Monte Carlo Methods and Simulations in Nuclear Technology

Home Assignment 03

Calculation of the mean distance that the fission neutrons fly till their first collision.

BY: FAISAL AHMED MOSHIUR
DATE:08/12/2022

Problem:

By Monte-Carlo method, calculate the mean distance that the fission neutrons fly till their first collision. Use an infinite system composed of a single fissile nuclide at a reasonable mass density.

Hints:

- Find the formula that gives the distance to the first collision (explained in Topic 4.3: Transition Kernel).

- You will need to locate the total microscopic cross section for specific neutron energy. For this, you need to download a table of cross sections for various neutron energies and program a script that locates the cross-section value for the nearest energy. You can download the cross-section tables e.g., from the JANIS database.

- In a loop, sample the energy of a fission neutron (use your code from Home Assignment 02 to do that), and for each sample locate the corresponding cross section and evaluate the distance to the nearest collision.

- Perform a simple sampling simulation to evaluate the mean distance that fission neutrons fly to the first collision. Collect at least several thousand samples and compute the mean distance and the standard deviation of the mean distance.

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 additional 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 accomplish 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 resides 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 located 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:

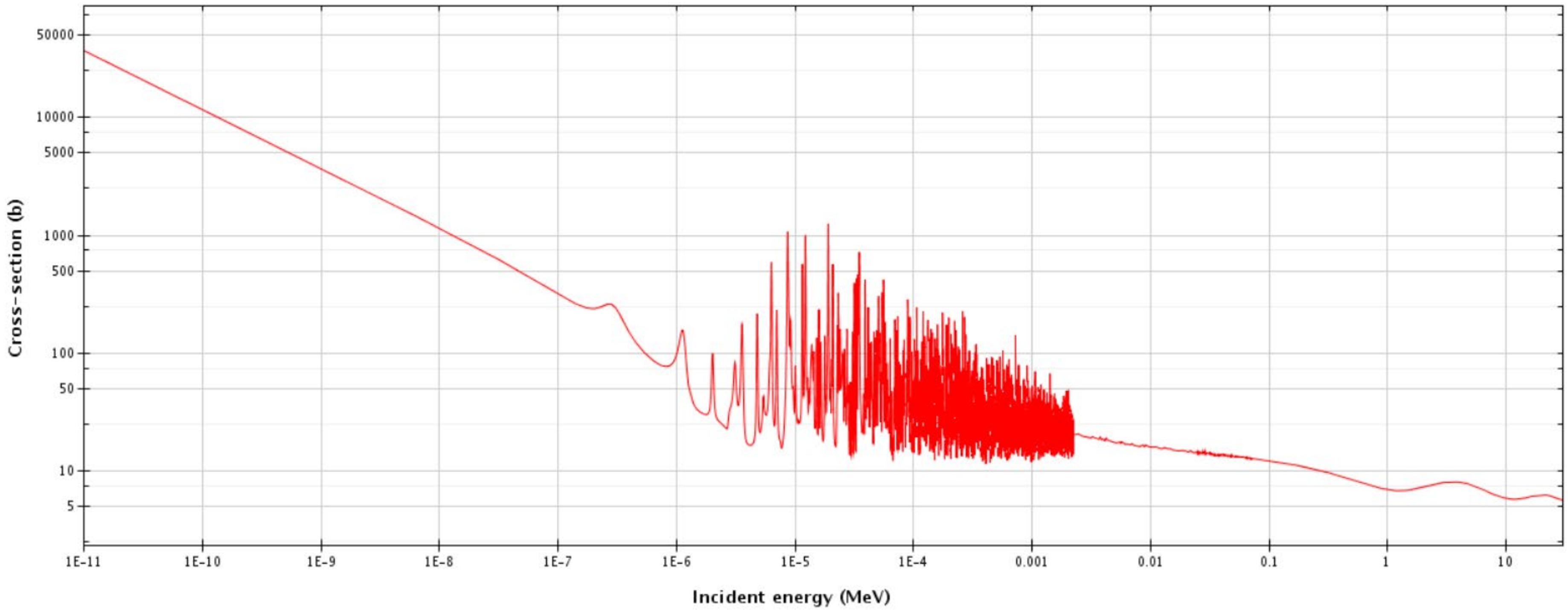

Fig. 1: NEA / Incident neutron data / ENDF/B-VIII.0 / Cross sections / U235 / MT=1: (n,total) / Cross section

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 large 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,

$$\sigma\,{}_{m_Y} = \frac{\sigma\,{}_Y}{\sqrt{n}} . - - - - \, [12]$$

Results:

For U-235 nuclide of atomic number density, $N = 7.98 \times 10^{21}$ atoms/cm^3 [2]:

Mean distance traversed by neutrons till first collision, $\bar{s}$ (in cm): 16.4919

Standard deviation of mean distance traversed by neutrons till first collision, σ_s (in cm): 0.0053

Discussions:

When collecting more random samples y_i, the estimated ${\sigma^2}_{m_Y}$ will usually decrease; however, the real error in m_Y is never known and it may even increase when more samples are collected. Hence, error might be present in the value of mean distance traversed by neutrons till first collision. That is why I have obtained a standard deviation of these values and observed a spread of 0.0053cm among the values.

However, if we have an in homogeneous medium, we can also use the above transition kernel and its corresponding cumulative distribution functions. We can make use of the phenomenon that neutrons have no history and use the cross-sections of the medium it is in when undergoing collision.

The computations used the values of (microscopic)cross-sections and their corresponding incident energies for neutrons are about <41000 in the database within the range of 10μeV to 20MeV. Nevertheless, it is not given that increasing the number of the values used will give better results in terms of lower of deviation. It can also increase the computational cost for larger sets of values due to searching of the values and interpolating them for nearest incident energies.

The interpolation used to obtain nearest incident energy of neutron for a particular microscopic cross-section in the regions of many sharp peaks might have contributed to some deviation in the results. [Fig. 1]

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: Topics- 2.4, 2.5 and 4.3.
- JANIS database for cross-section values with corresponding incident neutron energies (ENDF B-VIII.0).

Appendix:

```python
import numpy as np
import random
import scipy.interpolate as interpolate
import pandas as pd
import matplotlib.pyplot as plt
from pynverse import inversefunc

def display_distribution_of_RN(x, n=10):
    intervals = np.linspace(np.amin(x), np.amax(x), n + 1)
    points_in_intervals = np.zeros(n)

    points_in_intervals, intervals = np.histogram(x, bins=intervals)

    plt.plot(intervals[0:(n - 1)], points_in_intervals[0:(n - 1)], 'r')
    plt.xlabel('intervals')
    plt.ylabel('number of values in interval')
    plt.ylim(bottom=np.amin(points_in_intervals) - 0.2 * np.amin(points_in_intervals))
    plt.ylim(top=np.amax(points_in_intervals) + 0.2 * np.amax(points_in_intervals))
    plt.show()

# def inverse_transform_sampling(cdf, lower, upper, uni_rn):
#     inv_cdf = inversefunc(cdf, domain=[lower, upper])
#     return inv_cdf(uni_rn)

# the pdf
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
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
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
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, rng_seed=987654328):
    # np.random.seed(rng_seed)

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

    # aur.display_distribution_of_RN(uniform_rn, 10)

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

    # print(np.amax(prob_scaled_rn_1))

    # display_distribution_of_RN(prob_scaled_rn_1, 10)

    prob_scaled_rn_2 = np.zeros(n)
    count = 0
    # prob_scaled_rn_2_list = []
    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
            # prob_scaled_rn_2_list.append(prob_scaled_rn_1[i])

        if count >= n:
            break

    # prob_scaled_rn_2 = np.array(prob_scaled_rn_2_list)

    mean_rn = np.average(prob_scaled_rn_2)
    var_rn = np.var(prob_scaled_rn_2)
    sd_rn = np.std(prob_scaled_rn_2)

    return prob_scaled_rn_2, mean_rn, var_rn, sd_rn


# PART 1
def run(n):
    E_MeV, mean_E, var_E, sd_E = triangle_approach(n)

    sigma_t_vs_E = pd.read_csv('C://Users//faisa//OneDrive//Documents//RLT//Study_Materialz//MC_Methods//HA//HA03//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

    sigma_intp = np.interp(E_MeV, sigma_t_vs_E[:, 0], sigma_t_vs_E[:, 1])       # interpolating the sigma values for our E values

    SIGMA_intp = sigma_intp * 7.98e21 # SIGMA = N * sigma.

    s = np.zeros(len(SIGMA_intp))

    for i in range(0, len(SIGMA_intp)):
        s[i] = (- 1 / SIGMA_intp[i]) * np.log(np.random.rand())     # approximating distance to first collision

    mean_s = np.average(s)
    var_s = np.var(s)

    var_mean_s = var_s / len(s)
    sd_mean_s = np.sqrt(var_mean_s)

    print(f"Mean distance traversed by neutrons till first collision(in cm): {mean_s}\nStandard deviation of mean distance traversed by neutrons till first collision(in cm): {sd_mean_s}")


run(int(1e7))
```