MATH 676 – Final Report

Logan Harbour
April 30, 2019

1 Introduction

The majority of my dissertation is to involve acceleration for method of characteristics (MOC) radiation transport. Many acceleration methods for discrete ordinates (S_N) radiation transport are similar to those utilized in MOC transport. Therefore, my overarching goal for this course was to investigate and implement common acceleration methods for S_N transport utilizing Deal.ii. In specific, the focus is on diffusion synthetic acceleration (DSA) of source iteration [1] in discrete ordinates radiation transport.

With source iteration (see Section 2.2), the scattering term is lagged. For example, one first solves for the solution of particles that have not scattered. These particles are then used to determine the source of first-scattered particles, and so on. The issue lies in the fact that when the ratio of particles scattering to being absorbed is close to unity, the source iteration process requires a significant number of iterations. DSA is able to attenuate the most poorly attenuated errors very well, therefore the diffusion problem is used as a corrector to the scalar flux after each source iteration.

From this, the primary goals of this work are as follows:

1. Develop a one-group, discrete ordinates radiation transport code in Deal.ii
2. Develop a one-group, diffusion radiation transport code in Deal.ii
3. Utilize the diffusion code to estimate the error after a source iteration as a preconditioner

The primary goals were completed approximately a month before the deliverables for the course were due. Therefore, the following additional implementations were made:

1. Parallel support using the Trilinos wrappers
2. Reflecting boundary conditions, which require additional iteration within a source iteration
3. Acceleration of reflecting boundary conditions within the DSA solver

2 Methods

Begin with the spatial domain $\mathcal{D} \in \mathbb{R}^2$ in which $\delta\mathcal{D}$ is on the boundary of $\mathcal{D}$. The set of propagation directions $\mathcal{S}$ is the unit disk.

The linear Boltzmann equation for one-group transport is

$$\boldsymbol{\Omega} \cdot \nabla\Psi(\boldsymbol{\Omega}, \mathbf{x}) + \sigma_t(\mathbf{x})\Psi(\boldsymbol{\Omega}, \mathbf{x}) - \sigma_s(\mathbf{x})\Phi(\mathbf{x}) = q(\mathbf{x}) \,, \qquad \forall(\boldsymbol{\Omega}, \mathbf{x}) \in \mathcal{S} \times \mathcal{D} \,, \tag{1a}$$

$$\Phi(\boldsymbol{\Omega}, \mathbf{x}) = \Phi^{\text{inc}}(\boldsymbol{\Omega}, \mathbf{x}) \,, \qquad \forall(\boldsymbol{\Omega}, \mathbf{x}) \in \mathcal{S} \times \delta\mathcal{D} \,,\ \boldsymbol{\Omega} \cdot \mathbf{n}(\mathbf{x}) < 0 \,, \tag{1b}$$

where Φ is the scalar flux, defined by

$$\Phi = \frac{1}{2\pi} \int_{\mathcal{S}} \Phi(\mathbf{\Omega}, \mathbf{x}) d\Omega \,.$$

Define $\mathbb{T}_h$ as the set of all active cells of the triangulation for $\mathcal{D}$ and $\mathbb{F}_h$ as the set of all active interior faces and define a discontinuous approximation space for the scalar flux based on the mesh $\mathbb{T}_h$ as

$$V_h \in \{v \in L^2(\mathcal{D}) \mid \forall K \in \mathbb{T}_h \,, v|_K \in P_K\} \,, \tag{2}$$

where the finite-dimensional space P_K is assumed to contain $\mathbb{P}_k$, the set of polynomials of degree at most k. Denote interior edges as $\mathcal{E}_h^i$ and boundary edges as $\mathcal{E}_h^b$.

2.1 S_N discretization

Introduce the S_N discretization, which replaces the angular flux with a discrete angular flux, as

$$\psi(\mathbf{x}) = [\psi_1(\mathbf{x}), \psi_2(\mathbf{x}), \dots \psi_{N_\Omega}(\mathbf{x})]^T \,. \tag{3}$$

We then introduce the quadrature rule $\{(\mathbf{\Omega}_d, \omega_d), d = 1, \dots, N_\Omega\}$ where $\sum_d \omega_d = 1$. With said quadrature rule, we have

$$\int_{\mathcal{S}} f(\mathbf{\Omega}, \mathbf{x}) d\Omega \approx \sum_{d=1}^{N_\Omega} w_d f(\mathbf{\Omega}_d, \mathbf{x}) \,.$$

This discretization allows us to write the system in Equation (1) as

$$\mathbf{\Omega}_d \cdot \nabla \psi_d(\mathbf{x}) + \sigma_t(\mathbf{x}) \psi_d(\mathbf{x}) - \sigma_s(\mathbf{x}) \phi(\mathbf{x}) = q(\mathbf{x}) \,, \qquad \forall \mathbf{x} \in \mathcal{D} \tag{4a}$$

$$\psi_d(\mathbf{x}) = \Psi_d^{\text{inc}}(\mathbf{x}) \,, \qquad \forall \mathbf{x} \in \delta\mathcal{D} \,,\ \mathbf{\Omega}_d \cdot \mathbf{n}(\mathbf{x}) < 0 \,, \tag{4b}$$

where the discrete scalar flux, ϕ, is

$$\phi(\mathbf{x}) = \sum_{d=1}^{N_\Omega} w_j \psi_j(\mathbf{x}) \,.$$

To obtain the weak form, multiply Equation (4b) by the test function $v_d \in V_h$ and integrate as

$$\int_{\mathbb{T}_h} \mathbf{\Omega}_d \cdot \nabla \psi_d v_d + \int_{\mathbb{T}_h} \sigma_t \psi_d v_d - \int_{\mathbb{T}_h} \sigma_s \phi v_d = \int_{\mathbb{T}_h} q v_d \,, \tag{5}$$

and integrate the first term by parts to obtain

$$\int_{\mathbb{T}_h} -\mathbf{\Omega}_d \cdot \nabla v_d \psi_d + \int_{\mathcal{E}_h^i} \psi_d (\mathbf{\Omega}_d \cdot \mathbf{n}) v_d + \int_{\mathbb{T}_h} \sigma_t \psi_d v_d - \int_{\mathbb{T}_h} \sigma_s \phi v_d = \int_{\mathbb{T}_h} q v_d \,, \tag{6}$$

where $\mathbf{n}$ is the outward normal. Note that the surface integration in Equation (6) is double-valued due to the discontinuous approximation. We introduce the upwind approximation

$$\psi_d \mathbf{\Omega}_d \cdot \mathbf{n} = \psi_d^+ \mathbf{\Omega}_d \cdot \mathbf{n} \,, \tag{7}$$

where ψ_d^+ is the upwind value of ψ_d, that is, the value from the side of the face in which $\mathbf{\Omega} \cdot \mathbf{n} \geq 0$. The weak form is then defined as

$$\int_{\mathbb{T}_h} -\mathbf{\Omega}_d \cdot \nabla v_d \psi_d + \int_{\mathcal{E}_h^i} \psi_d^+ (\mathbf{\Omega}_d \cdot \mathbf{n}) v_d + \int_{\mathbb{T}_h} \sigma_t \psi_d v_d - \int_{\mathbb{T}_h} \sigma_s \phi v_d = \int_{\mathbb{T}_h} q v_d \,. \tag{8}$$

2.2 Source iteration

To converge the scattering source, cast Equation (4a) with iterative index ℓ as

$$\mathbf{\Omega}_d \cdot \nabla \psi_d^{(\ell+1)} + \sigma_t \psi_d^{(\ell+1)} = \sigma_s \phi^{(\ell)} + q \,, \tag{9}$$

with $\psi_d^{(0)} = \phi^{(0)} = \vec{0}$. After solving each direction, d, for an iteration ℓ in Equation (9), update the scalar flux with

$$\psi^{(\ell+1)} = \sum_{d=1}^{N_\Omega} w_d \psi_d^{(\ell+1)} \,. \tag{10}$$

The process is then to be repeated until $\left\|\phi^{(\ell+1)} - \phi^{(\ell)}\right\|_{L^2} < \tau$, where τ is some tolerance, typically 10^{-12}. $\psi^{(\ell+1)}$ is the particles that have scattered at most ℓ times. As $\sigma_s/\sigma_t \to 1$, particles scatter more and more before they are absorbed and the number of needed source iterations becomes significant.

2.3 Diffusion acceleration

Simple algebraic manipulations can show that the error in $\psi^{(\ell+1)}$ satisfies the same transport equation with a source equal to $R^{(\ell+1)} = \sigma_s(\psi^{(\ell+1)} - \psi^{(\ell)})$. Furthermore, a Fourier analysis shows that the transport equation with the diffusion approximation attenuates the errors most poorly attenuated by the transport sweep. Therefore, we will cast a diffusion problem with the previously defined source as a preconditioner to source iteration.

Consider then the approximated error in $\psi^{(\ell+1)}$, defined as $e^{(\ell+1)}$. The corresponding diffusion problem is

$$-\nabla \cdot D \nabla e^{(\ell+1)} + \sigma_a e^{(\ell+1)} = \sigma_s(\phi^{(\ell+1)} - \phi^{(\ell)}) \,, \tag{11}$$

where D is the diffusion coefficient, $D = 1/3\sigma_t$.

Cast Equation (11) in the same DG space with interior edges $\mathcal{E}_h^i$ and boundary edges $\mathcal{E}_h^b$ using a modified interior penalty method developed in [2] for the face terms to obtain

$$\begin{aligned}\int_{\mathbb{T}_h} (D\nabla e^{(\ell+1)} \cdot \nabla v + \sigma_a e^{(\ell+1)} v) + \int_{\mathcal{E}_h^i} \left(\{\!\{ D\delta_n e^{(\ell+1)} \}\!\} [\![v]\!] + \{\!\{ D\delta_n v \}\!\} [\![e^{(\ell+1)}]\!] + \kappa [\![e^{(\ell+1)}]\!] [\![v]\!] \right) \\ + \int_{\mathcal{E}_h^b} \left(\kappa e^{(\ell+1)} v - D v \delta_n e^{(\ell+1)} - D e^{(\ell+1)} \delta_n v \right) = \int_{\mathbb{T}_h} \sigma_s (\phi^{(\ell+1)} - \phi^{(\ell)}) v \,,\end{aligned} \tag{12}$$

where

$$\{\!\{ u \}\!\} \equiv \frac{u^+ + u^-}{2} \quad \text{and} \quad [\![u]\!] \equiv u^+ - u^- \,,$$

in which the penalty coefficient is

$$\kappa = \begin{cases} 2\left(\frac{D^+}{h_\perp^+} + \frac{D^-}{h_\perp^-}\right) & \text{for interior edges} \,, \\ 8\frac{D^-}{h_\perp^-} & \text{for boundary edges} \,, \end{cases}$$

and $h_\perp^\pm$ is a characteristic length of the cell in the direction orthogonal to the edge.

3 Code design and description

The bulk of the development resides in five classes, of which the `Problem` owns an instance of the other four. There are no public variables and the only access to private variables is through appropriately created getter

methods that ensure const-correctness when possible. All primary system (LHS, RHS, solution) storage is done using the Deal.ii wrappers for the Trilinos vector and sparse matrix. A summary of these five classes follows in the next subsections.

3.1 `Description` class

The `Description` class stores all of the materials and boundary conditions in the problem. It does not need access to any of the other classes as this information is initialized once at setup by the `Problem` and only read from `SNProblem` and `DSAProblem` during assembly. Its `setup_bcs()` and `setup_materials()` methods require access to a set of boundary and material ids, respectively, in order to confirm that total material and boundary condition coverage is provided.

The interface to materials and boundary conditions is provided to the other classes by the public methods `get_bc()` and `get_material()`, which return a reference to the custom `BC` and `Material` structs, respectively.

3.2 `Discretization` class

The `Discretization` class is storage for all that pertains to the triangulation and the degree of freedom handler. In specific, it generates the mesh (depending on user input), builds the degree of freedom handler, and distributes the degrees of freedom, builds the sparsity pattern, etc. It also builds the custom `AngularQuadrature` object, which contains the angular quadrature. The two classes that assemble and solve the problems, the `SNProblem` and `DSAProblem` have read-only access to some of the variables in the `Discretization`; namely the `DoFHandler`. Lastly, it also builds the listing of all boundary and material ids for use in verifying material and boundary condition coverage by the `Description`.

3.3 `Problem` class

As stated in the beginning of the section, the `Problem` stores instances of all of the classes that are discussed here. It stores the system matrix, system right hand side, solution vectors, and other data structures that are needed by both the `DSAProblem` and `SNProblem`. It stores the primary `solve()` method, which iterates and checks for convergence of the `DSAProblem` and `SNProblem` as needed. Lastly, it stores the `.vtu` output if requested via input.

3.4 `SNProblem` class

The `SNProblem` class assembles and solves the S_N problem. It does not interface with the `DSAProblem` at all, purposely chosen in the design. Its public method available to the `Problem` is `assemble_solve_update()`, which assembles, solves, and updates the solution for all of the directions in the problem.

The assembly and solve process for a specific direction are handled in the private methods `solve()` and `assemble()`. The assembly process is handled by the `MeshWorker::loop()`, which calls on the private methods `integrate_cell()`, `integrate_boundary()` and `integrate_face()` to integrate cells, boundaries, and interior faces, respectively. It is important to note that setting the `MeshWorker::LoopControl` `faces_to_ghost` parameter to `both` disables the need for communicating ghosted values of the system matrix. This is because the faces that are on ghosted cells are assembled on both sides of the processor boundary. The `solve()` method utilizes the Trilinos GMRES solver and AMG preconditioner.

The majority of the structure for the implementation of the `SNProblem` was provided by the Deal.ii step-12. The parallel implementation was derived from the Deal.ii step-40.

3.5 `DSAProblem` class

The `DSAProblem` class assembles and solves the diffusion problem utilized for acceleration of the S_N source iterations. It does not interface with the `SNProblem` at all, purposely chosen in the design. Its public method available to the `Problem` is `assemble_solve_update()`, which assembles, solves, and updates the error estimate for the scalar flux error.

At initialization, `assemble_initial()` is called, which utilizes the `MeshWorker::loop()` to build the non-changing system matrix for the diffusion problem into the sparse matrix `dsa_matrix`. This matrix is then to be utilized at the beginning of every full assembly step in `assemble()` in order to reduce the need to re-build every iteration.

The assembly and solve process for a single iteration are handled in the private methods `solve()` and `assemble()`. The assembly process is handled by the `MeshWorker::loop()`, which calls on the private methods `integrate_cell()` and `integrate_boundary()` to integrate cells and boundaries, respectively. The `solve()` method utilizes the Trilinos CG solver (the interior penalty method is symmetric) and AMG preconditioner.

The majority of the structure for the implementation of the `SNProblem` was provided by the Deal.ii step-12 and step-39. The parallel implementation was derived from the Deal.ii step-40.

4 Numerical results

4.1 Constant solutions

Take a problem with a domain of $[0, 10]^2$, 64^2 elements, S_2 quadrature, a homogeneous volumetric source of $q = 1$, and $\sigma_t = 10$. The top and right boundary conditions are reflective and the bottom and left boundary vary. One would expect that the constant solution in the interior of the domain is q/σ_a. With this, we the following four problems:

- $\sigma_s = 9.9$ and incident isotropic flux boundary conditions on the bottom and left of q/σ_a
- $\sigma_s = 9.9$ and vacuum boundary conditions on the bottom and left
- Without scattering and incident isotropic flux boundary conditions on the bottom and left of q/σ_a
- Without scattering and vacuum boundary conditions on the bottom and left

The inputs for these examples can be found in `examples/constant_solution` and the results are plotted below in Figure 1.

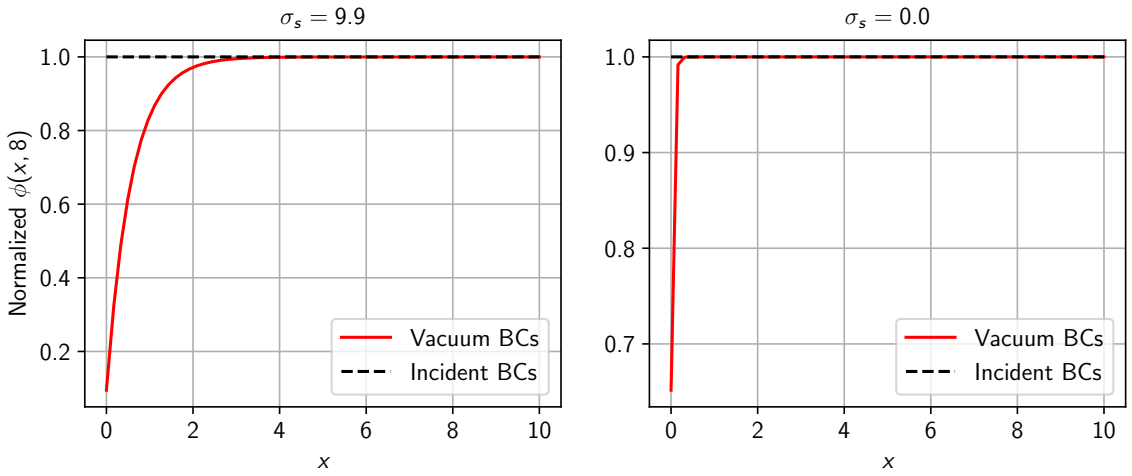

Figure 1: The scalar flux along $y = 8$ for the constant solution problems.

For the two cases with incident isotropic flux boundary conditions, as the flux condition is equal to the constant solution, we would expect the near the constant solution for the entire domain (as seen). For the case with scattering and vacuum boundary conditions, we would expect a decaying exponential to the constant solution due to the optical thickness of the domain. For the case without scattering and vaccuum boundary conditions, we would expect the constant solution near the vacuum boundary due to the optical thickness of the domain.

4.2 DSA

To verify the diffusion synthetic acceleration implementation, a problem was considered with increasing scattering ratio $c = \sigma_s/\sigma_t$. The spatial domain is $[0, 10]^2$, discretized into 16^2 elements with S_{10} angular quadrature. The cross sections are $\sigma_a + \sigma_s = \sigma_t = 100$ with a uniform volumetric source of 1. The input and run script are located in `examples/dsa_convergence`. The L_2 norm of the difference in scalar flux was plotted with varying c, as seen below in Figure 2.

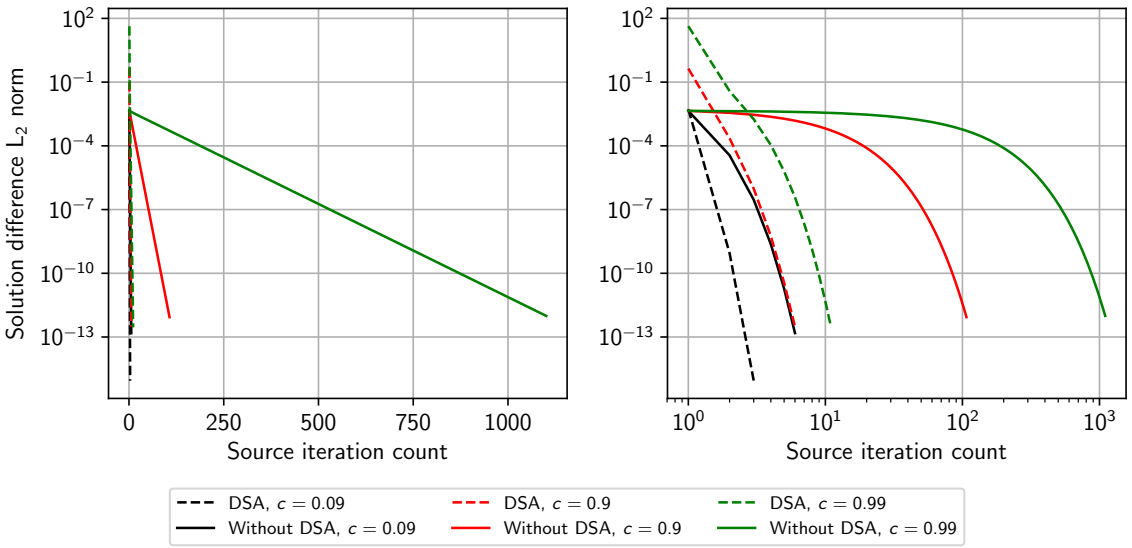

Figure 2: Source iteration norms with and without diffusion acceleration.

5 Conclusion

The primary goals have been met for this project, and have shown successful results. Additional goals were created after reaching the primary goals early, and were also successful. I was very pleased to develop in Deal.ii and the code created for this project has shown to be a very successful test bed for other acceleration techniques. This is primarily due to the ease of development using Deal.ii and the *plethora* of documentation available.

I have no doubt that both myself and my advisor will come to utilize this code again in the near future. A list of future works has already been developed for the coming months:

1. Multi-group support
2. Eigenvalue problem support
3. Acceleration of eigenvalue problems using DSA
4. Experimentation with other parallel solvers (pAIR!)

References

[1] Marvin L Adams and Edward W Larsen. Fast iterative methods for discrete-ordinates particle transport calculations. *Progress in nuclear energy*, 40(1):3–159, 2002.

[2] Jean C Ragusa and Yaqi Wang. A two-mesh adaptive mesh refinement technique for sn neutral-particle transport using a higher-order dgfem. *Journal of computational and applied mathematics*, 233(12):3178–3188, 2010.