CS 222: Prelim 1 Solutions

```
95-100    xxxxxxxx
90-94     xxxxxx
85-89     xxxxx
80-84     xxxxxxxx
75-79     xxxxxxxxx
70-74     xxxxxxxxxxxxxx
65-69     xxxxxxxxxxxxxxxxxxxxx
60-64     xxxxxxxxxxxxxxxxx                    Median = 58
55-59     xxxxxxxxxxxxxxx
50-54     xxxxxxxxxxxxxxxxxxx
45-49     xxxxxxxxxxxxxxxxxxxxx
40-44     xxxxxxxxxxxxxxxxxx
35-39     xxxxxxxxxx
30-34     xxxxxxxxx
< 30      xxxxxx
```

Rough grade guidelines A = [70,100], B = [50,64], C = [35,44]. Study the solutions before requesting a regrade. Regrades due by Wednesday March 29. Indicate on the cover or on separate pieces of paper what it is that you want us to regrade. DO NOT mark up your solutions so that we can be assured that academic integrity code is being upheld. Recall that it is our policy to make copies of a random subset of the exams.

1. (a) In a base-2 floating point number system with 10-bit mantissas, exactly how many floating point numbers are greater than or equal to 7 and less than or equal to 9? You must show work to receive full credit.

5points:
Spacing to the left of 8 is $2^{-10} * 2^3 = 2^{-7}$. So there are $2^7 + 1 = 129$ floating point numbers from 7 to 8 including 7 and 8.

5 points:
Spacing to the right of 8 is $2^{-10} * 2^4 = 2^{-6}$. So there are $2^6 + 1 = 65$ floating point numbers from 8 to 9 including 8 and 9.

So there are 129 + 65 -1 = 193 floating point numbers in between 7 and 9.

-1's for off-by-one mistakes

(b) Set up a 4-by-4 linear system that determines $a_1,\ a_2,\ a_3,\ a_4$ so that if

$$p(x) = a_1 + a_2x + a_3x^2 + a_4x^3$$

then $p(0) = 1$, $p'(1) = 0$, $p(1) = 3$, and $p(-2) = p(2)$. Do not solve the system.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \\ 0 & 4 & 0 & 16 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Equations 1,2, and 3 are 2 points each. Equation 4 is worth 4 points. Ok to write out in equation form rather than matrix-vector form.

2. Complete the following MATLAB function so that it performs as specified:

```
  function x = F(d,c,b)
% d is a column n-vector.
% c is a column m-vector.
% b is a column n-vector.
%
% x is a column n-vector that solves Ax = b where
%
%        A = c(1)*I + c(2)*D + c(3)*D^2 + ... + c(m)*D^(m-1)
%
% where I is the n-by-n identity matrix and D is the n-by-n diagonal
% matrix with D(i,i) = d(i), i=1:n. It may be assumed that the matrix A
% is nonsingular.
```

Make use of the nested multiplication idea for polynomial evaluation, e.g.,

$$((c_4 z + c_3)z + c_2)z + c_1 = c_4 z^3 + c_3 z^2 + c_2 z + c_1.$$

and vectorize your solution.

```
% 1 point:

m = length(c);
n = length(d);
y = c(m)*ones(n,1);

% 6 points:
for i=m-1:-1:1
   y = y.*d + c(i);
end

% 3 points
x = b./y;
```

-10 points for explicitly setting up D and I and then doing something like this

```
A = c(m)*I;
for i=m-1:-1:1
   A = A*D + c(i)*I;
end
x = A\b;
```

This involves $O(mn^3)$ flops.

3. We want to interpolate a function f on $[a, b]$ with error less than *tol*. When is it cheaper to set up a piecewise linear interpolant $L(z)$ with a uniform partition than a piecewise cubic hermite interpolant $C(z)$ with a uniform partition? Your answer should make use of the following facts and assumptions:

- If ℓ is the linear interpolant of f on an interval $[\alpha, \beta]$, then on that interval the error is no bigger than $M_2(\beta - \alpha)^2/8$, where M_2 is an upper bound for $|f^{(2)}(z)|$. Assume that M_2 is known.

- If p is the cubic hermite interpolant of f on an interval $[\alpha, \beta]$, then on that interval the error is no bigger than $M_4(\beta - \alpha)^4/384$, where M_4 is an upper bound for $|f^{(4)}(z)|$. Assume that M_4 is known.
- A vectorized MATLAB implementation of the function f is available and it requires σn seconds to execute when applied to an n-vector. Assume that σ is known.
- A vectorized MATLAB implementation of the function f' is available and it requires τn seconds to execute when applied to an n-vector. Assume that τ is known.

The following grading guidelines are only approximate. It was necessary to do a lot of "interpolation" to give partial credit in this problem. If you didn't look at this as a piecewise polynomial problem you didn't get more than 5-7 points, if that.

Let assume we base the interpolant on n points, $x = linspace(a, b, n)$. This means that the subinterval length is $h = (b - a)/(n - 1)$. (2 points)

In the linear case we must choose n to be the smallest integer so that

$$M_2 h^2/8 = M_2(b - a)^2/(8(n - 1)^2) <= tol \qquad (4pts)$$

i.e., set $n = n_1$ where

$$n_1 = ceil\left((b - a)\sqrt{\frac{M_2}{8tol}} + 1\right) \qquad (2pts)$$

In the piecewise cubic hermite case we must choose n to be the smallest integer so that

$$M_4 h^4/384 = M_4(b - a)^4/(384(n - 1)^4) <= tol \qquad (4pts)$$

i.e., set $n = n_2$ where

$$n_2 = ceil\left((b - a)\sqrt[4]{\frac{M_4}{384tol}} + 1\right) \qquad (2pts)$$

With n_1 and n_2 so determined we can compare execution times. We should do piecewise linear if

$$\sigma n_1 \leq (\tau + \sigma) n_2 \qquad (6pts)$$

and piecewise cubic otherwise.

-4 if τ not involved correctly.

4. Suppose `A` is a given n-by-n nonsingular matrix. To compute its LU factorization we start by adding multiples of the first row to rows 2 through n. The multiples are chosen so that components $(2, 1), \ldots, (n, 1)$ are zeroed. **(a)** Write a MATLAB script that modifies `A(2:n,:)` in this way. It is not necessary for you to vectorize your script. **(b)** Explain in English how partial pivoting changes the script in part **(a)**. **(c)** Explain in English why partial pivoting does not significantly add to the flop count of the overall algorithm.

```
% 4 pts
v(2:n) = A(2:n,1)/A(1,1);
% 6 pts
for i=2:n
   A(i,:) = A(i,:) - v(i)*A(1,:);
end
```

(b) Scan `abs(A(:,1))`for the largest value and interchange that row and row 1. (5 points)

(c) The search for the maximal element in a subcolumn is an order of magnitude less than the update of the submatrix that follows. This means that pivoting costs $O(n^2)$ while the elimination process itself is $O(n^3)$. (5 points)

5. Suppose $f(t)$ and $g(t)$ are functions that take a real scalar t and return a real column n-vector. Assume that A is a nonsingular n-by-n matrix. We wish to plot a spline approximation of

$$h(t) = f(t)^T A^{-1} g(t)$$

across the interval [0,1]. To that end, assume `n` and `A` are available along with functions `f(t)` and `g(t)` that implement $f(t)$ and $g(t)$ respectively.

Write a MATLAB script that plots the function S across [0,1] where S is the not-a-knot spline interpolant of $h(t)$ at $t = 0.0, 0.1, 0.2, \ldots, 0.9, 1.0$. Make effective use of the MATLAB functions `LU` and `SPLINE`:

```
[L,U,P] = LU(A) returns lower triangular matrix L, upper
triangular matrix U, and permutation matrix P so that
P*A = L*U.

YY = SPLINE(X,Y,XX) uses cubic spline interpolation
to find a vector YY corresponding to XX.  X and Y are the
given data vectors and XX is the new abscissa vector.
```

```
[L,U,P] = lu(A);
tvals = linspace(0,1,11);
hvals = zeros(11,1);
for i=1:11
   hvals(i) = f(tvals(i))'*(U\L\(P*g(tvals(i))));
end
tau = linspace(0,1)';
svals = spline(tvals,hvals,tau);
plot(tau,svals)
```

6 points for LU outside loop
8 points for using L,U, and P correctly to get hval(i)
6 for the stuff after the loop