

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

J. Leppänen, Development of a New Monte Carlo Reactor Physics Code, ISBN 978-951-38-7018-8, PhD thesis, VTT Publications (2007), Chapters 5 and 6.

A. Hébert, Applied Reactor Physics, ISBN 978-2-553-01436-9, Library and Archives Canada, Canada (2009), Chapter 3.11.

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).

Introduction: stochastic versus deterministic

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 ξ . All other random numbers will be derived from ξ .

In MATLAB exercise ξ is calculated using the Matlab pseudo-random number generator `rand()` based on the Mersenne-Twister algorithm (see Wikipedia). 10

Math: probability density function (PDF)

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 = ?

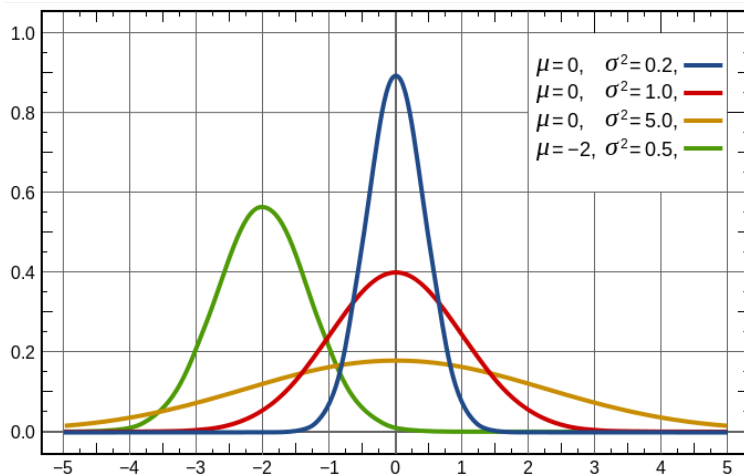
Math: cumulative distribution function (CDF)

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

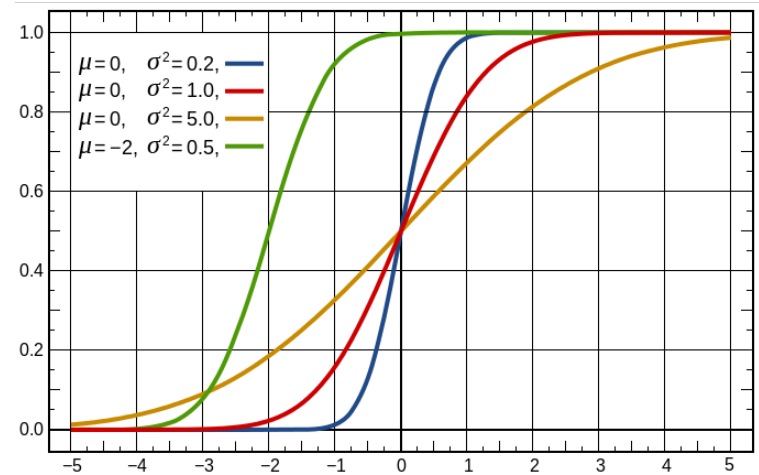
$$F(x) = \int_{-\infty}^x dP = \int_{-\infty}^x f(x') dx'$$

$F(x)$ changes from 0 to 1.

PDF



CDF



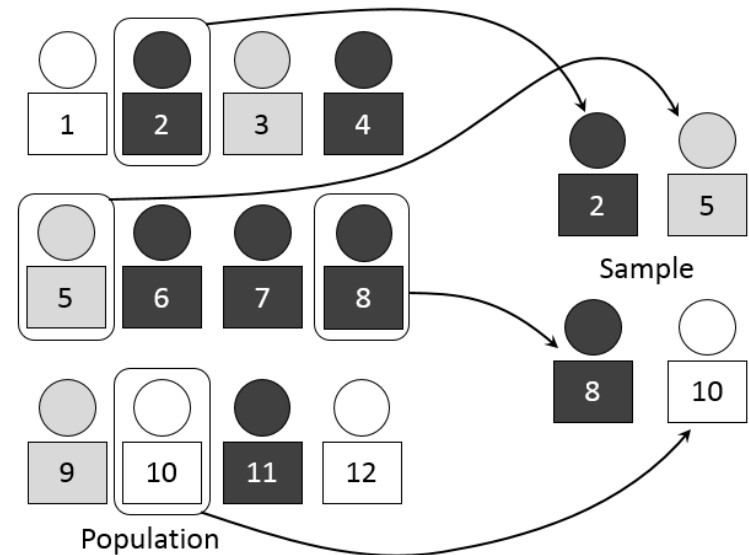
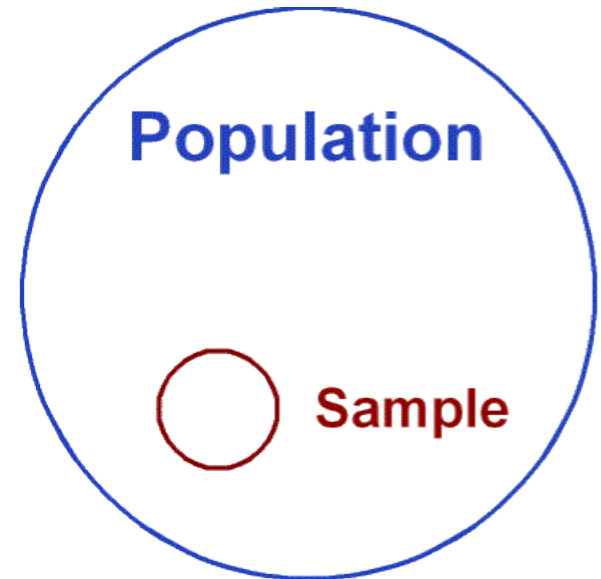
Math: sampling

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 ξ (uniformly distributed between 0 and 1)
2. Use ξ 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 *inverse method* is done using the inverse of the Cumulative Distribution Function $F(x)$.

1. Generate ξ (uniformly distributed between 0 and 1)
2. The cumulative probability of the event assumed equal to ξ : $F(x) = \xi$
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 *numerically* or by *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:

$$\Sigma_t = dP/dx$$

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 ξ (uniformly distributed between 0 and 1)



```
% 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 (Σ_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 XSs which equalize total XSs 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).

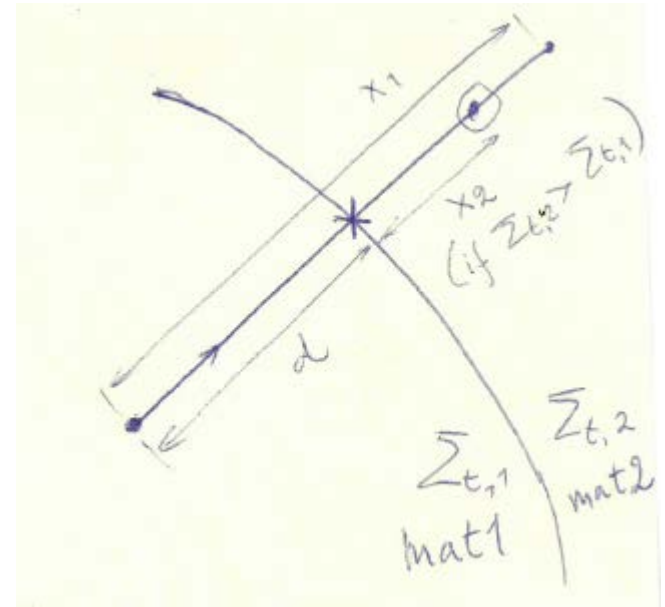
To re-adjust the coordinate of the next collision:


- 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 *without* handling the surface crossings

Is an *acceptance-rejection* technique.

Proposed by Woodcock in the 1960s.

Used in *Serpent* Monte Carlo code as a basic algorithm (optional in other codes)

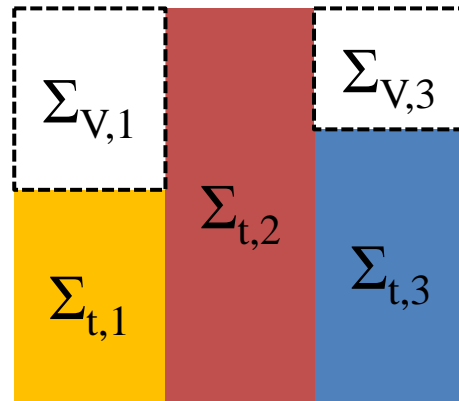
Based on a concept of *virtual collision* (or *pseudo-scattering*)

Scattering reaction (fictitious) in which angular and energy distributions are characterised by δ -functions ($\delta(E_0)$ and $\delta(\mathbf{\Omega}_0)$) and state of neutron is completely preserved

Neutron tracking: delta-tracking

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

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



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:

$$\Sigma_V(\mathbf{r}, E) = \Sigma_m(E) - \Sigma_t(\mathbf{r}, E)$$

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 ξ and comparing it with

$$P = \Sigma_V(\mathbf{r}, E)/\Sigma_m(E)$$

$P > \xi$ – 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:

$$R = \int_t \int_V \int_E f(\mathbf{r}, E) \phi(\mathbf{r}, E) dt dV dE$$

where $f(\mathbf{r}, E)$ is response function, e.g. 1 (to estimate flux) or Σ_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

$$\lim_{n \rightarrow \infty} \sigma = 0$$



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:

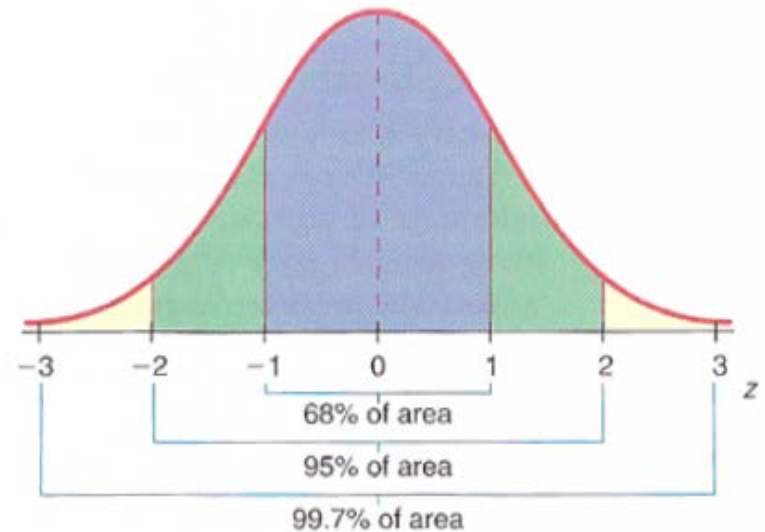
- distribution is the same for each term in the sum
- values are independent
- both mean and standard deviation exist and are finite

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 ± 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_{born} (fixed value = size of the batch).

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

```
% 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 ξ ;
- 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 ξ
- 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

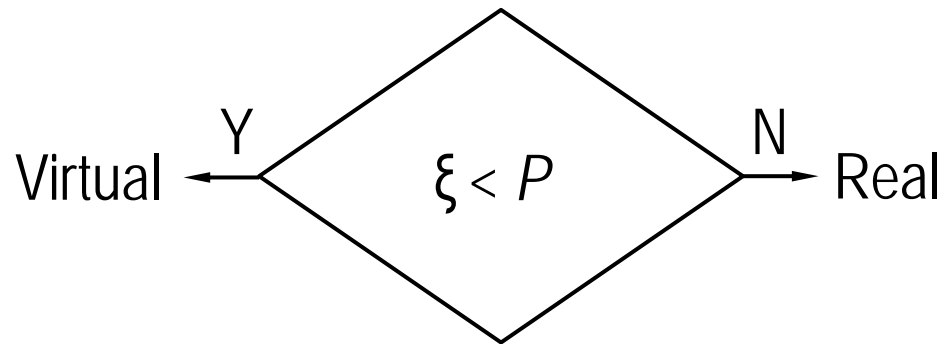
Interactions: real or virtual

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

$$P = \Sigma_V(\mathbf{r}, E) / \Sigma_m(E)$$

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



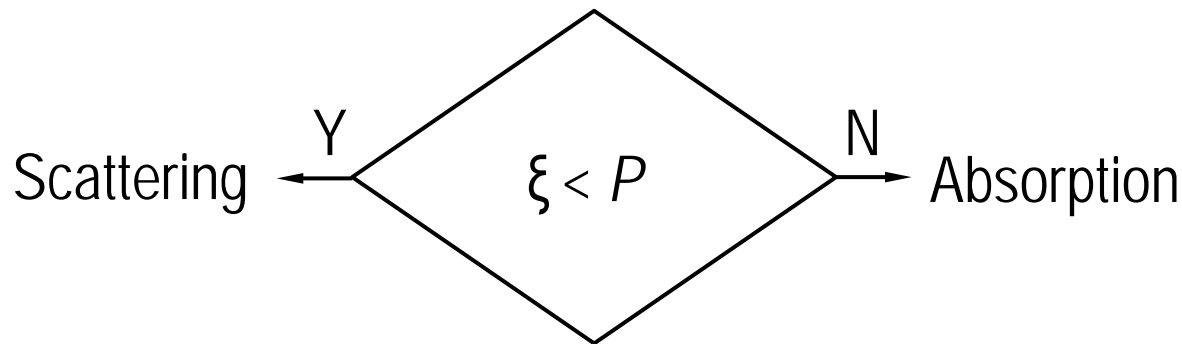
Interactions: collision type

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

- scattering
- absorption (capture + fission)

$$P = \Sigma_S(\mathbf{r}, E) / \Sigma_t(\mathbf{r}, E)$$

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



* (n,2n) reaction not considered

Interactions: scattering

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

Scattering assumptions:

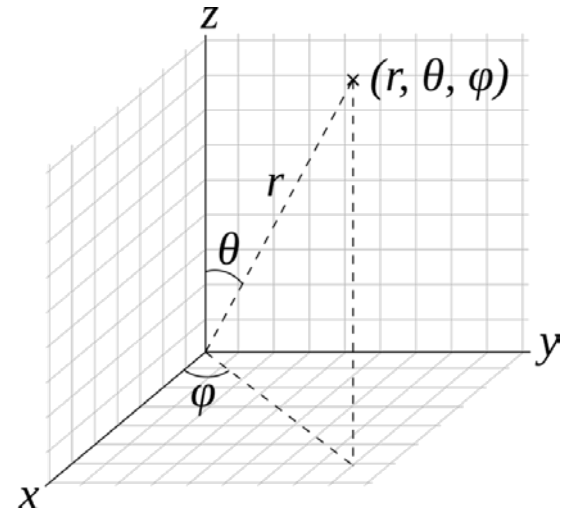
- **Isotropic**: new direction and energy are sampled independently.
- **Anisotropic**: new direction and new energy are not independent (not considered here, see Leppänen pp. 105-111 for more details)

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 = acos(2*rand()-1);  
phi = 2.0*pi*rand();  
dirX = sin(teta)*cos(phi);  
dirY = sin(teta)*sin(phi);  
x(iNeutron) = x(iNeutron) + freePath * dirX;  
y(iNeutron) = y(iNeutron) + freePath * 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 >= 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):

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

```
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 $F(E) \geq \xi$

```
iGroup(iNeutron) = find(cumsum(fuel.chi) >= rand(), 1, 'first');
```

MATLAB exercise

```

22  %-----
23  % Number of source neutrons
24 -   numNeutrons_born = 100;                                % INPUT
25
26  % Number of inactive source cycles to skip before starting k-eff
27  % accumulation
28 -   numCycles_inactive = 100;                              % INPUT
29
30  % Number of active source cycles for k-eff accumulation
31 -   numCycles_active = 2000;                              % INPUT
32
33  % Size of the square unit cell
34 -   pitch = 3.6; %cm                                       % INPUT
35
36  %-----
37  % Path to macroscopic cross section data:
38 -   path(path, '..\02.Macro.XS.421g');
39  % Fill the structures fuel, clad and cool with the cross sections data
40 -   fuel = macro421_UO2_03__900K;                          % INPUT
41 -   clad = macro421_Zry__600K;                             % INPUT
42 -   cool = macro421_H2OB__600K;                            % INPUT
43
44  % Define the majorant: the maximum total cross section vector
45 -   SigTmax = max([fuel.SigT; clad.SigT; cool.SigT]);
46
47  % Number of energy groups
48 -   ng = fuel.ng;

```

```

50 %-----
51 % Detectors
52 - detectS = zeros(1,ng);
53 Initialize your new detector here
54 %-----
55 % Four main vectors describing the neutrons in a batch
56 - x = zeros(1,numNeutrons_born*2);
57 - y = zeros(1,numNeutrons_born*2);
58 - weight = ones(1,numNeutrons_born*2);
59 - iGroup = ones(1,numNeutrons_born*2);
60
61 %-----
62 % Neutrons are assumed born randomly distributed in the cell with weight 1
63 % with sampled fission energy spectrum
64 - numNeutrons = numNeutrons_born;
65 - for iNeutron = 1:numNeutrons
66 -     x(iNeutron) = rand()*pitch;
67 -     y(iNeutron) = rand()*pitch;
68 -     weight(iNeutron) = 1;
69     % Sample the neutron energy group
70     iGroup(iNeutron) = find(cumsum(fuel.chi) >= rand(), 1, 'first');
71 - end
72
73 %-----
74 % Prepare vectors for keff and standard deviation of keff
75 - keff_expected = ones(1,numCycles_active);
76 - sigma_keff = zeros(1,numCycles_active);
77 - keff_active_cycle = ones(1,numCycles_active);
78 - virtualCollision = false;

```



```

80 % Main (power) iteration loop
81 - for iCycle = 1:(numCycles_inactive + numCycles_active)
82
83 % Normalize the weights of the neutrons to make the total weight equal to
84 % numNeutrons_born (equivalent to division by keff_cycle)
85 - weight = (weight ./ sum(weight,2)) * numNeutrons_born;
86 - weight0 = weight;
87
88 %-----
89 % Loop over neutrons
90 - for iNeutron = 1:numNeutrons
91
92 -     absorbed = false;
93
94 %-----
95 % Neutron random walk cycle: from emission to absorption
96
97 - while ~absorbed
98
99 % Sample free path length according to the Woodcock method
100 -     freePath = -log(rand())/SigTmax(iGroup(iNeutron));
101
102 -     if ~virtualCollision
103 -         % Sample the direction of neutron flight assuming both
104 -         % fission and scattering are isotropic in the lab (a strong
105 -         % assumption!)
106 -         teta = pi*rand();
107 -         phi = 2.0*pi*rand();
108 -         dirX = sin(teta)*cos(phi);
109 -         dirY = sin(teta)*sin(phi);
110 -     end

```

```

112 % Fly
113 -   x(iNeutron) = x(iNeutron) + freePath * dirX;
114 -   y(iNeutron) = y(iNeutron) + freePath * dirY;
115
116 % If outside the cell, find the corresponding point inside the
117 % cell
118 -   while x(iNeutron) < 0, x(iNeutron) = x(iNeutron) + pitch; end
119 -   while y(iNeutron) < 0, y(iNeutron) = y(iNeutron) + pitch; end
120 -   while x(iNeutron) > pitch, x(iNeutron) = x(iNeutron) - pitch; end
121 -   while y(iNeutron) > pitch, y(iNeutron) = y(iNeutron) - pitch; end
122
123 % Find the total and scattering cross sections
124 -   if x(iNeutron) > 0.9 && x(iNeutron) < 2.7 % INPUT
125 -       SigA = fuel.SigF(iGroup(iNeutron)) + fuel.SigC(iGroup(iNeutron)) + fuel.SigL(iGroup(iNeutron));
126 -       SigS = fuel.SigS{1+0}(iGroup(iNeutron),:)' ;
127 -       SigP = fuel.SigP(iGroup(iNeutron));
128 -   elseif x(iNeutron) < 0.7 || x(iNeutron) > 2.9 % INPUT
129 -       SigA = cool.SigC(iGroup(iNeutron)) + cool.SigL(iGroup(iNeutron));
130 -       SigS = cool.SigS{1+0}(iGroup(iNeutron),:)' ;
131 -       SigP = 0;
132 -   else
133 -       SigA = clad.SigC(iGroup(iNeutron)) + clad.SigL(iGroup(iNeutron));
134 -       SigS = clad.SigS{1+0}(iGroup(iNeutron),:)' ;
135 -       SigP = 0;
136 -   end
137
138 % Find the other cross sections ...
139 % ... scattering
140 -   SigS_sum = sum(SigS);
141 % ... total
142 -   SigT = SigA + SigS_sum;
143 % ... virtual
144 -   SigV = SigTmax(iGroup(iNeutron)) - SigT;

```

Insert your new detector here

```
146     % Sample the type of the collision: virtual (do nothing) or real
147     if SigV/SigTmax(iGroup(iNeutron)) >= rand() % virtual collision
148
149         virtualCollision = true;
150
151     else % real collision
152
153         virtualCollision = false;
154
155     % Sample type of the collision: scattering or absorption
156     if SigS_sum/SigT >= rand() % isotropic scattering
157
158         % Score scatterings with account for weight divided by the
159         % total scattering cross section
160         detectS(iGroup(iNeutron)) = detectS(iGroup(iNeutron)) + weight(iNeutron)/SigS_sum;
161
162         % Sample the energy group of the secondary neutron
163         iGroup(iNeutron) = find(cumsum(SigS)/SigS_sum >= rand(), 1, 'first');
164
165     else % absorption
166
167         absorbed = true;
168
169         % Neutron is converted to the new fission neutron with
170         % the weight increased by eta
171         weight(iNeutron) = weight(iNeutron) * (SigP/SigA);
172
173         % Sample the energy group for the new-born neutron
174         iGroup(iNeutron) = find(cumsum(fuel.chi) >= rand(), 1, 'first');
175
176     end % scattering or absorption
177 end % virtual or real
178 end % of neutron random walk cycle: from emission to absorption
179 end % of loop over neutrons
```

```

181 %-----
182 % Russian roulette
183 - for iNeutron = 1:numNeutrons
184 -     terminateP = 1 - weight(iNeutron)/weight0(iNeutron);
185 -     if terminateP >= rand()
186 -         weight(iNeutron) = 0; % killed
187 -     elseif terminateP > 0
188 -         weight(iNeutron) = weight0(iNeutron); % restore the weight
189 -     end
190 - end
191
192 %-----
193 % Clean up absorbed or killed neutrons
194
195 - x(weight == 0) = [];
196 - y(weight == 0) = [];
197 - iGroup(weight == 0) = [];
198 - weight(weight == 0) = [];
199 - numNeutrons = size(weight,2);
200

```

```

201 %-----
202 % Split too "heavy" neutrons
203
204 numNew = 0;
205 for iNeutron = 1:numNeutrons
206     if weight(iNeutron) > 1
207         % Truncated integer value of the neutron weight
208         N = floor(weight(iNeutron));
209         % Sample the number of split neutrons
210         if weight(iNeutron)-N > rand(), N = N + 1; end
211         % Change the weight of the split neutron
212         weight(iNeutron) = weight(iNeutron)/N;
213         % Introduce new neutrons
214         for iNew = 1:N-1
215             numNew = numNew + 1;
216             x(numNeutrons + numNew) = x(iNeutron);
217             y(numNeutrons + numNew) = y(iNeutron);
218             weight(numNeutrons + numNew) = weight(iNeutron);
219             iGroup(numNeutrons + numNew) = iGroup(iNeutron);
220         end
221     end
222 end
223 % Increase the number of neutrons
224 numNeutrons = numNeutrons + numNew;

```

```

226 %-----
227 % k-eff in a cycle equals the total weight of the new generation over
228 % the total weight of the old generation (the old generation weight =
229 % numNeutronsBorn)
230 keff_cycle = sum(weight,2)/sum(weight0,2);
231
232 iActive = iCycle - numCycles_inactive;
233 if iActive <= 0
234     fprintf('Inactive cycle = %3i/%3i; k-eff cycle = %8.5f; numNeutrons = %3i\n', ...
235           iCycle,numCycles_inactive,keff_cycle,numNeutrons);
236 else
237     % k-effective of the cycle
238     keff_active_cycle(iActive) = keff_cycle;
239
240     % k-effective of the problem
241     keff_expected(iActive) = mean(keff_active_cycle(1:iActive));
242
243     % Standard deviation of k-effective
244     sigma_keff(iActive) = sqrt( sum( ( keff_active_cycle(1:iActive) - keff_expected(iActive) ).^2 ) ...
245                               / max(iActive-1,1) / iActive );
246
247     fprintf('Active cycle = %3i/%3i; k-eff cycle = %8.5f; numNeutrons = %3i; k-eff expected = %9.5f; si
248           iCycle-numCycles_inactive, numCycles_active, keff_cycle, numNeutrons, keff_expected(iActive)
249     end
250
251 end % of main (power) iteration

```