SIAM Review*, 65(2):539–562, 2023.

[364] H. Wang, Z. Zhang, and J. Yang. Machine learning applications in geophysics: A review. *Advances in Geophysics*, 63:1–39, 2022.

[365] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy. Esrgan: Enhanced super-

Copyright © July 15, 2025 Brunton. All Rights Reserved.

DRAFT

resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018.

[366] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[367] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.

[368] G. Welch and G. Bishop. An introduction to the kalman filter, 1995.

[369] J. WILLEMS. Least squares stationary optimal control and the algebraic Riccati equation. *IEEE Trans. Automatic Control*, 16:621, 1971.

[370] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[371] D. M. Witten and R. Tibshirani. Penalized classification using fisher's linear discriminant. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(5):753–772, 2011.

[372] L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization*. John Wiley & Sons, 2014.

[373] M. A. Woodbury. The stability of out-input matrices. *Chicago, IL*, 9:3–8, 1949.

[374] J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 31(2):210–227, 2009.

[375] S. J. Wright. *Primal-dual interior-point methods*. SIAM, 1997.

[376] C. Yang, X. Ma, and T. Alkhalifah. Deep-learning-assisted full waveform inversion: Physics-based loss functions and applications. *Geophysics*, 86(3):R363–R377, 2021.

[377] J. Yang, J. Wright, T. S. Huang, and Y. Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010.

[378] S. Yu, S. M. Won, H. W. Baac, D. Son, and C. Shin. Quantitative evaluation of line-edge roughness in various finfet structures: Bayesian neural network with automatic model selection. *IEEE Access*, 10:26340–26346, 2022.

[379] X. Yuan and J. Yang. Sparse and low-rank matrix decomposition via alternating direction methods. *preprint*, 12, 2009.

[380] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

[381] L. Zhang and H. Schaeffer. On the convergence of the SINDy algorithm. *Multiscale Modeling & Simulation*, 17(3), 2019.

[382] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In *IEEE Robotics and Automation (ICRA)*, pages 528–535, 2016.

[383] P. Zheng, T. Askham, S. L. Brunton, J. N. Kutz, and A. Y. Aravkin. Sparse relaxed regularized regression: SR3. *IEEE Access*, 7(1):1404–1423, 2019.

[384] S. Zhou, K. Chan, C. Li, and C. C. Loy. Towards robust blind face restoration with codebook lookup transformer. *Advances in Neural Information Processing Systems*, 35:30599–30611, 2022.

[385] S. Zlobec. An explicit form of the moore-penrose inverse of an arbitrary complex matrix. *SIAM Review*, 12(1):132–134, 1970.

[386] F. Zou, L. Shen, Z. Jie, W. Zhang, and W. Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11127–11135, 2019.

[387] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

DRAFT

Copyright © July 15, 2025 Brunton. All Rights Reserved.