An Efficient MOC Linear Algebra Solver

Geoffrey Gunow

December 14, 2016

1 Introduction

The method of characteristics (MOC) is a popular method for efficiently resolving the neutron flux distribution within a nuclear reactor. The neutron flux is necessary to compute reaction rates and hence heat production rates. The neutron flux – which is proportional to neutron density – can be represented by the neutron transport equation, a partial differential equation (PDE). MOC solves PDEs by laying characteristic paths or "tracks" across the phase space. The characteristic paths are chosen so that the complicated set of PDEs simplify to ordinary differential equations across the tracks. By laying many tracks across the problem, the problem can be resolved. For solving the neutron transport equation this means laying tracks across the geometry of the problem. Across these paths, the "angular neutron flux" (the neutron flux traveling across a direction) can be analytically calculated with a simple exponential relation. Each of these tracks contribute to the neutron flux distribution within the region they cross.

MOC is often solved using codes that do not resemble matrix equations. Instead, they resemble nested loop structures with the inner-most loop containing an exponential evaluation[1]. Some implementations have attempted to implement a matrix structure but have been shown to be very inefficient[2]. Because of this the reactor physics community largely views matrix implementations of MOC as inefficient – a direct challenge to the assertion that "everything in scientific computing comes down to linear algebra." In this project I attempt to provide an implementation that is cast in common linear algebra terms but also nears the efficiency of traditional MOC methods.

2 The Neutron Transport Equation

When simulating the behavior of a nuclear reactor, it is important to have an accurate estimate of the neutron fission reaction rates. These reaction rates produce heat rates necessary for thermal-hydraulic analysis as well as govern transmutation and material damage of all materials within the reactor. To determine the fission rates, the neutron behavior in the reactor is needed. Specifically, the angular neutron flux ψ is defined as

$$\psi(\vec{r}, E, \Omega, t) = vn(\vec{r}, E, \Omega, t) \tag{1}$$

where $n(\vec{r}, E, \Omega, t)$ is the neutron population at position $\vec{r}$, with energy E traveling in direction Ω at time t and v is the neutron velocity which is related to the energy E by the simple kinematic relationship $E = \frac{1}{2}m_n v^2$ (for energies relevant in reactor applications) where m_n is the rest mass of a neutron. The reaction rate $R_x(t, E)$ for reaction x with neutrons at energy E at a given time t within a region V can be calculated as

$$R_x(E, \Omega, t) = \int_V d\vec{r}\, \Sigma_x(\vec{r}, E, \Omega, t)\psi(\vec{r}, E, \Omega, t) \tag{2}$$

where $\Sigma_x(\vec{r}, E, t)$ is the "macroscopic cross-section" of type x at position $\vec{r}$ for neutrons of energy E traveling in direction Ω at time t. For the purpose of this report cross-sections can be thought of as simply material properties which describe the interaction probability of neutrons with the material. When multiplied by the scalar neutron flux they produce reaction rates. For instance if we are concerned with the *total* fission reaction rate R_F within a region V of a reactor at time t (which is necessary for thermal-hydraulic analysis), we would simply integrate the product with the fission cross-section $\Sigma_f(\vec{r}, \Omega E, t)$ over all possible neutron energies and directions as

$$R_F(t) = \int_0^\infty dE \int_V d\vec{r} \int_{4\pi} d\Omega\, \Sigma_f(\vec{r}, E, \Omega, t)\psi(\vec{r}, E, \Omega, t). \tag{3}$$

Therefore, **in order to calculate any neutron reaction rate we simply need to know the neutron angular flux distribution** $\psi(\vec{r}, \Omega E, t)$. For computational reasons and convenience, these equations are *almost always* simplified. Specifically, the scalar flux $\phi(\vec{r}, E, t)$ is defined as the angular neutron flux ψ integrated across all directions as:

$$\phi(\vec{r}, E, t) = \int_{4\pi} d\Omega \, \psi(\vec{r}, E, \Omega, t). \tag{4}$$

Then the macroscopic cross-sections are *approximated* as being independent of neutron direction. This allows for calculation of neutron reaction rates only using the scalar flux. In addition, a multi-group approximation for the energy variable E is often made where it is discretized over a number of groups (often 100 for this purpose). Within each energy group, the behavior is approximated as being constant. This leads to neutron angular and scalar fluxes (ψ and ϕ) being defined for each energy group. To determine the neutron angular and scalar flux distributions, a simple balance equation if formed using all relevant gains and losses of neutrons within a volume, as described in Table 1 with variable definitions defined in Table 2. In this construction, all neutron interactions are seen as losses but re-added as a source if the neutron ends up in the same phase space (energy group, region, etc.).

Table 1: Relevant gains and losses of neutrons in neutron transport calculations

Gain Mechanism	Functional Form	Loss Mechanism	Functional Form
Fission	$\frac{\chi_g(\vec{r},t)}{4\pi} \sum_{g'=1}^{G} \nu\Sigma_{f,g'}(\vec{r},t)\phi_{g'}(\vec{r},t)$	All interactions	$\Sigma_{t,g}(\vec{r},t)\psi_g(\vec{r},\Omega,t)$
In-scattering	$\frac{1}{4\pi} \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r},t)\phi_{g'}(\vec{r},t)$	Leakage	$\Omega \cdot \nabla\psi_g(\vec{r},\Omega,t)$

Table 2: Variables introduced in Table 1

Variable	Definition
G	The number of energy groups modeled in the system
$\Sigma_{f,g}$	Fission cross-section for energy group g
ν	The average number of neutrons produced per fission interaction
χ_g	The probability of a fission neutron being born in group g
$\Sigma_{t,g}$	Total cross-section for energy group g
$\Sigma_s^{g'\to g}$	Scattering cross-section from energy group g' to g
ϕ_g	The neutron scalar flux distribution for group g
ψ_g	The neutron angular flux distribution for group g

For this project, the focus will be on steady state solutions. Under steady-state conditions, we would like the losses to balance the gains. However, this might not be possible for every region within the reactor. Therefore, a factor k_{eff} is introduced which forces balance in the steady-state multi-group neutron transport equation:

$$\Sigma_{t,g}(\vec{r})\psi_g(\vec{r},\Omega) + \Omega \cdot \nabla\psi_g(\vec{r},\Omega) = \frac{1}{4\pi}\left(\frac{\chi_g(\vec{r})}{k_{\mathit{eff}}} \sum_{g'=1}^{G} \nu\Sigma_{f,g'}(\vec{r})\phi_{g'}(\vec{r}) + \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\phi_{g'}(\vec{r})\right). \tag{5}$$

This factor k_{eff} is in fact an *eigenvalue* of the system. This will become more apparent in later sections. The value of $k_{\mathit{eff}} = 1.0$ indicates a perfectly balanced or "critical" system. A value less than unity indicates a tendency for losses to overtake gains (sub-critical) and a value greater than unity indicates a tendency for gains to overtake losses (super-critical). It is important to note that k_{eff} is only applied to the fission term since fission is the only *neutron production term*. Scattering only changes a neutron's energy whereas fission creates neutrons.

3 The Method of Characteristics

The Method of Characteristics (MOC) is a popular method to solve the steady-state multi-group neutron transport equation. With this method, we define the neutron source $Q_g(\vec{r}, \Omega)$ for group g to be

$$Q_g(\vec{r}) = \frac{1}{4\pi}\left(\frac{\chi_g(\vec{r})}{k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}(\vec{r})\phi_{g'}(\vec{r}) + \sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\vec{r})\phi_{g'}(\vec{r})\right). \tag{6}$$

Along a "characteristic path" which in this case is simply a straight line through the geometry, the angular flux $\psi_g(\vec{r}, \Omega)$ follows a simple ordinary differential equation relationship that can be solved analytically. Then, a few more approximations are often made to simplify the computation:

1. The geometry is discretized into "flat source regions" (FSRs). Across each FSR the source Q_g is assumed to be constant as well as all material properties and cross-sections.
2. The directional variable Ω is discretized by laying many "tracks" across the Geometry at different angles. Each track is essentially a line through the geometry representing the angular flux along a specific direction.

With these approximations, the source in region i and group g is given by

$$Q_{i,g} = \frac{1}{4\pi}\left(\frac{\chi_{i,g}}{k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,i,g'}\phi_{i,g'} + \sum_{g'=1}^{G}\Sigma_{s,i}^{g'\to g}\phi_{i,g'}\right). \tag{7}$$

Notice that now the scalar fluxes and cross sections take discrete values for a given region i rather than having a continuous functional form. The angular fluxes traversing FSR i then follow the simple relationship

$$\psi_{k,g}(l) = \psi_{k,g}(0)e^{-\Sigma_{t,i,g}l} + \frac{Q_{i,g}}{\Sigma_{k,i,g}}\left(1 - e^{-\Sigma_{t,i,g}l}\right) \tag{8}$$

where now the angular fluxes $\psi_{k,g}(l)$ refer to the angular flux along track k which has some starting position $\vec{r}_k$ and direction Ω_k and l refers to the distance traversed from it's starting position $\vec{r}_k$ along direction Ω_k.

For ease of notation we define $\psi_{k,g,s}$ as being the outgoing flux on segment s from track k in group g that traverses region $i = I_k(s)$ where the segments are connected so that the outgoing angular flux from segment s is the incoming angular flux from segment $s+1$ along track k. The incoming angular flux on segment zero is defined by the boundary conditions. For this report, all boundaries are assumed to be vacuum such that this value is always zero.

Then using the relationship in Eq. 4, the scalar flux of the region can be estimated from each track traversing the region. In practice, this means a quadrature is formed giving weights to each track. The weighted summation of the scalar flux estimates leads to the expression:

$$\phi_{i,g} = \frac{4\pi\sum\limits_{k\in V_i} w_k l_{k,i}\sin\theta_k\overline{\psi_{k,i,g}}}{\sum\limits_{k\in V_i} w_k l_{k,i}\sin\theta_k} \tag{9}$$

where θ_k is the polar angle of track k, V_i is the volume defined by region i, $l_{i,k}$ is the distance traversed by track k through region i, and $\overline{\psi_{k,i,g}}$ is the average angular flux of track k in group g traversing region i can be computed as

$$\overline{\psi_{k,i,g}} = \frac{1}{l_{k,i}}\int_{d_{k,i}}^{d_{k,i}+l_{k,i}} dl\,\psi_{k,g}(l) = \frac{1}{l_{k,i}}\left[\frac{\psi_{k,g,s-1}}{\Sigma_{t,i,g}}\left(1 - e^{-\Sigma_{t,i,g}l_{k,i}}\right) + \frac{l_{k,i}Q_{i,g}}{\Sigma_{t,i,g}}\left(1 - \frac{1 - e^{-\Sigma_{t,i,g}l_{k,i}}}{\Sigma_{t,i,g}l_{k,i}}\right)\right] \tag{10}$$

where $d_{k,i}$ is the distance traversed along track k until it reaches region i. With this relationship, the computation of $\phi_{i,g}$ can be simplified to

$$\phi_{i,g} = \frac{4\pi}{\Sigma_{t,i,g}}\left(Q_{i,g} + \frac{1}{V_i}\sum_{k\in V_i} w_k\sin\theta_k\Delta\psi_{k,i,g}\right) \tag{11}$$

where $\Delta\psi_{k,i,g} = \psi_{k,g,s-1} - \psi_{k,g,s}$.

In practice, the first step in an MOC algorithm is to lay tracks across the geometry and assign them weights. Next, the Geometry is discretized into regions across which the flat source approximation is applied. Then, tracks are segmented. In this process the tracks are split into segments where each segment traverses exactly one flat source region. Then the "transport sweep" computation begins where an initial guess of the scalar neutron flux distribution in every region and energy group $\phi_{i,g}$ is used to determine neutron sources $Q_{i,g}$. With the approximated neutron sources, each track is traversed, calculating the current angular neutron flux $\psi_{k,g}$ along the track at each intersection – but not saving the values since the contribution of the track to the scalar flux can easily be computed only using the difference between incoming and outgoing angular neutron fluxes from the region[3]. At the very end the contribution of the neutron source to the scalar flux for each region is added (i.e. $\frac{4\pi Q_{i,g}}{\Sigma_{t,i,g}}$). A description of this algorithm is shown in Algorithm 1.

Algorithm 1 Transport Sweep Algorithm

$\phi_{i,g} \leftarrow 0 \quad \forall\, i, g \in \{I, G\}$ ▷ Initialize FSR scalar fluxes to zero
while $\phi_{i,g}\ \forall\ i$ not converged **do**
  for all k **do** ▷ Loop over tracks
    $\psi_{\text{temp}} \leftarrow 0$ ▷ Vacuum B.C.'s
    for all $s \in S(k)$ **do** ▷ Loop over segments
      for all $g \in G$ **do** ▷ Loop over energy groups
        $i \leftarrow I_k(s)$ ▷ Get FSR for this segment
        $\Delta\psi_{k,i,g} \leftarrow \left(\psi_{\text{temp}} - \frac{Q_{i,g}}{\Sigma_{t,i,g}}\right)\left(1 - e^{-\Sigma_{t,i,g} l_{k,i}}\right)$ ▷ Compute angular flux change
        $\phi_{i,g} \leftarrow \phi_{i,g} + w_k \sin\theta_k \Delta\psi_{k,i,g}$ ▷ Increment FSR scalar flux
        $\psi_{\text{temp}} \leftarrow \psi_{\text{temp}} - \Delta\psi_{k,g,s}$ ▷ Update track outgoing flux
      end for
    end for
  end for
  for all $i \in I$ **do** ▷ Loop over regions
    for all $g \in G$ **do** ▷ Loop over energy groups
      $\phi_{i,g} \leftarrow \frac{4\pi}{V_i}\phi_{i,g} + \frac{4\pi Q_{i,g}}{\Sigma_{t,i,g}}$ ▷ Add source contribution
    end for
  end for
  Update k_{eff} and FSR sources $Q_{i,g}\ \forall\ i$
end while

4 Forming Matrix Equations

In order to cast MOC as a linear algebra problem, the operations performed on the angular and scalar fluxes are cast as matrices. We consider the angular fluxes and scalar fluxes as vectors ϕ and Ψ respectively. The size of ϕ is M and the size of Ψ is N where $N >> M$ since there are many more angular fluxes in a problem than scalar fluxes.

First, we define the source vector Q with components $Q_{i,g}$ given in Eq. 7 as

$$Q = \frac{1}{k_{eff}}\hat{F}\phi + \hat{S}\phi \tag{12}$$

where

$$\hat{F}\phi = \frac{\chi_{i,g}}{4\pi}\sum_{g'=1}^{G}\nu\Sigma_{f,i,g'}\phi_{i,g'} \tag{13}$$

and

$$\hat{S}\phi = \frac{\chi_{i,g}}{4\pi}\sum_{g'=1}^{G}\Sigma_{s,i}^{g'\to g}\phi_{i,g'}. \tag{14}$$

Since $\hat{F}$ and $\hat{S}$ only act on *scalar* fluxes ϕ they are of size $M \times M$. Then we consider the relationship for angular flux variation along a track presented in Eq. 8 as

$$T\Psi = HQ \tag{15}$$

where $T\Psi$ relates the difference between outgoing angular flux $\psi_{k,g,s}$ and the attenuation of the incoming angular flux $\psi_{k,g,s-1}$ over region i as

$$T\Psi = \psi_{k,g,s} - \psi_{k,g,s-1} e^{-\Sigma_{t,i,g} l_{k,i}} \tag{16}$$

and HQ is the operation selecting the source of the traversed region i and determining its contribution to the angular flux as

$$HQ = \frac{Q_{i,g}}{\Sigma_{k,i,g}} \left(1 - e^{-\Sigma_{t,i,g} l_{k,i}}\right). \tag{17}$$

Note that H is of size $N \times M$ as it operates on scalar fluxes to produce a quantity used in computation with angular flux computation. For each row in H there is exactly one nonzero entry corresponding to the source of the traversed region. Matrix T is of size $N \times N$ but very sparse. Specifically T contains ones along the diagonal relating to the outgoing angular fluxes and an off-diagonal element $-e^{-\Sigma_{t,i,g} l_{k,i})}$ applied to the incoming angular fluxes.

Lastly, we need to define how the scalar fluxes are computed as a weighted summation of angular fluxes. In Eq. 9 a relationship was given in terms of both the angular fluxes and the average angular fluxes where the average angular fluxes are dependent on both the angular fluxes $\psi_{k,g,s}$ and sources $_{i,g}$. However, we would like to have a relationship that is only cast in terms of angular fluxes so that it is possible to calculated scalar fluxes from angular fluxes alone. To do this, Eq. 8 is used to eliminate $Q_{i,g}$ from Eq. 10 yielding the relationship

$$\overline{\psi_{k,g,s}} = \frac{\psi_{k,g,s-1} - \psi_{k,g,s}}{\Sigma_{t,i,g} l_{k,i}} + \frac{\psi_{k,g,s} - \psi_{k,g,s-1} e^{-\Sigma_{t,i,g} l_{k,i}}}{1 - e^{\Sigma_{t,i,g} l_{k,i}}} \tag{18}$$

Used in conjunction with Eq. 9 this yields

$$\phi_{i,g} = \frac{4\pi}{V_i} \sum_{k \in V_i} w_k l_{k,i} \sin\theta_k \left[\frac{\psi_{k,g,s-1} - \psi_{k,g,s}}{\Sigma_{t,i,g} l_{k,i}} + \frac{\psi_{k,g,s} - \psi_{k,g,s-1} e^{-\Sigma_{t,i,g} l_{k,i}}}{1 - e^{\Sigma_{t,i,g} l_{k,i}}} \right] \tag{19}$$

which can be written in matrix form as

$$\phi = W\Psi. \tag{20}$$

Note that W is size $M \times N$ since it operates on angular fluxes to produce scalar fluxes. Altogether, the eigenvalue problem can then be written as

$$T\Psi = \frac{1}{k} H\hat{F}W\Psi + H\hat{S}W\Psi. \tag{21}$$

This can be re-arranged as a generalize eigenvalue problem (of the form $Ax = \lambda Bx$) as

$$H\hat{F}W\Psi = k\left(T - H\hat{S}W\right)\Psi \tag{22}$$

which of course can be re-arranged as a standard eigenvalue problem by taking an inverse as

$$\left(T - H\hat{S}W\right)^{-1} H\hat{F}W\Psi = k\Psi. \tag{23}$$

5 Solving the Eigenvalue Problem

Now that the eigenvalue problem has been formed, it is possible to solve the problem with a variety of methods. Since taking the inverse of a large matrix is infeasible, it would not be wise to try to explicitly form the matrix $\left(T - H\hat{S}W\right)^{-1} H\hat{F}W$ for use in a standard eigenvalue solver. However, there are many eigenvalue solvers that can solve the generalized eigenvalue problem described in Eq. 22. Indeed, Julia contains an optimized eigenvalue solver `eigs` which is capable of solving generalized eigenvalue problems. Most standard eigenvalue solvers fundamentally rely on performing matrix-vector multiplies with the matrix of which we wish to find the

eigenvalues and eigenvectors. For a generalized eigenvalue solver, this matrix vector multiply is replaced with solving a linear system.

This can clearly be seen if we look at a simple method for computing the dominant eigenvalue and eigenvector, the power method. This method takes advantage of the eigenvector dominating the result of repeatedly multiplying by a matrix. This algorithms is shown in Algorithm 2. It is important to note that while a matrix inversion is shown in the algorithm, this is not explicitly computed in the algorithm. Rather, the linear system is solved whose solution is equivalent to the matrix-vector product shown in the algorithm.

Algorithm 2 Power Method Algorithm

Take a starting guess for Ψ
while Ψ not converged **do**
  $\Psi \leftarrow \left(T - H\hat{S}W\right)^{-1} H\hat{F}W\Psi$
  Normalize Ψ
end while

A caveat of the neutron transport application is that we are only concerned with the dominant eigenvalue and the *scalar fluxes* associated with the dominant eigenvector. This is because scalar fluxes are used to compute reaction rates. Therefore, while the eigenvalue solver may present a solution for Ψ, we are only concerned with the solution $\phi = W\Psi$. Since the W is size $M \times N$ and $N >> M$, the storage requirements of ϕ are much less than Ψ. Indeed if it is possible to only compute the components of Ψ on-the-fly when needed with respect to determining ϕ, there could be gains in computational efficiency.

This leads to the idea of source iteration. With this method the eigenvalue problem is computed more as a fixed-point problem than a typical eigenvalue problem. We take note of the eigenvalue problem given in Eq. 21 and iteratively solve

$$\Psi \leftarrow T^{-1}H\left(\frac{\hat{F}}{k}\Psi + \hat{S}\right)W\Psi. \tag{24}$$

Note that since we are concerned with arriving at the solution of ϕ, we are really interested in

$$\phi \leftarrow WT^{-1}H\left(\frac{\hat{F}}{k} + \hat{S}\right)(W\Psi). \tag{25}$$

which can be re-written as

$$\phi \leftarrow WT^{-1}H\left(\frac{\hat{F}}{k} + \hat{S}\right)\phi. \tag{26}$$

After each iteration, we arrive at a new estimate of ϕ and from this updated estimate it is possible to compute a new value for k. The new value of k is computed by forcing

$$\frac{1}{k}\hat{F}\phi\mathbb{1} = C \tag{27}$$

where C is some constant and $\mathbb{1}$ is the vector of ones. Therefore at each iteration k is updated by taking

$$k \leftarrow k\frac{\hat{F}\phi_{\text{new}}\mathbb{1}}{\hat{F}\phi\mathbb{1}} \tag{28}$$

Physically, this is taking the eigenvalue k to be the ratio of fission sources between iterations. The process continues iteratively until a stationary point is reached and the eigenvalue problem is converged. This algorithm is detailed in Algorithm 3.

Source iteration is indeed the strategy used in standard MOC solvers. Benefiting from being entirely based on the scalar fluxes ϕ, angular flux vector Ψ does not need to be explicitly stored. During a transport sweep in the standard solver, $\phi_{\text{new}} \leftarrow WT^{-1}HQ$ is implicitly computed where Q is the sources formed from the current estimate of the scalar fluxes. During the transport sweep, angular fluxes are formed as needed in the computation associated with the elements of T.

Algorithm 3 Source Iteration Algorithm

Take a starting guess for ϕ
while ϕ not converged **do**
  $\phi_{\text{new}} \leftarrow WT^{-1}H\left(\frac{\hat{F}}{k} + \hat{S}\right)\phi$
  Normalize ϕ_{new}
  Compute $k \leftarrow k\frac{\hat{F}\phi_{\text{new}}}{\hat{F}\phi}$
  $\phi \leftarrow \phi_{\text{new}}$
end while

Trying to mimic this strategy in matrix form, an `MOCMatrix` type is created which defines how to perform matrix-vector multiplication with the matrix $WT^{-1}H$ wherein matrix elements are computed on-the-fly by forming the relevant elements of W, T, and H individually on-the-fly with some common terms saved to reduce computation. This leads to a large reduction in storage since neither the W, T, nor H matrices need to be stored explicitly. The idea is very similar to the `SymmetricArrow` matrix studied in homework 2 for this class. By taking advantage of matrix structure, both storage and computation can be greatly reduced.

6 The MOC.jl Package

To do this work, I created the `MOC.jl` package. This package contains all the up-front computation required by the MOC method including geometry descriptions, ray tracing, material definitions, track generation, and quadrature sets. However, at the moment it is limited to only uniform grid geometries with vacuum boundary conditions for two dimensional problems and quadrature sets are limited to a maximum of three polar angles – although experience has shown three polar angles to be sufficient to converge most neutron transport problems. The package also contains various eigenvalue solver options:

1. `StandardSolver`: An eigenvalue solver that computes the solution with source iteration by iterating over tracks in the usual fashion with minimal storage (as presented in Algorithm 1).

2. `SparseSolver`: An eigenvalue solver that explicitly creates the aforementioned matrices and saves them in sparse form. It then uses these matrices to compute the solution using either: Julia's `eigs` solver, power method, or source iteration.

3. `MOCMatrixSolver`: An eigenvalue solver that forms an `MOCMatrix` for use in source iteration and does not explicitly store angular fluxes. While similar in structure to `StandardSolver`, it computes the components of each necessary matrix element explicitly on-the-fly whereas StandardSovler takes advantage of certain cancellations that occur in the computation.

7 Results and Analysis

The various solvers implemented in Julia were tested on a simple benchmark problem. The benchmark problem was chosen to be a pure UO_2 fuel with cross-sections from the C5G7 benchmark. The number of azimuthal and polar angles used in the quadrature and the problem discretization were varied yielding several different problem complexities. The computational times required to solve these problems of various complexities or sizes are shown in Fig. 1.

Note that the best solver is the `StandardSolver` followed by the `MOCMatrixSolver`. It is important to note that it should be possible to make the `MOCMatrixSolver` as efficient as the `StandardSolver` by making the same simplifications and cancellation of terms in the computation. However, in that case the computation would be virtually the same and the `MOCMatrixSolver` would merely be copying the same code as the `StandardSolver`, just calling the transport sweep a matrix multiplication. For this reason, the choice was made to form each matrix term individually in the `MOCMatrixSolver` without any cancellations in order to quantify the performance benefit of the cancellations and simplifications commonly employed in efficient MOC solvers.

The worst performing solver is the `SparseSolver` which explicitly computes and stores all matrix terms in standard sparse format. Of the eigenvalue solver options, power method performed the worst. The Julia `eigs`

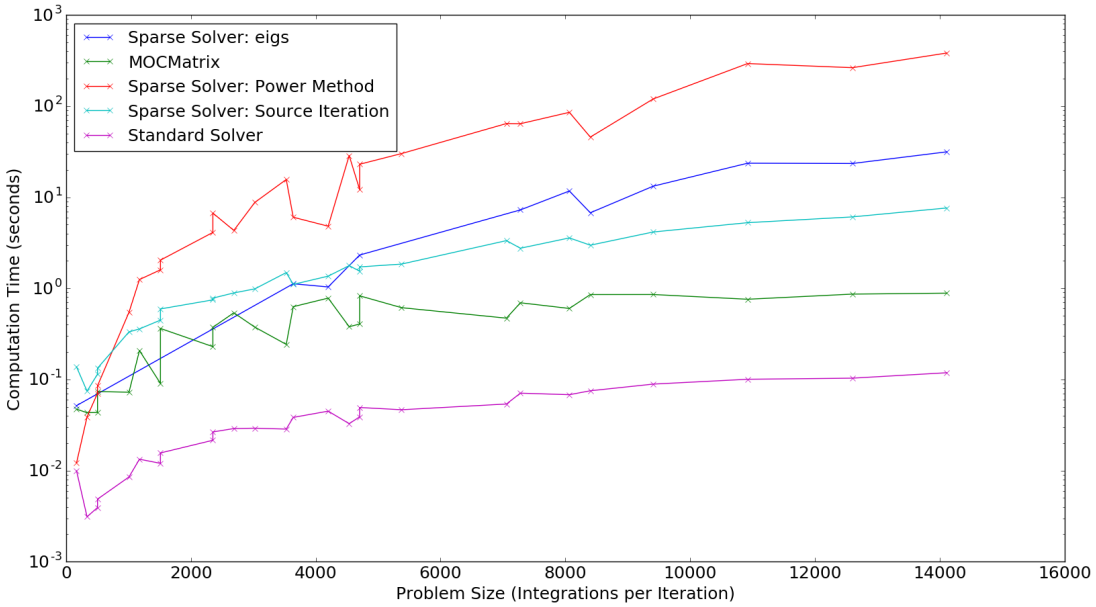


Figure 1: Computational time for various solvers to solve test problems of UO_2 fuel with vacuum boundaries.

function showed much better performance, as expected, since the algorithms it employs should be strictly better than power method for almost all use cases.

However, source iteration proved to be better than both of the more standard eigenvalue methods. To understand the reasons for this, we consider the computation performed at each iteration. First, note that source iteration works directly with ϕ which is size M whereas both power and `eigs` aim to calculate the dominant eigenvector Ψ which is of length N and then from that solution calculate $\phi = W\Psi$.

In addition to directly solving for ϕ, source iteration also benefits from an easier linear system to solve at each iteration. Consider that source iteration solves the system

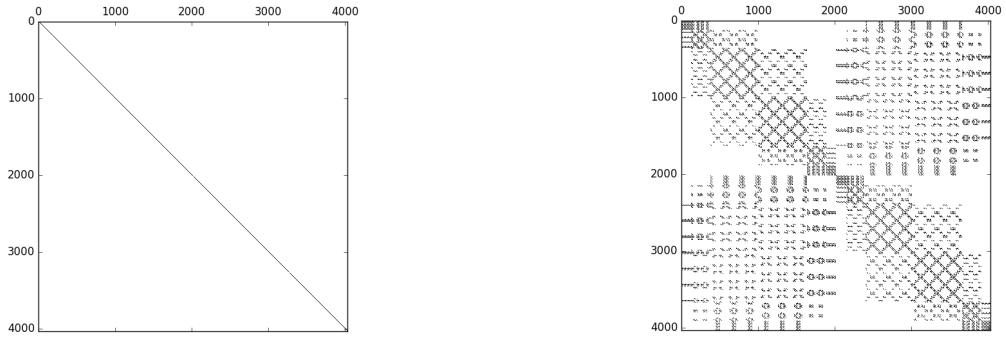
$$\phi \leftarrow WT^{-1}H\left(\frac{\hat{F}}{k} + \hat{S}\right)\phi \tag{29}$$

whereas both power method and `eigs` solve the linear system

$$\Psi \leftarrow \left(T - H\hat{S}W\right)^{-1} H\hat{F}W\Psi. \tag{30}$$

The complexity of solving linear systems depends strongly on the structure of the matrix being implicitly inverted. For source iteration that matrix is T whereas for power method and `eigs` that matrix is $T - H\hat{S}W$. A comparison of the structure of these two matrices is shown in Fig. 2.

The structure of T is extremely simple. As noted early it is just a diagonal matrix with the potential for an off-diagonal element at each row. It is also lower triangular. These features make solving the linear system almost trivial. In contrast, we see that $T - H\hat{S}W$ has a much more complicated structure.



(a) T (b) $T - H\hat{S}W$

Figure 2: A comparison of the matrix being implicitly inverted in source iteration (T) and the matrix being implicitly inverted in power method and eigs ($T - H\hat{S}W$)

This explanation is reinforced by comparing source iteration with power method. First, we look at the iteration count of both the methods. This is shown in Fig. 3. Although power method was shown to be far less computationally efficient, it is able to reach convergence in far fewer iterations.

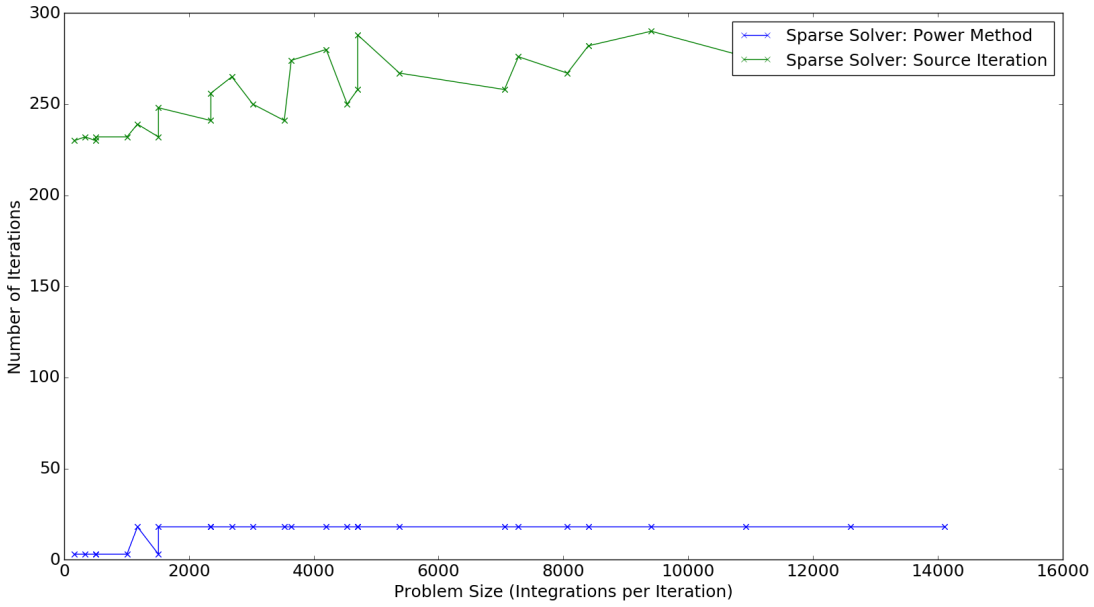


Figure 3: Required iterations for power method and source iteration to solve test problems of UO_2 fuel with vacuum boundaries.

This means that the discrepancy in run time must be accounted for by increased computational work at each iteration. This makes physical sense as power method is essentially using the updated guess of the flux to compute the scattering source whereas source iteration uses the flux from the previous iteration to compute the scattering source, essentially lagging the scattering source term. While using an updated scattering source requires fewer iterations, it is easy to see why it would be computationally cumbersome – especially when imagining a sweeping algorithm such as that used in the `StandardSolver` which uses the updated flux to compute the scattering source. The source iteration also seems to make more physical sense for the method of characteristics since all source terms are lagged whereas the power method and similarly `eigs` only lag the

fission source terms.

Lastly, it was noted previously that whenever we solve for the scalar flux ϕ with source iteration, a transport sweep is conducted that implicitly multiplies the neutron source Q by $WT^{-1}H$ at each iteration. If this matrix were formed explicitly, it would be only size $M \times M$ even though the internal matrices have some dimensions of length N. Therefore, if it were simple to explicitly compute the matrix $WT^{-1}H$, each iteration would just be a simple matrix-vector product with an $M \times M$ matrix, greatly reducing the computational cost. The structure of $WT^{-1}H$ is shown in Fig. 4.

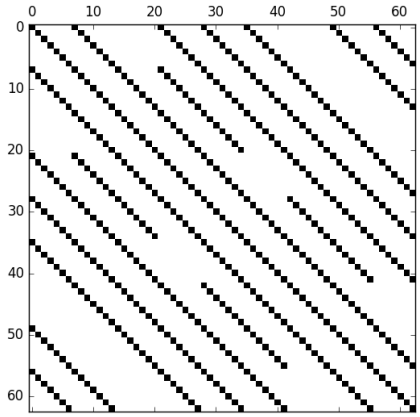


Figure 4: Structure of the $WT^{-1}H$ matrix for a 5×5 mesh of UO_2 fuel with C5G7 cross-sections.

This structure seems rather simple. The matrix is reasonably sparse and of greatly reduced size when compared with other matrices in this application (for example T). However, explicitly computing $WT^{-1}H$ is far more computationally expensive than solving the eigenvalue problem itself, making this strategy infeasible.

8 Conclusion

In this project, I was able to form a variety of solvers that were able to solve the neutron transport equation. These solvers were created using entirely Julia code. The best performing solver was the standard MOC solver, mimicking the algorithm used by many existing MOC solvers. While this solver does not seem to resemble a matrix solver, I discovered in this project that the transport sweep – which is the fundamental component in the solver – actually solves matrix-multiplication with the matrix $WT^{-1}H$ in an efficient manner. This matrix-vector multiplication arises from the source iteration solution to the eigenvalue problem which is more efficient than competing methods due to the inner linear solve being far easier to compute due to matrix structure. In addition using this method allows the solver to operate largely on reduced dimensionality since it is yields inputs and outputs which only depend on the new estimate of scalar fluxes at each iteration.

References

[1] William Boyd, Samuel Shaner, Lulu Li, Benoit Forget, and Kord Smith. The OpenMOC Method of Characteristics Neutral Particle Transport Code. *Annals of Nuclear Energy*, 2014.

[2] Abel Marin-lafleche, Micheal A Smith, and Changho Lee. PROTEUS-MOC : A 3D Deterministic Solver Incorporating 2D Method of Characteristics. pages 2759–2770, 2013.

[3] W. Boyd. Massively Parallel Algorithms for Method of Characteristics Neutral Particle Transport on Shared Memory Computer Architectures. M.S. Thesis, Massachusetts Institute of Technology, Department of Nuclear Science and Engineering (2014).