Blueprint for a Finite-Difference Eigensolver for the 2D Schrödinger Equation

1 Introduction

The finite-difference method approximates derivatives using differences between values at discrete points. Think of a smooth curve: instead of computing the exact slope at one point, you measure the change between two close points and divide by the distance between them. For example, for a function $f(x)$,

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x}.$$

For the second derivative, which appears in the Laplacian,

$$f''(x_i) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\Delta x^2}.$$

In our problem we solve the 2D time-independent Schrödinger equation

$$-\frac{1}{2}\nabla^2\psi(x,y) + V(x,y)\psi(x,y) = E\psi(x,y),$$

where ∇^2 is the Laplacian operator. We discretize the domain (i.e., replace the continuous region by a grid of points) and approximate the derivatives using finite differences.

2 Problem Setup and Discretization

We consider a rectangular domain of size $L_x \times L_y$ centered at the origin. The wavefunction ψ is set to zero on the boundary (Dirichlet conditions). For simplicity, let us use:

$$L_x = 2, \quad L_y = 2, \quad N_x = 3, \quad N_y = 3,$$

which means we have 3 *interior* points in each direction. The grid spacing is

$$\Delta x = \frac{L_x}{N_x + 1} = \frac{2}{4} = 0.5, \quad \Delta y = 0.5.$$

The interior points for x are:

$$x = [-0.5,\ 0,\ 0.5],$$

and similarly for y.

3 Step-by-Step Code with Matrix Visualizations

Below, we explain each code block and show how the matrices look before and after any updates.

1. Grid Setup

Listing 1: Grid Setup

```
dx = Lx / (Nx + 1)  # 2/4 = 0.5
dy = Ly / (Ny + 1)  # 2/4 = 0.5

x = np.linspace(-Lx/2 + dx, Lx/2 - dx, Nx)  # x = [-0.5, 0, 0.5]
y = np.linspace(-Ly/2 + dy, Ly/2 - dy, Ny)  # y = [-0.5, 0, 0.5]

X, Y = np.meshgrid(x, y, indexing='ij')
```

Visualization: The 2D grid (meshgrid) is

$$X = \begin{pmatrix} -0.5 & -0.5 & -0.5 \\ 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 \end{pmatrix}, \quad Y = \begin{pmatrix} -0.5 & 0 & 0.5 \\ -0.5 & 0 & 0.5 \\ -0.5 & 0 & 0.5 \end{pmatrix}.$$

Here, each pair (X_{ij}, Y_{ij}) represents a point in the domain where the wavefunction ψ will be approximated.

2. Potential Evaluation

Assume the potential is zero everywhere:

Listing 2: Potential Evaluation

```
V = Vfun(X, Y, *args)  # For our case, V(x,y)=0.
```

Before: The function $Vfun$ is applied to each grid point. **After:**

$$V = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

This matrix represents the potential at each interior grid point.

3. Constructing the Finite-Difference Laplacian

The Laplacian is approximated using the 5-point stencil:

$$\nabla^2\psi \approx \frac{\psi_{i+1,j} + \psi_{i-1,j} - 2\psi_{i,j}}{\Delta x^2} + \frac{\psi_{i,j+1} + \psi_{i,j-1} - 2\psi_{i,j}}{\Delta y^2}.$$

Since there are $N_{\text{total}} = N_x \times N_y = 9$ unknowns, we will construct arrays that represent the diagonals of a 9×9 matrix.

a) Main Diagonal Initialization

Listing 3: Main Diagonal of Laplacian

```
N_total = Nx * Ny  # 9 points
lap_main = (-2.0/dx**2 - 2.0/dy**2) * np.ones(N_total)
```

Explanation: With $\Delta x = \Delta y = 0.5$, we have:

$$\frac{1}{dx^2} = \frac{1}{0.25} = 4.$$

So,

$$-2\left(\frac{1}{dx^2}\right) - 2\left(\frac{1}{dy^2}\right) = -2(4) - 2(4) = -8 - 8 = -16.$$

Before: The call `np.ones(9)` creates:

$$[1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1].$$

After: Multiplying by -16 gives:

$$\texttt{lap\_main} = [-16,\ -16,\ -16,\ -16,\ -16,\ -16,\ -16,\ -16,\ -16].$$

Each element represents the contribution from the center point in the finite-difference formula.

b) Off-Diagonals in the y-Direction

Listing 4: Off-Diagonals in y-Direction

```
lap_off_y = 1.0/dy**2 * np.ones(N_total - 1)
```

Before Update: `np.ones(8)` creates:

$$[1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1].$$

Multiplying by $1/dy^2 = 4$ gives initially:

$$\texttt{lap\_off\_y} = [4,\ 4,\ 4,\ 4,\ 4,\ 4,\ 4,\ 4].$$

These values represent the connection between a grid point and its immediate neighbor in the y-direction.

Removing Wrap-Around Connections: We must not connect the last point in one row to the first point in the next row. This is done by:

Listing 5: Removing Wrap-Around in y-Direction

```
for i in range(1, Nx):
    idx = i*Ny - 1
    lap_off_y[idx] = 0.0
```

Updates:

- For $i = 1$: idx $= 1 \times 3 - 1 = 2$. Set `lap_off_y`$[2] = 0$ so that index 2 is not connected to index 3.
- For $i = 2$: idx $= 2 \times 3 - 1 = 5$. Set `lap_off_y`$[5] = 0$.

After Update:

$$\texttt{lap\_off\_y} = [4,\ 4,\ 0,\ 4,\ 4,\ 0,\ 4,\ 4].$$

Each element here connects adjacent points in the y-direction, except at the row boundaries where we set the connection to zero.

c) Off-Diagonals in the x-Direction

Listing 6: Off-Diagonals in x-Direction

```
lap_off_x = 1.0/dx**2 * np.ones(N_total - Ny)
```

Before and After: `np.ones(6)` creates:

$$[1,\ 1,\ 1,\ 1,\ 1,\ 1].$$

Multiplying by 4 gives:

$$\texttt{lap\_off\_x} = [4,\ 4,\ 4,\ 4,\ 4,\ 4].$$

These represent the connection between points in the x-direction (neighbors above and below in the grid).

d) Assembling the Laplacian Matrix

We combine the diagonals into a sparse matrix:

Listing 7: Assembling the Laplacian

```
diagonals = [lap_main, lap_off_y, lap_off_y, lap_off_x, lap_off_x]
offsets   = [0, 1, -1, Ny, -Ny]
L = sp.diags(diagonals, offsets, shape=(N_total, N_total), format='csr')
```

Interpretation:

- The **main diagonal** (offset 0) has the values from `lap_main` (all -16).
- The two **off-diagonals** with offsets $+1$ and -1 use `lap_off_y` to connect neighboring points in the y-direction.
- The off-diagonals with offsets $+Ny$ and $-Ny$ (i.e., $+3$ and -3) use `lap_off_x` to connect points in the x-direction.

Dense Form of L: For clarity, the full 9×9 matrix (if converted to a dense matrix) would look like

$$L = \begin{pmatrix} -16 & 4 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 4 & -16 & 4 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & -16 & 0 & 0 & 4 & 0 & 0 & 0 \\ 4 & 0 & 0 & -16 & 4 & 0 & 4 & 0 & 0 \\ 0 & 4 & 0 & 4 & -16 & 4 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 & 4 & -16 & 0 & 0 & 4 \\ 0 & 0 & 0 & 4 & 0 & 0 & -16 & 4 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 4 & -16 & 4 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 4 & -16 \end{pmatrix}.$$

Notice that the off-diagonal zeros correspond to the blocked connections (wrap-around issues).

4. Hamiltonian Assembly

The Hamiltonian is given by

$$H = -\frac{1}{2}L + V,$$

with V being added as a diagonal matrix.

Listing 8: Hamiltonian Assembly

```
H = -0.5 * L + sp.diags(V.ravel(order='C'), 0, format='csr')
```

Before: L is as shown above, and V is a 3x3 matrix of zeros (flattened to a 9-element vector). **After:** Every element of L is multiplied by -0.5. For example:

$$-0.5 \times (-16) = 8 \quad \text{(main diagonal)}, \quad -0.5 \times 4 = -2 \quad \text{(off-diagonals)}.$$

Thus, the dense form of H becomes

$$H = \begin{pmatrix} 8 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ -2 & 8 & -2 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 8 & 0 & 0 & -2 & 0 & 0 & 0 \\ -2 & 0 & 0 & 8 & -2 & 0 & -2 & 0 & 0 \\ 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -2 & 0 & -2 & 8 & 0 & 0 & -2 \\ 0 & 0 & 0 & -2 & 0 & 0 & 8 & -2 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & -2 & 8 & -2 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & -2 & 8 \end{pmatrix}.$$

Each element now represents the combined effect of the kinetic energy operator (from the Laplacian) and the potential (which is zero in this example).

5. Solving the Eigenvalue Problem and Normalization

The eigensolver finds solutions to

$$H\psi = E\psi.$$

Listing 9: Solving the Eigenvalue Problem

```
E, psi_flat = spla.eigsh(H, k=neigs, sigma=E0, which='LM')
```

Explanation: The vector ψ_{flat} contains eigenfunctions in a flattened (one-dimensional) form. Each eigenvector has 9 components corresponding to our 9 grid points.

Then, each eigenvector is reshaped into a 3×3 matrix and normalized:

Listing 10: Reshaping and Normalizing

```
psi = np.empty((neigs, Nx, Ny), dtype=complex)
for n in range(neigs):
    psi_n = psi_flat[:, n].reshape((Nx, Ny), order='C')
    norm = np.sqrt(np.sum(np.abs(psi_n)**2) * dx * dy)
    psi[n, :, :] = psi_n / norm
```

Before Normalization: An eigenvector might look like

$$\psi_{\text{flat}}^{(n)} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_8 \end{pmatrix}.$$

After Reshaping: It becomes a 3×3 matrix:

$$\psi^{(n)} = \begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{pmatrix}.$$

After Normalization: The entries are scaled so that

$$\sum_{i,j} |\psi_{ij}^{(n)}|^2 \Delta x \Delta y = 1.$$

4 Explanation of `lap_off_x` and `lap_off_y` Arrays

Below we provide a detailed, visual explanation of the off-diagonals for the x and y directions.

Visualizing the Grid

Consider the 3x3 grid with flattened indices:

$$\begin{array}{ccc} 0:(-0.5,-0.5) & 1:(-0.5,\ 0) & 2:(-0.5,\ 0.5) \\ 3:(0,\ -0.5) & 4:(0,\ 0) & 5:(0,\ 0.5) \\ 6:(0.5,-0.5) & 7:(0.5,\ 0) & 8:(0.5,\ 0.5) \end{array}$$

Horizontal Connections (`lap_off_y`)

- **Purpose:** Connect each grid point to its left and right neighbors.
- **Coefficient:** Each connection gets a value $1/\Delta y^2 = 4$ (for $\Delta y = 0.5$).

Connections:

$$\begin{array}{ll} 0 \leftrightarrow 1, & 1 \leftrightarrow 2, \quad \text{(Row 0)} \\ 3 \leftrightarrow 4, & 4 \leftrightarrow 5, \quad \text{(Row 1)} \\ 6 \leftrightarrow 7, & 7 \leftrightarrow 8, \quad \text{(Row 2)} \end{array}$$

The For Loop: A loop sets the connection between the last index of a row and the first index of the next row to zero:

- For row 0: Index 2 (end) should not connect to index 3 (start of row 1).
- For row 1: Index 5 (end) should not connect to index 6 (start of row 2).

Thus, the `lap_off_y` array becomes:

$$[4,\ 4,\ 0,\ 4,\ 4,\ 0,\ 4,\ 4].$$

Vertical Connections (`lap_off_x`)

- **Purpose:** Connect each grid point to its above and below neighbors.
- **Coefficient:** Each connection gets a value $1/\Delta x^2 = 4$ (for $\Delta x = 0.5$).

Connections:

$$\begin{array}{lll} 0 \leftrightarrow 3, & 1 \leftrightarrow 4, & 2 \leftrightarrow 5, \\ 3 \leftrightarrow 6, & 4 \leftrightarrow 7, & 5 \leftrightarrow 8. \end{array}$$

In the flattened array, a jump of N_y (here, 3) corresponds to moving vertically in the grid.

5 Conclusion and Final Words

This document provides a comprehensive, step-by-step explanation of the finite-difference method for solving the 2D Schrödinger equation. We illustrated the grid, showed how the Laplacian is built using finite differences, and explained the off-diagonals (`lap_off_y` and `lap_off_x`) with visual aids. The use of a loop in setting up `lap_off_y` ensures that the physical structure of the grid is respected by preventing connections that cross row boundaries.