UFR - Sciences et Techniques, Besancon

Numerical integration of the Schrödinger equation: vibrational energies and wave functions of the H2 molecule

Fortran implementation - Practical report

M1 CompuPhys 2023/2024 - Léo BECHET

Introduction

A notebook containing the python code used for automation and data analysis is available alongside that report. In case you do not see it, please contact me at leo.bechet@edu.univ-fcomte.fr . All the code, necessary file and reports are available on github : https://github.com/lele394/Fortran-Study-of-H2-molecule-vibrational-energies-and-wave-functions-M1-CompuPhys-2023-2024

Preliminary work was done prior to the practical. It included the computation of the parameters a, b and V0. Subsequent calculations revealed a = 1.3924, b = 1.484e-3 and V0 = 40970.35

Command line arguments

Due to the tasks asked during this practical, a modification was introduced to allow the use of command line arguments when launching the program. The way this is done is by adding the following lines inside the fortran program.

```
call get_command_argument(POSITION, arg)
read(arg, *) VARIABLE_TO_CHANGE
```

Where POSITION is an integer specifying the position of the argument and VARIABLE_TO_CHANGE being the name of the variable you wish to assign it to.

Comparing iteration values with Morse Potential

We wish to compare the values found using the fortran program with the values from the Morse potential. In order to do so we first compute the Morse potential and use it to seed the fortran program first guessing value. The program will then converge on its own to a node. We found out that though it starts to diverge rather quickly (see below), the Morse potential can be used as a starting value for the first 18 nodes.

v	theo values	fortran values	delta relatif	ratio	iters	node
0	-0.94708	-0.947366	0.000302	0.999698	2	0
1	-0.845556	-0.846869	0.001552	0.99845	2	1
2	-0.749787	-0.752887	0.004134	0.995883	2	2
3	-0.659772	-0.665169	0.00818	0.991887	2	3
4	-0.575511	-0.583508	0.013896	0.986295	3	4
5	-0.497004	-0.507655	0.021429	0.979021	3	5
6	-0.424252	-0.437477	0.031173	0.969769	4	6
7	-0.357254	-0.372815	0.043558	0.95826	3	7
8	-0.29601	-0.313474	0.058997	0.94429	3	8
9	-0.240521	-0.259391	0.078456	0.927251	5	9

10	-0.190786	-0.210471	0.10318	0.906471	5	10
11	-0.146805	-0.166641	0.135119	0.880965	5	11
12	-0.108578	-0.127866	0.177643	0.849154	4	12
13	-0.076106	-0.094109	0.236557	0.808697	5	13
14	-0.049388	-0.065403	0.324278	0.755129	6	14
15	-0.028424	-0.041781	0.469905	0.680316	7	15
16	-0.013215	-0.023312	0.764103	0.56686	9	16
17	-0.00376	-0.010101	1.686542	0.372226	11	17
18	-5.9e-05	-0.002271	37.533209	0.025952	5	18

To justify the use of the fortran iterative program, we choose to check how much the relative différence between the Morse potential and the fortran potential evolves with each nodes.

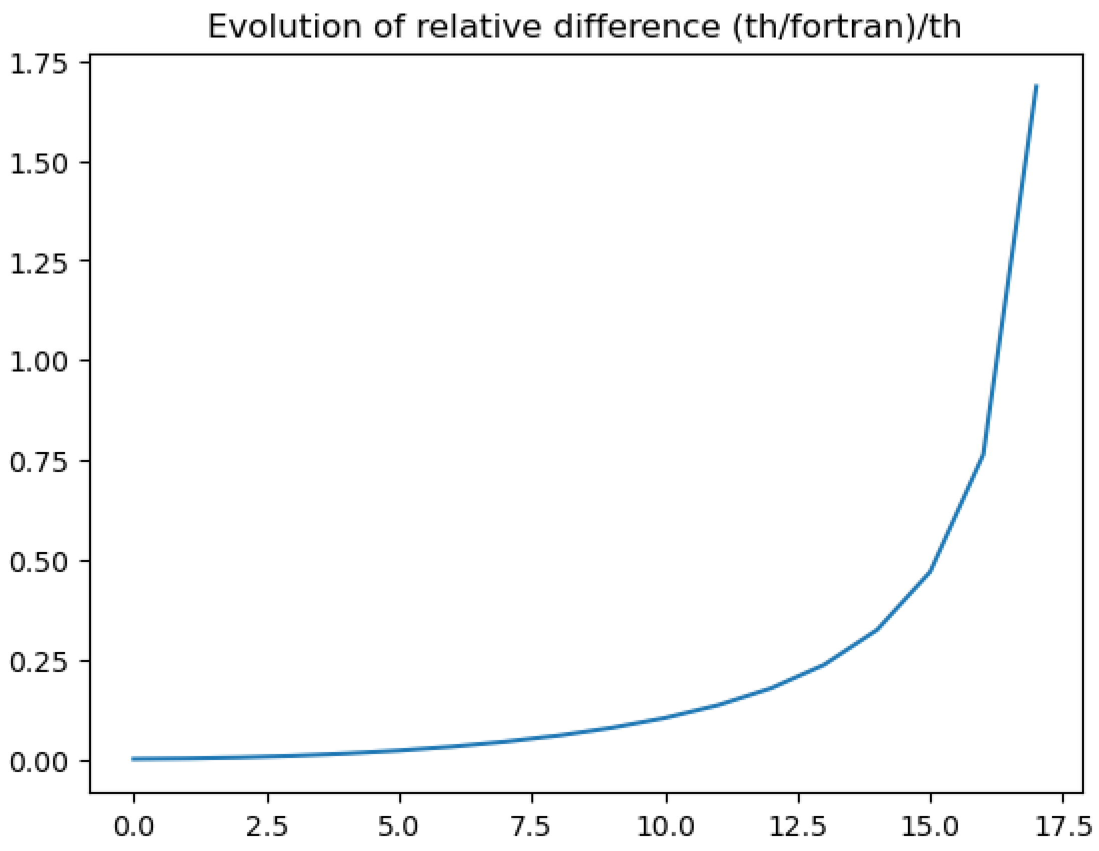

Fig 1. Relative difference between Morse potential and fortran iterative potential.

As one can easily see, the curve tends to grow exponentially. The fortran program is way more precise, which is why we will be using it.

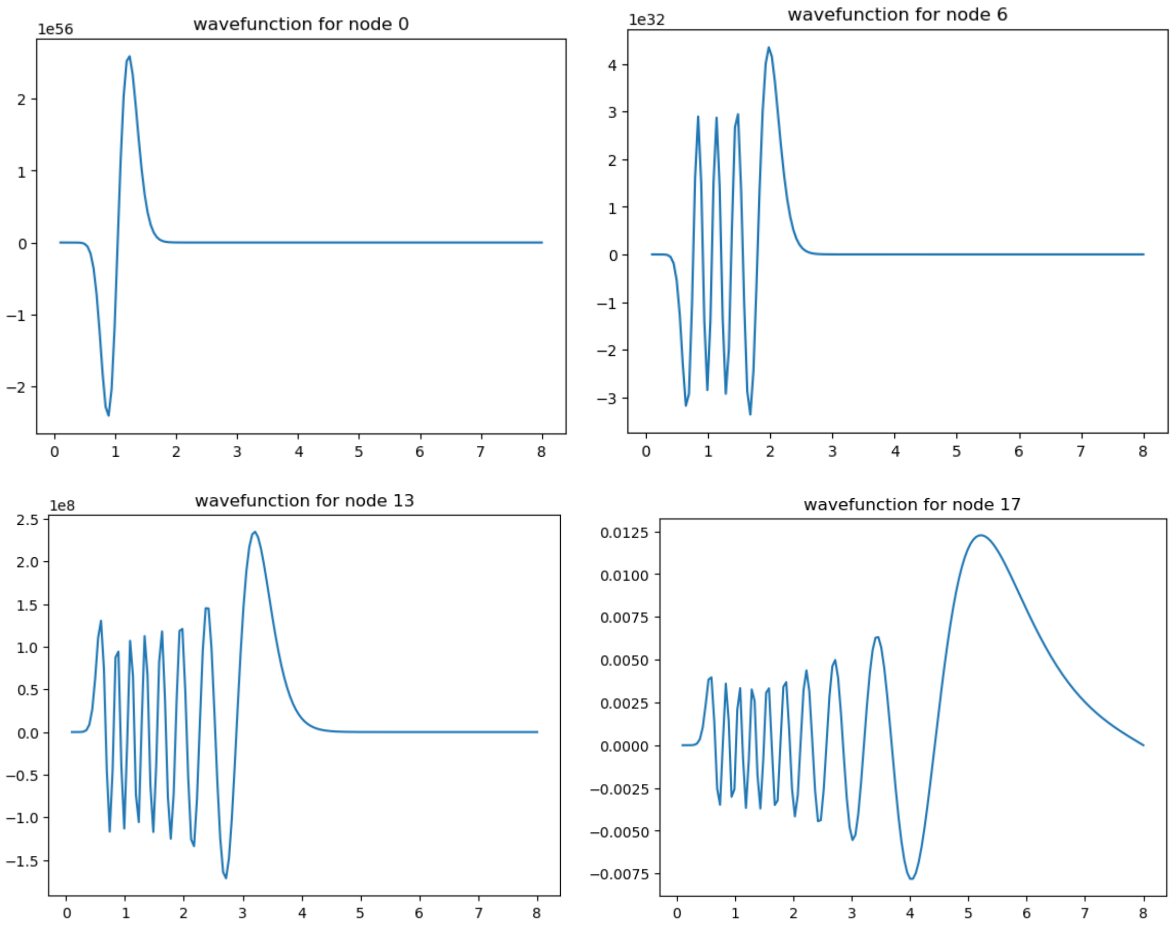

Fig 2. Examples of computed wavefunction

Evolution of iteration number in relation to the distance from a node

We would like to study the evolution of the number of iterations, closely linked to the run time, when changing the distance between the node and the starting value.

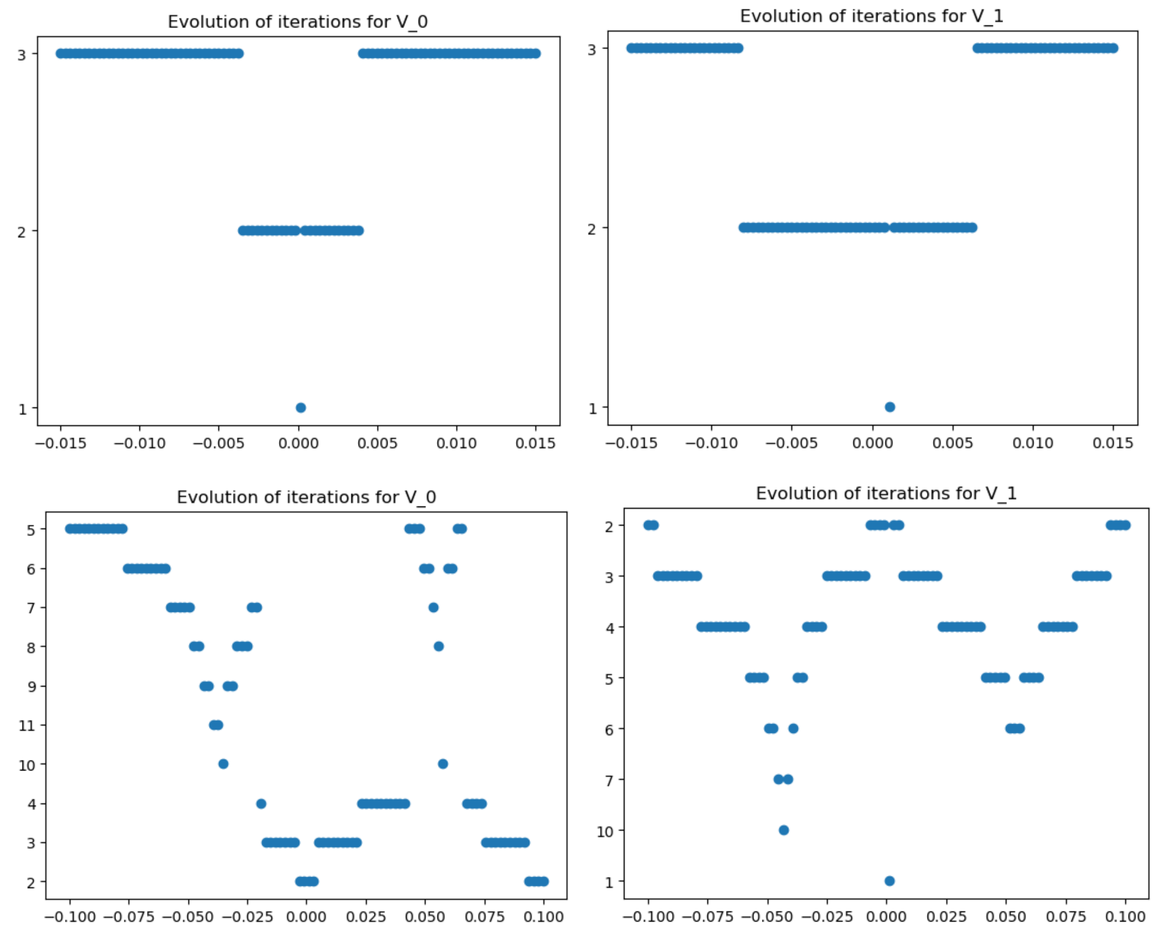

Fig 3. At the top, evolution from -0.015 to +0.015 from the node, at the bottom, evolution from -0.1 to +0.1

We used the values computed by fortran in the previous part to set the relative origin of each graph for both nodes. We can however spot an issue, on the graphs for the node 1, It is not centered. We have rerun the simulation multiple times, to no avail. It seems to be a systematic error that we have no explanation for.

The plot of the second row illustrates that, if the starting value is too high, the program will start to converge onto the next value instead of the one we want to. It underlines the importance to choose the right starting value when iterating for a specified node.

Influence of r0, rN , N, ε and τ

We wish to study the influence of each parameter on the number of iterations. To do so we'll plot graphs showing the evolution of the iterations in respect to the parameter variation. Let's start with N.

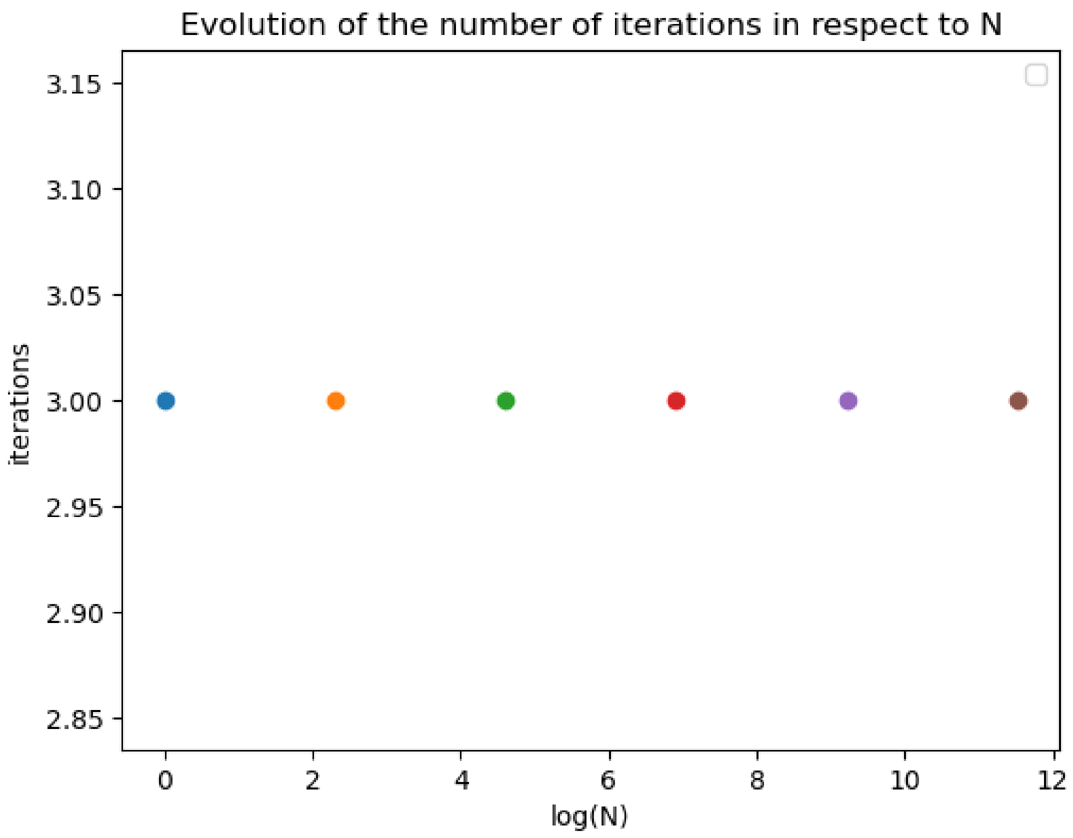

Fig 4. Evolution of iterations depending on N

N does not seem to have any impact on the number of iterations. We tested multiple powers of 10 without noticing any difference.

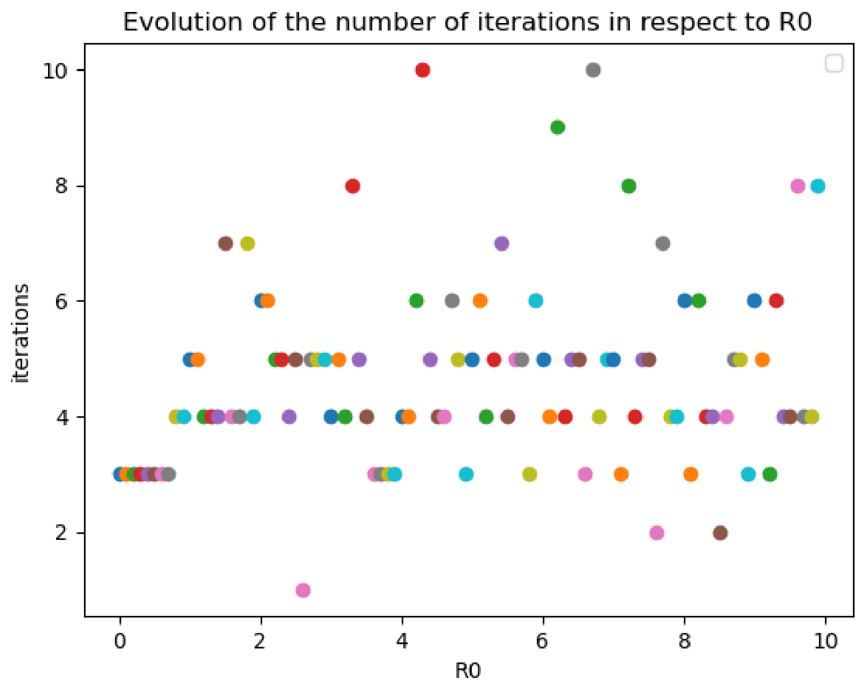

Fig 5. Evolution of iterations depending on R0

However, having R0 varies by a lot the number of iterations, going from 1 to 10. The effect of this variable will be studied later. The number of iterations being dependent on the proximity of the guess value from the real value, the further you are the longer it'll take.

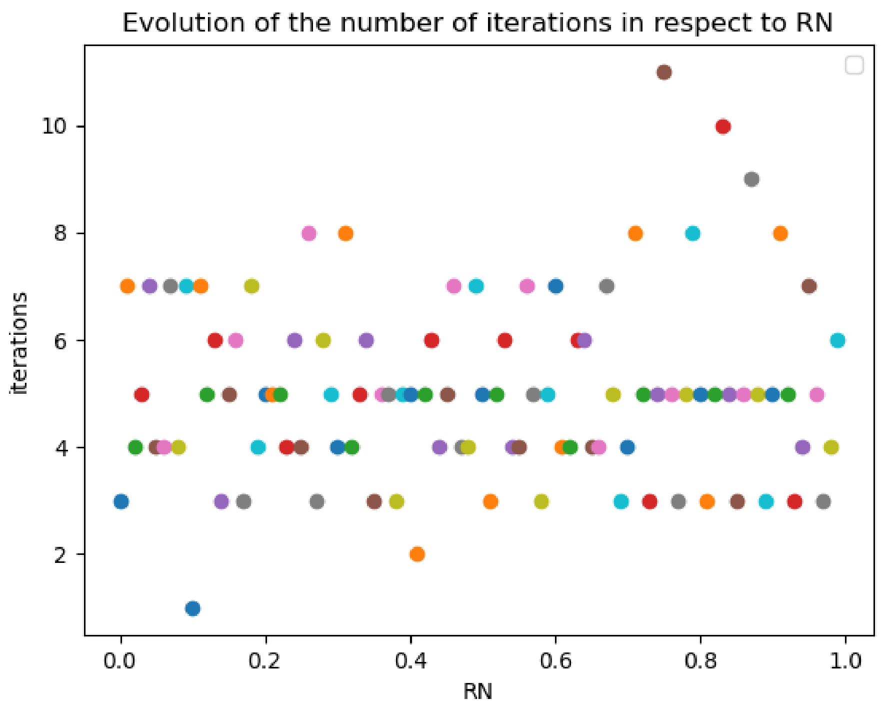

Fig 5. Evolution of iterations depending on RN

RN seems to have a similar effect as R0 on the iteration numbers

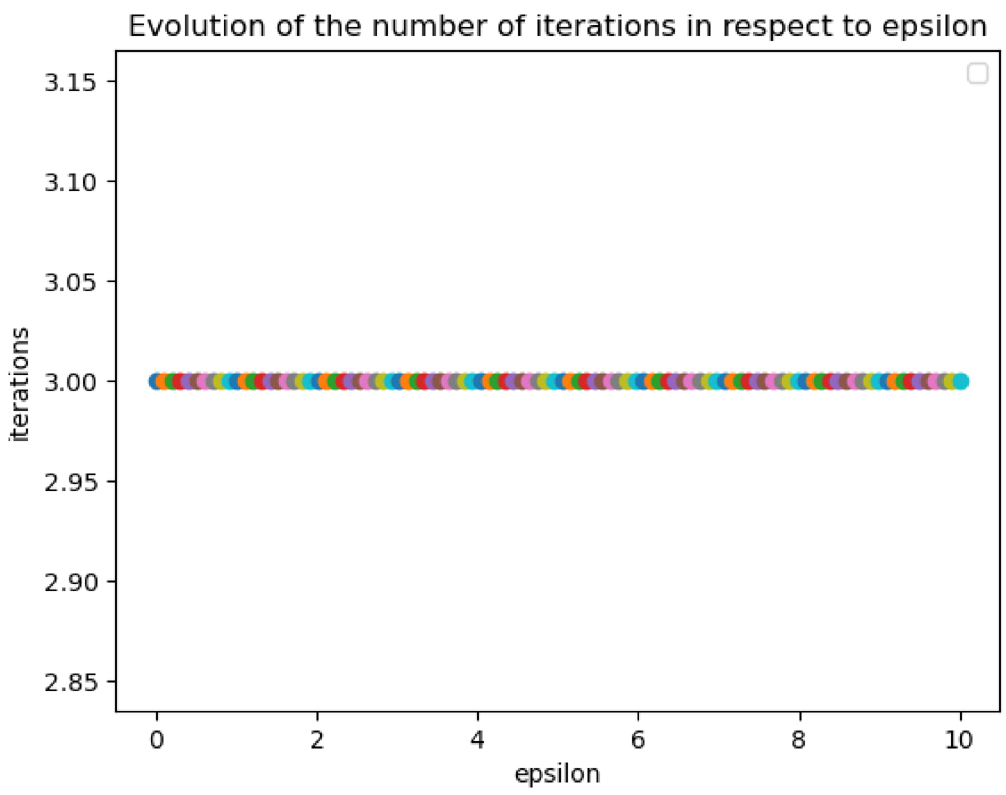

Fig 6. Evolution of iterations depending on ε

ε doesn't seems to have any effect on the iteration numbers

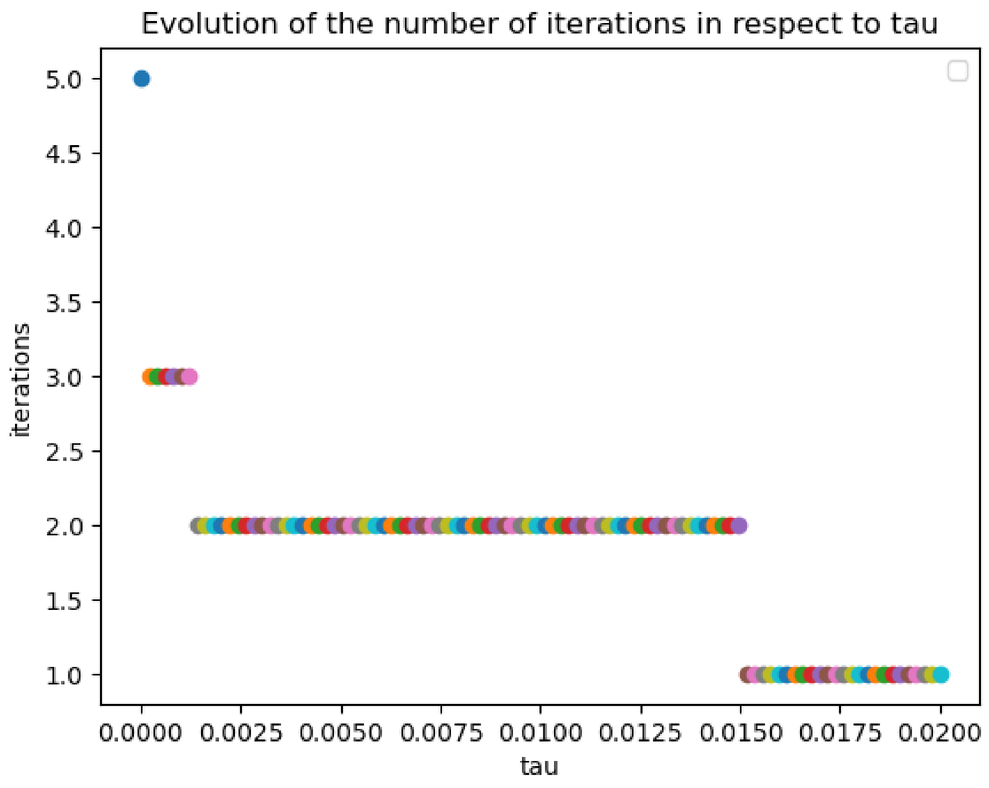

Fig 7. Evolution of iterations depending on τ

τ is tied to the precision of the final result. The program always gets closer but in smaller and smaller steps. The more precise you wish your result to be, the longer the program will take to reach the threshold, thus increasing exponentially the number of iterations.

Evolution of δe

In order to find the evolution of δe we will use a modified version of the code. The fix used removes the existing stop condition by setting the tolerance to 0.0. This effectively removes any possibility of the program stopping. We also modify the increment, by using a fixed incrementation instead of the one used by the original program. This effectively build a list of deltas that can be used in python and plotted.

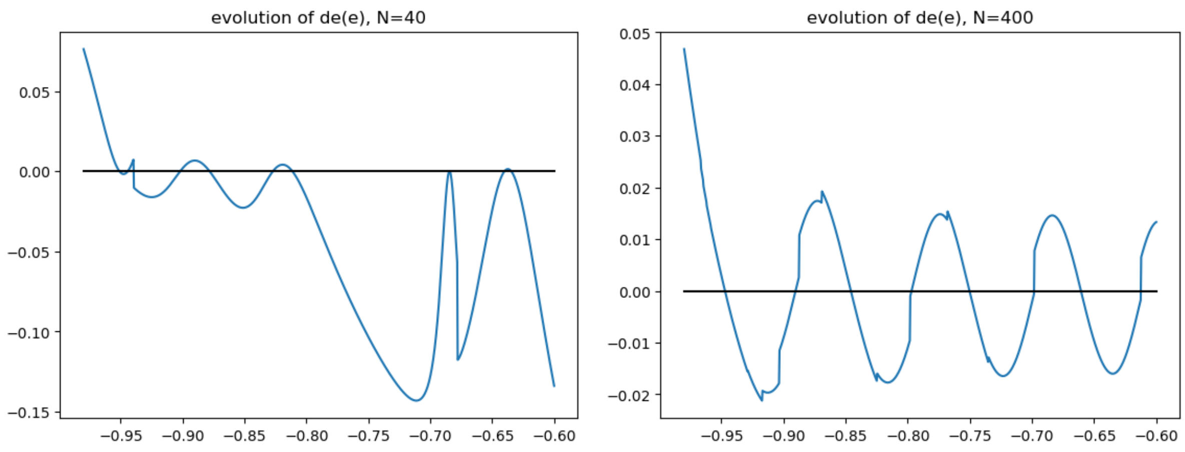

Fig 4. Graph using not enough points

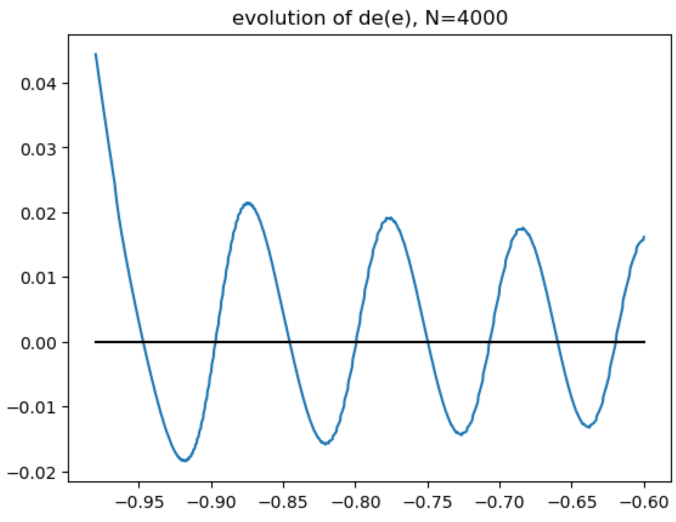

Fig 8. Evolution of δe using 4000 points.

Analyzing the graph, we can confirm that the first 4 eigenvalues are presented and correspond to the computed ones found in a previous section. This however exposes a problem. When choosing a guess value, we have to be aware that if we miss the node we're looking for, the program will converge on the next node, which is not the one we will be looking for.

Morse potential correction using B

When determining the parameters of the morse potential, we used the rigid motor model, which leads to unrealistic results in the position of the lines in the branch Q. In order to correct this approximation, we will now take into account the centrifugal term as a perturbation

```
real :: B = 60.80 ! defined in subject

psi_square = 0
do i=1, length
   psi_square = psi_square + (psi(i)**2)
end do

do i=1, length
   phi(i) = psi(i)/sqrt(psi_square)
end do

sum = 0
do i = 1, length
   sum = sum + phi(i)**2/position(i)**2
end do
print *, sum*B
```

The code snippet above allows us to compute the corrected B value using the given B in the subject. Psi is the list of positions on the Y axis of the wavefunction; phi contains psi squared. To compute B0 and B1 we will use the wave functions computed in question 2. Running this program for the first 2 wave functions gives us newly corrected values for B.

```
Computed number for B0 is 59.8182182
Computed number for B1 is 57.7194748
```

Comparison to Schrodinger's equation

In order to compare with the Schrodinger equation, we isolate B from the given equation.

$$E_{vj}^0 = E_v + BJ(J+1) \Leftrightarrow B = \frac{E_{vj}^0 - E_v}{J(J+1)}$$

We transform the potential formula in the program to be :

```
V(I)=DEXP(-2.D0*A*(RR(I)-1.D0))-2.D0*DEXP(-A*(RR(I)-1.D0)) +
B*(JLEV*(JLEV+1))/RR(I)**2
```

Where the second part accounts for the centrifugal term. This equation is implemented in the Notebook and then used for the calculations of new B0. The result of our computations gives us :

```
v=0 J=1, B: -59.669217740000995
v=0 J=2, B: -59.87338665083376
v=0 J=3, B: -59.288876324166964
v=0 J=4, B: -58.989724485249994
v=0 J=5, B: -58.43150346650005

v=1 J=1, B: -57.3339077899998
v=1 J=2, B: -57.36122135666649
v=1 J=3, B: -57.34995451041673
v=1 J=4, B: -56.80211264699997
v=1 J=5, B: -56.25625101716663
```

As we can see, the values stays quite precise for the first 2 J, but quickly diverge from the results of the previous questions. This can also be shown by graphing the error between both values.

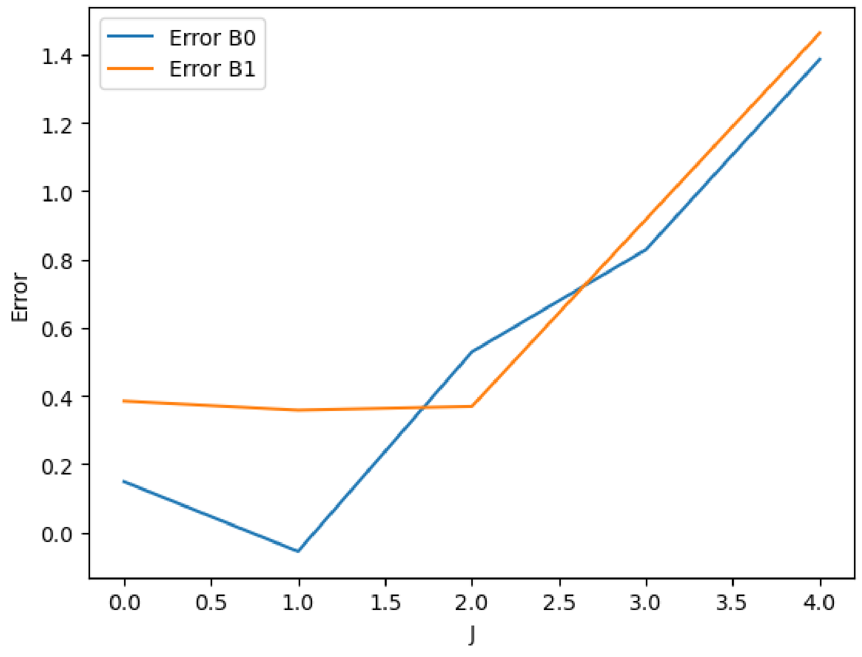

Fig 9. Error between computed B0 and B1 with different J

As we can see, the curves tend to ramp up pretty early, thus exposing the issue arising with the precision of the computed values.

Conclusion

The Morse potential approximation was tested against an iterative method. It appears that the Morse method is effective only for the few first nodes. However it quickly diverges from the real solution. We also discovered that using the Morse potential as an estimator for the guess value of the iterative program was a viable option for at least the first 20 nodes. A correction was added by taking into account the centrifugal term, but no further testing was realized.

We would like to indicate that in order to simplify subsequent calculations of node properties, the creation of a database of previously computed values for each node is recommended. Those values could be used as starting guess values for the iterative approach, when more precise calculations are needed.