January Report:
Adapting Particle Transport Monte Carlo Code to Solve the 1D Time Independent Schrödinger Equation

물리학과 2021313130 이유나

2025. 02. 04

Recap

- The following differential equations have the same form.

1D Time Independent Schrödinger Equation

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

Steady State Neutron Diffusion Equation

$$-D\frac{d^2}{dx^2}\phi(x) + \Sigma_a\phi(x) = \frac{\nu}{k}\Sigma_f\phi(x)$$

ϕ: neutron flux,
D: diffusion coefficient,
Σ_a: macroscopic absorption cross section,
k: multiplication factor,
Σ_f: macroscopic fission cross section ,
ν: average number of neutrons released per fission

Recap

- The particle transport Monte Carlo code already exists for neutron diffusion.

  ex) MCNP (Monte Carlo N-Particle)

- We can interpret the result of the Monte Carlo simulation as the solution of the Schrödinger equation.

Recap

- The Monte Carlo method is highly effective for solving problems with **complicated internal geometries.**
- Existing particle transport codes (MCNP, OpenMC, etc.) have been developed and validated in nuclear engineering. **Adapting these codes allows for efficient reuse of mature algorithms** and demonstrates their versatility in addressing new problems, such as quantum mechanics.
- While Quantum Monte Carlo (QMC) methods are well-established for solving the Schrödinger equation, the direct adaptation of particle transport Monte Carlo codes for quantum problems remains **a relatively unexplored approach.**

Current Progress

1) ~~Study Monte Carlo particle transport method~~ ✓ **Done**
2) Implement code for fundamental mode ✓ **In progress**
3) Implement code for higher modes
4) Interpret as a Schrödinger equation solution
5) Add quantum effects to code
6) Apply to more complex situations
   - Various potentials
   - Higher Dimensions
7) Study numerical and QMC methods
8) Compare performance between methods

Algorithm

1. Input boundary conditions, total number of neutrons, and cycles.
2. Initialize two stacks (**fission banks**) to store neutrons.
3. Fill one stack (**parent fission bank**) with the total number of neutrons. Each neutron has an initial position and direction, which is determined by the random number generator.

Algorithm

4. Simulate neutron diffusion.

- For the total number of cycles:
  - While the parent fission bank is not empty:
    - Pop a neutron from the parent fission bank.
    - While the neutron is not terminated:
      - Generate a random number and calculate the distance for the neutron to move.
      - Calculate the new position, considering boundary conditions.
      - Sample the collision type at the new position and add any produced neutrons to the child fission bank.
      - Count neutron flux at each position.
    - Tally neutron flux and k value.
    - Update the child fission bank to become the new parent fission bank.

Algorithm

5. Calculate mean neutron flux and standard deviation.
6. Calculate mean k value and standard deviation.
7. Plot mean neutron flux with one sigma confidence intervals.

Key Variables

boundary_condition // 0 for vacuum, 1 for reflective
N // total number of neutrons
M // total number of cycles
active_cycle // number of active cycles
inactive_cycle // number of inactive cycles
width // width of the reactor (potential well)
sigma_s // scattering cross section
sigma_c // capture cross section
sigma_f // fission cross section

Key Variables

struct neutron // stores the neutron's position and direction

struct neutron *parent_fission_bank // array of struct neutron

struct neutron *child_fission_bank // array of struct neutron

gsl_vector *neutron_flux // stores neutron flux for each cycle

double k_cycle // stores k value for each cycle

double k_average // average of k value over all cycles

double k_sample_standard_deviation // standard deviation of k values over all cycles

double k1, k2 // theoretical k values for the first and second mode

Key Functions

initialize_rng() // initializes the random number generator instance

random_number_generator() // generates a random number

initialize_fission_bank() // initializes neutron positions and directions in the parent fission bank

determine_direction() // determines neutron directions

simulate_neutron_diffusion() // simulates neutron diffusion for one cycle

sample_collision_type() // samples collision type

add_fission_bank() // adds neutrons to child fission bank when fission occurs

flux_counter() // updates the neutron_flux variable

fission_bank_size_adjustment() // adjusts the child fission bank size to N

How to Run the Program

- Ensure all files are in the same directory.
  - trial1.c // main source code
  - trial1_functions.s // custom function source code
  - trial1.h // header file
  - input.txt // stores input data
  - Makefile // build script
  - draw_flux_data.py // plotting script
  - flux_data.txt // neutron flux results
  - text_data.txt // k value results and information for debugging
  - position_data.txt // neutron position

How to Run the Program

- Open input.txt and specify the desired values.
  - ➢ boundary_condition, N, M

input.txt

```
1   Boundary condition (zero = vacuum, one = reflective): 0
2   Number of neutrons: 100000
3   Total number of cycles: 100
```

- Run the following command in the terminal.
  - ➢ make; ./trial1
- The program will take a few seconds to minutes to run.
- Check the terminal output for the k value and standard deviation.

How to Run the Program

- Adjust draw_flux_data.py to match boundary conditions and execute it.

```
60              # plt.ylim(0.3, 0.5)  # for reflective
61              plt.ylim(0.0, 0.4)  # for vacuum
```

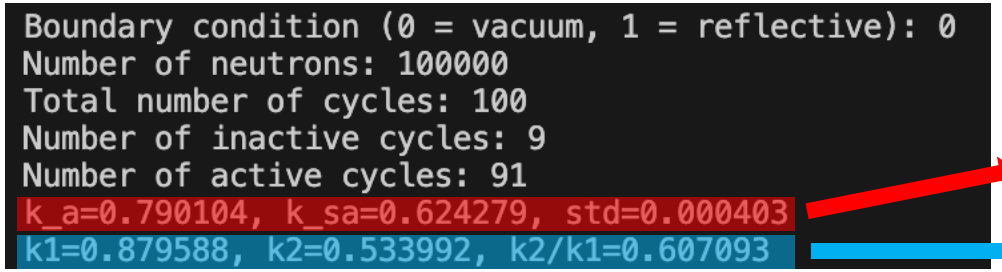
```
79          #plt.ylim(0.3, 0.5) # for reflective
80          plt.ylim(0.1, 0.5) # for vacuum
```

Results – (1) Vacuum Boundary Condition

- Input values
  - Boundary condition: 0
  - Number of neutrons: 100000
  - Total number of cycles: 100
- Variables
  - width = 20.0
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06

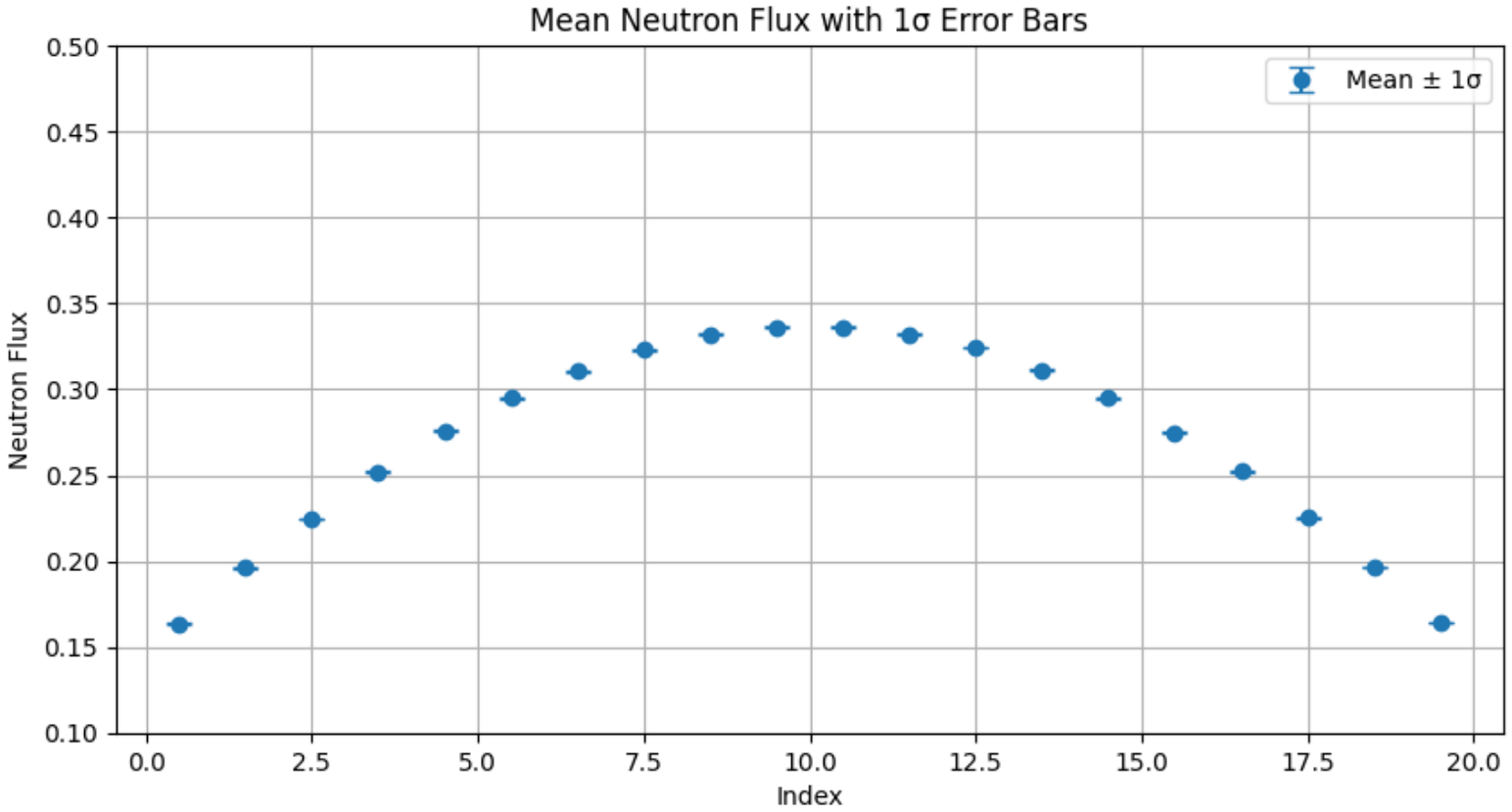
Results – (1) Vacuum Boundary Condition

- Terminal output



- $k_{average} = 0.790104 \pm 0.000403$ (68.27% CI)
- k1 is not within the range of $k_{average}$

Results – (1) Vacuum Boundary Condition



Results - (1) Vacuum Boundary Condition

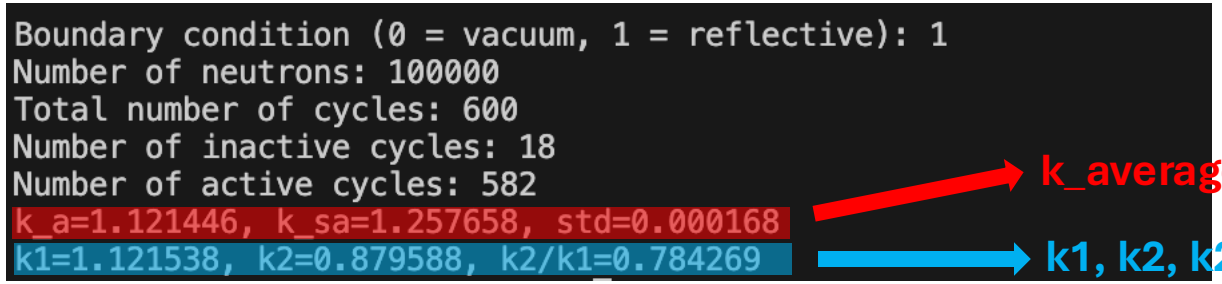
- $k_{average}$ being smaller than $k1$ means **bigger leakage**.
- In the Monte Carlo simulation, the flux at the boundary must be **theoretically zero**. Since every neutron that exits the boundary is terminated, and neutrons **cannot re-enter**, unlike in the reflective boundary condition, **this results in larger overall leakage, leading to a smaller k value**.
- In the **1D model**, the neutron's traveled distance represents its **3D movement**. However, since the full distance is applied in 1D, the neutron **effectively travels farther than it would in 3D**.
- Instead of **sampling the direction in 1D** (left or right), a better approach is to **sample the solid angle** and project the traveled distance onto the x-axis.
- **To Do:** Implement a function to refine both approaches and **verify the results**.

Results - (2) Reflective Boundary Condition

- Input values
  - Boundary condition: 1
  - Number of neutrons: 100000
  - Total number of cycles: 600
- Variables
  - width = 20.0
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06

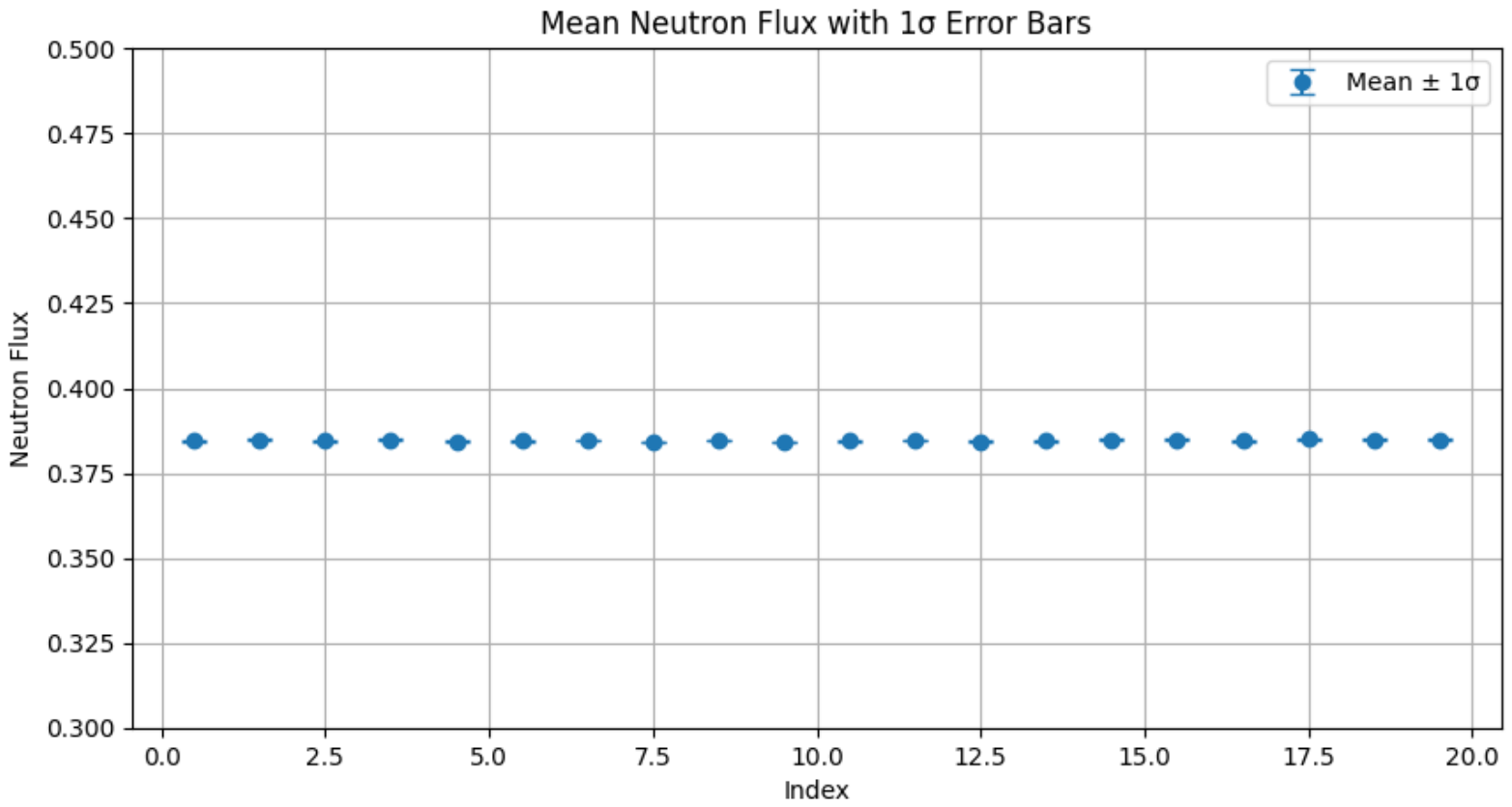
Results - (2) Reflective Boundary Condition

- Terminal output



- $k_{average} = 1.121446 \pm 0.000168$ (68.27% CI)
- k1 is within the range of $k_{average}$

Results - (2) Reflective Boundary Condition



To Do List

1) ~~Study Monte Carlo particle transport method~~
2) Implement code for fundamental mode
3) Implement code for higher modes
4) Interpret as a Schrödinger equation solution
5) Add quantum effects to code
6) Apply to more complex situations
   - Various potentials
   - Higher Dimensions
7) Study numerical and QMC methods
8) Compare performance between methods

Resources

- Alex F Bielajew . (2020). *Fundamentals of the Monte Carlo method for neutral and charged particle transport*.
- Brown, F. B. (n.d.). *Monte Carlo Techniques for Nuclear Systems*. Lecture.
- *Boundary conditions - diffusion equation*. Nuclear Power. (2021, October 28). https://www.nuclear-power.com/nuclear-power/reactor-physics/neutron-diffusion-theory/boundary-conditions-diffusion-equation/
- Leppänen, J. (2007). *Development of a new Monte Carlo Reactor Physics Code* (thesis). *Development of a new Monte Carlo reactor physics code*. VTT, Espoo.
- Shentu, J., Yun, S.-H., & Cho, N.-Z. (2007). A Monte Carlo method for solving heat conduction problems with complicated geometry. *Nuclear Engineering and Technology*, *39*(3), 214. https://doi.org/10.5516/net.2007.39.3.207