SDS386D: Monte Carlo Methods in Statistics **Cesar N. Yahia**
Homework 1 cy5799

1.

Rejection sampling: use rejection sampling to sample from density function
$f(x) \propto (-\log x)^2 x^3 (1-x)^2$ for $0 < x < 1$
Carefully detail methods, provide a histogram of samples you obtained, and find approximate or actual probability of acceptance.

To implement rejection sampling, $f(x)$ can be decomposed into a $f(x) = g(x)h(x)$ where $g(x)$ is a bounded function and $h(x)$ is a density that can be sample from.

One possible density $h(x)$ could be the uniform density on $U(0,1)$ as long as $g(x) = (-\log x)^2 x^3 (1-x)^2$ is bounded over $(0,1)$. Since $g(x)$ is proportional to a density function on $(0,1)$ that integrates to 1, it has to be bounded.

To find the bound on $g(x)$, check if the function is concave. If so, the upper bound is obtained by setting the first derivative to zero. Alternatively, the value of x that maximized $g(x)$ also maximizes $log(g(x))$. Therefore, check the concavity of $log(g(x))$ and find the value of x that maximizes $log(g(x))$. Note that $log(g(x))$ is defined on the given domain.

$$
\begin{aligned}
z = \log(g(x)) &= \log((-\log x)^2 x^3 (1-x)^2) \\
z = \log(g(x)) &= 2\log(-\log x) + 3\log x + 2\log(1-x) \\
\frac{dz}{dx} &= 2\frac{-1/x}{-\log x} + \frac{3}{x} + 2\frac{-1}{1-x} \\
\frac{dz}{dx} &= 2\frac{1/x}{\log x} + \frac{3}{x} - 2\frac{1}{1-x} \\
\frac{d^2z}{dx} &= 2\frac{(-1/x^2)(\log x) - (1/x)(1/x)}{(\log x)^2} - \frac{3}{x^2} - 2\frac{1}{(1-x)^2} \\
\frac{d^2z}{dx} &= 2\frac{(1/x^2)(-(\log x) - 1)}{(\log x)^2} - \frac{3}{x^2} - 2\frac{1}{(1-x)^2}
\end{aligned}
$$

Over the domain of interest, $\frac{d^2z}{dx}$ is negative and z is concave. This is further shown in the graphs below.

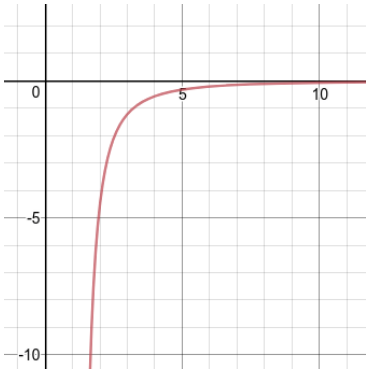

Figure 1: graph showing the double derivative $\frac{d^2 \log(g(x))}{dx}$

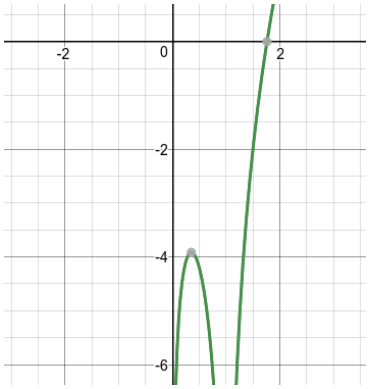

Figure 2: graph showing the function $\log(g(x))$

To maximize z set $\frac{dz}{dx} = 0$ and solve for x. Numerically, we obtain that $\tilde{x} = 0.352$
Therefore, $g(x)$ is maximized at $\tilde{x} = 0.352$
Substitute, $\tilde{x}$ in $g(x)$ to get $M = 0.019966$. However, since $f(x)$ is not normalized, this does not correspond to the acceptance probability. The acceptance probability is $p = \frac{1}{cM}$ where c is unknown.

However, we can find the value of c in this case via numerical integration since we are in a one dimensional space. $\int_0^1 cf(X) = 1$ gives $c = 117$. Therefore, the acceptance probability $p = \frac{1}{cM} = 0.426 = 42.6\%$. This is quite good for Monte Carlo methods.

Now to implement the rejection sampling algorithm, draw an instance x of a uniform random variable from $(0, 1)$. Draw u from another independent uniform random variable on $(0, 1)$. Evaluate $g(x)/M$, if $u < g(x)/M$, accept x as coming from $f(x)$. The algorithm was implemented in Python as shown below. The histogram showing the obtained samples is also shown below.

```
import numpy.random as nprand
import math
import matplotlib.pyplot as plt


# Define the function g(x)
def gx(x):
        if x <= 0 or x >= 1:
                raise Exception("Warning x is out of domain")
        else:
                y = float(((-math.log(x))**2) * (x**3) * ((1 - x)**2))
        return y

# implement the sampling algorithm
if __name__ == '__main__':
        M = 0.019966  # set the upper bound on g
        accepted = []
        numofiter = 0
        numaccept = 0
        while numofiter <= 50000:
                x = nprand.uniform(low=0, high=1)
                gofx = gx(x)  # calculate g(x)
                ratio = gofx / M
                u = nprand.uniform(low=0, high=1)
                if u < ratio:
                        accepted.append(x)
                        # computationally calculate acceptance probability
                        numaccept += 1
                numofiter += 1

        acceptprob = float(numaccept) / numofiter
        print(acceptprob)  # Should get in range of 40%
        print(numaccept)
        print(numofiter)

        # plot a histogram of accepted values
        plt.hist(accepted, bins=20, facecolor='g')
        plt.title("f(x)")
        plt.xlabel("x")
        plt.ylabel("frequency")
        plt.savefig('from_uniform.png')
        plt.show()
```

We ran the algorithm for $50,000$ iterations, and we evaluate the acceptance probability computationally as shown in the algorithm. Out of $50,000$ possible x values, $21,400$ were accepted, which gives a computational acceptance probability of $p = 0.428$. This is very close to the theoretical acceptance probability $p = 0.426$ which was calculated via numerical integration of the density function as shown above.

The histogram showing the resulting density $f(x)$ is shown below.

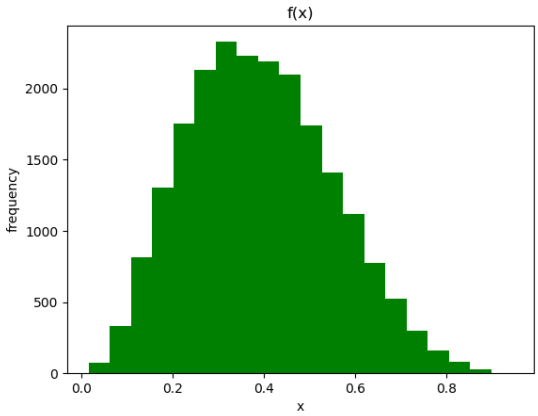

Figure 3: Histogram showing frequency of accepted values that correspond to $f(x)$

Here things are in reverse, we already have samples from a density, but we are looking to get integral with respect to some unnormalized integral - refer to the problem2_clarification handout

2.

Use Monte Carlo methods to evaluate the integral $I = \int_0^1 (-\log x)^2 x^3 (1-x)^{5/2} dx$
let $h(x) = (-\log x)^2 x^3 (1-x)^{5/2}$
let $f(x) = (-\log x)^2 x^3 (1-x)^2$
$l(x) = \frac{h(x)}{f(x)} = (1-x)^{1/2}$
$I = \int_0^1 h(x)dx = \int_0^1 \frac{h(x)}{f(x)} f(x)dx = \int_0^1 l(x)f(x)dx$
Therefore, $\hat{I}$, an estimator for the integral I can be given by:
$\hat{I} = \frac{\frac{\sum_1^N l(x_i)}{N}}{c}$
Where x_i are samples from $f(x)$, and c is a normalizing constant such that $c\int_0^1 f(x) = 1$ and $f_n(x) = cf(x)$ gives a normalized density.
Reasoning:
The integral $\tilde{I} = \int_0^1 l(x) f_n(x)$ is given by $\tilde{I} = \frac{\sum_1^N l(x_i)}{N}$
But, $\hat{I} = \tilde{I}/c$
Therefore, $\hat{I} = \frac{\frac{\sum_1^N l(x_i)}{N}}{c}$

This is the integral of the normalized density, but we want to find the integral of the unnormalized density. The normalized is the unnormalized multiplied by a normalization factor c.

1) Sampling from $f(x)$ was done in part 1 of this homework following the procedure outlined in the previous Python script. Using x_i's sampled from $f(x)$ we can evaluate $\tilde{I} = \int_0^1 l(x) f_n(x) = \frac{\sum_1^N l(x_i)}{N}$

where fn(x) is a density proportional to f(x) since in the sum we are taking samples based on how many times they appear then normalizing by dividing by the total number of samples N so implicitly multiplying by fn(x)

2) At the same time, estimate $c = \frac{1}{\int_0^1 f(x)} = \frac{1}{\sum_1^N f(x_i)/N}$ where x_i's are sampled from $U(0,1)$

3) After estimating values, divide $\tilde{I}$ by c to obtain the desired integral $\hat{I} = \tilde{I}/c = \int_0^1 l(x)f(x) = \int_0^1 (-\log x)^2 x^3 (1-x)^{5/2} dx$

The code implementing the previous steps is shown below.

```
import numpy.random as nprand
import math
import matplotlib.pyplot as plt


# Define the function g(x)
def gx(x):
        if x <= 0 or x >= 1:
                raise Exception("Warning x is out of domain")
        else:
                y = float(((-math.log(x))**2) * (x**3) * ((1 - x)**2))
        return y


# Define the function l(x)
def lx(x):
        l = float((1.0 - x)**0.5)
        return l
```

```
# The function f(x)
def fx(x):
        y = gx(x)
        return y

# implement the sampling algorithm
if __name__ == '__main__':
        M = 0.01996559  # set the upper bound on g
        accepted = []
        numofiter = 0
        numaccept = 0  # number of accepted x_i's
        lxi = []  # numerator of estimator
        fxi = []
        Ihat = []
        while numofiter <= 1000:
                x = nprand.uniform(low=0, high=1)
                gofx = gx(x)  # calculate g(x)
                ratio = float(gofx) / M
                u = nprand.uniform(low=0, high=1)
                if u < ratio:
                        accepted.append(x)
                        # computationally calculate acceptance probability
                        numaccept += 1
                        lxi.append(lx(x))
                fxi.append(fx(u))  # sample f(x) using uniform R.V. to obtain c
                numofiter += 1
                if numaccept > 0:
                        Itildaelem = float(sum(lxi)) / numaccept
                        celem = numofiter / float(sum(fxi))
                        Ihat.append(Itildaelem / celem)

        Itilda = float(sum(lxi)) / numaccept
        c = numofiter / float(sum(fxi))
        estimator = Itilda / c
        print("... Integral estimator ...")
        print(estimator)

        # plot a graph showing convergence to value of integral I
        Iterations = range(1, len(Ihat) + 1)
        plt.plot(Iterations, Ihat, 'g')
        plt.title("integral estimator across iterations")
        plt.xlabel("Iterations")
        plt.ylabel("Integral")
        plt.savefig('integral.png')
        plt.show()
```

For 1000 iterations, the integral estimator was $\hat{I} = 0.0065649$
The value of the integral estimator $\hat{I}$ across iteration is shown in the figure below.

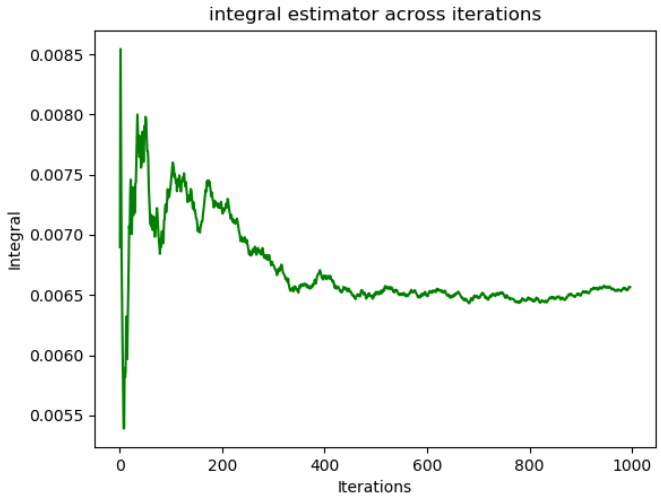

Figure 4: Variation in $\hat{I}$ across iterations

The variance of our estimator $\hat{I}$ can be computationally calculated from the sampled values of $\hat{I}$.
$Var(\hat{I}) = \frac{\sum_1^N (\hat{I}_i - avg(\hat{I}))}{N-1}$
Where $\hat{I}_i$ is the sampled values of $\hat{I}$ across iterations. N is the total number of sampled $\hat{I}$ values. $avg(\hat{I})$ is the average value of $\hat{I}$ across iterations.

The variance of the estimator after 1000 iterations was $Var(\hat{I}) = 8.18E-08$ which is approximately 0. The figure below shows the change in $Var(\hat{I})$ across iterations.

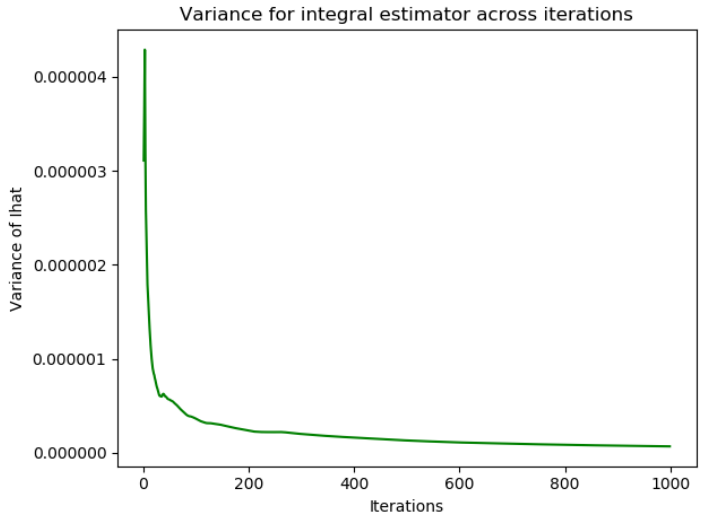

Figure 5: Variation in $Var(\hat{I})$ across iterations

3.

Find the acceptance probability when sampling a standard normal random variable with density $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ using a Cauchy density as a proposal $h(x) = \frac{1}{\pi(1+x^2)}$
For rejection sampling, we need to decompose $f(x) = g(x)h(x)$ where $g(x)$ is a bounded function. Since $f(x)$ and $h(x)$ are given.
$g(x) = \frac{f(x)}{h(x)} = \frac{\pi(1+x^2)e^{-x^2/2}}{\sqrt{2\pi}}$
This function is shown in the figure below. It is a bimodal function and attains a maximum value equal to 1.52

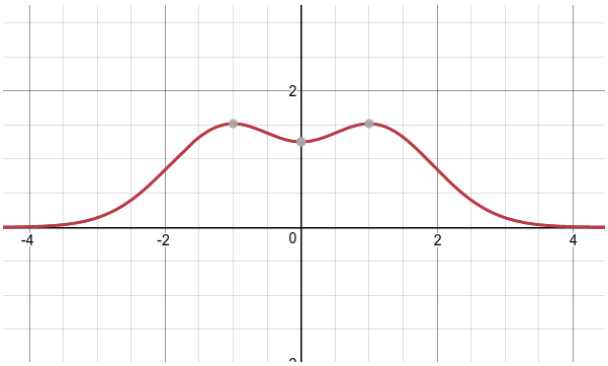

Figure 6: $g(x)$

Therefore, $M = 1.52$ is an upper bound on $g(x)$. This indicates that the acceptance probability for rejection sampling $p = \frac{1}{M} = 0.658$.
Note that here there is no need for a normalizing constant since the Cauchy density integrates to 1.

To verify we need to be able to sample from a Cauchy density. An easy way to sample from a Cauchy density is to use the inverse transform technique. The random variable $X = F^{-1}(U)$ has a Cauchy density where F^{-1} is the inverse of a Cauchy CDF and U is a uniform R.V. on $(0, 1)$.

A Cauchy CDF is given by $F = \frac{1}{\pi} \arctan(\frac{x-x_0}{\gamma}) + \frac{1}{2}$
In our case, $\gamma = 1$ and $x_0 = 0$, so the CDF is given by $F = \frac{1}{\pi} \arctan(x) + \frac{1}{2}$.
This gives us that $X = \tan(\pi(U - \frac{1}{2}))$ is a Cauchy R.V. where U is uniform on $(0, 1)$.

To implement the rejection sampling algorithm, draw an instance x from a Cauchy random variable. Draw u from a uniform random variable on $(0, 1)$. Evaluate $g(x)/M$, if $u < g(x)/M$, accept x as coming from the standard normal density $f(x)$. This algorithm was implemented in Python as shown below.

```
import numpy.random as nprand
import numpy as np
import matplotlib.pyplot as plt

# define a Cauchy R.V. obtained using
# inverse CDF technique
def cauchyRV():
        X = np.tan(np.pi * (nprand.uniform(low=0, high=1) - 0.5))
        return X

# Define the function g(x)
def gx(x):
        numerator = np.pi * (1 + x**2) * np.exp(-0.5 * x**2)
        denominator = np.sqrt(2 * np.pi)
        y = float(numerator) / denominator
        return y

if __name__ == '__main__':
        M = 1.52  # set the upper bound on g
        accepted = []
        numofiter = 0
        numaccept = 0  # number of accepted x_i's
        while numofiter <= 1000:
                x = cauchyRV()  # sample a Cauchy random variable
                gofx = gx(x)  # calculate g(x)
                ratio = float(gofx) / M
                u = nprand.uniform(low=0, high=1)  # sample a uniform on (0,1)
                if u < ratio:
                        accepted.append(x)
                        # computationally calculate acceptance probability
                        numaccept += 1
                numofiter += 1

        acceptprob = float(numaccept) / numofiter
        print("... acceptance probability ...")
        print(acceptprob)  # Should get in range of 65%
        print("... number of accepted proposals ...")
        print(numaccept)
        print("... total number of proposals ...")
        print(numofiter)

        # plot a histogram of accepted values
        plt.hist(accepted, bins=20, facecolor='g')
        plt.title("f(x)")
        plt.xlabel("accepted x")
        plt.ylabel("frequency")
        # fig = plt.gcf()
        plt.savefig('standardnormal.png')
        plt.show()
```

The code implemented above also calculates the acceptance probability computationally by counting the number of accepted proposals and dividing by the total number of proposals. As expected the acceptance probability was obtained to be 0.655 which is very close to the theoretical acceptance probability calculated earlier (0.658). The number of accepted proposals was 655 out of 1000.

The histogram showing the resulting density function $f(x)$ (standard normal) is shown below.

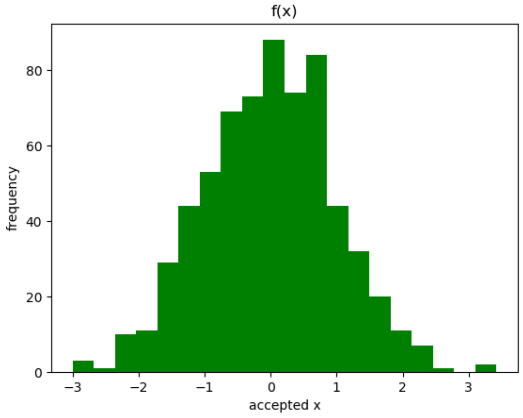

Figure 7: Histogram showing frequency of accepted values that correspond to $f(x)$ (standard normal)