Solución de la ecuación de Schrödinger en dos dimensiones usando el shooting method

Crsthian David Hinostroza Vargas Machuca
Código: 16130112

Curso: Análisis Numérico

Resumen

En el presente trabajo se resolverá la ecuación de Shrödinger para un potncial armónico bidimensional con simetría radial, obteniendo primero sus autovalores con el Shooting Method, para posteriormente, calcular numéricamente las funciones de onda correspondientes con el método de Runge-Kutta de cuarto orden.

1. Introducción

Una de las soluciones de la ecuación de Scrödinger más importante es la del oscilador armónico, usada en la construcción de modelos.

Esta ecuación es una con condiciones de frontera, lo que hace que no pueda resolverse con ciertos métodos numéricos adecuados para problemas con condiciones iniciales, además de buscar los parámetros qu se adecúan a las condiciones que nos da el problema.

Por esta razón, en este trabajo, se presenta el uso del Shooting Method, para buscar el parámetro necesario, la energía, para resolver estas ecuaciones.

2. Marco teórico

2.1. La ecuación de Schrödinger

El potencial que genera el movimiento armónico es una fuerza restauradora, proporcional al cuadrado del desplazamiento de la posición de equilibrio.

$$U = \frac{m\omega^2}{2}x^2$$

Con lo que la ecuación de Schrödinger con este potencial tendría la forma:

$$(-\frac{\hbar}{2m}\nabla^2 + \frac{m\omega^2}{2}r^2)\Psi = E\Psi$$

Entonces, para el caso bidimensional, y realizando una separación de variables, se obtienen 2 ecuaciones diferenciales de la forma:

$$(-\frac{\hbar}{2m}\frac{d^2}{dx^2} + \frac{m\omega^2}{2}x^2)\Phi_x E_x = \Phi_x$$
$$(-\frac{\hbar}{2m}\frac{d^2}{dy^2} + \frac{m\omega^2}{2}y^2)\Phi_y = E_y\Phi_y$$

Donde $\Psi = \Phi_x\Phi_y$

Ya que se van a realizar cálculos numéricos, es neceario tener las ecuaciones en su forma adimensional, para de esta manera, evitar posibles errores en la solución numérica. Para adimensionalizar la ecuación se realiza el cambio de variable $x = q\sqrt{\frac{\hbar}{m\omega}}$, obteniendo:

$$(-\frac{d^2}{dq^2} + q^2)\phi = E'_q\phi$$

Por la simetría radial del potencial, ambas dimensiones tendrán la misma forma, por lo que sólo es necesario resolver una de ellas, por lo que se puede obtener la solución bidimensional multiplicando las soluciones de ambas dimensiones.

2.2. Métodos numéricos

Se construyó el algoritmo para resolver las ecuaciones en el lenguaje Python 3.0, aprovechando el paquete Numpy para realizar los cálculos numéricos, y el paquete Matplotlib.pyplot para graficar las soluciones.

Ya que la ecuación de Schrödinger es una ecuación diferencial de segundo grado, se decidió usar el método de Runge-Kutta de cuarto orden para resolverla numéricamente, sin embargo, antes de poder hacerlo, se necesita del valor de los autovalores de energía. Con esto en mente, se emplea el Shooting method y el método de la bisección para obtenerlas.

2.2.1. Shooting method

Este método es usado para resolver ecuaciones diferenciales con problemas de valores en la frontera, reduciéndolas a ecuaciones con valores iniciales.

Si tenemos una ecuación diferencial de segundo orden con valores en la frontera:

$$y''(t) = f(t, y(t), y'(t)), \qquad y(t_0) = y_0, \qquad y(t_1) = y_1$$

Considerando $y(t, a)$, donde a es un parámetro variable, como la solución para el valor inicial:

$$y''(t) = f(t, y(t), y'(t)), \qquad y(t_0) = y_0, \qquad y'(t_0) = a$$

Se define una función $F(a)$, como la diferencia entre $y(t_1, a)$ y el valor en la frontera y_1

$$F(a) = y(t_1, a) - y_1$$

Donde F tiene una raíz a, entonces la solución $y(t, a)$ es también una solución del problema de valores en la frontera.

2.2.2. Método de la bisección

Es un método para obtener las raíces de una ecuación no lineal. Requiere de un intervalo inicial $[a, c]$, en el que se se sabe que existe una de las raíces de la ecuación b, y un rango de tolerancia para la solución que da este método.

Ya que en el intervalo inicial que se considera se encuentra una raíz, es mandatorio que la función evaluada en a y c, tengan signos opuestos. De esta forma se puede buscar la raíz de la ecuación en el punto medio. Si este valor medio no es cero, se le escoge como nuevo extremo del intervalo; elección que depende del signo de la función evaluada en ese punto, y del signo de la función evaluada en el otro extremo.

2.2.3. Método de Runge-Kutta

El método de Runge-Kutta parte de una aproximación que se hace al evaluar una integral con la regla de Simpson 1/3. con lo que tenemos:

$$y_{n+1} = y_n + \frac{h}{6}[f(y_n, t_n + f(\overline{y}_{n+\frac{1}{2}}, t_{n+\frac{1}{2}} + f(\overline{y}_{n+1}, t_{n+1}]$$

Donde $\overline{y}_{n+\frac{1}{2}}$ y $\overline{y}_{n+1}$ son aproximaciones que se deber hacer de modo que:

$$y_{n+1} = y_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

En donde:

$$k_1 = hf(y_n, t_n)$$

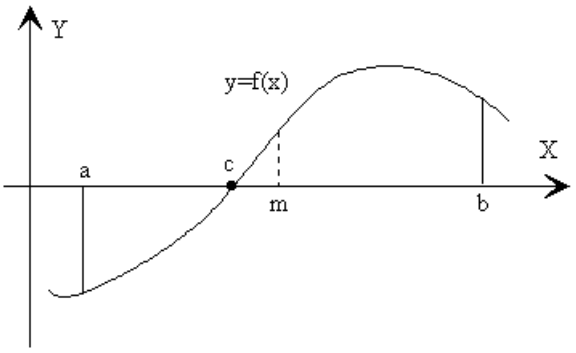


Figura 1: Visualización del método de la bisección.

$$k_2 = hf(y_n + \frac{k_1}{2}, t_n + \frac{h}{2})$$

$$k_3 = hf(y_n + \frac{k_2}{2}, t_n + \frac{h}{2})$$

$$k_4 = hf(y_n + k_3, t_n + h)$$

Si decidimos usar el método para resolver un conjunto de ecuaciones diferenciales $y' = f(y, z, t)$ y $z' = g(y, z, t)$ se tienen las aproximaciones:

$$y_{n+1} = y_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

$$z_{n+1} = z_n + \frac{1}{6}[l_1 + 2l_2 + 2l_3 + l_4]$$

En donde se considera:

$$k_1 = hf(y_n, z_n, t_n)$$
$$l_1 = hg(y_n, z_n, t_n)$$

$$k_2 = hf(y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}, t_n + \frac{h}{2})$$
$$l_2 = hg(y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}, t_n + \frac{h}{2})$$

$$k_3 = hf(y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}, t_n + \frac{h}{2})$$
$$l_3 = hg(y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}, t_n + \frac{h}{2})$$

$$k_4 = hf(y_n + k_3, z_n + l_3, t_n + h)$$
$$l_4 = hf(y_n + k_3, z_n + l_3, t_n + h)$$

3. Investigación numérica

Para resolver la ecuación de Schrödinger es necesario conseguir los valores correctos para los autovalores de energía, para posteriormente, calcular la solución de la ecuación diferencial. Primero discutiremos las dos funciones más importantes que se definieron en el código del programa, para luego explicar cómo se usaron.

3.1. Código del programa

Para realizar los diversos cálculos se definieron dos funciones importantes en él:

3.1.1. bisec

Esta función obtiene las raíces de una función dada. Sus parámetros son:

- **func**: Función a evaluar.
- **a**: Valor mínimo del intervalo de búsqueda.
- **c**: Valor máximo del intervalo de búsqueda.
- **tol**: Rango de tolerancia.

La función primero calcula la cantidad de iteraciones que realizará para obtener el rango de tolerancia deseado a partir del tamaño del intervalo inicial, para después iniciar la búsqueda de la raíz.

3.1.2. RK_4_1

Esta resuelve un sistema de dos ecuaciones diferenciales o, para este caso, una ecuación de segundo grado, con el método de Runge-Kutta basado en la regla de Simpson 1/3. Sus parámetros son:

- **func**: Segunda derivada de la ecuación diferencial.
- **y_0**: Valor de la función en el tiempo inicial.
- **dy_0**: Valor de la derivada de la función evaluado en el tiemo inicial.
- **t_0**: Tiempo inicial.
- **t_f**: Tiempo final.
- **E**: Energía
- **opcion**: Puede ser 1, para devolver un vector con todos los datos obtenidos para la funcion; o 2, para devolver únicamente el valor de la función en el tiempo final ingresado.

3.2. Obtención de los autovalores

La obtención de los autovalores de energía se realiza al considerar los valores en la frontera para el caso del potencial armónico.

Como se sabe, el potencial del oscilador armónico tiende a cero al evaluarse en el infinito, hecho que solo sucede cuando se escogen correctamente los autovalores, por lo que, se resolvió la ecuación de Schrodinger para $\Phi_x(-10)$, utilizando como valor inicial $x_0 = -10$ para un rango de energías de 0 a 6. De esta forma, se obtuvo una función que tendrá como raíces a los valores de energía permitidos.

Seguidamente, se localizaron las raíces con el método de la bisección, eligiendo el intervalo de búsqueda inicial a partir de la figura 2, para un rango de tolerancia de $\tau = 10^{-12}$:

$$E_0 = 0{,}9999999999995453$$
$$E_1 = 3{,}0000000000004547$$
$$E_2 = 5{,}0000000000004550$$

Analíticamente, los autovalores de energía, para el oscilador armónico en una dimensión, son proporcionales a $n + \frac{1}{2}$, pero debido a que usamos la ecuación adimensionalizada, nuestros autovalores de energía tendrán la forma $2n + 1$, lo que explica la ausencia de números pares.

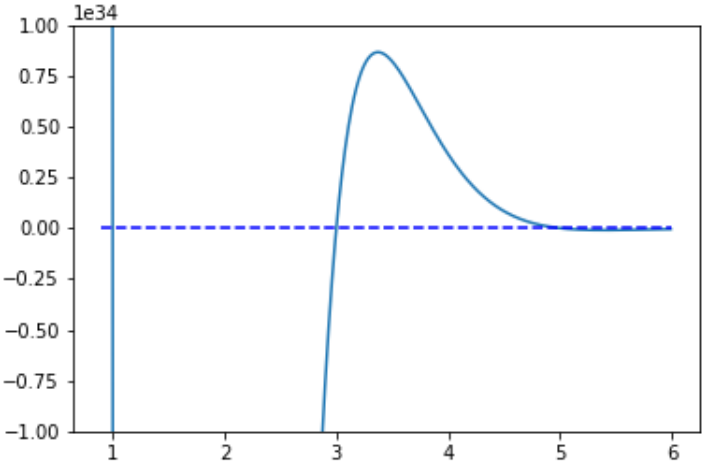
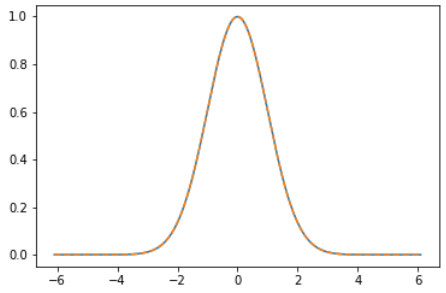


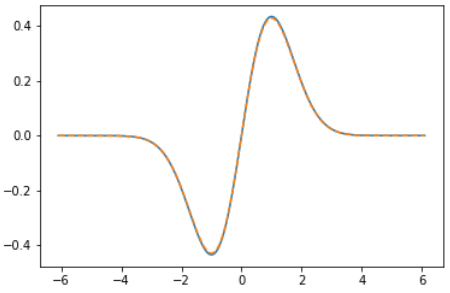
Figura 2: Obtiención de autovalores de energía.

3.3. Obtención de las funciones de onda

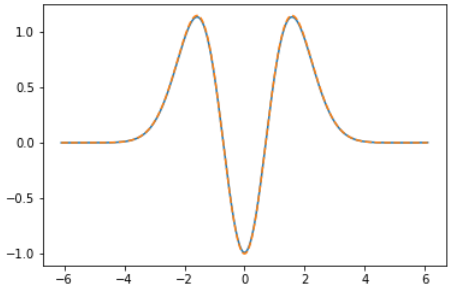
Luego de obtenidos los autovalores de energía, estos se usaron para obtener las soluciones de las ecuaciones de Shrödinger corespondientes a cada uno de ellos. De esta manera, E_0 está relacionado a la función de onda en el estado fundamental; mientras que E_1 y E_2, al primer y segundo estado excitado.



(a)



(b)

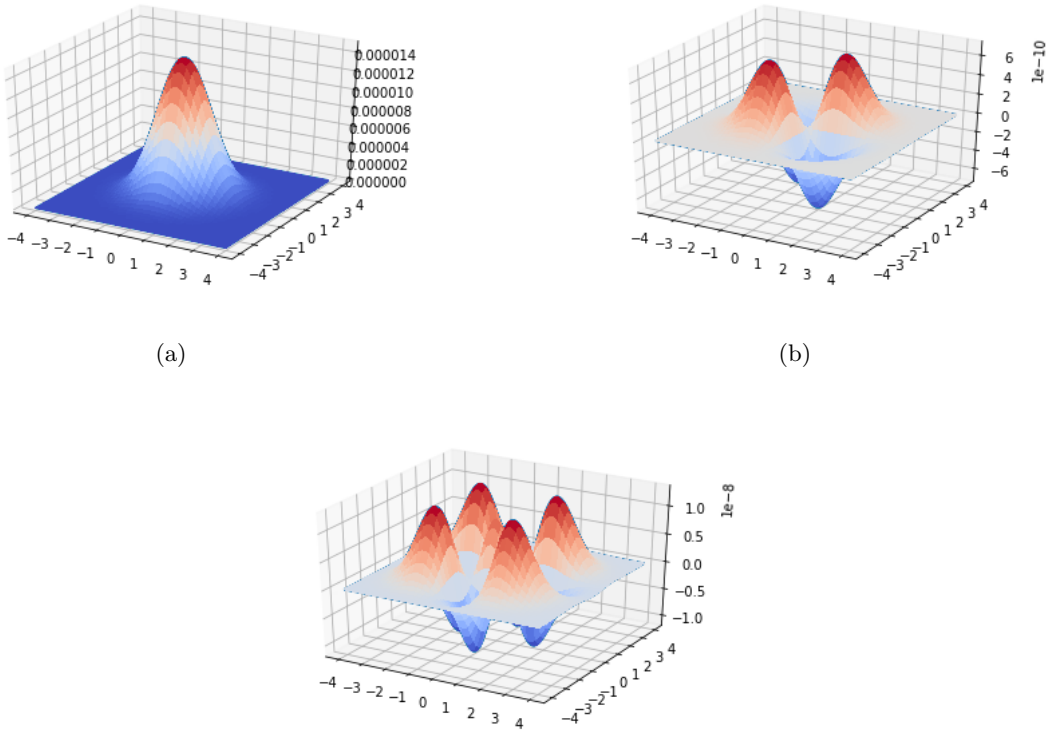


(c)

Figura 3: Soluciones para el oscilador armónico cuántico en dos dimensiones.

4. Resultados

Las gráficas obtenidas para la solución bidimensional se muestran en la Figura 4.



(a) (b)

(c)

Figura 4: Soluciones para el oscilador armónico cuántico en dos dimensiones.

Los resultados muestran el comportamiento esperado, aunque aún existen algunas fallas en la escala. Por otro lado, todas las soluciones complen con las condiciones de frontera del potencial armónico.

5. Conclusiones

Al intentar resolver una ecuación diferencial por primera vez, uno de los primeros pasos es encontrar los valores que se adecúen a la solución que desamos obtener. Por este motivo, la proximación por el Shooting method es una forma bastante versátil para lograrlo.

Por otro lado, al ser Python un lenguaje de alto nivel, le tomará más tiempo desarrollar una cantidad grande de cálculos. Esto sucedió al desear obtener las gráficas para as soluciones en dos dimensiones por lo que una solución para este problema podría ser mezclar la eficiencia de Python con lo ligero de otros lenguajes.

En este trabajo no se discutió el caso de un oscilador cuántico anisotrópico, o con perturbaciones. Por lo que la tarea de adecuar el código a esos potenciales es una tarea pendiente.

Referencias

M. Christine. *Solving The Stationary One Dimensional Schrödinger Equation With The Shooting Method.* Technische Universtität Wien, 2016.

A. García. *Numerical Methods for Physics.* Prentice Hall, 1999.

S. Nakamura. *Métodos numéricos aplicados con software.* Pearson Education, 1992.

W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes, The art of scientific computing.* Cambridge University Press, 2007.

Apéndice

```
import numpy as np
import matplotlib.pyplot as mp
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm


#Funcion para graficar el eje X
def ejeX(x):
    return x-x

#Funcion para evaluar el potencial armonico
def harmE(t,y,E):
    s = -(E - t**2)*y
    return s

#Funcion basada en el metodo de la biseccion
def bisec(func,a,c,tol):
    cota = (c-a)/2
    N = 0

    while cota >= tol:
        N = N + 1
        cota= cota/2
    N = N + 1

    if func(a) < 0 :
        for i in range(0,N+1):
            b = (a+c)/2
            if func(b) < 0:
                a = b
            if func(b) > 0:
                c = b
        return b

    if func(a) > 0 :
        for i in range(0,N+1):
            b = (a+c)/2
            if func(b) < 0:
                c = b
            if func(b) > 0:
                a = b
        return b


#Funcion basada en el metodo de Runge-Kutta
def RK_4_1(func,y_0,dy_0,t_0,t_f,E,opcion):


    #h = 0.001
    h = 0.05
    N = int((t_f-t_0)/h)

    Y = [y_0]
    tiempo = [t_0]

    dy_i = dy_0
    y_i = y_0
    t = t_0
    for i in range (0,N):
        k1 = h*dy_i
        l1 = h*func(t,y_i,E)

        k2 = h*(dy_i + l1/2)
        l2 = h*func(t + h/2,y_i + k1/2,E)
```

```
            k3 = h*(dy_i + l2/2)
            l3 = h*func(t + h/2,y_i + k2/2,E)

            k4 = h*(dy_i + l3)
            l4 = h*func(t + h,y_i + k3,E)

            y = y_i + (1/6)*(k1 + 2*k2 + 2*k3 + k4)
            dy = dy_i + (1/6)*(l1 + 2*l2 + 2*l3 + l4)

            Y.append(y)
            t = t + h
            tiempo.append(t)
            y_i = y
            dy_i = dy

        if opcion == 1 :
            Y = Y / np.max(Y)  #Normalizamos la funcion
            return Y
        if opcion == 0 :
            return y_i

#Funcion para aplicar el shooting method
def wf_hp4(E):
    y  = RK_4_1(harmE,0.00001,0,-10,10,E,0)
    return y

#Funciones que evaluan las soluciones analiticas de las funciones de onda
def wf0(x):
    return np.exp(-(x**2)/2)

def wf1(x):
    return x*(2**-0.5)*np.exp(-(x**2)/2)

def wf2(x):
    return (2*x**2 - 1)*np.exp(-(x**2)/2)



#Aplicacion de las funciones
e = np.arange(0.9,6,0.01)
mp.plot(e,wf_hp4(e),e,ejeX(e),'b--')
axes = mp.gca()

axes.set_ylim([-1.*10**34,1.*10**34])
mp.savefig('autovalores.png')

tol = 10**-12
energy = [bisec(wf_hp4,0.5,1.5,tol),bisec(wf_hp4,2.5,3.5,tol),bisec(wf_hp4,4.5,5.5,tol)]
print(energy)



#Graficas en 1D

s = np.arange(-6.1,6.1,(12)/12000)
mp.plot(s,RK_4_1(harmE,0,0.0000001,-6.1,6.1,0.9999999999995453,1),s,wf0(s),'--')

mp.plot(s,RK_4_1(harmE,0,-0.000000425,-6.1,6.1,2.999999999816737,1),s,wf1(s),'--')

mp.plot(s,RK_4_1(harmE,0,0.00000685,-6.1,6.1,4.999999995129201,1),s,wf2(s),'--')



#Graficas 2D
fig = mp.figure()
ax = fig.add_subplot(111, projection='3d')

X = RK_4_1(harmE,0,0.00001,-4,4,0.9999999999995453,1)
```

```
N = int(np.size(X))

Z = np.zeros((N,N))
for i in range(N):
    for j in range(N):
        Z[i,j] = X[i]* X[j]

t = np.arange(-4,4.05,0.05)
s = np.arange(-4,4.05,0.05)
t,s = np.meshgrid(s,t)


ax.plot_wireframe(t, s, Z , rstride=1, cstride=1)

surf = ax.plot_surface(t, s, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)

mp.savefig('estado_fundamental_2d.png')
mp.show()



fig = mp.figure()
ax = fig.add_subplot(111, projection='3d')

X = RK_4_1(harmE,0,-0.000000425,-4,4,2.999999999816737,1)

N = int(np.size(X))

Z = np.zeros((N,N))
for i in range(N):
    for j in range(N):
        Z[i,j] = X[i]* X[j]


t = np.arange(-4,4.05,0.05)
s = np.arange(-4,4.05,0.05)
t,s = np.meshgrid(s,t)

ax.plot_wireframe(t, s, Z , rstride=1, cstride=1)

surf = ax.plot_surface(t, s, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)

mp.savefig('estado_primero_2d.png')
mp.show()



fig = mp.figure()
ax = fig.add_subplot(111, projection='3d')

X = RK_4_1(harmE,0,0.00000685,-4,4,4.999999995129201,1)

N = int(np.size(X))

Z = np.zeros((N,N))
for i in range(N):
    for j in range(N):
        Z[i,j] = X[i]* X[j]

t = np.arange(-4,4.05,0.05)
s = np.arange(-4,4.05,0.05)
t,s = np.meshgrid(s,t)

ax.plot_wireframe(t, s, Z , rstride=1, cstride=1)

surf = ax.plot_surface(t, s, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)

mp.savefig('estado_segundo_2d.png')
mp.show()
```