Computational Physics Assignment

Sulaiman Rasool

I. Question 1

This question tackles floating point variables and asks to write a program to find the nearest representable real number above and below a given floating point variable. We are to return the fractional rounding range of the floating point variable, which is the sum of half the upper and lower difference divided by the floating point variable. The floating point variable for this question is 0.25, the upper closest value is 0.25000000000000006 and the lower closest value is 0.24999999999999997. Through halving the difference and dividing by 0.25 the fractional rounding range is $1.6653345369377348 \times 10{-}16$. The upper and lower values closest to 0.25000000000000006 are ; 0.2500000000000001 and 0.25 with a fractional rounding range $2.2204460492503126 \times 10{-}16$. The upper and lower values closest to 0.24999999999999997 are ; 0.25 and 0.24999999999999994 with a fractional rounding range of $1.1102230246251568 \times 10{-}16$. Floating-point numbers are represented as binary fractions, hence values such as 0.25 can be represented exactly. However most decimal numbers cannot be represented exactly and are approximated by binary floating-point numbers closest to what are store in the machine.

II. Question 2

This question tackles matrix methods, in specific we use LU decomposition with the Crout's algorithm to solve linear algebraic equations. Answering question 2b, I can express matrix A, provided in the question, as a product of upper and lower diagonal matrices. The upper matrix :

$$U = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 \\ 0 & 8 & 4 & 0 & 0 \\ 0 & 0 & 16 & 10 & 0 \\ 0 & 0 & 0 & 44.75 & -25 \\ 0 & 0 & 0 & 0 & 41.45 \end{bmatrix}$$

The lower matrix :

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1.38 & 1 & 0 \\ 0 & 0 & 0 & -0.78 & 1 \end{bmatrix}$$

The determinant of matrix A is 712224. I then made a program to solve the linear equation $A \cdot x = b$ and obtain the matrix x. Using the provided b and A in conjunction with the LU decomposition program, I get x as :

$$x = \begin{bmatrix} 0.46168059 \\ 0.61495822 \\ -0.47991643 \\ 0.06786629 \\ 0.18648066 \end{bmatrix}$$

I check the validity of x by doing the dot product of A with x, which results in the matrix b. Part e requires the use of the previous routines to calculate the inverse of matrix A.

$$A^{-1} = \begin{bmatrix} 0.3807 & -0.0473 & 0.0057 & -0.0034 & -0.0014 \\ -0.142 & 0.142 & -0.0171 & 0.0102 & 0.0042 \\ 0.0342 & -0.0342 & 0.0342 & -0.0206 & -0.0084 \\ 0.0452 & -0.0452 & 0.0452 & 0.0328 & 0.0134 \\ 0.0259 & -0.0259 & 0.0259 & 0.0188 & 0.0241 \end{bmatrix}$$

To confirm that A^{-1} is indeed the inverse of the matrix, I did the dot product of A^{-1} with A, resulting in an identity matrix of the size of A.

III. Question 3

In this question we plot a table of data using either the Lagrange polynomial interpolation method or the cubic spline interpolation method. I use the matrix solver I designed for question 2 to solve the matrix in the cubic spline. The following graphs display my answers for question 3.

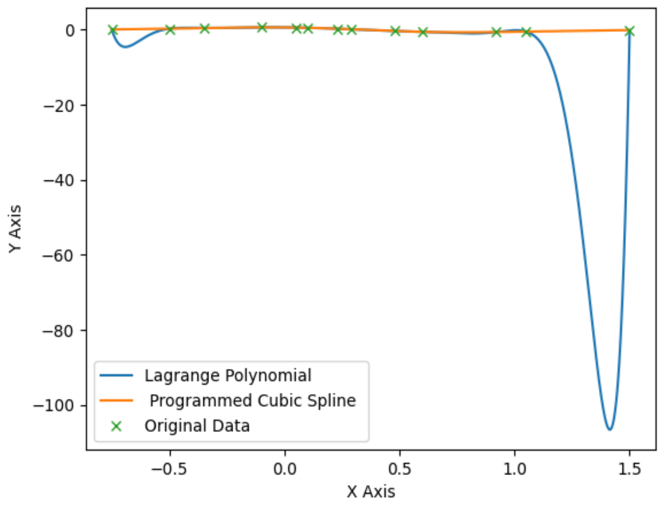

Fig. 1. shows the plot of the tabulated data and the programmed interpolation plots. The green points show the original data, the yellow line shows the Cubic spline and the green line shows the Lagrange polynomial.

Fig1 demonstrates that the cubic spline is a much better interpolation than the Lagrange and goes through the points

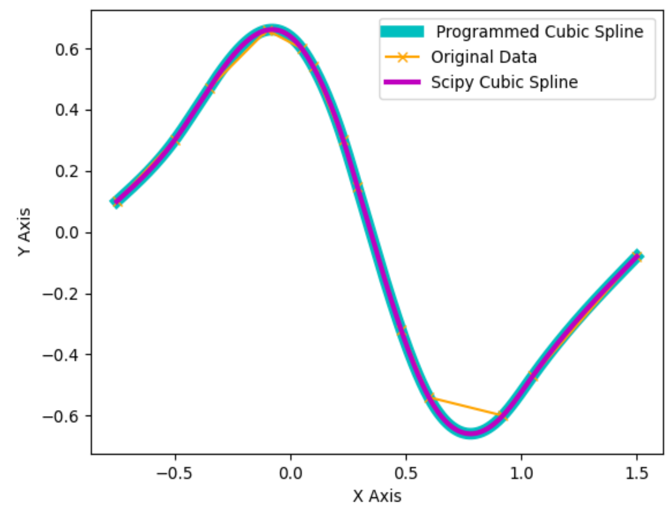

Fig. 2. shows the plot of my programmed cubic spline against an external cubic function, referenced against the original data. The blue line shows my programmed cubic spline, the purple line shows the cubic spline from scipy and the orange line shows the original data with a simple straight line function

as expected. I then use an external function in scipy to compare to my cubic spline, which is shown in figure 2.

Figure 2 shows that the cubic spline function that I programmed interpolated the data to a reasonable accuracy and is verified by an external cubic spline function.

IV. Question 4

This question required to write a program that uses numpy.fft to convolve a signal function with a response function. In this question, the signal function is h(t) as provided, which is a square wave for t in between 5 and 7 seconds and 0 for all other time. The response function is a Gaussian function plotted over the whole time array. For this question I take the range of t between -20 and 20 seconds which allows us to plot the Gaussian correctly. Figure 3 shows my initial plots of the signal function (h) and the response function (g),

To obtain the convolution of the functions, I first padded out the signal and response , adding zeros to the arrays. The took the individual Fourier transforms and multiplied them together, and took the inverse Fourier transform of this. To plot the convolution correctly I had to shift the convolution in the real domain, the following figure shows my results,

Figure 4 shows the plot of the convolution, in order to get the correct amplitude, I normalised the Fourier transforms of the individual equations before multiplying them, which resulted in the correct convolution that has been shifted in the real domain.

V. Question 5

This question tackles the ordinary differential equation for a simple impedance-divider circuit, and aims to find the output voltage as a function of time using two different methods, the Runge Kutta Fourth order and the Adam Bashforth Fourth order method. Answering part a, the ODE can be expressed as ;

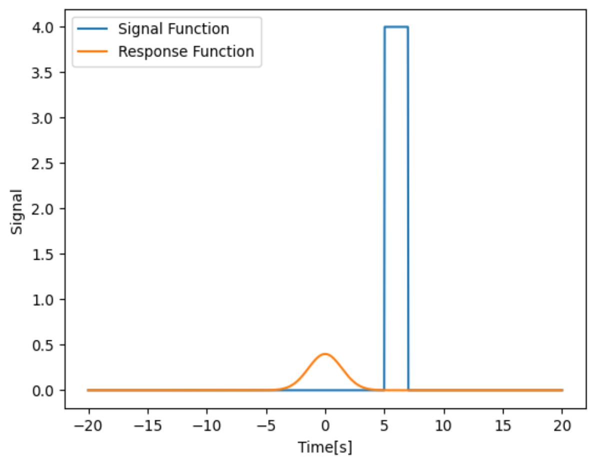

Fig. 3. shows the plots of my signal function in the blue line , h(t), and the response function in the orange line, g(t).

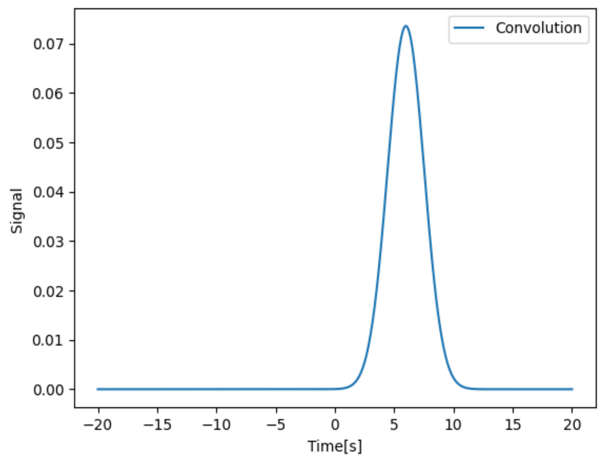

Fig. 4. shows the plot of convolution of the signal and the response function

$$\frac{dV_{out}}{dt'} = V_{in} - V_{out} \tag{1}$$

where t' is a scaled variable equivalent to $\frac{t}{RC}$, where t is the time, R is Resistance of the circuit and C is the capacitance of the circuit. In equation 1, v_{in} is the input voltage which is a function of time, and v_{out} is the output voltage which is a function of time. The question requires to use this ODE to write a program that finds the output voltage at a number time steps given the input voltage function, which has a switch so that we can decide which method to use. Answering part c, I set the initial input voltage to 5 volts, and for all other time I set it to 0. I then solve the ODE to obtain the analytical solution which will validate my methods using the following equation;

$$V_{out} = V_{in} \exp^{\frac{-t}{RC}} \tag{2}$$

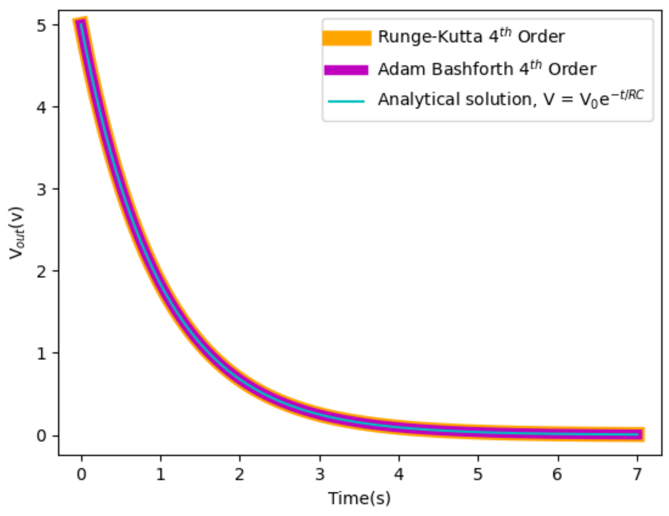

Fig. 5. shows the plot of the Runge Kutta fourth order method in orange, the Adam Bashforth Fourth order method in purple, and the analytical solution in cyan. The initial input voltage is 5v with the resistance $r = 1000\Omega$ and the capacitance c = $1 \times 10^{-3}F$

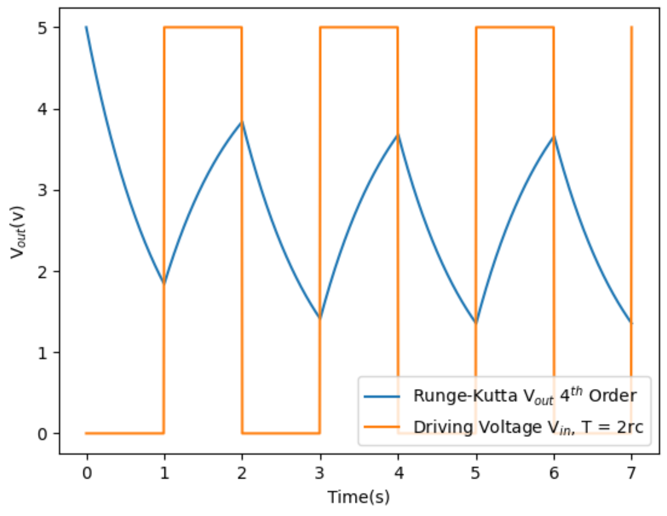

Fig. 7. shows the voltage out, evaluated using the Runge Kutta Fourth order method with a driving voltage $V_5 = 5$ and a square wave function with time period $T = rc$ where the resistance $r = 1000\Omega$ and the capacitance c = $1 \times 10^{-3}F$

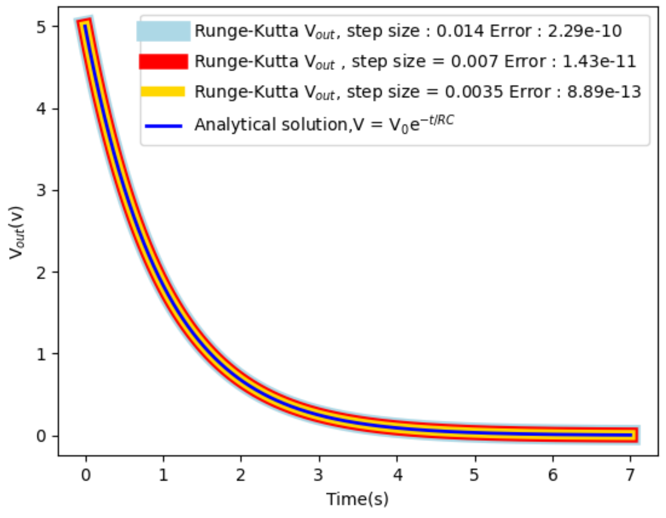

Fig. 6. shows the plot of the Runge Kutta method, plotted with different step sizes, each subsequent step is half the previous step. The initial input voltage is 5v with the resistance $r = 1000\Omega$ and the capacitance c = $1 \times 10^{-3}F$

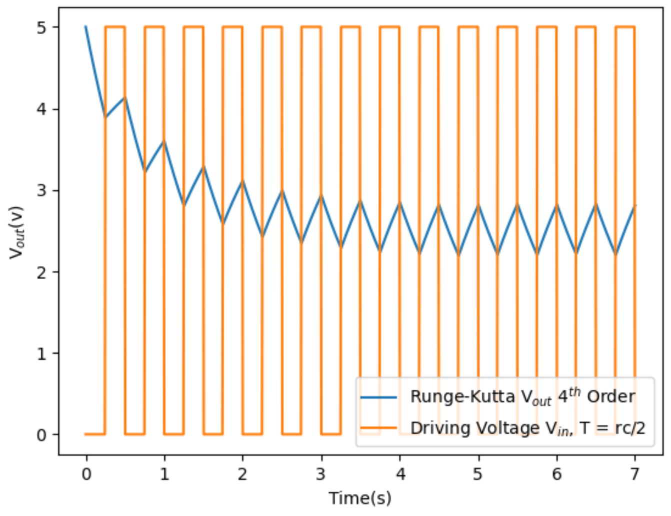

Fig. 8. shows the voltage out, evaluated using the Runge Kutta Fourth order method with a driving voltage $V_5 = 5$ and a square wave function with time period $T = \frac{rc}{2}$ where the resistance $r = 1000\Omega$ and the capacitance c = $1 \times 10^{-3}F$

Using this , I plot the two methods against the analytical solution shown in figure 5. As shown in figure 5, the two methods follow the analytical solution, with a sample rate of 1000 in the time period of 0 to 7 seconds. The Adam Bashforth method utilises the Euler step method to obtain its first four values. The time period is 7 seconds, which is sufficiently large for the voltage out to approximate close to 0. The sample rate is 1000, which allows for an accurate plot of the Runge Kutta and Adam Bashfourth. To answer part d of question 5, I rerun my Runge Kutta method using step sizes of 0.014, 0,007, 0,0035.

Figure 6 shows that when we half the step size while keeping the time period the same, the average error, which is the average difference between the analytical solution and the Runge Kutta curve, decreases by a factor of 16. This is as expected for the Runge Kutta fourth order method, as the total error scales by a factor of 0^4 (order of 4), so if we half the step size, the error should decrease by a factor of 16, as shown. Answering part e , figure 7 and 8 show how the voltage out changes when a square wave with a specified period is used to drive the circuit.

Figure 7 and 8 show how the output changes with a driving voltage with different time periods. Figure 7 has a larger time period, which results in fewer square waves pulses within the given time period to drive the circuit, this means that it takes longer to reach a steady state and also results in a lower output voltage once the steady state is reached, whereas figure 8 has more square waves, which allows the circuit to reach a steady state by with a higher steady state voltage than figure 7.