Robby Kevlishvili **Problem Set 2, Due: Mar. 18, 2025**

All the Python codes will be added on the last page, in the Appendix.

(1) Use the probability integral transformation method to simulate from the distribution

$$f(x) = \begin{cases} \frac{2}{a^2}x, & \text{if } 0 \leq x \leq a \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $a > 0$. Set a value for a, simulate various sample sizes, and compare results to the true distribution.

Solution:

1. The function F(X) is

$$(1/a^2)x^2 \; for \; 0 \leq x \leq a \tag{2}$$

Drawing $u \sim U(0,1)$

$$x = a\sqrt{u} \sim f(x) \tag{3}$$

Plotting the graphs:

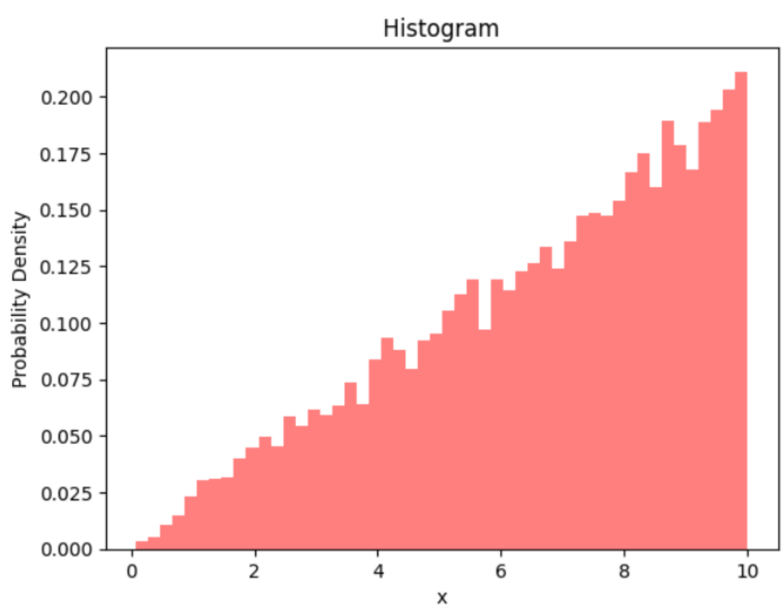


Figure 1: Distribution with a = 10, n = 10,000

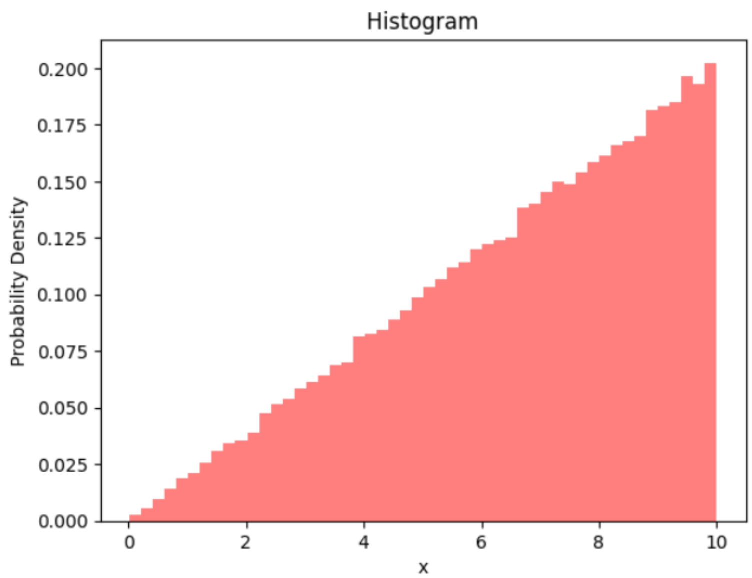


Figure 2: Distribution with a = 10, n = 100,000

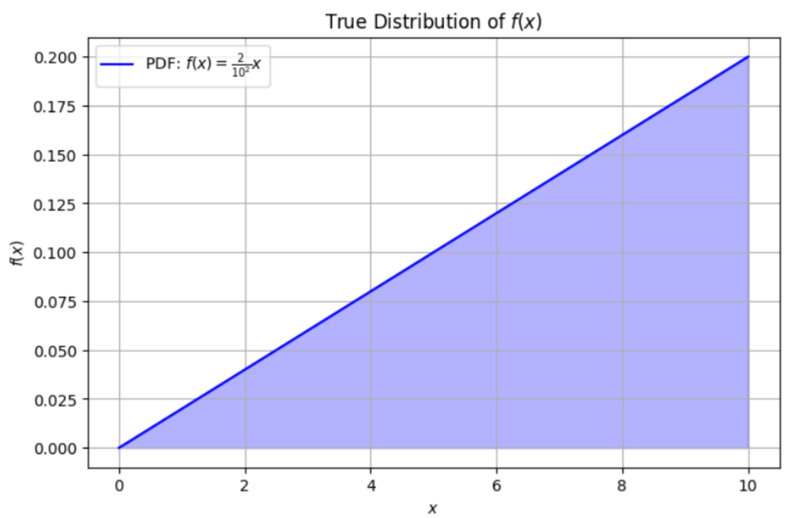


Figure 3: True Distribution of the PDF

We see that as n increases, the distribution becomes a lot more 'smooth', and more closely resembles the true distribution of the PDF.

(2) Generate samples from the distribution

$$f(x) = \frac{2}{3}e^{-2x} + 2e^{-3x} \tag{4}$$

using the finite mixture approach.

Solution:

1. The PDF of f(x) can be written as

$$f(x) = \frac{1}{3}2e^{-2x} + \frac{2}{3}3e^{-3x} \tag{5}$$

The CDF of an exponential distribution with rate λ is

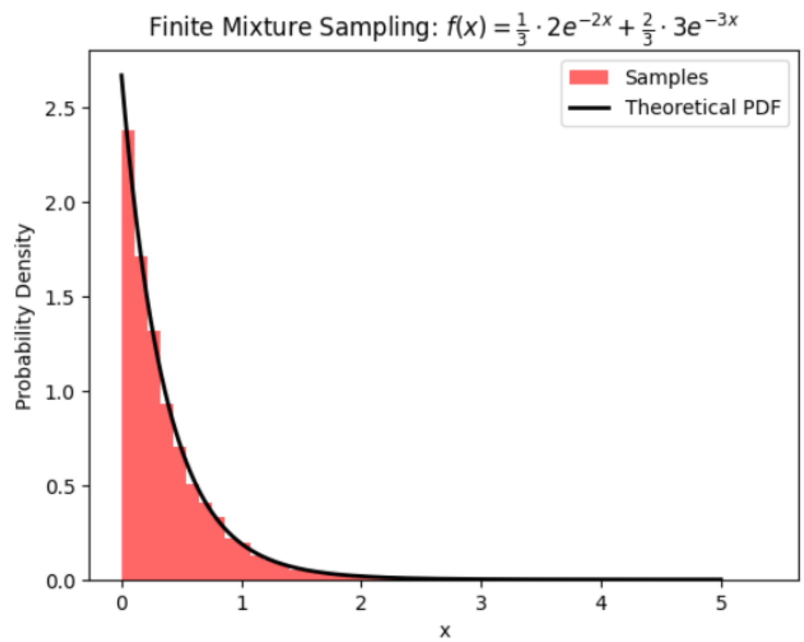
$$F(x) = 1 - e^{-\lambda x} \tag{6}$$

The inverse of the CDF for an exponential distribution with rate λ is

$$F^{-1}(u) = -\frac{1}{\lambda}ln(1-u), with\ u\ \epsilon\ (0,1) \tag{7}$$

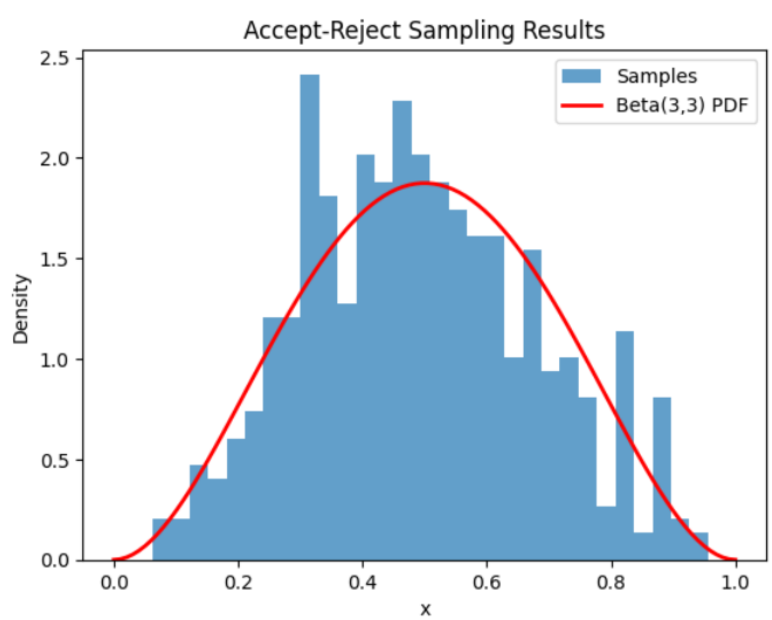
Steps:
Generate a random number u_1 from a uniform distribution between 0 and 1
If $u_1 < \frac{1}{3}$, select the first component $(\lambda_1 = 2)$. If $u_1 > \frac{2}{3}$, select the second component $(\lambda_1 = 3)$
If the first component was selected, generate a random number u_2 from a uniform distribution between 0 and 1. Calculate $x = -\frac{1}{2}ln(1-u_2)$.
If the second component was selected, generate a random number u_2 from a uniform distribution between 0 and 1. Calculate $x = -\frac{1}{3}ln(1-u_2)$.

Plotting the graph:



(3) Draw 500 observations from Beta$(3, 3)$ using the accept-reject algorithm. Compute the mean and variance of the sample and compare them to the true values. $\rightarrow$
Calculated values included in graph. Overall close resemblance.

```
Sample Mean: 0.4929 | True Mean: 0.5000
Sample Variance: 0.0350 | True Variance: 0.0357
```



Appendix

Python Code for Q1:

import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

Generate random numbers

a = 2
u = np. random . rand (10000)
x = a * u **(1/2)

Plot

plt . hist (x, bins =50 , density =True , color =”red ”, alpha =0.5)
plt.xlabel (”x”)
plt.ylabel (” Probability Density ”)
plt.title (” Histogram ”)
plt.show ()

Python Code for Q2:

import numpy as np
import matplotlib.pyplot as plt

Parameters $n_samples = 10000 Numberofsamples$
$weights = [1/3, 2/3] Mixtureweightsforcomponents$
$(= 2and = 3)$
$lambdas = [2, 3] Rateparametersfortheexponentialcomponents$

Step 1: Choose components based on weights
u1 = np.random.rand($n_samples$)
$component_choice = np.where(u1 < weights[0], 0, 1) 0for = 2, 1for = 3$

Step 2: Sample from the selected exponential components
x = np.zeros($n_samples$)
$foriinrange(len(lambdas)):$
$mask = (component_choice == i)$
$n_selected = np.sum(mask)$
$Inversetransformmethod : X = -ln(U)/$
$x[mask] = -np.log(np.random.rand(n_selected))/lambdas[i]$

Plot histogram vs theoretical PDF
plt.hist(x, bins=50, density=True, alpha=0.6, color="red", label="Samples")

Overlay theoretical PDF
x$_g rid = np.linspace(0, 5, 500)$
$theoretical_pdf = (weights[0] * 2 * np.exp(-2 * x_g rid)) + (weights[1] * 3 * np.exp(-3 * x_g rid))$
$plt.plot(x_g rid, theoretical_pdf, "k-", linewidth = 2, label = "TheoreticalPDF")$

plt.xlabel("x")
plt.ylabel("Probability Density")
plt.title("Finite Mixture Sampling: $f(x) =$
$frac13 \cdot 2e^{-2x} +$
$frac23 \cdot 3e^{-3x}$")
plt.legend()
plt.show()

Python code for Q3:
def target (x) :
return stats.beta.pdf(x, a=3 , b =3)
def proposal (x) :
return stats.uniform.pdf(x)
def accept $_reject(target, proposal, c, n)$:
$sample = []$
$whilelen(sample) < n$:
$x = np.random.uniform(0, 1)$
$u = np.random.uniform(0, 1)$
$ifu <= target(x)/(proposal(x) * c)$:
$sample.append(x)$
$returnnp.array(sample)$
$Samplebeta$
$c = target(0.5)/proposal(0.5)$
$sample = accept_reject(target, proposal, c, 10000)$