Yi Zheng Wong (638183)

PHYC30012 – Computational Physics Report 1: Semiclassical Quantisation of Molecular Vibrations

Aim

The aim for this project is to study and implement numerical integration and root-finding algorithms to determine the various allowed energy levels in a molecule subjected to semiclassical quantisation rules.

The accuracies of the numerical integration and root-finding algorithm will be assess by comparing to analytically solvable problems.

For the molecules, two models will be studied; the *quadratic potential* and the *Morse potential*. The numerical integration and root-finding algorithm will be used to evaluate the allowed energy levels and it will be analyzed how well it agrees with experimental data based on the measured energy levels of the hydrogen molecule.

Theory of Molecular Vibration

Molecular vibrations occur when the particle in consideration is bounded to the other particle, resulting in vibration, which results in a closed loop in its phase space trajectory.

According to Bohr's semi-classical quantization rule, the area of the closed loop in the phase space of bounded particles will be quantized according to the equation[1]:

$$S = \left(n + \frac{1}{2}\right) h$$

Where h is the Planck's constant. The quantization of phase space results in the particle's energy only allowed to take on certain values E_n. The $\frac{1}{2}$ appearing in the above quantization rule is to ensure the particle's energy does not collapse to zero, in violation to the uncertainty principle.

The particle's trajectory in the phase space is given by the following equation:

$$\frac{p^2}{2m} + V(x) = E_n$$

Where p is the momentum, m is the mass of the molecule, $V(x)$ is the potential energy in terms of the separation distance x and energy E_n. The above equation then can be expressed in terms of p:

$$p = \pm\sqrt{2m(E_n - V(x))}$$

The $\pm$ emphasizes the property that in the phase space trajectory forms a loop and is symmetric about the x axis.

Applying Bohr's quantization rule, it can be deduced that:

$$2\int_{r_{in}}^{r_{out}} \sqrt{2m(E_n - V(x))}\, dx = \left(n + \frac{1}{2}\right) h$$

Which can then be expressed in terms of the reduced Planck's constant $\hbar$:

$$\gamma \int_{r_{in}}^{r_{out}} \sqrt{(E_n - V(x))}\, dx = \pi \left(n + \frac{1}{2}\right) - (1)$$

Where $\gamma = \frac{\sqrt{2m}}{\hbar}$. To avoid the use of really small numbers, the units for energy and length used will be electron volts and Angstrom respectively, so the value of γ will be 31.0556. The adjusted value of γ by considering that since (1) works in the SI unit, by performing a direct conversion from m to Å and J to eV:

$$\begin{aligned}\gamma &= \frac{\sqrt{2 \times 2 \times 1.6737 \times 10^{-27}}}{1.0546 \times 10^{-34}} \times 10^{-10} \times \sqrt{1.602 \times 10^{-19}} \\ &= 31.0556\end{aligned}$$

Hence the values of allowed energy levels can be found by finding E_n for any given n such that the above equation is satisfied.

Unfortunately, for realistic models such as *Morse* and *Leonard-Jones* potential molecular systems, it is impossible to solve (1) analytically. However it can be done numerically, which is what will be done for this project.

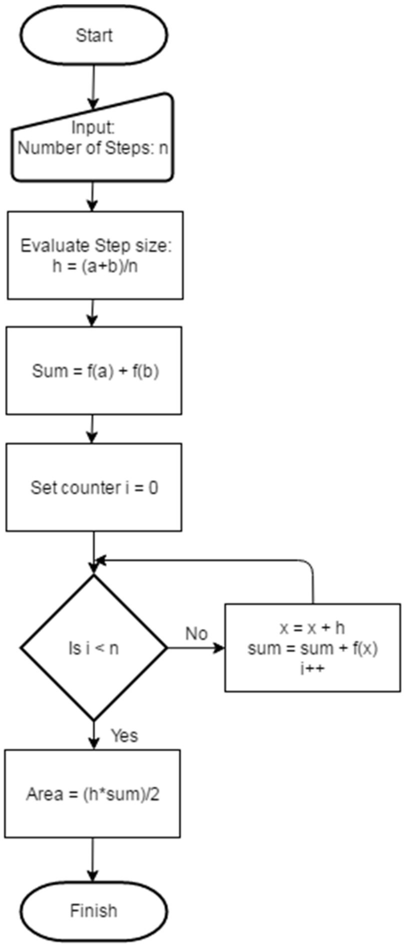
Numerical Integration

Numerical integration can be performed by employing either the *Trapezoid rule* or the *Simpson's rule*, which states that any definite integral can be approximated respectively by:

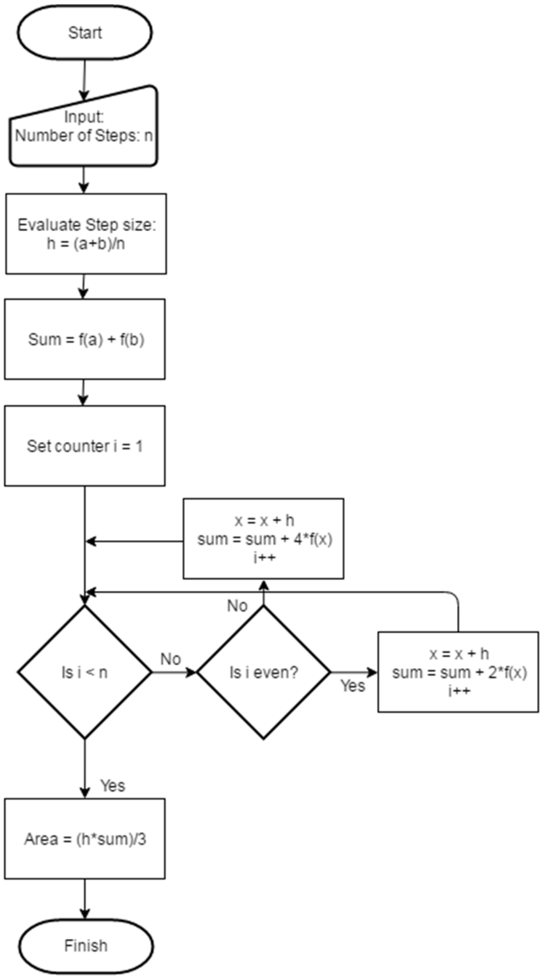
$$\int_a^b f(x)\, dx \approx \frac{\Delta x}{2}[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)] \rightarrow Trapezoid$$
$$\int_a^b f(x)\, dx \approx \frac{\Delta x}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 4f(x_{n-1}) + f(x_n)] \rightarrow Simpson$$

Where $x_0 = a$ and $x_n = b$, and x_i for $0 < i < n$ is evenly dividing along the interval $[a, b]$. It is important to note that Simpson's rules only works if n is even, where the interval is divided even number of times.

Trapezoid and Simpson's rule can be performed by in C, the following flow chart will illustrate its numerical algorithm respectively:



Trapezoid Rule's flowchart



Simpson Rule's flowchart

The implementation and testing of the above numerical integration rules are performed in the script file integral.c attached, and the number of steps are inputted in the terminal. The following known definite integral will be tested:

$$\int_0^1 \frac{\ln(1+x)}{x}dx = \frac{\pi^2}{12}$$

Also take note that as $\lim_{x\to 0}\frac{\ln(1+x)}{x} = 1$, which can be shown easily using L'Hopital's rule, in implementing the function into the c script file, it needs to be specified explicitly that $\frac{\ln(0)}{0} = 1$ to prevent a *nan* output.

Running integral.c and tabulating the percentage relative errors:

Number of steps	Rectangular	Trapezoid	Simpson
10	1.896510	0.031068	0.000089

50	0.374331	0.001244	0.000000
100	0.186855	0.000311	0.000000
500	0.037321	0.000012	0.000000
1000	0.018658	0.000003	0.000000

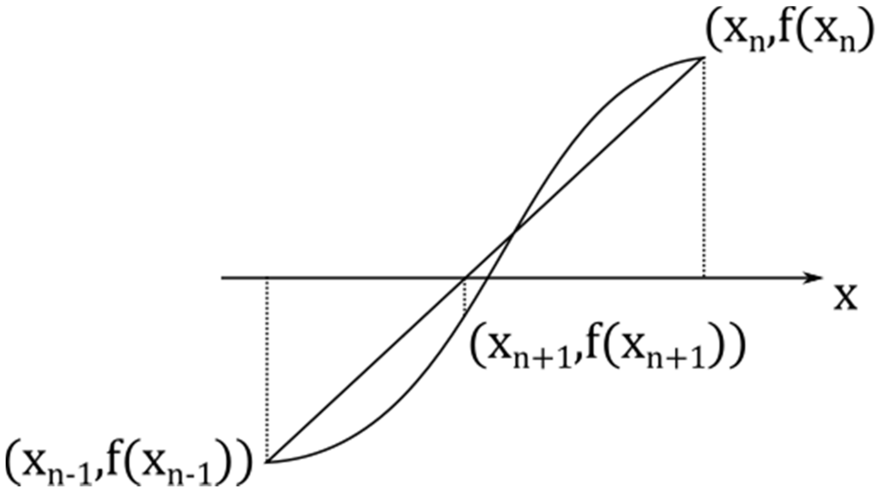
From the table it can be seen that Simpson's rule provides a much higher accuracy to the evaluation of a definite integral compared to the Rectangular and Trapezoid rule. Even at just 50 steps, the error is virtually non-existent.

Simpson's rule will be employed in the later analysis of molecular vibrations.

Root-Finding

Besides numerical integration, in order to find the allowed energy levels of a molecule a root-finding function is needed. The *false position* root-finding algorithm employed.

The false position algorithm works on the fact that on a given function $f(x)$, assuming continuity, that two values of x: x_{n-1} and x_n such that the signs of $f(x_{n-1})$ and $f(x_n)$ are opposite (where $f(x_{n-1})f(x_n) < 0$) have been found, the root must lie somewhere between x_{n-1} and x_n.

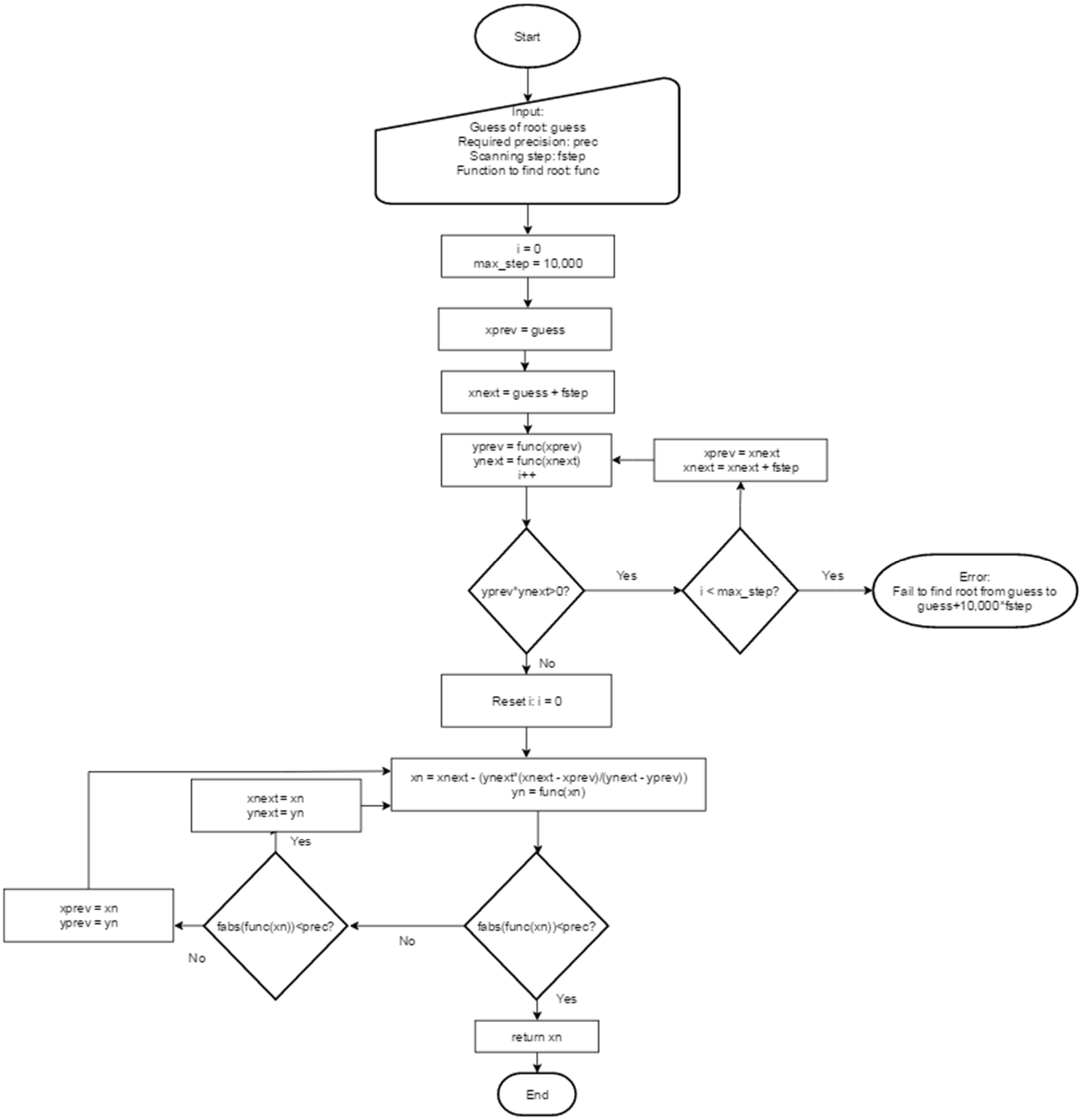


The false position algorithm allows us to narrow down to the next location of the root by connecting a secant between $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$ as shown above, where:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} - (2)$$

Depending on the sign of $f(x_{n+1})$, either x_n or x_{n-1} will be replaced by x_{n+1} such that the function will have opposite signs with the narrowed down root of x_{n+1}. This process will be repeated until the value of $f(x_{n+1})$ has the required precision.

The program to execute the false position method algorithm is first guess the lower bound of the root, and scan it till it finds a crossover, then applying equation (2), as exhibited by the following flow chart:



Putting the algorithm of Simpson rule and false position to the test to solve the equation:

$$\int_0^x t^2 dt = x$$

The file to solve this equation, which also requires the manual input of the lower bound guess, number of steps and the required precision in the terminal, is *integralroottest.c*.

As the above equation is solvable with roots of $\pm\sqrt{3} \approx 1.73205081$ and 0, this allows us to test the accuracy of the program as tabulated:

Lower bound guess	Number of Steps	Precision	Obtained roots
-2.0	10	0.01	-1.733453
0	100	0.01	0
1.0	100	0.01	1.731453
3.0	100	0.01	$Error$

From the obtained, the combination of Simpson's rule and false position algorithm has impressive accuracy up to the specified precision level up to 2 decimal places. Having more steps seems to have little effect on the accuracy.

Quadratic Potential

As a crude approximation, the potential of a molecule can be approximated by the following:

$$V(r) = 4V_0\left(\frac{r}{a} - 1\right)\left(\frac{r}{a} - 2\right)$$

Another big advantage of using quadratic potential is that the integral of a square root of a quadratic function can be evaluated analytically:

$$\int_{x_-}^{x_+} \sqrt{ax^2 + bx + c}\, dx = \frac{(4ac - b^2)\pi}{8a\sqrt{-a}} - (3)$$

Where x_- and x_+ are the smaller and the bigger roots respectively. Hence not only it serves as a crude model for the potential of a molecule, the numerical methods implemented can be tested.

Let $\varepsilon_n = \frac{E_n}{V_0}$, where the energy level is expressed as the proportion to the minimum point of the potential. The value of r_{min} and r_{max} can be shown to be:

$$r_{min} = \frac{a}{2}\left(3 - \sqrt{1 + \varepsilon_n}\right)$$
$$r_{max} = \frac{a}{2}\left(3 + \sqrt{1 + \varepsilon_n}\right)$$

Thus, equation (1) has to be modified into:

$$\gamma\sqrt{V_o}\int_{\frac{a}{2}\left(3-\sqrt{1+\varepsilon_n}\right)}^{\frac{a}{2}\left(3+\sqrt{1+\varepsilon_n}\right)} \sqrt{\varepsilon_n - 4\left(\frac{r}{a} - 1\right)\left(\frac{r}{a} - 2\right)}\, dx = \pi\left(n + \frac{1}{2}\right) - (3)$$

The integrand can be expressed as $\sqrt{-\frac{4r^2}{a^2} + \frac{12r}{a} - 2 + \varepsilon_n}$, using (2) ε_n can be shown to be:

$$\varepsilon_n = -1 + \frac{4\left(n + \frac{1}{2}\right)}{\gamma\sqrt{V_0}a}$$

The file to evaluate the value of ε_n both numerically and analytically is quad_molecule.c.

An additional algorithm to optimize the value of a is also implemented. It is done by first evaluating the relative error in the first two energy levels:

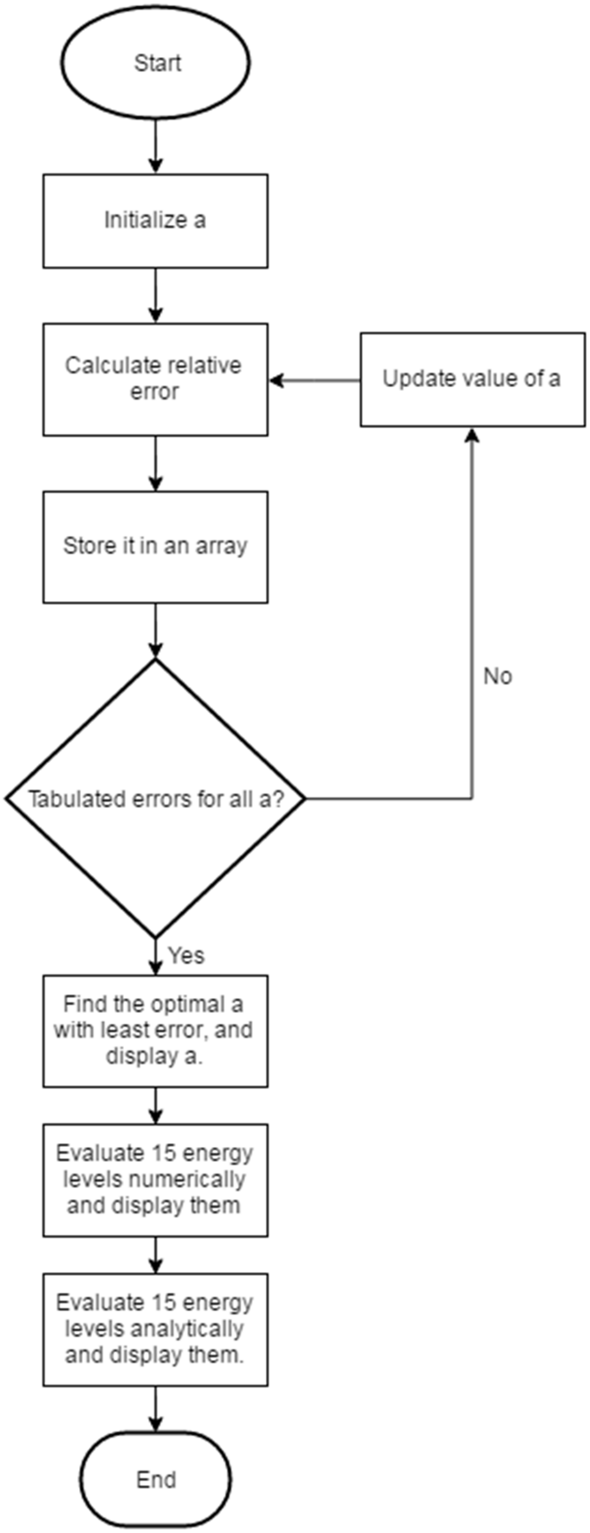
$$Relative\ error = (E_0 - E'_0)^2 + (E_1 - E'_1)^2$$

By scanning through the values of a incremental at a fixed step, tabulating the errors into an array, and use the implemented function *minimum* to locates the position of the lowest element in the array. This allows us to find the optimal value of a where the relative error in the first energy level is minimized.

a will start at the value of 0.3, and with an incremental value of 0.001, repeated 1000 times. This scan the optimal value from 0.3 to 1.3 up to 3 decimal places. In the implementation of Simpson's rule, the interval will be divided into $10{,}000$ parts; and in the false position root-finding algorithm will have a required precision of 10^{-6}.

In quad_molecule.c, the optimal value of a will be obtained first, then using this optimal value, the 15 energy levels will be evaluated both numerically and analytically and tabulated up to 3 decimal places, consistent with the experimental results' precision.

The structure of the program is exhibited by the following flow chart:


Start
Initialize a
Calculate relative error
Update value of a
Store it in an array
No
Tabulated errors for all a?
Yes
Find the optimal a with least error, and display a.
Evaluate 15 energy levels numerically and display them
Evaluate 15 energy levels analytically and display them.
End

The following output is obtained:

Optimal $a = 0.535$

Energy levels:

n^{th} Energy level	Numerical (eV)	Analytical (eV)	Actual (eV)
0	-4.485	-4.485	-4.477
1	-3.960	-3.960	-3.962
2	-3.436	-3.436	-3.475
3	-2.911	-2.911	-3.017
4	-2.387	-2.387	-2.587
5	-1.862	-1.862	-2.185
6	-1.338	-1.338	-1.811
7	-0.813	-0.813	-1.466
8	-0.288	-0.288	-1.151
9	0.236	0.236	-0.867
10	0.761	0.761	-0.615
11	1.285	1.285	-0.400
12	1.810	1.810	-0.225
13	2.334	2.334	-0.094
14	2.859	2.859	-0.017

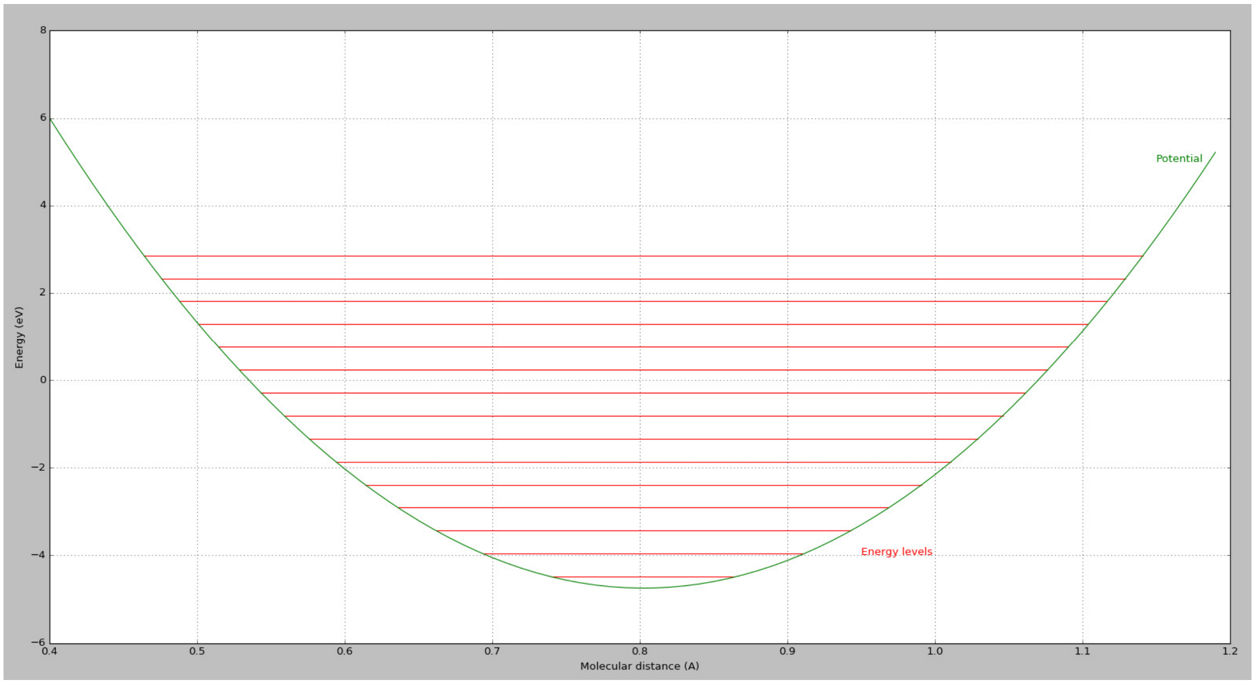
It can be seen that the numerical and analytical solutions obtained matches exactly up to 3 decimal places. Examining the output closely in the terminal, they only deviate at the 6^{th} decimal place for n greater or equal to 4. This is due to the limited precision imposed on the root-finding algorithm.

It can also be seen that the energy levels predicted by the quadratic model is fairly accurate for low values of n below 4; the calculated values starts to diverge away from the measured value above that. Furthermore it also predicts that the energy level is positive from $n = 9$ onwards, which is physically impossible as positive energy represents unbounded states. A more detailed analysis will be performed later when the quadratic potential model to the Morse Potential model are compared.

Incidentally, the quadratic model predicted the equilibrium distance of the molecule, fairly well at an equilibrium distance of:

$$r_{eq} \approx 0.803\text{Å}$$

The plot of the quadratic potential and the allowed energy levels can be done easily using Python matplotlib[2] in the file *quad_plot.py*. The following plot is obtained:



Morse Potential

The Morse potential is a much better approximation that takes into the account that the fact that the interaction between molecules gets weaker when they are further apart. It is given by:

$$V = V_0\left[\left(1-\exp\left(-\frac{(r-r_{eq})}{a}\right)\right)^2-1\right]$$

Where a in this case represents the strength of the bond. Lower value of a represents stronger bond.

As before, let $\varepsilon_n = \frac{E_n}{V_0}$. Hence to find the allowed energy levels, equation (1) can be expressed as:

$$\gamma\sqrt{V_0}\int_{r_{min}}^{r_{max}}\sqrt{\varepsilon_n+1-\left(1-\exp\left(-\frac{(r-r_{eq})}{a}\right)\right)^2}\,dr = \pi\left(n+\frac{1}{2}\right)$$

Where

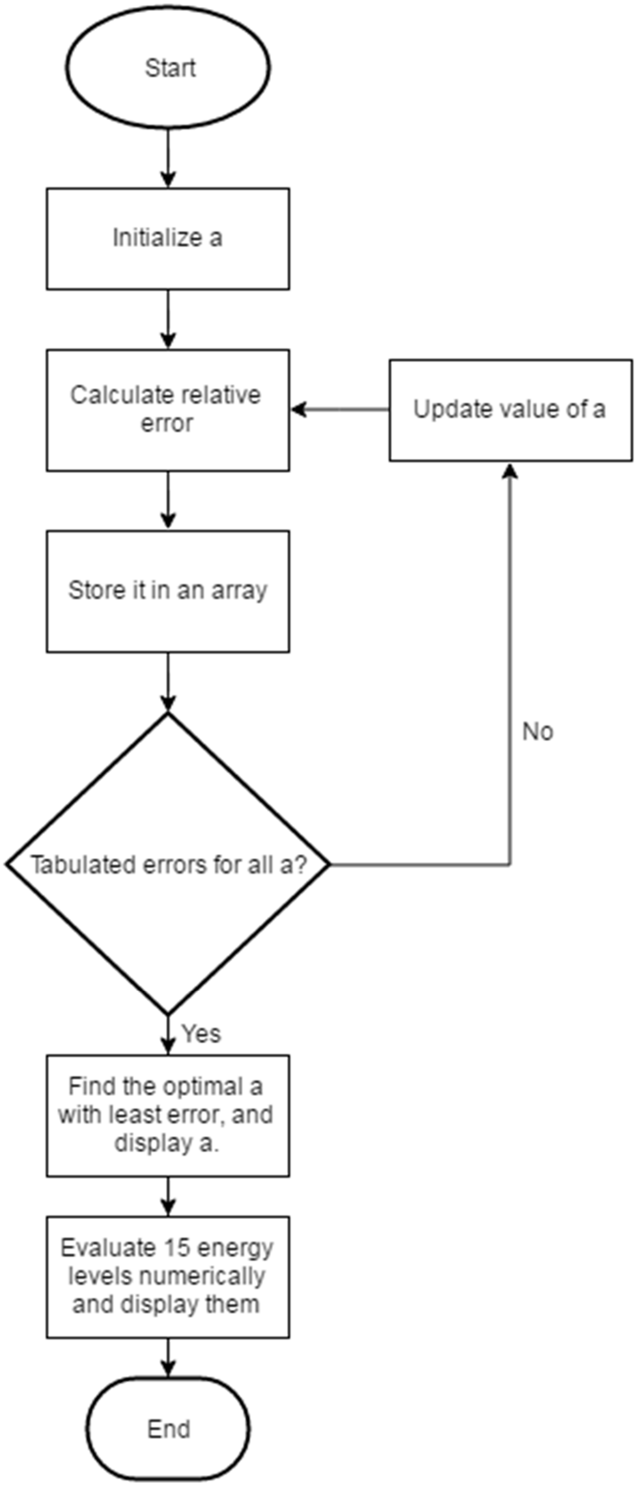
$$r_{max} = r_{eq} - a\ln\left(1-\sqrt{1+\frac{\varepsilon}{V_0}}\right)$$

$$r_{min} = r_{eq} - a\ln\left(1+\sqrt{1+\frac{\varepsilon}{V_0}}\right)$$

The evaluation of the allowed energy levels using the Morse potential is the same as the quadratic potential, where the value of a is first optimized by scanning the values of a, evaluating the relative errors, and picking a with the least error. Then using the optimal value to evaluate the energy levels. The values of a will be scanned from 0.1 to 1.1 at an incremental step of 0.001, allowing us to find the optimal a up to 3 decimal places.

The range where the optimal value of a lies can be found by experimenting with different initial values of a; if the optimal value of a is found to lie in the extreme ends of the range, very likely the optimal value lies outside of it.

The only difference for using the Morse potential is that it cannot be evaluated analytically, so only numerical values will be obtained. The structure of the program is illustrated as shown:


Start
Initialize a
Calculate relative error
Update value of a
Store it in an array
No
Tabulated errors for all a?
Yes
Find the optimal a with least error, and display a.
Evaluate 15 energy levels numerically and display them
End

The optimization and evaluation of the allowed energy levels are done in the morse_molecule.c file.

Compiling and running the program, the following values are obtained:

<u>Optimal $a = 0.256$</u>

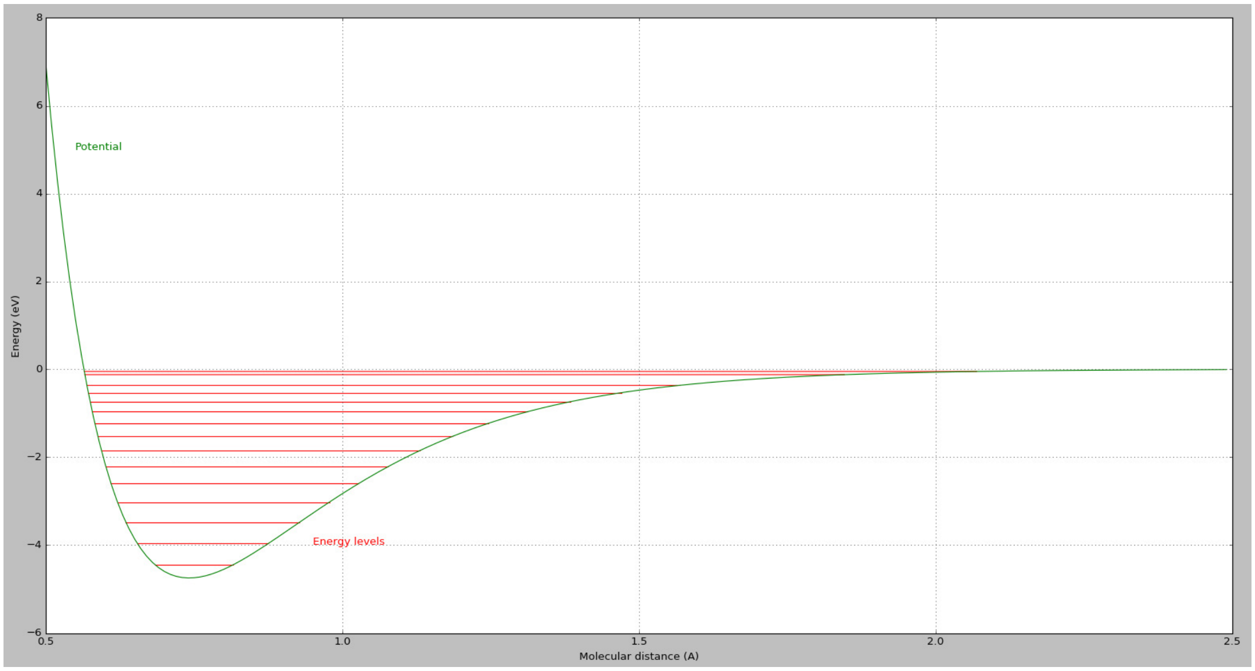
Energy levels

n^{th} Energy level	Numerical (eV)	Actual (eV)
0	-4.477	-4.477
1	-3.960	-3.962
2	-3.476	-3.475
3	-3.023	-3.017
4	-2.601	-2.587
5	-2.211	-2.185
6	-1.853	-1.811
7	-1.526	-1.466
8	-1.231	-1.151
9	-0.968	-0.867
10	-0.736	-0.615
11	-0.536	-0.400
12	-0.368	-0.225
13	-0.126	-0.094
14	-0.053	-0.017

It can been seen that the calculated energy levels from the Morse potential is a huge improvement over the quadratic potential. Nonetheless, up till $n = 5$, the energy levels Morse potential predicted agrees with the measured energy levels well (up to 2 significant figures). Higher than that, the calculated values start to diverge with the measured values.

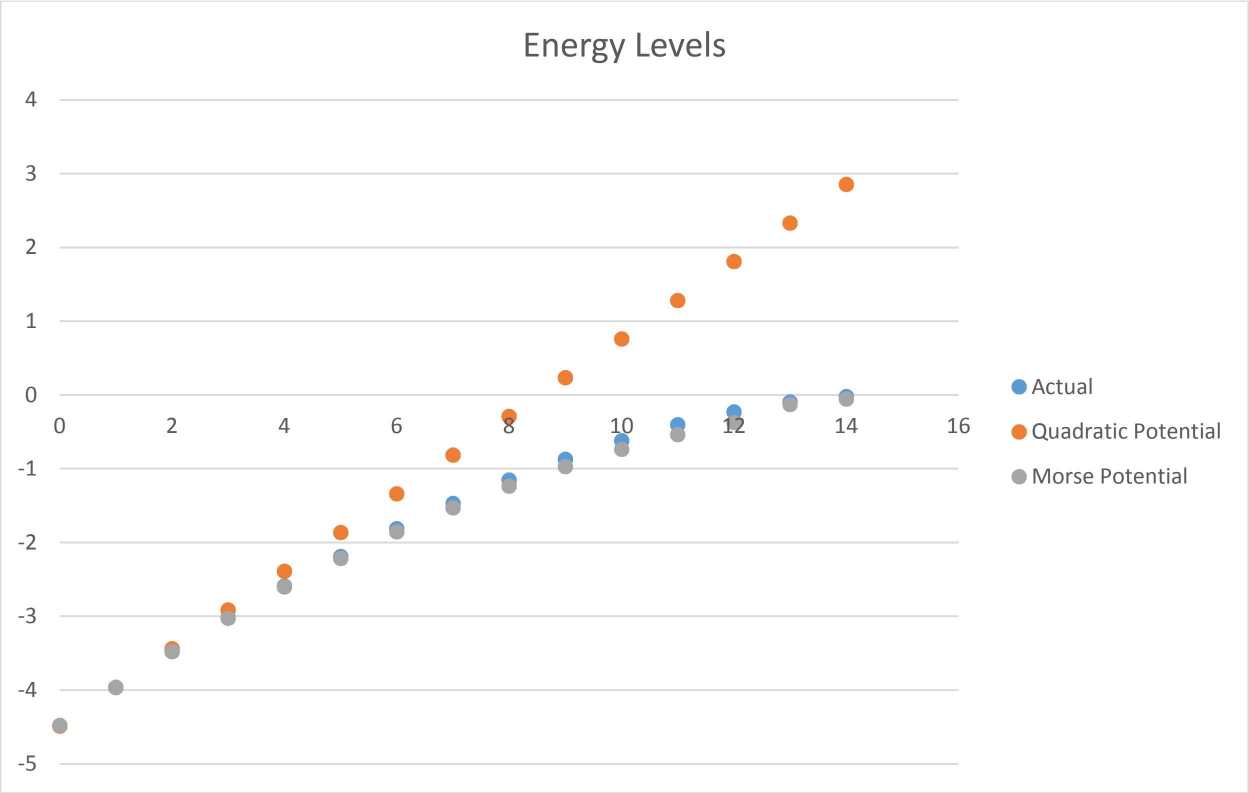
It is also worth noting that all the calculated energy levels are all negative, which is expected as Morse potential takes into account the asymptotic nature of the interaction between molecules.

Once again, the plot of can be obtained in the file *morse_plot.py*:



Analysis of Results and Program

The comparison of the energy level obtained via the quadratic and Morse potential with the actual energy levels can be compared graphically as shown:



For low n, the both potential fits the actual energy well, but the energies predicted by the quadratic potential increases with a linear trend to the positive, while the energy obtain with Morse potential stays true to its asymptotic nature below the x-axis.

It can be concluded that the Morse potential fits the experimental data a lot better, but it does not perfectly describe the actual potential as for n above 7, it starts to diverge away from the actual energy levels; hence a better model for the potential energy will be needed.

For the programs, they are found to work very well, obtaining highly accurate and precise calculations without performance issues. The codes admittedly can be made tidier by having all the functions used, namely *simp_int*, *find_root* and *minimum* placed into one file, and imported into the calculation files using the Makefile command, eliminating the need to include all the functions at the start of every program.

(2195 words)

References

1. Ter Haar, D (1967). *The Old Quantum Theory*. Pergamon Press. p. 206
2. Landau, R.H., PA, M. J., & Bordeianu, C. C. (2015). *Computational Physics: Problem Solving with Python*. John Wiley & Sons.