PHY407 Lab-04 Report

Patrick Sandoval[1] and Chin Chong Leong[2,3]

[1] *Q2 - Asymmetric Quantum Well*
[2] *Q1 - Solving Linear System*
[3] *Q3 - Solving Non-Linear Systems*

1. Q1 - SOLVING LINEAR SYSTEM

For the pseudo-code and comments of the implementation of `PartialPivot()` function for this question, please refer to the python script title `Lab04_Q1.py` and `SolveLinear.py`.

1.1. *Q1a - Implementation of Partial Pivoting function*

For this part of the exercise, we implemented the `PartialPivot()` function such that at each stage (at mth row), it will check the mth element of all lower rows and find which row has value furtherest away from zero (using `numpy.argmax()`); if that row is not the current row then it will perform row swapping, and proceed with the Gaussian Elimination process. We defined the matrix $\mathbf{A}$ and vector $\mathbf{v}$ as in Eq 6.2-6.3 in the textbook, where

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

$$\mathbf{v} = \begin{bmatrix} -4 \\ 3 \\ 9 \\ 7 \end{bmatrix} \tag{2}$$

and by calling `PartialPivot(A,v)`, we obtained the vector:

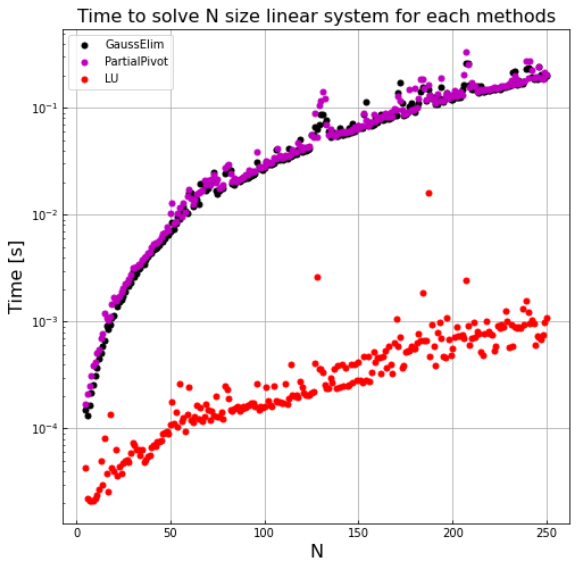
$$\mathbf{x} = \begin{bmatrix} 2 \\ -1 \\ -2 \\ 1 \end{bmatrix} \tag{3}$$

which matches up exactly with the values in Eq 6.16 in the textbook.

1.2. *Q1b - Comparison of accuracy and timing of 3 methods to solve linear system*

We implemented a for-loop to run from $N = 5$ to $N = 250$, where in each loop, a $N \times N$ matrix $\mathbf{A}$ and N-size vector $\mathbf{v}$ is randomly created, and the Gaussian Elimination, Partial Pivoting, and LU decomposition were used to solve the x vector satisfying $\mathbf{Ax} = \mathbf{v}$. The timing and error of the each method for each N are recorded and plotted in the Figure 1

As we can see from Figure 1, the LU decomposition method (`numpy.linalg.solve` in python) perform much better than the other two methods in both time and accuracy, where it takes about 10^{-4} to 10^{-3} seconds to solve the linear equations and the error is around 10^{-14} at large N. The Gaussian Elimination and Partial Pivoting method both perform similarly time-wise, with Partial Pivoting being slightly slower than Gaussian Elimination on average (which makes sense since it has to do one more step on row-swapping if necessary). However, the Partial Pivoting method is on average slightly more accurate than Gaussian Elimination, the error of the Partial Pivoting method is approximately 10^{-13} while the error of the Gaussian Elimination method is approximately 10^{-12} to 10^{-11}. (This also makes sense since Partial Pivoting would try to avoid errors from divide-by very small number.)



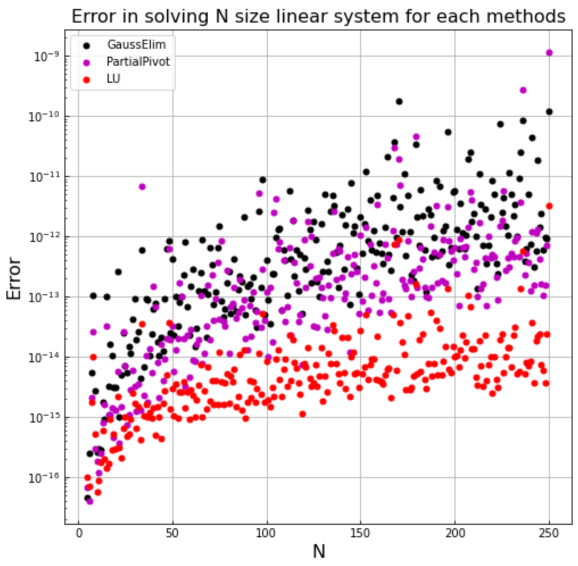


Figure 1: (Left) Time required to compute $\mathbf{x}$ for each N (Right) Error at each N. The black dots are results from using the Gaussian Elimination method; purple dots are from using the Partial Pivoting method; and red dots are from using the LU decomposition method.

2. Q2 - ASYMMETRIC QUANTUM WELL

For the code, pseudo-code, print outs and plots for this question please look at the python script titled `Lab04_Q2.py`.

2.1. *Q2c - Calculating* 10×10 *Hamiltonian Matrix*

For an electron of mass M in a 1-dimensional asymmetric well of width L given by the following relation

$$V(x) = \frac{ax}{L} \tag{4}$$

The matrix element of the infinite dimensional Hamiltonian matrix has the following piece-wise definition

$$\mathcal{H}_{mn} = \begin{cases} 0 & \text{if } m \neq n \text{ and both even/odd} \\ -\frac{8amn}{\pi^2(m^2-n^2)^2} & \text{if } m \neq n \text{ and one even, one odd} \\ \frac{1}{2}a + \frac{\pi^2\hbar^2 m^2}{2ML^2} & \text{if } m = n \end{cases} \tag{5}$$

From eq.5 we see that the Hamiltonian is symmetric and Hermitian because we can interchange m with n and the matrix is left unchanged. Using the matrix element interpretation of the Hamiltonian we can express it as a 10×10 matrix. Now that we have an actual matrix for the Hamiltonian we can compute its eigenvalues which represent the allowed energy states for the electron in the asymmetric potential well.

Table 1: Energy eigenvalues for (10×10) Hamiltonian calculated using `numpy.linalg.eigenvalsh`

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)
5.836	11.181	18.664	29.146	42.658	59.189	78.735	101.293	126.861	155.567

Note—For the full printed values of the energy eigenvalues please refer to the script titled `Lab04_Q2.py`

2.2. *Q2d - Calculating (*100×100*) Hamiltonian Matrix*

Following the same procedure as in subsection 2.1 we can compute a (100×100) Hamiltonian where we can compute its corresponding energy eigenvalues. We can then compare the precision of these eigenvalues with the results shown in table 1.

Table 2: Energy eigenvalues for (100×100) Hamiltonian calculated using `numpy.linalg.eigenvalsh`

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)	(eV)
5.836	11.181	18.664	29.146	42.658	59.189	78.735	101.292	126.860	155.437

Note—For the full printed values of the energy eigenvalues please refer to the script titled `Lab04_Q2.py`

If we compare the results of table 1 with the ones shown in table 2 we see that the lower energy eigenvalues have are almost identical, whereas the higher energy eigenvalues show some discrepancies.

2.3. *Q2e - Probability Amplitudes in an Asymmetric Well*

From the results of subsections 2.1 and 2.2 we see that for low energy levels the eigenvalues of the different size Hamiltonian's are very close to one another, which means that we can uses the smaller Hamiltonian eigenvalues as a good approximations for the lower energy levels of the electron in the potential. Consequently, we will be plotting the probability amplitudes of the ground, 1^{st}, and 2^{nd} excited states. In order to plot such quantities we need to find the wave function at each state, by noting that the eigenvectors form a complete basis and the wave function is a superposition of these basis.

$$\psi(x)_m = \sum_n \phi_{mn}(x) \sin\left(\frac{n\pi x}{L}\right) \tag{6}$$

Where $\phi_{mn}(x)$ is the n^{th} element of the m^{th} energy eigenvector for a wave function in the m^{th} energy state. Once we we calculate the wave function we need to ensure the wave function is normalized, i.e

$$\int_{-\infty}^{\infty} |\psi(x)|^2 \, dx = 1 \tag{7}$$

We can normalize the wave function by performing the integral shown in eq.7 and dividing the wave function by the square root of the result of our integration. Lastly if we want to plot the probability amplitude of the wave function we just need to take the absolute value square of our function and plot that over the spatial domain which is the width of the well. The results of plotting the probability amplitudes is the following. From eq.4 we know that $V \propto x$ which

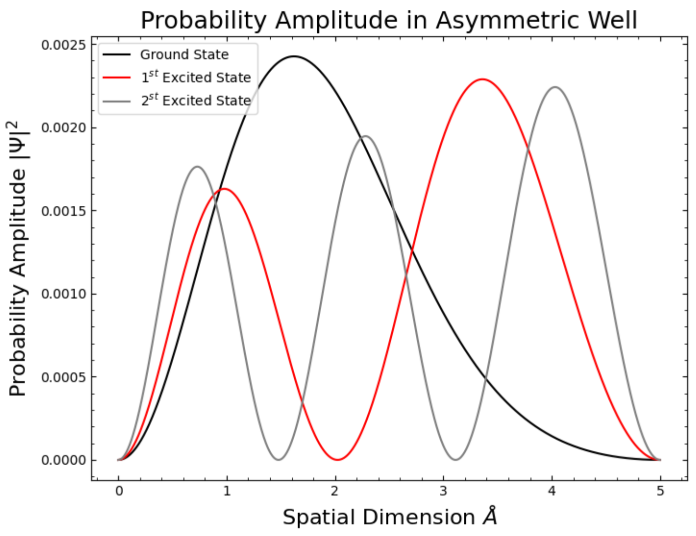


Figure 2: Probability of wave function for ground state, 1^{st} and 2^{nd} excited states

means that its makes sense for the peak of the probability amplitude for the ground states to lean towards the left as this region was lower potentials and is more "accessible" for the electron to be in. And for higher energy states the electron is able to access regions of higher potential with greater ease hence the increasing of peak magnitudes.

3. Q3 - SOLVING NON-LINEAR SYSTEMS

For the pseudo-code and print-outs of each sub-question please refer to the python script titled `Lab04_Q3.py`

3.1. *Q3a - Relaxation method*

For this part of the exercise, we wrote the function $f(x) = 1 - e^{-cx}$ and derive its derivative $f'(x) = ce^{-cx}$. Then we implemented the relaxation method similar to example 6.3 in the textbook as a function, where the error at each iteration would be Eq. 6.83 in the textbook:

$$\epsilon' \approx \frac{x - x'}{[1 - f'(x)]^{-1}} \tag{8}$$

By first setting $c = 2$ and having the first guess of x as 0.5, we obtained

$$x = 0.796811 \tag{9}$$

with the target accuracy as 10^{-6} (Error of this solution is 7.52×10^{-7}).

Then we initialized an array of c ranging $0.01 \leq c \leq 3$ with step size of 0.01. Again by having the first guess of x as 0.5 and using our relaxation method implementation, we calculated the numerical solution at each c and plotted x as a function c in the graph below.

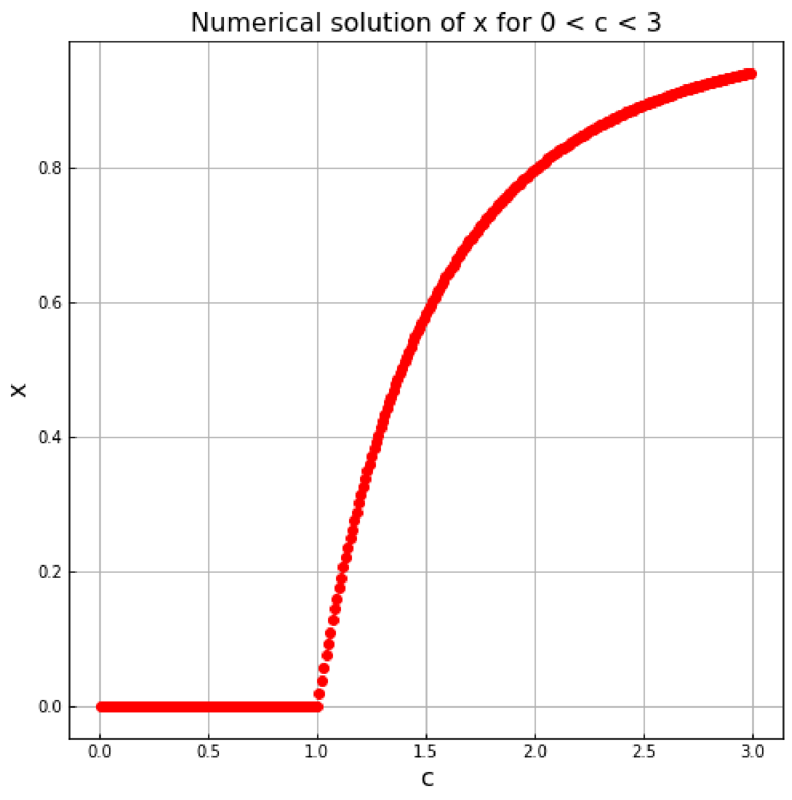


Figure 3: Numerical solution of x as a function of c. It is very clear from this graph there is a transition where x goes from $x = 0$ for $c \in [0, 1]$ to $x \neq 0$ for $c > 1$

3.2. *Q3b - Over-relaxation*

We modified our relaxation method implementation such that it will also return how many iterations it takes for the error of the numerical solution to be less than the target accuracy. Using the same setup as the first part of Q3a (with $c = 2$), we concluded that it takes 15 steps for the solution to converge and accurate to 10^{-6}.

We then implemented the over-relaxation method based on the relaxation function, where in each iteration we modified the next guess x' and the error ϵ' with a parameter ω according to part (a) of the exercise:

$$x' = (1+\omega)f(x) - \omega x \tag{10}$$

$$\epsilon' = \frac{x - x'}{1 - [(1+\omega)f'(x) - \omega]^{-1}} \tag{11}$$

Starting from $\omega = 0.5$, and with a couple trial and error by manually changing the value of ω in python, we eventually got x to converge to the same value ($x = 0.796812$) with the same target accuracy with only 4 steps with $\omega = 0.68$. So it converges almost 4 times as fast as the simple relaxation method.

Notice in the textbook it was stated that overrelaxation method deliberately overshoot the calculated value (parameterized by (1+ω) factor) in each iteration in hope that this will get us to the solution quicker. This works under the assumption that difference between each guess $\Delta x = f(x) - x$ is positive so that Δx can be "boosted" by the (1+ω) factor with $\omega > 0$. Indeed if we look at $f(x) = 1 - e^{-cx}$ we notice that $f(x)$ is a monotonic increasing function and therefore $\omega > 1$ would work in this case. With the same logic, we can see that $\omega < 0$ would help monotonic decreasing function (or at least monotonic decreasing in the neighbourhood containing the solution) to achieve the solution quicker.

3.3. *Q3c - Wien's displacement constant*

We now set $f(x) = 5e^{-x} + x - 5$ (such that $f(x) = 0$ is true for the solution), and implemented the binary search method using recursion with the procedure described in pg.264 in the textbook (see pseudocode and comments of the function for more detail). With the target accuracy set to 10^{-6}, and the initial guess lower and upper bound set to $(x_1, x_2) = (3.0, 5.0)$, we found the solution as:

$$x = 4.9651141 \tag{12}$$

in 21 steps with the binary search method. ($x = 4.9651142$ in 4 steps with Newton's method, $x = 4.9651142$ in 6 steps with simple relaxation method using initial guess $x_1 = 3.0$).

With the nonlinear equation solved, the Wien's displacement constant can be found by:

$$b = \frac{hc}{k_B x} = 0.0028978\, m\, K \tag{13}$$

where $h = 6.626 \times 10^{-34}\ m^2\ kg\ s$ is the Planck constant, $k_B = 1.38 \times 10^{-23}\ m^2\ kg\ s^{-2}\ K^{-1}$ is the Boltzmann constant, and $c = 2.998 \times 10^8\ m/s$ is the speed of light. Finally with Wien's displacement law, the surface temperature of the Sun can be found by:

$$T = \frac{b}{\lambda} = 5772.421\, K \tag{14}$$

where $\lambda = 502$ nm is the peak wavelength of the Sun's spectra.