# Fall 2022: Monte Carlo Methods
# Homework 2

**NAME:** Utkarsh Khandelwal
**Net Id:** uk2051

## Exercise 16

We are given a density function and we have to generate samples from it. As instructed we would have to use the inversion to generate samples from it. Given density function $\pi(x)$

$$\pi(x) = \frac{1}{2\sqrt{x}}$$

For the given density, lets calculate the CDF by integrating density function:

$$F(x) = \frac{1}{2}\frac{\sqrt{x}}{\frac{1}{2}} = \sqrt{x}$$

So, if we sample out $U$ from Uniform Distribution $U \sim U(0,1)$ then X would have the distribution of $F(X)$ if $X = F^{-1}(U)$ Calculating $F^{-1}(U)$

$$F(F^{-1}(U)) = U$$

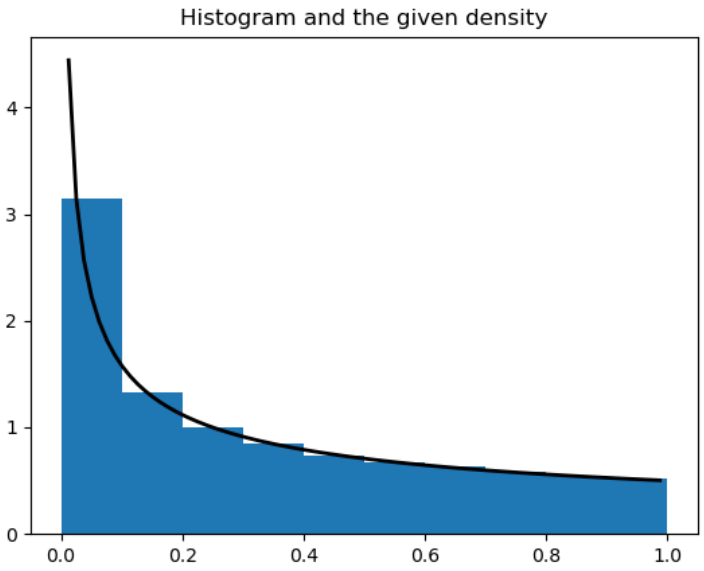
Therefore,

$$\sqrt{F^{-1}(U)} = U$$
$$X = F^{-1}(U) = U^2$$

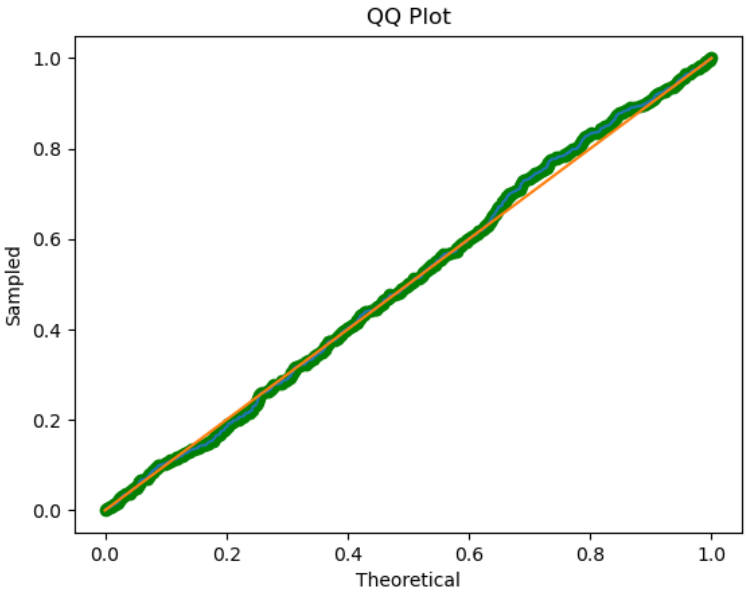
Here, algorithm would be to generate a sample $U$ from $U(0,1)$ and then squaring it to get the sample $X$ from density $\pi(x)$ Below is the code snippet for the same:

```
1: def GenerateNSamplesFromDistribution(N):
2:   points =  np.random.uniform(0,1,N)
3:   return (points ** 2)
```

Below is the image with the histogram of the sampled points and the curve generated analytically using $\pi(x)$



Also, the q-q plot is attached depicting the similarity of the distribution.




**Exercise 18**

This exercise wants us to sample points uniformly over a unit disc. So, let's consider a point in the 2D space. Let the region of the disc be D. So, we have to sample from the joint density

$$p(x,y) = \begin{cases} \frac{1}{2\pi}, \text{ for } x^2+y^2 \leq 1 \\ 0,, \text{ for otherwise} \end{cases}$$

We will convert this into polar coordinate form. So let there be two random variable $r$ and $\theta$, Let

$$x = r\cos\theta \text{ and } y = r\sin\theta$$

Let's calculate the joint density $p(r,\theta)$ So using change of variables,

$$p(x,y) = \frac{p(r,\theta)}{|D`(r,\theta)|}$$

$$|D`(r,\theta)| = \left| \begin{bmatrix} cos(\theta) & -rsin(\theta) \\ sin(\theta) & rcos(\theta) \end{bmatrix} \right| = r$$

$$\frac{1}{\pi} = \frac{p(r,\theta)}{r}$$

$$\frac{1}{\pi} = \frac{p(r,\theta)}{r}$$

$$\frac{r}{\pi} = p(r, \theta)$$

Now, let's independently sample $r$ and $\theta$ from $p(r)$ and $p(\theta)$ respectively. So, this could be simplified as

$$\frac{r}{\pi} = p(r)p(\theta)$$

Let us sample $\theta \sim p(\theta) \sim U[0, 2\pi]$

$$p(\theta) = \frac{1}{2\pi}$$

So,

$$p(r) = 2r$$

We can generate sample from the density $p(r)$ using the inversion. CDF of $p$ will be

$$F_p(r) = r^2$$

Let us sample a point $u_1 \sim U(0, 1)$ So, using inversion, we can write

$$r = F^{-1}(u_1)$$

So,

$$F(F^{-1}(u_1)) = u_1 \Rightarrow F^{-1}(u_1) = \sqrt{u_1} \Rightarrow r = \sqrt{u_1}$$

Now, let's focus on how to sample $\theta$ from $U(0, 2\pi)$ And by change of variables we can sample $\theta$ using a sample $u_2$ from $U(0, 1)$ Let,

$$\theta = 2\pi u_2$$

So,

$$p(\theta) = \frac{p(u_2)}{2\pi}$$

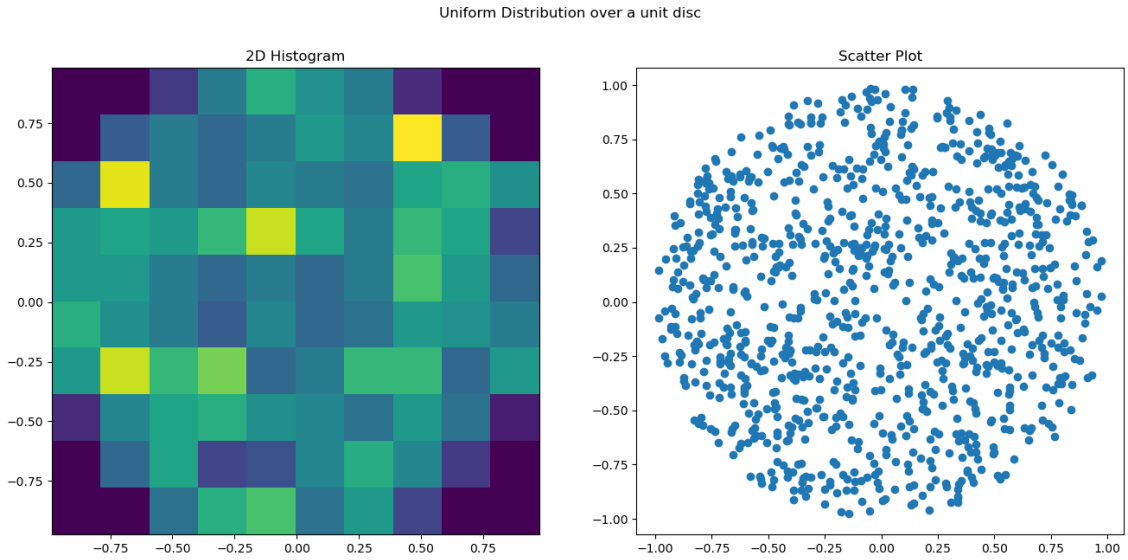
Our algorithm would be then

1. Sample $u_1 \sim U(0, 1)$ and $u_2 \sim U(0, 1)$
2. Compute $\theta$ and $r$ using $\theta = 2\pi u_2$ and $r = \sqrt{u_1}$
3. Compute $x$ and $y$ using $x = r\cos(\theta)$ and $y = r\sin(\theta)$
4. Return the sample $(x, y)$

Below is the code snippet generating $N$ such points

```
1: def GenerateUniformPointsOnADisc(N):
2:   u1 = np.random.uniform(0,1,N)
3:   u2 = np.random.uniform(0,1,N)
4:   r = np.sqrt(u1)
5:   theta = 2 * np.pi * u2
6:
7:   x = np.multiply(r, np.cos(theta))
8:   y = np.multiply(r, np.sin(theta))
9:   return (x, y)
```

Below are the images showing the 2D histogram as well as scatter plots of the points created.




We can see from the images that points generated as uniformly distributed around the disc.

**Exercise 19**

This problem is asking to again sample a unit disc uniformly but using "Rejection sampling" techniques. Let $p(x, y)$ be the density of sampling from the disc and $\tilde{p}(x, y)$ be the density of sampling from a square inscribing the circle. Now, first step is to find $K$,

$$p(x, y) \leq K\tilde{p}(x, y)$$

Therefore,

$$K \geq \frac{p(x, y)}{\tilde{p}(x, y)}$$

We are using uniform distribution over both square and disk. So,

$$\tilde{p}(x, y) = \frac{1}{\text{Area of Square with side 2r}} \mathbb{1}_{-1 \leq x, y \leq 1} = \frac{1}{4r^2} \mathbb{1}_{-1 \leq x, y \leq 1}$$

and

$$p(x, y) = \frac{1}{\text{Area of Circle with radius r}} \mathbb{1}_{x^2+y^2 \leq 1} = \frac{1}{\pi r^2} \mathbb{1}_{x^2+y^2 \leq 1}$$

Hence, using these two, we can write

$$K \geq \frac{\frac{1}{\pi r^2}}{\frac{1}{4r^2}} \times \frac{\mathbb{1}_{x^2+y^2 \leq 1}}{\mathbb{1}_{-1 \leq x, y \leq 1}}$$

This could be further simplified, since $\mathbb{1}_{-1 \leq x, y \leq 1} \subseteq \mathbb{1}_{x^2+y^2 \leq 1}$.So,

$$K \geq \frac{4}{\pi} \times \mathbb{1}_{x^2+y^2 \leq 1}$$

So according to the rejection sampling algorithm for 2D

1. Set $K = \frac{4}{\pi}$
2. Sample $(x, y)$ from $\tilde{p}(x, y)$ density function. In our case $\tilde{p}(x, y)$ is $Uniform(-1, 1) \times (-1, 1)$. So, sample $\tilde{p}(x, y) = \tilde{p}(x) \times \tilde{p}y$. So sample two independent random variable Sample $x \sim U(-1, 1)$ and $y \sim U(-1, 1)$.
3. Sample $u \sim U[0, 1]$
4. If $u \leq \frac{p(x,y)}{K\tilde{p}(x,y)} = \frac{4}{\pi} \times \mathbb{1}_{x^2+y^2\leq 1} \times \frac{\pi}{4} \Rightarrow u < \mathbb{1}_{x^2+y^2\leq 1}$, then $(\alpha, \beta) = (x, y)$, else go to step 2.
5. $(\alpha, \beta)$ will be a single sample point.

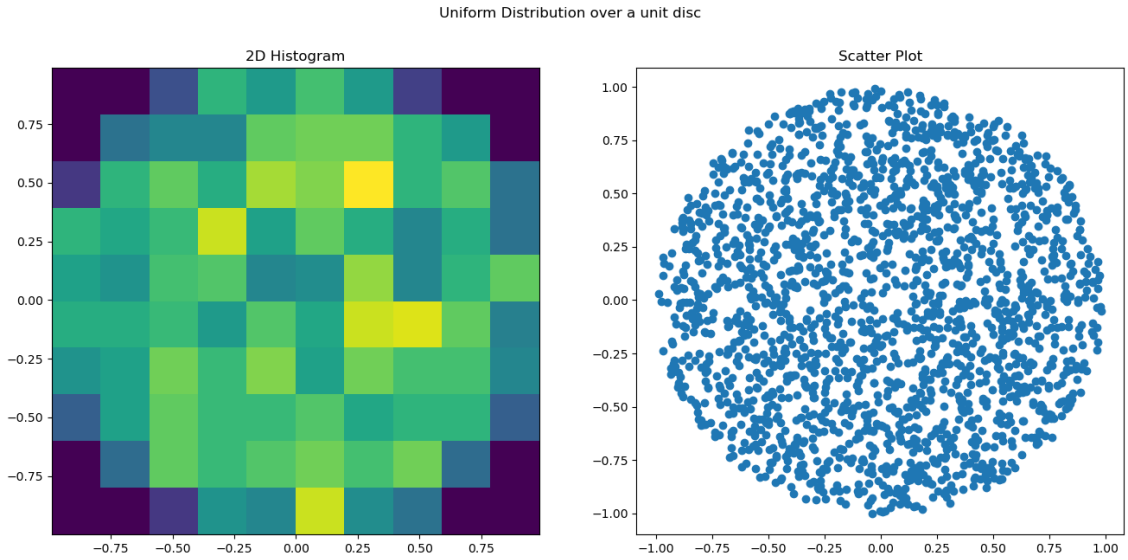
Below is the code snippet doing the same:

```
def GenerateOneSamplePointUsingRejectionSampling(k):
    while(True):
      xC =  np.random.uniform(-1,1,1)
      yC =  np.random.uniform(-1,1,1)
      u =  np.random.uniform(0,1,1)
      if (xC ** 2 + yC ** 2 < 1 and u < 4/(math.pi * k)):
        break
    return (xC, yC)
```

| Comparing Exercise 18 and Exercise 19 | | | |
|---|---|---|---|
| | Expected number of uniform RVs needed for per sample | Time Needed to generate one sample or Iterations needed Name | Theoretical K |
| Exercise 18 | 2.0 | 2.125000000006594e-07 | |
| Exercise 19 | 1.276 | 3.406150000006391e-05 | $\frac{4}{\pi} = 1.273$ |
| Exercise 19 | 1.9965 | 6.824349999997991e-05 | $2$ |
| Exercise 19 | 3.0315 | 0.00014066799999966405 | $3$ |
| Exercise 19 | 4.0285 | 0.00013260400000005744 | $4$ |
| Exercise 19 | 5.0145 | 0.00015445599999945614 | $5$ |
| Exercise 19 | 6.046 | 0.0001861155000001844 | $6$ |
| Exercise 19 | 7.015 | 0.0002158834999994639 | $7$ |
| Exercise 19 | 8.2345 | 0.00025261149999990094 | $8$ |

According to the data in the above table, we can see that for the exercise 18, expected number of uniform random variables needed are fixed while for the exercise 19, expected number needed varies with the value of K. Also, expected number of uniform random variables is approximately equal to the value of K used agreeing with the analytical calculation.

Below image of the samples generated using rejection sampling procedure




**Exercise 20**

We have to calculate $\mathbb{P}(X > 2)$ where X is sampled from $\mathcal{N}(0, 1)$ but we are not allowed to sample from it. Instead we have to sample from $\mathcal{N}(m, \sigma^2$ We know from the classroom lecture that

$$\mathbb{E}[f(x)] = \tilde{f_N} = \frac{1}{N}\sum_{k=1}^{N} f(Y^k) \times \frac{\pi(Y^k)}{\tilde{\pi}(Y^k)}$$

We know that

$$\mathbb{P}(X > 2) = \mathbb{E}[\mathbb{1}_{X>2}]$$

So, $f(Y^k) = \mathbb{1}_{Y^k>2}$ where $Y^k$ is sampled from $\mathcal{N}(m, \sigma^2)$. Below is the code snippet for the estimator of $\mathbb{P}(X > 2)$

```
def SamplePoint(m, sigma):
  return np.random.normal(m, sigma)

def GetWeight(x, m, sigma):
  return 1/( sigma) * np.exp(1/2 * (((x - m) ** 2)/(sigma * sigma) - x**2/1))
def CalculateProbabilityExercise20(m, sigma, N):
  samples = np.random.normal(m, sigma, N)
  weights = GetWeight(samples, m, sigma)
  indicators = np.where(samples > 2, 1, 0)

  fw = np.multiply(indicators, weights)
  weightsSum = np.sum(weights)

  expectation = np.mean(fw)

  trueMean = 1 - norm.cdf(2,loc = 0,scale = 1)

#  fMinusMu = fw - expectation
  fMinusMu = fw - trueMean
  squareFMinusMu = fMinusMu ** 2

```

```
22:     variance = np.mean(squareFMinusMu)
23:     rmse = np.sqrt(variance/N)
24:
25:     return (np.around(expectation, 6), np.around(variance , 6), np.around(rmse ,
                                                    6))
```

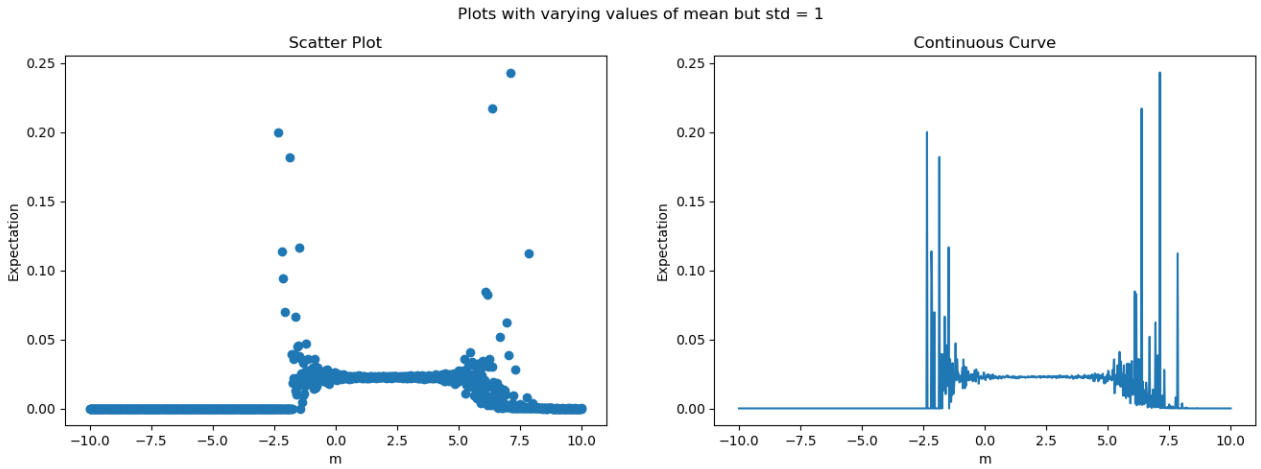
The estimator should predict the value probability to be $0227$ for it to be unbiased. For multiple combinations of $m$ and $\sigma$, I computed the mean, variance and rmse of the estimator. Estimator is unbiased for $m = 0.5$ and $\sigma = 1$, $m = 2$ and $\sigma = 1$ and $m = 2.5$ and $\sigma = 1$

While the minimum Root Mean Square Error I got is for some other combination of the values. Below, is the output of the results generated by the estimator.

0s

| m | sigma | Mean | Variance | RMSE |
|---|---|---|---|---|
| 0 | 1 | 0.0205 | 0.020085 | 0.001417 |
| 0 | 1.5 | 0.010309 | 0.001569 | 0.000396 |
| 0 | 2 | 0.005703 | 0.000648 | 0.000255 |
| 0 | 2.5 | 0.003794 | 0.000515 | 0.000227 |
| 0 | 3 | 0.002546 | 0.000483 | 0.00022 |
| 0.5 | 1 | 0.022169 | 0.007298 | 0.000854 |
| 0.5 | 1.5 | 0.010533 | 0.001039 | 0.000322 |
| 0.5 | 2 | 0.005585 | 0.000562 | 0.000237 |
| 0.5 | 2.5 | 0.003719 | 0.000486 | 0.000221 |
| 0.5 | 3 | 0.002612 | 0.000473 | 0.000217 |
| 1 | 1 | 0.021993 | 0.003053 | 0.000553 |
| 1 | 1.5 | 0.010113 | 0.000764 | 0.000276 |
| 1 | 2 | 0.005649 | 0.00051 | 0.000226 |
| 1 | 2.5 | 0.003582 | 0.000472 | 0.000217 |
| 1 | 3 | 0.002516 | 0.000469 | 0.000217 |
| 1.5 | 1 | 0.023152 | 0.001717 | 0.000414 |
| 1.5 | 1.5 | 0.009972 | 0.000612 | 0.000247 |
| 1.5 | 2 | 0.005795 | 0.000481 | 0.000219 |
| 1.5 | 2.5 | 0.003557 | 0.000464 | 0.000215 |
| 1.5 | 3 | 0.002495 | 0.000465 | 0.000216 |
| 2 | 1 | 0.022894 | 0.001218 | 0.000349 |
| 2 | 1.5 | 0.010058 | 0.000549 | 0.000234 |
| 2 | 2 | 0.00559 | 0.000469 | 0.000217 |
| 2 | 2.5 | 0.003581 | 0.000459 | 0.000214 |
| 2 | 3 | 0.002569 | 0.000463 | 0.000215 |
| 2.5 | 1 | 0.022132 | 0.001226 | 0.00035 |
| 2.5 | 1.5 | 0.009674 | 0.000555 | 0.000236 |
| 2.5 | 2 | 0.005788 | 0.000471 | 0.000217 |
| 2.5 | 2.5 | 0.003647 | 0.000458 | 0.000214 |
| 2.5 | 3 | 0.002531 | 0.000464 | 0.000215 |
| 3 | 1 | 0.023078 | 0.001865 | 0.000432 |
| 3 | 1.5 | 0.009939 | 0.000612 | 0.000247 |
| 3 | 2 | 0.005558 | 0.000485 | 0.00022 |
| 3 | 2.5 | 0.003714 | 0.000466 | 0.000216 |
| 3 | 3 | 0.002449 | 0.000467 | 0.000216 |
| 3.5 | 1 | 0.023252 | 0.003549 | 0.000596 |
| 3.5 | 1.5 | 0.010019 | 0.000756 | 0.000275 |
| 3.5 | 2 | 0.005926 | 0.000517 | 0.000227 |
| 3.5 | 2.5 | 0.003563 | 0.000473 | 0.000218 |
| 3.5 | 3 | 0.002609 | 0.000469 | 0.000217 |

Also, below is the image for the expectation vs m for the fixed value of $\sigma = 1$




We can infer from the image that for value of $\sigma = 1$, there is only a small range of $m$ for which the estimator is unbiased.

**Exercise 21**

Here, we have to find the normalization constant. From the class notes, we know that

$$\frac{1}{N} \times \sum_{k=1}^{N} \frac{p(X^k)}{q(X^k)} \rightarrow \frac{Z_p}{Z_q}$$

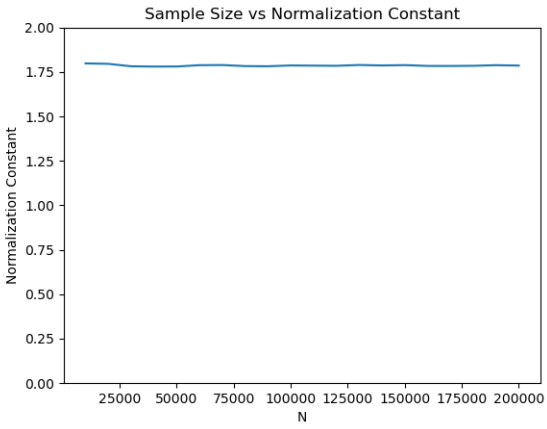
Here, $Y^k$ are sampled from the density $\frac{q(X^k)}{Z_q}$ Here, we need to compute $Zp$. Hence,

$$Z_q \times \frac{1}{N} \times \sum_{k=1}^{N} \frac{p(X^k)}{q(X^k)} = Z_p$$

Below is the code snippet demonstrating the same:

```
1: def ComputeNormalizationConstant(N):
2:   sum = 0
3:   for i in range(N):
4:     x = np.random.normal(0,1)
5:     sum += (math.exp(-(abs(x)** 3)))/(math.exp((-x * x/2)))
6:   Z_p = math.sqrt(2 * math.pi)/N * sum
7:   return Z_p
```

Following is the output of the image




**Exercise 22**

Here, we are asked to repeat exercise 20, but estimator in this case is $\frac{\tilde{f}_N}{\tilde{\mathbb{1}}_N}$. We will be using the following expression for computing it.

$$\frac{\tilde{f}_N}{\tilde{\mathbb{1}}_N} = \frac{\sum_{i=1}^{N} f(Y^k)\frac{p(Y^k)}{q(Y^k)}}{\sum_{i=1}^{N} \frac{p(Y^k)}{q(Y^k)}}$$

Below is the snippet of the routine

```
def CalculateProbabilityExercise22(m, sigma, N):
   samples = np.random.normal(m, sigma, N)
   weights = GetWeight(samples, m, sigma)
   indicators = np.where(samples > 2, 1, 0)
   weightsSum = np.sum(weights)

   normalizedWeights = weights/weightsSum

   fnw = np.multiply(indicators, normalizedWeights)
   expectation = np.sum(fnw)

   trueMean = 1 - norm.cdf(2,loc = 0,scale = 1)


#  fMinusMu = fnw - expectation
   fMinusMu = fnw - trueMean
   squareFMinusMu = fMinusMu ** 2

   variance = np.mean(squareFMinusMu)
   rmse = np.sqrt(variance/N)

   return (np.around(expectation, 6), np.around(variance , 6), np.around(rmse ,
                                                    6))
```

Below is the code snippet which is doing the computation:

```
means = [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5]
sigmas = [1.00, 1.50, 2.00, 2.50, 3.00]

data = []
for i in range(len(means)):
```

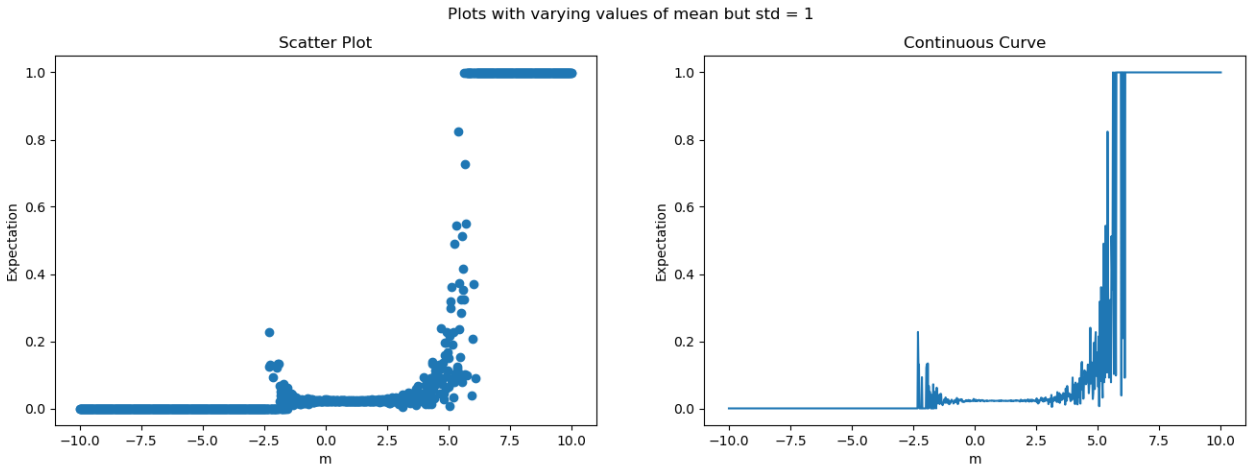
```
  for j in range(len(sigmas)):
    m_20, v_20, r_20 = CalculateProbabilityExercise20(means[i],sigmas[j],10000)
    m_22, v_22, r_22 = CalculateProbabilityExercise22(means[i],sigmas[j],10000)
    data.append((means[i], sigmas[j],m_20, r_20, m_22, r_22))

print(tabulate(data, headers=['m', 'sigma', 'Ex 20: Mean', 'Ex 20: RMSE', 'Ex
                                            22: Mean', 'Ex 22: RMSE']))
```

Below is the image of the results generated using the above comparison function

| m | sigma | Ex 20: Mean | Ex 20: RMSE | Ex 22: Mean | Ex 22: RMSE |
|---|---|---|---|---|---|
| 0 | 1 | 0.0221 | 0.00147 | 0.0223 | 0.000227 |
| 0 | 1.5 | 0.010427 | 0.000398 | 0.021299 | 0.000227 |
| 0 | 2 | 0.005422 | 0.00025 | 0.021258 | 0.000227 |
| 0 | 2.5 | 0.003766 | 0.000226 | 0.023546 | 0.000227 |
| 0 | 3 | 0.002551 | 0.00022 | 0.021842 | 0.000227 |
| 0.5 | 1 | 0.023299 | 0.000874 | 0.023782 | 0.000227 |
| 0.5 | 1.5 | 0.010129 | 0.000318 | 0.023773 | 0.000227 |
| 0.5 | 2 | 0.005893 | 0.000236 | 0.023987 | 0.000227 |
| 0.5 | 2.5 | 0.004054 | 0.00022 | 0.021603 | 0.000227 |
| 0.5 | 3 | 0.00258 | 0.000218 | 0.022572 | 0.000227 |
| 1 | 1 | 0.023825 | 0.000577 | 0.023943 | 0.000227 |
| 1 | 1.5 | 0.009842 | 0.00027 | 0.023925 | 0.000227 |
| 1 | 2 | 0.005505 | 0.000225 | 0.02195 | 0.000227 |
| 1 | 2.5 | 0.003776 | 0.000217 | 0.022763 | 0.000227 |
| 1 | 3 | 0.002618 | 0.000216 | 0.022114 | 0.000227 |
| 1.5 | 1 | 0.021808 | 0.000405 | 0.02338 | 0.000227 |
| 1.5 | 1.5 | 0.010236 | 0.000247 | 0.022872 | 0.000227 |
| 1.5 | 2 | 0.005578 | 0.000219 | 0.023536 | 0.000227 |
| 1.5 | 2.5 | 0.003634 | 0.000215 | 0.022749 | 0.000227 |
| 1.5 | 3 | 0.002516 | 0.000216 | 0.023187 | 0.000227 |
| 2 | 1 | 0.023086 | 0.000351 | 0.022525 | 0.000227 |
| 2 | 1.5 | 0.010229 | 0.000237 | 0.022519 | 0.000227 |
| 2 | 2 | 0.005615 | 0.000216 | 0.023091 | 0.000227 |
| 2 | 2.5 | 0.003553 | 0.000214 | 0.022612 | 0.000227 |
| 2 | 3 | 0.002544 | 0.000216 | 0.021708 | 0.000227 |
| 2.5 | 1 | 0.022878 | 0.000354 | 0.020197 | 0.000227 |
| 2.5 | 1.5 | 0.010033 | 0.000236 | 0.022803 | 0.000227 |
| 2.5 | 2 | 0.005407 | 0.000216 | 0.022232 | 0.000227 |
| 2.5 | 2.5 | 0.003712 | 0.000214 | 0.021696 | 0.000227 |
| 2.5 | 3 | 0.002636 | 0.000216 | 0.024022 | 0.000227 |
| 3 | 1 | 0.023155 | 0.000434 | 0.02661 | 0.000227 |
| 3 | 1.5 | 0.010254 | 0.00025 | 0.024 | 0.000227 |
| 3 | 2 | 0.005659 | 0.00022 | 0.022107 | 0.000227 |
| 3 | 2.5 | 0.003885 | 0.000215 | 0.023192 | 0.000227 |
| 3 | 3 | 0.002645 | 0.000216 | 0.021613 | 0.000227 |
| 3.5 | 1 | 0.022576 | 0.000581 | 0.039352 | 0.000227 |
| 3.5 | 1.5 | 0.010274 | 0.000279 | 0.022005 | 0.000227 |
| 3.5 | 2 | 0.005792 | 0.000227 | 0.022247 | 0.000227 |
| 3.5 | 2.5 | 0.00363 | 0.000218 | 0.021531 | 0.000227 |
| 3.5 | 3 | 0.002578 | 0.000217 | 0.024379 | 0.000227 |

Also, below is the image for the expectation vs m for the fixed value of $\sigma = 1$




We can infer from the image that for value of $\sigma = 1$, there is only a small range of $m$ for which the estimator is unbiased.

We can compare the estimators in exercise 20 and exercise 22, by the relation given in class notes.

$$\text{rmse}^2\left(\frac{\tilde{f}_N}{\tilde{\mathbb{1}}_N}\right) = var(\tilde{f}_N) + a\left(\frac{\pi(x)}{\tilde{\pi}(x)}\right) - b\left(\frac{\pi(x)}{\tilde{\pi}(x)}\right)$$

So, the variance of both the estimators are related by this complex relation where $a$ depends on the variance and $b$ depends on the covariance. So, according to the values of $m$ and $\sigma$ the ratio of $\frac{\pi(x)}{\tilde{\pi}(x)}$ will be change and so will the variance and covariance. Hence, we can't say with certainity without knowing the value of $m$ and $\sigma$.

So, the preference of the estimator will depend on the value of $m$ and $\sigma$.