CS 322: FinaL Grading Guide

P1-P5 (Max = 100)

97	x
96	xx
95	xxxx
94	xxxxxxxxxxxx
93	xxxxxxxxxx
92	xxxxxxxxx
91	xxxxxxxxxx
90	xxxxxxxxx
89	xxxxxxxxxxxx
88	xxxxxxxxxxx
87	xxxxx
86	xxxxxxxx
84	xxxxxxx
83	xxxxxxx
82	xxxxxxxxx
81	xxxxxxxx
80	xxxxxxx
79	xxxx
78	xx
77	xxxxx
76	xxxxx
75	xxxx
74	xxx
73	xxx
72	xx
71	xxx
70	xxx
< 70	xxxxxxxxxx

Final Exam (Max = 100)

85-90	xxxx
80-84	xxxxxxxxx
75-79	xxxxxxxxxxxxxxxxx
70-74	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
65-69	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
60-64	xxxxxxxxxxxxxxxxxxxxx
55-59	xxxxxxxxxxxxxxxxxxxxx
50-54	xxxxxxxxxxxxxxxxxxxx
45-49	xxxxxxxx
40-44	xxxxxx
< 40	xxxxxxxx

Total Scores = .30*(P1+P2+P3+P4+p5) + .20Pre1 + .20 Pre2 + .30*Final

85-100	xxxxxxx
82-84	xxxxxxxxx
79-81	xxxxxxxxxxxxxxxx
76-78	xxxxxxxxxxxxxxxxxxx
73-75	xxxxxxxxxxxxxxxxx
70-72	xxxxxxxxxxxxxxxxxx
67-69	xxxxxxxxxxxxxxxxxxxxxx
64-66	xxxxxxxxxxxxxxxx
61-63	xxxxxxxxxxxxxxxxxxxxxx
58-60	xxxxxxxx
55-57	xxxxxxx
52-54	xxxxx
< 51	xxxxxxxx

Approximate Course Grade distribution $(A, B, C) \approx (30\%, 40\%, \%25)$

1. (a) (10 points) Let P_m be the regular polygon with vertices $(\cos(2\pi k/m), \sin(2\pi k/m)$, $k = 0{:}m-1$. Complete the following script so that it displays $P_{n/2}$ and P_{2n} in the same window. Do not compute any more sines or cosines. Your solution should be vectorized. Do not worry about axis scaling.

```
% Assume n is a positive even integer >= 6
theta1 = linspace(0,2*pi,n+1); c1 = cos(theta1); s1 = sin(theta1);
theta2 = theta1 + pi/n;        c2 = cos(theta2); s2 = sin(theta2);
```

Solution

```
% Display P_(n/2). Every other P_n vertex is a P_(n/2) vertex.

plot(c1(1:2:n+1),s1(1:2:n+1))              % 4 points

% Every other P_(2n) vertex is a P_n vertex

c = zeros(1,2*n+1); c(1:2:2*n+1) = c1;     % 1 point
s = zeros(1,2*n+1); s(1:2:2*n+1) = s1;     % 1 point

% In between these we have the (c2(i),s2(i)), i=1:n

c(2:2:2*n) = c2(1:n);                      % 1 point
s(2:2:2*n) = s2(1:n);                      % 1 point
hold on                                    % 1 point
plot(c,s)                                  % 1 point
hold off
```

Two points if you compute the sines and cosines from scratch.

(b) (10 points) Assume that $A \in \mathbb{R}^{n\times n}$, $u \in \mathbb{R}^n$, and $x \in \mathbb{R}^n$ are stored in MATLAB arrays `A` (n-by-n), `u` (n-by-1), and `x` (n-by-1). Write a script that assigns $y = Auu^TA^Tx$ to `y`.

Solution:

```
w = A*u; alfa = w'*x; y = alfa*w;          % 10 points. Note: one matrix-vector product.

w = A*u; v = A'*x; beta = u'*v; y = beta*w  % 5 points because you didn't use the
                                           % fact that (u'*A') = (A*u)'

y = (A*u)*(v'*(A'*x))                      % Another 5-point solution because it
                                           % involves two matrix-vector products
                                           % like the previous solution.

y = A*u*u'*A'*x                            % 2 points because this is O(n^3),
                                           % (A*u*u') is an n-by-n matrix
```

2. **(a)** (10 points) Write a script that assigns values to `d` such that the polynomial $p(x) = d_1 + d_2(x-1) + d_3(x-1)^2 + d_4(x-1)^3$ has the property that $p(3) = 1$, $p'(3) = 2$, $p''(3) = 7$, and $p(4) = 10$.

Solution

```
% d(1) + 2d(2) + 4d(3) +  8d(4)  = 1        2 points
%        d(2) + 4d(3) + 12d(4)  = 2        2 points
%               2d(3) + 12d(4)  = 7        2 points
% d(1) + 3d(2) + 9d(3) + 27d(4)  = 10       2 points

d = [ 1 2 4 8 ; 0 1 4 12; 0 0 2 12; 1 3 9 27]\[1;2;7;10]      % 2 points


>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Miscellaneous Deductions:

1.  Entries backwards in A - Minus 2 pts
2.  Setting sA = sparse(A) and not using it - Minus 1 pt
3.  Made b a row vector (i.e., forgot transpose) - Minus 1 pt
4.  Rearranged the rows of A but not corresponding entries in b -
      Minus 3 pts.
5.  Set up A' rather than A - Minus 2 pts.
6.  Use A(i) instead of A(i,:) - Minus 2 pts.
7.  Initialize d vector to d=[d1 d2 d3 d4]'; - Minus 2 pts.
8.  b\A - Minus 2 pts.
9.  Discussion of using Cholesky even if used LU later - Minus 2 pts.
```

(b) (10 points) Consider the following script:

```
n = 5;
theta1 = linspace(0,2*pi,n);  S1 = spline(theta1,sin(theta1)); y1 = ppval(S1,4);
theta2 = linspace(pi,3*pi,n); S2 = spline(theta2,sin(theta2)); y2 = ppval(S2,4);
```

Ignoring roundoff, do you think `y1` and `y2` have the same value? Explain. Hint: Think about symmetry, periodicity, and the fact that `spline` returns the not-a-knot interpolant.
Solution:
Since we have 4 subintervals, each spline is made up of just two cubics. (Not-a-knot tells us that.)
Let p_1 (across $[0, \pi]$) and p_2 (across $[\pi, 2\pi]$) be the two cubics that define S_1.
Let q_1 (across $[\pi, 2\pi]$) and q_2 (across $[2\pi, 3\pi]$) be the two cubics that define S_2.
The main conclusion from symmetry: $S_2(3\pi - x) = S_1(x)$.
It **doesn't** follow that $S_1(4) = p_2(4)$ equals $q_1(4) = S_2(4)$.
Note:It doesn't follow that p_2 is $-p_1$ translated right, or that q_1 is $-q_2$ translated left, or that q_2 is p_1 translated right.

```
Points were given as follows:

1.  Observing that each spline involves just 2 cubics - 3 pts.
2.  Saying "No" - 2 pts.
3.  Seeing that it boils down to the question does p2=q1 on [pi,2*pi] -
      3 pts.
4.  Seeing that S2 is S1 in reverse - 2 pts.
5.  Thinking about p1 and p2 "is one the flip of the other" - 2 pts.
6.  Picture of sine function (or two of them) on interval(s) with
      breakpoints drawn in - 2 pts.
```

```
7.  Description of sine function being symmetric/reflection - 2 pts.
8.  Description of sine function being "out-of-phase" or periodic - 2 pts.
9.  Description of using S''' continuous at x2, x_{n-1} which will break
symmetry since 4 is closer to one of these points in one spline than in
the other - 2 pts.

MAXIMUM of 10 pts (no matter how many of these you wrote)!!!
```

3. (15 points) Implement the following function so that it performs as specified:

```
function x = FirstCol(A)
% A is m-by-n (m>=n) an gas rank n.
% x is the first column of G where G is n-by-n, lower triangular, and satisfies
% G*G' = A'*A
```

Your solution should be vectorized and efficient
Solution

```
% Compute the first column of C = A'*A
v = A'*A(:,1);                % 5 points

% Comparing firs columns of C = GG' we see C(:,1) = G(:,n)*G(1,1).
% It follows that G(1,1) = sqrt(C(1,1))
x(1) = sqrt(v(1));            % 5 points

% C(2:n,1) = G(2:n)*alfa so
x(2:n) = v(2:n)/alfa;         % 5 points
```

-3 for no vectorization.

-2 for `B = A'*A; c = B(:,1)`

Here is a 4-point solution since it is an order of magnitude more work:

```
G = chol(A'*A); x = G(:,1)
```

Working with LU in this way is worth 3 points.
Doing an n = 2 example correctly getting x from the first column of C is worth up to 8 points.

4. (15 points) How would you use `fmin` to find that point on the ellipse

$$x(t) = h + a\cos(t)$$
$$y(t) = k + b\sin(t)$$

which is furthest away from a given point (x_0, y_0). Your answer should include an implementation of the objective function that is passed to `fmin`, a justification of the search interval $[L, R]$, and the script that sets up `L` and `R` and calls `fmin` For your information

```
z = fmin('f',L,R,[],p1,p2,..)
```

returns a minimizer of the function $f(t, p1, p2, \ldots)$ on the interval $[L, R]$. Assume that `x0`, `y0`, `h`, `k`, `a`, and `b` are available. In the calling script, assign the x and y coordinates of the solution to `xstar` and `ystar`. Efficiency matters (as always). Do not worry about error tolerances. Do not worry if there is more than one point on the ellipse that is maximally separated from (x_0, y_0)

Solution

```
% 5 points for efficient deterinination of L and R
   if x0>=h & y0>=k; L = pi;     R = 3*pi/2 ; end
   if x0>=h & y0<k;  L =  pi/2;  R = pi ;     end
   if x0<h  & y0>=k; L = 3*pi/2; R = 2*pi ;   end
   if x0<h  & y0<k;  L = 0;      R = pi/2 ;   end

% 3 points for fmin call
   tstar = fmin('MyF',L,R,[],a,b,h,k,x0,y0);

% 2 points for this...
   xstar = h + a*cos(tstar);
   ystar = k + b*sin(tstar);

% 5 points for the objective function

   function z = MyF(t,a,b,h,k,x0,y0)
   xdiff = (h+a*cos(t) - x0);
   ydiff = (k+b*sin(t) - y0);
   z = -sqrt(xdiff^2 + ydiff^2);


Miscellaneous deductions:

1.  Objective function that minimizes - Minus 2 pts.
2.  L = 0, R = 2*pi - Minus 3 pts.
3.  Use of splines (inefficient) - Minus 3 pts.
4.  Didn't pass a,b,h,k - Minus 1 pt.
5.  Didn't pass a,b,h,k,x0,y0 - Minus 2 pts.
6.  Didn't compute x,y in objective function - Minus 2 pts.
7.  Objective function of 1/sqrt(dist) - Minus 2 pts. (won't work if
(x0,y0) is on ellipse.)
8.  Discrete version - Minus 5/10 pts. from objective function and script
(doesn't pertain to 5 pts for choice of [L,R].)
9.  Objective function of sqrt(dist)-maxdist_inside_ellipse - Minus 2 pts.
(would minimize, not maximize).
10. [L,R] - not angles at all - Minus 5 pts.
```

5. (10 points) Elementary calculus tells us that for small h

$$\left| \frac{f(x+h) - f(x)}{h} - f'(x) \right| = \frac{h}{2} |f''(\eta)|$$

where $x \leq \eta \leq x + h$. The script

```
disp('         h                  Error')
disp('-----------------------------------------')
for k = 1:12
   h = 1/16^k;
   Dh = (exp(1+h)-exp(1))/h;
   error = abs(Dh - exp(1));
   disp(sprintf('%20.15f  %20.15f',h,error))
end
```

explores this result for the function e^x at $x = 1$. Here is the output:

```
         h                      Error
-----------------------------------------
 0.062500000000000     0.086744022944412
 0.003906250000000     0.005316063900590
 0.000244140625000     0.000331848518794
 0.000015258789063     0.000020738972192
 0.000000953674316     0.000001296274671
 0.000000059604645     0.000000088814953
 0.000000003725290     0.000000036660889
 0.000000000232831     0.000000871125916
 0.000000000014552     0.000010407869080
 0.000000000000909     0.000020109709045
 0.000000000000057     0.000468171540955
 0.000000000000004     0.031718171540955
```

Explain why the error decreases and then grows as h gets smaller and smaller. You may assume that `exp` returns the nearest floating point number to the true exponential.

Solution

```
5 points:  Rounding Errors in numerator are maginified by 1/h.

3 points: As h goes to zero calculus error goes to zero but rounding error grow
as EPS/h.

2 points: The sum h + EPS/h minimizes around h = sqrt(EPS). Or Mention "tradeoff"
          between roundoff and calculus error.
```

6.(a) (10 points) Give two reasons why automatic stepsize control is important in an initial value problem solver like `ode23`.

Solution

```
Error control (5pts)

If they talked about accuracy, they also received 5 points.
If they talked around error, e.g. "prevents misbehviour", "a closer
answer", they received 3 points.

Reduce number of f-evals (5pts)

If they talked generally about increasing efficiency, reducing number of
iterations, smaller number of steps, run at higher speeds, etc, without
talking about function evaluations, they generally received 3 points.


Some answers comnined the two, e.g. "optimal flop count for given
tolerance" = 5/10 points; "allows the stepsize to be small where
appropriate but large where the result won't be severly corrupted (for a
given number of f-evals, greater accuracy)" = 7/10 points.

No points were awarded for "stability", "overshooting", or for issues
related to steepest descent methods.
```

(b) (10 points) The backwards Euler method for the initial value problem $y' = f(t,y),\ y(t_0) = y_0$ is defined by

$$y_{n+1} = y_n + h_n f(t_{n+1}, y_{n+1})$$

where $t_{n+1} = t_n + h_n$ and $y_n \approx y(t_n)$. Complete the following function so that it performs as advertised:

```
function Yvals = BackEuler(A,c,y0,h,N)
% A is an m-by-m matrix
% c and y0 are column m-vectors
% h > 0 and N is a positive integer
% Yvals is an N-by-m matrix with  the property that Yvals(n,:)' is the
% backward's Euler solution to
%
%            A*(dy/dt) = y + exp(-t)*c     y(0) = y0
%
% at t = nh.
```

Make effective use of `[L,U,P] = LU(·)` when solving linear systems. Assume nonsingularity of all the linear systems whose solution is required.

Solution

Note that $y_{n+1} = y_n + hA^{-1}(y_{n+1} + e^{-t_{n+1}}c)$ and so $(I - hA^{-1})y_{n+1} = y_n + he^{-t_{n+1}}A^{-1}c$. Apply A to both side and conclude:

$$(A - hI)y_{n+1} = Ay_n + he^{-t_{n+1}}c)$$

```
[m,m] = size(A);
Yvals = zeros(N,m);
[L,U,P] = LU(A-h*eye(m,m));
for n = 1:N
   % Have solution at t = (n-1)h, get solution at t = nh.
   b = A*y0 + exp(-n*h)*c;
   ynew = U\(L\P*b);
   Yvals(k,:) = ynew';
   y0 = ynew;
end
```

>>>>>>>>>>

3 for using LU outside of loop
2 for correct matrix
2 for correct rhs when finding ynew
3 for correct use of L, U, P

-2 for screwups like not updating y0 (if they were using y0 in their loop), or for having yvals(1,:) = y0.

If people got the correct answer, but explicitly found the inverse of A, (i.e. they used (I - hA^{-1})y_{n+1} = y_n + he^{-t_{n+1}}A^{-1}c) they received 8 out of 10 (-2 for not having the correct matrix).

Another common mistake was using y_{n+1} = y_n + h*(A\(y_n + exp(-h*(n+1))*c), i.e. y_n instead of y_{n+1} on the RHS. They would generally call LU on the matrix A. This generally received 6 out of 10, as they lost 2 points for the correct matrix for LU decomposition and two points for the correct rhs when finding ynew. (They could get less than this if they also e.g. put y0 in yvals)