PHY407 (Lab 7: Ordinary Differential Equations, Pt. II)

Utkarsh Mali[1] and Aslesha Pokhrel[1,2]

[1]Department of Physics, University of Toronto
[2]Department of Computer Science, University of Toronto

October 31, 2020

Contents

Work Allocation:
Utkarsh: Question 1, Question 2
Aslesha: Question 3

Question 1: Return of Space Garbage

(a) This question asks us to redo the code but using a varying step size. We use $N = 10000$ and are asked to plot each individual step sizes. We changed the code in order to accommodate the adaptive time step. **The effect of the adaptive can be clearly seen as the steps are closer along sharper curves.** We enforced a varying time step by including a `rho` function which calculates our characteristic value which was then used to change the value of our varying h. Note, as Newman suggested on page 359, we set our h to be a minimum of a constant and h' in order to avoid an large time step jump.

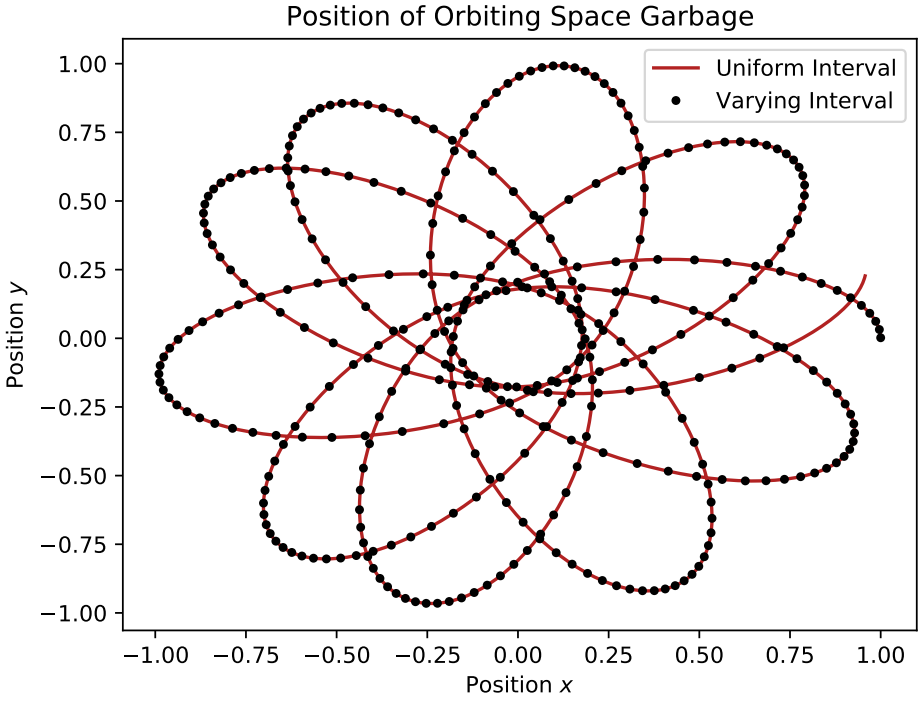

Figure 1: Here we observe the plot showing the orbit of our space garbage in a given time frame. **We observe that the varying time steps are closer together when the change in gradient is much higher than when there is a relative lower change in gradient.**

(b) We are now asked to use a timer to compare the time taken for both methods.

```
CODE OUTPUT:
    Uniform Interval Runtime: 0.5895 sec
    Varying Interval Runtime: 0.0753 sec
```

We observe that the varying interval runtime was an order of magnitude smaller than the uniform interval runtime. This matches Newman's hypothesis describing the efficiency of the time varying interval method.

(c) We now use our results to provide a plot of step size vs time

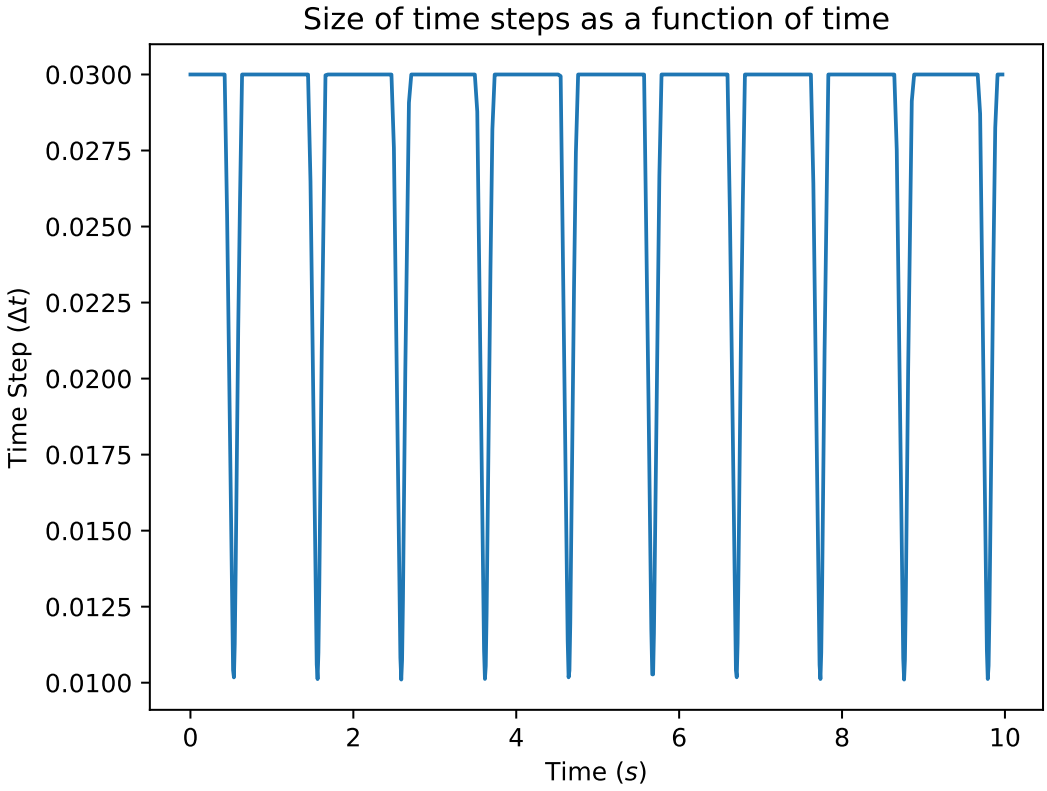

Figure 2: We can see that when needed the step size reduces down to 0.01 but most of the time it can be increased to larger values.

Note: We truncated our h to a certain value as suggested by Newman in order to eliminate overstepping which could make the program miss certain values. As a result, our step is is limited to a maximum of 0.03. If we did not truncate our h, then we would observe a sharp triangle like structure, but this would result in the program skipping entire loops of the orbit of the space garbage. In order to keep all the values, as well as keep the relative run-time low we made the decision to set `h = min(h, 0.015)`. This provided the optimum time jump while maintaining a sharp drop in run time. **When the gradient change is very steep, we notice that the time step is very short, and when the gradient change is very small, we observe a much larger time step.**

Question 2: Bulirsch-Stoer method: Earth and Pluto's orbits

(a) This question asks us to write a program which calculated the orbit of the earth using the Bulirsch-Stoer method. Our code is an adaptation of the `bulirsch.py` method as permitted by Prof Grisouard. We redefined our constants as stated in the question and chose our constant H to be 1 week. We also set a new definition for the given function to `f(r)`. And changed the dimension of the code execution as well as the error calculate to add in quadrature. We chose to run the loop for 54 weeks in order to get a complete year as requested by the question.

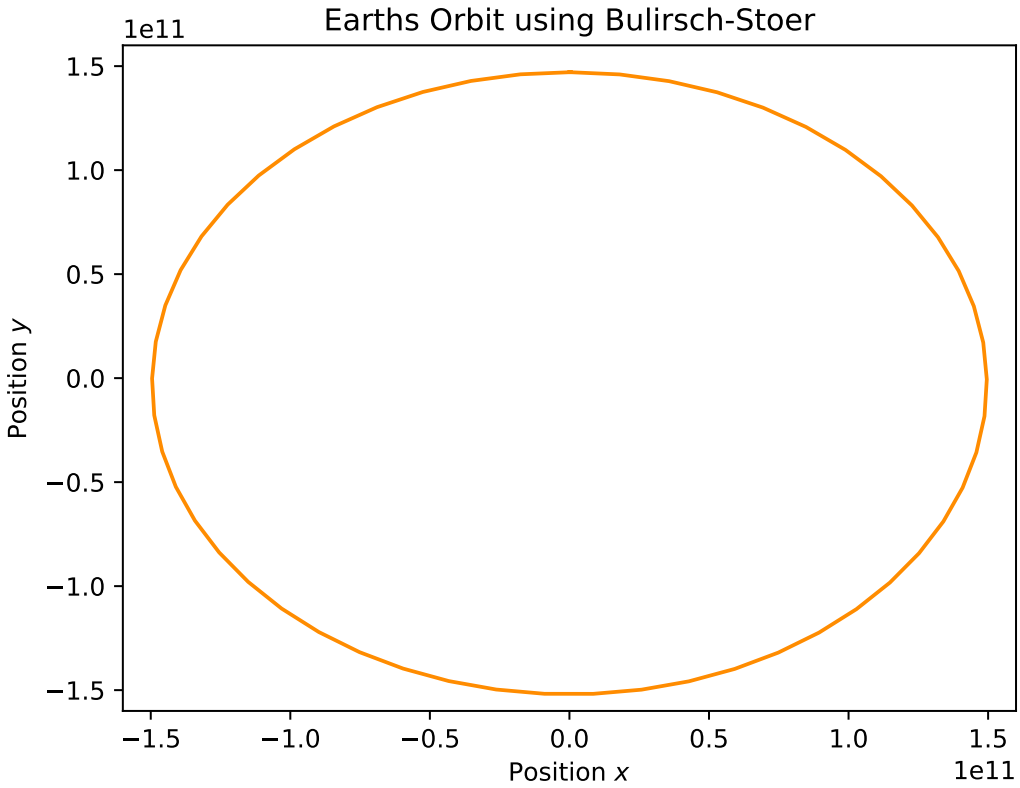

Figure 3: We observe Earth's orbit around the sun as a circular orbit with radius 1.5e11 which is equal to 1AU.

(b) We are now asked to modify our program in order to plot the orbit of Pluto, we should observe an ellipsoid orbit. We modified our initial conditions for the position and velocity of Pluto, we also change H to be H = 0.5 representing a jump every 6 months in order to save computing time. Our error and method of calculation was left unchanged. Once again we are using a modified version of `bulirsch.py`.

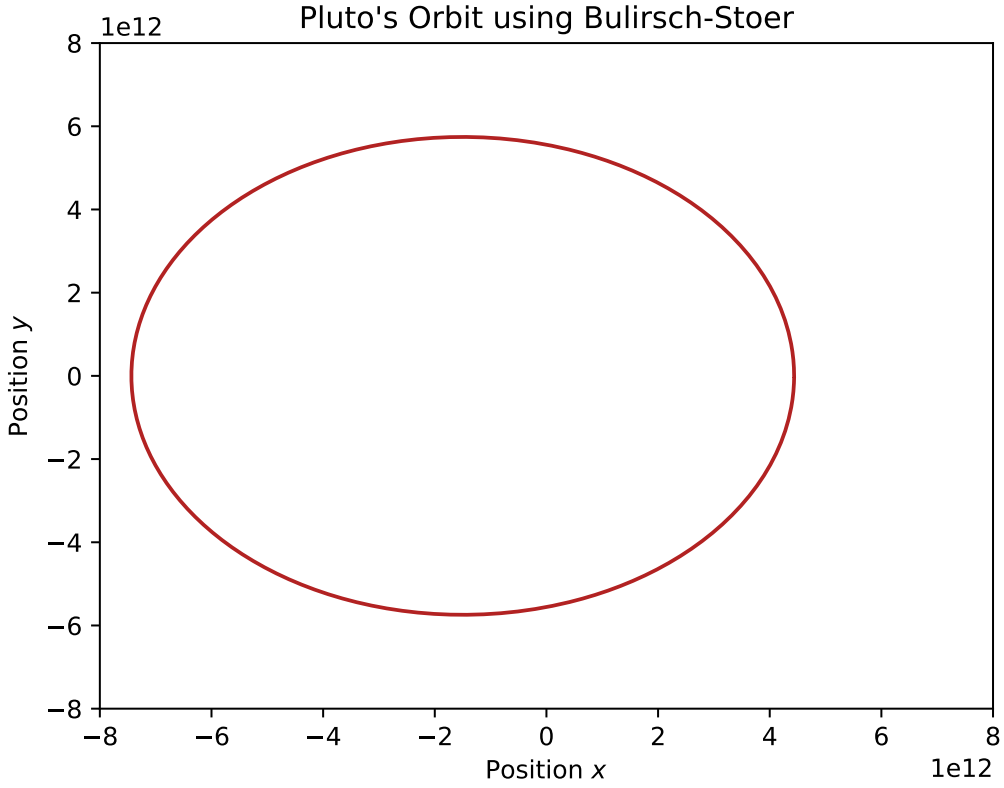

Figure 4: We observe Pluto's orbit around the sun as a elliptical orbit, slightly off centered.

```
PSEUDOCODE:
# import
# QUESTION 2A
# Setting constants, some are taken from Newman812a-sol.py provided by Prof Grisouard
# Mass of the Sun
# Constant to convert to year time
# Gravitational Constant
# Divide by 53 since we are working in years
# Iniital x
# Initial y
# Initial xdot
# Initial ydot
# Accuracy given in the question
# Define Function as given in the book.
# Set empty arrays
# Set initializing vector
# Do the "big steps" of size H, looping for 1 year
# Loop over using Newman's Bulirsch-Stoer Method
    # Update initial points into final arrays
    # Do one modified midpoint step to get things started
    # The array R1 stores the first row of the
    # extrapolation table, which contains only the single
    # modified midpoint estimate of the solution at the
    # end of the interval
    # Set the empty dimension to 4
    # Now increase n until the required accuracy is reached
        # Modified midpoint method
        # Calculate extrapolation estimates. Arrays R1 and R2
        # hold the two most recent lines of the table
        # Set the empty dimension to 4
        # Take the error of both x and y values.
    # Set r equal to the most accurate estimate we have,
    # before moving on to the next big step
# Plot the results

# QUESTION 2B
# Setting constants, like previously. Change values of H, and initial setup
# Repeat steps in Question 2a with Pluto's initial conditions.
```

Question 3: The hydrogen atom

(a) The code was adapted from Newman's squarewell.py online to solve equation (2) given in the handout using RK4 and find the energies using the shooting method. There is nothing to be submitted for this question.

(b) In this section, we numerically calculate the energy for given state n and order l by using the code from part (a). Let the energy for $n = 1$ and $l = 0$ be $E_{1,0}$, for $n = 2$ and $l = 0$ be $E_{2,0}$ and, for $n = 2$ and $l = 1$ be $E_{2,1}$. We will look at the effects of adjusting the parameters h, r_∞ and *target* as discussed in the handout on these energy values. Analytically, $E_{1,0} = -13.6$ eV and $E_{2,0} = E_{2,1} = -3.4$ eV. Let a be the Bohr radius in meters and e be the elementary charge. The numerical values of $E_{1,0}, E_{2,0}$ and $E_{2,1}$ rounded to 2 significant digits for different parameters are summarized in the table below:

h	r_∞	target	$E_{1,0}$ (eV)	$E_{2,0}$ (eV)	$E_{2,1}$ (eV)	$\lvert E_{2,0} - E_{2,1}\rvert$ (eV)
0.002a	20a	e/1000	-13.49	-3.39	-3.40	1.49×10^{-2}
0.002a	40a	e/1000	-13.49	-3.39	-3.40	1.48×10^{-2}
0.002a	50a	e/1000	-13.02	-3.39	-3.40	1.48×10^{-2}
0.0001a	20a	e/1000	-13.60	-3.40	-3.40	9.63×10^{-4}
0.0001a	20a	e/10000	-13.60	-3.40	-3.40	9.63×10^{-4}

Table 1: Table showing the energy values for change in parameters. Here, a is the Bohr radius in meters and e is the elementary charge.

From table 1 we can see that decreasing h improves the solution as all the energies get closer to the true value mentioned above and the difference between $E_{2,0}$ and $E_{2,1}$ also decreases as desired. However, decreasing h too much slows down the program significantly. Similarly, increasing r_∞ did not seem to improve the solution significantly but if r_∞ is too large, the value of $E_{1,0}$ starts to deviate from the true solution. This is likely due to the error accumulation in the RK4 iterations. Likewise, decreasing the target energy convergence further from $e/1000$ to $e/10000$ did not have any impact on the solution accuracy but it increased the run-time of the program.

(c) In this part, the program was modified to plot the normalized eigenfunction R. The function was normalized by calculating by $\int |R(r)|^2 dr$ over the range of r using the Simpson's rule and dividing the eigenfunction R by the square-root of this value. The in-built function `scipy.integrate.simps` was used to calculate the integral and the number of slices are determined by the value of h and r_∞ since $R(r)$ that we are integrating over is discrete here. There is nothing to be submitted for this question.

(d) In this section, we plot the explicit solutions for R(r) given in the handout along with the numerically calculated solution obtained in part (c). The numerical plots shown below are obtained by setting the parameters to $h = 0.0001a, r_\infty = 20a, target = e/1000$.

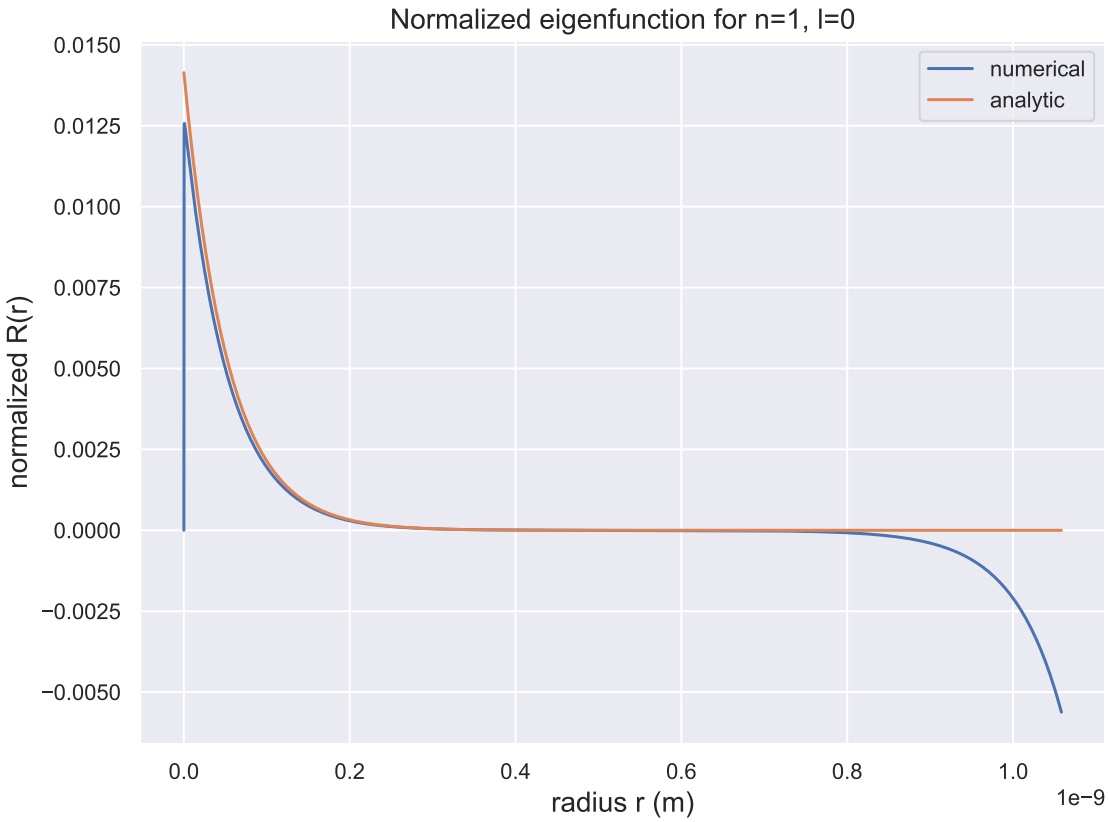

Figure 5: The plot showing the normalized eigenfunctions of numerically calculated and analytically determined R(r) for $n = 1, l = 0$. We obtain the spurious large values of R at the right-hand end of the wave function due to the reasons explained in question 3(c) in the handout i.e. the error in the approximation in energy value causes the R to diverge.

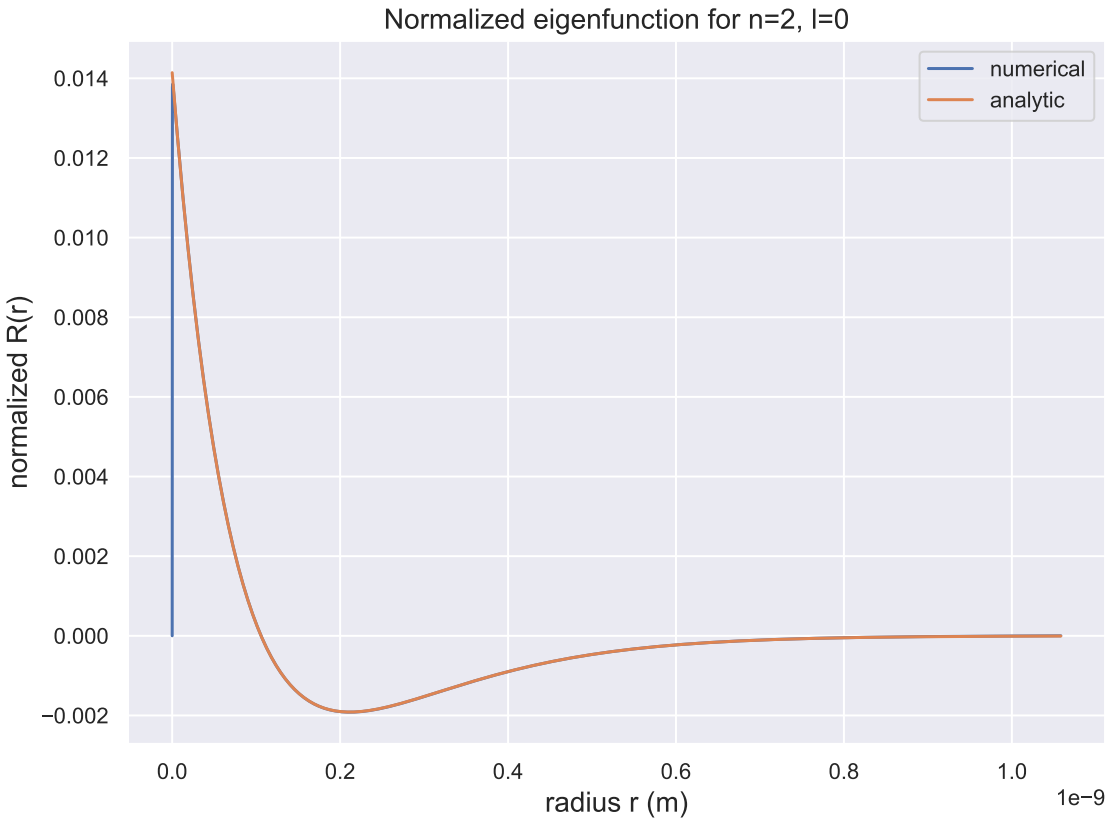

Figure 6: The plot showing the normalized eigenfunctions of numerically calculated and analytically determined R(r) for $n = 2, l = 0$.

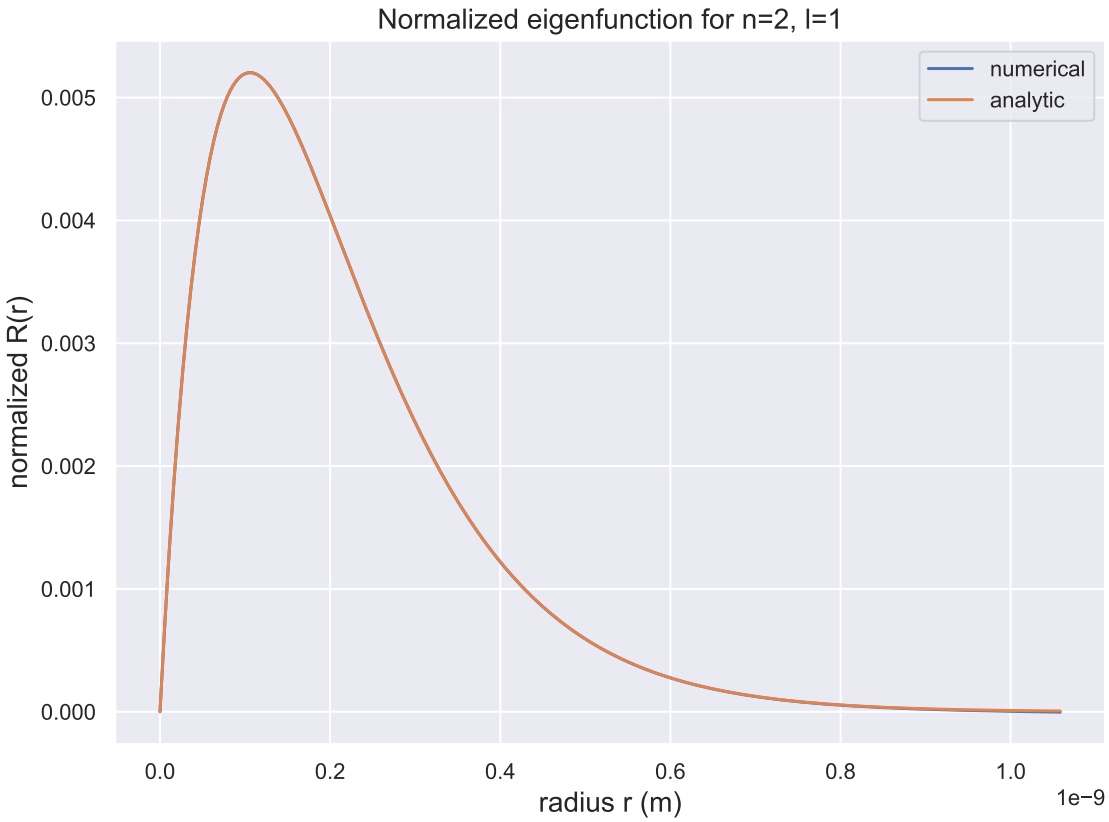

Figure 7: The plot showing the normalized eigenfunctions of numerically calculated and analytically determined R(r) for $n = 2, l = 1$.

From fig. 5, fig. 6 and fig. 7 we can see that the overall shape of the eigenfunction R calculated numerically closely matches the analytical solution. In fig. 5 and fig. 6, there is an initial disagreement between the numerical and the analytical solution because we initialize $R(h)$ to 0 for convenience which becomes a source of inaccuracy. Similarly, in fig. 5, the value of R diverges at the right-hand end (r_∞) of the wave function because the energy we have is not exact which causes the value of R to diverge as it is very sensitive to the energy value. Likewise, the zero-crossings for the the numerical solution matches the analytic solution for the wave functions except for the one in fig. 5. As discussed above, this happens because of the error in the approximation of energy.