2Simulation of 2D Quantum Wavepacket Dynamics using the Shrödinger equation and the Finite Element Method in Julia

Author: Victor Emmanuel Meimaris A.M: 58855

Democritus University of Thrace, Xanthi, Greece

Date: March 25, 2023 started / May 30, 2025 written document

Abstract

This document outlines a Julia-based numerical simulation framework for studying the time evolution of a two-dimensional quantum mechanical wavepacket. The core of the simulation relies on solving the Time-Dependent Schrödinger Equation (TDSE). Spatial discretization is achieved using the Finite Element Method (FEM) with bilinear quadrilateral elements, particularly tailored for an L-shaped domain. The temporal evolution is handled by the robust and unitary Crank-Nicolson scheme. The script is modular, allowing for different potential energy landscapes and initial wavefunction configurations. Key aspects such as mesh generation, FEM matrix assembly, initial state definition, and time-stepping solution are discussed, along with the influence of various simulation parameters on the output.

1. Introduction

The study of quantum mechanical systems is fundamental to understanding the behavior of matter at atomic and subatomic scales. The Time-Dependent Schrödinger Equation (TDSE) governs the evolution of a quantum system's wavefunction, from which all observable properties can be derived. Due to the complexity of analytical solutions, especially for systems with non-trivial potentials or geometries, numerical methods are indispensable.

This Julia script provides a framework to simulate the dynamics of a single quantum particle in a two-dimensional space. The primary objectives are:

1. To accurately model the spatial characteristics of the wavefunction using the Finite Element Method (FEM).
2. To propagate the wavefunction in time using the Crank-Nicolson method, known for its stability and preservation of probability (unitarity).

3. To visualize the evolution of the probability density of the particle under various user-defined potential fields and initial conditions.

The script is structured into a module Functions, encapsulating all necessary components for setting up and running the simulation.

2. Physical and Mathematical Framework

2.1. The Time-Dependent Schrödinger Equation (TDSE)

The cornerstone of this simulation is the 2D TDSE:

$$i\hbar\frac{\partial\Psi(\mathbf{r},t)}{\partial t} = \hat{H}\Psi(\mathbf{r},t)$$

where:

- The complex-valued wavefunction of the particle at position $\mathbf{r} = (x, y)$ and time t is represented as: $\Psi(\mathbf{r}, t)$
- The reduced Planck constant is: $\hbar$
- The Hamiltonian operator, which describes the total energy of the system, is: $\hat{H}$

  For a single particle of mass m in a potential $V(\mathbf{r})$, it is given by:
  $$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}) = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x, y)$$

2.2. Numerical Discretization

To solve the TDSE numerically, we discretize both space and time.

2.2.1. Spatial Discretization: Finite Element Method (FEM)

The FEM is employed for spatial discretization. This involves:

1. **Weak Formulation:** Multiplying the TDSE by a test function $N_i(\mathbf{r})$ and integrating over the spatial domain Ω:
   $$\int_\Omega N_i\left(i\hbar\frac{\partial\Psi}{\partial t}\right)d\Omega = \int_\Omega N_i\left(-\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi\right)d\Omega$$

   Applying Green's first identity (integration by parts) to the Laplacian term:

$$\int_\Omega N_i \left(-\frac{\hbar^2}{2m} \nabla^2 \Psi \right) d\Omega = \int_\Omega \frac{\hbar^2}{2m} \nabla N_i \cdot \nabla \Psi d\Omega - \oint_{\partial\Omega} N_i \frac{\hbar^2}{2m} \frac{\partial \Psi}{\partial n} dS$$

For Dirichlet boundary conditions ($\Psi = 0$ on $\partial\Omega$), the boundary integral vanishes.

2. **Domain Meshing:** The 2D domain is divided into smaller, non-overlapping quadrilateral elements. The script is specifically designed to handle L-shaped domains.
3. **Wavefunction Approximation:** Within each element e, the wavefunction $\Psi^e(x, y, t)$ is approximated as a linear combination of basis (or shape) functions $N_j(x, y)$ and time-dependent nodal values $\psi_j(t)$:

   $$\Psi^e(x, y, t) \approx \sum_{j=1}^{4} N_j(x, y) \psi_j(t)$$

   The script uses bilinear shape functions for quadrilateral elements.
4. **Element & Global Matrices:** Substituting the approximation into the weak form and summing over all elements leads to a system of ordinary differential equations in time:

   $$i\hbar M \frac{d\vec{\psi}}{dt} = H\vec{\psi}$$

   where:
   - $\vec{\psi}(t)$ is the vector of nodal wavefunction values.
   - M is the global **Mass Matrix**:

     $$M_{ij} = \int_\Omega N_i N_j d\Omega$$

   - H is the global **Hamiltonian Matrix**:

     $$H_{ij} = \int_\Omega \left(\frac{\hbar^2}{2m} \nabla N_i \cdot \nabla N_j + N_i V N_j \right) d\Omega$$

     This H matrix is referred to as tempo in the script before further scaling.

2.2.2. Temporal Discretization: Crank-Nicolson Method

The Crank-Nicolson scheme is used for time integration. It is an implicit method, unconditionally stable for this type of problem, and unitary (conserves probability). It approximates the time derivative as:

$$\frac{d\vec{\psi}}{dt} \approx \frac{\vec{\psi}^{n+1} - \vec{\psi}^{n}}{\Delta t}$$

and averages the right-hand side at times t_n and t_{n+1}:

$$i\hbar\frac{\vec{\psi}^{n+1} - \vec{\psi}^{n}}{\Delta t} = \frac{1}{2}(H\vec{\psi}^{n+1} + H\vec{\psi}^{n})$$

Rearranging this gives:

$$\left(M - \frac{\Delta t}{2i\hbar}H\right)\vec{\psi}^{n+1} = \left(M + \frac{\Delta t}{2i\hbar}H\right)\vec{\psi}^{n}$$

Or, equivalently, as implemented by scaling H first:

$$\left(M + \frac{i\Delta t}{2\hbar}H\right)\vec{\psi}^{n+1} = \left(M - \frac{i\Delta t}{2\hbar}H\right)\vec{\psi}^{n}$$

This can be written as:

$$A\vec{\psi}^{n+1} = B\vec{\psi}^{n}$$

where $A = M + \frac{i\Delta t}{2\hbar}H$ and $B = M - \frac{i\Delta t}{2\hbar}H$. At each time step Δt, this linear system is solved for $\vec{\psi}^{n+1}$. Consequently, both A and B are **complex-valued** matrices. They inherit the **sparsity** from M and H. These matrices are generally **non-Hermitian**. Because they are non-Hermitian, they are **not symmetric positive definite (SPD)** in the standard sense, nor are they generally symmetric in the complex sense unless H is zero. Furthermore, A and B are **not necessarily diagonally dominant**, as the off-diagonal elements of H, scaled by the complex factor, can be significant relative to the diagonal entries which also include contributions from M. The specific conditioning and other numerical properties of A and B depend on the characteristics of M, H, and the time step Δt. The matrix A must be **invertible** to solve the linear system at each time step, a condition satisfied by the unconditionally stable Crank-Nicolson method. Given that the system matrices A and B are not

Symmetric Positive Definite (SPD), iterative methods designed for such systems, like the Conjugate Gradient method, are not directly applicable. Consequently, a robust solver for non-symmetric systems, such as the **Biconjugate Gradient Stabilized (BiCGSTAB) method**, is an appropriate choice for solving the system at each time step.

Furthermore, considering that these matrices are also **not necessarily diagonally dominant**, the convergence of simpler iterative schemes like the damped Jacobi method, often employed as a smoother in multigrid approaches, cannot be guaranteed. This lack of guaranteed convergence for basic smoothers further underscores the suitability of employing a more general and robust iterative solver like BiCGSTAB for the overall solution of the linear system.

3. Code Implementation Details (Module Functions)

The script uses SparseArrays.jl for efficient handling of large matrices and Plots.jl for visualization (with gr() used for animations).

3.1. Constants

- h_bar = 0.65821220e-3: Reduced Planck constant in eV·ps.
- m = 9.1093837e-31 * (10e25 / 1.602117662): Mass of an electron, converted to units of (eV·ps²)/nm².
- gamma = (h_bar^2)/(2*m): A convenient constant appearing in the kinetic energy term, in units of eV·nm².

3.2. grid Function

- **Purpose:** Generates the computational mesh for an L-shaped domain, node coordinates, element connectivity (local-to-global mapping l2g), and identifies boundary nodes.
- **Inputs:** domain::Tuple{Real, Real} (e.g., (-1,1)), max_length::Float64 (desired maximum element edge length).
- **Outputs:** coords (matrix of node coordinates), l2g (local-to-global mapping for elements), noe (number of elements), nop (number of nodes), boundary_nodes (a Set of boundary node indices), max_length (potentially adjusted), lengthr (number of points along one dimension of the full square grid), a, b, c (node numbering matrices for sub-regions), nx_half, ny_half.
- **Logic:**
    - The L-shape is constructed by defining node numbering for three rectangular

subdomains: a top-left square (a), a bottom-left square (b), and a bottom-right square (c).
    - max_length is adjusted to ensure that the number of divisions along an axis (domain_length / max_length) is an even integer, which is a requirement for the specific FEM element structuring or subsequent solver logic.
    - Boundary nodes are explicitly collected into a Set for easy application of Dirichlet boundary conditions.

3.3. fem_matrices Function

- **Purpose:** Assembles the FEM matrices A and B required for the Crank-Nicolson scheme.
- **Inputs:** V_potential_func (a function (x,y) -> V(x,y)), dt (time step), coords, l2g, noe, boundary_nodes, step_size (related to max_length), lengthr.
- **Key Steps:**
    1. **Shape Functions (N):** Defines the four bilinear shape functions for a quadrilateral element in local coordinates (ξ, η).
    2. **Numerical Integration (Gaussian Quadrature):** Iterates over each element (e = 1:noe):
        - Retrieves element node coordinates (xe, ye).
        - Defines coordinate transformation $x(\xi, \eta)$, $y(\xi, \eta)$.
        - Calculates the Jacobian matrix components (J11, J12, J21, J22) and its determinant detJ.
        - Calculates derivatives of shape functions with respect to global coordinates (θNx, θNy).
        - Computes local element matrices:
            - Stiffness & Potential part (KV):
              $$\int_e (\gamma(\nabla N_i \cdot \nabla N_j) + N_i V N_j) \, |detJ| d\xi d\eta$$
            - Mass part (L):
              $$\int_e N_i N_j |detJ| d\xi d\eta$$

        These integrals are evaluated using gauss_quad_2D(integrand, 3) (3-point Gaussian quadrature).
    3. **Assembly:** The local matrices KV and L are assembled into global sparse matrices H_global (referred to as H initially, then tempo) and M_global (referred to as M).

4. **Dirichlet Boundary Conditions:** Rows and columns corresponding to boundary_nodes in H_global and M_global are zeroed out, and the diagonal entry is set to 1. This effectively enforces $\Psi = 0$ at these nodes and ensures the matrices remain well-conditioned.
5. **Crank-Nicolson Matrices:**
   - The Hamiltonian H_global (stored in tempo) is scaled: $H_{\text{scaled}} = H_{\text{global}} \cdot i \cdot \Delta t/(2\hbar)$.
   - $A = M_{\text{global}} + H_{\text{scaled}}$
   - $B = M_{\text{global}} - H_{\text{scaled}}$

- The variable tempo stores the original Hamiltonian matrix $\int (\gamma \nabla N_i \cdot \nabla N_j + N_i V N_j)$, which is used later for energy calculations.

3.4. wavefunction Function

- **Purpose:** Defines the initial wavefunction $\Psi(x, y, t = 0)$.
- **Form:** A Gaussian wave packet modulated by a plane wave:

$$\Psi(x, y; x_0, y_0, \sigma, k_x, k_y) = \left(e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}} \right) \cdot \left(e^{-i(k_x x + k_y y)} \right)$$

- **Parameters:**
  - x_0, y_0: Initial center coordinates of the Gaussian.
  - σ: Standard deviation, controlling the initial spatial spread of the packet.
  - k_x, k_y: Wavevector components, determining the initial momentum and direction of propagation.

3.5. V_function Function

- **Purpose:** Provides a selection of potential energy landscapes $V(x, y)$..
- **Inputs:** V_flag::Int64, optional parameters V0 (potential strength), x0, y0 (center), r (radius).
- **Returns:** A function (x,y) -> V(x,y).
- **Options (V_flag):**
  1. V_flag == 1: Infinite potential well (constant potential, default $V_0 = 0.0$). The "infinite" walls are handled by Dirichlet boundary conditions.
  2. V_flag == 2: Circular potential barrier of height V0 centered at (x0, y0) with radius r.
  3. V_flag == 3: Circular potential well of depth $-V_0$ centered at (x0, y0) with radius r.

3.6. solution Function

- **Purpose:** Solves the TDSE over a specified number of iterations or up to a certain time, yielding the final wavefunction and testing each solver.
- **Inputs:** coords, nop, psi_zero (the initial wavefunction function from wavefunction), time (tuple (t_start, t_end)), A, B (Crank-Nicolson matrices), lengthr, dt, boundary_nodes, tempo (unscaled Hamiltonian H), step_size, L-shape geometry matrices (a,b,c,nx_half,ny_half), overlaps, iterations.
- **Key Steps:**
  1. **Initialization:** The initial wavefunction vector $\vec{\psi}_0$ is populated using the psi_zero function evaluated at nodal coordinates. Dirichlet boundary conditions are enforced on $\vec{\psi}_0$.
  2. **Normalization:** $\vec{\psi}_0$ is normalized such that $\int |\Psi_0|^2 dxdy = 1$. The normalization constant A_ is calculated as sqrt(sum(psi_0' * psi_0 * step_size^2)). A check sum_check verifies this.
  3. **Initial Energy:** The initial energy $E_{\text{initial}} = \text{real}(\vec{\psi}_0^\dagger \cdot \text{tempo} \cdot \vec{\psi}_0)$ is calculated. tempo is the true Hamiltonian matrix.
  4. **Time Stepping and Solver Benchmarking:** The system $A\vec{\psi}^{n+1} = B\vec{\psi}^n$ is solved iteratively for iterations steps using four different approaches, and their execution times are recorded:
     a. Direct Solver (\): A_LU = lu(A) precomputes the LU factorization. Then psi = A_LU \ (B * psi_0) is used in the loop.
     b. BiCGSTAB Alone: psi = bicgstab_vic(A, B * psi_0, 1e-10, 150) is used.
     c. Domain Decomposition with BiCGSTAB: psi = domain_decomposition(A, B * psi_0, a, b, c, nx_half, ny_half, overlaps, 1) is used (where flag = 1 selects bicgstab_vic as the subdomain solver).
     d. Domain Decomposition with Multigrid: psi = domain_decomposition(A, B * psi_0, a, b, c, nx_half, ny_half, overlaps, 2) is used (where flag = 2 selects multigrid_vic as the subdomain solver).
  5. **Final Energy:** The energy is recalculated using the final wavefunction. For a closed system with a time-independent Hamiltonian, energy should be conserved.
  6. **Performance Reporting:** Timings for each solver approach are printed

3.7. animated_solution Function

- **Purpose:** Generates an animation of the probability density $|\Psi(x, y, t)|^2$ over time.

- **Inputs:** Similar to solution, plus no_fr (total number of frames to compute) and no_afr (compute every no_afr-th frame for animation).
- **Logic:**
  - Initializes and normalizes $\vec{\psi}_0$ as in solution.
  - Calculates global_max = maximum(abs2.(psi_0)) * 1.8 to ensure consistent z-axis scaling in plots.
  - Uses LU decomposition (A_LU = lu(A)) for solving at each time step.
  - In the time evolution loop (n = 1:no_fr):
    - Solves for psi.
    - If n % no_afr == 0 or n == 1, the probability density abs2(psi[k]) is mapped onto a 2D grid Z.
    - A surface plot is generated using Plots.surface() and stored.
    - $\vec{\psi}_0$ is updated to psi for the next step.
  - The collected frames are compiled into an animation using @animate.
  - Final energy is printed.

3.8. Helper Functions

- **gauss_quad_2D(funct, n):** Implements 2D Gaussian quadrature for numerical integration over a square domain $[-1, 1] \times [-1, 1]$. It supports $n = 2, 3, 4$ integration points in each direction. This is crucial for accurately computing the element matrices in fem_matrices.
- **bicgstab_vic(A, b, tol, Nmax):**
  - **Purpose:** Implements the BiConjugate Gradient Stabilized (BiCGSTAB) method, an iterative algorithm to solve $Ax = b$ on a non spd system.
  - **Mathematical Idea:** BiCGSTAB is a Krylov subspace method that iteratively refines a solution. It uses an Incomplete LU factorization ($K \approx A$) as a preconditioner (K^{-1} applied via K \ vector) to accelerate convergence. Each iteration involves computing step lengths (α, ω) and search directions (p) to minimize the residual $r = b - Ax$.
  - **Inputs:** Matrix A, vector b, tolerance tol, max iterations Nmax.
  - **Output:** Approximate solution vector x.
- **domain_decomposition(A, b, s1, s2, s3, nx_half, ny_half, overlap, flag):**
  - **Purpose:** Implements an Additive Schwarz domain decomposition method to iteratively solve $Ax = b$.
  - **Script Implementation Details:**
    1. **Setup:**
       - Validates overlap.
       - Initializes global solution x to zero.

- Defines inner1, inner3 (core non-overlapping parts of subdomains 1 and 3) and overlap matrices (over1, over21, etc.) based on s1, s2, s3 and overlap.
- Constructs global node index vectors (inds1, inds2, inds3) for the three extended (overlapping) subdomains.
- Extracts subdomain matrices A_1, A_2, A_3 from the global A.
- Creates mapping index vectors d and c to extract solutions from the core regions of extended subdomains 2 and 3, respectively.
- Sets overall DD tolerance tolDD = 10e-10 and max iterations Nmax = 200.

2. **Solver Branching (flag):**
   - **If flag == 1 (BiCGSTAB for subdomains):**
     - Sets subdomain solver tolerance tolBic = 10e-5 and max iterations it = 20.
     - In each DD iteration, solves $A_k \delta\tilde{x}_k = r_k^{(m)}$ using bicgstab_vic(Ak, r[inds_k], tolBic, it) for each of the three subdomains.
   - **If flag == 2 (Multigrid for subdomains):**
     - Sets multigrid smoother iterations n1=6 (pre), n2=4 (post), and multigrid tolerance tolMG = 10e-4.
     - **Crucially, calls multigrid_vic_hierarchy *once before the DD iteration loop*** for each subdomain (A1, A2, A3) to precompute their respective multigrid hierarchies (A1_, R1, P1, levels1, etc.). The time for this setup is printed.
     - In each DD iteration, solves $A_k \delta\tilde{x}_k = r_k^{(m)}$ using multigrid_vic(Ak_hierarchy, Rk_hierarchy, Pk_hierarchy, levels_k, r[inds_k], n1, n2, tolMG) for each subdomain, passing the precomputed hierarchies.
3. DD Iteration:
   a. Calculates global residual $r = b - Ax$.
   b. Checks for convergence (norm(r)/norm(b) < tolDD).
   c. Solves for local corrections x1, x2, x3 on subdomains using the method selected by flag.
   d. Extracts relevant parts of x1, x2, x3 (using d and c for x2, x3).
   e. Updates global solution x additively.

- **Output:** Approximate global solution vector x.
- **Mathematical Idea:** The L-shaped domain Ω is divided into three overlapping subdomains Ω_k. The global matrix A is restricted to each

extended (overlapping) subdomain as $A_k = A[\text{inds}_k, \text{inds}_k]$.
The iterative process is:

1. Initialize solution $x^{(0)}$.
2. For iteration $m = 0, 1, \ldots$
   a. Global residual: $r^{(m)} = b - Ax^{(m)}$
   b. For each subdomain k, solve a local correction problem on the extended subdomain using the restricted residual $r_k^{(m)} = r^{(m)}[\text{inds}_k]$:
   $A_k \delta\tilde{x}_k = r_k^{(m)}$
   (Solved via bicgstab_vic in the code). $\delta\tilde{x}_k$ is the correction on the extended subdomain.
   c. Map relevant parts of $\delta\tilde{x}_k$ to a global correction that applies to the core (non-overlapping) part of subdomain k. (The code uses index vectors d and c for this mapping for subdomains 2 and 3).
   d. Update global solution additively: `x(m+1)=x(m)+Σk(mapped core correction for δx~k).
3. Convergence is checked by $\|r^{(m)}\|/\|b\| < \text{tol}$.

- **Parameters Affecting Domain Decomposition Convergence and Performance:**
  - overlap: The number of shared node layers between subdomains. Generally, a larger overlap improves the convergence rate of the Schwarz iterations (as more information is exchanged per iteration) but also increases the size and cost of solving each subdomain problem. An optimal overlap balances these factors. Too small an overlap can lead to slow convergence or divergence.
  - Subdomain Solver (flag): The choice and efficiency of the solver for the local problems on A_k (bicgstab_vic or multigrid_vic) is critical. A more accurate or faster subdomain solver can lead to fewer outer Schwarz iterations or faster overall execution.
  - Tolerance (tol) and Max Iterations (Nmax): These control the accuracy of the final DD solution and the maximum computational effort for the outer Schwarz loop.
  - Problem Size and Condition Number: Larger, more ill-conditioned global systems $Ax = b$ will generally be harder for any iterative method, including DD, to solve. The properties of the subdomain matrices A_k also play a role.

- **multigrid_vic_hierarchy(nox, noy, overlap, A1, flag_grid_type = 1):**
  - **Purpose:** Precomputes and returns the components of a geometric multigrid

hierarchy for a given fine-grid subdomain operator A1.

- **Script Implementation Details:**
  1. Calculates grid dimensions (nx, ny) for each level down to a minimum size (<=3 nodes in a dimension). For L-shaped grids (flag_grid_type=2), n_inner (derived from nox - overlap) is used by build_Restriction_matrix.
  2. Builds restriction matrices $R^{(l-1)}$ for each level using build_Restriction_matrix.
  3. Builds prolongation matrices $P^{(l-1)} = 4(R^{(l-1)})^T$.
  4. Stores the input A1 as the operator for the finest level (A[1]).
  5. Builds coarse grid operators $A^{(l)} = R^{(l-1)} A^{(l-1)} P^{(l-1)}$ using Galerkin projection for $l > 1$.
- **Output:** A (vector of operators $A^{(l)}$), R (vector of restriction matrices $R^{(l)}$), P (vector of prolongation matrices $P^{(l)}$), levels (total number of levels).

- multigrid_vic(A_hierarchy, R_hierarchy, P_hierarchy, levels, B_fine, n1, n2, tol): (Note: A_hierarchy contains operators for all levels, B_fine is the RHS for the finest level of this specific MG solve).
  - **Purpose:** Performs a V-cycle multigrid solve using a precomputed hierarchy.
  - **Script Implementation Details:**
    1. Initializes solution vectors $x^{(l)}$ and RHS vectors $b^{(l)}$ for all levels. Sets $b[1] = B_fine$.
    2. Iterates for Nmax = 50 V-cycles or until convergence.
    3. V-Cycle Implementation:
       a. Down-stroke: For levels
    4. Convergence is checked on the finest level residual: norm(b[1] - A_hierarchy[1] * x[1]) < norm_b * tol.
  - **Output:** Approximate solution vector $$x[1]$$ for the (subdomain) problem.
  - **Mathematical Idea (Geometric Multigrid V-Cycle):**
    1. **Hierarchy of Grids:** A sequence of coarser grids is defined, starting from the fine grid (dimensions nox, noy). Node counts are approximately halved in each dimension for coarser levels until a minimum size is reached. For L-shaped subdomains (flag_grid_type=2), n_inner helps manage the geometry during coarsening.
    2. **Grid Operators ($A^{(l)}$):** The operator $A^{(0)} = A1$ is given on the finest level. For coarser levels $l > 0$, the operator is formed using the Galerkin approach:
       $A^{(l)} = R^{(l-1)} A^{(l-1)} P^{(l-1)}$

where $R^{(l-1)}$ is the restriction operator from level $l-1$ to l, and $P^{(l-1)}$ is the prolongation (interpolation) operator from level l to $l-1$.

3. **Restriction (R):** Transfers a vector (e.g., residual) from a finer grid to a coarser grid. The build_Restriction_matrix function constructs R based on weighted averaging (using a 9-point stencil for interior points).
4. **Prolongation (P):** Interpolates a vector (e.g., correction) from a coarser grid to a finer grid. In this code, $P = 4R^T$.
5. **Smoothing:** On each grid level l (except the coarsest), a "smoother" is applied to reduce high-frequency error components of the current approximate solution $x^{(l)}$ to $A^{(l)}x^{(l)} = b^{(l)}$. This implementation uses bicgstab_vic for n1 iterations (pre-smoothing) before going to a coarser grid, and for n2 iterations (post-smoothing) after returning from a coarser grid. The reason bicgstab_vic is used and not a smoother like damped Jacobi, is because A is not necessarily diagonally dominant as the smoother requires, as such not guaranteeing convergence.
6. V-Cycle Algorithm (Recursive View for solving $A^{(l)}x^{(l)} = b^{(l)}$):
   a. Coarsest Level: If on the coarsest grid, solve $A^{(\text{coarsest})}x^{(\text{coarsest})} = b^{(\text{coarsest})}$ directly (e.g., A[end] \ b[end]).
   b. Pre-Smoothing (Down-stroke): Perform n1 smoothing steps (e.g., bicgstab_vic) on $A^{(l)}x^{(l)} = b^{(l)}$ to get an improved $x^{(l)}$.
   c. Compute Residual: $d^{(l)} = b^{(l)} - A^{(l)}x^{(l)}$.
   d. Restrict Residual: Transfer residual to coarser grid: $d^{(l+1)} = R^{(l)}d^{(l)}$.
   e. Recursive Solve: Solve the coarse grid correction equation $A^{(l+1)}e^{(l+1)} = d^{(l+1)}$ by performing a V-cycle starting from level $l+1$. Set initial guess for $e^{(l+1)}$ to zero.
   f. Prolongate Correction: Interpolate coarse grid correction $e^{(l+1)}$ back to fine grid: $e^{(l)} = P^{(l)}e^{(l+1)}$.
   g. Correct Solution: Update solution: $x^{(l)} = x^{(l)} + e^{(l)}$.
   h. Post-Smoothing (Up-stroke): Perform n2 smoothing steps on $A^{(l)}x^{(l)} = b^{(l)}$.

- **Inputs:** Subdomain grid dimensions nox, noy, overlap (for L-shape handling), fine-grid operator A1, RHS B, smoother iterations n1, n2, tolerance tol, grid type flag_grid_type.
- **Output:** Approximate solution vector $x[1]$ for the subdomain problem.

3.9. Solver Benchmarking and Discussion

The solution function in the script systematically benchmarks four different

approaches to solve the linear system $A\vec{\psi}^{n+1} = B\vec{\psi}^{n}$ arising at each time step of the Crank-Nicolson scheme. **The observed performance ranking, from fastest to slowest, is reported by the user as:**

1. **Direct Solver (\ operator, using LU decomposition).**
2. **domain_decomposition with multigrid_vic as subdomain solver (flag=2).**
3. **domain_decomposition with bicgstab_vic as subdomain solver (flag=1).**
4. **bicgstab_vic (alone).**

Potential Explanations for the Observed Benchmark Ranking:

- **1.** Direct Solver **(\):**
  - **Efficiency for Moderate Systems:** For sparse matrices that are not excessively large (i.e., the number of unknowns nop is within a certain range), Julia's backslash operator (\) is highly optimized. It typically employs a direct sparse LU factorization (e.g., from UMFPACK, part of SuiteSparse).
  - **One-Time Factorization Cost:** The LU factorization $A = LU$ is precomputed once via A_LU = lu(A). In the time-stepping loop, each solution for ψ then only requires relatively fast sparse forward and backward substitutions.
  - **Robustness:** Direct solvers provide the solution (up to machine precision) without concerns about iteration counts or convergence criteria that affect iterative methods. For problems where factorization is feasible, they are often the most robust and can be the fastest in serial execution.
- **2. domain_decomposition with multigrid_vic (flag=2):**
  - **Effective Subdomain Solver:** multigrid_vic method, when applied as a solver for the three smaller subdomain problems ($A_k \delta\tilde{x}_k = r_k^{(m)}$), is performing very effectively. Multigrid methods, when well-tuned, can achieve convergence rates that are nearly independent of the problem size (or depend very weakly), aiming for $O(N)$ complexity for a problem with N unknowns.
  - **Reduced Outer Iterations:** If multigrid_vic provides accurate solutions to the subdomain problems quickly, the number of outer Schwarz iterations required for the global domain_decomposition method to converge will be small. This limits the accumulation of overhead from the DD framework.
  - **Specialized Multigrid Components:** The multigrid_vic implementation uses build_Restriction_matrix which has specific logic for L-shaped subdomains (flag_grid_type=2). This geometric adaptation of the restriction (and thus prolongation) operators might be crucial for its good performance on the L-shaped subdomains, particularly A_2.

    - **Synergy:** The combination of DD breaking down the problem and MG efficiently solving the resulting (still potentially complex) subdomain problems appears to be a strong pairing for this specific setup, outperforming other iterative strategies.
- **3. domain_decomposition with bicgstab_vic (flag=1):**
    - **Subdomain Solver Efficiency:** bicgstab_vic (with its ILU preconditioner) is less efficient at solving the subdomain problems than multigrid_vic if there are many elements, but for less elements due to bicgstab, this is very fast.
    - **DD Overheads:** The overheads of domain decomposition (subdomain setup, data transfers between global residual and local RHS, solution updates) are present.
- **4. bicgstab_vic (alone):**
    - **Global Problem Challenge:** bicgstab_vic applied directly to the global matrix A is the fastest for less elements as expected and is close to backslash.
    - **Preconditioner Effectiveness:** The matrix A resulting from the FEM discretization of the TDSE on this geometry has properties (e.g., conditioning, eigenvalue distribution) that are handled by ILU.

4. Influence of Parameters on Simulation Output

The behavior and accuracy of the simulation are highly dependent on several key parameters:

- **Mesh Parameters (domain, max_length):**
    - domain: Defines the physical extent of the simulation area (e.g., -1 nm to 1 nm).
    - max_length: Controls the element size and thus the mesh density. A smaller max_length results in a finer mesh, leading to higher spatial accuracy but significantly increases computational cost (more nodes and elements, larger and denser matrices, longer solution times). The choice of max_length should be small enough to resolve the shortest wavelength components of the wavefunction and the features of the potential.
- **Time Step (dt):**
    - Affects the temporal accuracy and stability of the simulation. The Crank-Nicolson scheme is unconditionally stable in theory for this problem, but a very large Δt will still lead to poor accuracy, failing to capture the dynamics correctly.
    - A smaller Δt provides better temporal resolution but increases the total number of steps required to simulate a given physical time, thus increasing

overall computation time.

- **Initial Wavefunction Parameters ($x_0, y_0, \sigma, k_x, k_y$ in wavefunction):**
  - x_0, y_0: The initial (mean) position of the wavepacket.
  - σ: The initial spatial spread (standard deviation) of the wavepacket. A smaller σ means the particle is more localized initially, which, by the uncertainty principle, implies a wider spread in momentum and thus faster spreading of the packet over time.
  - k_x, k_y: Components of the initial wavevector, determining the initial average momentum $\mathbf{p} = \hbar\mathbf{k}$. These dictate the initial direction and group velocity of the wavepacket. Higher magnitudes of k mean higher initial kinetic energy.
- **Potential Parameters (V_flag, V_0, x_0, y_0, r in V_function):**
  - V_flag: Selects the type of potential landscape (e.g., free particle, barrier, well).
  - V_0: The strength (height or depth) of the potential feature.
    - For a barrier (V_flag == 2): A higher V_0 will lead to more reflection and less transmission (tunneling). The energy of the incident wavepacket relative to V_0 is critical.
    - For a well (V_flag == 3): A deeper well (larger positive V_0 for a potential $-V_0$) can lead to bound states or trapping of the wavepacket.
  - x_0, y_0, r: Define the geometry (center and radius/extent) of the potential feature. These determine how the wavepacket interacts spatially with the potential (e.g., head-on collision, glancing interaction).
- **Simulation Duration/Frames:**
  - time (in solution, determining n_steps): The total physical time the simulation runs for. Longer times allow for observation of more evolved dynamics like spreading, reflection, transmission, or interference.
  - no_fr (in animated_solution): Total number of discrete time steps computed for the animation.
  - no_afr (in animated_solution): Frame capture rate (e.g., save a frame every no_afr steps). Affects the smoothness and computational cost of generating the animation.

5. Conclusion

The provided Julia script offers a capable tool for simulating and visualizing 2D quantum wavepacket dynamics using the Finite Element Method and the Crank-Nicolson scheme. Its modular design allows for easy modification of initial

conditions and potential landscapes, facilitating the study of various quantum phenomena such as tunneling, scattering, and particle confinement within an L-shaped geometry. The use of sparse matrices and appropriate numerical integration techniques contributes to the computational feasibility of the simulations. The script also provides a platform for exploring and comparing various advanced iterative linear solvers, highlighting the trade-offs between direct methods, standalone iterative methods, and more complex approaches like domain decomposition and multigrid in a serial computing context.

Further development could focus on optimizing the iterative solvers, particularly for parallel execution where methods like domain decomposition and multigrid can offer significant advantages, extending to 3D simulations, or incorporating different types of boundary conditions.

6. References and Further Reading

1. **Finite Element Method:**
   - Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z. (2013). *The Finite Element Method: Its Basis and Fundamentals* (7th ed.). Butterworth-Heinemann.
   - Reddy, J. N. (2019). *An Introduction to the Finite Element Method* (4th ed.). McGraw-Hill Education.
2. **Numerical** Solution of PDEs **& Crank-Nicolson:**
   - LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM.
   - Morton, K. W., & Mayers, D. F. (2005). *Numerical Solution of Partial Differential Equations: An Introduction* (2nd ed.). Cambridge University Press.
3. **Iterative Methods for Linear Systems (including BiCGSTAB):**
   - Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems* (2nd ed.). SIAM.
   - Van der Vorst, H. A. (1992). Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing, 13*(2), 631-644.
4. **Domain Decomposition Methods:**
   - Toselli, A., & Widlund, O. (2005). *Domain Decomposition Methods: Algorithms and Theory*. Springer.
   - Smith, B. F., Bjørstad, P. E., & Gropp, W. (1996). *Domain* Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press.
5. **Multigrid Methods:**
   - Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A Multigrid Tutorial* (2nd ed.). SIAM.

- Trottenberg, U., Oosterlee, C. W., & Schüller, A. (2001). *Multigrid*. Academic Press.