CHAPTER

22

Monte Carlo Variance Reduction and Scalar Flux Estimation

OUTLINE

"Man, this place is way too analog."

–*"Trotter" in the television show* The Upright Citizens Brigade

CHAPTER POINTS

- Abandoning analog tracking of simulated particles can have benefits for our calculations.
- Implicit capture allows neutrons to never be absorbed, rather we change a weight associated with the simulated particle.
- To estimate scalar fluxes we use one of two techniques based on either collisions or the track-length of particles in a mesh cell.
- We can play games with our sampling to randomly sample "better".

In the last chapter we talked about how to construct particles using a computer program that were analogs of real neutrons moving through a system. We saw that in many cases it took very many simulated neutrons to get reasonable answers. In this chapter we will see that by making our simulated particles behave in a non-analog way we can improve our answers.

We will not improve the convergence rate of Monte Carlo methods. Recall that the statistical error as measured by the standard deviation of the estimate converges as $O(N^{-1/2})$, where N is the number of simulated particles. This means that the standard deviation of our estimate is, to leading order, $CN^{-1/2}$. What variance reduction does is reduce the magnitude of C, the constant in the convergence.

22.1 IMPLICIT CAPTURE AND PARTICLE WEIGHTS

In problems where radiative capture is important, it can be beneficial to remove this absorption process from the types of interactions considered via a process called implicit capture. To do this we first introduce a particle weight, w. The weight is defined so that each particle represents a collection of neutrons, rather than a single one. Consider a neutron source in a volume, Q, with units neutrons per cm^3 per second. We will use N particles each with a weight w_i to represent this source in steady state calculation. The weights must satisfy

$$\sum_{i=1}^{N} w_i = \int_V dV\, Q.$$

From this relation we see that the units of w_i are neutrons per second.

To use implicit capture, as a particle moves a distance s in a material, we reduce its weight by a factor of $\exp(-\Sigma_\gamma s)$. This is done because this factor represents the likelihood of the neutron traveling a distance s without having a radiative capture reaction. With implicit capture, our simple shielding calculation has its procedure changed to be

1. Create a counter, $t = 0$ to track the number of neutrons that get through.
2. Create neutron with μ sampled from the uniform distribution $\mu \in [0, 1]$. Set $x = 0$. Set the particle's weight to be $w = 1/N$.
3. Sample randomly a distance to scatter, l, from the exponential distribution.
4. Move the particle to $x = x + l\mu$.
5. Reduce the weight of the particle by a factor $\exp(-\Sigma_\gamma s)$.
6. Check to see if $x > 3$. If so $t = t + w$, and go to 2. Otherwise, if $x < 0$ go to step 2.
7. Go back to step 3.

A couple of notes on this new algorithm. Now each particle has a weight and that is what we sum up to get the fraction of neutrons that leak out per unit time. Also, when we sample a distance to collision, we only sample a distance to scatter.

One drawback of implicit capture is that it can result in the tracking of particles that have a very small weight. After traveling a large distance, the weight could be much less than the initial weight and the particle could contribute only a small amount to the result. This is wasted computational effort tracking these low weight particles; it would be better to stop tracking them. For this purpose we introduce a cutoff weight. After decreasing from the initial weight by more than the cutoff, we treat the particle using analog tracking so that it can die via absorption.

The code to implement implicit capture for the slab transmission problem is below.

```
In [1]: def slab_transmission(Sig_s,Sig_a,thickness,N,
                              isotropic=False,
                              implicit_capture = True,
                              cutoff = 1.0e-3):
            """Compute the fraction of neutrons that leak through a slab
            Inputs:
            Sig_s:     The scattering macroscopic x-section
            Sig_a:     The absorption macroscopic x-section
            thickness: Width of the slab
            N:         Number of neutrons to simulate
            isotropic: Are the neutrons isotropic or a beam
            implicit_capture: Do we run implicit capture
            cutoff:    At what level do we stop implicit capture
            Returns:
            transmission:  The fraction of neutrons that made it through
            """
            imp_input = implicit_capture
            Sig_t = Sig_a + Sig_s
            iSig_t = 1.0/Sig_t
            iSig_s = 1.0/(Sig_s + 1.0e-14)
            transmission = 0.0
            N = int(N)
            initial_weight = 1.0/N
            for i in range(N):
                if (isotropic):
                    mu = random.random()
                else:
                    mu = 1.0
                x = 0
                alive = 1
                weight = initial_weight
                while (alive):
                    if (weight < cutoff*initial_weight):
                        implicit_capture = False

                    if (implicit_capture):
                        #get distance to collision
                        l = -math.log(1-random.random())*iSig_s
                    else:
                        #get distance to collision
                        l = -math.log(1-random.random())*iSig_t
                    #make sure that l is not too large
                    if (mu > 0):
                        l = min([l,(3-x)/mu])
                    else:
                        l = min([l,-x/mu])
                    #move particle
                    x += l*mu
                    if (implicit_capture):
                        weight *= np.exp(-l*Sig_a)
                    #still in the slab?
                    if (math.fabs(x-thickness) < 1.0e-14):
                        transmission += weight
                        alive = 0
                    elif (x<= 1.0e-14):
                        alive = 0
                    else:
                        if (implicit_capture):
                            mu = random.uniform(-1,1)
                        else:
                            #scatter or absorb
                            if (random.random() < Sig_s*iSig_t):
                                #scatter, pick new mu
                                mu = random.uniform(-1,1)
                            else: #absorbed
                        alive = 0
                implicit_capture = imp_input
        return transmission
```

The example problem of a beam incident on a pure absorber should be exact using implicit capture and only one particle.

```
In [2]: N = 1
        Sigma_s = 0.0
        Sigma_a = 2.0
        thickness = 3
        transmission = slab_transmission(Sigma_s,Sigma_a, thickness,
                                         N, isotropic=False, implicit_capture=True)
        print("The fraction that made it through using implicit capture was",
              transmission, "with a percent error of",
              np.abs(transmission - np.exp(-6))/np.exp(-6)*100,"%")
        transmission = slab_transmission(Sigma_s,Sigma_a, thickness,
                                         N, isotropic=False, implicit_capture=False)
        print("The fraction that made it through using analog tracking was",
              transmission, "with a percent error of",
              np.abs(transmission - np.exp(-6))/np.exp(-6)*100,"%")
```

```
The fraction that made it through using
implicit capture was 0.00247875217667 with a percent error of 0.0 %
The fraction that made it through using
analog tracking was 0.0 with a percent error of 100.0 %
```

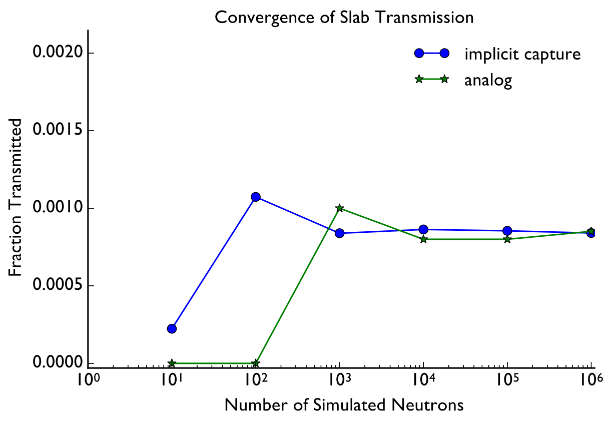
Indeed, implicit capture gave the correct answer For isotropic incident neutrons on the same slab, we expect implicit capture to be better than analog tracking, but not exact. In this test we will use 1000 simulated particles.

```
In [3]: true_sol = 0.000318257463690406467 27
        transmission = slab_transmission(Sigma_s,Sigma_a, thickness,
                                         N, isotropic=True, implicit_capture=True)
        print("The fraction that made it through using implicit capture was",
              transmission, "with a percent error of",
              np.abs(transmission - true_sol)/true_sol*100,"%")
        transmission = slab_transmission(Sigma_s,Sigma_a, thickness,
                                         N, isotropic=True, implicit_capture=False)
        print("The fraction that made it through using analog tracking was",
              transmission, "with a percent error of",
              np.abs(transmission - true_sol)/true_sol*100,"%")
```

```
The fraction that made it through using implicit capture was 0.000308930894796
     with a percent error of 2.93051065849 %
The fraction that made it through using analog tracking was 0.0
     with a percent error of 100.0 %
```

Implicit capture reduces our error from 100% to less than 5% with only 1000 particles. Previously, we needed about 2 million particles to get that accuracy.

The solution to the problem with scattering is below. In this figure we compare the results using the two approaches and different numbers of particles, up to 10^5.



These results indicate that implicit capture converges to the correct solution with many fewer simulated particles. A reasonable question is whether extra effort in performing implicit capture (because particles are not absorbed above the cutoff), is worth the cost.

22.1.1 A Figure of Merit

A way of measuring the benefit of a variance reduction technique is through the quantity called the figure of merit (FOM)

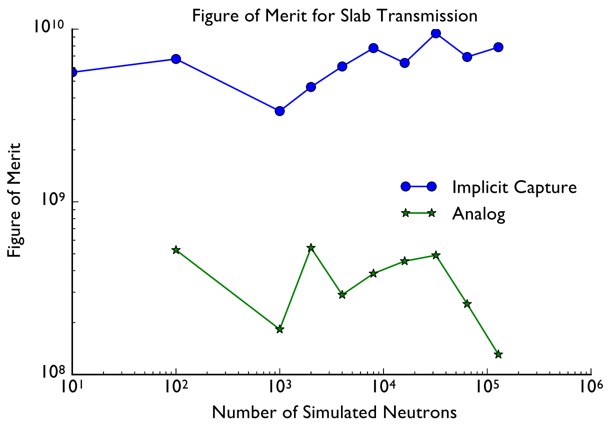
$$\mathrm{FOM} = \frac{1}{\sigma^2 T}.$$

It should be said that this is a figure of merit, not *the* figure of merit because others are possible. In the expression for FOM, σ^2 is the variance in an estimate and T is the time it takes to get the estimate. The benefit of the FOM is that it gives us a way to decide if the cost of a variance reduction technique is worth any increase in the computational time.

We can try this on our slab problem. We will run the problem 20 times with each number of particles, compute the time to solution, and the variance of the runs, and then plot the FOM. Remember that a larger number is better (lower variance and/or time). We expect that the implicit capture method should be better because it appears to give less error (see above) and the exponentials we evaluate should not be costly relative to the particle tracking.

```
In [4]: import time
        N_parts = [10,100,1000,2000,4000,8000,16000, 32000, 64000, 128000]
        Ntimes = 20
        solution_implicit = np.zeros((len(N_parts),Ntimes))
        solution_analog = np.zeros((len(N_parts),Ntimes))
        times_implicit = np.zeros(len(N_parts))
        times_analog = np.zeros(len(N_parts))
        var_implicit = np.zeros(len(N_parts))
        var_analog = np.zeros(len(N_parts))
        Sigma_s = 0.75
        Sigma_a = 2.0 - Sigma_s
        count = 0
        for N in N_parts:
            for replicate in range(Ntimes):
```

```
        tmp = time.clock()
        solution_implicit[count,replicate] = slab_transmission(Sigma_s,Sigma_a,
                                                               thickness, N,
                                                               isotropic=True,
                                                               implicit_capture=True,
                                                               cutoff=1e-2)
        times_implicit[count] += (time.clock()-tmp)/Ntimes
        tmp = time.clock()
        solution_analog[count,replicate] = slab_transmission(Sigma_s,Sigma_a,
                                                             thickness, N,
                                                             isotropic=True,
                                                             implicit_capture=False)
        times_analog[count] += (time.clock()-tmp)/Ntimes
    var_implicit[count] = np.std(solution_implicit[count,:])**2
    var_analog[count] = np.std(solution_analog[count,:])**2
    count += 1
```



In this example, we see that the FOM for implicit capture is about an order of magnitude larger than analog tracking. This means that implicit capture can get the same variance as analog tracking in one-tenth the time.

22.2 ESTIMATING SCALAR FLUX

Now that we have introduced our first variance reduction technique, we will discuss scalar flux estimators before moving on to other techniques. The first that we will consider is the collision estimator.

22.2.1 Collision Estimators

Consider the reaction rate in a volume, defined by

$$R = \int_V dV\ \Sigma_t(\mathbf{r})\phi(\mathbf{r}).$$

If inside the region the cross-section is constant, we can compute the average scalar flux via the relation

$$\bar{\phi} = \frac{1}{V}\int_V dV\,\phi(\mathbf{r}) = \frac{R}{V\,\Sigma_t}.$$

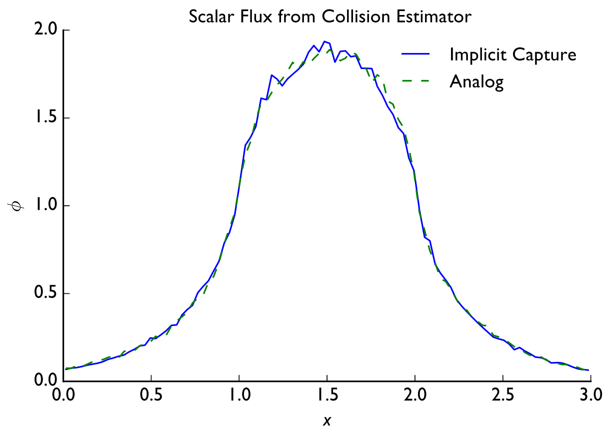
Therefore, if we sum, or tally, the weight from each collision inside the volume and divide that count by the total cross-section, we get an estimate of the scalar flux.

Notice, however, that we cannot use this in voids. We can, however, use other processes to estimate the scalar flux. For example, we could compute the scattering rate and divide by Σ_s, for implicit capture this is what we will do.

The slab problem from above will be modified for this purpose. We will introduce a mesh onto the problem and count the reactions in each mesh cell. Also, instead of a source on the boundary we will add a volumetric source to the problem. The source will be uniform between a and b. Therefore, we need to sample a position of the neutron's birth as well as an angle in $\mu \in [-1, 1]$.

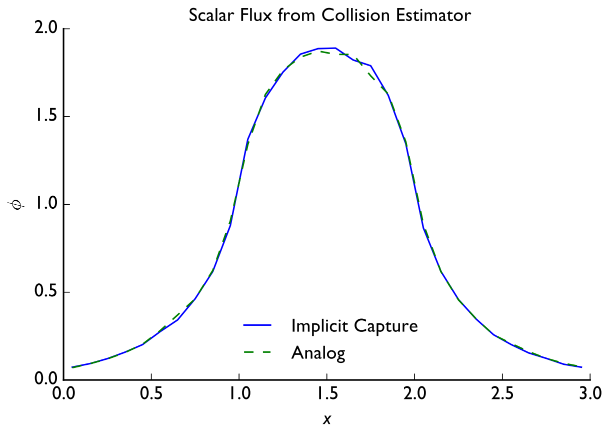
We hold off showing the code for the collision estimator and all of the other features we discuss until the end of the chapter in Section 22.4. This is done to avoid re-listing the code repeatedly with only minor changes.

If we run a calculation with the collision estimator on a slab of thickness 3 with $N = 1000$, $\Sigma_s = 2.0$, $\Sigma_a = 0.5$, and the source between 1 to 2, we get the following results for analog tracking and implicit capture with 100 mesh cells.

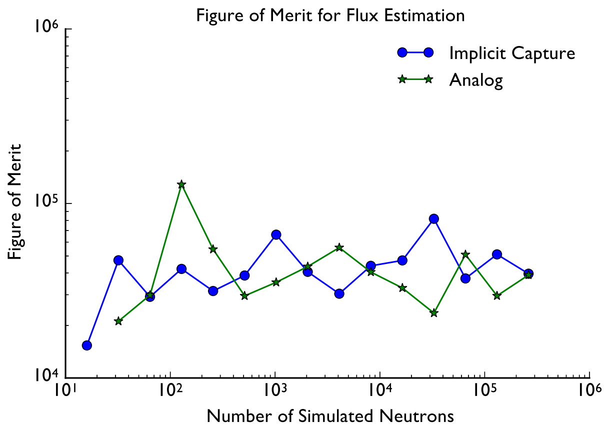


Note that using implicit capture, the collision estimator uses only scattering collisions.

If we use fewer mesh cells, 30 in this case, the answer looks better because we are averaging the scalar flux over a larger volume.



It is hard to tell which of the two methods (implicit capture or analog) is doing better. The figure of merit can help, but we need to select what quantity to measure the variance in. We could consider the variance in the scalar flux estimate in any of the cells as the quantity to estimate the variance in. In the following figure we look at the variance in the scalar flux at the left edge.



On this problem there are few collisions near the edge of the problem, so we do not see a large difference in the FOM. We expect implicit capture to outperform analog tracking because analog tracking will kill particles before they reach the edge of the problem. Implicit capture allows more particles to get to the edge, and therefore the estimation of the scalar flux will be better. However, we need to use something other than collisions to estimate the scalar flux here.

22.2.2 Track-Length Estimators

Another type of estimator for the scalar flux uses the definition of the scalar flux to estimate it. Recall that the scalar flux is the rate-density at which neutrons generate track length.

Therefore, for a given cell, every time a neutron moves inside it we sum the weight of the neutron times its path length in the cell. We then divide this by the volume of the region. We can write this in equation form as

$$\bar{\phi} = \frac{1}{V} \sum_{\text{neutrons}} \text{weight} \times \text{path length}.$$

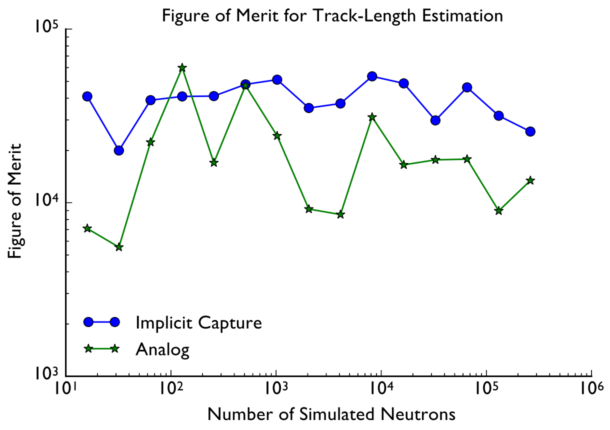
For implicit capture, the weight is changing while the neutron moves in the region. Therefore, we integrate the weight over the track to decide the contribution. A neutron traveling a distance s inside a region will contribute

$$\text{contribution} = \int_0^s ds' w_0 e^{-\Sigma_a s'} = \frac{1}{\Sigma_a} w_0 (1 - e^{-\Sigma_a s}),$$

where w_0 is the initial weight before moving the track-length s.

To implement this estimator we will reformulate how we do our tracking. We will make our method work by checking the distance to collision against the distance to the edge of a cell. Whichever distance is shorter, that event occurs: either the neutron has a collision or it moves to the next cell and a new distance to collision is sampled. This means that the neutrons will step through the problem cell by cell. This will slow down the code, because now the particles can only take steps limited by the width of the mesh cells.

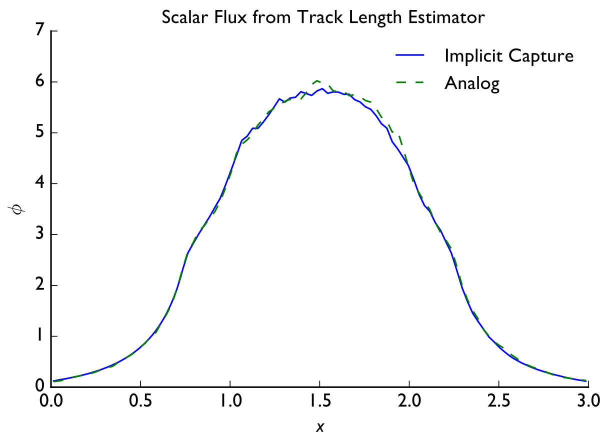
Comparing the figure of merit of the track-length estimator to analog Monte Carlo we see that, once again, on the edge of the problem the implicit capture result has a higher FOM.



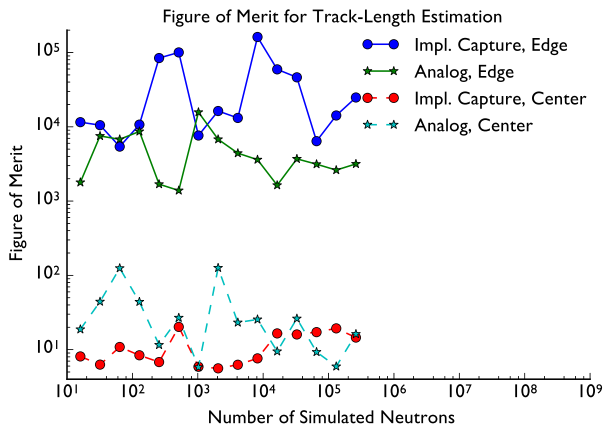
Moreover, the FOM for track-length estimation is higher than that for the collision estimator. In general, it can be beneficial to use both estimators for the scalar flux. By looking at discrepancies between the two methods we can tell where the statistical variance in the problem might be high.

22.2.3 Geometric Dependence of the FOM

As alluded to above, the Figure of Merit can change based on where we look in the problem. In particular, when neutrons have to cross an absorber to get to a particular cell, few of them will do that with analog tracking. Moreover, in a region of the problem where there are many neutrons, analog tracking may have low variance. We can demonstrate this by defining a problem that has a slab of thickness 3 cm $\Sigma_s = 1\ \text{cm}^{-1}$ everywhere and $\Sigma_a = 0.1\ \text{cm}^{-1}$ between $x = 1$ and $x = 2$, and $\Sigma_a = 2\ \text{cm}^{-1}$ otherwise. This problem has a strong absorber at the edges and a scatterer in the middle. We will source neutrons in the middle between $x = 1$ and $x = 2$. The following figure shows the estimate for the scalar flux in this problem using 10^4 simulated particles.



The figure of merit for this problem is quite different in the middle versus the edge.



In the middle of the problem there are many particles because that is where the source is. Furthermore, the center region is scattering dominated so the difference between analog and implicit capture is small in terms of the estimate. At the edge, the neutrons have to cross at least two mean-free paths; this results in implicit capture having a higher FOM. This result tells us that depending on what one cares more about, the Monte Carlo methods chosen may have to change.

22.3 STRATIFIED SAMPLING

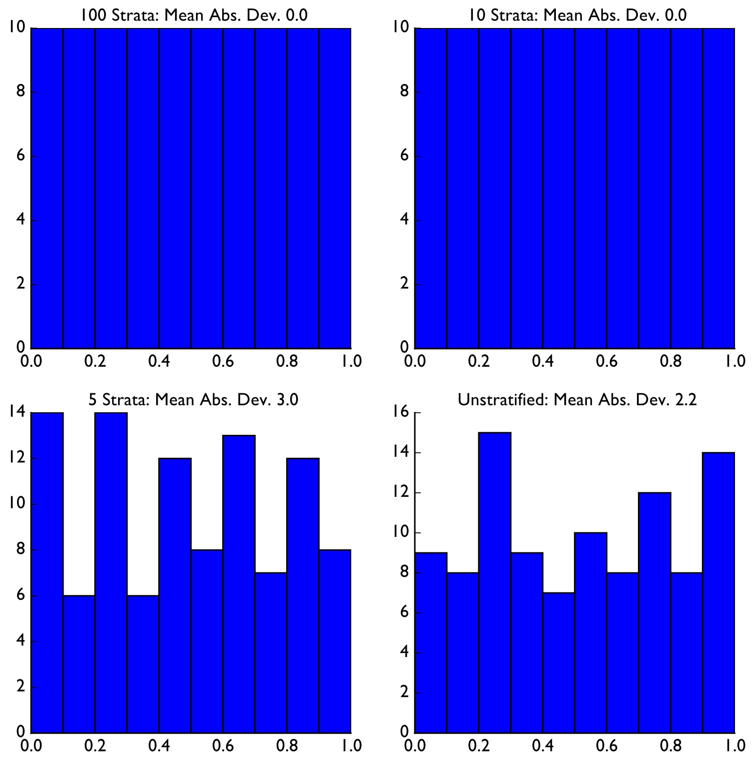
The idea behind stratified sampling is to control the randomness in the simulation. We want to use random numbers to simulate neutron interactions, but there is no guarantee that random numbers will not be close together. Stratified sampling is a way to spread out the numbers.

It is easiest to think about stratification in terms of a single random variable uniformly distributed between 0 and 1. There are several possible formulations, but the most straightforward to use divides the range between 0 and 1 into S bins of equal size. We then pick a bin at random by generating a random integer between 0 and $S-1$. Then inside that bin we randomly pick a location. If we perform this sampling so that each bin has the same number of samples, we expect that random samples will do a better job of filling the space between 0 and 1 than simple random sampling.

For this example, the code to produce the stratified samples is straightforward.

```
In [5]:  def strat_sample(N,S):
             N = N + (N % S)
             assert(N%S == 0 )
             dS = 1.0/S
             bins = np.zeros(N,dtype=int)
             count = 0
             for i in range(N//S):
                 bins[count:count+S] = np.random.permutation(S)
                 count += S
             place_in_bin = np.random.uniform(-0.5*dS,0.5*dS,N) + (bins+0.5)*dS
             return place_in_bin
```

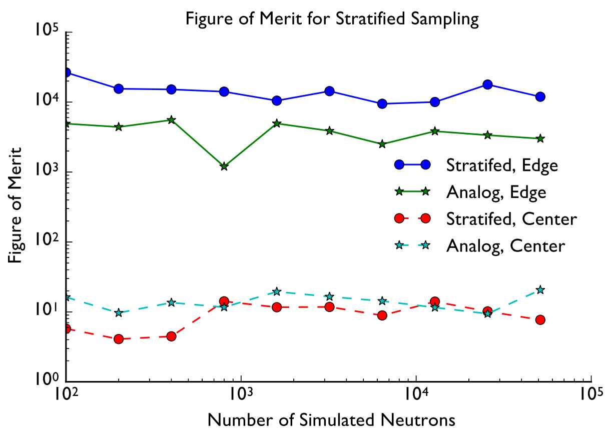
If we run an example with 100 samples and different numbers of strata, the histogram of the samples that we get is



Each of these histograms has 10 bars in the range from 0 to 1. Using 100 or 10 strata has the samples uniformly distributed among these bars. The mean absolute deviation from the mean for each of the bars is reported in the figure; this is a measure of how far away from 10 each of the bar heights are. When we have fewer bins, or do not use stratification, the samples are not uniformly distributed and the average deviation from 10 is larger. This means that we sample some parts of space more than others. The result for a particle transport calculation means that there could be more variance if we use these unstratified samples in our Monte Carlo code. Note that inside of the bins in this figure, the samples are randomly chosen. This means we still have randomness, it is just controlled to live in a particular bin.

Using stratified sampling we can decrease the variance in our solution. This can be most effective when the number of bins is equal to the number of samples. One can show that this is the best case scenario for filling out the range because the farthest apart two samples can be is twice the bin width, and maximizing the number of bins, minimizes the bin width.

Stratified sampling applied to the problem with a scatterer in the middle and an absorber on the edges, results in the following FOM. The stratified results include the implicit capture while the analog do not. In this case we used stratified sampling to choose the location where the neutrons are born in the source region. This is will make the neutron birth locations more uniform.

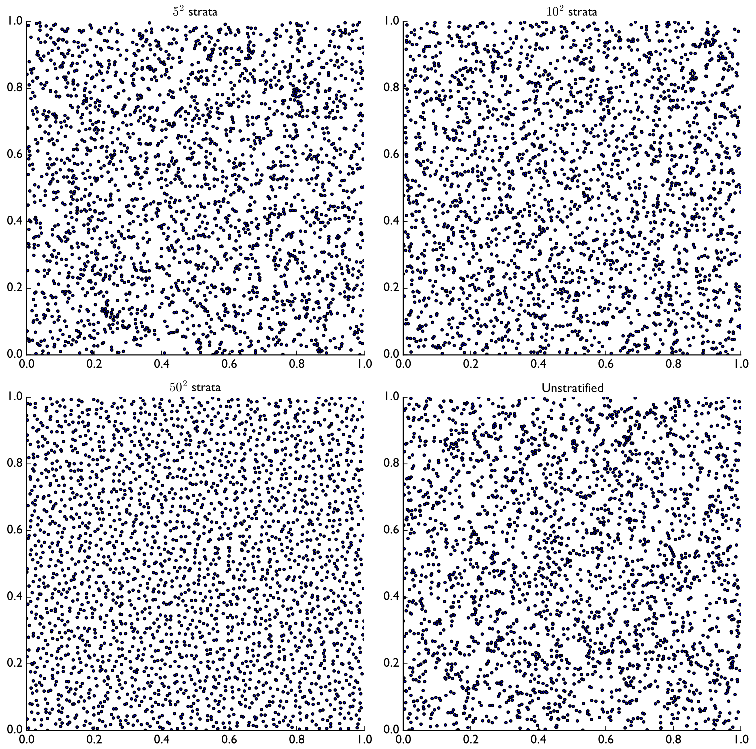


From these results, we can see that stratification provides an improvement in the FOM over previous sampling strategies, especially in the center. The stratification makes the neutrons be born more uniformly in the center of the problem. This makes the implicit capture results competitive with analog tracking. For the particles at the edge, the stratification also provided a benefit over the results from implicit capture with standard sampling.

We can generalize stratified sampling to multiple dimensions, though the number of strata increases as S^D where D is the number of dimensions, if the number of strata in each dimension is S. This can make it difficult to match the number of samples to the number of strata. The following code gives a stratification in two-dimensions. It will increase the number of samples to match the desired number of strata, if needed. It also allows the number of strata in each dimension to differ.

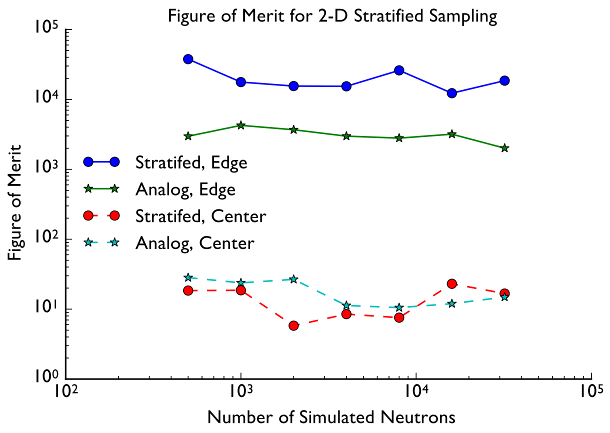
```
In [6]: def strat_sample_2D(N,S1,S2):
            """Create N samples in S1*S2 strata.
            Inputs:
            N:             number of samples
            S1:            number of strata in dimension 1
            S2:            number of strata in dimension 2
            Returns:
            samples:        N by 2 numpy vector containing the samples
            """
            #number of bins is S1*S2
            bins = S1*S2
            #make sure we have enough points
            if (N<bins):
                N = bins
            N -= (N % bins)
            Num_per_bin = N//bins
            assert(N % bins == 0)
            samples = np.zeros((N,2))
            count = 0;
            for bin_x in range(S1):
                for bin_y in range(S2):
                    for i in range(Num_per_bin):
                        center = (bin_x/S1 + 0.5/S1,bin_y/S2 + 0.5/S2)
                        samples[count,0] = center[0] + random.uniform(-0.5,0.5)/S1
                        samples[count,1] = center[1] + random.uniform(-0.5,0.5)/S2
                        count += 1

            return samples
```

Sampling a 2-D space with 5, 10, and 50 strata in each dimension (for a total of $5^2 = 25$, $10^2 = 100$, and $25^2 = 2500$ strata), all with 2500 samples, are compared with unstratified sampling in the following figure.



In this figures we see the benefit of stratification in 2-D. The 50^2 strata case fills the space nicely without the clumps that can be seen in the unstratified example. As such the 5^2 and the 10^2 strata cases do not appear to be much of an improvement over the unstratified case.

We can apply 2-D stratification by using it to pick the initial position and μ for the source particles in the slab. This should decrease the variance in our calculation when we increase the number of neutrons sampled to have a large number of strata. On the problem with a scatter in the middle and absorber on the edge, the FOM demonstrates that 2-D stratification does best when the number of simulated neutrons is high.



In these results, the FOM for stratified sampling is consistently higher at the edge of the problem when using analog tracking compared with analog tracking. It does appear, however, that in the center there is less benefit due to the large number of neutrons that are born here.

22.4 COMPLETE MONTE CARLO CODE FOR SLABS

The following code listing has all of the features discussed in this chapter.

```
def slab_source(Nx,Sig_s,Sig_a,thickness,a,b,N,Q,
                implicit_capture = True,
                cutoff = 1.0e-3,
                stratified = [1,1]):
            """Compute the fraction of neutrons that leak through a slab
            Inputs:
            Nx:         The number of grid points
            Sig_s:      The scattering macroscopic x-section
            Sig_a:      The absorption macroscopic x-section
            thickness: Width of the slab
            a,b:        Endpoints of Source
            N:          Number of neutrons to simulate
            implicit_capture: Do we run implicit capture
            cutoff:     At what level do we stop implicit capture
            stratified: Use stratified sampling in space and angle
                         Specify a list of length two with the number of
                         strata in each dimension; default [1,1] for unstratified

            Returns:
            transmission:  The fraction of neutrons that made it through
            scalar_flux:   The scalar flux in each of the Nx cells
            scalar_flux_tl:   The scalar flux in each of the Nx cells
                              from track length estimator
            X:             The value of the cell centers in the mesh
            """
              imp_input = implicit_capture
              dx = thickness/Nx
              X = np.linspace(dx*0.5, thickness - 0.5*dx,Nx)
              scalar_flux = np.zeros(Nx)
              scalar_flux_tl = np.zeros(Nx)
              assert (Sig_s.size == Nx) and (Sig_a.size == Nx)
```

```
Sig_t = Sig_a + Sig_s
iSig_t = 1.0/Sig_t
iSig_s = 1.0/(Sig_s+1.0e-14)
iSig_a = 1.0/(Sig_a+1.0e-14)
leak_left = 0.0
leak_right = 0
N = int(N)
#make a vector of the initial positions and mus
samples = strat_sample_2D(N,stratified[0],stratified[1])
xs = samples[:,0]*(b-a) + a #adjust to bounds of source
mus = (samples[:,1]-0.5)*2 #shift to range -1 to 1
N = int(xs.size)
#the initial weight does not change
init_weight = Q*thickness/N
for i in range(N):
    mu = mus[i]
    x = xs[i]
    alive = 1
    weight = init_weight
    #which cell am I in
    cell = int(x/dx)
    implicit_capture = imp_input
    while (alive):
        if (weight < cutoff*init_weight):
            implicit_capture = False
        if (implicit_capture):
            l = -math.log(1-random.random())*iSig_s[cell]
        else:
            #get distance to collision
            l = -math.log(1-random.random())*iSig_t[cell]
        #compare distance to collision to distance to cell edge
        distance_to_edge = ((mu > 0.0)*( (cell+1)*dx - x) +
                            (mu<0.0)*( x - cell*dx) + 1.0e-8)/math.fabs(mu)
        if (distance_to_edge < l):
            l = distance_to_edge
            collide = 0
        else:
            collide = 1
        x += l*mu #move particle
        #score track length tally
        if (implicit_capture):
            scalar_flux_tl[cell] += weight*(1.0 -
                                    math.exp(-l*Sig_a[cell]))*iSig_a[cell]
        else:
            scalar_flux_tl[cell] += weight*l
        if (implicit_capture):
            weight *= math.exp(-l*Sig_a[cell])
        #still in the slab?
        if (math.fabs(x-thickness) < 1.0e-14) or (x > thickness):
            leak_right += weight
            alive = 0
        elif (x<= 1.0e-14):
            alive = 0
            leak_left += weight
        else:
            cell= int(x/dx) #compute cell particle collision is in
            if (implicit_capture):
                if (collide):
                    mu = random.uniform(-1,1)
                scalar_flux[cell] += weight*iSig_s[cell]/dx
            else: #scatter or absorb
                scalar_flux[cell] += weight*iSig_t[cell]/dx
                if (collide) and (random.random() < Sig_s[cell]*iSig_t[cell]):
                    #scatter, pick new mu
                    mu = random.uniform(-1,1)
                elif (collide): #absorbed
```

```
                        alive = 0
        return leak_left,leak_right, scalar_flux, scalar_flux_tl/dx, X, N
```

CODA

In this chapter we have expanded our Monte Carlo toolkit to estimate the scalar flux and to make better estimates via Monte Carlo. These techniques, along with those in the previous chapter, give us all the tools we need to solve source-driven neutron transport problems via Monte Carlo. There is still an important class of problems that we do not have to tools to solve, yet: k-eigenvalue problems. We will discuss Monte Carlo techniques for these problems in the next chapter.

PROBLEMS

Short Exercises

22.1. Repeat Short Exercise 21.3 using the 2-D stratified sampling to pick the proposal points with the number of strata being 5^2, 10^2, 50^2, 100^2, and 200^2. Only keep the accepted proposed points (this will mean that you get fewer samples than the number of strata). How do these results compare to standard rejection sampling?

Programming Projects

The problems below ask you to modify the iron-shielded reactor example from the previous chapter to include the methods discussed in this chapter.

1. Iron-Shielded Reactor: Implicit Capture

Modify the iron shielding code to include implicit capture. Compare the analog to the implicit capture results with $N = 10^2, 10^3, 10^4$, and 10^5. Compute the figure of merit for the average transmitted energy per neutron for these values of N.

2. Iron-Shielded Reactor: Scalar Flux

Inside the shield compute the scalar flux of neutrons in the range 100 keV to 1 MeV and the scalar flux of neutrons above 1 MeV. Compare the collision estimator and track-length estimator for several different mesh resolutions.