Monoenergetic 1D Diffusion Solver for Eigenvalue Problems

Introduction:

This paper presents a one-dimensional diffusion solver capable of handling neutron diffusion problems with both fixed and fission sources. The solver supports vacuum and reflective boundary conditions and can work with heterogeneous media. It prompts the user to input material properties, the number of mesh points, and the error tolerance. The code is designed to manage incorrect user inputs effectively. Parallel computing was initially implemented to enhance performance, but the implementation was not effective and has been removed from the final version. This issue will be discussed further in the code discussion section. The solver builds a tridiagonal matrix based on the user's input and applies the appropriate boundary conditions. It then solves the system of linear equations using a memory-optimized Gauss-Seidel method and plots the results.

Mathematics:

The Finite Volume Method (FVM) was used to discretize the diffusion equation.

$$-\frac{1}{3\Sigma_t}\frac{d^2\phi}{dx^2}+\Sigma_a\phi=\mathrm{Q}\ \text{or}\ \frac{1}{\mathrm{k}}\nu\Sigma_f\phi$$

Case 1: Fix Source

$$-\frac{\frac{1}{3\Sigma_t}\left(\frac{\phi_{i+1}-\phi_i}{\Delta x}-\frac{\phi_i-\phi_{i-1}}{\Delta x}\right)}{\Delta x}+\Sigma_a\phi_i=Q_i$$

Which simplifies to:

$$-\frac{1}{3\Sigma_t}\frac{\phi_{i+1}-2\phi_i+\phi_{i-1}}{\Delta x^2}+\Sigma_a\phi_i=Q_i$$

Matrix A coefficients:

$$\left(-\frac{1}{3\Sigma_t\Delta x^2}\right)\phi_{i+1}+\left(\frac{2}{3\Sigma_t\Delta x^2}+\Sigma_a\right)\phi_i+\left(-\frac{1}{3\Sigma_t\Delta x^2}\right)\phi_{i-1}$$

Case 2: Fission Source

$$-\frac{\frac{1}{3\Sigma_t}\left(\frac{\phi_{i+1}^{m+1}-\phi_i^{m+1}}{\Delta x}-\frac{\phi_i^{m+1}-\phi_{i-1}^{m+1}}{\Delta x}\right)}{\Delta x}+\Sigma_a\phi_i^{m+1}=\frac{1}{\mathrm{k}^m}\nu\Sigma_f\phi^m$$

Which simplifies to:

$$-\frac{1}{3\Sigma_t}\frac{\phi_{i+1}^{m+1}-2\phi_i^{m+1}+\phi_{i-1}^{m+1}}{\Delta x^2}+\Sigma_a\phi_i^{m+1}=\frac{1}{\mathrm{k}^m}\nu\Sigma_f\phi^m$$

Matrix A coefficients:

$$\left(-\frac{1}{3\Sigma_t\Delta x^2}\right)\phi_{i+1}^{m+1}+\left(\frac{2}{3\Sigma_t\Delta x^2}+\Sigma_a\right)\phi_i^{m+1}+\left(-\frac{1}{3\Sigma_t\Delta x^2}\right)\phi_{i-1}^{m+1}$$

Updating k:

$$k^{m+1}=k^m\left(\frac{\sum_{i=0}^{n-1}\nu\Sigma_f\phi_i^{m+1}}{\sum_{i=0}^{n-1}\nu\Sigma_f\phi_i^m}\right)\ for\ a\ uniform\ mesh$$

Boundary Conditions:

Vacuum boundary conditions:

$$\phi_{-D}=0,\phi_{t+D}=0$$

The slab thickness (t) is extended to t +/- D

$$a_{0,0}=\ a_{n,n}=1$$
$$a_{0,1:n}=\ a_{n,0:n-1}=0$$
$$b_0=b_n=0$$

Reflective boundary conditions on left side:

$$a_{0,0}=\frac{2D_1}{\Delta x^2}+\Sigma_{a,1}$$
$$a_{0,1}=-\frac{2D_1}{\Delta x^2}$$

Algorithms:

1. Start Program

- Define main functions and helper plot functions.

2. Input Handling

- Prompt the user to specify the problem type (f for fission or anything else for fixed source).
- Collect relevant parameters for the selected problem type (Slab thickness (t), number of mesh points, boundary conditions, material properties, convergence tolerance).
- Handle incorrect user input

3. Solve the Problem

- For fission problem (Eigenvalue Solver):
    i. Build the tridiagonal matrix using mesh points and material properties.
    ii. Apply boundary conditions:
        - Reflective or vacuum.
    iii. Adjust extrapolated distances for vacuum boundaries and refine the mesh.
    iv. Start with initial guesses for the flux and multiplication factor (k)
    v. While k is not converged, and maximum number of iterations is not reached:
        - Solve the tridiagonal matrix using a linear equations iterative solver such as Gauss Seidel.
        - Update the k
        - Check k's convergence
- For fixed source problem:
    i. Build the tridiagonal matrix using mesh points and material properties.
    ii. Apply boundary conditions:
        - Reflective or vacuum.
    iii. Adjust extrapolated distances for vacuum boundaries and refine the mesh.
    iv. Start with initial guesses for the flux
    v. Solve the tridiagonal matrix using a linear equations iterative solver such as Gauss Seidel.

4. Plot Results

- Plot Neutron Flux vs Position.
- Plot Residuals vs Iteration (log scale).

Code Development and Discussion:

The code solves both fixed source and fission source neutron diffusion problems using a finite volume discretization scheme and a memory optimized Gauss-Seidel iterative solver. It allows the user to input parameters such as slab thickness, number of mesh points, boundary conditions (vacuum or reflective), material properties (e.g., cross-sections and source strengths), and solver settings like initial flux, eigenvalue guess, tolerance, and maximum iterations. For eigenvalue problems, it iteratively computes the neutron multiplication factor k and the neutron flux distribution.

The results include the neutron flux profile and the convergence behavior (residuals vs iteration), which are plotted for visualization. The code is dynamic and supports heterogeneous

media, adaptive mesh refinement to account for vacuum boundary extrapolation. It provides a comprehensive tool for studying 1D neutron diffusion.

The development of the code progressed through four versions. Version control software (GitHub) was used throughout the development. The initial version addressed a simple fixed source problem in homogeneous media with vacuum boundaries. Users provided inputs such as slab thickness, total cross section, scattering to total cross section ratio, fixed source strength, number of mesh points, convergence tolerance, and maximum iterations. The code successfully converged for a test case with slab thickness 10, total cross section 1, , scattering to total cross section ratio 0.9, fixed source strength 1, 100 mesh points, and a convergence tolerance of 10^{-6}, in 2640 iterations. The residuals (errors) were tracked and plotted to investigate the solver's convergence behavior, as shown in Figures 1 and 2.

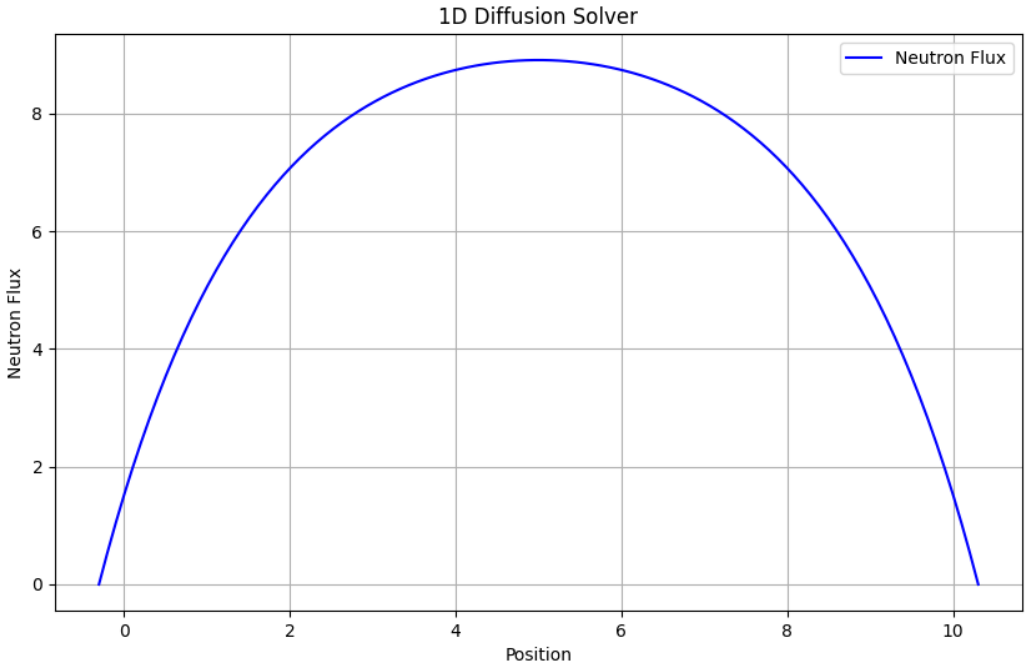

Figure 1: Fixed Source Diffusion with Vacuum Boundary Conditions

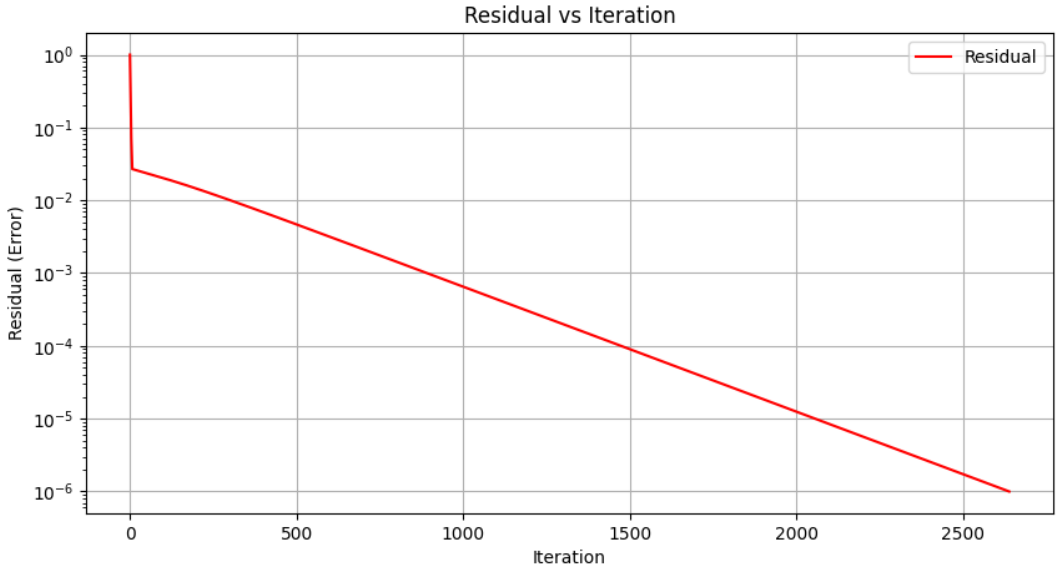

Figure 2: Residual vs Iterations of a Simple Fixed Source Diffusion Problem

Next, support for heterogeneous media and reflective boundary conditions was added. The user is prompted to select the left and right boundary conditions. The source strength and cross section parameters became medium specific, and the user input prompts were updated accordingly to account for these additional details. Figures 3 and 4 show simulations of a fuel pellet with a width of 10, surrounded by a moderator of width 5 on both sides, modeled in two different ways.

In the simulation shown in Figure 3, the slab thickness is set to 20, with three media and vacuum boundary conditions. The fuel is in the middle, represented as medium 2, with a thickness of 10 and a source strength of 1. The moderators are represented as media 1 and 3, each with a thickness of 5 and a source strength of 0. Scattering to total cross section ratios of 0.8 for the fuel and 0.99 for the moderators were used. The simulation converged in 3456 iterations with a runtime of 0.4317 seconds.

Due to the statistical and random nature of neutron diffusion, symmetry in the system can be assumed. The conditions on the left and right sides of the fuel pellet and moderators are identical, resulting in symmetry at $x = 10$, as observed in Figure 3. This symmetry allows the problem to be simplified by applying a reflective boundary condition on either the left or right side and modeling only half the domain. Figure 4 demonstrates this approach, which reduces the problem complexity and improves efficiency. Using the same parameters as in Figure 3 but changing the left boundary condition to reflective, the simulation converged in 3390 iterations with a reduced

runtime of 0.2164 seconds.

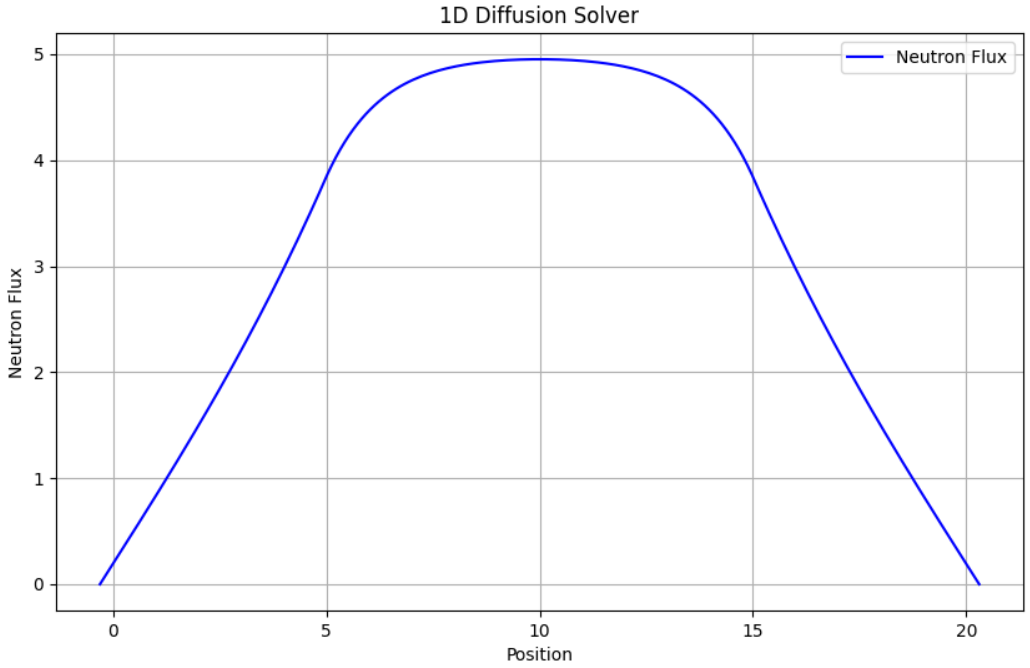

Figure 3: Fixed Source Diffusion with 3 Media and Vacuum Boundary Conditions

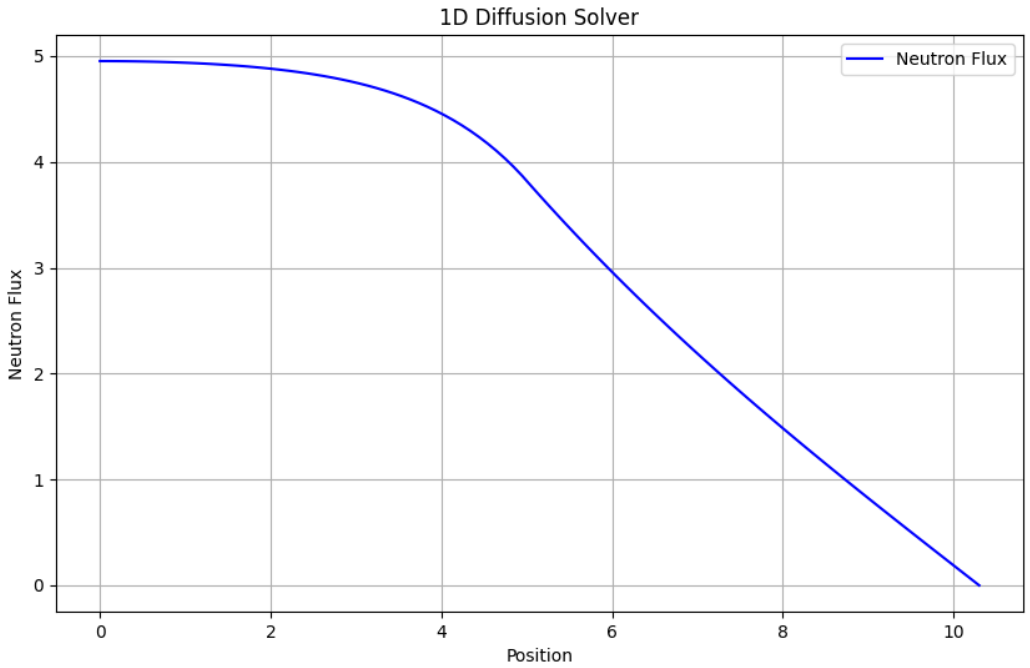

Figure 4: Fixed Source Diffusion with 2 Media and Reflective Boundary Condition on the Left

Support for fission sources was added, allowing users to select the type of problem they wish to solve (fixed source or fission source). For fission problems, the user is prompted to provide additional medium specific parameters, such as the fission to absorption cross section ratio and neutron yield, along with an initial guess for the multiplication factor (k). The code begins by using the initial guesses for flux and k to compute an update the flux. Once the flux converges, the code updates k using the new flux and checks for k convergence. This loop continues until both the flux and multiplication factor converge or the maximum number of iterations is reached.

Figures 5 and 6 demonstrate the same simulations as Figures 3 and 4 but with a fission source. The fuel has a fission to absorption cross section ratio of 0.5 and a neutron yield of 2.5, while the moderators have values of 0 for both.

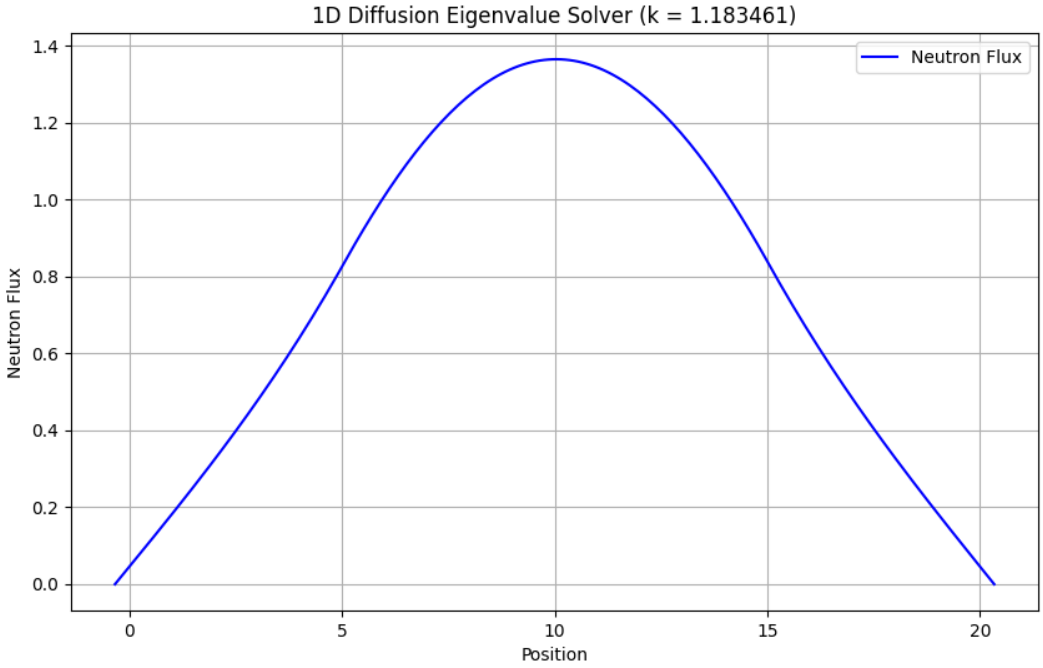

Figure 5: Fission Source Diffusion with 3 Media and Vacuum Boundary Conditions

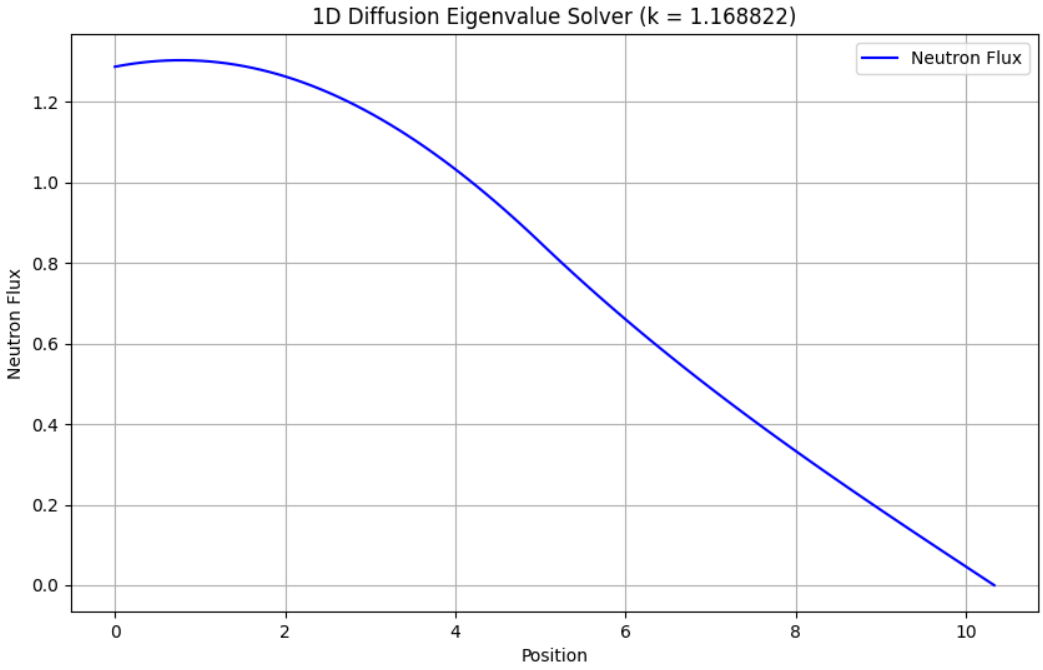

Figure 6: Fixed Source Diffusion with 2 Media and Reflective Boundary Condition on the Left

The final phase of development focused on optimization, formatting, and documentation. A dedicated get_user_input() function was implemented, utilizing helper functions to ensure that user inputs were logical and valid. Additionally, a write_to_file() function was introduced to save user inputs and outputs to a file for reproducibility. Sample runs were included in the code's GitHub repository for reference. ChatGPT was used to format, comment, and optimize the code further, including providing suggestions for improvement. Multiple iterative solvers were developed to evaluate performance and convergence. Initially, simple Jacobi and Gauss-Seidel solvers were implemented.

Recognizing the sparse nature of the finite volume discretization, which results in a tridiagonal matrix, an optimized Gauss Seidel solver was developed. Instead of constructing and solving a full N^2 matrix system A x = b, only the three non-zero diagonals were stored and passed to the solver. This optimization had no effect on runtime but significantly reduced memory usage, scaling it from N^2 to 3N.

A parallel computation solver was also explored, using Jacobi iterations with Python's Pool library. However, for the relatively simple simulated problems, the overhead of parallelization outweighed its benefits, resulting in slower runtimes compared to the standard Jacobi or Gauss-Seidel methods. Tests with up to 10,000 mesh points confirmed that the parallel implementation was inefficient for this use case. While parallel computation could be advantageous for more complex problems, it was ultimately excluded from the final version of the code due to its negative impact on performance in the current context.

Conclusion:

The development of the monoenergetic 1D neutron diffusion solver to solve the diffusion eigenvalue problem successfully addressed both fixed source and fission source problems using a finite volume discretization scheme and a memory-optimized Gauss-Seidel iterative solver. The code successfully implements all project requirements, a monoenergetic 1D diffusion solver to solve the diffusion eigenvalue problem in heterogeneous media with support for both vacuum and reflective boundary conditions. Its features include user-friendly input prompts for defining slab properties, material cross-sections, source strengths, and solver settings. Optional features such adaptive mesh refinement and parallel computation were also implemented. The outputs, including neutron flux profiles and convergence behavior, are visualized.

The development underwent four stages, developing a simple solver for homogenous media, adding support for heterogeneous media, reflective boundary conditions, and fission sources, then optimizing performance and memory usage. The introduction of a dedicated get_user_input() function ensured logical user inputs, while a write_to_file() function facilitated reproducibility by logging inputs and results. The code leveraged GitHub for version control, incorporating iterative improvements. Optimizations, such as storing only the three non-zero diagonals of the tridiagonal matrix, significantly reduced memory usage from N^2 to 3N.

The solver was also benchmarked for performance, with results demonstrating efficient convergence for various test cases. Parallel computation via Jacobi iterations using Python's Pool library was explored but ultimately excluded due to its inefficiency for simpler problems. However, this feature remains as an option for more complex simulations.

Overall, the final code represents a comprehensive starting framework for understanding and solving 1D neutron diffusion problems. It serves as a foundation for further enhancements and can be adapted to more complex neutron transport problems.