Code Overview

1 Introduction

This document provides a complete technical overview of the implementation used in the Quantum Tunneling Simulation Project. The numerical models include a one-dimensional Gaussian wavepacket evolved using Hamiltonian eigen-decomposition, and a two-dimensional split-step Fourier propagation scheme. The codebase is written in Python and relies on NumPy, Matplotlib, and FFT-based routines.

The objective of this document is to explain each computational component, describe how classes and functions interact, justify algorithmic design choices, and provide a full understanding of the simulation architecture.

2 Class: `GaussianWavePacket` (1D Simulation)

The class `GaussianWavePacket` implements exact time evolution of the one-dimensional time-dependent Schrödinger equation using eigen-decomposition. It constructs the spatial grid, potential, Hamiltonian, eigenstates, and evaluates the wavefunction at all times in `time_array`.

2.1 Constructor Parameters

- **num_grid_intervals** – Number of intervals in the spatial grid.
- **domain_length** – Length of the physical domain $[0, L]$.
- **barrier_start, barrier_height, barrier_width** – Define a rectangular potential barrier.
- **initial_center** – Initial Gaussian center x_0.
- **initial_momentum** – Wavepacket momentum k_0.
- **sigma_width** – Gaussian width parameter σ.
- **time_array** – Times at which $\psi(x, t)$ is evaluated.

2.2 Grid Construction

The spatial grid is defined by

$$x_i = \frac{iL}{N}, \qquad i = 0, 1, \ldots, N, \tag{1}$$

where $N =$ `num_grid_intervals`. The grid spacing is

$$\Delta x = x_1 - x_0.$$

Dirichlet boundary conditions are imposed by using interior points $x_1, \ldots, x_{N-1}$ in the Hamiltonian.

2.3 Initial Wavefunction

The initial Gaussian wavepacket is defined as

$$\psi_0(x) = \exp\left[-\frac{(x - x_0)^2}{2\sigma^2}\right] e^{ik_0 x}. \tag{2}$$

Normalization is enforced using the discrete approximation:

$$\sum_i |\psi_0(x_i)|^2 \, \Delta x = 1. \tag{3}$$

2.4 Kinetic Operator Construction

The Laplacian is implemented using the standard second-derivative finite-difference stencil:

$$T = -\frac{1}{2\Delta x^2} \begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix}. \tag{4}$$

2.5 Potential Construction

The potential operator is diagonal:

$$V_{ii} = \begin{cases} V_0, & a < x_i < a + w, \\ V_{\text{bg}}, & \text{otherwise}, \end{cases} \tag{5}$$

where V_{bg} is a small background term that prevents exact spectral degeneracy.

2.6 Hamiltonian Assembly

The full Hamiltonian is

$$H = T + V. \tag{6}$$

Since H is Hermitian, it is diagonalized via `numpy.linalg.eigh`.

3 Eigen-Decomposition Time Evolution

The stationary states satisfy

$$H\phi_n = E_n\phi_n. \tag{7}$$

3.1 Projection of the Initial State

The coefficients in the eigenbasis are computed as

$$c_n = \sum_i \phi_n^*(x_i)\,\psi_0(x_i)\,\Delta x. \tag{8}$$

3.2 Time Evolution

The exact solution of the TDSE is expressed as

$$\psi(x,t) = \sum_n c_n\, e^{-iE_n t}\,\phi_n(x). \tag{9}$$

This provides an unconditionally stable propagation method.

3.3 Animation

The `animation()` method uses Matplotlib's `FuncAnimation` to animate:

- $\mathrm{Re}[\psi(x,t)]$,
- $\mathrm{Im}[\psi(x,t)]$,
- or the probability density $|\psi(x,t)|^2$.

4 Transmission and Reflection Computation

For a sufficiently late time t_f, the reflection and transmission probabilities are:

$$R = \int_{x<a} |\psi(x,t_f)|^2\,dx, \tag{10}$$

$$T = \int_{x>a+w} |\psi(x,t_f)|^2\,dx. \tag{11}$$

The integrals are computed via discrete Riemann sums.

5 Parameter Sweep Functions

The function `compute_transmission()`:

1. Constructs the Hamiltonian.
2. Evolves the wavepacket to a large time.
3. Computes the transmission coefficient T.

Repeated evaluation produces transmission-vs-parameter curves.

6 2D Simulation Overview (Split-Step Fourier Method)

The two-dimensional component solves the TDSE using operator splitting.

6.1 Grid and Initial State

A 2D mesh is generated using `numpy.meshgrid`. The initial packet is

$$\psi(x, y, 0) = \exp\left[-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right] e^{ik_0(x-x_0)}. \tag{12}$$

6.2 Potential Definition

A rectangular vertical barrier is defined as

$$V(x, y) = \begin{cases} V_0, & x_b < x < x_b + w, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

6.3 Split-Step Algorithm

Each time step applies:

1. Potential half-step: $\psi \leftarrow e^{-iV\,dt/2}\psi$.
2. FFT to momentum space.
3. Kinetic propagation: $\psi \leftarrow e^{-i(k_x^2+k_y^2)dt/2}\psi$.
4. Inverse FFT.
5. Final potential half-step.
6. Absorbing boundary mask to reduce reflections.

6.4 Animation Generation

Selected $|\psi(x, y, t)|^2$ frames are plotted using `imshow()` to generate a 2D animation.

7 Summary

This document has presented a full description of the quantum tunneling simulation code, covering discretization methods, Hamiltonian construction, eigen-decomposition, exact time evolution, parameter studies, and two-dimensional FFT-based propagation. The implementation is modular, efficient, and suitable for pedagogical demonstrations as well as research-level numerical experiments.