Hw 7

Stefany Fabian Dubón*
Bryn Mawr College
(Dated: April 20th, 2023)

Continued to write a programs to solve first-order diffrential equations using the fourth-order Runge-Kutta and plot said results, as well as to learned how to use varying step size.

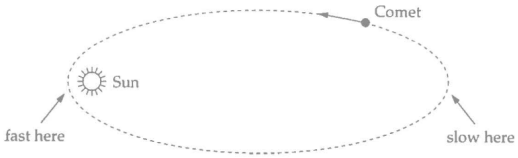
1. INTRODUCTION

After learning how to solve differential equations with more than one variable using the fourth-order Runge-Kutta method in the last assignment, we now explore the idea of letting the step size to vary.

So far we have used repeated steps of a fixed size h, which is chosen by the programmer. However, if we let the step size vary, and let the program choose the best value at each step we can get better results. This is because if we are solving some first-order differential equations, there are some regions where the function is slowly varying, in which case we can accurately capture its shape with a few, widely spaced points. But, if in the central region of the figure the function varies rapidly, then in this region we need points that are more closely spaced. This shows how if we can use this method called *adaptivestepsize*, then we can vary the size h of our steps, making them large in the regions where the solution varies little and small when we need more detail, then we can calculate the whole solution faster, and keeping the accuracy. The basic idea of an adaptive step size method is to vary the step size h so that the error introduced per unit interval in t is roughly constant. In practice this method has two parts, first we have to estimate the error on our step, then we compare the step size to achieve the accuracy we want.

2. EXERCISE 8.10: COMETARY ORBITS

As we know, many comets travel in highly elongated orbits around the Sun. Although they usually are far out in the solar system, moving very slowly; there are rare occasions their orbits bring them close to the Sun for a fly-by and for a small period of time they move very fast:



This is a great example of a system for which an adaptive step size method is useful, since for the large periods of time when the comet is moving slowly we can use long time-steps, so that the program can run quickly, but short time-steps are crucial in the brief but fast-moving period close to the Sun.

We can derive the differential equation obeyed by a comet if we keep in mind that the force between the Sun, with mass M at the origin, and a comet of mass m with position vector r is GMm/r^2 in direction $-\boldsymbol{r}/r$, and hence Newton's second law tells us that

$$m\frac{d^2\boldsymbol{r}}{dt^2} = -\left(\frac{GMm}{r^2}\right)\frac{\boldsymbol{r}}{r} \tag{1}$$

If we cancel the m and take the x component we have:

$$\frac{d^2x}{dt^2} = -GM\frac{x}{r^3} \tag{2}$$

Since the comet stays in a single plane as it orbits, we can throw out one of the coordinates. If we orient our aces so that this plane is perpendicular to the z-axis, we can forget about the z coordinate and we are left with just two second-order equations to solve:

$$\frac{d^2x}{dt^2} = -GM\frac{x}{r^3}, \frac{d^2y}{dt^2} = -GM\frac{y}{r^3} \tag{3}$$

where $r = \sqrt{x^2+y^2}$

For this problem we were asked to first turn these two second-order equations into four first-order equations, then we are have to write a program to solve the equations mentioned using the fourth-order Runge-Kutta method with a fixed step size. Using the mass of the Sun and Newton's gravitational constant G, with the initial conditions of the comet coordinates being x=4 billion kilometers and y= 0 with initial velocity $v_x = 0$ and $v_y = 500ms^{-1}$ and plot the trajectory. Then we are to

*Electronic address: sfabiandub@brynmawr.edu

modify a copy of the code to make the calculation using and adaptive step size instead with a target accuracy of $\delta = 1$ kilometer per year in the position of the comet and again plot the trajectory. Finally, we need to place dots on the graph showing the position of the comet at each Runge-Kutta step around a single orbit.

For part a, I turned the two second-order equations given into the following first-order equations:

$$\frac{dx}{dt} = vx, \frac{dvx}{dt} = -GM\frac{x}{r^3}, \tag{4}$$

and

$$\frac{dy}{dt} = vy, \frac{dvy}{dt} = -GM\frac{y}{r^3}, \tag{5}$$

where G and M are constants.

Part b) I started the program by defining the constants for G, and M as well as the initial conditions given in the problem. I then defined the function f(r,t) that calculates the derivatives of the comet's position and velocity with respect to time (four first-order equations). The function returns an array of derivatives that represent the velocities in the x and y directions (fx, and fv), and the accelerations in the x and y directions (fx, and fvy), which are calculated based on the current position of the comet and the gravitational force exerted by the Sun using Newston's law of gravitation. I then used defined ***T*** to calculate the period of the comet's orbit around the Sun using Kepler's third law $T = 2\pi\sqrt{\frac{a^3}{GM}}$, where a is the initial x coordiate of the comet (x0). Next, I defined a the variables that determine the time step h. I used the $h = 50$ because it let me calculate at least two full orbits of the comet (to make sure it was at least this I defined the T as seen previously). I choose 50 because, when I tried different values and when I used 100 the successive orbits of the comet didn't lie on top of each other. However, it did took a long time for the code to run and produce the graph, which I am still unsure. What I had done before was that I defined a fixed value for nsteps, and use the formula $h = (b-a)/N$ to defined h, but in the plot I would only get a quarter of the elliptical orbit showing. Next, I initialized array tpoints, xpoints, and ypoitns to store the values of time, x coordinates and y coordinates at each time step of the simulation. Then using a for loop, it iterates over the time steps, and for each step, it uses the fourth-order Runge-Kutta method to calculate the comet's position and velocity, and store the updated components in the xpoints and ypoints arrays, as well as the the distance r1 between the comet and the Sun at each time step. Finally, I plotted a graph showing the trajectory of the comet (a plot of y against x). Like I mentioned before, my first attempt with nsteps being 10000, I would only get a quarter of the elliptical orbit of the comet in the simulation, but with the new version (the one submitted) I got multiple periods of the orbit shown in the simulation, which means that I was able to simulate multiple periods, however the successive orbits of the comet don't lie on top of another than makes me believe I am using a value of h that is too high. However, the simulation with 100 lasted a long time, (at least 30 min) and when I tried lower values it either took too long or my computer would force me to quit the terminal because " the system has run out of application memory".

For part c) I used the same code as part b, but I modified to do the calculation using an adaptive step size with a target accuracy of $\delta = 1$ kilometer per year in the position of the comet. For this version of the code I kept the constants and initial conditions the same, as well as the f(r,t) function so that it could calculate the derivatives of the comet's position and velocity with respect to time. I kept the same formula (kepler's third law) to compute the period od the comet's orbit. For the time step and target accuracy I did changed, I kept the same value for a and b since I still want at least two full periods, and $h = 0.01$ as the initial time step since i am expecting it to adapt to the right value later in the code. I also defined the delta s the target accuracy as said in the problem. Then I created arrays for the t, x, and y points as in the previous program. Next, instead of a for loop, I used a while loop to integrate the motion of the comet using the Runge-Kutta method, where it continues as long as the time ***a*** is less that the total time ***b*** and the number of steps taken ***i*** is less than the maximum number of steps ***nsteps***. Here at each step the code calculates the derivatives of the position and velocity using the f function, and it also calculates the error in the position estimate, which it uses to adaptively adjust the time step size h to maintain the desired accuracy ***delta***. For the error formula, I decided to use the one in the book, where I first used the error formula for each variable:

$$\epsilon = \frac{1}{30}(x_1 - x)_2, h = \frac{1}{30}(y_1 - y_2), \tag{6}$$

Now, since we have variables x and y that represent coordinate of a point in a two-dimensional space, we want to perform a calculation that ensures that the Euclidean error in the position of the point meets the target established, we used $\sqrt{\epsilon_x^2 + \epsilon_y^2}$. For the plot, I wasn't able to get a full elliptical orbit like in part a, since it only shows a quarter of the actual orbit, I tried changing the initial step size, the maximum number of steps and even b, but when I tried to use the formula $(b-a)/h$ for the maximum number of steps or a high number like 100000, the code would run for so long my computer would run out of application memory. I tried a lot of variations of this code but this is the one that gave me the best results.

Finally for part d), I used a scatter command to add dots to the plot at each Runge-Kutta step to show the position of the comet at each step around a single orbit.

3. EXERCISE 8.14 A-B

For a single particle of mass m in one dimensional, time-independent Schrödinger equation is:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x) \tag{7}$$

where $\psi(x)$ is the wavefunction, V(x) is the potential energy at position x, and E is the total energy of the particle, potential plus kinetic.

a) We were asked to considered this one dimensional, time independent Schrödinger equation in a harmonic (i.e., quadratic) potential $V(x) = V_0\frac{x^2}{a^2}$, and convert it from a second-order equation to two first-order ones. Then we had to write a program to find the energies of the ground state and the first two states for these equations when m is the electron mass, $V_0 = 50eV$, and $a = 10^{-11}$, with $x = -10ato + 10a$

b) We were then asked to modify the program to calculate the same three energies for the anharmonic oscillator with $V(x) = V_0\frac{x^4}{a^4}$

I approached this exercise by first converting the second-order, one dimensional, time independent Schrödinger equation into two first-order equations, thus

$$\frac{d\psi}{dx} = \phi, \frac{d\phi}{dx} = \frac{2m}{\hbar^2}\left[\frac{a^2}{V_0x^2} - E\right]\psi, \tag{8}$$

using the values given.

For the program, I started with defining the constants and the initial values, I then defines the potential energy function, where using the defined constants to calculate the potential energy based on the given formula. Next, I created the function that defines the two first-order equations that represent the time-independent Schrödinger equation. I also added another function (solve(E)) that takes an energy value E as the input, and then calculates the wavefunction for that energy value using the fourth-order Runge-Kutta. I initialized both psi and phi as 0.0 and 1.0 respectively, and then wrote a for loop that iterates over a range of $x = -10ato + 10a$ as told in the problem, with the step size h. To find the value of the energies of the ground state and the first two excited states for these equations, I used the secant method and initialized E1, E2 and psi2. I then calculated the ground state by setting a a target value for the energy difference between consecutive iterations. Which then enters a while loop that iterated until the energy difference between consecutive iterations is smaller than the target value, and prints the value for the ground state energy. I then used the same process for the first two excited states, with energies initialized as E2, E3, E4 and wave functions calculated using the solve(E) function.

4. RESULTS

4.1. Exercise 8.10

For part a) I got the following four first-order equations from the two second-order equations given:

$$\frac{dx}{dt} = vx, \frac{dvx}{dt} = -GM\frac{x}{r^3}, \tag{9}$$

and

$$\frac{dy}{dt} = vy, \frac{dvy}{dt} = -GM\frac{y}{r^3}, \tag{10}$$

where G and M are constants.

For part b) I got the following plot:

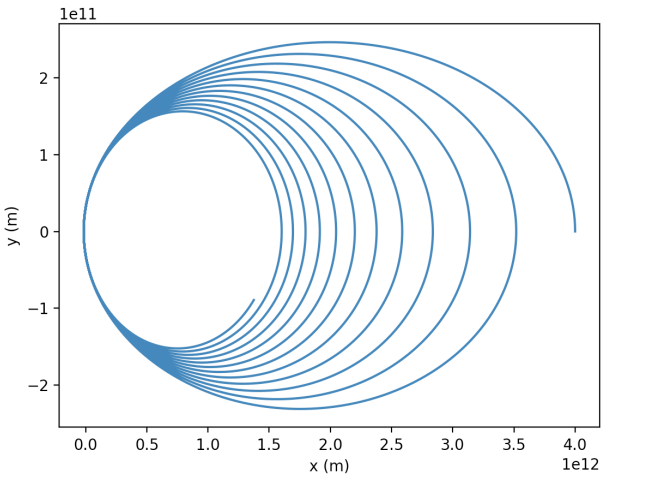


FIG. 1: Graph showing the trajectory of a comet (y against x) with a fixed step size

For part c) I got this graph:

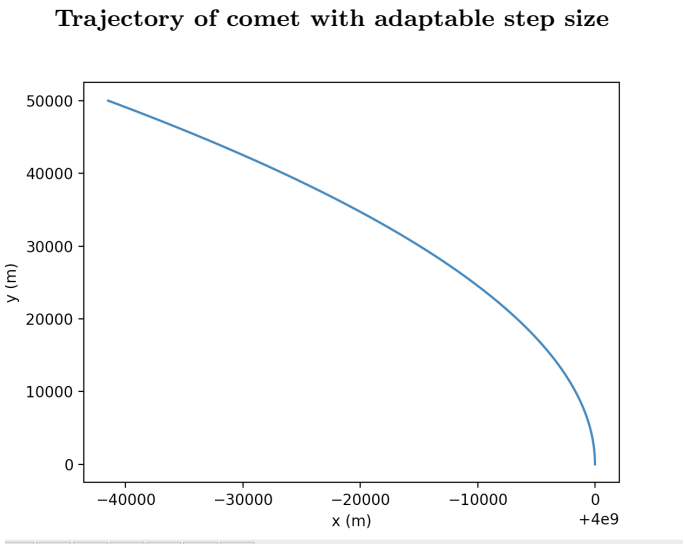


FIG. 2: Graph showing a quarter of the trajectory of a comet with an adaptable step size

Lastly for part d), I got:

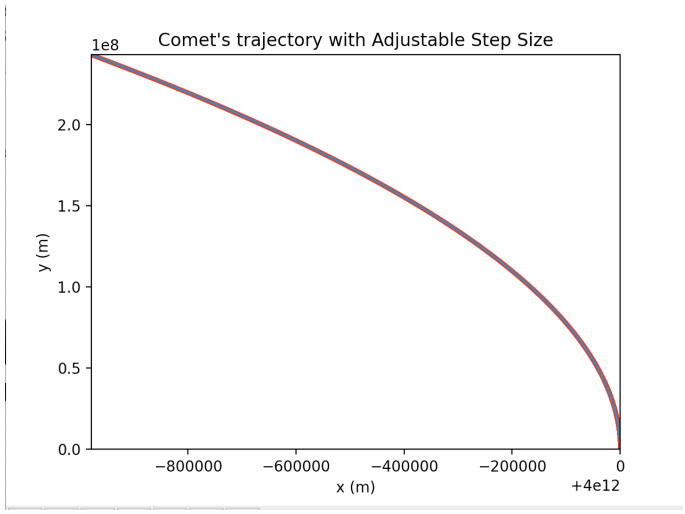


FIG. 3: Graph showing a quarter of the trajectory of a comet with an adaptable step size and dots showing the position of the comet at each Runge-Kutta step

For this graph, the points are so close together that it just look like the line, but once it's zoomed in, you can see the dots are separate and clearer.

4.2. Exercise 8.14

For this exercise, we use the one dimensional, time-independent Schrödinger equation:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x) \tag{11}$$

For part ***a*** we we found the energies of the ground state, and the first two excited states using the potential $V(x) = V_0\frac{x^2}{a^2}$ we get the following results:

Ground state = 138.02397130603683 eV
First exited state = 138.0239714617988 eV
Second exited state = 138.02397158898728 eV

For part ***b*** we used the potential $V(x) = V_0\frac{x^4}{a^4}$ to find the following energy values:
Ground state = 138.02397130603683 eV
First exited state = 138.0239714617988 eV
Second exited state = 138.02397158898728 eV

5. CONCLUSION

Survey Questions This has been the hardest assignment so far. I struggled so much in the first problem. I really like the first, since I got to use the secant method, and found it doable, but the first problem felt like I was doing everything wrong. For part a, I kept on getting either the wrong plot like positive exponential lines, only a part of the orbit or just a blank plot with no data being represented. I also had a hard time waiting for the plots to be computed since my computer would take so long, or it would continuously run out of memory and force me to quit. I now have a better understanding of what a step size is, and the importance of being able to have an adaptable step size (even if my plot is not totally correct). I personally thought the problem was extremely difficult and time consuming overall.