Physics 607: Project 1

Aleyna Akyüz

I. Coupled Harmonic Oscillator

A. Theory

System that is studied in this project is coupled harmonic oscillator consists of 3 masses and 2 strings. System is in equilibrium in the beginning. Mass's displacements from their equilibrium locations are given by x_1, x_2, x_3. We can calculate the forces on the masses as following

$$F_1 = k(x_2 - x_1)$$

$$F_2 = k(x_3 - x_2) + k(x_1 - x_2)$$

$$F_3 = k(x_2 - x_3)$$

To keep the code simple $k = m = 1$ .Equations of motion of this system are

$$F_1 = x_1''$$

$$F_2 = x_2''$$

$$F_3 = x_3''$$

This is a second order differential equation system. One can reduce the order by introducing a new variable as

$$u = x'$$

Then,

$$u' = x''$$

B. Methods

One can implement numerical methods by introducing a vector as

$$v = [x_1, x_2, x_3, u_1, u_2, u_3]$$

From the equations in the theory section, we can see that

$$v' = [v_3, v_4, v_5, v_1 - v_0, v_2 + v_0 - 2v_1, v_1 - v_2]$$

Indexes of v run from 0 to 5.

To solve this ODE, Runge-Kuta and Euler methods applied and compared with scipy solution.

C. Results

One can observe from the Fig 1 and Fig 2 that position solutions are sinusoidal waves in different amplitudes as expected from physical system.

On the other hand, velocity solutions seems lot more random in Figure 3 and 4. To see the normal mode behaviour we need to solve the eigenvalue problem which is not the scope of this project.

Also, one can clearly see that Runge Kuta and scipy solutions quite close to each other.

Moreover, Runge Kuta method takes so much more time to run (1.41 seconds). This is because scipy is a more optimized code. To improve my method, I can use only numpy arrays rather than lists.

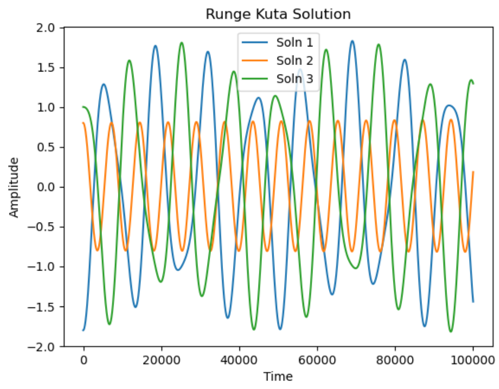

Fig. 1. Solutions from Runge Kuta

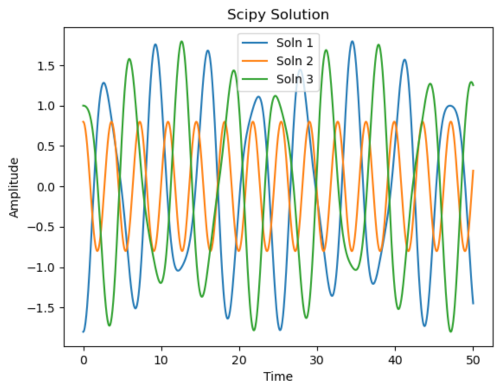

Fig. 2. Solutions from Scipy

II. Electric Field of Uniformly Charged Rod

A. Theory

Consider a rod of length L that has a uniform charge density λ. Electric field distance z above the midpoint of the rod can

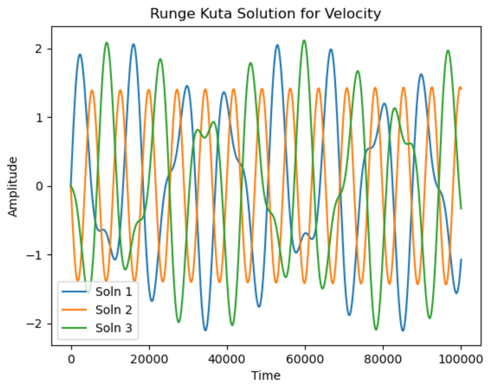

Fig. 3. Solutions from Runge Kuta

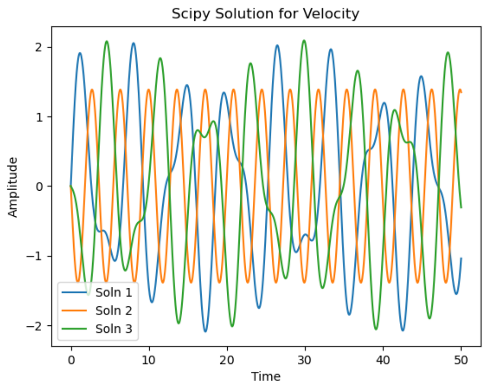

Fig. 4. Solutions from Scipy

be calculated as

$$\frac{1}{4\pi\epsilon_0}\int_0^{L/2}\frac{2\lambda z}{(z^2+x^2)^{3/2}}dx$$

To keep the code simpler, $\frac{\lambda}{2\pi\epsilon_0}=1$ and $z=1$. Also $L=10$

B. Methods

To solve this integral numerically, three methods are used.

The first one is the Riemann sum. Consider any real function f that is defined in a closed interval. Then Riemann sum can be defined as

$$\Sigma_{i=0}^{n} f(x_i)\Delta x_i$$

where $\Delta x_i = x_{i+1} - x_i$. The point that function is evaluated, x_i, can be chosen in different ways. In this implementation I used it as three different ways which are left, right and midpoint.

The second one is the Trapezoidal method. This method can be defined as

$$\frac{\Delta x}{2}(f(x_0)+2f(x_1)+2f(x_2)+2f(x_3)+...f(x_N))$$

. This method is implemented with 'Trapezoidal' function in integralmethods.py and compared with Scipy function scipy.integrate.trapezoid.

The third one is the Simpson method. This method can be defined as

$$\frac{\Delta x}{3}(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+...f(x_N))$$

. This method is implemented with 'Simpson' function in integralmethods.py and compared with Scipy function scipy.integrate.simpson.

C. Results

TABLE I
RESULTS OF NUMERICAL INTEGRALS WHEN Z=1

Method	Integral Value
Analytical Solution	0.9805806756909202
Left Riemann Sum	0.9830618093042169
Right Riemann Sum	0.97810995239455896
Mid-Point Riemann Sum	0.9805806802239275
Trapezoidal Method	0.9805806666249034
Scipy Implemented Trapezoidal	0.9805806666249033
Simpson Method	0.9805806756909093
Scipy Implemented Simpson	0.9805806756909093

From the table we can see that left and right Riemann sum methods are not as good as the other methods.

To observe a limiting condition one can take z as a very large number. Physically, it expected that electric field will go to zero as z increases. Results from table2 is obtained by taking $z = 10^{10}$.

TABLE II
RESULTS OF NUMERICAL INTEGRALS WHEN $z = 10^{10}$

Method	Integral Value
Analytical Solution	4.9999999937499996e-05
Left Riemann Sum	1.5811388298868436e-05
Right Riemann Sum	1.5811388298862507e-05
Mid-Point Riemann Sum	1.5811388298865475e-05
Trapezoidal Method	1.581138829886547e-05
Scipy Implemented Trapezoidal	1.5811388298865475e-05
Simpson Method	1.581138829886547e-05
Scipy Implemented Simpson	1.5811388298865475e-05

Left and right Riemann sums methods' timing quite close to each other since they are similar methods. On the other hand, midpoint methods takes much longer.

In Trapezoid method, my implementation and scipy's method are quite close to each other but in the Simpson method Scipy is way faster.