Monte Carlo Methods and Simulations in Nuclear Technology

Home Assignment 04

Computation of the relative change in the distance a fission neutron travels to its first collision when the material's atomic concentration is increased by 0.01% (compared to the concentration selected in home assignment 03).

BY: FAISAL AHMED MOSHIUR
DATE:15/12/2022

Problem:

This assignment builds on your solution of home assignment 03. Your task is to compute, by the Monte Carlo method, the relative change in the distance the fission neutron flies to its first collision when the atomic concentration of the material is increased by 0.01% (compared to the concentration that you selected in home assignment 03).

- First, try to compute this result by running two independent simple sampling simulations (at the two different atomic concentrations). Repeat the experiment and evaluate the standard deviation of the relative change of the distance the fission neutron flies to its first collision.

- Second, compute the result by the correlated sampling method. Evaluate the standard deviation of the result and compare it to the result you got in the first part of this assignment.

Nuclear Fission:

Each uranium-235 (U-235) atom has 92 protons and 143 neutrons, for a total of 235. The particle arrangement within uranium-235 is somewhat unstable, and the nucleus can dissolve if energized by an outside source. When a U-235 nucleus absorbs an added neutron, it swiftly splits into two halves. This is known as fission. When a U-235 nucleus divides, two or three neutrons are released. As a result, the chance of starting a chain reaction exists.

The task at hand is to calculate the mean distance that the fission neutrons fly till their first collision. To carry out the said task, I took into consideration a particular system mentioned before in the problem i.e., an infinite system composed of a single fissile nuclide at a reasonable mass density.

Transition Kernel:

We know that transition kernel, T, is the probability density function of the distance, s, traversed by a neutron to next collision. To perform a simple sampling simulation to evaluate the mean distance that fission neutrons fly to the first collision, I made use of the cumulative distribution function derived from transformation of the given probability density function

$$T = \sum_t (\vec{r}, E) e^{-\Sigma_t s} \quad ----- [1]$$

where, $\sum_t$ is the total macroscopic cross-section of a given nuclide. In addition to that, we assumed that the total macroscopic cross section is constant along the flight in between the two collisions.

Knowledge concerning the physics of the fission process is summarized in various nuclear data libraries such as JEFF, ENDF, or ENSDF. JEFF and ENDF list the cross sections of a wide range of nuclear reactions as well as the fission yields. ENSDF, on the other hand, is primarily concerned with nuclear structural data. We can find the distance, s, to first collision of neutrons using the microscopic cross-section values collected from ENDF B-VIII.0 including the incident neutron energies via the JANIS database. These data can be used to simulate the values of distances, s, to first collision of neutrons by the help of cumulative distribution function and inverse transform of it.

We know that

$$\sum_t = N\sigma_t ----[2]$$

where, N is the atomic number density and σ_t is the microscopic cross-section.

Therefore, the cumulative distribution function of the transition kernel, T,

$$F_s = \int_0^s \sum_t (\vec{r}, E) e^{-\Sigma_t \acute{s}}\, d\acute{s} = 1 - e^{-\Sigma_t s} ----- [3]$$

(Homogeneous material)

Let the generated random number be, $\mathcal{U}$, which lives in the interval $(0, 1)$.

From [3], we get

$$\mathcal{U} = 1 - e^{-\Sigma_t s} ----[4]$$

Therefore, we can make s the subject of the equation shown above to obtain

$$s = -\frac{1}{\Sigma_t}\ln(1-\mathcal{U}). - - - - -[5]$$

Since the term $(1-\mathcal{U})$ is also a random number from interval (0, 1). We can rewrite [5] as

$$s = -\frac{1}{\Sigma_t}\ln(\mathcal{U}). - - - - -[6]$$

By generating random values of $\mathcal{U}$, I got different values of s using cross section values. The cross-section values are found for the nearest incident energy of the neutron by using simple mathematical tool of linear interpolation. I have used the built-in interpolation module of Python "NumPy" library to find the cross-section values for nearest incident energy values of neutrons.

Graph:

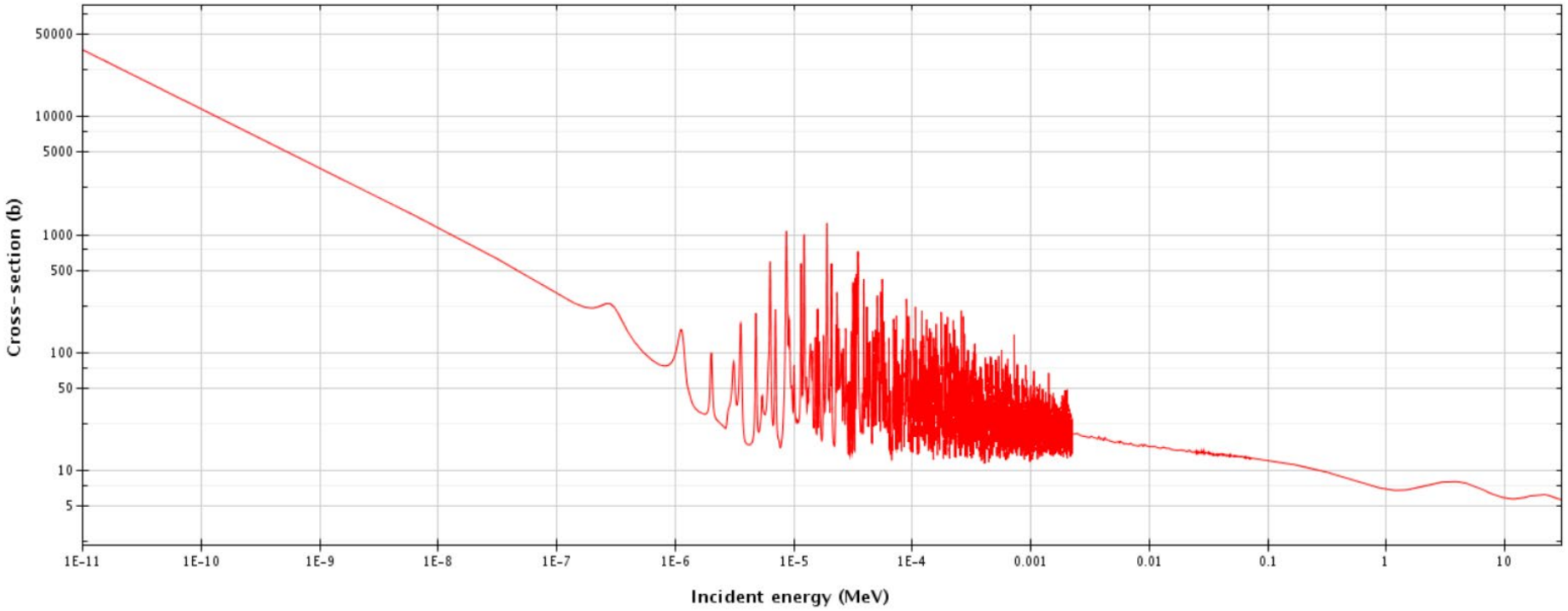

Fig. 1: NEA / Incident neutron data / ENDF/B-VIII.0 / Cross sections / U235 / MT=1: (n,total) / Cross section

Mean and standard deviation for sampling using random variables:

After sampling n values of unknown random variable Y, the expectation value of Y can be estimated by the mean value of those generated sampling n values.

$$m_Y = \frac{1}{n}\sum_{i=1}^{n} y_i ----[7]$$

According to the central limit theorem,

$$E[m_Y] = E[Y]. ----[8]$$

In addition to that, variance of mean values of generated samples of the unknown random variable Y, ${\sigma^2}_{m_Y}$:

$${\sigma^2}_{m_Y} = \frac{\sum E[{\xi_i}^2]}{n^2} = \frac{{\sigma^2}_Y}{n} ----[9]$$

where

$$\xi_i \equiv y_i - E[Y]. ----[10]$$

However, the value of ${\sigma^2}_Y$ is difficult to obtain but it can be estimated if a considerably substantial number of samples are taken e.g., $n > 10000$.

Therefore, variance

$$ {\sigma^2}_Y = \frac{1}{n}\sum_{i=1}^{n}(y_i - m_Y)^2 = \frac{1}{n}\sum_{i=1}^{n} {y_i}^2 - {m_Y}^2 \,.----[11] $$

Hence, we just need to update the values of $\sum {y_i}^2$ and $\sum y_i$ to estimate the $E[Y]$ and ${\sigma^2}_{m_Y}$ after collecting a new sample of Y.

And the standard deviations σ are found by,

$$ \sigma_{m_Y} = \frac{\sigma_Y}{\sqrt{n}}.----[12] $$

Correlated Sampling:

When calculating the difference between two extremely similar models, g_1 and g_2, correlated sampling might be employed.

$$Y_1 = g_1(X) - - - -[13]$$

$$Y_2 = g_2(X) - - - -[14]$$

Typically, g_2 is a significantly modified form of g_1. Since this difference between the expected values could be exceedingly small, the sampling would need to calculate $E[Y_1]$ and $E[Y_2]$ to a high degree of accuracy, which would take a significant amount of computational time.

Nevertheless, by examining the random variable, correlated sampling can easily solve this problem.

Let,

$$Z = Y_1 - Y_2 = (g_1 - g_2)(X) - - - -[15]$$

Variance of Z is

$$Var[Z] = Var[Y_1] - Var[Y_2] - 2Cov[Y_1, Y_2]. - - - - [16]$$

Where covariance of Y_1 and Y_2,

$$Cov[Y_1, Y_2] = E[(Y_1 - E[Y_1])(Y_1 - E[Y_1])] = E[Y_1 Y_2] - E[Y_1]E[Y_2], - - - - [17]$$

is the value showing how Y_1 and Y_2 are related to each other.

Sampling Z is more efficient than sampling Y_1 and Y_2 separately using independent input random variables X_1 and X_2 when Y_1 and Y_2 are significantly positively correlated to each other. That implies that g_1 and g_2 are remarkably similar.

Results:

Sampling of ten million random values of relative distance traversed by neutrons till first collision for U-235 nuclide of atomic number density, $N = 7.98 \times 10^{21}\ atoms/cm^3$ [2] and when N is increased by 0.01%:

For simple sampling:
Covariance: -0.238
Mean relative distance: 3681.799
Standard deviation of relative distance: 1944157.695
Simple sampling time (in seconds): 446.469

For correlated sampling:
Covariance of relative distance: 278.719
Mean relative distance: 9.999×10^{-5}
Standard deviation of relative distance: 1.154×10^{-16}
Correlated sampling time (in seconds): 264.250

Discussions:

The dataset obtained from slightly changing the value of atomic density, N, gives random values of mean distances a shift to the values. Results of mean and standard deviation of the relative distances show that simple sampling the distance individually will bring more uncertainty in the value. The standard deviation is increased by the fact that variance increased due to the covariance term being negative for simple sampling [16]. However, correlated sampling technique reduced the variance significantly as it had greater value of covariance in comparison to what we obtained from simple sampling of two variables [16]. Hence, the time to perform the simulation using correlated sampling is $\sim$1.7 times faster than what it took from running two simple sampling simulations (10^7 samples were taken for each technique). This is because, we are only using the time to sample random number once to plug it into the two systems to get a value.

References:

- J. R. Lamarsh, A. J. Baratta, Introduction to Nuclear Engineering, 3d ed., Prentice-Hall, 2001, ISBN: 0-201-82498-1.
- Lessons on Monte Carlo methods and simulations in nuclear technology.
- JANIS database for cross-section values with corresponding incident neutron energies (ENDF B-VIII.0).

Appendix:

```
import numpy as np
import random
import scipy.interpolate as interpolate
import pandas as pd
from time import process_time

# Random number with defined seed of 987654321
np.random.seed(987654321)

# The pdf for energy distribution
def pdf(x):
    a = 0.5535
    b = 1.0347
    c = 1.6214

    return a * np.exp(-x / b) * np.sinh(np.sqrt(c * x))

# The line pdf for energy distribution
def line_pdf(x, x1, y1, x2, y2):
    m = (y1 - y2) / (x1 - x2)
    c = y1 - x1 * (y1 - y2) / (x1 - x2)
    h_x =  m * x + c
    return h_x

# The line cdf for energy distribution
def line_cdf(x, x1, y1, x2, y2):
    m = (y1 - y2) / (x1 - x2)
    c = y1 - x1 * (y1 - y2) / (x1 - x2)
    F_x =  m * x**2 / 2 + c * x
    return F_x

# Inverse of the line cdf for energy distribution
def inv_line_cdf(x, x1, y1, x2, y2):
    m = (y1 - y2) / (x1 - x2)
    c = y1 - x1 * (y1 - y2) / (x1 - x2)
    F_inv_x = - c / m + (np.sqrt(c**2 + 2 * m * x))/m
    return F_inv_x

# Acceptance rejection method using triangle approach
def triangle_approach(n=1):

    uniform_rn = np.random.uniform(0, 1, 10 * n)

    prob_scaled_rn_1 = inv_line_cdf(uniform_rn, 0, 0.1, 20, 0)

    prob_scaled_rn_2 = np.zeros(n)
    count = 0
    for i in range(0, len(prob_scaled_rn_1)):
        c = 4
        h = line_pdf(prob_scaled_rn_1[i], 0, 0.1, 20, 0)
        u = np.random.rand()
        f = pdf(prob_scaled_rn_1[i])

        if u * c * h <= f:
            prob_scaled_rn_2[count] = prob_scaled_rn_1[i]
            count += 1

        if count >= n:
            break

    E = prob_scaled_rn_2[0]

    return E

sigma_t_vs_E =
pd.read_csv('C://Users//faisa//OneDrive//Documents//RLT//Git_clone_repo//Study_mater
ials-1//MC_Methods//HA//HA04//ENDF8_NT_InEnvsCRsec.csv')       # Reading the csv
file from Janis
sigma_t_vs_E = sigma_t_vs_E.to_numpy()      # Transforming the pandas dataframe into
a numpy array
sigma_t_vs_E[:, 0] = sigma_t_vs_E[:, 0] * 1e-6      # Turning E in eV from Janis
into E in MeV for our use case
sigma_t_vs_E[:, 1] = sigma_t_vs_E[:, 1] * 1e-24     # Turning sigma in barns from
Janis into sigma in cm^2 for our use case

def calculate_s(N, E, u, sigma_t_v_E):
    sigma_intp = np.interp(E, sigma_t_v_E[:, 0], sigma_t_v_E[:, 1])       #
Interpolating the sigma values for our E values
```

```
    SIGMA_intp = sigma_intp * N     # SIGMA = N * sigma
    s = (- 1 / SIGMA_intp) * np.log(u)     # Approximating distance to first collision

    return s

# Simple sampling for values of s for N and 1.0001N, and the relative change of values of s due to it.
def run_2_SSS(sigma_t_v_E, n):
    N1 = 7.98e21
    N2 = N1 * 1.0001
    s = np.zeros((3, n))
    rel_delta_s = np.zeros(n)

    for i in range(0, n):
        E1 = triangle_approach()
        u1 = np.random.rand()
        s1 = calculate_s(N1, E1, u1, sigma_t_v_E)
        s[0, i] = s1

        E2 = triangle_approach()
        u2 = np.random.rand()
        s2 = calculate_s(N2, E2, u2, sigma_t_v_E)
        s[1, i] = s2

        s[2, i] = s1 * s2

        rel_delta_s[i] = abs((s1 - s2) / s1)

    print("For simple sampling:")
    print(f"Covariance:{np.mean(s[2, :]) - np.mean(s[0, :]) * np.mean(s[1, :])}")
    print(f"Mean relative distance:{np.mean(rel_delta_s)}\nStandard deviation of relative distance:{np.std(rel_delta_s)}")
#   print(np.cov(s[0, :], s[1, :]))

# Correlated sampling of values for values of s for N and 1.0001N, and the relative change of values of s due to it.
def correlated_ss(sigma_t_v_E, n):
    N1 = 7.98e21
    N2 = N1 * 1.0001
    s = np.zeros((3, n))
    rel_delta_s = np.zeros(n)

    for i in range(0, n):
        E = triangle_approach()
        u = np.random.rand()

        s1 = calculate_s(N1, E, u, sigma_t_v_E)
        s[0, i] = s1

        s2 = calculate_s(N2, E, u, sigma_t_v_E)
        s[1, i] = s2

        s[2, i] = s1 * s2

        rel_delta_s[i] = abs((s1 - s2) / s1)

    print("For correlated sampling:")
    print(f"Covariance:{np.mean(s[2, :]) - np.mean(s[0, :]) * np.mean(s[1, :])}")
    print(f"Mean relative distance:{np.mean(rel_delta_s)}\nStandard deviation of relative distance:{ np.std(rel_delta_s)}")
#    print(np.cov(s[0, :], s[1, :]))


# Sampling the values 10000000 times
start_sss = process_time()
run_2_SSS(sigma_t_vs_E, int(1e7))
end_sss = process_time()
print(f"Simple sampling time:{end_sss - start_sss}")

start_css = process_time()
correlated_ss(sigma_t_vs_E, int(1e7))
end_css = process_time()
print(f"Correlated sampling time:{end_css - start_css}")

```