Computer Programming and Numerical Methods

Candidate Number: 1090111

January 2025

1 Introduction

The aim of this document is to answer some of the questions on the question paper and to give more insights into the workflow of this project. Please note that the submission document will be a ZIP file, in which there should be two Python documents (one for each set of wavefunctions), several .PNG files that show the different generated plots, and this PDF document that will explain some parts of this project.

2 Numerical algorithms

Part (c) of the long project involved writing a routine to numerically integrate the RDFs using two different methods of integration. The chosen methods were *Simpson's Rule* and the *Trapezium Rule*.

2.1 Simpson's Rule

Simpson's Rule was chosen as one of the numerical integration methods as it is typically used to approximate functions to the quadratic level. Since the RDF is a quadratic function, Simpson's Rule is an appropriate method of integration.
Generally, the compound Simpson's Rule is given by the equation below.

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_0+2i\Delta x}^{x_0+2(i+1)\Delta x} \tag{1}$$

$$\int_a^b f(x)dx = \frac{\Delta x}{3} \sum_{i=0}^{n-1} f(x_0 + 2i\Delta x) + 4f(x_0 + (2i+1)\Delta x) + f(x_0 + 2(i+1)\Delta x) \tag{2}$$

Now to implement this equation into the project, the following code was written:

```
def simpsons_rule(func, a, b, n):
if n % 2 != 0:
    raise ValueError("Number of intervals (n) must be even for Simpson's rule.")
h = (b - a) / n
x = np.linspace(a, b, n + 1)
y = func(x)
integral = y[0] + y[-1] + 4 * np.sum(y[1:-1:2]) + 2 * np.sum(y[2:-2:2])
integral *= h / 3
return integral
```

In simple terms, the code takes the interval from a to b and cuts it into n evenly-spaced segments. The function's value is calculated at all the cut-points and are then added up in a pattern: 4x the value for the odd indices and 2x the value for the even indices. The sum is then multiplied by the segment width (h) and divided by 3 as per Simpson's Rule. For this rule, n must be an even number because we're estimating the areas of regions of width 2Δx.

2.2 the Trapezium Rule

The other method of choice was the Trapezium Rule, which tends to be more accurate given the fact it has a smaller error, $O(\Delta x^5)$, compared to Simpson's Rule which sits at an error of $O(\Delta x^3)$. Generally, the compound Trapezium Rule is given by the equation:

$$\int_{x_0}^{x_1} f(x)dx = \sum_{i=0}^{n-1} \int_{x_0+i\Delta x}^{x_0+(i+1)\Delta x} f(x)dx \tag{3}$$

$$\int_{x_0}^{x_1} f(x)dx = \frac{\Delta x}{2}[f(x_0) + 2f(x_0 + \Delta x) + 2f(x_0 + 2\Delta x) + ... + 2f(x_0 + (n-1)\Delta x) + f(x_1)] \tag{4}$$

The implementation of the code is given by:

```
def trapezoidal_rule(func, a, b, n):
    h = (b - a) / n
    x = np.linspace(a, b, n + 1)
    y = func(x)
    integral = h * (np.sum(y) - 0.5 * (y[0] + y[-1]))
    return integral
```

This code splits the interval from a to b into n evenly-spaced segments, much like Simpson's Rule, and computes the function's value at each of those segments. It then approximates the area under the curve by treating the shape between each pair of adjacent points as a trapezium. Finally, it adds up the areas of those trapeziums by summing the function values with the endpoints each counting half as much and then multiplies by the width h.
In simple terms, the curve is replaced by straight lines between each pair of sample points and the areas of the resulting trapeziums are then summed to get an approximation of the integral.

Note that for both methods, a = 0, b = 30 and n = 1000.

3 Results and Discussion

3.1 Plotting the RDFs

When the code is executed, a plot will be obtained as seen in Figure 1 below.
The plot for set 1 shows the RDF for each of the different orbitals using the wavefunctions given on the question paper. Note how this plot is obtained by normalising the RDFs so the maximum of each RDF is one, as suggested by the question paper.

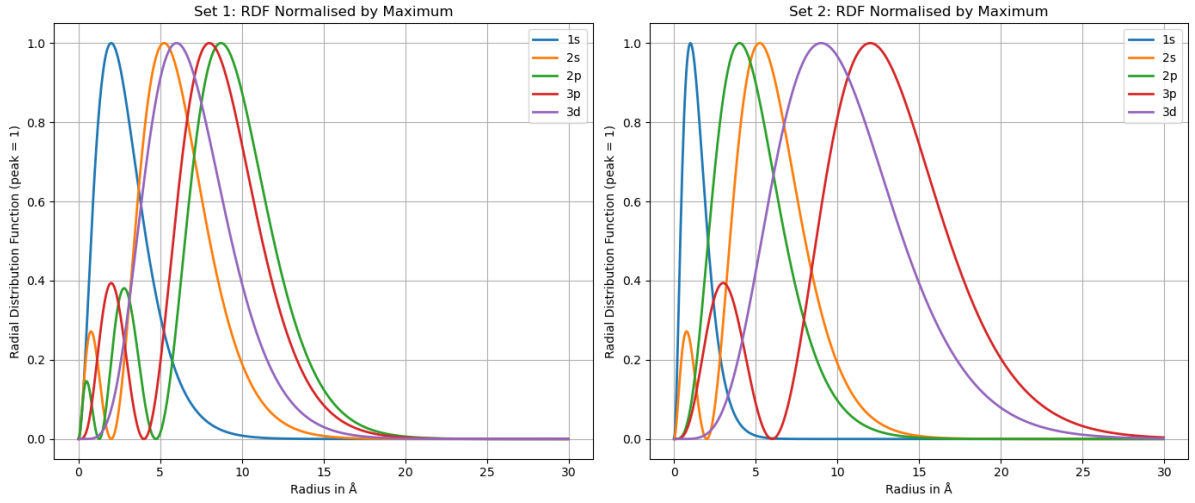

Figure 1: Plot of RDFs normalised by maximum using two different sets of wavefunctions.

The shape of each of the different RDFs is exactly as you would expect, except for the 2p orbital. For a 2p orbital one would not expect to see any nodes, but Figure 1 shows that there are two. Therefore, one of two things must have happened. Either (1), the given equation

$$(6 - 6r + r^2)e^{-r/2} \tag{5}$$

was incorrectly written into the Python code and therefore the RDF was incorrectly calculated or (2), the given equation is not an accurate representation of a 2p orbital. In this case, the observed output would be an accurate representation of the given wavefunction, but the wavefunction itself is not an accurate description of a 2p orbital. Realising that the given wavefunction may not be an accurate description of a 2p orbital, some research outside the scope of this project was done to examine what the correct wavefunction of a 2p orbital would look like.

According to *Perspectives of Modern Physics*[1], the hydrogen atom wavefunctions of different orbitals should instead be characterised by the equations in Table 1.

Type of Orbital	Orbital Wavefunction
1s	e^{-r}
2s	$(2 - r)e^{-r/2}$
2p	$re^{-r/2}$
3p	$(6r - r^2)e^{-r/3}$
3d	$r^2e^{-r/3}$

Table 1: Different types of orbitals and their (hydrogen atom) wavefunctions

Applying these wavefunctions to the existing code will generate the plot for set 2 as found in Figure 1. It is now clear to see that the RDFs behave much more like what one would expect from each of the different orbitals.

3.2 Numerical Integration of the RDFs

The next part of the project involved numerically integrating the RDFs for the different orbitals. This was done using Simpson's Rule and the Trapezium Rule, the output of which can be found below. Note that these are the values for the wavefunctions given on the question sheet.

```
Integration Results using Simpsons Rule:
Integral of 1s orbital: 25.13274
Integral of 2s orbital: 100.53097
Integral of 2p orbital: 904.77802
Integral of 3p orbital: 1809.55656
Integral of 3d orbital: 9047.78578

Integration Results using Trapezium Rule:
Integral of 1s orbital: 25.13274
Integral of 2s orbital: 100.53096
Integral of 2p orbital: 904.77797
Integral of 3p orbital: 1809.55656
Integral of 3d orbital: 9047.78578
```

As seen above, the integration results are about the same for the two different methods. This proves that for both methods, the orbitals were integrated successfully. Any substantial difference in results would mean that one of the two methods had been implemented incorrectly.
For completeness, the results of the numerical integration of the different orbitals as given by the equations in Table 1 are found below. Note that the difference in integration values is due to the difference in wavefunction equations.

[1] A. Beiser, Perspectives of Modern Physics, 1968-01-01, p.202, Accessed: 18-01-2024

```
Integration Results using Simpsons Rule:
Integral of 1s orbital: 3.14159
Integral of 2s orbital: 100.53097
Integral of 2p orbital: 301.59289
Integral of 3p orbital: 30891.96952
Integral of 3d orbital: 154550.48114

Integration Results using Trapezium Rule:
Integral of 1s orbital: 3.14159
Integral of 2s orbital: 100.53096
Integral of 2p orbital: 301.59289
Integral of 3p orbital: 30891.96911
Integral of 3d orbital: 154550.48047
```

Note that none of the integration values amount to 1, which is the value you would expect to see. This is because the numerical integration is performed using the RDFs that have been normalised in such a way that their maximum value will be 1. The equation for this normalisation method is:

$$RDF_{normalised}(r) = \frac{RDF(r)}{RDF_{max}} \tag{6}$$

However, true normalisation of the RDF is achieved by using a normalisation constant, not by rescaling the RDF such that their maximum value is one. The process for the determination of the normalisation constants can be found in section 2.3.

3.3 Determination of the Normalisation Constants

After determination of the integration values of the different RDFs, the normalisation constants were calculated using the trapezium integration values. The reported normalisation constants for the wavefunctions given on the question paper are found below.

```
Normalisation Constants (1 / Trapezium Integral):
Normalisation constant 1s orbital: 0.03979
Normalisation constant 2s orbital: 0.00995
Normalisation constant 2p orbital: 0.00111
Normalisation constant 3p orbital: 0.00055
Normalisation constant 3d orbital: 0.00011
```

The normalisation constants for the wavefunctions given in Table 1 are found to be:

```
Normalisation Constants (1 / Trapezium Integral):
Normalisation constant 1s orbital: 0.31831
Normalisation constant 2s orbital: 0.00995
Normalisation constant 2p orbital: 0.00332
Normalisation constant 3p orbital: 0.00003
Normalisation constant 3d orbital: 0.00001
```

Now that the normalisation constants have been determined, new plots can be generated to give a more accurate description of the shapes of the different orbitals. The area under the curve for each of the different RDFs will now equal one, as is always the case for the integration of properly normalised wavefunctions for RDFs. This plot can be found in Figure 2, where the plot for set 1 is obtained using the wavefunctions on the question paper and the plot for set 2 is generated using the wavefunctions from Table 1.

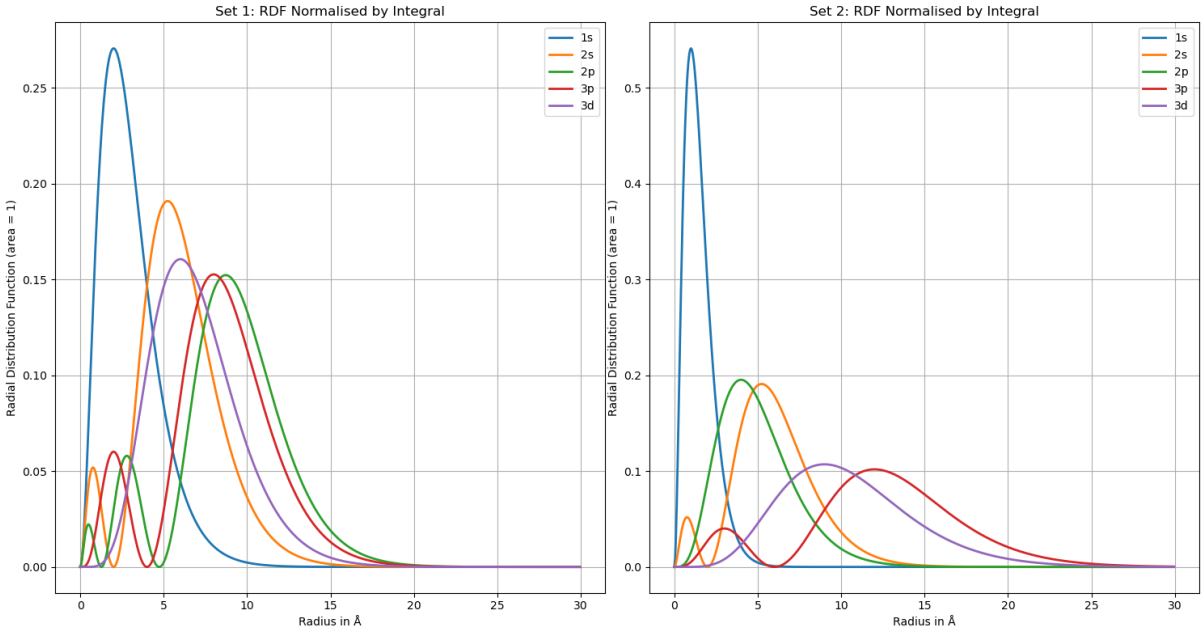

Figure 2: Plot of RDFs normalised by area using two different sets of wavefunctions.

Please note that the code for this project was written in several steps to reflect the order in which the questions are asked on the question paper. Realistically, many of the steps within the code could be combined or streamlined, but this would go against the flow of the assessment and therefore it was decided to write the code for the assessment as presented in the Python script.