Math 273 Final Exam

Carry out the *solution* of each problem: show steps of any required calculations; state reasons that justify any conclusions. Mere oracular *answers* will receive no credit.

1. Simulation of an RC circuit leads to the initial value problem

$$y' = -1000(y - \cos 2t) \qquad y(0) = 0.$$

Fill in *one* line in each MATLAB fragment below so that they compute the solution for $0 \le t \le 2\pi$ by (a) the Euler method, and (b) the backward Euler method. [The backward Euler method requires you to solve an algebraic equation for y_{n+1}.]

```
t = linspace(0,2*pi,nsteps+1)';
y = [y0; zeros(nsteps,1)];
h = 2*pi/nsteps;
for n = 1:nsteps
% Euler method
  y(n+1) =
end
```

```
t = linspace(0,2*pi,nsteps+1)';
y = [y0; zeros(nsteps,1)];
h = 2*pi/nsteps;
for n = 1:nsteps
% Backward Euler method
  y(n+1) =
end
```

2. Consider the 1-D *minimization* (*not* zero-finding) problem

$$\min_{x>-1} f(x) = x^3 - 9x^2 + 18x + 12$$

(a) Find the steepest descent direction and the Newton optimization step starting from $x = 2$.

(b) Find the steepest descent direction and the Newton optimization step starting from $x = 4$.

(c) Observe that Newton's method goes "downhill" in one case, "uphill" in the other. What accounts for the difference?

3. Vectorize the loops in the following fragment. Assume that `v` is a column vector of length `m`, and `w` is a column vector of length n.

```
a = zeros(m,n);
for i = 1:m
  for j = 1:n
    a(i,j) = v(i)*w(j);
  end
end
```

Exam Continues on Reverse

4. For the solution of the linear system $Mx = y$

$$M = \begin{bmatrix} .4107 & .1183 \\ .3923 & .1130 \end{bmatrix}, \qquad y = \begin{bmatrix} .5290 \\ .5053 \end{bmatrix}$$

the vector $z = (1.2313, 0.1970)^T$ is computed.

(a) Calculate the residual of z. What does this residual suggest about the accuracy of the proposed solution z?

(b) Observe that the *exact* solution is $x = (1, 1)^T$. Calculate the relative error of z.

(c) What do the results of parts (a) and (b) tell you about the matrix M?

5. Assume that `fun.m` and `dfun.m` are M-files that compute an n-component vector function of an n-component vector x, and the Jacobian of that function, respectively. Here are three MATLAB fragments that solve `dfun(x)=0` by the simplified Newton method (i.e. evaluating the Jacobian only once).

Determine the number of floating point operations required for each line of code marked "#", and then rank the fragments according to efficiency.

```
| F = fun(x);         | F = fun(x);         | F = fun(x);
| J = dfun(x);        | J = dfun(x);        | J = dfun(x);
#                     # [L,U] = lu(J);      #
| while norm(F)>tol   | while norm(F)>tol   | while norm(F)>tol
#   s = -J\F;         #   s = -(U\(L\F));   #   s = -1*inv(J)*F;
|   x = x + s;        |   x = x + s;        |   x = x + s;
|   F = fun(x);       |   F = fun(x);       |   F = fun(x);
| end                 | end                 | end
```

Exam Begins on Obverse