# 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_materials-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
```

```python
    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}")
```