Simulation of Quantum Wave Packet Dynamics through a Potential Barrier at Different Widths (Tunnel Effect)

Why Quantum Tunneling:

In the modern age of advancements in computer hardware, scientists are questioning the potential limits of chip manufacturing. Since Moore's law was first stated, the number of transistors in an integrated circuit (IC) has consistently followed the predicted trend. However, we are now re-evaluating the validity of these claims.

We might have actually reached the ceiling in terms of how small we can make our transistors. Current manufacturing standards are causing instability problems and entire chip meltdowns. Can we still go smaller?

Decreasing the size of our chips is not just bound by our manufacturing capabilities but also by the underlying principles of quantum mechanics. At an atomic level of precision, the barriers that control the flow of electrons as logical signals (transistors) lose their ability to control this flow due to the tunneling effect.

I) *1D Schrodinger equation:*

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

In order to simplify the resolution of this partial differential equation we can assume that the wave function $\Psi(x,t)$ can be separated into a space part and a time part. Therefore written as:

$$\Psi(x,t) = \phi(x)T(t)$$

Substituting into the original equation gives:

$$i\hbar\frac{\partial[\phi(x)T(t)]}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2[\phi(x)T(t)]}{\partial^2 x} + V(x)\phi(x)T(t)$$

Dividing both sides by $[\phi(x)T(t)]$ gives two ordinary differential equations that both must both equal a constant that we denote E:

$$i\hbar\frac{dT(t)}{dt} = ET(t)\ (1)$$

$$\frac{-\hbar^2}{2m}\frac{d^2\phi(x)}{d^2x} + V(x) = E\phi(x)\ (2)$$

Using "wolfram alpha" we can analytically solve equation (1) which gives us the solution to the time dependant Schrodinger equation:

$$T(t) = e^{\frac{-iEt}{\hbar}}$$

We are finally left with equation (2) that we are required to solve with numerical methods. For this particular instance the FDM or Finite Differences Method is used to solve this differential equation.

II) Discretization of the time independent Schrodinger equation

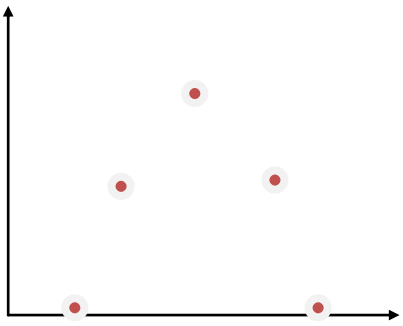
Generally at a step i

$$\frac{d^2\omega_i}{d^2x} = \frac{\frac{\omega_{i+1}-\omega_i}{\Delta x} - \frac{\omega_i-\omega_{i-1}}{\Delta x}}{\Delta \mathrm{x}}$$
$$\rightarrow$$
$$\frac{d^2\omega_i}{d^2x} = \frac{\omega_{i+1}-\omega_i-(\omega_i-\omega_{i-1})}{\Delta x^2}$$
$$\rightarrow$$
$$\frac{d^2\omega_i}{d^2x} = \frac{\omega_{i+1}-2\omega_i+\omega_{i-1}}{\Delta x^2}$$

The implication on the time independent Schrödinger equation is direct and gives:

$$\frac{-\hbar^2}{2m\Delta x^2}(\phi_{i+1}+\phi_{i-1}) + \phi_i(\frac{\hbar^2}{m\Delta x^2}+V_i) = E\phi_i$$

III) Simplification of the problem using finite elements



For the sake of simplifying the problem let's suppose that our wave function is in 5 elements $(\psi 0, \psi 1, \psi 2, \psi 3, \psi 4)$ *with* $(\psi 0 = \psi 4 = 0)$ *we can use the formula above to write a system of equations that is:*

$$\begin{cases} -\frac{-\hbar^2}{2m\Delta x^2}\phi_0 + \left(\frac{\hbar^2}{m\Delta x^2}+V_1\right)\phi_1 = E\phi_1 \\ -\frac{-\hbar^2}{2m\Delta x^2}\phi_1 + \left(\frac{\hbar^2}{m\Delta x^2}+V_1\right)\phi_2 = E\phi_2 \\ -\frac{-\hbar^2}{2m\Delta x^2}\phi_2 + \left(\frac{\hbar^2}{m\Delta x^2}+V_1\right)\phi_3 = E\phi_3 \end{cases}$$

$$where\ \phi_1, \phi_2, \phi_3\ are\ unknown$$

In order to write the problem to a computer we transform this system of equations into a system of matrices and vectors:

$$\frac{-\hbar^2}{2m\Delta x^2}\begin{bmatrix} -2-aV1 & 1 & 0 \\ 1 & -2-aV2 & 1 \\ 0 & 1 & -2-aV3 \end{bmatrix}\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix} = E\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}$$

This system of matrices and vectors expresses exactly wave function equation:

$$H\psi = E\psi$$

VI) Superpositions of eigenstates

Eigenstates are special states of a quantum system associated with specific measurable quantities (observables). When the system is in an eigenstate of a particular observable, measuring that observable will always yield the same result.

For a given $\phi_i(x) and\ E_i$ we can write:

$$\Psi(x,t) = \sum_i c_i \phi_i(x)\, e^{\frac{-iEt}{\hbar}}$$

To calculate these coefficients c_i we resort to the following definition:

$$c_i = \int_{-\infty}^{+\infty} \phi_i^* \phi(x,0)\, dx$$

$$Where\ \phi(x,0) = e^{\frac{-(x+x_0)^2}{\sigma}} e^{ip\,(x+x_0)} a\ Gaussian\ packet$$

We again use what we learned with the Finite Difference method to solve these integrations

$$c_i = \sum \phi_i^* \phi(x,0) \Delta x$$

V) Program and algorithms

This implementation uses python 3.11.3 and its linear algebra library numpy 1.26.4 and Vpython 6.11 as a graphical interface for rendering the actual plots, the project is saved in the main.py file and is commented for better understanding of the setup ,simulation and time evolution steps. This code is free of any charge and is open to every iteration and remark.

Algorithm InitialPacketInitialization

Input: x, x0, sig, p

Output: Normalized initial wave packet Psi0

Step 1: Compute the initial wave packet Psi0

```
x_trimmed <- x[1:-1]
decay_term <- exp(-(x_trimmed + x0)^2 / sig^2)
phase_term <- exp(i * p * (x_trimmed + x0))
Psi0 <- decay_term * phase_term
```

Step 2: Normalize the wave packet Psi0

```
    NormFact <- sum(abs(Psi0)^2 * dx)
    Psi0 <- Psi0 / sqrt(NormFact)
  Return Psi0
End Algorithm
```

Algorithm MatrixForHamiltonians

```
  Input: Repartitions, dx, y, hbar, m
  Output: Hamiltonian matrix H
  Step 1: Initialize H as a zero matrix of size (Repartitions-1) x (Repartitions-1)
  Step 2: Compute the main diagonal elements of H
    for i from 1 to Repartitions-1 do
      H[i][i] <- (hbar^2 / (m * dx^2)) + y[i+1]
    end for
  Step 3: Compute the off-diagonal elements of H
    for i from 1 to Repartitions-2 do
      H[i][i+1] <- -hbar^2 / (2 * m * dx^2)
      H[i+1][i] <- -hbar^2 / (2 * m * dx^2)
    end for
  Return H
End Algorithm
```

Algorithm MainTimeLoop

```
  Input: c, psi, E, hbar, x, len(Psi)
  Output: Update visualization data f1.data over time
  Initialize t to 0
  Set dt to 0.001
  While t < 0.2 do
    Call rate(24) to control the update rate
    Step 1: Compute the wave function Psi at time t
      Psi <- Sum over c[:, np.newaxis] * psi * exp(-1j * E[:, np.newaxis] * t / hbar) along axis=0
    Step 2: Update visualization data f1.data
      f1.data <- List of pairs [x[i], abs(Psi[i])] for each i in range(len(Psi))
    Increment t by dt
  End While
End Algorithm
```