AMath 584 Name: INSERT YOUR NAME HERE
Take-home Final UWNetID: INSERT YOUR NETID
Due to dropbox by 6pm PDT, Wednesday, December 14, 2011

The main part of the assignment (Problems 1–3) is worth 80 points.

The extra credit problem is worth 40 points. You will not be penalized if you don't do it.

Please use latex for this assignment! Submit a pdf file to the dropbox.

You can also submit Matlab or Python codes directly rather than including in `verbatim` environments. Please make it clear what code goes with which problem and write them nicely. Comments in the code are a good idea!

Problem #1.

(a) Using Matlab or Python (one of these!), write the following two functions and test them on a small example to make sure they work:

- `lu_nopivot`: Takes a square matrix A as input and returns matrices L and U corresponding to the LU factorization found by Gaussian elimination *without pivoting.* If the function encounters a divide-by-zero in the course of trying to do this (which might happen even if A is nonsingular), give an error message and abort.
- `solve_nopivot`: Takes as input `L, U, b`, the results of the LU factorization and a right hand side b of the appropriate length, and returns the solution x to the linear system, computed by first solving $Ly = b$ by forward substitution and then $Ux = y$ by back substitution.

Matlab hints: You can put each function in a separate m-file, e.g. `lu_nopivot.m`, starting with the line

```
function [L,U] = lu_nopivot(A)
```

You will need to determine the size of A by doing:

```
m = size(A,1)
```

This is the number of rows in A. Check that the number of columns is the same. If not, return an error, e.g.

```
if size(A,2) ~= m
    disp('** Error: A is not square')
    return
    end
```

You can use algorithm 20.1 on page 151 fairly directly. Note that the $m \times m$ identity matrix is obtained with `eye(m)` and the vector operation in the last line of the algorithm can be written as

```
    U(j,k:m) = u(j,k:m) - L(j,k)*U(k,k:m)
```

Python hints: The Python form of the function would be something like:

```
    def lu_nopivot(A):
        m = A.shape[0]
        if A.shape[1] != m:
            raise Exception('** Error: A is not square')
        #etc...
        return L,U
```

You can use algorithm 20.1 on page 151 fairly directly, but you'll need to correct for the fact that Python indexing starts at 0. Note that the $m \times m$ identity matrix is obtained with `eye(m)` and the vector operation in the last line of the algorithm can be written as

```
    U[j,k:] = u[j,k:] - L[j,k]*U[k,k:]
```

(b) Write a function `testge` that takes a matrix A (of arbitrary size) and a vector x (of the correct dimension) as input and does the following:

- Compute $b = Ax$ to use as a right hand side in tests below.
- Print the 2-norm condition number $\kappa(A)$ and also $\epsilon_m \kappa(A)$, where $\epsilon_m = 1.1 \times 10^{-15}$ is machine epsilon. This is the size of relative error you might hope for in solving the system $Ax = b$.
- Use your functions to factor A and solve $Ax = b$ without pivoting to obtain a result $\tilde{x}$.
- Compute and print the 2-norm of the error $\tilde{x} - x$ and the residual $A\tilde{x} - b$.
- Solve the system $Ax = b$ using backslash in Matlab or the NumPy routine `solve` to obtain a solution x^*.
- Compute and print the 2-norm of $x^* - x$ and the residual $Ax^* - b$.

These functions will be used in Problems 2 and 3 to see how well Gaussian elimination without pivoting works on some sample problems.

Solution:

Problem #2. (a)

A Toeplitz matrix has constant values along each diagonal, e.g.

$$A = \begin{bmatrix} c_0 & r_1 & r_2 & r_3 \\ c_1 & c_0 & r_1 & r_2 \\ c_2 & c_1 & c_0 & r_1 \\ c_3 & c_2 & c_1 & c_0 \end{bmatrix}.$$

A Toeplitz matrix can be specified in terms of its first column and first row in either Matlab or Python (NumPy) via

```
A = toeplitz(c,r)
```

(or simply `toeplitz(c)` if $r = c$, in which case the matrix is symmetric.) Consider the Toeplitz matrix in which

$$c_j = r_j = \alpha^j$$

for some value α. If $\alpha < 1$ then the values along each diagonal are decaying exponentially away from the main diagonal.

(a) Show that if $\alpha < 0.5$ then this matrix is *strictly column diagonally dominant*, meaning it satisfies

$$|a_{kk}| > \sum_{j=1}^{m} |a_{jk}|.$$

In other words, the diagonal term in each row is larger than the sum of all the off-diagonals. In this case it can be shown that no row interchanges will take place if Gaussian elimination with partial pivoting is applied. Hence factoring without pivoting should work fine in this case.

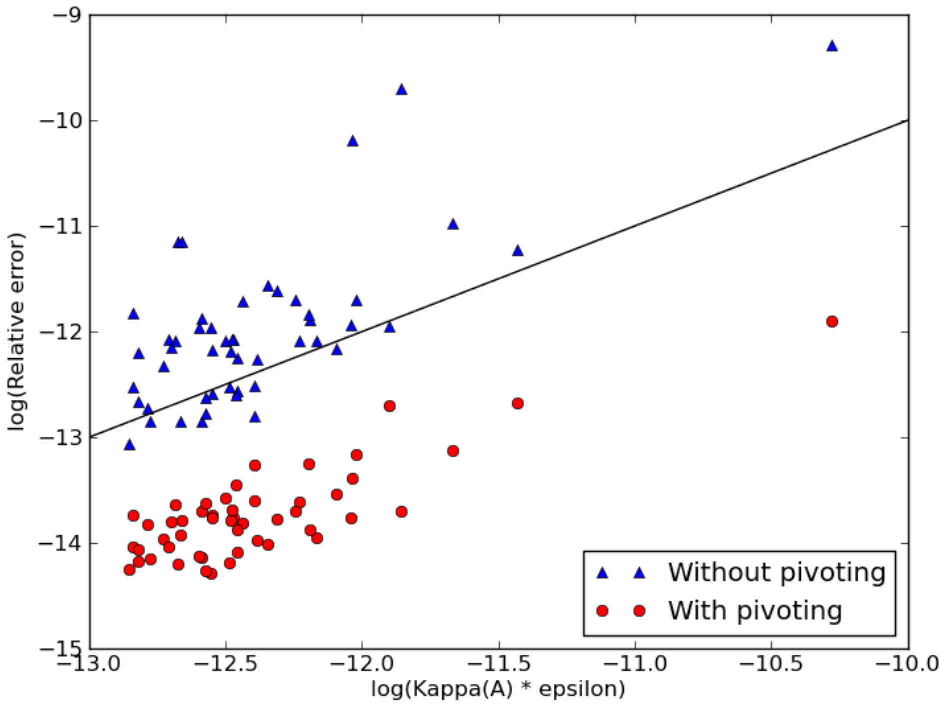
(b) Test out a Toeplitz matrix of this form for $\alpha = 0.45$ and $m = 200$ with your routines and see how the resulting solution compares to what the standard solver (backslash or `solve`) gives.

(c) Now try $\alpha = 0.95$ and $m = 200$. The matrix is not diagonally dominant but the off diagonal at least decay and you should get reasonable results and similar to what seen with the standard solver.

(d) Now try $\alpha = 1.1$ and $m = 200$. In this case the matrix elements grow exponentially away from the diagonal and pivoting would definitely interchange rows. Compare the accuracy of the solution obtained with your solver and with the standard solver and *comment* on whether each algorithm appears to be backward stable.

(e) Write a program that tests 50 random matrices of size 100×100 and produces a plot similar to the one below. You can generate a matrix whose entries are each normally distributed with mean 0 and variance 1 using the function `randn(100,100)`. Generate a vector x with random components using `randn(100,1)`. Use this to generate $b = Ax$ and then solve the system both with and without pivoting as in Problem 1.

Each matrix has a different condition number $\kappa(A)$ and the plot shows the relative error obtained with and without pivoting plotted against $\kappa(A)\epsilon_m$, on a base 10 logarithmic scale.



Solution:

Problem #3. A Toeplitz matrix is called *circulant* if

$$r_0 = c_0, \quad r_i = c_{m+1-i} \text{ for } i = 1,\ 2,\ \ldots,\ m-1,$$

for example

$$C = \begin{bmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{bmatrix} \qquad S = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \tag{1}$$

The specific example S is a *shift matrix* or *cyclic permutation* matrix that shifts each element in a vector by 1 position (with "periodic boundary conditions" so the last element moves to the first location).

Circulant matrices naturally arise if a finite difference operator is applied to a vector representing values of a function at m points on a domain with periodic boundary conditions. For example a centered approximation to a first derivative on a grid with spacing h might be approximated by such a matrix with $r_1 = 1/(2h)$ and $c_1 = -1/(2h)$ and all other elements 0. (Such matrices come up in 585 and 586).

(a) Show that any circulant matrix has eigenvectors of the form

$$x = \frac{1}{\sqrt{m}} \begin{bmatrix} 1 \\ \omega \\ \omega^2 \\ \vdots \\ \omega^{m-1} \end{bmatrix} \tag{2}$$

for any value of $\omega \in \mathbb{C}$ satisfying $\omega^m = 1$. (Note: the scalar factor $1/\sqrt{m}$ is immaterial for this problem, but useful below.)

(b) What is the corresponding eigenvalue λ (in terms of ω and $c_0,\ \ldots,\ c_{m-1}$)?

(c) The complex numbers ω satisfying $\omega^m = 1$ are the "mth roots of unity" and are equally spaced around the unit circle in the complex plane. For example, for $m = 4$ they are at ± 1 and $\pm i$. Determine the eigenvalues and eigenvectors of the 4×4 shift matrix S from (1)

(d) Let X be the matrix of eigenvectors of the 4×4 shift matrix S, normalized as in (2). Verify (by hand or computer) that $X^*X = I$. Hence the eigenvector matrix is unitary.

Notes on this problem to put it in context:

For general m, we can write the m possible values of ω as

$$\omega_k = e^{2\pi ik/m} = \cos(2\pi k/m) + i\sin(2\pi k/m), \quad k = 0,\ 1,\ \ldots,\ m-1,$$

where $i = \sqrt{-1}$.

The matrix of eigenvectors X thus has a very special form:

$$X_{jk} = \omega_k^j = e^{2\pi ijk/m}$$

This matrix of eigenvectors of an $m \times m$ circulant matrix (scaled as in (2)) is always unitary.

If we want to apply a circulant matrix C to a vector y we could write the product as

$$Cy = X \Lambda X^* y$$

where Λ is the matrix of eigenvalues of C.

This looks like a slow way to compute a matrix-vector product, replacing it with 3 matrix-vector products, first multiplying by X^*, then by Λ, and then by X. But in fact each of these can be done very quickly. Λ is diagonal so that is quick. The others are quick because multiplying by X^* and X correspond to doing a *Fourier transform* and its inverse, due to the special form of the ω_k. These products can therefore be computed using the *Fast Fourier Transform (FFT)* algorithm, which is a clever way of multiplying these matrices by a vector requiring only $\mathcal{O}(m \log m)$ work instead of $\mathcal{O}(m^2)$ work. For large values of m this is a huge speedup. (The trick relies on the fact that although X is dense, it can be written as the product of sparse matrices.) Routines for doing the FFT and inverse FFT are available in Matlab and NumPy (`fft` and `ifft`).

The fact that circulant matrices are related to Fourier transforms is also heavily used in the analysis of finite difference methods.

Solution:

Extra-Credit Problem.

Let $\mathcal{C}$ denote the set of continuous functions f defined on the interval $0 \leq s \leq 1$. $\mathcal{C}$ is a *linear space* since it is closed under addition of functions and scalar multiplication of a function by a real number.

Let K be a function that is continuous in $[-1, 1]$ and $f \in \mathcal{C}$. Then the function g defined by

$$g(t) = \int_0^1 K(t-s) f(s)\, ds \qquad \text{for any } t \in [0,1] \tag{3}$$

is also in $\mathcal{C}$ and is called the *convolution of K and f*.

If we fix the function K and consider the expression (3) it defines a *linear operator* $\mathcal{G} : \mathcal{C} \to \mathcal{C}$ by $g = \mathcal{G}(f)$. This mapping is linear because if $f_1, f_2 \in \mathcal{C}$ and $\alpha_1, \alpha_2 \in \mathbb{R}$, then

$$G(\alpha_1 f_1 + \alpha_2 f_2) = \alpha_1 G(f_1) + \alpha_2 G(f_2). \tag{4}$$

Functions in the infinite-dimensional space $\mathcal{C}$ can be approximated by vectors in $\mathbb{R}^m$ if we pick m points in the interval, say the equally spaced points $s_i = ih$ for $i = 1,\ 2,\ \ldots,\ m$ where $h = 1/m$, and then evaluate $f(s_i)$ to define a vector of length m.

$$\bar{f} = \begin{bmatrix} f(s_1) \\ \vdots \\ f(s_m) \end{bmatrix} \in \mathbb{R}^m.$$

The convolution integral of (3) can be replaced by a discrete version that maps a vector $\bar{f} \in \mathbb{R}^m$ to a new vector $\bar{g} \in \mathbb{R}^m$ if we replace the integral by a numerical approximation based on the m points, and define for example

$$g(t_i) = \int_0^1 K(t_i - s) f(s)\, ds \approx h \sum_{j=1}^{m} K(t_i - s_j) f(s_j). \tag{5}$$

This is just a matrix-vector product, so $\bar{g} = \bar{G}\bar{f}$ where the $m \times m$ matrix has elements

$$\bar{G}_{ij} = hK(t_i - s_j), \quad i, j = 1,\ 2,\ \ldots,\ m. \tag{6}$$

Note here that $t_i = ih$ and $s_j = jh$ (the same set of grid points are used for both s and t) and so

$$\bar{G}_{ij} = hK((i-j)h), \quad i, j = 1,\ 2,\ \ldots,\ m. \tag{7}$$

Note that $\bar{G}$ is a Toeplitz matrix.

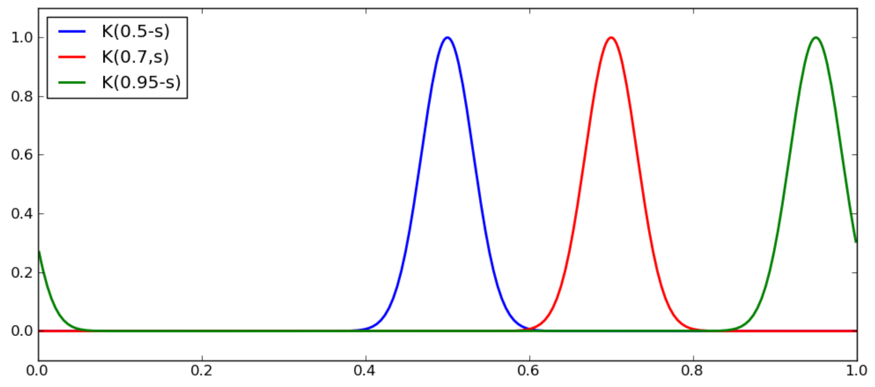
As a specific example, consider the Gaussian function

$$K(s) = \gamma \left(e^{-\beta s^2} + e^{-\beta (s-1)^2} + e^{-\beta (s+1)^2} \right). \tag{8}$$

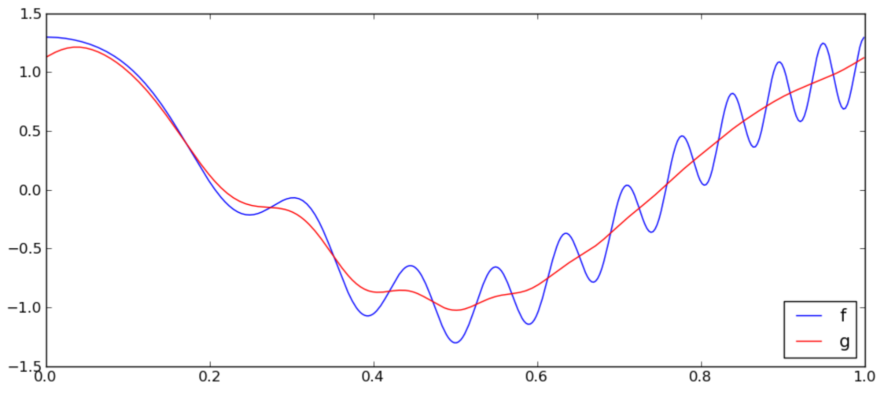
The three terms make the function periodic with period 1 over the interval $[-1, 1]$, and hence the matrix $\bar{G}$ will be circulant. The constant γ is a scale factor that depends on β, and should be chosen so that $\int_0^1 K(s)\, ds = 1$, but we'll choose it below to make the matrix have a related property.

This is a *smoothing kernal* in the sense that convolving K with a function f produces a smoother version of f. Intuitively, the integral (3) averages the values of f in a neighborhood of t to produce the value $g(t)$. The larger β is, the more this looks like a delta function so that $g(t)$ depends only on $f(s)$ for s very close to t. Smaller β gives more smoothing.

The figure below shows $K(t-s)$ as a function of s for three different values of t. Note that for $t = 0.95$ the periodic nature of the kernal is apparent. This means that in computing $g(0.95)$ the values of $f(s)$ for s near 0 also come into the average. Another way to view this is that we simply apply a Gaussian smoothing kernal, but to a periodic extension of the original $f(s)$ so that it is defined for all s and integrate over all s.



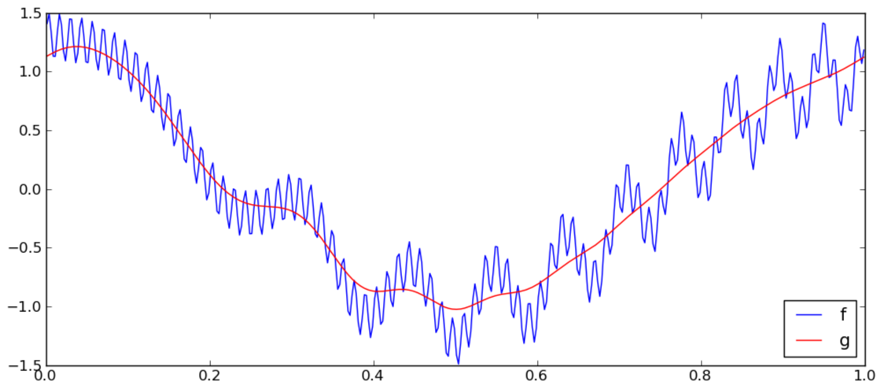
The next figure shows a function $f(s)$ and the resulting function g that is obtained by convolving with the K shown above.



Now consider the *inverse problem*: Given the smoothed version g, try to reconstruct the function f for which $g = \mathcal{G}(f)$. This sort of inverse problem comes up when trying to sharpen an image, for example.

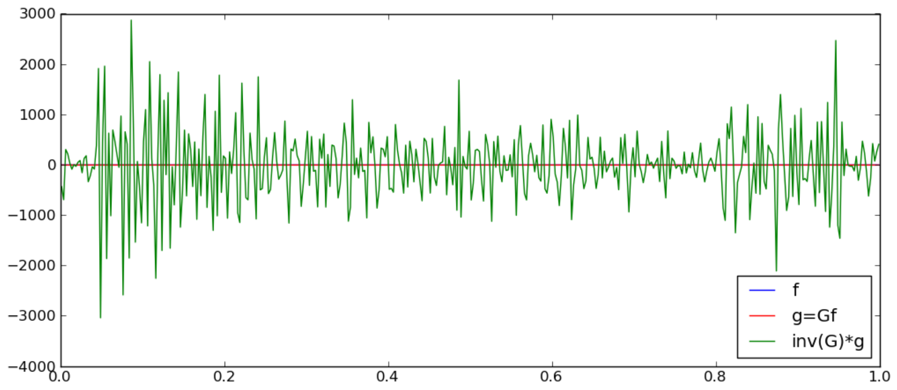
The discrete version corresponds being given a vector $\bar{g} \in \mathbb{R}^m$ and trying to determine $\bar{f} \in \mathbb{R}^m$ so that $\bar{G}\bar{f} = \bar{g}$. It can be shown that under certain conditions the matrix $\bar{G}$ should be nonsingular. However, it will often be very ill-conditioned.

To see why we expect ill-conditioning, suppose we change the function f used above by adding a perturbation that is highly oscillatory. The figure below shows the modified function and the result obtained by convolving with the same smoothing kernal as before.



The smoothed version g looks almost identical to the smoothed version of the original f. In fact they are identical to nearly machine accuracy. This means the inverse problem is very ill-conditioned: a tiny change in $\bar{g}$ can give a huge change in $\bar{G}^{-1}\bar{f}$.

So trying to solve the inverse problem by solving the system $\bar{G}\bar{f} = \bar{g}$ directly can not be expected to give good results. As an example of how bad, the figure below shows what we get if we solve the linear system (with $m = 400$).



(a) Compute the matrix $\bar{G}$ for this problem with $\beta = 500$ and $m = 100$. Use the formulas (7) and (8) with γ chosen so that the sum of elements in the first row of $\bar{G}$ is 1. (First form the matrix without γ and then compute the row sum in order to figure out how to scale the matrix.)

You should find that every row of $\bar{G}$ then sums to 1. Explain this fact based on what you learned in Problem 3.

(b) Let $\bar{f} \in \mathbb{R}^{100}$ be the vector obtained by evaluating the function

$$f(s) = \cos(2\pi s) + 0.3\cos(20\pi s^2)$$

at $s_j = jh$ with $h = 0.01$. This is the function that was used in the figures above.

Plot $\bar{f}$ vs. s, along with $\bar{g} = \bar{G}\bar{f}$, which should give a figure similar to the one above but on a coarser grid.

(c) Try solving the linear system to recover $\bar{f}$ from $\bar{g}$ and confirm that you get poor results.

(d) Compute the SVD of $\bar{G}$. Plot $\log_{10}(\sigma_i)$ vs. i to see how small some of the singular values are.

(e) You should find that some singular values are at the level of rounding error. These values (and the corresponding vectors) have no accuracy. Solve the linear system using a pseudo-inverse of $\bar{G}$ obtained by throwing away the singular values and vectors corresponding to any $\sigma_i < \epsilon$. Try various values of the tolerance ϵ to see how this affects the function you recover $\bar{G}^+\bar{f} = \hat{V}\hat{\Sigma}^{-1}\hat{U}^*\bar{f}$.

(Note: From Problem 3 you might note that this could be done more efficiently using the FFT, but do it using the SVD.)

(f) Try the same thing but starting with the function

$$f(s) = \begin{cases} 2 & \text{if } x < 0.5 \\ 2s & \text{if } x > 0.5 \end{cases}$$

How well can you recover this discontinuous function from its smoothed version? Experiment with different cutoff levels ϵ. Note that a discontinuous function requires many high frequencies to represent it as a Fourier series and the pseudo-inverse is essentially throwing some of these away.

Notes:

- A similar approach of setting small singular values to zero can be used for other inverse problems. It is also possible to define an approximate inverse by replacing $1/\sigma_j$ that would appear in the exact inverse (when written in terms of Σ^{-1}) by $\sigma_j/(\sigma_j^2 + \epsilon^2)$ for some small value of $\epsilon > \epsilon_m$. This is related to *Tikhonov regularization.*
- For the present problem this technique is also a variant of Fourier or band-pass filtering.
- The matrix $\bar{G}$ is symmetric and so the singular vectors are eigenvectors. Since the matrix is real, they are real vectors in spite of the fact that we know eigenvectors can have the form (2) where ω in general is complex. You might want to think about this and perhaps plot some of the singular vectors (plot a column of U against s). Hint: What is the multiplicity of each eigenvalue?

Solution: