Summer 2025

MATH 4650-E01
CSCI 4650-E01
MATH 5660-E01

Homework 3

Due on Monday June 30th 2025

Exercises:

1. EX.2.1.2.b2c handwritten or typed
2. EX.2.1.6 handwritten or typed
3. EX.2.1.8 handwritten or typed
4. EX.2.2.2.b2c handwritten or typed
5. EX.2.2.4.b handwritten or typed
6. EX.2.2.6 handwritten or typed
7. EX.2.2.8 handwritten or typed
8. CP.2.2.1.b2c colab
9. CP.2.2.2.b colab
10. EX.2.3.2.b2c handwritten or typed
11. EX.2.3.6 colab
12. CP.2.3.4 colab
13. EX.2.4.2.c2d handwritten or typed
14. EX.2.4.4 handwritten or typed
15. EX.2.7.2.b handwritten or typed
16. CP.2.7.1.b2c colab
17. CP.2.7.4 colab
18. CP.2.7.5.b colab
19. CP.2.7.7.b2c colab
20. CP.2.7.8.b2c colab

EX.2.1.2.b2c, Sauer3

Use Gaussian elimination to solve the systems:

b.

$$\begin{aligned} x + 2y - z &= 2 \\ 3y + z &= 4 \\ 2x - y + z &= 2 \end{aligned}$$

c.

$$\begin{aligned} 2x + y - 4z &= -7 \\ x - y + z &= -2 \\ -x + 3y - 2z &= 6 \end{aligned}$$

EX.2.1.6, Sauer3

Assume that your computer completes a 5000 equation back substitution in 0.005 seconds. Use the approximate operation counts n^2 for back substitution and $2n^3/3$ for elimination to estimate how long it will take to do a complete Gaussian elimination of this size. Round your answer to the nearest second.

EX.2.1.8, Sauer3

If a system of 3000 equations in 3000 unknowns can be solved by Gaussian elimination in 5 seconds on a given computer, how many back substitutions of the same size can be done per second?

EX.2.2.2.b2c, Sauer3

Find the LU factorization of the given matrices. Check by matrix multiplication.

(b) $\begin{pmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{pmatrix}$ (c) $\begin{pmatrix} 1 & -1 & 1 & 2 \\ 0 & 2 & 1 & 0 \\ 1 & 3 & 4 & 4 \\ 0 & 2 & 1 & -1 \end{pmatrix}$

Note: the LU factorization for these two matrices is done using Python in CP.2.2.1.b2c.

EX.2.2.4.b, Sauer3

Solve the system by finding the LU factorization and then carrying out the two-step back substitution

$$\text{(b)} \quad \begin{pmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

Note 1: the LU factorization for this matrix was found in EX.2.2.2.b.
Note 2: the two-step back substitution for this linear system of equations is done using Python in CP.2.2.2.b.

EX.2.2.6, Sauer3

Given the 1000×1000 matrix A, your computer can solve the 500 problems $Ax = b_1, \ldots, Ax = b_{500}$ in exactly one minute, using $A = LU$ factorization methods. How much of the minute was the computer working on the $A = LU$ factorization? Round your answer to the nearest second.

EX.2.2.8, Sauer3

Assume that your computer can solve a 2000×2000 linear system $Ax = b$ in 0.1 second. Estimate the time required to solve 100 systems of 8000 equations in 8000 unknowns with the same coefficient matrix, using the LU factorization method.
Clarification: Please assume that the 100 systems all have the same coefficient matrix A, but different right hand sides. So we want to solve 100 problems $Ax = b_1, \ldots, Ax = b_{100}$. (Same A, 100 different b's.)

CP.2.2.1.b2c, Sauer3

Use code fragments for Gaussian elimination in the previous section to write a python script to take a matrix A as input and output L and U. No row exchanges are allowed—the program should be designed to shut down if it encounters a zero pivot. Check your program by factoring the matrices in EX.2.2.2.b2c.

(b) $\begin{pmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{pmatrix}$ (c) $\begin{pmatrix} 1 & -1 & 1 & 2 \\ 0 & 2 & 1 & 0 \\ 1 & 3 & 4 & 4 \\ 0 & 2 & 1 & -1 \end{pmatrix}$

CP.2.2.2.b, Sauer3

Add two-step back substitution to your script from CP.2.2.1, and use it to solve the systems in EX.2.2.4.b.

$$\text{(b)} \quad \begin{pmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

EX.2.3.2.b2c, Sauer3

Find the infinity norm condition number of

(b) $A = \begin{pmatrix} 1 & 2.01 \\ 3 & 6 \end{pmatrix}$ (c) $A = \begin{pmatrix} 6 & 3 \\ 4 & 2 \end{pmatrix}$

EX.2.3.6, Sauer3

Please do this problem using Colab. Do not do this by hand.

Find the relative forward and backward errors and error magnification factor for the following approximate solutions of the system

$$\begin{cases} x_1 + 2\,x_2 = 3 \\ 2\,x_1 + 4.01\,x_2 = 6.01 \end{cases}$$

(a) $\begin{pmatrix} -10 \\ 6 \end{pmatrix}$ (b) $\begin{pmatrix} -100 \\ 52 \end{pmatrix}$ (c) $\begin{pmatrix} -600 \\ 301 \end{pmatrix}$ (d) $\begin{pmatrix} -599 \\ 301 \end{pmatrix}$

(e) What is the condition number of the coefficient matrix?

CP.2.3.4, Sauer3

Let A be the n-by-n matrix with entries

$$a_{ij} = \sqrt{(i-j)^2 + \frac{n}{10}}.$$

Define $x = (1, \ldots, 1)^T$ and $b = Ax$. For $n = 100$, 200, 300, 400, and 500, use the python program from Computer Problem 2.1.1 or Numpy's `numpy.linalg.solve` command to compute `xc`, the double precision computed solution. Calculate the infinity norm of the forward error for each solution. Find the five error magnification factors of the problems $Ax = b$, and compare with the corresponding condition numbers.

Hint: Note that this $a_{ij} = \sqrt{(i-j)^2 + \frac{n}{10}}$ formula is the same with 0-base indexing (Python) or 1-base indexing (Sauer and Matlab). Here is a code snippet to generate A with $n = 5$.

```
from math import sqrt
n = 5
A = np.zeros( [ n, n ], dtype=float )
for i in range(0,n):
  for j in range(0,n):
      A[i,j] = sqrt( ( i - j )**2 + n / 10. )
print(A)
```

```
[[0.70710678 1.22474487 2.12132034 3.082207   4.0620192 ]
 [1.22474487 0.70710678 1.22474487 2.12132034 3.082207  ]
 [2.12132034 1.22474487 0.70710678 1.22474487 2.12132034]
 [3.082207   2.12132034 1.22474487 0.70710678 1.22474487]
 [4.0620192  3.082207   2.12132034 1.22474487 0.70710678]]
```

EX.2.4.2.c2d, Sauer3

Find the $PA = LU$ factorization (using partial pivoting) of the following matrices:

c. $\begin{pmatrix} 1 & 2 & -3 \\ 2 & 4 & 2 \\ -1 & 0 & 3 \end{pmatrix}$

d. $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ -2 & 1 & 0 \end{pmatrix}$

EX.2.4.4, Sauer3

Solve the system by finding the $PA = LU$ factorization and then carrying out the two-step back substitution.

a. $\begin{pmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$

b. $\begin{pmatrix} -1 & 0 & 1 \\ 2 & 1 & 1 \\ -1 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -2 \\ 17 \\ 3 \end{pmatrix}$

Hint: The two $PA = LU$ factorization (with partial pivoting) are:

a.
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1/2 & 1/2 & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & 2 \end{pmatrix}$$

b.
$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} -1 & 0 & 1 \\ 2 & 1 & 1 \\ -1 & 2 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/2 & 1/5 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 \\ 0 & 5/2 & 1/2 \\ 0 & 0 & 7/5 \end{pmatrix}$$

EX.2.7.2.b, Sauer3

Use the Taylor expansion to find the linear approximation $L(x)$ to $F(x)$ near x_0.

$$\text{(b)} \left(\begin{array}{ccccc} F: & \mathbb{R}^2 & \to & \mathbb{R}^2 \\ & \begin{pmatrix} u \\ v \end{pmatrix} & \mapsto & \begin{pmatrix} u + e^{u-v} \\ 2u + v \end{pmatrix} \end{array} \right) \qquad \text{at } x_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

CP.2.7.1.b2c, Sauer3

Implement Newton's Method with appropriate starting points to find all solutions. Check with EX.2.7.3 to make sure your answers are correct.

$$\text{(b)} \begin{cases} u^2 + 4v^2 &= 4 \\ 4u^2 + v^2 &= 4 \end{cases} \qquad \text{(c)} \begin{cases} u^2 - 4v^2 &= 4 \\ (u-1)^2 + v^2 &= 4 \end{cases}$$

Hint from Sauer from his solution as in EX.2.7.3:

b. The curves are ellipses with semimajor axes 1 and 2 centered at zero and aligned with the x and y axes. Solving by substitution gives the four solutions

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}, \quad \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} = \begin{pmatrix} -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}, \quad \begin{pmatrix} u_3 \\ v_3 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{5}} \\ -\frac{2}{\sqrt{5}} \end{pmatrix}, \quad \begin{pmatrix} u_4 \\ v_4 \end{pmatrix} = \begin{pmatrix} -\frac{2}{\sqrt{5}} \\ -\frac{2}{\sqrt{5}} \end{pmatrix}.$$

c. The curves are a hyperbola and a circle that intersects one half of the hyperbola in two points. Solving by substitution gives the two solutions

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix} = \begin{pmatrix} \frac{4}{5}\left(1+\sqrt{6}\right) \\ \frac{1}{5}\sqrt{3+8\sqrt{6}} \end{pmatrix}, \quad \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} = \begin{pmatrix} \frac{4}{5}\left(1+\sqrt{6}\right) \\ -\frac{1}{5}\sqrt{3+8\sqrt{6}} \end{pmatrix}.$$

Special Instructions:

a. Please first compute the two solutions using either Sauer's exact solutions (see Hint above) or `scipy.optimize.fsolve` or `scipy.optimize.root`. This will be useful to compute the forward error.

b. Start Newton's method from a relative distance (as measured by the infinity norm) of at least 0.1 from the solution.

c. At each iteration of Newton's method, you must print:

(a) k, the iteration number

(b) the absolute backward error at iteration number k defined by

$$\|F(x_k)\|_\infty$$

where x_k is the current iterate.

(c) the relative forward error at iteration number k defined by

$$\|x_k - x\|_\infty / \|x\|_\infty$$

where x_k is the current iterate and x is the solution as computed by `scipy.optimize.fsolve` or `scipy.optimize.root`.

You can also print the current iterate x_k if you want.

CP.2.7.4, Sauer3

Apply Newton's Method to find both solutions of the system of three equations.

$$\begin{cases} 2u^2 - 4u + v^2 + 3w^2 + 6w + 2 &= 0 \\ u^2 + v^2 - 2v + 2w^2 - 5 &= 0 \\ 3u^2 - 12u + v^2 + 3w^2 + 8 &= 0 \end{cases}$$

Special Instructions:

a. Please first compute the two solutions using either `scipy.optimize.fsolve` or `scipy.optimize.root`. This will be useful to compute the forward error.

b. Start Newton's method from a relative distance (as measured by the infinity norm) of at least 0.1 from the solution.

c. At each iteration of Newton's method, you must print:

(a) k, the iteration number

(b) the absolute backward error at iteration number k defined by

$$\|F(x_k)\|_\infty$$

where x_k is the current iterate.

(c) the relative forward error at iteration number k defined by

$$\|x_k - x\|_\infty / \|x\|_\infty$$

where x_k is the current iterate and x is the solution as computed by `scipy.optimize.fsolve` or `scipy.optimize.root`.

You can also print the current iterate x_k if you want.

Hint: Here are the two roots:

$$\begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1.0960178410014330 \\ -1.1592471842154839 \\ -0.2611479367011116 \end{pmatrix}.$$

(Approximation for the second one.)

Hint:
Here are some function values and Jacobian function values that you can check before starting implementing Newton's method

$$F(\begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}) = \begin{pmatrix} 25 \\ 7 \\ 12 \end{pmatrix}, \quad F(\begin{pmatrix} 3 \\ 5 \\ -4 \end{pmatrix}) = \begin{pmatrix} 57 \\ 51 \\ 72 \end{pmatrix},$$

$$DF(\begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}) = \begin{pmatrix} 0 & -2 & 18 \\ 2 & -4 & 8 \\ -6 & -2 & 12 \end{pmatrix}, \quad DF(\begin{pmatrix} 3 \\ 5 \\ -4 \end{pmatrix}) = \begin{pmatrix} 8 & 10 & -18 \\ 6 & 8 & -16 \\ 6 & 10 & -24 \end{pmatrix}$$

CP.2.7.5.b, Sauer3

Use multivariate Newton's Method to find the two points in common of the three given spheres in three-dimensional space.

b. Each sphere has radius 5, with centers $\left(\ 1, \ \ -2, \ \ 0 \ \right)$, $\left(\ -2, \ \ 2, \ \ -1 \ \right)$, and $\left(\ 4, \ \ -2, \ \ 3 \ \right)$.

CP.2.7.7.b2c, Sauer3

Apply Broyden I with starting guesses $x_0 = (1, 1)$ and $A_0 = I$ to the systems in EX.2.7.3. Report the solutions to as much accuracy as possible and the number of steps required.

(b) $\begin{cases} u^2 + 4v^2 &= 4 \\ 4u^2 + v^2 &= 4 \end{cases}$ (c) $\begin{cases} u^2 - 4v^2 &= 4 \\ (u-1)^2 + v^2 &= 4 \end{cases}$

CP.2.7.8.b2c, Sauer3

Apply Broyden II with starting guesses $x_0 = (1, 1)$ and $B_0 = I$ to the systems in EX.2.7.3. Report the solutions to as much accuracy as possible and the number of steps required.

(b) $\begin{cases} u^2 + 4v^2 &= 4 \\ 4u^2 + v^2 &= 4 \end{cases}$ (c) $\begin{cases} u^2 - 4v^2 &= 4 \\ (u-1)^2 + v^2 &= 4 \end{cases}$