Simulation of neutronics for advanced reactors: Monte-Carlo method

Konstantin Mikityuk
Paul Scherrer Institut, Switzerland

Joint IAEA-ICTP Workshop on Physics and Technology of Innovative Nuclear Energy Systems
20-24 August 2018, ICTP, Trieste, Italy
Outline-2

Result estimates
- Scoring and collecting the results using batches
- Statistical accuracy and law of large numbers
- Central limit theorem and confidence intervals

Non-analog Monte Carlo
- Result estimates of neutron flux
- Statistical weight
- Russian roulette
- Splitting

Interactions
- Real or virtual
- Collision type: scattering (isotropic)
- Collision type: absorption = fission + capture
Literature


Introduction
Introduction: Monte Carlo method in neutron transport calculations

The Monte Carlo method is a technique for estimating the expected value of a random variable together with its standard deviation.

- In reactor physics it is done by a direct simulation of a population of neutrons by sampling individual neutrons.
- For each neutron a sequence of physical random events is simulated using a sequence of random numbers.
- Some parameters of average behavior of the population are recorded (scored).
Monte Carlo is a *stochastic* method differing from the *deterministic* methods.

- Deterministic methods (e.g. discrete ordinates method or method of characteristics) solve the neutron transport (Boltzmann) *equation* for angular flux and k-effective.

- Stochastic method (Monte Carlo) find the parameters of interest (e.g. k-effective, reaction rates) by simulating the random walk of individual neutrons. No neutron transport *equation* is solved.
Introduction: continuous-energy versus multi-group

Monte Carlo can use the following two representations of the nuclear data

– *Continuous-energy*, i.e. based on all data points available in ENDF files without any condensations. ACE format data libraries are prepared using the NJOY code.

– *Multi-group*, i.e. nuclear data condensed in energy using the energy group structures, similarly to conventional deterministic codes.

Most of the modern Monte Carlo codes (MCNP, Serpent) are based on the continuous-energy representation of the nuclear data.

In our Matlab exercises we will use multi-group representation of the nuclear data.
Introduction: analog versus non-analog

- *Analog* Monte Carlo:
  explicit, ‘as is’ simulation of individual neutrons from emission to absorption without any simplifications.

- *Non-analog* Monte Carlo:
  simulations using simplifications, tricks, acceleration techniques, etc.
Mathematical background
Math: a random variable

A random variable $x$ is a variable whose possible values are numerical outcomes of a random process (experiment), e.g. flipping a coin or rolling a die.

$x$ can be

- **discrete**, i.e. taking one of a specified finite list of values (e.g. number of dots on a dice face);

or

- **continuous**, i.e. taking any numerical value in a specified interval (e.g. atmospheric pressure).

Continuous random variable uniformly distributed between 0 and 1 is denoted $\xi$. All other random numbers will be derived from $\xi$.

In MATLAB exercise $\xi$ is calculated using the Matlab pseudo-random number generator `rand()` based on the Mersenne-Twister algorithm (see Wikipedia).
A probability density function (PDF) $f(x)$ describes the relative likelihood for the continuous random variable $x$.

Examples:
- angle between bike wheel valve and horizon (uniformly-distributed PDF)
- atmospheric pressure (normally-distributed PDF)

\[ dP = f(x)dx \] is the probability for $x$ to have a value between $x$ and $x+dx$.

The probability for $x$ to have a value between $a$ and $b$

\[
P(a < x < b) = \int_{a}^{b} dP = \int_{a}^{b} f(x)dx
\]

The total area below the PDF curve = ?
Math: cumulative distribution function (CDF)

Probability that a random variable takes a value less than or equal to $x$:

$$F(x) = \int_{-\infty}^{x} dP = \int_{-\infty}^{x} f(x') dx'$$

$F(x)$ changes from 0 to 1.
Math: sampling

*Sampling* is selection of random values according to the probability distributions (CDF or PDF) with the goal to represent with these few values the whole population.

**Sampling approach:**

1. Generate $\xi$ (uniformly distributed between 0 and 1)
2. Use $\xi$ to generate random values for parameters of interest using CDF by inverse method

*This is only one approach. There are much more techniques...*
Math: sampling by inverse method

Sampling of random variable by \textit{inverse method} is done using the inverse of the Cumulative Distribution Function $F(x)$.

1. Generate $\xi$ (uniformly distributed between 0 and 1)

2. The cumulative probability of the event assumed equal to $\xi$: $F(x) = \xi$

3. $x$ is found from the inverse function: $x = F^{-1}(\xi)$

Example: sampling of exponential distribution

The inverse of CDF is known only in simple case ($\exp \rightarrow \ln$, $\sin \rightarrow \arcsin$), in most real cases the inverse function is not known analytically. In such cases the inverse could be found \textit{numerically} or by \textit{acceptance-rejection technique} (not considered here)
Neutron tracking
Neutron tracking: introduction

Neutron tracking is simulation of a single neutron movement through the different material regions of the reactor core.

A *neutron track* – length of path that neutron makes between two interactions (collisions). The track can be cut short by the boundary between materials.

A *neutron history* – entire set of tracks made from initial emission to final absorption or escape.
Neutron tracking: sampling of free path length in homogeneous medium

Sampling of the free path length between two collision points (0 and \(x\)) is the basis of neutron tracking. For homogeneous infinite medium:

Macroscopic XS is interaction probability \(P\) per path length travelled by neutron:

\[
\Sigma_t = \frac{dP}{dx}
\]

Increase of probability to have the first interaction moving from \(x\) to \(x+dx\):

\[
dP_1(x) = P_0(x) dP
\]

Decrease of probability NOT to interact moving from \(x\) to \(x+dx\):

\[
dP_0 = -dP_1(x) = -P_0(x) dP = -P_0(x) \Sigma_t dx
\]

Non-interaction probability:

\[
P_0(x) = \exp(-x \Sigma_t)
\]
Neutron tracking: sampling of free path length in homogeneous medium

Increase of probability that neutron has first interaction moving from $x$ to $x+dx$:

$$dP_1(x) = P_0(x) dP = P_0(x) \Sigma_t dx = \Sigma_t \exp(-x\Sigma_t) \, dx$$

PDF of free path length: $f(x) = dP_1/dx = \Sigma_t \exp(-x\Sigma_t)$

CDF of free path length: $F(x) = 1 - \exp(-x\Sigma_t) = \xi$

The inverse function: $F^{-1}(\xi) = -\ln(1 - \xi)/\Sigma_t$

Sampling of free path length by inverse method:

$$x = -\ln(\xi)/\Sigma_t$$

where $\xi$ (uniformly distributed between 0 and 1)

```matlab
% Sample free path length according to the Woodcock method
freePath = -log(rand())/SigTmax(iGroup(iNeutron));
```
Neutron tracking: homogeneous versus heterogeneous materials

Neutron free path length sampling is valid in homogeneous material ($\Sigma_t$ is independent on space coordinate).

For heterogeneous materials (combination of several homogeneous materials or cells) collision probability changes each time when neutron crosses a cell boundary.

What to do?

- stop neutron at boundary surface and adjust or re-sample remaining distance to the next collision point (*ray tracing*);

- do not stop neutron at boundary surface but instead consider for each material fictitious X\(\text{s}\) which equalize total X\(\text{s}\) of all materials (*delta tracking*)
Neutron tracking: ray-tracing

Assume that the free path length sampled for mat1 is $x_1$. It can happen that neutron ends up at different material (mat2).

To re-adjust the coordinate of the next collision:

- we preserve the sampled non-interaction probability:
  $$\exp(-x_2 \Sigma_{t,2}) = \exp(-(x_1 - d) \Sigma_{t,1}) = \xi$$
  or (equivalent)

- we stop at the boundary and re-sample $x_2$

In both cases we should calculate distance to the boundary $d$:
could become very expensive for complicated geometry
Neutron tracking: delta-tracking

Goal: to sample the next collision point \textit{without} handling the surface crossings

Is an \textit{acceptance-rejection} technique.

Proposed by Woodcock in the 1960s.

Used in \textit{Serpent} Monte Carlo code as a basic algorithm (optional in other codes)

Based on a concept of \textit{virtual collision} (or \textit{pseudo-scattering})

Scattering reaction (fictitious) in which angular and energy distributions are characterised by $\delta$-functions ($\delta(E_0)$ and $\delta(\Omega_0)$) and state of neutron is completely preserved
Neutron tracking: delta-tracking

Key idea: to add an appropriate virtual collision XS ($\Sigma_V$) to each material in such a way that the modified total XS ($\Sigma_t$) has the same value in all materials.

Instead of heterogeneous material composition we obtain one pseudo-homogeneous material

![Diagram showing the relationship between $\Sigma_V$ and $\Sigma_t$ for different materials.]

This eliminates the need to adjust free path length each time neutron enters new material and the need to calculate surface distances.
Neutron tracking: delta-tracking

The virtual collision XS is given by:

\[ \Sigma_V(r, E) = \Sigma_m(E) - \Sigma_t(r, E) \]

where \( \Sigma_m(E) \) is the *majorant*, maximum of all total XSs in the system (the same for all materials).

Delta-tracking starts with sampling the free path using the *majorant*

\[ x = - \ln(\xi) / \Sigma_m \]

At the new collision point the collision type (real or virtual) is sampled by generating the random \( \xi \) and comparing it with

\[ P = \Sigma_V(r, E) / \Sigma_m(E) \]

\( P > \xi \) – virtual, otherwise real
Neutron tracking: delta-tracking

If the collision is real, the collision type is sampled, if virtual—nothing changes.

In other words: the neutron always travels by steps (free paths) determined by the most “opaque” material in the system and when it realizes that it is unnecessarily too short it just continues.
Neutron tracking: delta-tracking

Advantage

- it does not matter if the neutron crosses one or several material boundaries between two collision points, we just need to know where the collision point is (what is the total XSs at this point).

Disadvantages

- surface crossings are not recorded at all (only collision estimator of neutron flux available);
- surface flux and current can be easily estimates only at outer geometry boundary;
- when there is small-volume heavy absorber in the geometry, it determines the majorant and the efficiency is reduced.
Result estimates
Result estimates: scoring

Monte-Carlo game consists of two parts:
- simulation of neutron histories (discussed above)
- collection of results

Recorded events = scores are combined to obtain statistical estimates

Collection of results similar to measurements in an experiment and based on evaluation of flux integrals:

\[ R = \int_t \int_V \int_E f(r, E)\phi(r, E)dtdVdE \]

where \( f(r, E) \) is response function, e.g. 1 (to estimate flux) or \( \Sigma_x \) (to estimate reaction rate)

Integration over time is equivalent to averaging over many neutron histories. Normalization should be applied afterwards...
Result estimates: collecting the results using batches

All scores from one generation of neutrons $n$ are grouped in a single *batch*.

Batch = generation.

Number of neutron histories in one batch $I_n$ (may differ from batch to batch)

Number of batches $N$

Estimate of reaction rate in generation $n$:

$$R_n = \sum_{i=1}^{I_n} f^i \phi^i$$

More generally: estimate $X_n$ (e.g. could be ratio of reaction rates)

$X_n$ is random parameter, changing from batch to batch, not so interesting

More interesting – statistically averaged (mean) values + standard deviations
Result estimates: statistical accuracy

Mean value = the result

\[
\bar{X} = \frac{1}{N} \sum_{n=1}^{N} X_n
\]

Standard deviation = statistical accuracy

\[
\sigma(X) = \sqrt{\frac{1}{N(N-1)} \sum_{n=1}^{N} (X_n - \bar{X})^2}
\]

Frequently used quantities related to the standard deviation

- variance \( \sigma^2(X) \)
- relative statistical error \( E(X) = \sigma(X)/\bar{X} \)
Result estimates: statistical accuracy

The result of Monte Carlo simulation always given in the form \( \bar{X} \pm \sigma(X) \)

The longer simulation runs the closer the mean of the results to the *expected value* = Law of large numbers

\[
\lim_{n \to \infty} \sigma = 0
\]

Qualitative meanings of the statistical accuracy:
- how much the mean value is likely to deviate from the *expected value*
- how much results of two identical but independent simulations are likely to differ

In any case *statistical accuracy* of the simulation \( \neq \) *physical accuracy* of the simulation
Result estimates: central limit theorem

To find statistical accuracy of estimate $X$, we need in addition to standard deviation to know the probability distribution function (PDF).

Central limit theorem states that sum (or mean) of a large number of arbitrarily distributed random variables is itself a random variable following the normal distribution.

\[
f(x) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left( - \frac{x - \bar{x}}{2\sigma^2} \right)
\]

Assumptions:
- distribution is the same for each term in the sum
- values are independent
- both mean and standard deviation exist and are finite
Result estimates: confidence intervals

*Confidence interval* determines probability at which the result lies within a certain distance from the true mean value of the distribution.

In case of normal PDF, e.g. $1.02 \pm 0.01$ means that the true result lies

- with probability of 68% in the interval $1.01 – 1.03$ and
- with probability of 95% in the interval $1.00 – 1.04$
Non-analog Monte Carlo
Non-analog Monte Carlo: statistical trickery

Non-analog methods could be used instead of analog ones in order to make calculations faster

1. to improve statistics on reaction rates by estimating the flux (important when reaction rate is low)

2. to improve the random walk algorithm in order to score more frequently the neutrons having largest contribution to the results and to get rid of the neutrons with low importance
Non-analog Monte Carlo: result estimates

Analog
Score physical interactions for individual reactions (fission, capture, scattering, etc.)

Non-analog
Estimate flux and multiply it by the value of the response function (e.g. macro-XS).
The flux can be found by
- Collision estimate
- Track length estimate
- Surface and current estimate
Non-analog Monte Carlo: statistical weight

Analog
Each neutron history represents the transport of a single particle

Non-analog
Each neutron is assigned with a statistical weight $W$ and
- represents the contribution of several particles ($W > 1$); or
- has the same significance as analog simulation ($W = 1$); or
- has less significance than analog simulation ($W < 1$).
Non-analog Monte Carlo: statistical weight

k-effective of the cycle is the total weight of neutrons in the system divided by the number of neutrons born $N_{\text{born}}$ (fixed value = size of the batch).

At the beginning of each cycle the total weight of neutrons is normalised to $N_{\text{born}}$. This is equivalent to dividing the fission source by k-effective.

```matlab
% Normalize the weights of the neutrons to make the total weight equal to numNeutrons_born (equivalent to division by keff_cycle)
weight = (weight ./ sum(weight,2)) * numNeutrons_born;
weight0 = weight;
```

When $W > 1$ neutron splitting and when $W < 1$ neutron terminating are considered.
Non-analog Monte Carlo: Russian roulette

When weight of a neutron reduces, its contribution to overall results reduces too and tracking of such a neutron becomes a waste of computing time. How to get rid of too “light” neutrons?

Solution: assign a cut-off value for weight and play Russian roulette for neutrons with the weight below the cut-off.

One of the simple implementations:

− for each neutron set the terminate probability as $P = (1 - W / W_0)$, where $W_0$ is the weight at the beginning of the generation;
− generate random $\xi$;
− if $P > \xi$ terminate the neutron;
− otherwise and if $P > 0$ keep the neutron and set $W = W_0$
Non-analog Monte Carlo: splitting

The way around: when the weight of the neutron born in fission or \((n,2n)\) reaction is too high, it should be split.

One of the simplest algorithms is for every neutron with \(W > 1\):
- Generate random \(\xi\)
- calculate \(N = \text{floor}(W) = \lfloor W \rfloor\)
- if \(W - N > \xi\), split the neutron in \(N + 1\) identical neutrons with \(W / N\);
- otherwise, split the neutron in \(N\) identical neutrons with \(W / N\)
Interactions
Interactions: real or virtual

Once the collision point is sampled using either *ray-tracing* or *delta-tracking* method, the interaction type is sampled.

**Non-analog** (delta-tracking only): sample if collision real or virtual

\[ P = \frac{\sum_V (r, E)}{\sum_m (E)} \]

\[ \xi = \text{rand}() \]

Virtual \( \xrightarrow{\xi < P} \) \( Y \) \( \xleftarrow{\xi < P} \) \( N \) \( \text{Real} \)
Interactions: collision type

Both **non-analog** (when reaction is real) and **analog**: sample reaction type*

- scattering
- absorption (capture + fission)

\[
P = \frac{\Sigma_s(r, E)}{\Sigma_t(r, E)}
\]

\[
\xi = \text{rand}()
\]

* (n,2n) reaction not considered
Interactions: scattering

**Analog** and **non-analog** are the same (weight does not change).

Scattering assumptions:

- **Isotropic**: new direction and energy are sampled independently.
- **Anisotropic**: new direction and new energy are not independent (not considered here, see Leppänen pp. 105-111 for more details)
Interactions: isotropic scattering

Direction and energy of secondary neutron are sampled independently assuming isotropic scattering in the laboratory system (simplification)

Direction: \( \theta = \arccos[2\xi_1 - 1] \) and \( \varphi = 2\pi \xi_2 \)

\[
teta = \arccos(2*\text{rand}()-1);
phi = 2.0*\pi*\text{rand}();
dirX = \sin(teta)\times\cos(phi);
dirY = \sin(teta)\times\sin(phi);
x(iNeutron) = x(iNeutron) + \text{freePath} \times \text{dirX};
y(iNeutron) = y(iNeutron) + \text{freePath} \times \text{dirY};
\]

Energy \( E' \) is sampled by the inverse method:

\[
\xi = \text{rand}()
\]

Integrate numerically CDF

\[
F(E, E') = \frac{\int_0^{E'} \Sigma_s(E \rightarrow E'')dE''}{\int_0^{\infty} \Sigma_s(E \rightarrow E'')dE''}
\]

until \( F(E, E') \geq \xi \)

% Sample the energy group of the secondary neutron
iGroup(iNeutron) = find(cumsum(SigS)/SigS_sum >= \text{rand()}, 1, 'first');
Interactions: absorption = capture + fission

A simple method (combination of **analog** and **non-analog**) to be used in our Matlab exercise:

– Neutron is not terminated but its weight is changed by the eta-value (number of neutrons emitted per neutron absorbed):

$$W' = W \frac{\Sigma_p}{\Sigma_a}$$

```matlab
weight(iNeutron) = weight(iNeutron) * (SigP/SigA)
```

– Automatically the neutron is terminated in non-multiplying regions

– Energy $E'$ of neutron is sampled by the inverse method:

$$\xi = \text{rand}()$$

Integrate numerically CDF until $F(E) \geq \xi$

```matlab
iGroup(iNeutron) = find(cumsum(fuel.chi) >= rand(), 1, 'first');
```
MATLAB exercise
% Number of source neutrons
numNeutrons_born = 100; % INPUT

% Number of inactive source cycles to skip before starting k-eff accumulation
numCycles_inactive = 100; % INPUT

% Number of active source cycles for k-eff accumulation
numCycles_active = 2000; % INPUT

% Size of the square unit cell
pitch = 3.6; %cm % INPUT

% Path to macroscopic cross section data:
path(path,'..\02.Macro.XS.421g');

% Fill the structures fuel, clad and cool with the cross sections data
fuel = macro421_UO2_03__900K; % INPUT
clad = macro421_Zry__600K; % INPUT
cool = macro421_H2OB__600K; % INPUT

% Define the majorant: the maximum total cross section vector
SigTmax = max([fuel.SigT; clad.SigT; cool.SigT]);

% Number of energy groups
ng = fuel.ng;
% Detectors

detectorS = zeros(1,ng);

% Four main vectors describing the neutrons in a batch
x = zeros(1,numNeutrons_born*2);
y = zeros(1,numNeutrons_born*2);
weight = ones(1,numNeutrons_born*2);
iGroup = ones(1,numNeutrons_born*2);

% Neutrons are assumed born randomly distributed in the cell with weight 1
% with sampled fission energy spectrum
numNeutrons = numNeutrons_born;
for iNeutron = 1:numNeutrons
    x(iNeutron) = rand()*pitch;
y(iNeutron) = rand()*pitch;
    weight(iNeutron) = 1;
% Sample the neutron energy group
    iGroup(iNeutron) = find(cumsum(fuel.chi) >= rand(), 1, 'first');
end

% Prepare vectors for keff and standard deviation of keff
keff_expected = ones(1,numCycles_active);
sigma_keff = zeros(1,numCycles_active);
keff_active_cycle = ones(1,numCycles_active);
virtualCollision = false;
% Main (power) iteration loop
for iCycle = 1:(numCycles_inactive + numCycles_active)

% Normalize the weights of the neutrons to make the total weight equal to
% numNeutrons_born (equivalent to division by keff_cycle)
weight = (weight ./ sum(weight,2)) * numNeutrons_born;
weight0 = weight;

% Loop over neutrons
for iNeutron = 1:numNeutrons
    absorbed = false;

    % Neutron random walk cycle: from emission to absorption
    while ~absorbed
        % Sample free path length according to the Woodcock method
        freePath = -log(rand())/SigTmax(iGroup(iNeutron));

        if ~virtualCollision
            % Sample the direction of neutron flight assuming both
            % fission and scattering are isotropic in the lab (a strong
            % assumption!)
            teta = pi*rand();
            phi = 2.0*pi*rand();
            dirX = sin(teta)*cos(phi);
            dirY = sin(teta)*sin(phi);
        end
    end

end
% Fly
x(iNeutron) = x(iNeutron) + freePath * dirX;
y(iNeutron) = y(iNeutron) + freePath * dirY;

% If outside the cell, find the corresponding point inside the cell
while x(iNeutron) < 0, x(iNeutron) = x(iNeutron) + pitch; end
while y(iNeutron) < 0, y(iNeutron) = y(iNeutron) + pitch; end
while x(iNeutron) > pitch, x(iNeutron) = x(iNeutron) - pitch; end
while y(iNeutron) > pitch, y(iNeutron) = y(iNeutron) - pitch; end

% Find the total and scattering cross sections
if x(iNeutron) > 0.9 && x(iNeutron) < 2.7  % INPUT
    SigA = fuel.SigF(iGroup(iNeutron)) + fuel.SigC(iGroup(iNeutron)) + fuel.SigL(iGroup(iNeutron));
    SigS = fuel.SigS1+0(iGroup(iNeutron),:);
    SigP = fuel.SigP(iGroup(iNeutron));
elseif x(iNeutron) < 0.7 || x(iNeutron) > 2.9  % INPUT
    SigA = cool.SigC(iGroup(iNeutron)) + cool.SigL(iGroup(iNeutron));
    SigS = cool.SigS1+0(iGroup(iNeutron),:);
    SigP = 0;
else
    SigA = clad.SigC(iGroup(iNeutron)) + clad.SigL(iGroup(iNeutron));
    SigS = clad.SigS1+0(iGroup(iNeutron),:);
    SigP = 0;
end

% Find the other cross sections ...
% ... scattering
SigS_sum = sum(SigS);
% ... total
SigT = SigA + SigS_sum;
% ... virtual
SigV = SigTmax(iGroup(iNeutron)) - SigT;
% Sample the type of the collision: virtual (do nothing) or real
if SigV/SigTmax(iGroup(iNeutron)) >= rand() % virtual collision
    virtualCollision = true;
else % real collision
    virtualCollision = false;

% Sample type of the collision: scattering or absorption
if SigS_sum/SigT >= rand() % isotropic scattering

% Score scatterings with account for weight divided by the % total scattering cross section
detectS(iGroup(iNeutron)) = detectS(iGroup(iNeutron)) + weight(iNeutron)/SigS_sum;

% Sample the energy group of the secondary neutron
iGroup(iNeutron) = find(cumsum(SigS)/SigS_sum >= rand(), 1, 'first');

else % absorption
    absorbed = true;

% Neutron is converted to the new fission neutron with % the weight increased by eta
weight(iNeutron) = weight(iNeutron) * (SigP/SigA);

% Sample the energy group for the new-born neutron
iGroup(iNeutron) = find(cumsum(fuel.chi) >= rand(), 1, 'first');

end % scattering or absorption
end % virtual or real
end % of neutron random walk cycle: from emission to absorption
end % of loop over neutrons
% Russian roulette
for iNeutron = 1:numNeutrons
    terminateP = 1 - weight(iNeutron)/weight0(iNeutron);
    if terminateP >= rand()
        weight(iNeutron) = 0; % killed
    elseif terminateP > 0
        weight(iNeutron) = weight0(iNeutron); % restore the weight
    end
end

% Clean up absorbed or killed neutrons
x(weight == 0) = [];
y(weight == 0) = [];
iGroup(weight == 0) = [];
weight(weight == 0) = [];
numNeutrons = size(weight,2);
% Split too "heavy" neutrons

numNew = 0;
for iNeutron = 1:numNeutrons
    if weight(iNeutron) > 1
        % Truncated integer value of the neutron weight
        N = floor(weight(iNeutron));
        % Sample the number of split neutrons
        if weight(iNeutron)-N > rand(), N = N + 1; end
        % Change the weight of the split neutron
        weight(iNeutron) = weight(iNeutron)/N;
        % Introduce new neutrons
        for iNew = 1:N-1
            numNew = numNew + 1;
            x(numNeutrons + numNew) = x(iNeutron);
            y(numNeutrons + numNew) = y(iNeutron);
            weight(numNeutrons + numNew) = weight(iNeutron);
            iGroup(numNeutrons + numNew) = iGroup(iNeutron);
        end
    end
end

% Increase the number of neutrons
numNeutrons = numNeutrons + numNew;
% k-eff in a cycle equals the total weight of the new generation over
% the total weight of the old generation (the old generation weight =
% numNeutronsBorn)
    keff_cycle = sum(weight,2)/sum(weight0,2);

iActive = iCycle - numCycles_inactive;
if iActive <= 0
    fprintf('Inactive cycle = %3i/%3i; k-eff cycle = %8.5f; numNeutrons = %3i\n', ...
            iCycle,numCycles_inactive,keff_cycle,numNeutrons);
else
    % k-effective of the cycle
    keff_active_cycle(iActive) = keff_cycle;

    % k-effective of the problem
    keff_expected(iActive) = mean(keff_active_cycle(1:iActive));

    % Standard deviation of k-effective
    sigma_keff(iActive) = sqrt( sum( ( keff_active_cycle(1:iActive) - keff_expected(iActive) ).^2 ) / max(iActive-1,1) / iActive );

    fprintf('Active cycle = %3i/%3i; k-eff cycle = %8.5f; numNeutrons = %3i; k-eff expected = %9.5f; sigma = %9.5f\n', ...
            iCycle-numCycles_inactive, numCycles_active, keff_cycle, numNeutrons, keff_expected(iActive), sigma_keff(iActive));
end

end % of main (power) iteration