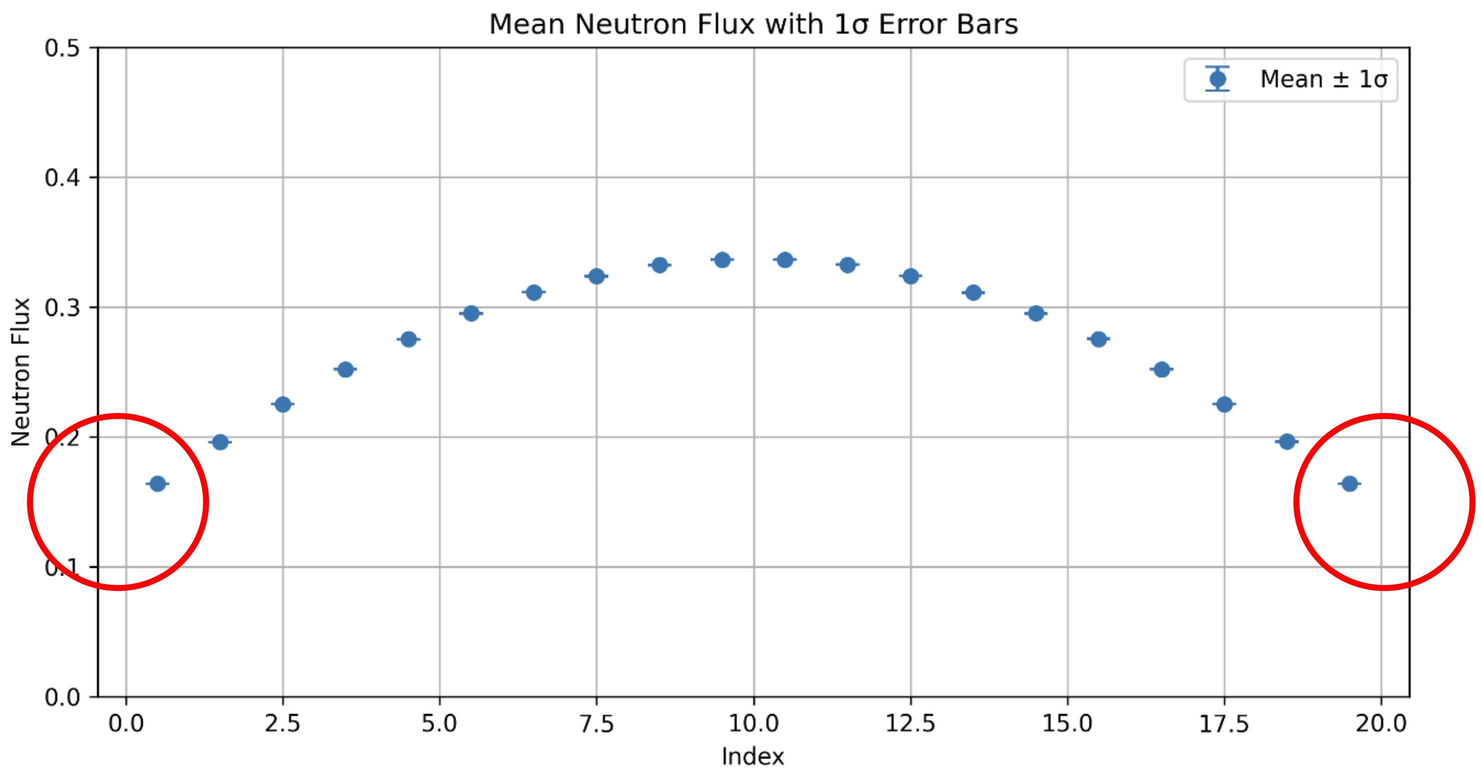
February Report: Adapting Particle Transport Monte Carlo Code to Solve the 1D Time Independent Schrödinger Equation

물리학과 2021313130 이유나

2025. 03. 07

Recap

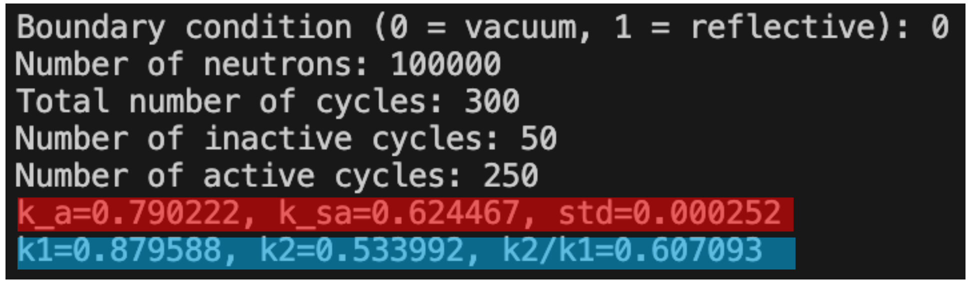
- Vacuum boundary condition



- Ideally, in vacuum boundary condition the neutron flux must be zero at the boundary.
- However, the results show us the flux is above zero.

Recap

- Vacuum boundary condition



```
Boundary condition (0 = vacuum, 1 = reflective): 0
Number of neutrons: 100000
Total number of cycles: 300
Number of inactive cycles: 50
Number of active cycles: 250
k_a=0.790222, k_sa=0.624467, std=0.000252
k1=0.879588, k2=0.533992, k2/k1=0.607093
```

- $\mathrm{k1} = 0.879588$
- $\mathrm{k}_{\mathrm{average}} = 0.790222 \pm 0.000252$ (68.27% Cl)
- k1 is not within the range of $\mathrm{k}_{\mathrm{average}}$

Recap

- 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.

Recap

- Conclusion: Two approaches can made.

1) Re-entering neutrons: Allow exiting neutrons to re-enter, reducing leakage but assign weights to differentiate them from non-exiting neutrons.
2) 1D Projection: Sample the solid angle and project the traveled distance onto the x-axis to represent the true movement in the 1D model.

Current Progress

1) ~~Study Monte Carlo particle transport method~~ ✓ **Done**
2) ~~Implement code for fundamental mode~~ ✓ **Done**
3) ~~Implement code for higher modes~~ ✓ **Done**
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

Original Algorithm

1. Define boundary conditions, total number of neutrons, and the number of inactive and active cycles.
2. Initialize two stacks **(fission banks)** for neutron storage.
   - The **parent fission bank** stores neutrons for the current cycle to simulate.
   - The **child fission bank** stores fission neutrons for the next cycle.
3. Assign each neutron in the parent fission bank an initial position and direction using a random number generator.

Original 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 alive:
            - Generate a random number and compute the distance for movement.
            - Compute the new position, considering boundary conditions.
            - If the neutron survives:
                - Sample the collision type at the new position.
                - If fission occurs, add produced neutrons to the child fission bank.
                - Tally neutron flux and k value.
        - Rescale the child fission bank to conserve the neutron population.
        - Update the child fission bank to become the new parent fission bank

Original 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.

Updated Algorithm for 1D Projection (First Version)

- Each neutron now only has distance.
- Direction is included in the distance calculation, so it does not need to be stored separately.

- Initial algorithm:
  - Generate a random number ξ
  - Compute distance:
$$distance = \frac{-log(\xi)}{\Sigma_s + \Sigma_c + \Sigma_f}$$
  - Update position based on neutron direction:
$$new\ position = initial\ position \pm distance$$

Updated Algorithm for 1D Projection (First Version)

- Updated algorithm:
  - Generate two random numbers ξ_1 and ξ_2
  - Compute distance:

$$distance = \frac{-log(\xi_1)}{\Sigma_s + \Sigma_c + \Sigma_f} * (2\xi_2 - 1)$$

  - Update position:

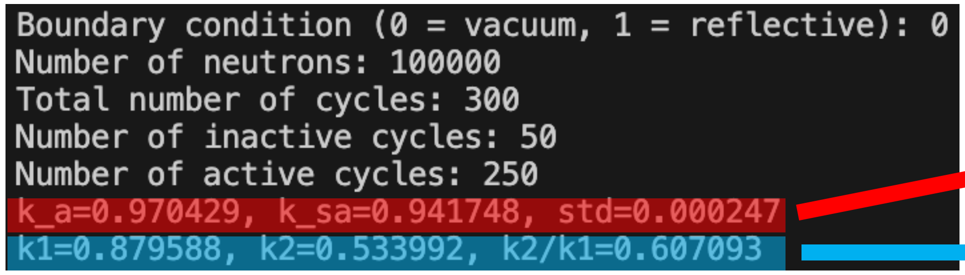
$$new\ position = initial\ position + distance$$

Result – (1) Vacuum Boundary Condition

- Input values
  - Boundary condition: 0
  - Number of neutrons: 100000
  - Number of inactive cycles: 50
  - Number of active cycles: 250
- Variables
  - **width = 20.0**
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06
  - num_bins = 20

Result – (1) Vacuum Boundary Condition

- Terminal output



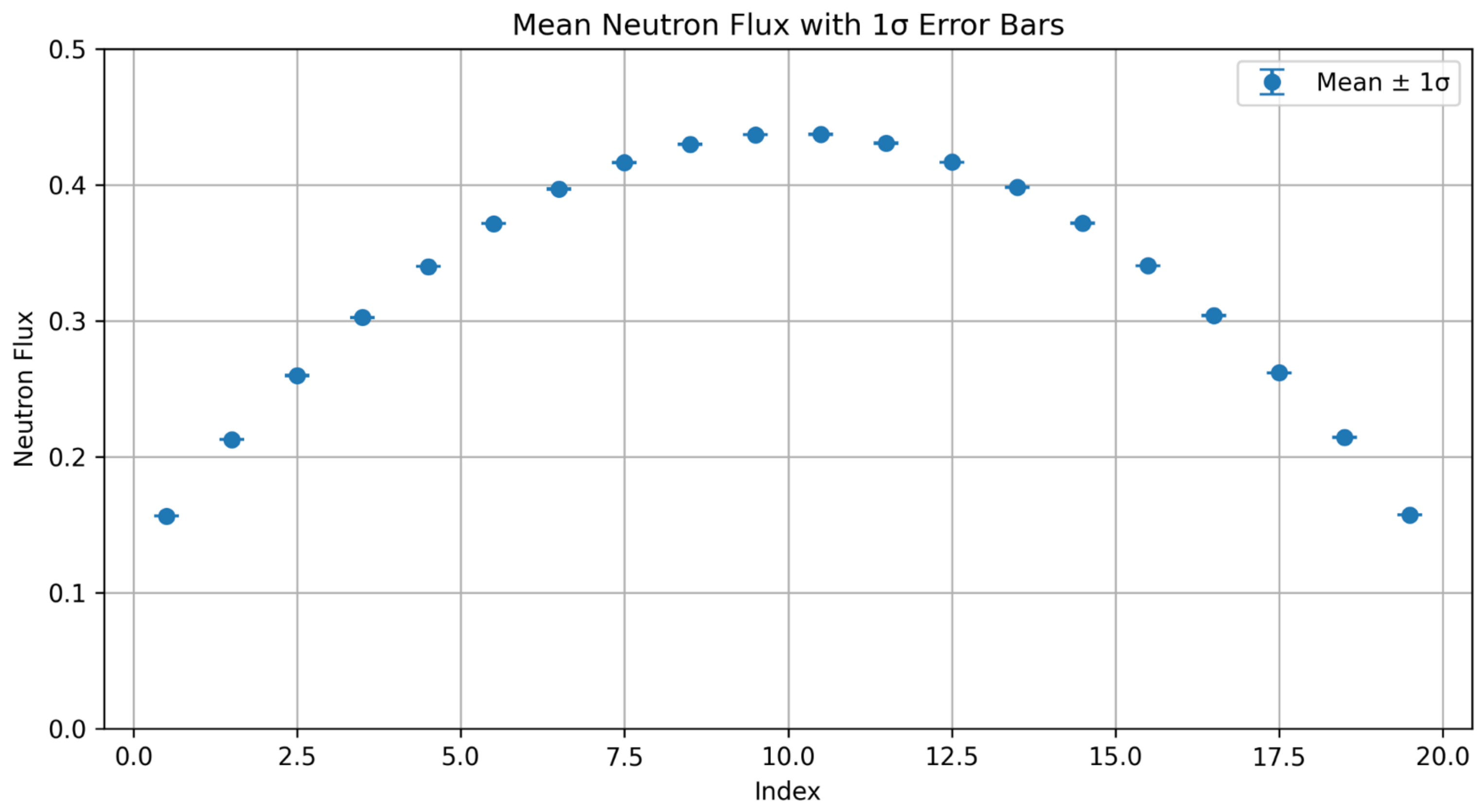
```
Boundary condition (0 = vacuum, 1 = reflective): 0
Number of neutrons: 100000
Total number of cycles: 300
Number of inactive cycles: 50
Number of active cycles: 250
k_a=0.970429, k_sa=0.941748, std=0.000247
k1=0.879588, k2=0.533992, k2/k1=0.607093
```

k_average, k_squared_average, k_sample_standard deviation

k1, k2, k2/k1

- $k_{\text{average}} = 0.970429 \ \pm 0.000247$ (68.27% Cl)
- k1 is bigger than k_{average} meaning leakage has been reduced
- k1 is not within the range of k_{average}

Result – (1) Vacuum Boundary Condition



Updated Algorithm for 1D Projection (Second Version)

- Non-analog simulation is used.
- Analog simulation (originally used)
  - Simulated neutrons behave analogously to real neutrons.
  - A neutron is born with **weight = 1.0,** which remains **unchanged** until it is killed.
  - Scores are tallied with a weight of 1.0.

- Non-analog simulation
  - Simulated neutrons no longer represent real neutrons.
  - A neutron is born with **weight = 1.0,** but its weight is **adjusted** through **force scattering and Russian roulette.**
  - **Scores are tallied using the neutron’s weight.**
  - Neutrons are not immediately killed; instead, their weight is modified. More neutrons are simulated in a single run, reducing the standard deviation.

Updated Algorithm for 1D Projection (Second Version)

1. Define boundary conditions, total number of neutrons, and the number of inactive and active cycles.
2. Initialize two stacks **(fission banks)** for neutron storage.
   - The **parent fission bank** stores neutrons for the current cycle to simulate.
   - The **child fission bank** stores fission neutrons for the next cycle.
3. Assign each neutron in the parent fission bank an initial position using a random number generator, with an initial weight if 1.0.

Updated Algorithm for 1D Projection (Second Version)

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 alive:
      - Generate a random number and compute the distance for movement.
      - Compute the new position, considering boundary conditions.
      - If the neutron survives:
        - Perform fission; if fission neutrons are produced, add them to the child fission bank.
        - Tally k value and neutron flux.
        - Forcefully scatter the neutron and adjust its weight.
        - Apply Russian roulette: if the neutron weight falls below a threshold, either kill it or reset its weight to 1.0.
    - Reweight the child fission bank to ensure total weight conservation.
    - Update the child fission bank to become the new parent fission bank.

Updated Algorithm for 1D Projection (Second Version)

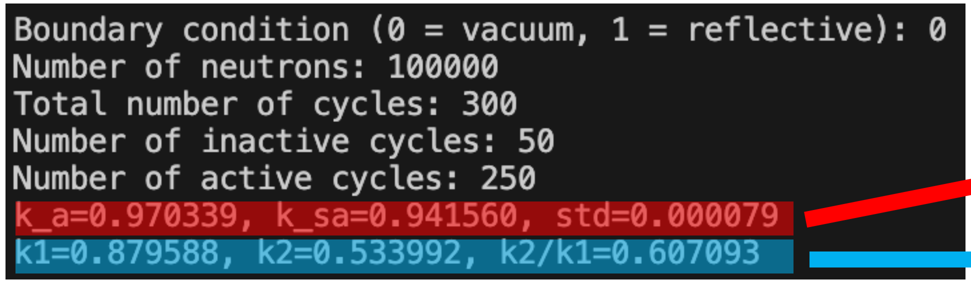
5. Calculate mean neutron flux and standard deviation.
6. Calculate mean k value and standard deviation.
7. Plot mean neutron flux with one sigma confidence intervals.

Result – (1) Vacuum Boundary Condition

- Input values
  - Boundary condition: 0
  - Number of neutrons: 100000
  - Number of inactive cycles: 50
  - Number of active cycles: 250
- Variables
  - **width = 20.0**
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06
  - num_bins = 20

Result – (1) Vacuum Boundary Condition

- Terminal output



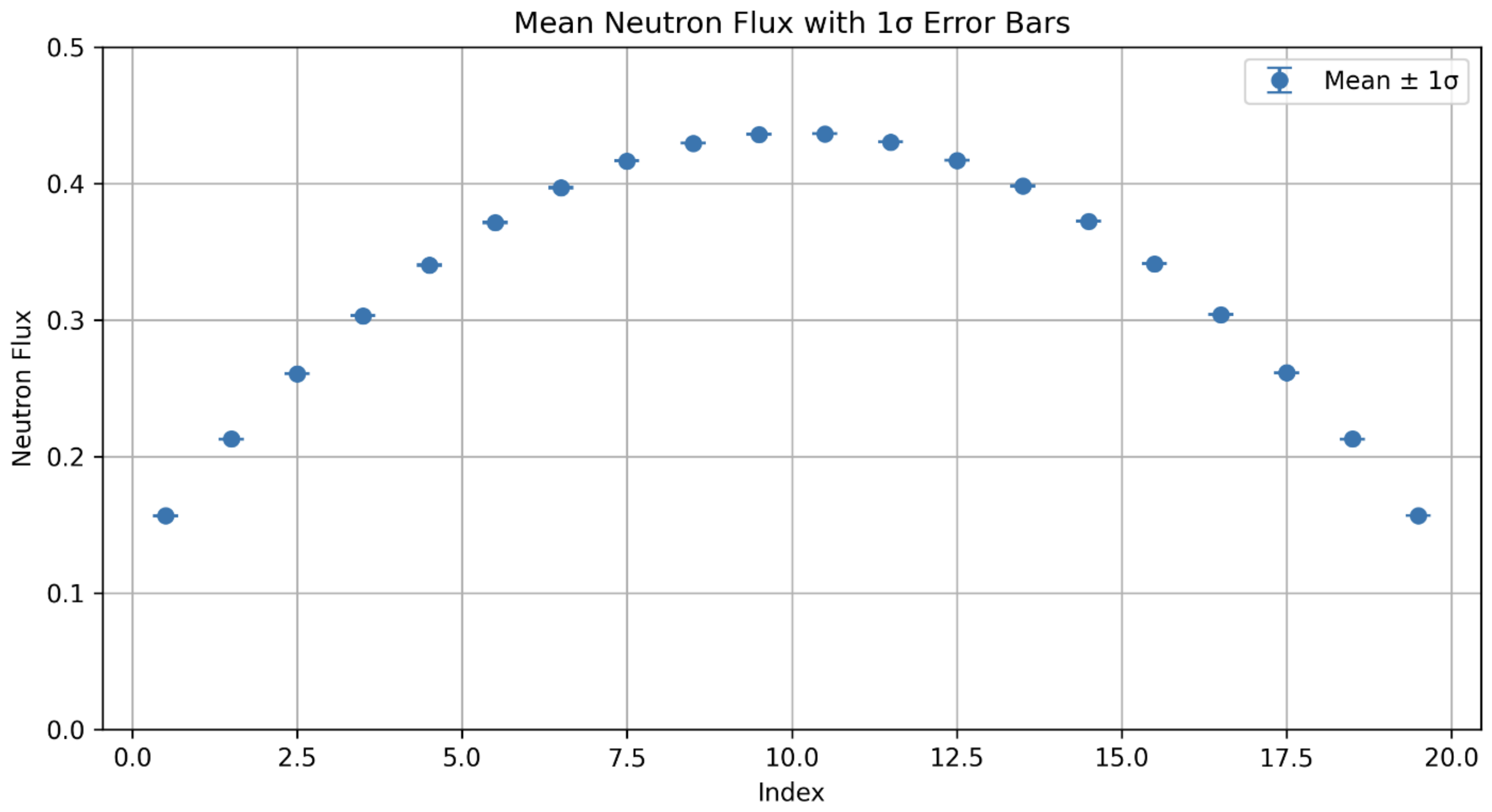
```
Boundary condition (0 = vacuum, 1 = reflective): 0
Number of neutrons: 100000
Total number of cycles: 300
Number of inactive cycles: 50
Number of active cycles: 250
k_a=0.970339, k_sa=0.941560, std=0.000079
k1=0.879588, k2=0.533992, k2/k1=0.607093
```

k_average, k_squared_average, k_sample_standard deviation

k1, k2, k2/k1

- $k_{average} = 0.970339 \pm 0.000079$ (68.27% CI)
- k1 is bigger than $k_{average}$ meaning leakage has been reduced
- k1 is not within the range of $k_{average}$

Result – (1) Vacuum Boundary Condition



Updated Algorithm for Re-entering Neutrons (First Version)

- For the vacuum boundary condition, if a neutron exits the boundaries, reflect it back inside, similar to a reflective boundary condition.
    - However, in this case, the parent neutron's weight is inverted: if initially w, it becomes $-w$.
- If the parent neutron has negative weight, any fission neutrons it produced will also have negative weight; otherwise, they will have positive weight.
- Negative weight neutrons are canceled out using the nearest positive weight neutron.
- **Disclaimer:** In this code, fission neutron weights were initially set to ± 1.0 instead of $\pm k$. This was later corrected in other codes upon realizing that the weights should be $\pm k$.

Updated Algorithm for Re-entering Neutrons (First Version)

4. Simulate neutron diffusion.

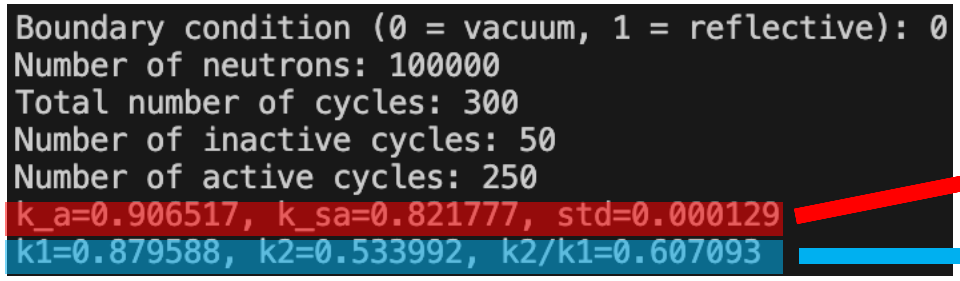
- For the total number of cycles:
    - While the parent fission bank is not empty:
        - Pop a neutron from the parent fission bank.
        - While the neutron is alive:
            - Generate a random number and compute the distance for movement.
            - Compute the new position and parent weight, considering boundary conditions.
            - Perform fission; if fission neutrons are produced, add them to the child fission bank.
            - Tally k value and neutron flux.
            - Forcefully scatter the neutron and adjust its weight.
            - Apply Russian roulette: if the neutron weight falls below a threshold, either kill it or reset its weight to 1.0.
        - **If vacuum boundary condition:**
            - **Sort the child fission bank using quicksort.**
            - **Cancel negative weight neutrons using positive weight neutrons.**
        - Rescale the child fission bank to ensure total weight conservation.
        - Update the child fission bank to become the new parent fission bank.

Result – (1) Vacuum Boundary Condition

- Input values
  - Boundary condition: 0
  - Number of neutrons: 100000
  - Number of inactive cycles: 50
  - Number of active cycles: 250
- Variables
  - **width = 20.0**
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06
  - num_bins = 20

Result – (1) Vacuum Boundary Condition

- Terminal output



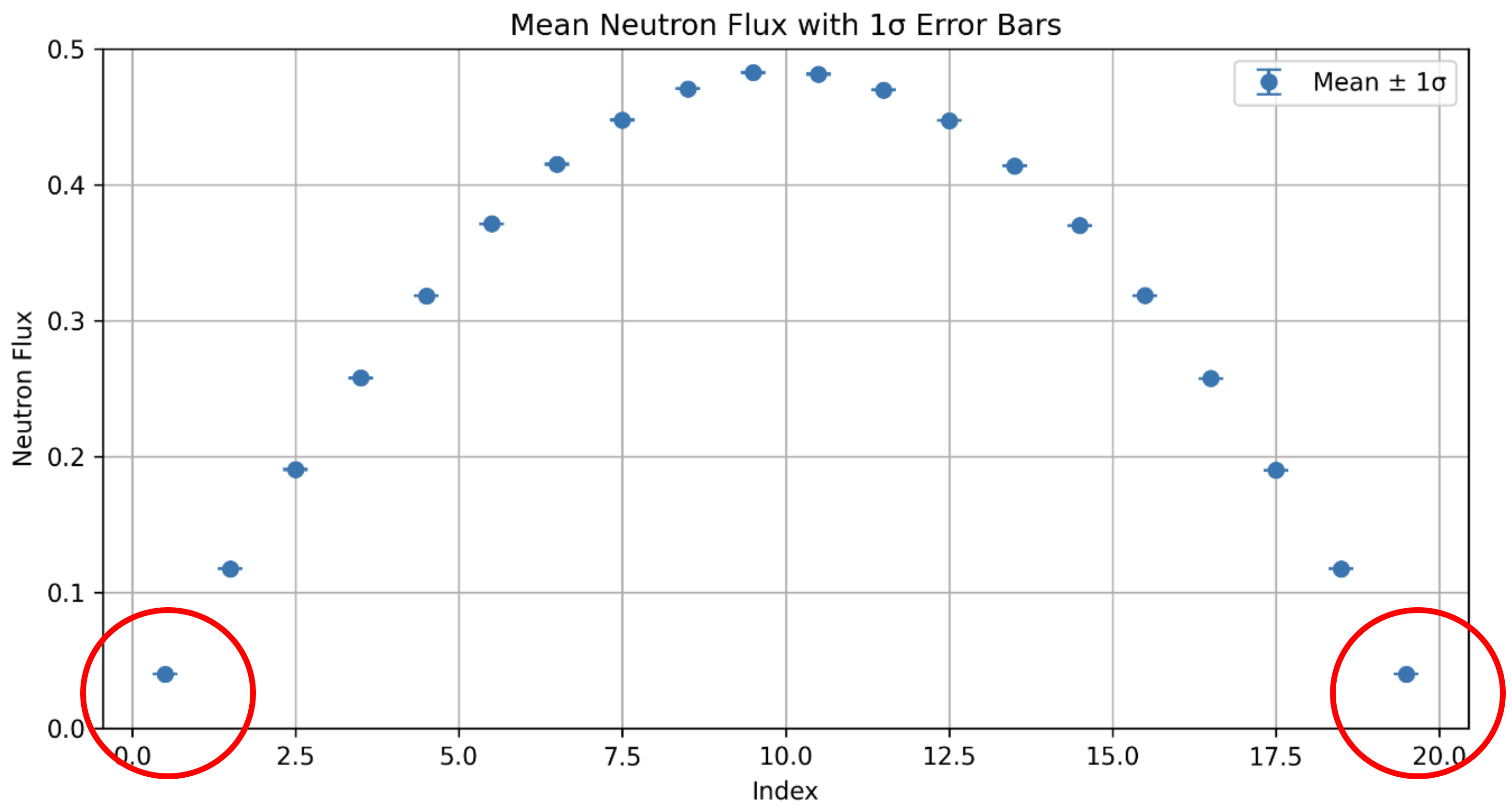
```
Boundary condition (0 = vacuum, 1 = reflective): 0
Number of neutrons: 100000
Total number of cycles: 300
Number of inactive cycles: 50
Number of active cycles: 250
k_a=0.906517, k_sa=0.821777, std=0.000129
k1=0.879588, k2=0.533992, k2/k1=0.607093
```

k_average, k_squared_average, k_sample_standard deviation

k1, k2, k2/k1

- $k_{average} = 0.906517 \pm 0.000129$ (68.27% Cl)
- k1 is similar to $k_{average}$
- k1 is not within the range of $k_{average}$

Result – (1) Vacuum Boundary Condition



- Neutron flux has moved closer to zero.

Updated Algorithm for Re-entering Neutrons (Second Version)

- For each bin, negative weight neutrons are canceled by counting the total number of neutrons and the number of negative neutrons, then adjusting the neutron count accordingly.
- Example: If bin 1 contains 10 neutrons, and 2 have negative weight, the final count in bin 1 becomes 6 neutrons.
- Since a large number of neutrons are simulated, each bin always contains more positive weight neutrons than negative weight neutrons.
- All neutron within each bin are **redistributed to the midpoint of the bin.**
- **Disclaimer:** In this code, fission neutron weights were initially set to ± 1.0 instead of $\pm k$. This was later corrected in other codes upon realizing that the weights should be $\pm k$.

Updated Algorithm for Re-entering Neutrons (Second Version)

4. Simulate neutron diffusion.

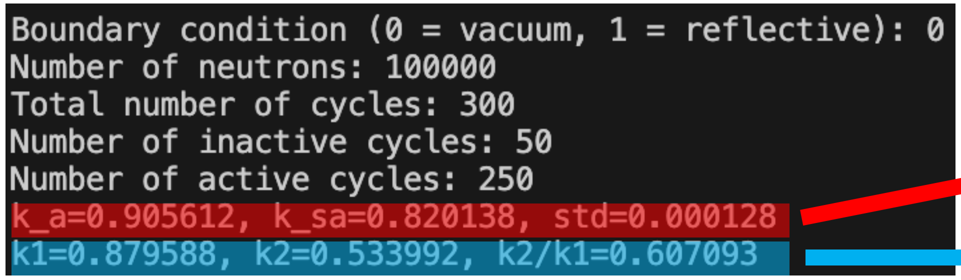
- For the total number of cycles:
    - While the parent fission bank is not empty:
        - Pop a neutron from the parent fission bank.
        - While the neutron is alive:
            - Generate a random number and compute the distance for movement.
            - Compute the new position and parent weight, considering boundary conditions.
            - Perform fission; if fission neutrons are produced, add them to the child fission bank.
            - Tally k value and neutron flux.
            - Forcefully scatter the neutron and adjust its weight.
            - Apply Russian roulette: if the neutron weight falls below a threshold, either kill it or reset its weight to 1.0.
        - **If vacuum boundary condition:**
            - **Count negative weight neutrons in each bin and adjust the total neutron count.**
            - **Redistribute the neutrons to the midpoint of each bin.**
        - Rescale the child fission bank to ensure total weight conservation.
        - Update the child fission bank to become the new parent fission bank.

Result – (1) Vacuum Boundary Condition

- Input values
  - Boundary condition: 0
  - Number of neutrons: 100000
  - Number of inactive cycles: 50
  - Number of active cycles: 250
- Variables
  - **width = 20.0**
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06
  - num_bins = 20

Result – (1) Vacuum Boundary Condition

- Terminal output



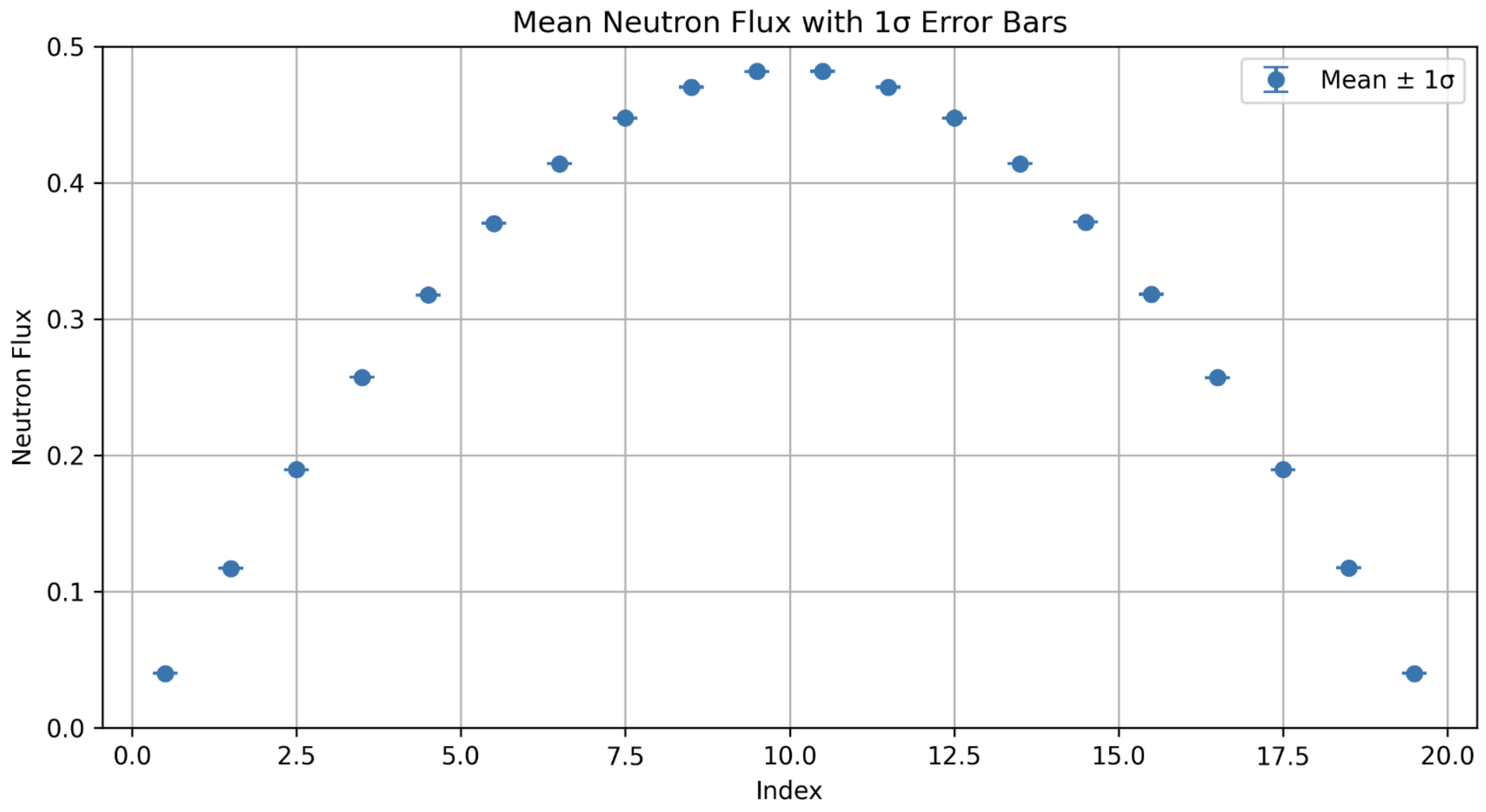
```
Boundary condition (0 = vacuum, 1 = reflective): 0
Number of neutrons: 100000
Total number of cycles: 300
Number of inactive cycles: 50
Number of active cycles: 250
k_a=0.905612, k_sa=0.820138, std=0.000128
k1=0.879588, k2=0.533992, k2/k1=0.607093
```

k_average, k_squared_average, k_sample_standard deviation

k1, k2, k2/k1

- $k_{\text{average}} = 0.906512 \pm 0.000128$ (68.27% Cl)
- k1 is similar to k_{average}
- k1 is not within the range of k_{average}

Result – (1) Vacuum Boundary Condition



Updated Algorithm for Re-entering Neutrons (Third Version)

- For each bin, negative weight neutrons are canceled by counting the total number of neutrons and the number of negative neutrons, then adjusting the neutron count accordingly.
- Example: If bin 1 contains 10 neutrons, and 2 have negative weight, the final count in bin 1 becomes 6 neutrons.
- Since a large number of neutrons are simulated, each bin always contains more positive weight neutrons than negative weight neutrons.
- All neutron within each bin are **redistributed uniformly throughout the bin.**
- **Disclaimer:** In this code, fission neutron weights were initially set to ± 1.0 instead of $\pm k$. This was later corrected in other codes upon realizing that the weights should be $\pm k$.

Updated Algorithm for Re-entering Neutrons (Third Version)

4. Simulate neutron diffusion.

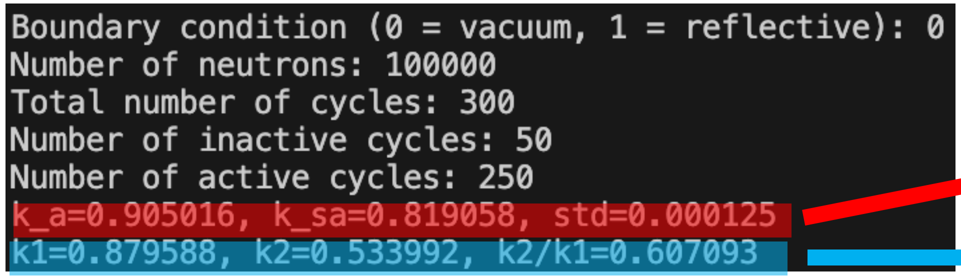
- For the total number of cycles:
  - While the parent fission bank is not empty:
    - Pop a neutron from the parent fission bank.
    - While the neutron is alive:
      - Generate a random number and compute the distance for movement.
      - Compute the new position and parent weight, considering boundary conditions.
      - Perform fission; if fission neutrons are produced, add them to the child fission bank.
      - Tally k value and neutron flux.
      - Forcefully scatter the neutron and adjust its weight.
      - Apply Russian roulette: if the neutron weight falls below a threshold, either kill it or reset its weight to 1.0.
    - **If vacuum boundary condition:**
      - **Count negative weight neutrons in each bin and adjust the total neutron count.**
      - **Redistribute the neutrons uniformly throughout the bin.**
    - Rescale the child fission bank to ensure total weight conservation.
    - Update the child fission bank to become the new parent fission bank.

Result – (1) Vacuum Boundary Condition

- Input values
  - Boundary condition: 0
  - Number of neutrons: 100000
  - Number of inactive cycles: 50
  - Number of active cycles: 250
- Variables
  - **width = 20.0**
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06
  - num_bins = 20

Result – (1) Vacuum Boundary Condition

- Terminal output



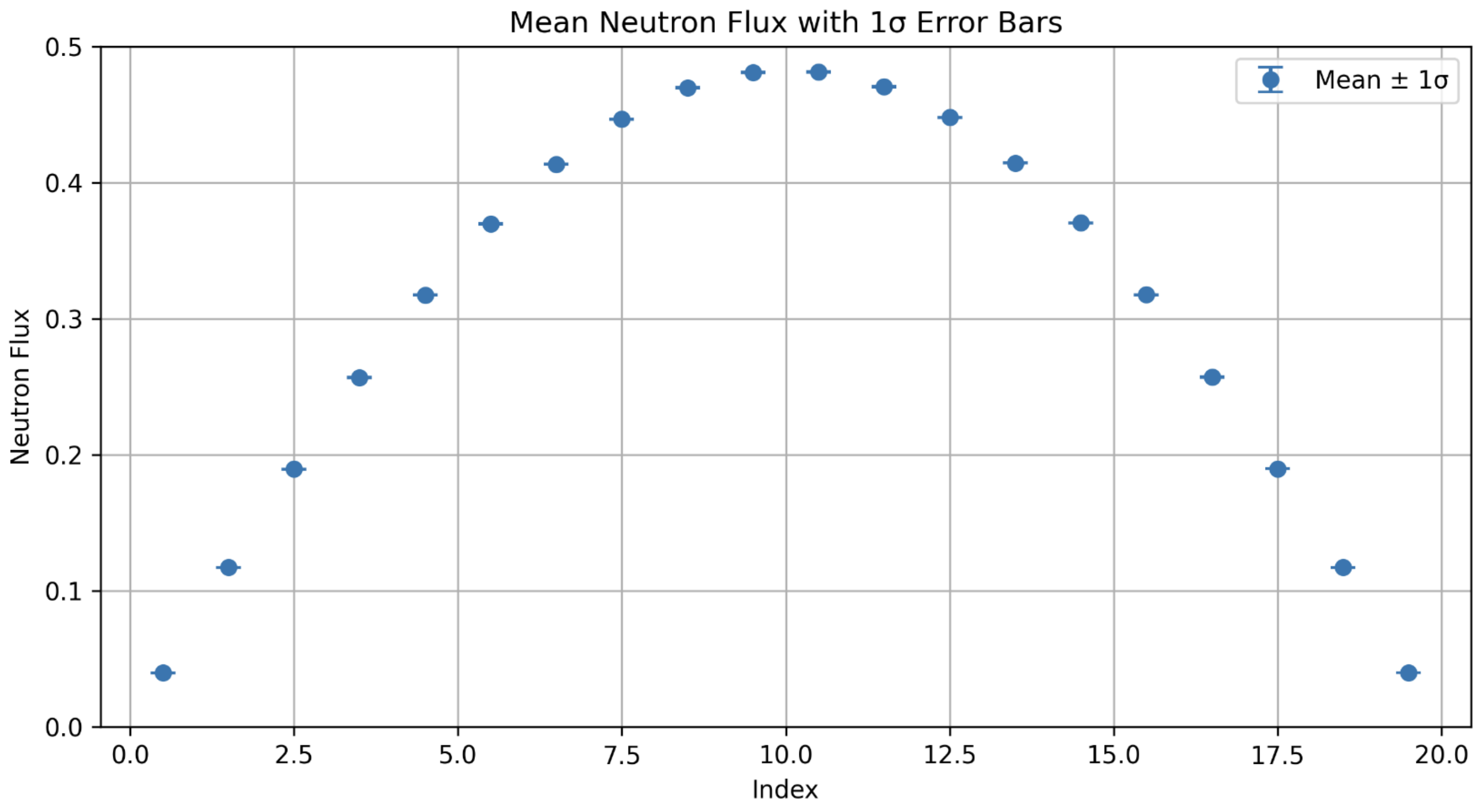
```
Boundary condition (0 = vacuum, 1 = reflective): 0
Number of neutrons: 100000
Total number of cycles: 300
Number of inactive cycles: 50
Number of active cycles: 250
k_a=0.905016, k_sa=0.819058, std=0.000125
k1=0.879588, k2=0.533992, k2/k1=0.607093
```

k_average, k_squared_average, k_sample_standard deviation

k1, k2, k2/k1

- $k_{\mathrm{average}} = 0.906516 \ \pm 0.000125$ (68.27% CI)
- k1 is similar to k_{average}
- k1 is not within the range of k_{average}

Result – (1) Vacuum Boundary Condition



Updated Algorithm for Re-entering Neutrons (Last Version)

- Negative weight neutrons are canceled by computing the total weight of the child fission bank in each bin and then reweighting each neutron accordingly.
- The child fission bank is then rescaled to ensure total weight conservation across each cycle.
- Example:
    - If bin 1 contains 10 neutrons with a total weight of 5.7, each neutron is reweighted to 5.7/10.
    - If bin 9 contains 7 neutrons with a total weight of 6.3, each neutron is reweighted to 6.3/7.
- If there are 70 neutrons in the child fission bank, and the initial total weight before the simulation started was 100, then each neutron is scaled by 100/70 to maintain weight conservation.

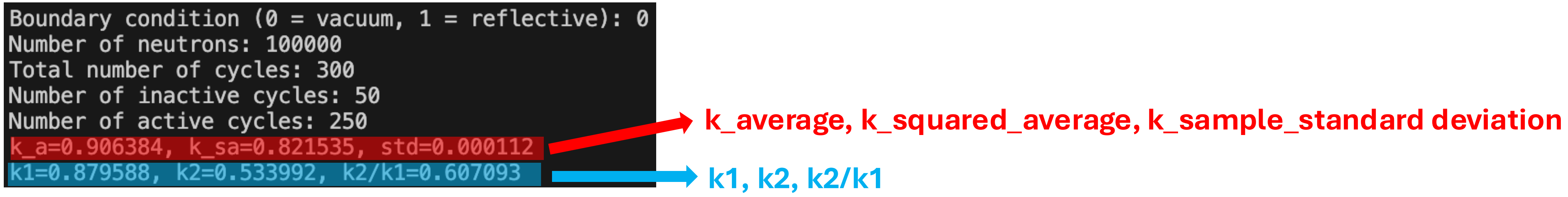
Updated Algorithm for Re-entering Neutrons (Last Version)

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 alive:
            - Generate a random number and compute the distance for movement.
            - Compute the new position and parent weight, considering boundary conditions.
            - Perform fission; if fission neutrons are produced, add them to the child fission bank.
            - Tally k value and neutron flux.
            - Forcefully scatter the neutron and adjust its weight.
            - Apply Russian roulette: if the neutron weight falls below a threshold, either kill it or reset its weight to 1.0.
        - **If vacuum boundary condition:**
            - **Compute the total weight of neutrons in each bin.**
            - **Reweight neutrons within each bin accordingly.**
        - Rescale the child fission bank to ensure total weight conservation.
        - Update the child fission bank to become the new parent fission bank.

Result – (1) Vacuum Boundary Condition

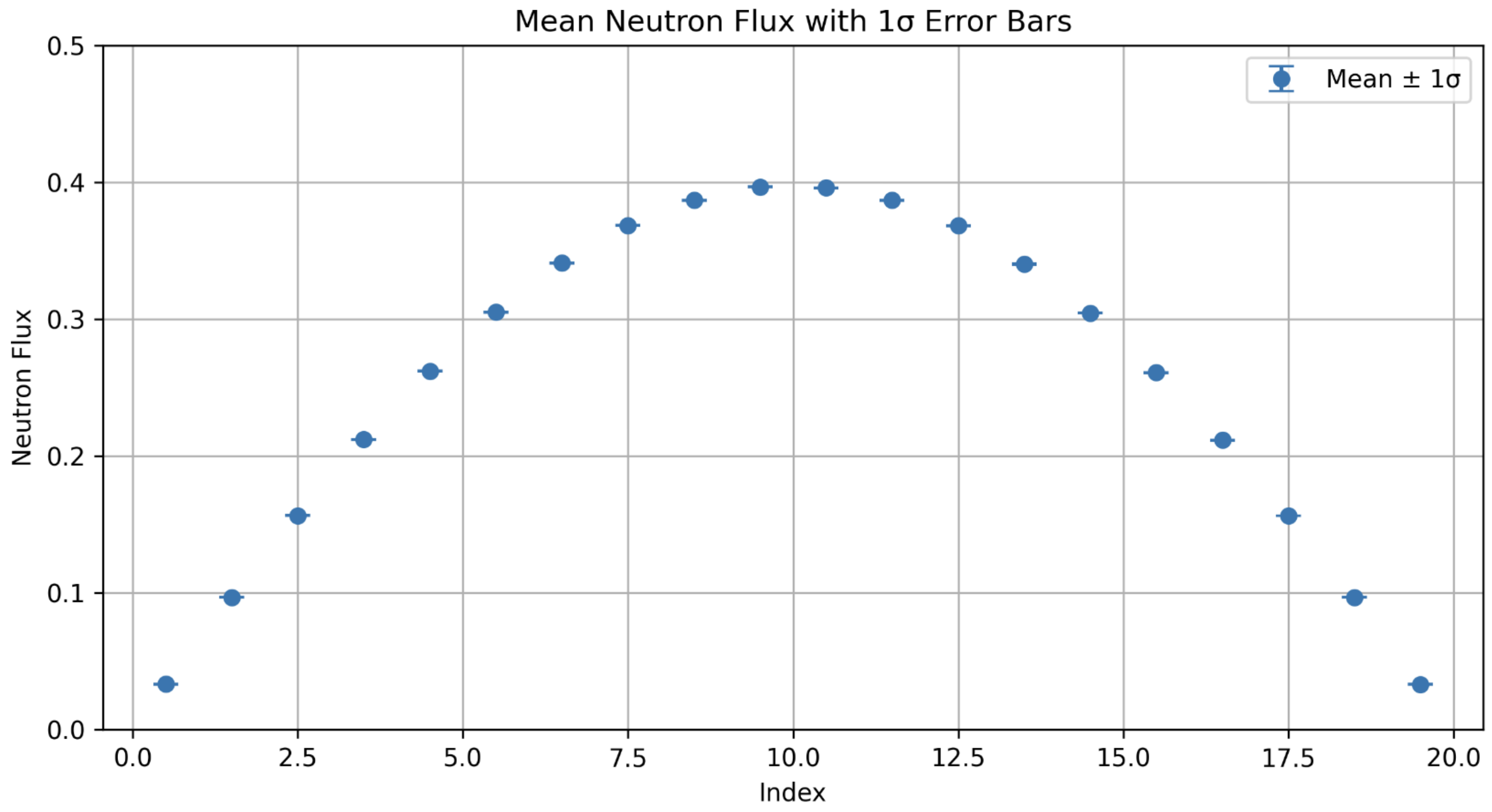
- Terminal output



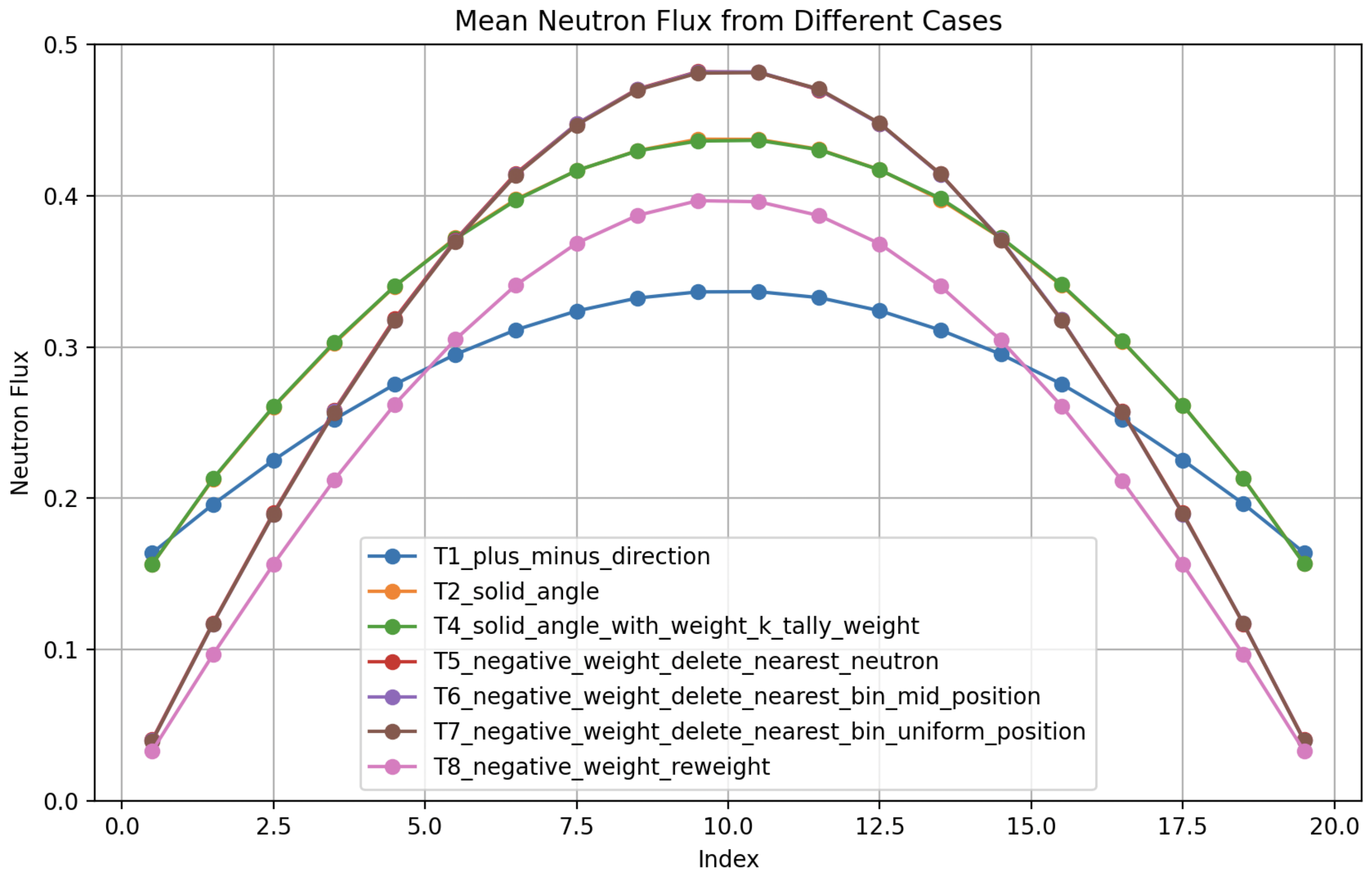
```
Boundary condition (0 = vacuum, 1 = reflective): 0
Number of neutrons: 100000
Total number of cycles: 300
Number of inactive cycles: 50
Number of active cycles: 250
k_a=0.906384, k_sa=0.821535, std=0.000112
k1=0.879588, k2=0.533992, k2/k1=0.607093
```

- $k_{\text{average}} = 0.906384 \pm 0.000112$ (68.27% CI)
- k1 is similar to k_{average}
- k1 is not within the range of k_{average}

Result – (1) Vacuum Boundary Condition



Overall Result



Conclusion

- **Use the latest version.**
- The latest version brings the neutron flux closest to zero at the boundaries.
- This version incorporates both approaches and correctly applies child neutron weights of $\pm k$.

Getting the Higher Modes

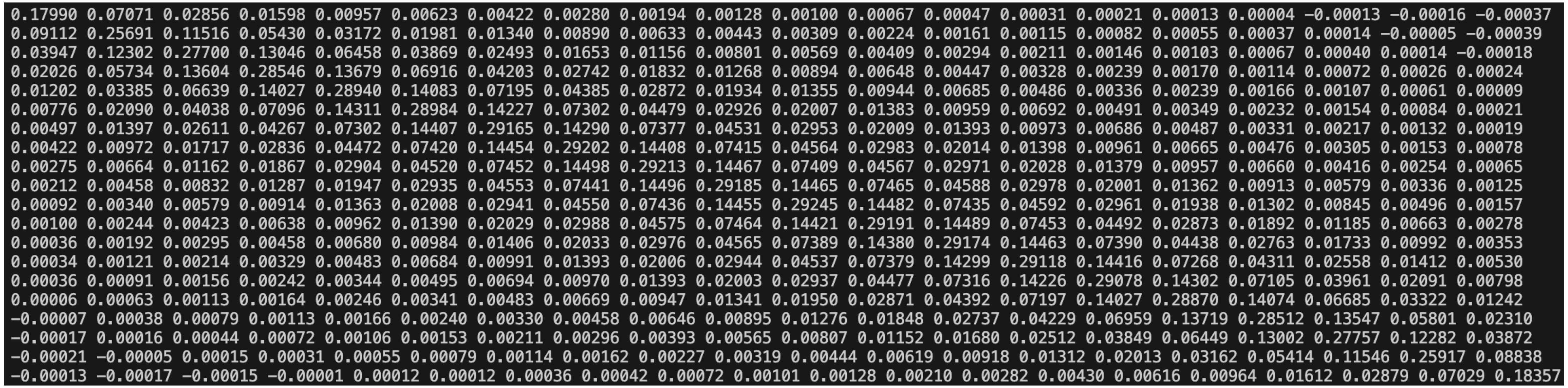
- To obain higher modes of neutron flux and k values, power method and delfation method is used.
- These methods require computing the fission matrix.
- The latest algorithm has been updated to compute the fission matrix.
- The power method and delfation method code is written in Python and utilizes the fission matrix data from the MC simulation.

Results – (1) Vacuum Boundary Condition

- Input values
  - Boundary condition: 0
  - Number of neutrons: 100000
  - Number of inactive cycles: 50
  - Number of active cycles: 250
- Variables
  - **width = 20.0**
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06
  - num_bins = 20

Results – (1) Vacuum Boundary Condition

- Fission matrix

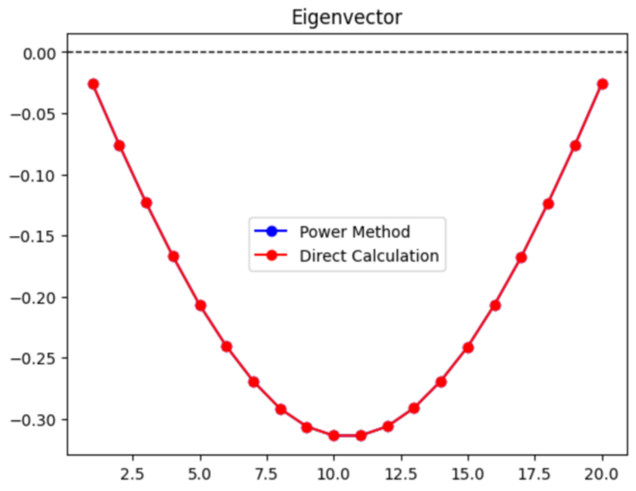


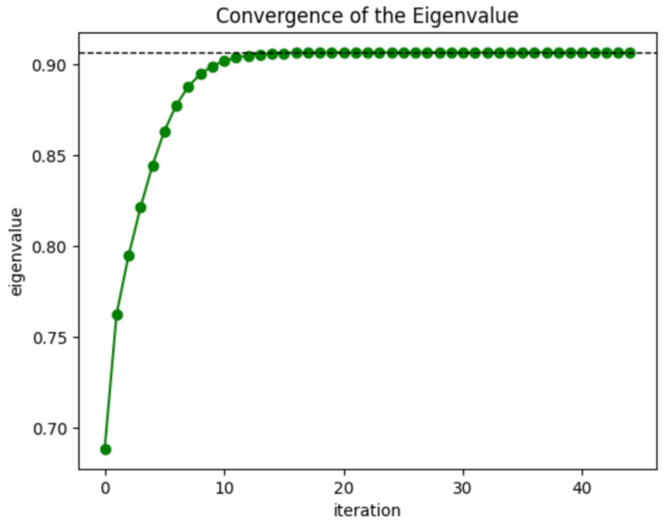
```
0.17990 0.07071 0.02856 0.01598 0.00957 0.00623 0.00422 0.00280 0.00194 0.00128 0.00100 0.00067 0.00047 0.00031 0.00021 0.00013 0.00004 -0.00013 -0.00016 -0.00037
0.09112 0.25691 0.11516 0.05430 0.03172 0.01981 0.01340 0.00890 0.00633 0.00443 0.00309 0.00224 0.00161 0.00115 0.00082 0.00055 0.00037 0.00014 -0.00005 -0.00039
0.03947 0.12302 0.27700 0.13046 0.06458 0.03869 0.02493 0.01653 0.01156 0.00801 0.00569 0.00409 0.00294 0.00211 0.00146 0.00103 0.00067 0.00040 0.00014 -0.00018
0.02026 0.05734 0.13604 0.28546 0.13679 0.06916 0.04203 0.02742 0.01832 0.01268 0.00894 0.00648 0.00447 0.00328 0.00239 0.00170 0.00114 0.00072 0.00026 0.00024
0.01202 0.03385 0.06639 0.14027 0.28940 0.14083 0.07195 0.04385 0.02872 0.01934 0.01355 0.00944 0.00685 0.00486 0.00336 0.00239 0.00166 0.00107 0.00061 0.00009
0.00776 0.02090 0.04038 0.07096 0.14311 0.28984 0.14227 0.07302 0.04479 0.02926 0.02007 0.01383 0.00959 0.00692 0.00491 0.00349 0.00232 0.00154 0.00084 0.00021
0.00497 0.01397 0.02611 0.04267 0.07302 0.14407 0.29165 0.14290 0.07377 0.04531 0.02953 0.02009 0.01393 0.00973 0.00686 0.00487 0.00331 0.00217 0.00132 0.00019
0.00422 0.00972 0.01717 0.02836 0.04472 0.07420 0.14454 0.29202 0.14408 0.07415 0.04564 0.02983 0.02014 0.01398 0.00961 0.00665 0.00476 0.00305 0.00153 0.00078
0.00275 0.00664 0.01162 0.01867 0.02904 0.04520 0.07452 0.14498 0.29213 0.14467 0.07409 0.04567 0.02971 0.02028 0.01379 0.00957 0.00660 0.00416 0.00254 0.00065
0.00212 0.00458 0.00832 0.01287 0.01947 0.02935 0.04553 0.07441 0.14496 0.29185 0.14465 0.07465 0.04588 0.02978 0.02001 0.01362 0.00913 0.00579 0.00336 0.00125
0.00092 0.00340 0.00579 0.00914 0.01363 0.02008 0.02941 0.04550 0.07436 0.14455 0.29245 0.14482 0.07435 0.04592 0.02961 0.01938 0.01302 0.00845 0.00496 0.00157
0.00100 0.00244 0.00423 0.00638 0.00962 0.01390 0.02029 0.02988 0.04575 0.07464 0.14421 0.29191 0.14489 0.07453 0.04492 0.02873 0.01892 0.01185 0.00663 0.00278
0.00036 0.00192 0.00295 0.00458 0.00680 0.00984 0.01406 0.02033 0.02976 0.04565 0.07389 0.14380 0.29174 0.14463 0.07390 0.04438 0.02763 0.01733 0.00992 0.00353
0.00034 0.00121 0.00214 0.00329 0.00483 0.00684 0.00991 0.01393 0.02006 0.02944 0.04537 0.07379 0.14299 0.29118 0.14416 0.07268 0.04311 0.02558 0.01412 0.00530
0.00036 0.00091 0.00156 0.00242 0.00344 0.00495 0.00694 0.00970 0.01393 0.02003 0.02937 0.04477 0.07316 0.14226 0.29078 0.14302 0.07105 0.03961 0.02091 0.00798
0.00006 0.00063 0.00113 0.00164 0.00246 0.00341 0.00483 0.00669 0.00947 0.01341 0.01950 0.02871 0.04392 0.07197 0.14027 0.28870 0.14074 0.06685 0.03322 0.01242
-0.00007 0.00038 0.00079 0.00113 0.00166 0.00240 0.00330 0.00458 0.00646 0.00895 0.01276 0.01848 0.02737 0.04229 0.06959 0.13719 0.28512 0.13547 0.05801 0.02310
-0.00017 0.00016 0.00044 0.00072 0.00106 0.00153 0.00211 0.00296 0.00393 0.00565 0.00807 0.01152 0.01680 0.02512 0.03849 0.06449 0.13002 0.27757 0.12282 0.03872
-0.00021 -0.00005 0.00015 0.00031 0.00055 0.00079 0.00114 0.00162 0.00227 0.00319 0.00444 0.00619 0.00918 0.01312 0.02013 0.03162 0.05414 0.11546 0.25917 0.08838
-0.00013 -0.00017 -0.00015 -0.00001 0.00012 0.00012 0.00036 0.00042 0.00072 0.00101 0.00128 0.00210 0.00282 0.00430 0.00616 0.00964 0.01612 0.02879 0.07029 0.18357
```

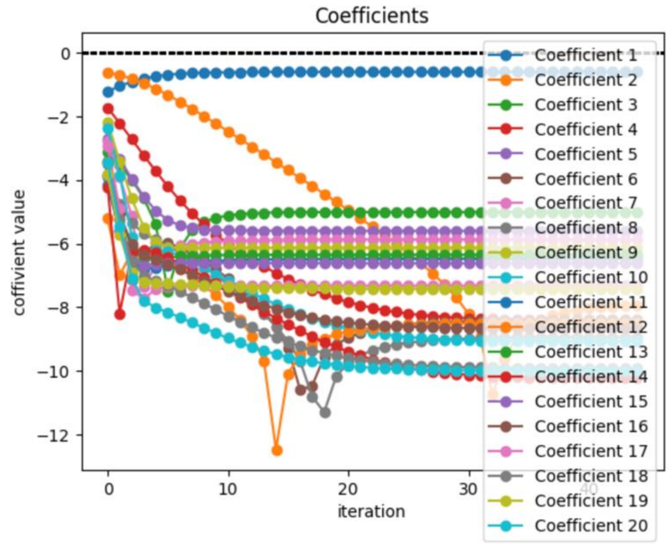
- The matrix is not perfectly symmetric, but a symmetric trend is observed.

Results – (1) Vacuum Boundary Condition

- The Power method was used to compute the dominant eigenvalue and its corresponding eigenvector.
- Direct calculation was performed using NumPy's linalg module.
- The coefficients represents the logarithm of the contributions of each eigenvector in the power method.

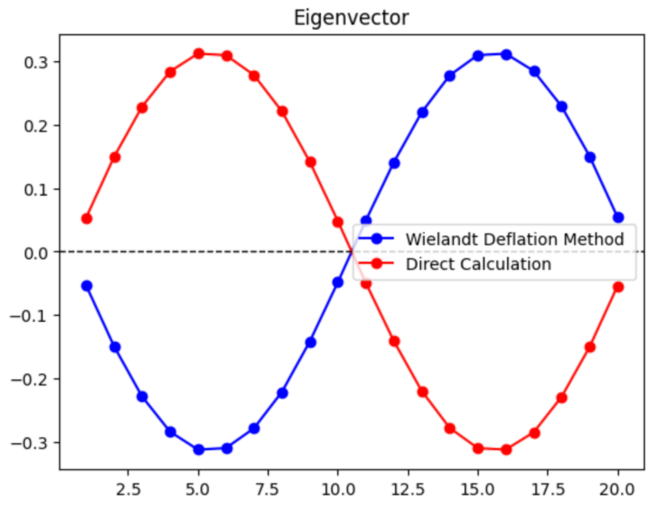


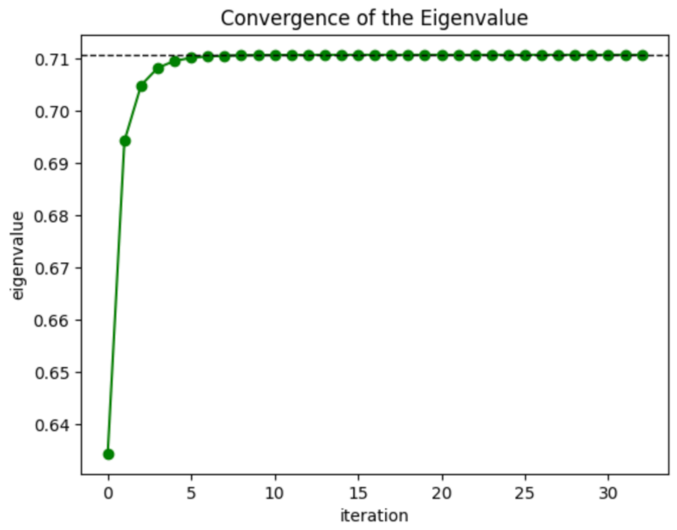


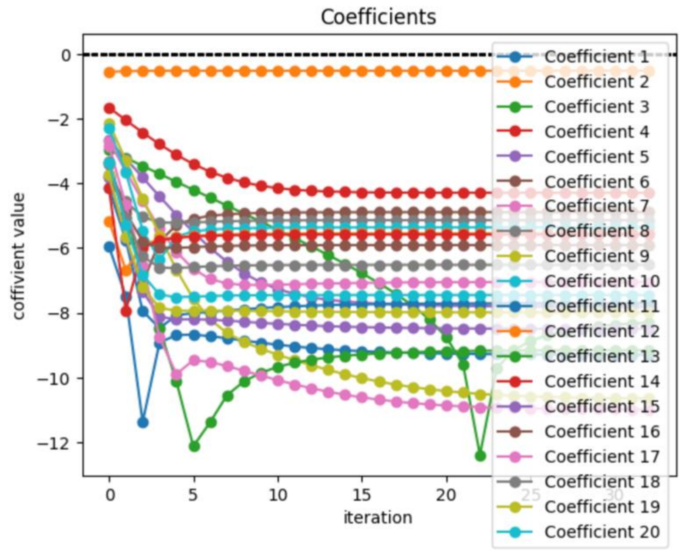


Results – (1) Vacuum Boundary Condition

- The Wielandt deflation method was used to compute the higher eigenvalues and their corresponding eigenvectors.
- The following figures show the second-largest eigenvalue and its corresponding eigenvector

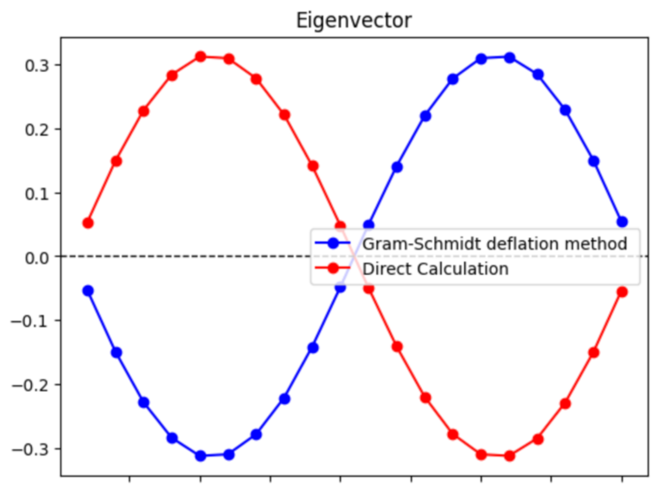


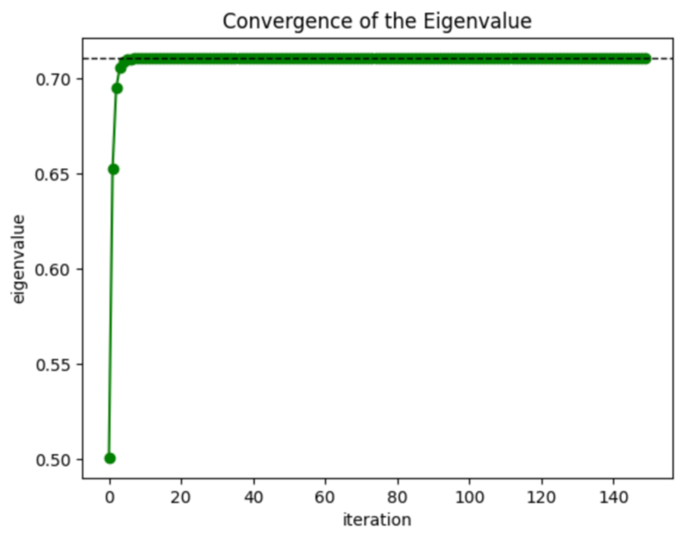


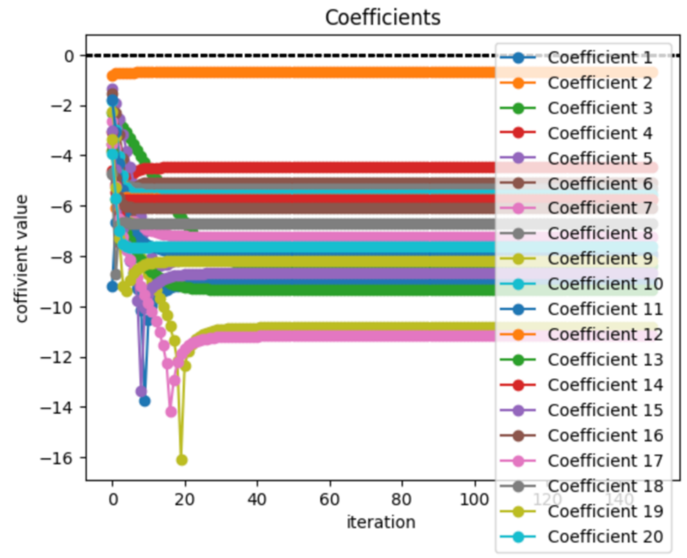


Results – (1) Vacuum Boundary Condition

- The Gram-Schmidt deflation method was used to compute the higher eigenvalus and their corresponding eigenvectors.
- The following figures show the second-largest eigenvalue and its corresponding eigenvector





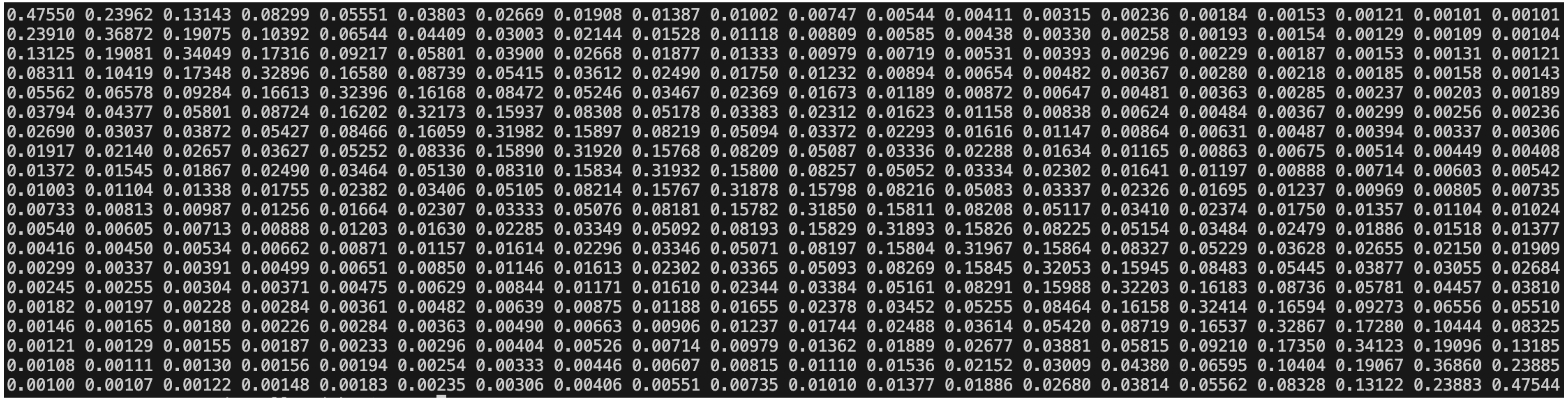


Results - (2) Reflective Boundary Condition

- Input values
  - Boundary condition: 1
  - Number of neutrons: 100000
  - Number of inactive cycles: 50
  - Number of active cycles: 250
- Variables
  - **width = 20.0**
  - sigma_s = 0.1
  - sigma_c = 0.07
  - sigma_f = 0.06
  - num_bins = 20

Results – (2) Reflective Boundary Condition

- Fission matrix

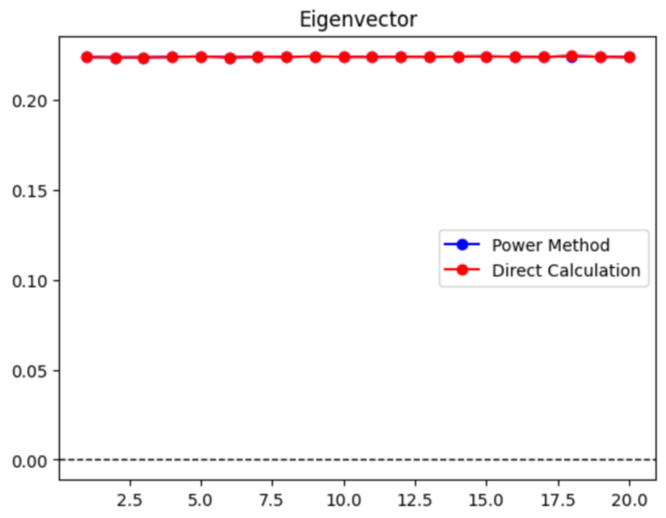


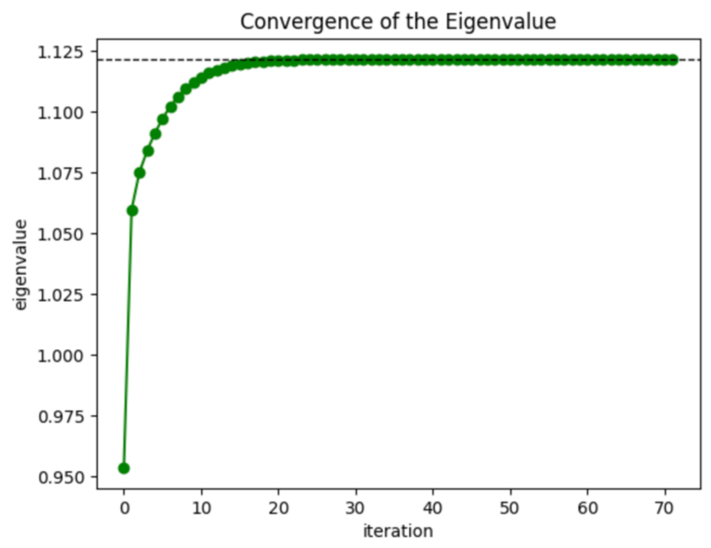
```
0.47550 0.23962 0.13143 0.08299 0.05551 0.03803 0.02669 0.01908 0.01387 0.01002 0.00747 0.00544 0.00411 0.00315 0.00236 0.00184 0.00153 0.00121 0.00101 0.00101
0.23910 0.36872 0.19075 0.10392 0.06544 0.04409 0.03003 0.02144 0.01528 0.01118 0.00809 0.00585 0.00438 0.00330 0.00258 0.00193 0.00154 0.00129 0.00109 0.00104
0.13125 0.19081 0.34049 0.17316 0.09217 0.05801 0.03900 0.02668 0.01877 0.01333 0.00979 0.00719 0.00531 0.00393 0.00296 0.00229 0.00187 0.00153 0.00131 0.00121
0.08311 0.10419 0.17348 0.32896 0.16580 0.08739 0.05415 0.03612 0.02490 0.01750 0.01232 0.00894 0.00654 0.00482 0.00367 0.00280 0.00218 0.00185 0.00158 0.00143
0.05562 0.06578 0.09284 0.16613 0.32396 0.16168 0.08472 0.05246 0.03467 0.02369 0.01673 0.01189 0.00872 0.00647 0.00481 0.00363 0.00285 0.00237 0.00203 0.00189
0.03794 0.04377 0.05801 0.08724 0.16202 0.32173 0.15937 0.08308 0.05178 0.03383 0.02312 0.01623 0.01158 0.00838 0.00624 0.00484 0.00367 0.00299 0.00256 0.00236
0.02690 0.03037 0.03872 0.05427 0.08466 0.16059 0.31982 0.15897 0.08219 0.05094 0.03372 0.02293 0.01616 0.01147 0.00864 0.00631 0.00487 0.00394 0.00337 0.00306
0.01917 0.02140 0.02657 0.03627 0.05252 0.08336 0.15890 0.31920 0.15768 0.08209 0.05087 0.03336 0.02288 0.01634 0.01165 0.00863 0.00675 0.00514 0.00449 0.00408
0.01372 0.01545 0.01867 0.02490 0.03464 0.05130 0.08310 0.15834 0.31932 0.15800 0.08257 0.05052 0.03334 0.02302 0.01641 0.01197 0.00888 0.00714 0.00603 0.00542
0.01003 0.01104 0.01338 0.01755 0.02382 0.03406 0.05105 0.08214 0.15767 0.31878 0.15798 0.08216 0.05083 0.03337 0.02326 0.01695 0.01237 0.00969 0.00805 0.00735
0.00733 0.00813 0.00987 0.01256 0.01664 0.02307 0.03333 0.05076 0.08181 0.15782 0.31850 0.15811 0.08208 0.05117 0.03410 0.02374 0.01750 0.01357 0.01104 0.01024
0.00540 0.00605 0.00713 0.00888 0.01203 0.01630 0.02285 0.03349 0.05092 0.08193 0.15829 0.31893 0.15826 0.08225 0.05154 0.03484 0.02479 0.01886 0.01518 0.01377
0.00416 0.00450 0.00534 0.00662 0.00871 0.01157 0.01614 0.02296 0.03346 0.05071 0.08197 0.15804 0.31967 0.15864 0.08327 0.05229 0.03628 0.02655 0.02150 0.01909
0.00299 0.00337 0.00391 0.00499 0.00651 0.00850 0.01146 0.01613 0.02302 0.03365 0.05093 0.08269 0.15845 0.32053 0.15945 0.08483 0.05445 0.03877 0.03055 0.02684
0.00245 0.00255 0.00304 0.00371 0.00475 0.00629 0.00844 0.01171 0.01610 0.02344 0.03384 0.05161 0.08291 0.15988 0.32203 0.16183 0.08736 0.05781 0.04457 0.03810
0.00182 0.00197 0.00228 0.00284 0.00361 0.00482 0.00639 0.00875 0.01188 0.01655 0.02378 0.03452 0.05255 0.08464 0.16158 0.32414 0.16594 0.09273 0.06556 0.05510
0.00146 0.00165 0.00180 0.00226 0.00284 0.00363 0.00490 0.00663 0.00906 0.01237 0.01744 0.02488 0.03614 0.05420 0.08719 0.16537 0.32867 0.17280 0.10444 0.08325
0.00121 0.00129 0.00155 0.00187 0.00233 0.00296 0.00404 0.00526 0.00714 0.00979 0.01362 0.01889 0.02677 0.03881 0.05815 0.09210 0.17350 0.34123 0.19096 0.13185
0.00108 0.00111 0.00130 0.00156 0.00194 0.00254 0.00333 0.00446 0.00607 0.00815 0.01110 0.01536 0.02152 0.03009 0.04380 0.06595 0.10404 0.19067 0.36860 0.23885
0.00100 0.00107 0.00122 0.00148 0.00183 0.00235 0.00306 0.00406 0.00551 0.00735 0.01010 0.01377 0.01886 0.02680 0.03814 0.05562 0.08328 0.13122 0.23883 0.47544
```

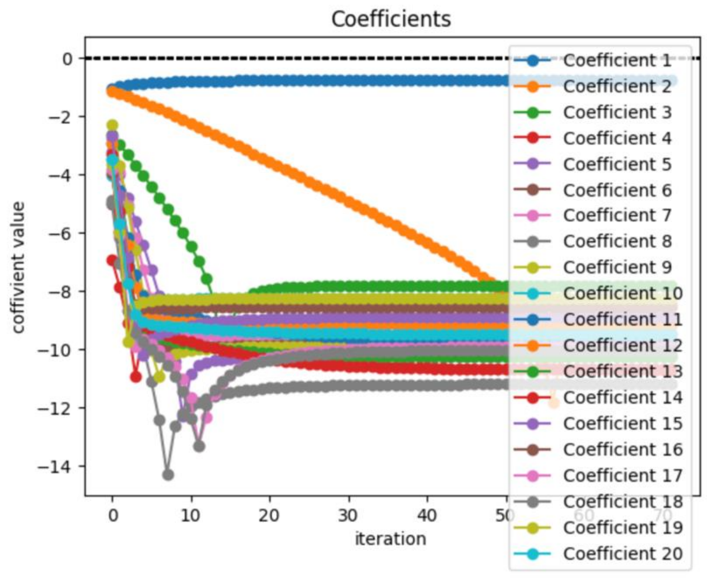
- The matrix is nearly symmetric.

Results – (2) Reflective Boundary Condition

- The Power method was used to compute the dominant eigenvalue and its corresponding eigenvector.

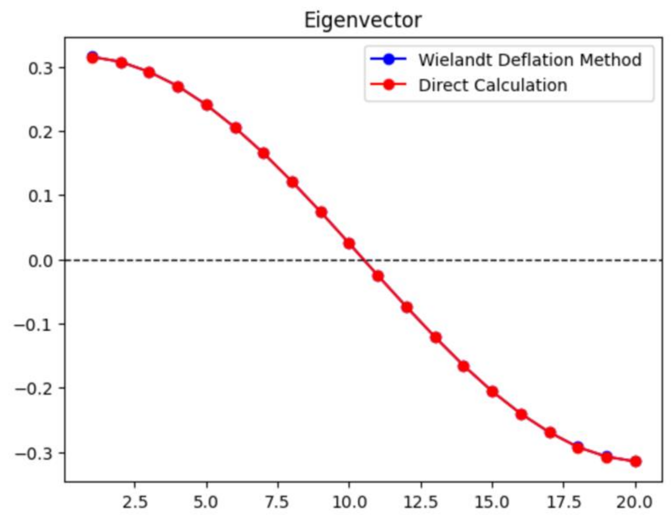


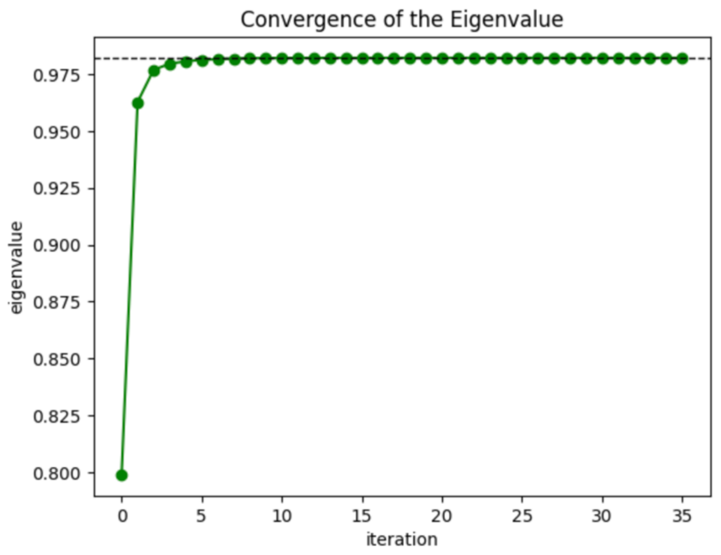


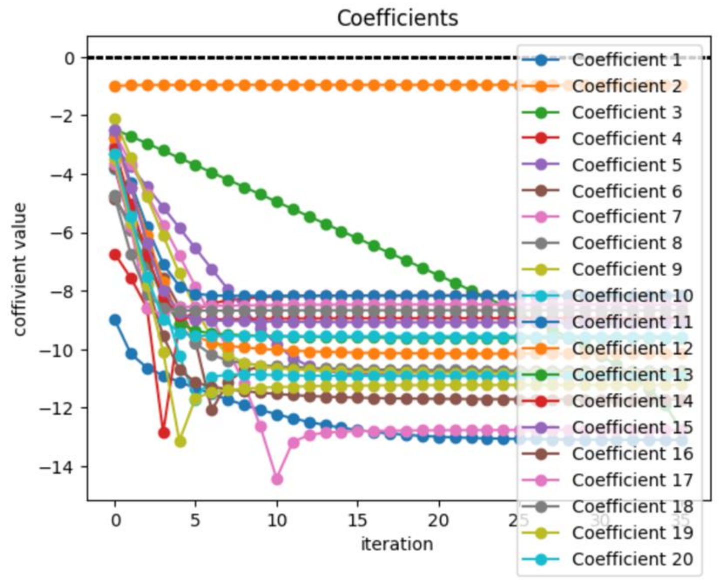


Results – (2) Reflective Boundary Condition

- The Wielandt deflation method was used to compute the higher eigenvalues and their corresponding eigenvectors.
- The following figures show the second-largest eigenvalue and its corresponding eigenvector

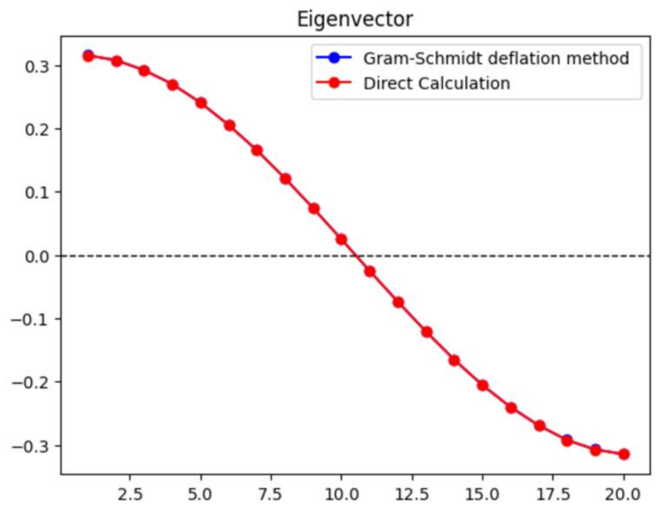


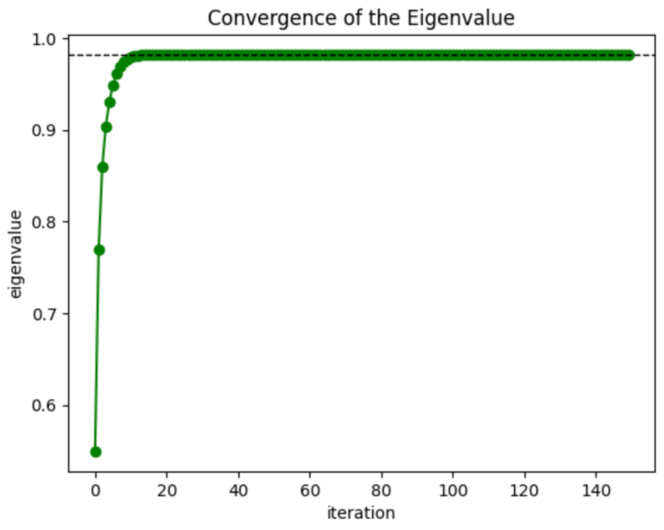


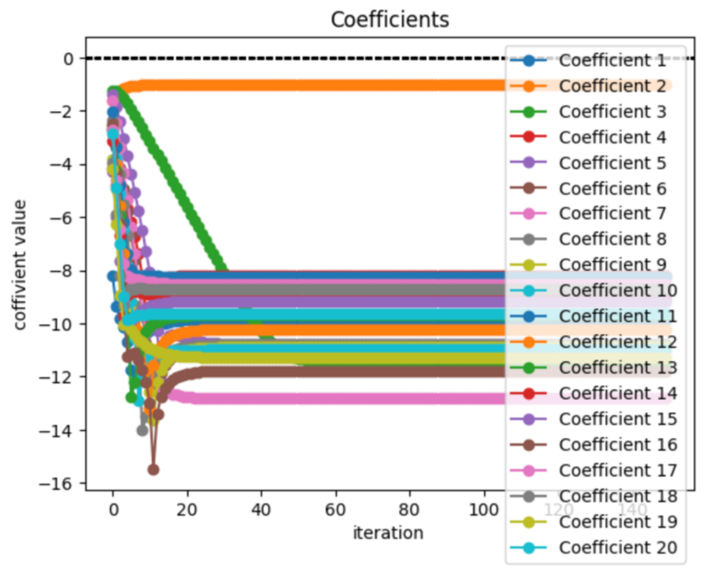


Results – (2) Reflective Boundary Condition

- The Gram-Schmidt deflation method was used to compute the higher eigenvalus and their corresponding eigenvectors.
- The following figures show the second-largest eigenvalue and its corresponding eigenvector







To-Do List

1) ~~Study Monte Carlo particle transport method~~ ✓ **Done**
2) ~~Implement code for fundamental mode~~ ✓ **Done**
3) ~~Implement code for higher modes~~ ✓ **Done**
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 (Not updated)

- 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