13 Time-Frequency Analysis: Fourier Transforms and Wavelets

Fourier transforms, and more generally, spectral transforms, are one of the most powerful and efficient techniques for solving a wide variety of problems arising the physical, biological and engineering sciences. The key idea of the Fourier transform, for instance, is to represent functions and their derivatives as sums of cosines and sines. This operation can be done with the Fast-Fourier transform (FFT) which is an $O(N \log N)$ operation. Thus the FFT is faster than most linear solvers of $O(N^2)$. The basic properties and implementation of the FFT will be considered here along with other time-frequency analysis techniques, including wavelets.

13.1 Basics of Fourier Series and the Fourier Transform

Fourier introduced the concept of representing a given function by a trigonometric series of sines and cosines:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx) \quad x \in (-\pi, \pi] . \tag{13.1.1}$$

At the time, this was a rather startling assertion, especially as the claim held even if the function $f(x)$, for instance, was discontinuous. Indeed, when working with such series solutions, many questions arise concerning the convergence of the series itself. Another, of course, concerns the determination of the expansion coefficients a_n and b_n. Assume for the moment that Eq. (13.1.1) is a uniformly convergent series, then multiplying by $\cos mx$ gives the following uniformly convergent series

$$f(x) \cos mx = \frac{a_0}{2} \cos mx + \sum_{n=1}^{\infty} (a_n \cos nx \cos mx + b_n \sin nx \cos mx) . \tag{13.1.2}$$

Integrating both sides from $x \in [-\pi, \pi]$ yields the following

$$\int_{-\pi}^{\pi} f(x) \cos mx dx = \frac{a_0}{2} \int_{-\pi}^{\pi} \cos mx dx + \sum_{n=1}^{\infty} \left(a_n \int_{-\pi}^{\pi} \cos nx \cos mx dx + b_n \int_{-\pi}^{\pi} \sin nx \cos mx dx \right) . \tag{13.1.3}$$

But the sine and cosine expressions above are subject to the following *orthogonality* properties.

$$\int_{-\pi}^{\pi} \sin nx \cos mx dx = 0 \quad \forall \ n, m \tag{13.1.4a}$$

$$\int_{-\pi}^{\pi} \cos nx \cos mx dx = \begin{cases} 0 & n \neq m \\ \pi & n = m \end{cases} \tag{13.1.4b}$$

$$\int_{-\pi}^{\pi} \sin nx \sin mx dx = \begin{cases} 0 & n \neq m \\ \pi & n = m \end{cases} . \tag{13.1.4c}$$

Thus we find the following formulas for the coefficients a_n and b_n

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx dx \quad n \geq 0 \tag{13.1.5a}$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin nx dx \quad n > 0 \tag{13.1.5b}$$

which are called the Fourier coefficients.

The expansion (13.1.1) must, by construction, produce 2π-periodic functions since they are constructed from sines and cosines on the interval $x \in (-\pi, \pi]$. In addition to these 2π-periodic solutions, one can consider expanding a function $f(x)$ defined on $x \in (0, \pi]$ as an *even periodic function* on $x \in (-\pi, \pi]$ by employing only the cosine portion of the expansion (13.1.1) or as an *odd periodic function* on $x \in (-\pi, \pi]$ by employing only the sine portion of the expansion (13.1.1).

Using these basic ideas, the complex version of the expansion produces the Fourier series on the domain $x \in [-L, L]$ which is given by

$$f(x) = \sum_{-\infty}^{\infty} c_n e^{in\pi x/L} \quad x \in [-L, L] \tag{13.1.6}$$

with the corresponding Fourier coefficients

$$c_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-in\pi x/L} dx \, . \tag{13.1.7}$$

Although the Fourier series is now complex, the function $f(x)$ is still assumed to be real. Thus the following observations are of note: (i) c_0 is real and $c_{-n} = c_n*$, (ii) if $f(x)$ is even, all c_n are real, and (iii) if $f(x)$ is odd, $c_0 = 0$ and all c_n are purely imaginary. These results follow from Euler's identity: $\exp(\pm ix) = \cos x \pm i \sin x$. The convergence properties of this representation will not be considered here except that it will be noted that at a discontinuity of $f(x)$, the Fourier series converges to the mean of the left and right values of the discontinuity.

The Fourier Transform

The *Fourier Transform* is an integral transform defined over the entire line $x \in [-\infty, \infty]$. The Fourier transform and its inverse are defined as

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \tag{13.1.8a}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k) dk \,. \tag{13.1.8b}$$

There are other equivalent definitions. However, this definition will serve to illustrate the power and functionality of the Fourier transform method. We again note that formally, the transform is over the entire real line $x \in [-\infty, \infty]$ whereas our computational domain is only over a finite domain $x \in [-L, L]$. Further, the Kernel of the transform, $\exp(\pm ikx)$, describes oscillatory behavior. Thus the Fourier transform is essentially an eigenfunction expansion over all continuous wavenumbers k. And once we are on a finite domain $x \in [-L, L]$, the continuous eigenfunction expansion becomes a discrete sum of eigenfunctions and associated wavenumbers (eigenvalues).

Derivative Relations

The critical property in the usage of Fourier transforms concerns derivative relations. To see how these properties are generated, we begin by considering the Fourier transform of $f'(x)$. We denote the Fourier transform of $f(x)$ as $\widehat{f(x)}$. Thus we find

$$\widehat{f'(x)} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f'(x) dx = f(x) e^{-ikx}|_{-\infty}^{\infty} + \frac{ik}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \,. \tag{13.1.9}$$

Assuming that $f(x) \to 0$ as $x \to \pm\infty$ results in

$$\widehat{f'(x)} = \frac{ik}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx = ik\widehat{f(x)} \,. \tag{13.1.10}$$

Thus the basic relation $\widehat{f'} = ik\widehat{f}$ is established. It is easy to generalize this argument to an arbitrary number of derivatives. The final result is the following relation between Fourier transforms of the derivative and the Fourier transform itself

$$\widehat{f^{(n)}} = (ik)^n \widehat{f} \,. \tag{13.1.11}$$

This property is what makes Fourier transforms so useful and practical.

As an example of the Fourier transform, consider the following differential equation

$$y'' - \omega^2 y = -f(x) \quad x \in [-\infty, \infty] \,. \tag{13.1.12}$$

We can solve this by applying the Fourier transform to both sides. This gives the following reduction

$$\begin{aligned} &\widehat{y''} - \omega^2 \widehat{y} = -\widehat{f} \\ &-k^2 \widehat{y} - \omega^2 \widehat{y} = -\widehat{f} \\ &(k^2 + \omega^2) \widehat{y} = \widehat{f} \\ &\widehat{y} = \frac{\widehat{f}}{k^2 + \omega^2} \,. \end{aligned} \tag{13.1.13}$$

To find the solution $y(x)$, we invert the last expression above to yield

$$y(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} \frac{\widehat{f}}{k^2 + \omega^2} dk \, . \tag{13.1.14}$$

This gives the solution in terms of an integral which can be evaluated analytically or numerically.

The Fast Fourier Transform

The Fast Fourier transform routine was developed specifically to perform the forward and backward Fourier transforms. In the mid 1960s, Cooley and Tukey developed what is now commonly known as the FFT algorithm [15]. Their algorithm was named one of the top ten algorithms of the 20th century for one reason: the operation count for solving a system dropped to $O(N \log N)$. For N large, this operation count grows almost linearly like N. Thus it represents a great leap forward from Gaussian elimination and LU decomposition ($O(N^3)$). The key features of the FFT routine are as follows:

1. It has a low operation count: $O(N \log N)$.
2. It finds the transform on an interval $x \in [-L, L]$. Since the integration Kernel $\exp(ikx)$ is oscillatory, it implies that the solutions on this finite interval have periodic boundary conditions.
3. The key to lowering the operation count to $O(N \log N)$ is in discretizing the range $x \in [-L, L]$ into 2^n points, i.e. the number of points should be $2, 4, 8, 16, 32, 64, 128, 256, \cdots$.
4. The FFT has excellent accuracy properties, typically well beyond that of standard discretization schemes.

We will consider the underlying FFT algorithm in detail at a later time. For more information at the present, see [15] for a broader overview.

The practical implementation of the mathematical tools available in MATLAB is crucial. This lecture will focus on the use of some of the more sophisticated routines in MATLAB which are cornerstones to scientific computing. Included in this section will be a discussion of the Fast Fourier Transform routines (*fft, ifft, fftshift, ifftshift, fft2, ifft2*), sparse matrix construction (*spdiag, spy*), and high end iterative techniques for solving $\mathbf{Ax} = \mathbf{b}$ (*bicgstab, gmres*). These routines should be studied carefully since they are the building blocks of any serious scientific computing code.

Fast Fourier Transform: FFT, IFFT, FFTSHIFT, IFFTSHIFT

The Fast Fourier Transform will be the first subject discussed. Its implementation is straightforward. Given a function which has been discretized with 2^n points and represented by a vector $\mathbf{x}$, the FFT is found with the command *fft(x)*. Aside from transforming the function, the algorithm associated with the FFT does three major things: it shifts the data so that $x \in [0, L] \to [-L, 0]$ and $x \in [-L, 0] \to [0, L]$, additionally it multiplies every other mode by -1, and it assumes you are working on a 2π periodic domain. These properties are a consequence of the FFT algorithm discussed in detail at a later time.

To see the practical implications of the FFT, we consider the transform of a Gaussian function. The transform can be calculated analytically so that we have the exact relations:

$$f(x) = \exp(-\alpha x^2) \quad \to \quad \widehat{f}(k) = \frac{1}{\sqrt{2\alpha}} \exp\left(-\frac{k^2}{4\alpha}\right). \tag{13.1.15}$$

A simple MATLAB code to verify this with $\alpha = 1$ is as follows

```
clear all; close all;  % clear all variables and figures

L=20;   % define the computational domain [-L/2,L/2]
n=128;  % define the number of Fourier modes 2^n

x2=linspace(-L/2,L/2,n+1);  % define the domain discretization
x=x2(1:n);   % consider only the first n points:  periodicity

u=exp(-x.*x);          % function to take a derivative of
ut=fft(u);             % FFT the function
utshift=fftshift(ut); % shift FFT

figure(1), plot(x,u)     % plot initial gaussian
figure(2), plot(abs(ut))      % plot unshifted transform
figure(3), plot(abs(utshift)) % plot shifted transform
```

The second figure generated by this script shows how the pulse is shifted. By using the command *fftshift*, we can shift the transformed function back to its mathematically correct positions as shown in the third figure generated. However, before inverting the transformation, it is crucial that the transform is shifted back to the form of the second figure. The command *ifftshift* does this. In general, unless you need to plot the spectrum, it is better not to deal with the *fftshift* and *ifftshift* commands. A graphical representation of the *fft* procedure and its shifting properties is illustrated in Fig. 66 where a Gaussian is transformed and shifted by the *fft* routine.

FFT versus Finite-Difference Differentiation

Taylor series methods for generating approximations to derivatives of a given function can also be used. In the Taylor series method, the approximations are *local* since they are given by nearest neighbor coupling formulas of finite differences. Using the FFT, the approximation to the derivative is *global* since it uses an expansion basis of cosines and sines which extend over the entire domain of the given function.

The calculation of the derivative using the FFT is performed trivially from the derivative relation formula (13.1.11). As an example of how to implement this differentiation formula, consider a specific function:

$$u(x) = \text{sech}(x) \tag{13.1.16}$$

which has the following derivative relations

$$\frac{du}{dx} = -\text{sech}(x)\text{tanh}(x) \tag{13.1.17a}$$

$$\frac{d^2u}{dx^2} = \text{sech}(x) - 2\,\text{sech}^3(x)\,. \tag{13.1.17b}$$

A comparison can be made between the differentiation accuracy of finite difference formulas of Tables 12-13 and the spectral method using FFTs. The following code generates the derivative using the FFT method along with the finite difference $O(\Delta x^2)$ and $O(\Delta x^4)$ approximations considered with finite differences.

```
clear all; close all;  % clear all variables and figures

L=20;   % define the computational domain [-L/2,L/2]
n=128;  % define the number of Fourier modes 2^n

x2=linspace(-L/2,L/2,n+1);  % define the domain discretization
x=x2(1:n);   % consider only the first n points:  periodicity
dx=x(2)-x(1);  % dx value needed for finite difference
u=sech(x);     % function to take a derivative of
ut=fft(u);     % FFT the function
k=(2*pi/L)*[0:(n/2-1) (-n/2):-1]; % k rescaled to 2pi domain

% FFT calculation of derivatives

ut1=i*k.*ut;         % first derivative
ut2=-k.*k.*ut;       % second derivative
u1=real(ifft(ut1)); u2=real(ifft(ut2)); % inverse transform
u1exact=-sech(x).*tanh(x);  % analytic first derivative
u2exact=sech(x)-2*sech(x).^3;  % analytic second derivative
```

```
% Finite difference calculation of first derivative

 % 2nd-order accurate
 ux(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);
 for j=2:n-1
     ux(j)=(u(j+1)-u(j-1))/(2*dx);
 end
 ux(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);

 % 4th-order accurate
 ux2(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);
 ux2(2)=(-3*u(2)+4*u(3)-u(4))/(2*dx);
 for j=3:n-2
   ux2(j)=(-u(j+2)+8*u(j+1)-8*u(j-1)+u(j-2))/(12*dx);
 end
 ux2(n-1)=(3*u(n-1)-4*u(n-2)+u(n-3))/(2*dx);
 ux2(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);

figure(1)
plot(x,u,'r',x,u1,'g',x,u1exact,'go',x,u2,'b',x,u2exact,'bo')
figure(2)
subplot(3,1,1), plot(x,u1exact,'ks-',x,u1,'k',x,ux,'ko',x,ux2,'k*')
axis([-1.15 -0.75 0.47 0.5])
subplot(3,1,2), plot(x,u1exact,'ks-',x,u1,'kv',x,ux,'ko',x,ux2,'k*')
axis([-0.9376 -0.9374 0.49848 0.49850])
subplot(3,1,3), plot(x,u1exact,'ks-',x,u1,'kv',x,ux,'ko',x,ux2,'k*')
axis([-0.9376 -0.9374 0.498487 0.498488])
```

Note that the real part is taken after inverse Fourier transforming due to the numerical round off which generates a small $O(10^{-15})$ imaginary part.

Of course, this differentiation example works well since the periodic boundary conditions of the FFT are essentially satisfied with the choice of function $f(x) = \text{sech}(x)$. Specifically, for the range of values $x \in [-10, 10]$, the value of $f(\pm 10) = \text{sech}(\pm 10) \approx 9 \times 10^{-5}$. For a function which does not satisfy periodic boundary conditions, the FFT routine leads to significant error. Much of this error arises from the discontinuous jump and the associated *Gibb's phenomenon* which results when approximating with a Fourier series. To illustrate this, Fig. 102 shows the numerically generated derivative to the function $f(x) = \tanh(x)$, which is $f'(x) = \text{sech}^2(x)$. In this example, it is clear that the FFT method is highly undesirable, whereas the finite difference method performs as well as before.

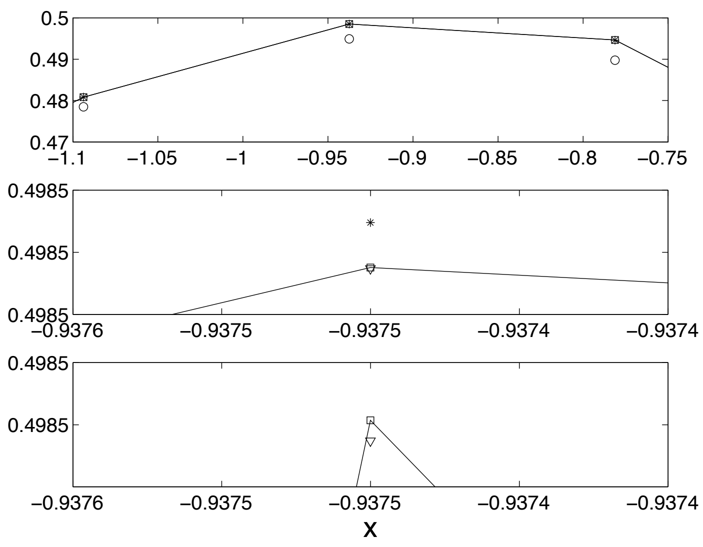


Figure 101: Accuracy comparison between second- and fourth-order finite difference methods and the spectral FFT method for calculating the first derivative. Note that by using the *axis* command, the exact solution (line) and its approximations can be magnified near an arbitrary point of interest. Here, the top figure shows that the second-order finite difference (circles) method is within $O(10^{-2})$ of the exact derivative. The fourth-order finite difference (star) is within $O(10^{-5})$ of the exact derivative. Finally, the FFT method is within $O(10^{-6})$ of the exact derivative. This demonstrates the spectral accuracy property of the FFT algorithm.

Higher dimensional Fourier transforms

For transforming in higher dimensions, a couple of choices in MATLAB are possible. For 2D transformations, it is recommended to use the commands **fft2** and **ifft2**. These will transform a matrix **A**, which represents data in the x and y direction, along the rows and then columns respectively. For higher dimensions, the **fft** command can be modified to **fft(x,[],N)** where N is the number of dimensions. Alternatively, use **fftn(x)** for multi-dimensional FFTs.

13.2 FFT Application: Radar Detection and Filtering

FFTs and other related frequency transforms have revolutionized the field of digital signal processing and imaging. The key concept in any of these applications is to use the FFT to analyze and manipulate data in the frequency domain.

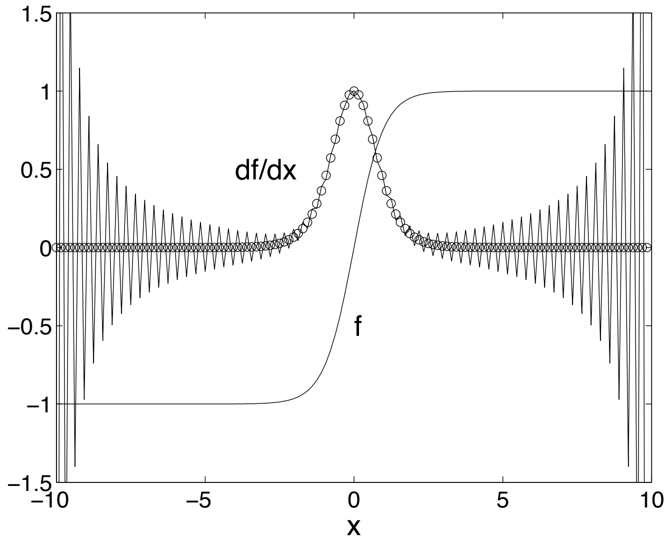


Figure 102: Accuracy comparison between the fourth-order finite difference method and the spectral FFT method for calculating the first derivative of $f(x) = \tanh(x)$. The fourth-order finite difference (circle) is within $O(10^{-5})$ of the exact derivative. In contrast, the FFT approximation (line) oscillates strongly and provides a highly inaccurate calculation of the derivative due to the non-periodic boundary conditions of the given function.

There are other methods of treating the signal in the time and frequency domain, but the simplest to begin with is the Fourier transform.

The Fourier transform is also used in quantum mechanics to represent a quantum wavepacket in either the spatial domain or the momentum (spectral) domain. Quantum mechanics is specifically mentioned due to the well-known *Heisenberg Uncertainty Principle*. Simply stated, you cannot know the exact position and momentum of a quantum particle simultaneously. This is simply a property of the Fourier transform itself. Essentially, a narrow signal in the time domain corresponds to a broadband source, whereas a highly confined spectral signature corresponds to a broad time domain source. This has significant implication for many techniques of signal analysis. In particular, an excellent decomposition of a given signal into its frequency components can render no information about *when* in time the different portions of signal actually occurred. Thus there is often a competition between trying to localize both in time and frequency a signal or stream of data. Time-frequency analysis is ultimately all about trying to resolve the time and frequency domain in an efficient and tractable manner. Various aspects of this time-frequency processing will be

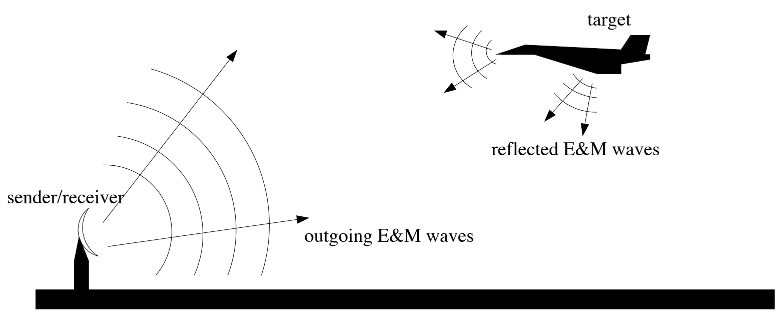


Figure 103: Schematic of the operation of a radar detection system. The radar emits an electromagnetic (E&M) field at a specific frequency. The radar then receives the reflection of this field from objects in the environment and attempts to determine what the objects are.

considered in later lectures, including wavelet based methods of time-frequency analysis.

At this point, we discuss a very basic concept and manipulation procedure to be performed in the frequency domain: noise attenuation via frequency (band-pass) filtering. This filtering process is fairly common in electronics and signal detection. As a specific application or motivation for spectral analysis, consider the process of radar detection depicted in Fig. 103. An outgoing electromagnetic field is emitted from a source which then attempts to detect the reflections of the emitted signal. In addition to reflections coming from the desired target, reflections can also come from geographical objects (mountains, trees, etc.) and atmospheric phenomena (clouds, precipitation, etc.). Further, there may be other sources of the electromagnetic energy at the desired frequency. Thus the radar detection process can be greatly complicated by a wide host of environmental phenomena. Ultimately, these effects combine to give at the detector a noisy signal which must be processed and filtered before a detection diagnosis can be made.

It should be noted that the radar detection problem discussed in what follows is highly simplified. Indeed, modern day radar detection systems use much more sophisticated time-frequency methods for extracting both the position and spectral signature of target objects. Regardless, this analysis gives a high-level view of the basic and intuitive concepts associated with signal detection.

To begin we consider an ideal signal, i.e. an electromagnetic pulse, generated in the time-domain.

```
clear all; close all;

L=30;    % time slot to transform
n=512;   % number of Fourier modes 2^9
```

```
t2=linspace(-L,L,n+1); t=t2(1:n);  % time discretization
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1];  % frequency components of FFT

u=sech(t);       % ideal signal in the time domain
figure(1), subplot(3,1,1), plot(t,u,'k'), hold on
```

This will generate an ideal hyperbolic secant shape in the time domain. It is this time domain signal that we will attempt to reconstruct via denoising and filtering. Figure 104(a) illustrates the ideal signal.

In most applications, the signals as generated above are not ideal. Rather, they have a large amount of noise integrated within them. Usually this noise is what is called *white noise*, i.e. a noise that effects all frequencies the same. We can add white noise to this signal by considering the pulse in the frequency domain.

```
noise=1;
ut=fft(u);
utn=ut+noise*(randn(1,n)+i*randn(1,n));
un=ifft(utn);
figure(1), subplot(3,1,2), plot(t,abs(un),'k'), hold on
```

These lines of code generate the Fourier transform of the function along with the vector **utn** which is the spectrum of the given signal with a complex and Gaussian distributed (mean zero, unit variance) noise source term added in. Figure 104 shows the difference between the ideal time-domain signal pulse and the more physically realistic pulse for which white noise has been added. In these figures, a clear *signal*, i.e. the original time-domain pulse, is still detected even in the presence of noise.

The fundamental question in signal detection now arises: is the time-domain structure received in a detector simply a result of noise, or is there an underlying signal being transmitted. For small amounts of noise, this is clearly not a problem. However, for large amounts of noise or low-levels of signal, the detection becomes a nontrivial matter.

In the context of radar detection, two competing objectives are at play: for the radar detection system, an accurate diagnosis of the data is critical in determining the presence of an aircraft. Competing in this regard is, perhaps, the aircraft's objective of remaining invisible, or undetected, from the radar. Thus an aircraft attempting to remain invisible will attempt to reflect as little electromagnetic signal as possible. This can be done by, for instance, covering the aircraft with materials that absorb the radar frequencies used for detection, i.e. stealth technology. An alternative method is to fly another aircraft through the region which bombards the radar detection system with electromagnetic energy at the detection frequencies. Thus the aircraft, attempting to remain undetected, may be successful since the radar system may not be able to distinguish

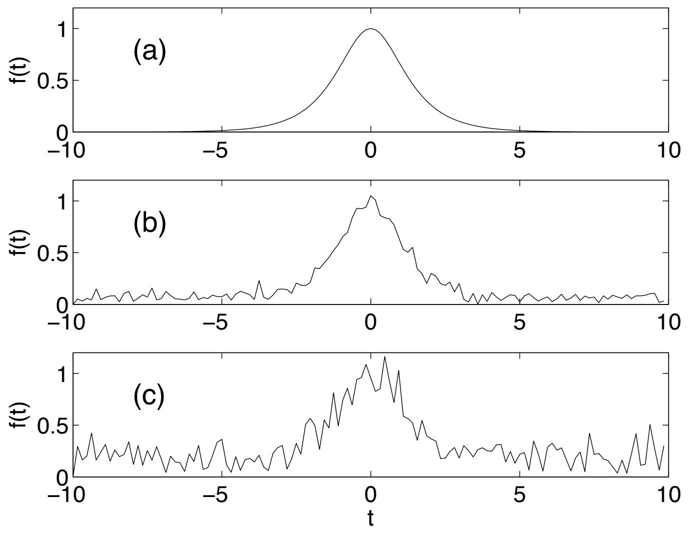


Figure 104: Ideal time-domain pulse (a) along with two realizations of the pulse with increasing noise strength (b) and (c). The noisy signals are what are typically expected in applications.

between the sources of the electromagnetic fields. This second option is, in some sense, like a radar jamming system. In the first case, the radar detection system will have a small signal-to-noise ratio and denoising will be critical to accurate detection. In the second case, an accurate time-frequency analysis is critical for determining the time localization and position of the competing signals.

The focus of what follows is to apply the ideas of spectral filtering to attempt to improve the signal detection by denoising. Spectral filtering is a method which allows us to extract information at specific frequencies. For instance, in the radar detection problem, it is understood that only a particular frequency (the emitted signal frequency) is of interest at the detector. Thus it would seem reasonable to filter out, in some appropriate manner, the remaining frequency components of the electromagnetic field received at the detector.

Consider a very noisy field created around the hyperbolic secant of the previous example (See Fig. 105):

```
noise=10;
ut=fft(u);
unt=ut+noise*(randn(1,n)+i*randn(1,n));
un=ifft(unt);
```

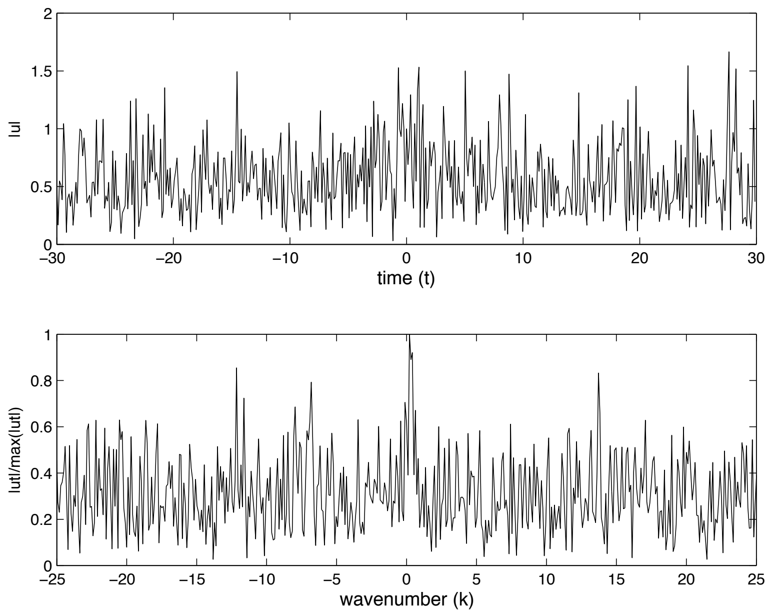


Figure 105: Time-domain (top) and frequency-domain (bottom) plots for a single realization of white-noise. In this case, the noise strength has been increased to ten, thus burying the desired signal field in both time and frequency.

```
subplot(2,1,1), plot(t,abs(un),'k')
axis([-30 30 0 2])
xlabel('time (t)'), ylabel('|u|')
subplot(2,1,2)
plot(fftshift(k),abs(fftshift(unt))/max(abs(fftshift(unt))),'k')
axis([-25 25 0 1])
xlabel('wavenumber (k)', ylabel('|ut|/max(|ut|)')
```

Figure 105 demonstrates the impact of a large noise applied to the underlying signal. In particular, the signal is completely buried within the white-noise fluctuations, making the detection of a signal difficult, if not impossible, with the unfiltered noisy signal field.

Filtering can help significantly improve the ability to detect the signal buried in the noisy field of Fig. 105. For the radar application, the frequency (wavenumber) of the emitted and reflected field is known, thus spectral filtering around this frequency can remove undesired frequencies and much of the white-noise picked up in the detection process. There are a wide range of filters that can be applied to such a problem. One of the simplest filters to consider here is the

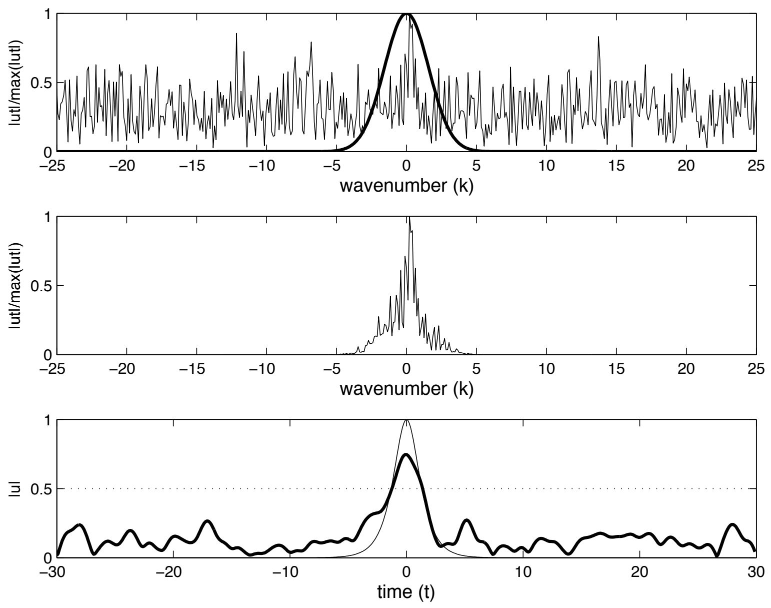


Figure 106: (top) White-noise inundated signal field in the frequency domain along with a Gaussian filter with bandwidth parameter $\tau = 0.2$ centered on the signal center frequency. (middle) The post-filtered signal field in the frequency domain. (bottom) The time-domain reconstruction of the signal field (bolded line) along with the ideal signal field (light line) and the detection threshold of the radar (dotted line).

Gaussian filter (See Fig. 106):

$$\mathcal{F}(k) = \exp(-\tau(k-k_0)^2) \tag{13.2.1}$$

where τ measures the bandwidth of the filter, and k is the wavenumber. The generic filter function $\mathcal{F}(k)$ in this case acts as a low-pass filter since it eliminates high-frequency components in the system of interest. Note that these are high-frequencies in relation to the center-frequency ($k = k_0$) of the desired signal field. In the example considered here $k_0 = 0$.

Application of the filter attenuates strongly those frequencies away from center frequency k_0. Thus if there is a signal near k_0, the filter isolates the signal input around this frequency or wavenumber. Application of the filtering results in the following spectral processing in MATLAB:

```
filter=exp(-0.2*(k).^2);
unft=filter.*unt;
unf=ifft(unft);
```

The results of this code are illustrate in Fig. 106 where the white-noise inundated signal field in the spectral domain is filtered with a Gaussian filter centered around the zero wavenumber $k_0 = 0$ with a bandwidth parameter $\tau = 0.2$. The filtering extracts nicely the signal field despite the strength of the applied white-noise. Indeed, Fig. 106 (bottom) illustrates the effect of the filtering process and its ability to reproduce an approximation to the signal field. In addition to the extracted signal field, a detection threshold is placed (dotted line) on the graph in order to illustrate that the detector would read the extracted signal as a target.

As a matter of completeness, the extraction of the electromagnetic field is also illustrated when not centered on the center frequency. Figure 107 shows the field that is extracted for a filter centered around $k_0 = 15$. This is done easily with the MATLAB commands

```
filter=exp(-0.2*(k-15).^2);
unft=filter.*unt;
unf=ifft(unft);
```

In this case, there is no signal around the wavenumber $k_0 = 15$. However, there is a great deal of white-noise electromagnetic energy around this frequency. Upon filtering and inverting the Fourier transform, it is clear from Figure 107 (bottom) that there is no discernible target present. As before, a decision threshold is set at $|u| = 0.5$ and the white noise fluctuations clearly produce a field well below the threshold line.

In all of the analysis above, filtering is done on a given set of signal data. However, the signal is evolving in time, i.e. the position of the aircraft is probably moving. Moreover, one can imagine that there is some statistical chance that a type I or type II error can be made in the detection. Thus a target may be identified when there is no target, or a target has been missed completely. Ultimately, the goal is to use repeated measurements from the radar of the airspace in order to try to avoid a detection error or failure. Given that you may launch a missile, the stakes are high and a high level of confidence is needed in the detection process. It should also be noted that filter design, shape, and parameterization play significant roles in designing optimal filters. Thus filter design is an object of strong interest in signal processing applications.

13.3 FFT Application: Radar Detection and Averaging

The last section clearly shows the profound and effective use of filtering in cleaning up a noisy signal. This is one of the most basic illustrations of the power of basic signal processing. Other methods also exist to help reduce noise and generate better time-frequency resolution. In particular, the filtering example given fails to use two key facts: the noise is white, and radar will continue to produce more signal data. These two facts can be used to produce useful

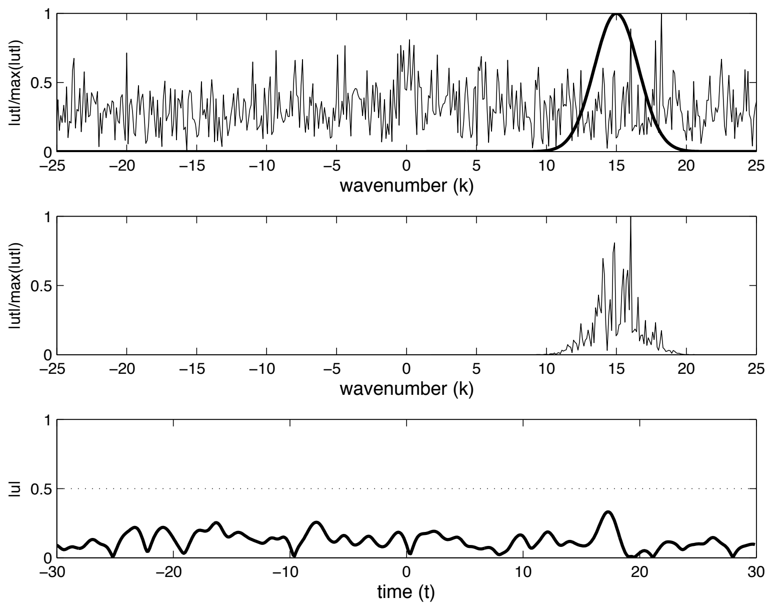


Figure 107: (top) White-noise inundated signal field in the frequency domain along with a Gaussian filter with bandwidth parameter $\tau = 0.2$ centered on the signal frequency $k = 15$. (middle) The post-filtered signal field in the frequency domain. (bottom) The time-domain reconstruction of the signal field (bolded line) along the detection threshold of the radar (dotted line).

information about the incoming signal. In particular, it is known, and demonstrated in the last section, that white-noise can be modeled adding a normally distributed random variable with zero mean and unit variance to each Fourier component of the spectrum. The key here: *with zero mean.* Thus over many signals, the white-noise should, on average, add up to zero. A very simple concept, yet tremendously powerful in practice for signal processing systems where there is continuous detection of a signal.

To illustrate the power of denoising the system by averaging, the simple analysis of the last section will be considered for a time pulse. In beginning this process, a time window must be selected and its Fourier resolution decided:

```
clear all; close all; clc

L=30;  % time slot
n=512; % Fourier modes
t2=linspace(-L,L,n+1); t=t2(1:n);
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1]; ks=fftshift(k);
```

```
noise=10;
```

In order to be more effective in plotting our spectral results, the Fourier components in the standard shifted frame (**k**) and its unshifted counterpart (**ks**) are both produced. Further, the noise strength of the system is selected.

In the next portion of the code, the total number of data frames, or realizations of the incoming signal stream, is considered. The word realizations is used here to help underscore the fact that for each data frame, a new realization of the white-noise is produced. In the following code, one, two, five, and one hundred realizations are considered.

```
labels=['(a)';'(b)';'(c)';'(d)'];
realize=[1 2 5 100];
for jj=1:length(realize)

  u=sech(t); ave=zeros(1,n);
  ut=fft(u);
  for j=1:realize(jj)
    utn(j,:)=ut+noise*(randn(1,n)+i*randn(1,n));
    ave=ave+utn(j,:);
    dat(j,:)=abs(fftshift(utn(j,:)))/max(abs(utn(j,:)));
    un(j,:)=ifft(utn(j,:));
  end
  ave=abs(fftshift(ave))/realize(jj);

  subplot(4,1,jj)
  plot(ks,ave/max(ave),'k')
  set(gca,'Fontsize',[15])
  axis([-20 20 0 1])
  text(-18,0.7,labels(jj,:),'Fontsize',[15])
  ylabel('|fft(u)|','Fontsize',[15])

end
hold on
plot(ks,abs(fftshift(ut))/max(abs(ut)),'k:','Linewidth',[2])
set(gca,'Fontsize',[15])
xlabel('frequency (k)')
```

Figure 108 demonstrates the averaging process in the spectral domain as a function of the number of realizations. With one realization, it is difficult to determine if there is a true signal, or if the entire spectrum is noise. Already after two realizations, the center frequency structure starts to appear. Five realizations has a high degree of discrimination between the noise and signal and 100 realizations is almost exactly converged to the ideal, noise-free signal. As expected, the noise in the averaging process is eliminated since its mean is

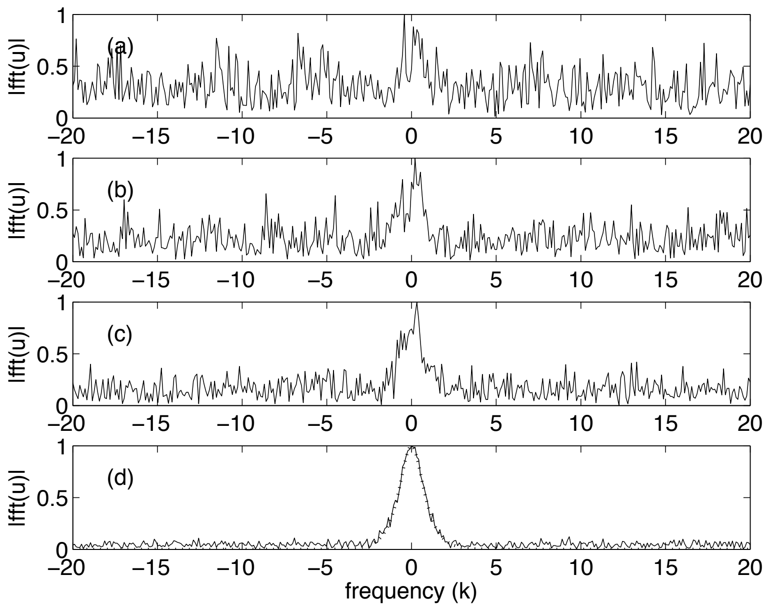


Figure 108: Average spectral content for (a) one, (b) two, (c) five and (d) one hundred realizations of the data. The dotted line in (d) represents the ideal, noise-free spectral signature.

zero. To view a few of the data realizations used in computing the averaging in Fig. 108, the first eight signals used to compute the average for 100 realizations are shown in Fig. 109. The following code is used to produce the waterfall plot used:

```
figure(2)
waterfall(ks,1:8,dat(1:8,:)), colormap([0 0 0]), view(-15,80)
set(gca,'Fontsize',[15],'Xlim',[-28 28],'Ylim',[1 8])
xlabel('frequency (k)'), ylabel('realization'),zlabel('|fft(u)|')
```

Figures 108 and 109 illustrate how the averaging process can ultimately extract a clean spectral signature. As a consequence, however, you completely loose the time-domain dynamics. In other words, one could take the cleaned up spectrum of Fig. 108 and invert the Fourier transform, but this would only give the averaged time domain signal, but not how it actually evolved over time. To consider this problem more concretely, consider the following bit of code that generates a time-varying signal shown in Fig. 110. There are a total of 21 realizations of data in this example.

```
slice=[0:0.5:10];
```

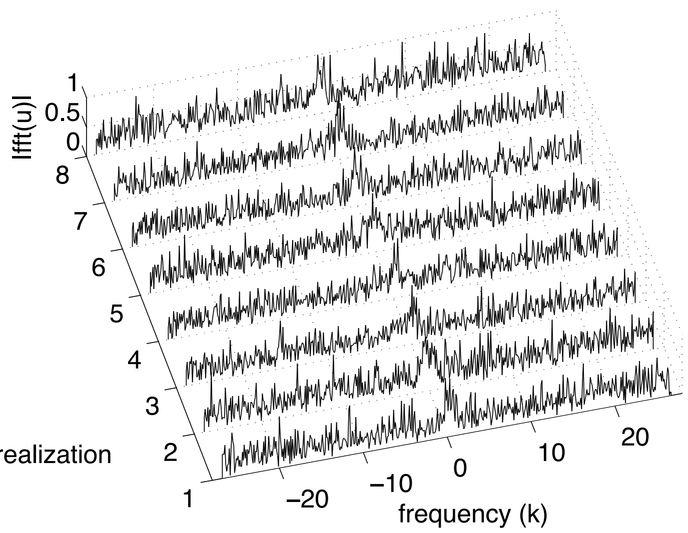


Figure 109: Typical time domain signals that would be produced in a signal detection system. At different times of measurement, the added white-noise would alter the signal stream significantly.

```
[T,S]=meshgrid(t,slice);
[K,S]=meshgrid(k,slice);

U=sech(T-10*sin(S)).*exp(i*0*T);
subplot(2,1,1)
waterfall(T,S,U), colormap([0 0 0]), view(-15,70)
set(gca,'Fontsize',[15],'Xlim',[-30 30],'Zlim',[0 2])
xlabel('time (t)'), ylabel('realizations'), zlabel('|u|')

for j=1:length(slice)
  Ut(j,:)=fft(U(j,:));
  Kp(j,:)=fftshift(K(j,:));
  Utp(j,:)=fftshift(Ut(j,:));
  Utn(j,:)=Ut(j,:)+noise*(randn(1,n)+i*randn(1,n));
  Utnp(j,:)=fftshift(Utn(j,:))/max(abs(Utn(j,:)));
  Un(j,:)=ifft(Utn(j,:));
end
figure(3)
subplot(2,1,2)
waterfall(Kp,S,abs(Utp)/max(abs(Utp(1,:)))), colormap([0 0 0]), view(-15,70)
```

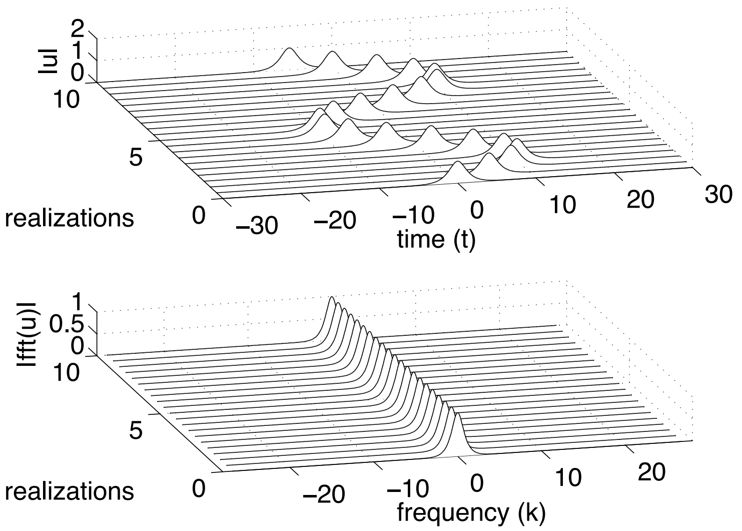


Figure 110: Ideal time-frequency behavior for a time-domain pulse that evolves dynamically in time. The spectral signature remains unchanged as the pulse moves over the time domain.

```
set(gca,'Fontsize',[15],'Xlim',[-28 28])
xlabel('frequency (k)'), ylabel('realizations'), zlabel('|fft(u)|')
```

If this were a radar problem, the movement of the signal would represent an aircraft moving in time. The spectrum, however, remains fixed at the transmitted signal frequency. Thus it is clear even at this point that averaging over the frequency realizations produces a clean, localized signature in the frequency domain, whereas averaging over the time-domain only smears out the evolving time-domain pulse.

The ideal pulse evolution in Fig. 110 is now inundated with white-noise as shown in Fig. 111. The noise is added to each realization, or data measurement, with the same strength as shown in Figs. 108 and 109. The following code plots the noise time-domain and spectral evolution:

```
figure(4)
subplot(2,1,1)
waterfall(T,S,abs(Un)), colormap([0 0 0]), view(-15,70)
set(gca,'Fontsize',[15],'Xlim',[-30 30],'Zlim',[0 2])
xlabel('time (t)'), ylabel('realizations'), zlabel('|u|')
subplot(2,1,2)
waterfall(Kp,S,abs(Utnp)), colormap([0 0 0]), view(-15,70)
```

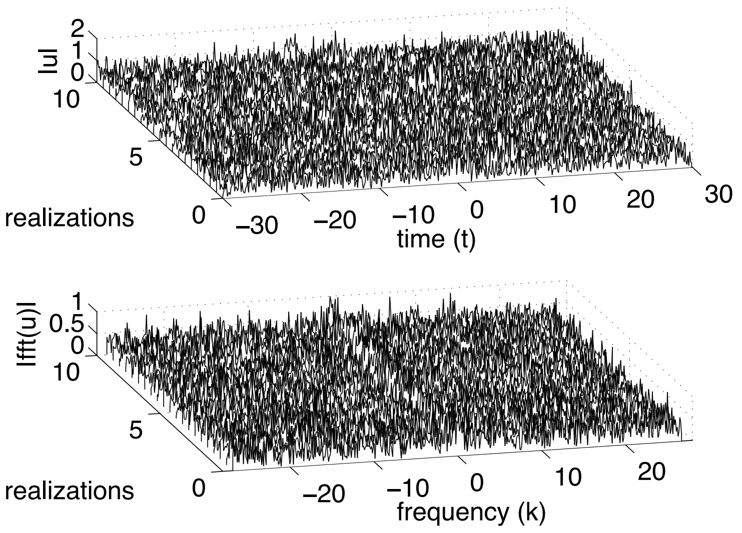


Figure 111: A more physically realistic depiction of the time-domain and frequency spectrum as a function of the number of realizations. The goal is to extract the meaningful data buried within the noise.

```
set(gca,'Fontsize',[15],'Xlim',[-28 28])
xlabel('frequency (k)'), ylabel('realizations'), zlabel('|fft(u)|')
```

The obvious question arises about how to clean up the signal and whether something meaningful can be extracted from the data or not. After all, there are only 21 data slices to work with. The following code averages over the 21 data realizations in both the time and frequency domains

```
figure(5)
Uave=zeros(1,n); Utave=zeros(1,n);
for j=1:length(slice)
  Uave=Uave+Un(j,:);
  Utave=Utave+Utn(j,:);
end
Uave=Uave/length(slice);
Utave=fftshift(Utave)/length(slice);

subplot(2,1,1)
plot(t,abs(Uave),'k')
set(gca,'Fontsize',[15])
xlabel('time (t)'), ylabel('|u|')
```

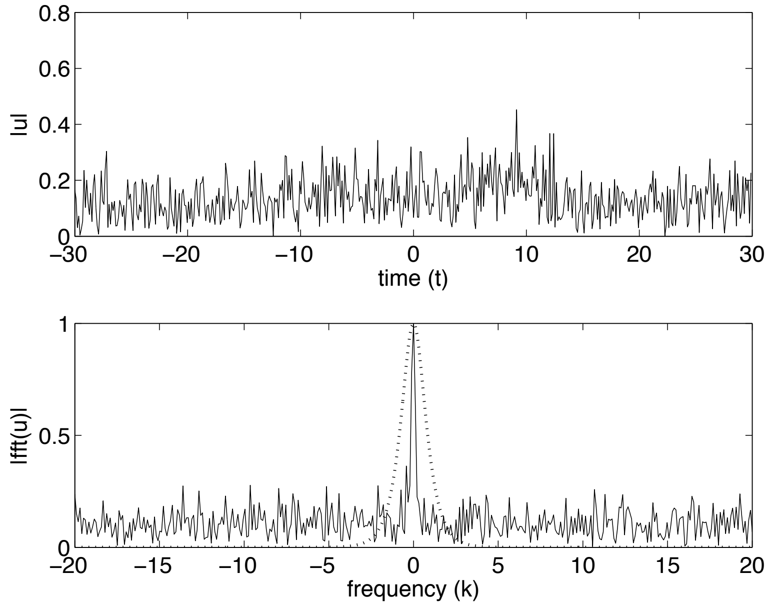


Figure 112: Averaged time-domain and spectral profiles for the 21 realizations of data shown in Fig. 111. Even with a limited sampling, the spectral signature at the center frequency is extracted.

```
subplot(2,1,2)
plot(ks,abs(Utave)/max(abs(Utave)),'k'), hold on
plot(ks,abs(fftshift(Ut(1,:)))/max(abs(Ut(1,:))),'k:','Linewidth',[2])
axis([-20 20 0 1])
set(gca,'Fontsize',[15])
xlabel('frequency (k)'), ylabel('|fft(u)|')
```

Figure 112 shows the results from the averaging process for a nonstationary signal. The top graph shows that averaging over the time domain for a moving signal produces no discernible signal. However, averaging over the frequency domain produces a clear signature at the center-frequency of interest. Ideally, if more data is collected a better average signal is produced. However, in many applications, the acquisition of data is limited and decisions must be made upon what are essentially small sample sizes.

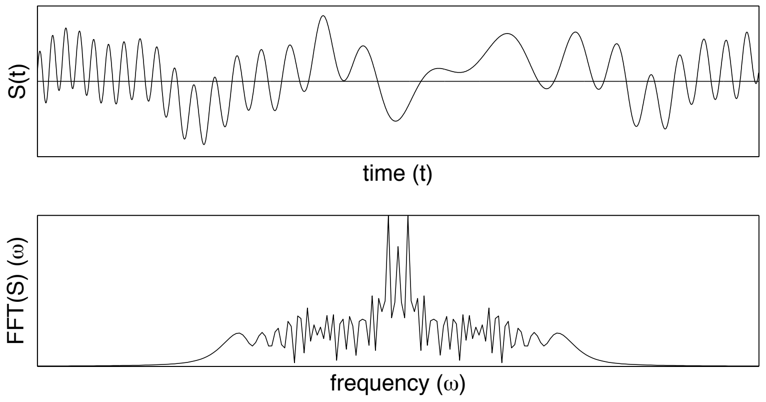


Figure 113: Signal $S(t)$ and its normalized Fourier transform $\hat{S}(\omega)$. Note the large number of frequency components that make up the signal.

13.4 Time-Frequency Analysis: Windowed Fourier Transforms

The Fourier transform is one of the most important and foundational methods for the analysis of signals. However, it was realized very early on that Fourier transform based methods had severe limitations. Specifically, when transforming a given time signal, it is clear that all the frequency content of the signal can be captured with the transform, but the transform fails to capture the *moment in time* when various frequencies were actually exhibited. Figure 113 shows a proto-typical signal that may be of interest to study. The signal $S(t)$ is clearly comprised of various frequency components that are exhibited at different times. For instance, at the beginning of the signal, there are high frequency components. In contrast, the middle of the signal has relatively low frequency oscillations. If the signal represented music, then the beginning of the signal would produce high notes while the middle would produce low notes. The Fourier transform of the signal contains all this information, but there is no indication of *when* the high or low notes actually occur in time. Indeed, by definition the Fourier transform eliminates all time-domain information since you actually integrated out all time in Eq. (13.1.8).

The obvious question to arise is this: What is the Fourier transform good for in the context of signal processing? In the previous sections where the Fourier transform was applied, the signal being investigated was fixed in frequency, i.e. a sonar or radar detector with a fixed frequency ω_0. Thus for a given signal, the frequency of interest did not shift in time. By using different measurements in time, a signal tracking algorithm could be constructed. Thus an implicit assumption was made about the invariance of the signal frequency. Ultimately, the Fourier transform is superb for one thing: characterizing *stationary* or periodic signals. Informally, a stationary signal is such that repeated measurements of the signal in time yield an average value that does not change in time. Most signals, however, do not satisfy this criteria. A song, for instance, changes its average Fourier components in time as the song progresses in time. Thus the generic signal $S(t)$ that should be considered, and that is plotted as an example in Fig. 113 is a *non-stationary signal* whose average signal value does change in time. It should be noted that in our application of radar detection of a moving target, use was made of the stationary nature of the spectral content. This allowed for a clear idea of where to filter the signal $\hat{S}(\omega)$ in order to reconstruct the signal $S(t)$.

Having established the fact that the direct application of the Fourier transform provides a nontenable method for extracting signal information, it is natural to pursue modifications of the method in order to extract time and frequency information. The most simple minded approach is to consider Fig. 113 and to decompose the signal over the time domain into separate time frames. Figure 114 shows the original signal $S(t)$ considered but now decomposed into four smaller time windows. In this decomposition, for instance, the first time frame is considered with the remaining three time frames zeroed out. For each time window, the Fourier transform is applied in order to characterize the frequencies present during that time frame. The highest frequency components are captured in Fig. 114(a) which is clearly seen in its Fourier transform. In contrast, the slow modulation observed in the third time frame (c) is devoid of high-frequency components as observed in Fig. 114(c). This method thus exhibits the ability of the Fourier transform, appropriately modified, to extract out both time and frequency information from the signal.

The limitations of the direct application of the Fourier transform, and its inability to localize a signal in both the time and frequency domains, were realized very early on in the development of radar and sonar detection. The Hungarian physicist/mathematician/electrial engineer Gábor Dénes (Physics Nobel Prize in 1971 for the discovery of holography in 1947) was first to propose a formal method for localizing both time and frequency. His method involved a simple modification of the Fourier transform kernel. Thus Gábor introduced the kernel

$$g_{t,\omega}(\tau) = e^{i\omega\tau} g(\tau - t) \tag{13.4.1}$$

where the new term to the Fourier kernel $g(\tau - t)$ was introduced with the aim of localizing both time and frequency. The *Gábor transform*, also known as the *short-time Fourier transform (STFT)* is then defined as the following:

$$\mathcal{G}[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau}d\tau = (f, \bar{g}_{t,\omega}) \tag{13.4.2}$$

where the bar denotes the complex conjugate of the function. Thus the function $g(\tau - t)$ acts as a time filter for localizing the signal over a specific window of time. The integration over the parameter τ slides the time-filtering window

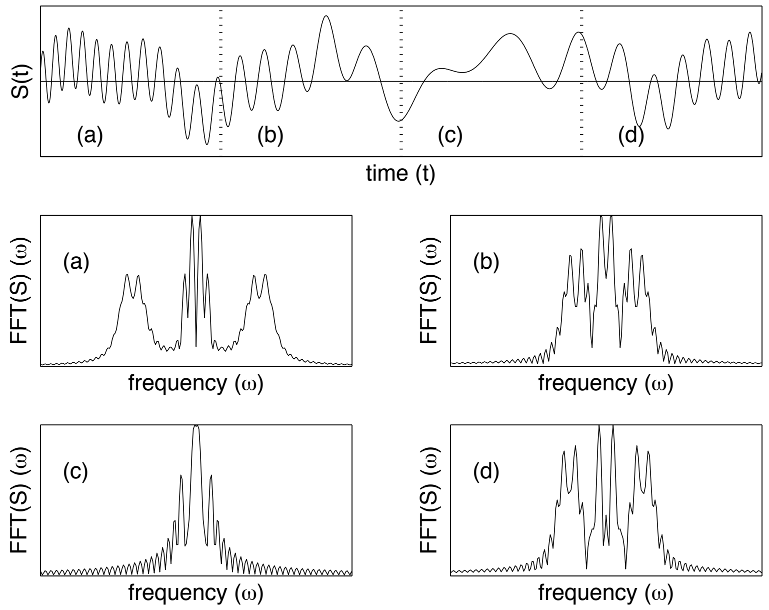


Figure 114: Signal $S(t)$ decomposed into four equal and separate time frames (a), (b), (c) and (d). The corresponding normalized Fourier transform of each time frame $\hat{S}(\omega)$ is illustrated below the signal. Note that this decomposition gives information about the frequencies present in each smaller time frame.

down the entire signal in order to pick out the frequency information at each instant of time. Figure 115 gives a nice illustration of how the time filtering scheme of Gábor works. In this figure, the time filtering window is centered at τ with a width a. Thus the frequency content of a window of time is extracted and τ is modified to extract the frequencies of another window. The definition of the Gábor transform captures the entire time-frequency content of the signal. Indeed, the Gábor transform is a function of the two variables t and ω.

A few of the key mathematical properties of the Gábor transform are highlighted here. To be more precise about these mathematical features, some assumptions about commonly used $g_{t,\omega}$ are considered. Specifically, for convenience we will consider g to be real and symmetric with $\|g(t)\| = 1$ and $\|g(\tau - t)\| = 1$ where $\|\cdot\|$ denotes the L_2 norm. Thus the definition of the Gábor transform, or STFT, is modified to

$$\mathcal{G}[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)g(\tau - t)e^{-i\omega\tau}d\tau \qquad (13.4.3)$$

with $g(\tau - t)$ inducing localization of the Fourier integral around $t = \tau$. With this definition, the following properties hold

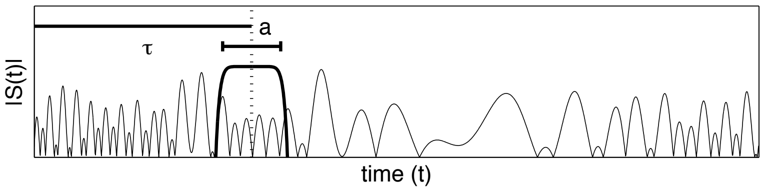


Figure 115: Graphical depiction of the Gábor transform for extracting the time-frequency content of a signal $S(t)$. The time filtering window $g(\tau - t)$ is centered at τ with width a.

1. The energy is bounded by the Schwarz inequality so that

$$|\tilde{f}_g(t,\omega)| \leq \|f\|\|g\| \tag{13.4.4}$$

2. The energy in the signal plane around the neighborhood of (t,ω) is calculated from

$$|\tilde{f}_g(t,\omega)|^2 = \left| \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau \right|^2 \tag{13.4.5}$$

3. The time-frequency spread around a Gábor window is computed from the variance, or second moment, so that

$$\sigma_t^2 = \int_{-\infty}^{\infty} (\tau - t)^2 |g_{t,\omega}(\tau)|^2 d\tau = \int_{-\infty}^{\infty} \tau^2 |g(\tau)|^2 d\tau \tag{13.4.6a}$$

$$\sigma_\omega^2 = \frac{1}{2\pi} \int_{-\infty}^{\infty} (\nu - \omega)^2 |\tilde{g}_{t,\omega}(\nu)|^2 d\nu = \frac{1}{2\pi} \int_{-\infty}^{\infty} \nu^2 |\tilde{g}(\nu)|^2 d\nu \tag{13.4.6b}$$

where $\sigma_t \sigma_\omega$ is independent of t and ω and is governed by the Heinsenberg uncertainty principle.

4. The Gábor transform is linear so that

$$\mathcal{G}[af_1 + bf_2] = a\mathcal{G}[f_1] + b\mathcal{G}[f_2] \tag{13.4.7}$$

5. The Gábor transform can be inverted with the formula

$$f(\tau) = \frac{1}{2\pi} \frac{1}{\|g\|^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \tilde{f}_g(t,\omega) g(\tau - t) e^{i\omega\tau} d\omega dt \tag{13.4.8}$$

where the integration must occur over all frequency and time-shifting components. This double integral is in contrast to the Fourier transform which requires only a single integration since it is a function, $\hat{f}(\omega)$, of the frequency alone.

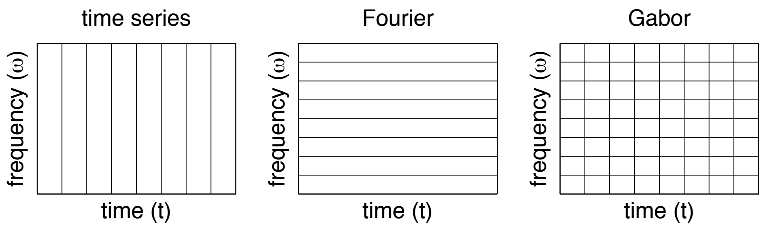


Figure 116: Graphical depiction of the difference between a time series analysis, Fourier analysis and Gábor analysis of a signal. In the time series method, good resolution is achieved of the signal in the time domain, but no frequency resolution is achieved. In Fourier analysis, the frequency domain is well resolved at the expense of losing all time resolution. The Gábor method, or short time Fourier transform, is constructed to give both time and frequency resolution. The area of each box can be constructed from $\sigma_t^2\sigma_\omega^2$.

Figure 116 is a cartoon representation of the fundamental ideas behind a time series analysis, Fourier transform analysis and Gábor transform analysis of a given signal. In the time series method, good resolution is achieved of the signal in the time domain, but no frequency resolution is achieved. In Fourier analysis, the frequency domain is well resolved at the expense of losing all time resolution. The Gábor method, or short-time Fourier transform, trades away some measure of accuracy in both the time and frequency domains in order to give both time and frequency resolution simultaneously. Understanding this figure is critical to understanding the basic, high-level notions of time-frequency analysis.

In practice, the Gábor transform is computed by discretizing the time and frequency domain. Thus a discrete version of the transform (13.4.2) needs to be considered. Essentially, by discretizing, the transform is done on a lattice of time and frequency. Thus consider the lattice, or sample points,

$$\nu = m\omega_0 \tag{13.4.9a}$$
$$\tau = nt_0 \tag{13.4.9b}$$

where m and n are integers and $\omega_0, t_0 > 0$ are constants. Then the discrete version of $g_{t,\omega}$ becomes

$$g_{m,n}(t) = e^{i2\pi m\omega_0 t}g(t - nt_0) \tag{13.4.10}$$

and the Gábor transform becomes

$$\tilde{f}(m,n) = \int_{-\infty}^{\infty} f(t)\bar{g}_{m,n}(t)dt = (f, g_{m,n}) \ . \tag{13.4.11}$$

Note that if $0 < t_0, \omega_0 < 1$, then the signal is over-sampled and time frames exist which yield excellent localization of the signal in both time and frequency.

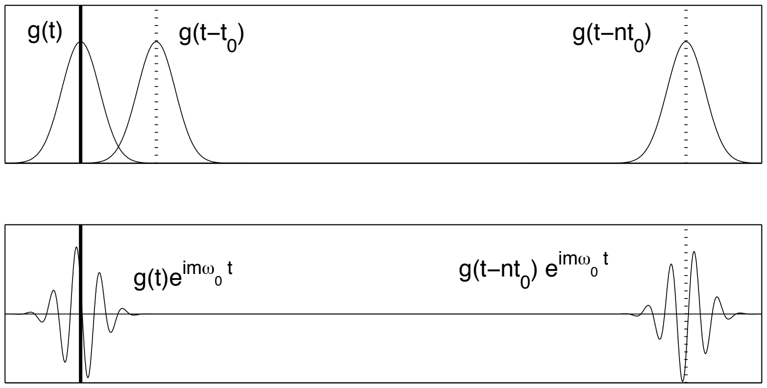


Figure 117: Illustration of the discrete Gábor transform which occurs on the lattice sample points Eq. (13.4.9). In the top figure, the translation with $\omega_0 = 0$ is depicted. The bottom figure depicts both translation in time and frequency. Note that the Gábor frames (windows) overlap so that good resolution of the signal can be achieved in both time and frequency since $0 < t_0, \omega_0 < 1$.

If $\omega_0, t_0 > 1$, the signal is under-sampled and the Gábor lattice is incapable of reproducing the signal. Figure 117 shows the Gábor transform on lattice given by Eq. (13.4.9). The overlap of the Gábor window frames ensures that good resolution in time and frequency of a given signal can be achieved.

Drawbacks of the Gábor (STFT) transform

Although the Gábor transform gives a method whereby time and frequency can be simultaneously characterized, there are obvious limitations to the method. Specifically, the method is limited by the time filtering itself. Consider the illustration of the method in Fig. 115. The time window filters out the time behavior of the signal in a window centered at τ with width a. Thus when considering the spectral content of this window, any portion of the signal with a wavelength longer than the window is completely lost. Indeed, since the Heinsenberg relationship must hold, the shorter the time filtering window, the less information there is concerning the frequency content. In contrast, longer windows retain more frequency components, but this comes at the expense of losing the time resolution of the signal. Figure 118 provides a graphical description of the failings of the Gábor transform. Specifically the trade offs that occur between time and frequency resolution, and the fact that high-accuracy in one of these comes at the expense of resolution in the other parameter. This is a consequence of a fixed time filtering window.

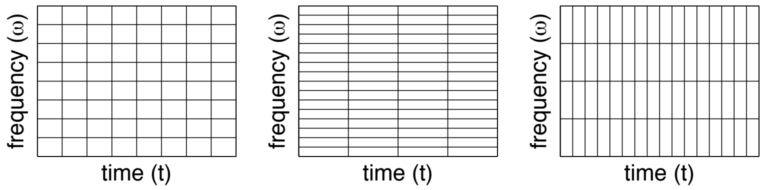


Figure 118: Illustration of the resolution trade-offs in the discrete Gábor transform. The left figure shows a time filtering window that produces nearly equal localization of the time and frequency signal. By increasing the length of the filtering window, increased frequency resolution is gained at the expense of worse time resolution (middle figure). Decreasing the time window does the opposite: time resolution is increased at the expense of poor frequency resolution (right figure).

Other short-time Fourier transform methods

The Gábor transform is not the only windowed Fourier transform that has been developed. There are several other well-used and highly developed STFT techniques. Here, a couple of these more highly used methods will be mentioned for completeness [27].

The *Zak transform* is closely related to the Gábor transform. It is also called the *Weil-Brezin* transform in harmonic analysis. First introduced by Gelfand in 1950 as a method for characterizing eigenfunction expansions in quantum mechanical systems with periodic potentials, it has been generalized to be a key mathematical tool for the analysis of Gábor transform methods. The Zak transform is defined as

$$\mathcal{L}_a f(t,\omega) = \sqrt{a} \sum_{n=-\infty}^{\infty} f(at+an)e^{-i2\pi n\omega} \tag{13.4.12}$$

where $a > 0$ is a constant and n is an integer. Two useful and key properties of this transform are as follows: $\mathcal{L}f(t,\omega+1) = \mathcal{L}f(t,\omega)$ (periodicity) and $\mathcal{L}f(t+1,\omega) = \exp(i2\pi\omega)\mathcal{L}f(t,\omega)$ (quasi-periodicity). These properties are particularly important for considering physical problems placed on a lattice.

The *Wigner-Ville Distribution* is a particularly important transform in the development of radar and sonar technologies. Its various mathematical properties make it ideal for these applications and provides a method for achieving great time and frequency localization. The Wigner-Ville transform is defined as

$$\mathcal{W}_{f,g}(t,\omega) = \int_{-\infty}^{\infty} f(t+\tau/2)\bar{g}(t-\tau/2)e^{-i\omega}\tau d\tau \tag{13.4.13}$$

where this is a standard Fourier kernel which transforms the function $f(t+$

$\tau/2)\bar{g}(t-\tau/2)$. This transform is nonlinear since $\mathcal{W}_{f_1+f_2,g_1+g_2} = \mathcal{W}_{f_1,g_1} + \mathcal{W}_{f_1,g_2} + \mathcal{W}_{f_2,g_1} + \mathcal{W}_{f_2,g_2}$ and $\mathcal{W}_{f+g} = \mathcal{W}_f + \mathcal{W}_g + 2\Re\{\mathcal{W}_{f,g}\}$.

Ultimately, alternative forms of the STFT are developed for one specific reason: to take advantage of some underlying properties of a given system. It is rare that a method developed for radar would be broadly applicable to other physical systems unless it were operating under the same physical principles. Regardless, one can see that specialty techniques exist for time-frequency analysis of different systems.

13.5 Time-Frequency Analysis and Wavelets

The Gábor transform established two key principles for joint time-frequency analysis: *translation* of a short-time window and *scaling* of the short-time window to capture finer time resolution. Figure 115 shows the basic concept introduced in the theory of windowed Fourier transforms. Two parameters are introduced to handle the *translation* and *scaling*, namely τ and a. The shortcoming of this method is that it trades off accuracy in time (frequency) for accuracy in frequency (time). Thus the fixed window size imposes a fundamental limitation on the level of time-frequency resolution that can be obtained.

A simple modification to the Gábor method is to allow the scaling window (a) to vary in order to successively extract improvements in the time resolution. In other words, first the low-frequency (poor time resolution) components are extracted using a broad scaling window. The scaling window is subsequently shortened in order to extract out higher-frequencies and better time resolution. By keeping a catalogue of the extracting process, both excellent time and frequency resolution of a given signal can be obtained. This is the fundamental principle of *wavelet theory.* The term wavelet means little wave and originates from the fact that the scaling window extracts out smaller and smaller pieces of waves from the larger signal.

Wavelet analysis begins with the consideration of a function known as the *mother wavelet*:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{13.5.14}$$

where $a \neq 0$ and b are real constants. The parameter a is the scaling parameter illustrated in Fig. 115 whereas the parameter b now denotes the translation parameter (previously denoted by τ in Fig. 115). Unlike Fourier analysis, and very much like Gábor transforms, there are a vast variety of mother wavelets that can be constructed. In principle, the mother wavelet is designed to have certain properties that are somehow beneficial for a given problem. Thus depending upon the application, different mother wavelets may be selected.

Ultimately, the wavelet is simply another expansion basis for representing a given signal or function. Thus it is not unlike the Fourier transform which represents the signal as a series of sines and cosines. Historically, the first wavelet was constructed by Haar in 1910 [28]. Thus the concepts and ideas of

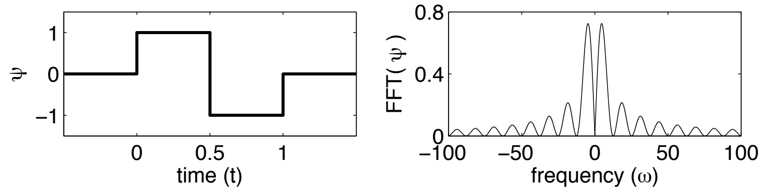


Figure 119: Representation of the compactly supported Haar wavelet function $\psi(t)$ and its Fourier transform $\hat{\psi}(\omega)$. Although highly localized in time due to the compact support, it is poorly localized in frequency with a decay of $1/\omega$.

wavelets are a century old. However, their widespread use and application did not become prevalent until the mid-1980s. The Haar wavelet is given by the piecewise constant function

$$\psi(t) = \begin{cases} 1 & 0 \leq t < 1/2 \\ -1 & 1/2 \leq t < 1 \\ 0 & \text{otherwise} \end{cases} . \tag{13.5.15}$$

Figure 119 shows the Haar wavelet step function and its Fourier transform which is a sinc like function. Note further that $\int_{-\infty}^{\infty} \psi(t)dt = 0$ and $\|\psi(t)\|^2 = \int_{-\infty}^{\infty} |\psi(t)|^2 dt = 1$. The Haar wavelet is an ideal wavelet for describing localized signals in time (or space) since it has compact support. Indeed, for highly localized signals, it is much more efficient to use the Haar wavelet basis than the standard Fourier expansion. However, the Haar wavelet has poor localization properties in the frequency domain since it decays like a sinc function in powers of $1/\omega$. This is a consequence of the Heinsenberg uncertainty principle.

To represent a signal with the Haar wavelet basis, the translation and scaling operations associated with the mother wavelet need to be considered. Depicted in Fig. 119 and given by Eq. (13.5.15) is the wavelet $\psi_{1,0}(t)$. Thus its translation is zero and its scaling is unity. The concept in reconstructing a signal using the Haar wavelet basis is to consider decomposing the signal into more generic $\psi_{m,n}(t)$. By appropriate selection of the m and n, finer scales and appropriate locations of the signal can be extracted. For $a < 1$, the wavelet is a compressed version of $\psi_{1,0}$ whereas for $a > 1$, the wavelet is a dilated version of $\psi_{1,0}$. The scaling parameter a is typically taken to be a power of two so that $a = 2^j$ for some integer j. Figure 120 shows the compressed and dilated Haar wavelet for $a = 0.5$ and $a = 2$, i.e. $\psi_{1/2,0}$ and $\psi_{2,0}$. The compressed wavelet allows for finer scale resolution of a given signal while the dilated wavelet captures low-frequency components of a signal by having a broad range in time.

The simple Haar wavelet already illustrates all the fundamental principles of the wavelet concept. Specifically by using scaling and translation, a given signal or function can be represented by a basis of functions which allows for

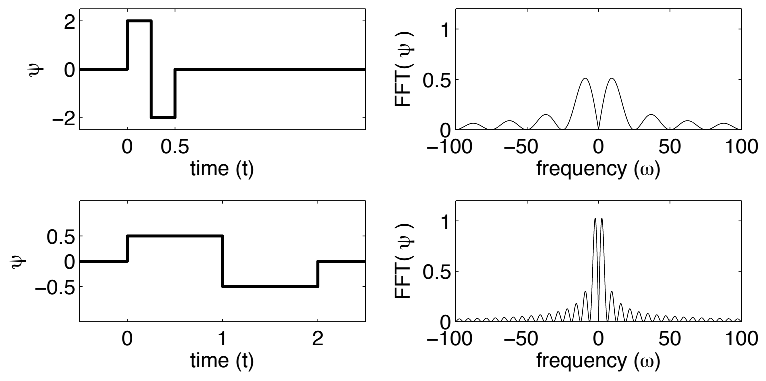


Figure 120: Illustration of the compression and dilation process of the Haar wavelet and its Fourier transform. In the top row, the compressed wavelet $\psi_{1/2,0}$ is shown. Improved time resolution is obtained at the expense of a broader frequency signature. The bottom row, shows the dilated wavelet $\psi_{2,0}$ which allows it to capture lower-frequency components of a signal.

higher and higher refinement in the time resolution of a signal. Thus it is much like the Gábor concept, except that now the time window is variable in order to capture different levels of resolution. Thus the large scale structures in time are captured with broad time-domain Haar wavelets. At this scale, the time resolution of the signal is very poor. However by successive rescaling in time, a finer and finer time resolution of the signal can be obtained along with its high-frequency components. The information at the low and high scales is all preserved so that a complete picture of the time-frequency domain can be constructed. Ultimately, the only limit in this process is the number of scaling levels to be considered.

The wavelet basis can be accessed via an integral transform of the form

$$(Tf)(\omega) = \int_t K(t,\omega) f(t) dt \tag{13.5.16}$$

where $K(t,\omega)$ is the kernel of the transform. This is equivalent in principle to the Fourier transform whose kernel are the oscillations given by $K(t,\omega) = \exp(-i\omega t)$. The key idea now is to define a transform which incorporates the mother wavelet as the kernel. Thus we define the *continuous wavelet transform (CWT):*

$$\mathcal{W}_\psi[f](a,b) = (f, \psi_{a,b}) = \int_{-\infty}^{\infty} f(t)\bar{\psi}_{a,b}(t) dt \tag{13.5.17}$$

Much like the windowed Fourier transform, the CWT is a function of the dilation parameter a and translation parameter b. Parenthetically, a wavelet is admissible if the following property holds:

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty \tag{13.5.18}$$

where the Fourier transform of the wavelet is defined by

$$\hat{\psi}_{a,b} = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} e^{-i\omega t} \psi\left(\frac{t-b}{a}\right) dt = \frac{1}{\sqrt{|a|}} e^{-ib\omega} \hat{\psi}(a\omega) \,. \tag{13.5.19}$$

Thus provided the admissibility condition (13.5.18) is satisfied, the wavelet transform can be well defined.

As an example of the admissibility condition, consider the Haar wavelet (13.5.15). Its Fourier transform can be easily computed in terms of the sinc-like function:

$$\hat{\psi}(\omega) = ie^{-i\omega/2} \frac{\sin^2(\omega/4)}{\omega/4} \,. \tag{13.5.20}$$

Thus the admissibility constant can be computed to be

$$\int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega = 16 \int_{-\infty}^{\infty} \frac{1}{|\omega|^3} \left|\sin\frac{\omega}{4}\right|^4 d\omega < \infty \,. \tag{13.5.21}$$

This then shows that the Haar wavelet is in the admissible class.

Another interesting property of the wavelet transform is the ability to construct new wavelet bases. The following theorem is of particular importance.

Theorem: *If ψ is a wavelet and ϕ is a bounded integrable function, then the convolution $\psi \star \phi$ is a wavelet.*

In fact, from the Haar wavelet (13.5.15) we can construct new wavelet functions by convolving with for instance

$$\phi(t) = \begin{cases} 0 & t < 0 \\ 1 & 0 \le t \le 1 \\ 0 & t \ge 1 \end{cases} \tag{13.5.22}$$

or the function

$$\phi(t) = e^{-t^2} \,. \tag{13.5.23}$$

The convolutions of these functions ϕ with the Haar wavelet ψ (13.5.15) are produced in Fig. 121. These convolutions could also be used as mother wavelets in constructing a decomposition of a given signal or function.

The wavelet transform principle is quite simple. First, the signal is split up into a bunch of smaller signals by translating the wavelet with the parameter b over the entire time domain of the signal. Second, the same signal is processed

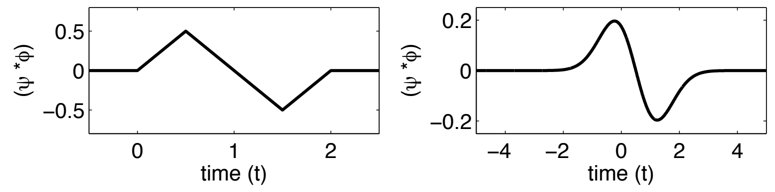


Figure 121: Convolution of the Haar wavelet with the functions (13.5.22) (left panel) and (13.5.23) (right panel). The convolved functions can be used as the mother wavelet for a wavelet basis expansion.

at different frequency bands, or resolutions, by scaling the wavelet window with the parameter a. The combination of translation and scaling allows for processing of the signals at different times and frequencies. Figure 121 is an upgrade of Fig. 116 that incorporates the wavelet transform concept in the time-frequency domain. In this figure, the standard time-series, Fourier transform and windowed Fourier transform are represented along with the multi-resolution concept of the wavelet transform. In particular, the box illustrating the wavelet transform shows the multi-resolution concept in action. Starting with a large Fourier domain window, the entire frequency content is extracted. The time window is then scaled in half, leading to finer time resolution at the expense of worse frequency resolution. This process is continued until a desired time-frequency resolution is obtained. This simple cartoon is critical for understanding wavelet application to time-frequency analysis.

Example: The Mexican Hat Wavelet. One of the more common wavelets is the Mexican hat wavelet. This wavelet is essentially a second moment of a Gaussian in the frequency domain. The definition of this wavelet and its transform are as follows:

$$\psi(t) = (1-t^2)e^{-t^2/2} = -d^2/dt^2\left(e^{-t^2/2}\right) = \psi_{1,0} \quad (13.5.24a)$$

$$\hat{\psi}(\omega) = \hat{\psi}_{1,0}(\omega) = \sqrt{2\pi}\omega^2 e^{-\omega^2/2} \quad (13.5.24b)$$

The Mexican hat wavelet has excellent localization properties in both time and frequency due to the minimal time-bandwidth product of the Gaussian function. Figure 123 (top panels) shows the basic Mexican wavelet function $\psi_{1,0}$ and its Fourier transform, both of which decay in t (ω) like $\exp(-t^2)$ ($\exp(-\omega^2)$). The Mexican hat wavelet can be dilated and translated easily as is depicted in Fig. 123 (bottom panel). Here three wavelets are depicted: $\psi_{1,0}$, $\psi_{3/2,-3}$ and $\psi_{1/4,6}$. This shows both the scaling and translation properties associated with any wavelet function.

To finish the initial discussion of wavelets, some of the various properties of

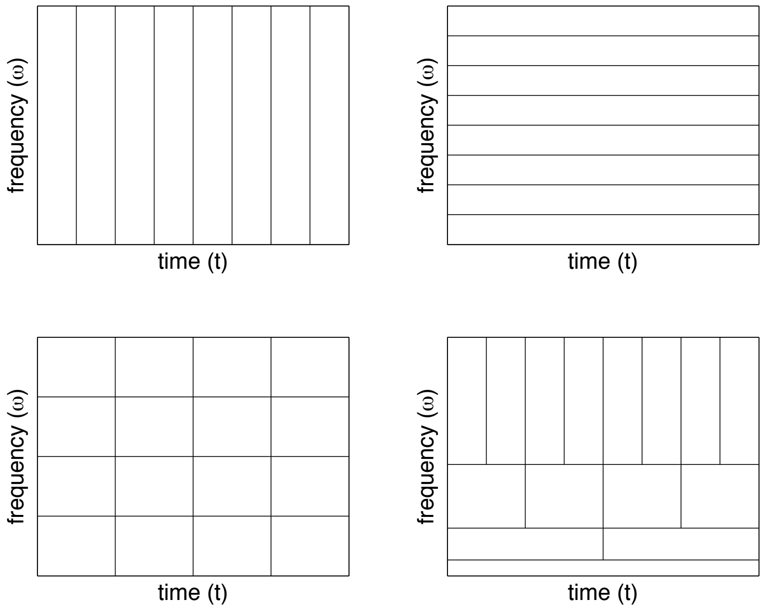


Figure 122: Graphical depiction of the difference between a time series analysis, Fourier analysis, Gábor analysis and wavelet analysis of a signal. This figure is identical to Fig. 116 but with the inclusion of the time-frequency resolution achieved with the wavelet transform. The wavelet transform starts with a large Fourier domain window so that the entire frequency content is extracted. The time window is then scaled in half, leading to finer time resolution at the expense of worse frequency resolution. This process is continued until a desired time-frequency resolution is obtained.

the wavelets are listed. To begin, consider the time-frequency resolution and its localization around a given time and frequency. These quantities can be calculated from the relations:

$$\sigma_t^2 = \int_{-\infty}^{\infty} (t- < t >)^2 |\psi(t)|^2 dt \tag{13.5.25a}$$

$$\sigma_\omega^2 = \frac{1}{2\pi} \int_{-\infty}^{\infty} (\omega- < \omega >)^2 |\hat{\psi}(\omega)|^2 d\omega \tag{13.5.25b}$$

where the variances measure the spread of the time and frequency signal around $< t >$ and $< \omega >$ respectively. The Heisenberg uncertainty constrains the localization of time and frequency by the relation $\sigma_t^2 \sigma_\omega^2 \geq 1/2$. In addition, the CWT has the following mathematical properties

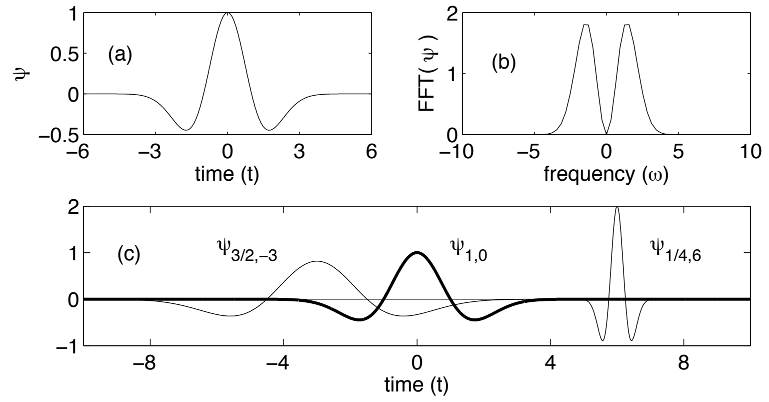


Figure 123: Illustration of the Mexican hat wavelet $\psi_{1,0}$ (top left panel), its Fourier transform $\hat{\psi}_{1,0}$ (top right panel), and two additional dilations and translations of the basic $\psi_{1,0}$ wavelet. Namely the $\psi_{3/2,-3}$ and $\psi_{1/4,6}$ are shown (bottom panel).

1. **Linearity:** the transform is linear so that

$$\mathcal{W}_\psi(\alpha f + \beta g)(a,b) = \alpha \mathcal{W}_\psi(f)(a,b) + \beta \mathcal{W}_\psi(g)(a,b)$$

2. **Translation:** the transform has the translation property

$$\mathcal{W}_\psi(T_c f)(a,b) = \mathcal{W}_\psi(f)(a, b-c)$$

where $T_c f(t) = f(t-c)$.

3. **Dilation:** the dilation property follows

$$\mathcal{W}_\psi(D_c f)(a,b) = \frac{1}{\sqrt{c}} \mathcal{W}_\psi(f)(a/c, b/c)$$

where $c > 0$ and $D_c f(t) = (1/c) f(t/c)$.

4. **Inversion:** the transform can be inverted with the definition

$$f(t) = \frac{1}{C_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathcal{W}_\psi(f)(a,b) \psi_{a,b}(t) \frac{dbda}{a^2}$$

where it becomes clear why the admissibility condition $C_\psi < \infty$ is needed.

To conclude this section, consider the idea of discretizing the wavelet transform on a computational grid. Thus the transform is defined on a lattice so that

$$\psi_{m,n}(x) = a_0^{-m/2} \psi\left(a_0^{-m} x - n b_0\right) \tag{13.5.26}$$

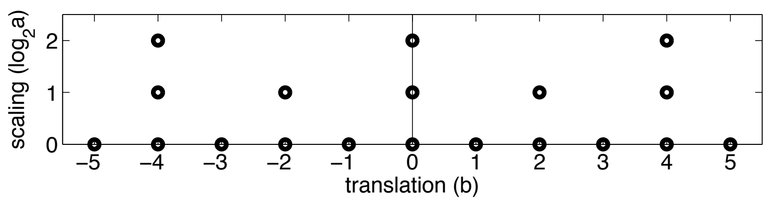


Figure 124: Discretization of the discrete wavelet transform with $a_0 = 2$ and $b_0 = 1$. This figure is a more formal depiction of the multi-resolution discretization as shown in Fig. 122.

where $a_0, b_0 > 0$ and m, n are integers. The *discrete wavelet transform* is then defined by

$$\begin{aligned} \mathcal{W}_\psi(f)(m,n) &= (f, \psi_{m,n}) \\ &= \int_{-\infty}^{\infty} f(t)\bar{\psi}_{m,n}(t)dt \\ &= a_0^{-m/2} \int_{-\infty}^{\infty} f(t)\bar{\psi}(a_0^{-m}t - nb_0)dt\,. \end{aligned} \qquad (13.5.27)$$

Futhermore, if $\psi_{m,n}$ are complete, then a given signal or function can be expanded in the wavelet basis:

$$f(t) = \sum_{m,n=-\infty}^{\infty} (f, \psi_{m,n})\psi_{m,n}(t)\,. \qquad (13.5.28)$$

This expansion is in a set of wavelet frames. It still needs to be determined if the expansion is in terms of a set of basis functions. It should be noted that the scaling and dilation parameters are typically taken to be $a_0 = 2$ and $b_0 = 1$, corresponding to dilations of 2^{-m} and translations of $n2^m$. Figure 124 gives a graphical depiction of the time-frequency discretization of the wavelet transform. This figure is especially relevant for the computational evaluation of the wavelet transform. Further, it is the basis of fast algorithms for multi-resolution analysis.

13.6 Multi-Resolution Analysis and the Wavelet Basis

Before proceeding forward with wavelets, it is important to establish some key mathematical properties. Indeed, the most important issue to resolve is the ability of the wavelet to actually represent a given signal or function. In Fourier analysis, it has been established that any generic function can be represented by a series of cosines and sines. Something similar is needed for wavelets in order to make them a useful tool for the analysis of time-frequency signals.

The concept of a wavelet is simple and intuitive: construct a signal using a single function $\psi \in L^2$ which can be written $\psi_{m,n}$ and that represents binary dilations by 2^m and translations of $n2^{-m}$ so that

$$\psi_{m,n} = 2^{m/2}\psi\left(2^m(x - n/2^m)\right) = 2^{m/2}\psi(2^m x - n) \tag{13.6.29}$$

where m and n are integers. The use of this wavelet for representing a given signal or function is simple enough. However, there is a critical issue to be resolved concerning the *orthogonality* of the functions $\psi_{m,n}$. Ultimately, this is the primary issue which must be addressed in order to consider the wavelets as basis functions in an expansion. Thus we define the orthogonality condition as

$$(\psi_{m,n}, \psi_{k,l}) = \int_{-\infty}^{\infty} \psi_{m,n}(x)\psi_{k,l}(x)dx = \delta_{m,k}\delta_{n,l} \tag{13.6.30}$$

where δ_{ij} is the Dirac delta defined by

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \tag{13.6.31}$$

where i, j are integers. This statement of orthogonality is generic, and it holds in most function spaces with a defined inner product.

The importance of orthogonality should not be underestimated. It is very important in applications where a functional expansion is used to approximate a given function or solution. In what follows, two examples are given concerning the key role of orthogonality.

Fourier expansions. Consider representing an even function $f(x)$ over the domain $x \in [0, L]$ with a cosine expansion basis. By Fourier theory, the function can be represented by

$$f(x) = \sum_{n=0}^{\infty} a_n \cos\frac{n\pi x}{L} \tag{13.6.32}$$

where the coefficients a_n can be constructed by using inner product rules and orthogonality. Specifically, by multiplying both sides of the equation by $\cos(m\pi x/L)$ and integrating over $x \in [0, L]$, i.e. taking the inner product with respect to $\cos(m\pi x/L)$, the following result is found:

$$(f, \cos m\pi x/L) = \sum_{n=0}^{\infty} a_n(\cos n\pi x/L, \cos m\pi x/L)\,. \tag{13.6.33}$$

This is where orthogonality plays a key role, the infinite sum on the right hand side can be reduced to a single index where $n = m$ since the cosines are orthogonal to each other

$$(\cos n\pi x/L, \cos m\pi x/L) = \begin{cases} 0 & n \neq m \\ L & n = m \end{cases}\,. \tag{13.6.34}$$

Thus the coefficients can be computed to be

$$a_n = \frac{1}{L}\int_0^L f(x)\cos\frac{n\pi x}{L}dx \tag{13.6.35}$$

and the expansion is accomplished. Moreover, the cosine basis is complete for even functions and any signal or function $f(x)$ can be represented, i.e. as $n \to \infty$ in the sum, the expansion converges to the given signal $f(x)$.

Eigenfunction expansions: The cosine expansion is a subset of the more general eigenfunction expansion technique that is often used to solve differential and partial differential equations problems. Consider the nonhomogeneous boundary value problem

$$Lu = f(x) \tag{13.6.36}$$

where L is a given self-adjoint, linear operator. This problem can be solved with an eigenfunction expansion technique by considering the associated eigenvalue problem of the operator L:

$$Lu_n = \lambda_n u_n\,. \tag{13.6.37}$$

The solution of (13.6.36) can then be expressed as

$$u(x) = \sum_{n=0}^{\infty} a_n u_n \tag{13.6.38}$$

provided the coefficients a_n can be determined. Plugging in this solution to (13.6.36) yields the following calculations

$$\begin{aligned}
&Lu = f\\
&L(\sum a_n u_n) = f\\
&\sum a_n L u_n = f\\
&\sum a_n \lambda_n u_n = f\,.
\end{aligned} \tag{13.6.39}$$

Taking the inner product of both sides with respect to u_m yields

$$\begin{aligned}
&(\sum a_n \lambda_n u_n, u_m) = (f, u_m)\\
&\sum a_n \lambda_n (u_n, u_m) = (f, u_m)\\
&a_m \lambda_m = (f, u_m)
\end{aligned} \tag{13.6.40}$$

where the last line is achieved by orthogonality of the eigenfunctions $(u_n, u_m) = \delta_{n,m}$. This then gives $a_m = (f, u_m)/\lambda_m$ and the eigenfunction expansion solution is

$$u(x) = \sum_{n=0}^{\infty} \frac{(f, u_n)}{\lambda_n} u_n\,. \tag{13.6.41}$$

Provided the u_n are a complete set, this expansion is guaranteed to converge to $u(x)$ as $n \to \infty$.

Orthonormal wavelets

The preceding examples highlight the importance of orthogonality for representing a given function. A wavelet ψ is called orthogonal if the family of functions $\psi_{m,n}$ are orthogonal as given by Eq. (13.6.30). In this case, a given signal or function can be uniquely expressed with the doubly infinite series

$$f(t) = \sum_{n,m=-\infty}^{\infty} c_{m,n}\psi_{m,n}(t) \tag{13.6.42}$$

where the coefficients are given from orthogonality by

$$c_{m,n} = (f, \psi_{m,n}) \, . \tag{13.6.43}$$

The series is guaranteed to converge to $f(t)$ in the L^2 norm.

The above result based upon orthogonal wavelets establishes the key mathematical framework needed for using wavelets in a very broad and general way. It is this result that allows us to think of wavelets philosophically as the same as the Fourier transform.

Multi-Resolution Analysis (MRA)

The power of the wavelet basis is its ability to take a function or signal $f(t)$ and express it as a limit of successive approximations, each of which is a finer and finer version of the function in time. These successive approximations correspond to different resolution levels.

A *multi-resolution analysis*, commonly referred to as an MRA, is a method that gives a formal approach to constructing the signal with different resolution levels. Mathematically, this involves a sequence

$$\{V_m : \;\; m \in \text{integers}\} \tag{13.6.44}$$

of embedded subspaces of L^2 that satisfies the following relations:

1. The subspaces can be embedded in each other so that
$$Course \;\; \cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \cdots V_m \subset V_{m+1} \cdots \;\; Fine$$
2. The union of all the embedded subspaces spans the entire L^2 space so that
$$\cup_{m=-\infty}^{\infty} V_m$$
is dense in L^2
3. The intersection of subspaces is the null set so that
$$\cap_{m=-\infty}^{\infty} V_m = \{0\}$$

4. Each subspace picks up a given resolution so that $f(x) \in V_m$ if and only if $f(2x) \in V_{m+1}$ for all integers m.

5. There exists a function $\phi \in V_0$ such that

$$\{\phi_{0,n} = \phi(x-n)\}$$

is an orthogonal basis for V_0 so that

$$\|f\|^2 = \int_{-\infty}^{\infty} |f(x)|^2 dx = \sum_{-\infty}^{\infty} |(f, \phi_{0,n})|^2$$

The function ϕ is called the *scaling function* or *father wavelet.*

If $\{V_m\}$ is a multiresolution of L^2 and if V_0 is the closed subspace generated by the integer translates of a single function ϕ, then we say ϕ generates the MRA.

One remark of importance: since $V_0 \subset V_1$ and ϕ is a scaling function for V_0 and also for V_1, then

$$\phi(x) = \sum_{-\infty}^{\infty} c_n \phi_{1,n}(x) = \sqrt{2} \sum_{-\infty}^{\infty} c_n \phi(2x-n) \tag{13.6.45}$$

where $c_n = (\phi, \phi_{1,n})$ and $\sum_{-\infty}^{\infty} |c_n|^2 = 1$. This equation, which relates the scaling function as a function of x and $2x$ is known as the *dilation equation*, or *two-scale equation*, or *refinement equation* because it reflects $\phi(x)$ in the refined space V_1 which as the finer scale of 2^{-1}.

Since $V_m \subset V_{m+1}$, we can define the orthogonal complement of V_m in V_{m+1} as

$$V_{m+1} = V_m \oplus W_m \tag{13.6.46}$$

where $V_m \perp W_m$. This can be generalized so that

$$\begin{aligned} V_{m+1} &= V_m \oplus W_m \\ &= (V_{m-1} \oplus W_{m-1}) \oplus W_m \\ &\vdots \\ &= V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_m \\ &= V_0 \oplus (\oplus_{n=0}^{m} W_n) \; . \end{aligned} \tag{13.6.47}$$

As $m \to \infty$, it can be found that

$$V_0 \oplus (\oplus_{n=0}^{\infty} W_n) = L^2 \, . \tag{13.6.48}$$

In a similar fashion, the resolution can rescale upwards so that

$$\oplus_{n=-\infty}^{\infty} W_n = L^2 \, . \tag{13.6.49}$$

Moreover, there exists a scaling function $\psi \in W_0$ (the mother wavelet) such that

$$\psi_{0,n}(x) = \psi(x-n) \tag{13.6.50}$$

constitutes an orthogonal basis for W_0 and

$$\psi_{m,n}(x) = 2^{m/2}\psi(2^m x - n) \tag{13.6.51}$$

is an orthogonal basis for W_m. Thus the mother wavelet ψ spans the orthogonal complement subset W_m while the scaling function ϕ spans the subsets V_m. The connection between the father and mother wavelet are shown in the following theorem.

Theorem: If $\{V_m\}$ is a MRA with scaling function ϕ, then there is a mother wavelet ψ

$$\psi(x) = \sqrt{2}\sum_{-\infty}^{\infty}(-1)^{n-1}\bar{c}_{-n-1}\phi(2x-n) \tag{13.6.52}$$

where

$$c_n = (\phi, \phi_{1,n}) = \sqrt{2}\int_{-\infty}^{\infty}\phi(x)\bar{\phi}(2x-1)dx\,. \tag{13.6.53}$$

That is, the system $\psi_{m,n}(x)$ is an orthogonal basis of L^2.

This theorem is critical for what we would like to do. Namely, use the wavelet basis functions as a complete expansion basis for a given function $f(x)$ in L^2. Further, it explicitly states the connection between the *scaling function* $\phi(x)$ (father wavelet) and *wavelet function* $\psi(x)$ (mother wavelet). It is only left to construct a desirable wavelet basis to use. As for wavelet construction, the idea is to build them to take advantage of certain properties of the system so that it gives an efficient and meaningful representation of your time-frequency data.

13.7 Spectrograms and the Gábor transforms in MATLAB

The aim of this lecture will be to use MATLAB's fast Fourier transform routines modified to handle the Gábor transform. The Gábor transform allows for a fast and easy way to analyze both the time and frequency properties of a given signal. Indeed, this windowed Fourier transform method is used extensively for analyzing speech and vocalization patterns. For such applications, it is typical to produce a *spectrogram* that represents the signal in both the time and frequency domain. Figures 125 and 126 are produced from the vocalization patterns in time-frequency of a human saying "do re mi" and a humpback whale vocalizing to other whales. The time-frequency analysis can can be used to produce speech

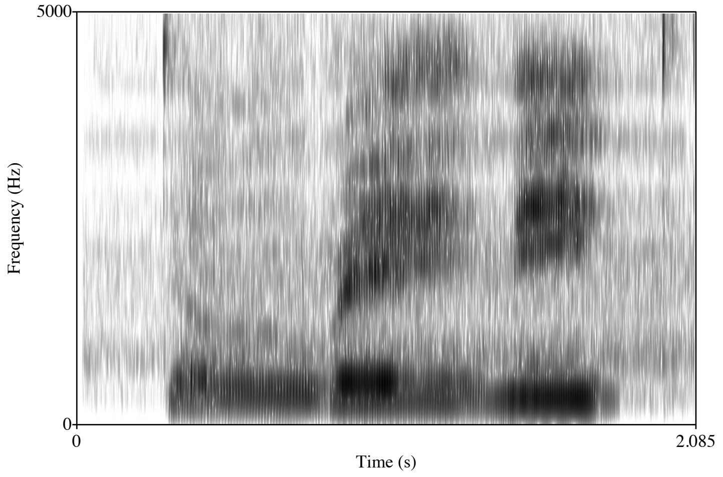


Figure 125: Spectrogram (time-frequency) analysis of a male human saying "do re mi." The spectrogram is created with the software program Praat, which is an open source code for analyzing phonetics.

recognition algorithms given the characteristic signatures in the time-frequency domains of sounds. Thus spectrograms are a sort of fingerprint of sound.

To understand the algorithms which produce the spectrogram, it is informative to return to the characteristic picture shown in Fig. 115. This demonstrates the action of an applied time filter in extracting time localization information. To build a specific example, consider the following MATLAB code that builds a time domain (t), its corresponding Fourier domain (ω), a relatively complicated signal ($S(t)$), and its Fourier transform ($\hat{S}(\omega)$).

```
  clear all; close all; clc

  L=10; n=2048;
  t2=linspace(0,L,n+1); t=t2(1:n);
  k=(2*pi/L)*[0:n/2-1 -n/2:-1]; ks=fftshift(k);

  S=(3*sin(2*t)+0.5*tanh(0.5*(t-3))+ 0.2*exp(-(t-4).^2)...
    +1.5*sin(5*t)+4*cos(3*(t-6).^2))/10+(t/20).^3;
  St=fft(S);
```

The signal and its Fourier tranform can be plotted with the commands

```
figure(1)
subplot(3,1,1)  % Time domain
```

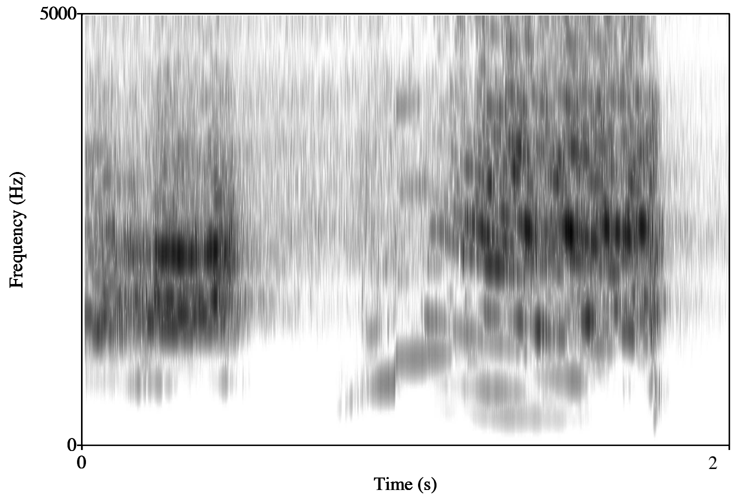


Figure 126: Spectrogram (time-frequency) analysis of a humpback whale vocalization over a short period of time. The spectrogram is created with the software program Praat, which is an open source code for analyzing phonetics.

```
plot(t,S,'k')
set(gca,'Fontsize',[14]),
xlabel('Time (t)'), ylabel('S(t)')

subplot(3,1,2)  % Fourier domain
plot(ks,abs(fftshift(St))/max(abs(St)),'k');
axis([-50 50 0 1])
set(gca,'Fontsize',[14])
xlabel('frequency (\omega)'), ylabel('FFT(S)')
```

Figure 127 shows the signal and its Fourier transform for the above example. This signal $S(t)$ will be analyzed using the Gábor transform method.

The simplest Gábor window to implement is a Gaussian time-filter centered at some time τ with width a. As has been demonstrated, the parameter a is critical for determining the level of time-resolution versus frequency resolution in a time-frequency plot. Figure 128 shows the signal under consideration with three filter widths. The narrower the time-filtering, the better resolution in time. However, this also produces the worst resolution in frequency. Conversely, a

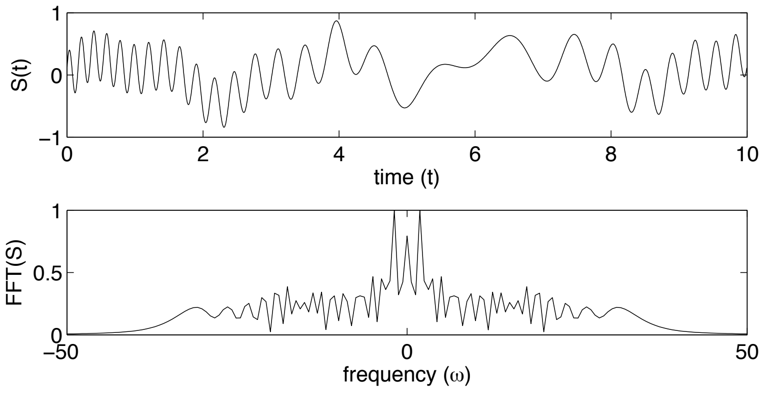


Figure 127: Time signal and its Fourier transform considered for a time-frequency analysis in what follows.

wide window in time produces much better frequency resolution at the expense of reducing the time resolution. A simple extension to the existing code produces a signal plot along with three different filter widths of Gaussian shape.

```
figure(2)
width=[10 1 0.2];
for j=1:3
  g=exp(-width(j)*(t-4).^2);
  subplot(3,1,j)
  plot(t,S,'k'), hold on
  plot(t,g,'k','Linewidth',[2])
  set(gca,'Fontsize',[14])
  ylabel('S(t), g(t)')
end
xlabel('time (t)')
```

The key now for the Gábor transform is to multiply the time filter Gábor function $g(t)$ with the original signal $S(t)$ in order to produce a windowed section of the signal. The Fourier transform of the windowed section then gives the local frequency content in time. The following code constructs the windowed Fourier transform with the Gábor filtering function

$$g(t) = e^{-a(t-b)^2}\,. \tag{13.7.54}$$

The Gaussian filtering has a width parameter a and translation parameter b. The following code constructs the windowed Fourier transform using the Gaussian with $a = 2$ and $b = 4$.

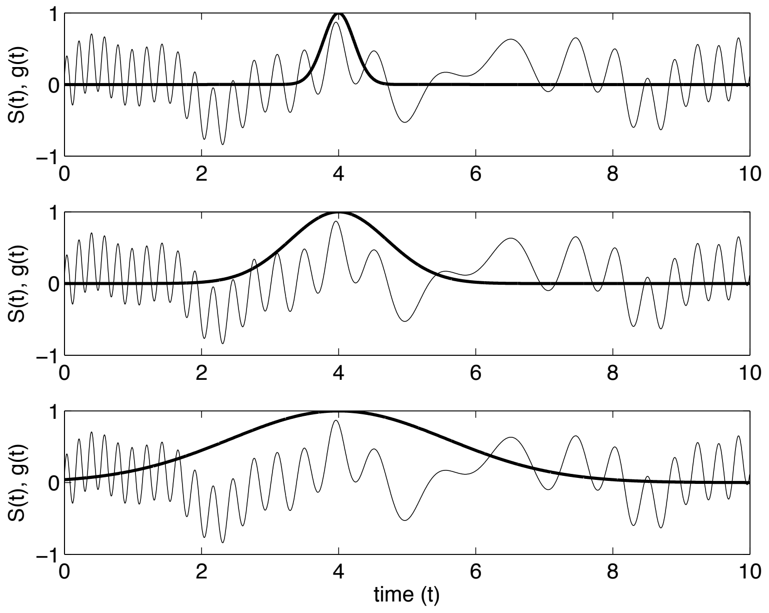


Figure 128: Time signal $S(t)$ and the Gábor time filter $g(t)$ (bold lines) for three different Gaussian filters: $g(t) = \exp(-10(x-4)^2)$ (top), $g(t) = \exp(-(x-4)^2)$ (middle), and $g(t) = \exp(-0.2(x-4)^2)$ (bottom). The different filter widths determine the time-frequency resolution. Better time resolution gives worse frequency resolution and vice-versa due to the Heinsenberg uncertainty principle.

```
figure(3)
g=exp(-2*(t-4).^2);
Sg=g.*S;
Sgt=fft(Sg);

subplot(3,1,1), plot(t,S,'k'), hold on
  plot(t,g,'k','Linewidth',[2])
  set(gca,'Fontsize',[14])
  ylabel('S(t), g(t)'), xlabel('time (t)')
subplot(3,1,2), plot(t,Sg,'k')
  set(gca,'Fontsize',[14])
  ylabel('S(t)g(t)'), xlabel('time (t)')
subplot(3,1,3), plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)),'k')
  axis([-50 50 0 1])
  set(gca,'Fontsize',[14])
  ylabel('FFT(Sg)'), xlabel('frequency (\omega)')
```

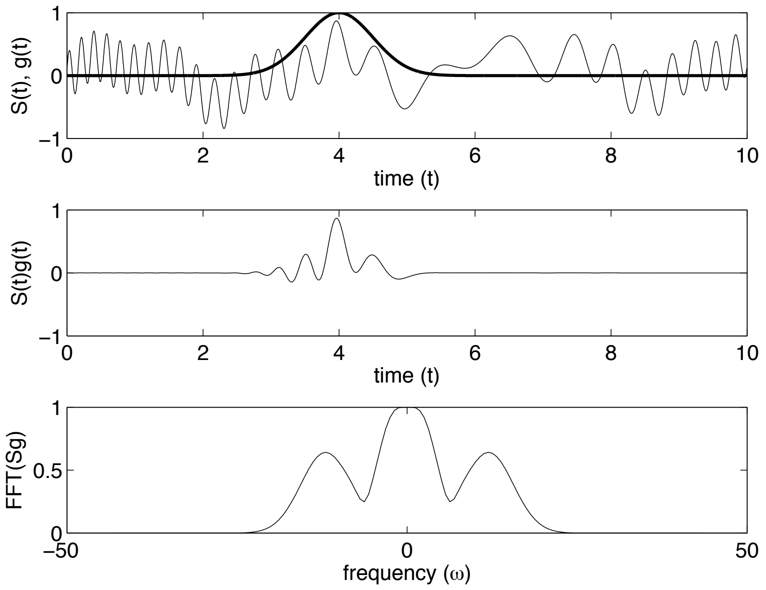


Figure 129: Time signal $S(t)$ and the Gábor time filter $g(t) = \exp(-2(x-4)^2)$ (bold line) for a Gaussian filter. The product $S(t)g(t)$ is depicted in the middle panel and its Fourier transform $\hat{Sg}(\omega)$ is depicted in the bottom panel. Note that the windowing of the Fourier transform can severely limit the detection of low-frequency components.

Figure 129 demonstrates the application of this code and the windowed Fourier transform in extracting local frequencies of a local time window.

The key to generating a spectrogram is to now vary the position b of the time filter and produce spectra at each location in time. In theory, the parameter b is continuously translated to produce the time-frequency picture. In practice, like everything else, the parameter b is discretized. The level of discretization is important in establishing a good time-frequency analysis. Specifically, finer resolution will produce better results. The following code makes a dynamical *movie* of this process as the parameter b is translated.

```
figure(4)
Sgt_spec=[];
tslide=0:0.1:10
for j=1:length(tslide)
  g=exp(-2*(t-tslide(j)).^2); % Gabor
  Sg=g.*S; Sgt=fft(Sg);
  Sgt_spec=[Sgt_spec; abs(fftshift(Sgt))];
  subplot(3,1,1), plot(t,S,'k',t,g,'r')
```

```
    subplot(3,1,2), plot(t,Sg,'k')
    subplot(3,1,3), plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)))
    axis([-50 50 0 1])
    drawnow
    pause(0.1)
  end
```

This movie is particularly illustrative and provides an excellent graphical representation of how the Gábor time-filtering extracts both local time information and local frequency content. It also illustrates, as the parameter a is adjusted, the ability (or inability) of the windowed Fourier transform to provide accurate time and frequency information.

The code just developed also produces a matrix **Sgt_spec** which contains the Fourier transform at each slice in time of the parameter b. It is this matrix that produces the spectrogram of the time-frequency signal. The spectrogram can be viewed with the commands

```
  pcolor(tslide,ks,Sgt_spec.'), shading interp
  set(gca,'Ylim',[-50 50],'Fontsize',[14])
  colormap(hot)
```

Modifying the code slightly, a spectrogram of the signal $S(t)$ can be made for three different filter widths $a = 5, 1, 0.2$ in Eq. (13.7.54). The spectrograms are shown in Fig. 130 where from left to right the filtering window is broadened from $a = 5$ to $a = 0.2$. Note that for the left plot, strong localization of the signal in time is achieved at the expense of suppressing almost all the low-frequency components of the signal. In contrast, the right most figure with a wide temporal filter preserves excellent resolution of the Fourier domain but fails to localize signals in time. Such are the tradeoffs associated with a fixed Gábor window transform.

13.8 MATLAB Filter Design and Wavelet Toolboxes

The applications of filtering and time-frequency analysis are so ubiquitous across the sciences, that MATLAB has developed a suite of toolboxes that specialize in these applications. Two of these toolboxes will be demonstrated in what follows. Primarily, screenshots will give hints of the functionality and versatility of the toolboxes.

The most remarkable part of the toolbox is that it allows for entry into high level signal processing and wavelet processing almost immediately. Indeed, one hardly needs to know anything to begin the process of analyzing, synthesizing, and manipulating data to one's own ends. For the most part, each of the toolboxes allows you to begin usage once you upload your signal, image or data. The only drawback is cost. For the most part, many academic departments have access to the toolboxes. And if you are part of a university environment,

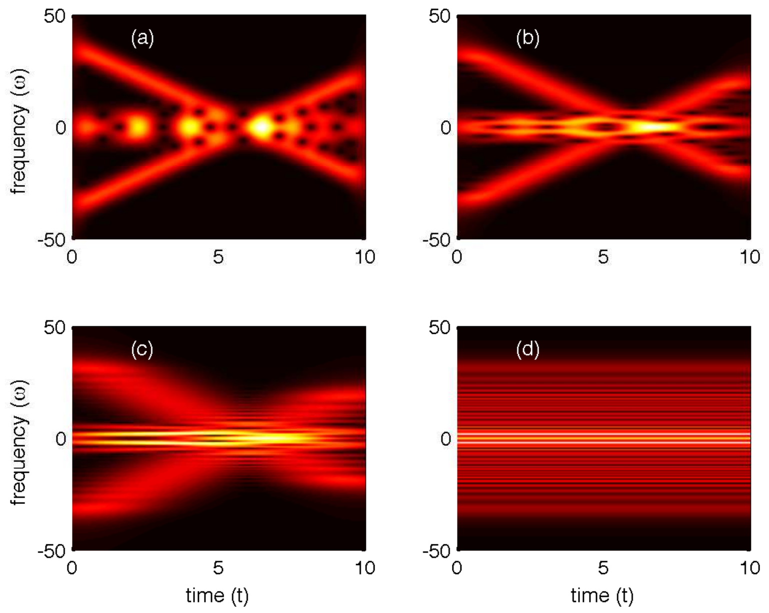


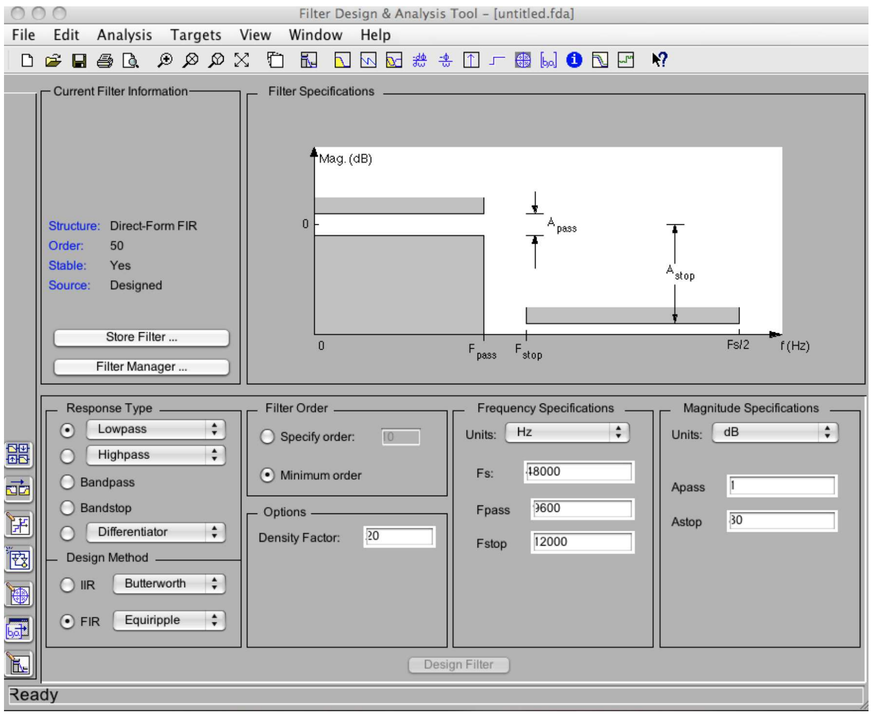
Figure 130: Spectrograms produced from the Gábor time-filtering Eq. (13.7.54) with (a) $a = 5$, (b) $a = 1$ and (c) $a = 0.2$. The Fourier transform, which has no time localization information is depicted in (d). It is easily seen that the window width trades off time and frequency resolution at the expense of each other. Regardless, the spectrogram gives a visual time-frequency picture of a given signal.

or enrolled in classes, the student versions of the toolboxes can be purchased directly from MATLAB at a very reasonable price.

Filter Design and Analysis Toolbox

The filter design toolbox is ideally suited to help create and engineer an ideal filter for whatever application you may need. The user has the option of using command line codes to directly access the filter design subroutines, or one can use the extremely efficient and useful GUI (graphical user interface) from MATLAB. Figure 131 shows a typical screenshot that MATLAB produces upon startup of the filter design toolbox. The command to start the toolbox is given by

```
FDAtool
```



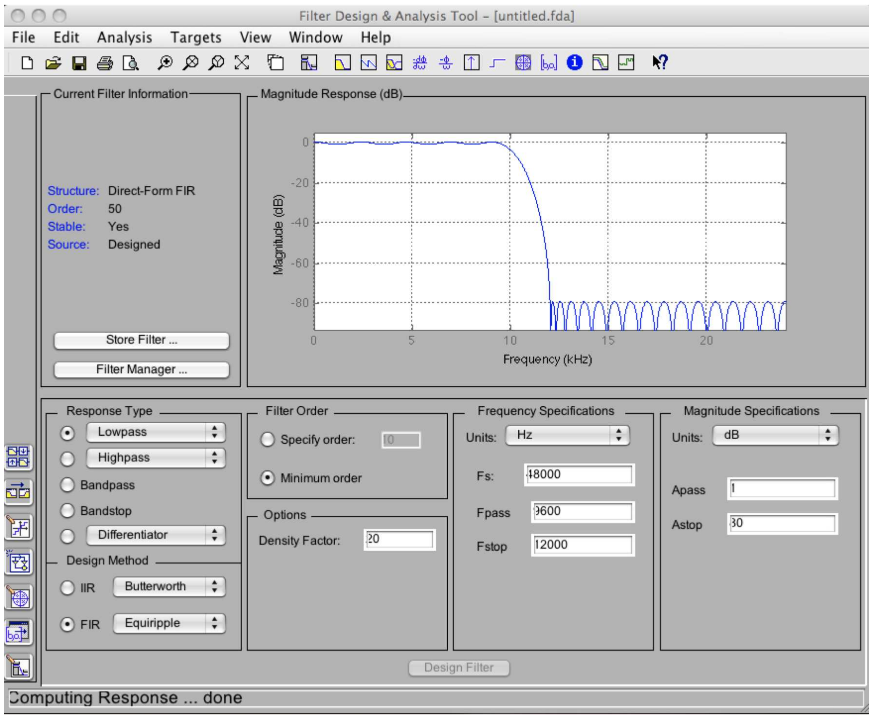
Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: ● Lowpass
Design Method: ● FIR Equiripple
Filter Order: Specify order: 10 [✓ Minimum order]
Density Factor: 20
Units: Hz
Fs: 48000
Fpass: 9600
Fstop: 12000
Units: dB
Apass: 1
Astop: 80

Figure 131: Screenshot of the filter design and analysis toolbox GUI. The top left contains information on the current filter design while the top right is a graphical representation of the filter characteristics. The lower portion of the GUI is reserved for the user interface functionality and design. Indeed, the type of filter, its spectral width, whether it is low-pass (filters high frequencies), high-pass (filters low frequencies), band-pass (filters a specified band of frequencies) or band-stop (attenuates a band of frequencies) can be chosen.

This launches the highly intuitive MATLAB GUI that guides you through the process of designing your ideal filter. Figure 132 demonstrates, for instance, the magnitude response of the filter along with its frequency and magnitude of response. The magnitude response can be accessed by dragging down the **Analysis** button at the top of the GUI panel. In addition to the amplitude response, the phase response, a combination of phase and amplitude response, group-velocity response, etc. Depending upon your needs, complete engineering of the filter can be performed with set objectives and performance aims.

As with all toolbox design and information, it can be exported from MATLAB for use in other programs or with other parts of MATLAB. The exporting of data, as well as saving of a given filter design, can all be performed from the **file** drag-down menu at the top right of the GUI. Moreover, if you desire to skip


Filter Design & Analysis Tool – [untitled.fda]
File Edit Analysis Targets View Window Help
Current Filter Information
Magnitude Response (dB)
Magnitude (dB)
Frequency (kHz)
Store Filter ...
Filter Manager ...
Design Filter

Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: IIR Butterworth [✓ FIR Equiripple]
Filter Order: Specify order: 10 [✓ Minimum order]
Options: Density Factor: 20
Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 9600; Fstop: 12000
Magnitude Specifications: Units: dB; Apass: 1; Astop: 80

Computing Response ... done

Figure 132: Screenshot of the magnitude response of the filter. The magnitude response, along with a host of other important characteristics of the filter can be obtained from the drag-down menu of the **Analysis** button on the top.

the filter design GUI, then all the filter functions can be accessed directly from the command line window. For instance, the command

```
[B,A]=butter(N,Wn)
```

generates a Butterworth digital and analog filter design. The parameter N denotes the Nth order lowpass digital Butterworth filter and returns the filter coefficients in length $N+1$ vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cutoff frequency Wn must be $0.0 < Wn < 1.0$, with 1.0 corresponding to half the sample rate. And besides this, you now know the butter command in MATLAB.

Wavelet Toolbox

Just like the signal processing, filter design toolbox, the wavelet toolbox is an immensely powerful toolbox for signal processing, image processing, multi-resolution analysis and any manner of time-frequency analysis. The wavelet

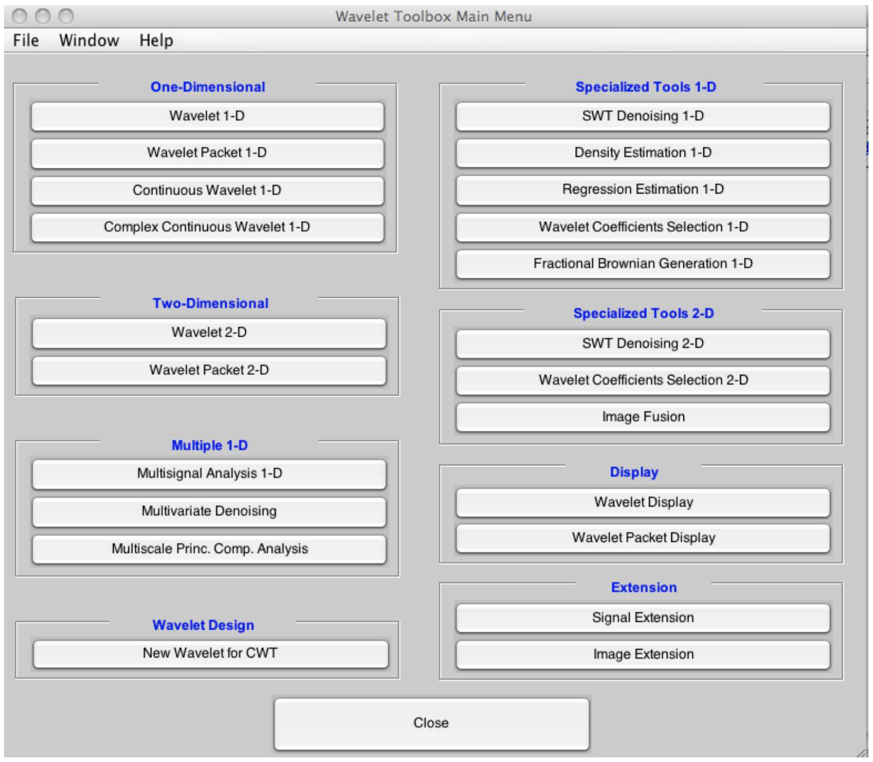


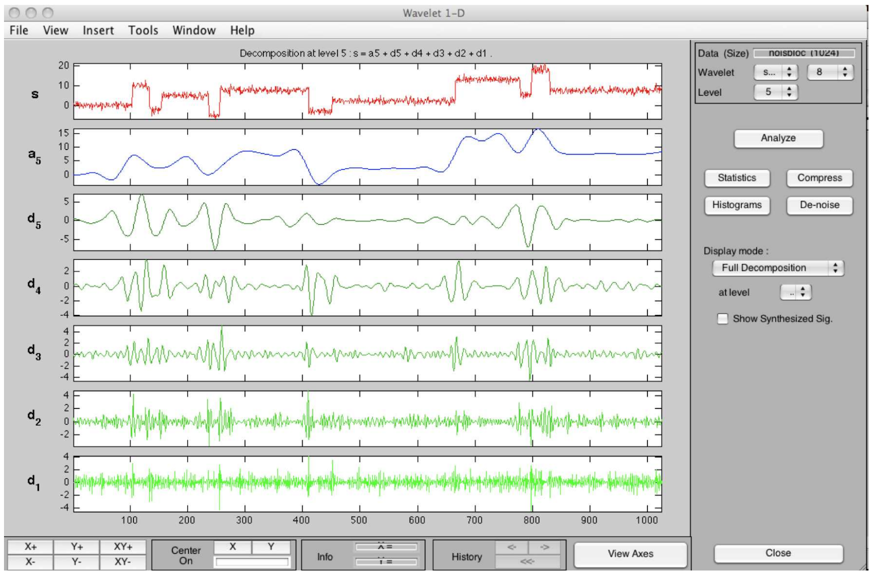
Figure 133: Screenshot of the entry point of the wavelet toolbox. A myriad of applications are available by simply clicking on one of the menu items. Both 1D and 2D problems can be analyzed with the toolbox..

toolbox is comprised of a large number of modules that allow for various applications and methodologies. To access the toolbox, simply type

```
wavemenu
```

and a selection of menu items is generated. Figure 133 demonstrates the assortment of possible uses of the wavelet toolbox that are available upon launching the wavelet toolbox. In addition to such applications as signal processing and time frequency analysis of one-dimensional signals or functions, one can proceed to denoise or manipulate image data. The program can also produce wavelet transforms and output the coefficients of a continuous or discrete wavelet transform.

As a specific example, consider the *wavelet 1-D* button and the example files it contains. The example files are accessed from the **file** button at the top left of the GUI. In this case, an example analysis of a noisy step function



Decomposition at level 5 : s = a5 + d5 + d4 + d3 + d2 + d1 .

Data (Size) noisbloc (1024)
Wavelet s... 8
Level 5
Display mode : Full Decomposition
Show Synthesized Sig.

Figure 134: Screenshot of the wavelet 1-D with a noisy step function signal. A multi-resolution decomposition of the signal is performed with *sym* wavelets.

signal is considered. It should be noted that your own signal could have easily been imported from the **file** button with the **load** menu item. In this example, a signal is provided with the full wavelet decomposition at different levels of resolution. It is easy to see that the given signal is decomposed into its *big* features followed by its *finer* features as one descends vertically down the plots. In the standard example, the *sym* wavelet is used with five levels of resolution. This shows the progression, as discussed in the wavelets lecture, of the multi-resolution analysis. The wavelet analysis can be easily modified to consider other wavelet bases. In fact, in addition to the sym wavelet, one can choose the Haar wavelets, Daubechies wavelets, and Coifman wavelets among others. This remarkable packaging of the best time-frequency domain methods is one of the tremendous aspects of the toolbox. In addition to the decomposition, a histogram, the statistics, or compression of the signal can be performed by simply clicking one of the appropriate buttons on the right.

The wavelet toolbox also allows for considering a signal with a wavelet expansion. The continuous wavelet 1-D provides a full decomposition of the signal into its wavelet basis. Figure 136 shows a given signal along with the calculation of the wavelet coefficients $C_{a,b}$ where a and b are dilation and translation operations respectively. This provides the basis for a time-frequency analysis. Additionally, given a chosen level of resolution determined by the parameter a, then the third panel shows the parameter $C_{a,b}$ for $a = a_{\text{max}}/2$. The local

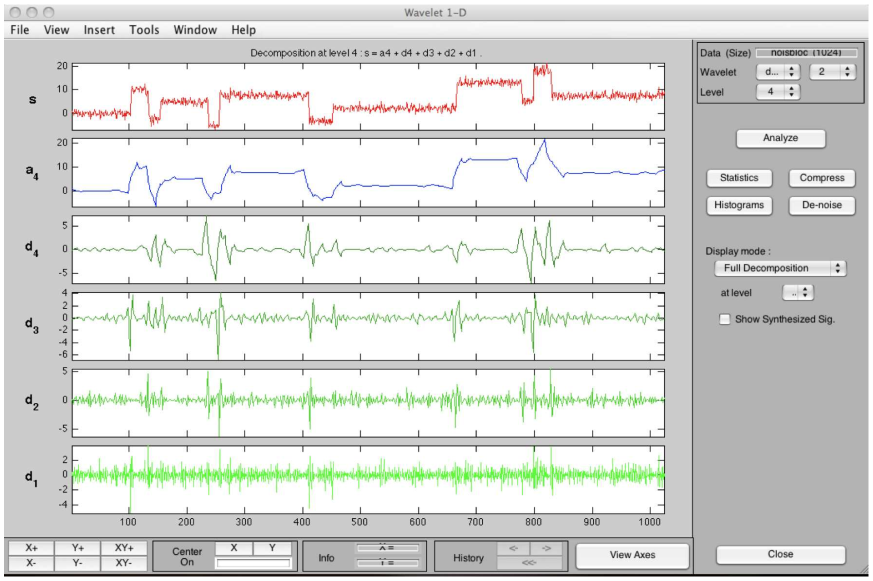
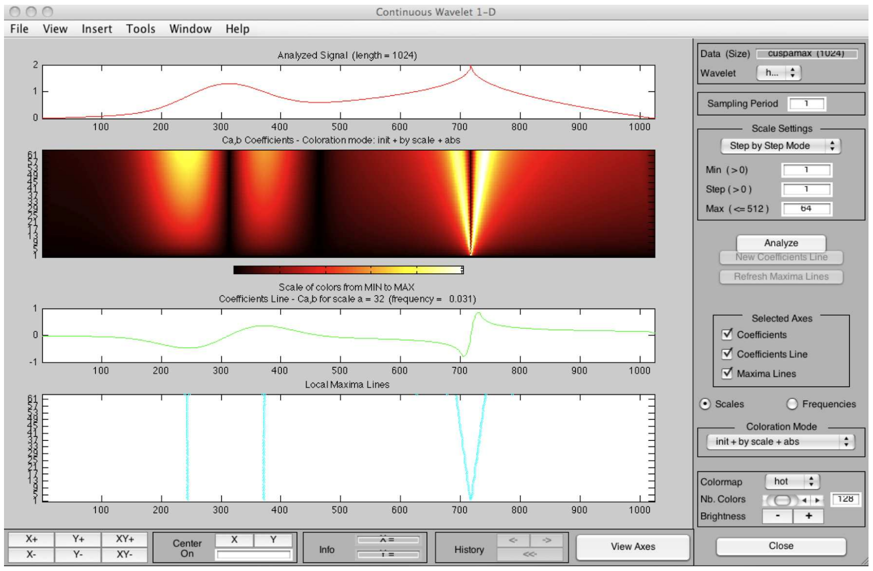


Figure 135: Screenshot of the wavelet 1-D with Daubechies wavelets.

maxima of the $C_{a,b}$ function is also shown. As before, different wavelet bases may be chosen along with different levels of resolution of a given signal.

Two final applications are demonstrated: One is an image denoising application, and the second is an image multi-resolution (compression) analysis. Both of these applications have important applications for image clean-up and size reduction, therefore they are prototypical examples of the power of the 2-D wavelet tools. In the first applications, illustrated in Fig. 137, a noisy image is considered. The application allows you to directly important your own image or data set. Once imported, the image is decomposed at the different resolution levels. To denoise the image, *filters* are applied at each level of resolution to remove the high-frequency components of each level of resolution. This produces an exceptional, wavelet based algorithm for denoising. As before, a myriad of wavelets can be considered and full control of the denoising process is left to the user. Figure 138 demonstrates the application of a multi-resolution analysis to a given image. Here the image is decomposed into various resolution levels. The program allows the user complete control and flexibility in determining the level of resolution desired. By keeping fewer levels of resolution, the image quality is compromised, but the algorithm thus provides an excellent compressed image. The discrete wavelet transform (**dwt**) is used to decompose the image. The inverse discrete wavelet transform (**idwt**) is used to extract back the image at an appropriate level of multi-resolution. For this example, two levels of decomposition are applied.


Continuous Wavelet 1-D
File View Insert Tools Window Help
Analyzed Signal (length = 1024)
Ca,b Coefficients - Coloration mode: init + by scale + abs
Scale of colors from MIN to MAX
Coefficients Line - Ca,b for scale a = 32 (frequency = 0.031)
Local Maxima Lines
Analyze
New Coefficients Line
Refresh Maxima Lines
Selected Axes
X+ Y+ XY+
X- Y- XY-
Center On X Y
Info
History
View Axes
Close

Data (Size) cuspamax (1024)
Wavelet h...
Sampling Period 1
Scale Settings
Step by Step Mode
Min (> 0) 1
Step (> 0) 1
Max (<= 512) 64
Selected Axes: ☑ Coefficients ☑ Coefficients Line ☑ Maxima Lines
[✓ Scales] Frequencies
Coloration Mode: init + by scale + abs
Colormap hot
Nb. Colors 128
Brightness - +

Figure 136: Screenshot of the continuous wavelet 1-D application where a signal is decomposed into a given wavelet basis and its wavelet coefficients are determined.

The wavelets transform and its algorithms can be directly accessed from the command line, just like the filter design toolbox. For instance, to apply a continuous wavelet transform, the command

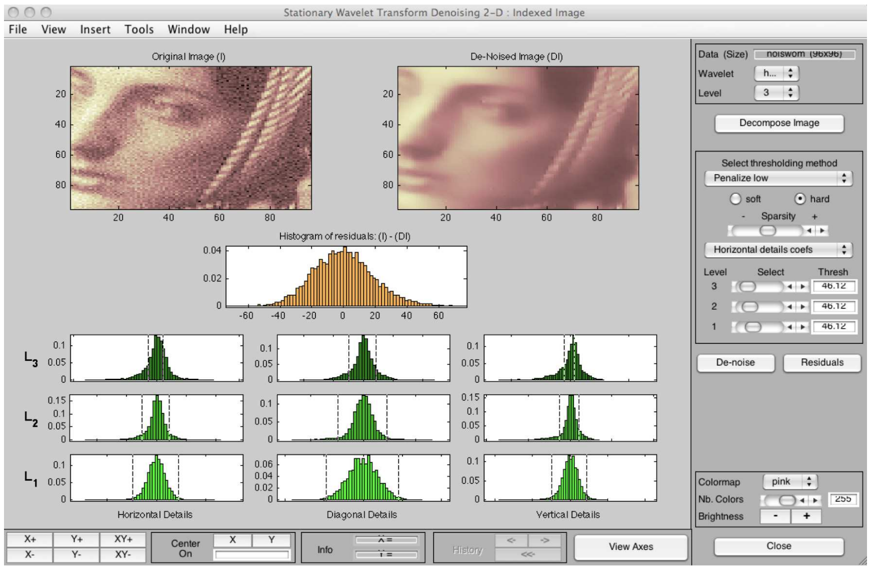
```
coeffs=cwt(S,scales,'wname')
```

can be used to compute the real or complex continuous 1-D wavelet coefficients. Specifically, it computes the continuous wavelet coefficients of the vector S at real, positive *scales*, using wavelet whose name is *wname* (for instance, haar). The signal S is real, the wavelet can be real or complex.

JPEG compression with wavelets

Wavelets can be directly applied to the arena of image compression. To illustrate the above concepts more clearly, we can consider a digital image with 600×800 pixels (rows × columns). Figure 139(a) shows the original image. The following MATLAB code imports the original JPEG picture and plots the results. It also constructs the matrix **Abw** which is the picture in matrix form.

```
A=imread('photo','jpeg');
Abw2=rgb2gray(A);
Abw=double(Abw2);
```



Data (Size) noiswom (96x96)
Wavelet h...
Level 3

Select thresholding method: Penalize low
soft [✓ hard]
Sparsity
Horizontal details coefs

Level	Select	Thresh
3		46.12
2		46.12
1		46.12

Colormap pink
Nb. Colors 255
Brightness

Figure 137: Screenshot of the denoising application using the wavelet bases.

```
[nx,ny]=size(Abw);
figure(1), subplot(2,2,1), imshow(Abw2)
```

The image is well defined and of high resolution. Indeed, it contains $600 \times 800 = 480,000$ pixels encoding the image. However, the image can actually be encoded with far less data. Specifically, it has been well-known for the past two decades that wavelets are an ideal basis in which to encode photos.

To demonstrate the nearly optimal encoding of the wavelet basis, the original photo is subjected to a two-dimensional wavelet transform:

```
[C,S]=wavedec2(Abw,2,'db1');
```

The **wavedec** command performs a two-dimensional transform and returns the wavelet coefficient vector **C** along with its ordering matrix **S**. In this example, a 2-level wavelet decomposition is performed (thus the value of 2 in the **wavedec** command call) using Daubechies wavelets (thus the **db1** option). Higher level wavelet decompositions can be performed using a variety of wavelets.

In the wavelet basis, the image is expressed as a collection of weightings of the wavelet coefficients. This is identical to representing a time-domain signal as a collection of frequencies using the Fourier transform. Figure 140(a) depicts the 480,000 wavelet coefficient values for the given picture. A zoom in of the dense looking region around 0 to 0.1 is shown in Fig. 141(a). In zooming in, one can clearly see that many of the coefficients that make up the image are

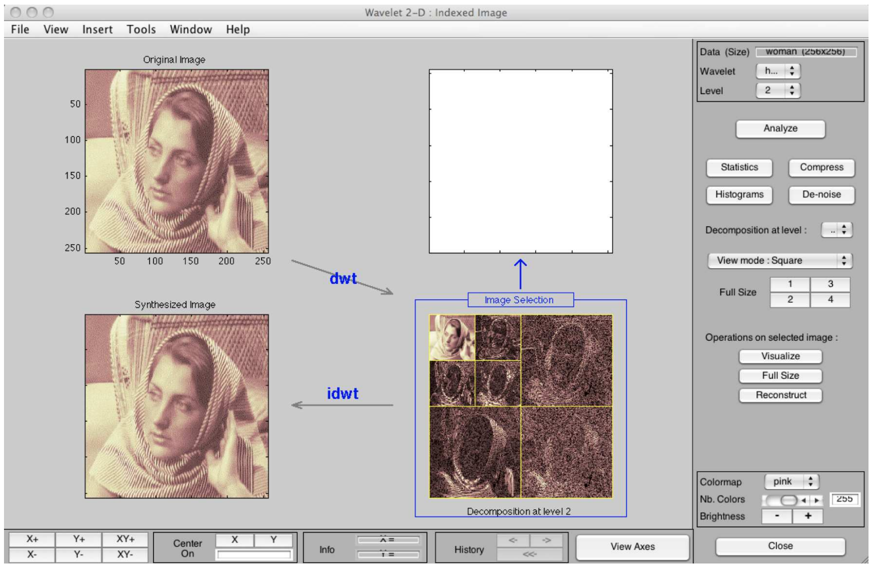


Figure 138: Screenshot of the decomposition of an image into different resolution levels. This application is ideal for image compression.

nearly zero. Due to numerical round-off, for instance, none of the coefficients will actually be zero.

Image compression follows directly from this wavelet analysis. Specifically, one can simply choose a threshold and directly set all wavelet coefficients below this chosen threshold to be identically zero instead of nearly zero. What remains is to demonstrate that the image can be reconstructed using such a strategy. If it works, then one would only need to retain the much smaller (sparse) number of non-zero wavelets in order to store and/or reconstruct the image.

To address the image compression issue, we can apply a threshold rule to the wavelet coefficient vector **C** that is shown in Fig. 140(a). The following code successively sets a threshold at 50, 100 and 200 in order to generate a sparse wavelet coefficient vector **C2**. This new sparse vector is then used to encode and reconstruct the image.

```
xw=(1:nx*ny)/(nx*ny);
th=[50 100 200];
for j=1:3
  count=0;
  C2=C;
  for jj=1:length(C2);
  if abs(C2(jj)) < th(j)
     C2(jj)=0;
```

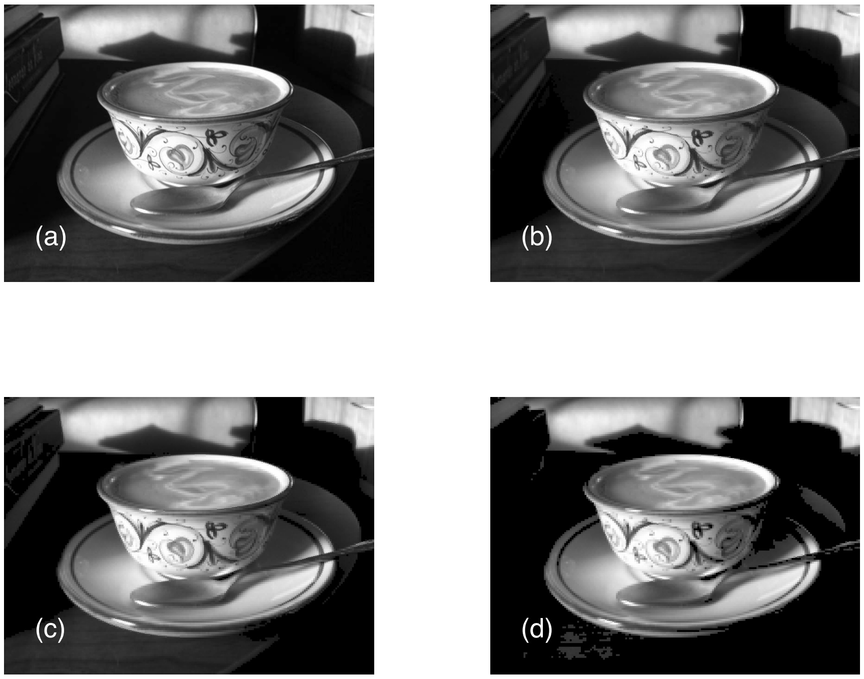


Figure 139: Original image (a) along with the reconstructed image using approximately (b) 5.6%, (c) 4.5% or (d) 3% of the information encoded in the wavelet coefficients. At 3%, the image reconstruction starts to fail. This reconstruction process illustrates that images are sparse in the wavelet representation.

```
     count=count+1;
  end
  end
  percent=count/length(C2)*100

  figure(2), subplot(4,1,j+1),plot(xw,C2,'k',[0 1],[th(j) th(j)],'k:')
  figure(3), subplot(4,1,j+1),plot(xw,C2,'k',[0 1],[th(j) th(j)],'k:')
    set(gca,'Xlim',[0.045 0.048])

  Abw2_sparse=waverec2(C2,S,'db1');
  Abw2_sparse2=uint8(Abw2_sparse);
  figure(1), subplot(2,2,j+1), imshow(Abw2_sparse2)
end
```

The results of the compression process are illustrated in Figs. 139-141.

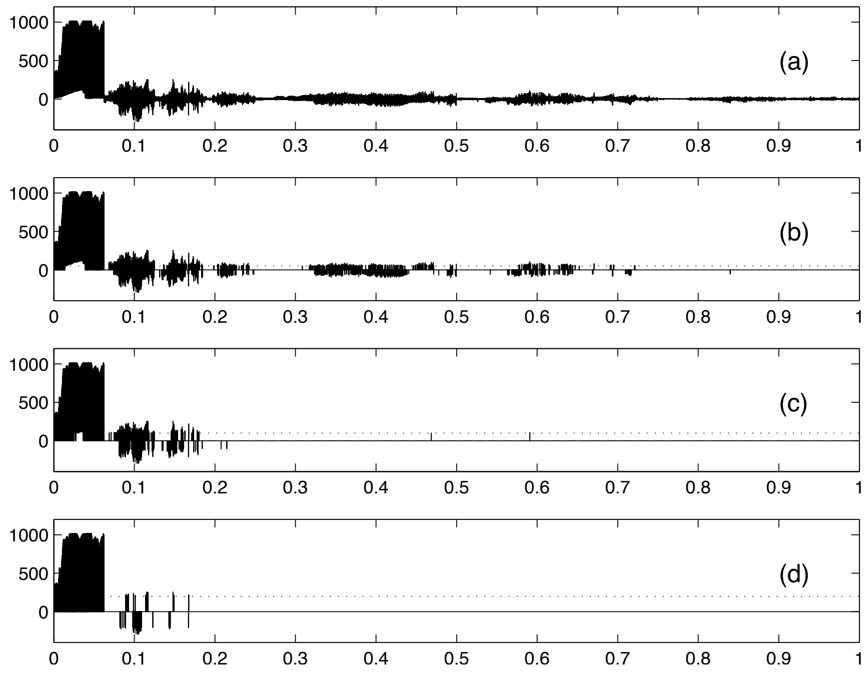


Figure 140: Original wavelet coefficient representation of the image (a) along with the thresholded coefficient vector for a threshold (dotted line) value of (b) 50, (c) 100 and (d) 200. The percentage of identically zero wavelet coefficients are 94.4%, 95.5% and 97.0% respectively. Only the non-zero coefficients are kept for encoding of the image. The x-axis has been normalized to unity.

To understand the compression process, first consider Fig. 140 which shows the wavelet coefficient vector for the three different threshold values of (b) 50, (c) 100 and (d) 200. The threshold line for each of these is shown as the dotted line. Anything below the threshold line is set identically to zero, thus it can be discarded when encoding the image. Figure 141 shows a zoom in of the region near 0 to 0.1. In addition to setting a threshold, the above code also calculates the percentage of zero wavelet coefficients. For the three threshold values considered here, the percentage of wavelet coefficients that are zero are 94.4%, 95.5% and 97.0% respectively. Thus only approximately 5.6%, 4.5% or 3% of the information is needed for the image reconstruction. Indeed, this is the key idea behind the JPEG2000 image compression algorithm: you only save what is needed to reconstruct the image to a desired level of accuracy.

The actual image reconstruction is illustrated in Fig. 139 for the three different levels of thresholding. From (b) to (d), the image is reconstructed from 5.6%, 4.5% or 3% of the original information. At 3%, the image finally becomes

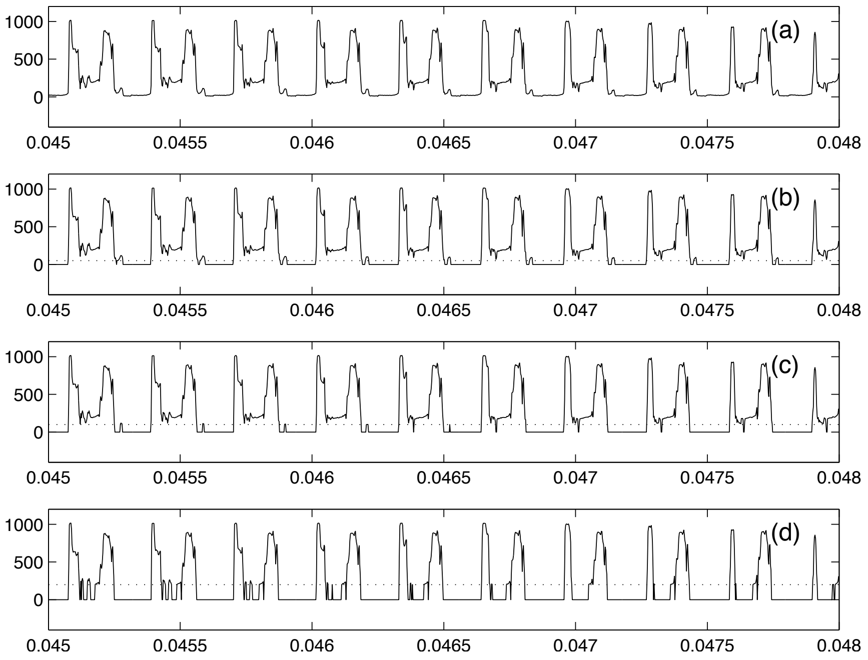


Figure 141: Zoom in of the original wavelet coefficient representation of the image (a) along with the thresholded coefficient vector for a threshold (dotted line) value of (b) 50, (c) 100 and (d) 200 (See Fig. 140). In the zoomed in region, one can clearly see that many of the coefficients are below threshold. The zoom in is over the entire wavelet domain that has been normalized to unity.

visibly worse and begins to fail in faithfully reproducing the original image. At 5.6% or 4.5%, it is difficult to tell much difference with the original image. This is quite remarkable and illustrates the *sparsity* of the image in the wavelet basis. Such sparsity is ubiquitous for image representation using wavelets.

14 Image Processing and Analysis

Over the past couple of decades, imaging science has had a profound impact on science, medicine and technology. The reach of imagining science is immense, spanning the range of modern day biological imaging tools to computer graphics. But the imaging methods are not limited to visual data. Instead, a general mathematical framework has been developed by which more general data analysis can be performed. The next set of subsections deal with some basic tools for image processing or data analysis.

14.1 Basic concepts and analysis of images

Imaging science is now ubiquitous. Indeed, it now dominates many scientific fields as a primary vehicle for achieving information and informing decisions. The following are a sample of the technologies and applications where imaging science is having a formative impact:

- **Ultrasound:** ultrasound refers to acoustic waves whose frequencies are higher than the audible range of human ears. Typically this is between 20Hz to 20KHz. Most medical ultrasound devices are above 1MHz and below 20MHz. Anybody who has had a baby will proudly show you their ultrasound pictures.

- **Infrared/thermal imaging:** Imperceptible to the human eye, infrared (IR) signatures correspond to wavelengths longer than 700nm. However, aided by signal amplification and enhancement devices, humans can perceive IR signals from above 700nm to a few microns.

- **Tomography (CAT scans):** Using the first Nobel Prize (1895) ideas of Wilhelm Röntgen for X-rays, tomography has since become the standard in the medical industry as the wavelengths, ranging from nanomenters to picometers, can easily penetrate most biological tissue. Imaging hardware and software have made this the standard for producing medical images.

- **Magnetic resonance imaging (MRI):** By applying a strong magnetic field to the body, the atomic spins in the body are aligned. High-frequency RF pulses are emitted into a slice plan of the external field. Upon turning off the RF waves, the relaxation process reveals differentials in tissue and biological material, thus giving the characteristics needed for imaging.

- **Radar and sonar imaging:** Radar uses radio waves for the detection and ranging of targets. The typical wavelengths are in the microwave range of 1cm to 1m. Applied to imaging, radar becomes a sophisticated tool for remote sensing applications. Sonar works in a similar fashion, but with underwater acoustic waves.

- **Digital photos:** Most of us are well aware of this application as we often would like to remove red-eye, undo camera blurring from out-of-focus shots or camera shakes, etc. Many software packages already exist to help you improve your image quality.

- **Computer graphics:** Building virtual worlds and targeted images falls within the aegis of the computer science community. There are many mathematically challenging issues related to representation of real-life images.

In the applications above, it goes without saying that most images acquired from experiment or enhancement devices are rarely without imperfections. Often the imperfections can be ignored. However, if the noise or imperfections in the images are critical for a decision making process, i.e. detection via radar of aircraft or detection of a tumor in biological tissue, then any enhancement of the image that could contribute to a statistically better decision is of high value.

Image processing and analysis addresses the fundamental issue of allowing an end user to have enhanced information, or statistically more accurate data, of a given image or signal. Thus mathematical method are the critical piece in achieving this enhancement. Before delving into specifics of mathematical methods that can be brought to bear on a given problem, it is important to classify the types of image analysis objectives that one can consider. The following gives a succinct list of image processing tasks:

- **image contrast enhancement:** Often there can be little contrast between different objects in a photo or in a given data set. Contrast enhancement is a method that tries to promote or enhance differences in an image in order to achieve sharper contrasts.

- **image denoising:** This is often of primary interest in analyzing data sets. Can noise, from either measurements or inaccuracies, be removed to give a more robust and fundamental picture of the underlying dynamics of a system.

- **image deblurring:** The leading cause of bad pictures: camera shakes and out-of-focus cameras. Image processing methods can often compensate and undo these two phenomena to produce a sharper image that is greatly deblurred. This is a remarkable application of image analysis, and is already an application that is included in many digital photo software bundles.

- **inpainting:** If data is missing over certain portions of an image or set of data, inpainting provides a mathematical framework for systematically generating the missing data. It accounts for both edges and the overall structure surrounding the missing pixels.

- **segmentation (edge detection):** Segmentation breaks up an image into blocks of different structures. This is particularly important for biomedical applications as well as military applications. Good algorithms for determining segmentation locations are critical.

The mathematical task is then to develop methods and algorithms capable of performing a set of tasks that addresses the above list. In the subsections that follow, only a small subset of these will be considered. However, a comprehensive mathematical treatment of the above issues can be performed [29].

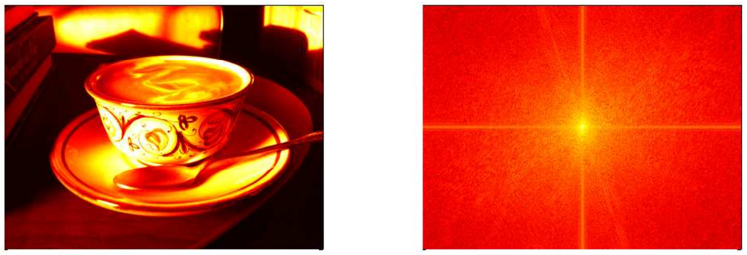


Figure 142: Image of a beautifully made cappuccino along with its Fourier transform (log of the absolute value of the spectrum) in two dimensions. The strong vertical and horizontal lines in the spectrum correspond to vertical and horizontal structures in the photo. Looking carefully at the spectrum will also reveal diagonal signatures associated with the diagonal edges in the photo.

Mathematical approaches

Given the broad appeal of imaging sciences and its applicability in such a wide variety of fields, it is not surprising that an equally diverse set of mathematical methods has been brought to bear on image processing. As with the previous lists of applications and tasks associated with image processing, the following list highlights some of the broad mathematical ideas that are present in thinking about image analysis.

Morphological approach: In this approach, one can think of the entire 2D domain of an image as a set of subdomains. Each subdomain should represent some aspect of the overall image. The fundamental idea here is to decompose the domain into fundamental shapes based upon some structure element S. The domain is then decomposed in a binary way, a small patch of the image (defined by the structure element S) is *in* the object of interest or *out* of the object of interest. This provides an efficient method for edge detection in certain sets of data or images.

Fourier analysis: As with time-frequency analysis, the key idea is to decompose the image into its Fourier components. This is easily done with a two-dimensional Fourier transform. Thus the image becomes a collection of Fourier modes. Figure 142 shows a photo and its corresponding 2D Fourier transform. The drawback of the Fourier transform is the fact that the Fourier transform of a sharp edged object results in a sinc-like function that decays in the Fourier modes like $1/k$ where k is the wavenumber. This slow decay means that localization of the image in both the spatial and wavenumber domain is limited. Regardless, the Fourier mode decomposition gives an alternative representation of the image. Moreover, it is clear from the Fourier spectrum that a great number of the Fourier mode components are zero or nearly so. Thus the concept of image compression can easily be seen directly from the spectrum, i.e. by saving 20% of the dominant modes only, the image can be nearly constructed. This then would compress the image five fold. As one might also expect, filtering with the Fourier transform can also help process the image to some desired ends.

Wavelets: As with the time-frequency analysis, wavelets provide a much more sophisticated representation of an image. In particular, wavelets allow for exceptional localization in both the spatial and wavenumber domain. In the wavelet basis, the key quantities are computed from the wavelet coefficients

$$c_\alpha = (u(x,y), \psi_\alpha) \tag{14.1.55}$$

where the wavelet coefficient c_α is equivalent to a Fourier coefficient for a wavelet ψ_α. Since the mid-1980s, the wavelet basis has taken over as the primary tool for image compression due to its excellent space-wavenumber localization properties.

Stochastic modeling: Consideration can be made in image processing of the statistical nature of the data itself. For instance, many images in nature, such as clouds, trees, sky, sandy beaches, are fundamentally statistical in nature. Thus one may not want to denoise such objects when performing image analysis tasks. In the stochastic modeling approach, hidden features are explored within a given set of observed data which accounts for the statistical nature of many of the underlying objects in an image.

Partial differential equations and diffusion: For denoising applications, or applications in which the data is choppy or highly pixelated, smoothing algorithms are of critical importance. In this case, one can take the original data as the initial conditions of a diffusion process so that the image undergoes, in the simplest case, the evolution

$$\frac{\partial u}{\partial t} = D\nabla^2 u \tag{14.1.56}$$

where $\nabla^2 = \partial_x^2 + \partial_y^2$. This diffusion process provides smoothing to the original data file u. The key is knowing when the smoothing process should be stopped so as not to remove too much image content.

Loading images, additive noise and MATLAB

To illustrate some of the various aspects of image processing, the following example will load the ideal image, apply Gaussian white noise to each pixel and show the noisy image. In MATLAB, most standard image formats are easily loaded. The key to performing mathematical manipulations is to then transform the data into double precision numbers which can be transformed, filtered, and manipulated at will. The generic data file uploaded in MATLAB is *uint8* which is an integer format ranging from 0 to 255. Moreover, color images have three sets of data for each pixel in order to specific the RGB color coordinates. The following code uploads and image file (600x800 pixels). Although there are 600x800 pixels, the data is stored as a 600x800x3 matrix where the extra dimensions contain the color coding. This can be made into a 600x800 matrix by turning the image into a black-and-white picture. This is done in the code that follows:

```
clear all; close all; clc;
A=imread('photo','tiff');   % load image
Abw=rgb2gray(A);         % make image black-and-white
subplot(2,2,1), image(A); set(gca,'Xtick',[],'Ytick',[])
subplot(2,2,3), imshow(Abw);

A2=double(A);          % change form unit8 to double
Abw=double(Abw);
```

Note that at the end of the code, the matrices produced are converted to double precision numbers.

To add Gaussian white noise to the images, either the color or black-and-white versions, the **randn** command is used. In fact, noise is added to both pictures and the output is produced.

```
noise=randn(600,800,3);  % add noise to RGB image
noise2=randn(600,800);  % add noise to black-and-white

u=uint8(A2+50*noise);   % change from double to uint8
u2=uint8(Abw+50*noise2);

subplot(2,2,2), image(u); set(gca,'Xtick',[],'Ytick',[])
subplot(2,2,4), image(u2); set(gca,'Xtick',[],'Ytick',[])
```

Note that at the final step, the images are converted back to standard image formats used for JPEG or TIFF images. Figure 143 demonstrates the plot produced from the above code. The ability to load and manipulate the data is fundamental to all the image processing applications to be pursued in what follows.

As a final example of data manipulation, the Fourier transform of the above image can also be considered. Figure 142 has already demonstrated the result of this code. But for completeness, it is illustrated here

```
Abw2=Abw(600:-1:1,:);
```

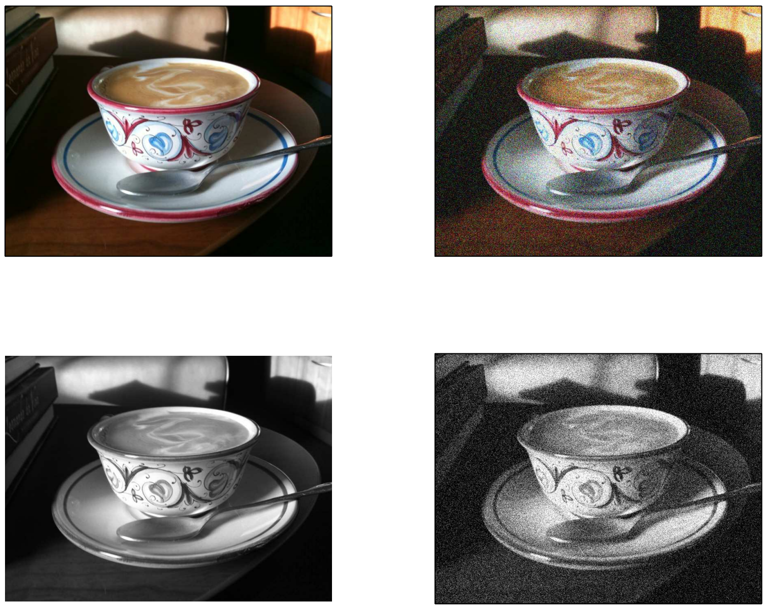


Figure 143: Ideal image in color and black-and-white (left panels) along with their noisy counterparts (right panels).

```
figure(2)
subplot(2,2,1), pcolor(Abw2), shading interp,
colormap(hot), set(gca,'Xtick',[],'Ytick',[])

Abwt=abs(fftshift(fft2(Abw2)));
subplot(2,2,2), pcolor(log(Abwt)), shading interp,
colormap(hot), set(gca,'Xtick',[],'Ytick',[])
```

Note that in this set of plots, we are once again working with matrices so that commands like **pcolor** are appropriate. These matrices are the subject of image processing routines and algorithms.

14.2 Linear filtering for image denoising

As with time-frequency analysis and denoising of time signals, many applications of image processing deal with cleaning up images from imperfections, pixelation, and graininess, i.e. processing of noisy images. The objective in any image processing application is to enhance or improve the quality of a given image. In this section, the filtering of noisy images will be considered with the aim of providing a higher quality, maximally denoised image.

The power of filtering has already been demonstrated in the context of radar detection applications. Ultimately, image denoising is directly related to filtering. To see this, consider Fig. 142 of the last section. In this image, the ideal image is represented along with the log of its Fourier transform. Like many images, the Fourier spectrum is particularly sparse (or nearly zero) for most high-frequency components. Noise, however, tends to generate many high-frequency structures on an image. Thus it is hoped that filtering of high-frequency components might remove unwanted noise fluctuations or graininess in an image. The top two panels of Fig. 144 show an initial noisy image and the log of its Fourier transform. These top two panels can be compared to the ideal image and spectrum shown in Fig. 142. The code used to generate these top two panels is given by the following:

```
A=imread('photo','tiff'); Abw=rgb2gray(A);
Abw=double(Abw);

B=Abw+100*randn(600,800);
Bt=fft2(B);  Bts=fftshift(Bt);

subplot(2,2,1), imshow(uint8(B)), colormap(gray)
subplot(2,2,2), pcolor((log(abs(Bts)))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
```

Note that the uploaded image is converted to a double precision number through the **double** command. It can be transformed back to the image format via the **uint8** command.

Linear filtering can be applied in the Fourier domain of the image in order to remove the high-frequency scale fluctuations induced by the noise. A simple filter to consider is a Gaussian that takes the form

$$F(k_x, k_y) = \exp(-\sigma_x(k_x - a)^2 - \sigma_y(k_y - b)^2) \tag{14.2.57}$$

where σ_x and σ_y are the filter widths in the x and y directions respectively and a and b are the center-frequency values for the corresponding filtering. For the image under consideration, it is a 600x800 pixel image so that the center-frequency components are located at $k_x = 301$ and $k_y = 401$ respectively. To build a Gaussian filter, the following lines of code are needed:

```
kx=1:800; ky=1:600; [Kx,Ky]=meshgrid(kx,ky);
F=exp(-0.0001*(Kx-401).^2-0.0001*(Ky-301).^2);
Btsf=Bts.*F;

subplot(2,2,3), pcolor(log(abs(Btsf))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
```

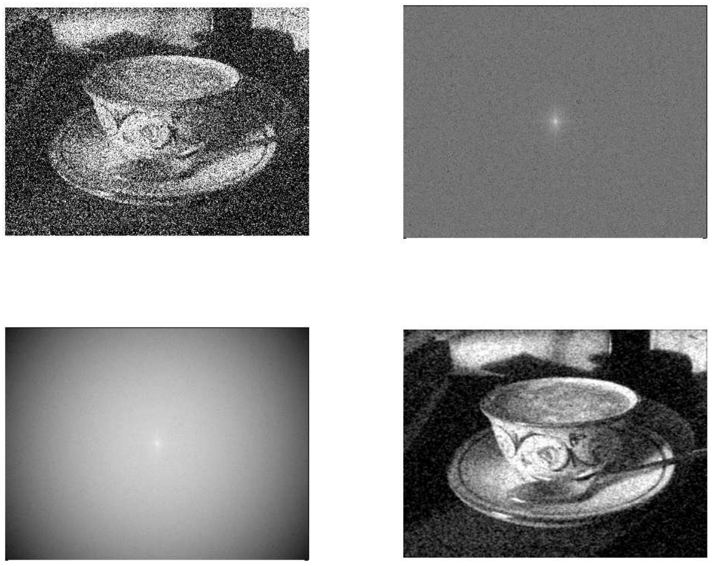


Figure 144: A noisy image and its Fourier transform are considered in the top panel. The ideal image is shown in Fig. 142. By applying a linear Gaussian filter, the high-frequency noise components can be eliminated (bottom left) and the image quality is significantly improved (bottom right).

```
  Btf=ifftshift(Btsf); Bf=ifft2(Btf);
  subplot(2,2,4), imshow(uint8(Bf)), colormap(gray)
```

In this code, the **meshgrid** command is used to generate the two-dimensional wavenumbers in the x and y direction. This allows for the construction of the filter function that has a two-dimensional form. In particular, a Gaussian centered around $(k_x, k_y) = (301, 401)$ is created with $\sigma_x = \sigma_y = 0.0001$. The bottom two panels in Fig. 144 show the filtered spectrum followed by its inverse Fourier transform. The final image on the bottom right of this figure shows the denoised image produced by simple filtering. The filtering significantly improves the image quality.

One can also over filter and cut out substantial information concerning the figure. Figure 145 shows the log of the spectrum of the noisy image for different values of filters in comparison with the unfiltered image. For narrow filters, much of the image information is irretrievably lost along with the noise. Finding an optimal filter is part of the image processing agenda. In the image spatial domain, the filtering strength (or width) can be considered. Figure 146 shows a series of filtered images and their comparison to the unfiltered image. Strong

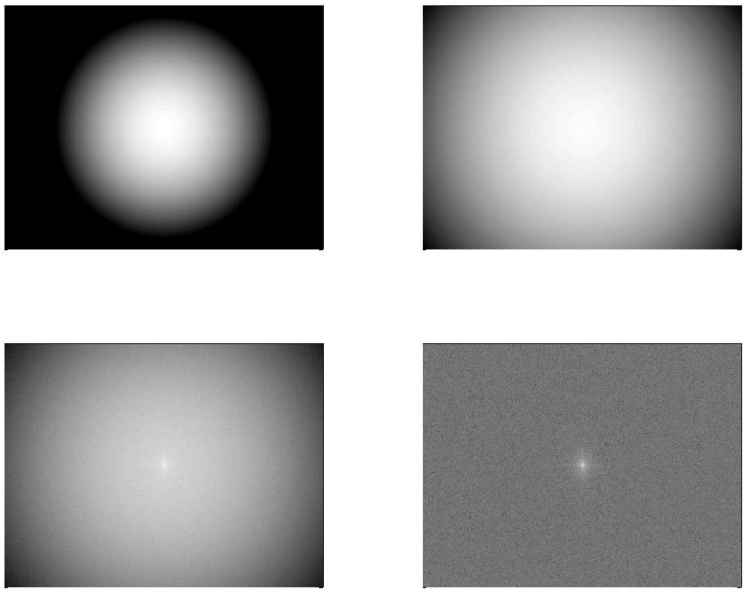


Figure 145: Comparison of the log of the Fourier transform for three different values of the filtering strength, $\sigma_x = \sigma_y = 0.01, 0.001, 0.0001$ (top left, top right, bottom left respectively), and the unfiltered image (bottom right).

filtering produces a smooth, yet blurry image. In this case, all fine scale features are lost. Moderate levels of filtering produce reasonable images with significant reduction of the noise in comparison to the non-filtered image. The following code was used to produce these last two figures.

```
% Gaussian filter
fs=[0.01 0.001 0.0001 0];
for j=1:4
  F=exp(-fs(j)*(Kx-401).^2-fs(j)*(Ky-301).^2);
  Btsf=Bts.*F; Btf=ifftshift(Btsf); Bf=ifft2(Btf);
  figure(4), subplot(2,2,j), pcolor(log(abs(Btsf)))
  shading interp,colormap(gray),set(gca,'Xtick',[],'Ytick',[])
  figure(5), subplot(2,2,j), imshow(uint8(Bf)), colormap(gray)
end
```

Note that the processed data is converted back to a graphics image before plotting with the **imshow** command.

Gaussian filtering is only one type of filtering that can be applied. There are, just like in signal processing applications, myriads of filters that can be

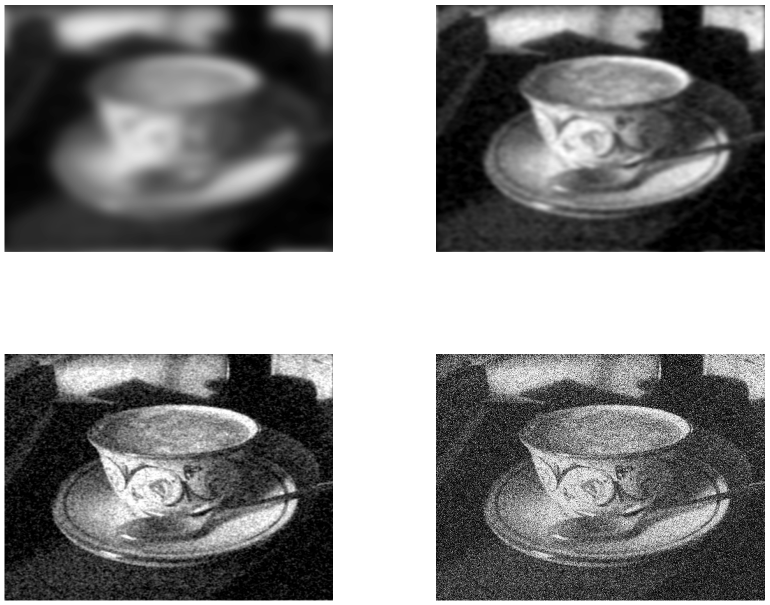


Figure 146: Comparison of the image quality for the three filter strengths considered in Fig. 145. A high degree of filtering blurs the image and no fine scale features are observed (top left). In contrast, moderate strength filtering produces exceptional improvement of the image quality (top right and bottom left). These denoised images should be compared to the unfiltered image (bottom right).

used to process a given image, including low-pass, high-pass, band-pass, etc. As a second example filter, the Shannon filter is considered. The Shannon filter is simply a step function with value of unity within the transmitted band and zero outside of it. In the example that follows, a Shannon filter is applied of width 50 pixels around the center frequency of the image.

```
width=50;
Fs=zeros(600,800);
Fs(301-width:1:301+width,401-width:1:401+width) ...
        =ones(1:2*width+1,1:2*width+1);
Btsf=Bts.*Fs;

Btf=ifftshift(Btsf); Bf=ifft2(Btf);

subplot(2,2,3), pcolor(log(abs(Btsf))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
```

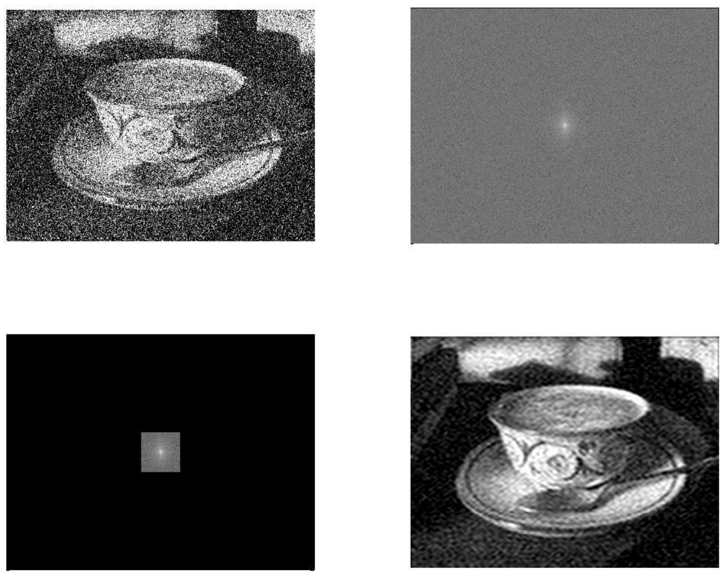


Figure 147: A noisy image and its Fourier transform are considered in the top panel. The ideal image is shown in Fig. 142. By applying a Shannon (step function) filter, the high-frequency noise components can be eliminated (bottom left) and the image quality is significantly improved (bottom right).

```
  subplot(2,2,4), imshow(uint8(Bf)), colormap(gray)
```

Figure 147 is a companion figure to Fig. 144. The only difference between them is the filter chosen for the image processing. The key difference is represented in the lower left panel of both figures. Note that the Shannon filter simply suppresses all frequencies outside of a given filter box. The performance difference between Gaussian filtering and Shannon filtering appears to be fairly marginal. However, there may be some applications where one or the other is more suitable. The image enhancement can also be explored as a function of the Shannon filter width. Figure 148 shows the image quality as the filter is widened along with the unfiltered image. Strong filtering again produces a blurred image while moderate filtering produces a greatly enhanced image quality. A code to produce this image is given by the following:

```
  fs=[10 50 100 200];
  for j=1:4
    Fs=zeros(600,800);
    Fs(301-fs(j):1:301+fs(j),401-fs(j):1:401+fs(j)) ...
```

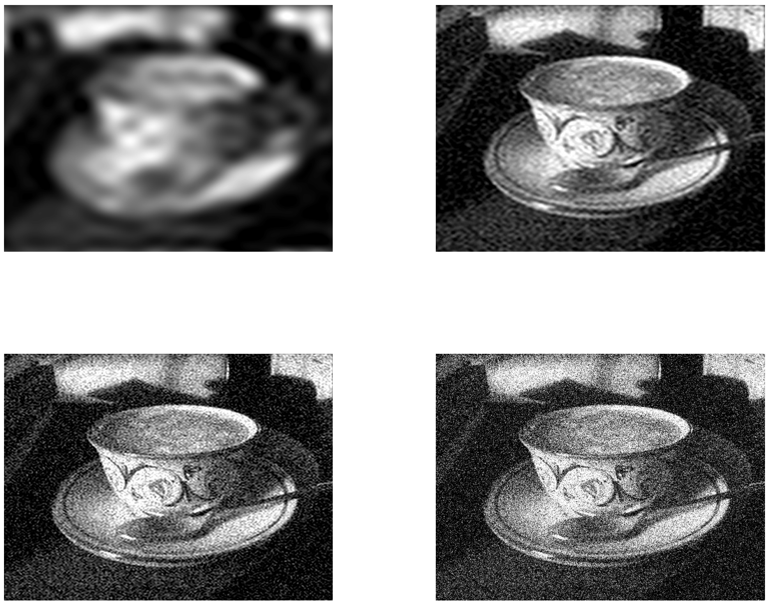


Figure 148: Comparison of the image quality for the three filter widths. A high degree of filtering blurs the image and no fine scale features are observed (top left). In contrast, moderate strength filtering produces exceptional improvement of the image quality (top right and bottom left). These denoised images should be compared to the unfiltered image (bottom right).

```
      =ones(1:2*fs(j)+1,1:2*fs(j)+1);
    Btsf=Bts.*Fs; Btf=ifftshift(Btsf); Bf=ifft2(Btf);
    figure(7), subplot(2,2,j), imshow(uint8(Bf)), colormap(gray)
  end
```

The width is adjusted by considering the number of filter pixels around the center frequency with value unity. One might be able to argue that the Gaussian filter produces slightly better image results since the inverse Fourier transform of a step function filter produces sinc-like behavior.

14.3 Diffusion and image processing

Filtering is not the only way to denoise an image. Intimately related to filtering is the use of diffusion for image enhancement. Consider for the moment the simplest spatial diffusion process in two dimensions, i.e. the heat equation:

$$u_t = D\nabla^2 u \tag{14.3.58}$$

where $u(x,y)$ will represent a given image, $\nabla^2 = \partial_x^2 + \partial_y^2$, D is a diffusion coefficient, and some boundary conditions must be imposed. If for the moment we consider periodic boundary conditions, then the solution to the heat equation can be found from, for instance, the Fourier transform

$$\hat{u}_t = -D(k_x^2 + k_y^2)\hat{u} \;\rightarrow\; \hat{u} = \hat{u}_0 e^{-D(k_x^2+k_y^2)t} \tag{14.3.59}$$

where the $\hat{u}$ is the Fourier transform of $u(x,y)$ and $\hat{u}_0$ is the Fourier transform of the initial conditions, i.e. the original noisy image. The solution of the heat equation illustrates a key and critical concept: the wavenumbers (spatial frequencies) decay according to a Gaussian function. Thus linear filtering with a Gaussian is equivalent to a linear diffusion of the image for periodic boundary conditions.

The above argument establishes some equivalency between filtering and diffusion. However, the diffusion formalism provides a more general framework in which to consider image cleanup since the heat equation can be modified to

$$u_t = \nabla \cdot (D(x,y)\nabla u) \tag{14.3.60}$$

where $D(x,y)$ is now a spatial diffusion coefficient. In particular, the diffusion coefficient could be used to great advantage to target trouble spots on an image while leaving relatively noise free patches alone.

To solve the heat equation numerically, we discretize the spatial derivative with a second-order scheme (see Table 12) so that

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{\Delta x^2}\left[u(x+\Delta x, y) - 2u(x,y) + u(x-\Delta x, y)\right] \tag{14.3.61a}$$
$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\Delta y^2}\left[u(x, y+\Delta y) - 2u(x,y) + u(x, y-\Delta y)\right]\,. \tag{14.3.61b}$$

This approximation reduces the partial differential equation to a system of ordinary differential equations. Once this is accomplished, then a variety of standard time-stepping schemes for differential equations can be applied to the resulting system.

The ODE system for 1D diffusion

Consider first diffusion in one-dimension so that $u(x,y) = u(x)$. The vector system for MATLAB can then be formulated. To define the system of ODEs, we discretize and define the values of the vector $\mathbf{u}$ in the following way.

$$\begin{aligned} u(-L) &= u_1 \\ u(-L+\Delta x) &= u_2 \\ &\vdots \end{aligned}$$

$$\begin{aligned} u(L-2\Delta x) &= u_{n-1} \\ u(L-\Delta x) &= u_n \\ u(L) &= u_{n+1}\,. \end{aligned}$$

If periodic boundary conditions, for instance, are considered, then periodicity requires that $u_1 = u_{n+1}$. Thus the system of differential equations solves for the vector

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}. \tag{14.3.62}$$

The governing heat equation is then reformulated in discretized form as the differential equations system

$$\frac{d\mathbf{u}}{dt} = \frac{\kappa}{\Delta x^2}\mathbf{A}\mathbf{u}\,, \tag{14.3.63}$$

where $\mathbf{A}$ is given by the sparse matrix

$$\mathbf{A} = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & & & \vdots \\ & & & & & 0 \\ \vdots & \cdots & 0 & 1 & -2 & 1 \\ 1 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix}, \tag{14.3.64}$$

and the values of one on the upper right and lower left of the matrix result from the periodic boundary conditions.

MATLAB implementation

The system of differential equations can now be easily solved with a standard time-stepping algorithm such as *ode23* or *ode45*. The basic algorithm would be as follows

1. Build the sparse matrix $\mathbf{A}$.

```
e1=ones(n,1);  % build a vector of ones
A=spdiags([e1 -2*e1 e1],[-1 0 1],n,n);  % diagonals
A(1,n)=1;  A(n,1)=1;  % periodic boundaries
```

2. Generate the desired initial condition vector $\mathbf{u} = \mathbf{u}_0$.

3. Call an ODE solver from the MATLAB suite. The matrix **A**, the diffusion constant κ and spatial step Δx need to be passed into this routine.

   ```
   [t,y]=ode45('rhs',tspan,u0,[],k,dx,A);
   ```

   The function *rhs.m* should be of the following form

   ```
   function rhs=rhs(tspan,u,dummy,k,dx,A)
   rhs=(k/dx^2)*A*u;
   ```

4. Plot the results as a function of time and space.

The algorithm is thus fairly routine and requires very little effort in programming since we can make use of the standard time-stepping algorithms already available in MATLAB.

2D MATLAB implementation

In the case of two dimensions, the calculation becomes slightly more difficult since the 2D data is represented by a matrix and the ODE solvers require a vector input for the initial data. For this case, the governing equation is

$$\frac{\partial u}{\partial t} = \kappa \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{14.3.65}$$

Provided $\Delta x = \Delta y = \delta$ are the same, the system can be reduced to the linear system

$$\frac{d\mathbf{u}}{dt} = \frac{\kappa}{\delta^2}\mathbf{A}\mathbf{u}\,, \tag{14.3.66}$$

where we have arranged the vector **u** by stacking slices of the data in the second dimension y. This stacking procedure is required for implementation purposes of MATLAB. Thus be defining

$$u_{nm} = u(x_n, y_m) \tag{14.3.67}$$

the collection of image (pixel) points can be arranged as follows

$$\mathbf{u} = \begin{pmatrix} u_{11} \\ u_{12} \\ \vdots \\ u_{1n} \\ u_{21} \\ u_{22} \\ \vdots \\ u_{n(n-1)} \\ u_{nn} \end{pmatrix}, \tag{14.3.68}$$

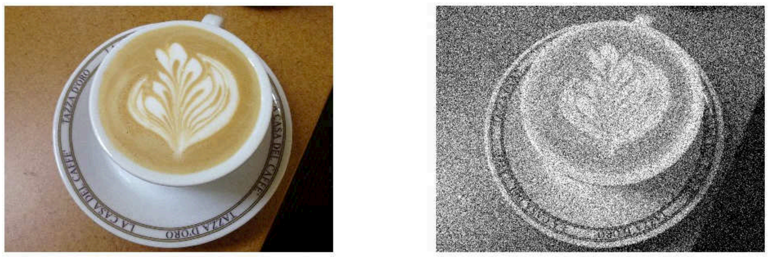


Figure 149: A beautiful looking image demonstrating the classic fernleaf pattern in a cappuccino (left panel). By adding noise to the ideal image, we are forced to use mathematics to cleanup this image (right panel).

The matrix **A** is a sparse matrix and so the sparse implementation of this matrix can be be used advantageously in MATLAB.

Again, the system of differential equations can now be easily solved with a standard time-stepping algorithm such as *ode23* or *ode45*. The basic algorithm follows the same course as the 1D case, but extra care is taken in arranging the 2D data into a vector.

Diffusion of an image

To provide a basic implementation of the above methods, consider the following MATLAB code which loads a given image file, converts it to double precision numbers, then diffuses the image in order to remove some of the noise content. The process begins once again with the loading of an ideal image on which noise will be projected.

```
clear all; close all; clc;

A=imread('espresso','jpeg');
Abw=rgb2gray(A); Abw=double(Abw);
[nx,ny]=size(Abw);
u2=uint8(Abw+20*randn(nx,ny));

subplot(2,2,1), imshow(A)
subplot(2,2,2), imshow(u2)
```

As before, the black-and-white version of the image will be the object consideration for diagnostic purposes. Figure 149 shows the ideal image along with a noisy black-and-white version. Diffusion will be used to denoise this image.

In what follows, a constant diffusion coefficient will be considered. To make the Laplacian operator in two dimensions, the **kron** command is used. The following code first makes the x- and $y-$derivatives in one dimension. The

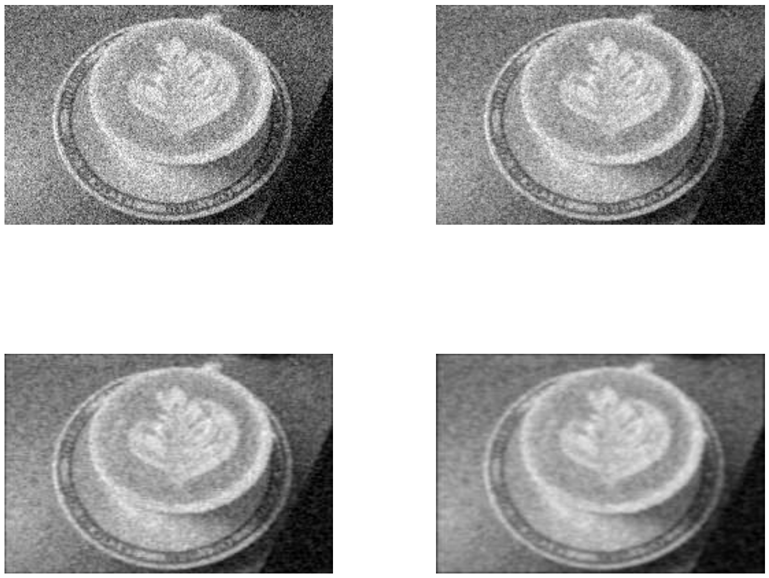


Figure 150: Image denoising process for a diffusion constant of $D = 0.0005$ as a function of time for $t = 0, 0.005, 0.02$ and 0.04. Perhaps the best image is for $t = 0.02$ where a moderate amount of diffusion has been applied. Further diffusion starts to degrade the image quality. The image was rescaled to 160×240 pixels.

kron command is much like the **meshgrid** in that it converts the operators into their two-dimensional versions.

```
x=linspace(0,1,nx); y=linspace(0,1,ny); dx=x(2)-x(1); dy=y(2)-y(1);
onex=ones(nx,1); oney=ones(ny,1);
Dx=(spdiags([onex -2*onex onex],[-1 0 1],nx,nx))/dx^2; Ix=eye(nx);
Dy=(spdiags([oney -2*oney oney],[-1 0 1],ny,ny))/dy^2; Iy=eye(ny);
L=kron(Iy,Dx)+kron(Dy,Ix);
```

The generated operator **L** takes two derivatives in x and y and is the Laplacian in two-dimensions. This operator is constructed to act upon data that has been stacked as in Eq. (14.3.68).

It simply remains to evolve the image through the diffusion equation. But now that the Laplacian operator has been constructed, it simply remains to reshape the image, determine a time-span for evolving, and choose a diffusion coefficient. The following code performs the diffusion of the image and produces a plot that tracks the image from its initial state to the final diffused image.

```
tspan=[0 0.005 0.02 0.04]; D=0.0005;
```

```
u3=Abw+20*noise2;
u3_2=reshape(u3,nx*ny,1);
[t,usol]=ode113('image_rhs',tspan,u3_2,[],L,D);

for j=1:length(t)
  Abw_clean=uint8(reshape(usol(j,:),nx,ny));
  subplot(2,2,j), imshow(Abw_clean);
end
```

The above code calls on the function **image_rhs** which contains the diffusion equation information. This function contains the following two lines of code:

```
function rhs=image_rhs(t,u,dummy,L,D)
rhs=D*L*u;
```

Figure 150 shows the evolution of the image through the diffusion process. Note that only a slight amount of diffusion is needed, i.e. a small diffusion constant as well as a short diffusion time, before the pixelation has been substantially reduced. Continued diffusion starts to degrade image quality. This corresponds to over-filtering the image.

Nonlinear filtering and diffusion

As mentioned previously, the linear diffusion process is equivalent to the application of a Gaussian filter. Thus it is not clear that the diffusion process is any better than simple filtering. However, the diffusion process has several distinct advantages: first, particular regions in the spatial figure can be targeted by modification of the diffusion coefficient $D(x, y)$. Second, nonlinear diffusion can be considered for enhancing certain features of the image. This corresponds to a nonlinear filtering process. In this case,

$$u_t = \nabla \cdot (D(u, \nabla u)\nabla u) \tag{14.3.69}$$

where the diffusion coefficient now depends on the image and its gradient. Nonlinear diffusion, with a properly constructed D above, can be used, for instance, as an effective method for extracting edges from an image [29]. Thus although this section has only considered a simple linear diffusion model, the ideas are easily generalized to account for more general concepts and image processing aims.

15 Linear Algebra and Singular Value Decomposition

Linear algebra plays a central role in almost every application area of mathematics in the physical, engineering and biological sciences. It is perhaps the

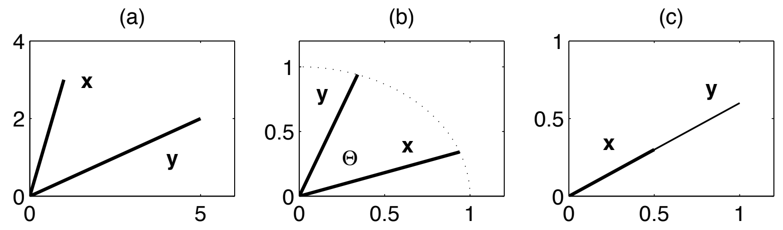


Figure 151: Transformation of a vector $\mathbf{x}$ under the action of multiplication by the matrix $\mathbf{A}$, i.e. $\mathbf{y} = \mathbf{A}\mathbf{x}$. (a) Generic rotation and stretching of the vector as given by Eq. (15.1.70). (b) rotation by 50^0 of a unit vector by the rotation matrix (15.1.71). (c) Stretching of a vector to double its length using (15.1.72) with $\alpha = 2$.

most important theoretical framework to be familiar with as a student of mathematics. Thus it is no surprise that it also plays a key role in data analysis and computation. In what follows, emphasis will be placed squarely on the *singular value decomposition* (SVD). This concept is often untouched in undergraduate courses in linear algebra, yet it forms one of the most powerful techniques for analyzing a myriad of applications areas.

15.1 Basics of The Singular Value Decomposition (SVD)

In even the earliest experience of linear algebra, the concept of a matrix transforming a vector via multiplication was defined. For instance, the vector $\mathbf{x}$ when multiplied by a matrix $\mathbf{A}$ produces a new vector $\mathbf{y}$ that is now aligned, generically, in a new direction with a new length. To be more specific, the following example illustrates a particular transformation:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix} \quad \rightarrow \quad \mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}. \tag{15.1.70}$$

Figure 151(a) shows the vector $\mathbf{x}$ and is transformed version, $\mathbf{y}$, after application of the matrix $\mathbf{A}$. Thus generically, matrix multiplication will rotate and stretch (compress) a given vector as prescribed by the matrix $\mathbf{A}$ (See Fig. 151(a)).

The rotation and stretching of a transformation can be precisely controlled by proper construction of the matrix $\mathbf{A}$. In particular, it is well known that in a two-dimensional space, the rotation matrix

$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{15.1.71}$$

takes a given vector $\mathbf{x}$ and rotates it by an angle θ to produce the vector $\mathbf{y}$. The transformation produced by $\mathbf{A}$ is known as a *unitary transformation* since

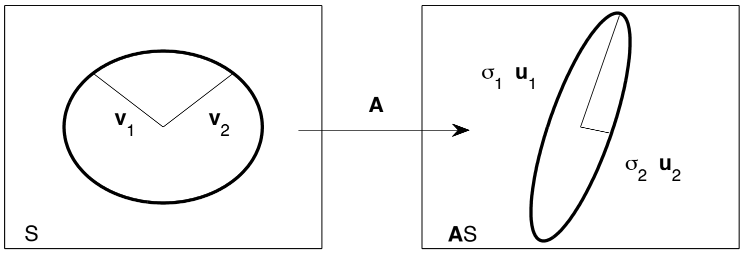


Figure 152: Image of a unit sphere S transformed into a hyperellipse $\mathbf{A}S$ in $\mathbb{R}^2$. The values of σ_1 and σ_2 are the singular values of the matrix $\mathbf{A}$ and represent the lengths of the semiaxes of the ellipse.

the matrix inverse $\mathbf{A}^{-1} = \bar{\mathbf{A}}^T$ where the bar denotes complex conjugation [30]. Thus rotation can be directly specified without the vector being scaled. To scale the vector in length, the matrix

$$\mathbf{A} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \tag{15.1.72}$$

can be applied to the vector $\mathbf{x}$. This multiplies the length of the vector $\mathbf{x}$ by α. If $\alpha = 2$ (0.5), then the vector is twice (half) its original length. The combination of the above two matrices gives arbitrary control of rotation and scaling in a two-dimentional vector space. Figure 151 demonstrates some of the various operations associated with the above matrix transformations.

A *singular value decomposition* (SVD) is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in applications. The SVD, much as illustrated in the preceding paragraph, is essentially a transformation that stretches/compresses and rotates a given set of vectors. In particular, the following geometric principle will guide our forthcoming discussion: the image of a unit sphere under any $m \times n$ matrix is a hyperellipse. A hyperellipse in $\mathbb{R}^m$ is defined by the surface obtained upon stretching a unit sphere in $\mathbb{R}^m$ by some factors $\sigma_1, \sigma_2, \cdots, \sigma_m$ in the orthogonal directions $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_m \in \mathbb{R}^m$. The stretchings σ_i can possibly be zero. For convenience, consider the $\mathbf{u}_j$ to be unit vectors so that $\|\mathbf{u}_j\|_2 = 1$. The quantity $\sigma_j \mathbf{u}_j$ is then the *principal semiaxes* of the hyperellipse with the length σ_j. Figure 152 demonstrates a particular hyperellipse created under the matrix transformation $\mathbf{A}$ in $\mathbb{R}^2$.

A few things are worth noting at this point. First, if $\mathbf{A}$ has rank r, exactly r of the lengths σ_j will be nonzero. And if the matrix $\mathbf{A}$ is an $m \times n$ matrix where $m > n$, at most n of the σ_j will be nonzero. Consider for the moment a full rank matrix $\mathbf{A}$. Then the n singular values of $\mathbf{A}$ are the lengths of the

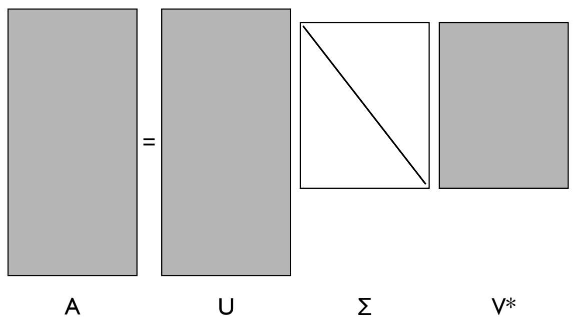


Figure 153: Graphical description of the reduced SVD decomposition.

principal semiaxes $\mathbf{A}S$ as shown in Fig. 152. Convention assumes that the singular values are ordered with the largest first and then in descending order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$.

On a more formal level, the transformation from the unit sphere to the hyperellipse can be more succinctly stated as follows:

$$\mathbf{A}\mathbf{v}_j = \sigma_j \mathbf{u}_j \quad 1 \leq j \leq n\,. \tag{15.1.73}$$

Thus in total, there are n vectors that are transformed under $\mathbf{A}$. A more compact way to write all of these equations simultaneously is with the representation

$$\left[\begin{array}{c} \\ \\ \mathbf{A} \\ \\ \\ \end{array}\right] \left[\begin{array}{cccc} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{array}\right] = \left[\begin{array}{cccc} \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ \\ \end{array}\right] \left[\begin{array}{cccc} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{array}\right] \tag{15.1.74}$$

so that in compact matrix notation this becomes

$$\mathbf{A}\mathbf{V} = \hat{\mathbf{U}}\hat{\boldsymbol{\Sigma}}\,. \tag{15.1.75}$$

The matrix $\hat{\boldsymbol{\Sigma}}$ is an $n \times n$ diagonal matrix with positive entries provided the matrix $\mathbf{A}$ is of full rank. The matrix $\hat{\mathbf{U}}$ is an $m \times n$ matrix with orthonormal columns, and the matrix $\mathbf{V}$ is an $n \times n$ unitary matrix. Since $\mathbf{V}$ is unitary, the above equation can be solved for $\mathbf{A}$ by multiplying on the right with $\mathbf{V}^*$ so that

$$\mathbf{A} = \hat{\mathbf{U}}\hat{\boldsymbol{\Sigma}}\mathbf{V}^*\,. \tag{15.1.76}$$

This factorization is known as the *reduced singular value decomposition*, or reduced SVD, of the matrix $\mathbf{A}$. Graphically, the factorization is represented in Fig. 153.

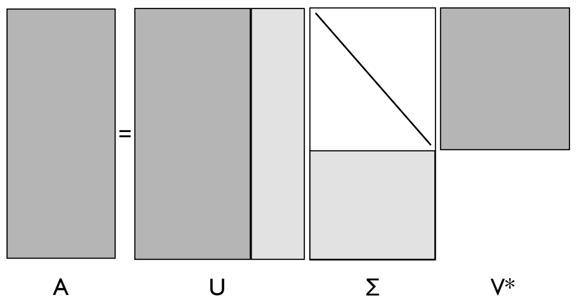


Figure 154: Graphical description of the full SVD decomposition where both **U** and **V** are unitary matrices. The light shaded regions of **U** and **Σ** are the silent rows and columns that are extended from the reduced SVD.

The reduced SVD is not the standard definition of the SVD used in the literature. What is typically done to augment the treatment above is to construct a matrix **U** from $\hat{\mathbf{U}}$ by adding an additional $m-n$ columns that are orthonormal to the already existing set in $\hat{\mathbf{U}}$. Thus the matrix **U** becomes an $m \times m$ unitary matrix. In order to make this procedure work, an additional $m-n$ rows of zeros is also added to the $\hat{\mathbf{\Sigma}}$ matrix. These "silent" columns of **U** and rows of **Σ** are shown graphically in Fig. 154. In performing this procedure, it becomes evident that rank deficient matrices can easily be handled by the SVD decomposition. In particular, instead of $m-n$ silent rows and matrices, there are now simply $m-r$ silent rows and columns added to the decomposition. Thus the matrix **Σ** will have r positive diagonal entries, with the remaining $n-r$ being equal to zero.

The full SVD decomposition thus take the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \,. \tag{15.1.77}$$

with the following three matrices

$$\mathbf{U} \in \mathbb{C}^{m\times m} \text{ is unitary} \tag{15.1.78a}$$
$$\mathbf{V} \in \mathbb{C}^{n\times n} \text{ is unitary} \tag{15.1.78b}$$
$$\mathbf{\Sigma} \in \mathbb{R}^{m\times n} \text{ is diagonal} \tag{15.1.78c}$$

Additionally, it is assumed that the diagonal entries of Σ are nonnegative and ordered from largest to smallest so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p = \min(m,n)$. The SVD decomposition of the matrix **A** thus shows that the matrix first applies a unitary transformation preserving the unit sphere via $\mathbf{V}^*$. This is followed by a stretching operation that creates an ellipse with principal semiaxes given by the matrix **Σ**. Finally, the generated hyperellipse is rotated by the unitary transformation $\mathbf{U}$. Thus the statement: the image of a unit sphere under any $m \times n$ matrix is a hyperellipse, is shown to be true. The following is the primary theorem concerning the SVD:

Theorem: *Every matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ has a singular value decomposition (15.1.77). Furthermore, the singular values $\{\sigma_j\}$ are uniquely determined, and, if $\mathbf{A}$ is square and the σ_j distinct, the singular vectors $\{\mathbf{u}_j\}$ and $\{\mathbf{v}_j\}$ are uniquely determined up to complex signs (complex scalar factors of absolute value 1).*

Computing the SVD

The above theorem guarantees the existence of the SVD, but in practice, it still remains to be computed. This is a fairly straightforward process if one considers the following matrix products:

$$\begin{aligned} \mathbf{A}^T\mathbf{A} &= (\mathbf{U\Sigma V}^*)^T(\mathbf{U\Sigma V}^*) \\ &= \mathbf{V\Sigma U}^*\mathbf{U\Sigma V}^* \\ &= \mathbf{V\Sigma}^2\mathbf{V}^* \end{aligned} \tag{15.1.79}$$

and

$$\begin{aligned} \mathbf{A}\mathbf{A}^T &= (\mathbf{U\Sigma V}^*)(\mathbf{U\Sigma V}^*)^T \\ &= \mathbf{U\Sigma V}^*\mathbf{V\Sigma U}^* \\ &= \mathbf{U\Sigma}^2\mathbf{U}^* . \end{aligned} \tag{15.1.80}$$

Multiplying (15.1.79) and (15.1.80) on the right by $\mathbf{V}$ and $\mathbf{U}$ respectively gives the two self-consistent eigenvalue problems

$$\mathbf{A}^T\mathbf{A}\mathbf{V} = \mathbf{V\Sigma}^2 \tag{15.1.81a}$$
$$\mathbf{A}\mathbf{A}^T\mathbf{U} = \mathbf{U\Sigma}^2 . \tag{15.1.81b}$$

Thus if the normalized eigenvectors are found for these two equations, then the orthonormal basis vectors are produced for $\mathbf{U}$ and $\mathbf{V}$. Likewise, the square root of the eigenvalues of these equations produces the singular values σ_j.

Example: Consider the SVD decomposition of

$$A = \begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix} . \tag{15.1.82}$$

The following quantities are computed:

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 4 \end{bmatrix} \tag{15.1.83a}$$
$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 4 \end{bmatrix} . \tag{15.1.83b}$$

The eigenvalues are clearly $\lambda = \{9, 4\}$, giving singular values $\sigma_1 = 3$ and $\sigma_2 = 2$. The eigenvectors can similarly be constructed and the matrices $\mathbf{U}$ and $\mathbf{V}$ are given by

$$\begin{bmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{bmatrix} \tag{15.1.84}$$

where there is an indeterminate sign in the eigenvectors. However, a self-consistent choice of signs must be made. One possible choice gives

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{15.1.85}$$

The SVD can be easily computed in MATLAB with the following command:

```
[U,S,V]=svd(A);
```

where the $[U, S, V]$ correspond to $\mathbf{U}$, $\boldsymbol{\Sigma}$ and $\mathbf{V}$ respectively. This decomposition is a critical tool for analyzing many data driven phenomena. But before proceeding to such applications, the connection of the SVD with standard and well-known techniques is elucidated.

15.2 The SVD in broader context

The SVD is an exceptionally important tool in many areas of applications. Part of this is due to its many mathematical properties and its guarantee of existence. In what follows, some of its more important theorems and relations to other standard ideas of linear algebra are considered with the aim of setting the mathematical framework for future applications.

Eigenvalues, eigenvectors and diagonalization

The concept of eigenvalues and eigenvectors is critical to understanding many areas of applications. One of the most important areas where it plays a role is in understanding differential equations. Consider, for instance, the system of differential equations:

$$\frac{d\mathbf{y}}{dt} = \mathbf{A}\mathbf{y} \tag{15.2.86}$$

for some vector $\mathbf{y}(t)$ representing a dynamical system of variables and where the matrix $\mathbf{A}$ determines the interaction among these variables. Assuming a solution of the form $\mathbf{y} = \mathbf{x}\exp(\lambda t)$ results in the eigenvalue problem:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}\,. \tag{15.2.87}$$

The question remains: How are the eigenvalues and eigenvectors found? To consider this problem, we rewrite the eigenvalue problem as

$$\mathbf{A}\mathbf{x} - \lambda\mathbf{I}\mathbf{x} = (\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}. \tag{15.2.88}$$

Two possibilities now exist.

Option I: The determinant of the matrix $(\mathbf{A}-\lambda\mathbf{I})$ is not zero. If this is true, the matrix is *nonsingular* and its inverse, $(\mathbf{A}-\lambda\mathbf{I})^{-1}$, can be found. The solution to the eigenvalue problem (15.2.87) is then

$$\mathbf{x} = (\mathbf{A}-\lambda\mathbf{I})^{-1}\mathbf{0} \tag{15.2.89}$$

which implies that $\mathbf{x}=\mathbf{0}$. This trivial solution could have easily been guessed. However, it is not relevant as we require nontrivial solutions for $\mathbf{x}$.

Option II: The determinant of the matrix $(\mathbf{A}-\lambda\mathbf{I})$ is zero. If this is true, the matrix is *singular* and its inverse, $(\mathbf{A}-\lambda\mathbf{I})^{-1}$, cannot be found. Although there is no longer a guarantee that there is a solution, it is the only scenario which allows for the possibility of $\mathbf{x}\neq\mathbf{0}$. It is this condition which allows for the construction of eigenvalues and eigenvectors. Indeed, we choose the eigenvalues λ so that this condition holds and the matrix is singular.

Another important operation which can be performed with eigenvalue and eigenvectors is the evaluation of

$$\mathbf{A}^M \tag{15.2.90}$$

where M is a large integer. For large matrices $\mathbf{A}$, this operation is computationally expensive. However, knowing the eigenvalues and eigenvectors of $\mathbf{A}$ allows for a significant ease in computational expense. Assuming we have all the eigenvalues and eigenvectors of $\mathbf{A}$, then

$$\begin{aligned}\mathbf{A}\mathbf{x}_1 &= \lambda_1\mathbf{x}_1\\ \mathbf{A}\mathbf{x}_2 &= \lambda_2\mathbf{x}_2\\ &\vdots\\ \mathbf{A}\mathbf{x}_n &= \lambda_1\mathbf{x}_n\,.\end{aligned}$$

This collection of eigenvalues and eigenvectors gives the matrix system

$$\mathbf{A}\mathbf{S} = \mathbf{S}\mathbf{\Lambda} \tag{15.2.91}$$

where the columns of the matrix $\mathbf{S}$ are the eigenvectors of $\mathbf{A}$,

$$\mathbf{S} = (\mathbf{x}_1 \;\; \mathbf{x}_2 \;\; \cdots \;\; \mathbf{x}_n)\,, \tag{15.2.92}$$

and $\mathbf{\Lambda}$ is a matrix whose diagonals are the corresponding eigenvalues

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & & 0\\ 0 & \lambda_2 & 0 & \cdots & 0\\ \vdots & & \ddots & & \vdots\\ 0 & & \cdots & 0 & \lambda_n \end{pmatrix}. \tag{15.2.93}$$

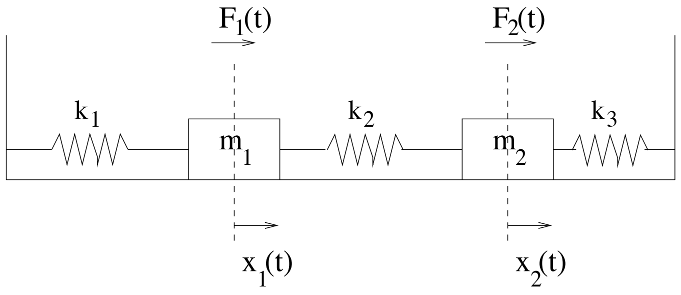


Figure 155: A two mass, three spring system. The fundamental behaviour of the system can be understood by decomposition the system via diagonalization. This reveals that all motion can be expressed as the sum of two fundamental motions: the masses oscillating in-phase, and the masses oscillating exactly out-of-phase.

By multiplying (15.2.91) on the right by $\mathbf{S}^{-1}$, the matrix $\mathbf{A}$ can then be rewritten as (note the similarity between this expression and Eq. (15.1.77) for the SVD decomposition)

$$\mathbf{A} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}\,. \tag{15.2.94}$$

The final observation comes from

$$\mathbf{A}^2 = (\mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1})(\mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}) = \mathbf{S}\mathbf{\Lambda}^2\mathbf{S}^{-1}\,. \tag{15.2.95}$$

This then generalizes to

$$\mathbf{A}^M = \mathbf{S}\mathbf{\Lambda}^M\mathbf{S}^{-1} \tag{15.2.96}$$

where the matrix $\mathbf{\Lambda}^M$ is easily calculated as

$$\mathbf{\Lambda}^M = \begin{pmatrix} \lambda_1^M & 0 & \cdots & & 0 \\ 0 & \lambda_2^M & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & \cdots & 0 & \lambda_n^M \end{pmatrix}. \tag{15.2.97}$$

Since raising the diagonal terms to the M^{th} power is easily accomplished, the matrix $\mathbf{A}$ can then be easily calculated by multiplying the three matrices in (15.2.96)

Diagonalization can also recast a given problem so as to elucidate its more fundamental dynamics. A classic example of the use of diagonalization is a two mass-spring system where the masses m_1 and m_2 are acted on by forces $F_1(t)$ and $F_2(t)$. A schematic of this situation is depicted in Fig. 155. For each mass, we can write down Newton's Law:

$$\sum F_1 = m_1\frac{d^2x_1}{dt^2} \qquad \text{and} \qquad \sum F_2 = m_2\frac{d^2x_2}{dt^2} \tag{15.2.98}$$

where $\sum F_1$ and $\sum F_2$ are the sum of the forces on m_1 and m_2 respectively. Note that the equations for $x_1(t)$ and $x_2(t)$ are coupled because of the spring with spring constant k_2. The resulting governing equations are then of the form:

$$m_1 \frac{d^2x_1}{dt^2} = -k_1x_1 + k_2(x_2 - x_1) + F_1 = -(k_1 + k_2)x_1 + k_2x_2 + F_1 \quad (15.2.99a)$$

$$m_2 \frac{d^2x_2}{dt^2} = -k_3x_2 - k_2(x_2 - x_1) + F_2 = -(k_2 + k_3)x_2 + k_2x_1 + F_2 . \quad (15.2.99b)$$

This can be reduced further by assuming, for simplicity, $m = m_1 = m_2$, $K = k_1/m = k_2/m$ and $F_1 = F_2 = 0$. This results in the linear system which can be diagonalized via Eq. (15.2.94). The pairs of complex conjugate eigenvalues are produced $\lambda_1^{\pm} = \pm i(2K + \sqrt{2K})^{1/2}$ and $\lambda_2^{\pm} = \pm i(2K - \sqrt{2K})^{1/2}$. Upon diagonalization, the full system can be understood as simply a linear combination of oscillations of the masses that in-phase with each other or out-of-phase with each other.

Diagonalization via SVD

Like the eigenvalue and eigenvector diagonalization technique presented above, the SVD method also makes the following claim: *the SVD makes it possible for every matrix to be diagonal if the proper bases for the domain and range are used.* To consider this statement more formally, consider that since $\mathbf{U}$ and $\mathbf{V}$ are orthonormal bases in $\mathbb{C}^{m\times m}$ and $\mathbb{C}^{n\times n}$ respectively, then any vector in these spaces can be expanded in their bases. Specifically, consider a vector $\mathbf{b} \in \mathbb{C}^{m\times m}$ and $\mathbf{x} \in \mathbb{C}^{n\times n}$, then each can be expanded in the bases of $\mathbf{U}$ and $\mathbf{V}$ so that

$$\mathbf{b} = \mathbf{U}\hat{\mathbf{b}}, \;\; \mathbf{x} = \mathbf{V}\hat{\mathbf{x}} \quad (15.2.100)$$

where the vectors $\hat{\mathbf{b}}$ and $\hat{\mathbf{x}}$ give the weightings for the orthonormal bases expansion. Now consider the simple equation:

$$\begin{aligned} \mathbf{Ax} = \mathbf{b} \quad \rightarrow \quad & \mathbf{U}^*\mathbf{b} = \mathbf{U}^*\mathbf{Ax} \\ & \mathbf{U}^*\mathbf{b} = \mathbf{U}^*\mathbf{U\Sigma V}^*\mathbf{x} \\ & \hat{\mathbf{b}} = \mathbf{\Sigma}\hat{\mathbf{x}} \, . \end{aligned} \quad (15.2.101)$$

Thus the last line shows $\mathbf{A}$ reduces to the diagonal matrix $\mathbf{\Sigma}$ when the range is expressed in terms of the of the basis vectors of $\mathbf{U}$ and the domain is expressed in terms of the basis vectors of $\mathbf{V}$.

Thus matrices can be diagonalized via either an eigenvalue decomposition or an SVD decomposition. However, there are three key differences in the diagonalization process.

- The SVD performs the diagonalization using two different bases, $\mathbf{U}$ and $\mathbf{V}$, while the eigenvalue method uses a single basis $\mathbf{X}$.

- The SVD method uses an orthonormal basis while the basis vectors in $\mathbf{X}$, while linearly independent, are not generally orthogonal.
- Finally, the SVD is guaranteed to exist for any matrix $\mathbf{A}$ while the same is not true, even for square matrices, for the eigenvalue decomposition.

Useful theorems of SVD

Having established the similarities and connections between eigenvalues and singular values, in what follows a number of important theorems are outlined concerning the SVD. These theorems are important for several reasons. First, the theorems play a key role in the numerical evaluation of many matrix properties via the SVD. Second, the theorems guarantee certain behaviors of the SVD that can be capitalized upon for future applications. Here is a list of important results. A more detailed account is given by Trefethen and Bau [30].

Theorem: *If the rank of* $\mathbf{A}$ *is* r*, then there are* r *nonzero singular values.*

The proof of this is based upon the fact that the rank of a diagonal matrix is equal to the number of its nonzero entries. And since the decomposition $\mathbf{A} = \mathbf{U\Sigma V}^*$ where $\mathbf{U}$ and $\mathbf{V}$ are full rank, then rank($\mathbf{A}$)=rank($\mathbf{\Sigma}$)=r. As a side note, the rank of a matrix can be found with the MATLAB command:

```
rank(A)
```

The standard algorithm used to compute the rank is based upon the SVD and the computation of the nonzero singular values. Thus the theorem is of quite useful in practice.

Theorem: *The* range($\mathbf{A}$)=$\langle \mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_r \rangle$ *and* null ($\mathbf{A}$)=$\langle \mathbf{v}_{r+1}, \mathbf{v}_{r+2}, \cdots, \mathbf{v}_n \rangle$.

Note that the range and null space come from the two expansion bases $\mathbf{U}$ and $\mathbf{V}$. The range and null space can be found in MATLAB with the commands:

```
range(A)
null(A)
```

This theorem also serves as the most accurate computation basis for determining the range and null space of a given matrix $\mathbf{A}$ via the SVD.

Theorem: *The norm* $\|\mathbf{A}\|_2 = \sigma_1$ *and* $\|\mathbf{A}\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}$.

These norms are known as the 2-norm and the Frobenius-norm respectively. They essentially measure the *energy* of a matrix. Although it is difficult to conceptualize abstractly what this means, it will become much more clear in the context of given applications. This result is established by the fact that $\mathbf{U}$ and $\mathbf{V}$ are unitary operators so that their norm is unity. Thus with $\mathbf{A} = \mathbf{U\Sigma V}^*$, the norm $\|\mathbf{A}\|_2 = \|\mathbf{\Sigma}\|_2 = \max\{|\sigma_j|\} = \sigma_1$. A similar reasoning holds for the Frobenius norm definition. The norm can be calculated in MATLAB with

```
norm(A)
```

Notice that the Frobenius norm contains the total matrix energy while the 2-norm definition contains the energy of the largest singular value. The ratio $\|\mathbf{A}\|_2/\|\mathbf{A}\|_F$ effectively measures the portion of the energy in the semiaxis $\mathbf{u}_1$. This fact will be tremendously important for us. Furthermore, this theorem gives the standard way for computing matrix norms.

Theorem: *The nonzero singular values of* $\mathbf{A}$ *are the square roots of the nonzero eigenvalues of* $\mathbf{A}^*\mathbf{A}$ *or* $\mathbf{AA}^*$. *(These matrices have the same nonzero eigenvalues).*

This has already been shown in the calculation for actually determining the SVD. In particular, this process is illustrated in Eqs. (15.1.80) and (15.1.79). This theorem is important for actually producing the unitary matrices $\mathbf{U}$ and $\mathbf{V}$.

Theorem: *If* $\mathbf{A} = \mathbf{A}^*$ *(self-adjoint), then the singular values of* $\mathbf{A}$ *are the absolute values of the eigenvalues of* $\mathbf{A}$.

As with most self-adjoint problems, there are very nice properties to the matrices, such as the above theorem. Eigenvalues can be computed with the command:

```
eig(A)
```

Alternatively, one can use the **eigs** to specify the number of eigenvalues desired and their specific ordering.

Theorem: *For* $\mathbf{A} \in \mathbb{C}^{m\times m}$, *the determinant is given by* $|\det(A)| = \prod_{j=1}^{m} \sigma_j$.

Again, due to the fact that the matrices $\mathbf{U}$ and $\mathbf{V}$ are unitary, their determinants are unity. Thus $|\det(A)| = |\det(\mathbf{U\Sigma V}^*)| = |\det(\mathbf{U})||\det(\mathbf{\Sigma})||\det(\mathbf{V}^*)| = |\det(\mathbf{\Sigma})| = \prod_{j=1}^{m} \sigma_j$. Thus even determinants can be computed via the SVD and its singular values.

Low dimensional reductions

Now comes the last, and most formative, property associated with the SVD: *low-dimensional approximations* to high-degree of freedom or complex systems. In linear algebra terms, this is also *low-rank approximations.* The interpretation of the theorems associated with these low-dimensional reductions are critical for the use and implementation of the SVD. Thus we consider the following:

Theorem: **A** *is the sum of* r *rank-one matrices*

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^* \, . \tag{15.2.102}$$

There are a variety of ways to express the $m \times n$ matrix **A** as a sum of rank one matrices. The bottom line is this: *the* N*th partial sum captures as much of the matrix* **A** *as possible.* Thus the partial sum of the rank one matrices is an important object to consider. This leads to the following theorem:

Theorem: *For any* N *so that* $0 \leq N \leq r$*, we can define the partial sum*

$$\mathbf{A}_N = \sum_{j=1}^{N} \sigma_j \mathbf{u}_j \mathbf{v}_j^* \, . \tag{15.2.103}$$

And if $N = \min\{m, n\}$*, define* $\sigma_{N+1} = 0$*. Then*

$$\|A - A_N\|_2 = \sigma_{N+1} \, . \tag{15.2.104}$$

Likewise, if using the Frobenius norm, then

$$\|A - A_N\|_N = \sqrt{\sigma_{N+1}^2 + \sigma_{N+2}^2 + \cdots + \sigma_r^2} \, . \tag{15.2.105}$$

Interpreting this theorem is critical. Geometrically, we can ask *what is the best approximation of a hyperellipsoid by a line segment? Simply take the line segment to be the longest axis, i.e. that associated with the singular value* σ_1*.* Continuing this idea, what is the best approximation by a two-dimensional ellipse? Take the longest and second longest axes, i.e. those associated with the singular values σ_1 and σ_2. After r steps, the total energy in **A** is completely captured. **Thus the SVD gives a type of least-square fitting algorithm, allowing us to project the matrix onto low-dimensional representations in a formal, algorithmic way.** Herein lies the ultimate power of the method.

15.3 Introduction to Principal Component Analysis (PCA)

To make explicit the concept of the SVD, a simple model example will be formulated that will illustrate all the key concepts associated with the SVD. The model to be considered will be a simple spring-mass system as illustrated in Fig. 156. Of course, this is a fairly easy problem to solve from basic concepts of $\mathbf{F} = m\mathbf{a}$. But for the moment, let's suppose we didn't know the governing

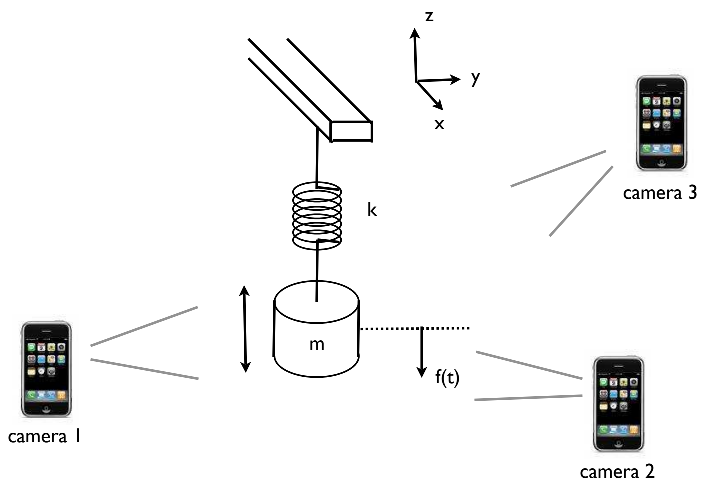


Figure 156: A prototypical example of how we might apply a principal component analysis, or SVD, is the simple mass-spring system exhibited here. The mass m is suspended with a spring with Hook's constant k. Three video cameras collect data about its motion in order to ascertain its governing equations.

equations. In fact, our aim in this lecture is to use a number of cameras (probes) to extract out data concerning the behavior of the system and then to extract empirically the governing equations of motion.

This prologue highlights one of the key applications of the SVD, or alternatively a variant of *Principal Component Analysis* (PCA), *Proper Mode Decomposition* (POD), *Empirical Mode Decomposition*, *Hotelling Transform*, *Empirical Orthogonal Functions* (EOF), *reduced order modeling* (ROM), *dimensionality reduction* or *Karhunen-Loéve Decomposition* as it also known in the literature. Namely, from seemingly complex, perhaps random data, can low-dimensional reductions of the dynamics and behavior be produced when the governing equations are not known? Such methods can be used to quantify low-dimensional dynamics arising in such areas as turbulent fluid flows [32], structural vibrations [33, 34], insect locomotion [35], damage detection [36], and neural decision making strategies [37] to name just a few areas of application. It will also become obvious as we move forward on this topic that the breadth of applications is staggering and includes image processing and signal analysis. Thus the perspective to be taken here is clearly one in which the data analysis of an unknown, but potentially low-dimensional system is to be analyzed.

Again we turn our attention to the simple experiment at hand: a mass suspended by a spring as depicted in Fig. 156. If the mass is perturbed or taken from equilibrium in the z-direction only, we know that the governing equations are simply

$$\frac{d^2f(t)}{dt^2} = -\omega^2 f(t) \tag{15.3.106}$$

where the function $f(t)$ measures the displacement of the mass in the z-direction as a function of time. This has the well known solution (in amplitude-phase form)

$$f(t) = A\cos(\omega t + \omega_0) \tag{15.3.107}$$

where the values of A and ω_0 are determined from the initial state of the system. This essentially states that the state of the system can be described by a one-degree of freedom system.

In the above analysis, there are many things we have ignored. Included in the list of things we have ignored is the possibility that the initial excitation of the system actually produces movement in the $x-y$ plane. Further, there is potentially noise in the data from, for instance, shaking of the cameras during the video capture. Moreover, from what we know of the solution, only a single camera is necessary to capture the underlying motion. In particular, a single camera in the $x-y$ plane at $z=0$ would be ideal. Thus we have over-sampled the data with three cameras and have produced redundant data sets. From all of these potential perturbations and problems, it is our goal to extract out the simple solution given by the simple harmonic motion.

This problem is a good example of what kind of processing is required to analyze a realistic data set. Indeed, one can imagine that most data will be quite noisy, perhaps redundant, and certainly not produced from an optimal viewpoint. But through the process of PCA, these can be circumvented in order to extract out the ideal or simplified behavior. Moreover, we may even learn how to transform the data into the optimal viewpoint for analyzing the data.

Data collection and ordering

Assume now that we have started the mass in motion by applying a small perturbation in the z-direction only. Thus the mass will begin to oscillate. Three cameras are then used to record the motion. Each camera produces a two-dimensional representation of the data. If we denote the data from the three cameras with subscripts a, b and c, then the data collected are represented by the following:

$$\text{camera 1: } (\mathbf{x}_a, \mathbf{y}_a) \tag{15.3.108a}$$
$$\text{camera 2: } (\mathbf{x}_b, \mathbf{y}_b) \tag{15.3.108b}$$
$$\text{camera 3: } (\mathbf{x}_c, \mathbf{y}_c) \tag{15.3.108c}$$

where each set $(\mathbf{x}_j, \mathbf{y}_j)$ is data collected over time of the position in the $x-y$ plane of the camera. Note that this is not the same $x-y$ plane of the oscillating mass system as shown in Fig. 156. Indeed, we should pretend we don't know the correct $x - y - z$ coordinates of the system. Thus the camera positions and their relative $x - y$ planes are arbitrary. The length of each vector $\mathbf{x}_i$ and $\mathbf{y}_i$ depends on the data collection rate and the length of time the dynamics is observed. We denote the length of these vectors as n.

All the data collected can then be gathered into a single matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{y}_a \\ \mathbf{x}_b \\ \mathbf{y}_b \\ \mathbf{x}_c \\ \mathbf{y}_c \end{bmatrix} \tag{15.3.109}$$

Thus the matrix $X \in \mathbb{R}^{m \times n}$ where m represents the number of measurement types and n is the number of data points taken from the camera over time.

Now that the data has been arranged, two issues must be addressed: noise and redundancy. Everyone has an intuitive concept that noise in your data can only deteriorate, or corrupt, your ability to extract the true dynamics. Just as in image processing, noise can alter an image beyond restoration. Thus there is also some idea that if the measured data is too noisy, fidelity of the underlying dynamics is compromised from a data analysis point of view. The key measure of this is the so-called signal-to-noise ratio: SNR$= \sigma^2_{\text{signal}}/\sigma^2_{\text{noise}}$, where the ratio is given as the ratio of variances of the signal and noise fields. A high SNR (much greater than unity) gives almost noiseless (high precision) data whereas a low SNR suggests the underlying signal is corrupted by the noise. The second issue to consider is redundancy. In the example of Fig. 156, the single degree of freedom is sampled by three cameras, each of which is really recording the same single degree of freedom. Thus the measurements should be rife with redundancy, suggesting that the different measurements are statistically dependent. Removing this redundancy is critical for data analysis.

The covariance matrix

An easy way to identify redundant data is by considering the covariance between data sets. Recall from our early lectures on probability and statistics that the covariance measures the statistical dependence/independence between two variables. Obviously, strongly statistically dependent variables can be considered as redundant observations of the system. Specifically, consider two sets of measurements with zero means expressed in row vector form:

$$\mathbf{a} = [a_1 \; a_2 \; \cdots \; a_n] \;\text{ and }\; \mathbf{b} = [b_1 \; b_2 \; \cdots \; b_n] \tag{15.3.110}$$

where the subscript denotes the sample number. The variances of $\mathbf{a}$ and $\mathbf{b}$ are given by

$$\sigma_{\mathbf{a}}^2 = \frac{1}{n-1}\mathbf{a}\mathbf{a}^T \tag{15.3.111a}$$

$$\sigma_{\mathbf{b}}^2 = \frac{1}{n-1}\mathbf{b}\mathbf{b}^T \tag{15.3.111b}$$

while the covariance between these two data sets is given by

$$\sigma_{\mathbf{ab}}^2 = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T \tag{15.3.112}$$

where the normalization constant of $1/(n-1)$ is for an unbiased estimator.

We don't just have two vectors, but potentially quite a number of experiments and data that would need to be correlated and checked for redundancy. In fact, the matrix in Eq. (15.3.109) is exactly what needs to be checked for covariance. The appropriate *covariance matrix* for this case is then

$$\mathbf{C}_{\mathbf{X}} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T\,. \tag{15.3.113}$$

This is easily computed with MATLAB from the command line:

```
cov(X)
```

The covariance matrix $\mathbf{C}_{\mathbf{X}}$ is a square, symmetric $m \times m$ matrix whose diagonal represents the variance of particular measurements. The off-diagonal terms are the covariances between measurement types. Thus $\mathbf{C}_{\mathbf{X}}$ captures the correlations between all possible pairs of measurements. Redundancy is thus easily captured since if two data sets are identical (identically redundant), the off-diagonal term and diagonal term would be equal since $\sigma_{\mathbf{ab}}^2 = \sigma_{\mathbf{a}}^2 = \sigma_{\mathbf{b}}^2$ if $\mathbf{a} = \mathbf{b}$. Thus large off-diagonal terms correspond to redundancy while small off-diagonal terms suggest that the two measured quantities are close to statistically independent and have low redundancy. It should also be noted that large diagonal terms, or those with large variances, typically represent what we might consider *the dynamics of interest* since the large variance suggests strong fluctuations in that variable. Thus the covariance matrix is the key component to understanding the entire data analysis.

Achieving the goal: The insight given by the covariance matrix leads to our ultimate aim of

i) removing redundancy
ii) identifying those signals with maximal variance.

Thus in a mathematical sense, we are simply asking to represent $\mathbf{C_X}$ so that the diagonals are ordered from largest to smallest and the off-diagonals are zero, i.e. our task is to diagonalize the covariance matrix. This is *exactly* what the SVD does, thus allowing it to becomes the tool of choice for data analysis and dimensional reduction. In fact, the SVD diagonalizes and each singular direction captures as much energy as possible as measured by the singular values σ_j.

15.4 Principal Components, Diagonalization and SVD

The example presented in the previous section shows that the key to analyzing a given experiment is to consider the covariance matrix

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \, . \tag{15.4.114}$$

where the matrix $\mathbf{X}$ contains the experimental data of the system. In particular, $\mathbf{X} \in \mathbb{C}^{m\times n}$ where m are the number of probes or measuring positions, and n is the number of experimental data points taken at each location.

In this setup, the following facts are highlighted:

- $\mathbf{C_X}$ is a square, symmetric $m \times m$ matrix.
- The diagonal terms of $\mathbf{C_X}$ are the variances for particular measurements. By assumption, large variances correspond to *dynamics of interest*, whereas low variances are assumed to correspond to *non-interesting dynamics.*
- The off-diagonal terms of $\mathbf{C_X}$ are the covariance between measurements. Indeed, the off-diagonals capture the correlations between all possible pairs of measurements. A large off-diagonal term represents two events that have a high-degree of redundancy, whereas a small off-diagonal coefficient means there is little redundancy in the data, i.e. they are statistically independent.

Diagonalization

The concept of diagonalization is critical for understanding the underpinnings of many physical systems. In this process of diagonalization, the correct coordinates, or basis functions, are revealed that reduce the given system to its low-dimensional essence. There is more than one way to diagonalize a matrix, and this is certainly true here as well since the constructed covariance matrix $\mathbf{C_X}$ is square and symmetric, both properties that are especially beneficial for standard eigenvalue/eigenvector expansion techniques.

The key idea behind the diagonalization is simply this: there exists an *ideal* basis in which the $\mathbf{C_X}$ can be written (diagonalized) so that in this basis, all redundancies have been removed, and the largest variances of particular measurements are ordered. In the language being developed here, this means that the system has been written in terms of its *principal components*, or in a *proper orthogonal decomposition.*

Eigenvectors and eigenvalues: The most straightforward way to diagonalize the covariance matrix is by making the observation that $\mathbf{X}\mathbf{X}^T$ is a square, symmetric $m \times m$ matrix, i.e. it is self-adjoint so that the m eigenvalues are real and distinct. Linear algebra provides theorems which state that such a matrix can be rewritten as

$$\mathbf{X}\mathbf{X}^T = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1} \tag{15.4.115}$$

as stated in Eq. (15.2.94) where the matrix $\mathbf{S}$ is a matrix of the eigenvectors of $\mathbf{X}\mathbf{X}^T$ arranged in columns. Since it is a symmetric matrix, these eigenvector columns are orthogonal so that ultimately the $\mathbf{S}$ can be written as a unitary matrix with $\mathbf{S}^{-1} = \mathbf{S}^T$. Recall that the matrix $\mathbf{\Lambda}$ is a diagonal matrix whose entries correspond to the m distinct eigenvalue of $\mathbf{X}\mathbf{X}^T$.

This suggests that instead of working directly with the matrix $\mathbf{X}$, we consider working with the transformed variable, or in the principal component basis,

$$\mathbf{Y} = \mathbf{S}^T\mathbf{X}\,. \tag{15.4.116}$$

For this new basis, we can then consider its covariance

$$\begin{aligned}
\mathbf{C}_\mathbf{Y} &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\
&= \frac{1}{n-1}(\mathbf{S}^T\mathbf{X})(\mathbf{S}^T\mathbf{X})^T \\
&= \frac{1}{n-1}\mathbf{S}^T(\mathbf{X}\mathbf{X}^T)\mathbf{S} \\
&= \frac{1}{n-1}\mathbf{S}^T\mathbf{S}\mathbf{\Lambda}\mathbf{S}\mathbf{S}^T \\
\mathbf{C}_\mathbf{Y} &= \frac{1}{n-1}\mathbf{\Lambda}\,.
\end{aligned} \tag{15.4.117}$$

In this basis, the *principal components* are the eigenvectors of $\mathbf{X}\mathbf{X}^T$ with the interpretation that the jth diagonal value of $\mathbf{C}_\mathbf{Y}$ is the variance of $\mathbf{X}$ along $\mathbf{x}_j$, the jth column of $\mathbf{S}$. The following codes of line produce the principal components of interest.

```
[m,n]=size(X);   %  compute data size
mn=mean(X,2); %  compute mean for each row
X=X-repmat(mn,1,n);  % subtract mean

Cx=(1/(n-1))*X*X';   % covariance
[V,D]=eig(Cx);      % eigenvectors(V)/eigenvalues(D)
lambda=diag(D);    % get eigenvalues
```

```
[dummy,m_arrange]=sort(-1*lambda);  % sort in decreasing order
lambda=lambda(m_arrange);
V=V(:,m_arrange);

Y=V'*X;  % produce the principal components projection
```

This simple code thus produces the eigenvalue decomposition and the projection of the original data onto the principal component basis.

Singular value decomposition: A second method for diagonalizing the covariance matrix is the SVD method. In this case, the SVD can diagonalize any matrix by working in the appropriate pair of bases $\mathbf{U}$ and $\mathbf{V}$ as outlined in the first lecture of this section. Thus by defining the transformed variable

$$\mathbf{Y} = \mathbf{U}^*\mathbf{X} \tag{15.4.118}$$

where $\mathbf{U}$ is the unitary transformation associated with the SVD: $\mathbf{X} = \mathbf{U\Sigma V}^*$. Just as in the eigenvalue/eigenvector formulation, we then compute the variance in $\mathbf{Y}$:

$$\begin{aligned}
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{YY}^T \\
&= \frac{1}{n-1}(\mathbf{U}^*\mathbf{X})(\mathbf{U}^*\mathbf{X})^T \\
&= \frac{1}{n-1}\mathbf{U}^*(\mathbf{XX}^T)\mathbf{U} \\
&= \frac{1}{n-1}\mathbf{U}^*\mathbf{U\Sigma}^2\mathbf{UU}^* \\
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{\Sigma}^2 .
\end{aligned} \tag{15.4.119}$$

This makes explicit the connection between the SVD and the eigenvalue method, namely that $\mathbf{\Sigma}^2 = \mathbf{\Lambda}$. The following codes of line produce the principal components of interest using the SVD (Assume that you have the first three lines from the previous MATLAB code).

```
[u,s,v]=svd(X'/sqrt(n-1));  % perform the SVD
lambda=diag(s).^2;  % produce diagonal variances
Y=u'*X;  % produce the principal components projection
```

This gives the SVD method for producing the principal components. Overall, the SVD method is the more robust method and should be used. However, the connection between the two methods becomes apparent in these calculations.

Spring Experiment

To illustrate this completely in practice, three experiments will be performed in class with the configuration of Fig. 156. The following experiments will attempt to illustrate various aspects of the PCA and its practical usefulness and the effects of noise on the PCA algorithms.

- **Ideal case:** Consider a small displacement of the mass in the z direction and the ensuing oscillations. In this case, the entire motion is in the z direction with simple harmonic motion being observed.
- **noisy case:** Repeat the ideal case experiment, but this time, introduce camera shake into the video recording. This should make it more difficult to extract the simple harmonic motion. But if the shake isn't too bad, the dynamics will still be extracted with the PCA algorithms.
- **horizontal displacement:** In this case, the mass is released off-center so as to produce motion in the $x-y$ plane as well as the z direction. Thus there is both a pendulum motion and a simple harmonic oscillation. See what the PCA tells us about the system.

In order to extract out the mass movement from the video frames, the following MATLAB code is needed. This code is a generic way to read in movie files to MATLAB for post-processing.

```
obj=mmreader('matlab_test.mov')

vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1 : numFrames
    mov(k).cdata = vidFrames(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames
  X=frame2im(mov(j));
  imshow(X); drawnow
end
```

This multi-media reader command **mmreader** simply characterizes the file type and its attributes. The bulk of time in this is the **read** command which actually uploads the movie frames into the MATLAB desktop for processing. Once the frames are extracted, they can again be converted to double precision numbers for mathematical processing. In this case, the position of the mass is to be determined from each frame. This basic shell of code is enough to begin the process of extracting the spring-mass system information.

15.5 Principal Components and Proper Orthogonal Modes

Now that the basic framework of the principal component analysis and its relation to the SVD has been laid down, a few remaining issues need to be addressed. In particular, the principal component analysis seems to suggest that we are simply expanding our solution in another *orthonormal* basis, one which can always diagonalize the underlying system.

Mathematically, we can consider a given function $f(x, t)$ over a domain of interest. In most applications, the variables x and t will refer to the standard space-time variables we are familiar with. The idea is to then expand this function in some basis representation so that

$$f(x,t) \approx \sum_{j=1}^{N} a_j(t)\phi_j(x) \tag{15.5.120}$$

where N is the total number of modes, $\phi_j(x)$, to be used. The remaining function $a_j(t)$ determines the weights of the spatial modes.

The expansion (15.5.120) is certainly not a new idea to us. Indeed, we have been using this concept extensively already with Fourier transforms, for instance. Specifically, here are some of the more common expansion bases used in practice:

$$\begin{aligned}
\phi_j(x) &= (x - x_0)^j \quad \text{Taylor expansion} && (15.5.121\text{a})\\
\phi_j(x) &= \cos(jx) \quad \text{Discrete cosine transform} && (15.5.121\text{b})\\
\phi_j(x) &= \sin(jx) \quad \text{Discrete sine transform} && (15.5.121\text{c})\\
\phi_j(x) &= \exp(jx) \quad \text{Fourier transform} && (15.5.121\text{d})\\
\phi_j(x) &= \psi_{a,b}(x) \quad \text{Wavelet transform} && (15.5.121\text{e})\\
\phi_j(x) &= \phi_{\lambda_j}(x) \quad \text{Eigenfunction expansion} && (15.5.121\text{f})
\end{aligned}$$

where $\phi_{\lambda_j}(x)$ are eigenfunctions associated with the underlying system. This places the concept of a basis function expansion in familiar territory. Further, it shows that such a basis function expansion is not unique, but rather can potentially have an infinite number of different possibilities.

As for the weighting coefficients $a_j(t)$, they are simply determined from the standard inner product rules and the fact that the basis functions are orthonormal (Note that they are not written this way above):

$$\int \phi_j(x)\phi_n(x)dx = \begin{cases} 1 & j = n \\ 0 & j \neq n \end{cases} \; . \tag{15.5.122}$$

This then gives for the coefficients

$$a_j(t) = \int f(x,t)\phi_j(x)dx \tag{15.5.123}$$

and the basis expansion is completed.

Interestingly enough, the basis functions selected are chosen often for simplicity and/or their intuitive meaning for the system. For instance, the Fourier transform very clearly highlights the fact that the given function has a representation in terms of *frequency* components. This is fundamental for understanding many physical problems as there is clear and intuitive meaning to the Fourier modes.

In contrast to selecting basis functions for simplicity or intuitive meaning, the broader question can be asked: what criteria should be used for selecting the functions $\phi_j(x)$. This is an interesting question given that any complete basis can approximate the function $f(x,t)$ to any desired accuracy given N large enough. But what is desired is the best basis functions such that, with N as small as possible, we achieve the desired accuracy. The goal is then the following: find a sequence of orthonormal basis functions $\phi_j(x)$ so that the first two terms give the best two-term approximation to $f(x,t)$, or the first five terms give the best five-term approximation to $f(x,t)$. These special, ordered, orthonormal functions are called the *proper orthogonal modes* (POD) for the function $f(x,t)$. With these modes, the expansion (15.5.123) is called the POD of $f(x,t)$. The relation to the SVD will become obvious momentarily.

Example 1: Consider an approximation to the following surface

$$f(x,t) = e^{-|(x-0.5)(t-1)|} + \sin(xt) \quad x \in [0,1],\, t \in [0,2]. \tag{15.5.124}$$

As listed above, there are a number of methods available for approximating this function using various basis functions. And all of the ones listed are guaranteed to converge to the surface for a large enough N in (15.5.120).

In the POD method, the surface is first discretized and approximated by a finite number of points. In particular, the following discretization will be used:

```
x=linspace(0,1,25);
t=linspace(0,2,50);
```

where x has been discretized into 25 points and t is discretized into 50 points. The surface is represented in Fig. 157(a) over the domain of interest. The surface is constructed by using the **meshgrid** command as follows:

```
[T,X]=meshgrid(t,x);
f=exp(-abs((X-.5).*(T-1)))+sin(X.*T);
subplot(2,2,1)
surfl(X,T,f), shading interp, colormap(gray)
```

The **surfl** produces a lighted surface that in this case is represented in grayscale. Note that in producing this surface, the matrix **f** is a 25×50 matrix so that the x-values are given as row locations while the t-values are given by column locations.

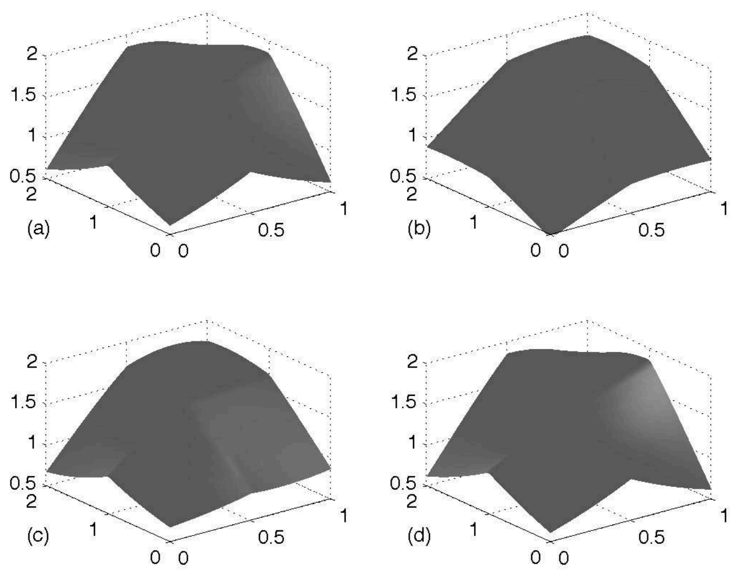


Figure 157: Representation of the surface (a) by a series of low-rank (low-dimensional) approximations with (b) one-mode, (c) two-modes and (d) three-modes. The energy captured in the first mode is approximately 92% of the total surface energy. The first three modes together capture 99.9% of the total surface energy, thus suggesting that the surface can be easily and accurately represented by a three-mode expansion.

The basis functions are then computed using the SVD. Recall that the SVD produces ordered singular values and associated orthonormal basis functions that capture as much energy as possible. Thus an SVD can be performed on the matrix **f** that represents the surface.

```
[u,s,v]=svd(f);  % perform SVD

for j=1:3
   ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)'; % modal projections
   subplot(2,2,j+1)
   surfl(X,T,ff), shading interp, colormap(gray)
   set(gca,'Zlim',[0.5 2])
end
subplot(2,2,1), text(-0.5,1,0.5,'(a)','Fontsize',[14])
subplot(2,2,2), text(-0.5,1,0.5,'(b)','Fontsize',[14])
```

```
   subplot(2,2,3), text(-0.5,1,0.5,'(c)','Fontsize',[14])
   subplot(2,2,4), text(-0.5,1,0.5,'(d)','Fontsize',[14])
```

The SVD command pulls out the diagonal matrix along with the two unitary matrices $\mathbf{U}$ and $\mathbf{V}$. The loop above processes the sum of the first, second and third modes. This gives the POD modes of interest. Figure 157 demonstrates the modal decomposition and the accuracy achieved with representing the surface with one, two and three POD modes. The first mode alone captures 92% of the surface while three modes produce a staggering 99.9% of the energy. This should be contrasted with Fourier methods, for instance, which would require potentially hundreds of modes to achieve such accuracy. *As stated previously, these are the best one, two and three mode approximations of the function $f(x,t)$ that can be achieved.*

To be more precise about the nature of the POD, consider the *energy* in each mode. This can be easily computed, or already has been computed, in the SVD, i.e. they are the singular values σ_j. In MATLAB, the energy in the one mode and three mode approximation is computed from

```
   energy1=sig(1)/sum(sig)
   energy3=sum(sig(1:3))/sum(sig)
```

More generally, the entire *spectrum* of singular values can be plotted. Figure 158 shows the complete spectrum of singular values on a standard plot and a log plot. The clear dominance of a single mode is easily deduced. Additionally, the first three POD modes are illustrated: they are the first three columns of the matrix $\mathbf{U}$. These constitute the orthonormal expansion basis of interest. The code for producing these plots is as follows:

```
  sig=diag(s);
  subplot(2,2,1), plot(sig,'ko','Linewidth',[1.5])
  axis([0 25 0 50])
  set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
  text(20,40,'(a)','Fontsize',[13])
  subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
  axis([0 25 10^(-18) 10^(5)])
  set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
      'Xtick',[0 5 10 15 20 25]);
  text(20,10^0,'(b)','Fontsize',[13])

  subplot(2,1,2)
  plot(x,u(:,1),'k',x,u(:,2),'k--',x,u(:,3),'k:','Linewidth',[2])
  set(gca,'Fontsize',[13])
  legend('mode 1','mode 2','mode 3','Location','NorthWest')
  text(0.8,0.35,'(c)','Fontsize',[13])
```

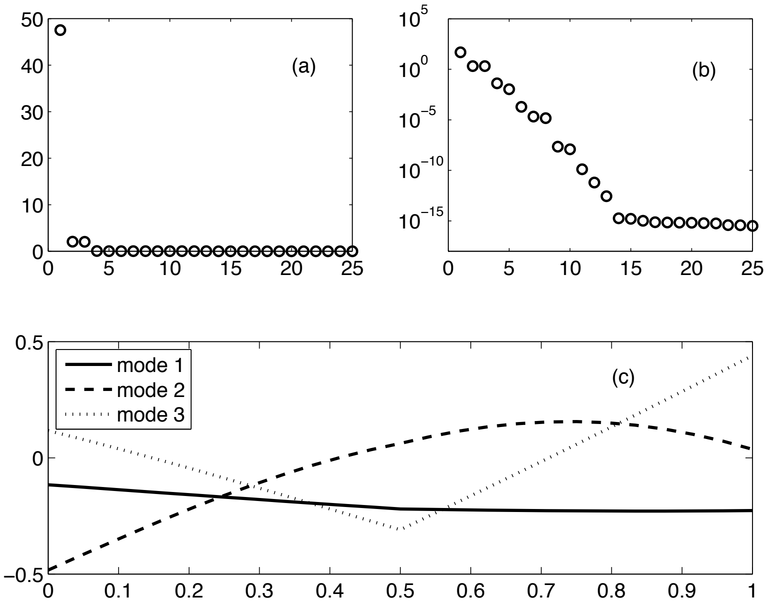


Figure 158: Singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. For the surface considered in Fig. 157, the bulk of the energy is in the first POD mode. A three mode approximation produces 99.9% of the energy. The linear POD modes are illustrated in panel (c).

Note that the key part of the code is simply the calculation of the POD modes which are weighted, in practice, by the singular values and the their evolution in time as given by the columns of $\mathbf{V}$.

Example 2: Consider the following time-space function

$$f(x,t) = [1 - 0.5\cos 2t]\text{sech}x + [1 - 0.5\sin 2t]\text{sech}x\tanh x\,. \qquad (15.5.125)$$

This represents an asymmetric, time-periodic breather of a localized solution. Such solutions arise in a myriad of contexts are often obtained via full numerical simulations of an underlying physical system. Let's once again apply the techniques of POD analysis to see what is involved in the dynamics of this system. A discretization of the system if first applied and made into a two-dimensional representation in time and space.

```
x=linspace(-10,10,100);
t=linspace(0,10,30);
[X,T]=meshgrid(x,t);
f=sech(X).*(1-0.5*cos(2*T))+(sech(X).*tanh(X)).*(1-0.5*sin(2*T));
```

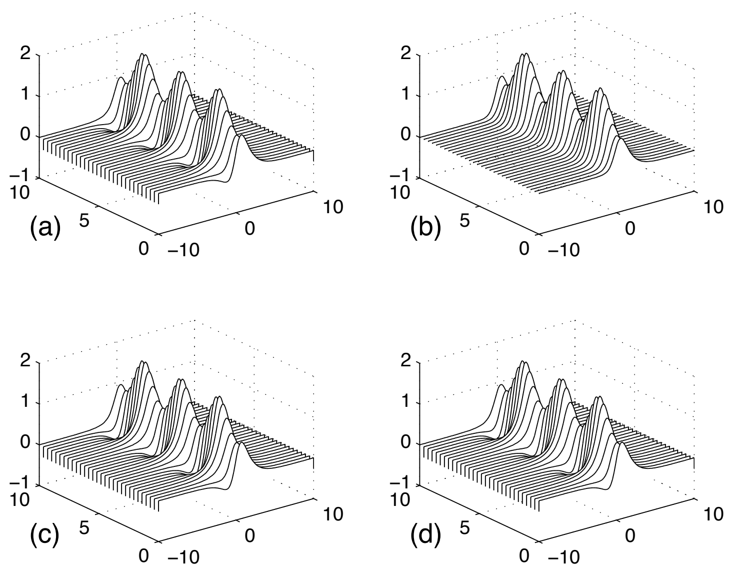


Figure 159: Representation of the time-space dynamics (a) by a series of low-rank (low-dimensional) approximations with (b) one-mode, (c) two-modes and (d) three-modes. The energy captured in the first mode is approximately 83% of the total energy. The first two modes together capture 100% of the surface energy, thus suggesting that the surface can be easily and accurately represented by a simple two-mode expansion.

```
subplot(2,2,1), waterfall(X,T,f), colormap([0 0 0])
```

Note that unlike before the matrix **f** is now a 30×100 matrix so that the x-values are along the columns and t-values are along the rows. This is done simply so that we can use the **waterfall** command in MATLAB.

The singular value decomposition of the matrix **f** produces the quantities of interest. In this case, since we want the x-value in the rows, the transpose of the matrix is considered in the SVD.

```
[u,s,v]=svd(f');
for j=1:3
  ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)';
  subplot(2,2,j+1)
  waterfall(X,T,ff'), colormap([0 0 0]), set(gca,'Zlim',[-1 2])
end
```

```
subplot(2,2,1), text(-19,5,-1,'(a)','Fontsize',[14])
subplot(2,2,2), text(-19,5,-1,'(b)','Fontsize',[14])
subplot(2,2,3), text(-19,5,-1,'(c)','Fontsize',[14])
subplot(2,2,4), text(-19,5,-1,'(d)','Fontsize',[14])
```

This produces the original function along with the first three modal approximations of the function. As is shown in Fig. 159, the two-mode and three-mode approximations appear to be identical. In fact, they are. The singular values of the SVD show that ≈83% of the energy is in the first mode of Fig. 159(a) while 100% of the energy is captured by a two mode approximation as shown in Fig. 159(b). Thus the third mode is completely unnecessary. The singular values are produced with the following lines of code.

```
figure(2)
sig=diag(s);
subplot(3,2,1), plot(sig,'ko','Linewidth',[1.5])
axis([0 25 0 50])
set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
text(20,40,'(a)','Fontsize',[13])
subplot(3,2,2), semilogy(sig,'ko','Linewidth',[1.5])
axis([0 25 10^(-18) 10^(5)])
set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
  'Xtick',[0 5 10 15 20 25]);
text(20,10^0,'(b)','Fontsize',[13])

energy1=sig(1)/sum(sig)
energy2=sum(sig(1:2))/sum(sig)
```

As before, we can also produce the first two modes, which are the first two columns of **U**, along with their time behavior, which are the first two columns of **V**.

```
subplot(3,1,2)  % spatial modes
plot(x,u(:,1),'k',x,u(:,2),'k--','Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','mode 2','Location','NorthWest')
text(8,0.35,'(c)','Fontsize',[13])

subplot(3,1,3)  % time behavior
plot(t,v(:,1),'k',t,v(:,2),'k--','Linewidth',[2])
text(9,0.35,'(d)','Fontsize',[13])
```

Figure 160 shows the singular values along with the spatial and temporal behavior of the first two modes. It is not surprising that the SVD characterizes the total behavior of the system as an exactly (to numerical precision) two mode

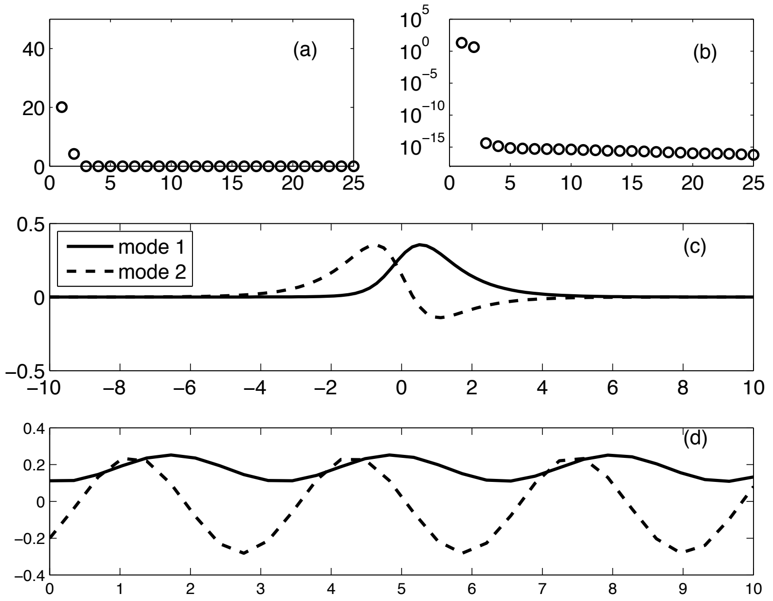


Figure 160: Singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. For the space-time surface considered in Fig. 159, the bulk of the energy is in the first POD mode. A two mode approximation produces 100% of the energy. The linear POD modes are illustrated in panel (c) which their time evolution behavior in panel (d).

behavior. Recall that is exactly what we started with! Note however, that our original two modes are different than the SVD modes. Indeed, the first SVD mode is asymmetric unlike our symmetric hyperbolic secant.

Summary and Limits of SVD/PCA/POD

The methods of PCA and POD, which are essentially related to the idea of diagonalization of any matrix via the SVD, are exceptionally powerful tools and methods for the evaluation of data driven systems. Further, they provide the most precise method for performing low-dimensional reductions of a given system. A number of key steps are involved in applying the method to experimental or computational data sets.

- Organize the data into a matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ where m is the number of measurement types (or probes) and n is the number of measurements (or trials) taken from each probe.
- Subtract off the mean for each measurement type or row $\mathbf{x}_j$.

- Compute the SVD and singular values of the covariance matrix to determine the principal components.

If considering a fitting algorithm, then the POD method is the appropriate technique to consider and the SVD can be applied directly to the $\mathbf{A} \in \mathbb{C}^{m\times n}$ matrix. This then produces the singular values as well as the POD modes of interest.

Although extremely powerful, and seemingly magical in its ability to produce insightful quantification of a given problem, these SVD methods are not without limits. Some fundamental assumptions have been made in producing the results thus far, the most important being the assumption of *linearity.* Recently some efforts to extend the method using nonlinearity prior to the SVD have been explored [39, 40, 41]. A second assumption is that larger variances (singular values) represent more important dynamics. Although generally this is believed to be true, there are certainly cases where this assumption can be quite misleading. Additionally, in constructing the covariance matrix, it is assumed that the mean and variance are sufficient statistics for capturing the underlying the dynamics. It should be noted that only exponential (Gaussian) probability distributions are completely characterized by the first two moments of the data. This will be considered further in the next section on independent component analysis. Finally, it is often the case that PCA may not produce the optimal results. For instance, consider a point on a propeller blade. Viewed from the PCA framework, two orthogonal components are produced to describe the circular motion. However, this is really a one-dimensional system that is solely determined by the phase variable. The PCA misses this fact unless use is made of the information already known, i.e. that a polar coordinate system with fixed radius is the appropriate context to frame the problem. Thus blindly applying the technique can also lead to non-optimal results that miss obvious facts.

15.6 Robust PCA

The singular value decomposition and its various guises, most notably principal component analysis, are perhaps the most widely used statistical tools for generating dimensionality reduction of data. As has been demonstrated and discussed, the SVD produces characteristic features (principal components) that are determined by the covariance matrix of the data. Fundamental in producing the SVD/PCA/POD modes is the L^2 norm for data fitting. Although the L^2 norm is highly appealing due to its centrality (and physical interpretation) in most applications, it does have potential flaws that can severely limit its applicability. Specifically, if corrupt data or large noise fluctuations are present in the data matrix, the higher-dimensional least-square fitting performed by the SVD algorithm squares this pernicious error; leading to significant deformation of the singular values and PCA modes describing the data. Although many physical systems and/or computations are relatively free of data corruption, many modern applications in image processing, PIV fluid measurements, web data analysis, and bioinformatics, for instance, are rife with arbitrary corruption of the measured data, thus ensuring that standard application of the SVD will be of limited use.

Ideally, in performing dimensionality reduction one should not allow the corrupt or large noise fluctuations to so grossly influence one's results. In order to limit the impact of such *data outliers*, a different measure, or norm, can be envisioned in measuring the best data fit (SVD/PCA/POD modes). One measure that has recently produced great success in this arena is the L^1 norm [42] which no longer squares the distance of the data to the best-fit mode. Indeed, the L^1 norm has been demonstrated to promote sparsity and is the underlying theoretical construct in *compressive sensing* or *sparse sensing* algorithms (See Sec. 18 for further details). Here, the L^1 norm is simply used as an alternative measure (norm) for performing the equivalent of the least-square fit to the data. As a result, a more robust method is achieved of computing PCA components, i.e. the so-called *robust PCA* method.

Matrix decomposition: low-rank plus sparse

To illustrate the entanglement of low-rank data with corrupt (sparse) measurement/matrix error, we can construct a new matrix which is composed of these two elements together

$$\mathbf{M} = \mathbf{L} + \mathbf{S} \tag{15.6.1}$$

where $\mathbf{M}$ is the data matrix of interest that is composed of low-rank structure $\mathbf{L}$ along with some sparse data $\mathbf{S}$. Our objective is the following: given observations $\mathbf{M}$, can the low-rank matrix $\mathbf{L}$ and sparse matrix $\mathbf{S}$ be recovered? Adding to the difficulty of this task is the following: we do not know the rank of the matrix $\mathbf{L}$, nor do we know how many non-zero (sparse) elements of the matrix $\mathbf{S}$ exist.

Remarkably, Candés and co-workers [42] have recently proved that this can be indeed solved through the formulation of a convex optimization problem. At first sight, the separation problem seems impossible to solve since the number of unknowns to infer for $\mathbf{L}$ and $\mathbf{S}$ is twice as many as the given measurements in $\mathbf{M}$. Furthermore, it seems even more daunting that we expect to reliably obtain the low-rank matrix $\mathbf{L}$ with errors in $\mathbf{S}$ of arbitrarily large magnitude. But not only can this problem be solved, it can be solved by tractable convex optimization [42].

Of course, in real applications, it is rare that the matrix decomposition is as ideal as (15.6.1). More generally, the decomposition is of the form

$$\mathbf{M} = \mathbf{L} + \mathbf{S} + \mathbf{N} \tag{15.6.2}$$

where the matrix $\mathbf{N}$ is a dense, small perturbation accounting for the fact that the low-rank component is only approximately low-rank and that small errors

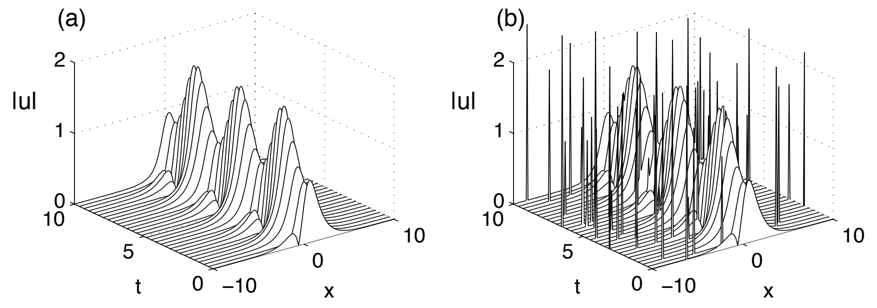


Figure 161: Representation of the space-time dynamics considered in Figs. 159 and 160. In (a), the ideal dynamics is demonstrated while in (b), a corrupted (noisy) image is assumed to be generated by the data collection process, for instance. The addition of noise brings into question the ability of the SVD to produce meaningful results. For this example, the low-rank matrix **L** has two modes of significance while the sparse matrix **S** has 60 non-zero entries.

can be added to all the entries (in some sense, this model unifies the classical PCA and the robust PCA by combining both sparse gross errors and dense small noise). But even in this case, the convex optimization algorithm formulated by Candés and co-workers [42] seems to do a remarkable job as separating low-rank from sparse data. This can be used to great advantage when certain types of data are considered, namely those with corrupt data or large, sparse noisy perturbations.

The details of the method and its proof are beyond the scope of this book. However, we will utilize the algorithms developed in the robust PCA literature in order to separate data matrices into low-rank and sparse components. A number of MATLAB algorithms can be downloaded from the web for this purpose (http://perception.csl.illinois.edu/matrix-rank/sample_code.html). Indeed, a wide variety of techniques have been developed for specifically providing a framework for robust PCA, including the Augmented Lagrange Multiplier (ALM) method [43], Accelerated Proximal Gradient method [44], Dual method [44], Singular Value Thresholding method [45] and the Alternating Direction method [46]. For our purposes, we will use the MATLAB program **inexact_alm_rpca** since it is extremely simple to use and exceedingly fast [43].

Example: As an example, consider the function from the previous section

$$f(x,t) = [1-0.5\cos 2t]\text{sech}x + [1-0.5\sin 2t]\text{sech}x\tanh x\,. \tag{15.6.3}$$

Previously, the SVD was applied to this function and a two-mode dominance was clearly demonstrated (See Figs. 159 and 160). To illustrate the problem to be considered, the above function is plotted along with a corrupted version in

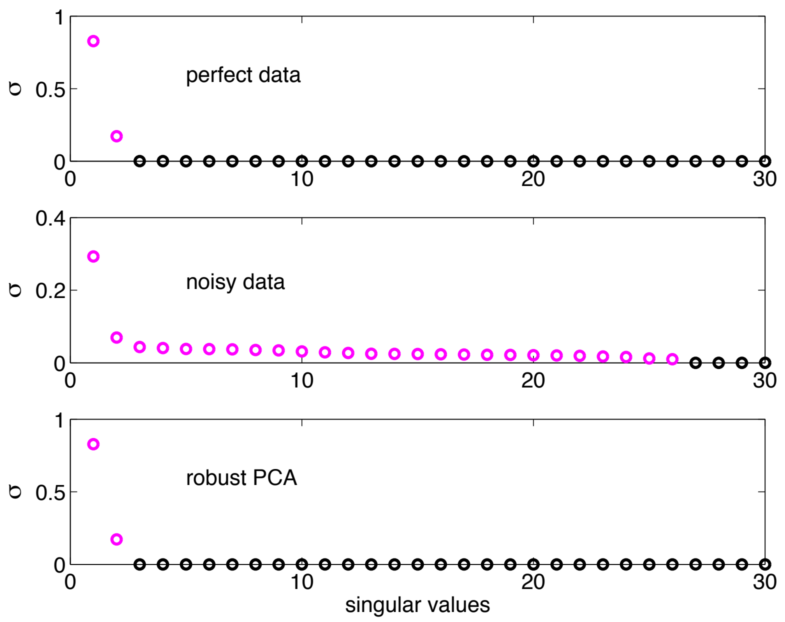


Figure 162: Singular values of the data matrices for the ideal data (top panel), the corrupted data (middle panel), and the low-rank data computed through robust PCA (bottom panel). The robust PCA construction produces almost a perfect match to the original, ideal data. The magenta dots represent the number of modes necessary to construct 99% of the original data. In this case, the ideal and robust PCA require two modes while the corrupt data requires 26 modes.

Fig. 161. In this example, the data was corrupted by the addition of sparse, but large, noise fluctuations to the ideal data. The following MATLAB code produces the ideal figure.

```
n=200;
x=linspace(-10,10,n);
t=linspace(0,10,30);
[X,T]=meshgrid(x,t);
usol=sech(X).*(1-0.5*cos(2*T))+(sech(X).*tanh(X)).*(1-0.5*sin(2*T));
subplot(2,2,1), waterfall(x,t,abs(usol)); colormap([0 0 0]);
```

To add sparse noise, the **randintrlv** is used to produce noise spikes (60 spikes in total) in certain matrix/pixel locations, thus corrupting the data matrix.

```
sam=60;
```

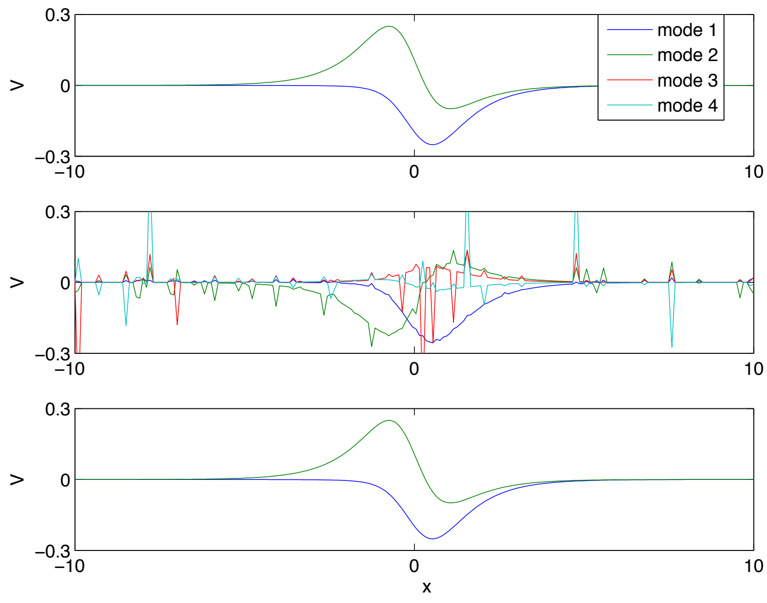
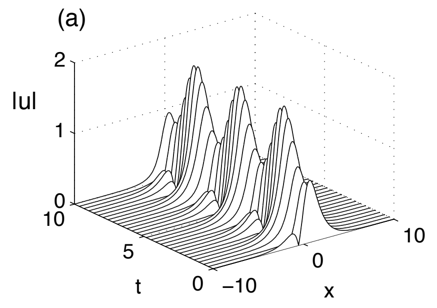


Figure 163: Dominant modes for the ideal data (top panel), the corrupted data (middle panel), and the low-rank data computed through robust PCA (bottom panel). The ideal and robust PCA produce the same dominant two modes while the corrupt data produces highly erratic modes.

```
Atest2=zeros(length(t),n);
Arand1=rand(length(t),n);
Arand2=rand(length(t),n);
r1 = randintrlv([1:length(t)*n],793);
r1k= r1(1:sam);
for j=1:sam
   Atest2(r1k(j))=-1;
end
Anoise=Atest2.*(Arand1+i*Arand2);
unoise=usol+2*Anoise;
subplot(2,2,2), waterfall(x,t,abs(unoise)); colormap([0 0 0]);
```

Figure 161 shows the ideal data along with the corrupt, or more physically relevant, data that might be collected in practice. The complex white-noise noise fluctuations are, of course, the problematic part of the dimensionality reduction issue. Recall that without such noise fluctuations, the ideal data can be faithfully approximated with a low-rank, two-mode approximation.

The PCA reduction of the data matrices can now be compared by using the singular value decomposition. The following code produces the SVD decompo-



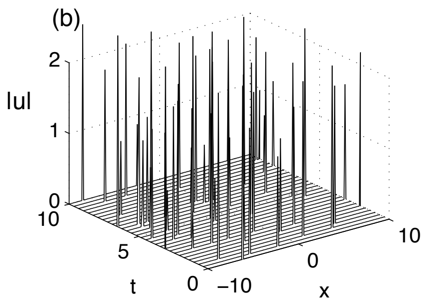


Figure 164: Separation of the original corrupt data matrix shown in Fig. 161(b) into a low-rank component (a) and a sparse component (b). The separation is almost perfect, with the low-rank matrix being dominated by two modes, i.e. they contain greater than 99% of the energy.

sition of both data matrices

```
A1=usol; A2=unoise;
[U1,S1,V1]=svd(A1);
[U2,S2,V2]=svd(A2);
```

Both the singular values and the modal structures are represented in Figs. 162 and 163 respectively. Specifically, the top panel of Fig. 162 shows the two-mode dominance (top panel of Fig. 163) of the ideal data while the middle panel of these figures shows the singular value distribution and the first four modes respectively. Note that the addition of the corrupt data activates many more modes. In fact, it takes 26 modes to capture 99% of the energy of the data compared to 2 modes previously. The modes are also highly perturbed from their ideal states due to the sparse (corrupt) noise that was added.

To apply the robust PCA algorithm, the corrupt data matrix is decomposed into real and imaginary parts before applying the **inexact_alm_rpca** algorithm. This algorithm effectively separates the data into low-rank and sparse components as given by (15.6.1). The low-rank portion is kept in order to then acquire the PCA modes necessary for reconstruction.

```
ur=real(unoise);
ui=imag(unoise);

lambda=0.2;
[R1r,R2r]=inexact_alm_rpca(real(ur.'),lambda);
[R1i,R2i]=inexact_alm_rpca(real(ui.'),lambda);

R1=R1r+i*R1i;
```

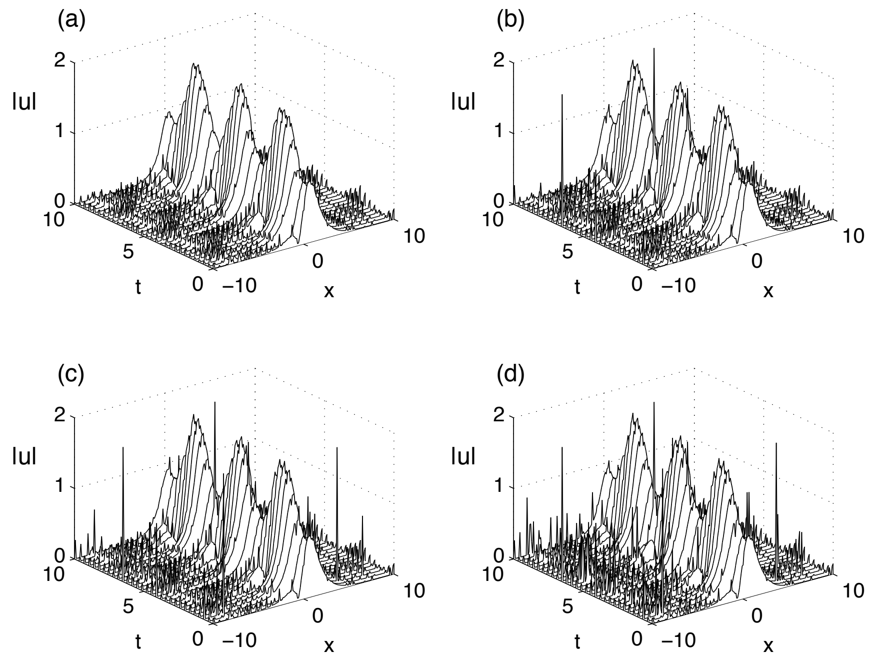


Figure 165: Two-, three-, four- and five-mode reconstruction of the matrix using the corrupted (sparse plus low-rank) data. Note that the addition of the higher modes adds sparse data spikes as is expected since they have not been filtered out of the data matrix.

```
R2=R2r+i*R2i;

[U3,S3,V3]=svd(R1.');

subplot(2,2,1), waterfall(x,t,abs(R1)'), colormap([0 0 0])
subplot(2,2,2), waterfall(x,t,abs(R2)'), colormap([0 0 0])
```

Note here that the real and imaginary portions are put back together at the end of the algorithm in order to SVD the low-rank portion. Further, the two matrices corresponding to the low-rank ($R1$) and sparse ($R2$) portions of the data matrix are plotted. The parameter *lambda* used in the **inexact_alm_rpca** algorithm can be tuned to best separate the sparse from low-rank matrices in (15.6.1). Here a value of 0.2 is used, which produces an almost ideal separation as is shown in Figs. 162(bottom panel), 163(bottom panel) and 164. Indeed, an almost perfect separation is achieved as advertised (guaranteed) by Candés and co-workers [42].

To further investigate the robust PCA algorithm and its ability to extract

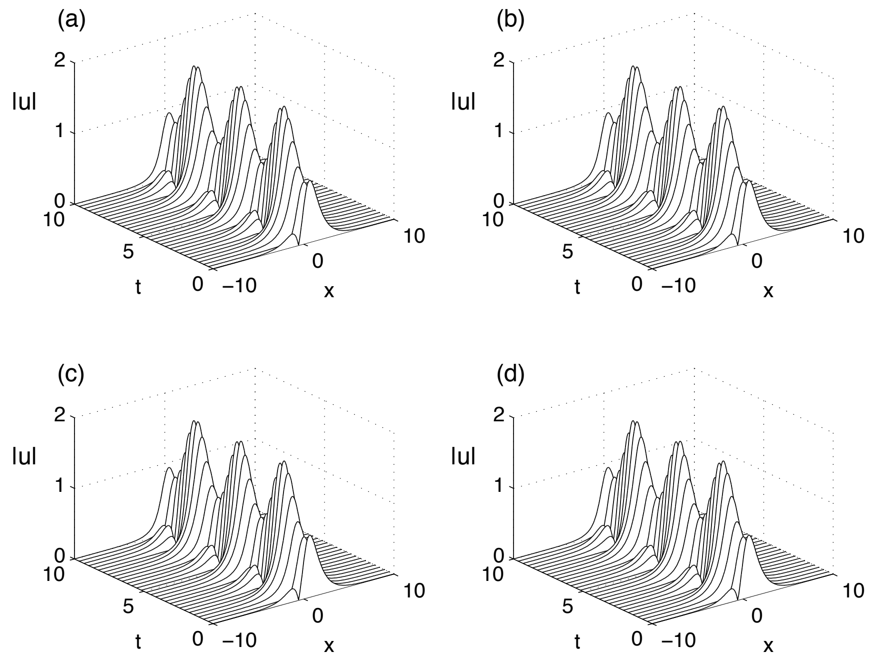


Figure 166: Two-, three-, four- and five-mode reconstruction of the matrix using the robust PCA algorithm for extracting the corrupted (sparse) components. Note that a two-mode expansion is sufficient to capture all the features of interest.

the meaningful, low-rank matrix from the full matrix corrupted by sparse fluctuations, a series of low-rank approximations are made of the data matrices using the PCA modes generated by the SVD (See Figs. 165 and 166). The low-rank approximations show how effective the robust PCA algorithm is in separating out the low-rank from sparse components.

```
% low-rank approximation of corrupt data matrix
figure(1)
for jj=1:4
  for j=1:jj+1
    ff=U2(:,1:j)*S2(1:j,1:j)*V2(:,1:j).';
    subplot(2,2,jj), waterfall(x,t,abs(ff)), colormap([0 0 0])
  end
end

% approximation of low-rank matrix from robust PCA
figure(2)
```

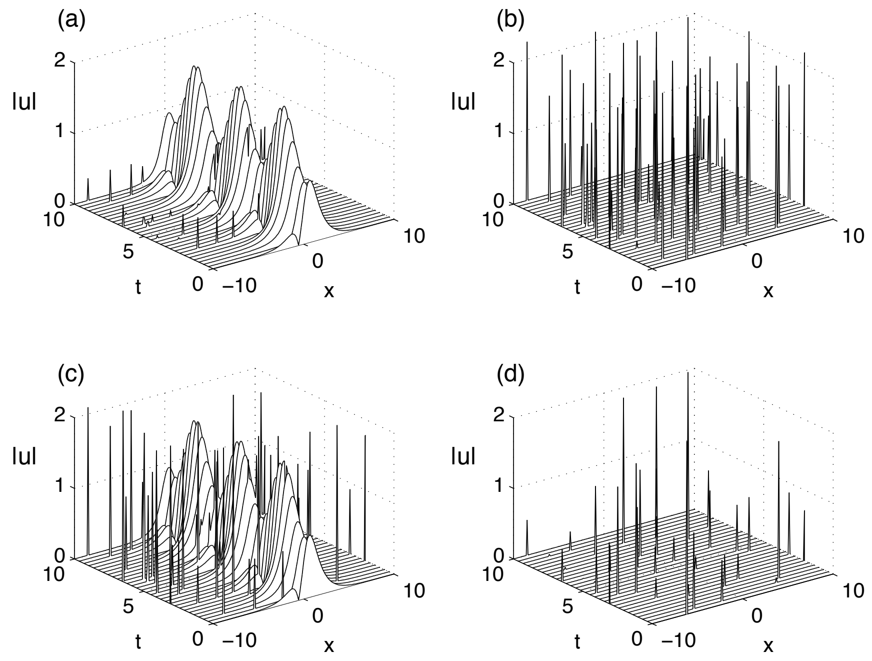


Figure 167: Separation of the original corrupt data matrix shown in Fig. 161(b) into a low-rank component (a) and (c) and a sparse component (b) and (d). The top panels are for *lambda* = 0.5 in the **inexact_alm_rpca** algorithm while the bottom panels are for *lambda* = 0.8. The separation deteriorates from the almost perfect separation illustrated previously with *lambda* = 0.2.

```
for jj=1:4
  for j=1:jj+1
    ff=U3(:,1:j)*S3(1:j,1:j)*V3(:,1:j).';
    subplot(2,2,jj), waterfall(x,t,abs(ff)), colormap([0 0 0])
  end
end
```

Finally, to end this discussion, the effects of the parameter *lambda* in the **inexact_alm_rpca** algorithm is illustrated. This parameter takes on values of zero to unity and can be tuned to achieve better or worse performance. The default, if *lambda* is not included, does a reasonable job generically in separating the low-rank from the sparse matrices. As *lambda* goes to zero, the computed low-rank matrix picks up the entire initial data matrix, i.e. the resulting sparse matrix is computed to be zero while the resulting sparse matrix contains the original data matrix. As *lambda* goes to unity, the opposite happens with the computed low-rank matrix containing nothing while the sparse matrix contains virtually the entire original matrix. Figure 167 depicts the results of the **inexact_alm_rpca** algorithm for *lambda* equal to 0.5 and 0.8. Note that as *lambda* is increased, the fidelity of the low-rank matrix is compromised and corrupted by the sparsity.

As a final comment, the robust PCA algorithm advocated for here is quite remarkable in its ability to separate sparse corruption from low-rank structure in a given data matrix. The application of robust PCA essentially acts as an advanced filter for cleaning up a data matrix. Indeed, one can potentially use this as a filter which is very unlike the time-frequency filtering schemes considered previously.

16 Independent Component Analysis

The concept of principal components or of a proper orthogonal decomposition are fundamental to data analysis. Essentially, there is an assertion, or perhaps a hope, that underlying seemingly complex and unordered data, there is in fact, some characteristic, and perhaps low-dimensional ordering of the data. The singular value decomposition of a matrix, although a mathematical idea, was actually a formative tool for the interpretation and ordering of the data. In this section on *independent component analysis* (ICA), the ideas turn to data sets for which there are more than one statistically independent objects in the data. ICA provides a foundational mathematical method for extracting interleaved data sets in a fairly straightforward way. Its applications are wide and varied, but our primary focus will be application of ICA in the context of image analysis.

16.1 The concept of independent components

Independent component analysis (ICA) is a powerful tool that extends the concepts of PCA, POD and SVD. Moreover, you are already intuitively familiar with the concept as some of the examples here will show. A simple way to motivate thinking about ICA is by considering the *cocktail party problem.* Thus consider two conversations in a room that are happening simultaneously. How is it that the two different acoustic signals of conversation one and two can be separated out? Figure 168 depicts a scenario where two groups are conversing. Two microphones are placed in the room at different spatial locations and from the two signals $s_1(t)$ and $s_2(t)$ a mathematical attempt is made to separate the signals that have been mixed at each of the microphone locations. Provided that the noise level is not too large or that the conversation volumes are sufficiently large, humans can perform this task with remarkable ease. In our case, the two microphones are our two ears. Indeed, this scenario and its mathematical foundations are foundational to the concept of eavesdropping on a conversation.

From a mathematical standpoint, this problem can be formulated with the

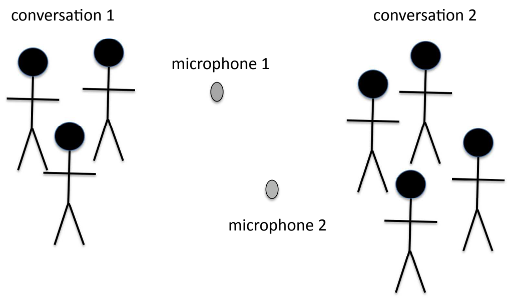


Figure 168: Envisioned cocktail party problem. Two groups of people are having separate conversations which are recorded from microphone one and two. The signals from the two conversations are mixed together at microphone one and two according to the placement relative to the conversations. From the two microphone recordings, the individual conversations can then be extracted.

following mixing equation

$$x_1(t) = a_{11}s_1 + a_{12}s_2 \tag{16.1.4a}$$
$$x_2(t) = a_{21}s_1 + a_{22}s_2 \tag{16.1.4b}$$

where $x_1(t)$ and $x_2(t)$ are the mixed, recorded signals at microphone one and two respectively. The coefficients a_{ij} are the mixing parameters that are determined by a variety of factors including the placement of the microphones in the room, the distance to the conversations, and the overall room acoustics. Note that we are omitting time-delay signals that may reflect off the walls of the room. This problem also resembles quite closely what may happen in a large number of applications. For instance, consider the following

- **Radar detection:** If there are numerous targets that are being tracked, then there is significant mixing in the scattered signal from all of the targets. Without a method for clear separation of the targets, the detector becomes useless for identifying location.
- **Electroencephalogram (EEG):** EEG readings are electrical recordings of brain activities. Typically these EEG readings are from multiple locations of the scalp. However, at each EEG reading location, all the brain activity signals are mixed, thus preventing a clear understanding of how many underlying signals are contained within. With a large number of EEG probes, ICA allows for the separation of the brain activity readings and a better assessment of the overall neural activity.
- **Terminator Salvation:** If you remember in the movie, John Connor and the resistance found a hidden signal (ICA) embedded on the normal

> signals in the communications sent between the terminators and skynet vehicles and ships. Although not mentioned in the movie, this was clearly somebody in the future who is reading this book now. Somehow that bit of sweet math only made it to the cutting room floor. Regardless, one shouldn't under-estimate how awesome data analysis skills are in the real world of the future.

The ICA method is closely related to the *blind source separation* (BSS) (or blind signal separation) method. Here the sources refer to the original signals, or independent components, and blind refers to the fact that the mixing matrix coefficients a_{ij} are unknown.

A more formal mathematical framework for ICA can be established by considering the following: *Given N distinct linear combinations of N signals (or data sets), determine the original N images.* This is the succinct statement of the mathematical objective of ICA. Thus to generalize (16.1.4) to the N dimensional system we have

$$x_j(t) = a_{j1}s_1 + a_{j2}s_2 + \cdots + a_{jN}s_N \quad 1 \leq j \leq N\,. \tag{16.1.5}$$

Expressed in matrix form, this results in

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{16.1.6}$$

where $\mathbf{x}$ are the mixed signal measurements, $\mathbf{s}$ are the original signals we are trying to determine, and $\mathbf{A}$ is the matrix of coefficients which determines how the signals are mixed as a function of the physical system of interest. At first, one might simply say that the solution to this is trivial and it is given by

$$\mathbf{s} = \mathbf{A}^{-1}\mathbf{x} \tag{16.1.7}$$

where we are assuming that the placement of our measurement devices are such that $\mathbf{A}$ is nonsingular and its inverse exists. Such a solution makes the following assumption: we know the matrix cofficients a_{ij}. The point is, we don't know them. Nor do we know the signal $\mathbf{s}$. Mathematically then, the aim of ICA is to approximate the coefficients of the matrix $\mathbf{A}$, and in turn the original signals $\mathbf{s}$, under as general assumptions as possible.

The statistical model given by Eq. (16.1.6) is called *independent component analysis*. The ICA is a *generative model*, which means that is describes how the observed data are generated by a process of mixing in the components $s_j(t)$. The independent components are *latent variables*, meaning they are never directly observed. The key step in the ICA models is to assume the following: *the underlying signals $s_j(t)$ are* statistically independent *with probability distributions that are* not *Gaussian.* Statistical independence and higher-order moments of the probably distribution (thus requiring non-Gaussian distributions) are the critical mathematical tools that will allow us to recover the signals.

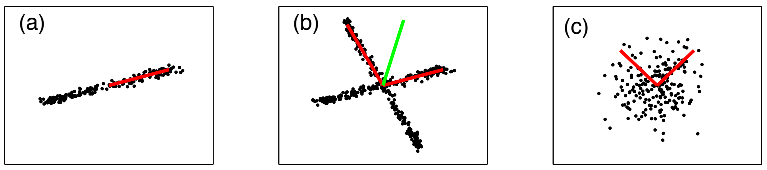


Figure 169: Illustration of the principals of PCA (a), ICA (b) and the failure of Gaussian distributions to distinguish principal components (c). The red vectors show the principal directions, while the green vector in the middle panel shows what would be the principal direction if a direct SVD where applied rather than an ICA. The principal directions in (c) are arbitrarily chosen since no principal directions can be distinguished.

Image separation and SVD

To make explicit the mathematical methodology to be pursued here, a specific example of image separation will be used. Although there are a variety of mathematical alternatives for separating the independent components [47], the approach considered here will be based upon PCA and SVD. To illustrate the concept of ICA, consider the example data represented in Fig. 169. The three panels are fundamental to understanding the concept of ICA. In the left panel (a), measurements are taken of a given system and are shown to project nicely onto a dominant direction whose leading principal component is denoted by the red vector. This red vector would be the principal component with a length σ_1 determined by the largest singular value. It is clear that the singular value σ_2 corresponding to the orthogonal direction of the second principal component would be considerably smaller. The middle panel (b), the measurements indicate that there are two principal directions in the data fluctuations. If an SVD is applied directly to the data, then the dominant singular direction would be approximately the green vector. Yet it is clear that the green vector does not accurately represent the data. Rather, the data seems to be organized *indepedently* along two different directions. An SVD of the two independent data sets would produce two principal components, i.e. two *independent component analysis* vectors. Thus it is critical to establish a method for separating out data of this sort into their independent directions. Finally, in the third panel (c), Gaussian distributed data is shown where no principal components can be identified with certainty. Indeed, there are an infinite number of possible orthogonal projections that can be made. Two arbitrary directions have been shown in panel (c). This graphic shows why Gaussian distributed random variables are disastrous for ICA, no projections onto the independent components can be accomplished.

We now turn our attention to the issue of image separation. Figure 170 demonstrates a prototypical example of what can occur when photographing

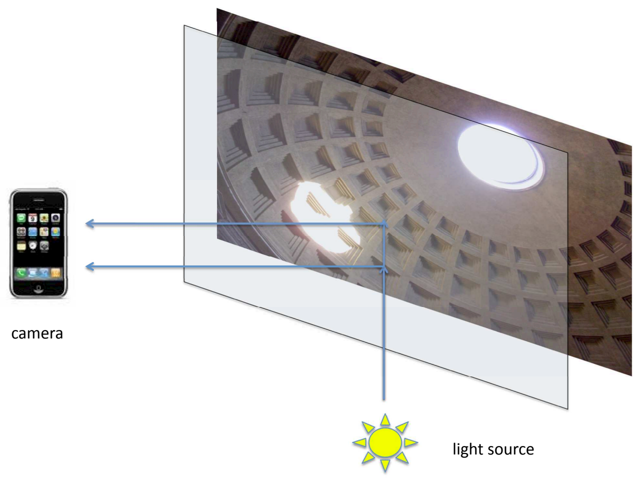


Figure 170: Illustration of image separation problem. A piece of glass reflects part of the background light source while some of the light penetrates to the image and is reflected to the camera. Thus the camera sees two images: the image (painting) and the reflection of the background light source.

an image behind glass. The glass encasing of the art work, or image, produces a reflection of the backlit source along with the image itself. Amazingly, the human eye can very easily compensate for this and *see through* much of the reflection to the original image. Alternatively, if we choose, we can also focus on the reflected image of the backlit source. Ultimately, there is some interest in training a computer to do this image processing automatically. Indeed, in artificial vision, algorithms such as these are important for mimicking natural human behaviors and abilities.

The light reflected off the glass is partially polarized and provides the crucial degree of freedom necessary for our purposes. Specifically, to separate the images, a photo is taken as illustrated in Fig. 170 but now with a linear polarizer placed in front of the camera lens. Two photos are taken where the linear polarizer is rotated 90 degrees from each other. This is a simple home experiment essentially. But you must have some control over the focus of your camera in terms of focusing and flash. The picture to be considered is hanging in my office: **The Judgement of Paris** by John Flaxman (1755-1826). Flaxman was an English sculptor, draughtsman and illustrator who enjoyed a long and brilliant career that included a stint at Wedgwood in England. In fact, at the age of 19, Josiah Wedgwood hired Flaxman as a modeler of classic and domestic friezes, plagues and ornamental vessels and medallion portraits. You can still find many Wedgwood ornamental vessels today that have signature Flaxman

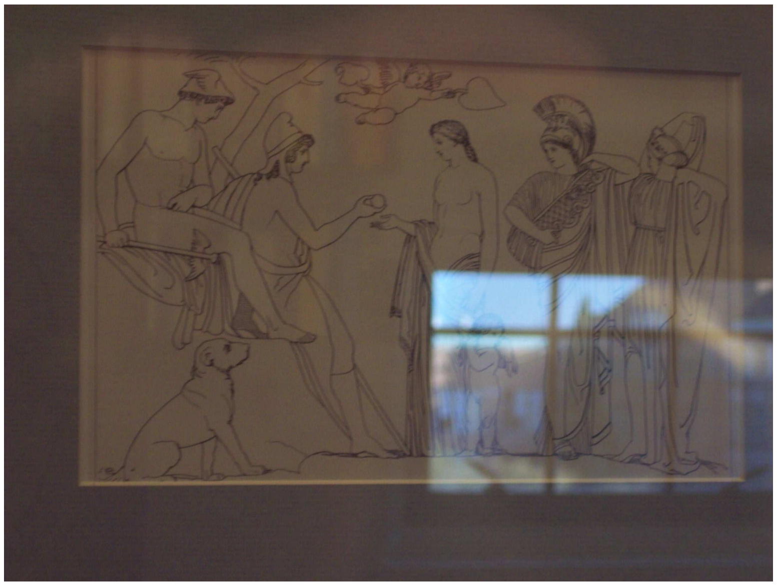


Figure 171: **The Judgement of Paris** by John Flaxman (1755-1826). This is a plate made by Flaxman for an illustrated version of Homer's Iliad. The picture was taken in my office and so you can see both a faint reflection of my books as well as strong signature of my window which overlooks Drumheller Fountain.

works molded on their sides. It was during this period that the manufactures of that time period perfected the style and earned great reputations. But what gained Flaxman his general fame was not his work in sculpture proper, but his designs and engravings for the **Iliad**, **Odyssey**, and **Dante's Inferno**. All of his works on these mythological subjects where engraved by Piroli with considerable loss to Flaxman's own lines. The greatest collection of Flaxman's work is housed in the Flaxman Gallery at the University College Longon.

In the specific Flaxman example considered here, one of the most famous scenes of mythology is depicted: **The Judgement of Paris**. In this scene, Hermes watches as young Paris plays the part of judge. His task is to contemplate giving the golden apple with the inscription *To the Fairest* to either Venus, Athena, or Hera. This golden apple was placed at the wedding reception of Peleus and Thetis, father and mother of peerless Achilles, by Discord since she did not receive a wedding invitation. Each lovely goddess in turn promises him something for the apple: Athena promises that he will become the greatest warrior in history, Juno promises him to be greatest ruler in the land, ruling over a vast and unmatched empire. Last comes the lovely Venus who promises him that the most beautiful woman in the world will be his. Of course, to sweeten up

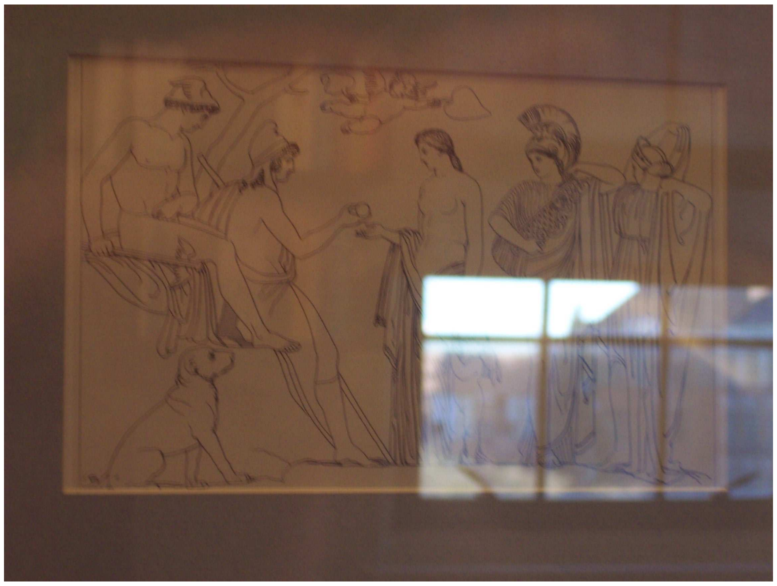


Figure 172: This is an identical picture to Fig. 171 with a linear polarizer rotated 90 degrees. The polarization state of the reflected field is effected by the linear polarizer, thus allowing us to separate the images.

her offer, she is wearing next to nothing as usual. Lo and behold, the hot chick wins! And as a result we have the Trojan War for which Helen was fought for. I could go on here, but what else is to be expected of a young 20-something year old male (an older guy would have taken the vast kingdom while a prepubescent boy would have chosen to be the great warrior). It is comforting to know that very little has changed since in some 4000 years. Figures 171 and 172 represent this Flaxman scene along with the reflected image of my office window. The two pictures are taken with a linear polarizer in front of a camera that has been rotated by 90 degrees between the photos.

SVD method for ICA

Given the physical problem, how is SVD used then to separate the two images observed in Figs. 171 and 172? Consider the action of the SVD on the image mixing matrix **A** of Eq. (16.1.6). In this case, there are two images $\mathbf{S}_1$ and $\mathbf{S}_2$ that we are working with. This gives us 6 unknowns $(\mathbf{S}_1, \mathbf{S}_2, a_{11}, a_{12}, a_{21}, a_{22})$ with only the two constraints from Eq. (16.1.6). Thus the system cannot be solved without further assumptions being made. The first assumption will be that the two images are statistically independent. If the pixel intensities are denoted by $\mathbf{S}_1$ and $\mathbf{S}_2$, then statistical independence is mathematically expressed

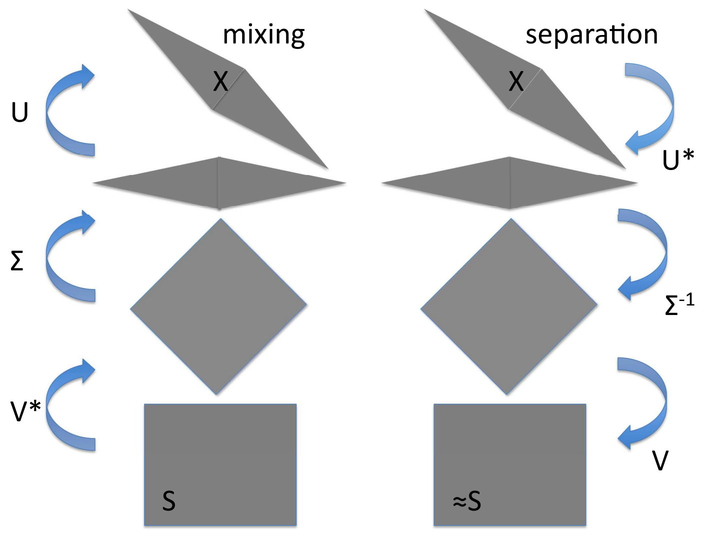


Figure 173: Graphical depiction of the SVD process of mixing the two images. The reconstruction of the images is accomplished by approximating the inversions of the SVD matrices so as to achieve a separable (statistically independent) probability distribution of the two images.

as

$$P(\mathbf{S}_1, \mathbf{S}_2) = P(\mathbf{S}_1)P(\mathbf{S}_2) \tag{16.1.8}$$

which states that the joint probability density is separable. No specific form is placed upon the probability densities themselves aside from this: they cannot be Gaussian distributed. In practice, this is a reasonable constraint since natural images are rarely Gaussian. A second assumption is that the matrix $\mathbf{A}$ is full rank, thus assuming that the two measurements of the image indeed have been filtered with different polarization states. These assumptions yield an estimation problem which is the ICA.

Consider what happens under the SVD process to the matrix $\mathbf{A}$

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{16.1.9}$$

where again $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices that simply lead to rotation and $\mathbf{\Sigma}$ stretches (scales) an image as prescribed by the singular values. A graphical illustration of this process is shown in Fig. 173 for distributions that are uniform, i.e. they form a square. The mixing matrix $\mathbf{A}$ can be thought to first rotate the square via unitary matrix $\mathbf{V}^*$, then stretch it into a parallelogram via the diagonal matrix $\mathbf{\Sigma}$ and then rotate the parallelogram via the unitary matrix $\mathbf{U}$.

This is now the mixed image **X**. *The estimation, or ICA, of the independent images thus reduces to finding how to transform the rotated parallelogram back into a square. Or mathematically, transforming the mixed image back into a separable product of one-dimensional probability distributions.* This is the defining mathematical goal of the ICA image analysis problem, or any general ICA reduction technique.

16.2 Image separation problem

The method proposed here to separate two images relies on reversing the action of the SVD on the two statistically independent images. Again, the matrix **A** in (16.1.9) is not known, so a direct computation of the SVD cannot be performed. However, each of the individual matrices can be approximated by considering its net effect on the assumed uniformly distributed images.

Three specific computations must be made: (i) the rotation of the parallelogram must be approximated, (ii) the scaling of the parallelogram according to the variances must be computed, and (iii) the final rotation back to a separable probability distribution must be obtained. Each step is outlined below.

Step 1: Rotation of the parallelogram

To begin, consider once again Fig. 173. Our first objective is to undo the rotation of the unitary matrix **U**. Thus we will ultimately want to compute the inverse of this matrix which is simply $\mathbf{U}^*$. In a geometrical way of thinking, our objective is to align the long and short axes of the parallelogram with the primary axis as depicted in the two top right shaded boxes of Fig. 173. The angle of the parallelogram relative to the primary axes will be denoted by θ, and the long and short axes correspond to the axes of the maximal and minimal variance respectively. From the image data itself, then, the maximal and minimal variance directions will be extracted. Assuming mean-zero measurements, the variance at an arbitrary angle of orientation is given by

$$Var(\theta) = \sum_{j=1}^{N} \left\{ [x_1(j) \quad x_2(j)] \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \right\}^2 . \qquad (16.2.10)$$

The maximal variance is determined by computing the angle θ that maximizes this function. It will be assumed that the corresponding angle of minimal variance will be orthogonal to this at $\theta - \pi/2$. These axes are essentially the principal component directions that would be computed if we actually knew components of the matrix **A**.

The maximum of (16.2.10) with respect to θ can be found by differentiating $Var(\theta)$ and setting it equal to zero. To do this, Eq. (16.2.10) is rewritten as

$$Var(\theta) \;=\; \sum_{j=1}^{N} \left\{ [x_1(j) \quad x_2(j)] \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \right\}^2$$

$$= \sum_{j=1}^{N} [x_1(j)\cos\theta + x_2(j)\sin\theta]^2 \tag{16.2.11}$$
$$= \sum_{j=1}^{N} x_1^2(j)\cos^2\theta + 2x_1(j)x_2(j)\cos\theta\sin\theta + x_2^2(j)\sin^2\theta.$$

Differentiating with respect to θ then gives

$$\begin{aligned}\frac{d}{d\theta}Var(\theta) &= 2\sum_{j=1}^{N} -x_1^2(j)\sin\theta\cos\theta + x_1(j)x_2(j)\left[\cos^2\theta - \sin^2\theta\right] + x_2^2(j)\sin\theta\cos\theta \\ &= 2\sum_{j=1}^{N}\left[x_2^2(j) - x_1^2(j)\right]\sin\theta\cos\theta + x_1(j)x_2(j)\left[\cos^2\theta - \sin^2\theta\right] \\ &= \sum_{j=1}^{N}\left[x_2^2(j) - x_1^2(j)\right]\sin 2\theta + 2x_1(j)x_2(j)\cos 2\theta \end{aligned} \tag{16.2.12}$$

which upon setting this equal to zero gives the value of θ desired. Thus taking $d(Var(\theta))/d\theta = 0$ gives

$$\frac{\sin 2\theta}{\cos 2\theta} = \frac{-2\sum_{j=1}^{N} x_1(j)x_2(j)}{\sum_{j=1}^{N} x_2^2(j) - x_1^2(j)}\,, \tag{16.2.13}$$

or in terms of θ alone

$$\theta_0 = \frac{1}{2}\tan^{-1}\left[\frac{-2\sum_{j=1}^{N} x_1(j)x_2(j)}{\sum_{j=1}^{N} x_2^2(j) - x_1^2(j)}\right]. \tag{16.2.14}$$

In the polar coordinates $x_1(j) = r(j)\cos\psi(j)$ and $x_2(j) = r(j)\sin\psi(j)$, the expression reduces further to

$$\theta_0 = \frac{1}{2}\tan^{-1}\left[\frac{\sum_{j=1}^{N} r^2(j)\sin 2\psi(j)}{\sum_{j=1}^{N} r^2(j)\cos 2\psi(j)}\right]. \tag{16.2.15}$$

The rotation matrix, or unitary transformation, associated with the rotation of the parallelogram back to its aligned position is then

$$\mathbf{U}^* = \begin{bmatrix} \cos\theta_0 & \sin\theta_0 \\ -\sin\theta_0 & \cos\theta_0 \end{bmatrix} \tag{16.2.16}$$

with the angle θ_0 computed directly from the experimental data.

Step 2: Scaling of the parallelogram

The second task is to undo the principal component scaling achieved by the singular values of the SVD decomposition. This process is illustrated as the second step in the right column of Fig. 173. This task, however, is rendered straightforward now that the principal axes have been determined from step 1. In particular, the assumption was that along the direction θ_0, the maximal variance is achieved, while along $\theta_0 - \pi/2$, the minimal variance is achieved. Thus the components, or singular values, of the diagonal matrix $\mathbf{\Sigma}^{-1}$ can be computed.

The variance along the two principal component axes are given by

$$\sigma_1 = \sum_{j=1}^{N} \left\{ [x_1(j) \quad x_2(j)] \left[\begin{array}{c} \cos\theta_0 \\ \sin\theta_0 \end{array} \right] \right\}^2 \tag{16.2.17a}$$

$$\sigma_2 = \sum_{j=1}^{N} \left\{ [x_1(j) \quad x_2(j)] \left[\begin{array}{c} \cos(\theta_0 - \pi/2) \\ \sin(\theta_0 - \pi/2) \end{array} \right] \right\}^2 . \tag{16.2.17b}$$

This then gives the diagonal elements of the matrix $\mathbf{\Sigma}$. To undo this scaling, the inverse of $\mathbf{\Sigma}$ is constructed so that

$$\mathbf{\Sigma}^{-1} = \left[\begin{array}{cc} 1/\sqrt{\sigma_1} & 0 \\ 0 & 1/\sqrt{\sigma_2} \end{array} \right] . \tag{16.2.18}$$

This matrix, in combination with that of step 1, easily undoes the principal component direction rotation and its associated scaling. However, this process has only decorrelated the images, and a separable probability distribution has not yet been produced.

Step 3: Rotation to produce a separable probability distribution

The final rotation to separate the probability distributions is a more subtle and refined issue, but critical to producing nearly separable probability distributions. This separation process typically relies on the higher moments of the probability distribution. Since the mean has been assumed to be zero and there is no reason to believe that there is an asymmetry in the probability distributions, i.e. higher-order odd moments (such as the skewness) are negligible, the next dominant statistical moment to consider is the fourth moment, or the kurtosis of the probability distribution. The goal will be to minimize this fourth-order moment, and by doing so we will determine the appropriate rotation angle. Note that the second moment has already been handled through steps and 1 and 2. Said in a different mathematical way, minimizing the kurtosis will be another step in trying to approximate the probability distribution of the images as separable functions so that $P(\mathbf{S}_1\mathbf{S}_2) \approx P(\mathbf{S}_1)P(\mathbf{S}_2)$.

The kurtosis of a probability distribution is given by

$$kurt(\phi) = K(\phi) = \sum_{j=1}^{N} \left\{ [\bar{x}_1(j) \;\; \bar{x}_2(j)] \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right\}^4 \tag{16.2.19}$$

where ϕ is the angle of rotation associated with the unitary matrix $\mathbf{U}$ and the variables $\bar{x}_1(j)$ and $\bar{x}_2(j)$ is the image that has undergone the two steps of transformation as outlined previously.

For analytic convenience, a normalized version of the above definition of kurtosis will be considered. Specifically, a normalized version is computed in practice. The expedience of the normalization will become clear through the algebraic manipulations. Thus consider

$$\bar{K}(\phi) = \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left\{ [\bar{x}_1(j) \;\; \bar{x}_2(j)] \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right\}^4 . \tag{16.2.20}$$

As in our calculation regarding the second moment, the kurtosis will be written in a more natural form for differentiation

$$\begin{aligned}
\bar{K}(\phi) &= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left\{ [\bar{x}_1(j) \;\; \bar{x}_2(j)] \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right\}^4 \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left[\bar{x}_1(j)\cos\phi + \bar{x}_2(j)\sin\phi \right]^4 \qquad (16.2.21) \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left[\bar{x}_1^4(j)\cos^4\phi + 4\bar{x}_1^3(j)\bar{x}_2(j)\cos^3\phi\sin\phi \right. \\
&\quad \left. + 6\bar{x}_1^2(j)\bar{x}_2^2(j)\cos^2\phi\sin^2\phi + 4\bar{x}_1(j)\bar{x}_2^3(j)\cos\phi\sin^3\phi + \bar{x}_2^4(j)\sin^4\phi \right] \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left[\frac{1}{8}\bar{x}_1^4(j)(3 + 4\cos 2\phi + \cos 4\phi) \right. \\
&\quad + \bar{x}_1^3(j)\bar{x}_2(j)(\sin 2\phi + (1/2)\sin 4\phi) + \frac{3}{4}\bar{x}_1^2(j)\bar{x}_2^2(j)(1 - \cos 4\phi) \\
&\quad \left. + \bar{x}_1(j)\bar{x}_2^3(j)(\sin 2\phi - (1/2)\sin 4\phi) + \frac{1}{8}\bar{x}_2^4(j)(3 - 4\cos 2\phi + \cos 4\phi) \right] .
\end{aligned}$$

This quantity needs to be minimized with an appropriate choice of ϕ. Thus the derivative $d\bar{K}/d\phi$ must be computed and set to zero.

$$\begin{aligned}
\frac{d\bar{K}(\phi)}{d\phi} &= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left[\frac{1}{8}\bar{x}_1^4(j)(-8\sin 2\phi - 4\sin 4\phi) \right. \\
&\quad + \bar{x}_1^3(j)\bar{x}_2(j)(2\cos 2\phi + 2\cos 4\phi) + 3\bar{x}_1^2(j)\bar{x}_2^2(j)\sin 4\phi
\end{aligned}$$

$$+\bar{x}_1(j)\bar{x}_2^3(j)(2\cos 2\phi - 2\cos 4\phi) + \frac{1}{8}\bar{x}_2^4(j)(8\sin 2\phi - 4\sin 4\phi)\Bigg]$$

$$= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left\{[\bar{x}_2^4(j) - \bar{x}_1^4(j)]\sin 2\phi\right.$$
$$+[2\bar{x}_1(j)\bar{x}_2(j)^3 + 2\bar{x}_1^3(j)\bar{x}_2(j)]\cos 2\phi + [2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi$$
$$\left.+[3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi\right\}$$
$$= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left\{[(\bar{x}_1^2(j)+\bar{x}_2^2(j))(\bar{x}_2^2(j) - \bar{x}_1^2(j))]\sin 2\phi\right.$$
$$+[2\bar{x}_1(j)\bar{x}_2(j)(\bar{x}_1^2(j)+\bar{x}_2^2(j))]\cos 2\phi + [2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi$$
$$\left.+[3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi\right\}$$
$$= \sum_{j=1}^{N}[\bar{x}_2^2(j) - \bar{x}_1^2(j)]\sin 2\phi + 2\bar{x}_1(j)\bar{x}_2(j)\cos 2\phi$$
$$+\frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)}\left\{[2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi\right.$$
$$\left.+[3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi\right\}$$
$$= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left\{[2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi\right.$$
$$\left.+[3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi\right\}, \qquad (16.2.22)$$

where the terms proportional to $\sin 2\phi$ and $\cos 2\phi$ add to zero since they are minimized when considering the second moment or variance, i.e. we would like to minimize both the variance and the kurtosis, and this calculation does both. By setting this to zero, the angle ϕ_0 can be determined to be

$$\phi_0 = \frac{1}{4}\tan^{-1}\left[\frac{-\sum_{j=1}^{N}[2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]/[\bar{x}_1^2(j)+\bar{x}_2^2(j)]}{\sum_{j=1}^{N}[3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]/[\bar{x}_1^2(j)+\bar{x}_2^2(j)]}\right] \qquad (16.2.23)$$

When converted to polar coordinates, this reduces very nicely to the following expression:

$$\phi_0 = \frac{1}{4}\tan^{-1}\left[\frac{\sum_{j=1}^{N} r^2 \sin 4\psi(j)}{\sum_{j=1}^{N} r^2 \cos 4\psi(j)}\right] \qquad (16.2.24)$$

The rotation back to the approximately statistically independent square is then given by

$$\mathbf{V} = \begin{bmatrix} \cos\phi_0 & \sin\phi_0 \\ -\sin\phi_0 & \cos\phi_0 \end{bmatrix} \qquad (16.2.25)$$

with the angle θ_0 computed directly from the experimental data. At this point, it is unknown wether the angle ϕ_0 is a minimum or maximum of the kurtosis and this should be checked. Sometimes this is a maximum, especially in cases when one of the image histograms has long tails relative to the other [48].

Completing the analysis

Recall that our purpose was to compute, through the statistics of the images (pixels) themselves, the SVD decomposition. The method relied on computing (approximating) the principal axis and scaling of the principal components. The final piece was to try and make the probability distribution as separable as possible by minimizing the kurtosis as a function of the final rotation angle.

To reconstruct the image, the following linear algebra operation is performed:

$$\begin{aligned}\mathbf{s} &= \mathbf{A}^{-1}\mathbf{x} = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^*\mathbf{x} \\ &= \begin{bmatrix} \cos\phi_0 & \sin\phi_0 \\ -\sin\phi_0 & \cos\phi_0 \end{bmatrix}\begin{bmatrix} 1/\sqrt{\sigma_1} & 0 \\ 0 & 1/\sqrt{\sigma_2} \end{bmatrix}\begin{bmatrix} \cos\theta_0 & \sin\theta_0 \\ -\sin\theta_0 & \cos\theta_0 \end{bmatrix}\mathbf{x}\end{aligned} \quad (16.2.26)$$

A few things should be noted about the inherent ambiguities in the recovery of the two images. The first is the ordering ambiguity. Namely, the two matrices are indistinguishable:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{21} & a_{22} \\ a_{11} & a_{12} \end{bmatrix}\begin{bmatrix} x_2 \\ x_1 \end{bmatrix} \quad (16.2.27)$$

Thus image one and two are arbitrary to some extent. In practice this does not matter since the aim was simply to separate, not label, the data measurements. There is also an ambiguity in the scaling since

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}/\alpha & a_{12}/\delta \\ a_{21}/\alpha & a_{22}/\delta \end{bmatrix}\begin{bmatrix} \alpha x_1 \\ \delta x_2 \end{bmatrix}. \quad (16.2.28)$$

Again this does not matter since the two separated images can then be rescaled to their full intensity range afterwards. Regardless, this shows the basic process that needs to be applied to the system data in order to construct a self-consistent algorithm capable of image separation.

16.3 Image separation and MATLAB

Using MATLAB, the previously discussed algorithm will be implemented. However, before proceeding to try and extract the images displayed in Figs. 171 and 172, a more idealized case is considered. Consider the two ideal images of Sicily in Fig. 174. These two images will be mixed with two different weights to produce two mixed images. Our objective will be to use the algorithm outlined in the last section to separate out the images. Upon separation, the images can be compared with the ideal representation of Fig. 174. To load in the images and start the processing, the following MATLAB commands are used.

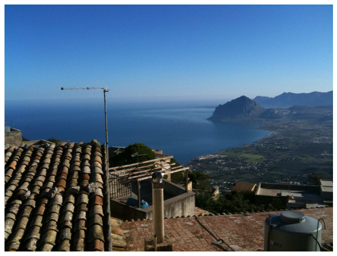
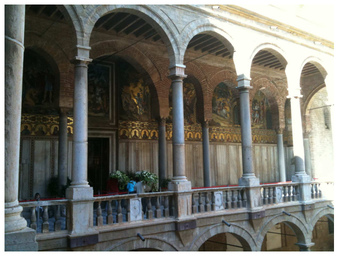



Figure 174: Two images of Sicily: an ocean view from the hills of Erice and an interior view of the Capella Palatina in Palermo.

```
S1=imread('sicily1','jpeg');
S2=imread('sicily2','jpeg');

subplot(2,2,1), imshow(S1);
subplot(2,2,2), imshow(S2);
```

The two images are quite different with one overlooking the mediterranean from the medieval village of Erice on the Northwest corner, and the second is of the Capella Palatina in Palermo. Recall that what is required is statistical independence of the two images. Thus the pixel strengths and colors should be largely decorrolated throughout the image.

The two ideal images are now mixed with the matrix **A** in Eq. (16.1.6). Specifically, we will consider the arbitrarily chosen mixing of the form

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 4/5 & \beta \\ 1/2 & 2/3 \end{bmatrix} \tag{16.3.29}$$

where in what follows we will consider the values $\beta = 1/5, 3/5$. This mixing produces two composite images with the strengths of image one and two altered between the two composites. This can be easily accomplished in MATLAB with the following commands.

```
A=[0.8 0.2; 1/2 2/3];  % mixing matrix
X1=double(A(1,1)*S1+A(1,2)*S2);
X2=double(A(2,1)*S1+A(2,2)*S2);

subplot(2,2,1), imshow(uint8(X1));
subplot(2,2,2), imshow(uint8(X2));
```

Note that the coefficients of the mixing matrix **A** will have a profound impact on our ability to extract one image from another. Indeed, just switching the parameter β from 1/5 to 3/5 will show the impact of a small change to the

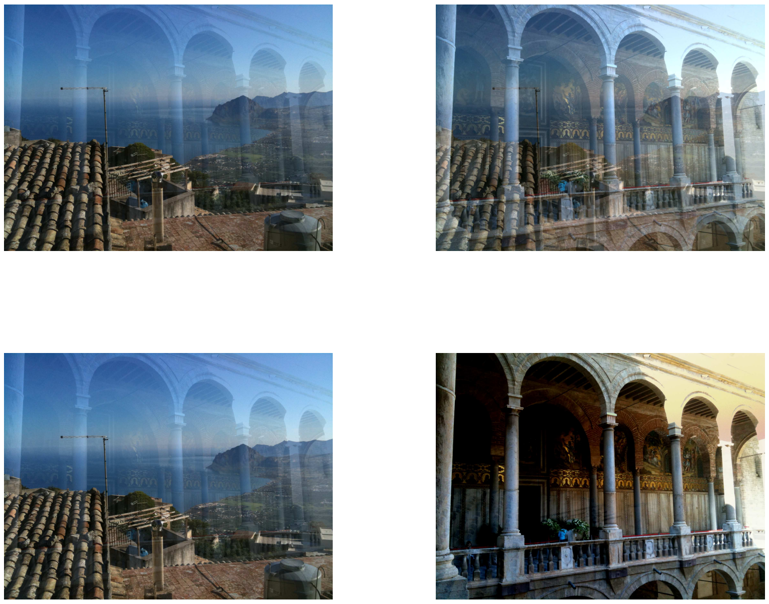


Figure 175: The top two figures show the mixed images produced from Eq. (16.3.29) with $\beta = 1/5$. The bottom two figures are the reconstructed images from the image separation process and approximation to the SVD. For the value of $\beta = 1/5$, the Capella Palatina has been exceptionally well reconstructed while the view from Erice has quite a bit of residual from the second image.

mixing matrix. It is these images that we wish to reconstruct by numerically computing and approximation to the SVD. The top row of Figs. 175 and 176 demonstrate the mixing that occurs with the two ideal images given the mixing matrix (16.3.29) with $\beta = 1/5$ (Fig. 175) and $\beta = 3/5$ (Fig. 176).

Image statistics and the SVD

The image separation method relies on the statistical properties of the image for reconstructing an approximation to the SVD decomposition. Three steps have been outlined in the last section that must be followed: first the rotation of the parallelogram must be computed by finding the maximal and minimal directions of the variance of the data. Second, the scaling of the principal component directions are evaluated by calculating the variance, and third, the final rotation is computed by minimizing both the variance and kurtosis of the data. This yields an approximately separable probability distribution. The three steps are each handled in turn.

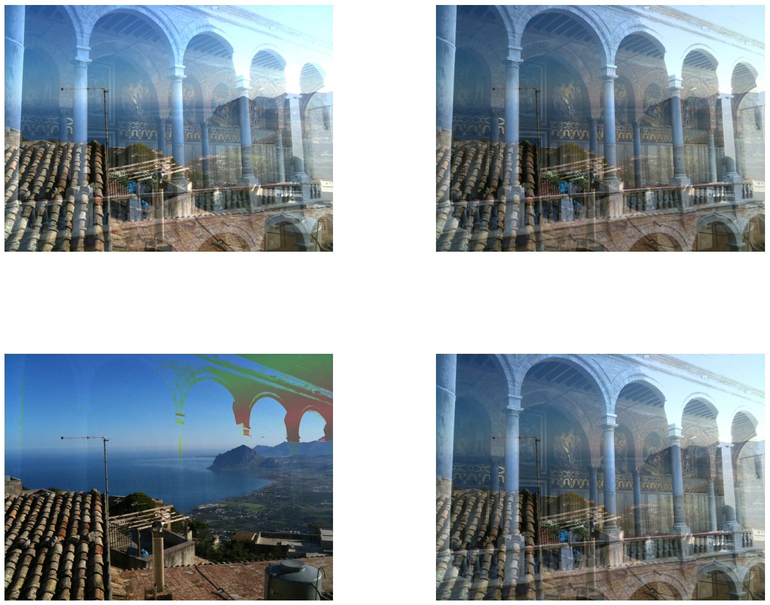


Figure 176: The top two figures show the mixed images produced from Eq. (16.3.29) with $\beta = 3/5$. The bottom two figures are the reconstructed images from the image separation process and approximation to the SVD. For the value of $\beta = 3/5$, the view from Erice has been has been well reconstructed aside from the top right corner of the image while the Capella Palatina remains of poor quality.

Step 1: maximal/minimal variance angle detection. This step is illustrated in the top right of Fig. 173. The actual calculation that must be performed for calculating the rotation angle is given by either Eq. (16.2.14) or (16.2.15). Once computed, the desired rotation matrix $\mathbf{U}^*$ given by Eq. (16.2.16) can be constructed. Recall the underlying assumption that a mean-zero distribution is considered. In MATLAB form, this computation takes the following form.

```
[m,n]=size(X1);
x1=reshape(X1,m*n,1);
x2=reshape(X2,m*n,1);
x1=x1-mean(x1); x2=x2-mean(x2);

theta0=0.5*atan( -2*sum(x1.*x2) / sum(x1.^2-x2.^2) )
Us=[cos(theta0) sin(theta0); -sin(theta0) cos(theta0)];
```

This completes the first step in the SVD reconstruction process.

Step 2: scaling of the principal components. The second step is to undo the scaling/compression that has been performed by the singular values along the principal component directions θ_0 and $\theta_0 - \pi/2$. The rescaling matrix thus is computed as follows:

```
sig1=sum( (x1*cos(theta0)+x2*sin(theta0)).^2 )
sig2=sum( (x1*cos(theta0-pi/2)+x2*sin(theta0-pi/2)).^2 )
Sigma=[1/sqrt(sig1) 0; 0 1/sqrt(sig2)];
```

Note that the variable called sigma above is actually the inverse of the diagonal matrix constructed from the SVD process. This completes the second step in the SVD process.

Step 3: rotation to separability. The final rotation is aimed towards producing, as best as possible, a separable probability distribution. The analytic method used to do this is to minimize both the variance and kurtosis of the remaining distributions. The angle that accomplishes this task is computed analytically from either (16.2.23) or (16.2.24) and the associated rotation matrix $\mathbf{V}$ is given by (16.2.25). Before computing this final rotation, the image after steps 1 and 2 must be produced, i.e. we compute the $\bar{\mathbf{X}}_1$ and $\bar{\mathbf{X}}_2$ used in (16.2.23) and (16.2.24). The MATLAB code used to produce the final rotation matrix is then

```
X1bar= Sigma(1,1)*(Us(1,1)*X1+Us(1,2)*X2);
X2bar= Sigma(2,2)*(Us(2,1)*X1+Us(2,2)*X2);

x1bar=reshape(X1bar,m*n,1);
x2bar=reshape(X2bar,m*n,1);

phi0=0.25*atan( -sum(2*(x1bar.^3).*x2bar-2*x1bar.*(x2bar.^3)) ...
  / sum(3*(x1bar.^2).*(x2bar.^2)-0.5*(x1bar.^4)-0.5*(x2bar.^4)) )

V=[cos(phi0) sin(phi0); -sin(phi0) cos(phi0)];
S1bar=V(1,1)*X1bar+V(1,2)*X2bar;
S2bar=V(2,1)*X1bar+V(2,2)*X2bar;
```

This final rotation completes the three steps of computing the SVD matrix. However, the images needed to be rescaled back to the full image brilliance. Thus the data points need to be made positive and scaled back to values ranging from 0 (dim) to 255 (bright). This final bit of code rescales and plots the separated images.

```
min1=min(min(min(S1bar)));
```

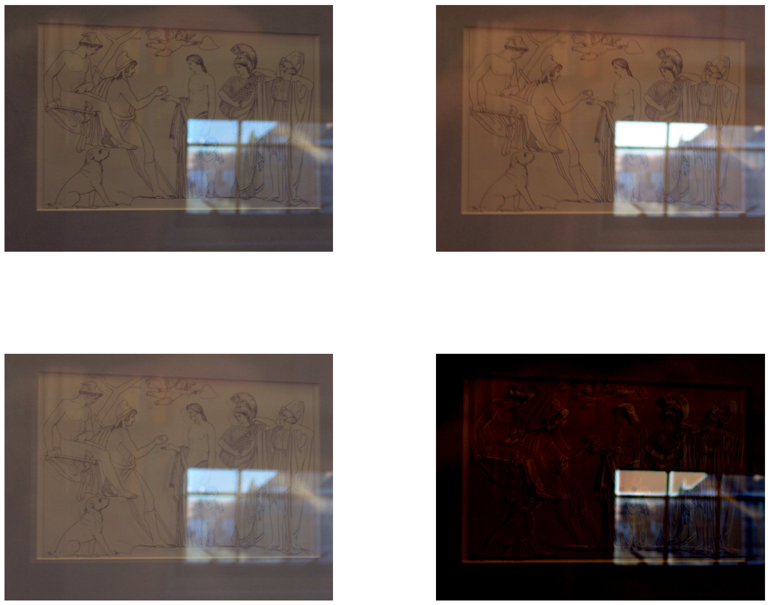


Figure 177: **The Judgement of Paris** by Flaxman with a reflection of my window which overlooks Drumheller Fountain. In the top row are the two original photos taken with linear polarizer rotated 90 degrees between shots. The bottom row represents the separated images. The brightness of the reflected image greatly effects the quality of the image separation.

```
S1bar=S1bar-min1;
max1=max(max(max(S1bar)));
S1bar=S1bar*(255/max1);

min2=min(min(min(S2bar)));
S2bar=S2bar-min2;
max2=max(max(max(S2bar)));
S2bar=S2bar*(255/max2);

subplot(2,2,3), imshow(uint8(S1bar))
subplot(2,2,4), imshow(uint8(S2bar))
```

Note that the successive use of the **min/max** command is due to the fact that the color images are $m \times n \times 3$ matrices where the 3 levels give the color mapping information.

Image separation from reflection

The algorithm above can be applied to the original problem presented of the **Judgement of Paris** by Flaxman. The original images are again illustrated in the top row of Fig. 177. The difference in the two figures was the application of a polarizer whose angle was rotated 90 degrees between shots. The bottom two figures shows the extraction process that occurs for the two images. Ultimately, the technique applied here did not do such a great job. This is largely due to the fact that the reflected image was so significantly brighter than the original photographed image itself, thus creating a difficult image separation problem. Certainly experience tells us that we also have this same difficulty: when it is very bright and sunny out, reflected images are much more difficult for us to see through to the true image.

17 Image Recognition: Basics of Machine Learning

One of the most intriguing mathematical developments of the past decade concerns the ideas of image recognition. In particular, mathematically plausible methods are being rapidly developed to mimic real biological processes towards automating computers to recognize people, animals, and places. Indeed, there are many applications, many of them available on smart phones, that can perform sophisticated algorithms for doing the following: tag all images in a photo library with a specific person, and automatically identify the location and name a set of mountain peaks given a photo of the mountains. These image recognition methods are firmly rooted in the basic mathematical techniques to be described in the following sections. It is remarkable the pace of this technology and its potential for scientific impact. The methods here will bring together statistics, time-frequency analysis and the SVD.

Taken as a whole, the dimensionality reduction and statistical methods presented here are a simple example of so-called *machine learning*, otherwise known as *statistical learning*. More broadly, such data classification schemes fall under pattern theory and are becoming a dominant paradigm in computer science for handling *big data*. In this example, a *supervised learning* technique is implemented in which a prescribed set of data is known and classified beforehand. Alternatively, one could modify the code to do *unsupervised learning* so that the code itself learns to classify and group the data in an appropriate fashion. We will not dwell on these issues here, but will simply present an intuitive example of the machine learning architecture.

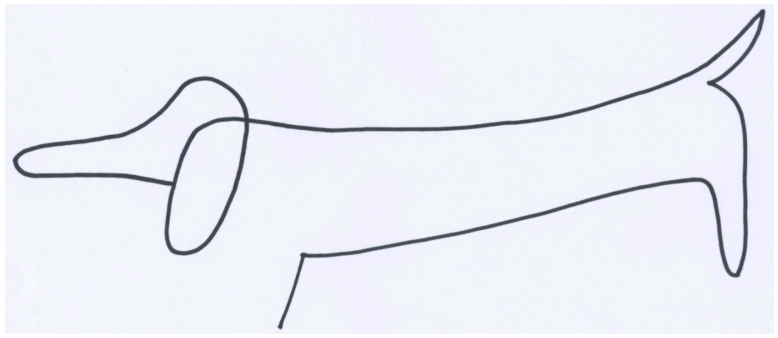


Figure 178: Author's sketch of a dog inspired by Picasso's own sketch of a dog. Note the identification of the image occurs simply from drawing the edges of the animal.

17.1 Recognizing dogs and cats

A basic and plausible methodology will be formulated in the context of image recognition. Specifically, we will generate a mathematically plausible mechanism by which we can discriminate between dogs and cats. Before embarking on this pursuit however, it is useful to consider our own perception and image recognition capabilities. To some extent, the art world has been far ahead to the mathematicians and scientists on this, often asking the question about how a figure can be minimally represented. For instance, consider the Picasso inspired image represented in Fig. 178. With very simple lines, Picasso was able to represent a dog. This example is particular striking given that not only do we recognize a dog, but we recognize the specific dog represented: a dauschung. It is clear from these examples that the concept of edge detection must play a critical role in the image recognition problem. The question to ask is this: could a computer be trained the recognize the figures and animals represented by Picasso in these works.

Picasso certainly pushed much further in the representation of figures and images. Indeed, many of his paintings challenge our ability to perceive the figures within them. But the fact remains, we often do perceive the figures represented even if they are remarkably abstract and distorted from our normal perception of what a human figure should look like, and yet there remains clear indications to our senses of the figures within the abstractions. This highlights the amazingly robust imaging system available to humans.

Wavelet analysis of images and the SVD and LDA

As discussed previously in the time-frequency analysis, wavelets represent an ideal way to represent multiscale information. As such, wavelets will be used here to represent images of dogs and cats. Specifically, wavelets are tremendously efficient in detecting and highlighting edges in images. In the image detection of cats versus dogs, the edge detection will be the dominant aspect of the discrimination process.

The mathematical objective is to develop an algorithm by which we can train the computer to distinguish between dogs and cats. The algorithm will consist of the following key steps:

- **Step 1:** Decompose images of dogs and cats into wavelet basis functions. The motivation is simply to provide an effective method for doing edge detection. The training sets for dogs and cats will be 80 images each.
- **Step 2:** From the wavelet expanded images, find the principal components associated with the dogs and cats respectively.
- **Step 3:** Design a statistical decision threshold for discrimination between the dogs and cats. The method to be used here will be a *linear discrimination analysis* (LDA).
- **Step 4:** Test the algorithm efficiency and accuracy by running through it 20 pictures of cats and 20 pictures of dogs.

The focus of the remainder of this lecture will simply be on step 1 of the algorithm and the appliation of a wavelet analysis.

Wavelet decomposition

To begin the analysis, a sample of the dog pictures is presented. The files **dogData.mat** contains a matrix **dog** which is a 4096×80 matrix. The 4096 data points correspond to a 64×64 resolution image of a dog while the 80 columns are 80 different dogs for the training set of dogs. A similar files exists for the cat data called **catData.mat**. The following command lines loads the data and plots the first nine dog faces.

```
load dogData
for j=1:9
subplot(3,3,j)
  dog1=reshape(dog(:,j),64,64);
  imshow(dog1)
end
```

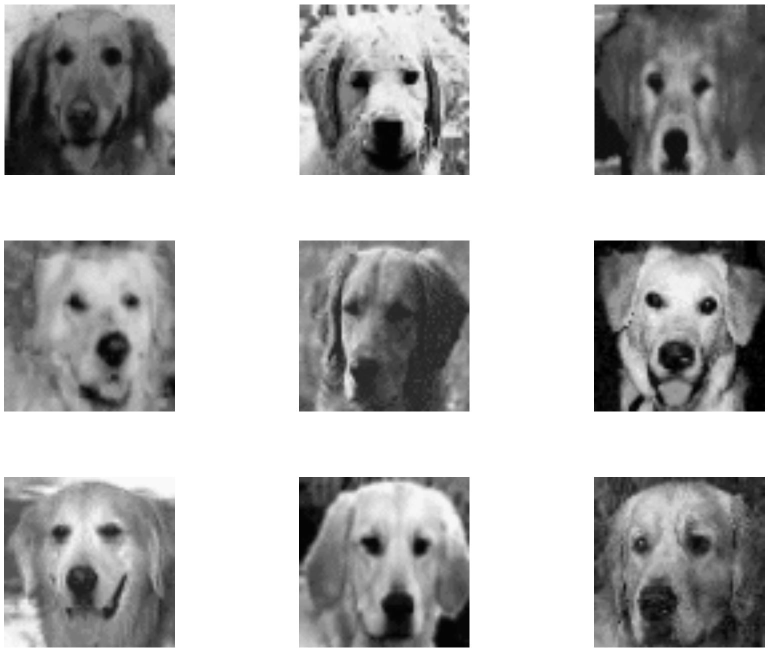


Figure 179: A sampling of nine dog faces to be used in the training set for the dog versus cat recognition algorithm.

Figure 179 demonstrates nine typical dog faces that are used in training the algorithm to recognize dogs versus cats. Note that high resolution images will not be required to perform this task. Indeed, a fairly low resolution can accomplish the task quite nicely.

These images are now decomposed into a wavelet basis using the discrete wavelet transform in MATLAB. In particular, the **dwt2** command performs a single-level discrete 2-D wavelet transform, or decomposition, with respect to either a particular wavelet or particular wavelet filters that you specify. The command

```
[cA,cH,cV,cD] = DWT2(X,'wname')
```

computes the approximation coefficients matrix **cA** and details coefficients matrices **cH**, **cV**, **cD**, obtained by a wavelet decomposition of the input image matrix **X**. The input **'wname'** is a string containing the wavelet name. Thus the large scale features are in **cA** and the fine scale features in the horizontal, vertical and diagonal directions are in **cH**, **cV**, **cD** respectively. The command lines

```
figure(2)
```

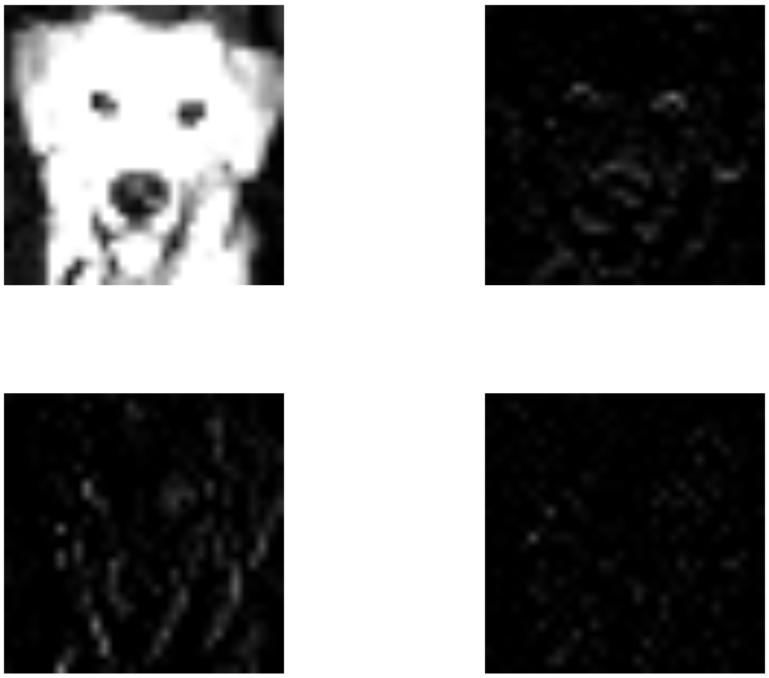


Figure 180: Wavelet decomposition into the matrices (a) **cA** (large scale structures), (b) **cH** (fine scales in the horizontal direction), (c) **cV** (fine scales in the vertical direction), and (d) **cD** (fine scales in the diagonal direction).

```
X=double(reshape(dog(:,6),64,64));
[cA,cH,cV,cD]=dwt2(X,'haar');
subplot(2,2,1), imshow(uint8(cA));
subplot(2,2,2), imshow(uint8(cH));
subplot(2,2,3), imshow(uint8(cV));
subplot(2,2,4), imshow(uint8(cD));
```

produce the Haar wavelet decomposition in four 32×32 matrices. Figure 180 shows the wavelet decomposition matrices for the sixth dog in the data set. Part of the reason the images are so dark in Fig. 180 is that the images have not been rescaled to the appropriate pseudocolor scaling used by the image commands. The following line of code note only rescales the images, but also combines the edge detection wavelets in the horizontal and vertical direction into a single matrix.

```
nbcol = size(colormap(gray),1);
cod_cH1 = wcodemat(cH,nbcol);
```

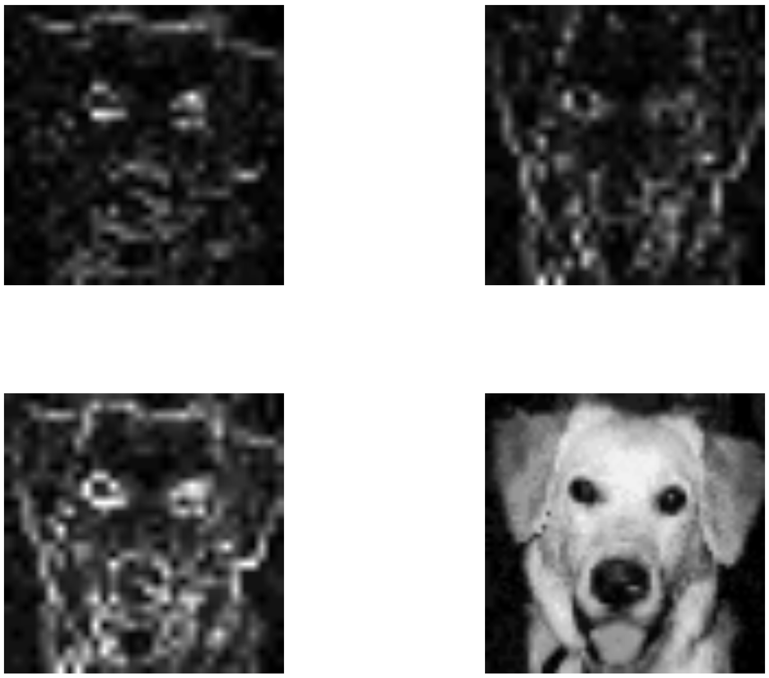


Figure 181: This figure represents the (a) horizontal and (b) vertical representations of the wavelet transform in the appropriately scaled image scale. The overall wavelet transformation process is represented in (c) where the edge detection in the horizontal and vertical directions are combined to represent the ideal dog in (d).

```
cod_cV1 = wcodemat(cV,nbcol);
cod_edge=cod_cH1+cod_cV1;

figure(3)
subplot(2,2,1), imshow(uint8(cod_cH1))
subplot(2,2,2), imshow(uint8(cod_cV1))
subplot(2,2,3), imshow(uint8(cod_edge))
subplot(2,2,4), imshow(reshape(dog(:,6),64,64))
```

The new matrices have been appropriate scaled to the correct image brightnesses so that the images now appear sharp and clean in Fig. 181. Indeed, the bottom two panels of this figure show the ideal dog along with the dog represented in its wavelet basis constructed by weighting the vertical and horizontal directions.

The cat data set can similarly be analyzed. Figures. 182 and 183 demonstrate the wavelet decomposition of a cat. Specifically, the bottom two panels of

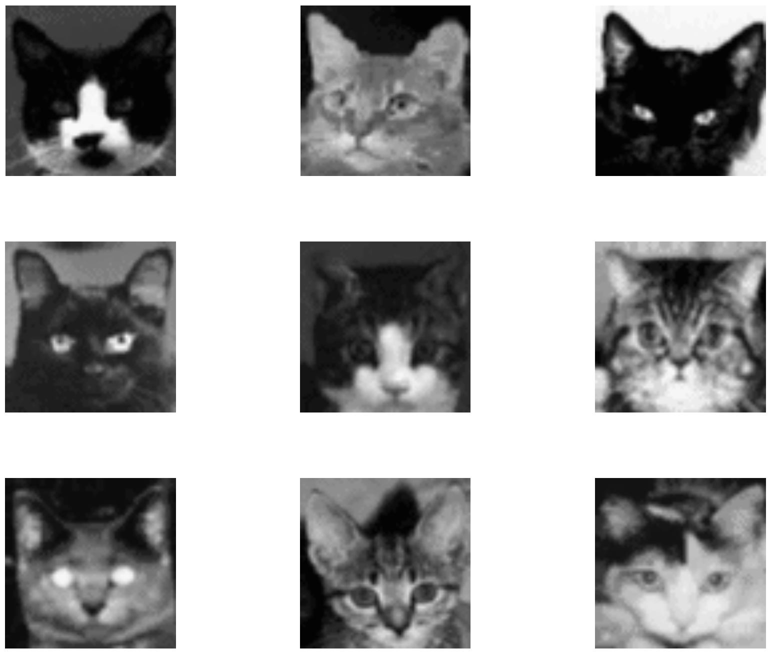


Figure 182: A sampling of nine cat faces to be used in the training set for the dog versus cat recognition algorithm.

Fig. 183 show the ideal cat along with the cat represented in its wavelet basis constructed by weighting the vertical and horizontal directions. Just as before, the analysis was done for sixth cat in the data set. This choice of cat (and dog) was arbitrary, but it does illustrate the general principal quite nicely. Indeed the bottom right figures of both Fig. 181 and 183 will form the basis of our statistical decision making process.

17.2 The SVD and Linear Discrimination Analysis

Having decomposed the images of dogs and cats into a wavelet representation, we now turn our attention to understanding the mathematical and statistical distinction between dogs and cats. In particular, steps 2 and 3 in the image recognition algorithm of the previous section will be addressed here. In step 2, our aim will be to decompose our dog and cat data sets via the SVD, i.e. dogs and cats will be represented by principal components or by a proper orthogonal mode decomposition. Following this step, a statistical decision based upon this representation will be made. The specific method used here is called a *linear discrimination analysis* (LDA).

To begin, the following main function file is created:

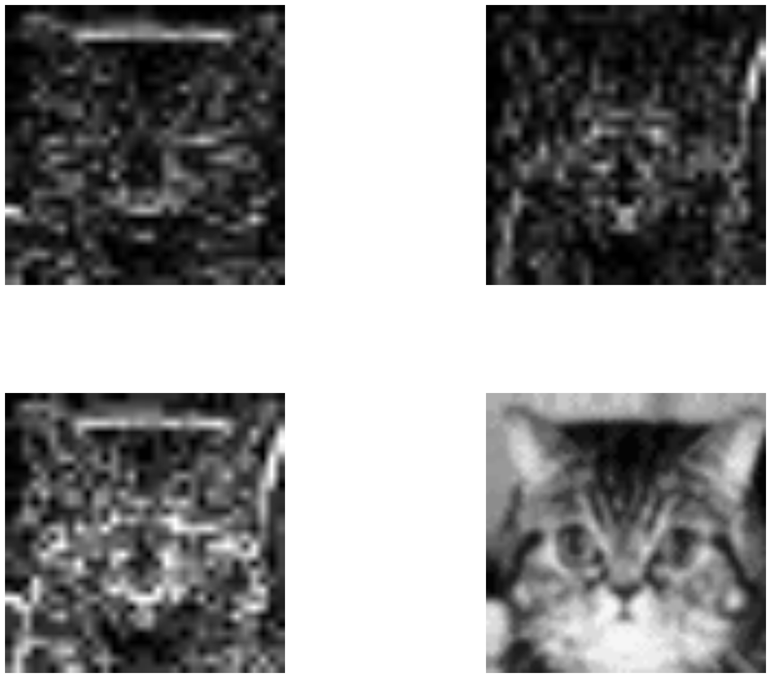


Figure 183: This figure represents the (a) horizontal and (b) vertical representations of the wavelet transform in the appropriately scaled image scale. The overall wavelet transformation process is represented in (c) where the edge detection in the horizontal and vertical directions are combined to represent the ideal dog in (d).

```
load dogData
load catData

dog_wave = dc_wavelet(dog);
cat_wave = dc_wavelet(cat);

feature=20;  %  1<feature<80
[result,w,U,S,V,threshold] = dc_trainer(dog_wave,cat_wave,feature);
```

In this algorithm, the dog and cat data are loaded and run through the wavelet algorithm of the last section. As a subroutine, this takes on the following form

```
function dcData = dc_wavelet(dcfile)
[m,n]=size(dcfile);  % 4096 x 80
nw=32*32;  % wavelet resolution
```

```
nbcol = size(colormap(gray),1);

for i=1:n
    X=double(reshape(dcfile(:,i),64, 64));
    [cA,cH,cV,cD]=dwt2(X,'haar');
    cod_cH1 = wcodemat(cH,nbcol);
    cod_cV1 = wcodemat(cV,nbcol);
    cod_edge=cod_cH1+cod_cV1;
    dcData(:,i)=reshape(cod_edge,nw,1);
end
```

where is should be recalled that the **cA**, **cH**, **cV** and **cD** are the wavelet decompositions with a resolution of 32×32. Upon generating this data, the subroutine **dc_trainer.m** is called. This subroutine will perform and SVD and LDA in order to train itself to recognize dogs and cats. The variable **feature** determines the number of principal components that will be used to classify the dog and cat features.

The SVD decomposition

The first step in setting up the statistical analysis is the SVD decomposition of the wavelet generated data. The subroutine **dc_trainer.m** accepts as input the wavelet decomposed dog and cat data along with the number of PCA/POD components to be considered with the variable **feature**. Upon entry into the subroutine, the dog and cat data are concatenated and the SVD is performed. The following code performs with the reduced SVD to extract the key components **U**, $\mathbf{\Sigma}$ and **V**.

```
function [result,w,U,S,V,th]=dc_trainer(dog0,cat0,feature)
nd=length(dog0(1,:)); nc=length(cat0(1,:));

[U,S,V] = svd([dog0,cat0],0); % reduced SVD

animals = S*V';
U = U(:,1:feature);
dogs = animals(1:feature,1:nd);
cats = animals(1:feature,nd+1:nd+nc);
```

Recall that only a limited number of the principal components are considered in the feature detection as determined by the variable **feature**. Thus upon completion of the SVD step, a limited number of columns of **U** and rows of $\mathbf{\Sigma V}^*$ are extracted. Further, new variables **cats** and **dogs** are generated giving the strength of the cat and dog projections on the PCA/POD modes generated by **U**.

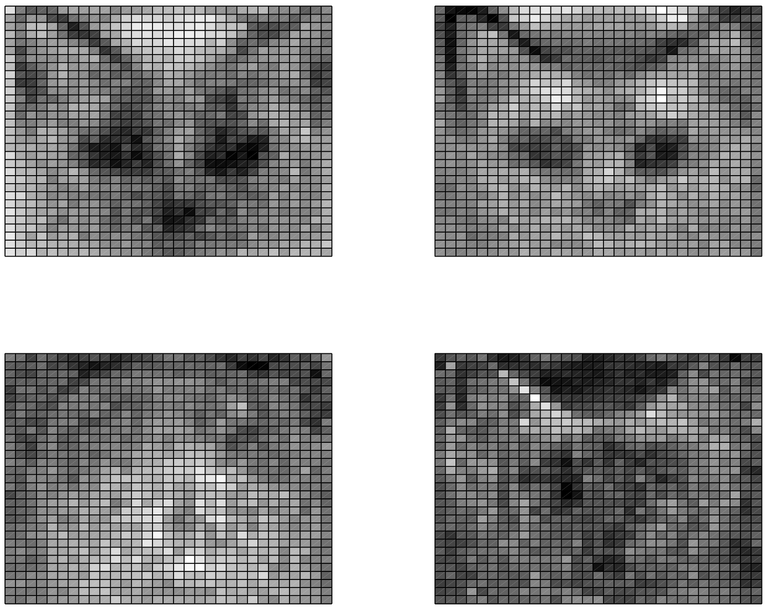


Figure 184: Representation of the first four Principal components, or proper orthogonal modes, of the cat and dog data. Note the dominance of the triangular ears in the first two modes, suggesting a strong signature of a cat. These modes are used as the basis for classifying a cat or dog image.

To get a better feeling of what has just happened, the SVD matrices **U**, **Σ** and **V** are all returned in the subroutine to the main file. In the main file, the decomposition can be visualized. What is of most interest is the PCA/POD that is generated from the cat and dog data. This can be reconstructed with the following code

```
for j=1:4
  subplot(2,2,j)
  ut1=reshape(U(:,j),32,32);
  ut2=ut1(32:-1:1,:);
  pcolor(ut2)
  set(gca,'Xtick',[],'Ytick',[])
end
```

Thus the first four columns, or PCA/POD modes, are plotted by reshaping them back into 32×32 images. Figure 184 shows the first four POD modes associated with the dog and cat data. The strength of each of these modes in a given cat or dog can be computed by the projection onto $\mathbf{\Sigma V}^*$. In particular, the singular (diagonal) elements of $\mathbf{\Sigma}$ gives the strength of the projection of

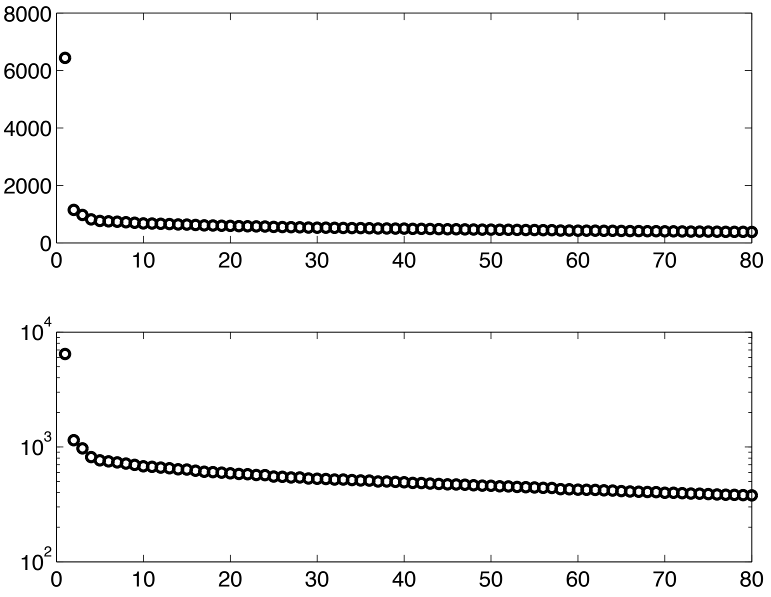


Figure 185: Singular values of the SVD decomposition of the dog and cat data. Note the dominance of a single singular value and the heavy-tail distribution (non-Gaussian) fall off of the singular values.

each cat onto the POD modes. Figures 185 and 186 demonstrate the singular values associated with the cat and dog data along with the strength of each cat projected onto the POD modes. These two plots are created from the main file with the commands

```
figure(2)
subplot(2,1,1)
plot(diag(S),'ko','Linewidth',[2])
set(gca,'Fontsize',[14],'Xlim',[0 80])
subplot(2,1,2)
semilogy(diag(S),'ko','Linewidth',[2])
set(gca,'Fontsize',[14],'Xlim',[0 80])

figure(3)
for j=1:3
    subplot(3,2,2*j-1)
    plot(1:40,V(1:40,j),'ko-')
    subplot(3,2,2*j)
    plot(81:120,V(81:120,j),'ko-')
end
```

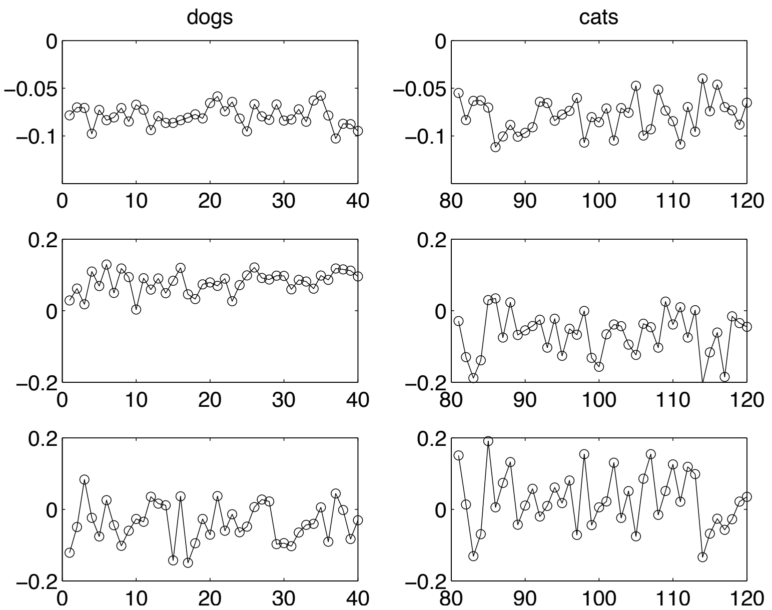


Figure 186: Projection of the first 40 individual cats and dogs onto the first three POD modes as described by the SVD matrix **V**. The left column is the first 40 dogs while the right column is the first 40 cats. The three rows of figures are the projections on to the first three POD modes.

```
  subplot(3,2,1), set(gca,'Ylim',[-.15 0],'Fontsize',[14]), title('dogs')
  subplot(3,2,2), set(gca,'Ylim',[-.15 0],'Fontsize',[14]), title('cats')

  subplot(3,2,3), set(gca,'Ylim',[-.2 0.2],'Fontsize',[14])
  subplot(3,2,4), set(gca,'Ylim',[-.2 0.2],'Fontsize',[14])
  subplot(3,2,5), set(gca,'Ylim',[-.2 0.2],'Fontsize',[14])
  subplot(3,2,6), set(gca,'Ylim',[-.2 0.2],'Fontsize',[14])
```

Note the dominance of a single mode and the heavy-tail distribution of the remaining singular values. The singular values in Fig. 185 are plotted on both a normal and log scale for convenience. Figure 186 demonstrates the projection strength of the first 40 individual cats and dogs onto the first three POD modes. Specifically, the first three mode strengths correspond to the rows of the figures. Note the striking difference in cats and dogs as shown in the second mode. Namely, for dogs, the first and second mode are of opposite sign and "cancel" whereas for cats, the signs are the same and the features of the first and second mode add together. Thus for cats, the triangular ears are emphasized. This starts to give some indication of how the discrimination might work.

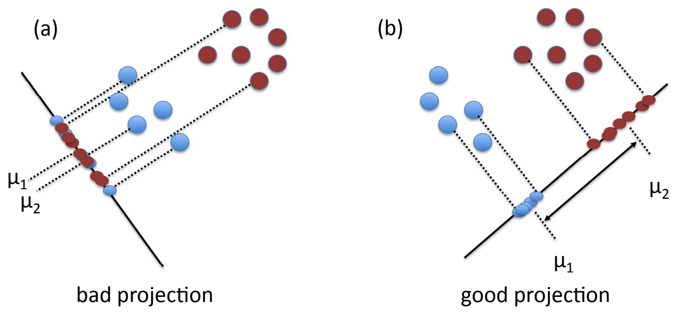


Figure 187: Graphical depiction of the process involved in a two-class linear discrimination analysis. Two data sets are projected onto a new basis function. In the left figure, the projection produces highly mixed data and very little distinction or separation can be drawn between data sets. In the right figure, the projection produces the ideal, well-separated statistical distribution between the data sets. The goal is to mathematically construct the optimal projection basis which separates the data most effectively.

Linear Discrimination Analysis

Thus far, two decompositions have been performed: wavelets and SVD. The final piece in the training algorithm is to use statistical information about the wavelet/SVD decomposition in order to make a decision about wether the image is indeed a dog or cat. Figure 187 gives a cartoon of the key idea involved in the LDA. The idea of the LDA was first proposed by Fisher [49] in the context of taxonomy. In our example, two data sets are considered and projected onto new bases. In the left figure, the projection shows the data to be completely mixed, not allowing for any reasonable way to separate the data from each other. In the right figure, which is the ideal charicature for LDA, the data are well separated with the means μ_1 and μ_2 being well apart when projected onto the chosen subspace. Thus the goal of LDA is two-fold: *find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data.*

For a two-class LDA, the above idea results in consideration of the following mathematical formulation. Construct a projection $\mathbf{w}$ such that

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \qquad (17.2.30)$$

where the scatter matrices for between-class $\mathbf{S}_B$ and within-class $\mathbf{S}_W$ data are given by

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{17.2.31}$$

$$\mathbf{S}_W = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T \, . \tag{17.2.32}$$

These quantities essentially measure the variance of the data sets as well as the variance of the difference in the means. There are a number of tutorials available online with details of the method, thus the mathematical details will not be presented here. The criterion given by Eq. (17.2.30) is commonly known as the generalized Rayleigh quotient whose solution can be found via the generalized eigenvalue problem

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w} \tag{17.2.33}$$

where the maximum eigenvalue λ and its associated eigenvector gives the quantity of interest and the projection basis. Thus once the scatter matrices are constructed, the generalized eigenvectors can easily be constructed with MATLAB.

Once the SVD decomposition has been performed in the **dc_trainer.m** subroutine, the LDA is applied. The following code produces the LDA projection basis as well as the decision threshold level associated with the dog and cat recognition.

```
 md = mean(dogs,2);
 mc = mean(cats,2);

 Sw=0;  % within class variances
 for i=1:nd
     Sw = Sw + (dogs(:,i)-md)*(dogs(:,i)-md)';
 end
 for i=1:nc
     Sw = Sw + (cats(:,i)-mc)*(cats(:,i)-mc)';
 end

Sb = (md-mc)*(md-mc)';  % between class

 [V2,D] = eig(Sb,Sw);  % linear discriminant analysis
 [lambda,ind] = max(abs(diag(D)));
 w = V2(:,ind);  w = w/norm(w,2);

 vdog = w'*dogs; vcat = w'*cats;
 result = [vdog,vcat];

 if mean(vdog)>mean(vcat)
```

```
    w = -w;
    vdog = -vdog;
    vcat = -vcat;
end
% dog < threshold < cat

sortdog = sort(vdog);
sortcat = sort(vcat);

t1 = length(sortdog);
t2 = 1;
while sortdog(t1)>sortcat(t2)
    t1 = t1-1;
    t2 = t2+1;
end
threshold = (sortdog(t1)+sortcat(t2))/2;
```

This code ends the subroutine **dc_trainer.m** since the statistical threshold level has been determined along with the appropriate LDA basis for projecting a new image of a dog or cat. For **feature=20** as carried through the codes presented here, a histogram can be constructed of the dog and cat statistics.

```
figure(4)
subplot(2,2,1)
hist(sortdog,30); hold on, plot([18.22 18.22],[0 10],'r')
set(gca,'Xlim',[-200 200],'Ylim',[0 10],'Fontsize',[14])
title('dog')
subplot(2,2,2)
hist(sortcat,30,'r'); hold on, plot([18.22 18.22],[0 10],'r')
set(gca,'Xlim',[-200 200],'Ylim',[0 10],'Fontsize',[14])
title('cat')
```

The above code not only produces a histogram of the statistical distributions associated with the dog and cat properties projected to the LDA basis, it also shows the decision threshold that is computed for the statistical decision making problem of categorizing a given image as a dog or cat. Note that as is the case in most statistical processes, the threshold level shows that some cats are mistaken as dogs and vice-versa. The more features that are used, the fewer “mistakes” are made in the recognition process.

17.3 Implementing cat/dog recognition in MATLAB

Up to now, we have performed three key steps in training or organizing our dog and cat data images: we have (i) wavelet decomposed the images, (ii) considered this wavelet representation and its corresponding SVD/PCA/POD, and (iii)

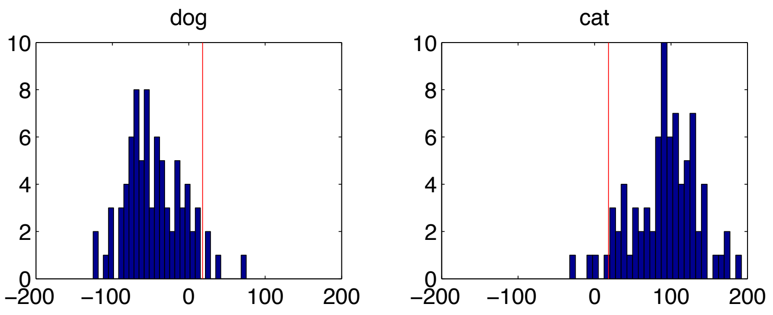


Figure 188: Histogram of the statistics associated with the dog and cat data once projected onto the LDA basis. The red line is the numerically computed decision threshold. Thus a new image would be projected onto the LDA basis a decision would be made concerning the image depending upon which side of the threshold line it sits.

projected these results onto the subspace provided by the linear discrimination analysis. The three steps combined are the basis of the training set of the dog versus cat recognition problem.

It remains now to test the training set (and machine learning) on a sample of dog and cat pictures outside of the training data set. The data set **PatternRecAns.mat** contains 38 new pictures of dogs and cats for us to test our recognition algorithm on. Specifially, two files are contained within this data set: **TestSet** which is a 4096×38 matrix containing the 38 pictures of resolution 64×64, and **hiddenlabels** which is a 1×38 vector containing either 1s (cat) or 0s (dog) identifying the image as a true dog or cat. To see part of the sample set of test pictures, the following line of code is used.

```
load PatternRecAns

figure(1)
for j=1:9
 subplot(3,3,j)
 test=reshape(TestSet(:,j+5),64,64);
 imshow(uint8(test))
end
```

Figure 189 shows 9 or the 38 test images of dogs and cats to be processed by our Wavelet, SVD, LDA algorithm.

To classify the images as dogs or cats, the new data set must now undergo the same process as the training images. Thus the following will occur

- Decompose the new image into the wavelet basis.

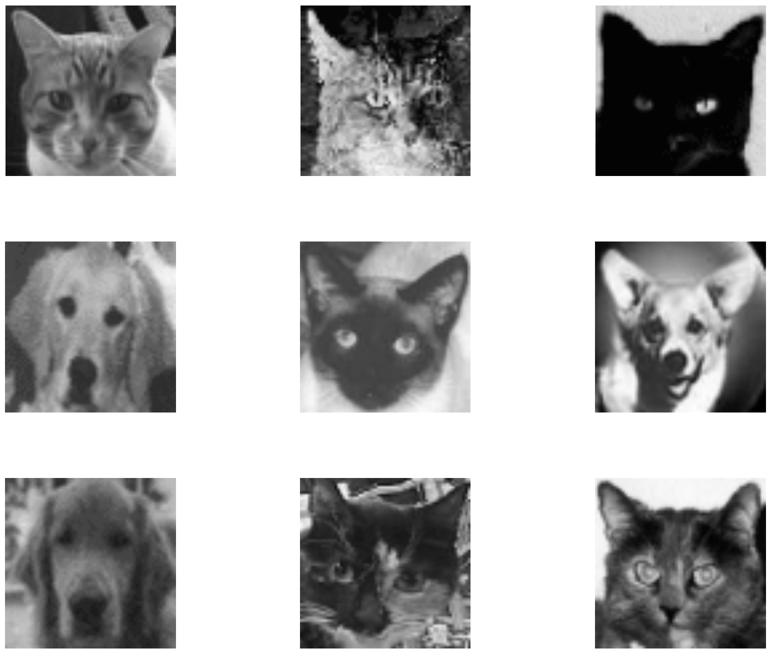


Figure 189: Nine sample images of dogs and cats that will be classified by the Wavelet, SVD, LDA scheme. The second dog in this set (6th picture) will be misclassified due to its ears.

- Project the images onto the SVD/PCA/POD modes selected from the training algorithm.
- Project this two-level decomposition onto the LDA eigenvector.
- Determine the projection value relative to the LDA threshold determined in the training algorithm.
- Classify the image as dog or cat.

The following lines of code perform the first three tasks above.

```
Test_wave = dc_wavelet(TestSet);  % wavelet transformation
TestMat = U'*Test_wave;        % SVD projection
pval = w'*TestMat;          % LDA projection
```

Note that the subroutine **dc_wavelet.m** is considered in the previous section. Further, it is assumed that the training algorithm of the last section has been run so that both the SVD matrix **U** has been computed, and the first 20 columns (features) have been preserved, and the eigenvector **w** of the LDA has been computed. Note that the above code performs the three successive transformations necessary for this problem: wavelet, SVD, LDA.

It only remains to see how the recognition algorithm worked. The variable **pval** contains the final projection to the LDA subspace and it must be compared with the threshold determined by the LDA. The following code produces a new vector **ResVec** containing 1s and 0s depending upon wether **pval** is above or below threshold. The vector **ResVec** is then compared with **hiddenlabels** which has the true identification of the dog and cat data. These are then compared for accuracy.

```
%cat = 1, dog = 0
ResVec = (pval>threshold)

disp('Number of mistakes');
errNum = sum(abs(ResVec - hiddenlabels))

disp('Rate of success');
sucRate = 1-errNum/TestNum
```

In addition to identifying and classifying the test data, the number of mistakes, or misidentified, images are reported and the percentage rate of success is given. In this algorithm, a nearly 95% chance of success is given which is quite remarkable given the low-resolution, 2D images and the simple structure of the algorithm.

The mistaken images in the 38 test images can now be considered. In particular, by understanding why mistakes where made in identification, better algorithms can be constructed. The following code identifies which images where mistaken and plots the results.

```
k = 1;
TestNum = length(pval);
figure(2)
for i = 1:TestNum
    if ResVec(i)~=hiddenlabels(i)
        S = reshape(TestSet(:,i),64,64);
        subplot(2,2,k)
        imshow(uint8(S))
        k = k+1;
    end
end
```

Figure 190 shows the two errors that were made in the identification process. These two dogs have the quintessential features of cats: triangular ears. Recall that even in the training set, the decision threshold was set at a value for

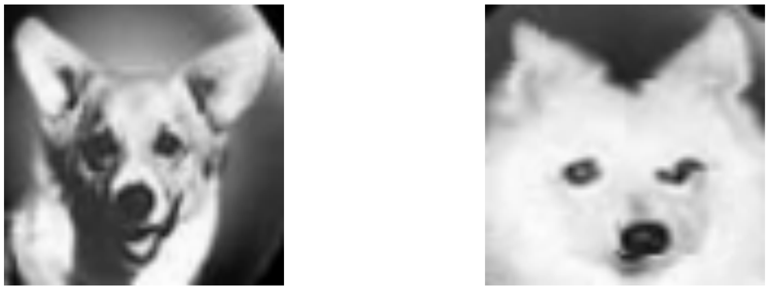


Figure 190: Of the 38 images tested, two mistakes are made. The two dogs above were both misclassified as cats. Looking at the POD of the dog/cat decomposition gives a clue to why the mistake was made: they have triangular ears. Indeed, the image recognition algorithm developed here favors, in some sense, recognition of cats versus dogs simply due to the strong signature of ears that cats have. This is the primary feature picked out in the edge detection of the wavelet expansion.

which some cats and dogs where misclassified. It is these kinds of features that can give the misleading results, suggesting that edge detection is not the only mechanism we use to identify cats versus dogs. Indeed, one can see that much more sophisticated algorithms could be used based upon size of the animal, color, coat pattern, walking cadence, etc. Of course, the more sophisticated the algorithm, the more computationally expensive. This is a fast, low-resolution routine that produces 95% accuracy and can be easily implemented.

18 Basics of Compressed Sensing

In most dimensionality reduction applications, a least-square fitting of the data in higher dimensions is traditionally applied due to both (i) well-developed factorization algorithms like the SVD and (ii) the intuitive appeal offered by an *energy* norm. However, this does not necessary make such an L^2 based reduction the best choice. Indeed, recent progress in the field of *compressed sensing* has shown that there is great advantage in reduction techniques based upon the L^1 norm instead. This will be explored in a series of three lectures where sparse sampling and an L^1 data projection can be used quite effectively for signal and image reconstruction.

18.1 Beyond Least-Square Fitting: The L^1 Norm

Understanding of compressed sensing comes from building an intuitive feel for data fitting and the norms we use to measure our error in such a process. In typical applications, a least-square fitting algorithm is used as the standard measure of error. However, other options are available beyond this standard choice. It should be noted that this standard choice is often chosen since the L^2 norm represents an *energy* norm and often it has a clear physical interpretation.

Curve Fitting: L^2 versus L^1

To begin understanding, let us first consider a standard curve fitting problem. Specifically, consider a best fit curve through a set of data points (x_i, y_i) where $i = 1, 2, ..., N$. This is a fairly standard problem that involves an underlying optimization routine. Section 3 outlines the basic idea which are reviewed here.

To solve this problem, it is desired to minimize some error between the line fit

$$y(x) = Ax + B \tag{18.1.1}$$

and the data points (x_i, y_i). Thus an error can be defined as follows:

$$E_p = \left(\frac{1}{N} \sum_{n=1}^{N} |y(x_i) - y_i|^p \right)^{1/p} \tag{18.1.2}$$

where p denotes the error norm to consider. For $p = 2$, this gives the standard least-square error E_2. However, other choices are possible and we will consider here $p = 1$ as well.

In order to solve this problem, an optimization routine must be performed to determine the coefficients A and B in the line fit. What is specifically required is to determine A and B from the minimization constraints: $\partial E_p/\partial A = 0$ and $\partial E_p/\partial B = 0$. For a line fit, these are trivial to calculate for either the L^2 or L^1 norms. However, when higher-degree of freedom fits are involved, the optimization routine can be costly. For L^2, the higher-degree of freedom fit is generated in $O(N^3)$ via a the singular-value decomposition. For L^1, a different optimization routine is required. In the end, these curve fitting techniques are simply methods for solving overdetermined systems of equations, i.e. there are N constraints/equations and two unknowns in A and B. Learning how to solve such overdetermined systems is of generic importance.

The difference in L^2 and L^1 norms is most readily apparent when considering data with large variance or many outliers. The L^2 norm squares the distance to these outlying points so that they have a big impact on the curve fit. In contrast, the L^1 norm is much less susceptible to outliers in data as the distance to these points is not squared, but rather only the absolute distance is considered. Thus the L^1 curve fit is highly robust to outliers in the data.

To demonstrate the difference between L^2 and L^1 line fitting, consider the following data set of (x_i, y_i) pairs

```
x=[0.1 0.4 0.7 1.2 1.3 1.7 2.2 2.8 3.0 4.0 4.3 4.4 4.9];
y=[0.5 0.9 1.1 1.5 1.5 2.0 2.2 2.8 2.7 3.0 3.5 3.7 3.9];
```

A line fit can be performed for this data set using both L^2 and L^1 norm minimization using the **fminsearch** command. By generically guessing values of $A = B = 1$, the following two **fminsearch** routines will converge to optimal values of A and B that minimize the L^2 and L^1 norm respectively

```
coeff_L2=fminsearch('line_L2_fit',[1,1],[],x,y)
coeff_L1=fminsearch('line_L1_fit',[1,1],[],x,y)
```

Note that these two function calls return 2×1 vectors with the updated values of A and B. The functions look like the following:

line_L2_fit.m

```
function E=line_L2_fit(x0,x,y)
E=sum( ( x0(1)*x+x0(2)  - y ).^2 )
```

line_L1_fit.m

```
function E=line_L1_fit(x0,x,y)
E=sum( abs( x0(1)*x+x0(2)  - y ) )
```

Note that in each of these, $p = 2$ or $p = 1$ respectively. This makes clear that **fminsearch** is simply an optimization routine and can be used for far more than curve fitting.

Figure 191 (top panel) demonstrates the line fitting process to the data proposed here. In this first example, which is devoid of outliers in the data, both the L^2 and L^1 fit look reasonable the same. Indeed, hardly any difference is seen between the dashed and dotted lines. However, if additional outliers are added to the data set (two outlying points):

```
x=[0.5 3.8];
y=[3.9 0.3];
```

Then significant differences can be seen between the L^2 and L^1 fits. This is demonstrated in Fig. 191 (bottom panel). Note how the L^2 fit is significantly skewed by the outlying data while the L^1 data remains a robust and locked along the bulk of the data. One should keep this example in mind for future reference as the L^2 and L^1 minimization process is effectively what is underlying SVD and compressed sensing in higher dimensions.

Underdetermined Systems

Continuing on this theme, consider now solving an underdetermined system of equations. As a simple example, solve $\mathbf{Ax} = \mathbf{b}$ with

```
A=[1/2 1/4 3 7 -2 10];
b=[3];
```

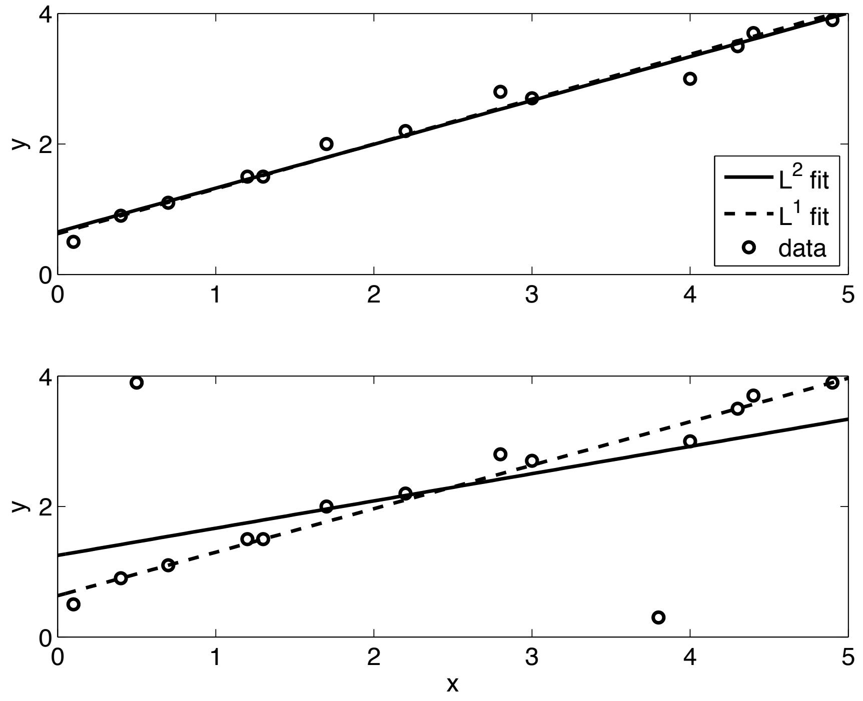


Figure 191: L^2 versus L^1 data fitting for data without outliers (top) and with outliers (bottom). Note the significant skewing of the data that occurs with the L^2 fitting algorithm (solid line) versus the L^1 algorithm (dotted line). The raw data is denoted with circles.

Of course, this has an infinite number of solutions. So one may ask how would an algorithm decide on what solution to pick. Two options that are immediately available to generate a solution are the backslash command and the pseudo-inverse (for underdetermined systems)

```
x1=A\b
x2=pinv(A)*b
```

The solution for both these two cases are strikingly different. For the backslash, the solution produced is sparse, i.e. it is almost entirely all zeros.

```
x1=
         0
         0
         0
         0
```

```
        0
   0.3000
```

In contrast, the solution produced with the pseudo-inverse produces the following solution vector

```
x2=
   0.0092
   0.0046
   0.0554
   0.1294
  -0.0370
   0.1848
```

Thus no sparsity is exhibited in the solution structure.

This example can be scaled up to a much more complex situation. In particular, consider the 100×500 system where there are 100 equations and 500 unknowns. As before, these can be solved using the backslash or pseudo-inverse commands.

```
m=100; n=500;
A=randn(m,n);
b=randn(m,1);

x1=A\b;
x2=pinv(A)*b;
```

In addition to the backslash and pseudo-inverse, L^2 or L^1 optimization can also be applied to the problem. That is, solve the system subject to minimizing the L^2 or L^1 norm of the solution vector. There are a number of open source convex optimization codes that can be downloaded from the internet. In the example here, a convex optimization package is used that can be directly implemented with MATLAB: **http://cvxr.com/cvx/**. In the implementation of the code, the following lines of code generate a solution that minimizes the L^1 residual and L^2 residual respectively

```
    cvx_begin;
       variable x3(n);
       minimize( norm(x3,1) );
       subject to
         A*x3 == b;
    cvx_end;
    cvx_begin;
       variable x4(n);
       minimize( norm(x4,2) );
```

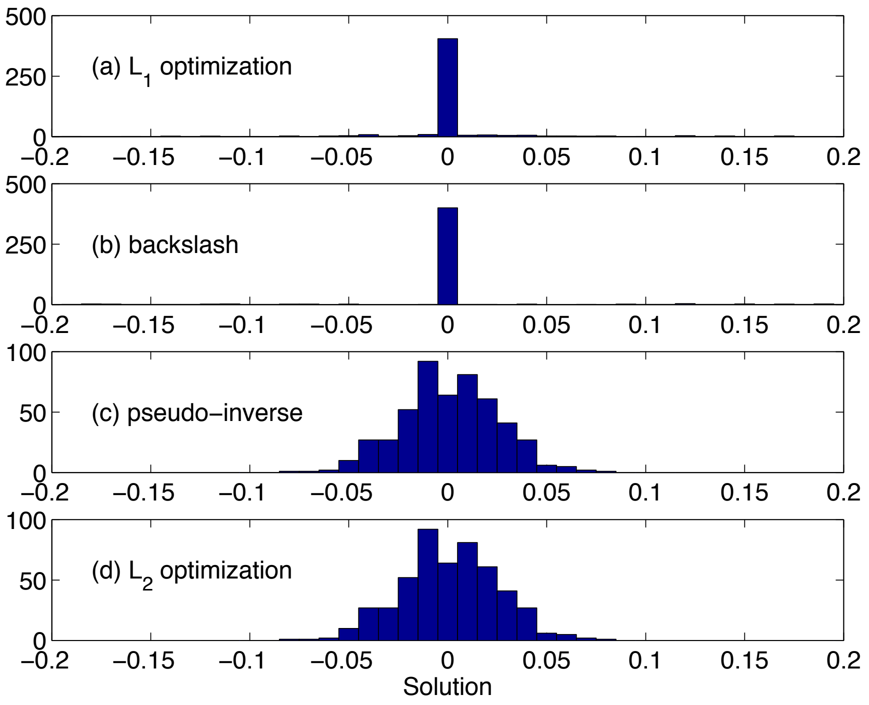


Figure 192: Histogram of the solution of an underdetermined system with 100 equations and 500 unknowns. The backslash and L^1 minimization produce sparse solution vectors while the pseudo-inverse and L^2 minimization do not.

```
    subject to
      A*x4 == b;
cvx_end;
```

Figure 192 demonstrates the use of all four methods for solving this underdetermined system, i.e. backslash, pseudo-inverse, L^1 minimization and L^2 minimization. What is plotted in this figure is a histogram of the values of the solution vector. Note that L^1 optimization and the backslash command both produce solutions that are sparse, i.e. there are a large number of zeros in the solution. Specifically, both these techniques produce 400 zeros and a remaining 100 nonzero elements that satisfy the remaining equations. In contrast, the L^2 optimization and pseudo-inverse produce identical, non-sparse solution vectors.

Figure 193 shows a detail of the backslash and L^1 optimization routines. The purpose of this figure is to show that the backslash and L^1 do indeed produce different results. Note that the backslash histogram is still much more tightly clustered around the zero solution while the L^1 minimization allows for larger

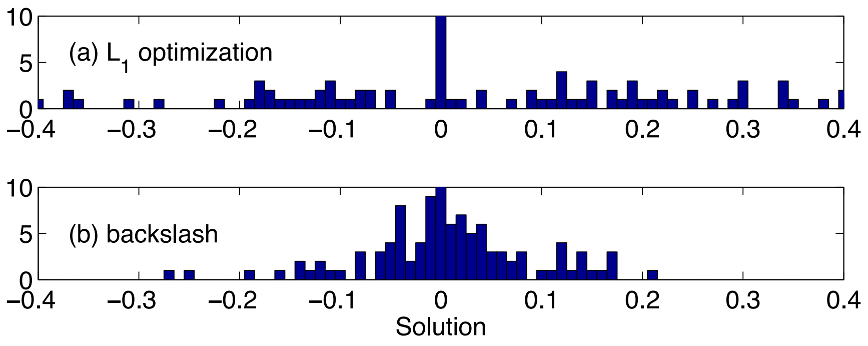


Figure 193: Blow up of the L^1 minimization and backlash solution histograms demonstrating the difference the two solution methods produce.

variance in the nonzero solutions. It is this solution sparsity that will play a critical role in the ability to reconstruct signals and/or images from sparse sampling.

Overdetermined Systems

This example can be also be modified to consider overdetermined systems of equations. In particular, consider the 500×150 system where there are 500 equations and 150 unknowns. As before, these can be solved using the backslash or pseudo-inverse commands.

```
m=500; n=150;
A=randn(m,n);
b=randn(m,1);

x1=A\b;
x2=pinv(A)*b;
```

In addition to the backslash and pseudo-inverse, L^2 or L^1 optimization can also be applied to the problem. That is, solve the system subject to minimizing the L^2 or L^1 norm of the solution vector. In the implementation of the code, the following lines of code generate a solution that minimizes the L^1 and L^2 norm respectively

```
cvx_begin;
    variable x3(n);
    minimize( norm(A*x3-b,1) );
cvx_end;
cvx_begin;
```

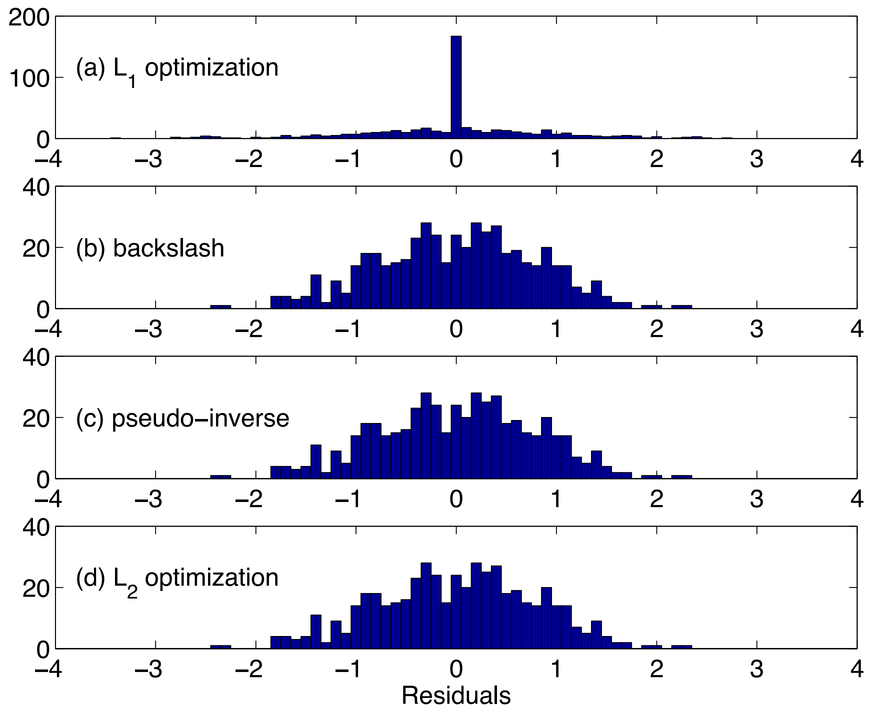


Figure 194: Histogram of the solution residual of an overdetermined system with 500 equations and 150 unknowns. The L^1 minimization produces sparse solution vectors while the pseudo-inverse, backslash, and L^2 minimization do not.

```
    variable x4(n);
    minimize( norm(A*x4-b,2) );
cvx_end;
```

As before, the sparsity of the resulting solution can be illustrated using all four solution methods. Figure 194 demonstrates the use of all four methods for solving this overdetermined system, i.e. backslash, pseudo-inverse, L^1 minimization and L^2 minimization. What is plotted in this figure is a histogram of the solution residual vector. Note that L^1 optimization produces solutions that are sparse, i.e. there are a large number of zeros in the residual. In contrast, the L^2 optimization, backslash, and pseudo-inverse produce identical, non-sparse solution vectors. Sparsity again plays a critical role in the ability to solve the resulting overdetermined system. The critical idea in compressed sensing is to take advantage of sparsity of signals or images in order to perform reconstruction. Indeed, the L^1 norm should be thought of as a proxy for sparsity as it promotes sparsity in its solution technique of over- or underdetermined systems.

18.2 Signal Reconstruction and Circumventing Nyquist

To illustrate the power of the L^1 norm, and ultimately of compressive sensing, it is informative to consider the reconstruction of a temporal signal. This example [50] makes connection directly to signal processing. However, it is fundamentally the same issue that was considered in the last section. Therefore, consider the "A" key on a touch-tone telephone which is the sum of two sinusoids with incommensurate frequencies [50]:

$$f(t) = \sin(1394\pi t) + \sin(3266\pi t) \,. \tag{18.2.3}$$

To begin, the signal will be sampled over 1/8 of a second at 40000 Hz. Thus a vector of length 5000 will be generated. The signal can also be represented using the discrete cosine transform (DCT). In MATLAB, the following lines are sufficient to produce the signal and its DCT:

```
n=5000;
t=linspace(0,1/8,n);
f=sin(1394*pi*t)+sin(3266*pi*t);
ft=dct(f);
```

As can be seen from the form of the signal, there should dominance by two Fourier modes in (18.2.3). However, since the frequencies are incommensurate and do not fall within the frequencies spanned by the DCT, there are a number of nonzero DCT coefficients. Figures 195 and 196 show the original signal over 1/8 of a second and a blow-up of the signal in both time and frequency. The key observation to make: *The signal is highly sparse in the frequency domain.*

The objective now will be to randomly *sample* the signal at a much lower frequency (i.e. sparse sampling) and try to reconstruct the original signal, in both frequency and time, as best as possible. Such an exercise is at the heart of compressive sensing. Namely, given that the signal is sparse (in frequency) to begin with, can we sample sparsely and yet faithfully reconstruct the desired signal [51, 52, 53]. Indeed, compressive sensing algorithms are data acquisition protocols which perform as if where possible to directly acquire just the important information about the signal, i.e. the sparse signal representation in the DCT domain. In effect, the idea is to acquire only the important information and ignore that part of the data which effectively makes no contribution to the signal. Figure 195 clearly shows that most of the information in the original signal can be neglected when considered from the perspective of the DCT domain. Thus the sampling algorithm must find a way to key in on the prominent features of the signal in the frequency domain. Since the L^1 norm promotes sparsity, this will be the natural basis in which to work to reconstruct the signal from our sparse sampling[51, 52, 53].

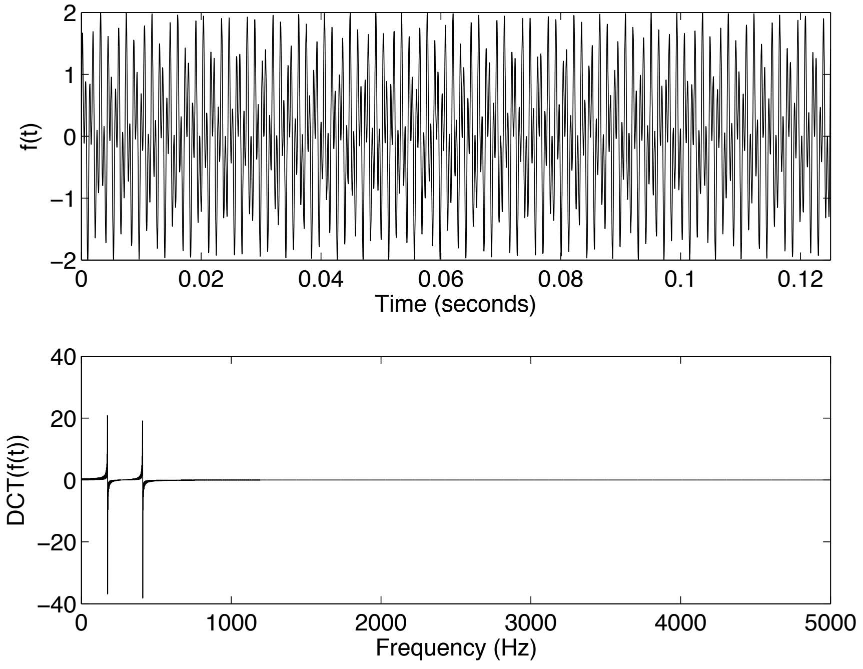


Figure 195: Original "A"-tone signal and its discrete cosine transform sampled at 40000 Hz for 1/8 seconds.

The Matrix Problem

Before proceeding too far along, it is important to relate the current exercise to the linear algebra problems considered in the last section. Specifically, this signal reconstruction problem is nothing more than a large underdetermined system of equations. To be more precise, the conversion from the time domain to frequency domain via the DCT can be thought of as a linear transformation

$$\psi \mathbf{c} = \mathbf{f} \tag{18.2.4}$$

where $\mathbf{f}$ is the signal vector in the time domain (plotted in the top panel of Figs. 195 and 196) and $\mathbf{c}$ are the cosine transform coefficients representing the signal in the DCT domain (plotted in the bottom panel of Figs. 195 and 196). The matrix ψ represents the DCT transform itself. The key observation is that most of the coefficients of the vector $\mathbf{c}$ are zero, i.e. it is sparse as clearly demonstrated by the dominance of the two cosine coefficients at $\approx$175 Hz and $\approx$410 Hz. Note that the matrix ψ is of size $n \times n$ while $\mathbf{f}$ and $\mathbf{c}$ are $n \times 1$ vectors. For the example plotted, $n = 5000$.

The choice of basis functions is critical in carrying out the compressed sensing

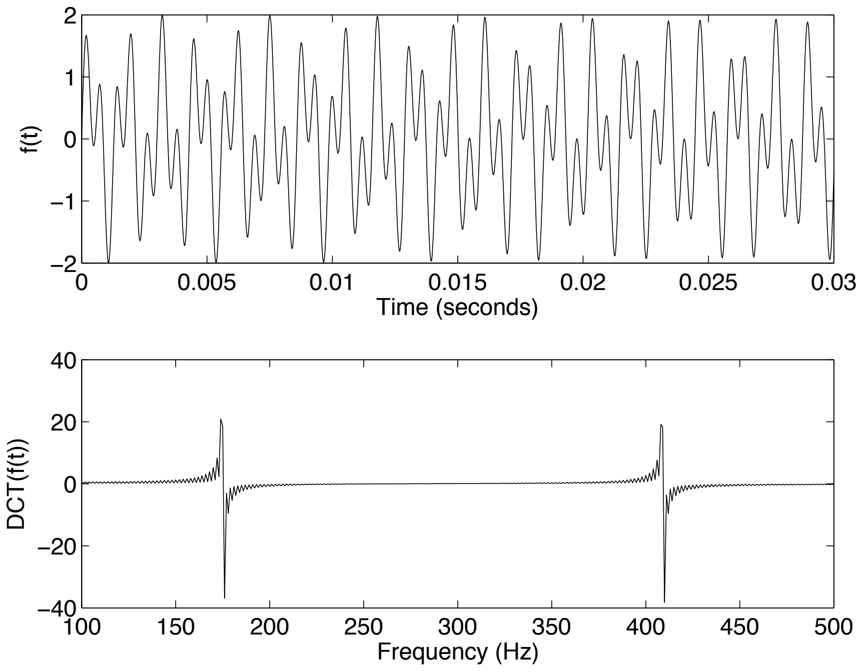


Figure 196: Zoom in of the original “A”-tone signal and its discrete cosine transform sampled at 40000 Hz for 1/8 seconds.

protocol. In particular, the signal must be sparse in the chosen basis. For the example here of a cosine basis, the signal is clearly sparse, allowing us to accurately reconstruct the signal using sparse sampling. The idea is to now sample the signal randomly (and sparsely) so that

$$\mathbf{b} = \phi \mathbf{f} \tag{18.2.5}$$

where $\mathbf{b}$ is a few (m) random samples of the original signal $\mathbf{f}$ (ideally $m \ll n$). Thus ϕ is a subset of randomly permuted rows of the identity operator. More complicated sampling can be performed, but this is a simple example that will illustrate all the key features. Note here that $\mathbf{b}$ is an $m \times 1$ vector while the matrix ϕ is of size $m \times n$.

Approximate signal reconstruction can then be performed by solving the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{18.2.6}$$

where $\mathbf{b}$ is an $m \times 1$ vector, $\mathbf{x}$ is $n \times 1$ vector and

$$\mathbf{A} = \phi\psi \tag{18.2.7}$$

is a matrix of size $m \times n$. Here the $\mathbf{x}$ is the sparse approximation to the full DCT coefficient vector. Thus for $m \ll n$, the resulting linear algebra problem is highly underdetermined as in the last section. The idea is then to solve the underdetermined system using an appropriate norm constraint that best reconstructs the original signal. As already demonstrated, the L^1 norm promotes sparsity and is highly appropriate given sparsity already demonstrated. The signal reconstruction is performed by using

$$\mathbf{f} \approx \psi \mathbf{x} . \tag{18.2.8}$$

If the original signal had exactly m non-zero coefficients, the reconstruction could be made exact.

Signal Reconstruction

To begin the reconstruction process, the signal must first be randomly sampled. In this example, the original signal $\mathbf{f}$ with $n = 5000$ points will be sampled at 10% so that the vector $\mathbf{b}$ above will have $m = 500$ points. The following code randomly rearranges the integers 1 to 5000 (**randintrlv**) so that the vector **perm** retreaves 500 random sample points.

```
m=500;
r1 = randintrlv([1:n],793);
perm=r1(1:m);

f2=f(perm);
t2=t(perm);
```

In this example, the resulting vectors **t2** and **f2** are the 500 random point locations (out of the 5000 original). The $m \times n$ matrix $\mathbf{A}$ is then constructed by constructing $\mathbf{A} = \phi\psi$.

```
D=dct(eye(n,n));
A=D(perm,:);
```

Recall that the matrix ψ was the DCT transform while the matrix ϕ was the permutation matrix of the identity yielding the sampling points for **f2**.

Although the L^1 norm is of primary importance, the underdetermined system will be solved in three ways: with the pseudo-inverse, the backslash and with L^1 optimization. It was already illustrated in the last section that the backslash promoted sparsity for such systems, but as will be illustrated, all sparsity is not the same! The following commands generate all three solutions for the underdetermined system.

```
x=pinv(A)*f2';
```

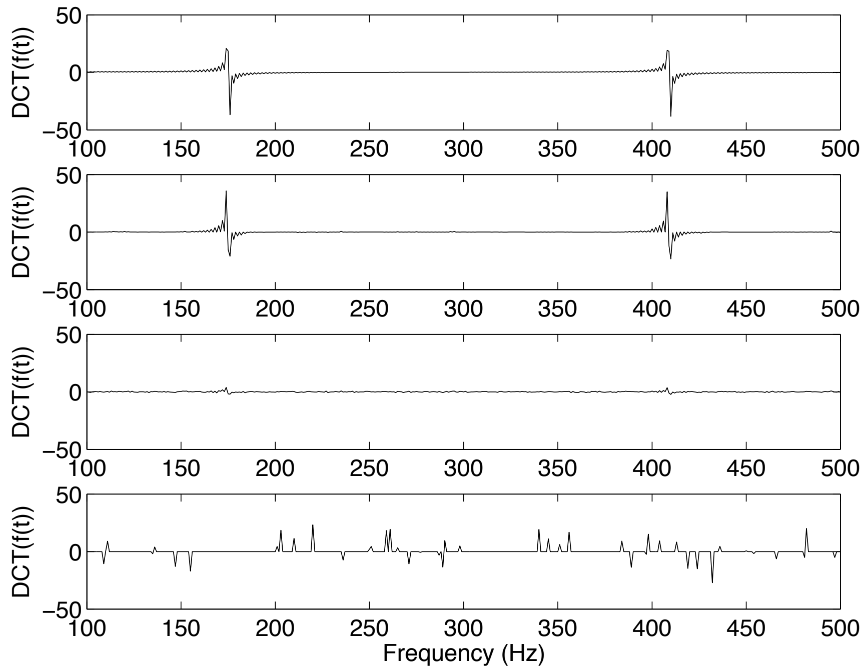


Figure 197: Original signal in the DCT domain (top panel) and the reconstructions using L^1 optimization (second panel), the pseudo-inverse (third panel) and the backslash (fourth panel) respectively. Note that the L^1 optimization does an exceptional job at reconstructing the spectral content with only 10% sampling. In contrast, both the pseudo-inverse and backslash fail miserably at the task.

```
x2=A\f2';
cvx_begin;
     variable x3(n);
     minimize( norm(x3,1) );
     subject to
     A*x3 == f2';
cvx_end;
```

The above code generates three vectors approximating the DCT coefficients, i.e. these solutions are used for the reconstruction (18.2.8). Figure 197 illustrates the results of the reconstructed signal in the DCT domain over the range of 100-500 Hz. The top panel illustrates the original signal in the DCT domain while the next three panels represent the approximation to the DCT coefficients using the L^1 norm optimization, the pseudo-inverse and the backslash respectively. As

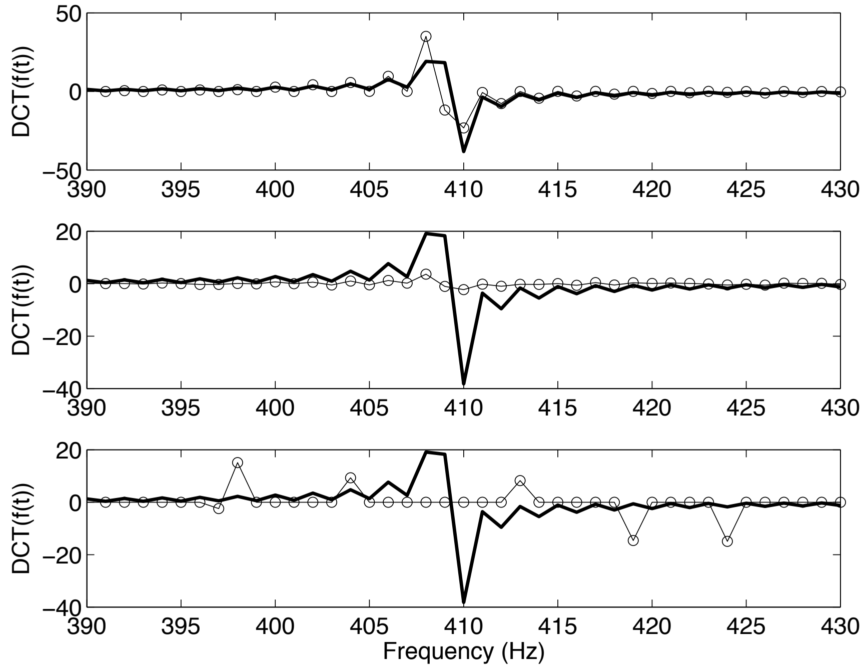


Figure 198: Zoom in of the DCT domain reconstructions using L^1 optimization (top panel), the pseudo-inverse (middle panel) and the backslash (bottom panel) respectively. Here a direct comparison is given to the original signal (bold line). Although both the L^1 optimization and backslash produce sparse results, it is clear the backslash produces a horrific reconstruction of the signal.

expected, both the L^1 and backslash methods produce sparse representations. However, the backslash produces a result that looks nothing like the original DCT signal. In contrast, the pseudo-inverse does not produce a sparse result. Indeed, the spectral content is quite full and only the slightest indication of the importance of the cosine coefficients at ≈175 Hz and ≈410 Hz is given.

To more clearly see the results of the cosine coefficient reconstruction, Fig. 198 zooms in near the ≈410 Hz peak of the spectrum. The top panel shows the excellent reconstruction yielded by the L^1 optimization routine. The pseudo-inverse (middle panel) and backslash fail to capture any of the key features of the spectrum. This example shows the almost "magical" ability of the L^1 optimization to reconstruct the original signal in the cosine domain.

It remains to consider the signal reconstruction in the time domain. To perform this task, the DCT domain data must be transformed to the time domain via the discrete cosine transform:

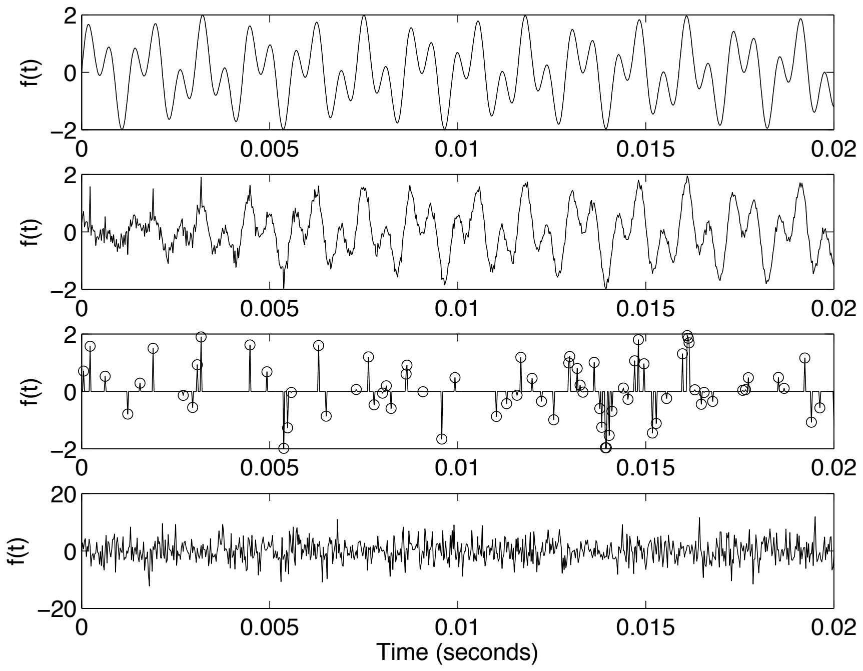


Figure 199: Original time-domain signal (top panel) and the reconstructions using L^1 optimization (second panel), the pseudo-inverse (third panel) and the backslash (fourth panel) respectively. Note that the L^1 optimization does an exceptional job at reconstructing the signal with only 10% sampling. In contrast, both the pseudo-inverse and backslash fail. Note in that in the pseudo-inverse reconstruction (third panel), the original sparse sampling points are included, demonstrating that the method does well in keeping the L^2 error in check for these points. However, the L^2 methods do poorly elsewhere.

```
sig1=dct(x);
sig2=dct(x2);
sig3=dct(x3);
```

As before, the L^1 optimization will be compared to the standard L^2 schemes of the pseudo-inverse and backslash. Figure 199 illustrates the original signal (top panel) along with the L^1 optimization, pseudo-inverse and backslash methods respectively. Even at 10% sampling, the L^1 does a very nice job in reconstructing the signal. In contrast, both L^2 based methods fail severely in the reconstruction. Note, however, that in this case that the pseudo-inverse does promote sparsity in the time domain. Indeed, when the original sparse sampling points are overlaid on the reconstruction, it is clear that the reconstruction does a very

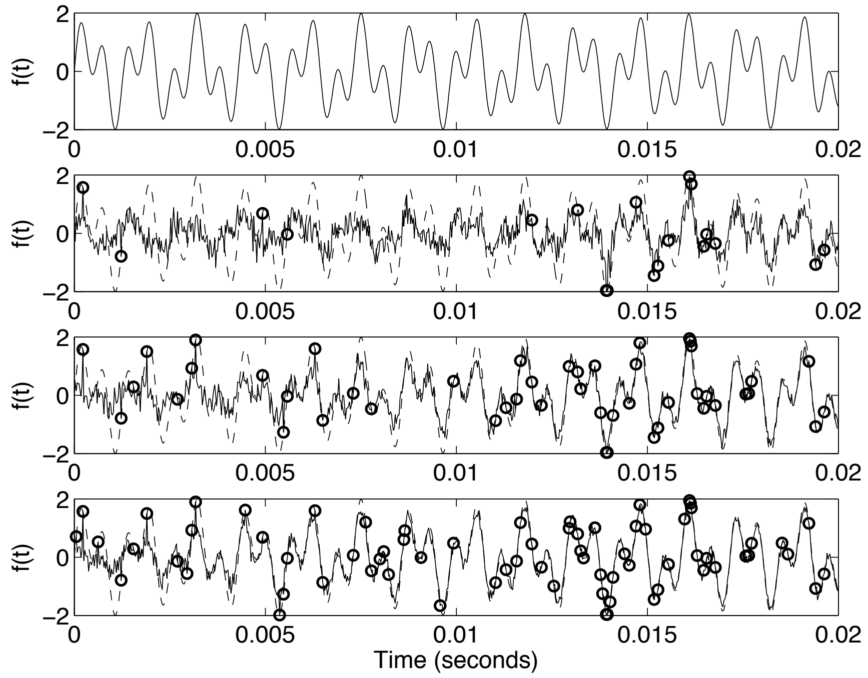


Figure 200: The original signal (top panel) and the reconstruction using $m = 100$ (second panel), 300 (third panel) and 500 (bottom panel) sampling points. In the top panel, the original signal is included for reference. In subsequent panels, the original signal is included as the dotted line. Also included is the sparse sample points (circles).

nice job of minimizing the L^2 error for these points. In contrast, the backslash produces a solution which almost resembles white-noise. However, as with the pseudo-inverse, it considering only the sampling points, the L^2 error is quite low.

Finally, Fig. 200 demonstrates the effect of sampling on the signal reconstruction. In what was considered in the previous plots, the sampling was $m = 500$ points out of a possible $n = 5000$, thus 10% sparse sampling was sufficient to approximate the signal. The sampling is lowered to see the deterioration of the signal reconstructions. In particular, Fig. 200 demonstrate the signal reconstruction using $m = 100$ (second panel), 300 (third panel) and 500 (bottom panel) sampling points. In the top panel, the original signal is included for reference. In subsequent panels, the original signal is included as the dotted line. Also included is the sparse sample points (circles). This provides a naked eye measure of the effectiveness of the compressed sensing algorithm and L^1 optimization method. At $m = 500$ (bottom panel), the reconstruction is quite faithful to the original signal. As m is lowered, the signal recovery becomes worse. But remarkably at even 2% sampling ($m = 100$), many of the key features are still visible in the signal reconstruction. This is especially remarkable given the amazing sparsity in the sampling. For instance, between $\approx 0.006 - 0.012$ Hz there are no sampling points, yet the compressed sensing scheme still picks up the key oscillatory features in this range. This is in direct contrast to the thinking and intuition established by Nyquist. Indeed, Nyquist (or Nyquist-Shannon sampling theory) states that to completely recover a signal, one must sample at twice the rate of the highest frequency of the signal. This is the gold standard of the information theory community. Here, however, the compressed sensing algorithm seems to be Nyquist. The apparent contradiction is overcome by one simple idea: sparsity. Specifically, Nyquist-Shannon theory assumes in its formulation that the signal is dense in the frequency domain. Thus it is certainly true that an accurate reconstruction can only be obtained by sampling at twice the highest frequency. Here however, compressed sensing worked on a fundamental assumption about the sparsity of the signal. This is a truly amazing idea that is having revolutionary impact across many disciplines of science [51, 52, 53].

18.3 Data (Image) Reconstruction from Sparse Sampling

Compressed sensing can now be brought to the arena of image reconstruction. Much like the signal recovery problem, the idea is to exploit the sparse nature of images in order to reconstruct them using greatly reduced sampling. These concepts also make natural connection to image compression algorithms. What is of particular importance is choosing a basis representation in which images are, in fact, sparse.

To illustrate the above concepts more clearly, we can consider a digital image with 600×800 pixels (rows × columns). Figure 201(a) shows the original image. The following MATLAB code imports the original JPEG picture and plots the results. It also constructs the matrix **Abw** which is the picture in matrix form.

```
A=imread('photo','jpeg');
Abw2=rgb2gray(A);
Abw=double(Abw2);
[nx,ny]=size(Abw);
figure(1), subplot(2,2,1), imshow(Abw2)
```

The image is well defined and of high resolution. Indeed, it contains $600 \times 800 = 480,000$ pixels encoding the image. However, the image can actually be encoded with far less data as illustrated with the JPEG wavelet compression algorithm (See Figs. 139-141). It also has been shown that the image is highly compressed (or sparse) in the Fourier basis as well (See Fig. 142). In what follows, the compressed sensing algorithm will be applied using the Fourier mode basis.

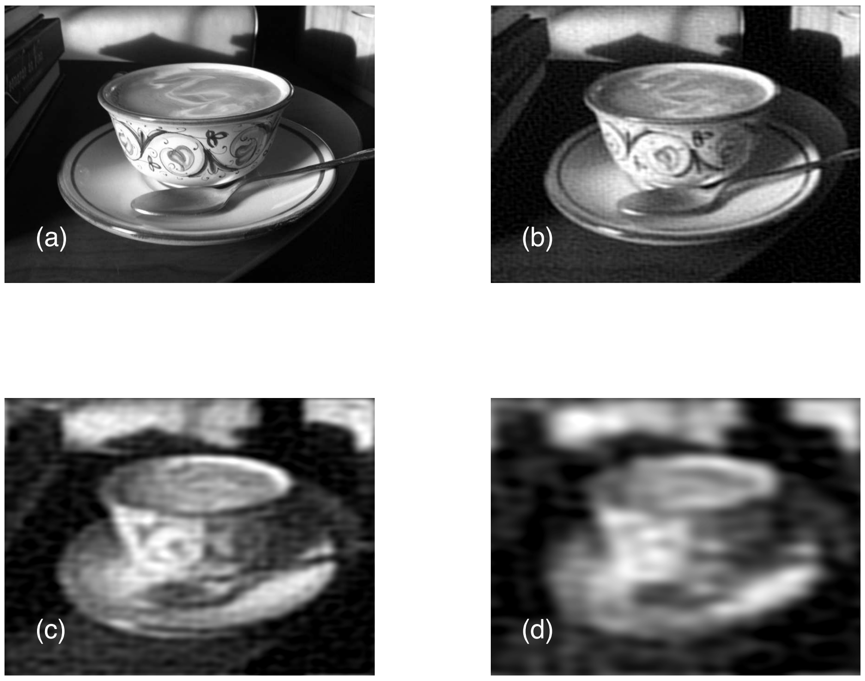


Figure 201: Original image (800×600 pixels) (a) along with the reconstructed image using approximately (b) 1.4%, (c) 0.2% or (d) 0.06% of the information encoded in the Fourier coefficients. At 0.06%, the image reconstruction starts to fail. This reconstruction process illustrates that images are sparse in the Fourier representation.

To demonstrate the nearly optimal encoding of the picture in the Fourier domain, the original photo is subjected to a two-dimensional Fourier transformation. The Fourier coefficients are reshaped into a single vector where the dominant portion of the signal is plotted on both a regular axis and semilog axis:

```
At=fftshift(fft2(Abw));
figure(2)
subplot(2,1,1)
plot(reshape(abs(At),nx*ny,1),'k','Linewidth',[2])
set(gca,'Xlim',[2.3*10^5 2.5*10^5],'Ylim',[-0.5*10^7 5*10^7])

subplot(2,1,2)
semilogy(reshape(abs(At),nx*ny,1),'k','Linewidth',[2])
set(gca,'Xlim',[2.3*10^5 2.5*10^5],'Ylim',[-0.5*10^7 5*10^7])
```

The **fft2** command performs a two-dimensional transform and returns the Fourier coefficient matrix **At** which is reshaped into an $nx \times ny$ length vector.

Figure 202 shows the Fourier coefficients over the dominant range of Fourier mode coefficients. In addition to the regular y-axis scale, a semilog plot is presented showing the low-amplitude (non-zero) energy in each of the Fourier mode coefficients. One can clearly see that many of the coefficients that make up the image are nearly zero. Due to numerical round-off, for instance, none of the coefficients will actually be zero.

Image compression follows directly from this Fourier analysis. Specifically, one can simply choose a threshold and directly set all wavelet coefficients below this chosen threshold to be identically zero instead of nearly zero. What remains is to demonstrate that the image can be reconstructed using such a strategy. If it works, then one would only need to retain the much smaller (sparse) number of non-zero wavelets in order to store and/or reconstruct the image.

To address the image compression issue, we can apply a threshold rule to the Fourier coefficient vector **At** that is shown in Fig. 202. The following code successively sets a threshold at three different levels (dotted lines in Fig. 202) in order to generate a sparse Forier coefficient vector **At2**. This new sparse vector is then used to encode and reconstruct the image.

```
count_pic=2;
for thresh=[0.005*10^7 0.02*10^7 0.05*10^7];
  At2=reshape(At,nx*ny,1);
  count=0;
  for j=1:length(At2);
     if abs(At2(j)) < thresh
        At2(j)=0;
        count=count+1;
     end
  end
  percent=100-count/length(At2)*100

  Atlow=fftshift(reshape(At2,nx,ny)); Alow=uint8(ifft2(Atlow));
  figure(1), subplot(2,2,count_pic), imshow(Alow);
  count_pic=count_pic+1;
end
```

The results of the compression process are illustrated in Figs. 201(b)-(d) where only 1.4%, 0.2% and 0.06% of the modes are retained respectively. The variable **percent** in the code keeps track of the percentage of nonzero modes in the sparse vector **At2**. Remarkably, the image can be very well reconstructed with only 1.4% of the modes, thus illustrating the sparse nature of image. Such sparsity is ubiquitous for image representation using wavelets or Fourier coefficients. Thus the idea is to take advantage of this fact. *Much like representing dynamics in*

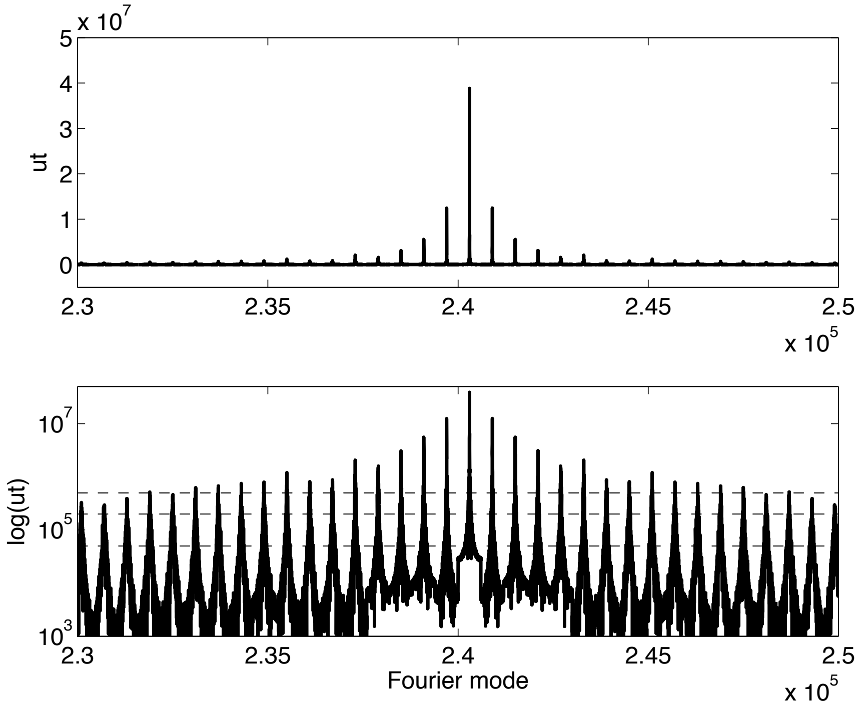


Figure 202: (a) Amplitude of the Fourier mode coefficients (in vector form) around the dominant Fourier modes. Due to the small amplitude of many of the Fourier modes, the amplitudes are also shown on a semilog scale (b). In panel (b), the three threshold lines (dotted) are demonstrated for reconstructing the images in Fig. 201(b)-(d). The percentage of non-zero Fourier coefficients for the three threshold lines is approximately 1.4%, 0.2% and 0.06% respectively. Only the non-zero coefficients are kept for encoding of the image.

a POD basis, the aim is always to take advantage of low-dimsional structure. This is exactly what compressed sensing does!

Compressed Sensing

Compressed sensing aims to take advantage of sparsity in information. In the wavelet or Fourier basis, images are clearly sparse and thus compressed sensing can play a definitive role. Specifically, note what the image compression algorithm does: (i) first the image is taken, (ii) a wavelet/Fourier transform is applied in order to apply a threshold rule, and (iii) the majority of information is promptly discarded. In the example here, 98.6% of the information is directly discarded and a faithful reconstruction can be produced. Overall, it seems quite astonishing to go through all the effort of acquiring the image only to discard 98.6% of what was procured. Could we instead sparse sample the image and faithfully reconstruct the image instead? In particular, of the 480,000 pixels sampled, could we instead randomly sample, let's say 5% (24,000 pixels), of the pixels and still reconstruct the image? Knowing that the image is sparse allows us to do just that. Specifically, we already know that the L^1 optimization routines already considered allows for a faithful reconstruction of sparse data.

To make the problem of a more manageable size, and to simply illustrate the process of compressed sensing, a greatly reduced image resolution will be considered since the processing time for the 800×600 image is significant. Thus the image if first resized to 100×75:

```
B=imresize(A,[75 100]);

Abw2=rgb2gray(B);
Abw=double(Abw2);
[ny,nx]=size(Abw)
```

At this size, the image is not nearly as sparse as the high-resolution picture, thus the compressed sensing algorithm is not as ideal as it would be in a more realistically applied problem. Regardless, the algorithm developed here is generic and allows for a signal reconstruction using sparse information.

The algorithm begins by first deciding how many samples of the image to take. Thus much like the signal problem of the last section, a random number of pixels will be sampled and an attempt will be made to completely reconstruct the image. The following code samples the image in 2500 randomly chosen pixel locations. The matrix **Atest2** shows the sample mask. At 2500 random points, only 1/3 of the image is sampled for the reconstruction.

```
k=2500;  % number of sparse samples
Atest2=zeros(ny,nx);
r1 = randintrlv([1:nx*ny],793);
r1k= r1(1:k);
for j=1:k
   Atest2(r1k(j))=-1;
end
imshow(Atest2), caxis([-1 0])
```

The matrix **Atest2** is what is illustrated in Fig. 203. The black dots represent the randomly chosen sampling points on the original image.

As was already stated, by using a low resolution image with only 100×75 pixels, the image is no longer remarkably sparse as with the 800×600 original. Thus more Fourier coefficients are required for reconstructing the image. The following code works much as the signal construction problem of the last section.

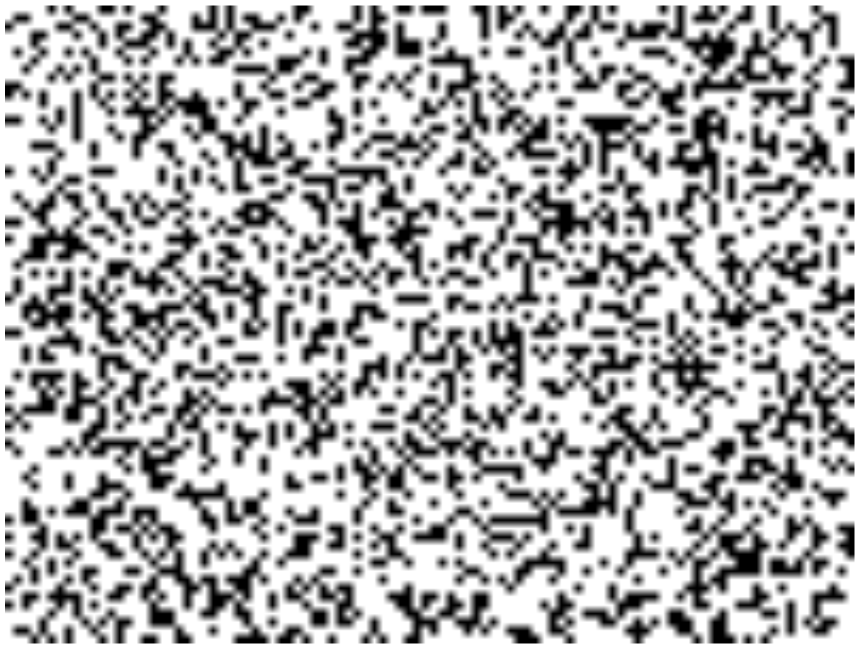


Figure 203: Typical realization of a random sampling of the image with 2500 random pixel points (black). The white pixels, which comprise 2/3 of the pixels, are not sampled. It should be noted that for images, the sampling mask can be extremely important in the image reconstruction [54].

Indeed, the image will now be represented in the Fourier domain using the two-dimensional discreet transform. Thus the randomly chosen sampling points, which are treated as delta functions, are responsible for producing the matrix leading to the highly undetermined system.

```
Atest=zeros(ny,nx);
for j=1:k
   Atest(r1k(j))=1;
   Adel=reshape(idct2(Atest),nx*ny,1);
   Adelta(j,:)=Adel;
   Atest(r1k(j))=0;
end
b=Abw(r1k).';
```

This code produces two key quantities: the matrix **Adelta** (which is 2500×7500) and the vector **b** (which is 2500×1). The objective is then to solve the underdetermined system $\mathbf{Adelta}\,\mathbf{y} = \mathbf{b}$, where the vector $\mathbf{y}$ (which is 7500×1)

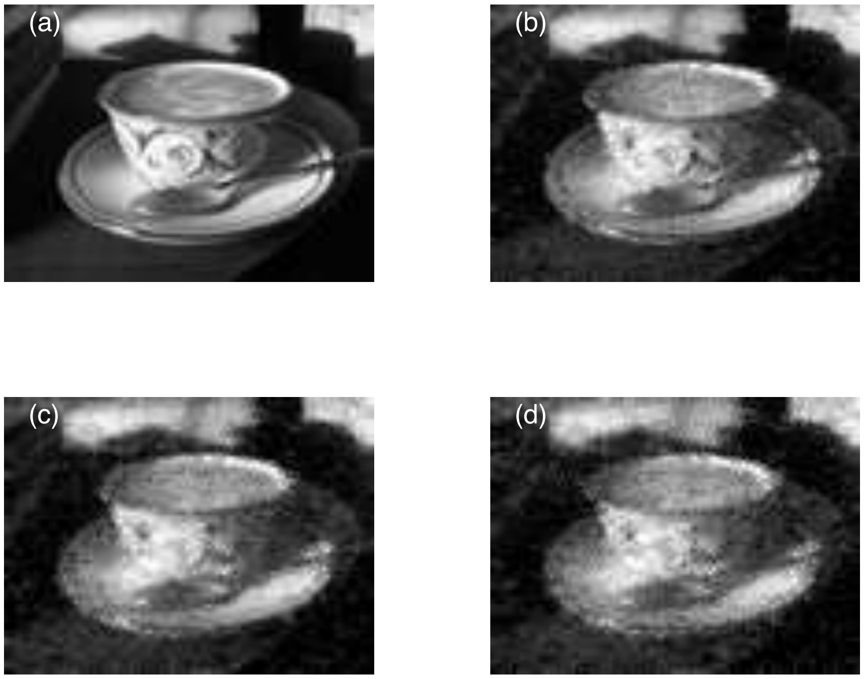


Figure 204: Compressed sensing recovery of the original image (a) which has 7500 pixels. Panels (b)-(d) demonstrate the image recovery using $k = 4000, 3000$ and 2500 random sample points of the image respectively.

is the sparse recover of the discrete cosine coefficients. The sparse recovery of this vector is done using the L^1 optimization.

```
 n=nx*ny;
 cvx_begin;
    variable y(n);
    minimize( norm(y,1) );
    subject to
       Adelta*y == b;
 cvx_end;

 Alow=uint8((dct2(reshape(y,ny,nx))));
figure(1), subplot(2,2,2), imshow(Alow)
```

As before, convex optimization is now performed to recover the discrete cosine coefficients. These are in turn used to reconstruct the original image.

Figure 204 demonstrates the image recovery using the compressed sensing format. Ultimately, the image recovery is not stellar given the sizeable number

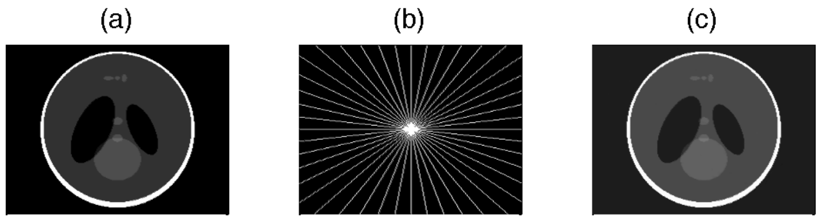


Figure 205: Compressed sensing recovery of the original image (a) which has 256×256 pixels. The sampling mask in the Fourier domain (22 diagonals) is shown in (b) and the reconstructed image sampled sparsely along the Fourier diagonals is shown in (c).

of pixels used. However, recall that the signal considered is already of low-resolution, thus loosing some of its sparsity. Furthermore, the random sampling is not optimal for recovering the image. Specifically, it is often desirable to apply a sampling mask in order to enhance the compressed sensing algorithm [54]. Regardless, the above code demonstrates the key features required in the compressed sensing algorithm.

Sampling Locations

The above example can be greatly improved by sampling with an appropriate sampling mask. As is the case with most images expressed in the Fourier basis, the dominant Fourier modes correspond to low frequency content. Indeed, most high frequency content can be easily neglected without much loss to the image quality. Thus when sampling, it would be highly advantageous to sample in the frequency domain with heavier sampling near the zero wavenumber. Candes and Romberg [54] have built such a sampling mask in their L^1-magic MATLAB routines. Specifically, they sample along diagonals in the frequency domain, thus allowing for dense sampling near the zero wavenumbers. Figure 205 demonstrates an efficient sampling of a particular image along diagonal lines in the Fourier domain. Here the construction is done by sampling only a small fraction of the total pixels.

Wavelet Basis

Although we demonstrated the compressed sensing algorithm with standard Fourier techniques, largely because most people have easy access to Fourier transforms via MATLAB, the compressive sensing is perhaps even more effective using wavelets. However, when using wavelets, the sampling must be performed in a very different way. Unlike the sampling using Fourier modes, which takes a delta function in the image space in order to create global spanning of the Fourier space, wavelets require something different since they are highly localized both in space and time. In particular, global and noisy sampling modes are required to perform the task in the most efficient manner [51, 56]. It is in this context that the compressive sensing can make serious performance gains in relation to what is demonstrated in the current example.

The One Pixel Camera

One can imagine taking this idea of compressed sensing further. Indeed, various research groups in image processing coupled with compressive sensing have proposed the idea of the one pixel camera [55]. The single pixel camera compresses the image data via its hardware before the pixels are recorded. As a result, it's able to capture an image with only thousands of pieces of information rather than millions.

The key benefit of such a camera is that it needs much less information to assemble an image. Massive CCD arrays collect millions of pixels worth of data, which are typically compressed to keep file sizes manageable. It's an approach the Rice researchers describe as "acquire first, ask questions later" [55]. Many pictures, however, have portions that contain relatively little information, such as a clear blue sky or a snowy white background. Conventional cameras record every pixel and later eliminate redundancy with compression algorithms. The idea is to eliminate the redundancy ahead of time.

Currently, the single pixel camera is simply not competitive with standard digital cameras. Mostly because of the processing (L^1 optimization) that must be performed in acquiring the image. Specifically, it takes about fifteen minutes to record a reasonable photo with the single pixel. However, one potential payoff of this sort of compressed sensing camera is that it may make conventional digital cameras much better. If a single pixel can do the job of an array of pixels, then you could potentially get each of the pixels in a megapixel camera to do extra duty as well. Effectively, you can multiply the resolution of your camera with the techniques developed in a single pixel camera.

19 Dimensionality Reduction for Partial Differential Equations

One of the most important uses of computational methods and techniques is in its applications to modeling of physical, engineering, or biological systems that demonstrate spatio-temporal dynamics and patterns. For instance, the field of fluid dynamics fundamentally revolves around being able to predict the time and space dynamics of a fluid through some potentially complicated geometry. Understanding the interplay of spatial patterns in time indeed is often the central focus of the field of partial differential equations. In systems

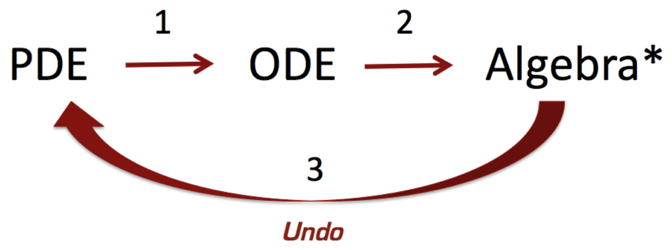


Figure 206: Basic solution technique for solving a partial differential equation. The first step is to reduce it to an appropriate set of ordinary differential equations and then algebra. If the reduction can be undone, then a complete analytic solution can be produced. The starred quantity, i.e. algebra, is the goal of any PDE or ODE reduction.

where the underlying spatial patterns exhibit low-dimensional (pattern forming) dynamics, then the application of the proper orthogonal decomposition can play a critical role in predicting the resulting low-dimensional dynamics.

19.1 Modal expansion techniques for PDEs

Partial differential equations (PDEs) are fundamental to the understanding of the underlying physics, biophysics, etc. of complex dynamical systems. In particular, a partial differential equation gives the governing evolution of a system in both time and space. Abstractly, this can be represented by the following PDE system:

$$\mathbf{U}_t = N(\mathbf{U}, \mathbf{U}_x, \mathbf{U}_{xx}, \cdots, x, t) \tag{19.1.9}$$

where $\mathbf{U}$ is a vector of physically relevant quantities and the subscripts t and x denote partial differentiation. The function $N(\cdot)$ captures the space-time dynamics that is specific to the system being considered. Along with this PDE are some prescribed boundary conditions and initial conditions. In general, this can be a complicated, nonlinear function of the quantity $\mathbf{U}$, its derivatives and both space and time.

In general, there are no techniques for building analytic solutions for (19.1.9). However for specific forms of $N(\cdot)$, such as the case where the function is linear in $\mathbf{U}$, time-indenpendent and constant coefficient, then standard PDE solution techniques may be used to characterize the dynamics analytically. Such solution methods are the core of typical PDE courses at both the graduate and undergraduate level. The specific process for solving PDE systems is illustrated in Fig. 206 where a reduction from a PDE to ODE is first performed and then a reduction from an ODE to algebra is enacted. Provided such reductions can be achieved and undone, then an analytic solution is produced.

To be more specific about the solution method concept in Fig. 206, a number of standard solution techniques can be discussed. The first to be highlighted is the technique of self-similarity reduction. This method simply attempts to extract a fundamental relationship between time and space by making a change of variable to, for instance, a new variable such as

$$\xi = x^\beta t^\alpha \tag{19.1.10}$$

where β and α are determined by making the change of variable in (19.1.9). This effectively performs the first step in the reduction process by reducing the PDE to an ODE in the variable ξ alone. A second, and perhaps the most common introductory technique, is the method of separation of variables where the assumption

$$\mathbf{U}(x,t) = \mathbf{F}(x)\mathbf{G}(t) \tag{19.1.11}$$

is made. Thus effectively separating time and space in the solution. When applied to (19.1.9) with $N(\cdot)$ being linear in $\mathbf{U}$, two ODEs result: one for the time dynamics $\mathbf{G}(t)$ and one for the spatial dynamics $\mathbf{F}(x)$. If the function $N(\cdot)$ is nonlinear in $\mathbf{U}$, then separation cannot be achieved.

As a specific example of the application of the method of separation, we consider here the eigenfunction expansion technique. Strictly speaking, to make analytic progress with the method of separation of variables, one must require that separation occurs. However, no such requirement needs to be made if we are simply using the method as the basis of a computational technique. In particular, the eigenfunction expansion techniques assumes a solution of the form

$$u(x,t) = \sum_{n=1}^{\infty} a_n(t)\phi_n(x) \tag{19.1.12}$$

where the $\phi_n(x)$ are an orthogonal set of eigenfunctions and we have assumed that the PDE is now described by a scalar quantity $u(x,t)$. The $\phi_n(x)$ can be any orthogonal set of functions such that

$$(\phi_j(x), \phi_k(x)) = \delta_{jk} = \begin{cases} 1 & j = k \\ 0 & j \neq q \end{cases} \tag{19.1.13}$$

where δ_{jk} is the Dirac function and the notation $(\phi_j, \phi_k) = \int \phi_j \phi_k^* dx$ gives the inner product. For a given physical problem, one may be motivated to use a specified set of eigenfunctions such as those special functions that arise for specific geometries or symmetries. More generally, for computational purposes, it is desired to use a set of eigenfunctions that produce accurate and rapid evaluation of the solutions of (19.1.9). Two eigenfunctions immediately come to mind: the Fourier modes and Chebyshev polynomials. This is largely in part due to their spectral accuracy properties and tremendous speed advantages such as $O(N \log N)$ speed in projection to the spectral basis.

To give a specific demonstration of this technique, consider the nonlinear Schrödinger (NLS) equation

$$iu_t + \frac{1}{2}u_{xx} + |u|^2 u = 0 \tag{19.1.14}$$

with the boundary conditions $u \to 0$ as $x \to \pm\infty$. If not for the nonlinear term, this equation could be solved easily in closed form. However, the nonlinearity mixes the eigenfunction components in the expansion (19.1.12) making a simple analytic solution not possible.

To solve the NLS computationally, a Fourier mode expansion is used. Thus use can be made of the standard fast Fourier transform. The following code formulates the PDE solution as an eigenfunction expansion technique (19.1.12) of the NLS (19.1.14). The first step in the process is to define an appropriate spatial and time domain for the solution along with the Fourier frequencies present in the system. The following code produces both the time and space domain of interest:

```
% space
L=40; n=512;
x2=linspace(-L/2,L/2,n+1); x=x2(1:n);
k=(2*pi/L)*[0:n/2-1 -n/2:-1].';
% time
t=0:0.1:10;
```

It now remains to consider a specific spatial configuration for the initial condition. For the NLS, there are a set of special initial conditions called solitons where the initial conditions are given by

$$u(x,0) = N\text{sech}(x) \tag{19.1.15}$$

where N is an integer. We will consider the soliton dynamics with $N = 1$ and $N = 2$ respectively. In order to do so, the initial condition is projected onto the Fourier modes with the fast Fourier transform. Rewriting (19.1.14) in the Fourier domain, i.e. Fourier transforming, gives the set of differential equations

$$\hat{u}_t = -\frac{i}{2}k^2\hat{u} + i\widehat{|u|^2 u} \tag{19.1.16}$$

where the Fourier mode mixing occurs due to the nonlinear mixing in the cubic term. This gives the system of differential equations to be solved for in order to evaluate the NLS behavior. The following code solves the set of differential equations in the Fourier domain.

```
% initial conditions
N=1;
```

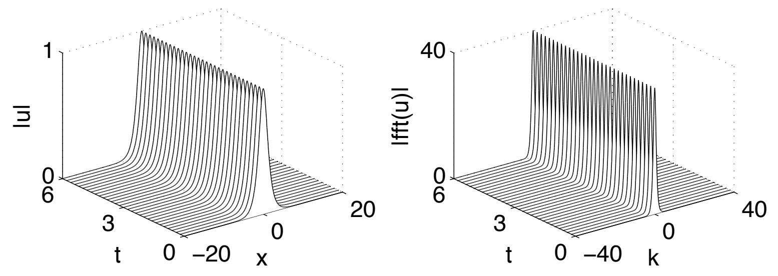


Figure 207: Evolution of the $N = 1$ soliton. Although it appears to be a very simple evolution, the phase is evolving in a nonlinear fashion and approximately 50 Fourier modes are required to model accurately this behavior.

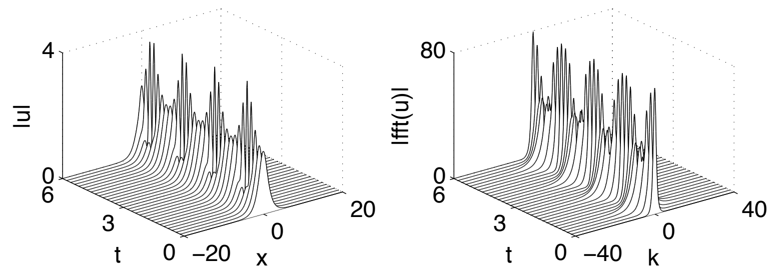


Figure 208: Evolution of the $N = 2$ soliton. Here a periodic dynamics is observed and approximately 200 Fourier modes are required to model accurately the behavior.

```
u=N*sech(x);
ut=fft(u);
[t,utsol]=ode45('nls_rhs',t,ut,[],k);
for j=1:length(t)
    usol(j,:)=ifft(utsol(j,:));  % bring back to space
end

subplot(2,2,1), waterfall(x,t,abs(usol)), colormap([0 0 0])
subplot(2,2,2), waterfall(fftshift(k),t,abs(fftshift(utsol)))
```

In the solution code, the following right hand side function **nls_rhs.m** is called that effectively produces the differential equation (19.1.16):

```
function rhs=nls_rhs(t,ut,dummy,k)
u=ifft(ut);
rhs=-(i/2)*(k.^2).*ut+i*fft( (abs(u).^2).*u );
```

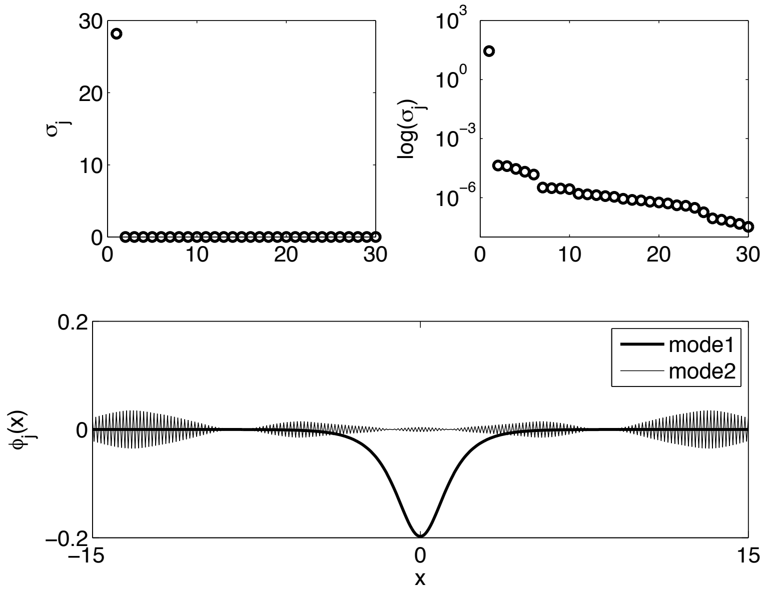


Figure 209: Projection of the $N = 1$ evolution to POD modes. The top two figures are the singular values σ_j of the evolution demonstrated in (207) on both a regular scale and log scale. This demonstrates that the $N = 1$ soliton dynamics is primarily a single mode dynamics. The first two modes are shown in the bottom panel. Indeed, the second mode is meaningless and is generated from noise and numerical round-off.

This gives a complete code that produces both the time-space evolution and the time-frequency evolution.

The dynamics of the $N = 1$ and $N = 2$ solitons are demonstrated in Figs. 207 and 208 respectively. During evolution, the $N = 1$ soliton only undergoes phase changes while its amplitude remains stationary. In contrast, the $N = 2$ soliton undergoes periodic oscillations. In both cases, a large number of Fourier modes, about 50 and 200 respectively, are required to model the simple behaviors illustrated.

The potentially obvious question to ask in light of our dimensionality reduction thinking is this: is the soliton dynamics really a 50 or 200 degree-of-freedom system as implied by the Fourier mode solution technique. The answer is no. Indeed, with the appropriate basis, i.e. the POD modes generated from the SVD, it can be shown that the dynamics is a simple reduction to 1 or 2 modes respectively. Indeed, it can easily be shown that the $N = 1$ and $N = 2$ are truly low dimensional by computing the singular value decomposition of the evolutions shown in Figs. 207 and 208 respectively. Indeed, just as in the example of

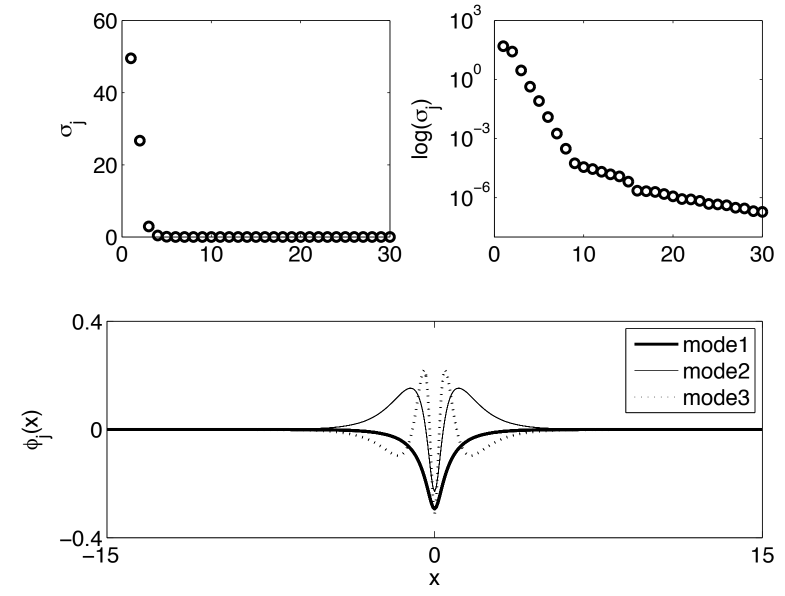


Figure 210: Projection of the $N = 2$ evolution to POD modes. The top two figures are the singular values σ_j of the evolution demonstrated in (208) on both a regular scale and log scale. This demonstrates that the $N = 2$ soliton dynamics is primarily a two mode dynamics as these two modes contain approximately 95% of the evolution energy. The first three modes are shown in the bottom panel.

Sec. 4.5, the simulations can be aligned in a matrix **A** where the columns are different segments of time and the rows are the spatial locations.

Figures 209 and 210 demonstrate the low-dimensional nature explicitly by computing the singular values of the numerical simulations along with the modes to be used in our new eigenfunction expansion. What is clear is that for both of these cases, the dynamics is truly low dimensional with the $N = 1$ soliton being modeled exceptionally well by a single POD mode while the $N = 2$ dynamics is modeled quite well with two POD modes. Thus in performing the expansion (19.1.12), the modes chosen should be the POD modes generated from the simulations themselves. In the next section, we will derive the dynamics of the modal interaction for these two cases and show that quite a bit of analytic progress can then be made within the POD framework.

19.2 PDE dynamics in the right (best) basis

The last section demonstrated the $N = 1$ and $N = 2$ soliton dynamics of the NLS equation (19.1.14). In particular, Figs. 207 and 208 showed the evolution dynamics in the Fourier mode basis that was used in the computation of the evolution. The primary motivation in performing a POD reduction of the dynamics through (19.1.12) is the observation in Figs. 209 and 210 that both dynamics are truly low-dimensional with only one or two modes playing a dominant role respectively.

$N = 1$ **Soliton Reduction**

Two take advantage of the low dimensional structure, we first consider the $N = 1$ soliton dynamics. Figure 207 shows that a single mode in the SVD dominates the dynamics. This is the first column of the **U** matrix (see Sec. 4.5). Thus the dynamics is recast in a single mode so that

$$u(x,t) = a(t)\phi(x) \tag{19.2.17}$$

where now there is now sum in (19.1.12) since there is only a single mode $\phi(x)$. Plugging in this equation into the NLS equation (19.1.14) yields the following:

$$ia_t\phi + \frac{1}{2}a\phi_{xx} + |a|^2 a|\phi|^2\phi = 0\,. \tag{19.2.18}$$

The inner product is now taken with respect to ϕ which gives

$$ia_t + \frac{\alpha}{2}a + \beta|a|^2 a = 0 \tag{19.2.19}$$

where

$$\alpha = \frac{(\phi_{xx}, \phi)}{(\phi, \phi)} \tag{19.2.20a}$$

$$\beta = \frac{(|\phi|^2\phi, \phi)}{(\phi, \phi)} \tag{19.2.20b}$$

and the inner product is defined as integration over the computational interval so that $(\phi, \psi) = \int \phi\psi^* dx$. Note that the integration is over the computational domain and the $*$ denotes the complex conjugate.

The differential equation for $a(t)$ given by (19.2.19) can be solved explicitly to yield

$$a(t) = a(0)\exp\left[i\frac{\alpha}{2}t + \beta|a(0)|^2 t\right] \tag{19.2.21}$$

where $a(0)$ is the initial value condition for $a(t)$. To find the initial condition, recall that

$$u(x,0) = \text{sech}(x) = a(0)\phi(x)\,. \tag{19.2.22}$$

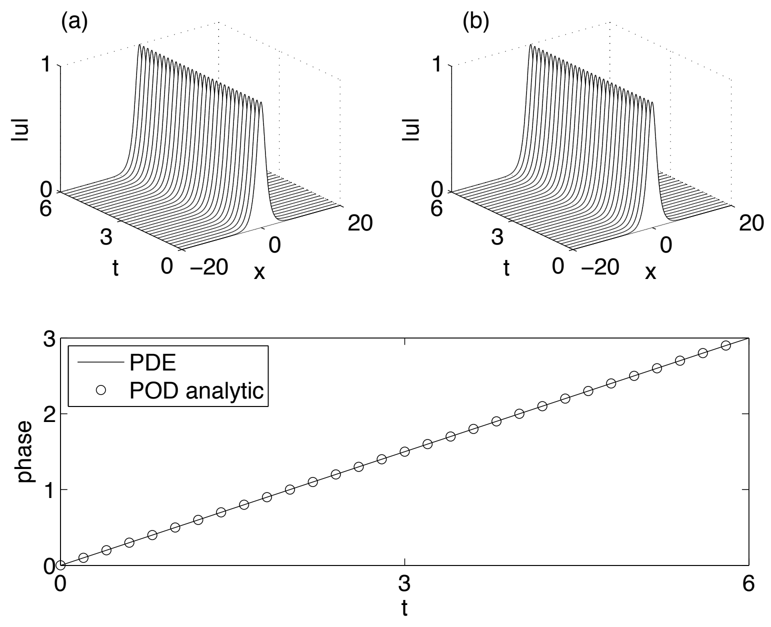


Figure 211: Comparison of (a) full PDE dynamics and (b) one-mode POD dynamics. In the bottom figure, the phase of the pulse evolution at $x = 0$ is plotted for the full PDE simulations and the one-mode analytic formula (circles) given in (19.2.24).

Taking the inner product with respect to $\phi(x)$ gives

$$a(0) = \frac{(\text{sech}(x), \phi)}{(\phi, \phi)}\,. \tag{19.2.23}$$

Thus the one mode expansion gives the approximate PDE solution

$$u(x,t) = a(0)\exp\left[i\frac{\alpha}{2}t + \beta|a(0)|^2 t\right]\phi(x)\,. \tag{19.2.24}$$

This solution is the low-dimensional POD approximation of the PDE expanded in the best basis possible, i.e. the SVD determined basis.

For the $N = 1$ soliton, the spatial profile remains constant while its phase undergoes a nonlinear rotation. The POD solution (19.2.24) can be solved exactly to give a characterization of this phase rotation. Figure 211 shows both the full PDE dynamics along with its one-mode approximation. In this example, the parameters α, β and $a(0)$ were computed using the following code:

```
% one soliton match
```

```
phi_xx=ifft(-(k.^2).*fft(U(:,1)));
norm=trapz(x,U(:,1).*conj(U(:,1)));
A0=trapz(x,(sech(x).').*conj(U(:,1)))/norm;
alpha=trapz(x,U(:,1).*phi_xx)/norm;
beta=trapz(x,(U(:,1).*conj(U(:,1))).^2)/norm;
```

This suggest that the $N = 1$ behavior is indeed a single mode dynamics. This is also known to be true from other solution methods, but it is a critical observation that the POD method for PDEs reproduces all of the expected dynamics.

$N = 2$ Soliton Reduction

The case of the $N = 2$ soliton is a bit more complicated and interesting. In this case, two modes clearly dominate the behavior of the system. These two modes are the first two columns of the matrix $\mathbf{U}$ and are now used to approximate the dynamics observed in Fig. (208). In this case, the two mode expansion takes the form

$$u(x,t) = a_1(t)\phi_1(x) + a_2(t)\phi_2(x) \tag{19.2.25}$$

where the ϕ_1 and ϕ_2 are simply taken from the first two columns of the $\mathbf{U}$ matrix in the SVD. Inserting this approximation into the governing equation (19.1.14) gives

$$i\left(a_{1t}\phi_1 + a_{2t}\phi_2\right) + \frac{1}{2}\left(a_1\phi_{1xx} + a_2\phi_{2xx}\right) + \left(a_1\phi_1 + a_2\phi_2\right)^2\left(a_1^*\phi_1^* + a_2^*\phi_2^*\right) = 0. \tag{19.2.26}$$

Multiplying out the cubic term gives the equation

$$\begin{aligned} & i\left(a_{1t}\phi_1 + a_{2t}\phi_2\right) + \frac{1}{2}\left(a_1\phi_{1xx} + a_2\phi_{2xx}\right) \\ & \quad + \big(|a_1|^2 a_1|\phi_1|^2\phi_1 + |a_2|^2 a_2|\phi_2|^2\phi_2 + 2|a_1|^2 a_2|\phi_1|^2\phi_2 + 2|a_2|^2 a_1|\phi_2|^2\phi_1 \\ & \quad + a_1^2 a_2^*\phi_1^2\phi_2^* + a_2^2 a_1^*\phi_2^2\phi_1^*\big)\ . \end{aligned} \tag{19.2.27}$$

All that remains is to take the inner product of this equation with respect to both $\phi_1(x)$ and $\phi_2(x)$. Recall that these two modes are orthogonal, thus the resulting 2×2 system of nonlinear equations results.

$$\begin{aligned} & ia_{1t} + \alpha_{11}a_1 + \alpha_{12}a_2 + \left(\beta_{111}|a_1|^2 + 2\beta_{211}|a_2|^2\right)a_1 && (19.2.28a) \\ & \quad + \left(\beta_{121}|a_1|^2 + 2\beta_{221}|a_2|^2\right)a_2 + \sigma_{121}a_1^2a_2^* + \sigma_{211}a_2^2a_1^* = 0 \\ & ia_{2t} + \alpha_{21}a_1 + \alpha_{22}a_2 + \left(\beta_{112}|a_1|^2 + 2\beta_{212}|a_2|^2\right)a_1 && (19.2.28b) \\ & \quad + \left(\beta_{122}|a_1|^2 + 2\beta_{222}|a_2|^2\right)a_2 + \sigma_{122}a_1^2a_2^* + \sigma_{212}a_2^2a_1^* = 0 \end{aligned}$$

where

$$\alpha_{jk} = (\phi_{j\,xx}, \phi_k)/2 \tag{19.2.29a}$$

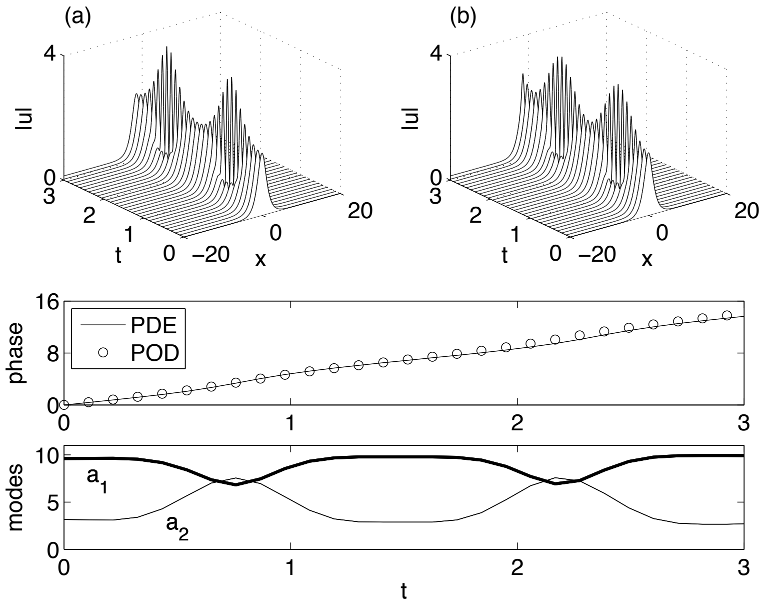


Figure 212: Comparison of (a) full PDE dynamics and (b) two-mode POD dynamics. In the bottom two figures, the phase of the pulse evolution at $x = 0$ (middle panel) is plotted for the full PDE simulations and the two-mode dynamics along with the nonlinear oscillation dynamics of $a_1(t)$ and $a_2(t)$ (bottom panel).

$$\beta_{jkl} = (|\phi_j|^2\phi_k, \phi_l) \tag{19.2.29b}$$

$$\sigma_{jkl} = (\phi_j^2\phi_k^*, \phi_l) \tag{19.2.29c}$$

and the initial values of the two components are given by

$$a_1(0) = \frac{(2\text{sech}(x), \phi_1)}{(\phi_1, \phi_1)} \tag{19.2.30a}$$

$$a_2(0) = \frac{(2\text{sech}(x), \phi_2)}{(\phi_2, \phi_2)} \,. \tag{19.2.30b}$$

This gives a complete description of the two mode dynamics predicted from the SVD analysis.

The 2×2 system (19.2.28) can be easily simulated with any standard numerical integration algorithm (e.g. fourth-order Runge-Kutta). Before computing the dynamics, the inner products given by α_{jk}, β_{jkl} and σ_{jkl} must be calculated along with the initial conditions $a_1(0)$ and $a_2(0)$. Starting from the code of the previous section and the SVD decomposition, these quantities can be computed as follows:

```
  % compute the second derivatives
  phi_1_xx=  ifft( -(k.^2).*fft(U(:,1))  );
  phi_2_xx=  ifft( -(k.^2).*fft(U(:,2))  );

  % compute the norms of the SVD modes
  norm1=trapz(x,U(:,1).*conj(U(:,1)));
  norm2=trapz(x,U(:,2).*conj(U(:,2)));

  % compute the initial conditions
  A0_1=trapz(x,(2.0*sech(x).').*conj(U(:,1)))/norm1;
  A0_2=trapz(x,(2.0*sech(x).').*conj(U(:,2)))/norm2;

  % compute inner products alpha, beta, sigma
  alpha11=trapz(x,conj(U(:,1)).*phi_1_xx)/norm1;
  alpha12=trapz(x,conj(U(:,1)).*phi_2_xx)/norm1;
  beta11_1=trapz(x,(U(:,1).*conj(U(:,1))).^2)/norm1;
  beta22_1=trapz(x,(U(:,2).*(abs(U(:,2)).^2).*conj(U(:,1))))/norm1;
  beta21_1=trapz(x,(U(:,1).*(abs(U(:,2)).^2).*conj(U(:,1))))/norm1;
  beta12_1=trapz(x,(U(:,2).*(abs(U(:,1)).^2).*conj(U(:,1))))/norm1;
  sigma12_1=trapz(x,(conj(U(:,2)).*(U(:,1).^2).*conj(U(:,1))))/norm1;
  sigma21_1=trapz(x,(conj(U(:,1)).*(U(:,2).^2).*conj(U(:,1))))/norm1;

  alpha21=trapz(x,conj(U(:,2)).*phi_1_xx)/norm2;
  alpha22=trapz(x,conj(U(:,2)).*phi_2_xx)/norm2;
  beta11_2=trapz(x,(U(:,1).*(abs(U(:,1)).^2).*conj(U(:,2))))/norm2;
  beta22_2=trapz(x,(U(:,2).*(abs(U(:,2)).^2).*conj(U(:,2))))/norm2;
  beta21_2=trapz(x,(U(:,1).*(abs(U(:,2)).^2).*conj(U(:,2))))/norm2;
  beta12_2=trapz(x,(U(:,2).*(abs(U(:,1)).^2).*conj(U(:,2))))/norm2;
  sigma12_2=trapz(x,(conj(U(:,2)).*(U(:,1).^2).*conj(U(:,2))))/norm2;
  sigma21_2=trapz(x,(conj(U(:,1)).*(U(:,2).^2).*conj(U(:,2))))/norm2;
```

Upon completion of these calculations, ODE45 can then be used to solve the system (19.2.28) and extract the dynamics observed in Fig. 212. Note that the two mode dynamics does a good job in approximating the solution. However, there is a phase drift that occurs in the dynamics that would require higher precision in both taking time slices of the full PDE and more accurate integration of the inner products for the coefficients. Indeed, the most simple trapezoidal rule has been used to compute the inner products and its accuracy is somewhat suspect. Higher-order schemes could certainly help improve the accuracy. Additionally, incorporation of the third mode could also help. In either case, this demonstrates sufficiently how one would in practice use the low dimensional structures for approximating PDE dynamics.

19.3 Global normal forms of bifurcation structures in PDEs

The preceding two sections illustrate quite effectively the role of POD in reducing the dynamics onto the optimal basis functions for producing analytically tractable results. Specifically, given the observation of low-dimensionality, the POD provides the critical method necessary for projecting the complicated system dynamics onto a limited set of physically meaningful modes. This technique can be used to characterize the underlying dynamical behavior, or bifurcation structure, of a given PDE. In what follows, a specific example will be considered where the POD reduction technique is utilized. In this example, a specific physical behavior, which is modified via a bifurcation parameter, is considered. This allows the POD method to reconstruct the *global normal form* for the evolution equations, thus providing a technique similar to normal form reduction for differential equations.

The specific problem considered here to illustrate the construction of a global normal form for PDEs arises in arena of laser physics and the generation of high-intensity, high-energy laser (optical) pulses [57]. Ultrashort lasers have a large variety of applications, ranging from small scale problems such as ocular surgeries and biological imaging to large scale problems such as optical communication systems and nuclear fusion. In the context of telecommunications and broadband sources, the laser is required to robustly produce pulses in the range of 10 femtoseconds to 50 picoseconds. The generation of such short pulses is often referred to as mode-locking [58, 59]. One of the most widely-used mode-locked lasers developed to date is a ring cavity laser with a combination of waveplates and a passive polarizer. The combination of such components act as an effective saturable absorber, providing an intensity discriminating mechanism to shape the pulse. It was first demonstrated experimentally in the early 90's that ultrashort pulse generation and robust laser operation could be achieved in such a laser cavity. Since then, a number of theoretical models have been proposed to characterize the mode-locked pulse evolution and its stability, including the present work which demonstrates that the key phenomenon of multi-pulsing can be completely characterized by a low-dimensional, four-mode interaction generated by a POD expansion.

The master mode-locking equation proposed by H. A. Haus [58, 59], which is the complex Ginzburg-Landau equation with a bandwidth-limited gain, was the first model used to describe the pulse formation and mode-locking dynamics in the ring cavity laser. The Ginzburg-Landau equation was originally developed in the context of particle physics as a model of super-conductivity, and has since been widely used as a proto-typical model for nonlinear wave propagation and pattern-forming systems. A large number of stationary solutions are supported by this equation. These stationary solutions can be categorized as single-pulse, double-pulse, and in general n-pulse solution depending on the strength of the cavity gain. The phenomenon for which a single mode-locked pulse solution splits into two or more pulses is known as multi-pulsing [58, 59], and has been observed in a wide variety of experimental and theoretical configurations. Specifically, when the pulse energy is increased, the spectrum of the pulse experiences a broadening effect. Once the spectrum exceeds the bandwidth limit of the gain, the pulse becomes energetically unfavorable and a double-pulse solution is generated which has a smaller bandwidth.

A number of analytical and numerical tools have been utilized over the past fifteen years to study the mode-locking stability and multi-pulsing transition. One of the earliest theoretical approaches was to consider the energy rate equations derived by Namiki et al. that describe the averaged energy evolution in the laser cavity [60]. Other theoretical approaches involved the linear stability analysis of stationary pulses. These analytical methods were effective in characterizing the stability of stationary pulses, but were unable to describe the complete pulse transition that occurs during multi-pulsing. In addition to the limitation of the analytic tools, the transition region is also difficult to characterize computationally even with the fastest and most accurate algorithms available. Moreover, there is no efficient (algorithmic) way to find bifurcations by direct simulations. Solutions that possess a small basin of attraction are very difficult to find. This difficulty has led to the consideration of reduction techniques that simplify the full governing equation and allow for a deeper insight of the multi-pulsing instability. The POD method provides a generic, analytic tool that can be used efficiently, unlike the direct methods.

To describe the evolution of the electric field in the laser, the dominant physical effects in the laser cavity must be included. Specifically, the leading order dynamics must account for the chromatic dispersion, fiber birefringence, self- and cross-phase modulations for the two orthogonal components of the polarization vector in the fast- and slow-fields, bandwidth-limited gain saturation, cavity attenuation, and the discrete application of the saturable absorber element after each cavity round-trip. Distributing the discrete saturable absorption over the entire cavity and using a low-intensity approximation, one obtains (in dimensionless form) the single-field evolution equation [59]

$$i\frac{\partial u}{\partial z}+\frac{D}{2}\frac{\partial^2 u}{\partial t^2}+|u|^2u-\nu|u|^4u=ig(z)\left(1+\tau\frac{\partial^2}{\partial t^2}\right)u-i\delta u+i\beta|u|^2u-i\mu|u|^4u \tag{19.3.31}$$

where

$$g(z)=\frac{2g_0}{1+\|u\|^2/e_0}=\frac{2g_0}{1+\frac{1}{e_0}\int_{-\infty}^{\infty}|u|^2\mathrm{d}t}\,. \tag{19.3.32}$$

The above equation is known as the cubic-quintic complex Ginzburg-Landau equation (CQGLE), and is a valid description of the averaged mode-locking dynamics when the intra-cavity fluctuations are small. Here u represents the complex envelope of the electric field propagating in the fiber. The independent variables z (the time-like variable) and t (the space-like variable) denote the propagating distance (number of cavity round-trips) and the time in the rest frame of the pulse, respectively. All the parameters are assumed to be positive throughout this manuscript, which is usually the case for physically realizable laser systems. The parameter D measures the chromatic dispersion, ν is the quintic modification of the self-phase modulation, β (cubic nonlinear gain) and μ (quintic nonlinear loss) arise directly from the averaging process in the derivation of the CQGLE.

For the linear dissipative terms (the first three terms on the right-hand-side of the CQGLE), δ is the distributed total cavity attenuation. The gain $g(z)$, which is saturated by the total cavity energy (L^2-norm) $\|u\|^2$, has two control parameters g_0 (pumping strength) and e_0 (saturated pulse energy). In principle the energy supplied by the gain can be controlled by adjusting either g_0 or e_0. In practice, however, one cannot change one of the parameters without affecting the other one. In what follows, we will assume without loss of generality that $e_0 = 1$ so that the cavity gain is totally controlled by g_0. The parameter τ characterizes the parabolic bandwidth of the saturable gain. Unless specified otherwise, the parameters are assumed to be $D = 0.4$, $\nu = 0.01$, $\tau = 0.1$, $\delta = 1$, $\beta = 0.3$ and $\mu = 0.02$ throughout the rest of this section. These parameters are physically achievable in typical ring cavity lasers.

In the appropriate parameter regime, the CQGLE supports single mode-locked pulses which are robust with respect to the initial conditions, as confirmed by experiments and numerical studies [59]. These mode-locked pulses are self-starting, and can be formed from low-amplitude white noise. When the gain parameter g_0 is increased, the stable pulse may undergo a multi-pulsing transition which is commonly observed in mode-locking systems [58, 59] and is illustrated in Fig. 213. At $g_0 = 3$ the initial condition quickly mode-locks into a single stationary pulse. When g_0 is increased, this stationary pulse eventually loses its stability at $g_0 \approx 3.15$ and a periodic solution is formed. The periodic solution consists a tall central pulse which is inherited from the previous stationary pulse, and one low-amplitude pulse on each side of the central pulse. The amplitudes of the pulses undergo small oscillations (on the order of 10^{-4}) which persist indefinitely over the propagation distance z. As g_0 keeps increasing, the two side pulses gradually move inwards. When the gain exceeds $g_0 \approx 3.24$, the periodic solution becomes unstable. At this gain value the central pulse experiences a little dip in intensity, and the system randomly chooses one of the small side pulses to be amplified and the other one to be suppressed, thus forming a double-pulse solution. The two pulses in the system have the same amount of energy, and the separation between them is fixed throughout the evolution. Additional pulses can be obtained by further increasing the value of g_0. The main focus of this manuscript is to develop a low-dimensional analytical model that describes the transition from one to two pulses as a function of the gain parameter g_0.

Figure 214 shows the POD modes obtained from a typical single-pulse evolution using a pulse-like initial condition (see left panel of Fig. 213). As mentioned before, the modal energy E_j of the j^{th} mode in a particular decomposition can

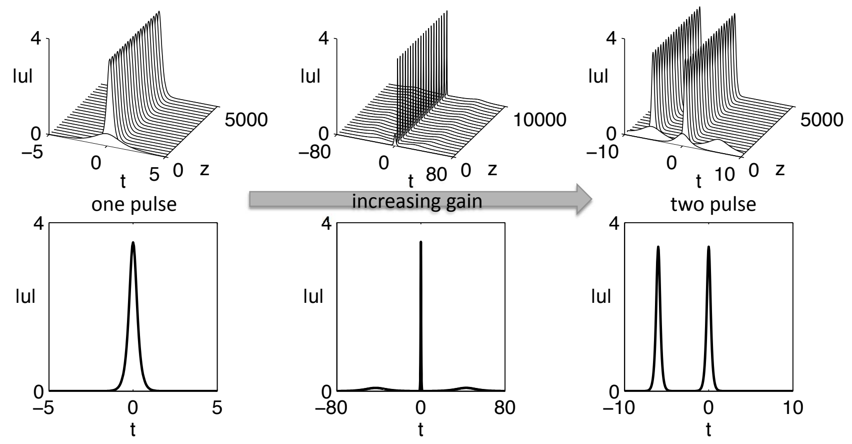


Figure 213: Multi-pulsing instability observed in the CQGLE (19.3.31). Top: As the gain is increased, the stable single mode-locked pulse (left, $g_0 = 3$) bifurcates to a periodic solution with small oscillations which persists indefinitely over z (middle, $g_0 = 3.24$), and finally to a stable double-pulse solution when the gain is too large (right, $g_0 = 3.6$). Bottom: The corresponding pulse profiles when $z \to \infty$.

be measured by the normalized eigenvalue λ_j, i.e.

$$E_j = \frac{\lambda_j}{\lambda_1 + \cdots + \lambda_M} \, . \tag{19.3.33}$$

We find that the first mode in the decomposition dominates the dynamics of the evolution by taking up over 99.9% of the total energy. All the other POD modes have negligible energy, and they represent the transient behavior in the evolution that shapes the initial pulse into the final form. One can also compute the POD modes by using random initial conditions. The higher order modes may look different and they may carry higher energy as the transient effect is different, but the fundamental mode ϕ_1 will still dominate the entire evolution.

To extend the reduced model to the periodic and double-pulse region we consider a combination of POD modes from different regions. The underlying idea is to use the dominating POD modes from different attractors of the CQ-GLE so that the resulting low-dimensional system carries as much information as possible, and thus approximates the mode-locking dynamics better. We illustrate this by combining the first m modes from the single-pulse region with the first n modes from the double-pulse region.

Denote S as the union of the two sets of orthonormal basis mentioned above, i.e.

$$S = \left\{ \phi_1^{(1)}, \cdots \phi_m^{(1)}, \phi_1^{(2)}, \cdots, \phi_n^{(2)} \right\} \, . \tag{19.3.34}$$

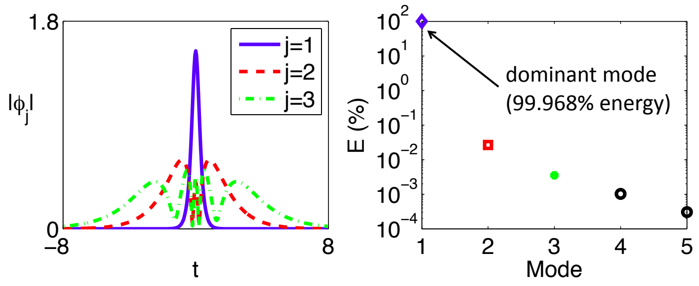


Figure 214: The POD modes and the corresponding normalized modal energies of a single-pulse evolution with $g_0 = 3$. The first three modes contain 99.968% (blue diamond), 0.0267% (red square), and 0.0035% (green asterisk) of the total energy respectively.

Here $\phi_j^{(1)}$ and $\phi_j^{(2)}$ are the POD modes computed from the single-pulse and double-pulse region respectively. The original field u can be projected onto the elements of S as

$$u = e^{i\theta_1}\left(a_1 S_1 + \sum_{j=2}^{m+n} a_j e^{i\psi_j} S_j\right) , \tag{19.3.35}$$

where S_j represent the elements of the combined basis S, a_j are the modal amplitudes, θ_1 is the phase of the first mode, and ψ_j denote the phase differences with respect to θ_1. One can obtain a low-dimensional system governing the evolutions of the modal amplitudes a_k and the phase differences ψ_k by substituting (19.3.35) into the CQGLE, multiplying the resulting equation by the complex conjugate of S_k $(k = 1, \cdots, m+n)$, integrating over all t, and then separating the result into real and imaginary parts. The reduced system has, however, a complicated form due to the fact the elements in the combined basis S are not all orthonormal to each other. To address this issue, one can orthogonalize the combined basis S prior to the projection to obtain a new basis $\{\Phi_j\}_{j=1}^{m+n}$. This can be achieved by the Gram-Schmidt algorithm given by

$$\left\{ \begin{array}{l} \Phi_1 = S_1 \ , \\ \Phi_j = \chi_j / \sqrt{\langle \chi_j, \chi_j \rangle} \ , \ j = 2, \cdots, m+n \end{array} \right. \tag{19.3.36}$$

where

$$\chi_j = S_j - \sum_{q=1}^{j-1} \langle S_j, \Phi_q \rangle \Phi_q \ , \tag{19.3.37}$$

and the inner product is defined in standard way. The reduced model derived from the new basis Φ_j has a simpler form than the one derived directly from S, since it does not have terms that are due to non-orthogonality of the basis

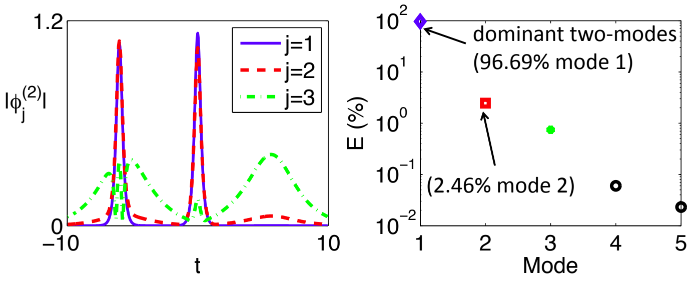


Figure 215: The POD modes and the corresponding normalized modal energies of a double-pulse evolution of the CQGLE with $g_0 = 3.6$ (cf. right column of Fig. 213). The first three modes contain 96.69% (blue diamond), 2.46% (red square), and 0.74% (green asterisk) of the total energy respectively.

functions. This reduced model will be called the $(m+n)$-model, which indicates the number of modes taken from the single-pulse (m) and double-pulse region (n).

Before moving on to the investigation of the mode-locking dynamics governed by the $(m+n)$-model, we first present the typical POD modes of a double-pulse evolution of the CQGLE in Fig. 215. The dominating modes in this case are $\phi_1^{(2)}$, $\phi_2^{(2)}$, and $\phi_3^{(2)}$, which takes up 99.9% of the total energy. As a first approximation to the CQGLE dynamics, we consider the (1+3)-model since the dominating mode in the single-pulse region is $\phi_1^{(1)}$ (see Fig. 214). The orthonormal basis $\{\Phi_j\}_{j=1}^4$ obtained using the Gram-Schmidt algorithm (19.3.36) is shown in Fig. 216. By construction, the first mode Φ_1 in the new basis is identical to $\phi_1^{(1)}$. The second mode Φ_2 contains a tall pulse which is identical to the first mode at $t = -6$ and a small bump in the origin. In general this tall pulse can be located on either the left or the right of the origin, depending on the data set U for which the POD modes are computed from. The other two modes have complicated temporal structures, and their oscillatory nature is mainly due to the orthogonalization. The (1+3)-model is derived from these four basis functions.

Classification of Solutions of the (1+3)-Model

The four-mode (1+3) model is 7-dimensional in the sense that the dynamic variables are the four modal amplitudes a_1, a_2, a_3, a_4, and the three phase differences ψ_2, ψ_3, ψ_4. To effectively characterize any periodic behavior in the

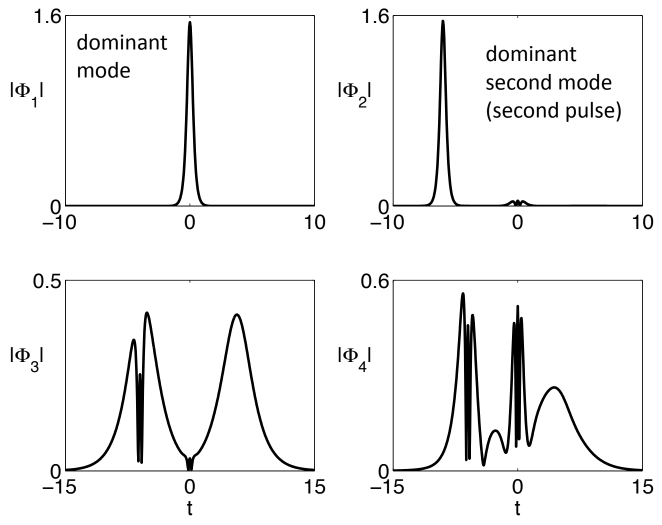


Figure 216: Basis functions used in the (1+3)-model. These functions are obtained by applying the Gram-Schmidt algorithm (19.3.36) to the set $S = \left\{\phi_1^{(1)}, \phi_1^{(2)}, \phi_2^{(2)}, \phi_3^{(2)}\right\}$.

system, it is customary to use the new set of variables

$$\begin{cases} x_1 = a_1 \ , \\ x_j = a_j \cos\psi_j \ , \ y_j = a_j \sin\psi_j \end{cases} \tag{19.3.38}$$

for $j = 2, 3, 4$ so that the formal projection of the field is given by

$$u = e^{i\theta_1} \left(x_1 \Phi_1 + \sum_{j \geq 2} (x_j + iy_j)\, \Phi_j \right) \ . \tag{19.3.39}$$

Figure 217 shows the solution of the (1+3)-model expressed in terms of the above variables and the reconstructed field u at different cavity gain g_0. At $g_0 = 3$ (top panel), the (1+3)-model supports a stable fixed point with $x_1 = 2.2868$ while the other variables are nearly zero, i.e. the steady state content of u mainly comes from Φ_1, and thus a single mode-locked pulse centered at $t = 0$ is expected. Increasing g_0 eventually destabilizes the fixed point and a periodic solution is formed (middle panel) which corresponds to a limit cycle in the 7-dimensional phase space. The reconstructed field has a tall pulse centered at the origin due to the significant contribution of x_1 (and hence Φ_1). The

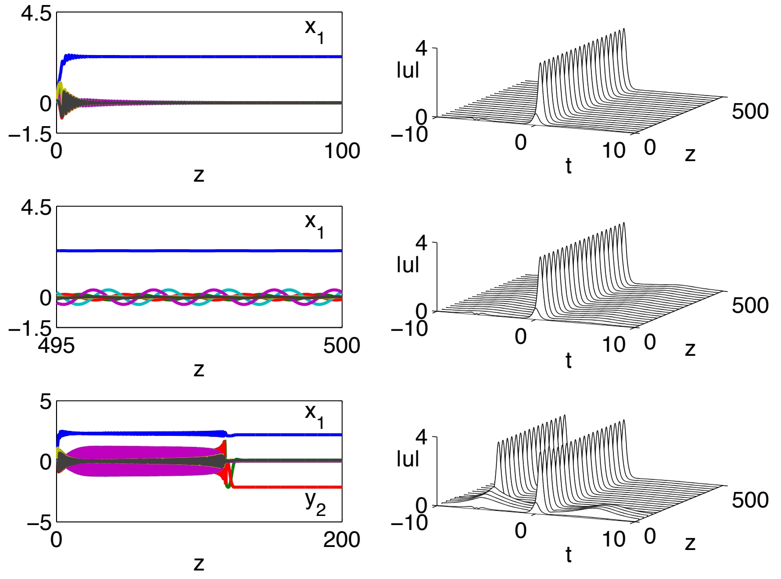


Figure 217: Left: The solution of the (1+3)-model at $g_0 = 3$ (top), $g_0 = 3.24$ (middle) and $g_0 = 3.8$ (bottom). Here only the significant variables (in absolute value) are labeled. Right: The corresponding reconstructed field u from the variables of the (1+3)-model. [From E. Ding, E. Shlizerman and J. N. Kutz, *Modeling multipulsing transition in ring cavity lasers with proper orthogonal decomposition*, Physical Review A **82**, 023823 (2010) [57] ©APS]

small-amplitude side pulses are resulted from the linear combination of the higher order modes present in the system, and their locations are completely determined by the data matrix U representing the double-pulse evolution of the CQGLE. Further increasing g_0 causes the limit cycle to lose its stability and bifurcate to another type of stable fixed point. Unlike the first fixed point, this new fixed point has significant contributions from both x_1 and y_2 which are associated to Φ_1 and Φ_2. Since the other variables in the system become negligible when $z \to \infty$, a double-pulse solution is formed.

The (1+3)-model is able to describe both the single-pulse and double-pulse evolution of the CQGLE, which is a significant improvement compared to the 2-mode model considered in the previous section. The small-amplitude side pulses in the reconstructed periodic solution do not appear at the expected locations predicted by the CQGLE (see middle column of Fig. 213), since we are using the data with a particular gain ($g_0 = 3.6$) to characterize the mode-locking behavior in the whole range of gain. Nevertheless, the two key features of the periodic solution of the CQGLE are explicitly captured: (i) the existence of side

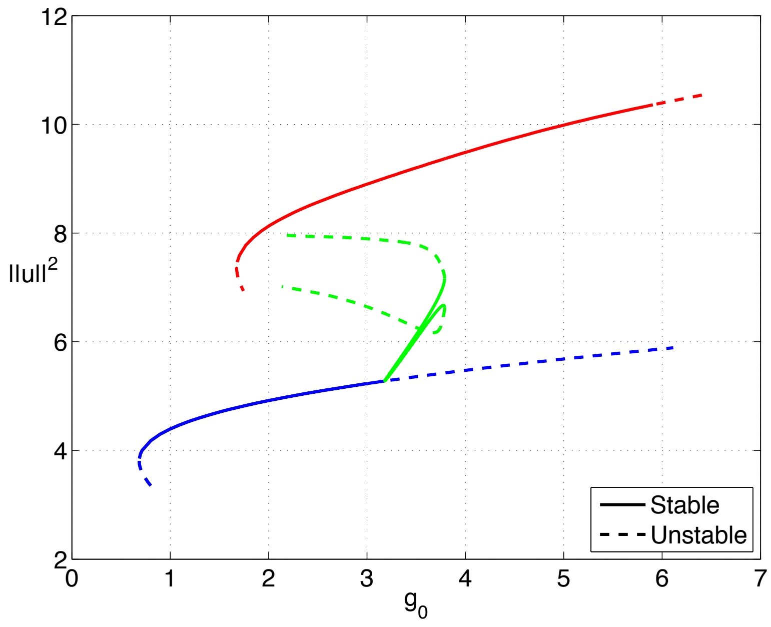


Figure 218: Bifurcation diagram of the total cavity energy of the reconstructed field u as a function of g_0. Here the bottom branch (blue) represents the single-pulse solution and the top branch (red) represents the double-pulse solution. The middle branch (green) denotes the extrema of the energy of the periodic solution. [From E. Ding, E. Shlizerman and J. N. Kutz, *Modeling multipulsing transition in ring cavity lasers with proper orthogonal decomposition*, Physical Review A **82**, 023823 (2010) [57] ©APS]

pulses, and (ii) the small-amplitude oscillations of the entire structure. These two features are not demonstrated by the reduced models considered before. The (1+3)-model also captures the amplification and suppression of the side pulses in the formation of the double-pulse solution in the CQGLE. Since the POD modes in the double-pulse region are computed from the numerical data where the left side pulse is chosen over the right side pulse (see Fig. 213), the second pulse in the reconstructed field always forms on the left. The case where the second pulse is formed on the right is not of particular interest since the underlying dynamics is essentially the same.

We construct a global bifurcation diagram in Fig. 218 to characterize the transition between the single-pulse and double-pulse solution of the (1+3)-model. The diagram shows the total cavity energy of the reconstructed field u as a function of the cavity gain g_0, which is obtained using MATCONT [63].

In terms of the dynamic variables, the energy of the field is given by

$$\|u\|^2 = x_1^2 + \sum_{j\geq 2} \left(x_j^2 + y_j^2\right) \ . \tag{19.3.40}$$

In the diagram, the branch with the lowest energy corresponds to the single mode-locked pulses of the (1+3)-model. With increasing g_0, the reconstructed mode-locked pulse readjusts its amplitude and width accordingly to accommodate the increase in the cavity energy. The branch of the single-pulse solutions is stable until $g_0 = 3.181$ where a Hopf bifurcation occurs. The fluctuation in the cavity energy as a result of the periodic solution is small at first, but increases gradually with g_0. At $g_0 = 3.764$ the periodic solution becomes unstable by a fold bifurcation of the limit cycle. In this case a discrete jump is observed in the bifurcation diagram and a new branch of solutions is reached. This new branch represents the double-pulse solution of the system and has higher energy than the single-pulse branch and the periodic branch.

The bifurcation diagram also illustrates that in the region $1.681 \leq g_0 \leq 3.181$ the single-pulse and the double-pulse solution are stable simultaneously. This bistability is not restricted to the (1+3)-model as it also happens in the CQGLE as well. Given a set of parameters, the final form of the solution is determined by the basin of attraction of the single-pulse and double-pulse branches rather than the initial energy content. When the initial condition is selected such that it is "close" to the double-pulse branch (characterized by the Euclidean distance between them), the result of linear analysis holds and a double-pulse mode-locked state can be achieved. For random initial data that is far away from both branches, the system tends to settle into the single-pulse branch as it is more energetically favorable.

Comparison of Different (m+n)-Models

To demonstrate that the (1+3)-model is the optimal (m+n)-model to characterize the transition from a single mode-locked pulse into two pulses that occurs in the CQGLE, we consider the (1+2)-model whose results are shown in the top row of Fig. 219. This model can be derived by setting the variables x_4 and y_4 in the (1+3)-model to zero. At $g_0 = 3$ a tall stable pulse in formed at $t = -6$ together with a small residual at the origin. The entire structure has close resemblance to Φ_2 from the Gram-Schmidt procedure. Except for the location, the tall pulse agrees with the single-pulse solution of the CQGLE quantitatively. One can think of this as the result of the translational invariance of the CQGLE. In the full simulation of the CQGLE, the small residual is automatically removed by the intensity discriminating mechanism (saturable absorption). When the structure loses its stability, a periodic solution is formed which persists even at unrealistically large values of cavity gain such as $g_0 = 7$.

The middle row of the same figure shows the numerical simulations of the (1+4)-model, which is derived from the combination of the first POD mode

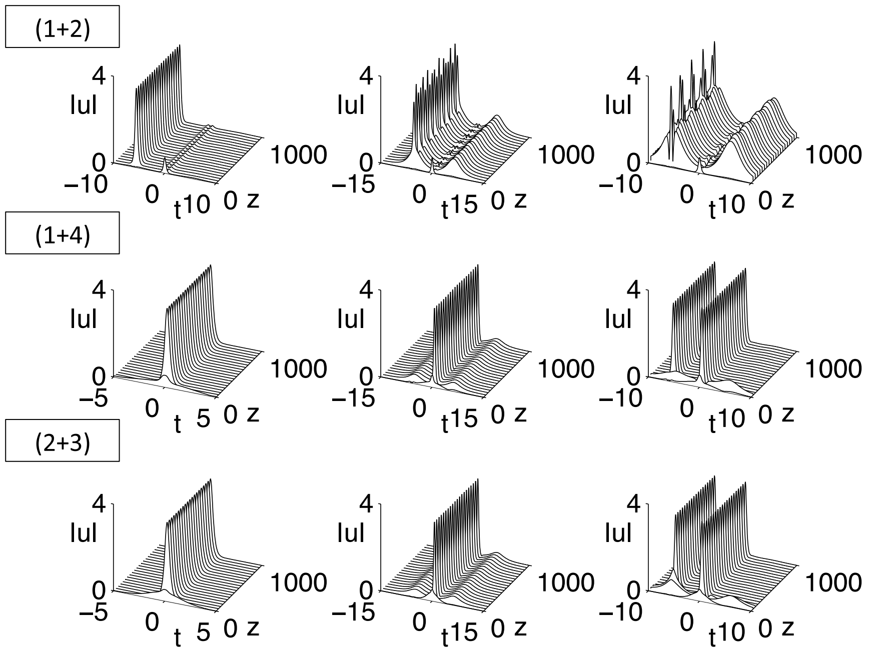


Figure 219: Top: The reconstructed field of the (1+2)-model at $g_0 = 3$ (left), $g_0 = 4.3$ (middle), and $g_0 = 7$ (right). Middle: The reconstructed field of the (1+4)-model at $g_0 = 3$ (left), $g_0 = 3.4$ (middle), and $g_0 = 3.8$ (right). Bottom: The reconstructed field of the (2+3)-model at $g_0 = 3$ (left), $g_0 = 3.4$ (middle), and $g_0 = 3.7$ (right). [From E. Ding, E. Shlizerman and J. N. Kutz, *Modeling multipulsing transition in ring cavity lasers with proper orthogonal decomposition*, Physical Review A **82**, 023823 (2010) [57] ©APS]

from the single-pulse region with the first four modes from the double-pulse region. This model is able to reproduce all the key dynamics observed during the multi-pulsing phenomenon shown in Fig. 218. The difference between the (1+4)-model and the (1+3)-model is the locations at which the bifurcations occur. In the previous section we showed that the single-pulse solution described by the (1+3)-model loses stability at $g_0 = 3.181$ (Hopf bifurcation), and the subsequent periodic solution bifurcates into the double-pulse solution at $g_0 = 3.764$. For the (1+4)-model these two bifurcations occur at $g_0 = 3.173$ and $g_0 = 3.722$ respectively, which are closer to the values predicted using the CQGLE ($g_0 \approx 3.15$ and $g_0 \approx 3.24$). The (2+3)-model (bottom row of Fig. 219) produces similar results as the (1+3)- and the (1+4)-model, with pulse transitions occurring at

$g_0 = 3.177$ and 3.678. The comparison here shows that the third POD mode $\phi_3^{(2)}$ from the double-pulse region is essential for getting the right pulse transition. Once this mode is included, further addition of the POD modes has only a minor effect on the accuracy of the low-dimensional model.

19.4 The POD method and symmetries/invariances

The POD reduction technique can be a powerful tool in projecting seemingly complex dynamics to a low-dimensional manifold where all the *important* dynamics occurs. Of course, it is not without its drawbacks and limitations. In particular, the method can often fail if the data sampled is simply put through the SVD algorithm without considering its basic structure and potential invariances. Invariances can arise due to various symmetry considerations in a given physical system, for instance, translational or rotational invariance.

As an example, we can once again consider the N-soliton dynamics as illustrated in Figs. 207-210. In this example, the N-soliton ($N = 1$ and $N = 2$ of the nonlinear Schrödinger equation was integrated using a spectral method in order to generate the data for sampling. A small change to this simulation can make a tremendous difference in the results of the POD reduction scheme. Specifically, consider integrating once again the nonlinear Schrödinger equation (19.1.14) but with the initial condition

$$u(x,0) = N\text{sech}(x + x_0)\exp(i\Omega x) \tag{19.4.41}$$

where Ω represents a center-frequency shift of the soliton and x_0 is a center-position offset variable. The center-frequency perturbation to the N-soliton solution causes the soliton solution to move at a prescribed group velocity. Indeed, such a group-velocity movement of the pulse, when the center-frequency is driven by noise, leads to a fundamental limit in fiber optical communications systems known as the Gordon-Haus jitter [61].

To observe the impact of the center-frequency on the soliton propagation, consider numerically integrating the nonlinear Schrödinger equation with the above initial conditions and $N = 2$, $\Omega = \pi$ and $x_0 = 10$. Figure 220 (left panel) shows the soliton evolution. It is clearly seen that the center frequency causes a group-velocity drift in the pulse. Other than that, however, the $N = 2$ soliton dynamics appears to be identical to what is observed and characterized in Figs. 207-210. As before, we could arrange our numerical simulations into a data matrix and perform an SVD reduction. The bottom left panel of Fig. 221 shows the singular values that would result from such a reduction. It is clear in this case that such a method would not result in a reduction of the dynamics as the translating data, when correlated across time and space, does no produce redundant data. Indeed, there is a very slow decay of the singular values and the POD method cannot be used to generate a low-dimensional manifold for the dynamics.

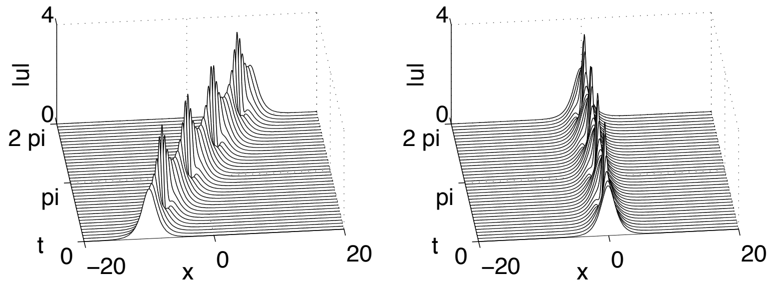


Figure 220: Evolution dynamics of the $N = 2$ soliton solution when subject to a center-frequency shift (left panel) and when the translational dynamics is factored out (right panel). The translating 2-soliton generates a high-dimensional singular value distribution. In contrast, by factoring out the translational mode, the distribution once again becomes dominated by two modes as previously shown in Figs. 207-210.

The failure of the method in this case is due simply to the translational invariance. If the invariance is *removed*, or factored out [62], before a data reduction is attempted, then the POD method can once again be used. In order to remove the invariance, the invariance must first be identified and an auxiliary variable defined. Thus we consider the dynamics rewritten as

$$u(x,t) \to u(x - c(t)) \tag{19.4.42}$$

where $c(t)$ corresponds to the translational invariance in the system responsible for limiting the POD method. The parameter c can be found by a number of methods. Rowley and Marsden [62] propose a template based technique for factoring out the invariance. Here, a simple center-of-mass calculation will be used to compute the location of the soliton.

The following code can be implemented to find the center position of the pulse as it evolves in time

```
for j=1:length(t)
   com=(x.*abs(usol(j,:)).^2);
   com2=(abs(usol(j,:)).^2);
   c(j)=trapz(x,com)/trapz(x,com2);
end
```

The end result of this computation is the production of the vector or function **c** which prescribes the location of the pulse. The center position can then be factored out by the following lines of code:

```
for j=1:length(t)
```

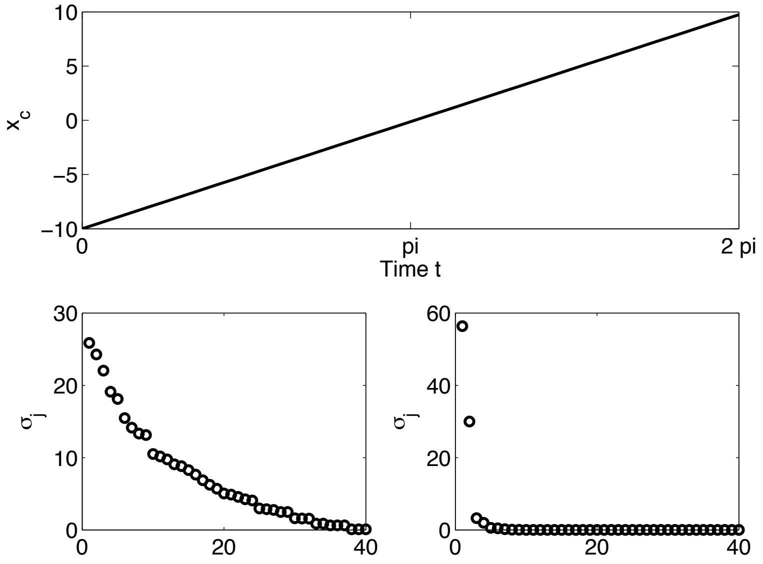


Figure 221: Dynamics of the center position $x_c(t)$ (top panel) showing the constant translation of the 2-soliton to the right in the computational domain. The distribution of singular values for the two data sets demonstrated in Fig. 220 are exhibited in the bottom panels. Without factoring out the translation, the SVD does not produce low-dimesional behavior (bottom left) whereas appropriate factoring of the data reveals the dominant two-mode dynamics (bottom right).

```
    [mn,jj]=min(abs(x-c(j)));
    ns=n/2-jj;
    usift=[usol(j,:) usol(j,:) usol(j,:)];
    usol_shift(j,:)=usift(1,n+1-ns:2*n-ns);
  end
```

The new matrix **usol_shift** has now had the group velocity drift factored out. Figure 220 (right panel) shows the net result of this. The center position vector as a function of time is demonstrated in the top panel of Fig. 221. Once the translation is factored out, the SVD reduction of the remaining data set produces the singular value in the bottom right panel of Fig. 221. This singular value distribution is identical to what was previously demonstrated in the simple $N = 2$ case. Thus the dynamics is now three dimensional: two modes interact nonlinearly to describe the pulse changes while one mode describes the translation. This example serves to demonstrate that if the invariance is handled properly, only a single degree of freedom extra is required to account for its action in the dynamics.

Advection-Diffusion Equations

A more sophisticated example of handling such an invariance is provided by the advection-diffusion equations (See Secs. 10.3 and 25.4):

$$\frac{\partial \omega}{\partial t} + [\psi, \omega] = \nu \nabla^2 \omega \tag{19.4.43a}$$

$$\nabla^2 \psi = \omega \tag{19.4.43b}$$

where

$$[\psi, \omega] = \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} - \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} \tag{19.4.44}$$

and $\nabla^2 = \partial_x^2 + \partial_y^2$ is the two dimensional Laplacian. A spectral method has been developed for solving this problem in Sec. 10.3. Here, the initial data applied to this problem will be a dipole-type initial condition

$$\omega(x, y, 0) = \exp[-(x+8)^2 - (y-2)^2] - \exp[-(x+8)^2 - (y+2)^2]\,. \tag{19.4.45}$$

Such an initial condition leads to the formation of a dipole pair that translates to the right as depicted in Fig. 222. The evolution over nine time frames is shown using the time span variable

```
tspan=linspace(0,135,9);
```

A nearly constant velocity is displayed as the dipole pair is advected to the right of the computational domain.

As previously noted, the dimensionality of the dynamics can be assessed by using the SVD on the computational data. In this case, the solution is first allowed to settle to the dipole state before sampling is applied. For this case, the time domain and resulting sampling in the advection-diffusion dynamics is given by

```
tspan=linspace(0,135,60);
```

where sixty slices of data are taken, but only the last forty slices of this data will be used in our POD projection. Given that the data is two-dimensional, it first must be reshaped into vector form before applying the SVD.

```
for j=21:length(t)
  w=real(ifft2(reshape(wtsol(j,:),nx,ny)));
  A(j-20,:)=reshape(w,1,nx*ny);
end
[U,S,V]=svd(A);
```

The top left panel of Fig. 223 shows the distribution of singular values after sampling the dynamics. Again, without taking care of the translation, the

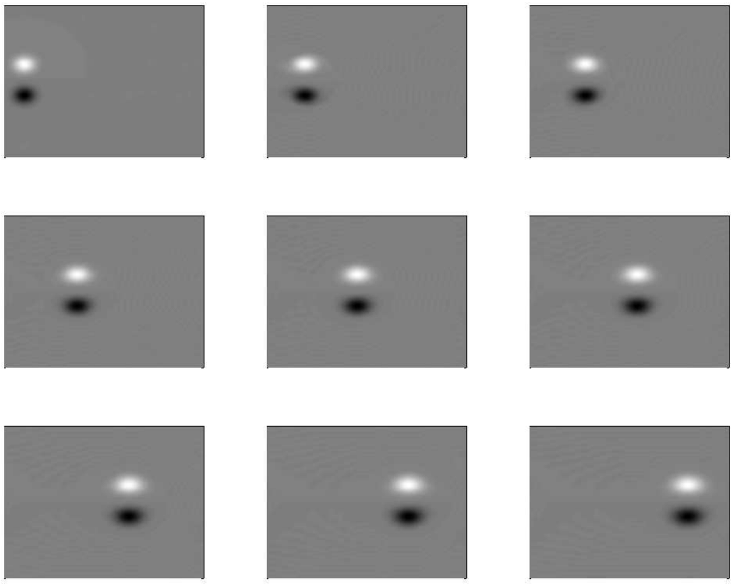


Figure 222: Evolution dynamics of a dipole-like initial condition in evenly spaced increments over the time frame $t \in [0, 135]$. The dipole is seen to generate a translation of the structure to the right of the computational domain.

dynamics appears to be high dimensional. However, it is clear from Fig. 222 that the dynamics should indeed by low-dimensional with a dominant dipole mode translating from left to right.

As previously, a computation must be made of the translation of the dipole mode. Here, since the data is symmetric about $y = 0$, it only remains to find the center-of-mass via a standard moment method. The following code finds the center position variable corresponding to the translation:

```
for j=1:length(t)
   w=ifft2(reshape(wtsol(j,:),nx,ny));
   for jj=1:nx
       wy(jj)=trapz(y,abs(w(:,jj)));
   end
   com=(x.*wy);
   com2=(wy);
   c(j)=trapz(x,com)/trapz(x,com2);
end
```

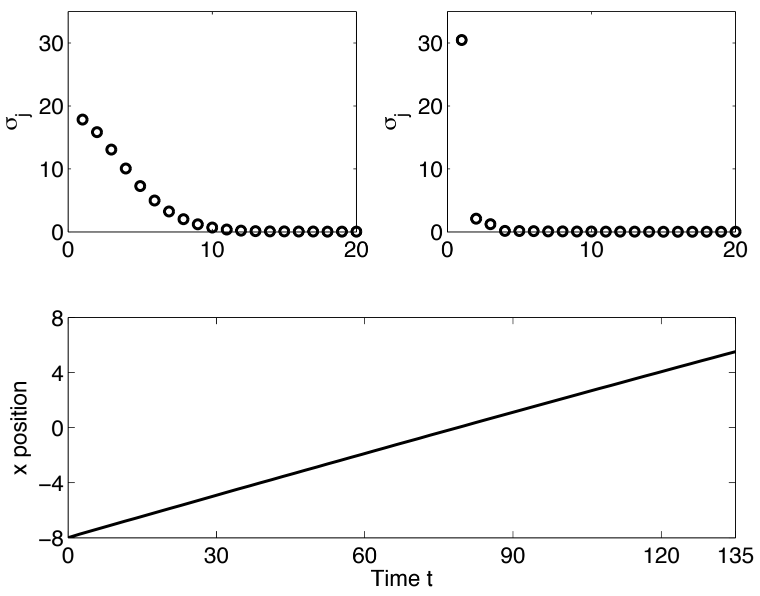


Figure 223: Dynamics of the center position (bottom panel) showing the constant translation of the dipole solution to the right in the computational domain. The distribution of singular values for the raw data and factored out data are exhibited in the top panels. Without factoring out the translation, the SVD does not produce low-dimesional behavior (top left) whereas appropriate factoring of the data reveals a dominant one-mode dynamic (top right).

The resulting translation vector is plotted in the bottom panel of Fig. 223. This gives the auxiliary variable that is to be factored out of the dynamics. The following code factors out the translation in order to align the data for the SVD and dimensionality reduction step.

```
for j=21:length(t)
  w=ifft2(reshape(wtsol(j,:),nx,ny));
  [mn,jj]=min(abs(x-c(j)));
  ns=nx/2-jj;
  usift=[w w w];
  w_shift=usift(:,nx+1-ns:2*nx-ns);
  A(j-20,:)=reshape(w_shift,nx*ny,1);
end
[U,S,V]=svd(A);
```

The top right panel of Fig. 223 shows the singular value distribution for this factored case. Note that the dynamics, as expected, is dominated by a single

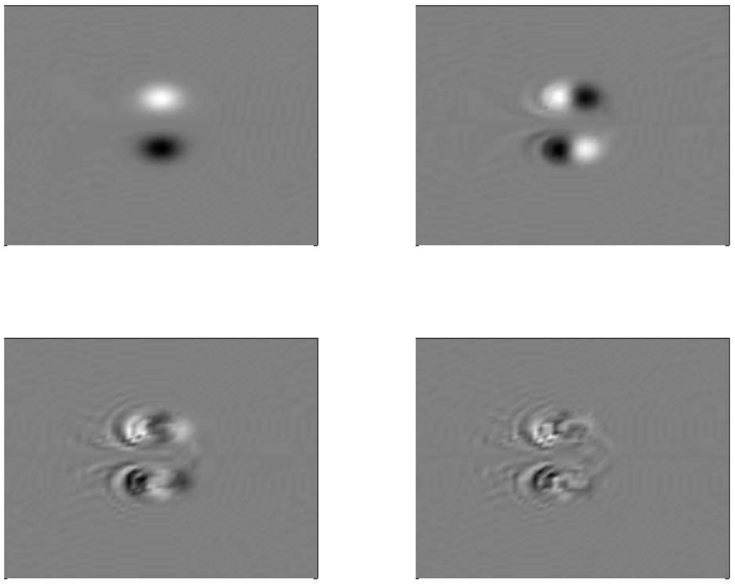


Figure 224: The four most dominant modes of the SVD decomposition for the data which has factored out the translational invariance. The first mode (top right) contains 88% of the modal energy while the first four contains 98%.

mode, i.e. the dipole mode. Figure 224 shows the first four modes of the SVD reduction. The first mode is clearly the dominant dipole solution. Modes two through four show the effects of the advection at the back of the dipole, i.e. fluid is ejected out the back of the dipole as it moves towards the left. Note that 88% of the energy is in the first dipole mode while 98% is in the first four modes.

A one-mode decomposition of the data can be performed by assuming a solution of the form

$$\omega(x,y,t) = a(t)\phi(x,y) \tag{19.4.46}$$

where $\phi(x,y)$ is the dominant dipole mode. Once the vorticity is determined, then the streamfunction can be computed from

$$\nabla^2\psi = \omega = a(t)\phi(x,y) \ \rightarrow \ \psi = a(t)L^{-1}\phi(x,y) \tag{19.4.47}$$

where the operator L^{-1} inverts the Laplacian operator. Plugging in the one-mode expansion into the governing equations and taking the inner product of both sides of the equation for vorticity yields

$$\frac{da}{dt} = \alpha a + \beta a^2 \tag{19.4.48}$$

where

$$\alpha = \nu \frac{(\nabla^2 \phi, \phi)}{(\phi, \phi)} \tag{19.4.49a}$$

$$\beta = \frac{((L^-1\phi)_y \phi_x, \phi)}{(\phi, \phi)} - \frac{((L^-1\phi)_x \phi_y, \phi)}{(\phi, \phi)} \tag{19.4.49b}$$

where the coefficients α and β are determined from inner products once the mode structures are computed from the SVD. Due to symmetry considerations of the dipole solution, $\beta = 0$, thus reducing everything to the calculation of α alone. The calculation of α can be performed with the following code:

```
w1=reshape(V(1,:),nx,ny);  w1t=fft2(w1);
w1lap=-real(ifft2(K.*w1t));

Q2=w1lap.*w1;
Q3=w1.*w1;
for j=1:nx
  s2(j)=trapz(y,Q2(j,:));
  s3(j)=trapz(y,Q3(j,:));
end
alpha=trapz(x,s2)/trapz(x,s3)
```

This yields a value of $\alpha = -1.771\nu$. Thus the leading order (dominant) POD solution for the dipole is simply exponential decay given by

$$a(t) = a(0)\exp(-1.771\nu t)\,, \tag{19.4.50}$$

where $a(0) = (\omega(x, y, 0), \phi(x, y))$. In total then, the overall solution has been deconstructed into its constitute parts. To reconstruct, one simply needs to put the three pieces together: the time dynamics of the mode-amplitude, the dominant mode-structure, and the translational dynamics. This gives the one-mode expansion for the dipole dynamics:

$$\omega(x, y, t) = a(0)\exp(-1.771\nu t)\phi(x - c(t), y)\,. \tag{19.4.51}$$

Of course, a more complicated dynamics would result if a higher-order mode expansion was assumed. However, 88% of the energy is in this single mode which was handled nicely by factoring out the translation.

To conclude this discussion, one can easily imagine factoring out a myriad of other symmetries. The most obvious is translation, but perhaps the second most obvious is factoring out a rotation. This might arise, for example, when considering spiral wave dynamics in reaction-diffusion systems. Ultimately, this is simply a data pre-processing step which must be performed prior to the dimensionality reduction step.

19.5 POD using robust PCA

The POD technique is ideal for sampling data, determining the low-dimensional structure therein, and exploiting the low-rank structure in the context of dynamical systems. The examples considered to this point demonstrate this by using the POD basis in combination with Galerkin projection in order to reduced the dynamics of a given complex system to a low-dimensional manifold on which the dynamics occurs. At the heart of this dimension reduction is the singular value decomposition and its various guises, most notably principal component analysis or proper orthogonal decomposition. The SVD produces characteristic features (principal components) that are determined by the covariance matrix of the data. Fundamental in producing the SVD/PCA/POD modes is the L^2 norm for data fitting. Although the L^2 norm is highly appealing due to its centrality (and physical interpretation) in most applications as an *energy*, it does have potential flaws that can severely limit its applicability when trying to integrate it with dynamical systems theory. In particular, if the data gathered through measurement is corrupt or suffers from large and sparse noise fluctuations, the higher-dimensional least-square fitting to the data matrix by the SVD algorithm squares this pernicious error; leading to significant deformation of the singular values and PCA/POD modes describing the data. Although many physical systems and/or computations are relatively free of data corruption, many modern applications, such as PIV fluid measurements, are rife with arbitrary corruption of the measured data due to the sensitivity of measuring *derivative*-type data. This ensures that the standard application of the SVD will be of limited use.

Ideally, in performing dimensionality reduction one should not allow the corrupt or large noise fluctuations to so strongly influence one's results. In order to limit the impact of such *data outliers*, a different measure, or norm, can be envisioned in measuring the best data fit (SVD/PCA/POD modes). One measure that has recently produced great success in this arena is the L^1 norm [51] which no longer squares the distance of the data to the best-fit mode. Indeed, the L^1 norm has been demonstrated to promote sparsity and is the underlying theoretical construct in *compressive sensing* or *sparse sensing* algorithms (See Sec. 18 for further details). Here, the L^1 norm is simply used as an alternative measure (norm) for performing the equivalent of the least-square fit to the data. As a result, a more robust method is achieved of computing PCA components, i.e. the so-called *robust PCA* method.

The robust PCA technique has already been considered in Sec. 15.6. Briefly, the idea of the robust PCA is to provide an algorithm capable of separating in an efficient manner low-rank data with corrupt (sparse) measurement/matrix error. Thus the idea would be to consider a matrix which is composed of these two elements together

$$\mathbf{M} = \mathbf{L} + \mathbf{S} \tag{19.5.1}$$

where $\mathbf{M}$ is the data matrix of interest that is composed of low-rank structure $\mathbf{L}$ along with some sparse data $\mathbf{S}$. The data-analysis objective is the following:

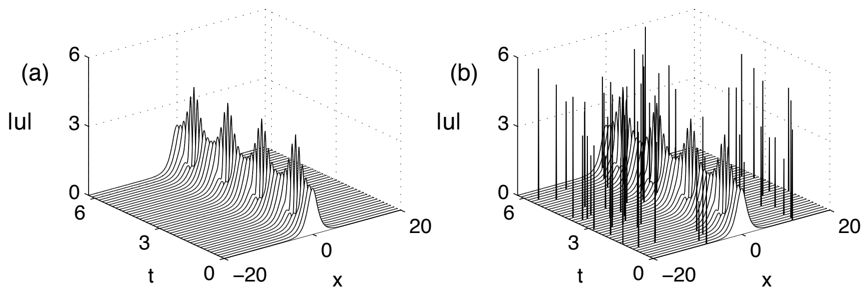


Figure 225: (a) Ideal evolution dynamics of the two-soliton of the nonlinear Schrödinger equation, and (b) the ideal data matrix corrupted by a sparse set of sixty measurements.

given measurements or observations **M**, can the low-rank matrix **L** and sparse matrix **S** be recovered? Adding to the difficulty of this task is the following: neither the rank of the matrix **L**, nor the number of non-zero (sparse) elements of the matrix **S** are known.

The separation problem seems impossible to solve since the number of unknowns to infer for **L** and **S** is twice as many as the given measurements in **M**. However, Candés and co-workers [51] have recently proved that this can be indeed solved through the formulation of a tractable convex optimization problem. In real applications, however, it is rare that the matrix decomposition is as ideal as (19.5.1). More generally, the decomposition is of the form

$$\mathbf{M} = \mathbf{L} + \mathbf{S} + \mathbf{N} \tag{19.5.2}$$

where the matrix **N** is a dense, small perturbation accounting for the fact that the low-rank component is only approximately low-rank and that small errors can be added to all the entries. But even in this case, the convex optimization algorithm for generating robust PCA seems to do a remarkable job as separating low-rank from sparse data. This can be used to great advantage when certain types of data are considered, namely those with corrupt data or large, sparse noisy perturbations.

POD mode selection via robust PCA

To illustrate the role of robust PCA in dimensionality reduction of dynamical systems, we once again revisit the two-soliton example considered throughout this chapter. The two-soliton dynamics are illustrated in Figs. 207-210. It is once again demonstrated in Fig. 225 along with a modified two-soliton data matrix which has been corrupted by sparse noise fluctuations. Assuming the two-soliton (nonlinear Schrödinger solver) has been executed with the initial condition $u(x, 0) = 2\text{sech}x$ and over the interval $t \in [0, 2\pi]$, then the resulting

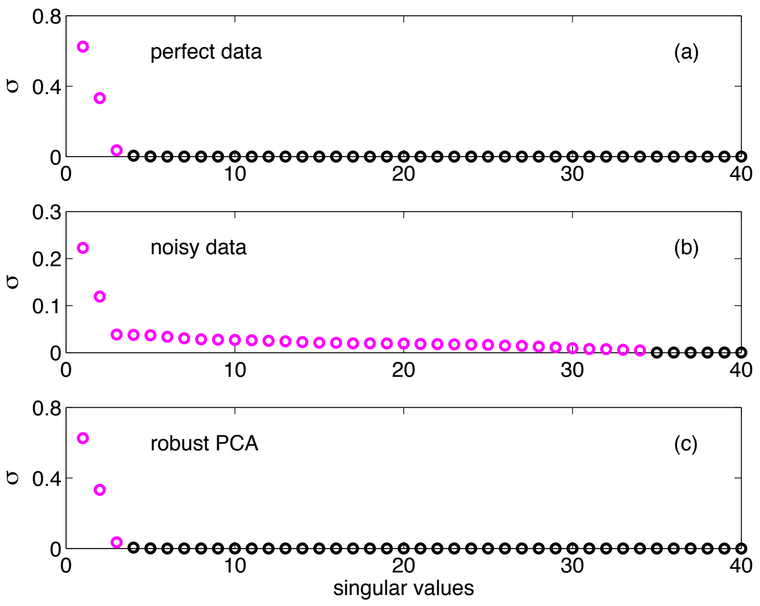


Figure 226: Singular values of the data matrices for the ideal data (top panel), the corrupted data (middle panel), and the low-rank data computed through robust PCA (bottom panel). The robust PCA construction produces almost a perfect match to the original, ideal data. The magenta dots represent the number of modes necessary to construct 99% of the original data. In this case, the ideal and robust PCA require three modes while the corrupt data requires 34 modes.

complex solution is contained in the matrix **usol**. This matrix is corrupted by 60 randomly chosen, sparse noise fluctuations with the following code:

```
sam=60; % corrupt data points
Atest2=zeros(length(t),n);
Arand1=rand(length(t),n);
Arand2=rand(length(t),n);
r1 = randintrlv([1:length(t)*n],793);
r1k= r1(1:sam);
for j=1:sam
    Atest2(r1k(j))=-1;
end
Anoise=Atest2.*(Arand1+i*Arand2);
unoise=usol+5*Anoise;

subplot(2,2,1), waterfall(x,t,abs(usol)); colormap([0 0 0]);
```

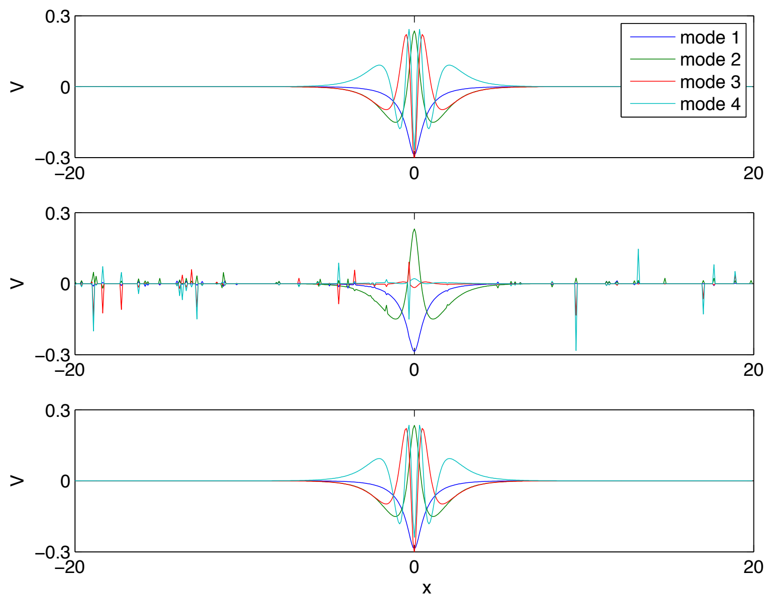


Figure 227: Dominant modes for the ideal data (top panel), the corrupted data (middle panel), and the low-rank data computed through robust PCA (bottom panel). The ideal and robust PCA produce the same dominant two- to three-modes while the corrupt data produces highly erratic modes.

```
subplot(2,2,2), waterfall(x,t,abs(unoise)); colormap([0 0 0]);
```

Recall that the noise added has both real and imaginary parts as is the case for the complex evolution field given by the nonlinear Schrödinger equation.

Previously, the data matrix was used in its raw (and perfect) form to generate the low-dimensional manifold spanned by a reduced set of POD modes. However, with the sparse corruption of the data, it seems unlikely that the desired low-dimensional manifold can be computed. Application of the SVD to the ideal data and corrupt data shows significant differences. The following code produces the singular values and POD modes for each.

```
A1=usol; A2=unoise;
[U1,S1,V1]=svd(A1);
[U2,S2,V2]=svd(A2);
```

Figure 226 and 227 show the singular values and POD modes for both reductions. In the top panel of both figures are the results for the ideal data, showing that a clear two- to three-mode dominance exists in the data, i.e. three-modes (magenta dots) captures more than 99% of the data matrix. Indeed, this was

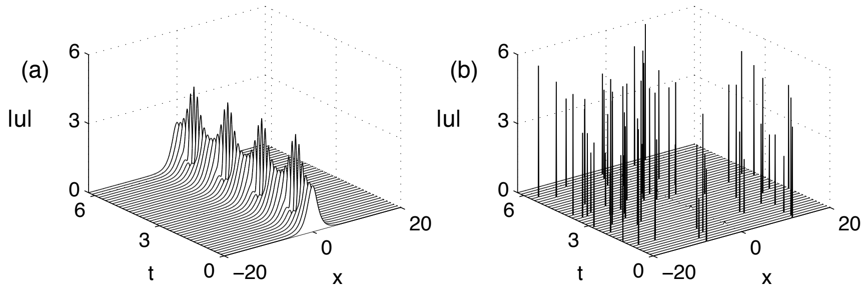


Figure 228: Separation of the original corrupt data matrix shown in Fig. 225(b) into a low-rank component (a) and a sparse component (b). The separation is almost perfect, with the low-rank matrix being dominated by two- to three-modes, i.e. the three modes contain greater than 99% of the energy.

directly exploited in Sec. 19.2 for the construction of a low-dimensional manifold for the dynamical evolution of the system. In contrast, the middle panels show that the corrupted data matrix activates a significant number of modes. Specifically, it now takes 34 modes (magenta circles) in order to capture 99% of the corrupt data matrix. The POD modes are also now severely misshaped from their ideal, noisy free state.

To remove the corrupted data, the **inexact_alm_rpca** algorithm is used which is based upon Alternating Direction method [46]. For our purposes, we will use this MATLAB program since it is extremely simple to use and exceedingly fast. The following code separates the low-rank from the sparse data as in (19.5.1)

```
ur=real(unoise);
ui=imag(unoise);

lambda=0.2;
[R1r,R2r]=inexact_alm_rpca(real(ur.'),lambda);
[R1i,R2i]=inexact_alm_rpca(real(ui.'),lambda);

R1=R1r+i*R1i;
R2=R2r+i*R2i;

[U3,S3,V3]=svd(R1.');
```

Note that the use of **inexact_alm_rpca** requires the data be real. Additionally, there is a parameter *lambda* that can adjusted to best separate the low-rank from sparse data. This is considered in more detail in Sec. 15.6.

The real and imaginary portions are put back together at the end of the algorithm in order to SVD the low-rank portion of interest, i.e. the portion of

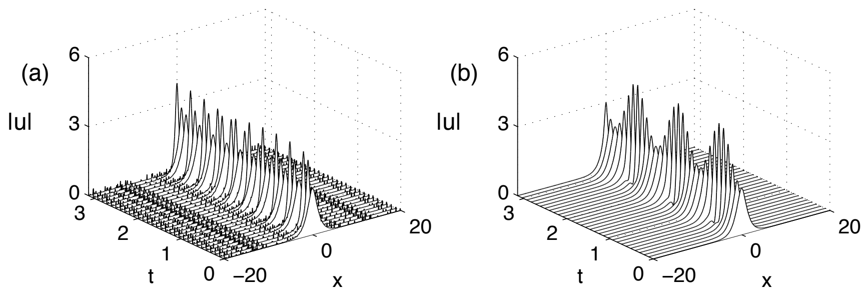


Figure 229: Comparison of the two-mode POD evolution given by (19.2.28) of Sec. 19.2 using the (a) SVD reduction of the corrupt data matrix and (b) using the low-rank data generated from the robust PCA algorithm. The use of the robust PCA algorithm renders the POD results almost identical to those with the ideal data, thus suggesting it can potentially be a filter for corrupt data before applying POD reductions.

the data considered to be without corrupt data. The two matrices corresponding to the low-rank ($R1$) and sparse ($R2$) portions of the data matrix are plotted in Fig. 228. The parameter *lambda* used in the **inexact_alm_rpca** algorithm can be tuned to best separate the sparse from low-rank matrices in (15.6.1). Here a value of 0.2 is used, which produces an almost ideal separation as is shown in Figs. 226(bottom panel), 227(bottom panel) and 228. Indeed, an almost perfect separation is achieved as advertised (guaranteed) by Candés and co-workers [51].

Finally, the POD dynamics are compared for a two-mode truncation of the corrupt data versus the low-rank approximation of the data achieved through robust PCA. Figure 229 shows the results of both two-mode expansions. The robust PCA modes produces almost identical results to those achieved with the standard POD reduction using the ideal data. In contrast, the POD reduction using the corrupt data produces inner products in (19.2.28) of Sec. 19.2 that greatly skew their value and their ability to predict accurately the true underlying evolution. Indeed, the two POD mode dynamics is significantly deteriorated from its robust PCA counterpart.

As was already pointed out in Sec. 15.6 the robust PCA algorithm advocated for here is quite remarkable in its ability to separate sparse corruption from low-rank structure in a given data matrix. The application of robust PCA essentially acts as an advanced filter for cleaning up a data matrix. Indeed, one can potentially use this as a filter which is very unlike the time-frequency filtering schemes considered previously. This has the potential to greatly enhance the POD reduction schemes advocated here when they are subject to corrupt data measurements.

20 Dynamic Mode Decomposition

Exploiting low-dimensionality in complex systems has already been demonstrated to be an effective method for either computationally or theoretically reducing a given system to a more tractable form. In what has been proposed so far with POD reductions, we have used *model-based* algorithms. Thus a given set of governing equations dictate the dynamics of the underlying system. These equations construe the *model*. However, an alternative to this approach exists when either the model equations are beyond their range of validity or governing equations simply are not known or well-formulated. In this alternative approach, experimental data itself drives the understanding of the system, without model equations being prescribed. Thus *data-based* algorithms can be constructed to understand, mimic and control the complex system. *Dynamic Mode Decomposition* (DMD) is a relatively new data-based algorithm that requires no underlying governing equations, rather snapshots of experimental measurements are used to predict and control a given system [64, 65, 66, 67]. In atmospheric sciences, a version of the DMD method is used to fit a set of a data to an underlying (best statistical fit) linear stochastic model. This is known as *linear inverse models* (LIMs) [68, 69, 70]. Thus the methodology presented here uses data alone as the source for informing us of the state and dimensionality of the system.

20.1 Theory of Dynamic Mode Decomposition (DMD)

Exploitation of low-dimensionality: that is the overarching goal of data-driven modeling methods. In the case of the technique of dynamic mode decomposition, it is the aim of the method to take advantage of low-dimensionality in the experimental data itself without having to rely on a given set of governing equations.

The DMD method provides a decomposition of experimental data into a set of dynamic modes that are derived from snap shots of the data in time. The mathematics underlying the extraction of dynamic information from time-resolved snapshots is closely related to the idea of the Arnoldi algorithm, one of the workhorses of fast computational solvers. To set the stage, we begin with the data collection process. To important parameters are required:

$$N = \text{number of spatial points saved per time snapshot} \quad (20.1.1a)$$
$$M = \text{number of snapshots taken} \quad (20.1.1b)$$

In this case, it is imperative to collect data at regularly spaced intervals of time

$$\text{data collection times}: \quad t_{m+1} = t_m + \Delta t \quad (20.1.2)$$

where the collection time starts at t_1 and ends at t_M, and the interval between data collection times is Δt.

The data can then be arranged into an $N \times M$ matrix

$$\mathbf{X} = [U(\mathbf{x}, t_1) \ \ U(\mathbf{x}, t_2) \ \ U(\mathbf{x}, t_3) \ \ \cdots \ \ U(\mathbf{x}, t_M)] \tag{20.1.3}$$

where $\mathbf{x}$ is a vector of data collection points of length N. A limitation of the DMD technique, at least applied in a straightforward manner, is apparent from the start: sampling must be done in equally spaced time intervals. Be that as it may, the objective is to mine the data matrix $\mathbf{X}$ for important dynamical information. For the purposes of the DMD method, the following matrix is also defined:

$$\mathbf{X}_j^k = [U(\mathbf{x}, t_j) \ \ U(\mathbf{x}, t_{j+1}) \ \ \cdots \ \ U(\mathbf{x}, t_k)] \tag{20.1.4}$$

Thus this matrix includes columns j through k of the original data matrix.

The DMD method approximates the modes of the so-called *Koopman operator.* The Koopman operator is a linear, infinite-dimentional operator that represents nonlinear, infinite-dimensional dynamics without linearization [71, 72], and is the adjoint of the Perron-Frobenius operator. The method can be viewed as computing, from the experimental data, the eigenvalues and eigenvectors (low-dimensional modes) of a linear model that approximates the underlying dynamics, even if the dynamics is nonlinear. Since the model is assumed to be linear, the decomposition gives the growth rates and frequencies associated with each mode. If the underlying model is linear, then the DMD method recovers the leading eigenvalues and eigenvectors normally computed using standard solution methods for linear differential equations.

Mathematically, the Koopman operator $\mathbf{A}$ is a linear, time-independent operator $\mathbf{A}$ such that

$$\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j \tag{20.1.5}$$

where j indicates the specific data collection time and $\mathbf{A}$ is the linear operator that maps the data from time t_j to t_{j+1}. The vector $\mathbf{x}_j$ is an N-dimensional vector of the data points collected at time j. The computation of the Koopman operator is at the heart of the DMD methodology. As already stated, the mapping over Δ is linear even though the underlying dynamics that generated $\mathbf{x}_j$ may be nonlinear. It should be noted that this is different than linearizing the dynamics.

To construct the appropriate Koopman operator that best represents the data collected, the matrix $\mathbf{X}_1^{M-1}$ is considered:

$$\mathbf{X}_1^{M-1} = [\mathbf{x}_1 \ \ \mathbf{x}_2 \ \ \mathbf{x}_3 \ \ \cdots \ \ \mathbf{x}_{M-1}] \ . \tag{20.1.6}$$

Making use of (20.1.5), this matrix reduces to

$$\mathbf{X}_1^{M-1} = \left[\mathbf{x}_1 \ \ \mathbf{A}\mathbf{x}_1 \ \ \mathbf{A}^2\mathbf{x}_1 \ \ \cdots \ \ \mathbf{A}^{M-2}\mathbf{x}_1\right] \ . \tag{20.1.7}$$

Here is where the DMD method connects to Krylov subspaces and the Arnoldi algorithm. Specifically, the columns of $\mathbf{X}_1^{M-1}$ are each elements in a Krylov space. This matrix attempts to fit the first $M-1$ data collection points using the Koopman operator (matrix) $\mathbf{A}$. In the DMD technique, the final data point $\mathbf{x}_M$ is represented, as best as possible, in terms of this Krylov basis, thus

$$\mathbf{x}_M = \sum_{m=1}^{M-1} b_m \mathbf{x}_m + \mathbf{r} \tag{20.1.8}$$

where the b_m are the coefficients of the Krylov space vectors and $\mathbf{r}$ is the residual (or error) that lies outside (orthogonal to) the Krylov space. Ultimately, this best fit to the data using this DMD procedure will be done in an L^2 sense using a pseudo-inverse.

Before proceeding further, it is at this point that the data matrix $\mathbf{X}_1^{M-1}$ in (20.1.7) should be considered further. In particular, our dimensionality reduction methods look to take advantage of any low-dimensional structures in the data. To exploit this, the SVD of (20.1.7) is computed:

$$\mathbf{X}_1^{M-1} = \mathbf{U\Sigma V}^* \tag{20.1.9}$$

where $*$ denotes the conjugate transpose, $\mathbf{U} \in \mathbb{C}^{N\times K}$, $\mathbf{\Sigma} \in \mathbb{C}^{K\times K}$ and $\mathbf{V} \in \mathbb{C}^{M-1\times K}$. Here K is the reduced SVD's approximation to the rank of $\mathbf{X}_1^{M-1}$. If the data matrix is full rank and the data has no suitable low-dimensional structure, then the DMD method fails immediately. However, if the data matrix can be approximated by a low-rank matrix, then DMD can take advantage of this low dimensional structure to project a future state of the system. Thus once again, the SVD plays the critical role in the methodology.

Armed with the reduction (20.1.9) to (20.1.7), we can return to the results of the Koopman operator and Krylov basis (20.1.8). Specifically, generalizing (20.1.5) to its matrix form yields

$$\mathbf{A}\mathbf{X}_1^{M-1} = \mathbf{X}_2^{M}\,. \tag{20.1.10}$$

But by using (20.1.8), the right hand side of this equation can be written in the form

$$\mathbf{X}_2^{M} = \mathbf{X}_1^{M-1}\mathbf{S} + \mathbf{r}e_{M-1}^* \tag{20.1.11}$$

where e_{M-1} is the $(M-1)$th unit vector and

$$\mathbf{S} = \begin{bmatrix} 0 & \cdots & & 0 & b_1 \\ 1 & \ddots & & 0 & b_2 \\ 0 & \ddots & \ddots & & \vdots \\ & \ddots & \ddots & 0 & b_{M-2} \\ 0 & \cdots & 0 & 1 & b_{M-1} \end{bmatrix}. \tag{20.1.12}$$

Recall that the b_j are the unknown coefficients in (20.1.8).

The key idea now is the observation that the eigenvalues of $\mathbf{S}$ approximate some of the eigenvalues of the unknown Koopman operator $\mathbf{A}$, making the DMD method similar to the Arnoldi algorithm and its approximations to the Ritz eigenvalues. Schmid [64] showed that rather than computing the matrix $\mathbf{S}$ directly, we can instead compute the *lower-rank* matrix

$$\tilde{\mathbf{S}} = \mathbf{U}^* \mathbf{X}_2^M \mathbf{V} \mathbf{\Sigma}^{-1} \tag{20.1.13}$$

which is related to $\mathbf{S}$ via a similarity transformation. Recall that the matrices $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ arise from the SVD reduction of $\mathbf{X}_1^{M-1}$ in (20.1.9).

Consider then the eigenvalue problem associated with $\tilde{\mathbf{S}}$:

$$\tilde{\mathbf{S}}\mathbf{y}_k = \mu_k \mathbf{y}_k \qquad k = 1, 2, \cdots, K \tag{20.1.14}$$

where K is the rank of the approximation we are choosing to make. The eigenvalues μ_k capture the time dynamics of the discrete Koopman map $\mathbf{A}$ as a Δt step is taken forward in time. These eigenvalues and eigenvectors can be related back to the similarity transformed original eigenvalues and eigenvectors of $\mathbf{S}$ in order to construct the DMD modes:

$$\psi_k = \mathbf{U}\mathbf{y}_k \, . \tag{20.1.15}$$

With the low-rank approximations of both the eigenvalues and eigenvectors in hand, the projected future solution can be constructed for all time in the future. By first rewriting for convenience $\omega_k = \ln(\mu_k)/\Delta t$ (recall that the Koopman operator time dynamics is linear), then the approximate solution at all future times, $\mathbf{x}_{DMD}(t)$, is given by

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k(0)\psi_k(\mathbf{x}) \exp(\omega t) = \mathbf{\Psi}\text{diag}(\exp(\omega t)\mathbf{b} \tag{20.1.16}$$

where $b_k(0)$ is the initial amplitude of each mode, $\mathbf{\Psi}$ is the matrix whose columns are the eigenvectors ψ_k, diag(ωt) is a diagonal matrix whose entries are the eigenvalues $\exp(\omega_k t)$, and $\mathbf{b}$ is a vector of the coefficients b_k.

It only remains to compute the initial coefficient values $b_k(0)$. If we consider the initial snapshot $(\mathbf{x}_1)$ at time zero, let's say, then (20.1.16) gives $\mathbf{x}_1 = \mathbf{\Psi}\mathbf{b}$. This generically is not a square matrix so that its solution

$$\mathbf{b} = \mathbf{\Psi}^+ \mathbf{x}_1 \tag{20.1.17}$$

can be found using a pseudo-inverse. Indeed, $\mathbf{\Psi}^+$ denotes the Moore-Penrose pseudo-inverse that can be accessed in MATLAB via the **pinv** command. As already discussed in the compressive sensing section, the pseudo-inverse is equivalent to finding the best solution $\mathbf{b}$ the in the least-squares (best fit) sense. This is equivalent to how DMD modes were derived originally.

Overall then, the DMD algorithm presented here takes advantage of low dimensionality in the data in order to make a low-rank approximation of the linear mapping that best approximates the nonlinear dynamics of the data collected for the system. Once this is done, a prediction of the future state of the system is achieved for all time. Unlike the POD method, which requires solving a low-rank set of dynamical quantities to predict the future state, no additional work is required for the future state prediction outside of plugging in the desired future time into (20.1.16). Thus the advantages of DMD revolve around the fact that (i) no equations are needed, and (ii) the future state is known for all time (of course, provided the DMD approximation holds).

The algorithm is as follows:

(i) Sample data at N prescribed locations M times. The data snapshots should be evenly spaced in time by a fixed Δt. This gives the data matrix $\mathbf{X}$.

(ii) From the data matrix $\mathbf{X}$, construct the two sub-matrices $\mathbf{X}_1^{M-1}$ and $\mathbf{X}_2^{M}$.

(iii) Compute the SVD decomposition of $\mathbf{X}_1^{M-1}$.

(iv) The matrix $\tilde{\mathbf{S}}$ can then be computed and its eigenvalues and eigenvectors found.

(v) Project the initial state of the system onto the DMD modes using the pseudo-inverse.

(vi) Compute the solution at any future time using the DMD modes along with their projection to the initial conditions and the time dynamics computed using the eigenvalue of $\tilde{\mathbf{S}}$.

20.2 Dynamics of DMD versus POD

To illustrate the application of the DMD method, a simple example is chosen in order to highlight the implementation of the algorithm advocated at the end of the last section. In fact, it is the same example used to illustrate the POD reduction technique, namely, the $N = 2$ soliton dynamics of the nonlinear Schrödinger (NLS) equation (See Sec. 19). For completeness, the NLS will be again given here:

$$iu_t + \frac{1}{2}u_{xx} + |u|^2 u = 0 \qquad (20.2.1)$$

with initial conditions $N\text{sech}(x)$. Thus the data to be used will be generated through simulations of this equation. The PDE solution can be computed using FFTs with the following code. First, the initialization of the space and time discretization is computed:

```
% space
L=40; n=512;
x2=linspace(-L/2,L/2,n+1); x=x2(1:n);
k=(2*pi/L)*[0:n/2-1 -n/2:-1].';
% time
slices=20;
t=linspace(0,2*pi,slices+1); dt=t(2)-t(1);
```

Note that the parameter **slices** will be critical in what follows ($M =$ **slices**$+1$). Specifically, this determines the number of data samples taken over the time interval $t \in [0, 2\pi]$. The initial conditions and time-stepping routines are as follows

```
u=2*sech(x).';   % initial conditions
ut=fft(u);
[t,utsol]=ode45('nls_rhs',t,ut,[],k);
for j=1:length(t)
    usol(j,:)=ifft(utsol(j,:));  % bring back to space
end
```

This implementation of the solver was also demonstrated in Sec. 19. Note that a call is made to the right-hand side function **nls_rhs.m** which is the following

```
function nls_rhs=nls_rhs(z,ut,dummy,k)
u=ifft(ut);
nls_rhs= -(i/2)*(k.^2).*ut + i*fft( (abs(u).^2).*u );
```

Having executed the above code, the matrix **usol** results which arranges the simulation data into a matrix with M time slices and N data points at each slice. Thus the matrix **usol** is the desired data matrix $\mathbf{X}$ required in the first step of the DMD algorithm.

The top left panel of Fig. 230 shows the result of the simulation of the NLS equation over the interval $t \in [0, 2\pi]$ with $M = 21$ and $N = 512$. Our goal is to sample from this set of data in order to (i) find a low-dimensional framework to exploit, and (ii) project the future state of the system in time without relying on computations of the PDE.

Our first objective is to then construct the data matrices in the second part of the algorithm: $\mathbf{X}_1^{M-1}$ and $\mathbf{X}_2^{M}$. Thus we have

```
X = usol.'; X1 = X(:,1:end-1); X2 = X(:,2:end);
```

Armed with these subsets of the full data matrix, the SVD decomposition of $\mathbf{X}_1^{M-1}$ can be computed and the eigenvalues and eigenvectors of $\tilde{\mathbf{S}}$ found. The following code performs these operations.

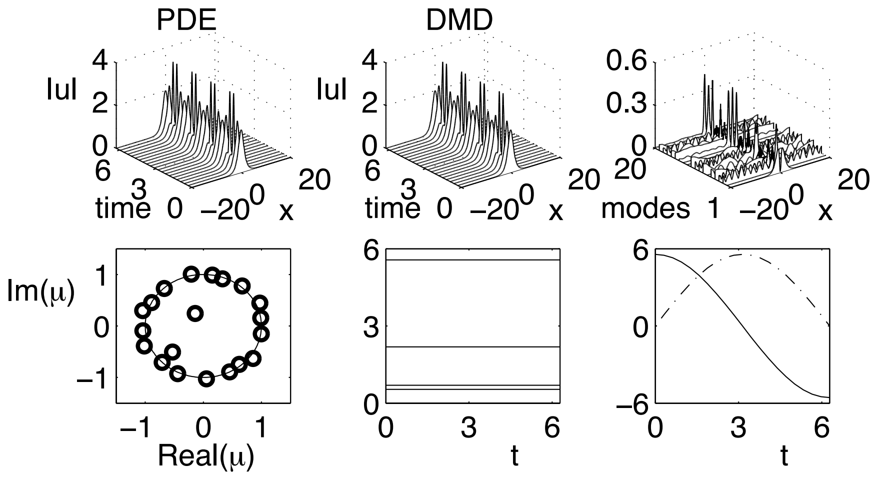


Figure 230: The top left panels represent the full PDE simulation and DMD reconstruction from sampling for $t \in [0, 2\pi]$ and $M = 21$ slices of data. The DMD modes used for the linear reconstruction are shown in the top right panel with its eigenvalue distribution μ_k shown in the bottom left panel. The solid line of the bottom left panel represents the unit circle for which eigenvalues outside of it are growth modes. The bottom right panels show the absolute value of the time dynamics (bottom middle) for each of the 20 DMD modes and the real (solid) and imaginary (dotted) amplitude of the time evolution of the first mode (bottom right).

```
[U,Sigma,V] = svd(X1, 'econ');
S = U'*X2*V*diag(1./diag(Sigma));
[eV,D] = eig(S);
mu = diag(D);
omega = log(mu)/(dt);
Phi = U*eV;
```

Note that the vector **vals** contains the eigenvalues μ of interest to the DMD dynamics and **Phi** contains the DMD modes which are the basis function used to predict the future state of the system.

The final steps in the DMD algorithm both project the initial conditions, via the pseudo-inverse, to the DMD modes and then predicts the future state using the DMD modes and their linear time dynamics dictated by the μ_k. The following gives a reconstruction of the dynamics over the same interval as used in collecting the data.

```
y0 = Phi\u;  % pseudo-inverse initial conditions
```

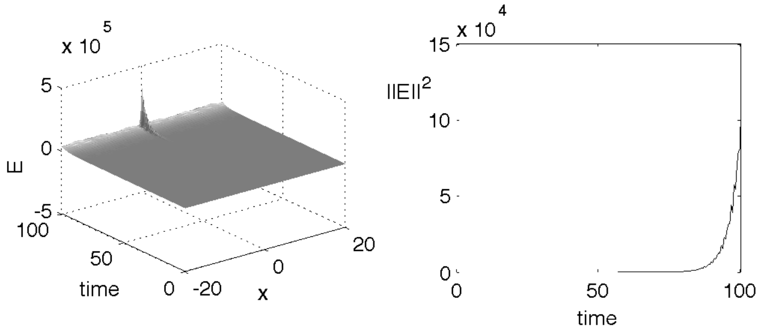


Figure 231: Evolution of the error, i.e. the absolute value of the difference between the DMD solution and the PDE solution, over the time interval $t \in [0, 100]$ (left panel) and the evolution of the norm of the error (right panel). Due to the DMD approximation having eigenvalue outside the unit circle, growth modes appear which blow up for long time projections of the dynamics. However, the error remains small for times less than $t \approx 50$, giving an upper range for when the DMD approximation holds.

```
u_modes = zeros(size(v1,2),length(t));
for iter = 1:length(t)
    u_modes(:,iter) =(y0.*exp(omega*t(iter)));
end
u_dmd = Phi*u_modes;
```

Figure 230 gives a summary of the DMD method, including demonstrating the PDE versus DMD dynamics. In fact, in the top panel of this figure, the PDE (exact solution) and DMD reconstruction are shown. In the top right panel, the 20 DMD modes constructed from the data reduction are illustrated. Thus these modes are used as the basis for reconstructing the best possible linear fit to the nonlinear evolution. Their time evolution is determined by the eigenvalues μ_k which are given in the left bottom panel. Also plotted on this panel is the unit circle. Those eigenvalues that lie outside the unit circle represent growth modes in the system. The absolute value of the the time dynamics for each mode is given in the middle bottom panel while the first mode time dynamics (the real and imaginary parts) is given in the bottom right panel.

Although the $N = 2$ soliton dynamics is oscillatory (nonlinearly) in time, the DMD reduction shows that a few of the eigenvalues μ_k sit slightly outside the unit circle when $M = 21$. This will ultimately lead to a long-term blow up of the predicted solution using the DMD method. Figure 231 shows the error as a function of time. Here, a comparison is made between the DMD predicted solution, using data only in $t \in [0, 2\pi]$, and the full numerical solution for the interval $t \in [0, 100]$. The agreement is quite good until about $t \approx 40$ when the

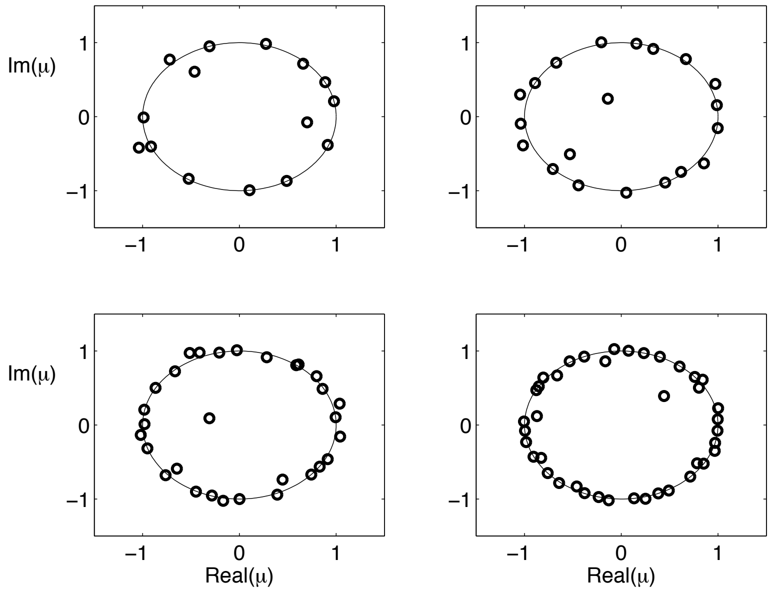


Figure 232: Spectral distribution of the eigenvalues μ_k as a function of the sampling rate over the interval $t \in [0, 2\pi]$. From top left to bottom right, the value of $M = 16, 21, 31$ and 41. Note that for higher sampling, the eigenvalues outside of the unit circle collapse to the unit circle, allowing for longer time dynamical predictions using the DMD modes. The corresponding eigenvectors are shown in Fig. 233.

solution starts to diverge exponentially from the true solution. This is simply a result of sampling error as will be shown in what follows. Either way, it is important to recognize that how one samples and frequently one samples can make a big impact on the DMD method. Moreover, any noise in the system and/or data can push an eigenvalue outside of the unit circle and limit the range (in time) of the DMD algorithm and prediction. Thus to continue to use it in this context, resampling must occur in order to re-project the data and avoid eventual (unphysical) blow up of the DMD predictions.

As already stated, the computation of the eigenvalues μ_k changes as a function of the sampling rate. To illustrate this, two new figures are produced. Figure 232 shows the eigenvalue distribution μ_k for $M = 16, 21, 31$ and 41 when sampling the same PDE dynamics over the interval $t \in [0, 2\pi]$. Note that as the sampling rate gets higher, the eigenvalues outside the unit circle draw closer to the unit circle, thus producing DMD predictions that are valid for a longer window in time since the growth rates of these eigenvalues is smaller. The modes ψ_k associated with these eigenvalues μ_k are shown in Fig. 233 for the corre-

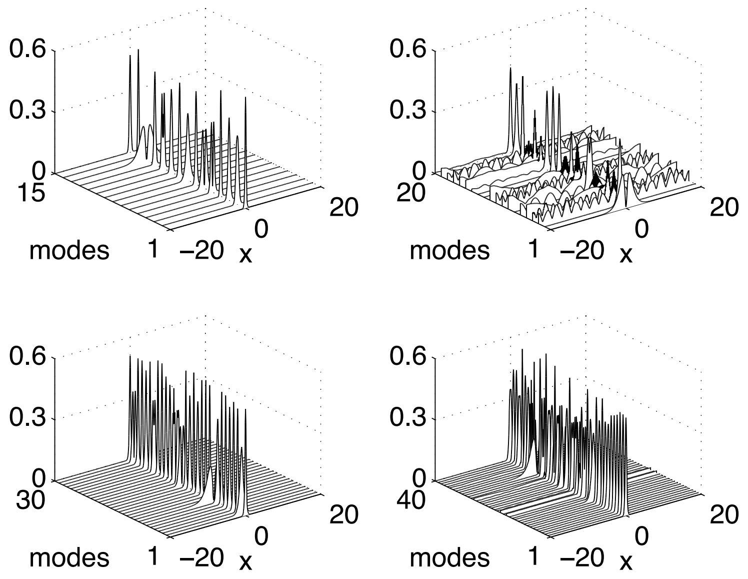


Figure 233: Eigenvectors associated with the eigenvalue distribution μ_k of Fig. 232. For $M = 16, 21, 31$ and 41, there are 15, 20, 30 and 40 eigenvectors generated respectively for reconstruction or approximation of the PDE solution using these DMD basis modes.

sponding number of samplings. These modes are used in linear combination with their time dynamics given by the μ_k to give the approximation to the true PDE dynamics.

An advantage of the DMD algorithm is that it actually can easily identify growth modes of a given physical system. In the example above, the growth modes were spurious. However, in many physical systems, there may indeed be a growth mode that one wishes to suppress. Thus the DMD framework is a natural way to apply the ideas of control theory, i.e. the growth modes are identified and an control algorithm is placed on the complex physical system with the aim of suppressing the pernicious growth mode(s). Such an application might be for the control of turbulent flow fields, for instance, where full simulations not only disagree with experimental observations quantitatively, but where such simulations are prohibitively expensive to do in real time. With the DMD framework, direct measurements can be performed in order to understanding the underlying dynamics and growth modes for short windows of time into the future, thus allowing for control and manipulation of the flow field.

Finally, a comment should be made to conclude the section about the DMD method in comparison with the POD method. In the POD method, a low-dimensional basis is selected from the data matrix **X** by use of the SVD. The dynamics of the governing equations are then projected onto this reduced basis and the resulting nonlinear dynamical system for the mode amplitudes evolved forward in time. With DMD, the dynamics are assumed to be linear so that an exact solution can be constructed. Thus there is no need to evolve a dynamical system forward in time for the mode amplitudes, rather, once the reduction is done and the DMD modes and their eigenvalues computed, then the future state of the system can be predicted without any addition work. Thus the POD method retains the *nonlinear dynamics* of the system through its projection of the low-dimensional modes onto the governing equations. DMD tries to construct the Koopman operator that best approximates the nonlinear dynamics directly without the need of a governing equation or additional evolution of any equation. Both have strengths and weaknesses, so that the the method of choice usually boils down to taking maximal advantage of specific aspects of the problem at hand. But if the equations are unknown, hard to parametrize, overly approximated, etc, then DMD provides a viable method for extracting and potentially controlling some underlying complex system.

20.3 Applications of DMD

The example given in the previous section is nice in the sense that the computations for the PDE can be performed quite easily and an intuitive understanding of the dynamics is already present. Thus the comparison of the POD, DMD and full PDE solution techniques can be made and each of their advantages and disadvantages compared. But just like the POD method considered in Sec. 19, the primary goal is to apply the DMD method to complex physical systems where full advantage can be taken of its strengths. In this section, two examples will be given for how one might use the DMD method for the modeling of physical and/or computational systems.

Reduced Order Integrator using DMD/POD

Modern methods in scientific computing aim to exploit any underlying mathematical structure to make computations more efficient. Adaptive time-stepping schemes for ordinary and partial differential equations are an example of a numerical technique that has continued to have significant impact on almost any field of the mathematical, physical, biological, and engineering sciences. Adaptive time stepping with Runge-Kutta methods are used so universally that they are now an integral component in most commercial and open source scientific computing software packages. Indeed, adaptive time-stepping is the default in every numerical integrator in MATLAB including **ode45**, **ode23**, **ode 113**, etc. The advantage of these adaptive integrators is particularly profound in time-dependent multi-scale physics problems where large steps can be taken in certain regimes but more refinement is needed to accurately capture fast time scales in other regimes.

Motivated by the increase in performance that can be obtained by adapting the integrators, it makes sense to consider hybrid numerical integration schemes that exploit, when possible, dimensionality reduction of a given system. To have maximal impact, the low-dimensional basis must be adaptively selected when the dynamics of the system changes, i.e. and initial low-dimensional basis will be unable to accommodate dynamical changes orthogonal to the reduced basis. The POD method outlined in the last Sec. 19 is a standard method for generating a reduced order model. Improvements to the POD technique include nonlinear methods based on manifold learning techniques [74, 75], balanced POD (BPOD) reductions [76, 77, 78], traveling POD [79], and Trust-Region POD [80] and Sequential POD (SPOD) [81]. for instance. All these extensions to the basic POD methodology are aimed to improve the ability of of the technique to model complex systems and/or to compute its underlying solution branches and bifurcation structure.

Although there are some exceptions, the generation of the POD modes is typically performed in a static and offline fashion where data for the modes are collected prior to generation of the model, and the POD basis itself remains unchanging despite any new incoming information. In order make the numerical integrator more robust while retaining some of the computational benefits of the reduced model, we adaptively switch between the accurate but expensive PDE simulations, to collect data or evolve the solutions in regimes that are high dimensional, and reduced order simulations that, while inexpensive to integrate, require representative data in order to be constructed and lose accuracy should the dynamics of the system travel too far away from the location in phase space where data was obtained.

Here, exploitation is made of the predictive and equation-free nature of the dynamic mode decomposition (DMD) in conjunction with the potentially accurate POD technique [73]. If the predictions of the two are close, then we will continue to use the reduced model. If not, we revert to the full PDE. The result is a robust yet simple integration scheme that can, for appropriate and non-trivial types of system dynamics, retain many of the computational benefits of reduced order models while avoiding spurious results. Such a *dimensionally adaptive* scheme has shown to be successful in reducing computational costs of complex systems [73].

As a specific example, consider application of these ideas to the complex Ginzburg-Landau equation (CQGLE) of mathematical physics.

$$u_t = \left(\frac{i}{2} - \tau\right) u_{xx} + \kappa u_{xxxx} + (i+\beta)|u|^2 u + (i\nu + \sigma)|u|^4 u + \gamma u \qquad (20.3.1)$$

where $\tau, \kappa, \beta, \nu, \sigma$ and γ are related to the specific application from which the equation arises. If these parameters are all zero, then the NLS equation re-

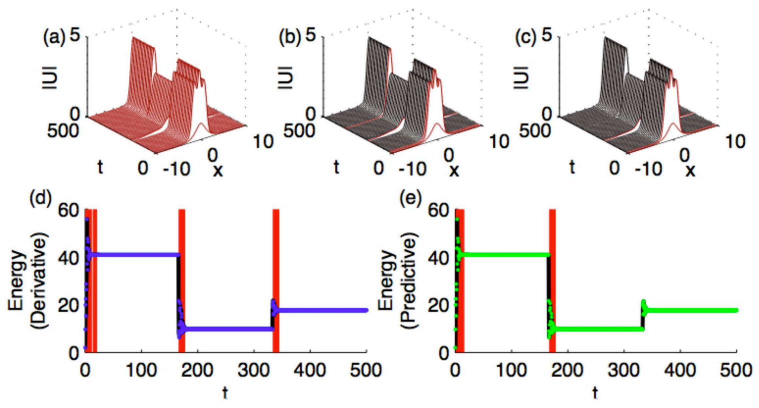


Figure 234: Comparison of the PDE evolution with the two proposed hybrid DMD/POD reductions. The full CQGLE PDE simulations are shown in the top left panel while the two hybrid DMD/POD steppers are shown in the remaining top two panels. The red regions are where full PDE simulations occur while the black region is where low-dimensional POD stepping occurs. The energy (L^2 norm) evolution for the two DMD/POD method are shown in the bottom panel. Note that because of the changes of γ in time, the DMD/POD method must return to the full PDE simulations. Regardless, the method is an order of magnitude faster than full PDE simulations. [from M. Williams, P. Schmid and J. N. Kutz, *Hybrid Reduced Order Integration with the Proper Orthogonal Decomposition and Dynamic Mode Decomposition*, SIAM Journal of Multiscale Modeling and Simulation (submitted) [73] ©SIAM]

sults. The CQGLE can be used, for instance, to model so-called mode-locked lasers [59]. For our purposes, the parameters are chosen to be $\tau = 0.08, \kappa = 0, \beta = 0.66, \nu = -0.1$ and $\sigma = -0.1$ with γ varying in time as a step-function. Specifically, $\gamma = -0.1$ aside from the interval $t \in [500/3, 1000/3]$ where $\tau = -0.2$. This causes the solution transitions and forces the POD/DMD solver back to the full PDE simulations.

The idea is to simulate (20.3.1). Methods for doing so are presented in previous sections of this book, thus it is not a complicated thing to do. However, when observing the resulting dynamics, much of it is low-dimensional in nature as there a variety of *attracting steady-states* in the system. Thus if the evolution becomes low-dimensional during the evolution, the goal is to switch to the appropriate POD basis expansion and simulate the system at a fraction of the cost. Checking for low-dimensional behavior is relatively easily and there are a variety of ways to do so [73]. The more difficult task is to assess wether the low-dimensional simulation needs to revert back to the full PDE simulation. This happens, for instance, when one of many parameters of the CQGLE changes

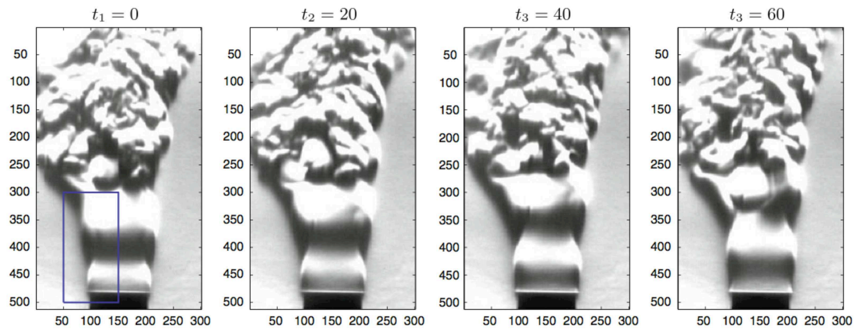


Figure 235: A selection of Schlieren snapshots of a helium fluid jet at Reynolds number $Re = 940$. The blue boxed region on the left panel indicates the data collection region for which time-resolved data is extracted for the DMD reduction. [from P. J. Schmid, L. Li, M. P. Juniper and O. Pust, *Applications of the dynamic mode decomposition*, Theoretical and Computational Fluid Dynamics **25**, 249-259 (2011) [65] ©Springer]

and the POD modes are incapable of representing the new (bifurcated) behavior. Two techniques are developed in Ref. [73] that set criteria for evaluating wether or not to continue in the POD basis or wether to return to the full PDE simulation.

Figure 234 illustrates the hybrid DMD/POD time-stepper that takes advantage of low-dimensionality. Here, the parameter γ is a step-function in time, causing the solution to change steady-state attractors in the evolution. In the top left panel, the full, high-resolution PDE simulation is shown for the time interval $t \in [0, 500]$. The remaining top panels show the results of the DMD/POD stepper for the two different low-dimensional criteria developed in the paper. The red region is where full PDE simulations occur while the black region represents the low-dimensional POD evolution. The bottom panels represent the energy (L^2 norm) in the CQGLE laser cavity for the two methods proposed. As can be seen, aside from the transition regions, the dynamics can almost always be represented in a low-dimensional POD basis, thus allowing for significant computational savings. Indeed, the simulation times drop by an order of magnitude in comparison with full PDE simulations [73].

DMD for a fluid jet

The previous example tied together the two concepts of DMD and POD. In this second example, DMD is used to extract the modal structure of a complex flow field and predict the dynamics of the modes. In this case, no governing equations, or underlying model, is assumed. Thus it is a purely data driven approach to the problem. Details of the experiment are found in Ref. [65].

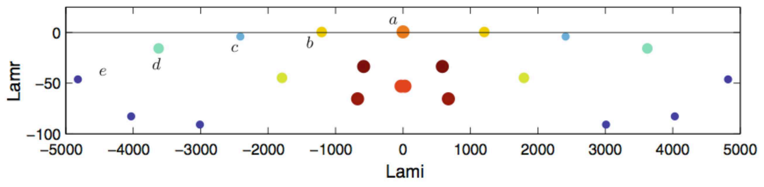


Figure 236: DMD spectrum for the flow field using Schlieren snapshots. Both the real and imaginary parts of the eigenvalue are shown along with a colored ball of a given size. The marker size indicates the spatial coherence of the associated DMD modes. Thus larger balls correspond to large-scale structures in the fluid flow while smaller balls correspond to small-scale structures. [from P. J. Schmid, L. Li, M. P. Juniper and O. Pust, *Applications of the dynamic mode decomposition*, Theoretical and Computational Fluid Dynamics **25**, 249-259 (2011) [65] ©Springer]

Basically, data for the flow of a helium jet is visualized using Schlieren snapshots in time. Figure 235 shows the visualization of the flow field. This constitutes the data. Note that only a small region of the flow field will be used to extract modal structures and their dynamics.

Once the data collection process is finished, the data matrix $\mathbf{X}$ can be constructed and the DMD algorithm as outlined in the previous section can be implemented. Schmid *et al* [65] uses a slightly modified implementation of the algorithm. Specifically, once the data preparation is accomplished and the two matrices $\mathbf{X}_1^{M-1}$ and $\mathbf{X}_2^M$ constructed, a QR-algorithm is used instead of the SVD. These, of course, are closely related and ultimately give the same thing, i.e. they both generate the DMD modes and their eigenvalues μ_k.

Figure 236 shows the DMD spectrum for the helium jet for the Schlieren snapshots of the data. Both the real and imaginary parts of the eigenvalue are shown along with a colored ball of a given size. The marker size indicates the spatial coherence of the associated DMD modes. Thus larger balls correspond to large-scale structures in the fluid flow while smaller balls correspond to small-scale structures. The DMD spectrum shows the typical spectrum for an advection-diffusion dominated flow. Specifically, the eigenvalues show a strong alignment along the horizontal neutral stability line, but are overall damped. An eigenvalue at the origin (labelled a) is present which represents the steady state; its corresponding eigenvector displays the mean flow field. This feature is also present in a POD analysis of the same flow field: the first, most dominant, right singular vector of the snapshot basis consists of the mean flow. The least stable modes then fall on parabolic arcs, denoted by b through e. Since over the temporal observation period the flow is in an equilibrated state, we do not expect and observe any unstable eigenvalue. For even longer observation periods and data sequences, the damping rates of the dominant dynamic

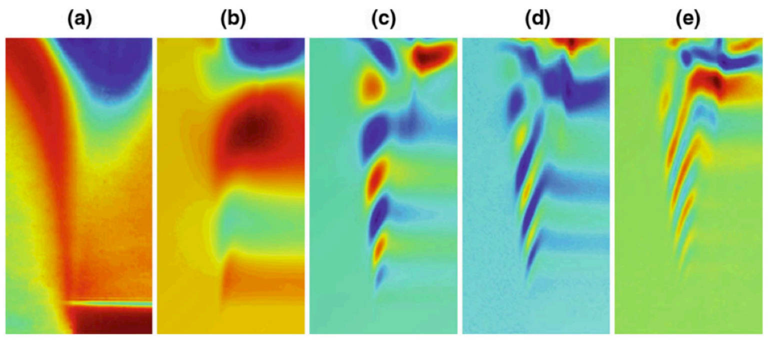


Figure 237: Five DMD modes generated from Schlieren snapshots. These five modes correspond to the eigenvalues a through e in Fig. 236. Note that the modes show both large-scale and small-scale structures in the flow field. [from P. J. Schmid, L. Li, M. P. Juniper and O. Pust, *Applications of the dynamic mode decomposition*, Theoretical and Computational Fluid Dynamics **25**, 249-259 (2011) [65] ©Springer]

modes will approach the neutral line. The color coding and symbol size of the eigenvalues represent a measure of coherence of the associated modes (similar to the energy content of POD modes) and help in the separation of relevant structures from noise-contaminated ones. This coherence measure is computed by projecting individual dynamic modes onto energy-ranked proper orthogonal modes (except the mean flow); the resulting coefficients of this projection give a ratio of large-scale to fine-scale structures. This feature adds information about spatial coherence and scales to the temporal information contained in the spectrum. Figure 237 presents five dynamic modes associated with the labeled eigenvalues in Fig. 236. As mentioned before the mode corresponding to label a represents the mean flow, while the higher modes b through e increasingly show the presence of small-scale (but coherent) structures. The support of the dynamic modes is clearly located at the diverging outer edge of the jet where vortex roll-up, mixing and entrainment processes are prevalent. The combination of the spectral content and DMD mode structures allows for prediction of the future state of the system. Further, if one were interested in flow control, such predictions could be used in a control algorithm for controlling the fluid jet nozzle or other parameters in the physical system. Much like the POD method, the DMD reduction highlights the key features for consideration in the complex system of interest. It is yet another tool that allows for understanding of a given system.

21 Data Assimilation Methods

Most of the data-driven techniques presented in this book were applied to systems where the underlying governing equations were prescribed. However, in the DMD method (or in the equation-free method), no governing equations were required to extract meaningful information about the dynamics of the complex system under consideration. The method of *data assimilation* is a hybrid method that uses both data measurements collected in time about the system in conjunction with a prescribed set of governing equations. The fact is, both the measurements and simulations are in practice heavily influenced by uncertainty and/or noise fluctuations. Thus neither the experiment or theoretical model can be fully trusted. However, combining the two so that the experimental measurements helps inform the model and vice-versa can greatly improve the predictive powers of the model, or analysis of the state of the system [83, 84, 85, 86]. Thus data is assimilated into the model predictions and hence the name. Data assimilation is potentially one of the most useful data-driven modeling techniques as we are rarely without some underlying governing equations or without experimental measurements. And to make optimal use of both, data assimilation techniques are ideal.

21.1 Theory of Data Assimilation

To begin thinking about data assimilation, we will first consider a given complex system and its underlying dynamical evolution. Specifically, let us begin by assuming that the system under consideration has the following evolution equation

$$\frac{d\mathbf{y}}{dt} = f(t, \mathbf{y}) \tag{21.1.1}$$

with the initial state of the system given by

$$\mathbf{y}(0) = \mathbf{y}_0 \, . \tag{21.1.2}$$

Of course, techniques for solving such a system have been considered extensively throughout this book. Indeed, provided that $f(t, \mathbf{y})$ is well behaved, i.e. continuous and differentiable, for instance, then a solution to the prescribed problem can be solved for uniquely. In fact, for an N-dimensional vector $\mathbf{y}$, N initial conditions are prescribed by $\mathbf{y}_0$ so that the number of unknowns and constraints match.

Up to this point, we have ignored and/or denied a simple and obvious truth regarding the evolution equation (21.1.1) and its initial conditions (21.1.2). Specifically, we are assuming that we can perfectly prescribe both! In practice, this is surely an impossible task. And for strongly nonlinear systems (21.1.1), even small changes in the initial conditions and/or slight noise perturbations to the system can lead to large changes in the dynamical behavior, stability and predictability of (21.1.1). To be more precise then about our formulation of the true system, (21.1.1) and (21.1.2) should be modified to

$$\frac{d\mathbf{y}}{dt} = f(t, \mathbf{y}) + \mathbf{q}_1(t) \tag{21.1.3}$$

with the initial conditions

$$\mathbf{y}(0) = \mathbf{y}_0 + \mathbf{q}_2 \,. \tag{21.1.4}$$

where $\mathbf{q}_1$ represents unknown model errors due to either noise fluctuations in the system or perhaps truncation of higher-order effects in the system that are thought to be negligible. Similarly, the vector $\mathbf{q}_2$ represents the error in the prescribed initial conditions either because we cannot accurately measure them in practice or prescribe them in a realistic physical system.

Even in the presence of the perturbations $\mathbf{q}_1$ and $\mathbf{q}_2$, the system of equations (21.1.3) and (21.1.4) remains well-posed with a unique solution specifying the dynamics in time. However, for a strongly nonlinear system, the presence of $\mathbf{q}_1$ and $\mathbf{q}_2$ make it virtually impossible to use the governing equations (21.1.3) and the initial conditions (21.1.4) for an accurate prediction of the future state of the system. This is largely due to the concept of *sensitivity to initial conditions* that is displayed in many complex systems. Of course, this then gives rise to the following fundamental question: Is modeling a worthwhile exercise if it cannot predict the future state of a realistic system? In practice, great engineering is often about eliminating large unknowns in a system by essentially suppressing, as much as possible, the vectors $\mathbf{q}_1$ and $\mathbf{q}_2$. But for highly complex systems, such as climate modeling and weather prediction, the system simply cannot be engineered and one must find effective strategies for dealing with the inherent effect of $\mathbf{q}_1$ and $\mathbf{q}_2$.

Data assimilation is a technique that attempts to mitigate the problems associated with having $\mathbf{q}_1$ and $\mathbf{q}_2$ present in a given system. As the name suggests, the idea is to assimilate experimental measurements directly into the theoretical model in order to inform the dynamics. Thus a set of measurements are taken so that

$$g(t, \mathbf{y}) + \mathbf{q}_3 = 0 \tag{21.1.5}$$

where $g(t, \mathbf{y})$ is a certain set of measurements, let's say M of them, on some quantities related to the state vector $\mathbf{y}$, and the vector $\mathbf{q}_3$ is the error measurement associated with the data collection process.

In principle, it is a great idea to incorporate real experimental measurements into the modeling process. In practice, the addition of (21.1.5) now makes for an overdetermined system (N unknowns and $N + M$ constraints) for $\mathbf{y}$ for which no solution exists in general. Thus the tradeoff of using the experimental data is that we went from a well-posed system with a unique solution to an overdetermined system with no general solution.

To deal with the overdetermined system, the following *quadratic form* is introduced:

$$J(\mathbf{y}) = \int_0^T \int_0^T \mathbf{q}_1^T(t_1)\mathbf{W}_1\mathbf{q}_1(t_2)dt_1dt_2 + \mathbf{q}_2^T\mathbf{W}_2\mathbf{q}_2 + \mathbf{q}_3^T\mathbf{W}_3\mathbf{q}_3 \qquad (21.1.6)$$

where the error vectors $\mathbf{q}_j$ are all directly included in the quadratic form. The matrices $\mathbf{W}_j$ are the inverse of the error covariance for the model, initial conditions and measurements respectively. The introduction of such a quadratic form is motivated by one primary purpose: optimization. In particular, one potential solution to the overdetermined system is to find the solution that minimizes the weighted error, weighted with respect to the model, initial condition and measurement error, as given by the quadratic form $J(\mathbf{y})$. Since it is a quadratic form, standard convex optimization methods can be directly applied to find a solution. The least-square error model defined by $J(\mathbf{y})$ is only one potential choice. However, this choice is particularly attractive when considering Gaussian statistics for the error vectors. In this case, minimizing $J(\mathbf{y})$ is equivalent to maximizing the probability density function $P(\mathbf{y}) = C\exp[-J(\mathbf{y})]$. Thus the minimum of (21.1.6) is also the maximum-likelihood estimate.

Thus to summarize the data assimilation method, we consider a theoretical model under the influence of some error (21.1.3) subject to initial conditions which are also subject to error (21.1.4). A number of experimental measurements (21.1.5), also subject to error, are then made to inform the model. The combined errors of the resulting overdetermined system is cast as a quadratic form for which convex optimization techniques can be used to find the best fit solution (in the L^2 sense). As a result, the experimental measurements are assimilated with the model predictions to generate better predictions of the dynamical behavior in time. The development of these key ideas will be carried forward in the next two sections.

Data Assimilation for a Single Random Variable

To illustrate the ideas of data assimilation, the simplest toy example will be considered (See, for example, the nice arguments by Holton and Hakim [82] in the context of atmospheric sciences). Specifically, consider a model that generates some prediction about the state of the system, thus effectively $\mathbf{q}_1$ and $\mathbf{q}_2$ would be combined in this model prediction process. Also, consider some experimental measurements on the systems that are also subject to error. Thus for the variable x, there are two predictions about its true value: one from the model and one from experiment. The idea is to combine these two measurements to arrive at a better prediction of the true value of x.

To begin this calculations, consider the following conditional probability statement from Bayes formula (See Sec. 12.2):

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \qquad (21.1.7)$$

where $p(x)$ represents the probability density for predicting the variable x. Here, y will represent the observation (experimental assimilation) that will be made. Thus $p(x|y)$ is the probability of finding x conditioned on having measured y. At this point, we will not concern ourselves with $p(y)$ since it will simply represent a scaling factor for (21.1.7). The probability density function $p(y|x)$ is a *likelihood function* since it is a function of the variable x.

As is the case with almost all problems in probability theory, great simplifications can be made by assuming Gaussian distributed random variables. In fact, our treatment of the data assimilation problem for higher dimensional problems relies explicitly on this assumption in order to derive something tractable. It also helps simplify the current one variable problem if this assumption is made. Thus consider the following probability density distributions

$$p(y|x) = c_1 \exp\left[-\frac{1}{2}\left(\frac{y-x}{\sigma_y}\right)^2\right] \tag{21.1.8}$$

$$p(x) = c_2 \exp\left[-\frac{1}{2}\left(\frac{x-x_0}{\sigma_0}\right)^2\right] \tag{21.1.9}$$

where σ_y is the error variance for the observation, and x_0, σ_0 are the predicted model mean and its associated error variance. The constants c_1 and c_2 are normalization constants ensuring that probability density functions integrate to unity. Thus the errors, both in measurement (σ_y) and theory (σ_0), are characterized by the variance parameters. Note that without any data assimilation, the model would predict the value x_0. Data assimilation attempts to give a correction to this value using the measurement data.

Using the conditional probability statement, which integrates in information about the observation y, along with the assumed Gaussian probability distributions (21.1.9) yields the following prediction (what is x given observation y) for the state of the system

$$p(x|y) = c_3 \exp\left[-\frac{1}{2}\left(\frac{y-x}{\sigma_y}\right)^2\right] \exp\left[-\frac{1}{2}\left(\frac{x-x_0}{\sigma_0}\right)^2\right] \tag{21.1.10}$$

where c_3 is another constant used for convenience.

Our goal now is to construct a quadratic form like (21.1.6) for this problem in order to determine how to modify x from its default prediction value of x_0 in light of our observation y. A very simple quadratic form to construct from the Gaussian distributions is as follows

$$J(x) = -\log[p(x|y)] + \log(c_3) = \frac{1}{2}\left(\frac{y-x}{\sigma_y}\right)^2 + \frac{1}{2}\left(\frac{x-x_0}{\sigma_0}\right)^2 . \tag{21.1.11}$$

But this quadratic form can be easily minimized as it is simply a parabola in one dimension, i.e. optimization can be performed in closed form unlike the

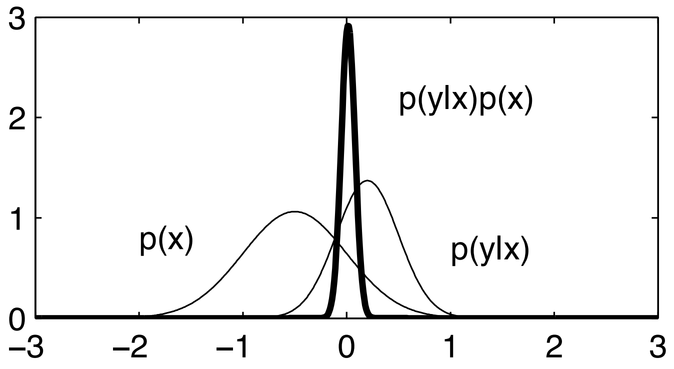


Figure 238: Distributions of the model (left Gaussian, $p(x)$), the observation (right Gaussian, $p(y|x)$), and the data assimilated distribution $p(\bar{x}) = p(y|x)p(x)$. Note that the error variance of the assimilated distribution is much narrower than either the model or observational data as shown in (21.1.13).

general formulation (21.1.6). To find its minimum, $dJ(\bar{x})/dx = 0$ is computed. This yields the following value for $\bar{x}$ at the minimum

$$\bar{x} = \left(\frac{\sigma_y^2}{\sigma_y^2 + \sigma_0^2}\right) x_0 + \left(\frac{\sigma_0^2}{\sigma_y^2 + \sigma_0^2}\right) y\,. \tag{21.1.12}$$

Thus $\bar{x}$, which is a weighted linear superposition of the model prediction x_0 and the observation y, is the new and improved prediction for the outcome of x given the observation y. Such is change of value when using the conditional probability argument.

Equation (21.1.12) has some very intuitive features. First and foremost: if there is no error in the experimental observation, then $\sigma_y = 0$ and $\bar{x} = y$. This is a statement of self-consistency essentially. It would be very bad if the data assimilation method failed to choose the observational data value if it was error free. The error variance for $\bar{x}$ can also be computed to be

$$\bar{\sigma}^2 = \frac{\sigma_0^2}{1 + (\sigma_0^2/\sigma_y^2)} = \frac{\sigma_y^2}{1 + (\sigma_y^2/\sigma_0^2)} < \sigma_0^2, \sigma_y^2\,. \tag{21.1.13}$$

This is also a self-consistency check. Namely, the error in the data assimilation prediction is always better than the error of either the model alone or observation alone. Again, the fact that you are using the combination of model and data should always improve your predictions and error.

Figure 238 shows the three probability densities of interest. In particular, what is shown is the probability density given by the model, $p(x)$, the observation, $p(y|x)$ and the data assimilation, $p(\bar{x}) = p(y|x)p(x)$. For this figure the following were assumed $x_0 = -0.5$, $\sigma_0 = 0.5$, $y = 0.2$ and $\sigma_y = 0.3$. This allows us to compute the distribution of the assimilated prediction through (21.1.12) and (21.1.13). Note that the variance of the assimilated distribution is quite narrow and shifted strongly towards the observational distribution. This is largely because the observation has a much smaller variance than the model data.

Another way to express these results if by noting that

$$\bar{x} = x_0 + K(y - x_0) \tag{21.1.14}$$

with $\bar{\sigma}^2 = (1-K)\sigma_0^2$ and where

$$K = \frac{\sigma_0^2}{\sigma_0^2 + \sigma_y^2} \leq 1\,. \tag{21.1.15}$$

The second term in the right hand side of (21.1.14), $K(y - x_0)$, is the so-called *innovation* since it brings in new information (an observation) to the prediction of x. The predicted value of x is a linear combination of its model prediction x_0 and the innovation. If there is no innovation, i.e. no new information, then the prediction remains x_0. The parameter $0 < K < 1$ is the gain weighting factor and is essentially the so-called *Kalman filter* or *Kalman gain*. This will be considered in more detail in the next section.

21.2 Data Assimilation, Sampling and Kalman Filtering

The preceding section outlined the high level view of the data assimilation method and showed how to implement it on the simplest example possible. What is desired now is to develop the theory further for implementation in realistic systems. Of particular note will be the role of the *Kalman Filter*, or as already hinted at in the last section, the method of incorporating *innovation* to the model predictions.

Relating Observations to the State Vector

Before proceeding to derive the Kalman filter for a general vector system, we address the issue of how observations (21.1.5) project onto the state variable $\mathbf{y}$ of the governing equation (21.1.1). For instance, the governing equation (21.1.1) may be the simulation of some underlying PDE system that is solved on a rectangular grid of a prescribed domain. Observations, however, may be made anywhere in the domain and will, in general, not be aligned with the grid used for (21.1.1). In weather prediction, the data is collected at observations points which may be on the coast, on top of mountains and/or metropolitan areas. Certainly the observation points are not aligned on a regular grid. The simulation of a geographic region, however, is most likely accomplished using a given discretization, perhaps in tens of meters to kilometers. The question is how to overlay the irregularly spaced observations onto the regularly spaced state variables. The simplest thing to do is to use a linear interpolation algorithm to generate values of the state variables at each grid point of the model.

The mapping of the experimental observations can be accomplished mathematically in the following way

$$\mathbf{y}(t) = \mathbf{H}\mathbf{x}(t) + \mathbf{q}_3 \tag{21.2.1}$$

where $\mathbf{y}(t)$ are the observations at a given time t of the state vector $\mathbf{x}$, $\mathbf{H}$ is a matrix that maps the state vector to the observations and $\mathbf{q}_3$ is the observational error. This is a linear version of the more general form (21.1.5). This step is always necessary once a data collection has been applied at a time t.

The One-Dimensional Kalman Filter

The derivation of the projection operator $\mathbf{H}$ in (21.2.1) is necessary for deriving the Kalman Filter. To start the derivation process, we once again return to one-dimensional considerations and a highly idealized version of the dynamics in discrete form. In particular, following Miller [84], we can consider the mapping from a time t_k to a time t_{k+1}. The full dynamics, without approximation, is assumed to be given by

$$x_{k+1} = f(x_k) + q_{k+1} \tag{21.2.2}$$

where x_{k+1} and x_k are the state of the one dimensional system at time t_{k+1} and t_k respectively, and q_{k+1} is a Gaussian white noise sequence. The model approximation to this system is given by

$$x_{0_{k+1}} = f\left(x_{0_k}\right) \tag{21.2.3}$$

where x_{0_k} is the best estimate of the state of the system at time t_k, sometimes called the *analysis*, and $x_{0_{k+1}}$ is the forecast of the dynamics using this estimate at time t_{k+1}.

The error between the truth and the forecast at time t_{k+1} is then given by

$$x_{k+1} - x_{0_{k+1}} = f(x_k) - f\left(x_{0_k}\right) + q_{k+1} \tag{21.2.4}$$

By Taylor expanding $f(x_k)$ around x_{0_k}, the above expression can be written as follows

$$\begin{aligned} x_{k+1} - x_{0_{k+1}} &= (x_k - x_{0_k})f'(x_{0_k}) + \frac{1}{2}(x_k - x_{0_k})^2 f''(x_{0_k}) \\ &+ \frac{1}{6}(x_k - x_{0_k})^3 f'''(x_{0_k}) + \cdots + q_{k+1} \end{aligned} \tag{21.2.5}$$

To find the error variance between the correct solution x_{k+1} and our prediction $x_{0_{k+1}}$, the expectation value of the quantity $(x_{k+1} - x_{0_{k+1}})^2$ is computed. Denoting this as $E[(x_{k+1} - x_{0_{k+1}})^2]$, we find by squaring the above expression and taking the expectation

$$E[(x_{k+1} - x_{0_{k+1}})^2] = E[(x_k - x_{0_k})^2](f'(x_{0_k}))^2 + \text{h.o.t} + E[q_{k+1}^2] \tag{21.2.6}$$

where h.o.t. denotes all higher-order moment terms, i.e. the skewness and kurtosis, for instance, of the error. Keeping these terms represents a closure problem for the error. However, as a first approximation, it is often the case that the higher-order moments are neglected. Thus we can discard them from the calculation at this point. However, it should be noted that there is a great deal of work that investigates the effect of preserving the higher-order moments in the calculation. One might imagine that the more information that is retained, the better the approximation should work.

To simplify the notation, we define the following two quantities

$$P_{k+1} = E[(x_{k+1} - x_{0_{k+1}})^2] \tag{21.2.7a}$$

$$P_k = E[(x_k - x_{0_k})^2] \tag{21.2.7b}$$

which are measures of the error variance between the true solution and the prediction at t_{k+1}, and the true solution and our best estimate of it at t_k respectively. This notation gives

$$P_{k+1} = P_k(f'(x_{0_k}))^2 + E[q_{k+1}^2] \tag{21.2.8}$$

Note that P_{k+1} accounts for the dynamics (and errors associated with it) while P_k accounts for errors in estimating the initial state.

To make a data assimilated prediction, $\bar{x}_{k+1}$, of the state of the system at time t_{k+1} using an observation y_{k+1}, we then apply the following formula

$$\bar{x}_{k+1} = x_{0_{k+1}} + K_{k+1}(y_{k+1} - x_{0_{k+1}}) \tag{21.2.9}$$

where the innovation is again given by the quantity $y_{k+1} - x_{0_{k+1}}$ and the Kalman gain K_{k+1} is given by

$$K_{k+1} = \frac{P_{k+1}}{P_{k+1} + R} \tag{21.2.10}$$

where R is the observation error variance. If there is no observational error, then $R = 0$ which forces $K_{k+1} = 1$. This in turn gives the assimilated prediction to be $\bar{x}_{k+1} = y_{k+1}$, i.e. the assimilated prediction is exactly the error free experimental observation. The variance associated with the error of our prediction $\bar{x}_{k+1}$ is given by

$$\bar{P}_{k+1} = (1 - K_{k+1})P_{k+1} \tag{21.2.11}$$

Taken together, the data assimilation results, which now directly tie in the dynamics of the system through (21.2.2), define what is called the *extended Kalman filter* (EKF).

The Vector EKF

The vector version of the above one-dimensional argument follows in a straightforward manner [84, 87]. The dynamics given by (21.2.2) and (21.2.3) are now written

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k) + \mathbf{q}_{k+1} \tag{21.2.12}$$

and

$$\mathbf{x}_{0_{k+1}} = f(\mathbf{x}_{0_k}) \tag{21.2.13}$$

respectively. Once again Taylor expanding now yields the covariance evolution

$$\mathbf{P}_{k+1} = \mathbf{J}(\mathbf{f})\mathbf{P}_k\mathbf{J}(\mathbf{f})^T + \mathbf{Q} \tag{21.2.14}$$

where $\mathbf{J}(\mathbf{f})$ is the Jacobian generated form the multi-dimensional Taylor expansion and higher-order moments have been neglected. This formula is equivalent to (21.2.8).

Given a data observation $\mathbf{y}_{k+1}$ at time t_{k+1}, the vector case requires an appropriate mapping of the observation space to the state space. But this was already discussed and is mathematically transcribed in (21.2.1). Finally, the data assimilated prediction of the correct state is given by

$$\bar{\mathbf{x}}_{k+1} = \mathbf{x}_{0_{k+1}} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{H}\mathbf{x}_{0_{k+1}}) \tag{21.2.15}$$

where $\mathbf{K}_{k+1}$ is the Kalman gain matrix given by

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k+1}\mathbf{H}^T + \mathbf{R})^{-1} \tag{21.2.16}$$

and $\mathbf{R}$ is the noise covariance matrix. The error covariance of the updated state vector is given by

$$\bar{\mathbf{P}}_{k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1} \tag{21.2.17}$$

where $\mathbf{I}$ is the identity matrix. In vector form, the innovation is given by $\mathbf{y}_{k+1} - \mathbf{H}\mathbf{x}_{0_{k+1}}$. Thus the matrix $\mathbf{H}$ serves an important role in overlaying the data measurement locations with the underlying grid used for computationally evolving forward the model dynamics. This is the EKF for generic complex systems.

With the EKF now established, its strengths and weaknesses are briefly considered. Its strengths are obvious, the EKF provides a systematic way to integrate observational data into the modeling process, thus improving the model predictions. Its most obvious drawbacks are computational. Specifically, for complex systems where the state vector is defined by potentially millions or billions of variables on a large computational grid, computing the innovation becomes unwieldy, especially as these large matrices need to be inverted and Jacobians found. Such enormous computational expense can render the method useless from a practical point of view. One potential way to deal with such computational complexity is to minimize (21.1.6) directly using, for instance, gradient descent algorithms. This is ultimately faster than computing Jacobians and inverses of the large matrices under consideration. Another potential way to solve the problem is to use Ensemble Kalman Filtering (EnKF) techniques which render the problem tractable by processing observations one at a time or by breaking the domain into smaller subdomains where the matrices are tractable. Such computational considerations are necessary to consider in the types of systems (weather prediction and climate modeling [82]) where data assimilation has becomes a standard method of analysis.

21.3 Data Assimilation for the Lorenz Equation

To demonstrate the data assimilation algorithm in practice, consideration will be given to the Lorenz equations

$$x' = \sigma(y - x) \tag{21.3.1a}$$
$$y' = rx - y - xz \tag{21.3.1b}$$
$$z' = xy - bz \tag{21.3.1c}$$

Thus the vector $\mathbf{x} = [x\ y\ z]^T$ is the three-degree of freedom state vector whose nonlinear evolution $f(\mathbf{x})$ is specified by the right hand side (Lorenz model) in (21.3.1) . The Lorenz equations are a highly simplified model of convective-driven atmospheric motion (See Sec. 24.3 for a derivation of the system).

To begin, a perfect simulation will be performed of the Lorenz system, i.e. a simulation that includes specified initial conditions without noise and an evolution which is free of stochastic forcing. This simulation will be the *truth* that the data assimilation will try to reproduce. The parameters to be simulated here, and in what follows, are standard in many examples: $\sigma = 10$, $b = 8/3$ and $r = 28$. The large value of r puts the system in a dynamical state that is highly sensitive to initial conditions. The perfect initial conditions will be $\mathbf{x}_0 = [5\ \ 5\ 5]^T$. The following MATLAB code solves this problem for the time domain $t \in [0, 20]$ and plots it in a parametric way in 3D.

```
t=0:0.01:20;
sigma=10; b=8/3; r=28;

x0=[5 5 5];
[t,xsol]=ode45('lor_rhs',t,x0,[],sigma,b,r)

x_true=xsol(:,1); y_true=xsol(:,2); z_true=xsol(:,3);
figure(1), plot3(x_true,y_true,z_true)
```

The right-hand side of the differential equation includes the dynamics through $f(\mathbf{x})$. In this case, the function **lor_rhs** is given by

```
function rhs=lor_rhs(t,x,dummy,sigma,b,r)
rhs=[sigma*(-x(1)+x(2))
    -x(1)*x(3)+r*x(1)-x(2)
    x(1)*x(2)-b*x(3)];
```

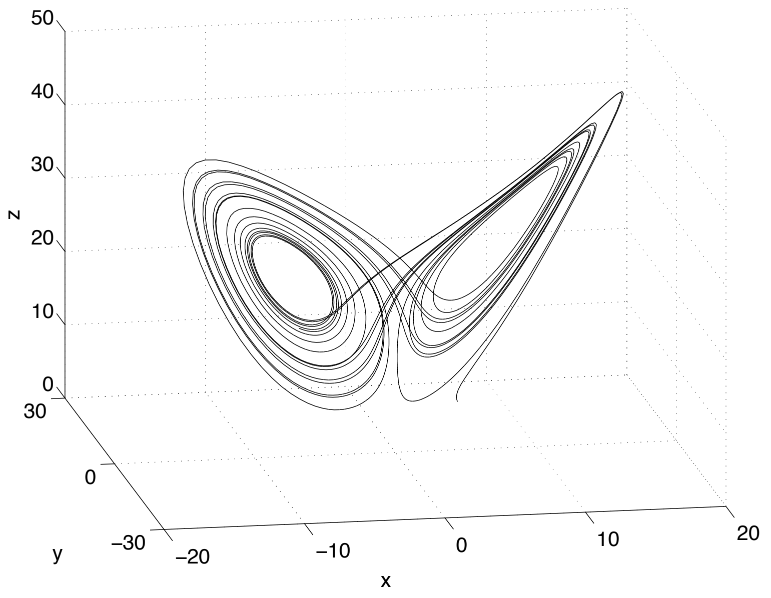


Figure 239: Evolution of the variables $x(t)$, $y(t)$ and $z(t)$ over the time period $t \in [0, 20]$ for $\sigma = 10$, $b = 8/3$ and $r = 28$. The evolution is shown parametrically as a function of time.

The results of simulating the Lorenz equation are shown in Figs. 239 and 240. The first of these figures demonstrates the standard *butterfly* pattern of evolution of the strange attractor in three dimensions. In this graph, time is a parametric quantity. The second of these figures illustrates the evolution of the variables $x(t)$, $y(t)$ and $z(t)$ over the time period $t \in [0, 20]$. In what follows, instead of tracking and comparing all the variables, we will focus on $x(t)$ for illustrative purposes only.

Sensitivity to initial conditions

The first thing that will be investigated is sensitivity of the evolution to small changes in the initial conditions. The mathematical statement of this problem is given in (21.1.2). Thus the effect of $\mathbf{q}_2$ on the dynamics will be considered. And in particular, it is the perturbation of the initial conditions that compromises the predictive power of the theoretical model. To simplify this, we will assume that the error has a Gaussian distribution so that

$$\mathbf{x}(0) = \mathbf{x}_0 + \sigma_2 \mathbf{q}(0, 1) \tag{21.3.2}$$

where $\mathbf{x}_0$ is the perfect initial conditions and $\mathbf{q}(0, 1)$ is a Gaussian distributed random variable with mean zero and unit variance. Thus σ_2 is chosen to make

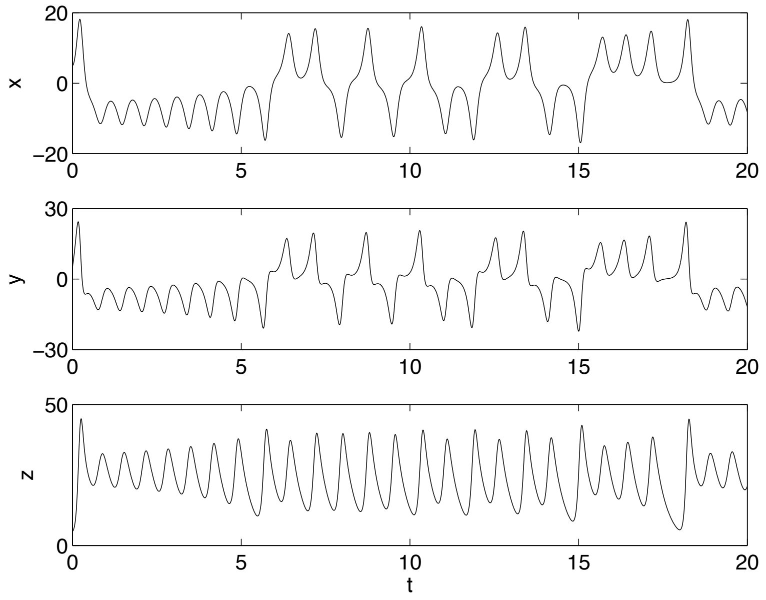


Figure 240: Time evolution of the variables $x(t)$, $y(t)$ and $z(t)$ over the time period $t \in [0, 20]$ for $\sigma = 10$, $b = 8/3$, $r = 28$ and $\mathbf{x}_0 = [5\ 5\ 5]^T$. In this parameter regime, specifically for this large a value of r, the dynamics of the Lorenz equations are highly sensitive to initial conditions.

the error variance either larger or smaller.

With this error, the evolution of (21.3.1) can once again be explored. In Fig. 241, eight realizations of the evolution are given using the initial conditions (21.3.2). The exact evolution which we are trying to model (with $\sigma_2 = 0$) is the thinner solid line while the initial conditions with error (with $\sigma_2 = 1$) is the bolded line. The following MATLAB code generates the eight realizations:

```
sigma2=1;  % error variance
for j=1:8
  xic=x0+sigma2*randn(1,3); % perturb initial conditions
  [t,xsol]=ode45('lor_rhs',t,xic,[],sigma,b,r);
  x=xsol(:,1);  % projected x values
  subplot(4,2,j), plot(t,x_true,'k'), hold on
  plot(t,x,'k','Linewidth',[2])
end
```

For all these realizations, the projected state fails to model the *true* dynamics after $t \approx 5$. Indeed, after this time, there is almost no correlation of closeness

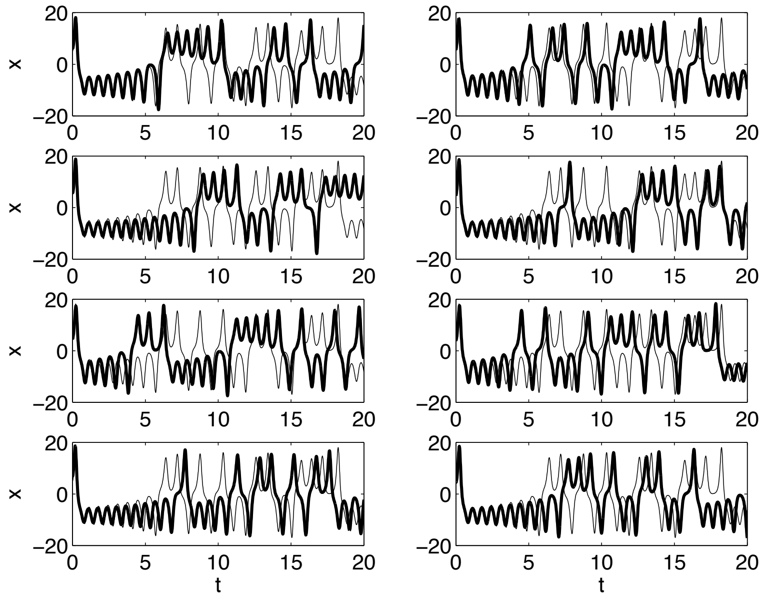


Figure 241: Time evolution of the variables $x(t)$, $y(t)$ and $z(t)$ over the time period $t \in [0, 20]$ for $\sigma = 10$, $b = 8/3$, $r = 28$ and $\mathbf{x}_0 = [5\ 5\ 5]^T$. Here, eight realizations are shown for the perturbed initial conditions given by (21.3.2) with $\sigma_2 = 1$. Note that for all simulations, after $t \approx 5$ the true dynamics (light line) differ from the dynamics with perturbed initial conditions (bold line). Thus prediction of the dynamics beyond this time is virtually impossible given such perturbations (errors) in the initial data.

between the truth and our projection based upon noisy initial data. This illustrates the need for data assimilation. Specifically, it is hoped that occasional measurements of the data would allow for an accurate prediction of the true future state far beyond $t \approx 5$.

Data assimilation for the Lorenz equations

The goal in this example will be to simply illustrate the EKF algorithm. Thus a simple case will be taken which can be completely coded in MATLAB in a fairly straightforward manner. In this simple example, no stochastic forcing of the differential equations will be considered so that the dynamics are *exactly* as specified in (21.3.1). Said another way, the error vector $\mathbf{q}_1$ in (21.1.6) is zero. However, there will be both error in the measurements ($\mathbf{q}_3$) and the initial conditions ($\mathbf{q}_2$). Knowing full well about the sensitivity to initial conditions in the Lorenz equations, the initial noise will greatly compromise the ability

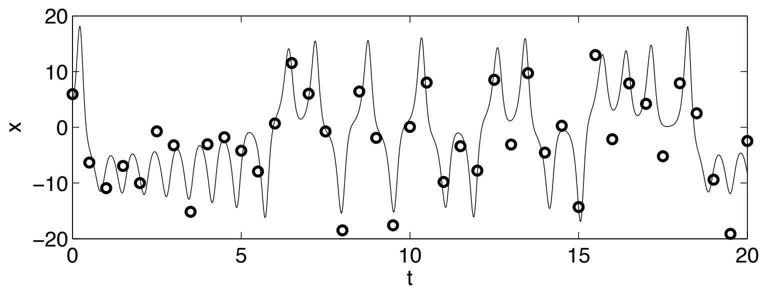


Figure 242: True time dynamics of the variable $x(t)$ over the time period $t \in [0, 20]$ for $\sigma = 10$, $b = 8/3$, $r = 28$ and $\mathbf{x}_0 = [5\ 5\ 5]^T$. The circles represent experimental measurements at every half-unit of time of the true dynamics with error given by (21.3.3) with $\sigma_3 = 4$. The dynamics of the variables $y(t)$ and $z(t)$ are similar. Data assimilation makes use of the experimental measurements to keep the model predictions closer to the true dynamics.

of the model to predict the future state of the system. The hope is that data assimilation will mitigate this problem to some extent and allow for much more accurate, and longer time, predictions for the future state of the system.

In our example, the data collection points will be at simulation time points already specified by our differential equation solver. Moreover, data will be taken from all variables. Thus the mapping matrix $\mathbf{H} = \mathbf{I}$. To illustrate the data collection process, consider then modification of (21.2.1) to

$$\mathbf{y}(t_n) = \mathbf{x}(t_n) + \sigma_3 \mathbf{q}(0, 1) \tag{21.3.3}$$

where t_n is a measurement time point and $\mathbf{q}(0, 1)$ is a Gaussian distributed random variable with mean zero and unit variance. Thus σ_3 is chosen to make the error variance either larger or smaller in the data measurements. Figure 242 shows the true dynamics (line) and data collected every half-unit of time with $\sigma_3 = 4$ (circles). As is clearly seen, the data is collected under error and does not match up perfectly with the true dynamics. However, it does follow the true dynamics fairly closely over the time period of integration. To compute these data points in MATLAB, the following code is used in conjunction with the code that generates the true dynamics:

```
% noisy obserations every t=0.5
tdata=t(1:50:end);
n=length(tdata)
xn=randn(n,1); yn=randn(n,1); zn=randn(n,1);
sigma3=4;  % error variance in data
xdata=x(1:50:end)+sigma3*xn;
ydata=y(1:50:end)+sigma3*yn;
```

```
zdata=z(1:50:end)+sigma3*zn;
```

The idea is to use these experimental points along with the noisy initial conditions shown in Fig. 241 in order to enhance our prediction for the future state.

Given that $\mathbf{H} = \mathbf{I}$ in this example, this reduces the EKF algorithm to computing

$$\bar{\mathbf{x}}_{k+1} = \mathbf{x}_{0_{k+1}} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{x}_{0_{k+1}}) \tag{21.3.4}$$

where $\mathbf{K}_{k+1}$ is the Kalman gain matrix given by

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1}(\mathbf{P}_{k+1} + \mathbf{R})^{-1} \tag{21.3.5}$$

and $\mathbf{R}$ is the noise covariance matrix. The error covariance of the updated state vector is given by

$$\bar{\mathbf{P}}_{k+1} = (\mathbf{I} - \mathbf{K}_{k+1})\mathbf{P}_{k+1}\,. \tag{21.3.6}$$

Note that the matrices involved are 3×3 matrices and the innovation is simply given by $\mathbf{y}_{k+1} - \mathbf{x}_{0_{k+1}}$. From (21.2.14), and using the fact that the dynamics propagates in an error free fashion ($\mathbf{q}_1 = 0$), then

$$\mathbf{P}_{k+1} = \mathbf{J}(\mathbf{f})\mathbf{P}_k\mathbf{J}(\mathbf{f})^T\,. \tag{21.3.7}$$

where the Jacobian for the Lorenz equation can be easily computed to give

$$\mathbf{J}(\mathbf{f}) = \begin{bmatrix} -\sigma & \sigma & 0 \\ r - z & -1 & -x \\ y & x & -b \end{bmatrix} \tag{21.3.8}$$

and the matrix $\mathbf{P}_k$ measures the error in estimating the initial state of the system at time t_k. This error is determined by (21.3.2) and the parameter σ_2.

Everything is in place then to implement the data assimilation procedure. The following is an algorithmic outline of what needs to occurs:

(i) Determine the sources of error and how to incorporate them. The error in the data measurement determines the matrix $\mathbf{R}$ while the error in the initial condition determines the matrix $\mathbf{P}_k$ (Note that we are ignoring errors generated in the dynamics themselves).

(ii) Compute the Jacobian at time t_k using the best estimate for the state vector $\mathbf{x}_0(t_k)$ and combine it with the computation of $\mathbf{P}_k$ in order to compute $\mathbf{P}_{k+1}$.

(iii) With $\mathbf{P}_{k+1}$ and $\mathbf{R}$, compute the Kalman gain matrix $\mathbf{K}_{k+1}$.

(iv) Compute the new state of the system using the innovation vector and the Kalman gain matrix.

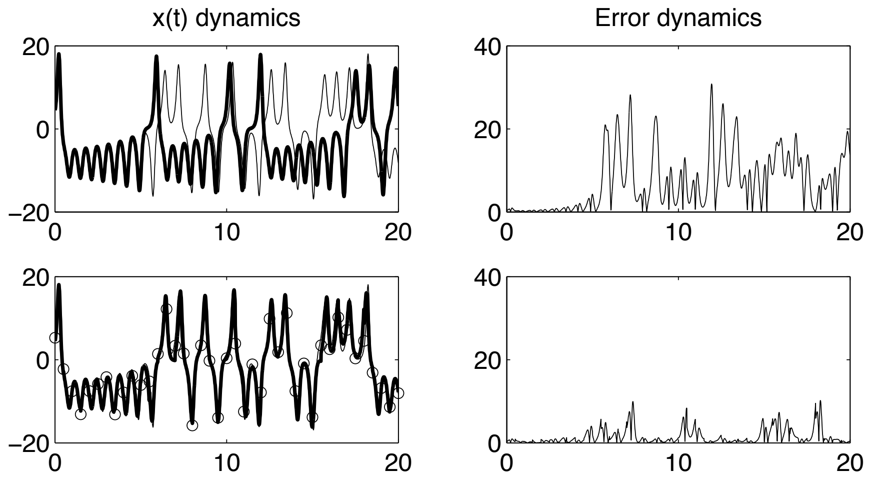


Figure 243: Comparison of the model dynamics using a direct numerical simulation of the noisy initial conditions (top panels) with the data assimilated solution with noisy initial conditions and noisy data measurements (bottom panels). The direct simulation (bold line) fails to predict the true dynamics (line) beyond $t \approx 5$ (top panel). Indeed, the error in this case grows quite large at this time (top right panel). When making use of the data assimilation technique (bottom panels) and the data measurements (circles), the solution stays close to the true solution for a much longer time with much smaller error (bottom right panel). Thus data assimilation can greatly extend the time window under which the model can be useful.

(v) Use the new state of the system to again project to another time into the future where observational data is once again available.

The Lorenz equation is a fairly trivial example to consider. Moreover, our treatment of the data assimilation will be for the simplest case possible. In this example, the error variance for both the initial conditions and data measurements will both be unity so that $\sigma_2 = \sigma_3 = 1$ respectively. The Kalman gain matrix will then be given as in the one-dimensional case: $K = \sigma_2/(\sigma_2 + \sigma_3)$. A code for simulating this system and making adjustments based upon the data measurements is as follows:

```
x_da=[];  % data assimilation solution
for j=1:length(tdata)-1  %  step through every t=0.5
  tspan=0:0.01:0.5;  % time between data collection
  [tspan,xsol]=ode45('lor_rhs',tspan,xic,[],sigma,b,r);

  xic0=[xsol(end,1); xsol(end,2); xsol(end,3)]  % model estimate
```

```
    xdat=[xdata(j+1); ydata(j+1); zdata(j+1)] % data estimate
    K=sigma2/(sigma2+sigma3);  % Kalman gain
    xic=xic0+(K*[xdat-xic0])  %  adjusted state vector

    x_da=[x_da; xsol(1:end-1,:)];  % store the data
  end
  x_da=[x_da; xsol(end,:)];  % store last data time
```

In this simulation, the vector **data-xic0** is the innovation vector that is weighted according to the Kalman gain matrix. Figure 243 shows the results of this simulation for one representative realization of the error vectors. In the top panels, the non-data assimilated computation is shown showing that the simulation solution diverges from the true solution around $t \approx 5$ (See also Fig. 241). The error between the true solution and the model solution is shown in the right panel. Note that the error is quite large around $t \approx 5$, thus making any prediction beyond this time fairly useless. In the bottom panel, the data assimilated solution is demonstrated (bold line) and compared to the true dynamics (line). The data assimilated solution is nearly indistinguishable from the true dynamics. The experimental observations are shown by circles at every half-unit of time. The error between the data assimilated solution and true dynamics is shown in the right panel. Note that the error remains quite small in comparison to the direct simulation from noisy initial data. Regardless, there is a build up of error that will eventually grow large as the data assimilated solution also diverges from the true dynamics. Ultimately, the data assimilated solution allows for significant extension of the time window where the model prediction is valid.

Data assimilation, in general, is much more sophisticated than what has been applied here to a simple 3×3 system. Indeed, for highly complex systems, the model error in the dynamics plays a fundamental role characterizing the behavior in addition to measurement and initial condition error. How one chooses to not only treat this error, but how to sample from the dynamics in time, gives rise to a great variety of mathematical techniques for enhancing the data assimilation method [83, 84, 85, 86]. Such data assimilation methods are at the heart of cutting-edge technology, for instance, in weather prediction and/or climate modeling. The hope here was simply to illustrate the basic ideas of this tremendously powerful methodology.

22 Equation Free Modeling

Multiscale phenomena, and their associated complex systems, are an increasingly important class of problems that need to be addressed from a mathematical point of view. As illustrated previously in the lecture on wavelets, some mathematical progress has been made in separating differing length scales by more sophisticated (wavelet) transforms. Alternatively, one can think about