CFRM 505 Homework 2

Eunki Chung eunkich@uw.edu

January 27, 2023

Problem 1

Consider the random variable X whose cdf is given by

$$F_X(x) = \int_0^\infty 2x^y e^{-2y}\,\mathrm{d}y \text{ for all } 0 < x < 1$$

Use Monte Carlo simulation to estimate the mean and variance of this distribution. Use the following methods to generate samples of X (using at least $10,000$ samples for each part).

Part 1

Find a closed form expression for F_X (i.e., do the integral) and then use the inverse transform method.

$$\begin{aligned}
\int_0^\infty 2x^y e^{-2y}\,\mathrm{d}y &= \int_0^\infty 2e^{\ln x^y} e^{-2y}\,\mathrm{d}y \\
&= \int_0^\infty 2e^{y\ln x - 2y}\,\mathrm{d}y \\
&= \int_0^\infty 2e^{y(\ln x - 2)}\,\mathrm{d}y \\
&= \lim_{n\to\infty} \frac{2}{\ln x - 2} e^{y(\ln x - 2)}\Big|_0^n \\
&= \lim_{n\to\infty} \frac{2}{\ln x - 2} e^{n(\ln x - 2)} - \frac{2}{\ln x - 2} \\
&= \lim_{n\to\infty} \frac{2}{\ln x - 2} e^{\ln x^n - 2n} - \frac{2}{\ln x - 2} \\
&= \lim_{n\to\infty} \frac{2}{\ln x - 2} x^n e^{-2n} - \frac{2}{\ln x - 2}
\end{aligned}$$

Notice that $x^n, e^{-2n} \to 0$ as $n \to \infty \quad$ for $0 < x < 1$

$$= -\frac{2}{\ln x - 2}$$

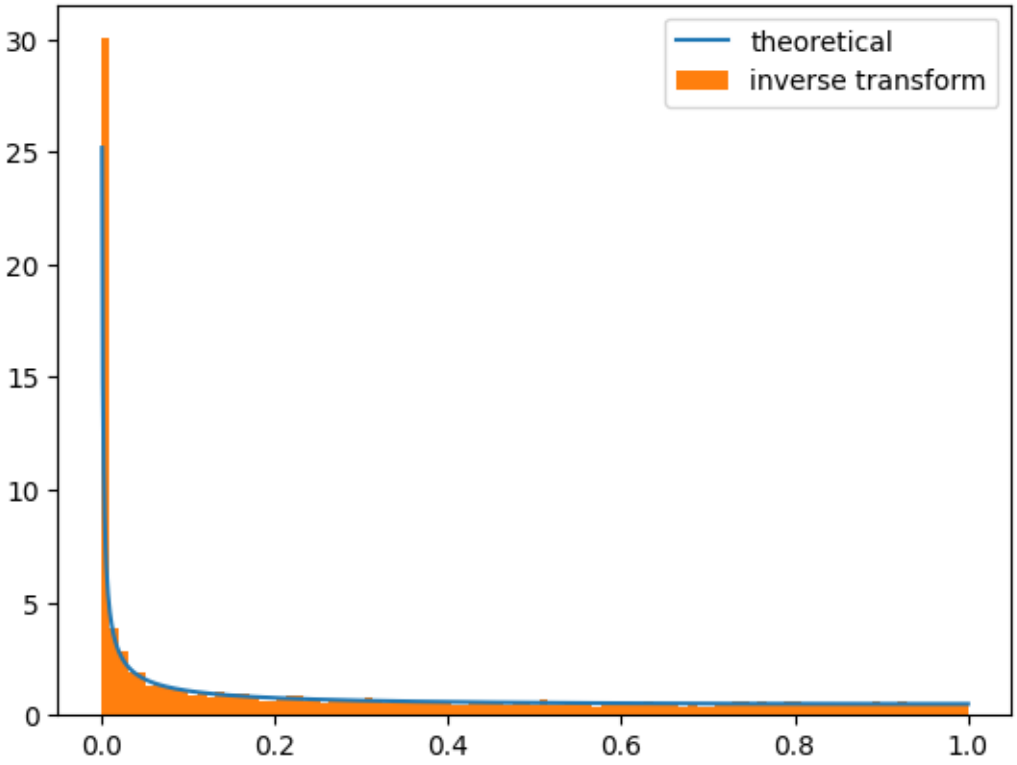
$$F_X(x) = -\frac{2}{\ln x - 2}$$

$$\begin{aligned}
f_X(x) &= \frac{dF_X}{dx} \\
&= \frac{2}{x(\ln x - 2)^2}
\end{aligned}$$

$$\begin{aligned}
u &= \frac{-2}{\ln x - 2} \\
(\ln x - 2)u &= -2 \\
\ln x - 2 &= \frac{-2}{u} \\
\ln x &= 2 - \frac{2}{u} \\
x &= \exp\left(2 - \frac{2}{u}\right)
\end{aligned}$$

$$\therefore \quad X = \exp\left(2 - \frac{2}{U}\right) \qquad U \sim \mathrm{Unif}(0,1)$$

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(12)
u = np.random.random(10000)
x = np.exp(2 - (2 / u))
bins = np.linspace(0, 1, 100)
supp = np.linspace(0, 1, 1000)[1:]
f = 2 / ((np.log(supp) - 2) ** 2 * supp)
plt.plot(supp, f, label="theoretical")
plt.hist(x, bins=bins, density=True, label="inverse transform")
plt.legend()
plt.show()
```



Part 2

Use the composition method. (Hint: Use $f_Y(y) = 2e^{-2y}$ and $F_{X|Y}(x; y) = x^y$.)

$$Y \sim \text{Exp}(2)$$

$$Y = \frac{-\ln U}{2} \quad U \sim \text{Unif}(0,1)$$

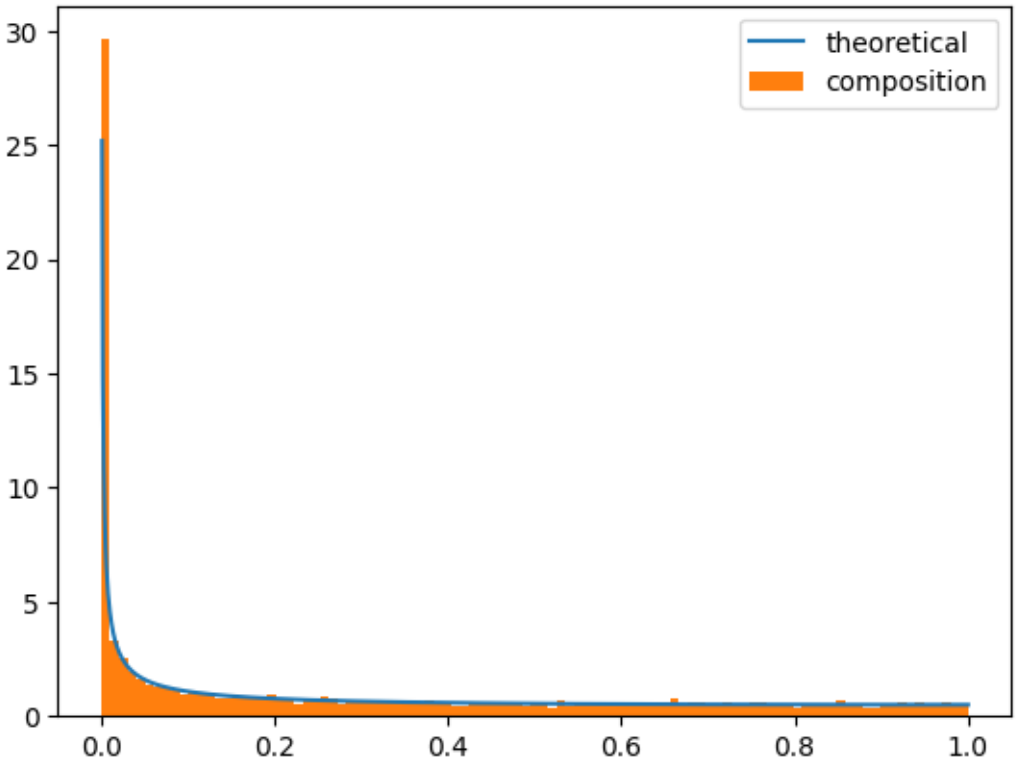
$$F_{X|Y}(x|y) = x^y$$

$$x^y = v$$

$$x = v^{\frac{1}{y}}$$

$$X|Y = V^{\frac{1}{Y}} \qquad V \sim \text{Unif}(0,1), \quad Y \sim \text{Exp}(2)$$

```
y = -np.log(u) / 2
v = np.random.random(10000)
x = v ** (1/y)
plt.plot(supp, f, label="theoretical")
plt.hist(x, bins=bins, density=True, label="composition")
plt.legend()
plt.show()
```



Problem 2

Use Monte Carlo simulation to estimate the mean and variance of $X \sim \text{Poisson}(10)$. Plot a histogram from your data. Use the inverse transform method to generate X and use at least $10,000$ samples.

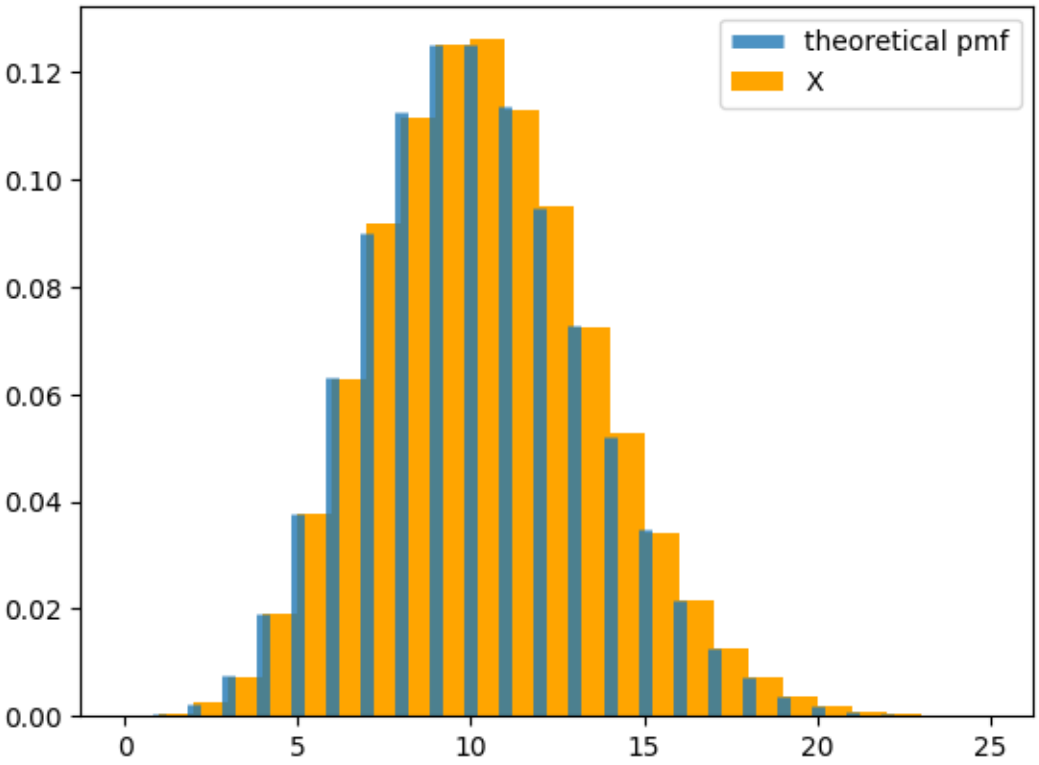
$$E[X] = Var[X] = \lambda = 10$$

```
lamda = 10
size = int(1e5)


def poisson(lamda):
    n = 0
    p = np.exp(-lamda)
    F = p
    U = np.random.random()
```

```
    while U > F:
        p = lamda * p / (n+1)
        F += p
        n += 1
    return n


X = np.empty(size)
for i in range(size):
    X[i] = poisson(10)

print(f"Mean: {np.mean(X)}\tVar: {np.var(X)}")
```

```
Mean: 9.99559   Var: 9.986870551900001
```

```
import scipy.stats as stats
x = np.linspace(0, 25, 26)
plt.vlines(x, 0, stats.poisson(10).pmf(x),
           label='theoretical pmf', alpha=.8, lw=5)
plt.hist(X, bins=25, density=True, color="orange", label="X")
plt.legend()
plt.show()
```



Problem 3

Consider the following "game":

1) Set $r = 1$ and $n = 1$
2) Generate a random vector (X, Y) uniformly distributed on the square $[-1, 1] \times [-1, 1]$.
3) If $X^2 + Y^2 > r^2$, then stop.
4) If $X^2 + Y^2 \leq r^2$, then set r equal to $\sqrt{r^2 - X^2 - Y^2}$ and set n equal to $n + 1$, then start over at step (2).

Your score at the end of this game is $N = n$ (the number of random vectors you calculated). Use Monte Carlo simulation with at least 10,000 samples to simulate this game and estimate the expected score $\mathbb{E}[N]$.

(This game is taken from the lesson by Grant Sanderson in 3b1b. The link has a lovely proof that the expected value is actually $e^{\pi/4}$.)

```
size = 10000


def game(size):
    out = np.empty(size)
    for i in range(size):
        r = n = 1
        while True:
            x, y = np.random.uniform(-1, 1, (2,))
            if x ** 2 + y ** 2 > r ** 2:
                break
            else:
                r = np.sqrt(r ** 2 - x ** 2 - y ** 2)
                n += 1
        out[i] = n

    return out


n = game(10000)
print(f"Estimate: {n.mean():.4f}  Theoretical: {np.e ** (np.pi / 4):.4f}")
```

```
Estimate: 2.2005  Theoretical: 2.1933
```

Problem 4

Use the inverse transform method to to estimate the mean and variance of the following random variables:

Part a

$X = \min\{X_1, X_2, X_3\}$ where X_1, X_2 and X_3 are i.i.d. exponentially distributed random variables with parameter $\lambda = 2$.

$$X_i \overset{iid}{\sim} \text{Exp}(2) \quad \text{for } i = 1, 2, 3$$

$$X = \min\{X_1, X_2, X_3\}$$

$$
\begin{aligned}
F_X(x) &= P[X \le x] \\
&= P[\min\{X_1, X_2, X_3\} \le x] \\
&= 1 - P[X_1 > x, X_2 > x, X_3 > x] \\
&= 1 - P[X_1 > x] \cdot P[X_2 > x] \cdot P[X_3 > x] \\
&= 1 - (1 - P[X_i \le x])^3 \\
&= 1 - (1 - F_{X_i}(x))^3 \\
&= 1 - (1 - (1 - e^{-\lambda x}))^3 \\
&= 1 - e^{-3\lambda x}
\end{aligned}
$$

$$
\begin{aligned}
u &= 1 - e^{-3\lambda x} \\
e^{-3\lambda x} &= 1 - u \\
-3\lambda x &= \ln(1-u) \\
x &= -\frac{\ln(1-u)}{3\lambda}
\end{aligned}
$$

$$
X = -\frac{\ln(1-U)}{3\lambda} = -\frac{\ln(U)}{3\lambda} \qquad U \sim \text{Unif}(0,1)
$$

```
size = 100000
lamda = 2

u = np.random.random(size)
x = - np.log(u) / (3 * lamda)
print(f"Mean: {x.mean():.4f}, Var: {x.var():.4f}")
```

```
Mean: 0.1667, Var: 0.0283
```

Part b

Y with the pdf

$$
f_Y(y) = \begin{cases} \frac{3}{8} y^2 e^{-y^3/8} & y > 0 \\ 0 & \text{otherwise} \end{cases}
$$

$$\begin{aligned}\int_0^y f_Y(t)dt &= \int_0^y \frac{3}{8}t^2 e^{-t^3/8}dy \\ &= -e^{-t^3/8}\Big|_0^y \\ &= -e^{-y^3/8} - (-1) \\ &= 1 - e^{-y^3/8}\end{aligned}$$

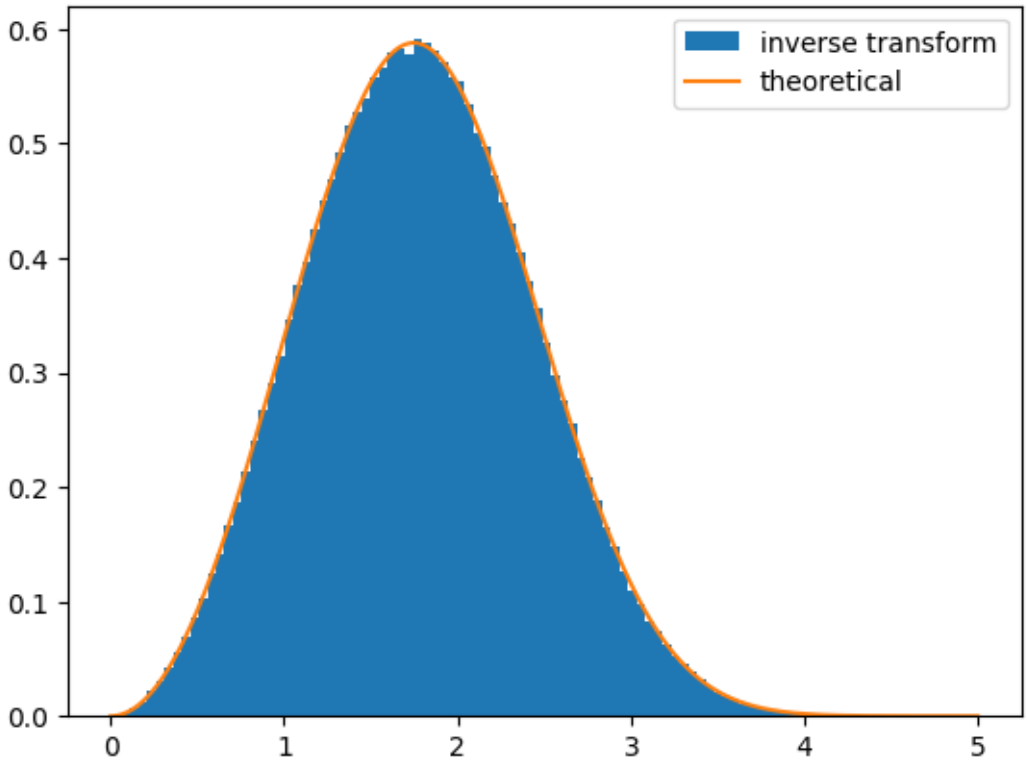
$$\begin{aligned}u &= 1 - e^{-y^3/8} \\ e^{-y^3/8} &= 1 - u \\ -y^3/8 &= \ln(1-u) \\ y^3 &= -8\ln(1-u) \\ y &= (-8\ln(1-u))^{1/3}\end{aligned}$$

$$Y = (-8\ln(1-U))^{1/3} = (-8\ln(U))^{1/3} \qquad U \sim \text{Unif}(0,\,1)$$

```
size = 1000000

u = np.random.random(size)
y = (-8 * np.log(u)) ** (1/3)

print(f"Mean: {y.mean():.4f}, Var: {y.var():.4f}")
```

```
Mean: 1.7868, Var: 0.4209
```

```
import seaborn as sns
supp = np.linspace(0, 5, 1000)
f = 3 / 8 * supp ** 2 * np.exp(-(supp ** 3) / 8)
plt.hist(y, bins=100, label='inverse transform', density=True)
plt.plot(supp, f, label='theoretical')
plt.legend()
plt.show()
```



Problem 5

Use the acceptance-rejection method to generate a random variable X with the pdf

$$f_X(x) = \begin{cases} \frac{3}{16}(3 + x - x^2) & 0 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

You can use a bounding rectangle (which amounts to using a uniform density $g(x)$), but you should choose the rectangle to be as small as possible.

Use your simulation code to generate at least $10,000$ samples. Estimate the expected value of X and compute the expected number of trials needed per accepted sample.

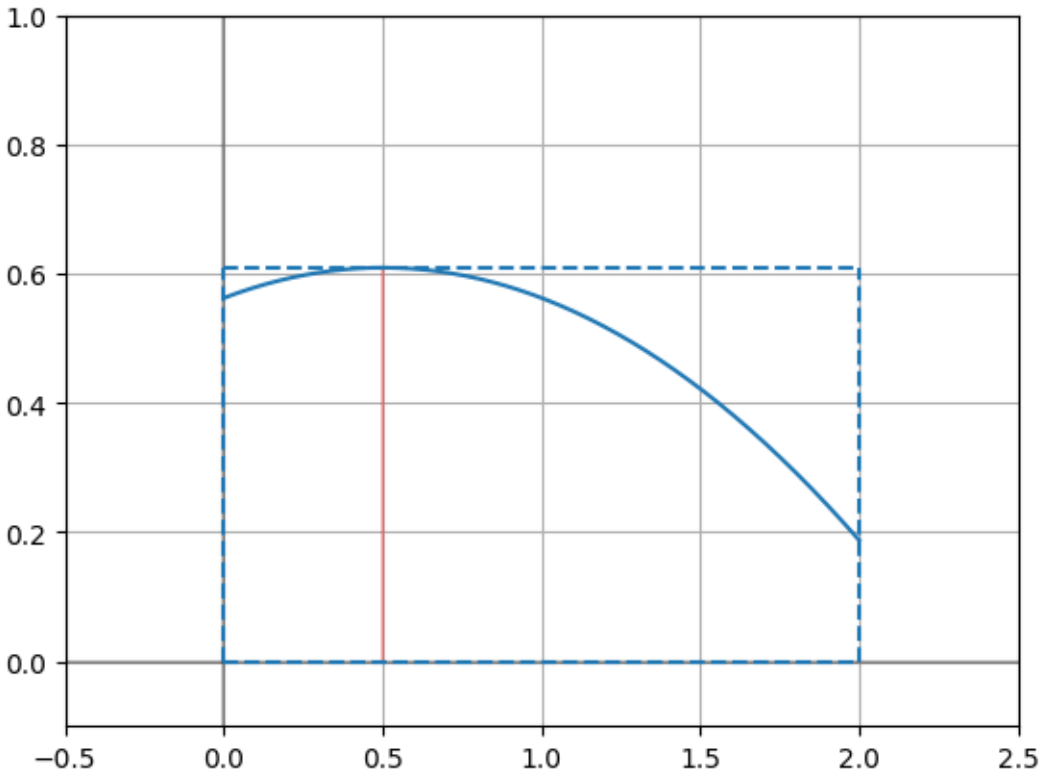
$$f_X(x) = \frac{3}{16}(3 + x - x^2)$$
$$f'_X(x) = \frac{3}{16}(1 - 2x^*) = 0$$
$$x^* = \frac{1}{2}$$

That is,

$$\arg\max_x f_X = \frac{1}{2} \max_x f_X = \frac{39}{64}$$

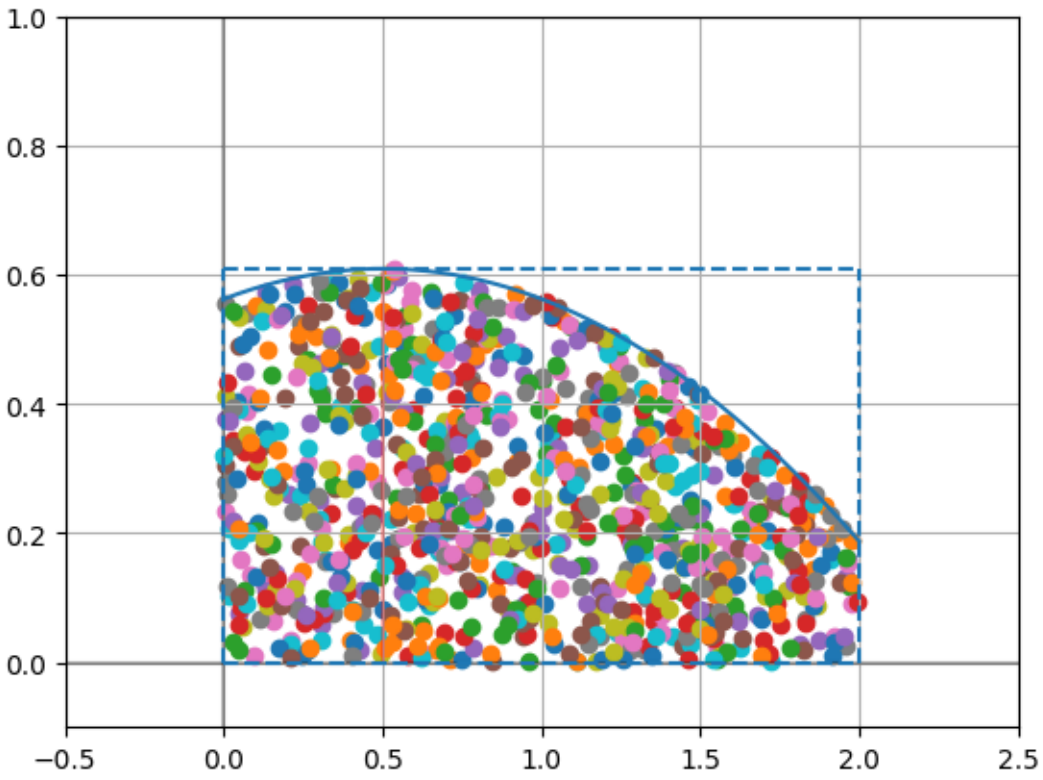
Thus, we set the height of the bounding rectangle to 39/64

```
supp = np.linspace(0, 2, 1000)


def f(x):
    return 3/16 * (3 + x - x ** 2)


plt.plot(supp, f(supp))
fmax = 3/16 * (3 + 1/2 - 1/2 ** 2)
plt.vlines(1/2, alpha=.3, color='r', ymin=0, ymax=fmax)
plt.axhline(0, alpha=.3, color='black')
plt.axvline(0, alpha=.3, color='black')
plt.hlines([0, fmax], xmin=0, xmax=2, ls='--')
plt.vlines([0, 2], ymin=0, ymax=fmax, ls='--')
plt.xlim(-.5, 2.5)
plt.ylim(-.1, 1)
plt.grid()
```




```
def generate_point():
    trial = 1
    while True:
        x = np.random.uniform(0, 2)
        y = np.random.uniform(0, fmax)
        if y <= f(x):
            return x, y, trial
        trial += 1


plt.plot(supp, f(supp))
plt.vlines(1/2, alpha=.3, color='r', ymin=0, ymax=fmax)
plt.axhline(0, alpha=.3, color='black')
plt.axvline(0, alpha=.3, color='black')
plt.hlines([0, fmax], xmin=0, xmax=2, ls='--')
plt.vlines([0, 2], ymin=0, ymax=fmax, ls='--')
plt.xlim(-.5, 2.5)
plt.ylim(-.1, 1)
```

```
plt.grid()

fmax = 3/16 * (3 + 1/2 - 1/2 ** 2)
size = 100000
num_trials = np.empty(size)
samples = np.empty(size)
ax = plt.gca()

for i in range(1000):
    x, y, trial = generate_point()
    num_trials[i] = trial
    samples[i] = x
    ax.scatter(x, y)

for i in range(size):
    x, y, trial = generate_point()
    num_trials[i] = trial
    samples[i] = x
```



```
print(f"E[X] = {samples.mean():.4f}, E[#(trials)]: {num_trials.mean():.4f}")
```

```
E[X] = 0.8748, E[#(trials)]: 1.2209
```

$$
\begin{aligned}
E[X] &= \int_0^2 x f_X(x) dx \\
&= \frac{3}{16} \int_0^2 3x + x^2 - x^3 dx \\
&= \frac{3}{16} \left[\frac{3}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 \right]_0^2 \\
&= \frac{3}{16} \left[6 + \frac{8}{3} - 4 \right] \\
&= \frac{7}{8} = 0.875
\end{aligned}
$$