FeS_NT - A Finite Element Neutron Transport Solver

Andrew Johnson

April 29, 2018

1 Introduction

The goal of this work is to introduce and explain the code written for the MATH6641 project. The time-dependent one-dimensional neutron transport equation was solved using the discrete ordinates formulation **cite** using discontinuous-Galerkin in space, and explicit Euler for time. The project was written in Python and works best with 3.5[1] and is included with this report in `fesnt.py`. This report, and the code included, are covered with the GNU Public License, found in `LICENSE`.

1.1 Problem Statement

The discrete-ordinates formulation of the neutron transport equation chooses to solve for the angular neutron flux, $\psi(x, \mu, t)$ along a discrete set of N angles $\vec{\mu}$. A quadrature set of weights and angles is chosen to satisfy some physical constraints, briefly summarized here:

$$\mu_m = -\mu_{N-m+1} \tag{1a}$$

$$w_m = w_{N-m+1} \tag{1b}$$

$$\sum_m w_m = 2 \tag{1c}$$

$$\int_{-1}^{1} f(x,\mu)d\mu \approx \sum_m w_m f(x, \mu_m) \tag{1d}$$

The weights and angles are, for this work, chosen from a Guass-Legendre polynomial set. Select sets of weights and angles are listed in section 6.1

This project seeks to solve the following initial-boundary value problem:

$$\frac{1}{\nu}\frac{\partial \psi_m}{\partial t} + \mu_m \frac{\partial \psi_m}{\partial x} + \sigma_t(x)\psi(x,t) = \frac{1}{2}\left(\sigma_{s0} + \chi\nu\sigma_f\right)\sum_m w_m \psi_m(x,t), \text{ for } a < x < b, t > 0, \mu_m \in \vec{\mu} \tag{2}$$

subject to initial and boundary conditions

$$\psi_m(x,0) = \psi_{m,0}(x), \text{ for } a < x < b, \mu_m \in \vec{\mu} \tag{3a}$$

$$\psi_m(a,t) = \Gamma_a(\mu_m, t) \text{ for } \mu_m > 0, t > 0 \tag{3b}$$

$$\psi_m(b,t) = \Gamma_b(\mu_m, t) \text{ for } \mu_m < 0, t > 0 \tag{3c}$$

The report is laid out as follows. Section 2 derives the numerical scheme implemented for this project. Section 3 gives some insight into the actual implementation and usage needed to reproduce the results. Section 4 presents results from three test cases of increasing difficulty.

2 Derivation of the Method

The author set out to apply a finite-element scheme for the spatial domain, with an implicit Euler scheme in time. This section shall detail the mathematical foundation for the project, and present the numeric scheme that was ultimately implemented for this work.

[1] not tested on other versions

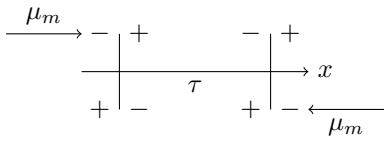

Figure 1: Notation used to denote upwind and downwind elements given flux angle μ

2.1 Finite-Element Method

Following the work of Reed and **other guy**, the author set out to apply a Discontinuous-Galerkin (DG) method for the spatial domain. This required subdividing the domain $a < x < b$ into non-overlapping elements τ. The finite-element space $\mathcal{V}$ was chosen to be piece-wise quadratic polynomials that are continuous within a single element.

2.1.1 Quadratic Flux

Within each element τ, the flux was represented as a quadratic function using the element boundaries and midpoint of the element. Specifically, the angular flux was represented using Lagrange interpolating polynomials, as

$$\psi_m(x) = \sum_{j=0}^{2} \psi_{m,j} \sum_{l=0}^{2} \alpha_{j,l} x^l = \sum_{j=0}^{2} \psi_{m,j} \prod_{l=0, l\neq j}^{2} \frac{x - x_l}{x_j - x_l} \tag{4}$$

This way, given three internal points of element τ, the Lagrange coefficients $\alpha_{j,l}$ could be determined, meaning that amplitudes $\psi_{m,j}$ remained to be solved. The benefit of using Lagrange polynomials is that, at a specific lattice site x_l, the angular flux evaluates to the specific polynomial amplitude, i.e.

$$\psi_m(x_j) = \psi_{m,j} \tag{5}$$

Using a quadratic representation of the flux within an element allows the use of Simpson's integration without introducing additional error, as such a method is exact for polynomials of order 2 or less. According to Simpson's rule, the spatial integral of a function can be expressed as

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\left(f(a) + 4f\left(\frac{b-a}{2}\right) + f(b)\right) \tag{6}$$

2.1.2 Upwind/Downwind Elements

The scheme implemented iterates through all angles in the quadrature set, and marches through elements in a manner akin to following the wind-direction. When mu is less than zero, this translates to moving left along the x axis, and the first mesh is taken to be the mesh with an edge at $x = b$. Such a scheme requires distinctions to be made between upwind and downwind sides, as will be utilized heavily in later sections. Figure 1 describes the upwind side of the edges of element τ as $-$ sides, and downwind sides as $+$.

3 Implementation

3.1 Using `fesnt.py`

4 Results

4.1 Case 1: Pure-Absorber

4.2 Case 2: Non-fissile

4.3 Case 3: Fissile Problem

5 Conclusion

6 Appendix

6.1 Quadrature Sets

The following tables contain the quadrature sets that were used in this work. For each set, only half of the values are presented, as symmetry rules from eq. (1) can be used to construct the full set.

Table 1: S-2 Weights and Angles

μ_m	w_m
0.5773502691	1.0

Table 2: S-4 Weights and Angles

μ_m	w_m
0.3399810435	0.6521451549
0.8611363115	0.3478548451

Table 3: S-8 Weights and Angles

μ_m	w_m
0.1834346424	0.3626837834
0.5255324099	0.3137066459
0.7966664774	0.2223810344
0.9602898564	0.1012285363