POLITECNICO
MILANO 1863

Nuclear Reactors Group

Reactor modelling with OpenMC

Prof. Enrico Padovani
Lorenzo Loi (lorenzo.loi@polimi.it)

Fission Reactor Physics I

21st-29th May, 2025

Neutronic modelling

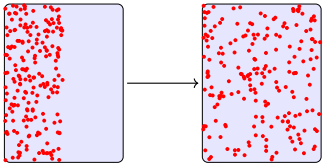
Introduction

Nuclear engineers ⇒ **How to model neutrons in a reactor?**

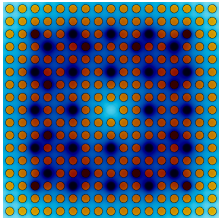
Point Kinetics

$$\begin{cases} \frac{dP}{dt} = \frac{\rho-\beta}{\Lambda} P + \sum_j \lambda_j C_j \\ \frac{dC_j}{dt} = \frac{\beta}{\Lambda} P - \lambda_j C_j \end{cases}$$

Diffusion theory



Transport/Monte Carlo



Accuracy

Computational efficiency

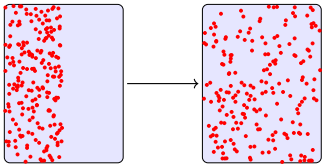
Introduction

Nuclear engineers ⇒ **How to model neutrons in a reactor?**

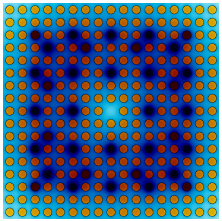
Point Kinetics

$$\begin{cases} \frac{dP}{dt} = \frac{\rho-\beta}{\Lambda} P + \sum_j \lambda_j C_j \\ \frac{dC_j}{dt} = \frac{\beta}{\Lambda} P - \lambda_j C_j \end{cases}$$

Diffusion theory



Transport/Monte Carlo



Accuracy

Computational efficiency

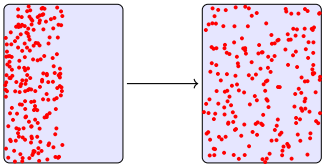
Introduction

Nuclear engineers ⇒ **How to model neutrons in a reactor?**

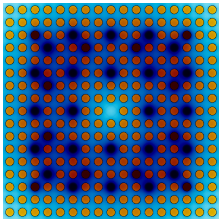
Point Kinetics

$$\begin{cases} \frac{dP}{dt} = \frac{\rho-\beta}{\Lambda} P + \sum_j \lambda_j C_j \\ \frac{dC_j}{dt} = \frac{\beta}{\Lambda} P - \lambda_j C_j \end{cases}$$

Diffusion theory



Transport/Monte Carlo



Accuracy

Computational efficiency

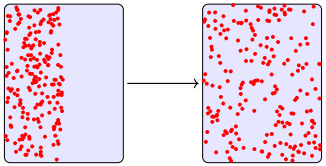
Introduction

Nuclear engineers ⇒ **How to model neutrons in a reactor?**

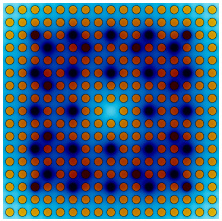
Point Kinetics

$$\begin{cases} \frac{dP}{dt} = \frac{\rho-\beta}{\Lambda} P + \sum_j \lambda_j C_j \\ \frac{dC_j}{dt} = \frac{\beta}{\Lambda} P - \lambda_j C_j \end{cases}$$

Diffusion theory



Transport/Monte Carlo



Accuracy

Computational efficiency

Transport theory: Boltzmann equation

This is the most general way of representing neutrons in a system

$$\underbrace{\frac{1}{v}\frac{\partial\varphi(\mathbf{r},E,\mathbf{\Omega},t)}{\partial t}}_{\text{Time rate of neutron density}} = -\underbrace{\mathbf{\Omega}\cdot\nabla\varphi(\mathbf{r},E,\mathbf{\Omega},t)}_{\text{Leakage}} - \underbrace{\Sigma_t(\mathbf{r},E)\varphi(\mathbf{r},E,\mathbf{\Omega},t)}_{\text{Collision}} + \underbrace{Q(\mathbf{r},E,\mathbf{\Omega},t)}_{\text{Source}}$$

Transport theory: Boltzmann equation

This is the most general way of representing neutrons in a system

$$\underbrace{\frac{1}{v}\frac{\partial\varphi(\mathbf{r},E,\boldsymbol{\Omega},t)}{\partial t}}_{\text{Time rate of neutron density}} = -\underbrace{\boldsymbol{\Omega}\cdot\nabla\varphi(\mathbf{r},E,\boldsymbol{\Omega},t)}_{\text{Leakage}} - \underbrace{\Sigma_t(\mathbf{r},E)\varphi(\mathbf{r},E,\boldsymbol{\Omega},t)}_{\text{Collision}} + \underbrace{Q(\mathbf{r},E,\boldsymbol{\Omega},t)}_{\text{Source}}$$

Where

$$Q(\mathbf{r},E,\boldsymbol{\Omega},t) = \underbrace{\iint \Sigma_s(\mathbf{r};\boldsymbol{\Omega}',E'\rightarrow\boldsymbol{\Omega},E)\varphi(\mathbf{r},E',\boldsymbol{\Omega}',t)\,d\boldsymbol{\Omega}'\,dE'}_{\text{Scattering}}$$

$$\underbrace{\frac{1}{4\pi}\int \chi(E)\,\nu\Sigma_f(\mathbf{r},E')\varphi(\mathbf{r},E',\boldsymbol{\Omega}',t)\,dE'}_{\text{Fission}}$$

Transport theory: Boltzmann equation

This is the most general way of representing neutrons in a system

$$\underbrace{\frac{1}{v}\frac{\partial\varphi(\mathbf{r},E,\boldsymbol{\Omega},t)}{\partial t}}_{\text{Time rate of neutron density}} = -\underbrace{\boldsymbol{\Omega}\cdot\nabla\varphi(\mathbf{r},E,\boldsymbol{\Omega},t)}_{\text{Leakage}} - \underbrace{\Sigma_t(\mathbf{r},E)\varphi(\mathbf{r},E,\boldsymbol{\Omega},t)}_{\text{Collision}} + \underbrace{Q(\mathbf{r},E,\boldsymbol{\Omega},t)}_{\text{Source}}$$

Where

$$Q(\mathbf{r},E,\boldsymbol{\Omega},t) = \underbrace{\iint \Sigma_s(\mathbf{r};\boldsymbol{\Omega}',E'\rightarrow\boldsymbol{\Omega},E)\varphi(\mathbf{r},E',\boldsymbol{\Omega}',t)\,d\boldsymbol{\Omega}'\,dE'}_{\text{Scattering}}$$

Note: Integro-differential equation in 7 variables: very difficoult to solve! Let's consider the stationary version.

$$\frac{1}{4\pi}\underbrace{\int \chi(E)\,\nu\Sigma_f(\mathbf{r},E')\varphi(\mathbf{r},E',\boldsymbol{\Omega}',t)\,dE'}_{\text{Fission}}$$

Eigenvalue problem

The stationary version of the Boltzmann equation can be view as an eigenvalue problem

$$\boldsymbol{\Omega} \cdot \nabla \varphi(\mathbf{r}, E, \boldsymbol{\Omega}) + \Sigma_t(\mathbf{r}, E)\varphi(\mathbf{r}, E, \boldsymbol{\Omega}) = \frac{1}{k} Q(\mathbf{r}, E, \boldsymbol{\Omega})$$

$$\Downarrow$$

This is the **mathematical problem** that the MC wants to solve

Eigenvalue formulation

$$\mathbb{L}\,\varphi = \tfrac{1}{k}\,\mathbb{F}\,\varphi$$

Eigenvalue problem

The stationary version of the Boltzmann equation can be view as an eigenvalue problem

$$\mathbf{\Omega} \cdot \nabla \varphi(\mathbf{r}, E, \mathbf{\Omega}) + \Sigma_t(\mathbf{r}, E)\varphi(\mathbf{r}, E, \mathbf{\Omega}) = \frac{1}{k} Q(\mathbf{r}, E, \mathbf{\Omega})$$

$$\Downarrow$$

This is the **mathematical problem** that the MC wants to solve

Eigenvalue formulation

$$\mathbb{L}\,\varphi = \tfrac{1}{k}\,\mathbb{F}\,\varphi$$

How is the MC going to solve this problem?

Eigenvalue problem

The stationary version of the Boltzmann equation can be view as an eigenvalue problem

$$\boldsymbol{\Omega} \cdot \nabla \varphi(\mathbf{r}, E, \boldsymbol{\Omega}) + \Sigma_t(\mathbf{r}, E)\varphi(\mathbf{r}, E, \boldsymbol{\Omega}) = \frac{1}{k} Q(\mathbf{r}, E, \boldsymbol{\Omega})$$

$$\Downarrow$$

This is the **mathematical problem** that the MC wants to solve

Eigenvalue formulation

$$\mathbb{L}\,\varphi = \frac{1}{k}\,\mathbb{F}\,\varphi$$

How is the MC going to solve this problem?

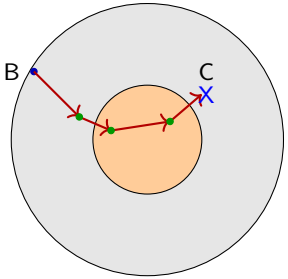
Idea

Simulate the $\overbrace{\text{life of the neutrons}}^{1}$ with $\overbrace{\text{probability density functions}}^{2}$ and perform $\underbrace{\text{statistical estimations}}_{3}$ of the quantities of interest

Transport with Monte Carlo - life

Idea

Simulate the **life of the neutrons** with probability density functions and perform statistical estimations of the quantities of interest

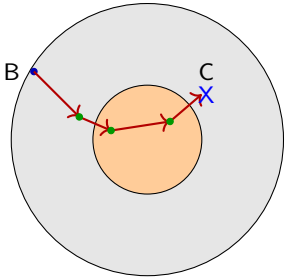


Life # 1

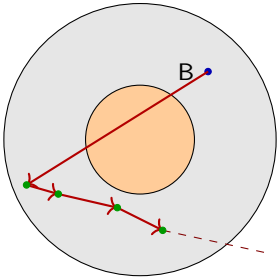
Transport with Monte Carlo - life

Idea

Simulate the **life of the neutrons** with probability density functions and perform statistical estimations of the quantities of interest



Life # 1

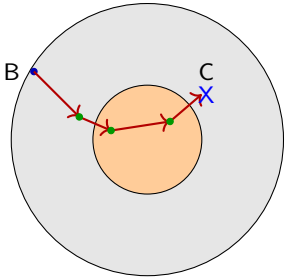


Life # 2

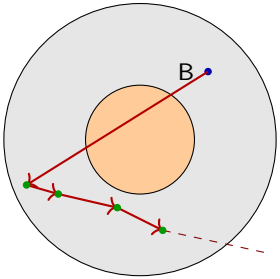
Transport with Monte Carlo - life

Idea

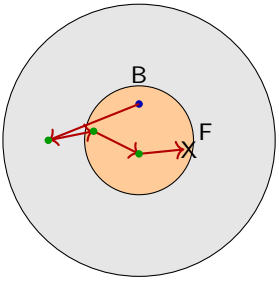
Simulate the **life of the neutrons** with probability density functions and perform statistical estimations of the quantities of interest



Life # 1



Life # 2



Life # 3

Transport with Monte Carlo - sampling

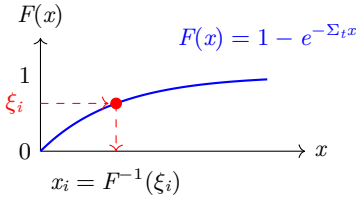
Idea

Simulate the life of the neutrons with **probability density functions** and perform statistical estimations of the quantities of interest

All the interactions in the neutron lifetime occur with a given probability (from XS). By knowing the CDF of an event, it is possible to *sample* from that using the *inverse technique*

1. Sample $\xi_i \in (0, 1)$
2. Set $\xi_i = F(x_i)$
3. Find the interaction position x_i inverting F

Namely, $x_i = -\frac{1}{\Sigma_t} \ln(1 - \xi_i)$



Event: neutron interaction in x_j

CDF of the event $= F(x_j)$

Probability that the neutron collided between 0 and x_j

Transport with Monte Carlo - collection

Idea

Simulate the life of the neutrons with probability density functions and perform **statistical estimations** of the quantities of interest

Simulating N groups of 100 neutrons:

- Captures $\{ \overbrace{23}^{g_1}, \overbrace{27}^{g_2}, \overbrace{32}^{g_3}, \overbrace{28}^{g_4}, \cdots, \overbrace{35}^{g_{N-1}}, \overbrace{42}^{g_N} \}$
- Fissions $\{ 33, 41, 30, 21, \cdots, 40, 22 \}$
- Leaks $\{ 44, 32, 38, 51, \cdots, 25, 36 \}$

$\Rightarrow$ Collection of information during the process

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

Eigenvalue problem

The stationary version of the Boltzmann equation can be view as an eigenvalue problem

$$\mathbf{\Omega} \cdot \nabla \varphi(\mathbf{r}, E, \mathbf{\Omega}) + \Sigma_t(\mathbf{r}, E)\varphi(\mathbf{r}, E, \mathbf{\Omega}) = \frac{1}{k} Q(\mathbf{r}, E, \mathbf{\Omega})$$

$$\Downarrow$$

This is the **mathematical problem** that the MC wants to solve

Eigenvalue formulation

$$\mathbb{L}\,\varphi = \tfrac{1}{k}\,\mathbb{F}\,\varphi$$

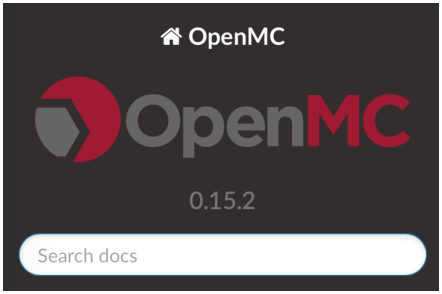
How is the MC going to solve this problem?

Note

The Monte Carlo is a **stochastic method**: it is able to solve the Boltzmann problem without any direct mathematical strategy!

OpenMC

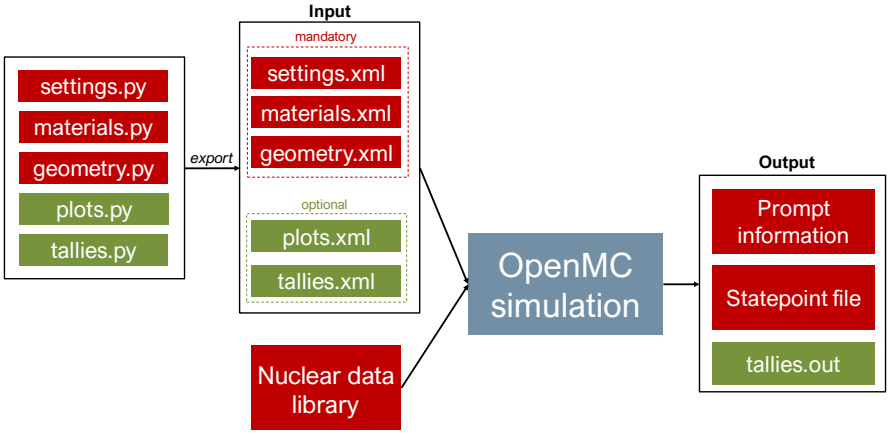
Code description



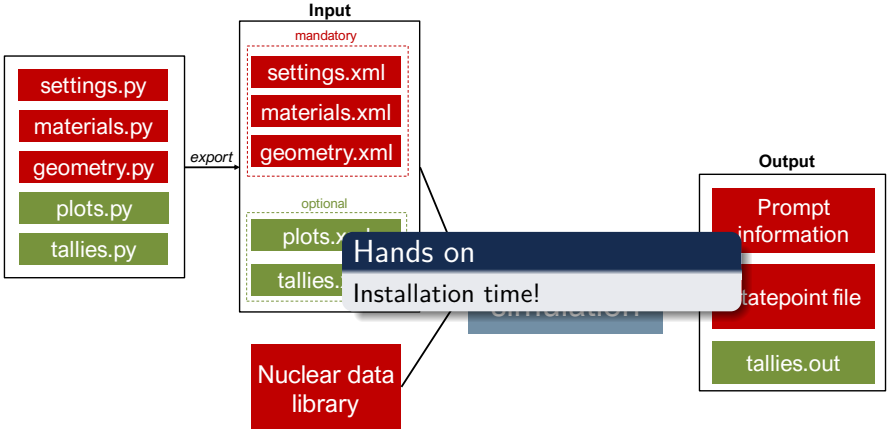
- Extended and updated documentation

- **Monte Carlo Particle Transport:** OpenMC performs high-fidelity neutron and photon transport using continuous-energy or multigroup methods.
- **Flexible Geometry Modeling:** Supports constructive solid geometry (CSG) and integration with CAD-based models
- **Advanced Physics Models:** Includes resonance treatment, temperature-dependent cross sections, and support for delayed neutrons and photon production.
- **Burnup and Depletion:** Fully integrated depletion solver with support for multi-step irradiation, decay chains, and fission product tracking.
- **Python API:** Extensive scripting capabilities for model setup and post-processing via Python.

Code infrastructure



Code infrastructure

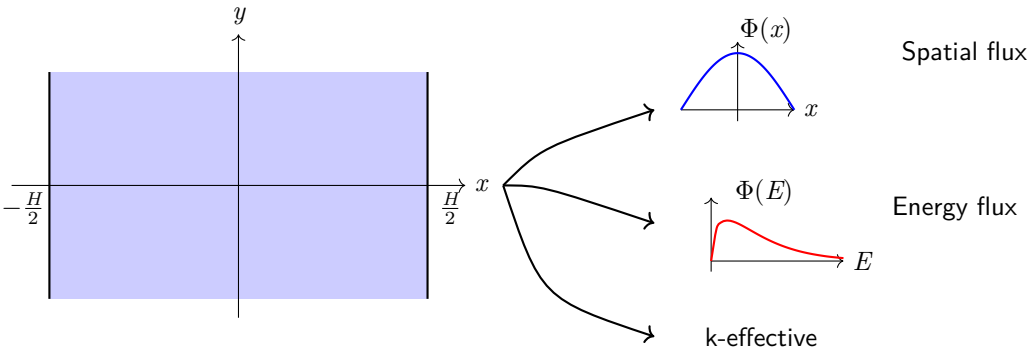


First example: homogeneous slab modelling

Description

Aim

Model a 1D slab having length H, evaluating its criticality, spatial flux and energy flux.



Materials

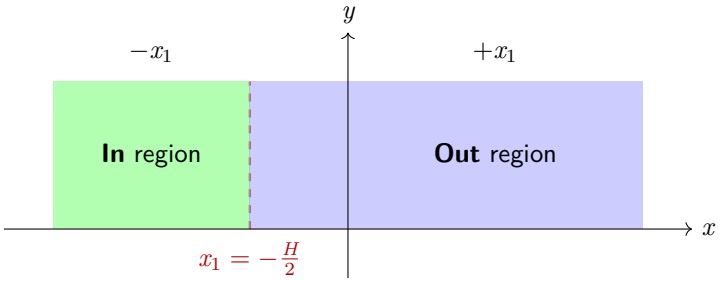

```
import numpy as np
import openmc

###############################################################
# ----- MATERIALS

fuel = openmc.Material(name='Homogeneous U-water mixture')
fuel.set_density('g/cm3', 4)
fuel.add_element('U', 1, enrichment=1.0) # 1% enrichment
fuel.add_element('O', 4)
fuel.add_nuclide('H1', 4)

# Create materials object
materials = openmc.Materials([fuel])

materials.export_to_xml()
```

- Isotopic compositions can be specified in either atomic concentrations (i.e., 'ao') or mass concentration (i.e., 'wo').
- Materials where thermal scattering plays an important rule must be aligned with the associated $S(\alpha, \beta)$ library.
- All the materials in the simulations have to be classified in a list and then exported to xml.

Geometry (I)

Constructive Solid Geometry (CSG) approach:

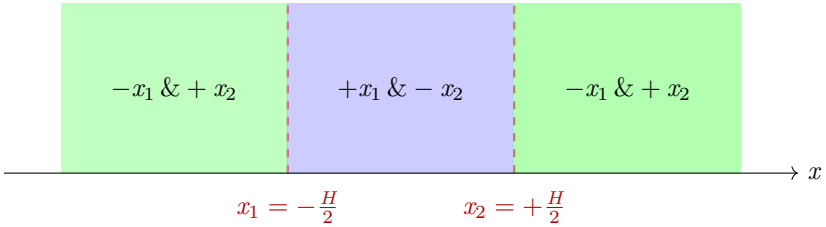
1. Define **surfaces** in the space;
2. Define **regions** with CSG syntax
3. Define **cells**: zones constrained by the surface regions and filled with materials.



Geometry (II)

Constructive Solid Geometry (CSG) approach:

1. Define **surfaces** in the space;
2. Define **regions** with CSG syntax
3. Define **cells**: zones constrained by the surface regions and filled with materials.



Geometry (III)


```
########################################
# ---- GEOMETRY

H = 40 # Total slab length (cm)

# Define slab surfaces
left = openmc.XPlane(x0=-H/2, boundary_type='vacuum')
right = openmc.XPlane(x0=H/2, boundary_type='vacuum')

# Define the slab cell
slab_region = +left & -right
slab_cell = openmc.Cell(region=slab_region)
slab_cell.fill = fuel

# Create universe and geometry
universe = openmc.Universe(cells=[slab_cell])
geometry = openmc.Geometry(universe)

geometry.export_to_xml()
```

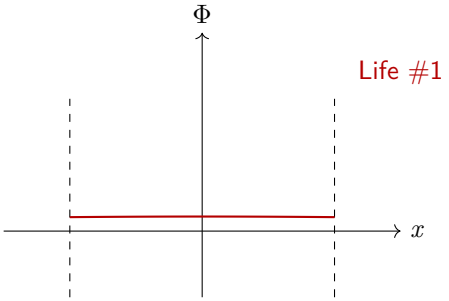
- The system is assembled through **constructive solid geometry**;
- First, surfaces are defined: a slab is identified by a couple of X-planes
- Vacuum BC will end the life of neutrons that are crossing that surface. Other type of BC: *reflective* or *periodic*
- Cells are defined, filled with materials and bounded by surfaces (convention: - is **inside**, + is **outside**)

Settings


```
################################################
# ---- SETTINGS

# Indicate the statistics: particles, active and inactive
    cycles
settings = openmc.Settings()
settings.batches = 100 # Total number of cycles
settings.inactive = 10 # Inactive cycles
settings.particles = 100 # Neutron per cycle

settings.export_to_xml()
```

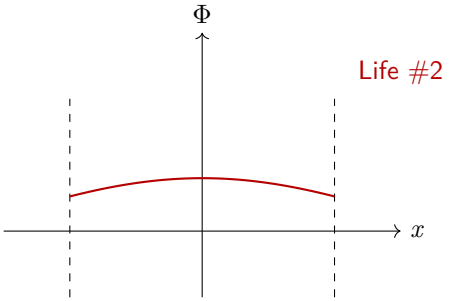
- Monte Carlo settings need to be specified;
- At first, neutrons will be sampled in an uniform way within the fissionable zones;
- For a certain number of cycles (*inactive*), information are not collected;
- During the remaining part of the run (i.e., *active cycles*) information are collected.

Importance of the inactive cycles



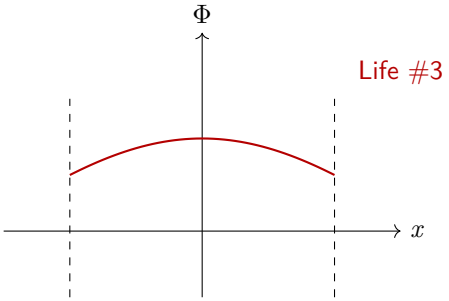
At the beginning of the simulation, the MC does not have any "physical information" about the fission source: before starting to collect data, **fission source** need to be **converged**.

Importance of the inactive cycles



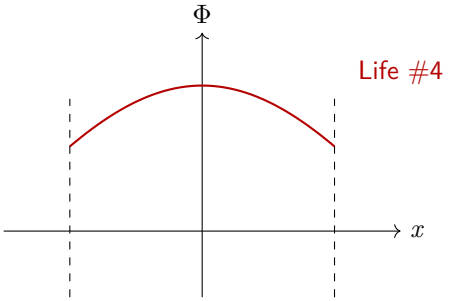
At the beginning of the simulation, the MC does not have any "physical information" about the fission source: before starting to collect data, **fission source** need to be **converged**.

Importance of the inactive cycles



At the beginning of the simulation, the MC does not have any "physical information" about the fission source: before starting to collect data, **fission source** need to be **converged**.

Importance of the inactive cycles



At the beginning of the simulation, the MC does not have any "physical information" about the fission source: before starting to collect data, **fission source** need to be **converged**.

Optimal value

There is not an easy way to estimate the correct number of inactive cycles. Precise analysis uses the **Shannon entropy**

First run!

The first simulation can be initialised

```
##############
# ---- RUN
openmc.run()
```

```
    2698/1    1.16491    1.24752 +/- 0.00139
    2699/1    1.28760    1.24754 +/- 0.00139
    2700/1    1.29664    1.24756 +/- 0.00139
Creating state point statepoint.2700.h5...

=======================>     TIMING STATISTICS     <=========

Total time for initialization     = 2.7385e+00 seconds
  Reading cross sections          = 2.7168e+00 seconds
Total time in simulation          = 4.2547e+01 seconds
  Time in transport only          = 4.1779e+01 seconds
  Time in inactive batches        = 3.5507e+00 seconds
  Time in active batches          = 3.8996e+01 seconds
  Time synchronizing fission bank = 2.1195e-01 seconds
    Sampling source sites         = 1.3403e-01 seconds
    SEND/RECV source sites        = 6.5285e-02 seconds
  Time accumulating tallies       = 1.8643e-01 seconds
  Time writing statepoints        = 1.5320e-02 seconds
Total time for finalization       = 3.1928e-04 seconds
Total time elapsed                = 4.5381e+01 seconds
Calculation Rate (inactive)       = 70408.5 particles/second
Calculation Rate (active)         = 28208 particles/second

============================>     RESULTS     <===============

k-effective (Collision)     = 1.24801 +/- 0.00118
k-effective (Track-length)  = 1.24756 +/- 0.00139
k-effective (Absorption)    = 1.24914 +/- 0.00097
Combined k-effective        = 1.24869 +/- 0.00084
Leakage Fraction            = 0.00000 +/- 0.00000
```

At the end, information are stored in the **statepoint file**

```
sp = openmc.StatePoint('statepoint.100.h5')

# k eff from OpenMC
keff = sp.keff

print(f"k-eff = {keff.nominal_value:.5f} +/- {keff.std_dev:.3e}")
```

Warning!

If *openmc.run()* is performed multiple times, there is a procedure to follow:

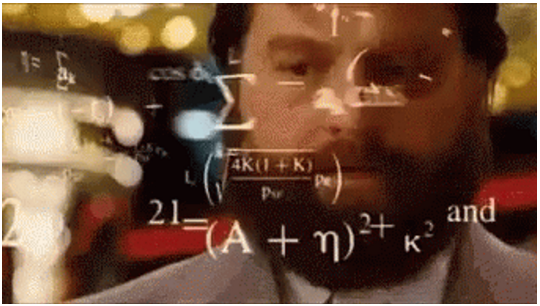
- Option 1: restart the kernel;
- Option 2: remove the file *summary.h5* + change name to the *statepoint.100.h5* file
- Option 3: include *sp.close()* before the next run

Your turn!

Ex1: Homogeneous slab

Find a way to make the system critical (i.e., k-eff $\simeq 1$)

Ideas?



Your turn!

Ex1: Homogeneous slab

Find a way to make the system critical (i.e., k-eff $\simeq 1$)

Ideas?

To modify the k-eff, you could modify the following

- Change composition: increase the 235U enrichment ($\simeq$ 2.5 % 235U @ 40 cm)
- Increase the slab width ($\simeq$ 400 cm @ 1 % 235U)

Other possibilities:

- Change temperature
- ...

Tallies

If no specifications are given, OpenMC will calculate only the k_{eff}. What about **flux**, **energy spectrum**, **scattering rate** etc.?

The **tallies** are the way the user *talks* with the code

Tally definition

$$R = \int_V \int_{4\pi} \int_{E_{g-1}}^{E_g} h(\mathbf{r}, E)\, \varphi(\mathbf{r}, \ , E)\, \mathrm{d}\mathbf{r}\, \mathrm{d}\ \mathrm{d}E$$

Where:

- The extremes of integration are called **filters** $\Rightarrow$ select the domain in which you want the information
- The response function h is called **score** $\Rightarrow$ select the kind of information

Tallies - filters

Full documentation **here**

Constructing Tallies

`openmc.Filter`	Tally modifier that describes phase-space and other characteristics.
`openmc.UniverseFilter`	Bins tally event locations based on the Universe they occurred in.
`openmc.MaterialFilter`	Bins tally event locations based on the Material they occurred in.
`openmc.MaterialFromFilter`	Bins tally event locations based on the Material they occurred in.
`openmc.CellFilter`	Bins tally event locations based on the Cell they occurred in.
`openmc.CellFromFilter`	Bins tally on which cell the particle came from.
`openmc.CellBornFilter`	Bins tally events based on which cell the particle was born in.
`openmc.CellInstanceFilter`	Bins tally events based on which cell instance a particle is in.
`openmc.CollisionFilter`	Bins tally events based on the number of collisions.
`openmc.SurfaceFilter`	Filters particles by surface crossing
`openmc.MeshFilter`	Bins tally event locations by mesh elements.
`openmc.MeshBornFilter`	Filter events by the mesh cell a particle originated from.
`openmc.MeshSurfaceFilter`	Filter events by surface crossings on a mesh.
`openmc.EnergyFilter`	Bins tally events based on incident particle energy.
`openmc.DelayedGroupFilter`	Bins fission events based on the produced neutron precursor groups.
`openmc.EnergyFunctionFilter`	Multiplies tally scores by an arbitrary function of incident energy.
`openmc.LegendreFilter`	Score Legendre expansion moments up to specified order.
`openmc.SpatialLegendreFilter`	Score Legendre expansion moments in space up to specified order.
`openmc.SphericalHarmonicsFilter`	Score spherical harmonic expansion moments up to specified order.
`openmc.TimeFilter`	Bins tally events based on the particle's time.
`openmc.ZernikeFilter`	Score Zernike expansion moments in space up to specified order.
`openmc.ZernikeRadialFilter`	Score the $m = 0$ (radial variation only) Zernike moments up to specified order.
`openmc.ParentNuclideFilter`	Bins tally events based on the parent nuclide
`openmc.ParticleFilter`	Bins tally events based on the particle type.
`openmc.RegularMesh`	A regular Cartesian mesh in one, two, or three dimensions
`openmc.RectilinearMesh`	A 3D rectilinear Cartesian mesh
`openmc.CylindricalMesh`	A 3D cylindrical mesh
`openmc.SphericalMesh`	A 3D spherical mesh
`openmc.UnstructuredMesh`	A 3D unstructured mesh

Tallies - scores

Full documentation **here**

Flux scores: units are particle-cm per source particle.

Score	Description
flux	Total flux.

Reaction scores: units are reactions per source particle.

Score	Description
absorption	Total absorption rate. For incident neutrons, this accounts for all reactions that do not produce secondary neutrons as well as fission. For incident photons, this includes photoelectric and pair production.
elastic	Elastic scattering reaction rate.
fission	Total fission reaction rate.
scatter	Total scattering rate.
total	Total reaction rate.

Response function examples:

- $h = 1 \rightarrow R = \iiint \varphi$, number of tracks per unit particle;
- $h = \Sigma_f \rightarrow R = \iiint \Sigma_f^i \varphi$ number of fissions per unit particle;
- $h = \Sigma_t \rightarrow R = \iiint \Sigma_t^i \varphi$ number of collisions per unit particle.

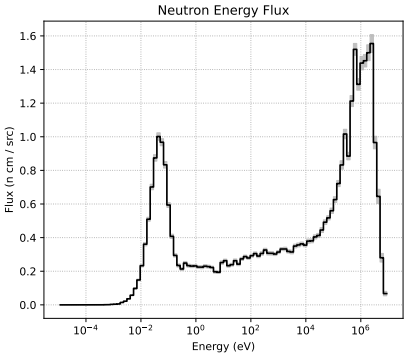
Tallies - in code


```
############################################################
tallies = openmc.Tallies()

# ── Filters ──
# 1. Cell
cell_filter = openmc.CellFilter(slab_cell)

# 2. Energy
energy_bins = np.logspace(-5, 7, num=100)  # 100 bins
energy_filter = openmc.EnergyFilter(energy_bins)

# ── Tally 1: Energy-dependent Flux ──
flux_tally = openmc.Tally(name='Phi(E)')
flux_tally.filters = [cell_filter, energy_filter]
flux_tally.scores = ['flux']

# ── Register the tallies ──
tallies = openmc.Tallies([flux_tally])

tallies.export_to_xml()
```

1. Choose filters
2. Choose scores
3. Export to xml

Tallies - read


```
# Energy-dependent flux
flux_tally = sp.get_tally(name='Phi(E)')
flux_vals = flux_tally.mean.flatten()
Uflux_vals = flux_tally.std_dev.flatten()

# Energy bins (midpoints)
energy_mid = 0.5 * (energy_bins[:-1] + energy_bins[1:])

# Plot the flux spectrum
plt.figure(figsize = (6,5))
plt.step(energy_mid, flux_vals, color = "black", where = "
    pre")
plt.fill_between(energy_mid, flux_vals - Uflux_vals,
    flux_vals + Uflux_vals, step='pre', color='black',
    alpha=0.25)
plt.xscale("log")
plt.xlabel("Energy (eV)")
plt.ylabel("Flux (n cm / src)")
plt.title("Neutron Energy Flux")
plt.grid(True, linestyle = ":")
plt.savefig("./phiE.pdf", format = "pdf", bbox_inches = "
    tight", dpi = 400)
plt.show()
```



Your turn!

Ex2: Homogeneous slab

Estimate the neutron mean free path

Hint: add some specific tallies, useful for the mean free path

```
#############################################################

# ---- Tally 1: Energy-dependent Flux ----
flux_tally = openmc.Tally(name='Phi(E)')
flux_tally.filters = [cell_filter, energy_filter]
flux_tally.scores = ['flux']

# ---- Tally 2: Find mean free path

RR_t = openmc.tally(name = "reaction rate ...")
RR_t.filters = [...]
RR_t.scores = ['...']

# ---- Register the tallies ----
tallies = openmc.Tallies([flux_tally, RR_t])

tallies.export_to_xml()
```

Your turn!

Ex2: Homogeneous slab

Estimate the neutron mean free path

Hint: add some specific tallies, useful for the mean free path

1. Find the total reaction rate $R = \iiint \Sigma_t \varphi$ with the 'total' score
2. Find the total flux $\Phi = \iiint \varphi$. Two options:
   - Or by specifying another tally, without energy filters
   - Or summing the energy flux previously obtained
3. The averaged total cross section is $\bar{\Sigma}_t = R/\Phi \quad (\text{cm}^{-1})$
4. Find the mean free path from the definition $\lambda = 1/\bar{\Sigma}_t \quad (\text{cm})$

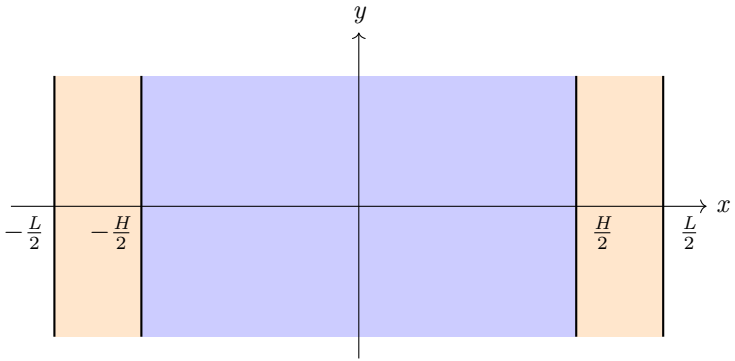
$$\Rightarrow \lambda \simeq 1.48 \text{ cm}$$

Second example: homogeneous slab + reflector

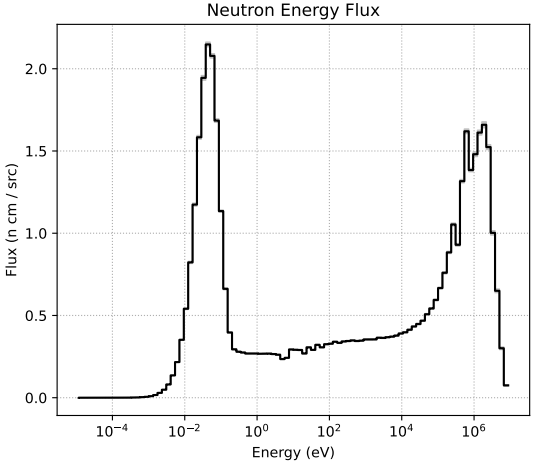
Description

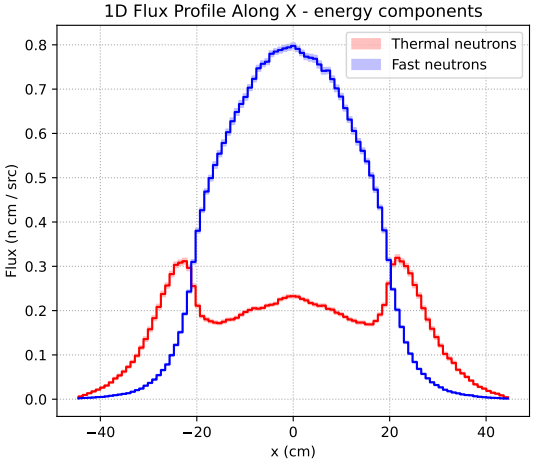
Aim

Model a 1D slab having length L: inner zone of fuel, (lenght H) and outer zone reflector, evaluating its criticality, spatial flux and energy flux.



Results


Neutron Energy Flux
Flux (n cm / src)
Energy (eV)
0.0
0.5
1.0
1.5
2.0
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
10^{6}


1D Flux Profile Along X - energy components
Thermal neutrons
Fast neutrons
Flux (n cm / src)
x (cm)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
−40
−20
0
20
40

Your turn!

Ex3: Homogeneous slab

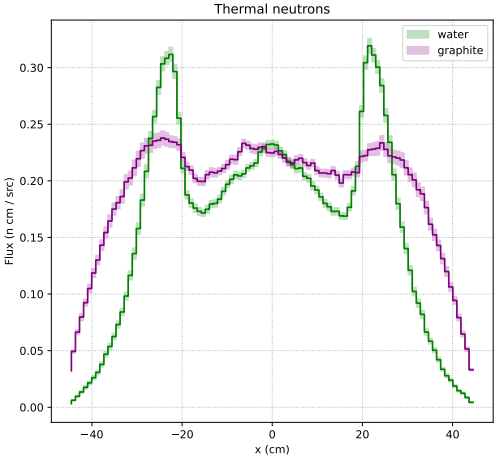
What is the reflector effect? Compare the k-eff and the fluxes in the case of

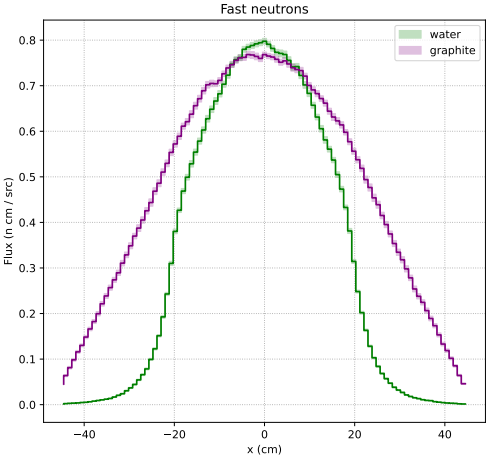
1. Light water reflector
2. Graphite reflector

Hint: save the water results by changing the statepoint name

- Depending on the library, graphite may be added to the openmc materials as:
  - option 1: *reflector.add_nuclide('C12', 1)*
  - option 2: *reflector.add_element('C', 1)*
- Graphite for nuclear reactors has a density of $\simeq 1.7$ g/cm^3

Results


Thermal neutrons
water
graphite
Flux (n cm / src)
x (cm)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
−40
−20
0
20
40

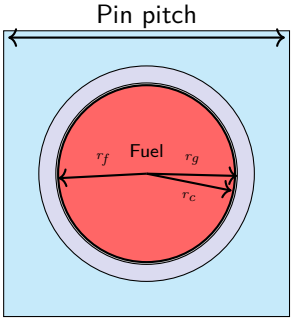

Fast neutrons
water
graphite
Flux (n cm / src)
x (cm)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
−40
−20
0
20
40

Third example: 2D pin

Description

Aim

Model a 2D pin having length with fuel, gap, cladding and water. Evaluate energy flux and spatial flux for both thermal and fast neutrons.



- **Fuel**: fresh UO_2, 4.5% U235 enrichment;
- **Gap**: Helium;
- **Cladding**: Ziconium;
- **Moderator**: H_2O, PWR density ($\simeq 0.75$ g/cm^3)

Thermal scattering problem

- During the thermalization, neutrons lose energy through scattering events: the speed of the neutrons at this stage is way higher that the nucleus thermal motion $\Rightarrow$ free nucleus approximation;
- As the neutrons reach the thermal energies, the fact that the interaction is with **bounded atoms**, become more and more important;
- Namely, atomic binding and lattice vibrations affect:
  - Neutron energy and scattering angle
  - Scattering cross section

  Also, this effect is as large as the atomic mass is low $\Rightarrow$ **very important for H in H_2O, C and Be**

Thermal scattering - correction

Total cross-section of H_2O

$$\sigma_t^{\mathrm{H_2}O}(E) \propto 2\sigma_t^{\mathrm{H}}(E) + \sigma_t^{\mathrm{O}}(E)$$

Total cross-section of H in H_2O

$$\sigma_t^{\mathrm{H}}(E) = \sigma_s(E) + \sigma_\gamma(E)$$

Scattering XS

$$\sigma_s(E) = \iint \frac{d^2\sigma_s}{d\theta\, dE'}\, dE'\, d\theta$$

Double-differential scattering XS

$$\frac{d^2\sigma_s}{d\theta\, dE'} = \frac{\sigma_b}{4\pi k_B T}\sqrt{\frac{E'}{E}}\, S(\alpha,\beta)$$

$$\alpha = \frac{E' + E - 2\sqrt{E'E}\cos\theta}{Ak_BT}$$

Momentum transfer

$$\beta = \frac{E' - E}{k_B T}$$

Energy transfer

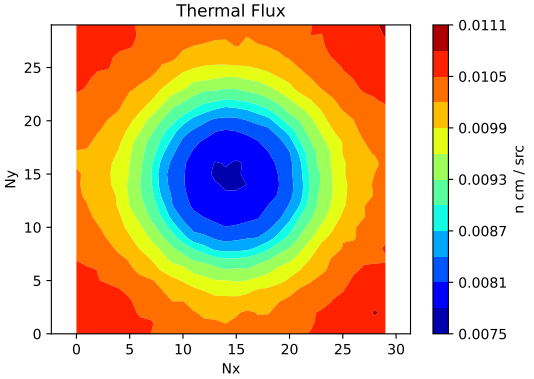
Thermal scattering - in OpenMC

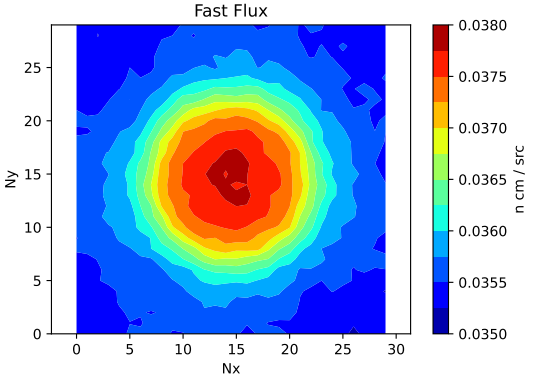
The $S(\alpha, \beta)$ correction may be set for specific materials in OpenMC

```
###############################################################
# ----- MATERIALS

fuel = openmc.Material(name='Homogeneous U-water mixture')
fuel.set_density('g/cm3', 4)
fuel.add_element('U', 1, enrichment=1.0) # 1% enrichment
fuel.add_element('O', 4)
fuel.add_nuclide('H1', 4)

### S(a,b) ###
fuel.add_s_alpha_beta('c_H_in_H2O')

# Create materials object
materials = openmc.Materials([fuel])

materials.export_to_xml()
```

- Full list of nomenclature **here**

Results


Thermal Flux
Ny
Nx
n cm / src
0.0111
0.0105
0.0099
0.0093
0.0087
0.0081
0.0075

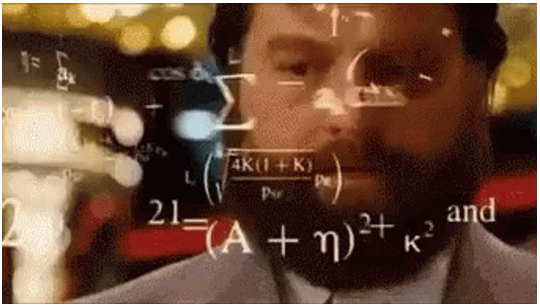

Fast Flux
Ny
Nx
n cm / src
0.0380
0.0375
0.0370
0.0365
0.0360
0.0355
0.0350

Your turn!

Ex4: 2D pin

Estimate the optimal pin pitch

Ideas?



Your turn!

Ex4: 2D pin

Estimate the optimal pin pitch

Ideas?

Balance between the **moderation capability** and the **absorption** from water

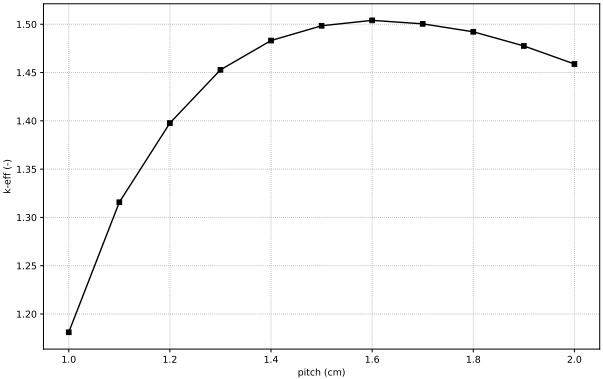
- Simulate the 2D pin with different pitches, saving each time the statepoint file;
- Load the statepoint togheter and look at the dependency

Your turn!

Ex4: 2D pin

Estimate the optimal pin pitch

Ideas?

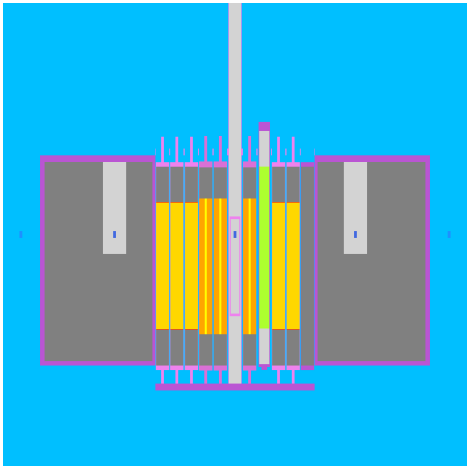


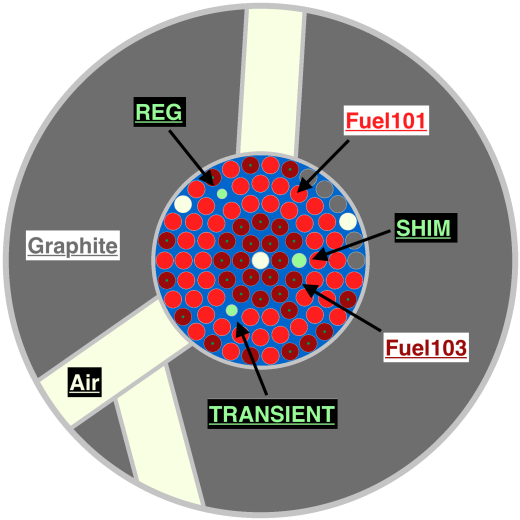
Fourth example: TRIGA

Introduction

- Dedicated course in Polimi ⇒ **Experimental Nuclear Reactor Kinetics**

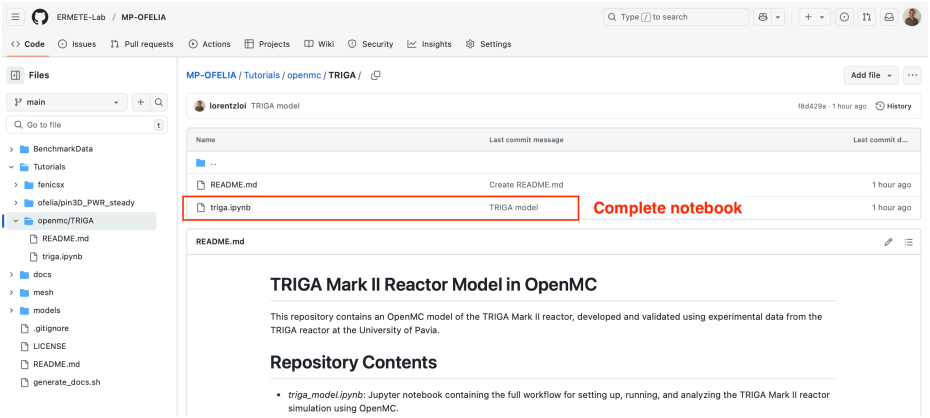
Structural composition





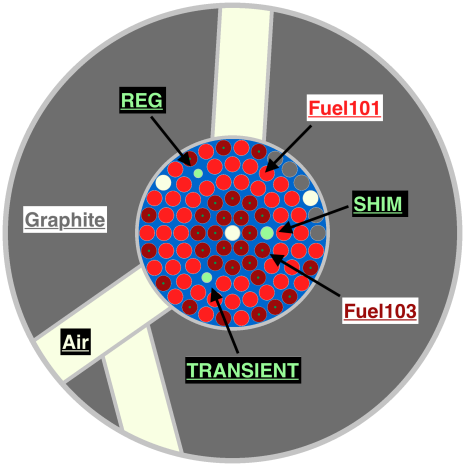
OpenMC model

https://github.com/ERMETE-Lab/MP-OFELIA/tree/main/Tutorials/openmc/TRIGA



Control rod usage

- **REGULATING Rod** – Fine control of reactivity during steady-state operation. Moves frequently to maintain criticality.
- **SHIM Rod** – Coarse adjustment of reactivity, mainly compensates for long-term fuel burnup and xenon poisoning.
- **TRANSIENT Rod** – Enables prompt reactivity insertions or withdrawals for pulse operations; fully withdrawn in steady-state.



Absorption capability

To quantify the characteristic of neutron absorption, each bar has a different value of control rod worth $\Rightarrow$ it measures the change in reactivity due to the full control rod insertion:

$$\text{CR worth}\,(-) = \overbrace{\frac{k_{out}-1}{k_{out}}}^{\rho \text{ @ rod extracted}} - \overbrace{\frac{k_{in}-1}{k_{in}}}^{\rho \text{ @ rod inserted}}$$

$$\text{CR worth}\,(\text{pcm}) = \left(\frac{k_{out}-1}{k_{out}} - \frac{k_{in}-1}{k_{in}}\right) \cdot 10^5$$

$$\text{CR worth}\,(\$) = \left(\frac{k_{out}-1}{k_{out}} - \frac{k_{in}-1}{k_{in}}\right) \cdot \frac{1}{\beta}$$

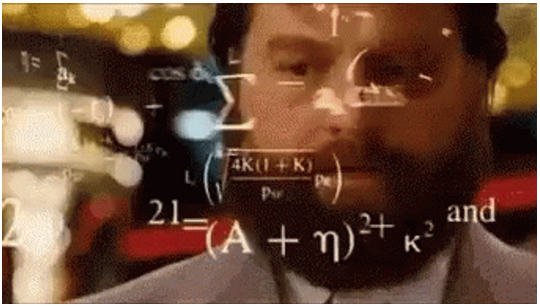
β = fraction of delayed neutrons
$\Rightarrow \beta^{\text{TRIGA}} \simeq 0.00730$

Your turn!

Ex5: CR worth

Estimate the CR worth of the SHIM rod in dollars

Ideas?



Your turn!

Ex5: CR worth

Estimate the CR worth of the SHIM rod in dollars

Ideas?

- CR SHIM $\simeq$ 3.1 \$
- CR REG $\simeq$ 1.3 \$
- CR TRANSIENT $\simeq$ 2.2 \$

Fifth example: Depletion

Background

Boltzmann equation

$$\mathbb{L}\,\varphi = \frac{1}{k}\,\mathbb{F}\,\varphi \quad \text{, where}$$

$$\mathbb{L} \propto \Sigma_t(E) \propto \sigma_t(E) N(\mathbf{r})$$

$$\mathbb{F} \propto \frac{1}{4\pi} \iint \nu(E) \underbrace{\Sigma_f(E,\mathbf{r})}_{\sigma_f(E) N(\mathbf{r})}\, dE d\mathbf{r}$$

Input/Output

$$N \Rightarrow \varphi$$

Background

Boltzmann equation

$$\mathbb{L}\,\varphi = \frac{1}{k}\,\mathbb{F}\,\varphi \quad \text{, where}$$

$$\mathbb{L} \propto \Sigma_t(E) \propto \sigma_t(E) N(\mathbf{r})$$

$$\mathbb{F} \propto \frac{1}{4\pi} \iint \nu(E)\, \underbrace{\Sigma_f(E,\mathbf{r})}_{\sigma_f(E) N(\mathbf{r})}\, dE d\mathbf{r}$$

Bateman equation

$$\frac{dN}{dt} = \mathbf{T} \cdot N \quad \text{, where}$$

$$T_{ii} = -\sum_{r_d} \int \phi(E)\, \sigma_i^{r_d}(E)\; dE \; - \lambda_i$$

$$T_{ij} = \sum_{r_c, j \to i} \int \phi(E)\, \sigma_j^{r_c}(E)\; dE \; + \lambda_j$$

Input/Output

$$N \Rightarrow \varphi$$

Input/Output

$$\phi \Rightarrow N$$

Note: $\phi = \int_\Omega \varphi$

Coupling

In the reality, one should solve the system

$$\begin{cases} \mathbb{L}\,\varphi = \frac{1}{k}\,\mathbb{F}\,\varphi \\ \frac{dN}{dt} = \mathbf{T}\,\cdot\,N \end{cases}$$

Coupling

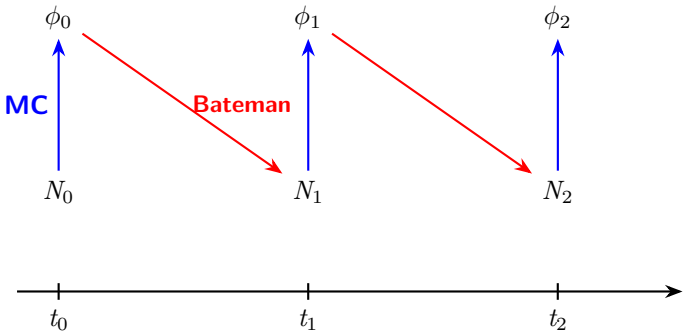
In the reality, one should solve the system

$$\begin{cases} \mathbb{L}\,\varphi = \frac{1}{k}\,\mathbb{F}\,\varphi \\ \frac{dN}{dt} = \mathbf{T} \cdot N \end{cases}$$

$\Rightarrow$ **Burnup algorithm!**

- Procedure that through a time discretization, solves iteratively the Boltzmann equation and the Bateman equation

Simple Burnup Scheme



- This is the example of the most basic scheme (Forward Euler fashion)
- Each Monte Carlo code has its own schemes: keep in mind that this is always the starting point!

How is the Bateman solved?

Boltzmann equation

$$\mathbb{L}\,\varphi = \frac{1}{k}\,\mathbb{F}\,\varphi \quad \text{, where}$$

$$\mathbb{L} \propto \Sigma_t(E) \propto \sigma_t(E) N(\mathbf{r})$$

$$\mathbb{F} \propto \frac{1}{4\pi} \iint \nu(E) \underbrace{\Sigma_f(E,\mathbf{r})}_{\sigma_f(E) N(\mathbf{r})} \, dE d\mathbf{r}$$

Input/Output

$$N \Rightarrow \varphi$$

Bateman equation

$$\frac{dN}{dt} = \mathbf{T} \cdot N$$

$$N(t) = N(0)\, e^{\mathbf{T}\, t}$$

- This solution is exact $\Rightarrow$ no statistical uncertainty associated
- This hold under the hypothesis that the flux is **constant within each step**

OpenMC data

- In order to run the burnup, OpenMC need *decay constants* and *fission yields* to solve the Bateman equation
- That data fall under the name of **depletion chain** and could be found at https://openmc.org/depletion-chains/
- Remember to 1) download and 2) store the chain in the folder where you are running the depletion

Code (I)

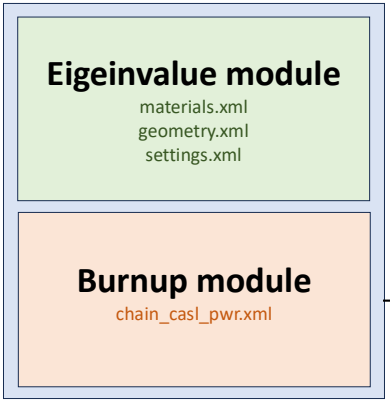
- The user has to **choose** which materials are going to be depleted and which not.
- OpenMC will burn all the materials with a *volume assigned*

```
############################################################
# ---- MATERIALS - burnup

fuel = openmc.Material(name='UO2')
fuel.set_density('g/cm3', 10.2)
fuel.add_element('U', 1, enrichment=3.0) # 1% enrichment
fuel.add_element('O', 2)
fuel.volume = 3.14 * 0.39**2 # if 2D, set the cross sectional area (cm2)
```

- **Volume** and **Power** are the input required by OpenMC for normalize the reaction rates

Code (II)

- Complete code structure

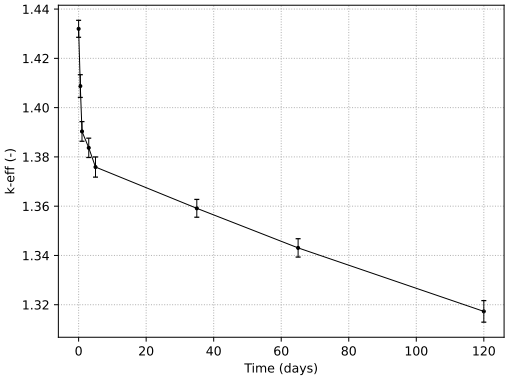



```
## BURNUP MODULE ##
import openmc.deplete

chain = openmc.deplete.Chain.from_xml("./chain_casl_pwr.
    xml")

model = openmc.Model(geometry=geometry, settings=settings)
operator = openmc.deplete.CoupledOperator(model, "./
    chain_casl_pwr.xml")

P = 170 # (W/cm) for non-3D systems

time_step = [0.5, 30, 30, 60] # example: simulate 4 months

integrator = openmc.deplete.PredictorIntegrator(operator,
    time_step, P, timestep_units='d')

integrator.integrate()
```

- All the data are stored in the **depletion_results.h5** file

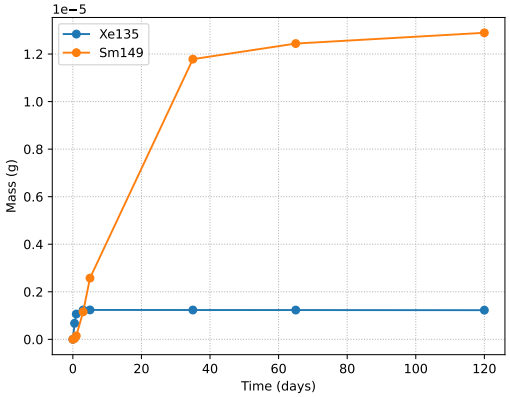
Multiplication factor in time


```
# Depletion post processing
results = openmc.deplete.Results("./
    depletion_results.h5")
time, k = results.get_keff()
time_days = time / (60 * 60 * 24) # time vector
    in days

# plot keff
plt.errorbar(time_days, k[:,0], k[:,1], color =
    "black", linestyle = '-', capsize = 2,
    linewidth = 0.5)
plt.scatter(time_days, k[:,0], s = 5, color = "
    black")
plt.xlabel("Time (days)")
plt.ylabel("k-eff (-)")
plt.grid(linestyle = ':')
```



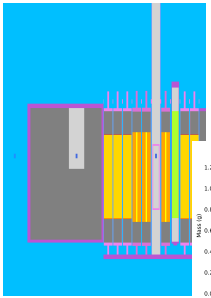
Nuclides in time

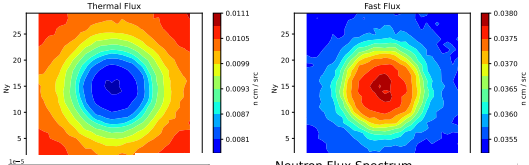

```
# read concentrations in time
time, Xe135 = results.get_mass("1", "Xe135") # (
    g)
time, Sm149 = results.get_mass("1", "Sm149") # (
    g)

# plot nuclides in time
plt.plot(time_days, Xe135, '-o', label = "Xe135"
    )
plt.plot(time_days, Sm149, '-o', label = "Sm149"
    )
plt.xlabel("Time (days)")
plt.ylabel("Mass (g)")
plt.legend()
plt.grid(linestyle = ':')
```

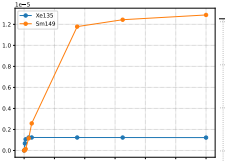


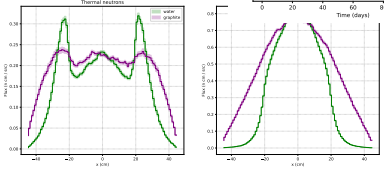
Conclusions

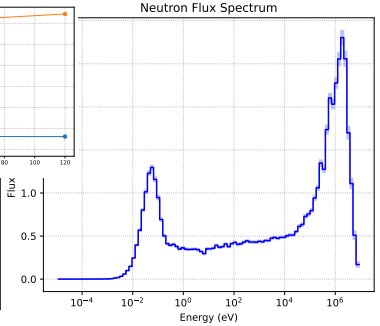
Recap











Full tutorial in OpenMC for reactor modelling

1. Bare homogeneous slab
2. Reflected homogeneous slab
3. 2D pincell
4. TRIGA research reactor
5. Depletion

For doubts/comments:
lorenzo.loi@polimi.it