Proyecto 1: Ecuación de Schrödinger en Python

Alejandro Restrepo Giraldo CC: 1001389709
Joseph Nicolay Ruíz Álvarez CC:1001362404

Ecuación de Schrödinger: La ecuación de Schrödinger describe la evolución temporal de sistemas cuánticos no relativistas. En una dimensión tiene la forma

$$\hat{H}\psi(x,t) = i\hbar\frac{\partial\psi(x,t)}{\partial t}$$

Las autofunciones independientes del tiempo con la energía potencial estacionaria cumplen el problema de Sturm-Liouville

$$\left(\frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right)\psi(x) = E\psi(x)$$

$$\frac{\partial^2\psi(x)}{\partial x^2} + \frac{2m}{\hbar^2}(E - V(x))\psi(x) = 0$$

Se interpreta la función de onda $\psi(x)$ como la descripción ondulatoria del sistema y $|\psi(x)|^2$ como la densidad de probabilidad de obtener alguna medida relacionada con la posición en un valor particular de esta. La función de onda debe cumplir

- $\psi(x)$ debe ser continua
- $\frac{\partial\psi(x)}{\partial x}$ debe ser continua
- $\psi(x)$ debe ser de cuadrado integrable, lo que implica $\lim_{x\to+\infty}\psi(x) = 0$

Algoritmo: Se considera la solución para diferentes potenciales en el intervalo [-5,5]. Se toma la condición $\psi(-5) = 0$ y $\frac{\partial\psi}{\partial x}|_{x=-5} = 0.1$ debido a las restricciones expuestas anteriormente. Se transforma la ecuación de Schrödinger en un sistema lineal de dos ecuaciones y se escibe como una función que tiene como argumentos un arreglo de la función de onda y su derivada, el valor en x en donde se soluciona y una función potencial arbitraria de x.

```
# Función de la ecuación de Schrödinger
def psi(Psi, x, V):

    # Linealizando la ec. de Schrödinger
    # x_1' = x_2 = psi
    # x_2' = (2m/hb**2)*(V(x)-E)x_1 = psi'

    return [Psi[1], (2*m)*(V(x) - E)*Psi[0]]
```

Figure 1: Función de la ecuación de Schrödinger

Los potenciales son funciones que toman como argumento el valor en x de la posición. En caso de requerir parámtros son definidos en la función. Para el pozo finito, se tomó $V = 0$ entre [-2,2] y $V = 4$ en las demás posiciones. Para el pozo infinito se considero el potencial $V = 0$ en todo el intervalo de solución [-5,5] para no tratar con valores de infinito y $V = 10^{100}$ en las demás posiciones. Para el potencial de oscilador armónico $\frac{1}{2}kx^2$ se tomó k = 2.

```python
# Recibe un valor del eje x, el ancho tal que el potencial es diferente de cero en [-l,l]
# y la profundidad del pozo V_0
def Pozo(x):

    # Ancho y profundida del pozo
    l = 2
    V_0 = 4

    if x<-l or x>l:
        return V_0
    if x>=-l and x<=l:
        return 0
```

Figure 2: Función de pozo finito.

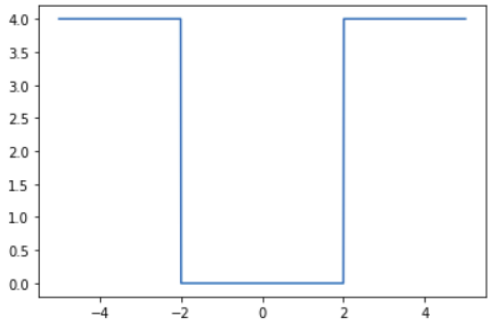

Figure 3: Gráfica del pozo finito.

```python
# Recibe un valor del eje x, el ancho tal que el potencial es diferente de cero en [-l,l]
def PozoInf(x):

    # Ancho y profundida del pozo
    l = 5.0
    V_0 = 1e100

    if x<-l or x>l:
        return V_0
    if x>=-l and x<=l:
        return 0
```

Figure 4: Función de pozo infinito.

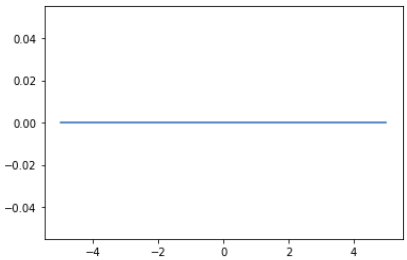

Figure 5: Gráfica del pozo infinito.

```python
# Función de potencial de oscilador armónico. Recibe un punto en el eje x y la constante
def Oscilador(x):

    # Constante de elasticidad
    k = 2

    return (1/2)*k*x**2
```

Figure 6: Función de potencial de oscilador armónico.

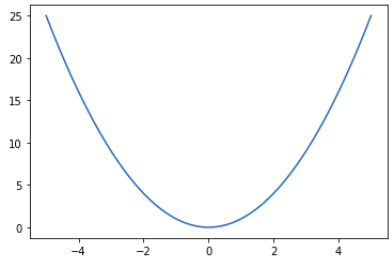

Figure 7: Gráfica de potencial de oscilador armónico.

Los autovaores de energía se calculan de acuerdo a sus valores analíticos. Se consideró la masa de la partícula del sistema $m = 1$, también para evitar tratar con números muy grandes o pequeños que pudiesen causar under- / over-flow se considera $\hbar = 1$.

Para el pozo infinito los autovalores con estas reducciones y las dimensiones particulares son $E_n = \frac{n^2\pi^2}{200}$ con L el ancho del pozo. Para el oscilador armónico $E_n = \sqrt{2}\left(n + \frac{1}{2}\right)$. El pozo finito presenta una complicación puesto que sus energías no tienen soluciones analíticas. Además de esto, la escogencia de una función de onda simétrica o antisimétrica impone ligaduras en los coeficientes de la solución y resultan en diferentes autovalores. Para el caso simétrico se hallaron numéricamente las primeras dos raíces de la ecuación $\sqrt{4-E} = \sqrt{E}tan(2\sqrt{2E})$, similarmente para el caso antisimétrico, las dos primeras raíces de $\sqrt{4-E} = -\sqrt{E}cot(2\sqrt{2E})$.

```
# Funciones de energías para las funciones de onda simétricas y antisimétrica
def E_sym(E):
    return np.sqrt(V_0-E) - np.sqrt(E)*np.tan(np.sqrt(E/2)*2*l)

def E_asym(E):
    return np.sqrt(V_0-E) + np.sqrt(E)*1/np.tan(np.sqrt(E/2)*2*l)


#Esta parte se utiliza para hallar las de energías más facilmente usando guess de la raíz
guess = np.round(np.arange(0.1, 4, 0.1),6)

# Arreglo de autovalores de energía
ENER = []

# Se depuran las raíces
for i in range(0,len(guess)):

    # Raíces usando el arreglo guess como valor cercano a la raíz real
    a = np.round(scp.fsolve(E_sym, guess[i])[0], 6)
    b = np.round(scp.fsolve(E_asym, guess[i])[0], 6)

    # El algoritmo a veces retorna el valor del guess por tanto se descartan estos
    if a not in guess:
        ENER.append(a)
        #print(a)

    if b not in guess:
        ENER.append(b)
        #print(b)

# Se quitan los valores repetidos
ENER = np.unique(ENER)
# Se organizan de menor a mayor
ENER = np.sort(ENER)
# Se toman solo los 4 primeros
ENER = ENER[:4]
```

Figure 8: Algoritmo para hallar los cuatro primeros autovalores de energía del pozo finito.

Con los autovalores se procede a resolver numéricamente la función de onda con la librería odeint de scipython. Para el pozo finito se obtuvieron los autovalores y funciones de onda

- $E_1 = 0.222087$
- $E_2 = 0.880048$
- $E_3 = 1.941936$
- $E_3 = 3.305328$

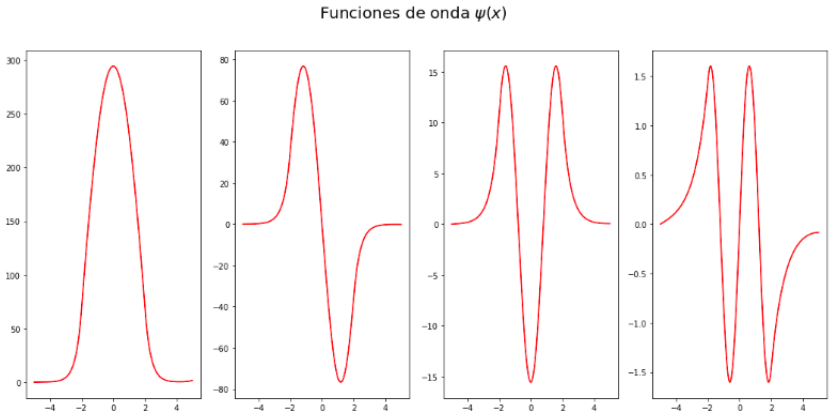

Figure 9: Cuatro primeras funciones de onda del pozo finito.

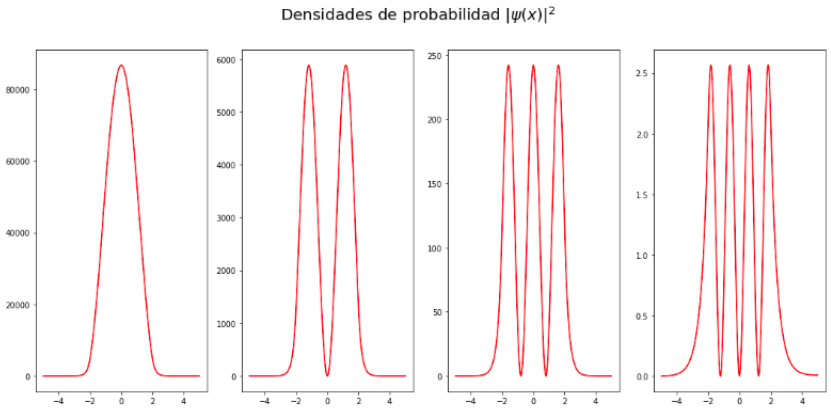

Figure 10: Densidades de probabilidad del pozo finito.

Para el pozo infinito se obtuvieron los autovalores y funciones de onda

- $E_1 = 0.049348$
- $E_2 = 0.197392$
- $E_3 = 0.444132$
- $E_3 = 0.789568$

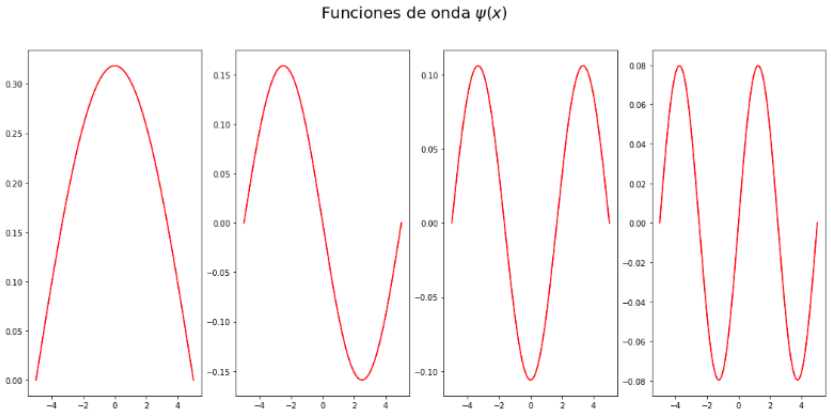

Figure 11: Autofunciones de pozo infinito.

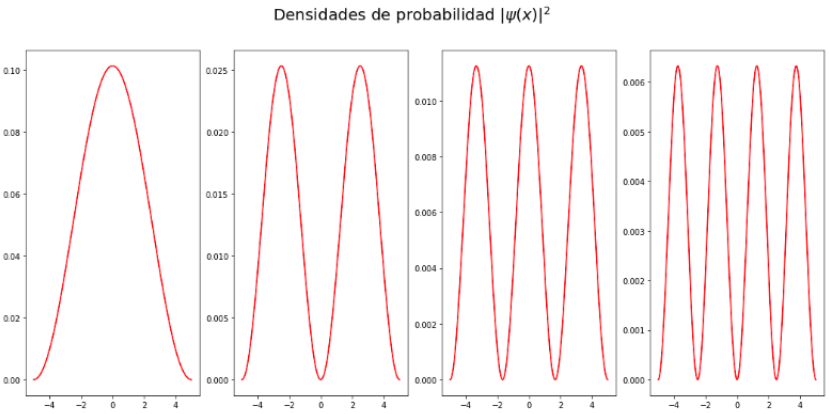

Figure 12: Densidad de probabilidad.

Para el potencial de oscilador armónico se obtuvieron los autovalores y funciones de onda:

- $E_1 = 0.707106$
- $E_2 = 2.121320$
- $E_3 = 3.535533$
- $E_3 = 4.949747$

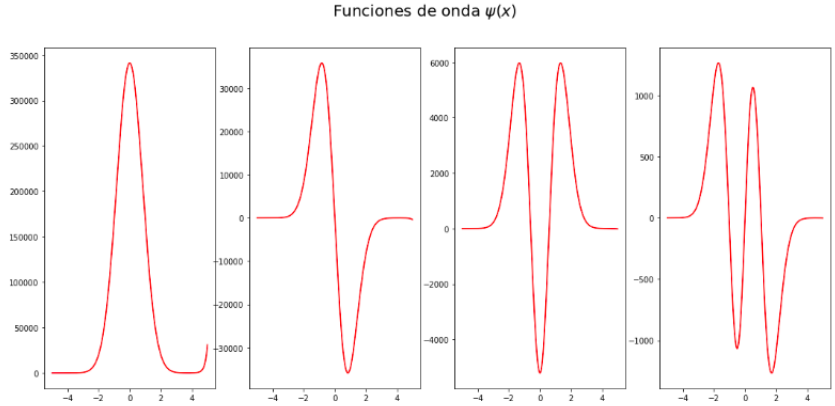

Figure 13: Autofunciones del potencial de oscilador armónico.

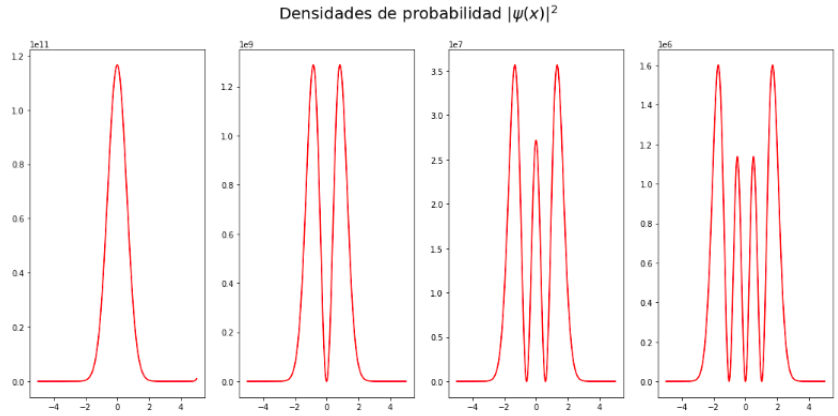

Figure 14: Densidades de probabilidad.

En adición, las autofunciones asociadas a los autovalores de energía expanden el espacio espacio de estados por superposición, y además, teniendo en cuenta una evolución espacio, cualquier función de estado posee representación,

$$\Psi(x,t) = \sum C_n \psi_n(x,t) e^{\frac{-itE_n}{\hbar}}$$

De modo que $\sum C_n^2 = 1$, pues la función pertenece al espacio de Hilbert.

El código usado en la programación se ve a continuación. Básicamente se almacenan en matrices los valores obtenidos de los autoestados y los auto valores y se determinan aleatoriamente siguiendo la condición sobre los coeficientes. Véase la figura 15.

```python
#Generador aleatorio de los coeficientes Cn
t = np.random.random((1,aa.shape[0]))
#norma de los coeficientes
norm = (t*t).sum()
t = t/(norm**0.5)
#sum coeficientes ==1, si desea comprobar
# print((t*t).sum())

#matriz diagonal cuyos autovalores estan dados por los términos exponenciales de las energías E_n
def mat_E(En,time):
    h = Planck
    #Matriz identidad de tamaño 4x4
    A = np.identity(4)
    for i in range(0,len(En)):
        n_complex = -1j
        #considerando la parte real de las exponenciales
        A[i] = A[i]*np.exp(n_complex*time*En[i]/h).real
    return A


fontP = FontProperties()

#arreglo de tiempos
time = np.linspace(0,2,5)

#graficación
fig, ax = plt.subplots(2,1,figsize=(8,6))
plt.subplots_adjust(hspace=0.5)


for k in time:
    #matriz diagonal con autovalores exponenciales de En
    En = mat_E(energy,k)
    #multiplicamos el Cada coeficiente exp con sus respectivos valores de autofunciones
    A = np.matmul(En,aa)
    #multiplicamos los coeficientes generados con la matriz anterio
    wf = np.matmul(t,A)
    plt.xlabel(r'$x$')
    ax[0].plot(val,(wf.T), label='time {:.1f}'.format(k))
    ax[0].set_title("Función de onda $\Psi (x,t)$", size=15)
    ax[1].plot(val,(wf.T)**2, label='time {:.1f}'.format(k))
    ax[1].set_title(r"Densidad de probabilidad $|\psi(x,t)|^2$" ,size=15)
    ax[0].legend(title='Arrays', bbox_to_anchor=(1.05, 1), loc='upper left', prop=fontP)
```

Figure 15: Algoritmo para el desarrollo de la parte temporal de Ψ

Ahora en las siguientes figuras se observa el estado resultante al haber sido expresado como combinación de los autoestados para diferentes instantes de tiempo y su respectiva densidad de probabilidad.

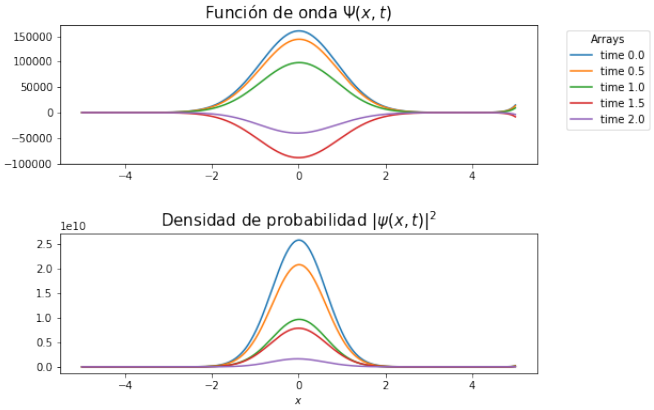

Figure 16: Función de estado y densidad de probabilidad para distintos tiempos. Oscilador armónico

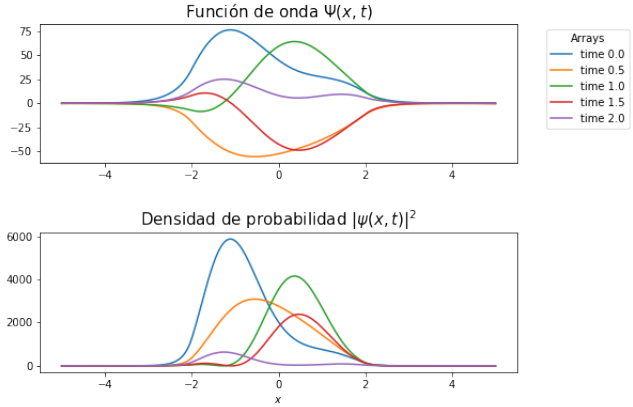

Figure 17: Función de estado y densidad de probabilidad para distintos tiempos. Pozo

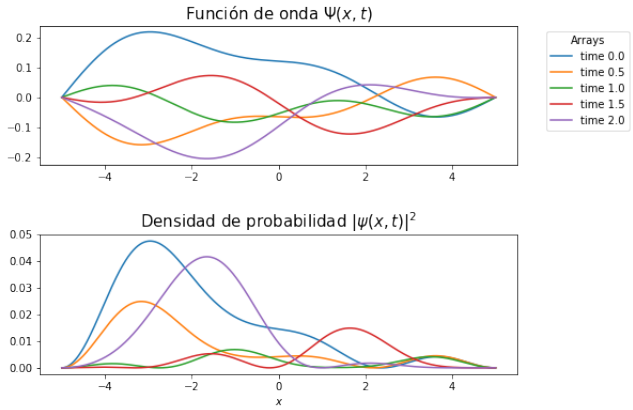

Figure 18: Función de estado y densidad de probabilidad para distintos tiempos. Pozo infinito