

TU WIEN

BACHELOR THESIS

Solving The Stationary One Dimensional Schrödinger Equation With The Shooting Method

Author:

Marie Christine ERTL
0725445

Supervisor:

Dipl. Ing., Dr. techn. Franz
SCHANOVSKY

Examiner:

Ao.Univ.Prof. Dipl.-Ing.
Dr.techn. Erasmus LANGER

*A thesis submitted in fulfilment of the requirements
for the degree of Bachelor of Science*

in the

Faculty of Electrical Engineering and Information Technology
Institute for Microelectronics

September 2016

Declaration of Authorship

I, Marie Christine ERTL, declare that this thesis titled, 'Solving The Stationary One Dimensional Schrödinger Equation With The Shooting Method' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

TU WIEN

Abstract

Faculty of Electrical Engineering and Information Technology
Institute for Microelectronics

Bachelor of Science

Solving The Stationary One Dimensional Schrödinger Equation With The Shooting Method

by Marie Christine ERTL
0725445

The Schrödinger equation is the fundamental quantum mechanical equation. However, only for a handful of cases it can be solved analytically, requiring a decent numerical method for systems where no analytical solution exists.

The shooting method is a numerical method to solve differential equations such as the Schrödinger equation where the boundary conditions are known and certain parameters to solve the equations have to be found. In this thesis we study the parameter energy as the eigenvalue of the system. We take the initial condition of the equation as the starting point and shoot with a defined initial value. Then we observe whether the solution comes close enough to the second boundary condition. If this is the case we then refine it further to a specified accuracy. Two different approaches for the shooting method are presented.

In this work the Schrödinger equation is solved for three cases: the infinite potential well, the quantum harmonic oscillator and the radial part of the hydrogen Schrödinger equation. Each case has an analytical solution which makes them perfect testing material for the suitability of the shooting method. To demonstrate the method's accuracy we compare the numerical solutions to their analytical counterparts. Overall, the results match the analytical solutions proving the shooting method to be a useful tool for obtaining numerical solutions for the Schrödinger equation.

Acknowledgements

First and foremost I want to thank my family for their support.

Very special thanks go to Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Erasmus Langer and to my advisor Dipl.-Ing. Dr. techn. Franz Schanovsky for their patience and guidance.

Contents

Declaration of Authorship	i
Abstract	ii
Acknowledgements	iii
Contents	iv
List of Figures	v
1 Introduction	1
2 Background and Related Work	3
2.1 The Stationary Schrödinger Equation	3
2.2 The Fourth Order Runge Kutta Method	4
2.3 Newton Raphson Method	6
2.4 The Shooting Method – Approach 1	7
2.5 The Shooting Method – Approach 2	7
2.6 Related Work	8
2.7 Python as Programming Language	9
3 Implementation	11
3.1 Infinite Potential Well	11
3.2 Quantum Harmonic Oscillator	18
3.3 Radial Hydrogen Schrödinger Equation	24
4 Lessons Learned	30
5 Summary and Outlook	33
5.1 Summary	33
5.2 Outlook	34
A Shooting Method – Approach 1	35
B Shooting Method – Approach 2	41
C Shooting Method Source Code for Hydrogen Atom	46
Bibliography	49

List of Figures

3.1	Infinite Potential Well.	12
3.2	Screening for Possible Energy Values for the Infinte Potential Well.	13
3.3	Wave Function Comparison for Ground State of the Infinite Potential Well.	14
3.4	Wave Function Comparison for First Excited State of the Infinite Potential Well.	15
3.5	Wave Function Comparison for Second Excited State of the Infinite Potential Well.	15
3.6	Wave Function Comparison for Third Excited State of the Infinite Potential Well.	16
3.7	Wave Function Comparison for Fourth Excited State of the Infinite Potential Well.	16
3.8	Console Output of the Program for Approach 2.	17
3.9	Quantum Harmonic Oscillator Potential.	18
3.10	Screening of Input Values for the Quantum Harmonic Oscillator with Analytical Solution for Comparison.	20
3.11	Wave Function Comparison for the Ground State of the Quantum Harmonic Oscillator.	21
3.12	Wave Function Comparison for the First Excited State of the Quantum Harmonic Oscillator.	22
3.13	Wave Function Comparison for the Second Excited State of the Quantum Harmonic Oscillator.	22
3.14	Wave Function Comparison for the Third Excited State of the Quantum Harmonic Oscillator.	23
3.15	Wave Function Comparison for the Fourth Excited State of the Quantum Harmonic Oscillator.	23
3.16	Hydrogen Atom Potential.	24
3.17	Screening of Input Values for the Radial Hydrogen Schrödinger Equation with Analytical Solution for Comparison.	26
3.18	Hydrogen Wave Function for 1s Orbital.	27
3.19	Hydrogen Wave Function for 2s Orbital.	28
3.20	Hydrogen Wave Function for 2p Orbital.	29
3.21	Hydrogen Wave Function for 3s Orbital.	29
4.1	Overshooting occurring due to coarse energy mesh.	32

Chapter 1

Introduction

Imagine you have a rifle and a target. You set the rifle on an aiming block where you can adjust the angle of fire. Now you aim the gun at the target, fire and see where the bullet hits. You can now use that information to adjust your aim. If the bullet flew over the target, you aim lower. If it was too low, you aim higher. In the event that the target is out of reach of your rifle, you have to move in closer. You repeat this step for as many times as needed, refining your trajectory until you hit the target where you want it to.

In general mathematical terms one has an equation and its boundary conditions. One needs to obtain a parameter that solves this equation for said boundary conditions. In some cases one might have a reference helping with the initial guess for this parameter. An important quantum mechanical equation is the Schrödinger equation, yielding wave functions as its solution, e.g.: for a particle trapped in a certain potential. However, it is rarely possible to solve this equation analytically. Therefore, this thesis' aim is to find a numerical method with the ability to provide accurate numerical solutions, where no analytical counterpart exists.

In this thesis two approaches of the shooting method are used to obtain energy eigenstates and their corresponding wave functions from the Schrödinger equation for three quantum mechanical systems: the infinite potential well, the quantum harmonic oscillator and the radial Schrödinger equation of the hydrogen atom. Knowing the Schrödinger equation and both boundary conditions, the solutions for arbitrary energies can be computed with a numerical integration method. Taking the value that approaches the second boundary conditions the best, the matching energy values can then be refined with either an interpolation method, or with shooting as often as needed. As the shooting method's mathematical tools the fourth order Runge Kutta integration method is used

as the solver and Newton Raphson's algorithm for refining initial results. Although all problems used in this thesis have an analytical solution, they are good testing material as the computed solutions are easily comparable with their analytical equivalent.

The motivation for this thesis is to test out the shooting method implemented with Python and to see how it can be used for solving the Schrödinger equation.

Python is a well suited language for scientific programming with clear, easily readable syntax and add-on packages for many computing needs. This thesis uses the `Sci-py` stack's extensive libraries and the `matplotlib` plotting environment. It is free and operating system independent, making it easily transferable. Python enthusiasts all over the world help to extend its features on a daily basis [1].

Outline of this thesis

This thesis is outlined as follows:

Introduction: This chapter familiarizes the reader with the thesis topic and briefly explains how the shooting method works in general terms. It further shows the motivation for this work.

Background and Related Work: This chapter introduces the Schrödinger equation and the mathematical methods used for the shooting method. It reasons why python was chosen as the programming language. Related work to this thesis is presented.

Implementation: This chapter acquaints the reader with the preparation and implementation of the chosen examples. It presents the solutions and compares them to their analytical counterparts to better highlight the suitability of the shooting method.

Lessons Learned: Outlines the difficulties experienced and how they were overcome.

Summary: This chapter summarizes this thesis.

Chapter 2

Background and Related Work

The Schrödinger equation is the fundamental equation in quantum mechanics. As stated before in Chapter 1 it is not possible to solve it analytically for most quantum mechanical systems. Therefore, this Chapter presents two approaches of the shooting method aiming to solve cases of the stationary one dimensional Schrödinger equation. This Chapter introduces the Schrödinger equation and the numerical methods used. Related work is presented and the advantages of using `Python` for scientific computing is discussed briefly.

2.1 The Stationary Schrödinger Equation

The stationary three dimensional Schrödinger equation for some arbitrary potential V is given by [2]:

$$\left\{-\frac{\hbar^2}{2m}\nabla^2 + V\right\}\Psi = \epsilon\Psi, \quad (2.1)$$

with the electron mass $m = 9,1 \times 10^{-31}$ kg, its charge $e = 1.6 \times 10^{19}$ C and the reduced Planck constant $\hbar = 1.05 \times 10^{-34}$ Js, ϵ representing the energy and ∇^2 being a three dimensional partial second derivative operator. In this thesis we will reduce equation 2.1 to one dimension (equ.:2.2)

$$-\frac{\hbar}{2m} \frac{d^2}{dx^2} \Psi(x) + V(x)\Psi(x) = \epsilon\Psi(x). \quad (2.2)$$

We then solve it numerically for the potentials of the infinite potential well (equ.: 2.3)

$$V(x = 0) = \infty, V(x = L) = \infty, \quad (2.3)$$

the quantum harmonic oscillator (equ.: 2.4)

$$V(x) = \frac{1}{2}kx^2 = \frac{1}{2}m(\omega x)^2, \quad (2.4)$$

and the radial part of the hydrogen Schrödinger equation¹ (equ.: 2.5).

$$V(r) = \frac{e^2}{4\pi\epsilon_0 r}. \quad (2.5)$$

2.2 The Fourth Order Runge Kutta Method

The fourth order Runge Kutta method, often abbreviated as RK4, is a numerical integration method based on the Euler formalism. It is used to numerically solve first order ordinary differential equations (ODEs) requiring only one initial value. While the Euler formalism uses only the previous value of the function to obtain the value at the next grid point, the Runge Kutta method calculates the function at intermediate intervals between two grid points. This increases accuracy and the accumulated error corresponds to $O(h^4)$ as compared to the error of the Euler formalism which is in the order of $O(h)$ shown by Brooks [2], with h as the chosen step size.

The equation system for the slopes between the grid points of the fourth order Runge Kutta method is shown in equation 2.6 [3]. Starting at a grid point y_i equation 2.6a calculates the estimated slope at the beginning of the interval. Equation 2.6b then calculates the slope at midpoint using $\Delta y^{(1)}$ to estimate the value in the middle of the two grid points. Equation 2.6c does the same now using $\Delta y^{(2)}$'s slope to obtain a better estimate. Equation 2.6d determines the slope at the end of the interval using $\Delta y^{(3)}$. The next grid point is then calculated using equation 2.7.

¹Due to the Schrödinger equation for hydrogen being more reasonably displayed in polar coordinates, we use the radius r in place of x .

$$\Delta y^{(1)} = hf(x_i, y_i) \quad (2.6a)$$

$$\Delta y^{(2)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(1)}}{2}\right) \quad (2.6b)$$

$$\Delta y^{(3)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(2)}}{2}\right) \quad (2.6c)$$

$$\Delta y^{(4)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(3)}}{2}\right) \quad (2.6d)$$

$$y_{i+1} = y_i + \frac{1}{6}(\Delta y^{(1)} + 2\Delta y^{(2)} + 2\Delta y^{(3)} + \Delta y^{(4)}) \quad (2.7)$$

The step size h is obtained from the boundary values and the number of chosen grid points n through

$$h = \frac{x_n - x_0}{n}. \quad (2.8)$$

Since equation 2.2 is a second order differential equation we cannot solve it directly with RK4. Therefore, we need to split it into an equation system of coupled first order differential equations $f(x_i, y_i, z_i)$ and $g(x_i, y_i, z_i) = f'(x_i, y_i, z_i)$. Equation 2.9 shows the adapted equation system from equation 2.6 [2, 3]. Now that we have two first order differential equations which are dependent on each other we need to apply the Runge Kutta method for each one. Instead of having only slopes Δy^m from one equation, we now need to consider also the slopes from the second equation Δz^m , with $m = 1, 2, 3, 4$.

$$\Delta y^{(1)} = hf(x_i, y_i, z_i) \quad (2.9a)$$

$$\Delta z^{(1)} = hg(x_i, y_i, z_i) \quad (2.9b)$$

$$\Delta y^{(2)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(1)}}{2}, z_i + \frac{\Delta z^{(1)}}{2}\right) \quad (2.9c)$$

$$\Delta z^{(2)} = hg\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(1)}}{2}, z_i + \frac{\Delta z^{(1)}}{2}\right) \quad (2.9d)$$

$$\Delta y^{(3)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(2)}}{2}, z_i + \frac{\Delta z^{(2)}}{2}\right) \quad (2.9e)$$

$$\Delta z^{(3)} = hg\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(2)}}{2}, z_i + \frac{\Delta z^{(2)}}{2}\right) \quad (2.9f)$$

$$\Delta y^{(4)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(3)}}{2}, z_i + \frac{\Delta z^{(3)}}{2}\right) \quad (2.9g)$$

$$\Delta z^{(4)} = hg\left(x_i + \frac{h}{2}, y_i + \frac{\Delta y^{(3)}}{2}, z_i + \frac{\Delta z^{(3)}}{2}\right) \quad (2.9h)$$

We now obtain the next grid points for each equation through equation 2.10. The step size is the same as before (equ.: 2.8).

$$y_{i+1} = y_i + \frac{1}{6}(\Delta y^{(1)} + 2\Delta y^{(2)} + 2\Delta y^{(3)} + \Delta y^{(4)}) \quad (2.10a)$$

$$z_{i+1} = z_i + \frac{1}{6}(\Delta z^{(1)} + 2\Delta z^{(2)} + 2\Delta z^{(3)} + \Delta z^{(4)}) \quad (2.10b)$$

2.3 Newton Raphson Method

The Newton Raphson method is an algorithm for finding a function's zero, helping us to refine our rather inadequate first guesses obtained through the Runge Kutta method. The Newton Raphson method we use here is taken from *Bakkalaureats-Vertiefung Mathematik, Teil 1 Numerische Methoden* [3]. The mathematical formulation is shown in equation 2.11. Using the function's value and dividing it with its first derivative gives a step size (equ.: 2.11a) which is in turn used to calculate the next step (equ.: 2.11b). If Δx reaches a certain small value (e.g. in the order of 10^{-6} to 10^{-12}), depending on how exact a solution is needed, the algorithm terminates. The adequately refined solution is then x_{n+1} . The Newton method used in the script is imported from the `scipy.optimize` module.

$$\Delta x = -f(x_n) \cdot \frac{1}{f'(x_n)} \quad (2.11a)$$

$$x_{n+1} = x_n + \Delta x \quad (2.11b)$$

While the Newton Raphson method is very handy as only one initial guess is needed for obtaining the root or zero of a function it has some serious drawbacks that need to be considered [4].

Divergence at turning point. If the initial guess is too close to the actual root of the function, the method may first diverge away from it.

Division by zero. This may prevent the method from converging to the actual zero. Also a compiling error is going to happen as division by zero is not allowed

Oscillations near local maximum and minimum. As the Newton Raphson method is always applied locally, it may converge not to an actual root or zero but to a minimum or maximum of the function. Eventually this will result in no convergence.

Root jumping. If a function has lots of oscillations and therefore multiple zeros, if the initial guess is not good enough, the method may jump and converge to a different zero.

2.4 The Shooting Method – Approach 1

The shooting method is a handy tool for solving one dimensional ordinary differential equations numerically and to obtain necessary eigenvalues. In this thesis we use it to calculate the eigenvalues of the stationary one dimensional Schrödinger equation. These eigenvalues represent the energy values for bound states of a confined particle in a box potential. Equation 2.2 is solved for the three potentials in equations 2.3, 2.4 and 2.5 and their eigenvalues are obtained. Furthermore, one can then calculate the corresponding wave function to each eigenvalue.

In this first approach we combine two numerical calculation methods as used in the Goethe University lecture notes [5]. First a numerical integration method is used to obtain possible solutions for eigenvalues. Saving the right boundary values in a list or a `numpy` array, the positions where the energy axis is crossed marks a possible eigenstate. As a second step these initial guesses are then refined with an algorithm for finding the zeros (or roots) of a function.

For the numerical integration we use the fourth order Runge Kutta method which is described in Section 2.2. For refining our initial results we chose the Newton Raphson algorithm (Section 2.3). The Newton Raphson method used in this thesis was imported from the `scipy.optimize` module. This approach to the shooting method is quite handy as it allows the user to find multiple eigenstates in one go. As stated before in Section 2.3 when the Newton method is employed this approach has one unfortunate drawback. While many eigenvalues may be found rather effortlessly and at once, if the energy mesh is too coarse, or if the eigenstates are very close together, the probability to skip some eigenvalues is quite high. Chapter 4 will discuss this problem in further detail and show an example of an *overshot* solution.

2.5 The Shooting Method – Approach 2

Brooks et al. [2] propose a slightly altered shooting method. This method does not require a root-finding algorithm like the Newton Raphson method. Instead one shoots as often as required to obtain the energy value that fulfills the second boundary condition. For obtaining the wave function and therefore the correct second boundary condition a numerical integration method is needed. The solutions of the one dimensional

Schrödinger equation have a certain number of nodes which correspond directly to their respective excited state, e.g.: the ground state has zero nodes, the first excited state has one, the second excited state has two. Brooks et al. use this to build in a safety precaution that makes it impossible to skip eigenstates. Starting with an energy value that is lower than the ground state energy and one that should be much larger than the highest calculated energy, an interval is obtained in which all possible wave functions have the right number of nodes. Then the energy is refined until the right boundary condition meets the required one. In our case that would be zero.

Finding the interval works as follows. First the Schrödinger equation is solved with the mean value of the two energy values and the nodes are counted. For the top range if the number of nodes is larger than the required number of nodes, the top energy value must be too high. Therefore, the upper energy value is assigned the current mean value of the interval. If the number of counted nodes is less than or equal the required number of nodes, the bottom energy is set to the current mean value of the energies.

To find the numerically exact solution² a similar method is employed. If the value at the second boundary point is less than zero, the bottom energy is set to the mean value for an even number of nodes and the top energy value is updated if the number of nodes is odd. Otherwise if the rightmost boundary value is greater than zero, the top energy is lowered to the mean value for an even number of nodes or the bottom energy is lifted to the mean value for an odd number of nodes. This is repeated until the difference between the top and bottom energy values equals 10^{-12} .

2.6 Related Work

In *How to build an atom* Brooks et al. [2] introduced us to the Schrödinger equation for a single free particle in atomic Rydberg units³. It showed how second order ordinary differential equations are solved numerically. First the equation is split into two coupled first order differential equations. The simplest numerical method for solving one dimensional differential equations is the Euler formula, which is also quite inaccurate. After presenting a predictor corrector method, Brooks et al. settle for the fourth order Runge Kutta method for coupled order systems as only one initial value is needed. They presented FORTRAN programs for solving the Schrödinger equation. Amongst them is a solution for free particles and one for the radial Schrödinger equation of the hydrogen atom. Brooks et al. introduce us to a few simplifications, making it easier to compute a solution. In this thesis we adapted their concept for solving the Schrödinger equation

²In this case it is an error of 10^{-12} as in numerical mathematics no exact solution exists.

³The energy is measured in Rydberg (Ry). One Rydberg equals approximately 13.6eV

in the second approach for the three cases we present.

In 1986 Killingbeck et al. [6] presented a more stable shooting method for the quantum harmonic oscillator when the energy is not held at an eigenvalue. With early approaches the shooting method became unstable at a certain point. They present a shooting method which can be used with many numerical integration methods over a large region for any energy value. Although we do not use any of the specified methods, the concept of the shooting method is introduced.

Binesh et al. [7] presented suitable boundary conditions for solving the Schrödinger equation for three different potentials. They solved these with the fourth order Runge Kutta method. A suggestion for boundary conditions for the radial Schrödinger equation for hydrogen and positronium atoms were given. They obtained their results for the ground state energies and the corresponding radial probabilities from a FORTRAN code. To simplify calculations made with the computer Binesh et al. use atomic Hartree units⁴. As the atomic units are very handy for making the hydrogen Schrödinger equation computable, we make use of them. Additionally we consider not only the ground state of the hydrogen atom, but also calculate the radial wave function of the 2s, the 3s and the 2p orbitals.

Very useful is the *hyperphysics* collection [8], a Web site started in 2005 by physics teachers to connect and share their knowledge. It has extensive material on the Schrödinger equation and the hydrogen concept.

2.7 Python as Programming Language

For numerical mathematics there is usually one programming language that comes to mind: **Fortran**. It is a rather old but very useful programming language for numerical computations. When memory was limited **Fortran** was the tool of choice. However, **Fortran** is quite old and outdated. Code written in **Fortran** is rather hard to read. In scientific computing readability and faster code generation is essential to make progress. Using old programming languages or hardware near programming languages like **C** might slow down the process.

In the lecture *Numerische Methoden der Physik* (Numerical Methods for Physics) [5] the solutions for the infinite potential well and the quantum harmonic oscillator were

⁴In Hartree atomic units the reduced Planck constant, the Bohr radius, the electron mass and the atomic charge are set to 1: $\hbar = a_0 = m_e = e = 1$

achieved using C. C is a very powerful language; its strong point is hardware near programming and therefore it is most suitable for runtime and memory sensitive applications and not necessarily for scientific computing.

Python is the swiss army knife of programming languages. It has packages to do almost everything. It is an intuitive language with clear and easy to read syntax. The basic Python functionality can be expanded with (third party) libraries for very many different uses. The most important ones for scientific computing are the `numpy` -package for n-dimensional array operations and the `scipy` - package containing predefined physical constants and a vast collection of numerical methods. Also the `Matplotlib` package for visualizing results is useful for scientific computing. `Numpy` arrays can be directly turned into graphs or figures, with lots of features to enhance the output plots. It is a high level programming language but with features to increase speed like wrapping C code with e.g. *Cython*. Also Matlab and Fortran wrappers exist if one is in need of functions from those languages. It handles small scripting projects and big development projects equally well. It also supports object oriented programming. Documentation is incorporated into the code itself and exists as command line comments for programming and docstrings for documenting functionality [9]. It is operating system independent and the standard functionality and many additional libraries are free of charge.

Chapter 3

Implementation

In this chapter we discuss three special cases the Schrödinger equation is solved for and why these are important. First is the infinite potential well which, while not really existing, serves as an easy to understand example on how to obtain the eigen-energies of a confined particle. Second comes the quantum harmonic oscillator. Here the particle is trapped in a potential that changes relatively to its position. Most arbitrary potentials can be approximated as harmonic near an equilibrium point, making this one of the most important model systems in quantum mechanics. Third is the radial part of the hydrogen Schrödinger equation, modeling the Coulomb attraction between the positively charged particle at the atom's core with the electron in the orbitals. The solution yields the radial wave functions of where to find the electron in the orbitals. We obtain the results for the first four orbitals: 1s, 2s, 3s and 2p.

Also, when calculating physical problems in a computer some precautions must be taken, e.g. one must be unit consistent. Lastly, we compare our obtained numerical results with their analytical counterpart, making it easy to judge the suitability of the method.

3.1 Infinite Potential Well

A particle, e.g. an electron, is trapped between two regions of infinite potential, thus forming the so called infinite potential well. The wave function outside the well is zero and, since it does not jump at the boundaries, this also true for the boundary values at the walls. In a classical system the particle, like a ball lying on a table, would be allowed to take any energy in the room. In the quantum mechanical system however, Schrödinger's equation allows a particle to exist only in a few quantized energy states. This is true for any bound state where the particle exists in a small confinement. Although it is not a real physically existing system, it is interesting for demonstrating the

properties of particles in bound states, as it can be solved fairly easily. Also it is one of very few quantum mechanical problems with an analytical solution [10].

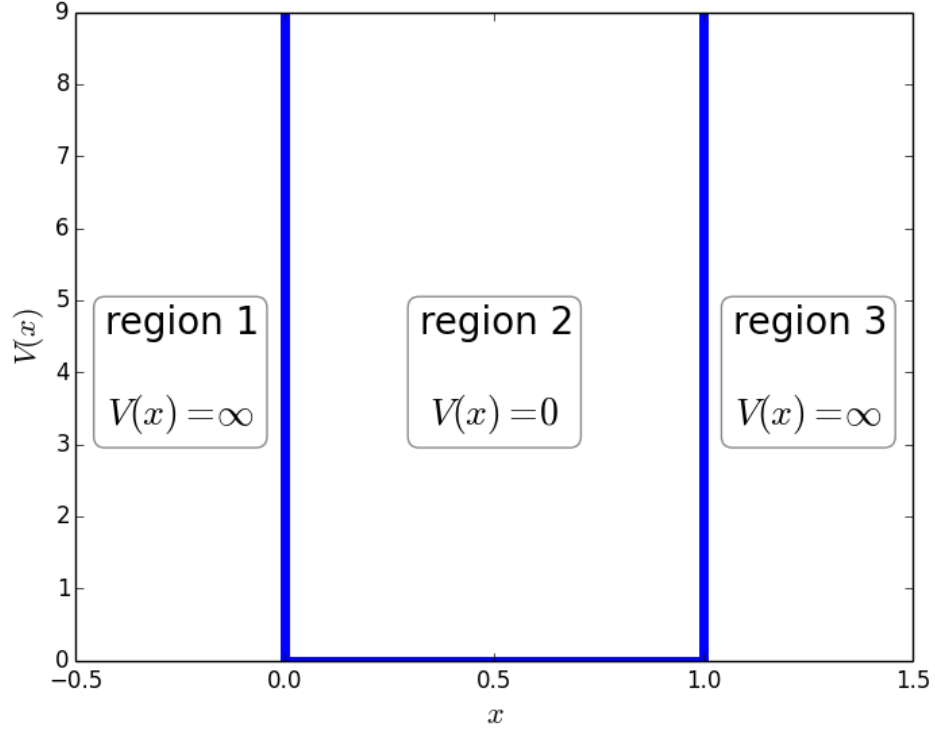


FIGURE 3.1: Infinite Potential Well.

The one dimensional, stationary Schrödinger equation for the infinite potential well reads as

$$\frac{d^2}{dx^2}\psi(x) = \frac{2m_e E}{\hbar^2} \cdot \psi(x) \quad (3.1)$$

with length L , electron mass $m_e = 9,11 \cdot 10^{-31}$ [kg], energy E and the reduced Planck constant $\hbar = \frac{h}{2\pi}$ [kg/s] [5].

As boundary conditions we have

$$\psi(x = 0) = 0, \psi(x = L) = 0. \quad (3.2)$$

The infinite potential well is defined as a three region system (fig. 3.1), where 1 and 3 are regions of infinite potential (equ.: 3.3)

$$V(x = 0) = \infty, V(x = L) = \infty. \quad (3.3)$$

As mentioned above it is necessary to transform the equation into a dimensionless one. For this purpose we take a new constant \hat{x} which we define as $\hat{x} = \frac{x}{L}$ to make the space

coordinate unit independent. With $\hat{E} = \frac{2mEL^2}{\hbar}$ substituted for our energy we obtain equation 3.4.

$$\frac{d^2}{dx}\psi(\hat{x}) = \hat{E}\psi(\hat{x}). \quad (3.4)$$

Having prepared our equation for computing a solution, we now implement the fourth order Runge Kutta formalism as our numerical integration tool¹. The Python implementation for the infinite potential well can be found in Appendix A for approach one and Appendix B for approach two.

We suspect four possible eigenstates for \hat{E} between 0 to 160. We set up the energy mesh with a spacing of $\Delta\hat{E} = 5.0$. In a loop we now solve equation 3.4 with the fourth order Runge Kutta implementation. The output is shown in figure 3.2 together with the analytic solutions for the energy states as red diamonds. The values closest to zero are now ready for refining. This is done as suggested in Chapter 2 with the Newton Raphson method. The algorithm terminates as soon as an error $\Delta\hat{E} = 10^{-12}$ is reached.

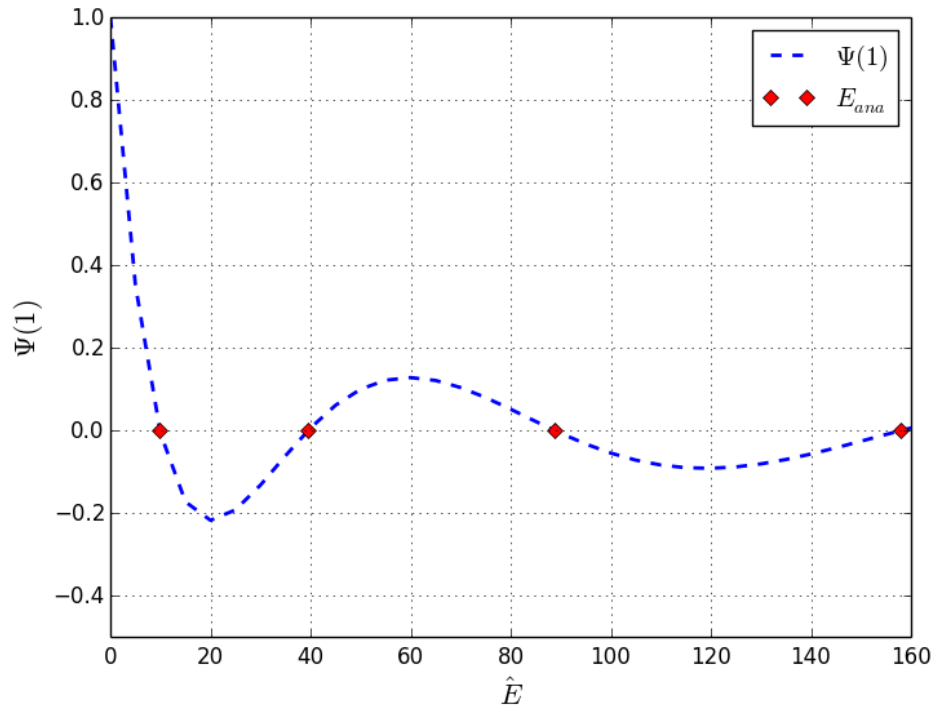


FIGURE 3.2: Screening for Possible Energy Values for the Infinte Potential Well.

Now that we computed our numerical solution for the energy eigenstates and their matching wave function we want to judge the suitability of the method. Thus, we compare the wave function we obtained numerically with the analytical equivalent. Figure

¹Note that the Sci-Py stack offers numerical integration methods in the integrate package.

3.3 to 3.7 depict the numerical solution as the broken line and the analytical solution as the solid line. In this case the energy mesh was fine enough to produce the eigenvalues we were looking for.

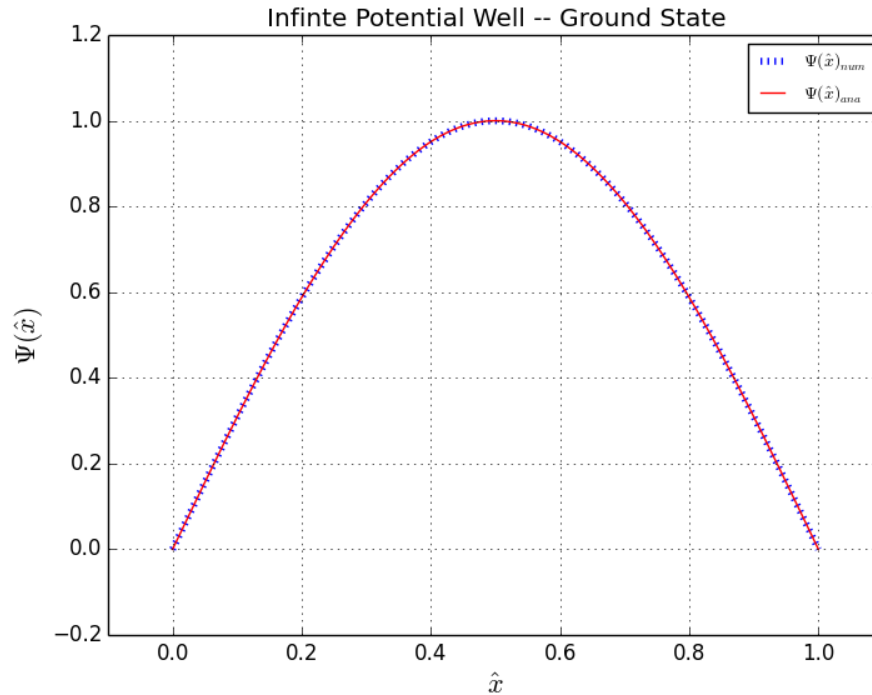


FIGURE 3.3: Wave Function Comparison for Ground State of the Infinite Potential Well.

The second approach results in a console output which contains the quantum state, the obtained energy value and produces plots for comparing the analytical wave function with the numerical wave function. The script for the second shooting method approach solves both the infinite potential well and the quantum harmonic oscillator problem in one go for a predefined number of nodes.

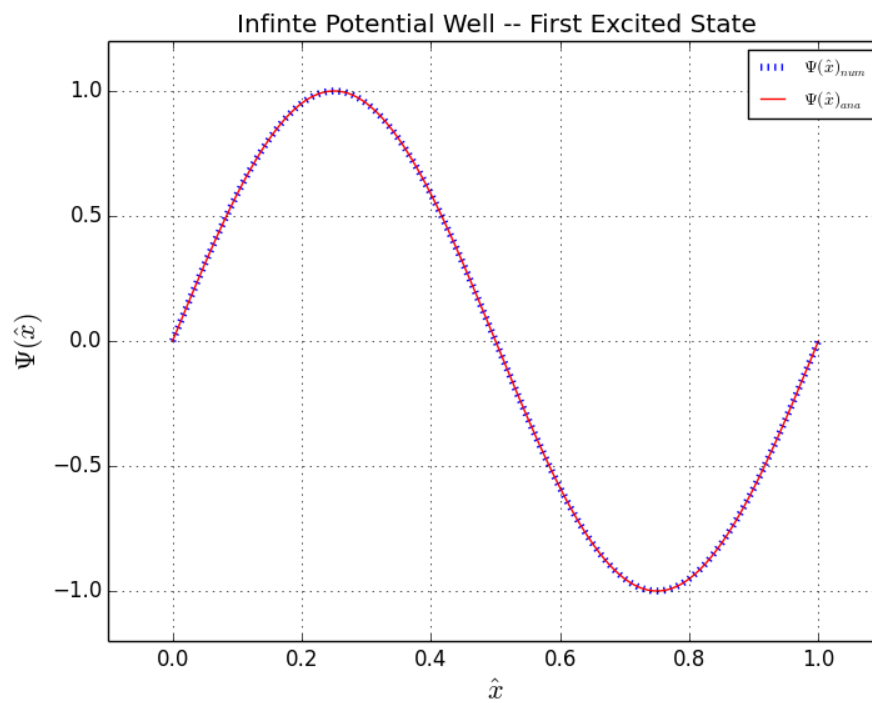


FIGURE 3.4: Wave Function Comparison for First Excited State of the Infinite Potential Well.

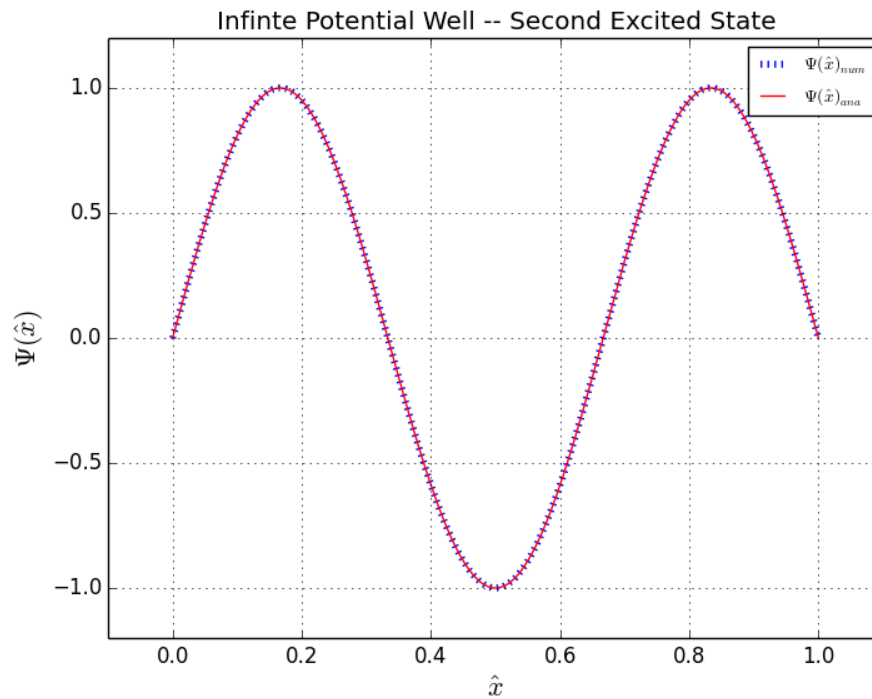


FIGURE 3.5: Wave Function Comparison for Second Excited State of the Infinite Potential Well.

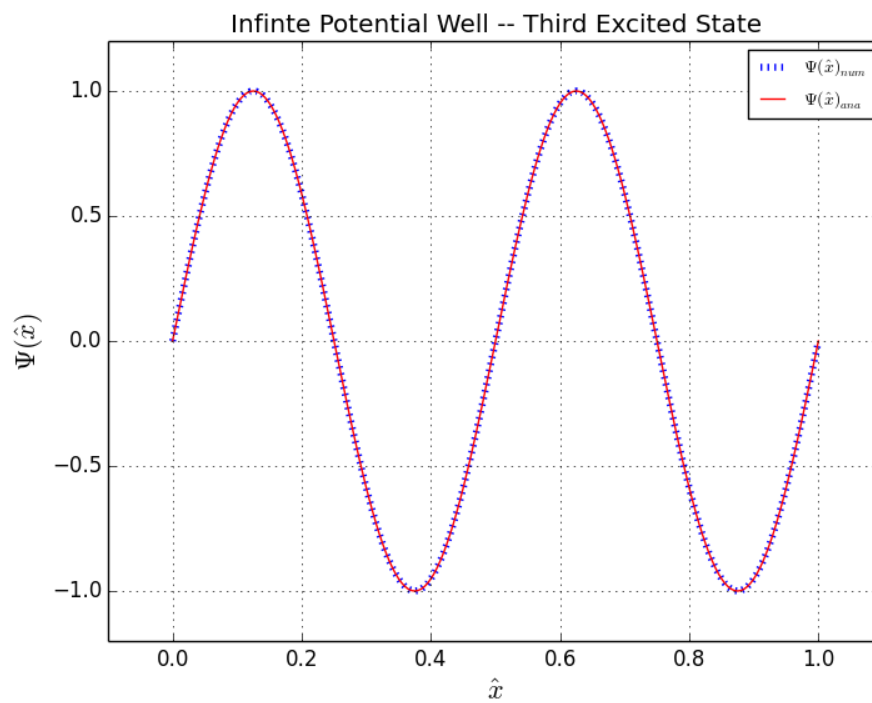


FIGURE 3.6: Wave Function Comparison for Third Excited State of the Infinite Potential Well.

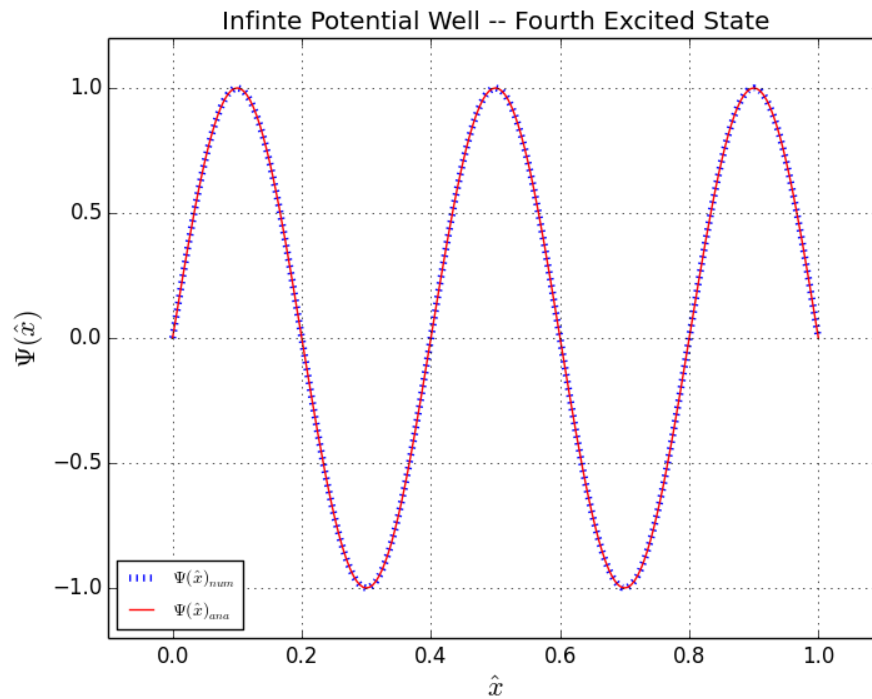
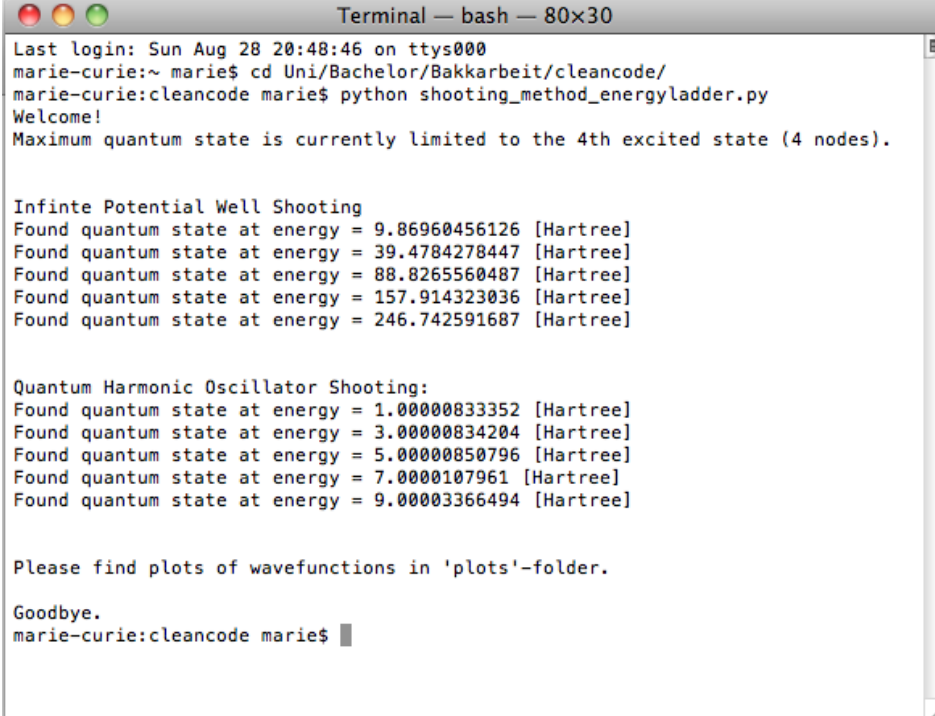


FIGURE 3.7: Wave Function Comparison for Fourth Excited State of the Infinite Potential Well.

A terminal window titled "Terminal — bash — 80x30" with standard macOS window controls (red, yellow, green buttons). The terminal shows the following text:

```
Last login: Sun Aug 28 20:48:46 on ttys000
marie-curie:~ marie$ cd Uni/Bachelor/Bakkarbeit/cleancode/
marie-curie:cleancode marie$ python shooting_method_energyladder.py
Welcome!
Maximum quantum state is currently limited to the 4th excited state (4 nodes).

Infinte Potential Well Shooting
Found quantum state at energy = 9.86960456126 [Hartree]
Found quantum state at energy = 39.4784278447 [Hartree]
Found quantum state at energy = 88.8265560487 [Hartree]
Found quantum state at energy = 157.914323036 [Hartree]
Found quantum state at energy = 246.742591687 [Hartree]

Quantum Harmonic Oscillator Shooting:
Found quantum state at energy = 1.00000833352 [Hartree]
Found quantum state at energy = 3.00000834204 [Hartree]
Found quantum state at energy = 5.00000850796 [Hartree]
Found quantum state at energy = 7.0000107961 [Hartree]
Found quantum state at energy = 9.00003366494 [Hartree]

Please find plots of wavefunctions in 'plots'-folder.

Goodbye.
marie-curie:cleancode marie$
```

FIGURE 3.8: Console Output of the Program for Approach 2.

3.2 Quantum Harmonic Oscillator

In classical physics the harmonic oscillator refers to a particle in equilibrium that, when displaced by some force, is pulled back by a restoring force proportional to the displacement. An example would be a mass connected with a spring to a wall. From classical mechanics we know this particle now oscillates around its equilibrium point with frequency $\nu = \frac{\omega_0}{2\pi}$. Its energy or amplitude can be any arbitrary value. This is not the case for the quantum equivalent of the harmonic oscillator. We want the wave function $\psi(x)$ to go to zero for large distances otherwise normalizing the wave function would not be possible. Therefore, this allows only for some eigenstates with quantized energies $E = \hbar\omega_0(n + \frac{1}{2})$ where n is an integral number. Also the ground state energy is not zero but $\frac{1}{2}\hbar\omega_0$ above the potential's minimum [10].

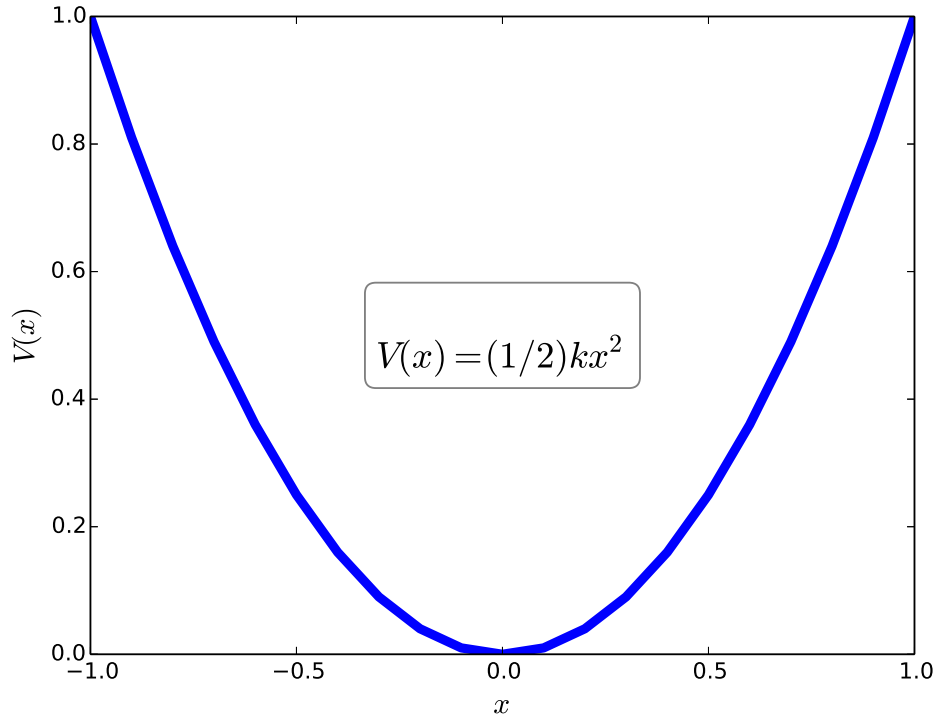


FIGURE 3.9: Quantum Harmonic Oscillator Potential.

The potential well for the quantum harmonic oscillator is shown in Figure 3.9 with its potential defined in equation 3.5, where the spring constant k is substituted with the angular frequency $\omega = \sqrt{\frac{k}{m}}$.

$$V(x) = \frac{1}{2}kx^2 = \frac{1}{2}m(\omega x)^2 \quad (3.5)$$

The Schrödinger equation for the one dimensional quantum harmonic oscillator is (equ. 3.6)

$$\frac{-\hbar}{2m} \frac{d^2}{dx^2} \psi(x) + \frac{m\omega^2}{2} x^2 \psi(x) = E\psi(x), \quad (3.6)$$

with the boundary conditions (equ 3.7)

$$\psi(x \rightarrow -\infty) = 0, \psi(x \rightarrow \infty) = 0. \quad (3.7)$$

When solving equation 3.6 analytically we obtain eigenvalues and the corresponding wave functions at energies shown in equation 3.8.

$$E = \hbar\omega_0(n + \frac{1}{2}), n = 0, 1, 2 \dots \quad (3.8)$$

To be able to solve equation 3.6 we first transform it into a dimensionless one by forming a length from the reduced Planck constant \hbar , the electron mass m_e and the frequency ω :

$$a = \sqrt{\frac{\hbar}{m_e\omega}} \quad (3.9)$$

Dividing x by a yields the normalized substitution variable $y = \frac{x}{a} = \frac{x}{\sqrt{\frac{\hbar}{m\omega}}}$ which we plug into equation 3.6. Also, to be unit consistent we substitute $\hat{E} = \frac{2E}{\hbar\omega}$.

$$\frac{d^2}{dy^2} \Psi(y) + y^2 \Psi(y) = \hat{E} \Psi(y) \quad (3.10)$$

To solve equation 3.10 with the Runge Kutta Method, we first split the second order differential equation into a first order coupled system by setting the first derivative of ψ to a new variable ϕ . and splitting equation 3.10 into two first order differential equations.

$$\Psi'(y) = \Phi \quad (3.11)$$

$$\Phi'(x) = (\hat{E} - y^2) \Psi \quad (3.12)$$

This leaves us to figure out how to implement the boundary conditions since infinity poses an implementation challenge in most computer programs. Since we cannot simply tell our program to set the wave function at point infinity to zero, we will start with a length deep in the forbidden zone where the energy E is a lot smaller than the value of the potential: $E \ll V(L)$.

Now we choose some arbitrary boundary conditions for the wave functions. This raises

the issue of parity². For the python program odd parity wave functions were chosen, with the corresponding boundary conditions (equ.: 3.13).

$$\Psi(y = \frac{L}{a}) = 1, \Phi(y = \frac{L}{a}) = 0 \quad (3.13)$$

With equations 3.10 and 3.13 implemented in a Python script we plug in numbers for \hat{E} and have a look at the second boundary $\Psi(y = 1)$, where our wave function should go to zero. In this case the mesh spacing for \hat{E} is $\Delta\hat{E} = 1.0$. From the first shooting method approach we obtain figure 3.10 depicting values for $\Psi(x = 1)$ over values for \hat{E} ranging from 0 to 6 and the corresponding expected energy values.

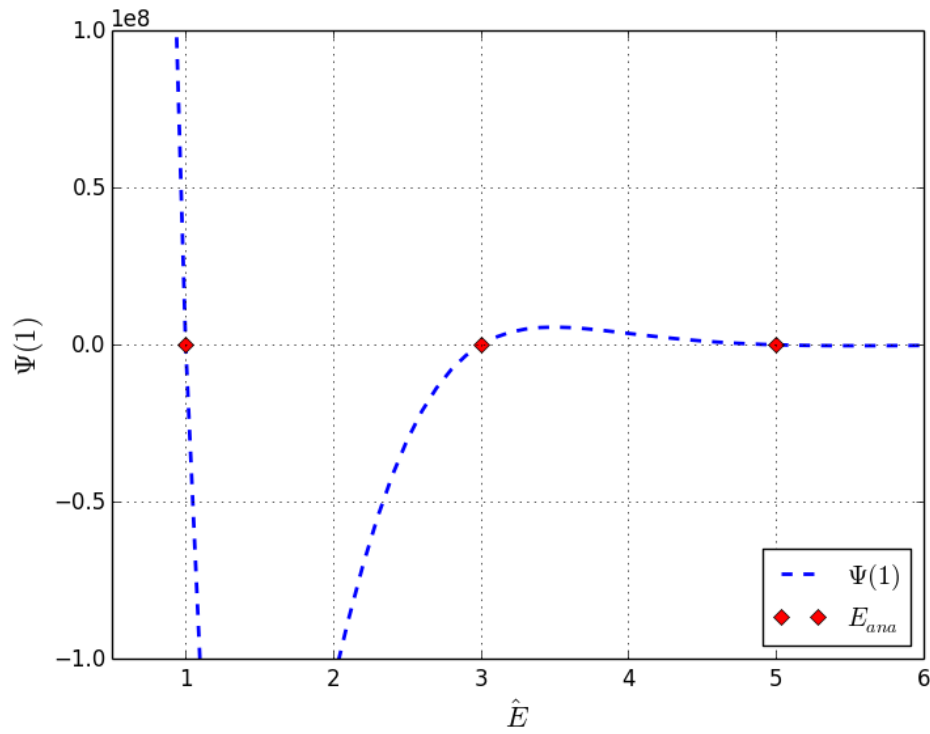


FIGURE 3.10: Screening of Input Values for the Quantum Harmonic Oscillator with Analytical Solution for Comparison.

Again much like in section 3.1 those energy values were chosen for which the wave function went closest to zero at the second boundary. They were then refined to an error of $\Delta E = 10^{-6}$ with Newton Raphson's method.

²When transforming all coordinates to their inverse, all physical properties have to stay the same. The wave function has two possibilities for its behaviour: $\psi(r) \rightarrow -\psi(-r)$ or $\psi(r) = \psi(-r)$. The first case is called odd parity, the second even parity. Since the occupation density is the square of the wave function, both cases are valid solutions.

As we did in the first section, we compared the numerically calculated wave functions with the analytical equivalents. Again the method works, as the numerical solution pictured in figures 3.11 to 3.15 as the broken line is aligned with the solid line, which represents the analytical solution. The wave functions for the first three energy eigenstates are pictured with their analytical analogue. The energy spacing was fine enough to produce the expected solutions and no overshooting occurred.

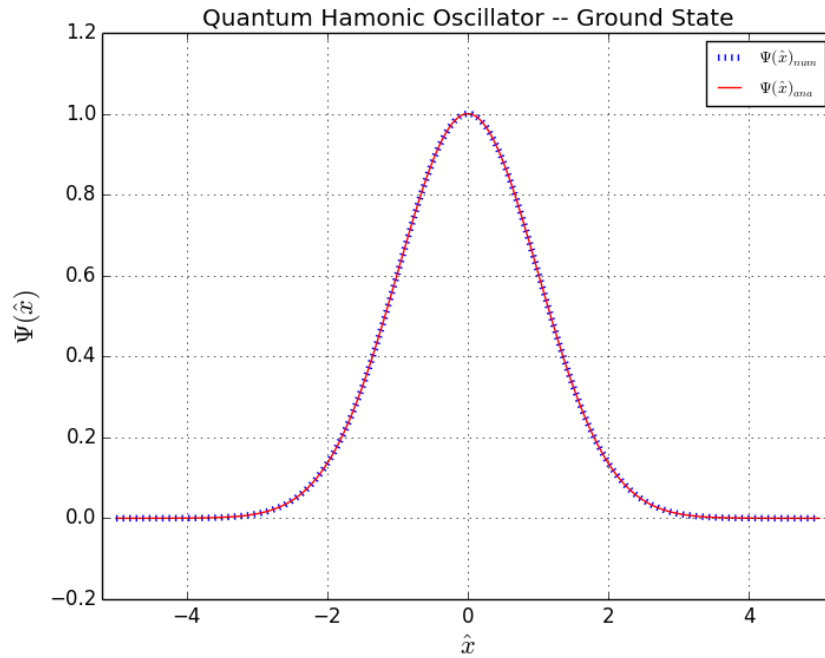


FIGURE 3.11: Wave Function Comparison for the Ground State of the Quantum Harmonic Oscillator.

The second approach's result can be seen in Section 3.1 in figure 3.8 as both problems have been solved in one go of the script.

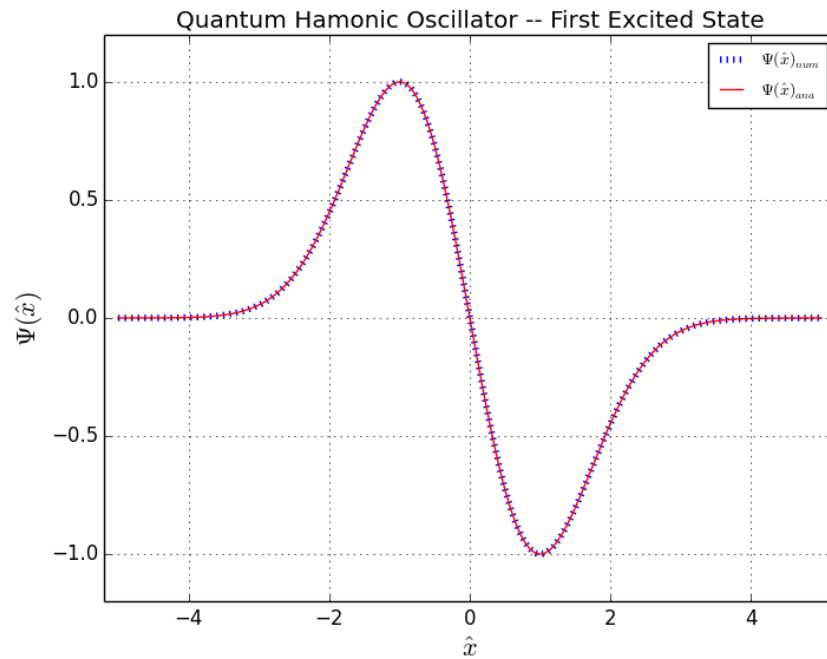


FIGURE 3.12: Wave Function Comparison for the First Excited State of the Quantum Harmonic Oscillator.

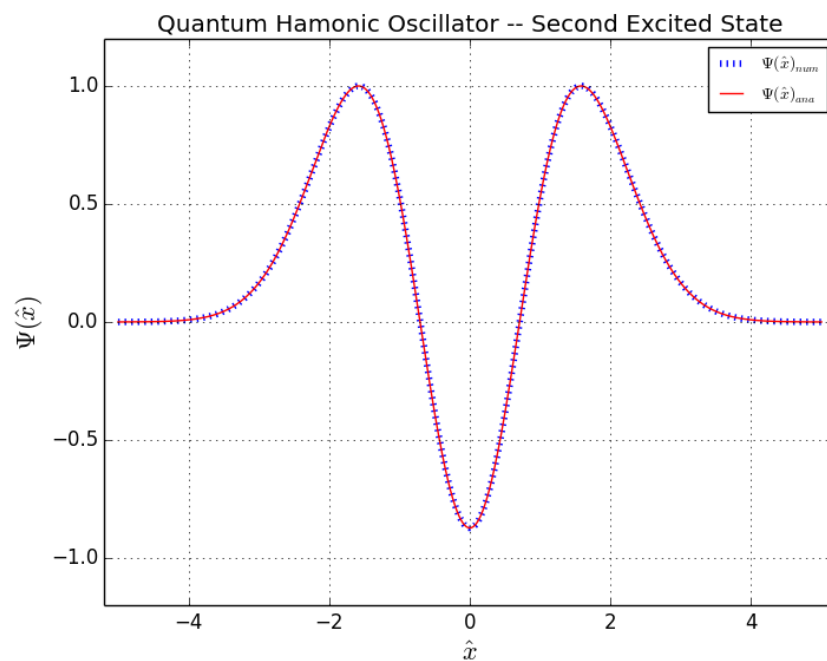


FIGURE 3.13: Wave Function Comparison for the Second Excited State of the Quantum Harmonic Oscillator.

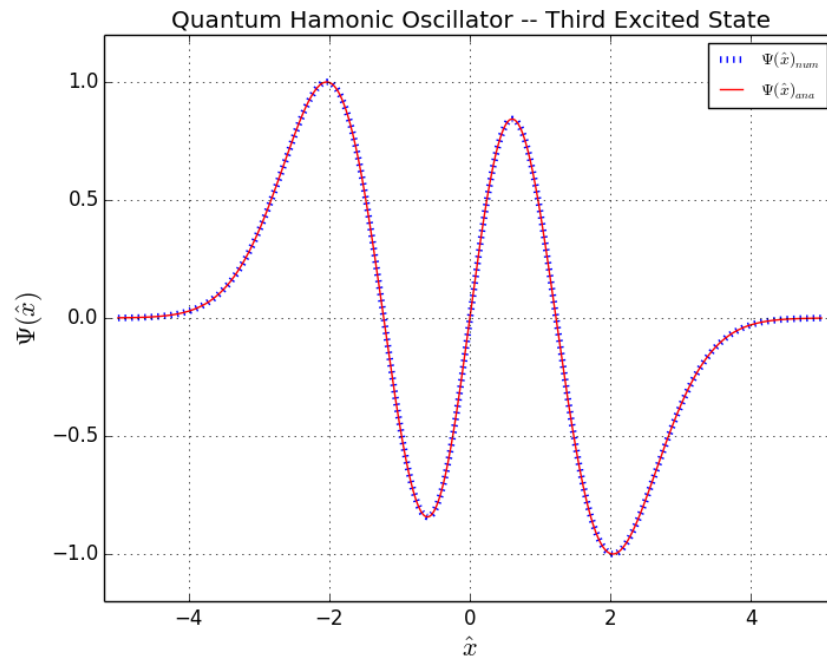


FIGURE 3.14: Wave Function Comparison for the Third Excited State of the Quantum Harmonic Oscillator.

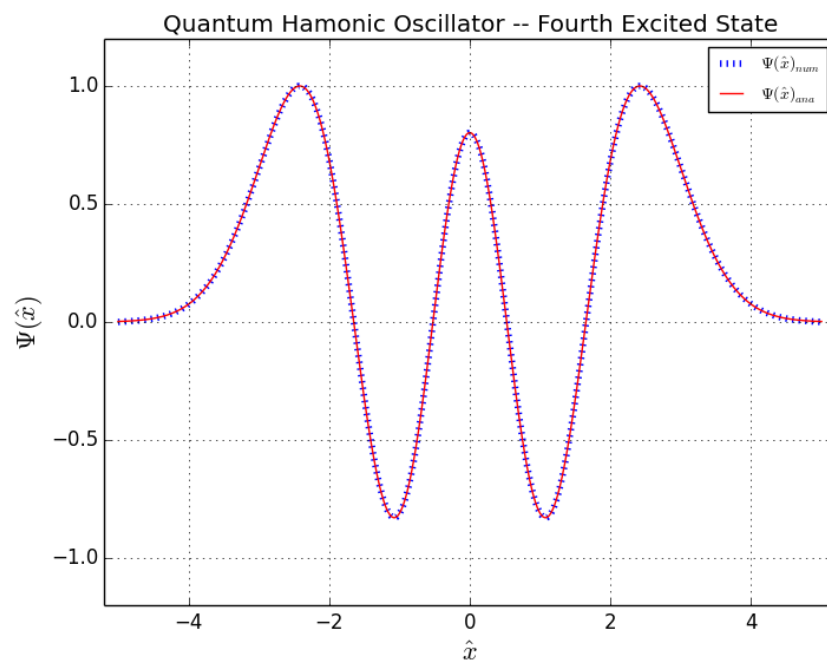


FIGURE 3.15: Wave Function Comparison for the Fourth Excited State of the Quantum Harmonic Oscillator.

3.3 Radial Hydrogen Schrödinger Equation

Hydrogen is the smallest and lightest element known, with a positive charged proton in its core and a negatively charged electron orbiting around it. Its movement is given by the Schrödinger equation (equ.: 3.14) when taking the electrostatic potential (equ.: 3.15) into account, describing the attraction between the proton and the electron [10]. The potential for s orbitals, where the orbital quantum number L^3 equals zero, is plotted in figure 3.16.

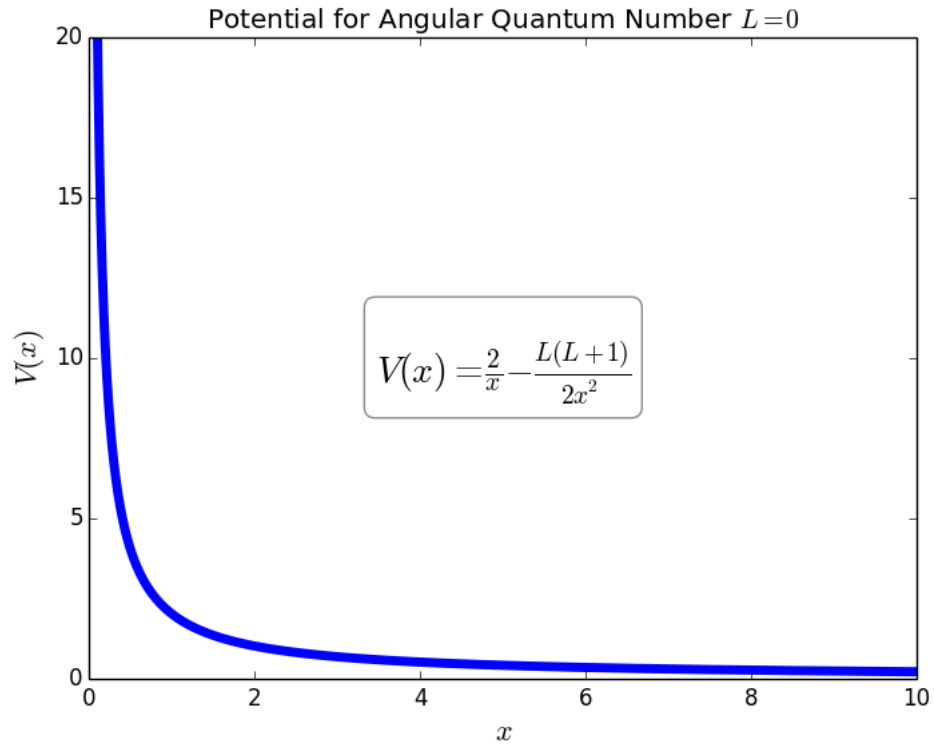


FIGURE 3.16: Hydrogen Atom Potential.

In this case the Schrödinger equation is a three dimensional second order differential equation and it is most often seen in polar coordinates (equation 3.14). With the potential (equ.: 3.15) only depending on the radial coordinate and the system being rotationally invariant⁴, equation 3.14 can be split into three one dimensional second order differential equations 3.16.

$$\begin{aligned} & -\frac{\hbar^2}{2\mu} \frac{1}{r^2 \sin \theta} \left[\sin \theta \frac{\partial}{\partial r} \left(r^2 \frac{\partial \Psi}{\partial r} \right) + \frac{\partial}{\partial \theta} \left(\sin \theta \frac{\partial \Psi}{\partial \theta} \right) + \frac{1}{\sin \theta} \frac{\partial^2 \Psi}{\partial \phi^2} \right] + \\ & + V(r) \Psi(r, \theta, \phi) = E \Psi(r, \theta, \phi) \end{aligned} \quad (3.14)$$

³The orbital quantum number is usually a lowercase l. For reading and coding purposes it is here defined as an uppercase L.

⁴Most physical systems are rotational invariant, their potential only depending on the distance between particles and not their direction [7].

$$V(r) = \frac{e^2}{4\pi\epsilon_0 r} \quad (3.15)$$

$$\Psi(r, \theta, \phi) = R(r)P(\theta)F(\phi) \quad (3.16)$$

When substituting equation 3.16 into equation 3.14, the partial derivatives can be expressed as ordinary derivatives. The equation can then be separated into a radial part (equ.: 3.17) and an angular part with a colatitude equation and an azimuthal equation⁵. The radial part collects all terms that depend on the radial coordinate r and sets them equal to a constant which contains the orbital quantum number L . For the 1s, 2s and 3s orbitals the orbital angular momentum is zero, while for p orbitals L can be $n - 1$, with n as the principal quantum number taking values of $n = 1, 2, 3, \dots$ [8].

$$\frac{1}{R} \left[r^2 \frac{dR}{dr} \right] + \frac{2\mu}{\hbar^2} (Er^2 + \frac{1}{4\pi\epsilon_0} e^2 r) = L(L+1) \quad (3.17)$$

Analytical solutions to equation 3.17 are in the form of equation 3.18,

$$R_{n,l} = r^l L_{n,l} e^{-\frac{r}{na_0}}, \quad (3.18)$$

with $L_{n,l}$ being corresponding Laguerre functions.

Equation 3.17 might pose some problems for a computer program, i.e. very small numbers. To simplify our solution we use Hartree atomic units where the reduced Planck constant, the Bohr radius, the electron mass and the atomic charge equal unity ($\hbar = a_0 = m_e = e = 1$). The energy of a system in atomic units is defined as the Hartree energy. With another handy substitution of $R(r)$ with $\frac{U(r)}{r}$ this leads to equation 3.19.

$$\frac{d^2}{dx} U(r) + (2E + \frac{2}{r} - \frac{L(L+1)}{r^2}) U(r) = 0. \quad (3.19)$$

Again, as equation 3.19 is a second order differential equation we first have to split it into a first order coupled system of two one dimensional differential equations.

$$V(r) = \frac{d}{dr} U(r) \quad (3.20)$$

$$\frac{d}{dr} V(r) = - (2E + \frac{2}{r} - \frac{L(L+1)}{r^2}) U(r) \quad (3.21)$$

Furthermore we define our boundary conditions as follows (equ.: 3.22)

$$U(r=0) = 0, V(r=0) = 1. \quad (3.22)$$

⁵As we will not be using these, they have not been included in this thesis but can be found here: [8].

With application of the first shooting method approach we find the first three quantized energy states which are depicted in figure 3.17, with the red dots representing the analytical values of the energy eigenstates for the 1s, 2s and 3s orbitals.

The eigenvalues for the energies of bound states are the following for above's equation

$$E = -2\frac{1}{n^2}, \quad (3.23)$$

with the principal quantum number n as an integral number $n = 1, 2, 3 \dots$

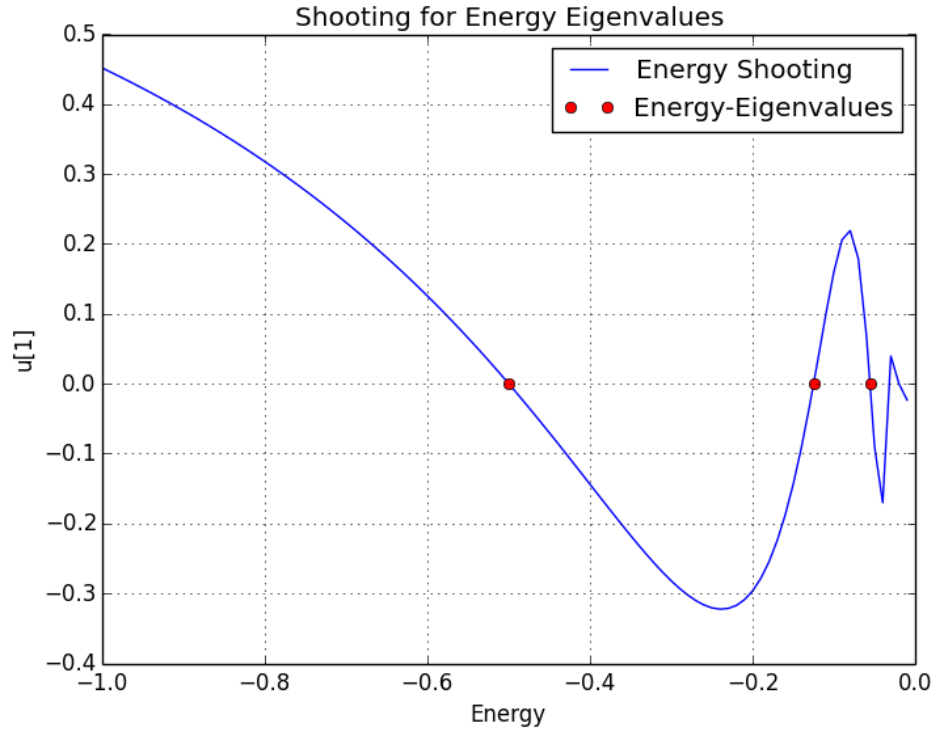


FIGURE 3.17: Screening of Input Values for the Radial Hydrogen Schrödinger Equation with Analytical Solution for Comparison.

Note that the energy eigenvalues for higher energies are spaced close together. With the first approach presented here it is very likely to miss a few states. Also the Newton Raphson method will very likely fail. As it requires only one initial guess for finding local zeros actually optimizing to a neighbouring zero is a known drawback of this method. A method like bisection, which requires an interval for finding a zero is thoroughly recommended.

Again, once the energy eigenvalues are computed to a more or less exact number the respective wave functions are calculated and compared to their analytical equivalents. These are depicted in figures 3.19 to 3.21

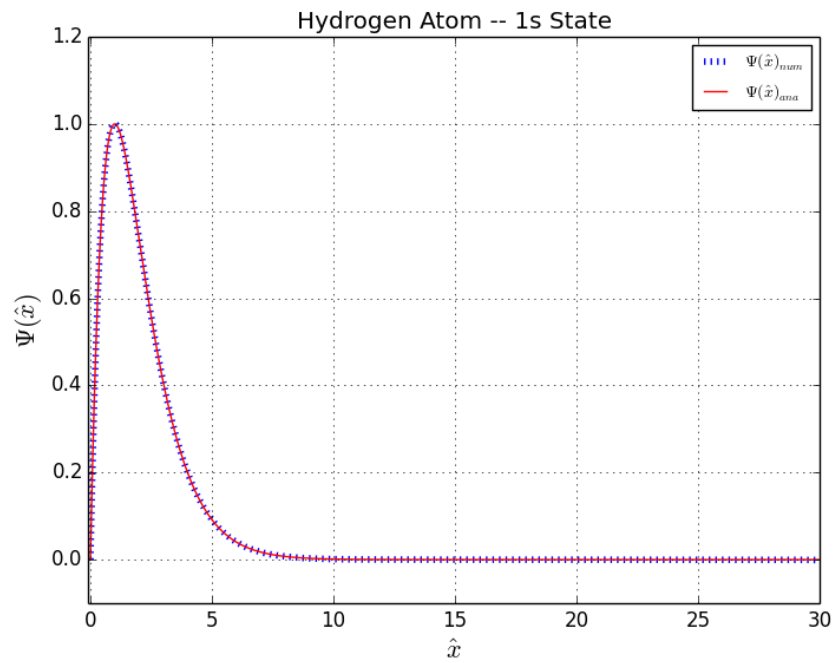


FIGURE 3.18: Hydrogen Wave Function for 1s Orbital.

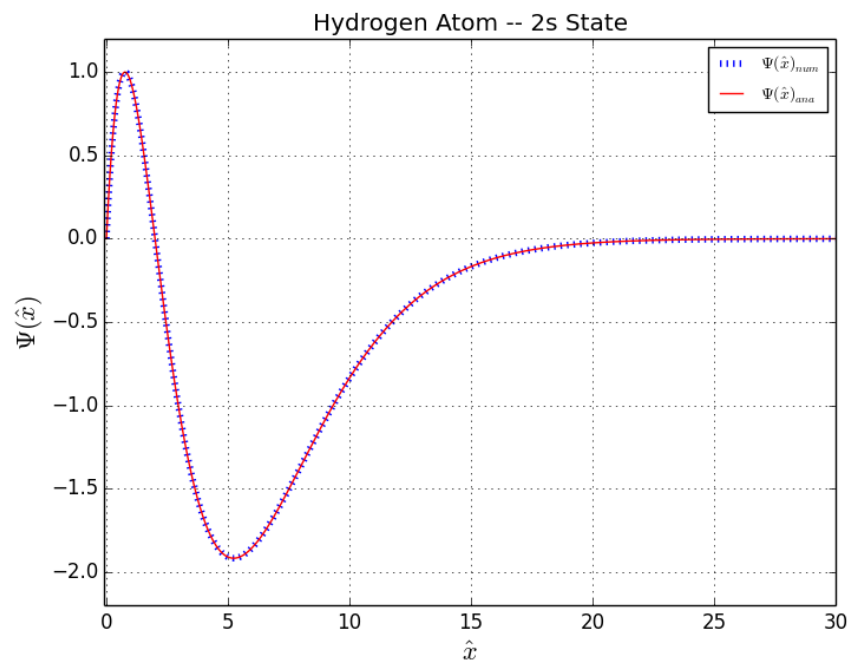


FIGURE 3.19: Hydrogen Wave Function for 2s Orbital.

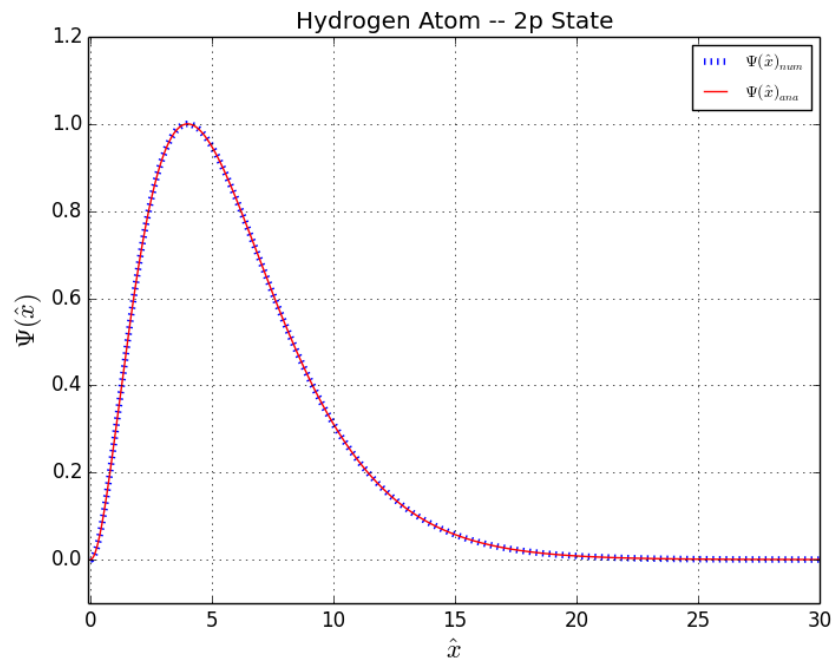


FIGURE 3.20: Hydrogen Wave Function for 2p Orbital.

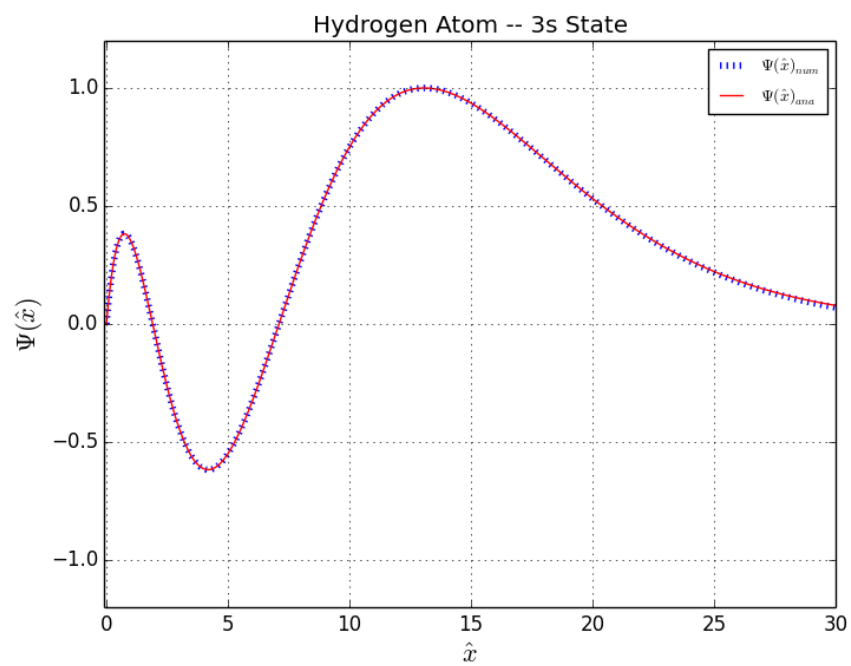


FIGURE 3.21: Hydrogen Wave Function for 3s Orbital.

Chapter 4

Lessons Learned

Coming from a C background, adapting to Python was easier than expected, although the first few code samples looked more like C code translated line for line into Python. After revising the first source code, it was still very much C code and did not incorporate the key strengths of scientific computing with Numpy and Scipy. Employing a tailored Runge Kutta method to each example is not at all compatible with an elegant Python approach. In the final script presented in Appendix A and B only one function for the fourth order Runge Kutta method exists which can be applied with any function and every potential, as long as it is part of the defined function [11]. Also the process of finding the zero crossings is automated. Step for step the code evolved into proper Python code, replacing bulky C concepts with Python's beautiful syntax, always making sure to follow the style guidelines defined in pep8 [12] and the docstring conventions in pep257 [9].

Plugging equations into a computer is not always a straight forward process. It requires some preparation to make sure of unit consistency, to use length scaling and normalization criteria. The normalization is a way of defining the probability of a particle to exist somewhere in space (equ.:4.1).

$$\int_{-\infty}^{\infty} |\Psi(x)|^2 dx = 1. \quad (4.1)$$

While the infinite potential well was by far the easiest case to adapt, the concepts mentioned above still had to be incorporated. It was a good and simple first example to try the shooting method. The quantum harmonic oscillator also was quite straight forward to prepare and solve. Again a length scaling was necessary, as the solution has an exponential part that may obscure the results rather quickly. Also the boundary conditions for this case were not suitable for numerical methods as infinity always poses problems for boundary conditions when using numerical methods. To avoid this, the

boundary values for our spatial coordinate were taken deep in the forbidden zone, where the energy is very small compared to the value of the potential.

By far the most challenging was the process of solving the radial Schrödinger equation of the hydrogen atom. The equation at first is three dimensional and had to be split into a part of radial dependence and one for angular dependence.

Trying out the first approach of the shooting method was educational. It seemed like a very handy method as one can find several eigenvalues at once. However, once the energy mesh chosen for computing a solution is too coarse, all of the states that might have been in between two mesh points are lost. One might obtain a wave function for a much higher energy state, instead of the expected one. In figure 4.1 this problem is shown. The expected state was the first excited, instead one obtained the wavefunction for the fifth excited state. While this might not be too much of a problem when the expected solution is known it is devastating for solving functions where the solution is not. Also for very shallow quantum wells, the eigenvalues will be very close together. Applying the Newton method which requires only one initial rough estimate for the zero can result in finding a neighbouring one. Again leading to a wave function we do not expect. To eliminate this drawback of the Newton Raphson algorithm, a method like bisection is recommended. The `scipy.optimize` module has numerous methods available for one dimensional functions and even for multi-dimensional functions. Therefore, the second approach employed here will be the safer bet. Although the definite node quantity has to be known in advance, we eliminate both weak points from approach one. The nodes are being watched and also no root finding algorithm is required for obtaining the final solution.

Solving the Schrödinger equation for the hydrogen atom was by far the most complicated, as the solution goes up exponentially for even the smallest deviations from the numerically exact solution. Also to increase computation speed and accuracy instead of using the fourth order Runge Kutta method, we used the numerical integration method `odeint` from the `scipy.integrate` module. The documentation of this method can be found here [13].

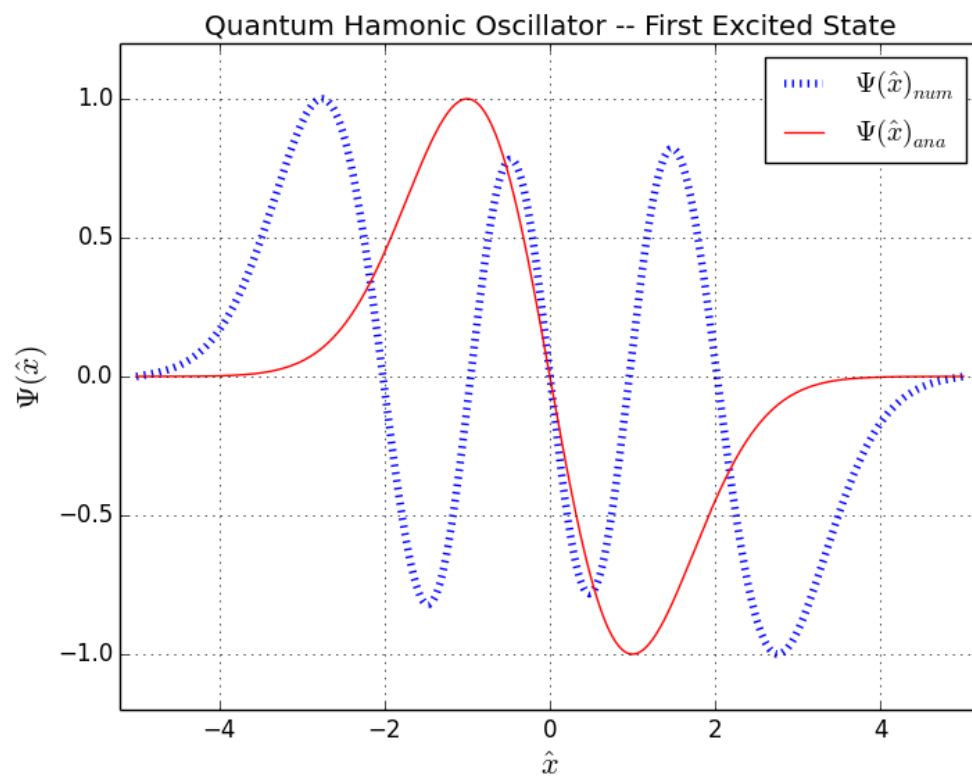


FIGURE 4.1: Overshooting occurring due to coarse energy mesh.

Chapter 5

Summary and Outlook

In this section we provide the summary and discuss the outlook for potential future work.

5.1 Summary

The Schrödinger equation is the most important problem to be solved in quantum physics with very few analytically solvable cases. This work's aim was to present a dependable numerical tool for solving the stationary one dimensional Schrödinger equation, obtaining both the eigenstates of the system's energy and the corresponding wave functions. The shooting method was demonstrated for three quantum mechanical problems: the infinite potential well, the quantum harmonic oscillator and the radial part of hydrogen's Schrödinger equation. As these cases all have an analytical solution we used it to compare our obtained results to. For applying the shooting method two numerical formalisms are required: a numerical integration method and some approximation algorithm. We chose the fourth order Runge Kutta method for the former and the Newton Raphson method for the latter.

In Chapter 3 we presented our computed energy solutions together with a comparison to their analytical counterparts. We also plotted the wave functions for each of the three cases for three to four states. The suitability of the shooting method is shown, as the results match nicely.

Chapter 4 briefly outlines the lessons learned while working on this thesis.

5.2 Outlook

Even if this thesis is self-contained we can think of further activities related to this work.

Other one dimensional differential equations

We have shown that the shooting method can be applied to the Schrödinger equation. However, there are other one dimensional differential equations — such as the Van der Pol oscillator — where the shooting method is interesting to study in order to solve it numerically.

Open Source Software

It is good practice in the scientific community to share ones code as Open Source Software. We support this and want to contribute our source code in an appropriate Python package that is related to solving numerical problems.

Web Application

It could be of interest for other scientists to study the shooting method with a configurable graphical user interface. Here, a browser-based app seems to be a right choice. This can be combined with the Open Source package mentioned above.

Appendix A

Shooting Method – Approach 1

```
"""
Script for solving the one dimensional Schroedinger equation numerically.

Numerical integration method used is the fourth order Runge Kutta.
Counts the nodes of the wave function and determines the harmonic.
Then refines the solution until proper energy is found.

Potentials:
Infinite Potential Well
 $V(x_{-}<0) = \text{inf}$ ,  $V(x_{-}=0,1) = 0$ ,  $V(x_{-}>1) = \text{inf}$ 

Harmonic Oscillator:
 $V(x_{-}) = x_{-}^2$ 

Radial Hydrogen Atom Coulomb attraction:
 $V(r) = 2/r - (L(L+1))/(r^2)$  a.u.
"""

import numpy as np
import scipy
from scipy import integrate
from scipy.optimize import newton
import matplotlib.pyplot as plt

def Schroed(y, r, V, E):
    """Return one dim Schroedinger equation with Potential V."""
    psi, phi = y
    dphidx = [phi, (V-E)*psi]
    return np.asarray(dphidx)

def rk4(f, psi0, x, V, E):
    """Fourth-order Runge-Kutta method to solve  $\phi'=f(\psi,x)$  with  $\psi(x[0])=\psi_0$ .

    Integrates function f with initial values psi0 and potential V numerically.
    Output is possible multidimensional (in psi) array with len(x).
    """
```



```

n = len(x)
psi = np.array([psi0]*n)
for i in xrange(n - 1):
    h = x[i+1] - x[i]
    k1 = h*f(psi[i], x[i], V[i], E)
    k2 = h*f(psi[i] + 0.5*k1, x[i] + 0.5*h, V[i], E)
    k3 = h*f(psi[i] + 0.5*k2, x[i] + 0.5*h, V[i], E)
    k4 = h*f(psi[i] + k3, x[i+1], V[i], E)
    psi[i+1] = psi[i] + (k1 + 2.0*(k2 + k3) + k4) / 6.0
return psi

def shoot(func, psi0, x, V, E_arr):
    """Shooting method: find zeroes of function func for energies in E_arr.

    func: Schroedinger equation to solve.
    psi0: initial conditions on left side, can be array.
    V : Potential to solve SE with.
    E_arr: array of energy values: find possible zeroes.
    """
    psi_rightb = []
    for EN in E_arr:
        psi = rk4(func, psi0, x, V, EN)
        psi_rightb.append(psi[len(psi)-1][0])
    return np.asarray(psi_rightb)

def shoot1(E, func, psi0, x, V):
    """Helper function for optimizing results."""
    psi = rk4(func, psi0, x, V, E)
    return psi[len(psi)-1][0]

def shoot_ode(E, psi_init, x, L):
    """Helper function for optimizing results."""
    sol = integrate.odeint(Schrod_deriv, psi_init, x, args=(L,E))
    return sol[len(sol)-1][0]

def findZeros(rightbound_vals):
    """Find zero crossing due to sign change in rightbound_vals array.

    Return array with array indices before sign change occurs.
    """
    return np.where(np.diff(np.signbit(rightbound_vals)))[0]

def optimizeEnergy(func, psi0, x, V, E_arr):
    """Optimize energy value for function using brentq."""
    shoot_try = shoot(func, psi0, x, V, E_arr)
    crossings = findZeros(shoot_try)
    energy_list = []
    for cross in crossings:
        energy_list.append(newton(shoot1, E_arr[cross],

```

```

        args=(func, psi0, x, V))
    return np.asarray(energy_list)

def normalize(output_wavefunc):
    """A function to roughly normalize the wave function."""
    normal = max(output_wavefunc)
    return output_wavefunc*(1/normal)

def shoot_potwell(psi_init, h_):
    """Shooting method for infinte potential well.

    500 mesh points.
    Returns the numerical and analytical solution as arrays.
    """
    x_arr_ipw = np.arange(0.0, 1.0+h_, h_)
    V_ipw = np.zeros(len(x_arr_ipw))
    E_arr = np.arange(1.0, 100.0, 5.0)
    eigE = optimizeEnergy(Schroed, psi_init, x_arr_ipw, V_ipw, E_arr)
    ipw_out_list = []
    for EE in eigE:
        out = rk4(Schroed, psi_init, x_arr_ipw, V_ipw, EE)
        ipw_out_list.append(normalize(out[:, 0]))
    out_arr = np.asarray(ipw_out_list)
    # analytical solution for IPW
    k = np.arange(1.0, 4.0, 1.0)
    ipw_sol_ana = []
    for kk in k:
        ipw_sol_ana.append(np.sin(kk*np.pi*x_arr_ipw))
    ipw_sol_ana_arr = np.asarray(ipw_sol_ana)
    return x_arr_ipw, out_arr, ipw_sol_ana_arr

def shoot_QuantumHarmonicOscillator(psi_init, h_):
    """Shooting method for quantum harmonic oscillator.

    500 mesh points.
    Returns the numerical and analytical solution as arrays.
    """
    x_arr_qho = np.arange(-5.0, 5.0+h_, h_)
    V_qho = x_arr_qho**2
    E_arr = np.arange(1.0, 15.0, 1.0)
    eigEn = optimizeEnergy(Schroed, psi_init, x_arr_qho, V_qho, E_arr)
    qho_out_list = []
    for EN in eigEn:
        out = rk4(Schroed, psi_init, x_arr_qho, V_qho, EN)
        qho_out_list.append(normalize(out[:, 0]))
    qho_out_arr = np.asarray(qho_out_list)
    # analytical solution for QHO
    qho_sol_ana_0 = np.exp(-(x_arr_qho)**2/2)
    qho_sol_ana_1 = np.sqrt(2.0)*(x_arr_qho)*np.exp(-(x_arr_qho)**2/2)*(-1)
    qho_sol_ana_2 = (1.0/np.sqrt(2.0))*(2.0*(x_arr_qho)**2-1.0)*np.exp(-(x_arr_qho)**2/2)
    qho_sol_list = []
    qho_sol_list.append(qho_sol_ana_0)

```

```

qho_sol_list.append(qho_sol_ana_1)
qho_sol_list.append(qho_sol_ana_2)
return x_arr_qho, qho_out_arr, np.asarray(qho_sol_list)

def Schrod_deriv(y, r, L, E):
    """Odeint calls routine to solve Schroedinger equation of the Hydrogen atom.
    """
    du2 = y[0]*((L*(L+1))/(r**2) - 2./r - E)
    return [y[1], du2]

def shoot_hydrogen(psi_init, h_, L):
    """ """
    x_arr_hydro = np.arange(0.0001, 35.0+h_, h_)
    E_arr = np.arange(-1., 0., 0.001)
    rightb = []
    for EE in E_arr:
        psi = integrate.odeint(Schrod_deriv, psi_init,
                               x_arr_hydro, args=(L,EE))[:, 0]
        rightb.append(psi[len(psi)-1])
    rightb_arr = np.asarray(rightb)
    crossings = findZeros(rightb_arr)
    energy_l = []
    for cross in crossings:
        energy_l.append(newton(shoot_ode, E_arr[cross],
                               args=(psi_init, x_arr_hydro, L)))
    psi_out = []
    for En in energy_l:
        psi_out.append(integrate.odeint(Schrod_deriv, psi_init,
                                         x_arr_hydro, args=(L,En))[:, 0])
    return x_arr_hydro, np.asarray(psi_out)

def HYDR0_ana(x, N, L):
    """Return analytical solution for Hydrogen SE."""
    # analytical solution hydrogen for N=1
    if((N-L-1) == 0) and (L == 0):
        #return 2.0*np.exp(-x/2)*x
        return x*np.exp(-x)
    elif((N-L-1) == 1) and (L == 0):
        return (np.sqrt(2.)*(-x + 2.)*np.exp(-x/2.)/4.)*x
    elif((N-L-1) == 2):
        return (2.*np.sqrt(3.)*(2.*x**2./9. - 2.*x + 3.)*np.exp(-x/3.)/27.)*x
    elif((N-L-1) == 0) and (L == 1):
        return (np.sqrt(6.)*x*np.exp(-x/2.)/12.)*x
    else:
        print "No analytic wave function found. Please try again."
        print "Output will be zero array."
        return np.zeros(len(x))

def plot_wavefunction(fig, title_string, x_arr, num_arr, ana_arr, axis_list):
    """Output plots for wavefunctions."""
    # clear plot

```

```

plt.cla() # clear axis
plt.clf() # clear figure
plt.plot(x_arr, num_arr, 'b:', linewidth=4,
         label=r"$\Psi(\hat{x})_{\text{num}}$")
plt.plot(x_arr, normalize(ana_arr), 'r-',
         label=r"$\Psi(\hat{x})_{\text{ana}}$")
plt.ylabel(r"$\Psi(\hat{x})$", fontsize=16)
plt.xlabel(r'$\hat{x}$', fontsize=16)
plt.legend(loc='best', fontsize='small')
plt.axis(axis_list)
plt.title(title_string)
plt.grid()
fig.savefig("plots/wavefunc_"+title_string+".png")

# Initial conditions for pot.well and harmonic osc
psi_0 = 0.0
phi_0 = 1.0
psi_init = np.asarray([psi_0, phi_0])
h_ = 1.0/200.0 # stepsize for range arrays

fig = plt.figure()

ipw_x, ipw_num, ipw_ana = shoot_potwell(psi_init, h_)
qho_x, qho_num, qho_ana = shoot_QuantumHarmonicOscillator(psi_init, h_)
hydro_x, hydro_num = shoot_hydrogen(psi_init, h_, 0)
hydro_x2p, hydro_num2p = shoot_hydrogen(psi_init, h_, 1)
hydro_ana1s = HYDR0_ana(hydro_x, 1, 0)
hydro_ana2s = HYDR0_ana(hydro_x, 2, 0)
hydro_ana3s = HYDR0_ana(hydro_x, 3, 0)
#print hydro_num
hydro_ana2p = HYDR0_ana(hydro_x, 2, 1)

print "IPW shooting"
plot_wavefunction(fig, "Infinte Potential Well -- Ground State",
                  ipw_x, ipw_num[0, :], ipw_ana[0, :], [-0.1, 1.1, -0.2, 1.2])

plot_wavefunction(fig, "Infinte Potential Well -- First Excited State",
                  ipw_x, ipw_num[1, :], ipw_ana[1, :], [-0.1, 1.1, -1.2, 1.2])

plot_wavefunction(fig, "Infinte Potential Well -- Second Excited State",
                  ipw_x, ipw_num[2, :], ipw_ana[2, :], [-0.1, 1.1, -1.2, 1.2])

print "QHO shooting"
plot_wavefunction(fig, "Quantum Hamonic Oscillator -- Ground State",
                  qho_x, qho_num[0, :], qho_ana[0, :], [-5.2, 5.2, -1.2, 1.2])

plot_wavefunction(fig, "Quantum Hamonic Oscillator -- First Excited State",
                  qho_x, qho_num[1, :], qho_ana[1, :], [-5.2, 5.2, -1.2, 1.2])

plot_wavefunction(fig, "Quantum Hamonic Oscillator -- Second Excited State",
                  qho_x, qho_num[2, :], qho_ana[2, :], [-5.2, 5.2, -1.2, 1.2])

print "Hydrogen Atom shooting"
plot_wavefunction(fig, "Hydrogen Atom -- 1s State",

```

```
hydro_x, normalize(hydro_num[0, :]), hydro_ana1s, [-0.1, 30., -0.1, 1.2])

plot_wavefunction(fig, "Hydrogen Atom -- 2s State",
                  hydro_x, normalize(hydro_num[1, :]), hydro_ana2s, [-0.1, 30., -2.2, 1.2])

plot_wavefunction(fig, "Hydrogen Atom -- 2p State",
                  hydro_x2p, normalize(hydro_num2p[0, :]), hydro_ana2p, [-0.1, 30., -0.1, 1.2])

plot_wavefunction(fig, "Hydrogen Atom -- 3s State",
                  hydro_x, normalize(hydro_num[2, :]), hydro_ana3s, [-0.1, 30., -1.2, 1.2])
```

Appendix B

Shooting Method – Approach 2

```
"""
Script for solving the one dimensional Schroedinger equation numerically.

Numerical integration method used is the fourth order Runge Kutta.
Counts the nodes of the wave function and determines the harmonic.
Then refines the solution until proper energy is found.

Potentials:
Infinite Potential Well
 $V(x_{-} < 0) = \text{inf}$ ,  $V(x_{-} = 0, 1) = 0$ ,  $V(x_{-} > 1) = \text{inf}$ 
Analytic solution:
 $\sin(k * \pi * x)$ 

Harmonic Oscillator:
 $V(x_{-}) = x_{-} ** 2$ 
Analytic solution:
 $(1 / (\sqrt{(2 * n) * n!}) * H(x)) * \exp(-x ** 2 / 2)$ 
"""

import numpy as np
import scipy
from scipy import integrate
from scipy.signal import argrelextrema
import matplotlib.pyplot as plt

def Schroed(y, r, V, E):
    """Return one dim Schroedinger equation with Potential V."""
    psi, phi = y
    dphidx = [phi, (V-E)*psi]
    return np.asarray(dphidx)

def rk4(f, psi0, x, V, E):
    """Fourth-order Runge-Kutta method to solve  $\phi' = f(\psi, x)$  with  $\psi(x[0]) = \psi_0$ .

    Integrates function  $f$  with initial values  $\psi_0$  and potential  $V$  numerically.
    Output is possible multidimensional (in  $\psi$ ) array with  $\text{len}(x)$ ."
    """
```

```

"""
n = len(x)
psi = np.array([psi0]*n)
for i in xrange(n - 1):
    h = x[i+1] - x[i]
    k1 = h*f(psi[i], x[i], V[i], E)
    k2 = h*f(psi[i] + 0.5*k1, x[i] + 0.5*h, V[i], E)
    k3 = h*f(psi[i] + 0.5*k2, x[i] + 0.5*h, V[i], E)
    k4 = h*f(psi[i] + k3, x[i+1], V[i], E)
    psi[i+1] = psi[i] + (k1 + 2.0*(k2 + k3) + k4) / 6.0
return psi

def findZeros(rightbound_vals):
    """Find zero crossing due to sign change in rightbound_vals array.

    Return array with array indices before sign change occurs.
    """
    return np.where(np.diff(np.signbit(rightbound_vals)))[0]

def normalize(output_wavefunc):
    """A function to roughly normalize the wave function to 1."""
    normal = max(output_wavefunc)
    return output_wavefunc*(1/(normal))

def countNodes(wavefunc):
    """Count nodes of wavefunc by finding Minima and Maxima in wavefunc."""
    maxarray = argrelextrema(wavefunc, np.greater)[0]
    minarray = argrelextrema(wavefunc, np.less)[0]
    nodecounter = len(maxarray)+len(minarray)
    return nodecounter

def RefineEnergy(Ebot, Etop, Nodes, psi0, x, V):
    tolerance = 1e-12
    ET = Etop
    EB = Ebot
    psi = [1]
    while (abs(EB - ET) > tolerance or abs(psi[-1]) > 1e-3):
        initE = (ET + EB)/2.0
        psi = rk4(Schroed, psi0, x, V, initE)[: , 0]
        nodes_ist = len(findZeros(psi))-1
        if nodes_ist > Nodes + 1:
            ET = initE
            continue
        if nodes_ist < Nodes - 1:
            EB = initE
            continue

        if (nodes_ist % 2 == 0):
            if ((psi[len(psi)-1] <= 0.0)):
                ET = initE
            else:

```

```

        EB = initE
    elif nodes_ist > 0:
        if ((psi[len(psi)-1] <= 0.0)):
            EB = initE
        else:
            ET = initE
    elif nodes_ist < 0:
        EB = initE
    return EB, ET

def ShootingInfinitePotentialWell(E_interval, nodes):
    """Implementation of Shooting method for Infinite PotWell

    INPUT: E_interval array with top and bottom value, len(E_interval)=2
           nodes: Number wavefunction nodes => determines quantum state.
    OUTPUT: refined energy value
           numerical wavefunction as array.
    """
    psi_0 = 0.0
    phi_0 = 1.0
    psi_init = np.asarray([psi_0, phi_0])
    h_mesh = 1.0/100.0 # stepsize for range arrays
    x_arr_ipw = np.arange(0.0, 1.0+h_mesh, h_mesh) # set up mesh
    V_ipw = np.zeros(len(x_arr_ipw)) # set up potential
    EBref, ETref = RefineEnergy(E_interval[0], E_interval[1], nodes, psi_init,
                                x_arr_ipw, V_ipw)
    psi = rk4(Schroed, psi_init, x_arr_ipw, V_ipw, EBref)[: , 0]
    return EBref, normalize(psi), x_arr_ipw

def IPW_ana(x, k):
    """Return analytical wavefunc of respective state (k) of IPW."""
    return np.asarray(np.sin(k*np.pi*x))

def ShootingQuantumHarmonicOscillator(E_interval, nodes):
    """Shooting QHO."""
    psi_0 = 0.0
    phi_0 = 1.0
    psi_init = np.asarray([psi_0, phi_0])
    h_mesh = 1.0/100.0 # stepsize for range arrays
    x_arr_qho = np.arange(-5.0, 5.0+h_mesh, h_mesh) # set up mesh
    V_qho = x_arr_qho**2 # set up potential
    EBref, ETref = RefineEnergy(E_interval[0], E_interval[1], nodes, psi_init,
                                x_arr_qho, V_qho)
    psiB = rk4(Schroed, psi_init, x_arr_qho, V_qho, EBref)[: , 0]
    psiT = rk4(Schroed, psi_init, x_arr_qho, V_qho, ETref)[: , 0]
    return EBref, ETref, normalize(psiB), normalize(psiT), x_arr_qho

def QHO_ana(x, nodes):
    """Return analytic solution for QHO for up to 5 nodes."""
    if(nodes == 1):
        return np.exp(-(x)**2/2)

```



```

elif(nodes == 2):
    return np.sqrt(2.0)*(x)*np.exp(-(x)**2/2)*(-1)
elif (nodes == 3):
    return (1.0/np.sqrt(2.0))*(2.0*(x)**2-1.0)*np.exp(-(x)**2/2)
elif (nodes == 4):
    return (1.0/np.sqrt(3.0))*(2.0*(x)**3-3.0*x)*np.exp(-(x)**2/2)*(-1)
elif (nodes == 5):
    return (1.0/np.sqrt(24.0))*(4.0*(x)**4-12.0*x**2+3.)*np.exp(-(x)**2/2)
else:
    print "No analytic wave function found. Please try again."
    print "Output will be zero array."
    return np.zeros(len(x))

# Start
E_qho = [0.1, 100.0]
E_ipw = [1.0, 500.0]

nodes_arr = np.arange(1, 6, 1)
L = 0.0
N = 1.0

print "Welcome!"
print "Maximum quantum state is currently limited to the 4th excited state."
print "\n"

print "Infinte Potential Well Shooting"
figipw = plt.figure()
for ii in nodes_arr:
    Energy, psi_ipw, x_ipw = ShootingInfinitePotentialWell(E_ipw, ii)
    psi_ana = normalize(IPW_ana(ii, x_ipw))
    print "Found quantum state at energy = % s [Hartree]" % (Energy, )
    plt.cla() # clear axis
    plt.clf() # clear figure
    plt.plot(x_ipw, psi_ipw, 'b-.', label=r'$\Psi(x)_{\text{num}}$')
    plt.plot(x_ipw, psi_ana, 'r--', label=r'$\Psi(x)_{\text{ana}}$')
    plt.title('Eigenstate: %s' % (ii, ))
    plt.legend(loc='best', fontsize='small')
    plt.grid()
    figipw.savefig('plots/ipw_shoottest_state_'+str(ii)+'.png')

print "\n"
print "Quantum Harmonic Oscillator Shooting:"
figqho = plt.figure()
for ii in nodes_arr:
    EB, ET, psibot, psitop, x_qho = ShootingQuantumHarmonicOscillator(E_qho, ii)
    psi_ana = QHO_ana(x_qho, ii)
    print "Found quantum state at energy = %s [Hartree]" % (ET, )
    plt.cla() # clear axis
    plt.clf() # clear figure
    plt.plot(x_qho, psitop, 'b-.', label=r'$\Psi(x)_{\text{num}}$')
    plt.plot(x_qho, normalize(psi_ana), 'r--',
             label=r'$\Psi(x)_{\text{ana}}$')
    plt.title('Eigenstate: %s' % (ii, ))
    plt.legend(loc='best', fontsize='small')

```

```
plt.grid()
figqho.savefig('plots/qho_shoottest_state_'+str(ii)+'.png')

print "\n"
print "Please find plots of wavefunctions in 'plots'-folder."
print "\nGoodbye."
```

Appendix C

Shooting Method Source Code for Hydrogen Atom

```
"""
Script for solving the one dimensional Schroedinger equation numerically.

Numerical integration method used is scipy.integrate.odeint.
Counts the nodes of the wave function and determines the harmonic.
Then refines the solution until proper energy is found.

Radial Hydrogen Atom:
 $V(r) = 2/r - (L(L+1))/(r**2)$  a.u.
"""

import numpy as np
import scipy
from scipy import integrate
# from scipy.signal import argrelextrema
from scipy.optimize import newton
import matplotlib.pyplot as plt

# for solving with scipy integrate package
def Schrod_deriv(y, r, L, E):
    """Odeint calls routine to solve Schroedinger equation of the Hydrogen atom.
    """
    du2 = y[0]*((L*(L+1))/(r**2) - 2./r - E)
    return [y[1], du2]

def shoot1(E, psi_init, x, L):
    """Helper function for optimizing results."""
    sol = integrate.odeint(Schrod_deriv, psi_init, x, args=(L,E))
    return sol[len(sol)-1][0]

def findZeros(rightbound_vals):
    """Find zero crossing due to sign change in rightbound_vals array.
```

```

    Return array with array indices before sign change occurs.
    """
    return np.where(np.diff(np.signbit(rightbound_vals)))[0]

def normalize(output_wavefunc):
    """A function to roughly normalize the wave function to 1."""
    normal = max(output_wavefunc)
    return output_wavefunc*(1/(normal))

def RefineEnergy(Ebot, Etop, Nodes, psi0, x, L):
    tolerance = 1e-12
    ET = Etop
    EB = Ebot
    psi = [1]
    while (abs(EB - ET) > tolerance or abs(psi[-1]) > 1e-3):
        print ET, EB
        initE = (ET + EB)/2.0
        psi = integrate.odeint(Schrod_deriv, psi0, x, args=(L, initE))[:, 0]
        nodes_list = len(findZeros(psi))-1
        if nodes_list > Nodes + 1:
            ET = initE
            continue
        if nodes_list < Nodes - 1:
            EB = initE
            continue

        if (nodes_list % 2 == 0):
            if ((psi[len(psi)-1] <= 0.0)):
                ET = initE
                print "ET!!"
            else:
                EB = initE
        elif nodes_list > 0:
            if ((psi[len(psi)-1] <= 0.0)):
                EB = initE
            else:
                ET = initE
        elif nodes_list < 0:
            EB = initE
    return EB, ET

def ShootingHydrogenAtom(psi_init_hydro, N, L, x_arr_hydro):
    """Shooting method for quantum harmonic oscillator.

    Returns the numerical wave function as array.
    """
    nodes = N-L-1 # Number of should be nodes
    E_hydro_top = 30.0 # top boundary energy
    E_hydro_bot = -9.0 # bottom boundary energy
    EBref, ETref = RefineEnergy(E_hydro_bot, E_hydro_top, nodes+1, psi_init_hydro,
                                x_arr_hydro, L)

```

```

    Enewton = newton(shoot1, EBref, args=(psi_init_hydro, x_arr_hydro, L))
    EBOT = 0
    ETOP = 0
    return EBOT, ETOP, EBref, ETref, Enewton

def HYDRO_ana(x, N, L):
    """Return analytical solution for Hydrogen SE."""
    # analytical solution hydrogen for N=1
    if ((N-L-1) == 0) and (L == 0):
        #return 2.0*np.exp(-x/2)*x
        return x*np.exp(-x)
    elif ((N-L-1) == 1) and (L == 0):
        return (np.sqrt(2.)*(-x + 2.)*np.exp(-x/2.)/4.)*x
    elif ((N-L-1) == 2):
        return (2.*np.sqrt(3.)*(2.*x**2./9. - 2.*x + 3.)*np.exp(-x/3.)/27.)*x
    elif ((N-L-1) == 0) and (L == 1):
        return (np.sqrt(6.)*x*np.exp(-x/2.)/12.)*x
    else:
        print "No analytic wave function found. Please try again."
        print "Output will be zero array."
        return np.zeros(len(x))

# Quantum numbers
L = 0. # angular quantum number
N = 1. # principal quantum number

h_ = 1./200.
#x_arr_hydrois = np.arange(0.0001, 20.0+h_, h_)
x_arr_hydro = np.arange(1e-7, 35.0+h_, h_)
# Initial conditions as array
psi_init = np.asarray([0., 1.]) # Init cond for hydrogen

nodes = np.arange(1,4,1)
for ii in nodes:
    EB, ET, Bref, Tref, newtonE = ShootingHydrogenAtom(psi_init, ii, 0, x_arr_hydro)
    hydro_ana = HYDRO_ana(x_arr_hydro, ii, 0)
    psiB = integrate.odeint(Schrod_deriv, psi_init,
                           x_arr_hydro, args=(L,Tref,))[0]
    plt.plot(x_arr_hydro, normalize(psiB), 'g:', linewidth = 5,
             label='wavefunction odeint from ebot')
    plt.plot(x_arr_hydro, normalize(hydro_ana), 'r-', label='wavefunction analytic')
    plt.show()

EB, ET, Bref, Tref, newtonE = ShootingHydrogenAtom(psi_init, 2, 1, x_arr_hydro)
hydro_ana = HYDRO_ana(x_arr_hydro, 2, 1)
psiB = integrate.odeint(Schrod_deriv, psi_init,
                       x_arr_hydro, args=(1,Tref,))[0]
plt.plot(x_arr_hydro, normalize(psiB), 'g:', linewidth = 5,
        label='wavefunction odeint from ebot')
plt.plot(x_arr_hydro, normalize(hydro_ana), 'r-', label='wavefunction analytic')
plt.show()

```

Bibliography

- [1] 2016. URL <https://www.scipy.org/>.
- [2] M.S.S Brooks. How to build an atom, September 2009.
- [3] E. Langer. *Bakkalaureats-Vertiefung Mathematik Teil 1: Numerische Verfahren*. Institute of Microelectronics, 2010.
- [4] Author: Autar Kaw et al. *Numerical Methods with Applications*. Number 978-0-578-05765-1. <http://www.autarkaw.com>, May 2010.
- [5] Prof. Dr. Marc Wagner. *Numerische Methoden der Physik, Lecture Notes*. Goethe Universitaet Frankfurth am Main, beta-version 0.6 edition, 2011/12.
- [6] J. Killingbeck. Shooting methods for the schrödinger equation. *J. Phys. A: Math. Gen.*, 20:1411 – 1417, 1987.
- [7] H.Arabshahi A. Binesh, A.A. Mowlavi. Suggestion of proper boundary conditions to solving schrodinger equation for different potentials by runge-kutta method. *Research Journal of Applied Sciences*, 5((6)):383–387, 2010.
- [8] 2005. URL <http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/hydsch.html>.
- [9] 2001. URL <https://www.python.org/dev/peps/pep-0257/>.
- [10] Bernhard Broecker. *dtv-Atlas Atomphysik*. Dtv, 6th editon edition, 1997.
- [11] 2015. URL <http://www.math-cs.gordon.edu/courses/ma342/python/>.
- [12] 2001. URL <https://www.python.org/dev/peps/pep-0008/>.
- [13] July 2016. URL <http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html>.