Lab 4 Report

PHY407 Week 3 Assignment

Landon Wang, Yishi Liu

October 7, 2022

Contribution: Yinshi Liu writes the entirety of question 1 and 3, Landon Wang writes the entirety of question 2 and completes this document.

Question 1

(a)

Code Printout

```
Solution using Gaussian elimination = [ 2. -1. -2.  1.]
Solution using Partial Pivot = [ 2. -1. -2.  1.]
Solution using Numpy = [ 2. -1. -2.  1.]
```

The solution using partial pivot is identical to Gaussian elimination, which is also identical to the solution using numpy. This means that both method are working as expected.

(b)

Pseudocode

1. Define function Gaussian_elimination (A, v), use the code outlined in Newman (Example 6.1, Pg. 219)
2. On the basis of Gaussian_elimination, adapt the code for Partial_pivot (A, v):

For each row m of the matrix:

Find the diagonal value A_{mm}(pivot) , and check to see if the value of A_{mm} is small.

If the value is close to zero, find the value $A_{m+i,m}$ for each row beneath m.

If the absolute value of $A_{m+i,m}$ is greater than the value of A_{mm}, then swap the rows of the matrix and elements of the respective vector v. (Swap around the pivot)

Iterate for every row beneath then break the loop. (Switch pivot)

Continue the elimination process.

3. Define function LU_decomposition (A, v)

Set the value of the upper matrix (U) to be A, since this matrix can be obtained through elimination.

Set the value of the lower matrix (L) to be an empty/identity matrix. Then fill in the values of the matrix using the following scheme:

$$L = \begin{pmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & b_{11} & 0 & 0 \\ a_{20} & b_{21} & c_{22} & 0 \\ a_{30} & b_{31} & c_{23} & d_{33} \end{pmatrix} \tag{1.1}$$

(Eqn. 6.32, Newman, Pg. 226)

where b, c, d represents the elements of matrix A after 1, 2, and 3 steps of Gaussian elimination. This pattern continues for larger matrices.

Create an empty array of y-values, which is the result of:

$$\begin{pmatrix} l_{00} & 0 & 0 & 0 \\ l_{10} & l_{11} & 0 & 0 \\ l_{20} & l_{21} & l_{22} & 0 \\ l_{30} & l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{1.2}$$

to obtain the value of y_n, use the following formula:

$$y_n = \frac{(v_n - (l_{n,0} \quad \cdots \quad l_{n,n-1}) \cdot (y_0 \quad \cdots \quad y_{n-1}))}{l_{nn}} \tag{1.3}$$

(Note: in order to properly calculate the y values, use the forward method from n = 0 and increase the indices)

calculate the x value using the formula:

$$x_n = \frac{(y_n - (u_{0,n} \quad \cdots \quad u_{n-1,n}) \cdot (v_0 \quad \cdots \quad v_n)}{u_{nn}} \tag{1.4}$$

(Note: in order to properly calculate the x values, use the backward method from n = size of matrix and decrease the indices)

return x.

4. Create a list of select values ranging from 5 to a few hundred.
5. Create empty arrays for variables
6. For each value in this list, create a matrix and vector of this dimension.
7. Call all 3 functions to solve created matrix. Time the computing time for each method, then append it into the time array for each method.

Calculate the vector v using the following method:

$$v_{sol} = A \cdot x \tag{1.5}$$

calculate the error using:

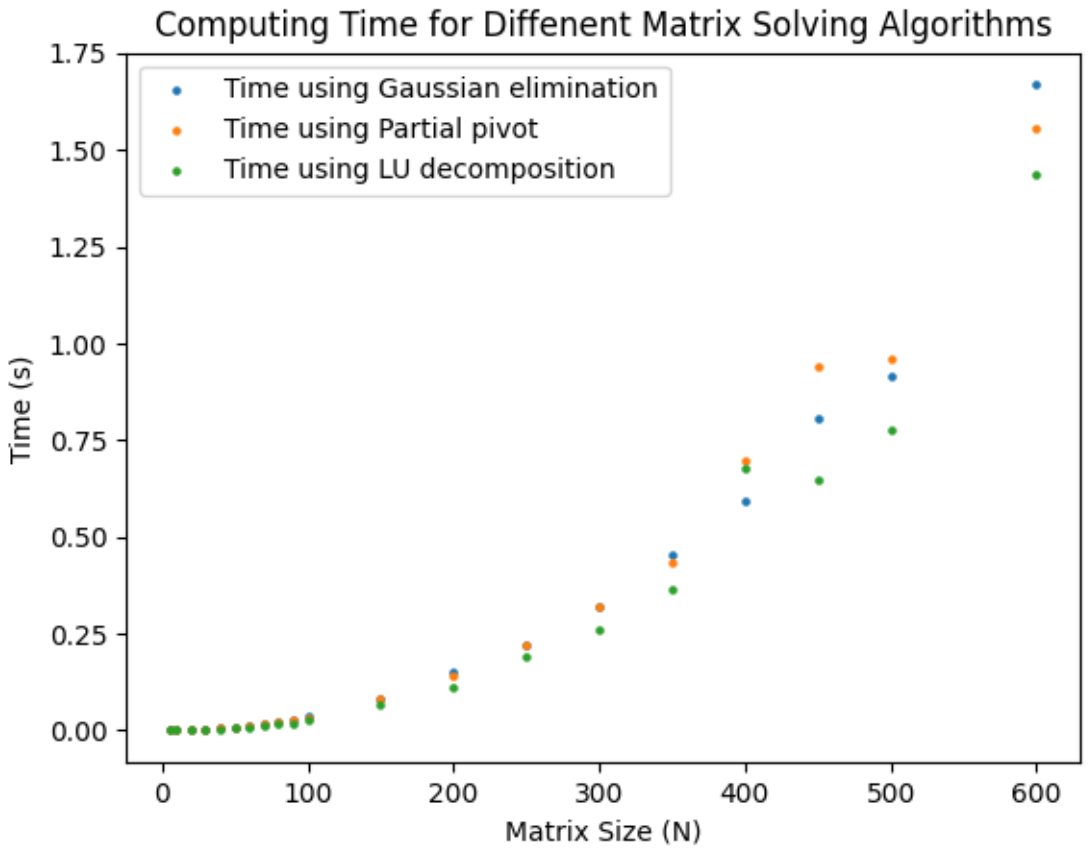
$$error = mean(|vector - v_{sol}|) \tag{1.6}$$

append the error for each method into respective arrays.

8. Plot the results.

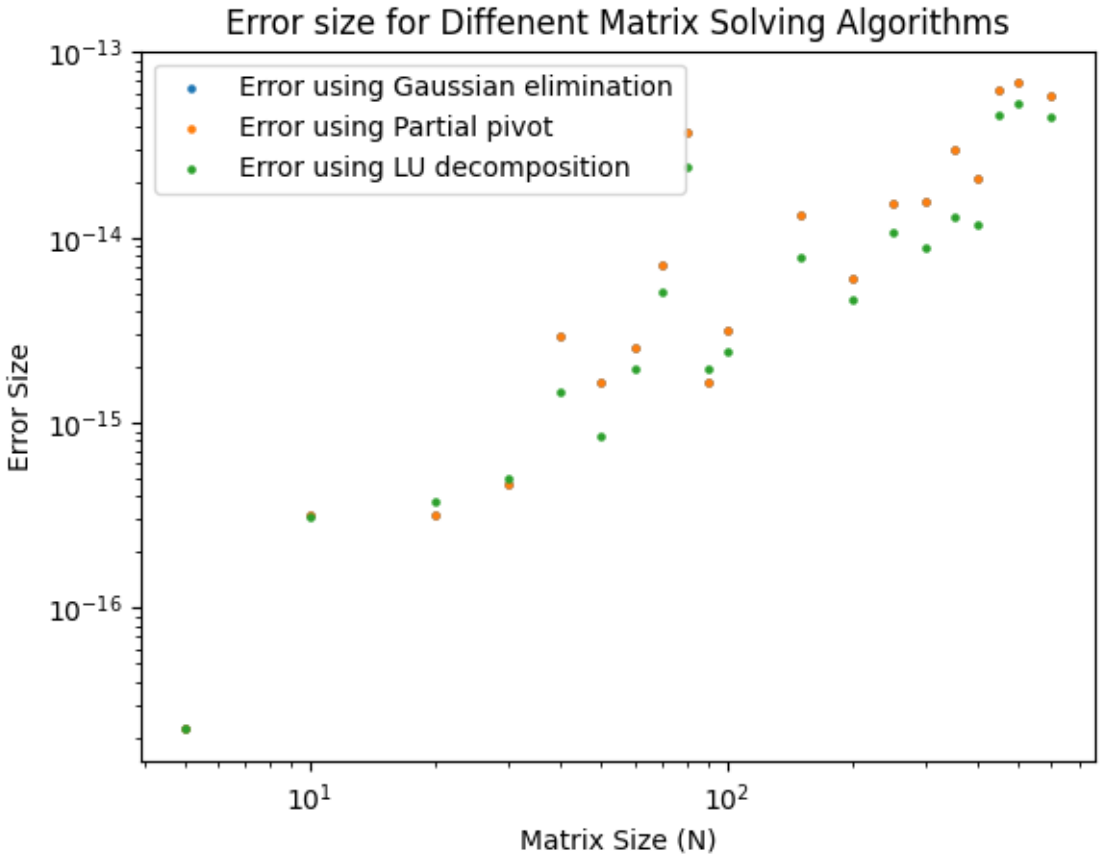
Code Outputs

Figure 1.1: Computing time for different matrix solving algotithms



As the plot has shown, the computing time for all methods follows a non-linear increase as the size of the matrix N increases. This is to be expected as all methods have $O(n^3)$ complexity. Generally, the time taken for Gaussian elimination and partial pivoting are quite similar, and the computing time used by LU decomposition is noticeably less than both of them. Aside from random fluctuations, this behaviour agrees with theory very well, where LU decomposition has a time complexity of $\frac{2}{3}O(n^3)$.

Figure 1.2: Error size for different matrix solving algorithms



According to Figure 1.2, the size of error increases as the size of the matrices increases. Since the general trend follows a straight line on a logarithmic scale, the error size increases exponentially as the size of the matrix increases. It is notable that the error size of Gaussian elimination and partial pivoting is identical, which is expected since all matrices used in this test only contains non-zero integers. As a consequence of that, Gaussian elimination would be mathematically equivalent to partial pivoting. Aside from random errors, the error sizes of LU decomposition slightly smaller than both methods mentioned above, which is expected as LU decomposition is generally superior compared to Gaussian elimination in most cases.

Question 2

A time independent Schrödinger's Equation states that

$$\hat{H}\psi = E\psi \tag{2.1}$$

Where $\hat{H}$ is the Hamiltonian defined as

$$\hat{H} = \frac{\hbar^2}{2M}\frac{d^2}{dx^2} + V(x) \tag{2.2}$$

Equation 2.1 is a shortened version using the Hamiltonian operator.

The most significance of writing the Schrödinger's Equation in the equation 2.1 form is that matrix operation gives a very easy to understand, yet complete solution. Notice the definition of eigenvalue and eigenvectors:

$$A\xi = \lambda\xi \tag{2.3}$$

Where A is a matrix, ξ is a column vector and λ is a number

This means, that for a given matrix A, there exist a vector ξ and value λ such that equation 2.3 stands. (Restrictions apply)

It is obvious that equation 2.1 is in the same form of equation 2.3, which means the Schrödinger's Equation can be solved with eigenvalue and eigenvector calculations.

When we restrict the dimension of ψ to 1, we can easily find the solution to Schrödinger's Equation by the method of serration of variables. Using Fourier sin series, we can express the solution as

$$\psi(x) = \sum_{n}^{\infty} \psi_n \sin\left(\frac{n\pi x}{L}\right) \tag{2.4}$$

Where n is an integer, L is the length of the well.

This implies that Schrödinger's Equation can be expressed as

$$\sum_{n}^{\infty} \psi_n \int_0^L \sin\left(\frac{n\pi x}{L}\right)\hat{H}\sin\left(\frac{m\pi x}{L}\right)dx = \frac{1}{2}LE\psi_m \tag{2.5}$$

We have:

$$H_{mn} = \frac{2}{L}\int_0^L \sin\left(\frac{m\pi x}{L}\right)\hat{H}\sin\left(\frac{n\pi x}{L}\right)dx \tag{2.6}$$

By the orthogonality relations, we (Quark and Qubit) can calculate H_{mn} as

$$H_{mn} = \begin{cases} 0 & m \neq n, both\ even\ or\ odd \\ -\dfrac{8amn}{\pi^2(m^2-n^2)^2} & m \neq n, one\ even\ one\ odd \\ \dfrac{1}{2}a + \dfrac{\pi^2\hbar^2m^2}{2ML} & m = n \end{cases} \quad if \tag{2.7}$$

With the fact that $V(x) = \frac{ax}{L}$

(a)
Thank you, Quark and Qubit!

(b)
Thank you again, Quark and Qubit!

With the help from our lovely guinea piggies, we have a full square, symmetric matrix of H_{mn}, which we can carry on the eigenvalue and eigenvector calculation.

(c)
Let a empty 10×10 matrix that and carry on the calculation in equation 2.7, and than follow a eigenvalue and eigenvector calculation using np.linalg.eigh function because the fact that H_{mn} is a symmetric matrix. The output of the eigenvalues (the energy level) is shown in following table 2.1.

Table 2.1: Energy level outputs for 10x10 H matrix

n	E_n (eV)
1	5.83634
2	11.181
3	18.6627
4	29.1438
5	42.6545
6	59.1844
7	78.7282
8	101.284
9	126.849
10	155.553

(d)
Now, modify the code from (c), we use a 100×100 matrix for H_{mn}, the data outputs follows in table 2.2:

Table 2.2: Energy level outputs for the first 10 states for a 100x100 H matrix

n	E_n (eV)
1	5.83634
2	11.181
3	18.6627
4	29.1438
5	42.6545
6	59.1843
7	78.7281
8	101.283
9	126.849
10	155.423

Compare the data from table 2.1 and 2.2, we observe that small deviation in every energy states with more substantial difference when energy level is high. Notice the number in the table has very limited accuracy. For more accuracy result should be check with the printed outputs.
In theory though, it should be more precise if we have a larger H_{mn}.

(e)
Figure 2.1 and 2.2 shows the probability density function $|\psi(x)|^2$ and the wave function $\psi(x)$ in the first three energy states.
As described earlier, we used the numpy function to calculate the eigenvectors and eigenvalues, which gives the out puts eigenvector as a matrix:

$$V = \left[\overrightarrow{V_1} \quad \cdots \quad \overrightarrow{V_n}\right] \tag{2.8}$$

Which $\overrightarrow{V_n}$ is the eigenvector corresponds to the eigenvalue E_n of the column vector E:

$$\vec{E} = \begin{bmatrix} E_1 \\ \vdots \\ E_n \end{bmatrix} \tag{2.9}$$

Therefore, when we calculate the wave function $\psi(x)$, we use the expression in equation 2.4:

$$\psi_n(x) = \sum_m^\infty \psi_m \sin\left(\frac{m\pi x}{L}\right) = \sum_m^\infty \overrightarrow{V_{n,m}} \sin\left(\frac{m\pi x}{L}\right) \tag{2.10}$$

As required for the problem, the first three states is calculated and plotted, which requires another layer of for loops.

Pseudocode
1. Define the steps of the set of x
2. Create an array of x using step defined in 1
3. Create an empty array for ψ_n
4. Calculate the step size using the length L and step defined in 1

PLOT and Calculation
5. Set up the first loop for three energy states
6. Set up the second state for every element in the eigenvector
7. Calculate ψ_m by equation 2.10
8. Sum up all ψ_m for ψ_n
9. Integrate $\psi_m^2(x)$ using trapz method for the area under the curve
10. Plot the ψ_m^2 plot for the three energy states with normalization
11. Repeat 5-10 for ψ_m

Notice that a function was defined earlier in the supposed (b) part of the response that uses the category indicated in equation 2.7.

Figure 2.1: $|\psi_n(x)|^2$ for n = 1, 2, 3

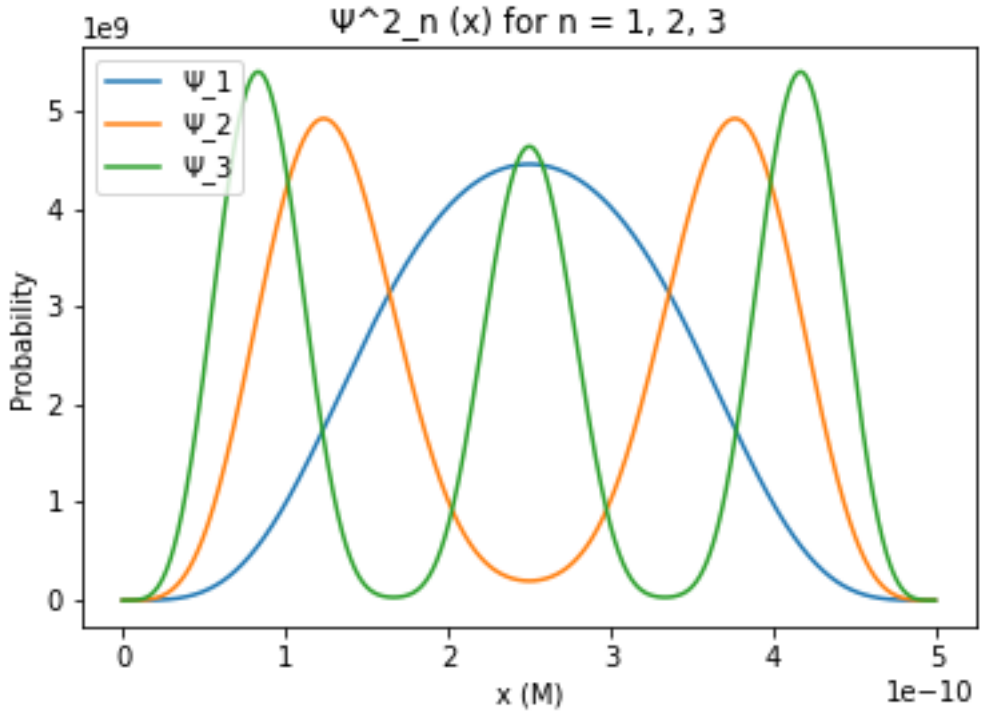
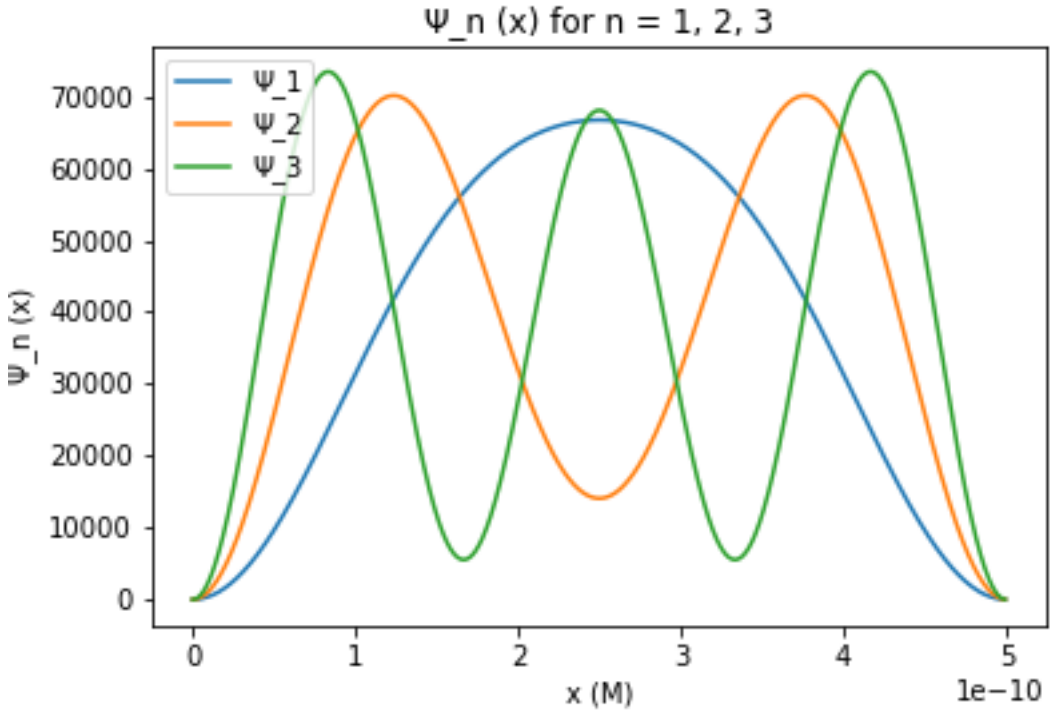


Figure 2.2: $\psi_n(x)$ for n = 1, 2, 3



Question 3

(a)

Textbook 6.10 a

Pseudocode

1. Define function f(x) and relative constants, in this case, f(x) is the following:

$$f(c,x) = 1 - e^{-cx} \tag{3.1}$$

2. Setup variable x and initial error. In this case x = 1.0, error = 1.0
3. While the error is higher than desired accuracy:

Update the x values using eqn. 3.1.

Update error using the error estimates:

$$\epsilon' \simeq \frac{x - x'}{1 - \frac{1}{f'(x)}} \tag{3.2}$$

in this case,

$$f'(x) = ce^{-cx} \tag{3.3}$$

$$\epsilon' \simeq \frac{x - x'}{1 - \frac{e^{cx}}{c}} \tag{3.4}$$

4. Print the final code output.

Code Output

```
Solution of x: 0.7968126311118457
Number of iterations: 14
```

Textbook 6.10 b

Pseudocode

1. Define arrays and relative constants
2. For each value of c:

Define values of x1 and error to 1.0 (m1 in actual code to avoid confusion)

While error is greater than accuracy:

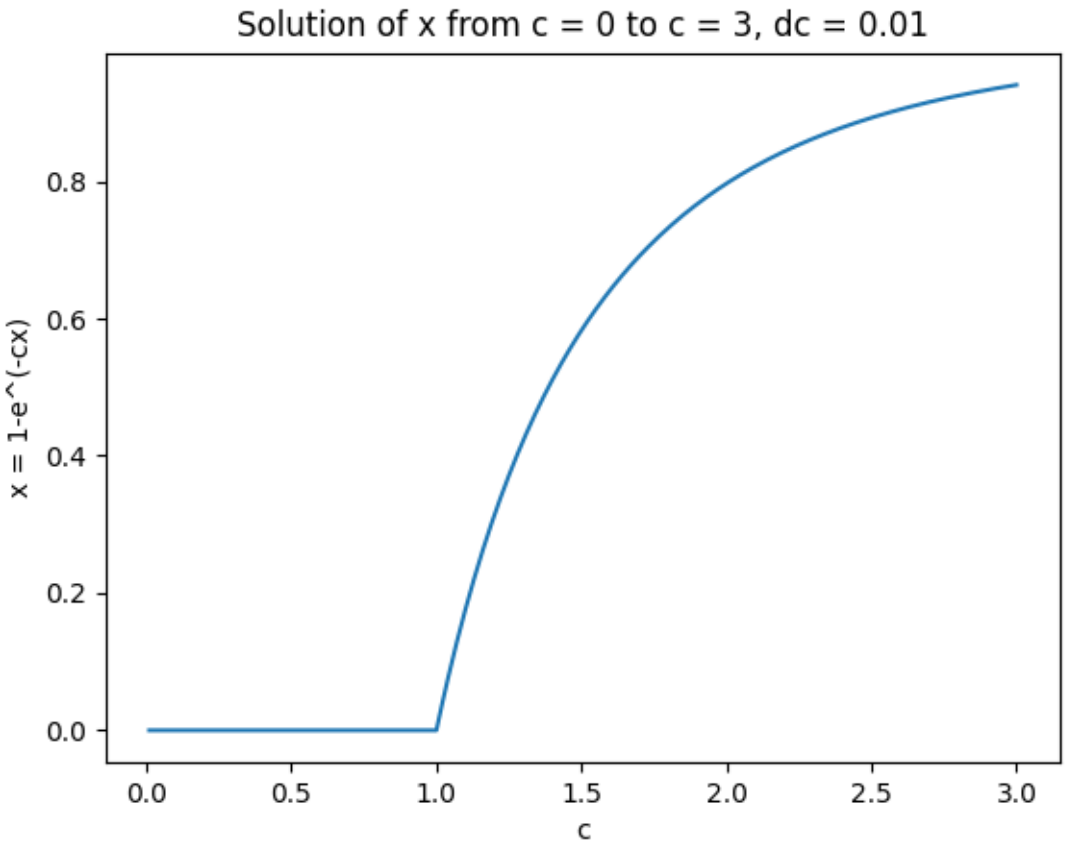
Redefine x1 using eqn. 3.1, update error using eqn. 3.4

Append values of x to prepared array

3. Plot the result.

Code Output

Figure 3.1: Solution of x from c=0 to c=3



As expected from eqn. 3.1, the solution of x is zero when $c < 1$, then it started to grow rapidly as c becomes larger than $c = 1.0$, which is the location of epidemic threshold.

(b)

Textbook 6.11 b

Pseudocode

1. Update the while loop in 6.10 with a variable i to count the number of iterations
2. Update and print the outputs

Code Output

```
Solution of x: 0.7968126311118457
Number of iterations: 14
```

Textbook 6.11 c

Pseudocode

1. Redefine constants and functions
2. Define over relaxation constant ω to be a number between 0 and 1.
3. Adapt the error estimate and function for over relaxation method, where:

$$\Delta x = x' - x = f(x) - x \tag{3.5}$$

$$x' = x + (1 + \omega)\Delta x \tag{3.6}$$

$$\epsilon' \simeq \frac{x - x'}{1 - \frac{1}{[(1 + \omega)f'(x) - \omega]}} \tag{3.7}$$

Code output

```
Solution of x: 0.7968123729832618
Number of iterations: 4
```

After some testing, the optimum over relaxation constant turns out to be $\omega = 0.5$, and the number of iterations is 4. This agrees with the textbook, which states that the calculation would converge at least twice as fast as normal relaxation method.

Textbook 6.11 d

There can be cases when over relaxation constant ω is smaller than zero. In such cases, each step of the calculation consistently undershoots the calculated result, which effectively reduces the step size for each step. This can work in cases where there are multiple zeros close together, but only one of them is a stable fixed point.

(c)

Textbook 6.13 b

Pseudocode

1. Define constants and function g(x):

$$g(x) = 5e^{-x} + x - 5 \tag{3.8}$$

2. Estimate the position of root. (In this case, it is known that the x = 0 root is trivial, so find the other root by plotting the function. According to the plot, there exists another root between 4 and 5)
3. Set up $x_1 = 4$ and $x_2 = 5$, check if $g(x_1)$ and $g(x_2)$ have opposite signs.
4. While $x_2 - x_1$ is larger than desired accuracy:

Calculate midpoint $x' = \frac{1}{2}(x_1 + x_2)$, evaluate sign of $g(x')$.

If the sign is the same as $g(x_1)$, update the value of x_1, else, update the value of x_2.
Update error $\epsilon' = |x_2 - x_1|$, iterate.

5. Update the midpoint $x' = \frac{1}{2}(x_1 + x_2)$ and print the result.

Code Output

```
root using binary search: 4.965114116668701
```

It is given that:

$$b = \frac{hc}{k_b x} \tag{3.9}$$

therefore, the Wien's displacement constant roughly equals to:

```
value of Wein's displacement constant: 0.0029011111301005563
```

which fits the theoretical value of $2.898 \times 10^{-3} mK$.

Using the formula:

$$\lambda = \frac{b}{T} \tag{3.10}$$

get:

$$T = \frac{b}{\lambda} \tag{3.11}$$

to obtain the surface temperature of the Sun:

```
Temperature of the Sun (K): 5779.105836853698
```

Which is very close to the theoretical value of 5778K.