HW 9

ENGR 21, Fall 2026.

November 12, 2026

Due Date	Thu, Nov 12
Turn in link	Gradescope PDF and Code
URL	`emadmasroor.github.io/E21-F26/Homework/HW9`

What to turn in

Combine all written HW into one PDF file.

Problem	Format
1.1	PDF
1.2	Python function named `eliminate_pivot`
2	Python function named `Jacobi`
3.1	Python function named `GaussSeidel`
3.2	PDF
4	Python code for generating plots
4	PDF of (graphs and short text)

Changelog:

- Slight modification to requirements of part 2 of 1.1.3, added Monday 4:30 PM.

1 Pivoting

1.1 By hand

Consider the matrix

$$A = \begin{bmatrix} 0 & 2 & 5 & -1 \\ 2 & 1 & 3 & 0 \\ -2 & -1 & 4 & 1 \\ 3 & 7 & -1 & 2 \end{bmatrix}$$

If this matrix is decomposed into L and U following Doolittle's factorization, determine the upper triangular matrix U, showing **all** intermediate stages of the matrix in the Gaussian elimination procedure and describing what each step accomplishes.

1.2 Python program for pivoting

Write a Python function named `eliminate_pivot` that takes as input a single matrix and performs Gaussian elimination with pivoting on that matrix.

Using the array defined here

```
A = np.array([[4  ,2   ,5   ,-1 ,0   ],
              [-3 ,1   ,0   ,9  ,0.1 ],
              [3.2,3   ,-4.3,6  ,-2  ],
              [0.5,1   ,5.1 ,.3 ,-7.3],
              [9.3,-1.4,4   ,2.1,6.0 ]
             ],dtype=np.float64)
```

your procedure should be numerically equal, or approximately equal, to the following:

```
array([[ 9.30e+00, -1.40e+00,  4.00e+00,  2.10e+00,  6.00e+00],
       [ 0.00e+00,  2.60e+00,  3.28e+00, -1.90e+00, -2.58e+00],
       [ 0.00e+00,  0.00e+00, -1.01e+01,  7.82e+00, -6.12e-01],
       [ 0.00e+00,  0.00e+00,  0.00e+00,  1.05e+01,  2.54e+00],
       [ 0.00e+00,  0.00e+00,  4.44e-16,  0.00e+00, -7.67e+00]])
```

2 The Jacobi Algorithm

Jacobi with `while` loop

In lecture, we have seen an algorithm for Jacobi's method that carries out `n_steps` iterations in a `for` loop. Write a Python function named `Jacobi` that instead makes use of a `while` loop, terminating the iterations when the norm of the residual is less than `tol`.

Your function should take as input the following quantities in order: A, the system matrix; b, the right-hand-side vector; $x^{(0)}$, the first guess for x, and `tol`, a tolerance parameter that is typically set to a very small number such as 10^{-6}. The function should return the most recent vector $x^{(k)}$.

> **Tip**
>
> You can use `b - numpy.dot(A,x)` to calculate the vector $b - Ax$.

> **Tip**
>
> You may wish to check your code on the following matrix and right-hand side:

```
A = np.array([[4   ,1 ,2  ],
              [-1/2,3 ,3/2],
              [0   ,2 ,4  ]],dtype=np.float64)
b = np.array([1,
              2,
              1],dtype=np.float64)
```

According to my solution, calling the function on this A and b with an initial guess of `[1, 1, 1]` and a tolerance parameter of `1e-6` gives:

```
>>> Jacobi(A,b,np.array([1,1,1],dtype=np.float64),1e-6)
array([ 0.12499984,  0.7499996 , -0.12499963])
```

and for a different set of inputs, we get

```
>>> Jacobi(A,b,np.array([0,0,0],dtype=np.float64),1e-3)
array([ 0.1249076 ,  0.74977215, -0.12478681])
```

3 Gauss-Seidel

As we saw in lecture 17, the Gauss-Seidel algorithm modifies the Jacobi algorithm so that it makes use of the new x values as they are computed. The first row's update is based entirely on the old values; the second row's update is based on the old values for x_3, x_4, ..., x_n, but it makes use of the newly-computed value for x_1; the third row's update is based on the old values for x_4, x_5, ..., x_n, but it makes use of the newly-computed value for x_1 and x_2.

3.1 Python code

Write a Python function named `GaussSeidel` that carries out a specified number of iterations of the Gauss-Seidel algorithm.

Your function should take as input the following quantities in order: A, the system matrix; b, the right-hand-side vector; $x^{(0)}$, the first guess for x, and n, the number of iterations. The function should return the most recent vector $x^{(k)}$.

3.2 Gauss-Seidel vs Jacobi

You will solve the same equation $Ax = b$ iteratively using the two iterative methods, Gauss-Seidel and Jacobi. You are welcome to use a computer to help you with this problem, but your answers should be written out by hand.

Solve the system of equations represented by the augmented matrix

$$\left[\begin{array}{ccc|c} 6 & 3 & 1 & 1 \\ 2 & 7 & 4 & 0 \\ 5 & 1 & 8 & 0 \end{array}\right]$$

starting with the guess $x_1 = 1, x_2 = 0, x_3 = 2$ using:

1. Two iterations of the Gauss-Seidel algorithm, and
2. Two iterations of the Jacobi algorithm.

For each iteration, show all your work, and also calculate and write down the norm of the residual vector at the end of each iteration.

4 Convergence of Iterative Methods

The norm of the residual vector after each iteration gives us information about how close to the true solution a numerical method gets after n iterations. In this problem, you will examine this for a simple 3x3 system (which could easily be solved exactly by direct methods) to illustrate the convergence properties of the Jacobi and Gauss-Seidel algorithms.

In practice, engineers need to solve systems of equations that have thousands or millions of rows, and the speed with which a numerical method can get close to the right answer becomes extremely important.

For the system represented by the augmented matrix

$$\left[\begin{array}{ccc|c} 6 & 3 & 1 & 1 \\ 2 & 7 & 4 & 0 \\ 5 & 1 & 8 & 0 \end{array}\right],$$

starting from the guess $[1, 0, 2]$,

1. Make a plot of the norm of the residual vector after 10 iterations of the Gauss-Seidel method and of the Jacobi method.
2. Make a plot of the norm of the residual vector after 1,000 iterations of the two methods on a log-log scale. Use `matplotlib.pyplot.xscale("log")` and `matplotlib.pyplot.yscale("log")`.
3. Describe and explain what these graphs tell us. You may wish to look at lecture 8.

You must make your plots in `matplotlib` and include the code with your submission.