Particle in a box

Let's apply the whole thing to the problem of a particle in a box. This means, we look at a quantum mechanical object in a potential well.

```
#| edit: false
#| echo: false
#| execute: true

import numpy as np
from scipy.sparse import diags
from scipy.sparse.linalg import eigsh
import matplotlib.pyplot as plt
# Set default plotting parameters
plt.rcParams.update({
    'font.size': 12,
    'lines.linewidth': 1,
    'lines.markersize': 5,
    'axes.labelsize': 11,
    'xtick.labelsize': 10,
    'ytick.labelsize': 10,
    'xtick.top': True,
    'xtick.direction': 'in',
    'ytick.right': True,
    'ytick.direction': 'in',
})

def get_size(w, h):
    return (w/2.54, h/2.54)
```

The problem is sketched below.

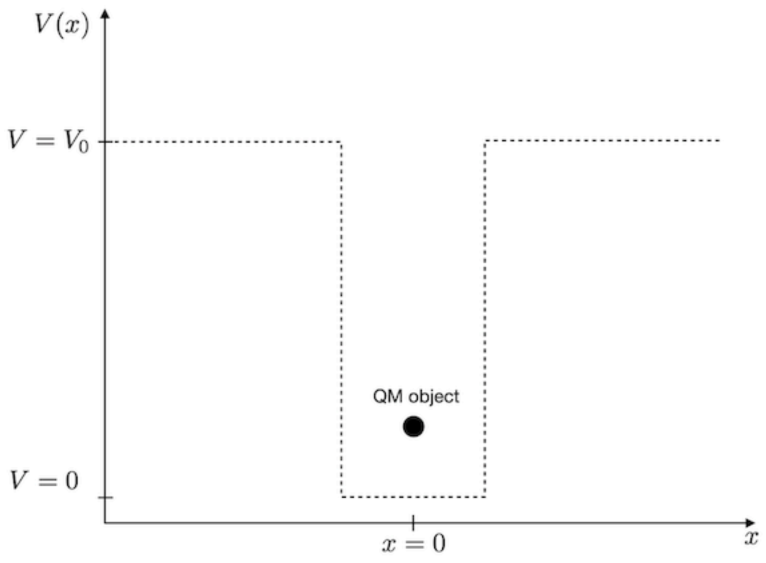

Figure 1: Particle in a Box

We need to define this rectangular box with zero potential energy inside the box and finite potential energy outside. Since the quantum mechanical object is a wave, we expect that only certain standing waves of particular wavelength can exist inside the box. These waves are connected to certain probability densities of finding the particle at certain positions and specific energy values. These are the energy levels, which are often characteristic of the quantum realm.

Definition of the problem

Before we start, we need to define some quantities:

- we will study a box of d=1 nm in width in a domain of L=10 nm
- we will use N=1001 points for our x_i
- our potential energy shall have a barrier height of 1 eV
- the potential energy inside the box will be zero

```
#| autorun: false
# Fundamental physical constants
# define some useful constants
hbar=1.678e-35 #joule seconds
m_e=9.10938356e-31 # kg
V_0=1.602e-19/2 # J
```

```
N=1001

d=1e-9 # m
L= 10e-9 #m

x = np.linspace(-L/2, L/2, N)
dx = x[1] - x[0]
```

Potential energy

We first define the diagonal potential energy matrix. The potential is zero inside the box and V_0 outside.

```
#| autorun: false
U_vec = np.zeros(N)
U_vec[np.abs(x)>d/2]=V_0

# potential energy is only on the diagonal, no deritvative
U = diags([U_vec], [0])
```

Kinetic energy

Next we construct the kinetic energy matrix using the finite difference representation of the second derivative.

```
#| autorun: false
# Kinetic energy matrix using finite differences
# T = - ²/(2m) * d²/dx²

T = -hbar**2 * diags([-2., 1., 1.], [0, -1, 1], shape=(N, N)) / (dx**2 * 2 * m_e)
```

Hamiltonian and boundary conditions

Finally, we construct the total Hamiltonian operator matrix. For hard wall boundary conditions, we need to ensure that the wavefunction vanishes at the boundaries.

```
#| autorun: false
# Sum of kinetic and potential energy
H = T + U
```

Solution

The last step is to solve the eigenvalue problem using the `eigsh` method from `scipy`. We use `which='SM'` to find the smallest magnitude eigenvalues, which correspond to the lowest energy states.

```
#| autorun: false
# diagonalize the matrix and take the first n eigenvalues and eigenvectors
n=20
vals, vecs = eigsh(H, k=n,which='SM')
```

Plotting the results

Let's visualize the energy levels and wavefunctions:

```
#| autorun: false
# define some scaling to make a nice plot
scale=1e9 # position scale
escale=6.242e18 # energy scale in eV
psiscale=5 # wavefunction scale

plt.figure(figsize=get_size(10,8))

# Plot the potential
plt.plot(x*scale, U_vec*escale, 'k-')

for k in range(11):
    vec = vecs[:, k]
    mag = np.sqrt(np.dot(vecs[:, k],vecs[:, k]))
    vec = vec/mag
    plt.axhline(y=vals[k]*escale,ls='--')
    plt.plot(x*scale, psiscale*np.real(vec)**2+vals[k]*escale)

plt.xlabel(r"position $x$ [nm]")
plt.ylabel(r"energy $E$ in eV, Wavefunction $\Psi(x)$")
plt.xlim([-L/4*scale,L/4*scale])

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

The diagram shows the corresponding energy states (the eigenvalues of the solution) and the value $|\Psi|^2$, which gives the probability to find the particle inside the box. The latter shows that, in contrast to what we expect from classical theory, where we would expect the particle to be with equal probability found at all positions inside the box, we get in quantum mechanics only certain positions at which we would find the particle. Also, the higher the energy state, the more equally is the particle distributed over the box. For a finite box depth, however, we get only a finite number of energy states in which the particle is bound.

A second interesting observation here is that the particle has a finite probability to enter the potential barrier. Especially for the higher energy states, the wavefunction decays exponentially into the barrier. This is similar to the evanescent wave we studied during the last lecture.

Comparison with analytical solution

For an infinite potential well, we can compare our numerical results with the analytical solution. The analytical energy levels for an infinite well are:

$$E_n = \frac{n^2\pi^2\hbar^2}{2md^2}$$

Let's compare:

```
#| autorun: false

fig = plt.figure(figsize=get_size(10,8))
plt.xlabel('quantum number n')
plt.ylabel(r"$E$ [eV]")

# Plot numerical solution
quantum_numbers = np.array(range(len(vals[:11])))+1
plt.plot(quantum_numbers, vals[:11]*escale, 'o', alpha=0.3, label='Numerical')


# E_n = (n² ²ħ²)/(2md²)
analytical_energies = [(n**2 * np.pi**2 * hbar**2)/(2 * m_e * d**2) * escale for n in quantu
plt.plot(quantum_numbers, analytical_energies, '.', color='red', alpha=0.7, label='Analytica


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

Energies of bound states

In the case of the particle in a box, only certain energies are allowed. The energies which correspond to these distributions are increasing nonlinearly with its index. Below we plot the energy as a function of the index of the energy value. This index is called *quantum number* as we can enumerate the solutions in quantum mechanics. The graph shows that the energy of the bound states increases with the square of the quantum number, i.e. $E_n \propto n^2$.

```
#| autorun: false
# Plot energy vs quantum number with quadratic fit
fig = plt.figure(figsize=get_size(10, 8))

# Plot numerical results
quantum_numbers = np.arange(1, len(vals[:11]) + 1)
energies = vals[:8] * escale
plt.scatter(quantum_numbers, energies, s=50, alpha=0.7, label='Numerical')

# Fit a quadratic function
coeffs = np.polyfit(quantum_numbers, energies, 2)
fit_x = np.linspace(1, 8, 100)
fit_y = np.polyval(coeffs, fit_x)
plt.plot(fit_x, fit_y, 'r-', linewidth=2,
         label=f'Fit: E = {coeffs[0]:.3f}n² + {coeffs[1]:.3f}n + {coeffs[2]:.3f}')

plt.xlabel('quantum Number n')
plt.ylabel('energy [eV]')
plt.tight_layout()
plt.show()
```

Where to go from here?

You may try at this point to create two closely spaced potential wells, e.g. two of 1 nm width with a distance of 0.1 nm or with a distance of 2 nm. You should see that for large distances of the wells the energy values in the individual wells are the same, while for the smaller distance they split up into two due to the interaction.

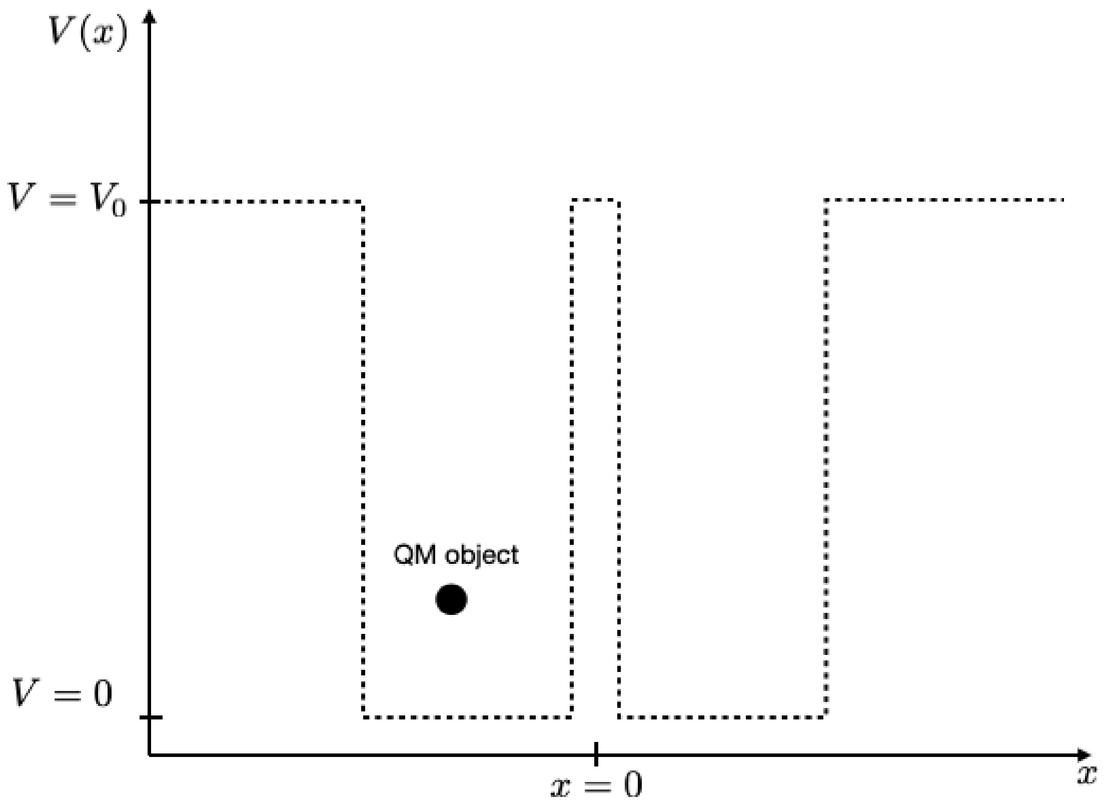

Figure 2: Double Well

Here's a starting point for exploring a double well:

```python
#| autorun: false
# Example: Create a double well potential
def double_well_potential(x, well_width=1e-9, separation=0.5e-9, V0=1.0*eV):
    """Create a double well potential"""
    V = np.ones_like(x) * V0
    # First well
    V[np.abs(x + separation/2) < well_width/2] = 0
    # Second well
    V[np.abs(x - separation/2) < well_width/2] = 0
    return V

# Try it out!
# V_double = double_well_potential(x)
# U_double = diags([V_double], [0])
# H_double = T + U_double
# ... continue with eigenvalue calculation
```