HOMEWORK # 12

04/25/2025

---

1 Question 1

For Question 1, we are given the following system of equations found in (1).

$$\begin{aligned} 6x + 2y + 2z &= -2 \\ 2x + \frac{2}{3}y + \frac{1}{3}z &= 1 \\ x + 2y - z &= 0 \end{aligned} \tag{1}$$

Using the above system, we are asked to verify a given solution to the system as well as use Gaussian Elimination with and without partial pivoting to solve the system. We are then asked to compare the results and look at each method's stability.

1.1 Verifying Exact Solution

For this part, we are given $(x, y, z) = (2.6, -3.8, -5)$ and are asked to verify that it is the exact solution to the system in (1). To do so, we can first re-write the system in (1) in matrix the matrix form $\mathbf{A}\vec{x} = \vec{b}$ where

$$\mathbf{A} = \begin{bmatrix} 6 & 2 & 2 \\ 2 & \frac{2}{3} & \frac{1}{3} \\ 1 & 2 & -1 \end{bmatrix}, \vec{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \text{ and } \vec{b} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}$$

By plugging in the given values of $\vec{x}$ for $\mathbf{A}\vec{x}$, we have the following.

$$\begin{aligned} \begin{bmatrix} 6 & 2 & 2 \\ 2 & \frac{2}{3} & \frac{1}{3} \\ 1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 2.6 \\ -3.8 \\ -5 \end{bmatrix} &= \begin{bmatrix} 6(2.6) + 2(-3.8) + 2(-5) \\ 2(2.6) + \frac{2}{3}(-3.8) + \frac{1}{3}(-5) \\ 2.6 + 2(-3.8) + 5 \end{bmatrix} \\ &= \begin{bmatrix} 15.6000 - 7.6000 - 10.0000 \\ 5.2000 - 2.5333 - 1.6667 \\ 2.6000 - 7.6000 + 5.0000 \end{bmatrix} \\ &= \begin{bmatrix} -2.0000 \\ 1.0000 \\ 0.0000 \end{bmatrix} \end{aligned}$$

As shown above, the given vector, $\vec{x}$, is the exact solution to the system in (1). This is further verified using a python script which uses `Numpy`. The script can be found on the GitHub repository under `Lab_12` directory.

1.2 Gaussian Elimination

For this part, we are asked to solve the system in (1) using Gaussian Elimination while making sure to use four-digit floating point arithmetic with rounding. For Gaussian Elimination, the system in (1) was converted to an augmented matrix form as shown below.

$$\mathbf{A}|\vec{b} = \left[\begin{array}{ccc|c} 6 & 2 & 2 & -2 \\ 2 & \frac{2}{3} & \frac{1}{3} & 1 \\ 1 & 2 & -1 & 0 \end{array}\right]$$

Algorithm 1 Gaussian Elimination Without Pivoting

Require: Matrix $A \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$
Ensure: Solution vector x s.t $Ax = b$

```
for i = 1 to n − 1 do
    for j = i + 1 to n do
        m ← A[j][i]/A[i][i]
        for k = i to n do
            A[j][k] ← A[j][k] − m · A[i][k]
        end for
        b[j] ← b[j] − m · b[i]
    end for
end for
                                                    ▷ Back substitution
for i = n to 1 step −1 do
    sum ← 0
    for j = i + 1 to n do
        sum ← sum + A[i][j] · x[j]
    end for
    x[i] ← (b[i] − sum)/A[i][i]
end for
```

Using the augmented matrix, the following algorithm was followed in order to perform Gaussian Elimination without pivoting. Using the above algorithm, a python script was written to perform the Gaussian Elimination on the augmented matrix from (1). The python script produced the following output in the form of the augmented matrix.

$$\mathbf{A}|b = \left[\begin{array}{ccc|c} 6.0000e+00 & 2.0000e+00 & 2.0000e+00 & -2.0000e+00 \\ 2.0000e-05 & 1.0000e-05 & -3.3333e-01 & 1.6667e+00 \\ -2.0000e-05 & 0.0000e+00 & 5.5555e+04 & -2.7779e+05 \end{array}\right]$$

As shown, the results of the Gaussian Elimination is in upper-triangular form. Additionally, the resulting solution that was found was $\vec{x} = \begin{bmatrix} 1.3335 & 0. & -5.0003 \end{bmatrix}^T$ which differs substantially from the solution we verified in part (1) of this question.

1.3 Partial Pivoting

For this part, we are asked to repeat the process in the previous part but while using partial pivoting. Similar to the previous section, a python script was developed to aid in computing the Gaussian Elimination with pivots. The code that was produced follows the following algorithm. Using the

Algorithm 2 Gaussian Elimination With Partial Pivoting

Require: Matrix $A \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$
Ensure: Solution vector x such that $Ax = b$

for $i = 1$ to $n - 1$ **do**
  $max\_row \leftarrow$ row index of maximum $|A[j][i]|$ for $j = i$ to n
  if $max\_row \neq i$ **then**
    Swap rows $A[i] \leftrightarrow A[max\_row]$
    Swap entries $b[i] \leftrightarrow b[max\_row]$
  end if
  for $j = i + 1$ to n **do**
    $m \leftarrow A[j][i]/A[i][i]$
    for $k = i$ to n **do**
      $A[j][k] \leftarrow A[j][k] - m \cdot A[i][k]$
    end for
    $b[j] \leftarrow b[j] - m \cdot b[i]$
  end for
end for
▷ Back substitution
for $i = n$ to 1 step -1 **do**
  $sum \leftarrow 0$
  for $j = i + 1$ to n **do**
    $sum \leftarrow sum + A[i][j] \cdot x[j]$
  end for
  $x[i] \leftarrow (b[i] - sum)/A[i][i]$
end for

above algorithm, the following augmented matrix in upper triangular form was produced.

$$\mathbf{A}|b = \left[\begin{array}{ccc|c} 6.0000e+00 & 2.0000e+00 & 2.0000e+00 & -2. \\ -2.0000e-05 & 1.6667e+00 & -1.3333e+00 & 0.33334 \\ 2.0000e-05 & -3.3330e-11 & -3.3332e-01 & 1.6667 \end{array}\right]$$

In addition to the upper-triangular matrix, the vector $\vec{x}$ was found to be $\begin{bmatrix} 2.6002 & -3.8001 & -5.0003 \end{bmatrix}^T$ which matches more closely to what was found in part (1) earlier compared to the no pivoting method.

1.4 Comparison and Stability Analysis

As shown in the previous two sections, the stability and accuracy of the different methods of Gaussian Elimination vary depending on whether or not partial pivoting was allowed. In short, the non-pivoting method was able to compute an upper-triangular matrix but failed to accurately find the exact solution that was given in part (1). Therefore, this method is highly unstable, and as a result, not always accurate. The partial pivoting method yielded far more accurate results as the solution it provided matched the one given in part (1) to the third decimal place.

2 Question 2

For this question, we are given the following symmetric matrix found in (2).

$$\mathbf{A} = \begin{bmatrix} 12 & 10 & 4 \\ 10 & 8 & -5 \\ 4 & -5 & 3 \end{bmatrix} \tag{2}$$

We are then asked to bring the matrix $\mathbf{A}$ to tri-diagonal form via a similarity transform using a Householder matrix. We know that a similarity transform of the matrix $\mathbf{A}$ is given by

$$\mathbf{A}' = \mathbf{Q}^T\mathbf{A}\mathbf{Q}$$

where $\mathbf{Q}$ is an orthogonal matrix. That is $\mathbf{Q}^T = \mathbf{Q}^{-1}$. For this problem, we are asked to perform the similarity transform using a Householder Matrix so we will let $\mathbf{H}$ denote our Householder Matrix. We know that a $\mathbf{H}$ is given by the following

$$\mathbf{H} = \mathbf{I} - 2ww^T$$

where $\mathbf{I}$ is the identity matrix and w is the normalized vector $v = x + \text{sign}(x_1)||x||_2 e_1$. In this form, x is the vector from the matrix $\mathbf{A}$ containing x_{21} and x_{31}. So we have the following.

$$x = \begin{bmatrix} 10 \\ 4 \end{bmatrix} \text{ and } e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{aligned} v &= \begin{bmatrix} 10 \\ 4 \end{bmatrix} + \sqrt{10^2 + 4^2} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 10 \\ 4 \end{bmatrix} + \sqrt{116} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 10 \\ 4 \end{bmatrix} + \begin{bmatrix} \sqrt{116} \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 10 + \sqrt{116} \\ 4 \end{bmatrix} \\ &= \begin{bmatrix} 20.77032961 \\ 4 \end{bmatrix} \end{aligned}$$

We then normalize v to obtain w as shown below.

$$\begin{aligned} v' &= \frac{1}{||v||_2} v \\ &= \frac{1}{\sqrt{20.77032961^2 + 4^2}} \begin{bmatrix} 20.77032961 \\ 4 \end{bmatrix} \\ &= \frac{1}{21.1519879} \begin{bmatrix} 20.77032961 \\ 4 \end{bmatrix} \\ &= \begin{bmatrix} 0.9819563867 \\ 0.1891075212 \end{bmatrix} \end{aligned}$$

Since we are working with a 3 by 3 matrix, let $w = \begin{bmatrix}0 & 0.9819563867 & 0.1891075212\end{bmatrix}^T$. Plugging into the Householder formula, we get the following.

$$\begin{aligned}
\mathbf{H} &= \mathbf{I} - 2ww^T \\
&= \mathbf{I} - 2\begin{bmatrix}0 & 0.9819563867 & 0.1891075212\end{bmatrix}\begin{bmatrix}0 \\ 0.9819563867 \\ 0.1891075212\end{bmatrix} \\
&= \mathbf{I} - 2\begin{bmatrix}0 & 0 & 0 \\ 0 & 0.96423 & 0.18569 \\ 0 & 0.18569 & 0.03576\end{bmatrix} \\
&= \begin{bmatrix}1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1\end{bmatrix} - \begin{bmatrix}0 & 0 & 0 \\ 0 & 1.92846 & 0.37138 \\ 0 & 0.37138 & 0.07152\end{bmatrix} \\
&= \begin{bmatrix}1 & 0 & 0 \\ 0 & -0.92846 & -0.37138 \\ 0 & -0.37138 & 0.92848\end{bmatrix}
\end{aligned}$$

We can now use $\mathbf{H}$ to evaluate the similarity transformation $\mathbf{A}' = \mathbf{HAH}$ which gives use the following tri-diagonal matrix.

$$\mathbf{A}' = \begin{bmatrix}12 & -10.77032961 & 0 \\ -10.77032961 & 3.86206897 & 5.34482759 \\ 0 & 5.34482759 & 7.13793103\end{bmatrix}$$

It is important to note that while most calculations and computation was done by hand for this problem, a python script was also written which follows the same methodology and uses `Numpy` to calculate both the Householder Matrix, $\mathbf{H}$, as well as the similarity transformation and tri-diagonal matrix $\mathbf{A}'$. This code can be found on the Github repository under `Homework_12`.

3 Question 3

This question revolves around the power method for finding the dominant eigenvalue. In the following sections, we are asked to develop python code using the power method for finding both the dominant eigen value and the smallest eigenvalue. We are then tasked with evaluating the error estimate and providing a case where the power method does not work.

3.1 Implementing the Power Method

For this part, we are asked to implement the power method for finding the dominant eigenvalue λ_1 where $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. The code that was produced can be found in the GitHub Repository under `Homework_12`. Using the produced code, we are then asked to find the dominant eigenvalue and corresponding eigenvector of the Hilbert matrix with entries

$$\mathbf{A}_{i,j} = \frac{1}{i+j-1}$$

for $n = 4, 8, 12, 16, 20$. The results of the power method are shown below

$$\begin{aligned}
n = 4 \quad &\text{Dominant eigenvalue} \approx 1.500214, \quad \text{iterations} = 19 \\
n = 8 \quad &\text{Dominant eigenvalue} \approx 1.695939, \quad \text{iterations} = 27 \\
n = 12 \quad &\text{Dominant eigenvalue} \approx 1.795372, \quad \text{iterations} = 1000 \\
n = 16 \quad &\text{Dominant eigenvalue} \approx 1.860036, \quad \text{iterations} = 31 \\
n = 20 \quad &\text{Dominant eigenvalue} \approx 1.907135, \quad \text{iterations} = 29
\end{aligned}$$

As shown above, the dominant eigenvalue increases as n increases. Additionally, the number of iterations needed to converge also increases as the size of the Hilbert matrix increases. It is important to note that for $n = 12$, the method required the maximum number of iterations allowed, indicating slow convergence due to the tightly clustered eigenvalues for larger Hilbert matrices. Additionally, I should note that my output does not match what I found when researching the dominant eigenvalues of the Hilbert matrix. From what I could find, the dominant eigenvalue should be around 1.75. Despite hours of debugging, I could not figure out what was wrong with my code.

3.2 Alterations for the Smallest Eigenvalue

For this part, we are asked to modify the power method to find the smallest eigenvalue of the Hilbert matrix for $n = 16$. To do this, the **inverse power method** was implemented. Instead of applying the matrix $\mathbf{A}$ at each iteration, the inverse power method solves the system

$$\mathbf{A}\vec{y} = \vec{x}$$

Using the developed code, the following results were obtained:

$$n = 16 \quad \text{Smallest eigenvalue} \approx 7.679967 \times 10^{-19}, \quad \text{iterations} = 1$$

It is important to note that the smallest eigenvalue is extremely small for larger Hilbert matrices. This is consistent with the fact that Hilbert matrices are extremely ill-conditioned, and their smallest eigenvalues are on the order of 10^{-19} for $n = 16$. The warning issued by `Scipy` about the ill-conditioning of $\mathbf{A}$ is expected and reflects the numerical difficulty of solving systems involving the Hilbert matrix. Furthermore, convergence after a single iteration is reasonable in this case, as the inverse operation greatly magnifies the component associated with the smallest eigenvalue, rapidly aligning the iterate with the corresponding eigenvector.