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:
```

```python
            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))
```