DEMONSTRATION OF A ZERO-VARIANCE BASED SCHEME FOR VARIANCE REDUCTION TO A MINI-CORE MONTE CARLO CALCULATION

Stavros Christoforou
Kirinthou 17, 34100, Chalkida, Greece
stavros.christoforou@gmail.com

J. Eduard Hoogenboom*
Department of Applied Sciences
Delft University of Technology
Mekelweg 15, 2629 JB, Delft, The Netherlands
j.e.hoogenboom@tudelft.nl

ABSTRACT

A zero-variance based scheme is implemented and tested in the MCNP5 Monte Carlo code. The scheme is applied to a mini-core reactor using the adjoint function obtained from a deterministic calculation for biasing the transport kernels. It is demonstrated that the variance of the $k_{eff}$ estimate is halved compared to a standard criticality calculation. In addition, the biasing does not affect source distribution convergence of the system. However, since the code lacked optimisations for speed, we were not able to demonstrate an appropriate increase in the efficiency of the calculation, because of the higher CPU time cost.

Key Words: Monte Carlo, biasing, adjoint, criticality

1. INTRODUCTION

Zero-variance schemes for Monte Carlo simulations have been a long-standing target of research in the field. Ever since variance reduction schemes became more common in Monte Carlo, researchers have attempted to reduce the variance of a calculation to zero. An important factor in this effort has been the adoption of the adjoint function as a measure of the importance of a region in the spatial, angular and energy domains of the calculation. This way, in source-detector problems, the adjoint function can be used as a biasing function for the calculation, in order to direct the simulated particles towards the most important regions of the system and reduce the variance of the estimate in those regions. In fact, it has been proven that using the appropriate biasing, one can get the variance of the Monte Carlo estimate down to zero, at least in theory. However, it is not straightforward how to reduce the variance in a criticality calculation, where the reactor serves both as the source and “detector” of particles simulated, therefore having a global problem that cannot easily be localised.

In previous publications we offered a solution by devising a zero-variance scheme for criticality calculations [1, 2]. Treating the criticality problem as a source-detector one, we showed that,

*Corresponding author
when using a collision estimator and biasing the transport kernels by the adjoint function, with an appropriate manipulation of particle weights, it is possible to reach an estimate with zero variance. A simplified two-direction model was used for demonstration, proving that when biasing using analytically calculated source and adjoint functions, the only limit to the reduction of variance of the estimate obtained from a single neutron generation is the Russian roulette threshold. This threshold can in theory get down to zero, therefore reducing the variance to zero, but in practice it is dictated by computational limits. It was also shown that zero variance cannot be obtained in practice, due to the need for renormalization and averaging of particle weights at the beginning of each successive generation of neutrons. However, a reduction in variance can occur even using approximate adjoint functions that have been obtained computationally.

It is now important to see if the scheme can be used in everyday Monte Carlo criticality calculations, rather than just as a theoretical possibility. In this paper, we implemented and then tested the scheme in the widely-used MCNP5 Monte Carlo code [3] by running a more realistic calculation of a cluster of fuel assemblies, what is sometimes called a mini-core calculation. As MCNP5 has already been used as the basis for several hybrid deterministic-Monte Carlo schemes, we considered it the prime target for application of our scheme.

2. IMPLEMENTATION OF THE SCHEME IN MCNP5

In order to obtain the adjoint functions used for biasing, a deterministic code that could output adjoint functions for a 3-D geometry was required. Our code of choice was PARTISN4 [4], a multigroup $S_N$ code, mainly because of its ease of use and its proven capabilities in producing adjoint functions from source-detector calculations.

The binary output file from PARTISN4 containing the adjoint functions was processed and read by a modified version of MCNP5 via a special option in the input file. The binary format was necessary as using linear interpolation to calculate the adjoint functions for biasing requires the adjoint functions at several points in the grid, rather than the piecewise-constant ones, making the amount of data for a 3-D calculation too large and inefficient. Since the output from PARTISN4 is that of an adjoint calculation, the directions had to be reversed in order to convert them to the equivalent MCNP5 ones, while the groups were printed out in a reverse order, which had to be taken into account.

A number of Fortran subroutines were written that dealt with biasing the source and collision and transition kernels. The particle tracking routine required some changes, since we wanted the code to allow particles to be tracked all the way to the boundaries of the system, so that the biased probability table is generated. Because of that, special attention was given to treatment of boundary conditions in the code. Also, as the code allows for repeated geometrical structures, we had to take them into account since particle transport is done in the local geometrical structures, while retrieval and use of the adjoint functions is done in the global geometry. In addition, the code could easily accommodate discrete directions, using the PARTISN4 output file as an input for the number and data for those directions.

Regarding creation and use of the adjoint function mesh, the current framework in the code that deals with weight window meshes was used. This had the advantage that the code already had routines available to retrieve or recalculate any data necessary upon entering a virtual mesh, a geometrical surface or a combination of the two.
3. A MINI-CORE CALCULATION

3.1 Description of the Problem

In our mini-core calculation, a 3 \times 3 cluster of 9 PWR fuel assemblies is used, with boundary conditions set to vacuum. Each assembly consists of a 17 \times 17 array of heterogeneous fuel pins, control rod guide tubes with the rods fully extracted and a central instrumentation tube. For simplicity the central instrumentation tube, which normally contains measuring devices, is assumed to have the same geometry and composition as the control rod guide tube. The dimensions for both the fuel and the water hole cells are shown in Fig. 1. Each MOX assembly contains fuel pins with three different values for plutonium weight percent in heavy metal, 3.2\%, 5.2\% and 7.80\%, defined as MOX-1, MOX-2 and MOX-3, respectively, with a density of 10.25 g/cm$^3$. In the lattice, the 8 outer assemblies contain a mixture of MOX-1, MOX-2 and MOX-3 fuel as shown in Fig. 2, while the central assembly replaces all fuel types with MOX-1 type fuel.

The composition of Uranium and Plutonium in the MOX fuel is given in Table I. For simplicity, the isotopic composition of the cladding is assumed to be natural Zr, with a reduced density of 5.77 g/cm$^3$ as it is smeared over the gap between the cladding and fuel. The guide tubes are also of Zr, but with a density of 6.55 g/cm$^3$. The coolant in both types of cells is water with a density of 0.7164 g/cm$^3$, containing 500 ppm of boron with an assumed boron isotopic composition of 18 wt \% $^{10}$B and 82 wt \% $^{11}$B.

<table>
<thead>
<tr>
<th></th>
<th>Uranium</th>
<th>Plutonium</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-234</td>
<td>0.01%</td>
<td>Pu-238 2.10%</td>
</tr>
<tr>
<td>U-235</td>
<td>0.25%</td>
<td>Pu-239 54.50%</td>
</tr>
<tr>
<td>U-238</td>
<td>99.74%</td>
<td>Pu-240 25.00%</td>
</tr>
</tbody>
</table>
Using this data, the geometry and composition inputs were manually generated for both MCNP5 and PARTISN4.

### 3.2 Cross-section Preparation

In order to provide better comparison to PARTISN4, the cross-sections were homogenised for each different material. This was done via multiple continuous-energy MCNP5 calculations and a special program called MgXsect [5], using specific material zones for cross-section tallying, for 3 energy groups. The temperature of the fuel was 813 K, while the cladding, guide tubes and coolant were at 578 K. The upper energy boundaries of the groups were at 20 MeV, 100 keV and 0.625 eV for groups 1, 2 and 3, respectively.

Since PARTISN4 would only be used for the generation of adjoint functions, we chose to use simplified, homogenised cells for the deterministic calculation. Six different types of cells were therefore created: 3 types for the MOX fuel in the outer assemblies, a water hole/guide tube cell, plus the water hole and MOX-1 fuel cell in the central assembly. The cross-section data
Monte Carlo zero-variance based scheme demonstration

was again generated using the same MCNP5/MgXsect calculation, then converted and used as direct ASCII input in PARTISN4, since it provides the option.

Because of the way MCNP5 uses the cross-sections, when generating them there was an issue with \((n,2n)\) reactions, because of which the sum of all interaction cross-sections was higher than \(\Sigma_t\). In order to solve that problem and because the probability for \((n,2n)\) was minute, we took it into account implicitly, by setting that cross-section to zero and then increasing the weight of a particle at each collision by the \((n,2n)\) probability.

### 3.3 Generation of Adjoint Functions

In order to generate adjoint functions using PARTISN4, we had to run the code in adjoint source-detector mode, using a special option in the code in order to remove fission (and therefore get the correct importances per source particle). The adjoint source was then normalised to \(\nu \Sigma_f\) of the forward \(k_{eff}\) calculation:

\[
S_{tot}^* = \sum_g \int \nu \Sigma_f(g) dV \tag{1}
\]

Note that the source term in the adjoint integro-differential equation solved by PARTISN4 is \(\eta^* = \nu \Sigma_f(P)\), while in the integral equation solved by Mote Carlo it is \(\eta^* = \nu \Sigma_f(P)/\Sigma_t(P)\).

A coarse mesh of one bin per cell was generated, since we wanted to test the performance of the scheme when rather imprecise adjoint functions are used for biasing. Running PARTISN4 generated directional and scalar adjoint functions for all 3 groups. We can see the output for the scalar, group dependent adjoint function \(\chi^*\) in Fig. 3. The CPU time cost for this calculation was extremely low, as it was completed in less than 2 seconds, on a modern dual-core Intel CPU.

### 4. RESULTS

Using the data mentioned above, the calculation was run for a total of 1000 active cycles, with 50 initial (inactive) cycles to allow for the fission source to converge. 2000 particles per cycle were used, while the Russian roulette survival weight was set to 0.02, resulting in a threshold weight of 0.01.
Three different calculations were performed. One standard criticality calculation using implicit capture only. Two calculations were preformed with the zero-variance based biasing scheme using either a piece-wise constant adjoint function over each mesh of the adjoint function and a linearly interpolated adjoint function between the values at successive mesh boundaries as provided by PARTISN4. The results can be seen in Table II, while in Fig. 4 we can see the evolution of $k_{\text{eff}}$ for the 1000 active cycles of the calculation.

### Table II: Calculation results for the mini-core system.

<table>
<thead>
<tr>
<th>Type of calculation</th>
<th>$k_{\text{eff}}$</th>
<th>st. dev.</th>
<th>Relative FOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit capture (no biasing)</td>
<td>0.98924</td>
<td>0.00034</td>
<td>1</td>
</tr>
<tr>
<td>Biasing with piecewise-constant adjoint function</td>
<td>0.98854</td>
<td>0.00020</td>
<td>0.49</td>
</tr>
<tr>
<td>Biasing with linearly-interpolated adjoint function</td>
<td>0.98911</td>
<td>0.00018</td>
<td>0.56</td>
</tr>
</tbody>
</table>

Both unbiased and biased calculations converge rather quickly, even for the variation shown in the first hundred active cycles, if we take into account the scale of the plot. Also, the results are all within $3\sigma$ of each other, which indicates that there were no errors (which would in any case be apparent in the particle contributions) in the implementation of either biasing method.

![Figure 4: $k_{\text{eff}}$ during the active cycles.](image)

### 4.1 Variance

It is more useful to look at the evolution of the standard deviation of the calculation during the active cycles, shown in Fig. 5, where we see that the biasing scheme performs consistently...
better than the implicit capture scheme. In fact, the implicit capture calculation needs over three times the cycles in order to achieve the same relative standard deviation as the biased one. The biased calculation achieves a $10^{-3}$ relative standard deviation in around cycles, while the implicit capture one requires around 140 cycles. Similarly, a $5 \times 10^{-4}$ relative standard deviation is achieved in 130 cycles, while for the implicit capture calculation, this requires almost 500 cycles.

It is interesting to see that the two different types of interpolation produce almost identical results. This can be explained by the fact that, during the calculation of the adjoint functions, the cells are homogenised. That way, the adjoint function is not significantly different at the boundaries of each cell, which means that linear interpolation is not much more accurate, compared to the piecewise-constant one.

However, since the extra CPU time necessary for calculation of the linearly interpolated adjoint functions was not significant, our suggestion would be to always use this type of implementation. One should only fall back to the piecewise-constant adjoint functions when the number of mesh cells is so large that the recalculation of the adjoint functions becomes a problem.

### 4.2 Efficiency

We earlier discussed the reduction in variance the scheme resulted in, but not of any increase in the efficiency of the calculation. This is because we found that our implementation made the code run extremely slow when running biased calculations. In fact, testing the time of the MCNP5 calculation to that of our own code, using a loosely coupled core as a testing system [2], we saw that MCNP5 was around an order of magnitude slower. This was also the difference between the biased and unbiased runs with MCNP5, which resulted in a relative figure of merit.
of 0.49 and 0.56 for the piecewise-constant and linearly-interpolated results, with the implicit capture result given a reference value of 1.

Of course, this was not performed as a direct comparison of the code systems, since our code was built just for showcasing the biasing scheme, while MCNP5 has a significantly broader range of applications. However, it was able to show us that our implementation in MCNP5 was certainly lacking optimisations that could bring the performance of the biasing scheme on par with the current capability of the code.

By profiling the code, it is quite obvious that most of the time is spent during the tracking of the particle throughout the geometry. This is even more pronounced in the biased calculations, where the particle must be tracked all the way to the boundaries of the system (or, theoretically, to infinity if the system uses reflective boundaries). This, of course, is not a matter of code optimisation, rather a direct consequence of the zero-variance scheme: The location of a new particle interaction must be selected by a distribution biased by $\psi^*$, which only becomes zero outside the boundaries of the system. In our scheme, we chose to stop particle tracking after 10 mean free paths, since the probability of the particle to reach any further without interactions approaches zero. Still, this requires significant calculation time compared to the simple inversion method used to find the neutron’s path length between collisions.

In addition, we noticed that whenever a particle would cross a virtual surface, the code immediately had to search for both the cell and adjoint function mesh the particle was now in, in order to be able to obtain the material data and the adjoint functions, both necessary for the biasing process.

As our main interest lied in showcasing the reduction in variance obtained by the scheme, we did not try to perform further optimisations towards the efficiency of the coding.

### 4.3 Source Convergence

MCNP5 automatically calculated the Shannon entropy of the source, with the results for all 1050 cycles shown in Fig. 6. The initial 50 cycles, which are discarded for estimating $k_{\text{eff}}$, are shown magnified in the inset graph. It is obvious that source convergence has already been achieved within those 50 cycles (the MCNP5 code output confirms this), so no more cycles need to be discarded before calculating $k_{\text{eff}}$ statistics.

Also, we see that the evolution of the source distribution does not differ between the biased and unbiased calculations. This indicates on one hand that the scheme does not offer a speed-up in the source convergence for the system, but on the other hand biasing with the adjoint functions does not adversely affect the fundamental behaviour of the calculation.

### 5. CONCLUSIONS

In this paper a zero-variance based scheme was implemented in the MCNP5 Monte Carlo code. Running a mini-core reactor problem, it was demonstrated that the theoretical reduction in variance still applies for practical nuclear reactor calculations, without affecting source distribution convergence of the system. However, we were not able to demonstrate an appropriate increase
in the efficiency of the calculation, since lack of code optimisation resulted in significantly higher CPU time cost.

ACKNOWLEDGEMENTS

This work is partially funded by the Integrated Project NURESIM (Contract n° 516560) in the 6th Euratom Framework Programme of the European Union.

REFERENCES


