55 · 53.5 · .5 = 113.5

$\boxed{113.5/117}$

Phys 304: Homework 9

Mary Smith[*]
Haverford College Department of Physics

(Dated: April 12, 2024)

[The homework took me 8 hours to complete. I ran into a lot of issues when trying to graph the quantum anharmonic oscillator, and never got the graph to behave like an anharmonic oscillator should. I learned how to use both the fixed and adaptive Runge-Kutta 4th order method. The most interesting problem was the cometary orbits problem, because it visually depicted how the adaptive step size works. The problem set was just the right length.]

✓ .5

1. PROBLEM 1: COMETARY ORBITS

Many comets travel in highly elongated orbits around the Sun. For much of their lives, they are far out in the solar system, moving very slowly, but on rare occasions their orbit brings them close to the Sun for a fly-by and for a brief period of time they move incredibly quickly. ✓

The differential equation obeyed by a comet is straightforward to derive. The force between the Sun, with mass$M$ at the origin, and a comet of mass m with position vector $\mathbf{r}$ is GMm/r^2 in the direction $-\mathbf{r}/r$ (the direction towards the Sun), and hence Newton's second law means that ✓

$$m\frac{d^2r}{dt^2} = -(\frac{GMm}{r^2})\frac{\mathbf{r}}{r}. \tag{1}$$

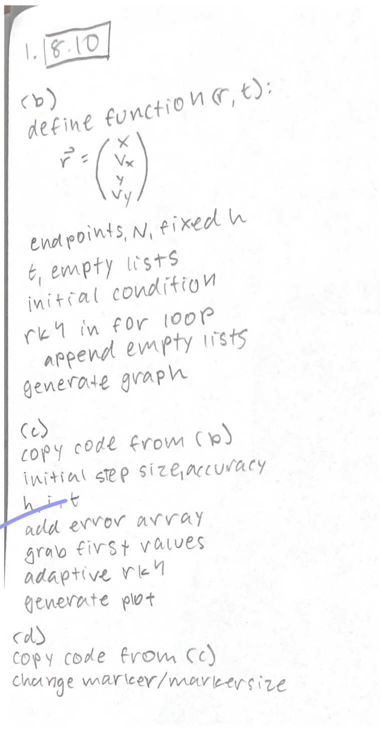
Canceling the m and taking the x component, this gives

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

and similarly for the y and z coordinates. However, if the axes are oriented such that a single plane is perpendicular to the z-axis, the z coordinate can be disregarded and there are just two second-order equations to solve:

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

where $r = \sqrt{x^2+y^2}$. The pseudocode for Python programming for this problem can be found in Figure 1.



```
1. [8.10]
(b)
define function(r, t):
    $\vec{r} = \begin{pmatrix} x \\ v_x \\ y \\ v_y \end{pmatrix}$
endpoints, N, fixed h
t, empty lists
initial condition
rk4 in for loop
    append empty lists
generate graph

(c)
copy code from (b)
initial step size, accuracy
h_init
add error array
grab first values
adaptive rk4
generate plot

(d)
copy code from (c)
change marker/markersize
```

✓

FIG. 1: [The pseudocode for Problem 1, cometary orbits.]

1.1. Part (a): Generating the first-order equations

The two second-order equations in Equation 3 can be simplified into two first-order equations that will be solved in the following sections. Setting dx/dt equal to v_x, and similarly dy/dt equal to v_y, the two equations in Equation 3 can be expressed as four first-order equations:

$$\begin{aligned}\frac{dx}{dt} &= v_x\\ \frac{dv_x}{dt} &= -GM\frac{x}{r^3}\\ \frac{dy}{dt} &= v_y\\ \frac{dv_y}{dt} &= -GM\frac{y}{r^3}.\end{aligned} \tag{4}$$

✓

[*]Electronic address: masmith@haverford.edu

1.2. Part (b): Solving with a fixed step size

A Python program was written according to the pseudocode in Figure 1, using the fourth-order Runge-Kutta method with a fixed step size. As an initial condition, a comet was taken at coordinates x=4 billion kilometers and y=0, with initial velocity $v_x = 0$ and $v_y = 500ms^{-1}$.

In order to do this, a function was defined of $(\mathbf{r}, t)$, where $\mathbf{r}$ is the vector of $(\frac{dx}{dt}, \frac{dv_x}{dt}, \frac{dy}{dt}, \frac{dv_y}{dt})$. A fixed step size h was chosen in order to accurately calculate at least two full orbits of the comet. Then the fixed 4th order Runge-Kutta method was used to append to empty lists for the values of $\mathbf{r}$, given the initial conditions provided. The graph in Figure 2 was then generated. A fixed step size of $h = 31,557.6$ was used to create this graph. The ellipse doesn't perfectly overlap on the right side of the graph, because the fixed step size doesn't account for the speed of the comet increasing as it gets closer to the Sun. The calculation takes about 5 seconds to run.

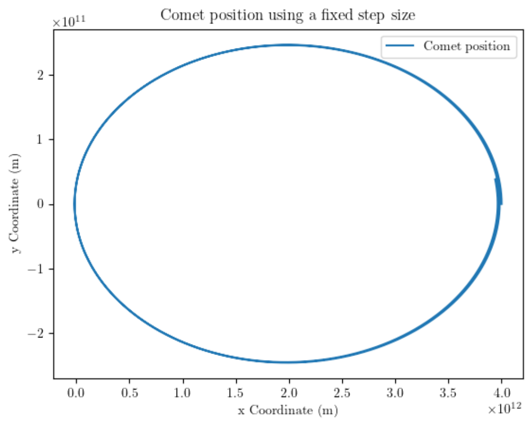


FIG. 2: [The position of a comet orbiting the Sun, approximated using the fixed 4th order Runge-Kutta method.]

1.3. Part (c): Solving with an adaptive step size

An adaptive step size method is extremely useful for cases like this, because for the large periods of time when the comet is moving slowly, long time-steps can be used so that the program runs quickly, but short time steps are crucial in the brief but fast-moving period close to the Sun. The Python program was then modified according to the pseudocode in Figure 1 in order to do the calculation with an adaptive step size. An accuracy of $\delta = 1$ kilometer per year in the position of the comet was set and the trajectory was once again plotted.

In order to do this, the first initial values for the comet were grabbed. Then for the times of interest to this problem, a while loop was created. Inside this while loop, the arrays for different steps were created and then one large step and two small steps were taken according to the adaptive Runge-Kutta 4th order. If the ρ value was sufficiently large, the $\mathbf{r}$ vector was set to be the small step, with an increase in t and h. Else, $\mathbf{r}$ was approximated for an h values based on ρ. This was used to generate the graph in Figure 3. The ellipse in this graph is perfectly aligned. The program runs almost instantly and is extremely accurate when compared to the plot in Figure 2.

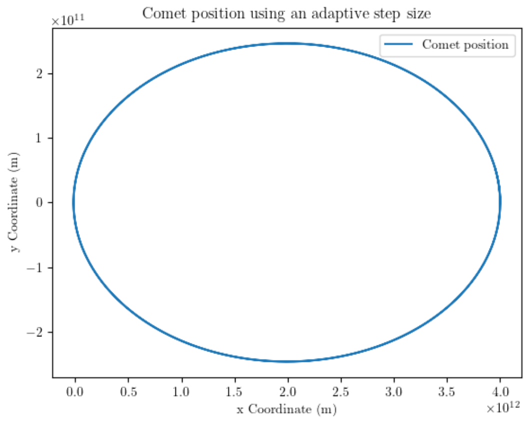


FIG. 3: [The position of a comet orbiting the Sun, approximated using the adaptive 4th order Runge-Kutta method.]

1.4. Part (d): The position at each step

The Python program was then further modified according to the pseudocode in Figure 1. Adjusting the marker and the marker size, dots can be placed on the graph showing the position of the comet at each Runge-Kutta step around a single orbit. This can be found in Figure 4. The steps, represented by dots in the figure, get closer together when the comet is close to the Sun and further apart when it is far out in the solar system.

2. PROBLEM 2: QUANTUM OSCILLATORS

Consider the case of the one-dimensional, time-independent Schrodinger equation in a harmonic potential $V(x) = V_0x^2/a^2$, where V_0 and a are constants.

The pseudocode for this section can be found in Figure 5.

2.1. Part (a): Energy states for the harmonic oscillator

The Time Independent Schrodinger Equation (TISE) for this problem can be written and then converted from a second-order equation to two first-order ones. The TISE for this problem is as follows:

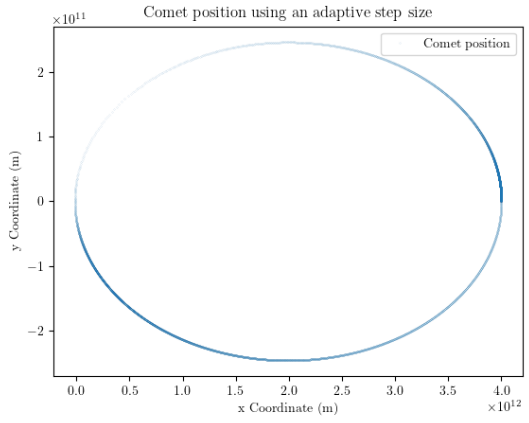
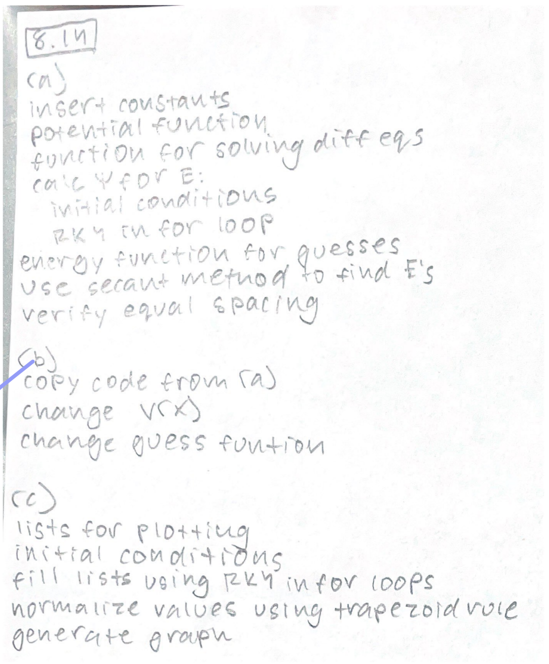


FIG. 4: [The position of a comet orbiting the Sun, showing the step size of the adaptive Runge-Kutta 4th order approximation of trajectory.]



```
8.14
(a)
insert constants
potential function
function for solving diff eqs
calc Ψ for E:
  initial conditions
  RK4 in for loop
energy function for guesses
use secant method to find E's
verify equal spacing

(b)
copy code from (a)
change V(x)
change guess funtion

(c)
lists for plotting
initial conditions
fill lists using RK4 in for loops
normalize values using trapezoid rule
generate graph
```

FIG. 5: [The pseudocode for Problem 2, quantum oscillators.]

$$\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + V(x)\psi(x) = E\psi(x) \tag{5}$$

This means that the TISE can be separated into the two first-order equations as:

$$\begin{aligned}\frac{d\psi}{dx} &= \phi \\ \frac{d\phi}{dx} &= \frac{2m}{\hbar^2}(V(x)-E)\psi.\end{aligned} \tag{6}$$

Then, according to the pseudocode in Figure 5, a Python program was written in order to solve for the first three energy states of the quantum harmonic oscillator for the system being described.

In order to do this, a function was defined of $(\mathbf{r}, t)$, where $\mathbf{r}$ is the vector of (ψ, ϕ). A fixed step size of h was chosen, and then the fixed 4th order Runge-Kutta was applied to solve for $\mathbf{r}$. Then, the secant method was applied to solve for the energies.

To generate guesses for for the secant method, a guess function was created. The potential energy of a harmonic oscillator is $V(x) = \frac{1}{2}kx^2$. Using the given potential energy in the problem, and the fact that $\omega = \sqrt{k/m}$, a guess function using the energy of a harmonic oscillator is created, where

$$E_n = (n + \frac{1}{2})\hbar\omega. \tag{7}$$

This yields energy levels of $E_1 = 138$ eV, $E_2 = 414$ eV, and $E_3 = 690$ eV. The spacings between the energy states is equal to the accuracy from which they were calculated.

2.2. Part (b): Energy states for the anharmonic oscillator

The Python program in Section 2.1 can then be altered for the case of the anharmonic oscillator, with $V(x) = V_0x^4/a^4$. Using the pseudocode in Figure 5, the same three energies can be calculated.

For the anharmonic oscillator, the spacings between energy states get smaller as the energy state number gets larger. This means that the energies vary approximately as $1/n$. Thus, the guess function is built with

$$E_n = (n + \frac{1}{2n})\hbar\omega. \tag{8}$$

This generates energy levels of $E_1 = 205$ eV,$E_2 = 735$ eV, and $E_3 = 1443$ eV. The energy spacings are not equal, which is expected for the anharmonic oscillator.

2.3. Part (c): The anharmonic graph

Using the pseudocode in Figure 5, a graph can be generated from x=-5a to x=5a. Empty lists were created for the wavefunction $\psi(x)$ for the three energy states, with initial conditions applied. Then, using the 4th order Runge-Kutta method with a fixed step size, these lists were populated. The wavefunctions were then normalized using the trapezoidal rule. Finally, the graph in Figure 6 was created.

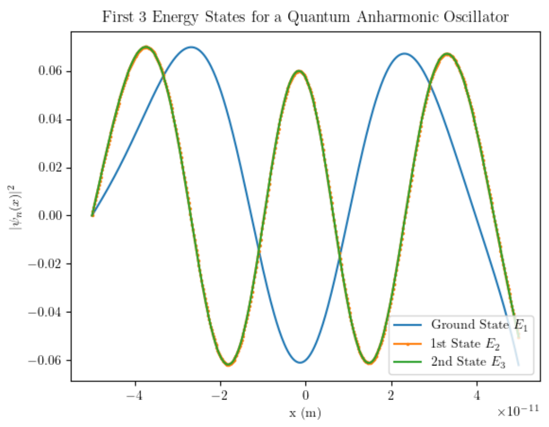


FIG. 6: [The first three energy states for the quantum anharmonic oscillator.]

The graph does not look as expected. For the quantum anharmonic oscillator, there should be one peak for the ground state, 2 peaks for the first excited state, and three peaks for the second excited state. My best guess for the unusual behavior of the graph is that it is an artifact of how I generated my guesses for the secant method.

8.10

55/56

Computational Physics/Astrophysics, Winter 2024:

Grading Rubrics [1]

Haverford College, Prof. Daniel Grin

For coding assignments, roughly 56 points will be available per problem. Partial credit available on all non-1 items.

4 1. Does the program complete without crashing in a reasonable time frame? (+4 points)

2 2. Does the program use the exact program files given (if given), and produce an answer in the specified format? (+2 points)

3 3. Does the code follow the problem specifications (i.e numerical method; output requested etc.) (+3 points)

5 4. Is the algorithm appropriate for the problem? If a specific algorithm was requested in the prompt, was it used? (+5 points)

4 5. If relevant, were proper parameters/choices made for a numerically converged answer? (+4 points)

3.5 6. Is the output answer correct? (+4 points).

2.5 7. Is the code readable? (+3 points)

. 5.1. Are variables named reasonably?

. 5.2. Are the user-functions and imports used?

error with adaptive step size, we expect smaller steps on left side of graph -0.5

[1] Inspired by rubric of D. Narayanan, U. Florida, and C. Cooksey, U. Hawaii

. 5.3. Are units explained (if necessary)? missing some units -0.5

. 5.4. Are algorithms found on the internet/book/etc. properly attributed?

3 8. Is the code well documented? (+3 points)

. 6.1. Is the code author named?

. 6.2. Are the functions described and ambiguous variables defined?

. 6.3. Is the code functionality (i.e. can I run it easily enough?) documented?

9. Write-up (up to 28 points)

5 . Is the problem-solving approach clearly indicated through a flow-chart, pseudo-code, or other appropriate schematic? (+5 points)

✓. Is a clear, legible LaTeX type-set write up handed in?

3 . Are key figures and numbers from the problem given? (+ 3 points)

4 . Do figures and or tables have captions/legends/units clearly indicated. (+ 4 points)

3 . Do figures have a sufficient number of points to infer the claimed/desired trends? (+ 3 points)

2 . Is a brief explanation of physical context given? (+2 points)

1 . If relevant, are helpful analytic scalings or known solutions given? (+1 point)

3 . Is the algorithm used explicitly stated and justified? (+3 points)

2 . When relevant, are numerical errors/convergence justified/shown/explained? (+2 points)

2 . Are 3-4 key equations listed (preferably the ones solved in the programming assignment) and algorithms named? (+2 points)

1 . Are collaborators clearly acknowledged? (+1 point)

2 . Are any outside references appropriately cited? (+2 point)

8.14

53.5/56

Computational Physics/Astrophysics, Winter 2024:

Grading Rubrics [1]

Haverford College, Prof. Daniel Grin

For coding assignments, roughly 56 points will be available per problem. Partial credit available on all non-1 items.

4 1. Does the program complete without crashing in a reasonable time frame? (+4 points)

2 2. Does the program use the exact program files given (if given), and produce an answer in the specified format? (+2 points)

2.5 3. Does the code follow the problem specifications (i.e numerical method; output requested etc.) (+3 points)

missing groundstate plot -0.5

5 4. Is the algorithm appropriate for the problem? If a specific algorithm was requested in the prompt, was it used? (+5 points)

4 5. If relevant, were proper parameters/choices made for a numerically converged answer? (+4 points)

3 6. Is the output answer correct? (+4 points).

Error calculating psi, wrong E2/E3 values causes ground state to not plot and 2nd level plot twice -1

3 7. Is the code readable? (+3 points)

. 5.1. Are variables named reasonably?

. 5.2. Are the user-functions and imports used?

[1] Inspired by rubric of D. Narayanan, U. Florida, and C. Cooksey, U. Hawaii

. 5.3. Are units explained (if necessary)?

. 5.4. Are algorithms found on the internet/book/etc. properly attributed?

3 8. Is the code well documented? (+3 points)

. 6.1. Is the code author named?

. 6.2. Are the functions described and ambiguous variables defined?

. 6.3. Is the code functionality (i.e. can I run it easily enough?) documented?

9. Write-up (up to 28 points)

5 . Is the problem-solving approach clearly indicated through a flow-chart, pseudo-code, or other appropriate schematic? (+5 points)

✓ . Is a clear, legible LaTeX type-set write up handed in?

3 . Are key figures and numbers from the problem given? (+ 3 points)

4 . Do figures and or tables have captions/legends/units clearly indicated. (+ 4 points)

3 . Do figures have a sufficient number of points to infer the claimed/desired trends? (+ 3 points)

2 . Is a brief explanation of physical context given? (+2 points)

0 . If relevant, are helpful analytic scalings or known solutions given? (+1 point) compare to expected −1

3 . Is the algorithm used explicitly stated and justified? (+3 points)

2 . When relevant, are numerical errors/convergence justified/shown/explained? (+2 points)

2 . Are 3-4 key equations listed (preferably the ones solved in the programming assignment) and algorithms named? (+2 points)

1 . Are collaborators clearly acknowledged? (+1 point)

2 . Are any outside references appropriately cited? (+2 point)