A trivial Monte Carlo eigenvalue calculation

James Paul Holloway

May 12, 2008

1 Introduction

Often, in order to understand a technique, we should reduce it to its simplest possible application, for if we can't understand it there, we can't understand it anywhere. These notes outline the Monte Carlo approach to k-eigenvalue calculations in the simplest possible case: an infinite homogeneous medium. Of course, the value of k_{eff} for such a system is known. Using Σ to denote macroscopic cross sections, and subscripts f and γ to denote fission and radiative capture, we have

$$k_{\text{eff}} = k_\infty = \frac{\nu\Sigma_f}{\Sigma_a} \tag{1}$$

where $\Sigma_a = \Sigma_f + \Sigma_\gamma$.

The "transport equation", such as it is in our simple case, is

$$\Sigma_t\phi = \nu\Sigma_f\phi + \Sigma_s\phi\,, \tag{2}$$

where Σ_s is the scattering cross section and $\Sigma_t = \Sigma_a + \Sigma_s$. The k eigenvalue problem arises by scaling ν in such a way that this homogeneous equation supports a non-zero solution, so we consider

$$\Sigma_t\phi = \frac{\nu\Sigma_f}{k}\phi + \Sigma_s\phi\,. \tag{3}$$

Note that in this infinite homogeneous medium, the various "operators" of transport theory are just multiplication by various constants.

2 Inverse power iteration

Inverse power iteration for k comes from isolating the $1/k$ scaled fission source

$$\Sigma_a\phi = \frac{\nu\Sigma_f}{k}\phi\,. \tag{4}$$

and "inverting" the transport operator on the left

$$\phi = \frac{1}{\Sigma_a}\left[\frac{\nu\Sigma_f}{k}\phi\right] = \frac{1}{k}\frac{1}{\Sigma_a}\left[Q\right]. \tag{5}$$

Note that if the fission source $Q = \nu\Sigma_f\phi$ is known, then evaluating the expression on the right simply requires solving a fixed source transport problem, a task to which Monte Carlo is well suited.

We can apply the fission operator (multiply by $\nu\Sigma_f$) to Eq. 5 and arrive at

$$Q = \frac{1}{k}\frac{\nu\Sigma_f}{\Sigma_a}\left[Q\right] \tag{6}$$

This is identical to Martin's Eq. 2 in his 6/23/06 document, where his M corresponds to my multiplication by $\nu\Sigma_f$, and his A^{-1} to my division by Σ_a. A Monte Carlo algorithm can evaluate A^{-1}, because that just means "solve a fixed source problem," and it can tally the fission source that results from that solution in order to compute $MA^{-1}Q$. A Monte Carlo algorithm can also divide by Σ_a, as we will see, and such an algorithm can tally the effect of the multiplication by $\nu\Sigma_f$.

The inverse power iteration is then based on the iteration

$$Q^{n+1} = \frac{1}{k^n}\frac{\nu\Sigma_f}{\Sigma_a}\left[Q^n\right]. \tag{7}$$

To compute an update for k. in general we would take the inner product of Eq. 6 with some function U, converting it into a scalar equality, and solve for k. Because we have only scalar operators it does not matter what "U" would be in this case, and we simply rearrange Eq. 6 into the form

$$k = \frac{\nu\Sigma_f\Sigma_a^{-1}Q}{Q}. \tag{8}$$

Then, we can compute

$$k^{n+1} = \frac{\nu\Sigma_f\Sigma_a^{-1}Q^n}{Q^n}. \tag{9}$$

This can be simplified because, by Eq. 7, $\nu\Sigma_f\Sigma_a^{-1}Q^n = k^nQ^{n+1}$, and so

$$k^{n+1} = k^n\frac{Q^{n+1}}{Q^n}. \tag{10}$$

The complete iteration is thus

$$Q^{n+1} = \frac{1}{k^n}\frac{\nu\Sigma_f}{\Sigma_a}\left[Q^n\right] \tag{11}$$

$$k^{n+1} = k^n\frac{Q^{n+1}}{Q^n}. \tag{12}$$

2.1 Convergence of the iteration

It is easy to see that if $k_n = k_\infty$ then $Q^{n+1} = Q^n$ in Eq. 11 and then that $k^{n+1} = k^n = k_\infty$ in Eq. 12. Thus, if the iteration converges, it converges to the correct result for k, and Q is of course arbitrary.

Indeed, for any Q_0, k_0 we have

$$Q^1 = \frac{1}{k^0}\frac{\nu\Sigma_f}{\Sigma_a}Q^0 = \frac{k_\infty}{k^0}Q^0 \tag{13}$$

$$k^1 = k^0\frac{Q^1}{Q^0} = k^0\frac{k_\infty}{k^0}\frac{Q^0}{Q^0} = k_\infty\,. \tag{14}$$

In other words, the iteration will converge in one step! But what would a Monte Carlo implementation look like?

2.2 Monte Carlo implementation

At the start of each iteration we have an estimate for Q^n and k^n. In this infinite dimensional context Q^n is just the number of fission neutrons that were released during iteration n. To evaluate Q^{n+1} we must compute the number of fission neutrons that are caused to be released by the known source Q^n/k_n. But we must interpret this statement carefully; Eq. 11 says to take neutrons from the source Q^n/k^n, follow them until they are absorbed, and then to compute the fission source that would result from the implied flux. The neutrons from *this* source are *not* transported as part of Eq. 11. We can equivalently say that we transport particles from the source Q_n, and when they are absorbed we compute the fission neutrons that they would release *using a mean number of secondaries* $\nu^n = \nu/k^n$; we will use this latter view.

To solve Eq. 11 we select a number N of histories to run per iteration (particles per cycle they say in the code world). In our problem, all of our neutrons will be absorbed eventually (they have nowhere else to go). When they are absorbed, the number, m^n, of fission neutrons released on average is ν/k^n times the probability of fission given that an absorption has occurred, or $m^n = (\nu/k^n)(\Sigma_f/\Sigma_a)$. For reasons that have to do with the needs of space dependent cases, we need a pdf to sample that has this mean. The one used is $p(s|m^n)$, where

$$p(s|m) = \begin{cases} 1-(m-\lfloor m\rfloor) & s = \lfloor m\rfloor \\ (m-\lfloor m\rfloor) & s = \lfloor m\rfloor + 1 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where s is the integer number of particles to be released in fission, and $\lfloor m\rfloor$ is the nearest integer less than or equal to m (the floor function applied to m). A direct calculation will show that $\langle s\rangle = m$ as desired. To sample $p(s|m)$ for s, the expression

$$s = \lfloor m + \xi\rfloor \tag{16}$$

is used, where $0 \leq \xi < 1$ is a uniformly distributed random number.

Note that the mean number of particles released per fission, m^n, and the pdf used to sample it, change from one iteration to the next because the mean number of secondaries $\nu^n = \nu/k_n$ varies from one iteration to the next.

2.3 The Monte Carlo algorithm

For each iteration (cycle) we:

1. Initialize a count of fission neutrons $\tilde{Q}^{n+1}$ to zero
2. compute
$$m^n = \frac{\nu\Sigma_f}{\Sigma_a k^n} \tag{17}$$
3. Then for each history in the iteration we sample $p(s|m^n)$, and add this sample $\tilde{Q}^{n+1}$.
4. At the end of the N histories, we can compute the tally
$$Q^{n+1} = \frac{\tilde{Q}^{n+1}}{N} Q^n\,. \tag{18}$$
Note in this that we compute the number of fission neutrons released per source particle (divide by the number of histories), times the magnitude of the previous source.
5. Finally, we can compute the new k estimate
$$k^{n+1} = k^n \frac{Q^n \tilde{Q}^{n+1}/N}{Q^n} = k^n \frac{\tilde{Q}^{n+1}}{N} = k^n \frac{\text{number of fission neutrons banked}}{\text{particles per cycle}} \tag{19}$$

Note that the final two steps implement Eq. 12. We don't need Eq. 11 in this infinite medium case.

The algorithm can be compressed into, for each cycle,

1. Initialize a count of fission neutrons $\tilde{Q}^{n+1}$ to zero
2. For each history $\tilde{Q}^{n+1} = \tilde{Q}^{n+1} + \lfloor \nu\Sigma_f/(\Sigma_a k^n) + \xi \rfloor$
3. At the end of the N histories, we can compute the new k estimate
$$k^{n+1} = k^n \frac{\tilde{Q}^{n+1}}{N} = k^n \frac{\text{number of fission neutrons banked}}{\text{particles per cycle}} \tag{20}$$

3 The k_∞ code

The Matlab code below will implement the k calculation. It contains two methods, method 2 is what is described above. Method 1 only contributes particles to the "fission bank" when there is a fission. Because fewer particles contribute, it has a higher variance. Here is a sample run:

```
>> sigF = 1; sigG = 1; sigS = 0; nu = 2.5;
>> histPerCycle = 1000; cycles = 100;
>> k1 = kInfinity(histPerCycle, cycles, sigS, sigG, sigF, nu, 1);
>> k2 = kInfinity(histPerCycle, cycles, sigS, sigG, sigF, nu, 2);
>>
```

Figure 1 shows the k estimates over the first 100 cycles, using 1000 histories per cycle. Note that the results are as good as they are going to be in about one cycle, because the deterministic iteration needs only one iteration in this trivial case. After that. it's just fluctuations.

Here is the code to calculate k_∞ by Monte Carlo.

```
function [kHistory] = kInfinity(historiesPerCycle, cycles, SigmaS, SigmaG, SigmaF,...
                                nu, method)
% [kHistory] = kInfinity(historiesPerCycle, cycles, SigmaS, SigmaG, SigmaF, nu)
%
% Estimate k_\inf using a Monte Carlo
% historiesPerCycle - number of histories to sample from the
%                     fission source during each cycle
% cycles - number of cycles to run
% SigmaS, SigmaG, SigmaF - Macroscopic cross sections for
%                          scatter, radiative capture, fission
% nu - mean number of fission neutrons born per fission
% method - a flag to select a method to use
%
% kHistory - the history of k values
%
% There are no dead cycles because the system is an infinite
% medium, so we don't need to get a critical fission source
% shape set up
%
% $Id$

validateData(historiesPerCycle, cycles, SigmaS, SigmaG, SigmaF, nu, method)

% ============================================================
% Initialization of data
% ============================================================

% Set up number of particles in previous bank.  We don't need
% to store phase space data for individual events becasue we
% don't care where the particles were created
fissionSource = 1;  % initial source

% Compute the absorption and total cross sections
SigmaA = SigmaG + SigmaF;
SigmaT = SigmaS + SigmaA;

% Initialize the history of k values
kHistory = zeros(1,cycles+1);
kHistory(1) = rand()+0.0001;  % Not zero, but don't care what

for i = 2:length(kHistory)
```

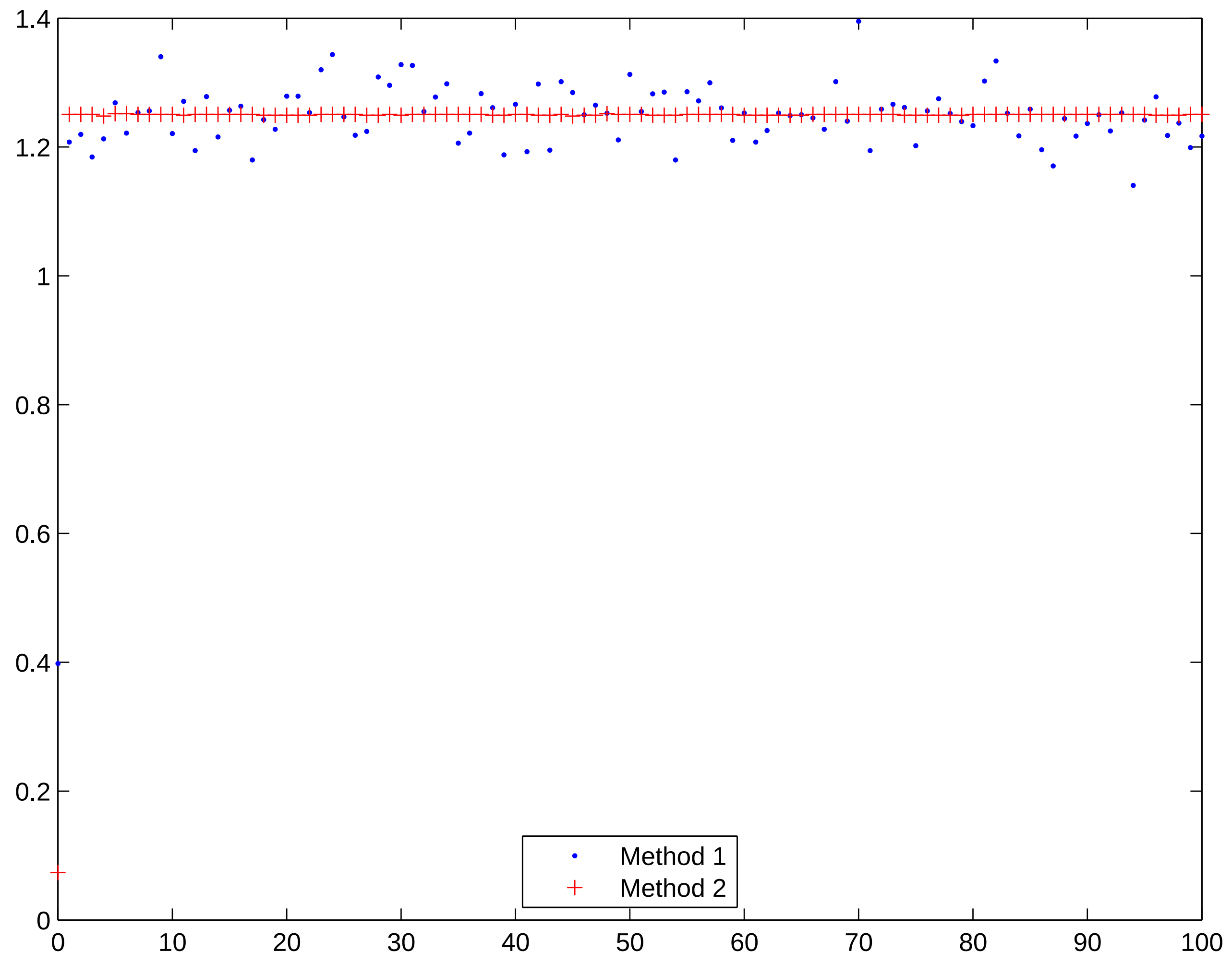

Figure 1: k estimates computed using two methods, showing 100 cycles and using 1000 histories per cycle. The exact value is $k_\infty = 2.5 \times 1/(1+1) = 1.25$. Note that Method 2 is lower variance. The spread in Method 1 will reduce with more particles per history.

```
   kHistory(i) = ...
         computeK(kHistory(i-1), historiesPerCycle, SigmaS, SigmaG, SigmaF, nu, SigmaT, SigmaA
   % disp([num2str(i-1) ': ' num2str(kHistory(i))])
   if (kHistory(i) <= 0)
       disp(['k negative or zero on cycle ' num2str(i-1)])
       break
   end
end

% End of kInfinity

% ============================================================
% ===>> computeK <<==========================================
% ============================================================
function k = computeK(kold, histories, SigmaS, SigmaG, SigmaF, nu, SigmaT, SigmaA, method)
% Dispatch the k computation to one of several methods.  Oh, for a decent
% OOP system!

if method == 1
  k = kMethod1(kold, histories, SigmaS, SigmaG, SigmaF, nu, SigmaT, SigmaA, method);
elseif method == 2
  k = kMethod2(kold, histories, SigmaS, SigmaG, SigmaF, nu, SigmaT, SigmaA, method);
else
  error(['Method number ', num2str(method), ' is invalid'])
end

% ============================================================
% ===>> kMethod1 <<==========================================
% ============================================================
function [k, newFissionSource] = kMethod1(kold, histories, SigmaS, SigmaG, SigmaF, ...
                                          nu, SigmaT, SigmaA, method)

% Sample histories particles from the fissionSource.  Note that
% since we are infinite dimensional the old fissionSource is
% not really used for sampling, becasue we don't need any phase
% space data.  Each particle is followed until it is absorbed,
% at which point we either add particles to the newFissionSource
% or not, depending on the probability of fission capture.
newFissionSource = 0;

for i = 1:histories
    % Track particle to absorbtion - nothing really to do

    % Particles has been absorbed, was it fission or radiative capture?
    interaction = rand();
    if(interaction * SigmaA > SigmaG)
        % it was a fission
        newParticles = floor(nu/kold + rand());
        newFissionSource = newFissionSource + newParticles;
    else
        % it was an absorbtion
    end
end
if(newFissionSource == 0)
    disp('Warning: there were no fissions on this cycle')
end
k = kold * newFissionSource / histories;

% ============================================================
% ===>> kMethod2 <<===========================================
% ============================================================
function [k, newFissionSource] = kMethod2(kold, histories, SigmaS, SigmaG, SigmaF, ...
                                          nu, SigmaT, SigmaA, method)

% Sample histories particles from the fissionSource.  Note that
% since we are infinite dimensional the old fissionSource is
% not really used for sampling, becasue we don't need any phase
% space data.  Each particle is followed until it is absorbed,
% at which point we always add particles to the newFissionSource
newFissionSource = 0;

for i = 1:histories
    % Track particle to absorbtion - nothing really to do

    interaction = rand();
    newParticles = floor(nu/kold * SigmaF/SigmaA + rand());
    newFissionSource = newFissionSource + newParticles;
end
if(newFissionSource == 0)
    disp('Warning: there were no fissions on this cycle')
end
k = kold * newFissionSource / histories;

% ============================================================
% ===>> validateData <<=======================================
% ============================================================
function validateData(historiesPerCycle, cycles, SigmaS, SigmaG, SigmaF, nu, method)

if(historiesPerCycle < 1)
    error('historiesPerCycle must be positive')
end
if(cycles < 0)
    error('cycles must be non-negative')
end
```

```
if(SigmaS < 0.0)
    error('SigmaS must be non-negative')
end
if(SigmaG < 0.0)
    error('SigmaS must be non-negative')
end
if(SigmaF < 0.0)
    error('SigmaS must be non-negative')
end
if(nu < 0.0)
    error('nu must be non-negative')
end
```