Shumet Getahun ***Roll no. 22203262***

Programming Assignment 1: Monte Carlo Simulation

CE605A Probability and Statistics for Civil Engineers 2023-24 I

Instructor: Balaji Devaraju [dbalaji] TA: Ibaad Anwar [ibaadanwar20] Pankaj Kumar [pankajkmr22]

Issued: 07.11.2023 Submission: 15.11.2023 15:00 h

Introduction

Monte Carlo Simulation Method is a computational tools for the simulation of random variables and the approximation of integrals/expectations. Monte Carlo methods are used in many fields including statistical physics, computational chemistry, statistical inference, genetics, Finance etc.

1. Inverse Transform Method

This method leverages the cumulative distribution function (CDF) of a probability distribution to transform uniform random numbers into values that follow the desired distribution.

a. Let us consider X is an independent and identically distributed (iid) random variable having given pmf:

X	1	2	3	4	5	6
pmf	0.1	0.3	0.2	0.1	0.2	0.1

Generate 500 random samples using the inverse transform method and produce a histogram for those samples.

b. Let us consider X as an independent and identically distributed (iid) random variable following exponential distribution having with CDF is

$P(X \leq x) = F_x(x) = 1 - e^{-\lambda x}\ \lambda \geq 0$

Write a Python function to get pdf from CDF (Use inverse CDF method). Generate 1000 realizations of the random variable X and produce a histogram of your realizations. (Assume $\lambda = 1$).

Solution a

Step 1 Define CDF distribution function Fx (X) = p(X<x)

F(X) = P(X≤x)

F (1) = 0.1

F (2) = 0.1+0.3 = 0.4

F (3) = 0.4+0.2 = 0.6

F (4) = 0.4+0.1 = 0.7

F (5) = 0.7+0.2 = 0.9

F (6) = 0.9+0.1 = 1.0

Step 2 Convert CDF to inverse CDF $X=F^{-1}(u)$

$$F^{-}(u) = \begin{cases} 1\ if\ 0 \leq u < 0.1 \\ 2\ if\ 0.1 \leq u < 0.4 \\ 3\ if\ 0.4 \leq u < 0.6 \\ 4\ if\ 0.6 \leq u < 0.7 \\ 5\ if\ 0.7 \leq u < 0.9 \\ 6\ if\ 0.9 \leq u \leq 1.0 \end{cases}$$

Step 3 Generate random samples using the discrete inverse CDF method

Generate 500 random samples ui from a uniform distribution between 0 and 1 and then apply the inverse CDF to get the corresponding x value using python.

```
[ ]  print(uniform_samples)

[0.98996973 0.04592403 0.80493196 0.96448251 0.19063721 0.54504127
 0.99908008 0.77676358 0.48071534 0.90202887 0.48928847 0.33369332
 0.23335191 0.80295423 0.23076589 0.84387125 0.03548588 0.73509414
 0.353763   0.91747199 0.56390422 0.7990023  0.90176839 0.95751531
 0.71703034 0.12630553 0.65713506 0.94199432 0.32817867 0.60891506
 0.5297497  0.47659635 0.93659119 0.38980713 0.88793288 0.12688
 0.16443662 0.65804202 0.10050879 0.88136165 0.97180616 0.00374697
 0.10901986 0.04627244 0.53289641 0.41391663 0.45417683 0.98140429
 0.31266578 0.40525422 0.3328886  0.23867853 0.5832142  0.75787592
 0.10529948 0.50967126 0.67748291 0.44692332 0.28955338 0.84356264
 0.05313309 0.74922901 0.55785799 0.59469262 0.50109295 0.73037404
 0.81423088 0.5398705  0.50213142 0.60533955 0.35558898 0.51560389
 0.88053823 0.13334018 0.18036719 0.86255297 0.96254638 0.33998204
 0.40564328 0.7574059  0.2480231  0.12175909 0.607546   0.13837704
 0.50380839 0.05217168 0.19555106 0.01824949 0.25719704 0.55946304
```

Figure 1 Random samples (some) Generated Using Python Coding

Step 4 $X=F^{-1}(u)$ has CDF $F(x)$

Step 5 visualization

Figure 1. Shows the probability distribution of a given function inverse using the inverse transform method, which is generated by using colab/python coding.

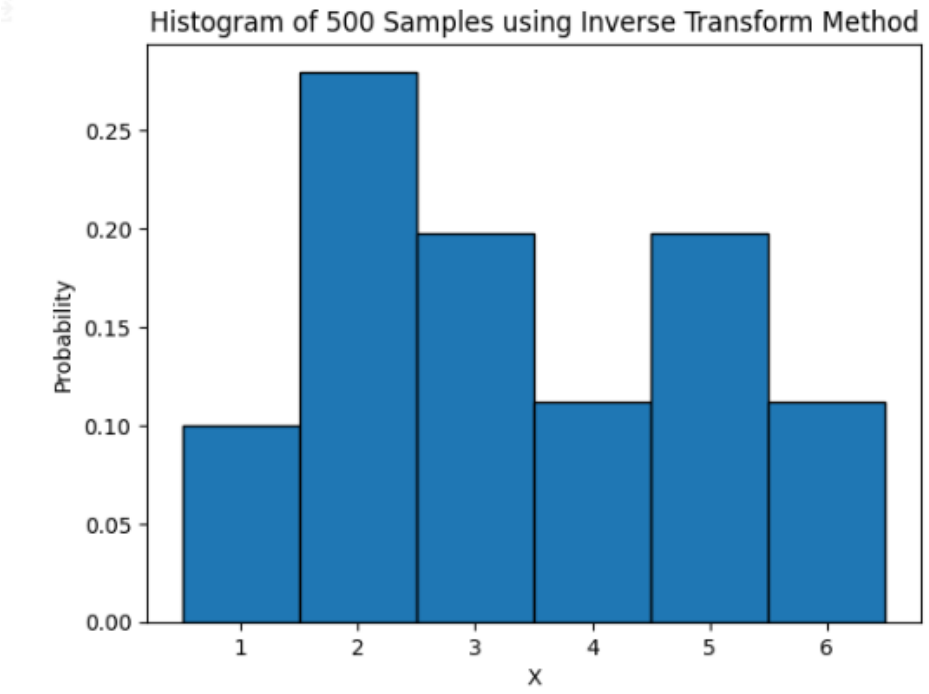


Figure 2: Histogram for 500 Samples Using Inverse Transform Method (python))

Solution b.

Step 1 Define CDF distribution function Fx (X) = p(X<x)

f (x)=1 − $e^{-\lambda x}$ λ ≥ o

Step 2 Convert CDF to inverse CDF X=F^{-1}(u)

$$u = F(x)$$

Step 3 Generate random samples using the continuous inverse CDF method

Generate 1000 realizations of the random variable X and produce a histogram of your realizations. (Assume $\lambda = 1$).

```
print(samples)
```

```
3.60936459e-01 2.98380807e-01 3.79258947e-01 3.61119356e-01
1.83020754e+00 2.80148814e-01 1.37588964e+00 2.47739444e-01
3.24757355e-01 4.44230331e-01 1.80201684e-01 1.03762140e+00
1.52585965e+00 8.46676367e-01 2.85647802e-01 1.21011074e-01
3.31173431e-01 5.96234150e-01 3.02222182e+00 4.71119830e-01
1.28145817e-01 2.31944394e+00 3.56388857e-01 6.95326356e-01
6.38116406e-02 1.20524709e+00 4.94063952e-01 1.71731939e+00
1.60120801e+00 2.91626938e-01 6.97882105e-01 8.66504662e-01
2.68692913e+00 7.41786376e-01 1.11108041e-01 2.47775024e+00
2.05187008e-01 4.07480361e-01 8.73224105e-02 1.40785065e+00
8.99549690e-02 2.01198376e+00 1.24270353e-01 2.62306951e-01
4.18614177e+00 1.16005056e-01 5.68864917e-01 1.59485298e-01
4.49974388e-01 5.92006751e-01 2.68761871e+00 1.97311072e+00
2.29895282e-01 7.71902602e-01 1.71976782e+00 3.03099266e-01
3.32977440e-01 1.02807203e-01 6.39316524e-02 2.95036104e-01
8.67555838e-01 1.63526625e+00 1.28478335e+00 2.64708517e+00
6.34270701e-02 7.21776034e-01 2.00620247e+00 5.55746835e-01
7.57908566e-01 3.96783371e-01 2.85654427e-01 1.73418498e+00
3.82501899e+00 1.05743941e+00 1.41290743e+00 3.16338091e-01
```

Figure 3: Generated Samples (some) using Python Coding for Given Function

Step 4 X=F^{-1}(u) has CDF F(x)

Step 5 visualization

The red line indicated in Fig.4 indicates the boundary condition for acceptance or rejection of random variables. The points outside the region can be rejected in this case, since it doesn't satisfy the given criteria if acceptance criteria is exponential distribution.

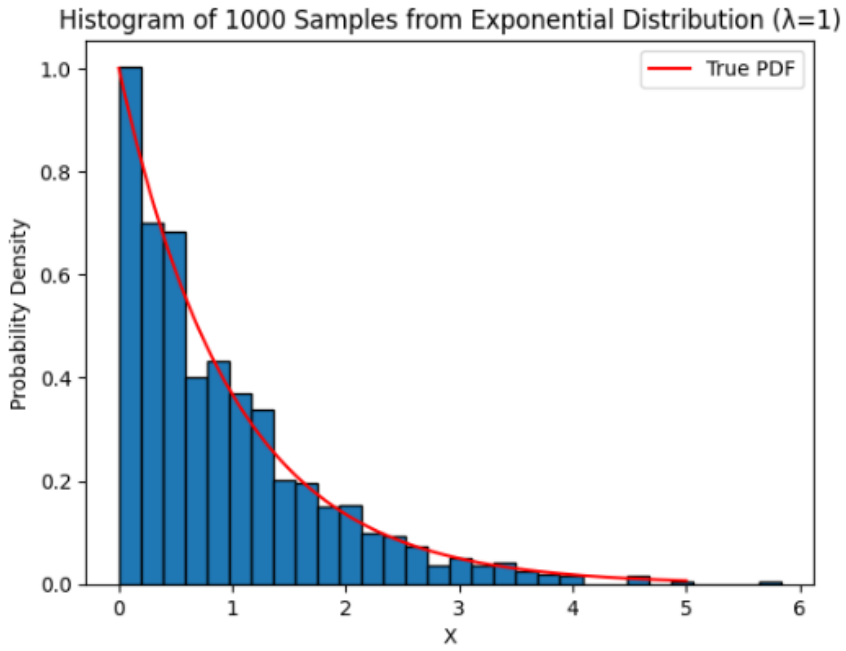


Figure 4: Histogram of 1000 realizations of the Random Variable (Colab)

2. Rejection Method

The basic idea behind the rejection method is to generate samples from a simple and easily sampled "envelope" distribution that entirely encompasses the target distribution of interest. By comparing the generated samples from the envelope distribution with the target distribution, some of them are accepted if they fall within the target distribution, and others are rejected if they fall outside it. This technique involves the following key steps:

a. Generate a random sample from the envelope distribution: This is usually a simple and known distribution, such as a uniform or normal distribution.

b. Generate a random value from a uniform distribution to determine acceptance or rejection.

c. Accept the sample if it falls within the target distribution; otherwise, reject it.

Let us consider X is an independent and identically distributed (iid) continuous random variable following uniform distribution X (0, 10). Draw random variables from the larger space (sample size = 1000) defined by the proposed pdf (Uniform distribution) and reject those not in the region defined by the standard normal distribution ($\mu = 0$, $\sigma =$

1). Change the sample size and discuss the impact of sample size.

Solution (rejection method)

Step 1: Draw a graph of x~ f(x) $f(x) = \frac{e^{\frac{-(x-\mu)^2}{2*\sigma^2}}}{\sqrt{2*\Pi*\sigma}}$

Step 2: Draw u~ g(x), g(x) = $\frac{1}{b-a}$ (a, b) = (0, 10) uniform distribution g(x) = y = (0, 0.1)

Step 3: Acceptance region determination

Normal distribution $f(x) = \frac{e^{\frac{-(x-\mu)^2}{2*\sigma^2}}}{\sqrt{2*\Pi*\sigma}}$ for μ = 0, σ = 1 $f(x) = \frac{e^{\frac{-(x)^2}{2}}}{\sqrt{2*\Pi}}$ for an interval (0, 10)

f (0) = $\frac{e^{\frac{-(0)^2}{2}}}{\sqrt{2*\Pi}}$ = **0.39894** f (10) = $\frac{e^{\frac{-(10)^2}{2}}}{\sqrt{2*\Pi}}$ ~ **0.000**

Accept if the acceptance region is c<=0.39894 is the maximum c value to encapsulate samples.

$\boldsymbol{u \leq \frac{f(x)}{c.g(x)}} = \frac{f(x)}{0.39894*g(x)}$ As we increase the value of C, the encapsulation of target sample increases or we can cover more area.

Uniform distribution g(x) = $\frac{1}{b-a}$ $a \leq x \leq b$ g(x) =0.1 constant value from x= 0 to 10

For encapsulation or covering the target random sample under normal distribution function, we can adjust the value of C, so that it will satisfy the coverage of the target area.

Therefore the area under the normal distribution curve (0, 10) and $u \leq \frac{f(x)}{0.39894*g(x)}$ are accepted values for random variables.

```
0.86754972 0.66452112 0.72142251 0.81627085 0.1245504  0.49531498
0.41068938 0.48430011 0.56287812 0.70742493 0.08798165 0.25121504
0.44038294 0.24668401 0.25016248 0.52077208 0.0044397  0.4937049
0.96468039 0.42641164 0.85797506 0.31218457 0.36815929 0.46565527
0.25244011 0.14919742 0.86004791 0.13176422 0.4927047  0.27160101
0.97272005 0.28704845 0.34378067 0.19327005 0.10809431 0.88603108
0.96651815 0.28761761 0.92822503 0.76745546 0.86318818 0.89909318
0.86044816 0.02589969 0.90576526 0.80958615 0.84755869 0.93199602
0.97637643 0.70061523 0.61973542 0.96720009 0.33348598 0.97198607
0.28386387 0.86425293 0.6238621  0.84748879 0.81401116 0.738042
0.57899701 0.42172856 0.59665397 0.6326614  0.38343486 0.39116115
0.8308603  0.19638558 0.33420548 0.21356799 0.26534754 0.49541049
0.11428673 0.08059944 0.29427321 0.27848186 0.03620023 0.17433678
0.55856533 0.28836218 0.37063436 0.87773859 0.88988012 0.68219691
0.44062842 0.1600052  0.28525941 0.1833349  0.90746685 0.71426495
0.51119795 0.41079258 0.95574693 0.34562423 0.54575877 0.2431192
0.48461166 0.9290532  0.91768104 0.53134962 0.02482811 0.02157248
0.63865264 0.59056738 0.8291505  0.07347775 0.3599743  0.22680929
0.8777737  0.37516069 0.94920759 0.90786268 0.90109504 0.84839349
0.67689188 0.53398485 0.39561106 0.50188634 0.02168939 0.5171844
0.88683755 0.56893006 0.93079261 0.22637582 0.78621695 0.73420063
0.94755305 0.44372322 0.91256202 0.82479864 0.54406907 0.32091526
0.39118074 0.87632688 0.62421611 0.21648856 0.47399159 0.00172293
0.2276988  0.2022041  0.59664207 0.26874076 0.80003864 0.09895793
0.94854201 0.62016055 0.75467943 0.33981243 0.29708589 0.96965371
0.52461157 0.60583511 0.49395445 0.07233172 0.3142776  0.65842547
0.87563393 0.43387712 0.36511767 0.88126257 0.66449787 0.0240639
```

Figure 5: Accepted sample (some) using Rejection Method for 1000 samples

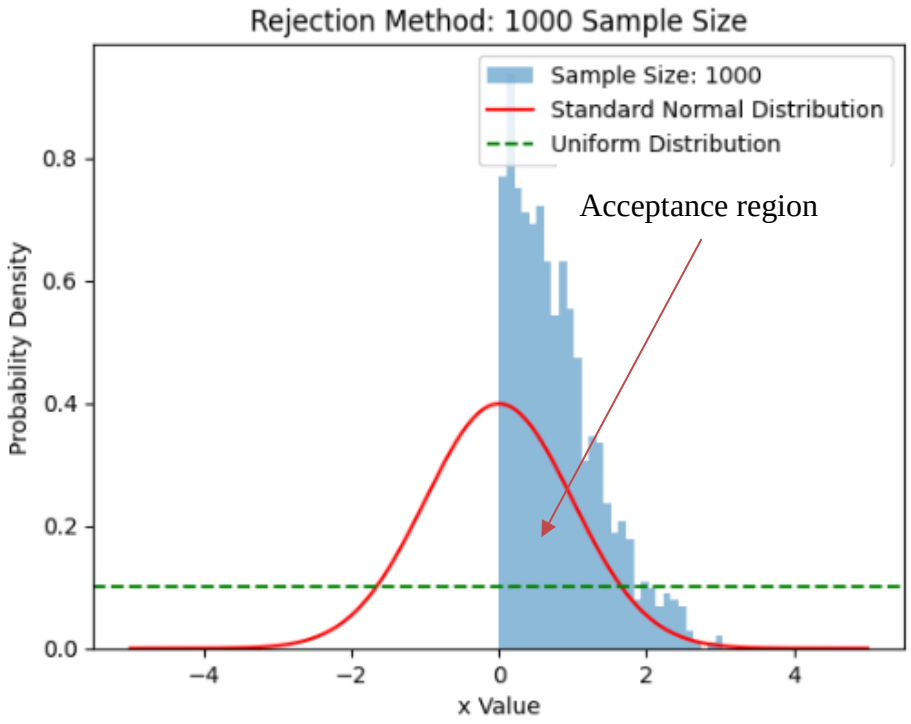


Figure 6 : Rejection Method using 1000 samples

Fig. 6 shows the rejection method using standard and uniform distribution. The blue region inside red line shows the acceptance region for a given criteria and x range (0, 10). Whereas the rejection region is shown by blue color outside of redline for a given condition.

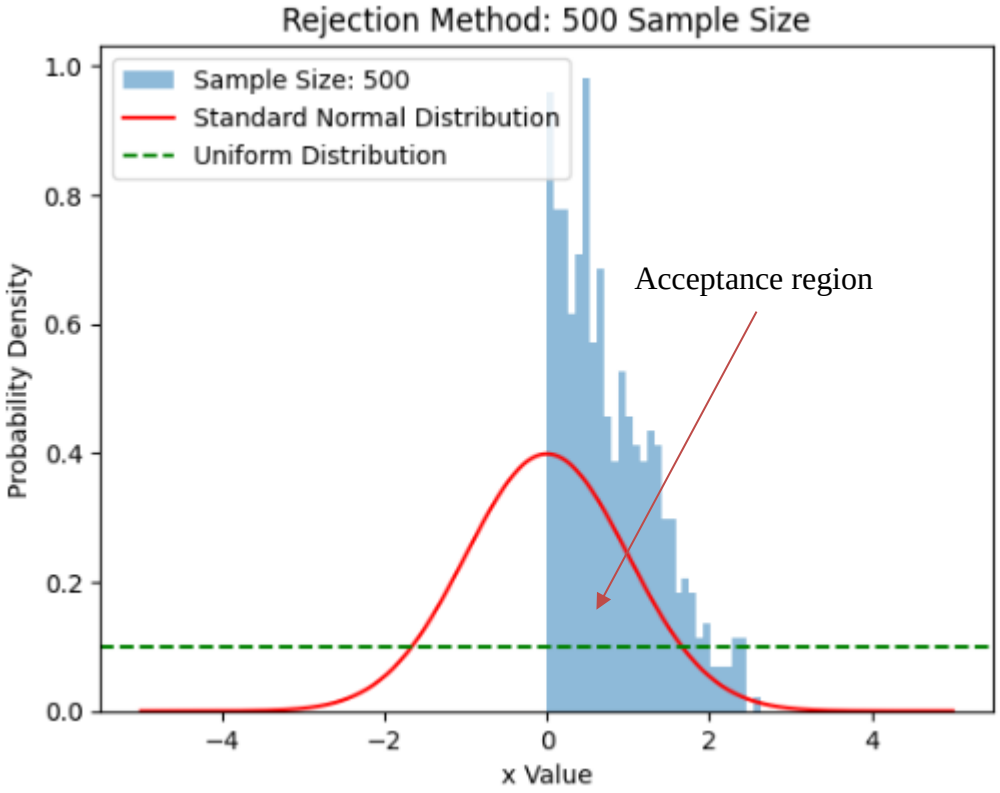


Figure 7: Rejection Method using 500 sample sizes

Fig. 7 shows the rejection method using standard and uniform distribution. The blue region inside red line shows the acceptance region for a given criteria and x range (0, 10). Whereas the rejection region is shown by blue color outside of redline for a given condition.

```
print(accepted_samples)
```

```
0.79840316 0.66605977 0.82381978 0.48796242 0.43471025 0.85606074
0.70854285 0.72742558 0.96564379 0.03761488 0.77221969 0.50642671
0.52484717 0.93363053 0.48507532 0.38530007 0.48396021 0.48178603
0.33221259 0.01519926 0.1881561  0.27327986 0.78567314 0.91217827
0.37906758 0.7727731  0.29720674 0.47586277 0.93595695 0.90299581
0.00102252 0.75660139 0.85996381 0.78196382 0.87151457 0.98569158
0.8926426  0.25028478 0.37205128 0.114099   0.94918673 0.76484918
0.30493323 0.89127512 0.41967483 0.66006019 0.44517541 0.12918773
0.97931882 0.83265512 0.77531432 0.54347336 0.44043001 0.99349712
0.33636232 0.06134735 0.51580774 0.67005201 0.51778663 0.3411065
0.7390309  0.06468742 0.97461504 0.84870152 0.93355038 0.48399568
0.61549737 0.97143073 0.14080236 0.88622561 0.49311492 0.57802055
0.233266   0.92127614 0.34576706 0.29836144 0.11886696 0.87714251
0.89404493 0.58409402 0.99278807 0.62737218 0.72164529 0.61627946
0.8815429  0.4805111  0.80781244 0.79448416 0.49721367 0.73117535
0.14151457 0.49241563 0.97868978 0.92761898 0.75738685 0.18653543
0.21931794 0.58768378 0.04281821 0.59259096 0.66352668 0.79849637
0.7322071  0.74503119 0.78511219 0.19141146 0.79964487 0.68727429
0.9000983  0.45053539 0.4063935  0.01403468 0.97144963 0.98756954
0.59325321 0.02024604 0.58531804 0.84059557 0.4422908  0.40139911
0.40779903 0.47443038 0.71476557 0.12944758 0.95970182 0.15695393
0.81769366 0.02532025 0.9555888  0.55777054 0.7744348  0.72079108
0.12101346 0.81024152 0.65141801 0.6385769  0.75744017 0.39396547
0.22036255 0.05146878 0.2592199  0.66447011 0.17678628 0.15067513
```

Figure 8: Accepted sample (some) using Rejection Method for 500 samples

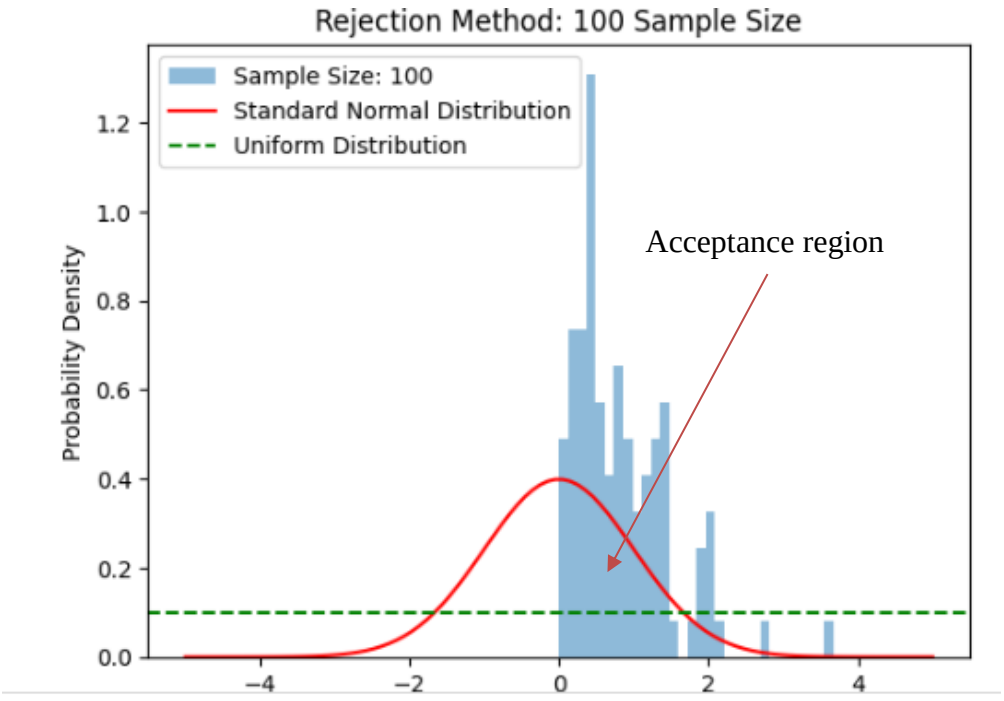


Figure 9: Rejection Method using 100 sample sizes

Fig. 9 shows the rejection method using standard and uniform distribution. The blue region inside red line shows the acceptance region for a given criteria and x range (0, 10). Whereas the

rejection region is shown by blue color outside of redline for a given condition.

```
[0.07940389 0.34247387 0.86311675 0.83695418 0.56261592 0.01351372
 0.61339275 0.67238013 0.91335711 0.84285451 0.45280064 0.9041813
 0.73497984 0.8806933  0.50250988 0.75421552 0.05574266 0.17893733
 0.91348423 0.91550217 0.21331369 0.70606005 0.21050039 0.01388732
 0.11698117 0.68541541 0.89869253 0.90507941 0.49698183 0.53684832
 0.59749726 0.42518511 0.46789528 0.52774799 0.13195931 0.46137993
 0.43046949 0.475818   0.22771821 0.30653891 0.86132647 0.96016413
 0.36730308 0.78150761 0.19776342 0.73267124 0.93351791 0.97134771
 0.45102141 0.63728841 0.81308504 0.21666311 0.87423991 0.00273675
 0.8945787  0.19264624 0.77600419 0.90787538 0.43381195 0.40817811
 0.59225166 0.40169205 0.47116655 0.78479673 0.77152855 0.26279177
 0.92595335 0.92559163 0.75052523 0.69290552 0.84201161 0.33108014
 0.30412039 0.69687155 0.21885336 0.19801752 0.24022101 0.27513291
 0.43001565 0.12994522 0.95958474 0.96927474 0.27131869 0.48800452
 0.0051763  0.69170565 0.00115622 0.089481   0.51493867 0.90995663
 0.82487487 0.88196465 0.59810403 0.06009107 0.92527248 0.62945712
 0.75702046 0.0126023  0.51705736 0.34517097]
```

Figure 10: Accepted sample using Rejection Method for 100 samples

From Fig. 5, 6,7,8,9, and 10, it is shown that the number of the accepted samples decreases as the number of target sample decreases form 1000 to 100. As we increase the sample size, it improves the approximation of the target distribution (standard normal and uniform distribution in this case) Fig.11. With a larger sample size, we are more likely to capture the characteristics of the target distribution, resulting in fewer rejections and a more accurate representation.

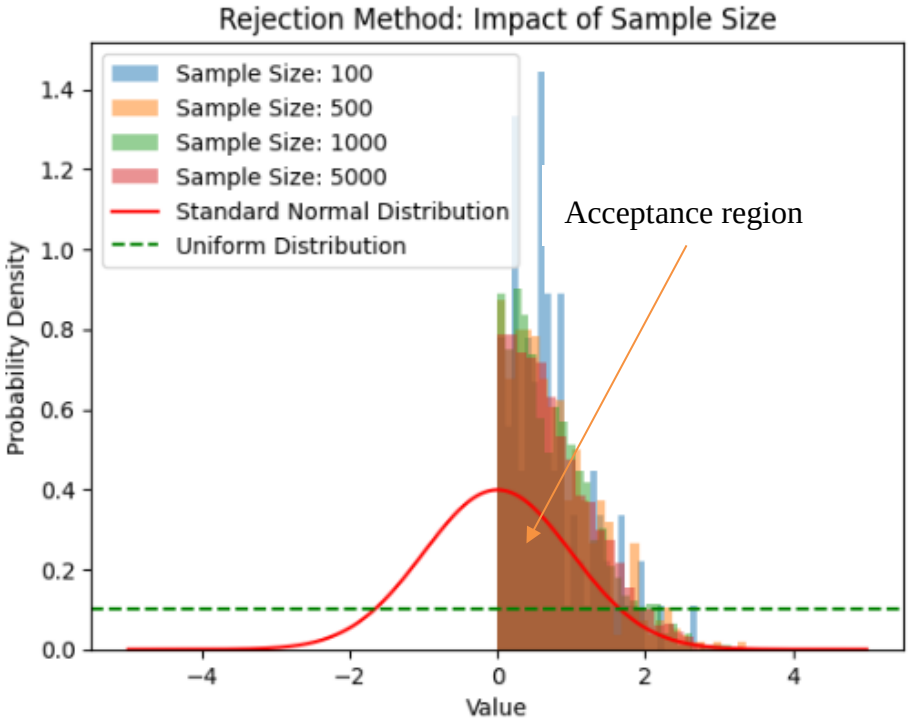


Figure 11: Rejection method for 100,500, 1000 and 5000 sample size

Bonus

Suppose there's a medical test for a rare disease, and this test is not perfect. The test has a sensitivity of 95%, which means it correctly identifies 95% of people who have the disease. It has a specificity of 90%, which means it correctly identifies 90% of people who do not have the disease. The disease is rare, with a prevalence rate in the population of 1%. A person takes the test and receives a positive result. What is the probability that they actually have the disease? (Take the number of Simulation = 1000).

Solution

The given problem can be solved by using Bayes' theorem, using different probability conditions.

P(S): is the total probability of testing positive

P (p) = P (Disease), the prevalence rate of disease in the population (0.01 or 1%).

P (D) = probability of not having disease P (D) = P (No Disease) = 1 – P (Disease) =1-0.01 = 0.99 (99%).

P(S/p): is the probability of testing positive given that the person has the disease, is the sensitivity of the test P (Positive\ Disease) = P (sensitivity) = 0.95 = 95%

P (N) = P (Negative \ Disease) = 1 – P (Positive \Disease) =1-0.95= 0.05 =5%

P (FP) = P (Positive \ No Disease) = P (false positive rate) = 0.10= 10%

P (ND) = (Negative \ No Disease) = P (specificity) = 0.90 = 90%

The probability of having the disease given that positive

$$\text{P (Disease/Positive)} = \text{P(S)} = \frac{P(S/p)*P(P)}{P(S/P)*P(P)+P(Fp)*P(D)} = \frac{(0.95*0.01)}{0.95*0.01 + 0.1*0.99} = 0.08755 = \mathbf{\underline{8.755\%}}$$

Therefore, given a positive test result, the probability that the person actually has the disease is approximately 8.755%.

It is similar with the python calculated value using Bayesian method (0.0876). However, there is slight difference between the Bayesian method (0.0876) and Monte Carlo simulation (0.091) results. It is possibly due to the inherent randomness nature in the Monte Carlo method.

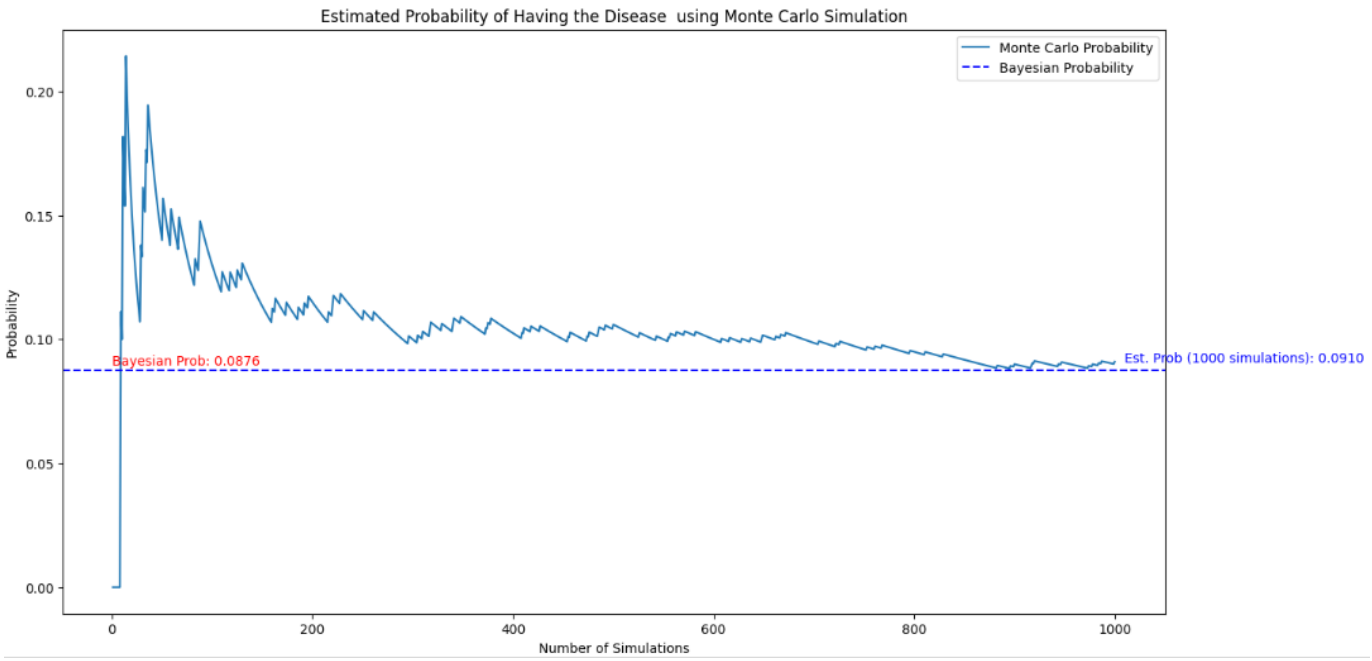


Figure 12: Monte Carlo simulation using 1000 simulation

```
0.08044693 0.08035714 0.08026756 0.08017817 0.08008899 0.08
0.08102109 0.08203991 0.08194906 0.08185841 0.08287293 0.08278146
0.08269019 0.08259912 0.08250825 0.08241758 0.08232711 0.08223684
0.08324206 0.08315098 0.08306011 0.08296943 0.08287895 0.08278867
0.08269859 0.0826087  0.08360478 0.0835141  0.08342362 0.08333333
0.08324324 0.08315335 0.08306365 0.08297414 0.08288482 0.0827957
0.08270677 0.08261803 0.08360129 0.08351178 0.08449198 0.08440171
0.08431163 0.08528785 0.08519702 0.08617021 0.08607864 0.08598726
0.08589608 0.08580508 0.08677249 0.08668076 0.08658923 0.08649789
0.08640674 0.08631579 0.08622503 0.08613445 0.08604407 0.08595388
0.08586387 0.08682008 0.08672936 0.08663883 0.08654849 0.08645833
0.08636837 0.08627859 0.08618899 0.08713693 0.08704663 0.08695652
0.0868666  0.08677686 0.08668731 0.08659794 0.08650875 0.08641975
0.08633094 0.0862423  0.08615385 0.08606557 0.08597748 0.08691207
0.08682329 0.08673469 0.08664628 0.08655804 0.08646999 0.08638211
0.08730964 0.08823529 0.0881459  0.08805668 0.08796764 0.08787879
0.08779011 0.08770161 0.08761329 0.08853119 0.08844221 0.08835341
0.08826479 0.08817635 0.08808809 0.088     ]
```

Figure 13: Probability Generated (some) Using Monte Carlos Generation using Python

From the Fig. 12 and Fig.13, it is shown that Monte Carlo's probability increases or decreases randomly. The Monte Carlo simulations is important due to its ability to analyze sophisticated models, probabilistic systems and provide perceptions into the range of possible outcomes. They are valuable for decision-making in situations with uncertainty, allowing for the exploration of diverse scenarios and the quantification of risk. The simulation shows convergence after many trials.

References

1. *Devore, J. L. (2016). Probability and statistics for Engineering and Science.*
2. *Given Assignment references and coding*

```
# part 1


# import modules
import numpy as np
import matplotlib.pyplot as plt


### Part 1 :  Inverse CDF method to drive PMF for discrete variable


pmf = np.array([0.1,0.3,0.2,0.1,0.2,0.1])                              # Defining pmf of a given function


print(pmf)                                                             # printing pmf of a given function
```

```
     [0.1 0.3 0.2 0.1 0.2 0.1]
```

```
# Cumulative distribution function (CDF)-                              # calcualating the cumulative distribution fucntion
cdf = np.cumsum(pmf)


print(cdf)                                                             # printing the cumulative distribution fucntion
```

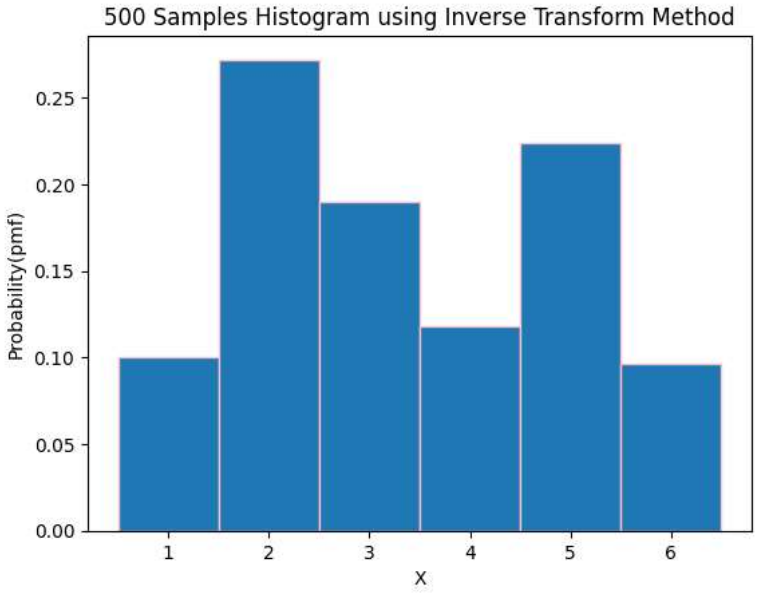
```
     [0.1 0.4 0.6 0.7 0.9 1. ]
```

```
def inverse_transform_sampling(cdf):
    u = np.random.rand()                                               # defines the inverse function u
    for i in range(len(cdf)):
        if u <= cdf[i]:                                                # initalizes iteration
            return i


# Number of samples to generate


sample_size = 500                                                      # determines the number of samples


uniform_samples = np.random.rand(500)                                  # detrmines the number of samples


print(uniform_samples)                                                 # generates the number of samples
```

```
     [0.77337961 0.94179495 0.53373609 0.07581401 0.72345915 0.71722106
      0.88758617 0.18072326 0.95710659 0.58449064 0.09781462 0.20619748
      0.92489628 0.40236189 0.58118079 0.62947915 0.89818709 0.65615626
      0.03588001 0.34292877 0.45408112 0.93022032 0.81828738 0.30341129
      0.85908986 0.80232204 0.70290529 0.36452024 0.68934178 0.03455874
      0.61372961 0.70835478 0.70062212 0.12962479 0.619227   0.35241664
      0.3453029  0.03867056 0.02751781 0.802521   0.86983528 0.07752895
      0.59401688 0.31795113 0.61275772 0.69181435 0.78692284 0.16053498
      0.40566842 0.61689957 0.18259981 0.77892973 0.86194433 0.75956431
      0.02662998 0.08109184 0.42320619 0.96326397 0.53917451 0.58049113
      0.62711251 0.48119394 0.64763248 0.54126004 0.3806434  0.66449454
      0.31386742 0.88891952 0.73737661 0.10383462 0.35768762 0.65345053
      0.78574325 0.81888552 0.919592   0.57735747 0.22821549 0.86462783
      0.65220622 0.84609712 0.48499152 0.42148208 0.98225227 0.61669315
      0.03261893 0.80275323 0.38821516 0.2201547  0.17188717 0.98313354
      0.67774642 0.53054114 0.16702796 0.37518873 0.41643354 0.63002526
      0.54931433 0.12840617 0.40597495 0.09698444 0.04263197 0.83570431
      0.02332066 0.00893696 0.3082699  0.25526159 0.37885453 0.94552614
      0.82885328 0.5253768  0.32728183 0.64007232 0.49102331 0.68026771
      0.71960604 0.19916347 0.65750163 0.02687057 0.54115927 0.05829121
      0.39804816 0.43192516 0.24669739 0.67414461 0.84680555 0.87785628
      0.32061381 0.79070206 0.80664586 0.94044672 0.80542638 0.29727658
      0.44753451 0.42176184 0.29111869 0.72006935 0.43304187 0.06004562
      0.8733544  0.4013872  0.52873529 0.38249692 0.75485825 0.60875795
      0.3577617  0.33531651 0.7666463  0.04723711 0.69947776 0.57266675
      0.05611849 0.44613957 0.92741244 0.37639698 0.94365535 0.85700167
      0.15844034 0.88550481 0.754954   0.43961106 0.05973341 0.02975736
      0.81828698 0.15973772 0.07612713 0.89537689 0.35358166 0.5489033
      0.89328596 0.59070864 0.64433929 0.09353746 0.38593655 0.80201355
      0.98332689 0.85436861 0.44117041 0.03936653 0.15320819 0.92462588
      0.34123558 0.41788034 0.511657   0.45265142 0.58690314 0.77739502
      0.60690011 0.01338826 0.64931403 0.97142921 0.11477986 0.09385557
      0.62494916 0.44908671 0.56120988 0.45044852 0.17448558 0.68708938
      0.23492723 0.02725592 0.44736225 0.47440242 0.86605629 0.7225678
      0.36435383 0.42731898 0.75744518 0.4937845  0.40056282 0.56045413
      0.85879027 0.71172171 0.35923301 0.8491197  0.73994141 0.16150143
      0.23471984 0.99298136 0.79494536 0.05289187 0.06447314 0.3504392
      0.7238525  0.26243785 0.92217042 0.04584385 0.72933759 0.3435783
      0.42778491 0.85891288 0.07378358 0.00945026 0.99177793 0.00772819
      0.56106577 0.17754594 0.85129928 0.68867148 0.84163749 0.13441188
      0.53654608 0.94755134 0.67534448 0.63683448 0.44945367 0.59567022
      0.46281068 0.80409008 0.49758659 0.56200709 0.39933032 0.70356873
```

```
       0.35789571 0.63246878 0.81294052 0.48724061 0.03969298 0.72478704
       0.2471689  0.77613214 0.79580481 0.78597068 0.91266738 0.71888338
       0.19851696 0.59633825 0.01999365 0.74517621 0.04543259 0.8888464
       0.93691927 0.11340365 0.74182289 0.83118937 0.0360216  0.62941474
       0.11595482 0.82783852 0.95428753 0.6941871  0.44052724 0.22062138
       0.87888526 0.93945634 0.95147108 0.82221756 0.11815175 0.14108002
       0.087689   0.94339067 0.15977848 0.74773129 0.80456276 0.69950812
       0.5158054  0.86839736 0.50858567 0.99287489 0.88807501 0.79152239
       0.43158191 0.50598092 0.16918942 0.95990806 0.58538154 0.58712854
       0.25105315 0.63537304 0.3737734  0.17514864 0.98497458 0.62449107
       0.28274461 0.07605921 0.35811158 0.62263606 0.60033983 0.57763273
       0.28233833 0.69850876 0.20892152 0.76014846 0.3243679  0.69891774
       0.59005886 0.15377356 0.16420072 0.03243582 0.63995248 0.67946358
       0.97936786 0.36933168 0.33456931 0.51343016 0.9857273  0.33274408
       0.3887501  0.32485189 0.84084469 0.61637272 0.25604951 0.18467683
       0.52806281 0.00000404 0.6110018  0.89279728 0.2222705  0.72094712
```

```
# Generate random samples using the inverse transform method
samples = np.array([1,2,3,4,5,6])


# Generate random samples using the inverse transform method


inverse_transform_samples = np.searchsorted(cdf, uniform_samples) + 1                          # Adding 1 to convert from 0-based to 1-ba


# Plot a histogram of the generated samples


plt.hist(inverse_transform_samples, bins=np.arange(0.5,7.5,1), density=True, edgecolor='pink')  # defines the histogram bin, range and co
plt.xticks(samples)                                                                             # specify xticks
plt.xlabel('X')                                                                                 # X axis label specification
plt.ylabel('Probability(pmf)')                                                                  # Y axis label specification
plt.title('500 Samples Histogram using Inverse Transform Method')                               # Histogram Legend
plt.show()                                                                                      # plot displaying
```



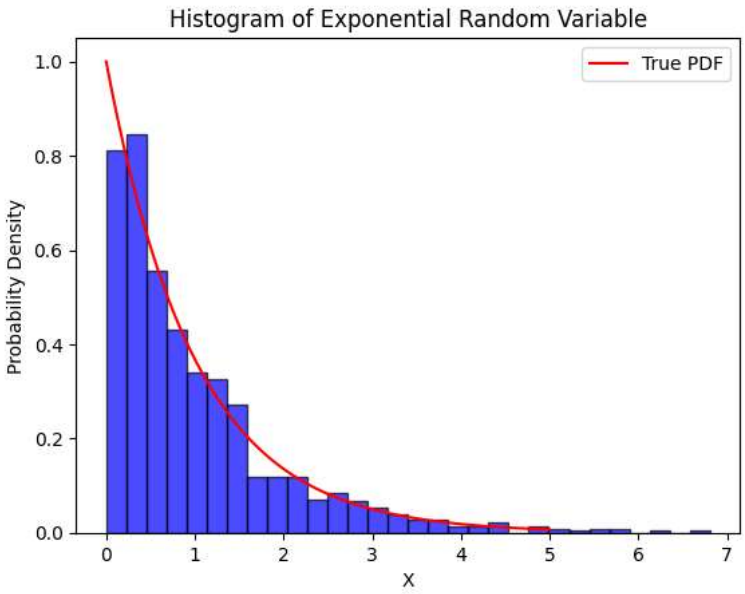

```
# part 2


import numpy as np
import matplotlib.pyplot as plt


# Define the cumulative distribution function (CDF) for the exponential distribution
def exponential_cdf(x, lambda_val):                                                # defines given exponential function
    return 1 - np.exp(-lambda_val * x)


#  Define the inverse CDF for the exponential distribution
def exponential_inv_cdf(u, lambda_val):                                            # defines cdf of inv function
    return -np.log(1 - u) / lambda_val


#  Define the probability density function (PDF) for the exponential distribution
def exponential_pdf(x, size=1):
    return lambda_val * np.exp(-lambda_val * x)
```

```
#  Generate random samples using the inverse transformation method
def generate_exponential_samples(num_samples, size=1):
    # Generate uniform random numbers
    u = np.random.uniform(0, 1, num_samples)                                    # initalize u function

    # Use the inverse CDF to get samples from the exponential distribution
    x = exponential_inv_cdf(u, lambda_val)                                      # x inversion function formulation

    return x


#  Parameters for the standard exponential distribution
lambda_val = 1                                                                  # Given lambda value
num_samples = 1000                                                              # Given sample value

# Generate 1000 random samples using inverse CDF method
samples = generate_exponential_samples(num_samples, lambda_val)
print(samples)
```

```
[3.11255362e-01 3.57740998e-01 4.49573106e-01 9.02028562e-01
 1.15584866e-01 5.14851772e-01 5.79398984e-01 1.40356121e-01
 2.11070125e+00 6.90966074e-01 4.84210109e-01 9.34267315e-01
 4.97529379e-01 3.95167458e-01 1.71203095e+00 1.38691223e+00
 8.26018764e-01 2.55032577e+00 2.44875129e+00 1.31137621e+00
 1.09649072e+00 1.12932446e+00 1.20042456e+00 1.69773407e-01
 8.13078875e-01 3.29236063e-01 3.67930366e-01 7.27663867e-01
 1.96297187e+00 2.37594201e+00 4.76203472e-01 1.96670202e-01
 9.64607091e-01 8.57266169e-01 9.87805115e-01 1.35433985e+00
 2.63480640e+00 3.84248226e-01 5.53444660e-02 1.12297346e+00
 4.83094932e-01 2.11622440e-01 1.15066064e-01 2.57254913e-01
 1.36678273e+00 8.32616780e-02 3.33774157e+00 2.07422484e-01
 4.07466580e-02 4.87103200e-02 6.17234037e-01 8.42009299e-01
 4.22466910e-01 2.45336609e+00 9.18480940e-01 8.35571730e-01
 1.08048334e+00 4.86752386e+00 2.97829798e+00 3.09427423e-01
 8.55949165e-01 7.61393958e-01 1.03134035e-02 1.96650326e+00
 7.76985691e-01 9.87606665e-01 5.03870200e-01 1.12891415e-01
 3.28676560e-01 1.70877354e-01 1.29160879e+00 3.05979531e-01
 1.69647284e+00 1.80134669e+00 6.12255238e-01 5.74249667e-01
 2.10474941e-01 2.24861571e-01 4.37192130e-01 2.87209296e-01
 9.59570838e-01 8.31019186e-01 1.12337669e-01 9.94023335e-01
 8.91642978e-01 1.74507965e-01 4.70065097e-01 2.75129362e-01
 2.07972192e+00 1.16755590e+00 5.79425664e+00 5.55360334e-01
 4.80658967e+00 1.49518617e-01 3.32173644e-01 1.52948620e+00
 1.07416438e-01 6.69067490e-01 4.36831725e-01 8.07455723e-02
 1.05714227e+00 1.75092018e+00 3.13059917e+00 3.02296166e-01
 3.66968258e-01 5.31754751e-01 6.02741403e-01 1.81694126e+00
 4.07859793e+00 1.82883411e+00 1.31086559e+00 3.07533286e+00
 1.04433060e+00 1.20598241e+00 5.92608770e-02 7.21265135e-01
 2.70807341e-01 1.43113469e+00 1.28253003e+00 2.37423076e-01
 3.51073767e-01 7.43347615e-01 1.18321380e+00 2.88454172e-01
 1.57006202e+00 4.19342445e-01 1.15366689e+00 1.17521976e+00
 2.67088012e+00 9.50936839e-01 1.09511139e+00 1.33598693e+00
 9.49116151e-01 5.18991598e-02 1.46271468e+00 4.79630826e-01
 2.54629670e+00 3.49590350e-01 2.92367247e-01 5.14229976e-02
 3.53102438e-01 5.75208593e-01 8.78349936e-01 1.04384985e+00
 4.39488837e-01 1.95340644e+00 9.96836325e-01 8.47050044e-01
 3.27629165e-01 3.10844104e+00 1.28303359e-02 1.55695615e+00
 6.25368449e-02 8.33224926e-01 1.37537010e+00 9.41817500e-01
 1.31018574e+00 2.22032892e+00 5.62307750e-01 1.39444196e+00
 4.36526477e-01 1.86867845e-01 8.48399583e-01 8.77597435e-01
 2.32391609e-01 1.16935218e+00 2.35252132e-01 2.86076587e+00
 6.19324586e-02 1.15164638e+00 5.27744430e-01 6.51838465e-01
 2.79916636e+00 2.38457564e+00 3.67119283e-01 7.86196671e-01
 1.39131224e+00 9.60136481e-01 3.02263550e-01 1.60050028e+00
 5.17924186e-01 5.81697363e-01 4.48671658e-01 1.17003838e+00
 4.85505787e-01 7.20871753e-02 2.34194644e+00 1.70637073e-01
 3.50901874e-01 8.00713186e-02 5.65480440e+00 1.64061110e-01
 2.29431448e+00 3.20226379e-01 3.73257442e-01 1.58089672e+00
 2.17060748e-01 3.93059794e-02 2.88214046e-01 9.51513297e-01
 4.11860777e-01 3.49867662e-01 2.02753032e-01 5.91457735e-02
 3.61674108e-02 6.91493084e-01 5.15140249e+00 2.98045564e+00
 6.26088485e-01 2.77757449e-01 4.96811214e-01 8.68313809e-03
 9.01128924e-01 1.64968021e-01 1.18628533e+00 1.96078747e+00
 3.14293913e-02 5.52529877e-02 2.04224898e-01 4.43455961e-01
 1.86855459e+00 3.31723536e+00 4.26889887e+00 2.07556622e-01
 6.54192275e-02 2.39568928e-01 4.88511674e-01 8.78172915e-01
 1.84190102e-01 1.16067908e+00 5.52873008e-01 2.10873062e-01
```

```
# Plot a histogram of generated samples
plt.hist(samples, bins=30, density=True, alpha=0.7, color='blue', edgecolor='black')    # defining histogram bin size values and color

# Plot the true probability density function (PDF) for comparison
x_values = np.linspace(0, 5, 100)                                               # x values space
pdf_values = exponential_pdf(x_values, lambda_val)                              # defining pdf values using x and lambda
plt.plot(x_values, pdf_values, 'r-', label='True PDF')                          # creating boundary to accept samples
plt.title('Histogram of Exponential Random Variable')                           # legend of histogram
plt.xlabel('X')                                                                 # specifing x label
```

```
plt.ylabel('Probability Density')                                   # specifying y labe
plt.legend()                                                        # plotting legend
plt.show()
```



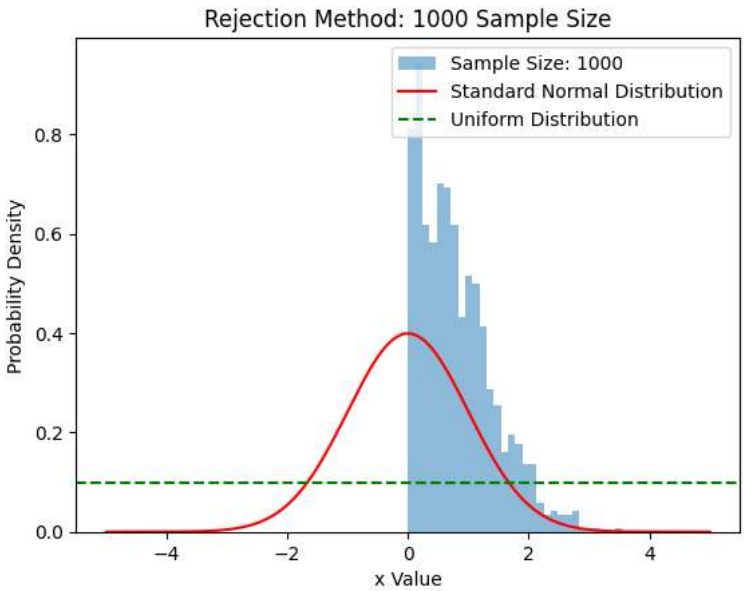

```
# part 3 1000 sample size


import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, uniform

# Parameters for the uniform distribution                           # Defining uniform distribution function
a, b = 0, 10

# Parameters for the standard normal distribution                   # Defining normal distribution function
mu, sigma = 0, 1


def rejection_method(sample_size):
    accepted_samples = []

    while len(accepted_samples) < sample_size:                       # acceptance criteria
        # Generate a random sample from the uniform distribution
        x = np.random.uniform(a, b)

        # Calculate the probability of x under the standard normal distribution    # Defining probability of normal distributi
        p_standard_normal = norm.pdf(x, mu, sigma)

        # Generate a random value from a uniform distribution for acceptance or rejection    # Defining probability of uniform distribut
        u = np.random.uniform(0, 10)

        # Accept the sample if u is less than the ratio of target to envelope distribution    # sample acceptance criteria
        if u < p_standard_normal / uniform.pdf(x, a, b):
            accepted_samples.append(x)

    return accepted_samples


# Number of samples to generate
sample_size =1000                                                   # target sample size determination
accepted_samples=np.random.rand(sample_size)
print(accepted_samples)                                             # printing accepted samples
```

```
[0.02491004 0.4277451  0.71158233 0.56518904 0.91984869 0.88291511
 0.4579625  0.51768748 0.58146576 0.49088371 0.49900626 0.55668897
 0.22920313 0.28432695 0.51952795 0.18397694 0.84438413 0.01867896
 0.891123   0.52386072 0.34830564 0.96952653 0.7786236  0.01021623
 0.76243237 0.6502395  0.37740825 0.87825434 0.12717337 0.70577704
 0.11737589 0.75677513 0.69768343 0.78759029 0.82831892 0.76213379
 0.4054991  0.32743961 0.73369659 0.00555743 0.8619104  0.98866283
 0.40543232 0.43410275 0.82411661 0.00884875 0.08814247 0.44541739
 0.31692839 0.75992908 0.74624473 0.13771954 0.68292778 0.37702741
 0.8826318  0.93580549 0.91656052 0.40999948 0.1409814  0.1782165
 0.58140297 0.33858947 0.87259264 0.6215038  0.12975906 0.04454757
 0.25798375 0.94739686 0.35987931 0.50723186 0.22352448 0.31572568
 0.51782545 0.79529982 0.56556851 0.30725006 0.67106195 0.3857334
 0.56638293 0.91929953 0.28648169 0.3542089  0.3589064  0.67575993
 0.87505799 0.55703593 0.71902569 0.39820479 0.25980832 0.16394712
 0.26396376 0.3626163  0.90940067 0.62026545 0.72959587 0.39510954
```

```
0.50240324 0.45689206 0.75978003 0.08937816 0.89272354 0.53257282
0.80123587 0.52152881 0.00181166 0.13718083 0.93412055 0.9635411
0.2917173  0.79593154 0.32871463 0.32219933 0.77134759 0.49813002
0.2895338  0.49963691 0.35795106 0.39243698 0.97655822 0.84608163
0.56201707 0.64657678 0.72718513 0.46768362 0.98094262 0.49289764
0.77538682 0.71871058 0.46046946 0.84369736 0.00125371 0.22897036
0.07678859 0.73227591 0.93627202 0.85018876 0.13198699 0.77526239
0.73404699 0.08702753 0.31755732 0.38367534 0.50866913 0.6671031
0.55213155 0.80837936 0.89550007 0.42782069 0.78534014 0.0128715
0.29791421 0.0850765  0.4949239  0.14315665 0.12725985 0.22277641
0.90897939 0.05147008 0.37913021 0.53885233 0.82009745 0.47410277
0.14472746 0.35971256 0.56779724 0.04732038 0.16538118 0.65006509
0.09503333 0.58119067 0.25226151 0.38770777 0.26775392 0.87763232
0.09281158 0.05677507 0.91296207 0.19031577 0.89277106 0.32538626
0.07711248 0.6709992  0.70421112 0.59626014 0.89567322 0.13498876
0.28206092 0.65194005 0.01346759 0.58504573 0.19707981 0.99484278
0.34434899 0.52519335 0.77313329 0.23341753 0.1780046  0.2016436
0.1234747  0.11915421 0.31108996 0.83206539 0.51414099 0.83035405
0.73161303 0.89736849 0.65006876 0.29650515 0.54864404 0.6372295
0.85814046 0.14995586 0.01799907 0.29161672 0.79083528 0.51093803
0.29045818 0.24232103 0.15707921 0.75959827 0.67895154 0.84252252
0.9516579  0.06296047 0.01609111 0.10527218 0.17456897 0.22123731
0.88319263 0.1289077  0.06661765 0.90429345 0.19740884 0.53609725
0.36927888 0.44584893 0.19816257 0.83594166 0.10107633 0.72797128
0.95723532 0.29686306 0.38186887 0.82341252 0.10609374 0.83130407
0.57640565 0.51160144 0.45655792 0.47783165 0.97609564 0.01284706
0.72844189 0.47213408 0.59108602 0.70837719 0.22554199 0.94357809
0.92261511 0.02834883 0.45361402 0.83689704 0.40000427 0.71966323
0.75595126 0.24198149 0.5941641  0.83923851 0.74486651 0.94612735
0.76422042 0.78353656 0.93799198 0.22954724 0.68966523 0.06637383
0.23136274 0.9408792  0.74787913 0.19236005 0.54685044 0.35603871
0.97942524 0.14450201 0.73659399 0.06274215 0.69666388 0.73858442
0.30383179 0.6424185  0.29232659 0.43744368 0.85821081 0.2342918
0.88477077 0.74716755 0.46609777 0.85403272 0.57926391 0.24162895
0.22881079 0.10980152 0.49341397 0.49219349 0.77071252 0.03069354
0.64779942 0.09933026 0.18123572 0.83013141 0.0340493  0.66578896
0.2877704  0.34160331 0.52809825 0.60260254 0.62180166 0.55024689
0.0651689  0.05883093 0.06934853 0.38089396 0.91134841 0.20908059
0.03863375 0.60180606 0.49097553 0.0079036  0.2036155  0.99282562
0.91447606 0.23629485 0.31519314 0.51562536 0.87291428 0.18917755
0.51659136 0.40218995 0.1622443  0.41669384 0.76584721 0.31460704
0.86508524 0.41461234 0.91772822 0.24476826 0.7076761  0.50161091
```

```
# Test the rejection method with different sample sizes
sample_sizes = [1000]                                                                    # testing sample size


for size in sample_sizes:
    accepted_samples = rejection_method(size)                                            # generating accepted sample size
    plt.hist(accepted_samples, bins=30, density=True, alpha=0.5, label=f'Sample Size: {size}')  # parameters of histogram

# Plot the standard normal distribution for comparison
x_range = np.linspace(-5, 5, 1000)                                                       # x spacing
y_uniform = 1 / (b - a)                                                                  # y function
plt.plot(x_range, norm.pdf(x_range, mu, sigma), 'r', label='Standard Normal Distribution')  # plot standard normal distributi
plt.axhline(y_uniform, color='g', linestyle='--', label='Uniform Distribution')          # plot standard uniform distribut
plt.title('Rejection Method: 1000 Sample Size')                                          # description of rejection sample
plt.xlabel('x Value')                                                                    # x label
plt.ylabel('Probability Density')                                                        # y label
plt.legend()                                                                             # legend of graph
```

```
<matplotlib.legend.Legend at 0x7950bff05b40>
```



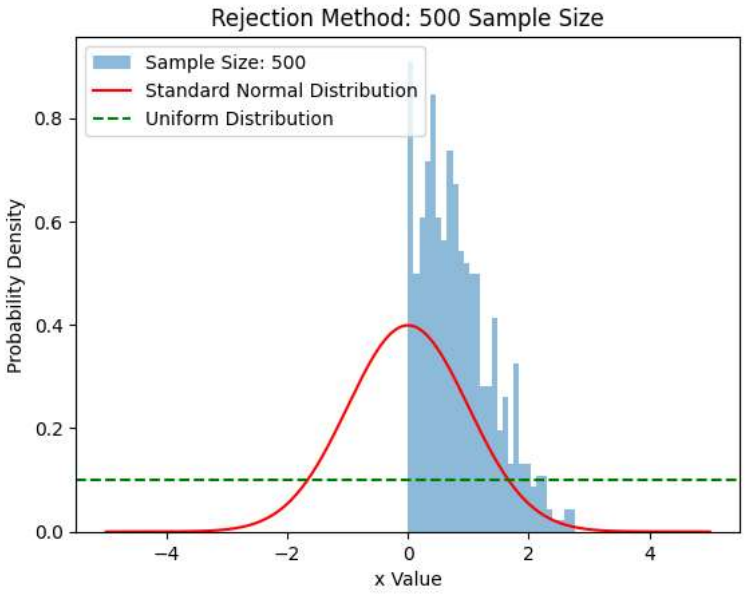
```
# part 3      500 sample size


import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, uniform

# Parameters for the uniform distribution                                                  # Defining uniform distribution function
a, b = 0, 10

# Parameters for the standard normal distribution                                          # Defining normal distribution function
mu, sigma = 0, 1


def rejection_method(sample_size):
    accepted_samples = []

    while len(accepted_samples) < sample_size:                                             # acceptance criteria
        # Generate a random sample from the uniform distribution
        x = np.random.uniform(a, b)

        # Calculate the probability of x under the standard normal distribution          # Defining probability of normal distributi
        p_standard_normal = norm.pdf(x, mu, sigma)

        # Generate a random value from a uniform distribution for acceptance or rejection     # Defining probability of uniform distribut
        u = np.random.uniform(0, 10)

        # Accept the sample if u is less than the ratio of target to envelope distribution    # sample acceptance criteria
        if u < p_standard_normal / uniform.pdf(x, a, b):
            accepted_samples.append(x)

    return accepted_samples


# Number of samples to generate
sample_size =500                                                                           # target sample size determination
accepted_samples=np.random.rand(sample_size)
print(accepted_samples)                                                                    # printing accepted samples
```

```
[2.84108987e-01 1.14380056e-01 9.34662591e-01 1.97937212e-01
 7.81415919e-01 5.92306876e-01 9.46387668e-01 5.82835135e-01
 9.53481957e-01 4.29324887e-01 5.91512179e-01 4.07029821e-01
 9.30657424e-01 9.14728752e-01 2.81504144e-01 2.53994193e-01
 3.30350362e-01 2.29922017e-01 3.71589920e-02 9.90428276e-01
 6.43719470e-01 5.13888388e-01 3.62905102e-01 3.94551862e-02
 5.53582690e-01 3.25150442e-02 8.78838714e-01 7.59008654e-01
 8.16235045e-02 9.87406784e-01 2.22646545e-01 2.79302981e-01
 3.28751361e-01 8.23730345e-01 9.20660467e-01 6.84806718e-04
 6.09661020e-01 6.21895124e-01 5.51128366e-01 4.80407228e-01
 8.78033497e-01 7.79316150e-01 9.38228719e-01 7.42048271e-01
 3.26914202e-01 1.90266344e-01 4.09912295e-01 1.23508238e-01
 1.57221083e-01 6.69386430e-01 3.73253659e-01 3.91203916e-02
 8.63839043e-01 3.20216467e-01 8.83380825e-01 5.25511682e-01
 9.50629918e-01 2.56527786e-01 7.19742167e-01 9.49068759e-01
 2.99650612e-01 7.51139962e-01 8.18863130e-01 3.67719878e-01
 2.73008333e-01 7.09340269e-01 3.30469408e-01 3.08604867e-01
 8.46882582e-01 8.49485369e-01 3.21778236e-01 3.20421945e-02
 8.59906981e-01 8.88263006e-01 8.28283645e-02 1.91860211e-01
 8.08694832e-01 1.78537916e-01 5.92325105e-01 9.51009099e-01
 6.88015216e-01 2.96469081e-01 8.89055626e-01 5.23512136e-01
 8.91642843e-01 9.29254926e-01 7.47495384e-01 1.81437356e-01
 1.53108821e-01 8.23170358e-01 2.31410676e-01 6.99152570e-01
 8.88620629e-01 4.09681477e-01 2.87974638e-01 7.73460746e-01
 3.09198382e-01 5.47249418e-01 8.67700953e-01 9.80866833e-01
 3.98964951e-01 5.93112291e-01 8.63188172e-01 3.55266089e-01
 6.01265949e-01 2.04292769e-01 9.85649797e-01 2.12086920e-01
 7.12750927e-01 9.67275954e-01 5.00348640e-01 4.90247588e-01
 9.20633474e-01 5.49931902e-02 2.62457779e-01 3.36243158e-01
 8.69598504e-01 2.26430663e-01 1.99291793e-01 9.71647965e-01
 4.52436248e-01 5.55073088e-01 4.55807943e-01 7.45314808e-01
 2.93203298e-01 2.37331058e-01 1.36765715e-01 6.69774179e-01
 4.86814868e-01 1.79801789e-01 5.11959205e-01 1.94115795e-01
 9.78607485e-01 6.53939000e-01 6.59413765e-01 5.90729805e-01
 8.21951889e-01 9.69193831e-01 1.76072474e-01 3.15988585e-01
 8.15210731e-01 9.08502759e-01 1.33049563e-01 8.41847123e-01
 9.50918543e-01 8.80435150e-01 1.67368336e-01 7.60241897e-01
 3.11062571e-01 6.66734600e-01 8.15943092e-01 2.56007782e-01
 4.10181136e-01 4.24799489e-01 8.38825491e-02 3.75546285e-01
 5.54220705e-03 1.28057949e-01 3.16864767e-02 6.30665131e-01
 4.55913748e-01 1.66428385e-01 3.08989615e-01 9.98594512e-01
 7.30200217e-01 2.55339702e-01 8.03537456e-01 7.52990434e-01
 7.04609992e-01 3.86720640e-01 6.43919693e-02 4.95510356e-01
 6.17809684e-01 9.64569416e-01 8.60483813e-01 4.85822129e-01
 5.76112220e-01 4.45095627e-01 9.54489493e-01 8.26863131e-01
```

```
4.28621178e-02 3.16219550e-01 5.36565967e-01 7.80891107e-02
8.96219934e-01 6.71675276e-01 5.20115278e-01 2.42972679e-01
4.09002426e-01 4.73340249e-01 3.40809649e-01 6.03230285e-01
8.33984064e-01 7.41178015e-01 8.61767291e-01 4.42173272e-01
6.12286638e-01 5.14106152e-01 7.29553017e-01 5.16284185e-01
3.43140466e-01 1.71539722e-01 4.89719418e-01 3.42397604e-01
9.33423164e-01 9.15237327e-01 7.53302701e-01 9.51528742e-01
2.57747436e-01 1.18746290e-01 4.96060673e-01 4.02367967e-01
8.06947032e-01 6.66587235e-01 2.42972982e-01 1.57880787e-01
5.63701668e-01 7.73711218e-01 5.01208774e-01 8.39836513e-01
2.96412698e-02 1.91837031e-01 5.99108507e-01 8.64968059e-01
7.44714446e-02 9.98270997e-01 6.59593122e-01 6.04053594e-01
8.51526021e-01 9.02533121e-01 7.17464357e-01 1.28056719e-01
```

```
# Test the rejection method with different sample sizes
sample_sizes = [500]                                                                    # testing sample size


for size in sample_sizes:
    accepted_samples = rejection_method(size)                                           # generating accepted sample size
    plt.hist(accepted_samples, bins=30, density=True, alpha=0.5, label=f'Sample Size: {size}')          # parameters of histogram

# Plot the standard normal distribution for comparison
x_range = np.linspace(-5, 5, 500)                                                       # x spacing
y_uniform = 1 / (b - a)                                                                 # y function
plt.plot(x_range, norm.pdf(x_range, mu, sigma), 'r', label='Standard Normal Distribution')          # plot standard normal distributi
plt.axhline(y_uniform, color='g', linestyle='--', label='Uniform Distribution')         # plot standard uniform distribut
plt.title('Rejection Method: 500 Sample Size')                                          # description of rejection sample
plt.xlabel('x Value')                                                                   # x label
plt.ylabel('Probability Density')                                                       # y label
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7950c2806080>
```



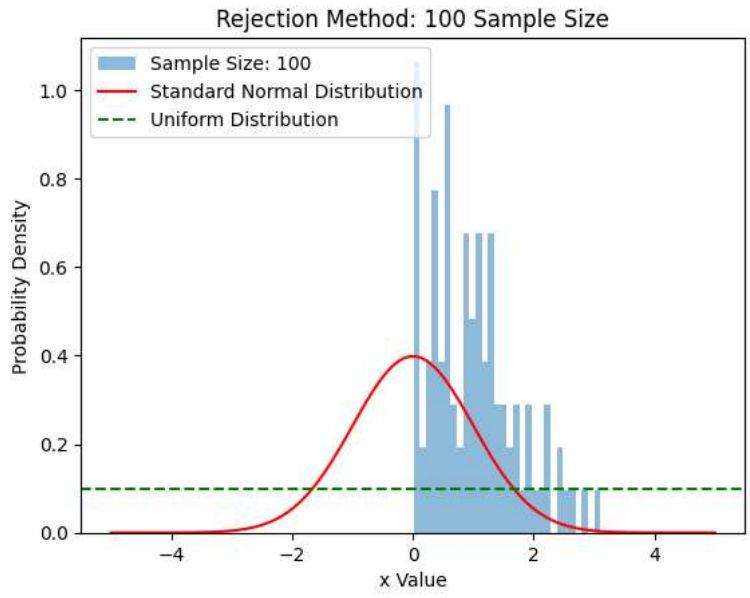

```
# part 3    100 sample size


import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, uniform

# Parameters for the uniform distribution                                               # Defining uniform distribution function
a, b = 0, 10

# Parameters for the standard normal distribution                                       # Defining normal distribution function
mu, sigma = 0, 1


def rejection_method(sample_size):
    accepted_samples = []

    while len(accepted_samples) < sample_size:                                          # acceptance criteria
        # Generate a random sample from the uniform distribution
        x = np.random.uniform(a, b)

        # Calculate the probability of x under the standard normal distribution         # Defining probability of normal distributi
        p_standard_normal = norm.pdf(x, mu, sigma)
```

```
        # Generate a random value from a uniform distribution for acceptance or rejection     # Defining probability of uniform distribut
        u = np.random.uniform(0, 10)

        # Accept the sample if u is less than the ratio of target to envelope distribution    # sample acceptance criteria
        if u < p_standard_normal / uniform.pdf(x, a, b):
            accepted_samples.append(x)

    return accepted_samples


# Number of samples to generate
sample_size =100                                                                              # target sample size determination
accepted_samples=np.random.rand(sample_size)
print(accepted_samples)                                                                       # printing accepted samples
```

```
[0.61426466 0.83742622 0.35551784 0.91499506 0.08887815 0.23276429
 0.31442981 0.69649195 0.3984711  0.270504   0.00903288 0.77269552
 0.99194874 0.98312759 0.80736386 0.06208781 0.86714465 0.6901345
 0.53440943 0.61975454 0.16622499 0.28024405 0.70636177 0.21065507
 0.32606787 0.28765882 0.29767773 0.01788084 0.81396512 0.86195119
 0.78529883 0.78456086 0.25804695 0.55825699 0.2315002  0.1275646
 0.80768649 0.33939285 0.50843151 0.27438452 0.21506506 0.9661047
 0.10817163 0.17307437 0.09926744 0.50673366 0.26660487 0.97094549
 0.66251737 0.9640128  0.57070292 0.82697522 0.9116013  0.6887472
 0.43975714 0.34695021 0.03228203 0.94216105 0.20344917 0.97129222
 0.82347255 0.25626834 0.71288457 0.56520877 0.19337092 0.02833276
 0.22459047 0.25068825 0.53316296 0.46182212 0.97150449 0.61020808
 0.9973088  0.70262489 0.20063346 0.91393044 0.36431198 0.35711882
 0.26828421 0.62486232 0.28497906 0.57364051 0.01474763 0.17818192
 0.46227155 0.30385487 0.77720971 0.25819164 0.08960626 0.07088272
 0.52253151 0.55423499 0.19838889 0.6951495  0.13051134 0.70825748
 0.29537848 0.23075667 0.57739006 0.79697972]
```

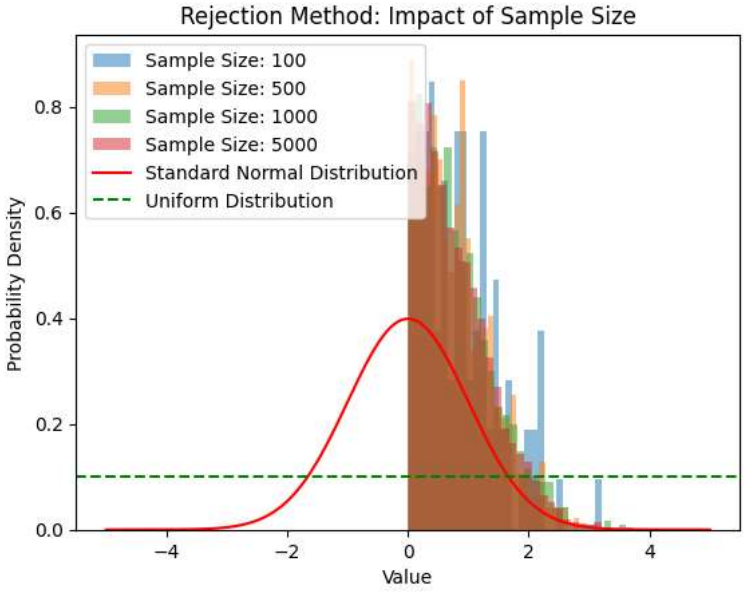
```
# Test the rejection method with different sample sizes
sample_sizes = [100]                                                                          # testing sample size


for size in sample_sizes:
    accepted_samples = rejection_method(size)                                                  # generating accepted sample size
    plt.hist(accepted_samples, bins=30, density=True, alpha=0.5, label=f'Sample Size: {size}')  # parameters of histogram

# Plot the standard normal distribution for comparison
x_range = np.linspace(-5, 5, 100)                                                              # x spacing
y_uniform = 1 / (b - a)                                                                        # y function
plt.plot(x_range, norm.pdf(x_range, mu, sigma), 'r', label='Standard Normal Distribution')     # plot standard normal distributi
plt.axhline(y_uniform, color='g', linestyle='--', label='Uniform Distribution')                # plot standard uniform distribut
plt.title('Rejection Method: 100 Sample Size')                                                 # description of rejection sample
plt.xlabel('x Value')                                                                          # x label
plt.ylabel('Probability Density')                                                              # y label
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7950c0220430>
```




```
# part 3    100,500,1000 and 5000 sample size


import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, uniform
```

```
# Parameters for the uniform distribution                                                    # Defining uniform distribution function
a, b = 0, 10

# Parameters for the standard normal distribution                                            # Defining normal distribution function
mu, sigma = 0, 1


def rejection_method(sample_size):
    accepted_samples = []

    while len(accepted_samples) < sample_size:                                                 # acceptance criteria
        # Generate a random sample from the uniform distribution
        x = np.random.uniform(a, b)

        # Calculate the probability of x under the standard normal distribution               # Defining probability of normal distributi
        p_standard_normal = norm.pdf(x, mu, sigma)

        # Generate a random value from a uniform distribution for acceptance or rejection    # Defining probability of uniform distribut
        u = np.random.uniform(0, 10)

        # Accept the sample if u is less than the ratio of target to envelope distribution   # sample acceptance criteria
        if u < p_standard_normal / uniform.pdf(x, a, b):
            accepted_samples.append(x)

    return accepted_samples


# Number of samples to generate
sample_sizes = [100, 500, 1000, 5000]                                                          # testing sample sizes 100,500,100


for size in sample_sizes:                                                                      # generating accepted sample sizes
    accepted_samples = rejection_method(size)                                                  # accepted samples
    plt.hist(accepted_samples, bins=30, density=True, alpha=0.5, label=f'Sample Size: {size}')  # parameters of histogram

# Plot the standard normal distribution for comparison
x_range = np.linspace(-5, 5, 1000)                                                              # x spacing
plt.plot(x_range, norm.pdf(x_range, mu, sigma), 'r', label='Standard Normal Distribution')      # plot standard normal distributi
plt.axhline(y_uniform, color='g', linestyle='--', label='Uniform Distribution')                 # plot uniform distribution
y_uniform = 1 / (b - a)
plt.title('Rejection Method: Impact of Sample Size')                                            # description of rejection sample
plt.xlabel('Value')                                                                             # x label
plt.ylabel('Probability Density')                                                               # y label
plt.legend()                                                                                    # legend
plt.show()                                                                                      # displaying plot
```




```
# Bonus


import numpy as np
import matplotlib.pyplot as plt

# Define the parameters
sensitivity = 0.95                                                                              # defining sensitivity probability
```

```
prevalence = 0.01                                                    # defining the prevalance probability
specificity = 0.90                                                   # defining the specificity probability


# Number of simulations
num_simulations = 1000                                               # initialize simulations


# Simulate the problem and calculate the probability using inverse CDF method
positive_given_disease = sensitivity * prevalence                     # positive disease probability
negative_given_no_disease = (1 - specificity) * (1 - prevalence)      # negative disease probability


# Normalize the given probabilities
total_probability = positive_given_disease + negative_given_no_disease     # total positive disease probability
positive_given_disease /= total_probability                                # positive given disease probability
negative_given_no_disease /= total_probability                             # negative given disease probability
samples = np.random.choice([1, 0], size=num_simulations, p=[positive_given_disease, negative_given_no_disease])


#  Calculate the probability that the person has the disease given a positive result
probability_disease_given_positive = (sensitivity * prevalence) / ((sensitivity * prevalence) + ((1 - specificity) * (1 - prevalence)))
print("Estimated probability that the person has the disease given a positive test result:", probability_disease_given_positive)  # print
```

```
Estimated probability that the person has the disease given a positive test result: 0.08755760368663597
```

```
# Calculate the probability that the person has the disease given a positive result
probability_estimation = np.cumsum(samples) / np.arange(1, num_simulations + 1)
print("x", probability_estimation)                                                  # generating probability estimation for simulatio
```

```
x [0.         0.         0.         0.         0.         0.
 0.         0.         0.11111111 0.1        0.18181818 0.16666667
 0.15384615 0.21428571 0.2        0.1875     0.17647059 0.16666667
 0.15789474 0.15       0.14285714 0.13636364 0.13043478 0.125
 0.12       0.11538462 0.11111111 0.10714286 0.13793103 0.13333333
 0.16129032 0.15625    0.15151515 0.17647059 0.17142857 0.19444444
 0.18918919 0.18421053 0.17948718 0.175      0.17073171 0.16666667
 0.1627907  0.15909091 0.15555556 0.15217391 0.14893617 0.14583333
 0.14285714 0.14       0.15686275 0.15384615 0.1509434  0.14814815
 0.14545455 0.14285714 0.14035088 0.13793103 0.15254237 0.15
 0.14754098 0.14516129 0.14285714 0.140625   0.13846154 0.13636364
 0.14925373 0.14705882 0.14492754 0.14285714 0.14084507 0.13888889
 0.1369863  0.13513514 0.13333333 0.13157895 0.12987013 0.12820513
 0.12658228 0.125      0.12345679 0.12195122 0.13253012 0.13095238
 0.12941176 0.12790698 0.13793103 0.14772727 0.14606742 0.14444444
 0.14285714 0.14130435 0.13978495 0.13829787 0.13684211 0.13541667
 0.13402062 0.13265306 0.13131313 0.13       0.12871287 0.12745098
 0.12621359 0.125      0.12380952 0.12264151 0.12149533 0.12037037
 0.11926606 0.12727273 0.12612613 0.125      0.12389381 0.12280702
 0.12173913 0.12068966 0.11965812 0.12711864 0.12605042 0.125
 0.12396694 0.12295082 0.12195122 0.12096774 0.128      0.12698413
 0.12598425 0.125      0.12403101 0.13076923 0.12977099 0.12878788
 0.12781955 0.12686567 0.12592593 0.125      0.12408759 0.12318841
 0.12230216 0.12142857 0.12056738 0.11971831 0.11888112 0.11805556
 0.11724138 0.11643836 0.11564626 0.11486486 0.11409396 0.11333333
 0.11258278 0.11184211 0.11111111 0.11038961 0.10967742 0.10897436
 0.10828025 0.10759494 0.10691824 0.1125     0.11180124 0.11111111
 0.11656442 0.11585366 0.11515152 0.11445783 0.11377246 0.11309524
 0.11242604 0.11176471 0.11111111 0.11046512 0.10982659 0.11494253
 0.11428571 0.11363636 0.11299435 0.11235955 0.11173184 0.11111111
 0.11049724 0.10989011 0.10928962 0.10869565 0.10810811 0.11290323
 0.11229947 0.11170213 0.11111111 0.11052632 0.10994764 0.11458333
 0.11398964 0.11340206 0.11282051 0.11734694 0.11675127 0.11616162
 0.11557789 0.115      0.11442786 0.11386139 0.11330049 0.1127451
 0.11219512 0.11165049 0.11111111 0.11057692 0.11004785 0.10952381
 0.10900474 0.10849057 0.10798122 0.10747664 0.10697674 0.11111111
 0.11059908 0.11009174 0.10958904 0.11363636 0.11764706 0.11711712
 0.11659193 0.11607143 0.11555556 0.11504425 0.11453744 0.11842105
 0.11790393 0.1173913  0.11688312 0.11637931 0.11587983 0.11538462
 0.11489362 0.11440678 0.11392405 0.11344538 0.11297071 0.1125
 0.1120332  0.11157025 0.11111111 0.11065574 0.11020408 0.1097561
 0.10931174 0.10887097 0.10843373 0.108      0.11155378 0.11111111
 0.11067194 0.11023622 0.10980392 0.109375   0.10894942 0.10852713
 0.10810811 0.10769231 0.11111111 0.11068702 0.11026616 0.10984848
 0.10943396 0.10902256 0.10861423 0.10820896 0.10780669 0.10740741
 0.10701107 0.10661765 0.10622711 0.10583942 0.10545455 0.10507246
 0.10469314 0.10431655 0.10394265 0.10357143 0.10320285 0.10283688
 0.1024735  0.10211268 0.10175439 0.1013986  0.1010453  0.10069444
 0.10034602 0.1        0.09965636 0.09931507 0.09897611 0.09863946
 0.09830508 0.10135135 0.1010101  0.10067114 0.10033445 0.1
 0.09966777 0.09933775 0.0990099  0.09868421 0.10163934 0.10130719
 0.1009772  0.10064935 0.10032362 0.10322581 0.10289389 0.1025641
 0.10223642 0.10191083 0.1015873  0.10126582 0.10410095 0.10691824
 0.10658307 0.10625    0.105919   0.10559006 0.10526316 0.10493827
 0.10461538 0.10429448 0.10397554 0.10365854 0.10638298 0.10606061
 0.10574018 0.10542169 0.10510511 0.10479042 0.10447761 0.10416667
 0.10385757 0.1035503  0.10324484 0.10588235 0.1085044  0.10818713
 0.10787172 0.10755814 0.10724638 0.10693642 0.10662824 0.1091954
```

```
fig, ax = plt.subplots(figsize=(15, 8))  # Increased plot size
ax.plot(range(1, num_simulations + 1), probability_estimation, label='Monte Carlo Probability')

ax.set_xlabel('Number of Simulations')
ax.set_ylabel('Probability')
ax.axhline(y=positive_given_disease, color='b', linestyle='--', label='Bayesian Probability')

# Adding a title
ax.set_title('Estimated Probability of Having the Disease  using Monte Carlo Simulation')

# Set y-axis limits
#ax.set_ylim(0,1)

# Add numerical value annotation for the true probability
ax.text(0, positive_given_disease + 0.002, f'Bayesian Prob: {positive_given_disease:.4f}', color='r')

# Add numerical value annotation for the estimated probability after 1000 simulations
ax.text(num_simulations + 10, probability_estimation[-1], f'Est. Prob (1000 simulations): {probability_estimation[-1]:.4f}', color='b')

ax.legend()
plt.show()
```