***Numerical Methods Project Report***

For this project, we used 3 physics equations to simulate physics questions. These parts are independent from each other. In part one, we tried to simulate a radioactive decay and plotting it in python. In part two, we tried to simulate two different pendulums motion tautochrone and parabola. In part three, we tried to simulate the velocity of gas particles in a box and collision of these particles. The working papers used while creating this projects are posted at Work Book.

Part One:

We tried to simulate radioactive decay of a sample of atoms given half life. The half-life of the atoms is set to 10, and the decay constant (lambda) is calculated as the natural logarithm of 2 divided by the half-life. The decay probability (pD) and survival probability (pS) are also calculated. The initial number of atoms is set to 1000.

The function "decay" is defined to simulate the decay of the atoms over time. It takes the initial number of atoms as an input and returns a list of the number of atoms that have survived at each time step. The function uses a for loop to iterate over the range of time steps, and for each time step, it uses the numpy function "random.random" to generate random numbers between 0 and 1 for each atom. It then checks whether the random number is less than the survival probability, and if so, the atom is considered to have survived. The number of surviving atoms is then appended to the population list, and the number of atoms is updated for the next time step.

The function is then called, and the resulting population is plotted over time. The mean deviation and standard deviation of the decay are also calculated and plotted. The exact decay of the atoms is also calculated and plotted for comparison.

Shape

Description automatically generated

Chart

Description automatically generated with medium confidence

Shape

Description automatically generated

Part Two:

We tried to simulate the motion of a simple pendulum using the Lagrangian formalism. We first define the position of the pendulum as a function of time using the symbol 'theta' and calculate the kinetic and potential energy of the pendulum. Then, we use the Lagrange equations to derive the equation of motion for the angular velocity and angular acceleration of the pendulum. Then we use the odeint function from the scipy library to integrate these equations of motion and plot the angular position of the pendulum as a function of time for two different initial conditions. Finally, we use matplotlib's animation function to animate the pendulum's motion along the path defined by x and y, which is either a tautochrone or parabolic path.

In result we get 2 different animations with taut or parabolic path.

In taut path, tautochrone, we got 2 pendulums dropped from different positions, but the time taken for these objects to reach the lowest point was same.

Chart

Description automatically generatedChart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

In parab path ,parabola, we got 2 pendulums dropped from different positions, and the time taken for these objects to reach the lowest point was different.Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Part three:

We tried to simulate the motion of particles in a two-dimensional space. We first created an array of random positions for a specified number of particles, which are divided into two groups based on their x-coordinate (particles to the right of x=0.5 are colored red, while those to the left are colored blue). We then assigned initial velocities to the particles and calculates all possible pairs of particles. Then we calculate the distance between each pair of particles and checks if the distance is less than a specified cutoff distance (2\*radius). If the distance is less than the cutoff, we use the law of conservation of momentum to compute the new velocities of the colliding particles. Then we define a function to compute the new positions and velocities of the particles over a specified number of time steps (ts) and with a specified time step (dt) using the euler method. We also included a function to compute the distances between all pairs of particles, and a function to compute the new velocities of colliding particles based on their positions and velocities. We gave elastic properties to the box. Then We animate the motion of particles over time using the matplotlib library.

And we created histogram using Maxwell-Boltzmann distribution to compare it to the velocity distribution of particles in animationChart, scatter chart

Description automatically generatedChart, histogram

Description automatically generatedChart, histogram

Description automatically generated