PROBLEM 3 - Solution of non-stationary Schrödinger Equation using the Crank-Nicolson Method

Gisela Martí Guerrero, ICC 2023

This program simulates the time evolution of a quantum wave packet using the Schrödinger equation in one dimension, for the reaction $H + H_2$. It uses the Crank-Nicolson method, which is a numerical techniqu used for solving time-dependent partial differential equations such as the Schrödinger equation. The method is implicit and combines information from the current and next time steps to update the solution.

The time-dependent Schrödinger equation is given by:

$$i\hbar\frac{\partial\Psi}{\partial t} = -\frac{\hbar^2}{2\mu}\frac{\partial^2\Psi}{\partial x^2} + V(x)\Psi$$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv
from scipy.sparse import diags
from matplotlib.animation import FuncAnimation, PillowWriter
```

We start defining some constants and parameters. μ is the reduced mass, expressed as $\mu = \frac{m_H \cdot m_{H_2}}{m_H + m_{H_2}}$, and E_h the Hartree energy, expressed as $E_h = \frac{hbar^2}{m_e \cdot a_0^2}$. The set of Gaussian wave packet parameters used are: $\langle x \rangle = 5.5\ au$, $\langle \Delta x^2 \rangle = 0.04\ au^2$, $2\pi\langle P_x \rangle / h = 5.5\ au^{-1}$.

In [2]:
```
# Constants
hbar = 1.0545718e-34          # Reduced Planck's constant in J*s
me = 9.1093837015e-31         # mass of an electron
a0 = 5.29177210903e-11        # bohr radius
amu_to_me = 1836

# System parameters
mH = 1.0                      # amu
mH2 = 2.0                     # amu
Eh = hbar**2/(me*a0**2)       # Hartree energy
mu = mH*mH2 / (mH+mH2) * amu_to_me # Reduced mass (me)

# Initial conditions for Gaussian wave packet
x0 = 5.5               # au
delta_x_squared = 0.04 # au^2
px0_over_hbar = 5.5    # au^-1
```

In [3]:
```
# Grid parameters
Lx = 10.0               # Length
Nx = 100                # Grid points
dx = Lx/Nx              # Subintervals
xrange = np.arange(0,Lx+dx,dx)

dt = 1.979e-16 *Eh/hbar # Time step (s)
Nt = 1000               # Time points
t = Nt*dt                # Total time
trange = np.arange(0,t+dt,dt)

# Animation parameters
animation_frames = 100
animation_name = "Schroedinger.gif"
```

We now create two tridiagonal matrices, which are used in the numerical solution of the time-dependent Schrödinger equation. The coefficient is calculated with: $a = \frac{\Delta t}{4\mu\Delta x^2}$. The matrices are constructed with the following expressions:

$$\begin{bmatrix} 2j \pm 2a & \pm a & 0 & 0 & \cdots & 0 \\ \pm a & 2j \pm 2a & \pm a & 0 & \cdots & 0 \\ 0 & \pm a & 2j \pm 2a & \pm a & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \pm a & 2j \pm 2a & \pm a \\ 0 & \cdots & 0 & 0 & \pm a & 2j \pm 2a \end{bmatrix}$$

where j represents the imaginary unit ($j = \sqrt{-1}$), Δt is the time step and Δx is the grid spacing. These matrices are used to discretize the spatial derivative in the Schrödinger equation.

In [4]:
```
# Create tridiagonal matrix with scipy
dim = len(xrange)
a = dt/(4*mu*dx**2)      # Parameter a
off_diag = a
matrix_A = diags([-a,2j+2*a,-a], [-1, 0, 1], shape=(dim, dim), format='csr')
matrix_B = diags([a,2j-2*a,a], [-1, 0, 1], shape=(dim, dim), format='csr')
```

The wavefunction is now initialized with a Gaussian wave packet. Its expression is given by:

$$\Psi = \frac{1}{\left(2\pi\delta_x^2\right)^{0.25}} \exp\left(\frac{ip_{x_0}}{\hbar}(x - x_0)\right) \exp\left(-\frac{(x - x_0)^2}{4\delta_x^2}\right)$$

Where $\frac{1}{\left(2\pi\delta_x^2\right)^{0.25}}$ is a normalization factor to ensure that the Gaussian wave packet is properly normalized. The second term, $\exp\left(\frac{ip_{x_0}}{\hbar}(x - x_0)\right)$, represents the initial momentum of the wave packet, where p_{x_0} is the initial momentum. The third term, $\exp\left(-\frac{(x-x_0)^2}{4\delta_x^2}\right)$, represents the spatial distribution of the wave packet.

In [5]:
```
WF = np.array(1/(2*np.pi*delta_x_squared)**0.25 * np.exp(1*px0_over_hbar*(xrange-x0))*
               np.exp(-(xrange-x0)**2/(4*delta_x_squared)))
WF_init = WF.copy()
```

The main integration loop is done for evolving the quantum wave packet over time using the Crank-

Nicolson method. We ensure periodicity in the spatial domain by setting periodic boundary conditions. The Crank-Nicolson Update consists in updating the wavefunction, $\Psi$, by solving a linear system of equations to evolve the wavefunction to the next time step.

In [6]:
```
WFt = []
for i,ti in enumerate(trange):
    WF[-1] = WF[0]
    if i%int(Nt/animation_frames) == 0 :
        WFt.append(WF.copy())
    WF = inv(matrix_A.toarray())@matrix_B.toarray()@WF
WFt = np.array(WFt)
```

Finally, an animation of the quantum wave packet evolution over time is generated and saved as a GIF.

In [7]:
```
# Start animation
def GIF(frame):
    """Function that creates a frame for the GIF."""
    ax.clear()
    WF0_frame, = ax.plot(xrange,WF_init,c="blue",alpha=0.2,label="WF inicial")
    WFt_frame, = ax.plot(xrange,WFt[frame],c="red",label="WF final")
    wall1 = ax.axvline(Lx,ymin=0,c="k",alpha=0.9)
    wall2 = ax.axvline(0,ymin=0,c="k",alpha=0.9)
    ax.set_xlabel("$x$")
    ax.set_ylabel("$\Psi$")
    ax.set_ylim([-1.5,5])
    ax.set_title("Wavefunction evolution")
    ax.legend(loc="upper right")
    return WF0_frame,WFt_frame,wall1,wall2

fig,ax = plt.subplots(figsize=(6,5))
animation = FuncAnimation(fig,GIF,frames=animation_frames,interval=20,blit=True,repeat=
animation.save(animation_name,dpi=120,writer=PillowWriter(fps=25))
fig.tight_layout()
plt.show()
```

...