Applications of the Numerov method to simple quantum systems using Python

Francisco Caruso*[1,2] ⓘ, Vitor Oguri[2] ⓘ, Felipe Silveira[2] ⓘ

[1]Centro Brasileiro de Pesquisas Físicas, Coordenação de Física de Altas Energias, Rio de Janeiro, RJ, Brasil.
[2]Universidade do Estado do Rio de Janeiro, Instituto de Física Armando Dias Tavares, Rio de Janeiro, RJ, Brasil.

Numerov's numerical method is developed in a didactic way by using Python in its *Jupyter Notebook* version 6.0.3 for three different quantum physical systems: the hydrogen atom, a molecule governed by the Morse potential and a quantum dot. After a brief introduction to the Numerov method, the complete code to calculate the eigenfunctions and eigenvalues of the hydrogen atom is presented. The necessary code changes to compute the other two examples are also provided in the sequel.

Keywords: Hydrogen Atom, Morse Potential, Quantum Dot, Numerov method, Python.

1. Introduction

In the last decades, the teaching of computing techniques has become more present and also increasingly essential in the development of students from all areas, and, therefore, it would not be different for physics teaching. Having said that, it is of paramount importance that we always produce new teaching materials for new technologies such as the Python programming language, which, despite of being relatively new, has already dominated the market becoming one of the most important languages today.

We will disclose here a powerful numerical calculation method originally developed by Boris Vasil'evich Numerov [1–3] – see also [4–7] – applying it to the time-independent Schrödinger equation describing physical systems like the hydrogen atom, a diatomic molecule governed by the Morse potential and a particular model for the quantum dot atom [8–10]. These three examples will be solved and the parameters needed for each solution using Numerov's method will be shared in their respective sections.

In summary, this work aims to provide the complete code developed in Python with the *Jupyter Notebook* for Numerov's numerical method. However, it is important to emphasize that we do not aim to teach Python to the reader, who must have a basic knowledge of programming to be able to keep up with the examples.

The code used in this article can be found in full and downloaded at the following link: https://1drv.ms/u/s!Ai_Lqkgh1kiskp5_cfOtwCfqX-LpTw?e=srdmzS, as well as in our repository on Github https://github.com/Felipecrdss/Numerov.

---

* Correspondence email address: francisco.caruso@gmail.com

2. Numerov's Method

Numerov's initial motivation was to be able to calculate corrections to the trajectory of Halley's comet. Therefore, Numerov's method was initially developed to determine solutions to eigenvalue problems associated with second order ordinary differential equations of celestial mechanics, which did not contain terms involving the first derivative of an unknown function $y(x)$, that is, equations of the form

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = f(y, x). \tag{1}$$

Every differential equation equal to equation (1) can be replaced by the following system of first order equations

$$\begin{cases} \dfrac{\mathrm{d}z}{\mathrm{d}x} = f(x, y), \\ z = \dfrac{\mathrm{d}y}{\mathrm{d}x}. \end{cases}$$

Traditional methods for numerically solving this system of equations, such as those of Euler or Runge-Kutta, consider that the values of $y(x)$ and of $\mathrm{d}y/\mathrm{d}x$ are known at a given point in the domain $[a, b]$ of system validity, *i.e.*, are suitable for the so-called seed problems.

In non-relativistic quantum mechanics, more specifically in bound state problems involving a particle of mass m confined in a well of potential $V(x)$, in a given interval $a < x < b$, the allowed energies (E) and the corresponding wave functions $\psi(x)$ that describe these steady states satisfy the Schrödinger's eigenvalue equation

$$\frac{\mathrm{d}^2 \psi}{\mathrm{d}x^2} + k^2(x)\psi = 0 \tag{2}$$

where $k = \sqrt{2m[E - V(x)]}/\hbar$ and $\hbar \simeq 1.055 \times 10^{-34} J.s$ is the reduced Planck constant.

In these cases, as the value of the first derivative of the wave function is not known, the Euler and Runge-Kutta cannot be employed. Nonetheless, it is possible to establish continuity conditions for the values of ψ and $d\psi/dx$ at two or more points of the domain of the wave function, which characterizes the so-called boundary value problems.

In addition, making the transformation of a second order differential equation in a first order system, the Numerov method allows the simultaneous determination of the particle energy spectrum and the eigenfunctions associated with each energy value.

Like any iterative numerical method, the solution of equation (2) is constructed by successive integrations. In Numerov's method, initially, the solution is considered to be known at two subsequent points of the interval $[a, b]$; for example, at $\psi(x - \delta)$ and $\psi(x)$, where δ is an arbitrarily small quantity, called the integration step. Next, we try to establish an algorithm to determine the solution at the next point, $\psi(x + \delta)$.

The starting point for establishing this algorithm is the expansion of $\psi(x \pm \delta)$ in Taylor series, up to fourth-order derivatives, that is,

$$\psi(x \pm \delta) = \psi(x) \pm \delta\psi'(x)$$
$$+ \frac{\delta^2}{2}\psi''(x) \pm \frac{\delta^3}{6}\psi'''(x)$$
$$+ \frac{\delta^4}{24}\psi^{iv}(x) \tag{3}$$

Adding the terms $\psi(x + \delta)$ and $\psi(x - \delta)$, only the derivatives of even order survive and, therefore, a relationship between the values of a function in three points and its second derivative is reached, given by

$$\frac{\psi(x + \delta) + \psi(x - \delta) - 2\psi(x)}{\delta^2}$$
$$= \psi''(x) + \frac{\delta^2}{12}\psi^{iv}(x)$$
$$\equiv \left(1 + \frac{\delta^2}{12}\frac{d^2}{dx^2}\right)\psi''x \tag{4}$$

Writing the unidimensional Schrödinger equation, equation (2), in a more convenient form

$$\left(1 + \frac{\delta^2}{12}\frac{d^2}{dx^2}\right)\psi''(x) = -k^2(x)\psi(x)$$
$$- \frac{\delta^2}{12}\frac{d^2}{dx^2}\left[k^2(x)\psi(x)\right] \tag{5}$$

and using equation (4) to replace the terms that contain second order derivatives, one obtains

$$\frac{\psi(x + \delta) + \psi(x - \delta) - 2\psi(x)}{\delta^2}$$

$$= -k^2(x)\psi(x) - \frac{\delta^2}{12}$$
$$\times \left[\frac{k^2(x + \delta)\psi(x + \delta) + k^2(x - \delta)\psi(x - \delta) - 2k^2(x)\psi(x)}{\delta^2}\right] + \mathcal{O}(\delta^4) \tag{6}$$

Regrouping the terms, we obtain the Numerov difference formula for the problem of a particle under action of a one-dimensional potential:

$$\left[1 + \frac{h^2}{12}k^2(x + \delta)\right]\psi(x + \delta)$$
$$= 2\left[1 - \frac{5\delta^2}{12}k^2(x)\right]\psi(x)$$
$$- \left[1 + \frac{\delta^2}{12}k^2(x - \delta)\right]\psi(x - \delta) \tag{7}$$

Since the problem of interest is an eigenvalue problem, the numerical integration technique of the one-dimensional Schrödinger equation for a particle in a well depends on attaching arbitrary values conveniently to eigenvalues and the respective (possible) eigenfunctions in 2 points of the domain of the problem. But how to do it? Regarding the choice of the initial value for the energy (first eigenvalue), just remember that, according to Heisenberg's uncertainty relation, the energy E of a particle in a well of potential $V(x)$ must be greater than the minimum value of the well. Thus, it is considered, initially, that $E_{initial} = V_{min} + \Delta E$, with $\Delta E > 0$.

The choice of an energy value, determines two turning points, x_ℓ and x_r, where the energy value is equal to the potential energy value, whose motion obeys the classical Newtonian mechanics. That is, from the point of view of classical mechanics, the movement of the particle is restricted only to the region $[x_\ell, x_r]$, in which the energy is greater than or equal to the potential energy. The regions $x < x_\ell$ and $x > x_r$ are called classically prohibited regions, and are indicated in Figure 1.

As the Schrödinger equation admits solutions for these classically prohibited regions, for each energy value, initially, values are assigned to a possible eigenfunction at two points of the classically prohibited regions, in which the function practically cancels itself. In general, these are the boundary points a and $b > a$ of the function's integration domain.

However, the implementation of Numerov's method to solve the problem still requires an iteration scheme that uses the Numerov formula in two steps: from a, or to the left of a from the classic turning points, hereinafter called match point (x_{match}), and from b, or to the right of the match point.

Thus, arbitrarily taking an initial value for the energy, and two successive arbitrary values for the solution, starting from the lower extremes and upper part of
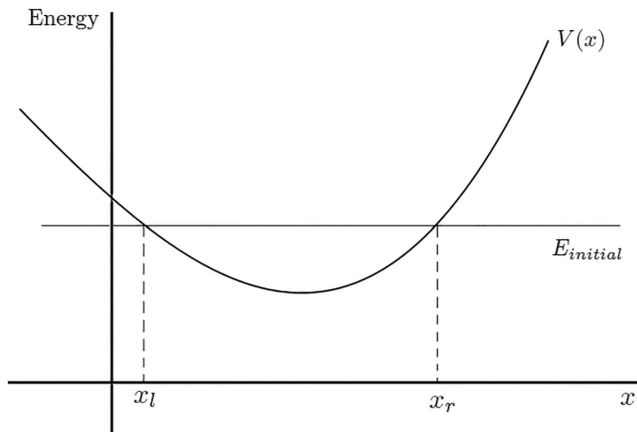
Figure 1: Meeting points between the potential curve and the initial energy, also called turning points.

the integration interval $[a, b]$, one can implement the method's iteration scheme in the two senses, such as:

- Solution to the left of match point $(x < x_{match})$

Being $E_{initial} = V_{min} + \Delta E \left(\frac{\Delta E}{|V_{min}|} \ll 1 \right)$ an arbitrary value for the energy of the particle. Also arbitrating values for the wave function, in two successive points, from a,

$$\begin{cases} \psi^\ell(a) = 0, \\ \psi^\ell(a + \delta) = \delta^\ell, \quad (\delta^\ell \ll 1) \end{cases}$$

and using the formula of differences, equation (7), the solution on the left is built sequentially until match point (x_{match}), in what $\psi^\ell(x_{match}) = \psi^\ell_{match}$.

- Solution to the right of match point $(x > x_{match})$

In a similar way, for the same values $E_{initial}$, arbitrating

$$\begin{cases} \psi^r(b) = 0, \\ \psi^r(b + \delta) = \delta^r, \quad (\delta^r \ll 1) \end{cases}$$

the solution to the right, from b, is constituted sequentially until the points x_{match} and $x = x_{match} - \delta$, as

$$\begin{cases} \psi^r(x_{match}) = \psi^r_{match}, \\ \psi^r(x_{match} - \delta) = \psi^r_{match-1}, \quad (\delta^r \ll 1) \end{cases}$$

To guarantee the boundary condition of the solution, we redefine the solution to the left according to equation (8) given below, and the boundary condition of the first derivatives, according to equation (9).

The procedure is repeated step by step, in the two ways, $a \rightleftharpoons b$. Starting from a, using the Numerov recurrence formula associated with an equation, if we build the solution ψ^ℓ until the classic rewind point, nearest of b, where $E = V(x_{match})$, called the match

point. Then, from b, the analog is made, building a solution ψ^r to the match point. In principle, the possible solutions ψ^ℓ and ψ^r will not necessarily be equal in this x_{match} stitch. To ensure the continuity of the solution redefines itself ψ^ℓ like

$$\psi^\ell(x) \rightarrow \psi^\ell(x) \frac{\psi^r(x_{match})}{\psi^\ell(x_{match})} \quad (a \le x \le x_{match}) \qquad (8)$$

Finally, it is verified how close are the values of the respective first derivatives of ψ^r and the new function ψ^ℓ It is staggered, at match point. To test the boundary condition of the derivatives first, taking into account Taylor's series for $\psi(x + \delta$ and $\psi(x - \delta)$, up to the first order, you can write

$$\begin{cases} \left. \dfrac{d\psi^\ell}{dx} \right|_{x_r} = \dfrac{\psi^\ell_{match+1} - \psi^\ell_{match-1}}{2\delta}, \\ \left. \dfrac{d\psi^r}{dx} \right|_{x_r} = \dfrac{\psi^r_{match+1} - \psi^r_{match-1}}{2\delta}, \end{cases} \qquad (9)$$

in what, $\psi_{match\pm1} = \psi(x_{match\pm\delta})$.

If the difference between these values is less than the values of a predefined error, the process is interrupted, confirming the searched eigenvalue and the respective eigenfunction as being

$$\begin{cases} \psi^\ell(x), \quad (a \le x < x_{match}) \\ \psi^r(x), \quad (x_{match} \le x \le b) \end{cases}$$

If the continuity condition of the derivatives is not satisfied, the value of the energy is increased and we restarted a search for a new value which is really an eigenvalue of the problem, and its respective eigenfunction. The process can be repeated until the desired number of eigenvalues and eigenfunctions of the problem.

Because it is based on Taylor's serial expansion to the fourth order, the error in Numerov's method is much smaller than the errors that come out from the expansion-based methods in lower order, like that of Runge-Kutta.

3. Hydrogen Atom

Numerical solutions of hydrogen atom were previously obtained in [1] using Numerov's method. The program was written in C++ for the ROOT cint compiler.

Although it was originally developed for second order linear and homogeneous ordinary differential equations that do not contain terms of the first derivative, the Numerov's method can be generalized to cover the presence of terms that contain the first derivative in the differential equation, so that eigenvalue problems can also be considered.

In fact, in the case of linear equations, every second order differential equation of the type

$$\frac{d^2y}{dx^2} + P(x)\frac{dy}{dx} + Q(x)y = 0$$

can be written in its normal form

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + q(x)y = 0$$

where

$$q(x) = Q(x) - \frac{1}{4}P^2(x) - \frac{1}{2}\frac{\mathrm{d}P}{\mathrm{d}x}$$

Schrödinger's radial equation for a particle of mass m under the action of a Coulombic electric field, like the electron in the hydrogen atom, can be written as

$$\frac{\mathrm{d}^2 R(r)}{\mathrm{d}r^2} + \frac{2}{r}\frac{\mathrm{d}R(r)}{\mathrm{d}r}$$
$$+ \frac{2m}{\hbar^2}\left[E + \frac{e^2}{r} - \frac{\hbar^2}{2m}\frac{\ell(\ell+1)}{r^2}\right]R(r) = 0 \quad (10)$$

Making the substitution $r = xa_B$ with $a_B = \hbar^2/(me^2)$ being the Bohr radius, equation (10) can be rewritten, for a new function $y(x) = R(r)$, as

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -\frac{2}{x}\frac{\mathrm{d}y}{\mathrm{d}x} - [\varepsilon - V(x)]\,y(x) \quad (11)$$

where, $\varepsilon = \frac{E}{e^2/(2a_B)}$ and $V(x) = \frac{\ell(\ell+1)}{x^2} - \frac{2}{x}$ are, respectively, the energy and the so-called effective potential in atomic units. So, in possession of equation (11), we can start building our program code.

First of all, we must import the functions matplotlib.pyplot (for plotting) and NumPy (for Mathematics and working with arrays); also, to avoid any mistakes, we need to import the function random. We then declare the effective potential $V(x)$. During the construction of all the examples we will use the value $\ell = 0$.

```
import matplotlib.pyplot  as plt
import numpy  as np
from numpy  import random
from random  import random


x = linspace(-10,10,1001)
 def V(x):
    L=0
    Vx=(L*(L+1))/(x*x)-2./x
     return  Vx
```

```
def vec_max(dim,x):              #left maximum
    xmax=0
    N=dim
    for j in range(int(N)):
        if j<int(N) and abs(x[j])>xmax:
                xmax=abs(x[j])
        else:
                continue
    return xmax
```

The equation, in this case, that is intended to be solved by Numerov's method presents a term involving

the first derivative, and can be expressed by

$$\psi''(x) = -p(x)\psi'(x) - s(x)\psi(x), \quad (12)$$

where

$$\begin{cases} p(x) = \dfrac{2}{x} \quad \Rightarrow \quad p'(x) = -\dfrac{2}{x^2}, \\ s(x) = \varepsilon - V(x). \end{cases}$$

From the Taylor expansion, equation (3), we can rewrite equation (12) as

$$\left(1 + \frac{\delta^2}{12}\frac{\mathrm{d}^2}{\mathrm{d}x^2}\right)\psi''(x) = -p(x)\psi'(x) - s(x)\psi(x)$$
$$- \frac{\delta^2}{12}\frac{\mathrm{d}^2}{\mathrm{d}x^2}[p(x)\psi'(x) + s(x)\psi(x)]. \quad (13)$$

In a similar way to the previous case, according to equation (4), you can write the term on the right side of the equation (13) which contains derivatives of order 2, such as

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}[p(x)\psi'(x) + s(x)\psi(x)]$$
$$= \frac{1}{\delta^2}[p(x+\delta)\psi'(x+\delta) + s(x+\delta)\psi(x+\delta)$$
$$+ p(x-\delta)\psi'(x-\delta) + s(x-\delta)\psi(x-\delta)$$
$$+ - 2p(x)\psi'(x) - 2s(x)\psi(x)]$$

Replacing first order derivatives with approximations

$$\begin{cases} \psi'(x) = [\psi(x+\delta) - \psi(x-\delta)]/(2\delta), \\ \psi'(x+\delta) = [\psi(x+\delta) - \psi(x)]/(\delta), \\ \psi'(x-\delta) = [\psi(x) - \psi(x-\delta)]/(\delta), \end{cases}$$

we obtain

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}[p(x)\psi'(x) + s(x)\psi(x)]$$
$$= \frac{1}{\delta^2}\left\{\left[\frac{p(x+\delta) - p(x)}{\delta} + s(x+\delta)\right] \times \psi(x+\delta)\right.$$
$$+ \left[\frac{p(x) - p(x-\delta)}{\delta} + s(x-\delta)\right]\psi(x-\delta)$$
$$\left. + 2\left[\frac{p(x-\delta) - p(x+\delta)}{2\delta} + s(x)\right]\psi(x)\right\}$$

or

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}[p(x)\psi'(x) + s(x)\psi(x)]$$
$$= \frac{1}{\delta^2}\left\{[p'(x) + s(x+\delta)]\psi(x+\delta)\right.$$
$$+ [p'(x) + s(x-\delta)]\psi(x-\delta)$$
$$\left. - 2[p'(x) + s(x)]\psi(x)\right\} \quad (14)$$

Taking into account that the left side of the equation (13) is equal to

$$[\psi(x+\delta)+\psi(x-\delta)-2\psi(x)]/\delta^2$$

we can write

$$\frac{\psi(x+\delta)+\psi(x-\delta)-2\psi(x)}{\delta^2}$$
$$= -p(x)\left[\frac{\psi(x+\delta)-\psi(x-\delta)}{2\delta}\right]$$
$$-s(x)\psi(x)+\frac{1}{12}\left[p'(x)+s(x+\delta)\right]\psi(x+\delta)$$
$$-\frac{1}{12}\left[p'(x)+s(x-\delta)\right]\psi(x-\delta)$$
$$+\frac{1}{6}\left[p'(x)+s(x)\right]\psi(x)$$

Regrouping the terms, and making

$$\begin{cases}\psi(x-\delta)=\psi_0,\\ r\psi(x)=\psi_1,\\ \psi(x+\delta)=\psi_2,\end{cases}$$

one obtains the Numerov difference equation for the problem, suitable for the propagation of the solution from the integration limits interval:

$$\psi_2 = \frac{2\left\{1-\left[s(X)-\frac{p'(x)}{5}\right]\frac{5\delta^2}{12}\right\}\psi_1}{\left\{1+p(x)\frac{\delta}{2}+[s(x+\delta)+p'(x)]\frac{\delta}{12}\right\}} \\ \frac{-\left\{1-p(x)\frac{\delta}{2}+[s(x-\delta)+p'(x)]\frac{\delta^2}{12}\right\}\psi_0}{\left\{1+p(x)\frac{\delta}{2}+[s(x+\delta)+p'(x)]\frac{\delta}{12}\right\}} \tag{15}$$

From this formula, a procedure analogous to the previous case can be implemented for the construction of solutions of the radial Schrödinger equation in the interval $(0,\infty)$.

Now, in order to introduce the Numerov difference formula (15), we first need to insert equation (7) in our code, for that, let's break it down into different pieces p_0, p_1 and p_2, with $h = \delta$, $q_0 = s(x)$. Thus, equation (15) is now called y_2, where $\psi_i = y_i$.

```python
def nrovl(y0, y1, x0, E, h, iflag):
    q0   =  (E-V(x0))
    q1   =  (E-V(x0+h))
    q2   =  (E-V(x0+h+h))
    p0   =  (1 + h*h*q0 /12)
    p1   =   2*(1 - 5*h*h*q1 /12)
    p2   =    1 + h*h*q2 /12
    y2   =  (p1*y1-p0*y0) /p2
    if iflag<1:
        print(" x0 = ", x0," y0 = ", y0," V = ",V(x0))
        print(" x1 = ", x0+h," y1 = ",y1," V = ",V(x0+h),
        " y2 = ",y2)
    return  y2
```

And, for the case $x - \delta$, we repeat the previous step, changing the appropriate sign.

```python
def nrovr(y0, y1, x0, E, h, iflag):
    q0   =  (E-V(x0))
    q1   =  (E-V(x0-h))
    q2   =  (E-V(x0-h-h))
    p0   =  (1 + h*h*q0 /12)
    p1   =   2*(1 - 5*h*h*q1 /12)
    p2   =    1 + h*h*q2 /12
    y2   =  (p1*y1-p0*y0) /p2
    if iflag<1:
        print(" x_100 = ", x0," y0_100 = ", y0," V = ",V(x0))
        print(" x_99 = ", x0-h," y_99 = ",y1," V = ",V(x0-h),
        " y_98 = ",y2)
    return  y2

def espectro(xl,xu,h,delta,eps,dim,nmax,kmax,Ein,Vmax,dE,iflag):
```

We now create a list for each variable up to the value of *dim* or *nmax* which will also be defined in a future step.

```python
xx=list(range(dim));        yy=list(range(dim))
yl=list(range(dim));        yr=list(range(dim))
ee=list(range(nmax));       ff=list(range(nmax))
ff2=list(range(nmax));      yy1=list(range(dim))
yy2=list(range(dim));       yy3=list(range(dim))
nk=list(range(nmax));

E_old = Ein;    E = Ein + dE
```

In the next steps, the program will determine the interaction for the eigenvalue candidates and determine their solutions and will print the parameters found on the screen.

```python
for M in range(nmax):
    print("  *******   Eigenvalue  #",M+1,"  ******* ")

    f_old=0
    for k in range(kmax): #iteration for eigenvalue candidate
        imatch=0

    for j in range(dim-1):  #classical right turning point
            xx[0]=xl;           xx[dim-1]=xu
            DE1 = E - V(xx[j])
            xx[j+1] = xx[j]+h
            DE2 = E - V(xx[j+1])
            D1D2=DE1*DE2
            if D1D2<=0  and DE1 > 0:  #match point
                imatch = j+1
                print(" imatch = ",imatch," xmatch = %2.3f"
xmatch=xx[imatch]
        ii=range(imatch+2)
        i_lim = ii[2:imatch+2]

        xx[0]=xl;  xx[1]=xl+h   #initial values
        yy[0]=0;   yy[1] = delta
```

```python
        for i in i_lim:      #numerov left solution
            yy[i]=nrovl(yy[i-2],yy[i-1],xx[i-2],E,h,iflag)
            xx[i]= xx[i-1]+h

        jjj=list(range(dim+1))
        j_lim = list(jjj[imatch-1:dim+1])
        comp_j=len(j_lim)
        jj=sorted(j_lim,key=abs,reverse=True)

        for i in range(dim):
            if i<=imatch+1:
                yl[i]=yy[i]
            if i>imatch+1:
                yl[i]=0

        for i in jj:        #numerov right solution
            if i==dim:
                yr[dim-1]=0
```

```
        if i==(dim-2):
            yr[dim-2]=2*delta
        if i<(dim-2):
            yr[i]=nrovr(yr[i+2],yr[i+1],xx[i+2],E,h,
            iflag)
            xx[i]= xx[i+1]-h

    for i in range(imatch-1):
        yr[i]=0

    ymatch=yy[imatch]
    yrmatch=yr[imatch]
    ylmatch=yl[imatch]

    if ymatch != 0:
        scale=yrmatch /ymatch
    else:
        continue
```

```
    for t in range(imatch+1):              #  y_left
        yy[t] = yy[t]*scale
        yl[t] = -yl[t]*scale
    yl[imatch+1]=-yl[imatch+1]*scale
    ymatch=yy[imatch]
    dlmatch=yy[imatch+1]*scale-yy[imatch-1]  # dif1_left

    t_lim=list(range(dim+1))
    tt=list(t_lim[imatch+1:dim])

    drmatch=yr[imatch+1]-yr[imatch-1]         # dif1_right

    f =(dlmatch-drmatch) /(2*h)

    for t in tt:                           #  y_right
        yy[t] = yr[t]



    delta_E=-f*(E-E_old) /(f-f_old)


    if abs(delta_E)<eps:  # determination of the root
                          (energy)
                        # of f(E) by the secant method
        ee[M] = E
        nk[M] = k
        ff[M] = f
        k = kmax
        break
```

So far, the only thing we need to change in our program is the equation that defines our effective potential $V(x)$. The rest of the code will be the same for any equation that is in the same form as equation (9). From now on, we must change the code whenever we are looking for solutions with a different potential.

The parameter M indicates the eigenvalue that we are determining, in the next step, in order to avoid that the program needs to sweep the entire potential well in search of solutions, we can give increments between one eigenvalue and another in the form of multiples of the dE parameter, which will be duly defined in a next step.

Usually, this process involves trial and error, where we adjust the dE multiplier for each case, until we find the desired eigenvalue. However, as we are developing this first example for the hydrogen atom, which already has its energy eigenvalues well defined, this task becomes much easier. Before we find the ground state ($M = 0$) we must use the step $24 \times dE$. Thus, we find for ground state energy the value of $\varepsilon = -0.99453$, which must be compared with the known value of the ground state energy for the hydrogen atom $\varepsilon = -1$. And the next

multipliers so that we can find at least the first three solutions which are:

```
        else:
            f_old=f;  E_old=E;   E = E + 24*dE

    if M==0:
            E = E + 126*dE
        for j in range(dim):
            yy1[j]=yy[j]
    if M==1:
        E = E + 3*dE
        for j in range(dim):
            yy2[j]=yy[j]
    if M==2:
        E = E_old + 6*dE
        for j in range(dim):
            yy3[j]=yy[j]
    print(" k = %2d   E_old = %2.3f    Eingen = %2.3f
           f_old = %2.3e   f = %2.3e   delta_E = %2.3e"
           %(nk[M],E_old,ee[M],f_old,f,delta_E))
    print()

return ee, xx, yy1, yy2, yy3
```

If you are interested in finding other solutions, you should add the steps for large values of M.

We finally reach the part of the program where we must introduce the initial values to proceed with the solution. Whenever we start to develop a new code, these must be the first values to be changed, right after choosing the effective potential $V(x)$.

The parameters a and b correspond to the boundary points mentioned in the second section. At first, our wave function should have its initial value (a) equal to 0; thus, to avoid divisions by 0 throughout the program, we should start from a relatively small value ($a = 0.001$). In the piece of code below h represents the δ increment, $kmax$ is the maximum number of interactions for each eigenvalue, $nmax$ is the number of eigenvalues that we are trying to find, Ein is the minimum energy of de effective potential that we are trying to solve, and $Vmax$ is the maximum energy of the same effective potential.

```
a=0.001; b= 60.001; h=0.01
xl=a; xu=b; D = xu-xl
delta = 0.02; eps = 0.0001
dim=int(D /h); kmax=300; nmax=3
n =0; iflag=0
Rydberg=13.605693122994

x0=xl; y0=0. ; y1=delta; iflag=1
nrovl(y0,y1,x0,0,0,h,iflag)
nrovr(y0,y1,xu,0,0,h,iflag)

dE = delta /4
Ein = -1.6
Vmax= 0.
```

With the inputs given in the previous step, we can now print the initial values of our program on the screen.

```
print(" =================================================")
print("  ")
print("        Potential : V(x) = (L*(L+1))/(x*x)-2./x
(Hydrogen L=0)    ")
print("  ")
print(" =================================================")

print(" E_in = %2.4f" %(Ein)," dE = ",dE," h = ",h," dim = ",
dim)
print("  ")
```

```
ee,xx,yy1,yy2,yy3=espectro(xl,xu,h,delta,eps,dim,nmax,kmax,
                                      Ein,Vmax,dE,iflag)

A=1.                    # amplitude normalization in 1 unit
ymax1=vec_max(dim,yy1)
ymax2=vec_max(dim,yy2)
ymax3=vec_max(dim,yy3)

for i in range(dim):    # amplitude normalization
    yy1[i] = yy1[i] /ymax1;  yy2[i] = yy2[i] /ymax2
    yy3[i] = yy3[i] /ymax3
```

Finally, at this point the program was able to determine the eigenvalues and eigenfunctions associated with the first three states of the hydrogen atom. Initially, the program prints the eigenvalues ($Eingen$) on the screen according to the Figure 2.

Table 1 shows the comparison between our results, extract from the Figure 2, and the well known analytical values for the hydrogen atom, in Rydberg units, given by the formula $E_n = -1/n^2$.

Table 1: Comparison between the energy values, in Rydberg units, found by the Numerov's method and the analytical ones for $\ell = 0$ and n=1, 2 and 3.

n	Numerov's Energy	Analytical value
1	-0.995	-1
2	-0.245	-0.25
3	-0.110	-0.111

From now on we will introduce the codes necessary to generate the graphs, as well as calculate the normalization of the wave functions. First, let's plot the effective potential graph. During the process of setting the code for different potentials, it is important to know the effective potential in order to adjust the parameters accordingly.

```
colors=['b','r','g','c','m','y','k'] #color list (blue, red,
                                      green, cyan,
                                      #yellow, magenta and black)
plt.figure(figsize=(8, 6), dpi=800)
plt.plot(x,V(x),color=colors[6],linewidth=2)
plt.ylim(-10,1)
plt.xlim(a,10)
plt.xlabel('x')
plt.ylabel('Effective potential')
plt.grid()
plt.show())
```

The code above plots the effective potential, as we can see in Figure 3
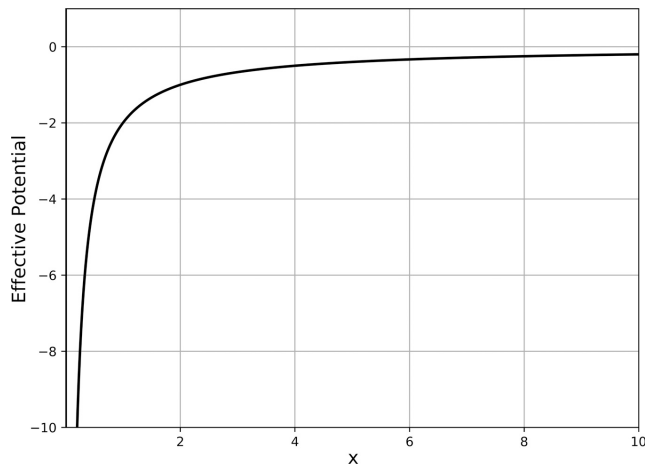

Figure 3: Effective potential of equation (11) for $\ell = 0$.

And the graph for the eigenfunctions with an arbitrary normalization, is built by

```
plt.figure(figsize=(8, 6), dpi=800)
plt.plot(xx,yy1,color=colors[0])
plt.plot(xx,yy2,'--',color=colors[1])
plt.plot(xx,yy3,':',color=colors[2])
```

```
========================================================================
                Potential : V(x) = -2/x     (hydrogen L=0)

========================================================================
E_in = -1.6000  dE =  0.005  h =  0.01  dim =  6000

 *************    Eigenvalues  # 1   *************
imatch =  126  xmatch = 1.261  V(xmatch) = -1.58604  E = -1.5950
imatch =  136  xmatch = 1.361  V(xmatch) = -1.46951  E = -1.4750
imatch =  148  xmatch = 1.481  V(xmatch) = -1.35044  E = -1.3550
imatch =  162  xmatch = 1.621  V(xmatch) = -1.23381  E = -1.2350
imatch =  180  xmatch = 1.801  V(xmatch) = -1.11049  E = -1.1150
imatch =  201  xmatch = 2.011  V(xmatch) = -0.99453  E = -0.9950
k =  5   E_old = -1.115     Eingen = -0.995   f_old = 1.052e+25   f = -3.124e+21   delta_E = -3.560e-05

 *************    Eigenvalues  # 2   *************
imatch =  548  xmatch = 5.481  V(xmatch) = -0.36490  E = -0.3650
imatch =  817  xmatch = 8.171  V(xmatch) = -0.24477  E = -0.2450
k =  1   E_old = -0.365     Eingen = -0.245   f_old = 5.129e+12   f = -4.319e+08   delta_E = -1.011e-05

 *************    Eigenvalues  # 3   *************
imatch =  870  xmatch = 8.701  V(xmatch) = -0.22986  E = -0.2300
imatch =  1819  xmatch = 18.191  V(xmatch) = -0.10994  E = -0.1100
k =  1   E_old = -0.230     Eingen = -0.110   f_old = -1.184e+09   f = -2.755e+03   delta_E = 2.792e-07
```

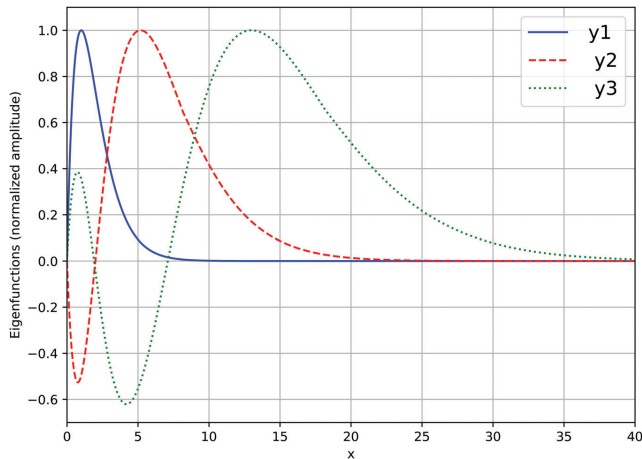**Figure 2:** Output of eigenvalues displayed by the code.

Figure 4: First three eigenfunctions of equation (11), with $\ell = 0$, calculated with the Numerov method.

Figure 5: First three normalized eigenfunctions of equation (11), with $\ell = 0$, calculated with the Numerov method.

```
plt.legend([' y1',' y2',' y3'],prop="size":15,frameon=True)
plt.ylim(-0.7,1.1)
plt.xlim(0,40)
plt.xlabel('x')
plt.ylabel('Eigenfunctions (normalized amplitude)')
plt.grid()
plt.show()
```

Which generates the plot shown in the Figure 4.

As a last step, let's include in our code, the calculation of the normalization of the wave functions, given by:

```
efic=list(range(nmax));  I=list(range(nmax))
eI=list(range(nmax))

S=A*(xu-xl)                    # Born normalization
l0=0; l1=0; l2=0
N=10000
for i in range(N):     # integral of y*y with the Monte Carlo
method
    y = A*random(1)
    j=np.random.randint(dim)
    yj1=yy1[j]*yy1[j];   yj2=yy2[j]*yy2[j];   yj3=yy3[j]*yy3[j]
    if y <= yj1:
        l0 += 1
    if y <= yj2:
        l1 += 1
    if y <= yj3:
        l2 += 1

efic[0] = float(l0) /N;  efic[1] = float(l1) /N
efic[2] = float(l2) /N
I[0] = S*efic[0];        I[1] = S*efic[1];        I[2] = S*efic[2]
eI[0] = (S /np.sqrt(N))*np.sqrt(efic[0]*(1-efic[0]))
eI[1] = (S /np.sqrt(N))*np.sqrt(efic[1]*(1-efic[1]))
eI[2] = (S /np.sqrt(N))*np.sqrt(efic[2]*(1-efic[2]))

print(" ")
print("  probability normalization ")
print("efic1=%2.3f I1=%2.3f eI1 = %2.3f" %(efic[0],I[0],eI[0]))
print("efic2=%2.3f I2=%2.3f eI2 = %2.3f" %(efic[1],I[1],eI[1]))
print("efic2=%2.3f I2=%2.3f eI2 = %2.3f" %(efic[2],I[2],eI[2]))
```

```
for i in range(dim): # probability normalization
    yy1[i]=yy1[i] /np.sqrt(I[0]); yy2[i]=yy2[i] /np.sqrt(I[1])
    yy3[i]=yy3[i] /np.sqrt(I[2])


ymax1=vec_max(dim,yy1);    ymax2=vec_max(dim,yy2)
ymax3=vec_max(dim,yy3)

S0=ymax1*ymax1*(xu-xl);    S1=ymax2*ymax2*(xu-xl)
S2=ymax3*ymax3*(xu-xl)
```
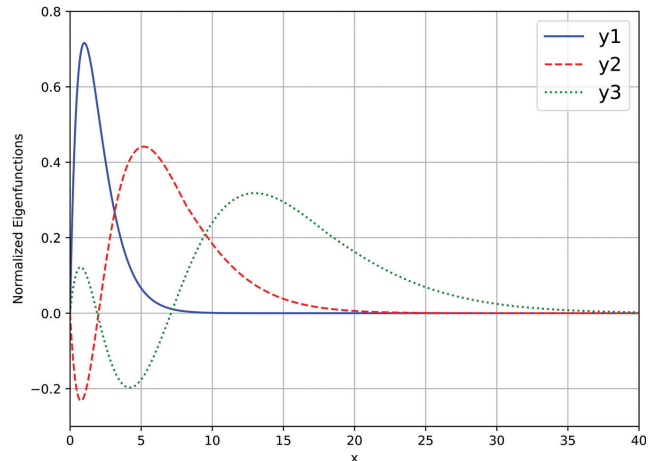
```
l0=0;  l1=0; l2=0              # checking probability
N=10000
for i in range(N):
    y1 = ymax1*ymax1*random(1); y2 = ymax2*ymax2*random(1)
    y3 = ymax3*ymax3*random(1)
    j=np.random.randint(dim)
    yj1=yy1[j]*yy1[j];   yj2=yy2[j]*yy2[j];    yj3=yy3[j]*yy3[j]
    if y1<= yj1:
        l0 += 1
    if y2 <= yj2:
        l1 += 1
    if y3 <= yj3:
        l2 += 1

efic[0] = float(l0) /N;  efic[1] = float(l1) /N
efic[2] = float(l2) /N
I[0] = S0*efic[0];     I[1] = S1*efic[1];     I[2] = S2*efic[2]
eI[0] = (S0 /np.sqrt(N))*np.sqrt(efic[0]*(1-efic[0]))
eI[1] = (S1 /np.sqrt(N))*np.sqrt(efic[1]*(1-efic[1]))
eI[2] = (S2 /np.sqrt(N))*np.sqrt(efic[2]*(1-efic[2]))

print(" ")
print("  checking probability ")
print(" efic1=%2.3f  I1=%2.3f  eI1=%2.3f" %(efic[0],I[0],eI[0]))
print(" efic2=%2.3f  I2=%2.3f  eI2=%2.3f" %(efic[1],I[1],eI[1]))
print(" efic3=%2.3f  I3=%2.3f  eI3=%2.3f" %(efic[2],I[2],eI[2]))
```

```
figure(figsize=(8, 6), dpi=800)
plot(xx,yy1,color=cores[0])
plot(xx,yy2,'--',color=cores[1])
plot(xx,yy3,':',color=cores[2])
legend(['y1','y2','y3'],prop="size":15,frameon=True)
ylim(-0.3,0.8)
xlim(0,40)
xlabel('x')
ylabel('Normalized Eigenfunctions')
grid(True)
show()
```

Thus, we were able to obtain our final graph, composed of the first three wave functions for the hydrogen atom (Figure 5).

4. Morse Potential

The Morse potential is a common model for the interatomic interaction of a diatomic molecule [11–13]. In this section, in order to learn how we can use the Phyton code

in other problems, we will see what we must change in the code that was made available in the previous section so that the program will be able to solve equation (2) for the quantum number $\ell = 0$ and the Morse potential with arbitrary parameters given by:

$$V(x) = 16(1 - e^{-2x})^2 \qquad (16)$$

In this example, we will calculate the first two bounded states, for that, we must adjust the dE multiplier for each case as:

```
        else:
            f_old=f;  E_old=E;   E = E + 7.2*dE

    if m==0:
                E = E + 70*dE
        for j in range(dim):
            yy1[j]=yy[j]
    if m==1:
        E = E + dE/10
        for j in range(dim):
            yy2[j]=yy[j]
    if m>=2:
        E = E_old + 50*dE
        for j in range(dim):
            yy3[j]=yy[j]
    print(" k = %2d   E_old = %2.3f     Eingen = %2.3f
            f_old = %2.3e    f = %2.3e   delta_E = %2.3e"
            %(nk[m],E_old,ee[m],f_old,f,delta_E))
    print()

 return ee, xx, yy1, yy2, yy3
```

And, the most important part which is the adjustment of the initial data of our problem. Analyzing the effective potential, we can verify that the value os parameter a must be negative, and its minimum value is zero, so the code must be set as follows:

```
a=-1.01; b= 5.01; h=0.006
xl=a; xu=b; D = xu-xl
delta = 0.01; eps = 0.00001
dim=int(D /h); kmax=100; nmax=2
n =0; iflag=0
Rydberg=13.605693122994

x0=xl; y0=0. ; y1=delta; iflag=1
nrovl(y0,y1,x0,0,h,iflag)
nrovr(y0,y1,xu,0,h,iflag)

dE = delta
Ein = 0.0
Vmax= 16.
```

From now on, the program is able to find the energies of the first two states, which are respectively 7.1380 and 15.0380. As well as already determined the wave functions also for the first two states.

Then, all that remains is to get all the aesthetic part of the code right, adjusting the limits of graphics, subtitles and other factors. Thus we first get the effective potential, as shown in Figure 6 and finally arrive at the normalized wave functions shown in Figure 7.

In Table 2, we can compare the eigenvalues found by the Numerov's method with the analytical values given by $E_n = 16 \left[\left(n + \frac{1}{2}\right) - \frac{1}{4}\left(n + \frac{1}{2}\right)\right]$.
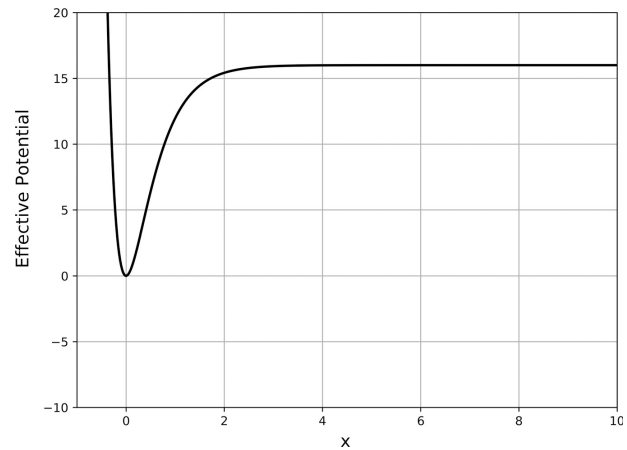
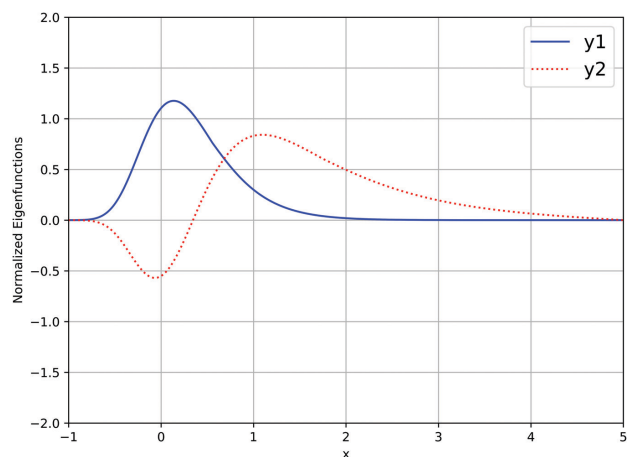**Figure 6:** Effective Potential given by equation (16).

Figure 7: Normalized Eigenfunctions of equation (11), with the potential given by equation (16) with $\ell = 0$.

Table 2: Comparison between the energy values for the Morse potential, in Rydberg units, found by the Numerov's method and the analytical ones for $\ell = 0$ and n=0 and 1.

n	Numerov's Energy	Analytical value
0	7.1380	7.0
1	15.0380	15.0

5. Quantum Dot

The development of technology based on quantum dots is quite recent, but it is already showing signs that it is the next great technology, when we talk about a kind of optical devices. In a simple model for a quantum dot composed of two electrons, they can be described with an external harmonic oscillator potential of frequency $\Omega = 2\omega$. Following the steps of reference [10], we have the effective potential to be introduced into equation (2) for the quantum number $\ell = 0$ is given by:

$$V(x) = \frac{1}{x} + \omega^2 x - \frac{0.25}{x^2} \qquad (17)$$

Introducing this potential into our code, let's now calculate the first five solutions that the program is capable of finding. We can observe that in our code used so far, we only have up to three wave functions, in this example we will show how we included two more solutions in the code. The procedure is very simple, just add the $yy4$ and $yy5$ functions along the code and all the other parts that are related to them, for example,

```
xx=list(range(dim));      yy=list(range(dim))
yl=list(range(dim));      yr=list(range(dim))
ee=list(range(nmax));     ff=list(range(nmax))
ff2=list(range(nmax));    yy1=list(range(dim))
yy2=list(range(dim));     yy3=list(range(dim))
 yy4=list(range(dim));  yy5=list(range(dim))
nk=list(range(nmax))
E_old = Ein;   E = Ein + dE
```

Now, as in the previous example, let's include the dE multipliers, remembering to include the new wave functions.

```
        else:
            f_old=f;  E_old=E;   E = E + dE/25

    if m==0:
        E = E + 6*dE
        for j in range(dim):
            yy1[j]=yy[j]
    if m==1:
        E = E + 6*dE
        for j in range(dim):
            yy2[j]=yy[j]
    if m>=2:
        E = E_old + 13*dE/2
        for j in range(dim):
            yy3[j]=yy[j]
    if m>=3:
        E = E_old + 13*dE/2
        for j in range(dim):
            yy4[j]=yy[j]
    if m>=4:
        E = E_old + 4*dE
        for j in range(dim):
            yy5[j]=yy[j]
    print(" k = %2d   E_old = %2.3f    Eingen = %2.3f
            f_old = %2.3e   f = %2.3e   delta_E = %2.3e"
            %(nk[m],E_old,ee[m],f_old,f,delta_E))
    print()

return ee, xx, yy1, yy2, yy3. yy4, yy5
```

And finally, we must include the initial conditions for our problem, as shown below.

```
a=0.001; b= 60.001; h=0.02
xl=a; xu=b; D = xu-xl
delta = 0.01; eps = 0.00001
dim=int(D /h); kmax=100; nmax=5
n =0; iflag=0
Rydberg=13.605693122994

x0=xl; y0=0. ; y1=delta; iflag=1
nrovl(y0,y1,x0,0,h,iflag)
nrovr(y0,y1,xu,0,h,iflag)

dE = delta/2.8
Ein = 0.086857
Vmax= 2.
```

Thus, adjusting the rest of the code, we obtain the first five eigenvalues. The effective potential and the eigenfunctions can be seen in Figures 8 and 9.

In Table 3, we can compare the eigenvalues found by the Numerov's method with the analytical values for the quantum dot given by $\eta_{n\ell} = 2(n + \ell + 1)\omega$.
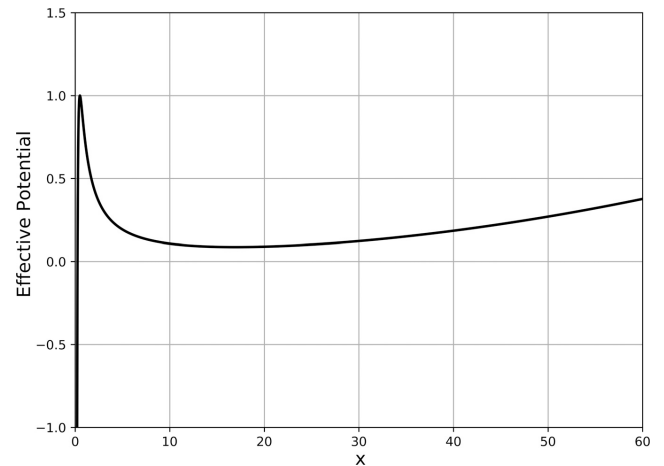
Figure 8: Effective Potential given by equation (17).
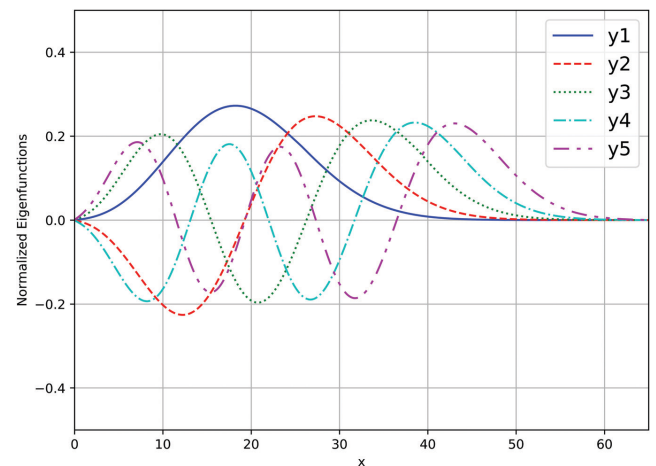

Figure 9: Normalized Eigenfunctions of equation (17), with the potential given by equation (16) with $\ell = 0$.

Table 3: Comparison between the energy values for the Quantum dot, in Rydberg units, found by the Numerov's method and the analytical ones for $\ell = 0$. Results are shown for different values of n and fixed ω ($\omega = 0.01$ Ha).

n	Numerov's Energy	Analytical value
4	0.1046	0.10
6	0.1403	0.14
8	0.1760	0.18
10	0.2134	0.22
12	0.2507	0.26

6. Conclusion

Analyzing the results arranged in Tables 1, 2 and 3 we can conclude that the method used here is able to reproduce the analytical results within tiny errors.

Therefore, it is evident that the Numerov numerical method is a powerful tool, easy to use, which can help in the development of not only new knowledge in the area of programming, but also in solving the Schrödinger

equations outside the usual results found in the examples of modern physics textbooks. We hope that this short highlight to the code, which can be found in full at the links provided in the Introduction, will open doors for students to create their own versions, increasingly improving the versatility of this tool.

Acknowledgments

References

[1] F. Caruso and V. Oguri, Rev. Bras. Ens. Fis. **36**, 2310 (2014).

[2] B.V. Numerov, Not. Roy. Astro. Soc. **84**, 592 (1924).

[3] B.V. Numerov, Astronomische Nach. **230**, 359 (1927).

[4] J.M. Blatt, Jour. Comput. Phys. **1**, 382 (1967).

[5] A.C. Allison, Jour. Comput. Phys. **6**, 378 (1970).

[6] J.P. Leroy and R. Wallace, Jour. Phys. Chem. **89**, 1928 (1985).

[7] A. Bağcı and Z. Güneş, arXiv:2111.11379 (2021).

[8] F. Caruso, J. Martins and V. Oguri, Ann. Phys. **347**, 130 (2014).

[9] F. Caruso, J. Martins, V. Oguri and F. Silveira, Ann. Phys. **377**, 518 (2017).

[10] F. Caruso, V. Oguri and F. Silveira, Braz. J. Phys. **49**, 432 (2019).

[11] P.M. Morse and E.C.G. Stueckelberg, Phys. Rev. **33**, 932 (1929).

[12] P.M. Morse, Phys. Rev. **34**, 57 (1929).

[13] M.N. Srnec, U. Shiv and D.M. Jeffry, J. Chem. Edu. **94**, 669 (2017).