NERS 544 Class Project

Brody Bassett

February 17, 2016

1 Introduction

This code will solve the one-dimensional, slab geometry neutron transport problem with isotropic scattering and weight windows. This section assumes knowledge of standard Monte Carlo techniques, and as such does not describe concepts such as rouletting, splitting, implicit capture and sampling from a random distribution that are included in the course lecture notes.

1.1 Weight windows biased by adjoint scalar flux

Define the matrices

$$\boldsymbol{w} = \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{I-1,1} & w_{I,1} \\ w_{1,2} & \ddots & & & w_{I,2} \\ \vdots & & \ddots & & \vdots \\ w_{1,G-1} & & & \ddots & w_{I,G-1} \\ w_{1,G} & w_{2,G} & \cdots & w_{I-1,G} & w_{I,G} \end{bmatrix} \quad \boldsymbol{\phi}^\dagger = \begin{bmatrix} \phi^\dagger_{1,1} & \phi^\dagger_{2,1} & \cdots & \phi^\dagger_{I-1,1} & \phi^\dagger_{I,1} \\ \phi^\dagger_{1,2} & \ddots & & & \phi^\dagger_{I,2} \\ \vdots & & \ddots & & \vdots \\ \phi^\dagger_{1,G-1} & & & \ddots & \phi^\dagger_{I,G-1} \\ \phi^\dagger_{1,G} & \phi^\dagger_{2,G} & \cdots & \phi^\dagger_{I-1,G} & \phi^\dagger_{I,G} \end{bmatrix} \tag{1}$$

for $\boldsymbol{w}$, the matrix of weight windows $w_{i,g}$, and $\boldsymbol{\phi}^\dagger$, the discrete values for the adjoint scalar flux from a deterministic transport calculation, where i and g are, respectively, a spatial cell (region) and an energy group. In all cases, the lower ($\boldsymbol{w}_{low}$), ideal, ($\boldsymbol{w}_{id}$), and upper ($\boldsymbol{w}_{up}$) weight windows were related by the equation

$$\boldsymbol{w}_{low} = k\boldsymbol{w}_{id} = k^2\boldsymbol{w}_{up} \tag{2}$$

with $k = 2$. The weight window values were initialized to $\boldsymbol{w}_{id} = \mathbf{1}$ for problems in which the adjoint was not used to bias the weight windows. To bias the weight windows, the spatially averaged adjoint scalar flux is computed using the cell length (or relative volume) Δx_i as

$$\overline{\phi}^\dagger = \frac{\sum_{i,g} \Delta x_i \phi^\dagger_{i,g}}{\sum_i \Delta x_i}. \tag{3}$$

Next, the lower weight windows were calculated as

$$\boldsymbol{w}_{low} = \frac{\overline{\phi}^\dagger}{\boldsymbol{\phi}^\dagger}, \tag{4}$$

where the division is pointwise, with the ideal and upper weight window values following from Eq. 2. Finally, to prevent particles from being born and immediately rouletted, the weight windows were normalized so that

the average of the weight windows in locations where source particles were born would be equal to one. The source-biased average of the weight windows was calculated as

$$\overline{w}_S = \frac{\sum_{i,g} w_{i,g} \Delta x_i S_{i,g}}{\sum \Delta x_i S_{i,g}}, \tag{5}$$

after which the upper, lower and ideal weight windows were renormalized from their values calculated in Eq. 4:

$$\tilde{\boldsymbol{w}} = \frac{\boldsymbol{w}}{\overline{w}_S}. \tag{6}$$

This is similar to and derived from the response method used by Wagner and Haghighat [2].

1.2 Source distribution

The total source for the problem was calculated as

$$\bar{S} = \sum_{i,g} \Delta x_i S_{i,g}, \tag{7}$$

after which source distribution (or the likelihood of a source particle being born into a certain cell and energy group) was calculated as

$$\tilde{S}_{i,g} = \frac{\Delta x_i S_{i,g}}{\bar{S}}. \tag{8}$$

after which the function $\tilde{S}_{i,g}$ was sampled using standard methods, with the position in cell i determined using the cell-center position $\overline{x}_i$ and a random $\xi \in [0,1]$ as

$$x = \bar{x}_i + \Delta x_i \left(\xi - \frac{1}{2} \right). \tag{9}$$

1.3 Current and scalar flux estimators

Table 1 includes four different estimators for this problem in a slightly different form than in the lecture notes. $\phi_{i,g}$ represents the scalar flux in cell i and group g, while $J_{i\pm1/2,g}$ and $\phi_{i\pm1/2,g}$ represent the total current and scalar flux, respectively, at the positive or negative cell edge of cell i and in group g . Let $n \in N$ be the current history and m represent some event within the history. For the radiation current and surface flux, m is incremented each time a particle crosses a surface. For the radiation volume flux, m is incremented each time a new track is produced. For the collision flux, m is incremented each time a collision occurs.

Estimator	Mean	Mean squared
Radiation current	$J_{i\pm1/2,g} = \frac{\bar{S}}{N} \sum_{n=1}^{N} \left(\sum_m w_{m,n,g,i} \right)$	$J^2_{i\pm1/2,g} = \frac{\bar{S}^2}{N} \sum_{n=1}^{N} \left(\sum_m w_{m,n,g,i} \right)^2$
Radiation surface flux	$\phi_{i\pm1/2,g} = \frac{\bar{S}}{N} \sum_{n=1}^{N} \left(\sum_m \frac{w_{m,n,g,i}}{\lvert\mu\rvert} \right)$	$\phi^2_{i\pm1/2,g} = \frac{\bar{S}^2}{N} \sum_{n=1}^{N} \left(\sum_m \frac{w_{m,n,g,i}}{\lvert\mu\rvert} \right)^2$
Radiation volume flux	$\phi_{i,g} = \frac{\bar{S}}{NV_i} \sum_{n=1}^{N} \left(\sum_m w_{m,n,g,i} \ell_{m,n,g,i} \right)$	$\phi^2_{i,g} = \frac{\bar{S}^2}{NV_i^2} \sum_{n=1}^{N} \left(\sum_m w_{m,n,g,i} \ell_{m,n,g,i} \right)^2$
Collision flux	$\phi_{i,g} = \frac{\bar{S}}{NV_i} \sum_{n=1}^{N} \left(\sum_m \frac{w_{m,n,i,g}}{\Sigma_{t;m,n,i,g}} \right)$	$\phi^2_{i,g} = \frac{\bar{S}^2}{NV_i^2} \sum_{n=1}^{N} \left(\sum_m \frac{w_{m,n,i,g}}{\Sigma_{t;m,n,i,g}} \right)^2$

Table 1: Estimators

1.4 Linearly anisotropic scattering

The total cross section for linearly anisotropic scattering is

$$\Sigma_s(\mu_0) = \frac{1}{2}\left(\Sigma_{s,0} + 3\Sigma_{s,1}\mu_0\right), \tag{10}$$

where $\mu_0 = \mu\mu'$ is the dot product of the direction vectors of the incoming and outgoing particles. The average value of this function is simply

$$\int_{-1}^{1} \Sigma_s(\mu_0)\, d\mu_0 = \Sigma_{s,0}, \tag{11}$$

while the probability density function for scattering from the angle μ' to μ can be calculated using the above normalization as

$$f(\mu,\mu') = \frac{1}{2\Sigma_{s,0}} \int_{-1}^{1} \left(\Sigma_{s,0} + 3\mu\mu''\Sigma_{s,1}\right)\delta(\mu''-\mu')\, d\mu'' = \frac{\Sigma_{s,0} + 3\mu\mu'\Sigma_{s,1}}{2\Sigma_{s,0}}. \tag{12}$$

Note that for this PDF to be positive for all values of $\mu\mu'$, the cross sections must obey the condition

$$\frac{\Sigma_{s,0}}{3\Sigma_{s,1}} > 1. \tag{13}$$

The cumulative density function will then follow as

$$F(\mu,\mu') = \int_{-1}^{\mu} \frac{\Sigma_{s,0} + 3\mu\mu'\Sigma_{s,1}}{2\Sigma_{s,0}} d\mu = \frac{(\mu+1)\left(2\Sigma_{s,0} + 3(\mu-1)\mu'\Sigma_{s,1}\right)}{4\Sigma_{s,0}}. \tag{14}$$

Setting $\xi = F(\mu,\mu')$ then gives the outgoing scattering angle μ in terms of the incoming scattering angle μ' and a random $\xi \in [0,1]$:

$$\mu = \frac{-\Sigma_{s,0} + \sqrt{\left(\Sigma_{s,0} - 3\mu_0\Sigma_{s,1}\right)^2 + 12\mu_0\Sigma_{s,0}\Sigma_{s,1}\xi}}{3\mu_0\Sigma_{s,1}}. \tag{15}$$

If $\Sigma_{s,1} = 0$, the equation for isotropic scattering must be used, in which an angle is sampled as

$$\mu = 2\xi - 1. \tag{16}$$

1.5 Figure of merit

The precision was calculated as

$$P_{i,g} = \sqrt{\frac{\phi_{i,g}^2 - \left(\phi_{i,g}\right)^2}{N}}, \tag{17}$$

the relative error as

$$R_{i,g} = \frac{P_{i,g}}{\phi_{i,g}}, \tag{18}$$

and the figure of merit as

$$FOM_{i,g} = \frac{1}{R_{i,g}^2 T}, \tag{19}$$

where T is the wall time the simulation took to run [1].

2 Code

Class	Description
Bank	Holds the particles (see the class below) waiting in the stack.
Estimators	Holds the various flux estimators and calculates values such as variance and figure of merit
Monte Carlo	Computes source and cross section distributions. Calculates weight windows. Runs individual particle simulations.
Particle	Holds the cell, energy, weight, position and angle of each particle.

Table 2: Monte Carlo classes

In addition to the mesh, data handling and deterministic transport classes, the code has the Monte Carlo-specific codes as described in Table 2. The Monte Carlo class does most of the simulation. The routine to run a set of histories is included in part in Listing 1. Weight windows for the problem are checked whenever a particle was born, has a collision, or crosses a cell boundary. At the end of the simulations, the estimators are normalized by the total source weight and (for some) the cell volume.

Listing 1: Manatee Solution Routine

```
void Monte_Carlo::
solve()
{
    Teuchos::Time timer("solve");
    timer.start();

    for (unsigned h = 0; h < number_of_histories_; ++h)
    {
        get_source_particle();

        estimators_.begin_history();

        while(bank_.particles_remain())
        {
            Particle particle = bank_.get_particle();

            check_weight_windows(particle);

            while (particle.is_alive())
            {
                double mfp = sample_mfp();

                while (mfp > 0)
                {
                    if (particle.angle() > 0)
                    {
                        stream_right(mfp,
                                     particle);
                    }
                    else
                    {
                        stream_left(mfp,
                                    particle);
                    }
```

```
                    if (particle.is_alive())
                    {
                        check_weight_windows(particle);
                    }
                    else
                    {
                        break;
                    }
                }

                if (particle.is_alive())
                {
                    simulate_collision(particle);

                    check_weight_windows(particle);
                }
            }
        }

        estimators_.end_history();
    }

    estimators_.normalize(total_source_,
                          number_of_histories_,
                          cell_volume_);

    runtime_ = timer.stop();
}
```

The collisions are calculated for either implicit or analog capture. This process is shown in Listing 2, which excludes the fission sampling for brevity. Note that for analog capture, the maximum value for the CDF of the cross sections is equal to Σ_t, while for implicit capture that value is $\Sigma_t - \Sigma_a$, to prevent a particle whose weight has already been reduced by the absorption probability from being absorbed.

Listing 2: Manatee Collision Routine

```
void Monte_Carlo::
simulate_collision(Particle &particle)
{
    estimators_.add_collision(particle.cell(),
                              particle.group(),
                              particle.weight(),
                              data_.sigma_t(particle.cell(), particle.group())
                                  );

    double cross_rand = 0;

    if (implicit_capture_)
    {
        particle.weight() *= 1 - data_.sigma_a(particle.cell(), particle.group
            ()) / data_.sigma_t(particle.cell(), particle.group());

        cross_rand = get_rand() * (data_.sigma_t(particle.cell(), particle.
            group()) - data_.sigma_a(particle.cell(), particle.group()));
```

```
    }
    else
    {
        cross_rand = get_rand() * data_.sigma_t(particle.cell(), particle.
            group());
    }

    double cross_sum = 0;

    // scattering
    for (unsigned g = 0; g < data_.number_of_groups(); ++g)
    {
        cross_sum += data_.sigma_s(particle.cell(), particle.group(), g);

        if (cross_rand < cross_sum)
        {
            if (data_.number_of_scattering_moments() == 1)
            {
                particle.angle() = get_isotropic_angle();
            }
            else if (data_.number_of_scattering_moments() == 2)
            {
                particle.angle() = get_anisotropic_angle(g,
                                                         particle);
            }
            else
            {
                cerr << "only first moment of scattering supported" << endl;
            }

            particle.group() = g;

            return;
        }
    }

    // absorption
    cross_sum += data_.sigma_a(particle.cell(), particle.group());

    if (cross_rand < cross_sum)
    {
        particle.is_alive() = false;

        return;
    }

    // cross sections not equal to total
    cerr << "cross sections do not sum to total" << endl;
}
```

Finally, the code for streaming in the right direction is included in Listing 3. If the mean free path sampled puts the particle outside of the current cell, the mean free path is reduced by the distance travelled and the position of the particle is placed at the next cell edge. If the particle has a collision, the mean free path is set to zero, after which the particle breaks from the streaming loop in the solution routine and has a collision

as sampled in the collision routine. Each time a particle streams any distance, crosses a cell boundary, or collides with the material, the result is tallied into the estimator for the current history.

Listing 3: Manatee Streaming Routine

```
void Monte_Carlo::
stream_right(double &mfp,
             Particle &particle)
{
    double distance = mfp / data_.sigma_t(particle.cell(), particle.group());
    double boundary_distance = distance_to_boundary(particle.cell(), particle.
        position(), particle.angle());

    if (distance > boundary_distance) // cross cell boundary
    {
        mfp -= boundary_distance * data_.sigma_t(particle.cell(), particle.
            group());

        estimators_.add_track(particle.cell(),
                              particle.group(),
                              particle.weight(),
                              boundary_distance);

        estimators_.add_crossing(particle.cell() + 1,
                                 particle.group(),
                                 particle.weight(),
                                 particle.angle());

        particle.position() = mesh_.cell_edge_position(particle.cell(), 1);

        if (particle.cell() == mesh_.number_of_cells() - 1) // at left
            boundary
        {
            if (data_.boundary_condition(1).compare("reflected") == 0)
            {
                particle.angle() = - particle.angle();

                estimators_.add_crossing(particle.cell() + 1,
                                         particle.group(),
                                         particle.weight(),
                                         particle.angle());
            }
            else if (data_.boundary_condition(1).compare("vacuum") == 0)
            {
                mfp = 0;

                particle.is_alive() = false;
            }
            else
            {
                cerr << "boundary condition does not exist" << endl;
            }
        }
        else // at another cell boundary
        {
```

```
            particle.cell() += 1;
        }
    }
    else
    {
        estimators_.add_track(particle.cell(),
                              particle.group(),
                              particle.weight(),
                              distance);

        mfp = 0;

        particle.position() += distance * particle.angle();
    }
}
```

Whenever the weight windows are checked, if the weight is below the minimum, the particles are rouletted with a probability that either raises the weight of the particle to the ideal weight or causes their weight to drop to zero. Where their weight is greater than the weight window, a number of new particles is sampled and these particles are added to the bank. Listing 4 shows this process.

Listing 4: Manatee Weight Window Checking Routine

```
void Monte_Carlo::
check_weight_windows(Particle &particle)
{
    if (particle.weight() < weight_windows(particle.cell(), particle.group()).
        lower)
    {
        double survival_probability = particle.weight() / weight_windows(
            particle.cell(), particle.group()).ideal;

        if (get_rand() < survival_probability)
        {
            particle.weight() = weight_windows(particle.cell(), particle.group
                ()).ideal;
        }
        else
        {
            particle.is_alive() = false;
        }
    }
    else if (particle.weight() > weight_windows(particle.cell(), particle.
        group()).upper)
    {
        unsigned particles_to_create = floor(particle.weight() /
            weight_windows(particle.cell(), particle.group()).ideal) - 1;

        particle.weight() /= (particles_to_create + 1);

        for (unsigned p = 0; p < particles_to_create; ++p)
        {
            bank_.add_particle(particle.cell(),
                               particle.group(),
                               particle.weight(),
```

```
                                        particle.position(),
                                        particle.angle());
            }
        }
}
```

The full source code is available at `https://github.com/brbass/manatee`. The most relevant files are in the following locations:

```
proj/proj_544.cc
src/monte_carlo/Bank.cc
src/monte_carlo/Bank.hh
src/monte_carlo/Estimators.cc
src/monte_carlo/Estimators.hh
src/monte_carlo/Monte_Carlo.cc
src/monte_carlo/Monte_Carlo.hh
src/monte_carlo/Particle.cc
src/monte_carlo/Particle.hh
src/transport_model/Sn_Transport.cc
src/transport_model/Sn_Transport.hh
src/transport_model/SP1_Transport.cc
src/transport_model/SP1_Transport.hh
```

See `https://github.com/brbass/NERS-544/tree/master/proj/code` for the full, tabulated results of the simulations, in the folders:

```
scattering_slab7/
thick_slab7/
```

3 Results and testing

The two material cases are as follows. The first problem is a scattering problem. The particles are born into the first group and then eventually scatter down to the second group. The response function is placed as a uniform source in the second group, which otherwise would have a lower population of neutrons. The boundaries of this problem are vacuum. For this problem, isotropic scattering was used.

Number of cells	$\Sigma_{t,1}$	$\Sigma_{t,2}$	$c_{1\to1}$	$c_{1\to2}$	$c_{2\to1}$	$c_{2\to2}$	S_1	S_2	R_1	R_2	x_{max}
300	10.0	10.0	0.6	0.39	0.19	0.8	1.0	0.0	0.0	1.0	1.0

Table 3: Scattering problem data

The second material case is a shielding problem. The first region is a thin, purely scattering medium in the first group and a thin, partially absorbing medium in the second group. It is this region into which the particles are born. The second region is a very optically thick region that is entirely scattering in the first group and mostly scattering in the second. Neutrons scatter only from the first to the second group. The third region is mostly scattering in the first group and completely absorbing in the second. This is the region where the response function was placed, since getting particles to this region would not occur otherwise. The slab length is 10.0, and is split evenly between the three regions, each of which has 100 spatial cells. The boundaries of this problem are reflecting. For the shielding problem, linearly anisotropic scattering was used with $\Sigma_{s,1,g'\to g} = \Sigma_{s,0,g'\to g}/4$ for all g' and g. In the table below, the scattering cross sections are defined in terms of the scattering ratio, in which $\Sigma_{s,g'\to g} = c_{g'\to g}\Sigma_{t,g'}$.

Region	Number of cells	$\Sigma_{t,1}$	$\Sigma_{t,2}$	$c_{1\to1}$	$c_{1\to2}$	$c_{2\to1}$	$c_{2\to2}$	S_1	S_2	R_1	R_2	x_{max}
1	100	0.1	0.1	0.9	0.1	0.0	0.9	1.0	0.0	0.0	0.0	$3.\overline{3}$
2	100	10.0	10.0	0.9	0.1	0.0	0.9	0.0	0.0	0.0	0.0	$6.\overline{6}$
3	100	1.0	1.0	0.45	0.45	0.0	0.0	0.0	0.0	0.0	1.0	10

Table 4: Shielding problem data

Material case	Analog, non-adjoint	Implicit, non-adjoint	Analog, adjoint	Implicit, adjoint
Scattering	1.000×10^7	1.000×10^7	1.008×10^7	1.008×10^7
Shielding	1.000×10^7	1.000×10^7	1.074×10^8	9.065×10^7

Table 5: Number of particles simulated

The adjoint functions used to calculate the weight windows were calculated in a multigroup, continuous finite element SP_1 code with data and mesh structures shared with the Monte Carlo code. For each Monte Carlo simulation, 10^7 histories were run. The total number of particles to be simulated was equal to the number of histories for the non-adjoint cases due to the constant weight windows. For the adjoint cases, the number of particles simulated varied due to splitting. Results are included below for each of the test problems and the following data:

- Adjoint functions and comparison to forward solution
- Comparison of estimators for the scalar flux when the weight windows are initialized with the adjoint function
- Comparison of estimators for the scalar flux when the weight windows are constant through the problem
- Combinations of implicit capture and initialization of the weight windows with the adjoint function
- The figure of merit for each of these combinations
- The volume flux standard error for each of these combinations

Each will be discussed in turn.

The adjoint problems correctly simulated the "importance" of the particles to the response function. In the scattering problem, the response function was in Group 2, and so the adjoint was higher in that region, causing lower weight windows for that group and more simulation time spent there.

In the second case the adjoint function had a more pronounced change, as particles on the very left of the problem had such a small probability of reaching the detector in the right region. The adjoint function remained fairly constant in the first region, where the material is optically thin. Then in the highly scattering region, particles gain importance exponentially (note the log scale) until they reach the third region, where they're most important to the detector response (the absorption in group 2 of the third region).

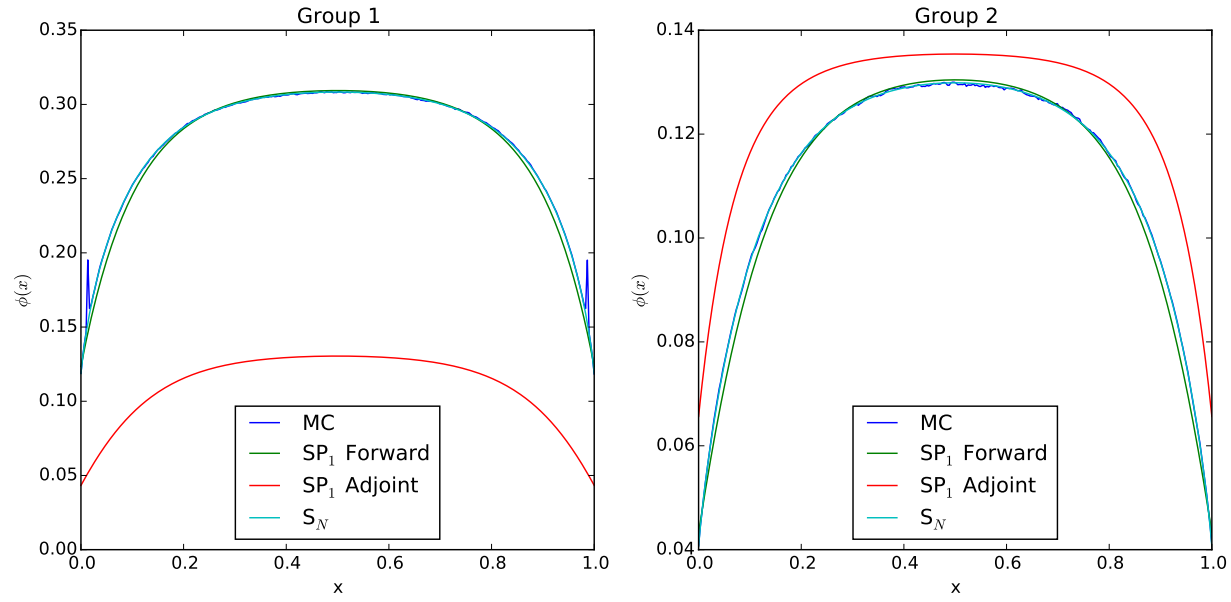

Figure 1: Adjoint and foward solutions, scattering problem

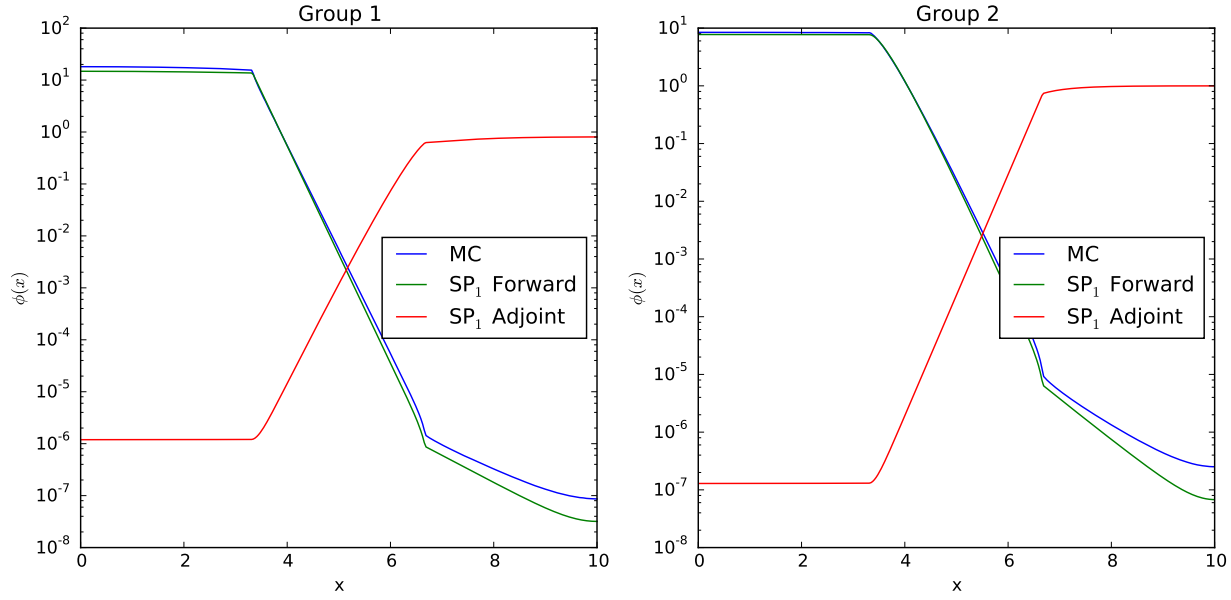

Figure 2: Adjoint and foward solutions, shielding problem

These figures show the difference between the estimators when the weight windows are initialized with the adjoint function and when they are not. In this case, the estimators look very similar regardless of whether they were initialized with the weight windows. The surface and volume estimators were relatively smooth (with the volume superior), while the collision estimator has definite peaks in its mean. The S_N transport model more closely approximated the Monte Carlo solution than the SP_1 model did.

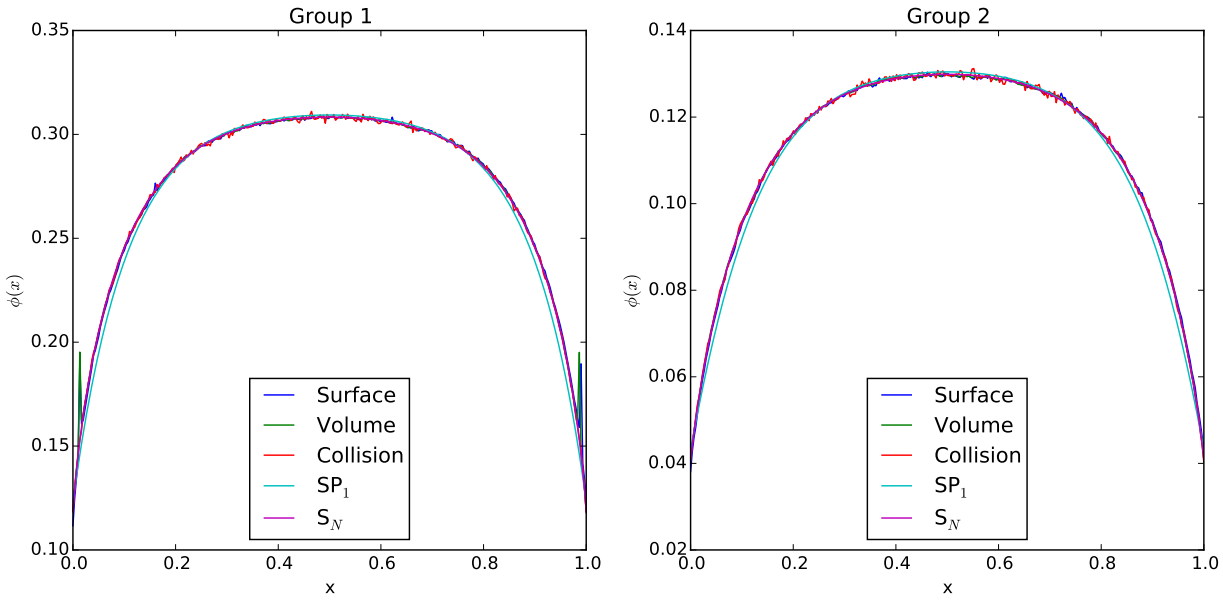

Figure 3: Estimators for the scalar flux, initialized with adjoint, implicit capture, scattering problem

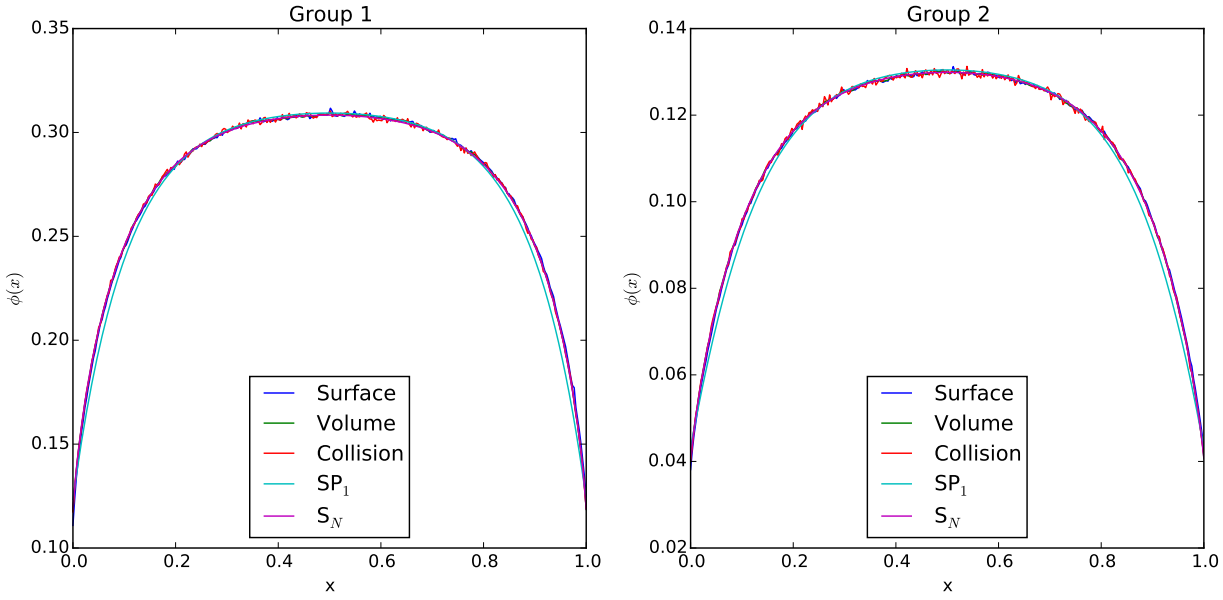

Figure 4: Estimators for the scalar flux, not initialized with adjoint, implicit capture, scattering problem

For this problem, the initialization with the adjoint changed the simulation entirely. Particles were highly unlikely to even reach the third region in the simulation with constant weight windows with the collision flux in particular behaving erratically. Then the problem was initialized with the adjoint, the estimators all agreed wonderfully with one another. Their solution was similar to the SP_1 solution except in the highly absorbing region, where the SP_1 equations are not accurate. Note that while initializing with the weight windows did not significantly affect the number of particles simulated in the scattering problem, in this case it raised the number of particles simulated by an order of magnitude, which increased the solution time significantly.

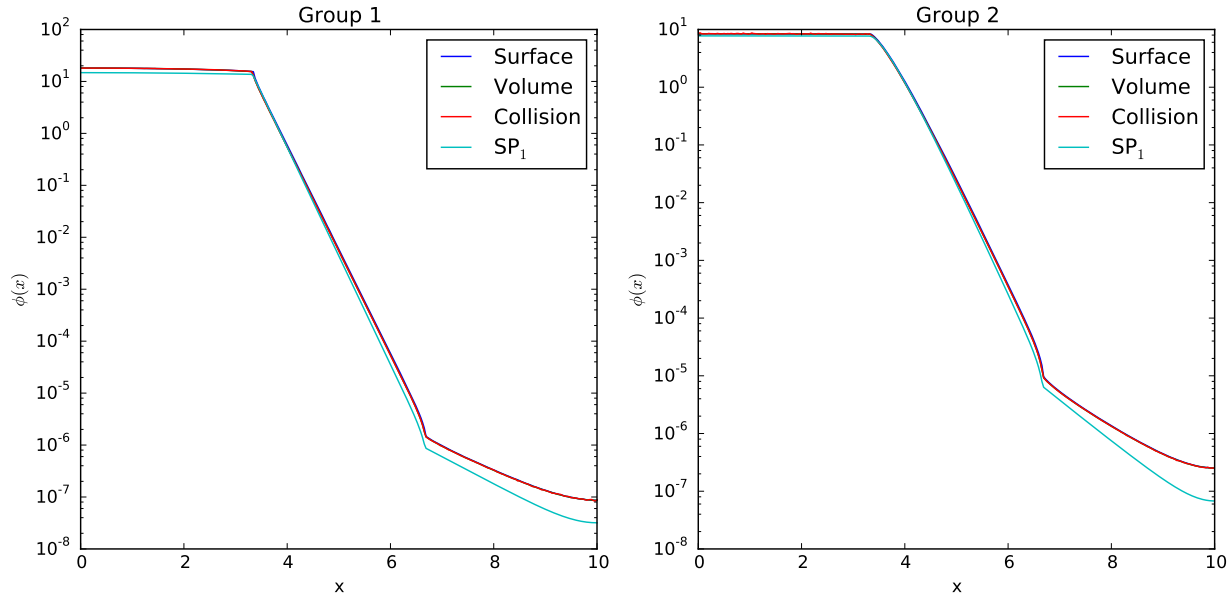

Figure 5: Estimators for the scalar flux, initialized with adjoint, implicit capture, shielding problem

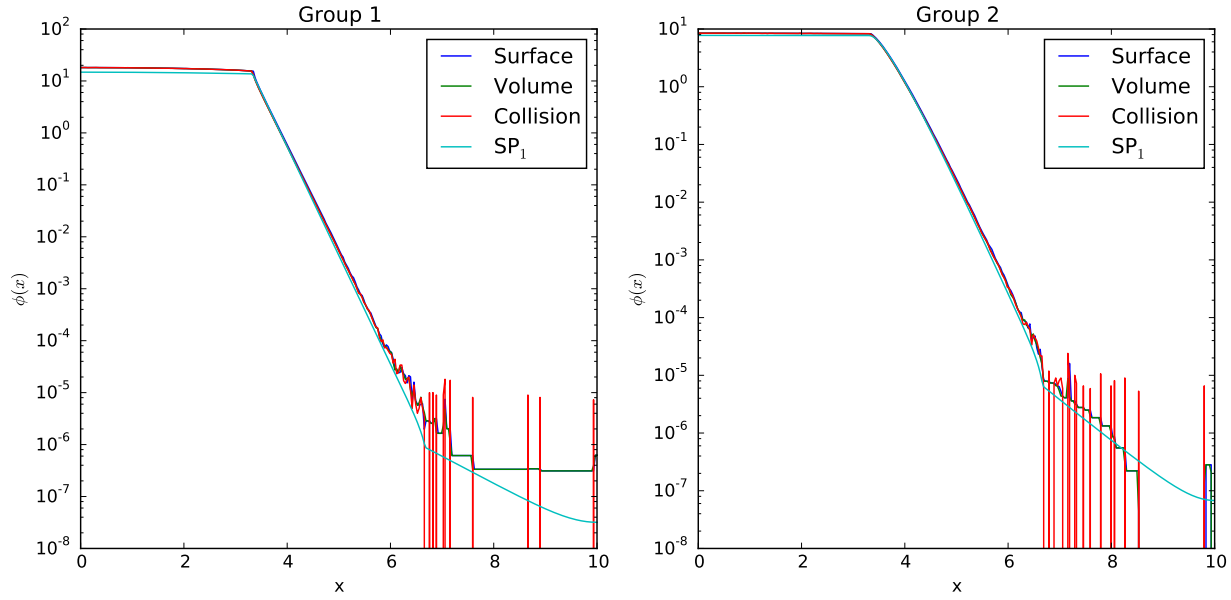

Figure 6: Estimators for the scalar flux, not initialized with adjoint, implicit capture, shielding problem

The implicit capture didn't appear to add much to the simulation in terms of the mean solution. In the cases where the adjoint was used, the implicit and analog scattering cases produced almost idential results. Interestingly, where the adjoint was used for the scattering problem, peaks appeared at the edges of the problem, which could be attributed to the higher weight windows at the boundaries (due to the lower adjoint near the boundaries) that caused particles that did survive the rouletting to have greater weight.

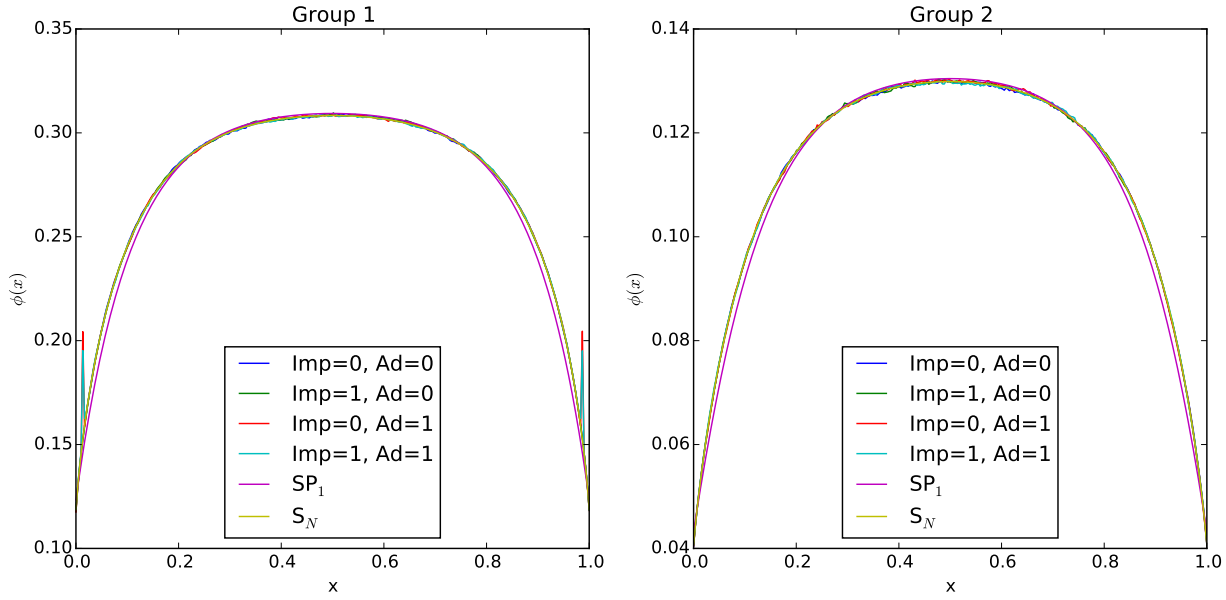

Figure 7: Volume flux, combinations of adjoint and implicit capture, scattering problem

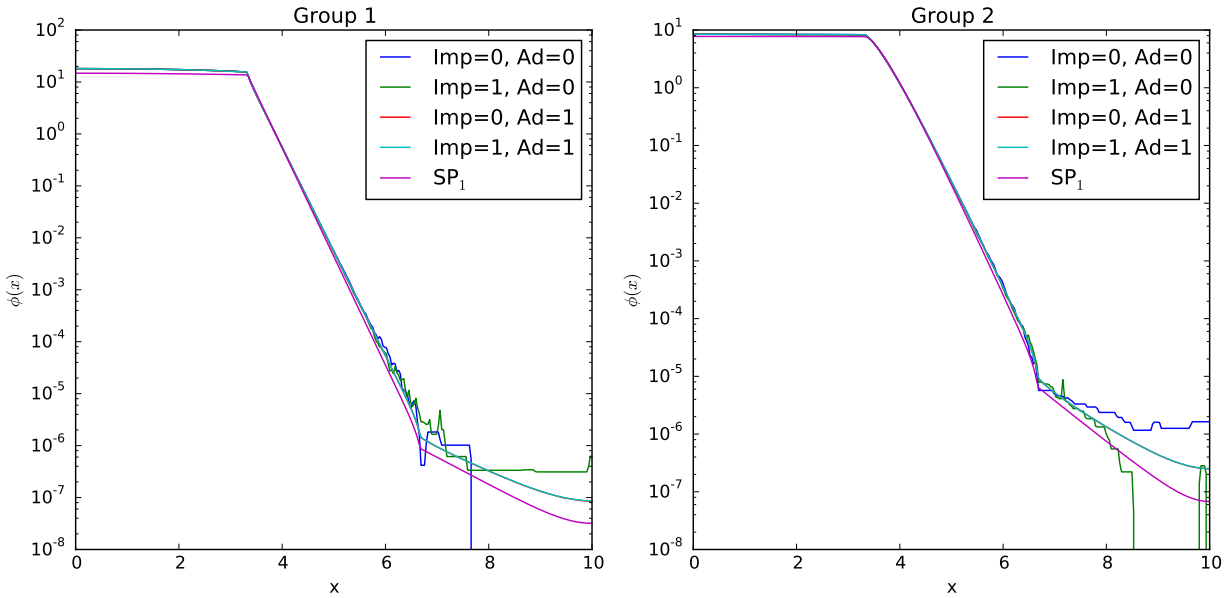

Figure 8: Volume flux, combinations of adjoint and implicit capture, shielding problem

The figure of merit for the scattering problem shows simply that either initializing with the adjoint or using implicit capture raises the figure of merit despite more computation time due to the additional particles. The case in which the adjoint and implicit capture were used had the highest figure of merit.

For the shielding problem, the figure of merit went down significantly in the third region for the non-adjoint cases. For the adjoint cases, however, the figure of merit remained relatively constant throughout the problem. The figure of merit was an order of magnitude lower for the first region in the second group, since more simulation time would be spent there in the non-adjoint case when those particles weren't preferentially rouletted. In the third region, however, the figure of merit was four orders of magnitude higher, since only one or a few particles reached that side of the problem in the non-adjoint case. The figure of merit was higher for implicit absorption than for analog in both cases.

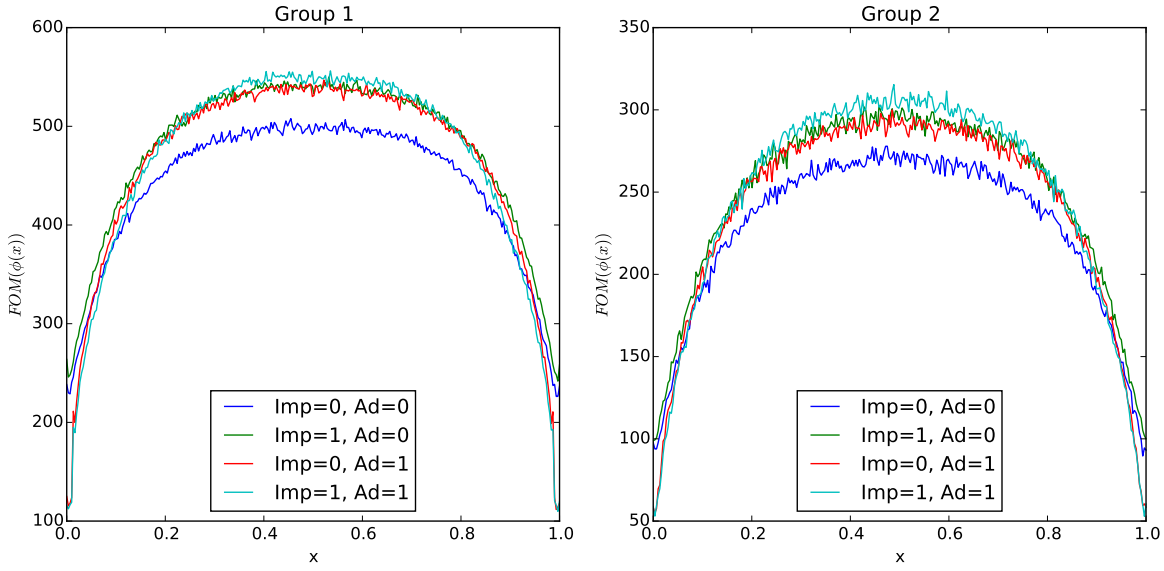

Figure 9: Volume flux figure of merit, combinations of adjoint and implicit capture, scattering problem

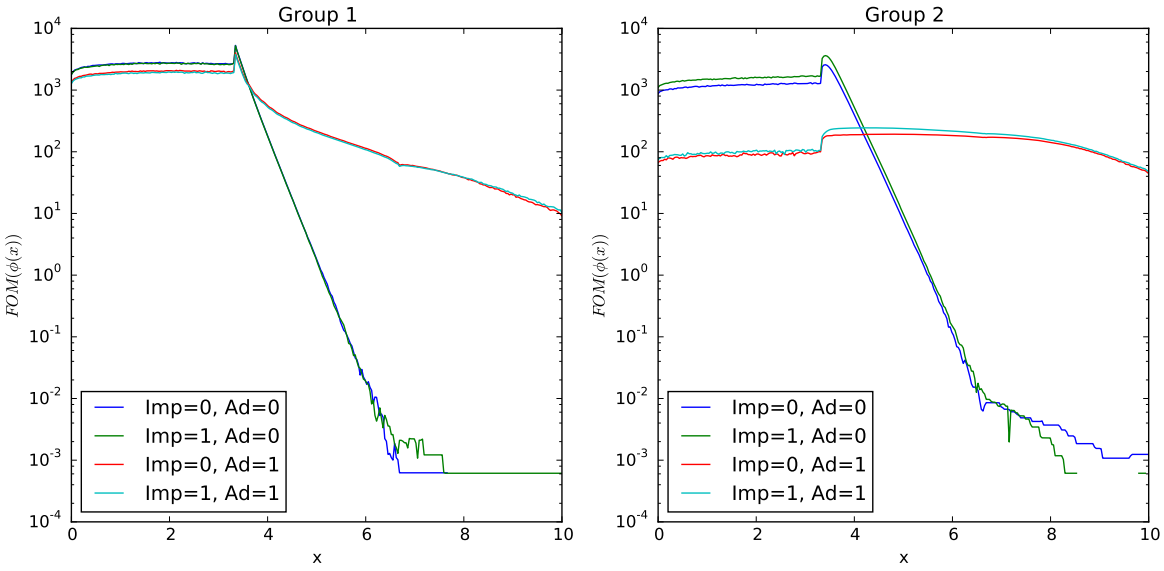

Figure 10: Volume flux figure of merit, combinations of adjoint and implicit capture, shielding problem

The standard error was lower for the implicit and adjoint cases. Separately they gave some improvement to the scattering solution, while together the net change was larger. The standard error was lower for the scattering case than for the first region of the shielding case, while for the second and third regions of the shielding case the error was very low due to the very low solution and number of particles that reached that point. In all test cases, implicit absorption and adjoint-weighted windows gave better results than analog absorption and constant weight windows.

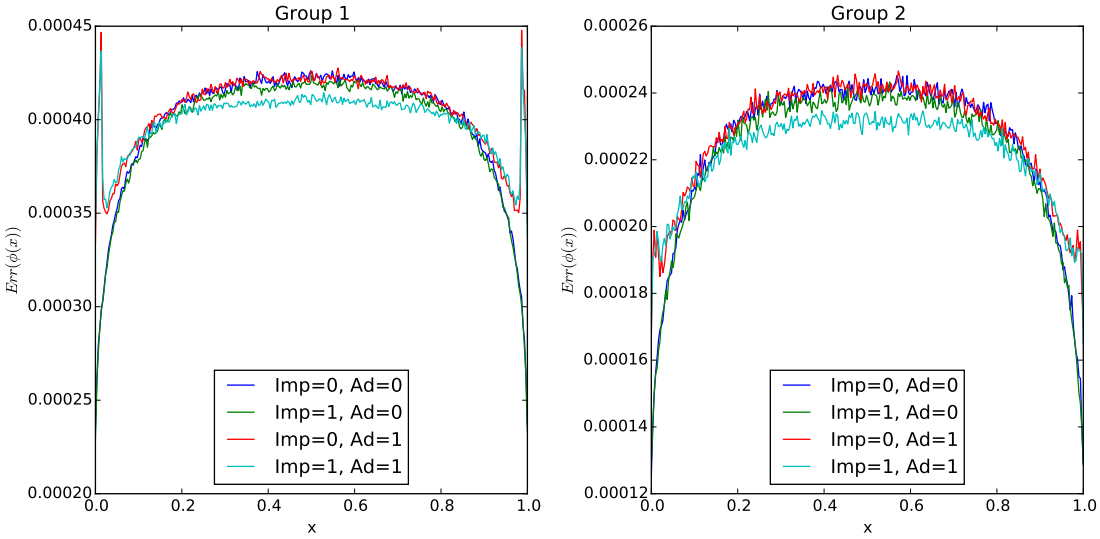

Figure 11: Volume flux standard error, combinations of adjoint and implicit capture, scattering problem

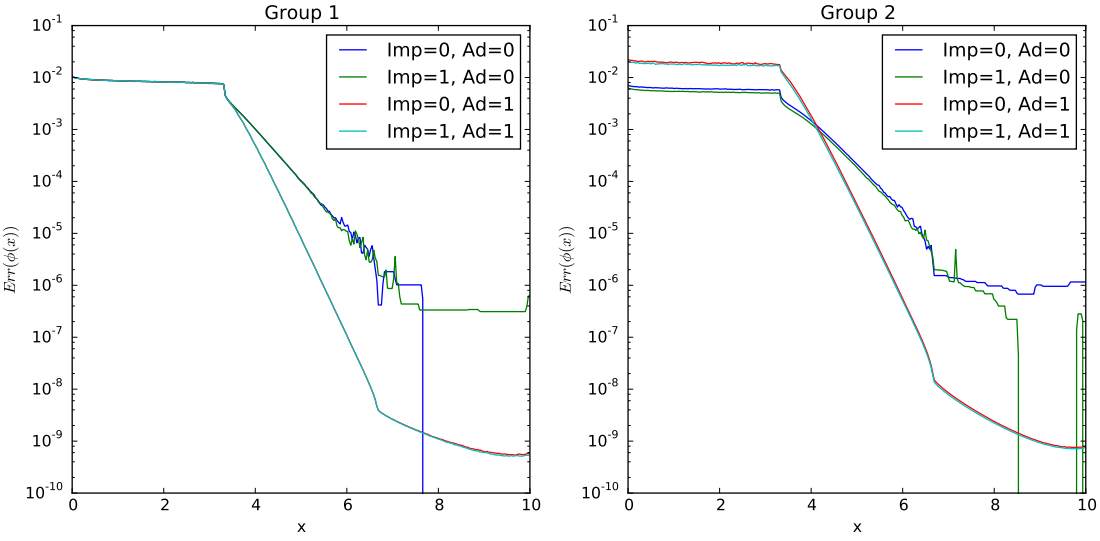

Figure 12: Volume flux standard error, combinations of adjoint and implicit capture, shielding problem

References

[1] Pooneh Saidi, Claudio Tenreiro, and Mahdi Sadeghi. *Variance reduction of Monte Carlo simulation in nuclear engineering field*. INTECH Open Access Publisher, 2013.

[2] John C Wagner and Alireza Haghighat. Automated variance reduction of monte carlo shielding calculations using the discrete ordinates adjoint function. *Nuclear Science and Engineering*, 128(2):186–208, 1998.