Ejemplo 1: Simulación de la Intensidad del Láser Cuántico

Vamos a modelar la intensidad de un láser cuántico, que puede ser afectada por efectos como la modulación de amplitud y frecuencia. Utilizaremos Python para calcular y graficar cómo cambia la intensidad del láser con el tiempo.

Intensidad del Láser:

Supongamos que la intensidad de un láser cuántico en el tiempo está dada por:

$$I(t) = I_0 \left[1 + \epsilon \cos(\omega_m t)\right] \cos^2(\omega t)$$

donde:

- I0I_0I0 es la intensidad máxima.
- ϵ\epsilonϵ es el índice de modulación.
- ωm\omega_mωm es la frecuencia de modulación.
- ω\omegaω es la frecuencia angular del láser.
- ttt es el tiempo.

Vamos a usar Python para graficar la intensidad I(t)I(t)I(t) en función del tiempo:

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

# Parámetros
I0 = 1.0              # Intensidad máxima
epsilon = 0.1         # Índice de modulación
omega = 2 * np.pi * 1  # Frecuencia del láser (1 Hz)
omega_m = 2 * np.pi * 0.5  # Frecuencia de modulación (0.5 Hz)
t = np.linspace(0, 10, 1000)  # Tiempo desde 0 a 10 segundos

# Intensidad del láser
I_t = I0 * (1 + epsilon * np.cos(omega_m * t)) * np.cos(omega
* t)**2

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(t, I_t, label='Intensidad del láser')
plt.xlabel('Tiempo (s)')
plt.ylabel('Intensidad')
plt.title('Intensidad de un Láser Cuántico en función del
Tiempo')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Parámetros**: Definimos la intensidad máxima, el índice de modulación, las frecuencias y el rango de tiempo.
- **Cálculo**: Calculamos la intensidad en función del tiempo.
- **Graficar**: Utilizamos matplotlib para visualizar cómo varía la intensidad del láser.

Ejemplo 2: Simulación de la Potencia del Láser Cuántico

La potencia del láser puede variar en función de la longitud de onda y otros parámetros. Supongamos que la potencia de un láser cuántico está dada por:

$$P(\lambda) = P_0 \left(\frac{\lambda_0}{\lambda} \right)^2$$

donde:

- P0P_0P0 es la potencia máxima.
- λ0\lambda_0λ0 es la longitud de onda central.
- λ\lambdaλ es la longitud de onda variable.

Vamos a usar Python para graficar la potencia en función de la longitud de onda.

```
# Parámetros
P0 = 10.0            # Potencia máxima
lambda_0 = 800e-9     # Longitud de onda central (800 nm)
lambda_range = np.linspace(700e-9, 900e-9, 1000)  # Rango de
longitudes de onda desde 700 nm a 900 nm

# Potencia en función de la longitud de onda
P_lambda = P0 * (lambda_0 / lambda_range)**2

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(lambda_range * 1e9, P_lambda, label='Potencia del
láser')
plt.xlabel('Longitud de onda (nm)')
plt.ylabel('Potencia')
plt.title('Potencia del Láser Cuántico en función de la
Longitud de Onda')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Parámetros**: Definimos la potencia máxima y la longitud de onda central.
- **Cálculo**: Calculamos la potencia en función de la longitud de onda.
- **Graficar**: Utilizamos matplotlib para visualizar cómo cambia la potencia del láser con la longitud de onda.

Ejemplo 3: Automatización del Control de un Láser Cuántico

Vamos a automatizar el ajuste de la potencia de un láser cuántico en función del tiempo utilizando Python. Imaginemos que queremos ajustar la potencia del láser para seguir un perfil de modulación dado.

Perfil de Modulación:

Supongamos que la potencia del láser sigue un perfil sinusoidal:

$$P(t) = P_{\text{base}} + P_{\text{mod}} \sin(\omega_m t)$$

donde:

- PbaseP_{\text{base}}Pbase es la potencia base.
- PmodP_{\text{mod}}Pmod es la amplitud de modulación.
- ωm\omega_mωm es la frecuencia de modulación.
- t es el tiempo.\

Usaremos un controlador de láser simulado para ajustar la potencia en función de este perfil:

```
import time

# Parámetros
P_base = 5.0          # Potencia base
P_mod = 2.0           # Amplitud de modulación
omega_m = 2 * np.pi * 0.1  # Frecuencia de modulación (0.1 Hz)
t_max = 10            # Tiempo máximo de operación

def set_laser_power(power):
    # Simula el ajuste de la potencia del láser
    print(f"Potencia ajustada a: {power:.2f} unidades")

# Simulación del ajuste de potencia
t = 0
while t < t_max:
    P_t = P_base + P_mod * np.sin(omega_m * t)
    set_laser_power(P_t)
    time.sleep(1)  # Espera 1 segundo
    t += 1  # Incrementa el tiempo
```

```
print("Simulación completada.")
```

Explicación:

- **Función set_laser_power**: Simula el ajuste de la potencia del láser.
- **Bucle de Simulación**: Ajusta la potencia en función del perfil sinusoidal y espera 1 segundo entre ajustes.

Ejemplo 4: Simulación del Perfil de Emisión de un Láser Cuántico

Para un láser cuántico, la función de emisión en función de la frecuencia puede ser importante. Supongamos que la emisión sigue una distribución gaussiana centrada en una frecuencia ν0\nu_0ν0.

$$E(\nu) = E_0 \exp\left(-\frac{(\nu - \nu_0)^2}{2\sigma^2}\right)$$

donde:

- E0E_0E0 es la intensidad máxima.
- ν0\nu_0ν0 es la frecuencia central.
- σ\sigmaσ es el ancho de la distribución en frecuencia.

Vamos a graficar el perfil de emisión en función de la frecuencia.

```
# Parámetros
E0 = 1.0              # Intensidad máxima
nu_0 = 5.0            # Frecuencia central
sigma = 1.0           # Ancho de la distribución en frecuencia
nu_range = np.linspace(0, 10, 1000)  # Rango de frecuencias

# Perfil de emisión
E_nu = E0 * np.exp(- (nu_range - nu_0)**2 / (2 * sigma**2))

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(nu_range, E_nu, label='Perfil de Emisión del Láser')
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Emisión')
plt.title('Perfil de Emisión de un Láser Cuántico en función
de la Frecuencia')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Parámetros**: Definimos la intensidad máxima, la frecuencia central y el ancho de la distribución.
- **Cálculo**: Calculamos el perfil de emisión en función de la frecuencia.

- **Graficar**: Utilizamos matplotlib para visualizar el perfil de emisión del láser.

Resumen

Estos ejemplos muestran cómo modelar y simular aspectos de un láser cuántico utilizando Python. Desde la simulación de la intensidad y potencia del láser hasta la automatización del control y el perfil de emisión, estos ejemplos proporcionan una base sólida para trabajar con láseres cuánticos en un entorno de programación.

Ejemplo 1: Generación y Detección de Fotones

Simularemos un láser cuántico que emite fotones y cómo se pueden contar estos fotones. Para simplificar, supongamos que el número de fotones emitidos sigue una distribución de Poisson.

Distribución de Poisson:

La distribución de Poisson se usa para modelar el número de eventos (fotones emitidos) en un intervalo de tiempo dado. La probabilidad de observar kkk fotones cuando el promedio es λ\lambdaλ está dada por:

$$P(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

donde λ\lambdaλ es la tasa media de emisión de fotones:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

# Parámetros
lambda_rate = 5   # Tasa media de emisión de fotones
k = np.arange(0, 20)  # Número de fotones

# Distribución de Poisson
probabilities = poisson.pmf(k, lambda_rate)

# Graficar
plt.figure(figsize=(10, 6))
plt.bar(k, probabilities, alpha=0.7, color='b',
label='Distribución de Poisson')
plt.xlabel('Número de Fotones')
plt.ylabel('Probabilidad')
plt.title('Distribución de Fotones Emitidos por un Láser
Cuántico')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Parámetros**: Definimos la tasa media de emisión de fotones.
- **Distribución**: Calculamos la probabilidad de observar diferentes números de fotones.
- **Graficar**: Utilizamos matplotlib para visualizar la distribución de fotones.

Ejemplo 2: Simulación de la Interferencia de Fotones

Simularemos la interferencia de fotones en un experimento de doble rendija, un experimento fundamental en la óptica cuántica.

Interferencia de Doble Rendija:

Para fotones que pasan por dos rendijas, la intensidad de la luz en función de la posición en una pantalla se describe por la interferencia constructiva y destructiva.

La intensidad en una pantalla está dada por:

$$I(x) = I_0 \left[\cos^2 \left(\frac{kdx}{2L} \right) \right]$$

donde:

- I0I_0I0 es la intensidad máxima.
- ddd es la separación entre las rendijas.
- LLL es la distancia a la pantalla.
- kkk es el número de onda.

Ejemplo en código:

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

# Parámetros
I0 = 1.0          # Intensidad máxima
d = 0.01          # Separación entre rendijas (m)
L = 1.0           # Distancia a la pantalla (m)
wavelength = 500e-9  # Longitud de onda (500 nm)
k = 2 * np.pi / wavelength
x = np.linspace(-0.1, 0.1, 1000)  # Posiciones en la pantalla

# Intensidad en función de la posición
I_x = I0 * np.cos(k * d * x / (2 * L))**2

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(x, I_x, label='Intensidad de Interferencia')
plt.xlabel('Posición en la pantalla (m)')
plt.ylabel('Intensidad')
plt.title('Interferencia de Fotones en un Experimento de
Doble Rendija')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Parámetros**: Definimos la separación entre rendijas, la distancia a la pantalla y la longitud de onda.
- **Cálculo**: Calculamos la intensidad en función de la posición en la pantalla.
- **Graficar**: Visualizamos el patrón de interferencia.
- **Ejemplo 3: Automatización del Registro de Eventos de Fotones**

  Simularemos el registro de eventos de detección de fotones en un detector cuántico. Vamos a automatizar la recolección de datos sobre la detección de fotones en diferentes intervalos de tiempo.

- **Registro de Eventos:**
  Supongamos que queremos registrar eventos de detección de fotones con una tasa promedio de emisión. Simularemos estos eventos y almacenaremos los resultados en un archivo.

```
import numpy as np
import pandas as pd
from scipy.stats import poisson

# Parámetros
lambda_rate = 5    # Tasa media de emisión de fotones
duration = 10      # Duración del experimento en segundos
interval = 1       # Intervalo de muestreo en segundos
times = np.arange(0, duration, interval)

# Registro de eventos
events = [poisson.rvs(lambda_rate * interval) for _ in times]

# Crear DataFrame
df = pd.DataFrame({'Tiempo (s)': times, 'Número de Fotones':
events})

# Guardar en archivo CSV
df.to_csv('registro_fotones.csv', index=False)

print("Registro de eventos guardado en
'registro_fotones.csv'")
```

Explicación:

- **Parámetros**: Definimos la tasa media de emisión y los parámetros del experimento.
- **Registro**: Simulamos la detección de fotones en intervalos de tiempo.
- **Guardar**: Almacenamos los resultados en un archivo CSV usando pandas.

Ejemplo 4: Simulación del Estado de Partículas Cuánticas

Simularemos un sistema de partículas cuánticas en un estado de superposición utilizando la mecánica cuántica. En este caso, consideraremos un sistema de dos niveles (estados |0⟩|0\rangle|0⟩ y |1⟩|1\rangle|1⟩).

Estado de Superposición:

El estado de una partícula en superposición puede describirse como:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

donde α y β son coeficientes complejos que cumplen $|\alpha|^2 + |\beta|^2 = 1$.

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

# Parámetros
alpha = np.exp(1j * np.pi / 4) / np.sqrt(2)  # Coeficiente
para |0>
beta = np.exp(1j * np.pi / 4) / np.sqrt(2)   # Coeficiente
para |1>

# Probabilidades
prob_0 = np.abs(alpha)**2
prob_1 = np.abs(beta)**2

# Graficar
plt.figure(figsize=(8, 6))
plt.bar(['|0⟩', '|1⟩'], [prob_0, prob_1], color=['blue',
'red'])
plt.xlabel('Estado')
plt.ylabel('Probabilidad')
plt.title('Probabilidad de Medición en un Estado de
Superposición')
plt.grid(True)
plt.show()
```

Explicación:

- **Coeficientes**: Definimos los coeficientes de los estados en superposición.
- **Probabilidades**: Calculamos las probabilidades de encontrar la partícula en cada estado.
- **Graficar**: Visualizamos las probabilidades de medición.

Resumen

Estos ejemplos cubren diversas facetas de la automatización y simulación de láseres cuánticos y fotones en Python. Desde la simulación de la distribución de fotones hasta la interferencia y el registro de eventos, estos ejemplos proporcionan una base sólida para explorar aplicaciones cuánticas en programación.

Ejemplo: Simulación Simple de Emisión de Fotones en Intervalos de Tiempo

Supongamos que un láser cuántico emite fotones a una tasa promedio constante. Queremos simular la cantidad de fotones detectados en intervalos de tiempo discretos y visualizar los resultados.

Modelo:

Vamos a usar una distribución de Poisson para modelar el número de fotones detectados en cada intervalo de tiempo. La distribución de Poisson es adecuada para este propósito ya que se utiliza comúnmente para describir eventos que ocurren de manera independiente con una tasa promedio constante.

Objetivos:

1. Simular la detección de fotones en intervalos de tiempo.
2. Visualizar el número de fotones detectados en un gráfico de barras.

Aquí está el código en Python:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

# Parámetros
lambda_rate = 10  # Tasa media de emisión de fotones por
intervalo
num_intervals = 20  # Número de intervalos de tiempo
intervals = np.arange(num_intervals)  # Intervalos de tiempo
(0, 1, 2, ..., 19)

# Simular el número de fotones detectados en cada intervalo
detected_photons = poisson.rvs(lambda_rate,
size=num_intervals)

# Graficar el número de fotones detectados
plt.figure(figsize=(10, 6))
plt.bar(intervals, detected_photons, color='skyblue',
edgecolor='black')
plt.xlabel('Intervalo de Tiempo')
plt.ylabel('Número de Fotones Detectados')
plt.title('Número de Fotones Detectados en Diferentes
Intervalos de Tiempo')
plt.grid(True)
plt.show()
```

Explicación:

1. **Parámetros:**
    - lambda_rate: La tasa media de emisión de fotones por intervalo de tiempo.

- num_intervals: El número de intervalos de tiempo para los cuales simularemos la detección de fotones.

2. **Simulación:**
   - Utilizamos la función poisson.rvs de scipy.stats para simular el número de fotones detectados en cada intervalo de tiempo. Esto genera una serie de números aleatorios que siguen una distribución de Poisson con la tasa media especificada.
3. **Visualización:**
   - Creamos un gráfico de barras con matplotlib para mostrar el número de fotones detectados en cada intervalo de tiempo. El gráfico proporciona una visión clara de cómo varía el número de fotones detectados a lo largo de los intervalos de tiempo.

Síntesis:

Sintetizar el cálculo manual de la segunda derivada de la función de onda en Python. La función de onda dada es:

$$\psi_n(x) = \frac{2}{L} \sin\left(\frac{n\pi x}{L}\right)$$

Cálculo Manual

1. **Primera Derivada**

La función de onda es:

ψn(x)=2Lsin(nπxL)\psi_n(x) = \frac{2}{L} \sin\left(\frac{n \pi x}{L}\right)

$$\psi_n(x) = \frac{2}{L} \sin\left(\frac{n\pi x}{L}\right)$$

Para encontrar la primera derivada respecto a x:

$$\frac{\partial}{\partial x}\psi_n(x) = \frac{2}{L} \cdot \frac{n\pi}{L} \cos\left(\frac{n\pi x}{L}\right)$$

Simplificando:

Simplificando:

$$\frac{\partial}{\partial x}\psi_n(x) = \frac{2n\pi}{L^2} \cos\left(\frac{n\pi x}{L}\right)$$

Implementación en Python

Ahora implementemos esto en Python para verificar los cálculos.

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

# Definimos los parámetros
L = 1.0    # Longitud del intervalo
n = 1      # Número cuántico

# Función de onda
def psi_n(x, L, n):
    return (2 / L) * np.sin(n * np.pi * x / L)

# Primera derivada
def dpsi_n_dx(x, L, n):
    return (2 * n * np.pi / L**2) * np.cos(n * np.pi * x / L)

# Segunda derivada
def d2psi_n_dx2(x, L, n):
    return - (2 * n**2 * np.pi**2 / L**3) * np.sin(n * np.pi
* x / L)

# Generar valores de x
x = np.linspace(0, L, 500)

# Calcular valores de la función de onda, primera y segunda
derivada
psi_values = psi_n(x, L, n)
dpsi_values = dpsi_n_dx(x, L, n)
d2psi_values = d2psi_n_dx2(x, L, n)

# Graficar
plt.figure(figsize=(12, 6))

plt.subplot(3, 1, 1)
plt.plot(x, psi_values, label=r'$\psi_n(x)$', color='b')
plt.title('Función de Onda $\psi_n(x)$')
plt.xlabel('$x$')
plt.ylabel(r'$\psi_n(x)$')
plt.grid(True)

plt.subplot(3, 1, 2)
plt.plot(x, dpsi_values, label=r'$\frac{\partial
\psi_n(x)}{\partial x}$', color='r')
plt.title('Primera Derivada $\frac{\partial
\psi_n(x)}{\partial x}$')
plt.xlabel('$x$')
plt.ylabel(r'$\frac{\partial \psi_n(x)}{\partial x}$')
plt.grid(True)

plt.subplot(3, 1, 3)
```

```
plt.plot(x, d2psi_values, label=r'$\frac{\partial^2
\psi_n(x)}{\partial x^2}$', color='g')
plt.title('Segunda Derivada $\frac{\partial^2
\psi_n(x)}{\partial x^2}$')
plt.xlabel('$x$')
plt.ylabel(r'$\frac{\partial^2 \psi_n(x)}{\partial x^2}$')
plt.grid(True)

plt.tight_layout()
plt.show()
```

Explicación del Código

1. **Definición de Funciones:**
    - psi_n(x, L, n): Calcula la función de onda ψn(x)\psi_n(x)ψn(x).
    - dpsi_n_dx(x, L, n): Calcula la primera derivada de ψn(x)\psi_n(x)ψn(x).
    - d2psi_n_dx2(x, L, n): Calcula la segunda derivada de ψn(x)\psi_n(x)ψn(x).
2. **Generación de Valores:**
    - Se genera un conjunto de valores para xxx y se calculan los valores correspondientes para ψn(x)\psi_n(x)ψn(x), su primera derivada y su segunda derivada.
3. **Visualización:**
    - Se grafican la función de onda, la primera derivada y la segunda derivada para visualizar cómo cambian con xxx.

Este código te permite ver gráficamente la función de onda, así como sus derivadas primera y segunda, facilitando la comprensión de cómo cambia la función en función de xxx.

Schrödinger, el **Enfoque de Heisenberg** y el concepto de **Interacción**.

Conceptos Claves

1. **Enfoque de Schrödinger**
    - **Definición:**
        - En este enfoque, los operadores observables son estáticos, mientras que el estado cuántico ∣ψ(t)⟩|\psi(t)\rangle∣ψ(t)⟩ evoluciona en el tiempo.
    - **Ecuaciones:**
        - Evolución del estado: ∣ψ(t)⟩=e−iHtℏ∣ψ(0)⟩|\psi(t)\rangle = e^{-\frac{iHt}{\hbar}} |\psi(0)\rangle∣ψ(t)⟩=e−ℏiHt∣ψ(0)⟩
        - Valor esperado: ⟨A(t)⟩=⟨ψ(t)∣A∣ψ(t)⟩\langle A(t) \rangle = \langle \psi(t) | A | \psi(t) \rangle⟨A(t)⟩=⟨ψ(t)∣A∣ψ(t)⟩
    - **Variables:**

        - HHH: Hamiltoniano del sistema.
        - ħ\hbarħ: Constante de Planck reducida.
        - AAA: Operador observable.

2. **Enfoque de Heisenberg**
    - **Definición:**
        - En este enfoque, los estados cuánticos |ψ(t)⟩|\psi(t)\rangle|ψ(t)⟩ son estáticos y los operadores observables evolucionan en el tiempo.
    -

- Ecuaciones:
    - Evolución del operador: $A_h(t) = e^{\frac{iHt}{\hbar}} A e^{-\frac{iHt}{\hbar}}$
    - Valor esperado: $\langle A(t) \rangle = \langle \psi(0) | A_h(t) | \psi(0) \rangle$
    - Ecuación de Heisenberg: $i\hbar \frac{\partial A_h(t)}{\partial t} = [A_h(t), H]$

- Variables:
    - $A_h(t)$: Operador observable en el enfoque de Heisenberg.
    - $[A_h(t), H]$: Conmutador entre el operador y el Hamiltoniano.

1. **Enfoque de Interacción (en general)**
    - **Definición:**
        - En el enfoque de interacción, tanto los estados cuánticos como los operadores pueden evolucionar en el tiempo, pero este enfoque se suele usar para manejar interacciones en sistemas cuánticos complicados.

Ejemplo Práctico en Python

Vamos a implementar ejemplos simples en Python para calcular la evolución del estado en el Enfoque de Schrödinger y la evolución del operador en el Enfoque de Heisenberg.

1. Enfoque de Schrödinger

Objetivo: Simular la evolución del estado |ψ(t)⟩|\psi(t)\rangle|ψ(t)⟩ bajo el Hamiltoniano HHH en el tiempo.

Código Python:

Python:

```
import numpy as np
```

```
import matplotlib.pyplot as plt

# Parámetros
hbar = 1.0  # Constante de Planck reducida
H = np.array([[1, 0], [0, -1]])  # Hamiltoniano 2x2 para
simplicidad
psi_0 = np.array([1, 0])  # Estado inicial

# Evolución del estado
def evolve_state(H, psi_0, t, hbar):
    U = np.linalg.matrix_exponential(-1j * H * t / hbar)  #
Operador de evolución
    return U @ psi_0

# Tiempo
t_values = np.linspace(0, 10, 100)
psi_t_values = [evolve_state(H, psi_0, t, hbar) for t in
t_values]

# Magnitudes de los componentes del estado
psi_t_norms = [np.linalg.norm(psi_t) for psi_t in
psi_t_values]

# Graficar
plt.figure(figsize=(8, 6))
plt.plot(t_values, psi_t_norms, label='Norma del estado
$\psi(t)$')
plt.xlabel('Tiempo $t$')
plt.ylabel('Norma de $|\psi(t)\rangle$')
plt.title('Evolución del Estado Cuántico en el Enfoque de
Schrödinger')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Hamiltoniano (H)**: Definimos un Hamiltoniano simple.
- **Evolución del Estado**: Calculamos la evolución del estado |ψ(t)⟩|\psi(t)\rangle|ψ(t)⟩ usando el operador de evolución U(t)U(t)U(t).
- **Visualización**: Graficamos la norma del estado a lo largo del tiempo.

2. Enfoque de Heisenberg

Objetivo: Calcular la evolución del operador Ah(t)A_h(t)Ah(t) en el tiempo y su valor esperado.

Código Python:

```
from scipy.linalg import expm

# Operador observable
A = np.array([[0, 1], [1, 0]])  # Operador de Pauli X para
simplicidad

# Evolución del operador
def evolve_operator(A, H, t, hbar):
    A_h = expm(1j * H * t / hbar) @ A @ expm(-1j * H * t /
hbar)
    return A_h

# Tiempo
t_values = np.linspace(0, 10, 100)
A_t_values = [evolve_operator(A, H, t, hbar) for t in
t_values]

# Valor esperado
def expectation_value(A, psi):
    return np.conj(psi).T @ A @ psi

# Estado fijo
psi_0 = np.array([1, 0])

# Calcular valores esperados
expectation_values = [expectation_value(A_h, psi_0) for A_h
in A_t_values]

# Graficar
plt.figure(figsize=(8, 6))
plt.plot(t_values, np.real(expectation_values), label='Valor
Esperado $\langle A(t) \rangle$')
plt.xlabel('Tiempo $t$')
plt.ylabel('Valor Esperado')
plt.title('Evolución del Valor Esperado en el Enfoque de
Heisenberg')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Operador Observable (A)**: Definimos un operador simple.
- **Evolución del Operador**: Calculamos la evolución del operador usando la evolución de Heisenberg.
- **Valor Esperado**: Calculamos y graficamos el valor esperado del operador.

Resumen

- **Enfoque de Schrödinger**: Los estados cuánticos evolucionan en el tiempo, mientras que los operadores observables son estáticos.
- **Enfoque de Heisenberg**: Los operadores observables evolucionan en el tiempo, mientras que los estados cuánticos son estáticos.

- **Python**: Utilizamos bibliotecas como numpy y matplotlib para calcular y visualizar la evolución de estados y operadores en ambos enfoques.

Estos ejemplos te permiten ver cómo se puede programar la dinámica cuántica utilizando Python y entender cómo los conceptos se traducen en cálculos prácticos.

Explicación:

- **Hamiltoniano (H)**: Es una matriz simple que define la energía del sistema.
- **Evolución del Estado**: Calculamos la evolución del estado |ψ(t)⟩|\psi(t)\rangle|ψ(t)⟩ usando la matriz exponencial del Hamiltoniano.
- **Visualización**: Graficamos las probabilidades de los componentes del estado cuántico en función del tiempo.

2. Enfoque de Heisenberg

Vamos a calcular la evolución de un operador observable usando el enfoque de Heisenberg. Utilizaremos un operador simple para ilustrar el concepto.

Objetivo: Calcular cómo cambia un operador observable en el tiempo y su valor esperado en el enfoque de Heisenberg.

Código Python:

python

```
import numpy as np

from scipy.linalg import expm

import matplotlib.pyplot as plt


# Parámetros

hbar = 1.0  # Constante de Planck reducida

H = np.array([[1, 0], [0, -1]])  # Hamiltoniano simple 2x2

A = np.array([[0, 1], [1, 0]])  # Operador observable
(Pauli-X)


# Función para evolucionar el operador

def evolve_operator(A, H, t, hbar):

    A_h = expm(1j * H * t / hbar) @ A @ expm(-1j * H * t /
hbar)
```

```
    return A_h


# Tiempo
t_values = np.linspace(0, 10, 100)
A_t_values = [evolve_operator(A, H, t, hbar) for t in
t_values]


# Valor esperado
def expectation_value(A, psi):
    return np.real(np.conj(psi).T @ A @ psi)


# Estado fijo
psi_0 = np.array([1, 0])


# Calcular valores esperados
expectation_values = [expectation_value(A_h, psi_0) for A_h in
A_t_values]


# Graficar
plt.figure(figsize=(8, 6))
plt.plot(t_values, expectation_values, label='Valor Esperado
$\\langle A(t) \\rangle$')
plt.xlabel('Tiempo $t$')
plt.ylabel('Valor Esperado')
plt.title('Evolución del Valor Esperado del Operador en el
Enfoque de Heisenberg')
plt.legend()
plt.grid(True)
plt.show()
```

Explicación:

- **Operador Observable (A)**: Definimos un operador simple (Pauli-X).
- **Evolución del Operador**: Calculamos la evolución del operador usando la matriz exponencial del Hamiltoniano.

- **Valor Esperado**: Calculamos el valor esperado del operador en el estado fijo ∣ψ0⟩|\psi_0\rangle∣ψ0⟩ y lo graficamos.

Resumen

- **Enfoque de Schrödinger**: Se calcula la evolución temporal del estado cuántico ∣ψ(t)⟩|\psi(t)\rangle∣ψ(t)⟩ usando un Hamiltoniano.
- **Enfoque de Heisenberg**: Se calcula cómo cambia el operador observable A(t)A(t)A(t) en el tiempo usando el Hamiltoniano y se evalúa su valor esperado.

Estos ejemplos simplificados ilustran cómo puedes implementar estos conceptos cuánticos en Python para obtener y visualizar resultados prácticos. Puedes modificar los parámetros y operadores para experimentar con diferentes sistemas cuánticos.