Monte Carlo Methods and Simulations in Nuclear Technology

Home Assignment 02

BY: FAISAL AHMED MOSHIUR
DATE:24/11/2022

Problem:

1) Having the probability density function that describes the energy distribution of fission neutrons coming from a specific fissile nuclide (the first assignment), generate at least ten thousand samples randomly from this distribution by the acceptance-rejection method, and use these samples to estimate:

- the mean value of the fission neutron energy,
- the variance and the standard deviation of the energy of the fission neutrons,
- confidence intervals for the estimated mean value,
- the variance and the standard deviation of the mean value.

2) Compare the results with those obtained deterministically in the previous assignment.

3) Repeat the Monte Carlo simulation with different RNG seeds. How often does the accurate expectation value (computed in the previous assignment) fall into the computed confidence intervals?

You can use the RNG provided by the programming language you use, or you can implement the RNG yourself if you wish.

Acceptance-rejection method

This is a technique which generates samples from any probability density function, $f_X(x)$, using another probability density function, $h(x)$, for that holds that $f_X(x) \leq h(x)c$ where, $c = sup_x[\frac{f_X(x)}{h(x)}]$ and $c \geq 1$.

Generation of samples randomly from given distribution by the acceptance-rejection method entails following procedure:

- Generate two random numbers, e.g., x from probability density function, $h(x)$, and u from probability density function, $\mathcal{u}(0,1)$, such that $x = F_X^{-1}(u)$ with u being randomly sampled from, $\mathcal{u}(0,1)$.
- Accept x if $u \,.\, c \,.\, h(x) < f_X(x)$.

The proportion of proposed samples which are accepted is:

$$\frac{\int_{-\infty}^{\infty} f_X(x)dx}{\int_{-\infty}^{\infty} c \,.\, h(x)} = \frac{1}{c}$$

For good efficiency, c should be close to unity without compromising on the fact that the inverse transform method can be deployed to easily generate samples from $h(x)$.

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:

$$m_Y = \frac{1}{n}\sum_{i=1}^{n} y_i$$

According to the central limit theorem,

$$E[m_Y] = E[Y]$$

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

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

where, $\xi_i \equiv y_i - E[Y]$

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

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, $\sigma_{m_Y} = \frac{\sigma_Y}{\sqrt{n}}$

Confidence Interval

The confidence interval is the range of values within which we expect our estimate to fall a certain percentage of the time if we repeat our procedures or re-sample the population in the same way.

Therefore, explaining in terms of the sampling case mentioned above the probability that $E[Y]$ is inside the interval $[m_Y - \delta, m_Y + \delta]$ equals the probability that m_Y is inside the interval $[E[Y] - \delta, E[Y] + \delta]$.

Moreover, it should also be mentioned that the values of δ i.e., the intervals are governed by significance of the σ associated with it, which in terms dictate the probability of such said intervals.

In the case of the energy distribution of fission neutrons coming from a U-235 fissile nuclide, let the probability density function, $\chi(\bar{E})$, describing it be:

$$\chi(\bar{E}) = ae^{-\frac{\bar{E}}{b}}\sinh(\sqrt{c\bar{E}})$$

where,

$$a = 0.5535\,,\ b = 1.0347\ MeV, and\ c = 1.6214\ MeV^{-1}$$

The exact value of $E[\bar{E}] = \int_0^\infty \bar{E}\chi(\bar{E})dx = \int_0^\infty a\bar{E}e^{-\frac{E}{b}}\sinh(\sqrt{c\bar{E}})dx = 2.0\ MeV$

I have used the triangle approach to find the values of random samples with procedures explained in the above acceptance-rejection method.

Results

Time taken to generate 10000 means = 621.375 seconds

Mean of means = 1.9885685758346887

Variance of means = 0.00024325403448789435

Standard deviation of means = 0.015596603299689787

Ratio of random numbers within 1 SD = 0.6839

Confidence intervals obtained from different seeds to generate different random numbers:

Intervals with levels of σ_{m_Y} signf.	Chance to fall in given interval with 100 diff. seeds	Chance to fall in given interval with 10000 diff. seeds
1σ	0.23	0.2521
2σ	0.63	0.6244
3σ	0.91	0.904

Remarks

When collecting more 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.

Taking more and more samples of mean could enhance the values falling into the intervals as well as investigating the triangle function to fit into the probability density function could also result into much better values of chances of values falling into the confidence intervals.

```python
import numpy as np
import random
import time

# 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)
    Finv_x = - c / m + (np.sqrt(c**2 + 2 * m * x))/(m)
    return Finv_x

# Acceptance rejection method using triangle approach
def triangle_approach():
    uniform_rn = np.random.uniform(0, 1, 100000)

    prob_scaled_rn_1 = inv_line_cdf(uniform_rn, 0, 0.2, 10, 0)

    prob_scaled_rn_2_list = []

    for i in range(0, len(prob_scaled_rn_1)):
        c = 2
        h = line_pdf(prob_scaled_rn_1[i], 0, 0.2, 10, 0)
        u = np.random.rand()
        f = pdf(prob_scaled_rn_1[i])

        if u * c * h <= f:
            prob_scaled_rn_2_list.append(prob_scaled_rn_1[i])

        if len(prob_scaled_rn_2_list) >= 10000:
            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(seed):
    np.random.seed(seed)

    rns, mean_rns, var_rns, sd_rns = triangle_approach()
    '''
    print(f'mean of random numbers = {mean_rns}')
    print(f'variance of random numbers = {var_rns}')
    print(f'SD of random numbers = {sd_rns}')
    '''
    var_mean = var_rns / len(rns)
    sd_mean = np.sqrt(var_mean)
    '''
    print(f'\nvariance of mean = {var_mean}')
    print(f'SD of mean = {sd_mean}')
    '''
    interval_1_left = mean_rns - sd_mean
    interval_1_right = mean_rns + sd_mean
    interval_2_left = mean_rns - 2 * sd_mean
    interval_2_right = mean_rns + 2 * sd_mean
    interval_3_left = mean_rns - 3 * sd_mean
    interval_3_right = mean_rns + 3 * sd_mean
    '''
    print(f'\nConfidence interval 1 = ({interval_1_left}, {interval_1_right})')
    print(f'Confidence interval 2 = ({interval_2_left}, {interval_2_right})')
    print(f'Confidence interval 3 = ({interval_3_left}, {interval_3_right})')
    '''
    return interval_1_left, interval_1_right, interval_2_left, interval_2_right, interval_3_left, interval_3_right


# PART 2
def mean_var_sd_for_means_1(seed, m): # m = number of means we want
    np.random.seed(seed)

    means = np.zeros(m)

    start = time.process_time()

    for i in range(0, m):
        means[i] = triangle_approach()[1]

    end = time.process_time()
    print(f'time taken to generate {m} means = {end - start} seconds')

    mean_means = np.average(means)
    var_means = np.var(means)
    sd_means = np.std(means)

    print(f'mean of means = {mean_means}')
    print(f'variance of means = {var_means}')
    print(f'SD of means = {sd_means}')

    dev_means = abs(means - mean_means)
    acc = 0
    rej = 0
    for k in dev_means:
        if abs(k) <= sd_means:
```

```
            acc += 1
        else:
            rej += 1

    print(f'Ratio of random numbers within 1 SD = {acc / (acc + rej)}')


def check(actual, n):
    one = 0
    two = 0
    three = 0

    for i in range(987654321, 987654321 + n):
        interval_1_left, interval_1_right, interval_2_left, interval_2_right, interval_3_left, interval_3_right = run(i)

        if actual >= interval_1_left and actual <= interval_1_right:
            one += 1

        if actual >= interval_2_left and actual <= interval_2_right:
            two += 1

        if actual >= interval_3_left and actual <= interval_3_right:
            three += 1

        #print(i - 987654320)

    print(f'Chance to fall in interval 1 = {one / n}')
    print(f'Chance to fall in interval 2 = {two / n}')
    print(f'Chance to fall in interval 3 = {three / n}')


#run(987654321)
check(2, 10000)
#mean_var_sd_for_means_1(987654321, 10000)
triangle_approach()
```