1. **32 points:** 8 parts, 4 points each
True or False? Give a reason to justify your answer.

(a) Roundoff error is a consequence of computers' finite memory.

True. Infinite decimals must be represented with a finite number of bits. Only a finite number of floating point numbers can make up a field; hence, even rational numbers or integers may experience roundoff error, as they will be rounded to the nearest member of the floating point number system.

(b) In solving a system of n equations by Gaussian elimination, partial pivoting is recommended in order to reduce the operation count.

False. Partial pivoting is used to avoid ill-conditioning; it actually increases the operation count.

(c) For a general square matrix A with factorization $PA = LU$, $\det A = \det U$.

False. Permuting the rows of A during partial pivoting will change the sign of the determinant, hence $\det A = \pm \det U$.

(d) For a symmetric matrix A with factorization $A = LU$, $\det A = \det U$.

True. Symmetric matrices do not require pivoting.

(e) Let $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. If $|a| > |b|$ and $|d| > |c|$, then Jacobi's method for solving $Ax = b$ will converge.

True. $|a| > |b|$ and $|d| > |c|$ implies that A is diagonally dominant; hence, the Jabobi method will converge.

(f) For $A = \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 5 & 6 \\ 1 & 0.1 & 8 \end{pmatrix}$, the first step of factorizing $A \mapsto PA = LU$ will be to left-multiply A by $P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

False. The first step would be $P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$. Partial pivoting finds the largest element (not just a nonzero element) in the column and performs a row exchange with that element's location.

(g) Multiplying a matrix A by a scalar α does not change the condition number (i.e., $\kappa(\alpha A) = \kappa(A)$).

True. $\kappa(A) = \|A\| \cdot \|A^{-1}\|$

$\|\alpha A\| = |\alpha| \cdot \|A\|$

$(\alpha A)^{-1} = \frac{1}{\alpha} A^{-1} \Rightarrow \|(\alpha A)^{-1}\| = \frac{1}{|\alpha|} \cdot \|A^{-1}\|$

$\kappa(\alpha A) = |\alpha| \, \|A\| \cdot \frac{1}{|\alpha|} \, \|A^{-1}\| = \kappa(A)$

(h) It is impossible to estimate error without knowing the exact solution.

False. Error can be estimated using error bounds, or theorems involving successive terms, condition number, etc.
You might say that this is one of the huge themes of our class – if you know the exact answer, you probably don't need a numerical method!

2. **(18 points)** Secant method

(a) List the errors (if any) in the following algorithm for the secant method, and a way to correct them.

Algorithm 1: Secant method

Input : $f(x)$, x_0, ϵ_{tol}
Output: $x \approx p$ where $f(p) = 0$

1 $x_k = x_1$
2 $x_{k-1} = x_0$
3 $k = 2$
4 **while** $|x_k + x_{k-1}| < \epsilon_{tol}$ **do**
5 $\quad x_{k+1} = x_k - \dfrac{f(x_k) \cdot (x_k - x_{k-1})}{f'(x_k) - f'(x_{k-1})}$ `% current estimate of root`
6 $\quad x_{k-1} = x_k$
7 $\quad x_k = x_{k+1}$
8 $\quad k = k + 2$
9 **end**

1. Input x_1 is also required
2. Line 3 should be $k = 1$
3. Line 4 should be `while` $|x_k - x_{k-1}| > \epsilon_{\text{tol}}$ `do`
4. Line 5 should have $(f(x_k) - f(x_{k-1}))$ in the denominator, not f'
5. Line 8 should be $k = k + 1$
6. There is no protection against an infinite loop if the convergence criterion is not met. The maximum number of iterations $k_{\max}$ should be another input, and there should be another stopping criterion. This can be achieved by adding

`if` $k \geq k_{\max}$ `then stop`

inside the loop after line 8.
This is particularly important since the secant method (unlike bisection) is not guaranteed to converge.

The following Matlab code implements the revised algorithm.

```
clear;
% Secant Method
%% INPUT

fnstring = 'log(x)';
f = inline(fnstring);

x0 = 0.5;
x1 = 1.25;

tol = 1e-8;
maxIter = 1000;

exact = 1;
%% ALGORITHM
xkMinus1 = x0;
xk = x1;
fxkMinus1 = f(x0);

k = 1;

while abs(xk - xkMinus1) > tol
    % cost for current iteration
    fxk = f(xk);
    % calculate next estimate
    xkPlus1 = xk - fxk * (xk - xkMinus1) / ( fxk - fxkMinus1);
    % reset for next iteration
    fxkMinus1 = fxk;
    xkMinus1 = xk;
    xk = xkPlus1;

    datastring1 = sprintf('k = %d : x_k = %18.15f,   ',k,xkMinus1);
    datastring2 = sprintf(' e_k = %18.15f',abs(exact-xkMinus1));
    disp(strcat(datastring1,datastring2));
    k = k+1;
    if k == maxIter
        break
    end
end
if k >= maxIter
    errstring = sprintf('no convergence after %d steps',k);
```

```
    disp(errstring);

else
    convstring = sprintf('convergence achieved at step %d',k-1);
    disp(convstring);
    rootstring = sprintf(' x = %24.15f',xk);
    tolstring = sprintf(' tol = %18.15f',tol);
    disp([fnstring,' has a zero at ',rootstring,...
          ' plus/minus',tolstring]);
end
```

(b) What is the main cost per iteration? What are the minimum storage requirements? List the scalars that need to be stored.

The main cost is one evaluation of $f(x)$ per step. The minimum storage requirements would keep x_{k-1}, x_k, x_{k+1}, f_{k-1} and f_k. As written, the algorithm also stores k and the inputs ϵ_{tol} and $k_{\max}$.

(c) State one advantage of the secant method over Newton's method. State one advantage of Newton's method over the secant method.

The secant method does not require knowledge of the derivative; Newton's method does.
The secant method only requires one function evaluation per step; it therefore has a smaller cost per iteration than Newton's method.
Newton's method converges faster than the secant method.

(d) Test your revised algorithm on the function $f(x) = \ln(x)$. Fill in the table below using initial guesses $x_0 = 0.5$ and $x_1 = 1.25$.

k	x_k	$e_k = \lvert x_k - p \rvert$
0	0.5	0.5
1	1.25	0.25
2	1.06735	0.06735
3	0.991985	0.008015

3. **(10 points)** Let λ_1 denote the largest eigenvalue of $A = \begin{pmatrix} 10 & 1 \\ 1 & 100 \end{pmatrix}$. Use the power method to estimate λ_1 by performing two iterations using $x = \frac{1}{\sqrt{2}}[1,1]^T$ for the initial vector.

Hint: Note that this matrix is *almost* diagonal, that is, its off-diagonal elements are $\leq 10\%$ of its diagonal elements. You might guess, therefore, that its eigenvalues will be pretty close to 10 and 100, as they would be if $A = \begin{pmatrix} 10 & 0 \\ 0 & 100 \end{pmatrix}$.

k	λ_k
0	56
1	99.1602
2	100.002

$$\lambda_0 = \frac{x_0^T A x_0}{x_0^T x_0} = 56 \text{ or } 55.99999999$$

$$Ax_0 = \begin{pmatrix} 10 & 1 \\ 1 & 100 \end{pmatrix} \begin{pmatrix} 0.7071968 \\ 0.7071968 \end{pmatrix} = \begin{pmatrix} 7.7781746 \\ 71.417785 \end{pmatrix}, \quad x_1 = \frac{1}{\|Ax_0\|} Ax_0 = \begin{pmatrix} 0.1082706 \\ 0.9941214 \end{pmatrix}$$

$$\lambda_1 = \frac{x_1^T A x_1}{x_1^T x_1} = 99.1602402$$

$$Ax_1 = \begin{pmatrix} 10 & 1 \\ 1 & 100 \end{pmatrix} \begin{pmatrix} 0.1082706 \\ 0.9941214 \end{pmatrix} = \begin{pmatrix} 2.076827988 \\ 99.52041607 \end{pmatrix}, \quad x_2 = \frac{1}{\|Ax_1\|} Ax_1 = \begin{pmatrix} 0.020863819 \\ 0.997823268 \end{pmatrix}$$

$$\lambda_2 = \frac{x_2^T A x_2}{x_2^T x_2} = 100.00254165$$

4. **(5 points)** The data below comes from computing the derivative $f'(1)$ using a finite difference approximation with step size h. The exact value is known to be $f'(1) = 3$. Which approximation was used, $D_+ f(1)$ or $D_0 f(1)$?

h	approximation
0.4	3.1599999
0.2	3.0399999
0.1	3.0100000
0.05	3.0025000
0.025	3.0006240

h	error	error/h	error/h^2
0.4	0.1599999	0.3999999	0.999999
0.2	0.0399999	0.1999999	0.999999
0.1	0.0100000	0.1000000	1.000000
0.05	0.0025000	0.0500000	1.000000
0.025	0.0006240	0.0249999	0.999999

Since error/h^2 is approximately constant, the rate of convergence of this approximation is $O(h^2)$; of the two choices, only $D_0 f(x)$ is second order accurate, so the approximations are given by $D_0 f(1)$.

Alternatively, note that each time the step size is decreased by a factor of 2, the error decreases by a factor of 4 $\Rightarrow$ second order convergence.

5. **(10 points)** Consider the fixed point iteration $x_{n+1} = g(x_n)$ with $g(x) = \frac{1}{4}x^2 - \frac{1}{4}x + \frac{3}{2}$.

   (a) Find the fixed points of $g(x)$.

   Fixed points satisfy $x = g(x)$.

$$x = \frac{1}{4}x^2 - \frac{1}{4}x + \frac{3}{2}$$
$$0 = \frac{1}{4}x^2 - \frac{5}{4}x + \frac{3}{2}$$
$$0 = x^2 - 5x + 6 = (x-2)(x-3)$$
$$\Rightarrow \text{fixed points at } x = 2, 3$$

(b) If the starting guess is $x_0 = \frac{3}{2}$, find $\lim_{n\to\infty} x_n$.

$$g'(x) = \frac{1}{2}x - \frac{1}{4}$$
$$g'(2) = 1 - \frac{1}{4} = \frac{3}{4} < 1$$
$$g'(3) = \frac{3}{2} - \frac{1}{4} = \frac{5}{4} > 1$$

This implies that the fixed point iteration will diverge from the $x = 3$ fixed point, and converge toward $x = 2$, since $|g'(2)| < 1$.

$$\lim_{n\to\infty} x_n = 2$$

6. **(25 points)** Consider the linear system defined by $A = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$, $b = \begin{pmatrix} 6 \\ -6 \end{pmatrix}$.

(a) Solve for x by Gaussian elimination.

$$\left(\begin{array}{cc|c} 4 & 1 & 6 \\ 1 & 4 & -6 \end{array}\right) \mapsto \left(\begin{array}{cc|c} 4 & 1 & 6 \\ 0 & \frac{15}{4} & -\frac{15}{2} \end{array}\right) \Rightarrow x_2 = -2, \quad 4x_1 - 2 = 6 \Rightarrow x_1 = 2$$

$$x = \begin{pmatrix} 2 \\ -2 \end{pmatrix}$$

(b) Find the LU factorization of A.

From part (a), we know that $m_{21} = \frac{1}{4} \Rightarrow M_1 = \begin{pmatrix} 1 & 0 \\ -\frac{1}{4} & 1 \end{pmatrix}$

$$M_1 A = \begin{pmatrix} 1 & 0 \\ -\frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 0 & \frac{15}{4} \end{pmatrix} = U$$

$$L = \begin{pmatrix} 1 & 0 \\ \frac{1}{4} & 1 \end{pmatrix}$$

(c) Write Jacobi's method in component form and take one step starting from the zero vector.

$$A = L + D + U, \quad L = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}, \quad U = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$Dx^{(k+1)} = -(L+U)x^{(k)} + b$$

$$\begin{aligned} 4x_1^{(k+1)} &= 6 - x_2^{(k)} & &\Rightarrow x_1^{(1)} = \frac{3}{2} \\ 4x_2^{(k+1)} &= -6 - x_1^{(k)} & &\Rightarrow x_2^{(1)} = -\frac{3}{2} \end{aligned}$$

(d) Write the Gauss-Seidel method in component form and take one step starting from the zero vector.

$$(L+D)x^{(k+1)} = -Ux^{(k)} + b$$

$$\begin{aligned} 4x_1^{(k+1)} &= 6 - x_2^{(k)} & &\Rightarrow x_1^{(1)} = \frac{3}{2} \\ 4x_2^{(k+1)} &= -6 - x_1^{(k+1)} & &\Rightarrow x_2^{(1)} = \frac{1}{4}\left(-6 - \frac{3}{2}\right) = -\frac{15}{8} = -1.875 \end{aligned}$$

(e) Find the iteration matrices B_J, B_{GS}. Which method converges faster?

$$B_J = -D^{-1}(L+U) = -\begin{pmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{1}{4} \\ -\frac{1}{4} & 0 \end{pmatrix}$$

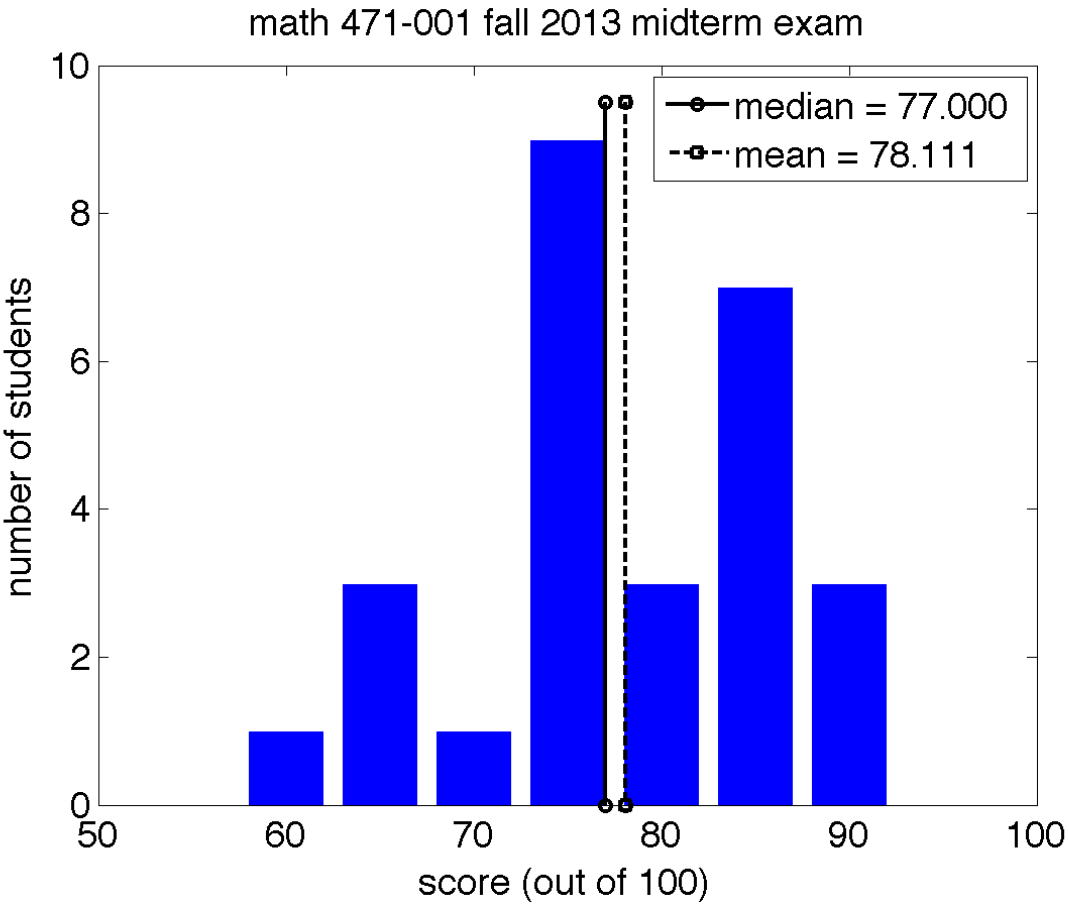
$$\|B_J\|_\infty = \frac{1}{4}$$

$$\det\begin{pmatrix} -\lambda & -\frac{1}{4} \\ -\frac{1}{4} & -\lambda \end{pmatrix} = \lambda^2 - \frac{1}{16} \Rightarrow \rho(B_J) = \frac{1}{4}$$

$$B_{GS} = -(L+D)^{-1}U; \quad L + D = \begin{pmatrix} 4 & 0 \\ 1 & 4 \end{pmatrix} \Rightarrow (L+D)^{-1} = \begin{pmatrix} \frac{1}{4} & 0 \\ -\frac{1}{16} & \frac{1}{4} \end{pmatrix}$$

$$B_{GS} = \begin{pmatrix} \frac{1}{4} & 0 \\ -\frac{1}{16} & \frac{1}{4} \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{4} \\ 0 & -\frac{1}{16} \end{pmatrix} \quad \Rightarrow \|B_{GS}\|_\infty = \frac{1}{4}, \ \rho(B_{GS}) = \frac{1}{16}$$

Since $\rho(B_{GS}) = \frac{1}{16} < \rho(B_J) = \frac{1}{4}$, the Gauss-Seidel method converges faster as $k \to \infty$.


math 471-001 fall 2013 midterm exam
median = 77.000
mean = 78.111
number of students
score (out of 100)
0
2
4
6
8
10
50
60
70
80
90
100