ORIE 4580/5580/5581 Assignment 3

Students: Kahei Lam(kl2235) and J. von Neuman (jvn001)

Github link: https://github.com/Althealam/ORIE-5580-Simulation-Modeling-Analysis/blob/main/HW3/ORIE_5580_hw3.ipynb

(**Please replace this with your own link!**)

Instructions

- Due Thursday September 25, at 11.59pm on Gradescope.
- Assignment .ipynb files available for download on Canvas.
- Do all your work in provided notebook (text answers typeset in markdown; show all required code and generate plots inline), and then generate and submit a pdf.
- Ideally do assignments in groups of 2, and submit a single pdf with both names
- Please show your work and clearly mark your answers.
- You can use any code fragments given in class, found online (for example, on StackOverflow), or generated via Gemini/Claude/ChatGPT (you are encouraged to use these for first drafts) **with proper referencing**.
- You can also discuss with others (again, please reference them if you do so); but you must write your final answers on your own as a team.

Suggested reading

Chapters 7 (you can skim through this), and chapters 8 and 9 of Introduction to Probability by Grinstead and Snell.

Chapter 3 and chapter 4 (up to section 4.5) of Simulation by Ross.

In [ ]:
```
#importing necessary packages
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
%matplotlib inline
```

Question 1: Combining LCGs (20 points)

In order to avoid biases, simulations should not use anywhere near the full period of an LCG (otherwise, the random sequence repeats\ldots). For example, a typical traffic simulator may have $10,000$ vehicles, each experiencing thousands of random disturbances, thus needing around 10^7 random samples per replication -- for this, an LCG using $m = 2^{31} - 1 \approx 2 \times 10^9$ is insufficient, as after 100 replications the sequences get correlated.

One method to combine multiple LCGs to obtain a generator with a longer period is to add a smaller period LCG to it. For example, suppose we have two generators $X_{n+1} = (a_1 X_n) \bmod m_1$ and $Y_{n+1} = (a_2 Y_n) \bmod m_2$, with $m_1 > m_2$. We can derive a combined generator by setting $Z_n = (X_n + Y_n) \bmod m_1$. If properly designed, the resulting period can be on the order of $m_1 m_2$. We will now study a small example to see how this works.

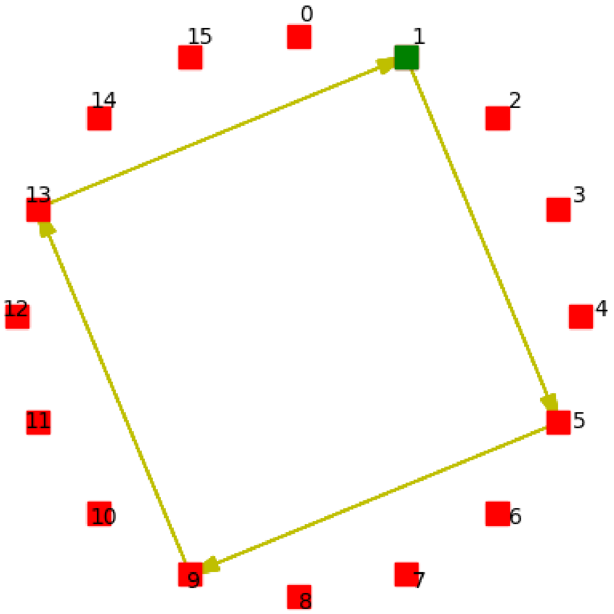
(a) Consider two LCGs, $x_{n+1} = (5x_n) \mod 16$ and $y_{n+1} = (2y_n) \mod 7$. Starting both with seed $x_0 = y_0 = 1$, plot the sequences x_n, y_n using the clock visualization introduced in class (separate plot for each sequence; you can use and modify the code in Demo-PRNGs.ipynb on Canvas).

In [ ]:
```
# Functions to visualize LCG sequence on clock (see demo notebook)
def plot_clock_face(m, fig, annotate=False):
    """
    Plot points on a unit circle representing the LCG sequence on a clock face.
    Parameters:
    m (int): The modulus value for the LCG sequence.
    fig (matplotlib.figure.Figure): The figure object to draw on.
    annotate (bool): Whether to annotate points with their index.

    Returns:
    None
    """
    # Plot m points on the unit circle
    for i in range(m):
        theta = 2.0 * np.pi * i / m
        plt.plot(np.sin(theta), np.cos(theta), 'rs', markersize = 10)
        if annotate:
            plt.annotate(str(i), (np.pi/2 - theta, 1.05), xycoords='polar')

def plot_clock_path(m, x, fig, color='y'):
    """
    Plot the path of an LCG sequence on a clock face.
    Parameters:
    m (int): The modulus value for the LCG sequence.
    x (numpy.ndarray): The LCG sequence.
    fig (matplotlib.figure.Figure): The figure object to draw on.
    color (str): The color for the path.

    Returns:
    None
    """
    # Plot the seed node
    theta_0 = 2.0 * np.pi * (x[0] * (m + 1) - 1) / m
    plt.plot(np.sin(theta_0), np.cos(theta_0), 'gs', markersize = 10)

    # Plot the path of the LCG sequence
    for i in range(len(x) - 1):
        theta_start = 2.0 * np.pi * (x[i] * (m + 1) - 1) / m
        theta_end = 2.0 * np.pi * (x[i + 1] * (m + 1) - 1) / m

        x_start = np.sin(theta_start)
        y_start = np.cos(theta_start)
        del_x = np.sin(theta_end) - np.sin(theta_start)
        del_y = np.cos(theta_end) - np.cos(theta_start)

        if abs(del_x) > 0 or abs(del_y) > 0:
            plt.arrow(x_start, y_start, del_x, del_y,
                      length_includes_head=True, head_width=0.05, head_length=0.1, fc=color, ec
```

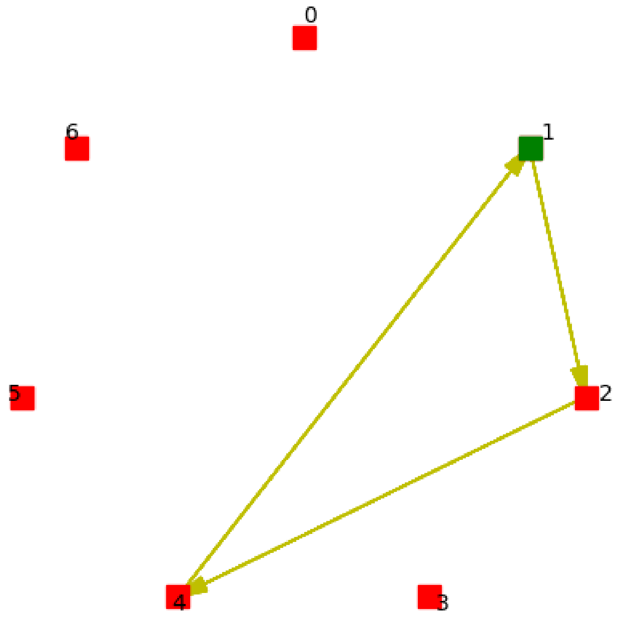
In [ ]:
```
# Function to generate pseudorandom sequence using LCG
# Set default parameters to glibc specifications (see demo notebook)

def LCG(n, m=2**31-1, a=1103515245, c=12345, seed=1):
    """
    Generate a pseudorandom sequence using a Linear Congruential Generator (LCG).

    Parameters:
    n (int): The number of pseudorandom numbers to generate.
    m (int): The modulus value (default is 2^31-1, following glibc specifications).
    a (int): The multiplier value (default is 1103515245, following glibc specifications).
    c (int): The increment value (default is 12345, following glibc specifications).
    seed (int): The initial seed value (default is 1).

    Returns:
    numpy.ndarray: An array of pseudorandom numbers in the range [0, 1).
    """
    # Initialize an array to store the generated pseudorandom numbers
    output = np.zeros(n)

    x = seed
    for i in range(n):
        # Calculate the pseudorandom number and normalize it to [0, 1)
        output[i] = (x + 1.0) / (m + 1.0)
```

```
        # Update the LCG state using the specified parameters
        x = (a * x + c) % m

    return output
```

In [ ]:
```
# Ans
n = 20 # number of samples

## 1. X(n+1) = (5*X(n)) mod 16
m1 = 16
a1 = 5
c1 = 0
seed1 = 1
fig = plt.figure(figsize = (5, 5))
x_seq = LCG(n=n, m=m1, a=a1, c=c1, seed=seed1)

plot_clock_face(m1, fig, annotate=True)
plot_clock_path(m1, x_seq, fig)
plt.axis('off')
plt.show()
```

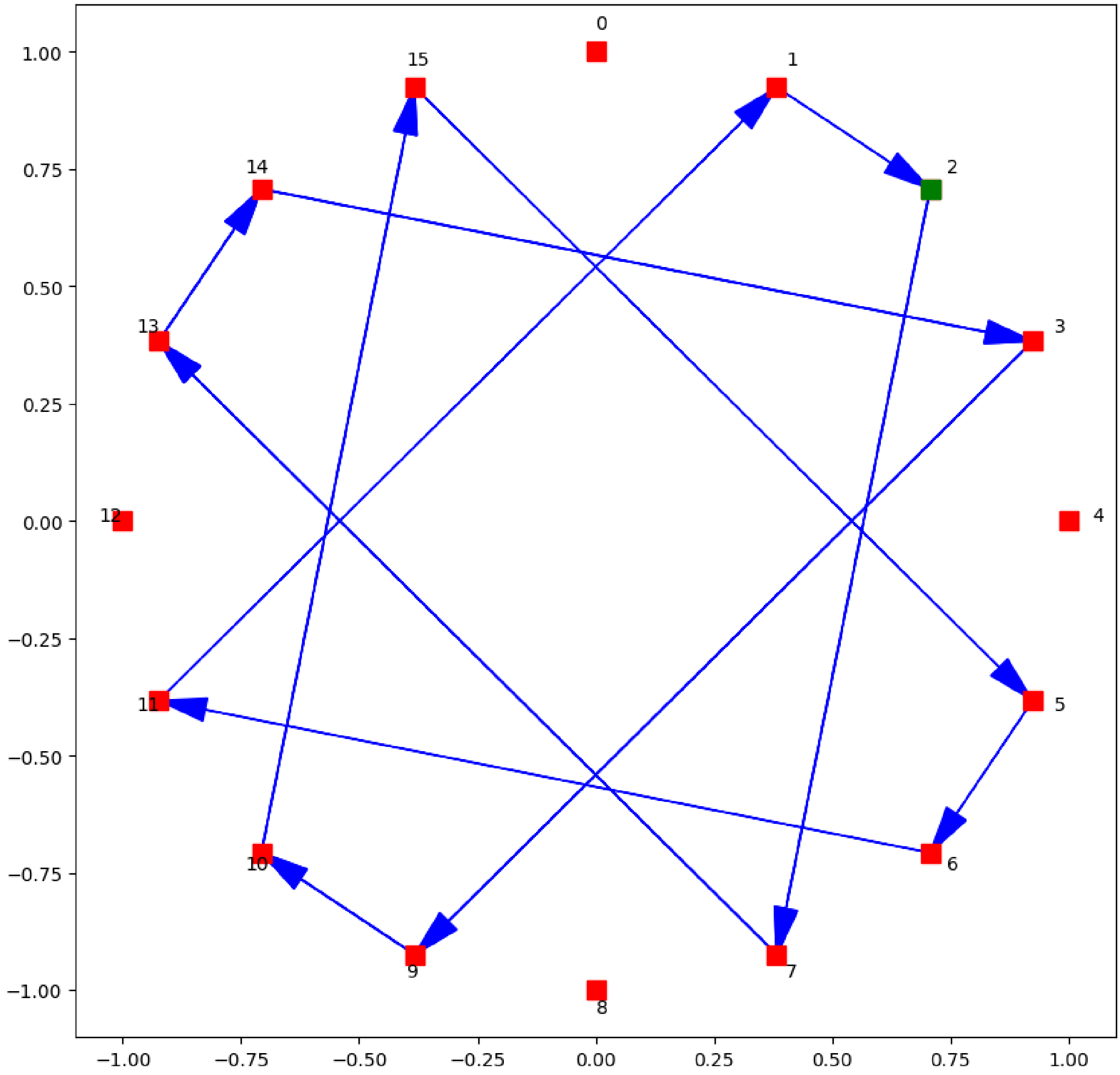


In [ ]:
```
# Ans
## 2. Y(n+1) = (2*Y(n)) mod 7
m2 = 7
a2 = 2
c2 = 0
seed2 = 1
fig = plt.figure(figsize = (5, 5))
y_seq = LCG(n=n, m=m2, a=a2, c=c2, seed=seed2)

plot_clock_face(m2, fig, annotate=True)
plot_clock_path(m2, y_seq, fig)
plt.axis('off')
plt.show()
```



(b) Next, define a combined LCG as $z_n = (x_n + y_n) \mod 16$. Starting both the base LCGs with seed $x_0 = y_0 = 1$, plot the sequence z_n using the clock visualization given in class.

In [ ]:
```
x_prime = np.round(x_seq*(m1+1)-1).astype(int)
y_prime = np.round(y_seq*(m2+1)-1).astype(int)

z_prime = (x_prime+y_prime) % 16
z_seq = (z_prime+1.0)/(16+1.0)

fig = plt.figure(figsize=(10, 10))
plot_clock_face(m=16, fig=fig, annotate= True)
plot_clock_path(m=16, x=z_seq, fig=fig, color='b')
```



(c) What are the periods of the pseudo-random sequences x_n, y_n and z_n?

Ans.

1. The period of x_n

$$x_0 = 1$$

$$x_1 = (5 \times 1) mod 16 = 5$$

$$x_2 = (5 \times 5) mod 16 = 9$$

$$x_3 = 13$$

$$x_4 = 1$$

We can know that x_4=x_0=1, so the sequence start repeating and the period of x_n is 4

2. The period of y_n

$$y_0 = 1$$

$$y_1 = (2 \times 1) mod 7 = 2$$

$$y_2 = 4$$

$$y_3 = 1$$

The period of y_n is 3

3. The period of z_n

We know that the period of x_n is 4 and the period of y_n is 3. So the period of z_n is $lcm(4, 3) = 12$

Question 2: inverting cdfs (25 pts)

In class, we defined $F^{-1}(y)$ for a continuous increasing cdf $F(x)$ as the unique x such that $F(x) = y$ (for $y \in [0, 1]$). More generally, for any cdf F we can use the inversion method based on its generalized inverse or *pseudoinverse*:

$$F^{-1}(y) = \inf\{x|F(x) \geq y\}$$

(where $\inf$ denotes the \href{https://en.wikipedia.org/wiki/Infimum_and_supremum}{infimum}; if you have not seen this before, treat it as minimum).

(a) Find the pseudoinverse $F^{-1}(y)$ for the following mixed (discrete/continuous) cdf

$$F(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } 0 \leq x < \frac{1}{2}, \\ \frac{1}{2} & \text{for } \frac{1}{2} \leq x < 1, \\ 1 & \text{for } x \geq 1 \end{cases}$$

Ans.

Case 1: $y \in t(0, \frac{1}{2}]$

We need the smallest x with $F(x) \geq y$.

- For $x < 0$: $F(x) = 0$. Since $y \geq 0$, $0 \geq y$ only holds when $y = 0$; for $y \in (0, \frac{1}{2})$, $0 < y$, so $x < 0$ fails.
- For $0 \leq x < \frac{1}{2}$: $F(x) = x$. Solve $x \geq y$. The smallest x is y (as $F(x)$ increases here, and $y < \frac{1}{2}$ keeps $x = y$ in $[0, \frac{1}{2})$).

Thus, $F^{-1}(y) = y$ for $y \in [0, \frac{1}{2})$.

Case 2: $y = \frac{1}{2}$

We need the smallest x with $F(x) \geq \frac{1}{2}$.

- For $x < \frac{1}{2}$: $F(x) < \frac{1}{2}$, so no solution.
- For $x \geq \frac{1}{2}$: $F(x) \geq \frac{1}{2}$ (since $F(x) = \frac{1}{2}$ when $\frac{1}{2} \leq x < 1$, and $F(x) = 1$ when $x \geq 1$). The smallest x is $\frac{1}{2}$.

Thus, $F^{-1}(\frac{1}{2}) = \frac{1}{2}$.

Case 3: $y \in \left(\frac{1}{2}, 1\right]$

We need the smallest x with $F(x) \geq y$.

- For $x < 1$: $F(x) \leq \frac{1}{2} < y$, so no solution.
- For $x \geq 1$: $F(x) = 1 \geq y$ (since $y \leq 1$). The smallest x is 1.

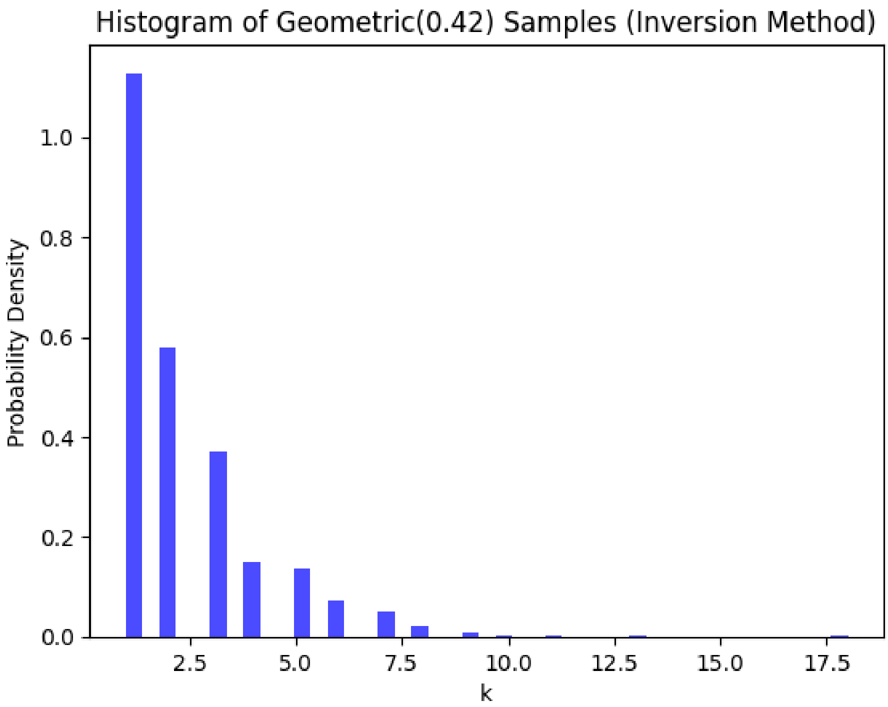
Thus, $F^{-1}(y) = 1$ for $y \in \left(\frac{1}{2}, 1\right]$.

Final Pseudoinverse

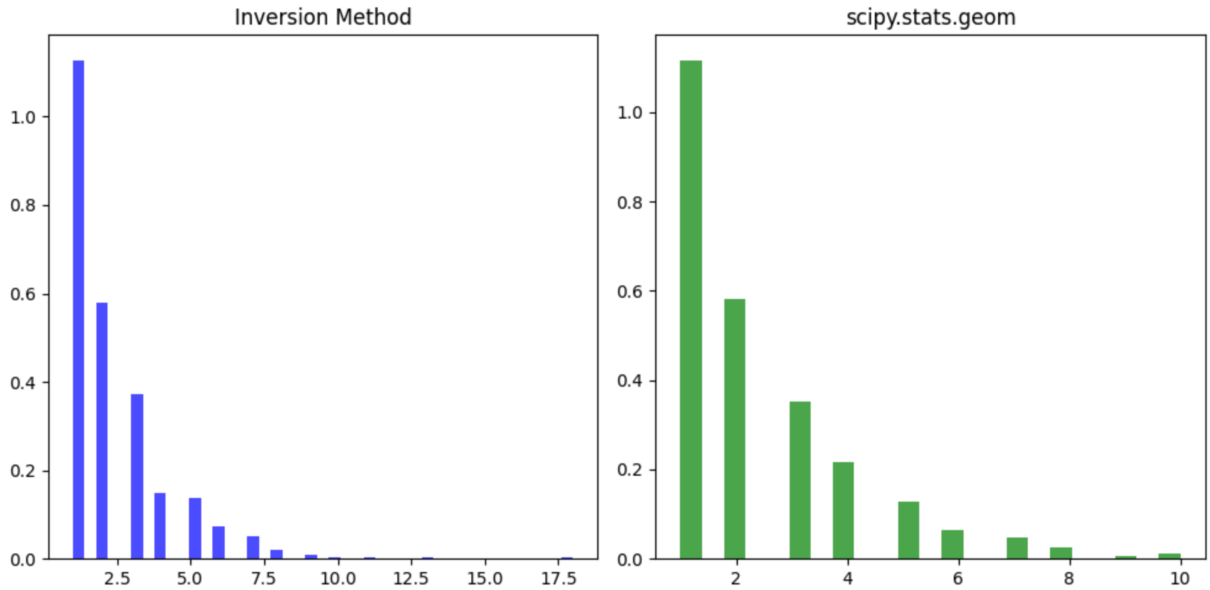
$$F^{-1}(y) = \begin{cases} y & \text{if } y \in [0, \frac{1}{2}), \\ \frac{1}{2} & \text{if } y = \frac{1}{2}, \\ 1 & \text{if } y \in \left(\frac{1}{2}, 1\right]. \end{cases}$$

(b) Use the above definition to get an inversion algorithm for the $Geometric(p)$ distribution (with pmf $p(k) = p(1-p)^{k-1} \, \forall \, k \in \{1, 2, 3, \ldots\}$). Implement this, and generate and plot the histogram of 1000 samples from a $Geometric(0.42)$ distribution. (For this, it may be useful for you to first understand how the scipy.stats library works, and in particular, how it provides methods to compute various statistics for many different random variables, including the geometric r.v.)

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# implement the inversion algorithm
def geometric_inversion(n, p):
    """
    Generate n samples from Geometric(p) using the inversion method.
    """
    # Generate uniform random variables
    u = np.random.uniform(0, 1, n)
    # Apply the inverse CDF
    samples = np.ceil(np.log(1 - u) / np.log(1 - p)).astype(int)
    return samples

# Parameters
n = 1000  # Number of samples
p = 0.42  # Success probability

# Generate samples
samples = geometric_inversion(n, p)

# Plot histogram
plt.hist(samples, bins='auto', density=True, alpha=0.7, color='blue')
plt.title('Histogram of Geometric(0.42) Samples (Inversion Method)')
plt.xlabel('k')
plt.ylabel('Probability Density')
plt.show()
```



In [ ]:
```
# verification with scipy.stats
from scipy.stats import geom

# Generate samples using scipy
scipy_samples = geom.rvs(p, size=n)

# Plot both histograms
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.hist(samples, bins='auto', density=True, alpha=0.7, color='blue')
plt.title('Inversion Method')

plt.subplot(1, 2, 2)
plt.hist(scipy_samples, bins='auto', density=True, alpha=0.7, color='green')
plt.title('scipy.stats.geom')

plt.tight_layout()
plt.show()
```



(c) The p.d.f. of the random variable X is given by

$$f(x) = \begin{cases} e^{x-2} & \text{for } 0 \le x \le 2, \\ e^{-x} & \text{for } x > 2, \\ 0 & \text{otherwise,} \end{cases}$$

Describe and implement an inversion algorithm to generate samples of X. Generate 1,000 samples and plot a histogram. Compare the histogram and the p.d.f.

Ans.

1. Piecewise Probability Density Function (PDF)

$$f(x) = \begin{cases} e^{x-2} & \text{if } 0 \le x \le 2, \\ e^{-x} & \text{if } x > 2, \\ 0 & \text{otherwise.} \end{cases}$$

2. Derivation of Cumulative Distribution Function (CDF)

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

- For $x < 0$:

$$F(x) = 0$$

- For $0 \leq x \leq 2$:

$$F(x) = \int_0^x e^{t-2} dt = \left[e^{t-2}\right]_0^x = e^{x-2} - e^{-2}$$

- For $x > 2$:

$$F(x) = \int_0^2 e^{t-2} dt + \int_2^x e^{-t} dt$$

$$= \left(1 - e^{-2}\right) + \left(e^{-2} - e^{-x}\right) = 1 - e^{-x}$$

3. Derivation of Inverse CDF ($F^{-1}(u)$)

$$F^{-1}(u) = \text{the value of } x \text{ satisfying } F(x) = u$$

- For $0 \leq u \leq 1 - e^{-2}$:

$$e^{x-2} - e^{-2} = u \implies x = \ln\left(u + e^{-2}\right) + 2$$

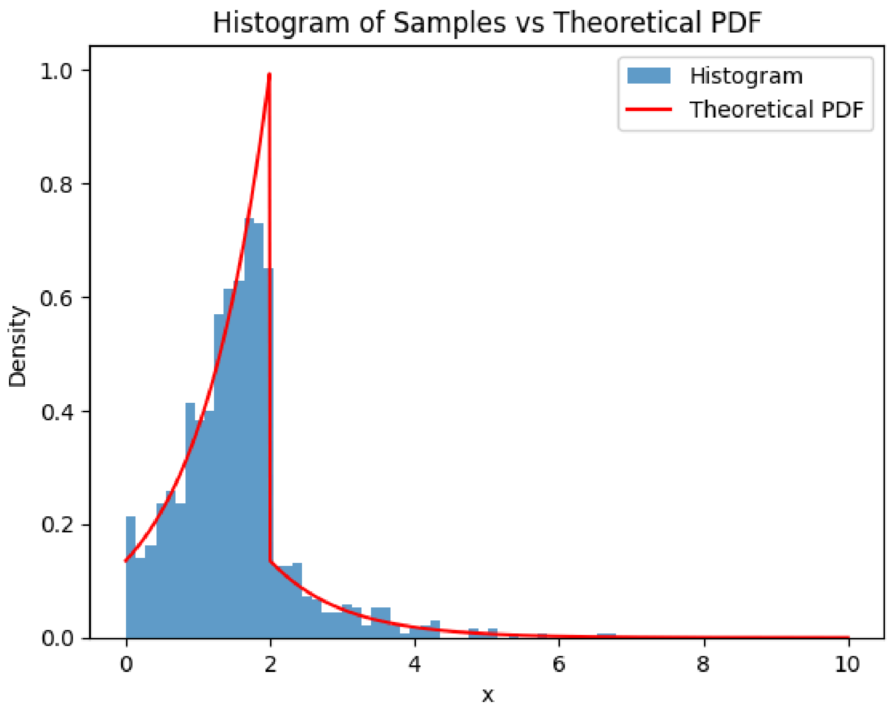
- For $u > 1 - e^{-2}$:

$$1 - e^{-x} = u \implies x = -\ln(1-u)$$

4. Final Inverse Transformation Formula

$$F^{-1}(u) = \begin{cases} \ln\left(u + e^{-2}\right) + 2 & \text{if } 0 \leq u \leq 1 - e^{-2}, \\ -\ln(1-u) & \text{if } u > 1 - e^{-2}. \end{cases}$$

In [ ]:
```python
# implement the inversion algorithm

import numpy as np
import matplotlib.pyplot as plt

def inverse_transform_sampling(n):
    """Generate n samples from the given PDF using inversion."""
    u = np.random.uniform(0, 1, n)
    x = np.zeros(n)

    # Case 1: 0 <= u <= 1 - exp(-2)
    mask = u <= (1 - np.exp(-2))
    x[mask] = np.log(u[mask] + np.exp(-2)) + 2

    # Case 2: u > 1 - exp(-2)
    x[~mask] = -np.log(1 - u[~mask])

    return x

# Generate 1000 samples
n = 1000
samples = inverse_transform_sampling(n)

# Plot histogram of samples
plt.hist(samples, bins=50, density=True, alpha=0.7, label='Histogram')

# Plot the theoretical PDF
x_vals = np.linspace(0, 10, 1000)
pdf = np.zeros_like(x_vals)
# PDF for 0 <= x <= 2
mask_pdf1 = (x_vals >= 0) & (x_vals <= 2)
pdf[mask_pdf1] = np.exp(x_vals[mask_pdf1] - 2)
# PDF for x > 2
mask_pdf2 = x_vals > 2
pdf[mask_pdf2] = np.exp(-x_vals[mask_pdf2])
plt.plot(x_vals, pdf, 'r-', label='Theoretical PDF')

plt.title('Histogram of Samples vs Theoretical PDF')
plt.xlabel('x')
plt.ylabel('Density')
plt.legend()
plt.show()
```



Question 3: Acceptance-Rejection (25 pts)

Let the random variable X have density

$$f(x) = \begin{cases} (5x^4 + 4x^3 + 3x^2 + 1)/4 & \text{for } 0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

(a) Give an acceptance-rejection algorithm to generate samples of X.

Ans.

In [1]:
```
import numpy as np


def f(x):
    return (5*x**4 + 4*x**3 + 3*x**2 + 1) / 4

c = 13/4 # c is maximum when plug in x=1 in to f(x)

def sample_X(n_samples=1):
    samples = []
    while len(samples) < n_samples:
        x = np.random.uniform(0, 1)
        u = np.random.uniform(0, 1)
        if u <= f(x)/c:
            samples.append(x)
    return np.array(samples)

samples = sample_X(1000)
samples
```

```
Out[1]: array([0.93034468, 0.93804381, 0.95432786, 0.67174348, 0.84747004,
       0.98148818, 0.77095777, 0.86520859, 0.69342324, 0.84604829,
       0.04316345, 0.75265934, 0.42628348, 0.28410728, 0.97476949,
       0.34387113, 0.76935239, 0.65647153, 0.94103698, 0.7717421 ,
       0.55263106, 0.61991537, 0.91550548, 0.87544287, 0.87692616,
       0.89806509, 0.61525441, 0.10041846, 0.97366584, 0.32386921,
       0.75456393, 0.18238689, 0.800179  , 0.98904061, 0.97967015,
       0.95662684, 0.93539742, 0.95602143, 0.94109531, 0.86011476,
       0.9408876 , 0.8651341 , 0.37337688, 0.95524927, 0.5381595 ,
       0.49505358, 0.98442766, 0.79344766, 0.18915559, 0.98046928,
       0.96055   , 0.37226172, 0.57568022, 0.52672544, 0.96277072,
       0.87921633, 0.2974513 , 0.27704008, 0.8292173 , 0.78019728,
       0.58214926, 0.96585545, 0.30541679, 0.73183723, 0.69037155,
       0.08909471, 0.70542355, 0.68763184, 0.45486204, 0.59002195,
       0.88482049, 0.45468343, 0.34530234, 0.83252325, 0.67235537,
       0.8039732 , 0.29245894, 0.8189732 , 0.64708076, 0.85544663,
       0.75716251, 0.9173797 , 0.58314731, 0.76497655, 0.41160383,
       0.86645888, 0.85535188, 0.47085069, 0.69056739, 0.68531151,
       0.99405529, 0.80970968, 0.8804866 , 0.65674092, 0.94228543,
       0.78608926, 0.22249578, 0.683876  , 0.52128995, 0.8958511 ,
       0.98911531, 0.45346007, 0.47415553, 0.57220844, 0.89857418,
       0.67438672, 0.80091525, 0.71306722, 0.74520401, 0.86453846,
       0.83128361, 0.87641048, 0.89716624, 0.80705969, 0.87078423,
       0.95168681, 0.92408116, 0.94274736, 0.330892  , 0.97425568,
       0.89308082, 0.66526671, 0.93529396, 0.72314901, 0.82137212,
       0.78740888, 0.84838684, 0.98607908, 0.81443094, 0.61918597,
       0.53313619, 0.81350435, 0.9378408 , 0.24230981, 0.71476634,
       0.92492915, 0.97772622, 0.63202169, 0.92999956, 0.82047391,
       0.77065207, 0.83854339, 0.79316732, 0.61754193, 0.60917316,
       0.94848468, 0.90135932, 0.79828275, 0.91806529, 0.22339576,
       0.63266967, 0.74297017, 0.8193859 , 0.78397382, 0.72582974,
       0.29673817, 0.8179492 , 0.75269588, 0.79121134, 0.92929198,
       0.9654778 , 0.93661297, 0.94370499, 0.58709534, 0.74817155,
       0.91562122, 0.64271669, 0.89714489, 0.79917169, 0.84369557,
       0.48219534, 0.45502132, 0.95292155, 0.22526414, 0.7992545 ,
       0.99528858, 0.92546569, 0.92161162, 0.93635003, 0.63820788,
       0.98653208, 0.94236883, 0.95707926, 0.76190402, 0.83718782,
       0.55365332, 0.75612203, 0.41314106, 0.86319837, 0.59143466,
       0.54816609, 0.73650846, 0.75311823, 0.77065348, 0.80624463,
       0.82390506, 0.96459658, 0.91850711, 0.42447219, 0.75591068,
       0.76397834, 0.87132347, 0.94158742, 0.81197581, 0.49600701,
       0.88745554, 0.59051936, 0.99961825, 0.79302088, 0.87609901,
       0.83233643, 0.98965105, 0.07481454, 0.70842601, 0.64159569,
       0.44748265, 0.11223682, 0.79916898, 0.70132612, 0.84981895,
       0.36602742, 0.9089699 , 0.28429853, 0.86584061, 0.99262729,
       0.98844959, 0.97311009, 0.41600583, 0.96690056, 0.76834583,
       0.54574956, 0.87980821, 0.12456904, 0.93258566, 0.74395869,
       0.41563037, 0.93580935, 0.95943655, 0.74008696, 0.57619983,
       0.96555723, 0.49111551, 0.99363587, 0.50997133, 0.94414557,
       0.76611838, 0.74073137, 0.72218025, 0.91870009, 0.96529516,
       0.75026403, 0.43475822, 0.58311243, 0.90905209, 0.81939602,
       0.82054215, 0.72508028, 0.89984478, 0.99526729, 0.65631766,
       0.505964  , 0.9936042 , 0.94835566, 0.4874044 , 0.97616701,
       0.01687331, 0.68636243, 0.85774073, 0.90673268, 0.57490913,
       0.88357697, 0.08886816, 0.807205  , 0.56301335, 0.23303378,
       0.47849232, 0.92110218, 0.64673909, 0.95963981, 0.86111044,
       0.93761306, 0.36379705, 0.70203019, 0.18367574, 0.41655765,
       0.47151288, 0.61586602, 0.40729446, 0.00254907, 0.33922633,
       0.89901172, 0.13697697, 0.58173091, 0.18269588, 0.59748353,
       0.9281389 , 0.62255695, 0.48998768, 0.5959483 , 0.4297742 ,
       0.60047906, 0.88483705, 0.3207245 , 0.93848763, 0.83163106,
       0.85151873, 0.95614016, 0.50642653, 0.66126971, 0.80940745,
       0.93285131, 0.46704074, 0.91019143, 0.72751714, 0.95483679,
       0.55377393, 0.16781508, 0.88637494, 0.40753061, 0.35586203,
       0.36944662, 0.96733494, 0.72169803, 0.95995449, 0.80900296,
       0.31471248, 0.9386607 , 0.69168735, 0.36862434, 0.53785397,
       0.50213617, 0.28416379, 0.80775178, 0.89628779, 0.90182628,
       0.97971172, 0.94401537, 0.72904662, 0.77438384, 0.52185496,
       0.88366254, 0.92834768, 0.99195796, 0.82089641, 0.60368007,
       0.28429832, 0.92177498, 0.68518625, 0.56984042, 0.90841397,
       0.45777813, 0.85115317, 0.98724683, 0.81048093, 0.47539342,
       0.79794558, 0.84480031, 0.72219092, 0.69965495, 0.91630158,
       0.27235545, 0.29565876, 0.68351564, 0.7942635 , 0.65279523,
       0.90269289, 0.95142205, 0.79925094, 0.4368868 , 0.7005285 ,
       0.73767859, 0.88341255, 0.94383269, 0.93603035, 0.25710106,
```

```
       0.86109725, 0.96453422, 0.9828212 , 0.47997105, 0.00902588,
       0.68616434, 0.90500024, 0.90118042, 0.42140221, 0.79831899,
       0.41582837, 0.96323907, 0.91760015, 0.52098926, 0.2478581 ,
       0.98863921, 0.25950176, 0.8009879 , 0.87110357, 0.59174505,
       0.83302608, 0.9203631 , 0.7922534 , 0.7354831 , 0.80126849,
       0.73404882, 0.99984293, 0.49136737, 0.99002847, 0.88617583,
       0.67039421, 0.41745857, 0.73347957, 0.00215456, 0.73322037,
       0.90165825, 0.55351001, 0.89304795, 0.82976175, 0.84466086,
       0.89260809, 0.57522116, 0.54859381, 0.62655834, 0.76678789,
       0.39651336, 0.71468516, 0.9179508 , 0.94954292, 0.93585785,
       0.94665307, 0.96495905, 0.37207362, 0.63721126, 0.89315604,
       0.92493618, 0.36897584, 0.94002069, 0.56051404, 0.80194132,
       0.97285944, 0.78156784, 0.83780177, 0.9112435 , 0.82007286,
       0.00300568, 0.03339187, 0.71819367, 0.66446295, 0.78889459,
       0.96293942, 0.97470227, 0.04505568, 0.59768933, 0.79239293,
       0.24062694, 0.99410459, 0.65518316, 0.54726463, 0.74289771,
       0.90459055, 0.74551085, 0.80598956, 0.85902908, 0.91532462,
       0.81772099, 0.84972501, 0.676689  , 0.76566857, 0.94950138,
       0.86458234, 0.94634434, 0.85132825, 0.74630097, 0.85304526,
       0.8270513 , 0.83228734, 0.76614029, 0.41294189, 0.51900554,
       0.73478588, 0.94829668, 0.79042147, 0.76676923, 0.83606543,
       0.69373761, 0.68881386, 0.38398977, 0.8199792 , 0.33637843,
       0.76199111, 0.25431509, 0.75496815, 0.45117714, 0.65887278,
       0.95788013, 0.92003939, 0.86255997, 0.9711657 , 0.8843077 ,
       0.57364895, 0.68404439, 0.98510364, 0.70214289, 0.91952427,
       0.98222485, 0.8193035 , 0.09063309, 0.98391771, 0.98877739,
       0.50246657, 0.72960304, 0.64317409, 0.63545051, 0.98046994,
       0.92816677, 0.87930078, 0.98288055, 0.86701025, 0.75834455,
       0.99302808, 0.70110821, 0.7450207 , 0.20059134, 0.67921483,
       0.74809245, 0.9579014 , 0.74834579, 0.65683777, 0.4996393 ,
       0.99107703, 0.64454191, 0.38649558, 0.9941947 , 0.99279643,
       0.87629249, 0.45780292, 0.98896962, 0.91818001, 0.82922422,
       0.84463528, 0.94677884, 0.94021752, 0.84687195, 0.40977701,
       0.96799425, 0.9848749 , 0.8929706 , 0.87651028, 0.18519777,
       0.97235583, 0.21009382, 0.64885763, 0.99731571, 0.80875126,
       0.88572211, 0.68596756, 0.92990863, 0.73900598, 0.99402742,
       0.74147845, 0.98986981, 0.70907981, 0.95811631, 0.87078558,
       0.80600749, 0.75828699, 0.98437883, 0.98427682, 0.72124462,
       0.84458957, 0.97529724, 0.99050283, 0.97650179, 0.68350716,
       0.90182488, 0.74302741, 0.94655752, 0.97238525, 0.74503521,
       0.60724807, 0.75786374, 0.97785494, 0.97258278, 0.82873462,
       0.97164465, 0.33483394, 0.92189858, 0.91690388, 0.98865735,
       0.6908014 , 0.88888068, 0.91405975, 0.53970159, 0.76864899,
       0.97996592, 0.91252066, 0.32961852, 0.72982467, 0.26362332,
       0.6966858 , 0.71986812, 0.96514413, 0.84398433, 0.53985958,
       0.42962185, 0.45279033, 0.90735306, 0.19550334, 0.95752902,
       0.7374677 , 0.57387867, 0.63783921, 0.67611202, 0.27232721,
       0.79185089, 0.00755357, 0.80329791, 0.78393029, 0.35792253,
       0.90363782, 0.85277536, 0.17714987, 0.96273157, 0.80399069,
       0.57728015, 0.58784522, 0.99885335, 0.68264287, 0.9483067 ,
       0.59626003, 0.41340602, 0.97234991, 0.9919077 , 0.81680039,
       0.97605256, 0.9198921 , 0.92292403, 0.9009042 , 0.64818392,
       0.07563707, 0.96373533, 0.76322745, 0.94215046, 0.68213138,
       0.57488626, 0.56539914, 0.62353263, 0.99189273, 0.67694447,
       0.77810276, 0.24615071, 0.9328779 , 0.97249454, 0.66858629,
       0.88889307, 0.88276342, 0.61941292, 0.16870184, 0.87856556,
       0.06354614, 0.48449583, 0.96621779, 0.75215869, 0.68569701,
       0.90407976, 0.91359069, 0.74922252, 0.70875037, 0.6741256 ,
       0.01758782, 0.02540126, 0.90154551, 0.64521844, 0.57611576,
       0.25214903, 0.55881552, 0.69688863, 0.78159657, 0.118443  ,
       0.74953159, 0.14006031, 0.68622915, 0.82107897, 0.13574138,
       0.46105451, 0.15881049, 0.80742618, 0.32741275, 0.54983639,
       0.71695359, 0.88790165, 0.90504194, 0.5967728 , 0.84983911,
       0.90154827, 0.91229251, 0.99735681, 0.60211376, 0.95597669,
       0.86106394, 0.7796457 , 0.90052338, 0.99237155, 0.99524875,
       0.86169941, 0.18428563, 0.9105058 , 0.42009182, 0.98406178,
       0.91982151, 0.89909653, 0.88475494, 0.98362803, 0.72287186,
       0.77680543, 0.9799218 , 0.923116  , 0.46860738, 0.83540949,
       0.07991936, 0.75940078, 0.8149098 , 0.79286907, 0.95570405,
       0.71026066, 0.29029796, 0.82069383, 0.6460035 , 0.85379129,
       0.98980069, 0.72175419, 0.82266208, 0.38780641, 0.96903186,
       0.81405058, 0.71595801, 0.8520814 , 0.63752892, 0.46014163,
       0.87028983, 0.79005796, 0.95016863, 0.87458414, 0.84988852,
       0.34520934, 0.52206521, 0.5114635 , 0.46250364, 0.92326969,
       0.69568345, 0.92801354, 0.98358801, 0.84288067, 0.94896854,
```

```
0.90252845, 0.89862482, 0.45865609, 0.88670322, 0.91365715,
0.38800304, 0.82298227, 0.81896068, 0.82876786, 0.30991864,
0.97048952, 0.35311489, 0.94305341, 0.93762957, 0.9916878 ,
0.92651042, 0.65744037, 0.93300388, 0.74874851, 0.73585149,
0.37648818, 0.99488724, 0.77591772, 0.99359856, 0.18616606,
0.94313544, 0.93285228, 0.91033135, 0.23448464, 0.81998022,
0.64141143, 0.98302601, 0.78867623, 0.78496007, 0.97319891,
0.82037177, 0.708271  , 0.89161627, 0.9115576 , 0.95808718,
0.84292276, 0.91648467, 0.81084142, 0.66863543, 0.67733097,
0.84905839, 0.9892423 , 0.95480508, 0.55891125, 0.76961609,
0.89388976, 0.33977875, 0.87359686, 0.83136787, 0.61376654,
0.95993242, 0.86083483, 0.98036558, 0.91853993, 0.73721797,
0.79515757, 0.8901948 , 0.94712961, 0.87530472, 0.54911527,
0.72719289, 0.82497862, 0.85963216, 0.31304449, 0.04840995,
0.15693461, 0.79970341, 0.98603118, 0.66887689, 0.92238138,
0.63510963, 0.51896381, 0.46682113, 0.65443862, 0.91031516,
0.99866804, 0.83977927, 0.13340409, 0.47759811, 0.83488193,
0.70558147, 0.7178214 , 0.90004202, 0.76643548, 0.87468331,
0.66789578, 0.63942166, 0.68798004, 0.24932862, 0.97993006,
0.93873947, 0.99561958, 0.58909503, 0.85343788, 0.76421022,
0.90720714, 0.66771999, 0.66338204, 0.78295846, 0.71143693,
0.96651489, 0.84691559, 0.42888672, 0.9821854 , 0.76862079,
0.5246699 , 0.93703737, 0.78667287, 0.78992515, 0.74359261,
0.38461512, 0.8590153 , 0.28362739, 0.74165777, 0.05143247,
0.55139785, 0.595707  , 0.85690848, 0.75405312, 0.85227204,
0.6994883 , 0.68028082, 0.63038485, 0.86633659, 0.5196427 ,
0.81665162, 0.74236875, 0.78450147, 0.39798913, 0.95318795,
0.75664279, 0.71982445, 0.96305143, 0.85183692, 0.97535771,
0.24402705, 0.96183989, 0.96961159, 0.8568859 , 0.87734076,
0.7956464 , 0.99245264, 0.93434668, 0.90188875, 0.38880162,
0.77651225, 0.9234912 , 0.85398249, 0.68174482, 0.66284271,
0.87412183, 0.35241154, 0.99659201, 0.89051126, 0.85733848,
0.79631229, 0.13642217, 0.93447813, 0.70307343, 0.90508343,
0.54704654, 0.55458331, 0.65790147, 0.62137549, 0.55480837,
0.41023766, 0.64269527, 0.89033538, 0.92295374, 0.75353987,
0.95112727, 0.60455565, 0.66763692, 0.98873187, 0.7459918 ,
0.4833735 , 0.79252854, 0.61176012, 0.79177403, 0.96402422,
0.48379274, 0.65691955, 0.44425861, 0.92942234, 0.61025638,
0.75299831, 0.73547151, 0.87232631, 0.50555371, 0.76789693,
0.87668813, 0.07710387, 0.94812183, 0.86815403, 0.7559856 ,
0.98941657, 0.90096163, 0.28014703, 0.99051441, 0.9293684 ,
0.84907089, 0.64403631, 0.743497  , 0.14531041, 0.9777997 ,
0.91046953, 0.97490688, 0.26331923, 0.88245897, 0.51341849,
0.71151607, 0.9940347 , 0.70066125, 0.5697211 , 0.98762933,
0.98985969, 0.96856282, 0.99273894, 0.6563272 , 0.37995812,
0.99430255, 0.71207432, 0.43837019, 0.96716322, 0.83167786,
0.61445138, 0.83362311, 0.9945238 , 0.59998084, 0.92504335,
0.45151287, 0.88632378, 0.844241  , 0.3850618 , 0.79961397,
0.84410367, 0.39384153, 0.9737171 , 0.85508709, 0.99702365,
0.99815398, 0.61000051, 0.78860591, 0.64765366, 0.68886442])
```

(b) On average, how many samples from the uniform distribution over $[0,1]$ would your acceptance-rejection algorithm need in order to generate one sample of X?

Ans.

$$p_{\text{accept}} = \frac{1}{c},$$

$$\mathbb{E}[N] = \frac{1}{p_{\text{accept}}} = c.$$

For the given density function

$$f(x) = \frac{5x^4 + 4x^3 + 3x^2 + 1}{4}, \quad 0 \le x \le 1,$$

the maximum occurs at $x = 1$:

$$f(1) = \frac{13}{4} = 3.25.$$

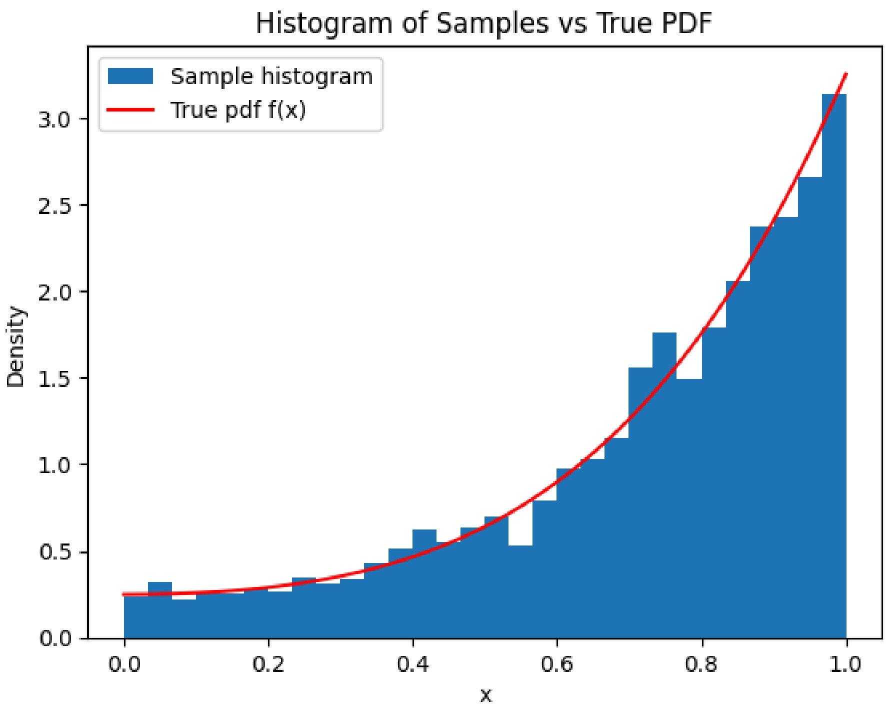
Therefore,

$$p_{\text{accept}} = \frac{1}{3.25}$$
$$\mathbb{E}[N] = 3.25$$

(c) Use your algorithm in (a) to generate 2,500 samples of X. Note that this will require more than 2500 uniform random variables.

Plot a histogram of your sample and compare it against the true pdf.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(111)
samples = sample_X(2500)

x_vals = np.linspace(0,1,200)
pdf_vals = f(x_vals)

plt.hist(samples, bins=30, density=True, label="Sample histogram")
plt.plot(x_vals, pdf_vals, 'r-', label="True pdf f(x)")
plt.xlabel("x")
plt.ylabel("Density")
plt.title("Histogram of Samples vs True PDF")
plt.legend()
plt.show()
```



Question 4: Generalized Acceptance-Rejection (30 pts)

We want to generate a $\mathcal{N}(0,1)$ rv X, with pdf $f(x) = \frac{e^{-x^2/2}}{\sqrt{2\pi}}$, using generalized acceptance-rejection.

(a) First, suppose we choose the proposal distribution to be a \emph{Laplace} (i.e., two-sided Exponential) distribution, which has pdf $g(x) = e^{-|x|}/2$. Describe (and implement) an inversion algorithm to get samples from this distribution.

The CDF is

$$G(x) = \int_{-\infty}^{x} g(t), dt.$$

For $x < 0$: $|x| = -x$, so

$$g(x) = \tfrac{1}{2}e^{x}, \quad x < 0.$$

Hence

$$G(x) = \tfrac{1}{2}e^{x}.$$

For $x \geq 0$: $|x| = x$, so

$$g(x) = \tfrac{1}{2}e^{-x}.$$

The CDF is the sum of the probability mass on $(-\infty, 0]$ and the integral from 0 to x:

$$G(x) = \tfrac{1}{2} + \left(\tfrac{1}{2} - \tfrac{1}{2}e^{-x}\right) = 1 - \tfrac{1}{2}e^{-x}.$$

Let $U \sim \text{Uniform}(0, 1)$. Set $G(x) = U$ and solve for x:

If $0 < U < 0.5$:

$$U = \tfrac{1}{2}e^{x} \quad \Rightarrow \quad x = \ln(2U).$$

If $0.5 \leq U < 1$:

$$U = 1 - \tfrac{1}{2}e^{-x} \quad \Rightarrow \quad x = -\ln(2(1-U)).$$

In [3]:
```
def sample_laplace(n_samples=1):
    U = np.random.rand(n_samples)
    X = np.where(U < 0.5,
                 np.log(2*U),
                 -np.log(2*(1-U)))
    return X

samples = sample_laplace(10)
samples
```

Out[3]:
```
array([ 1.74284943, -1.8709212 ,  0.01563474,  1.3820094 ,  4.03449315,
        3.31980923, -0.99662524, -0.11495978,  0.06469833,  0.05960708])
```

(b) Determine the smallest k such that $kg(x) \geq f(x) \, \forall \, x \in \mathbb{R}$. Using this, propose (and implement) an acceptance-rejection algorithm for sampling $X \sim \mathcal{N}(0, 1)$, and compute the expected number of samples needed for generating each sample.

Ans.

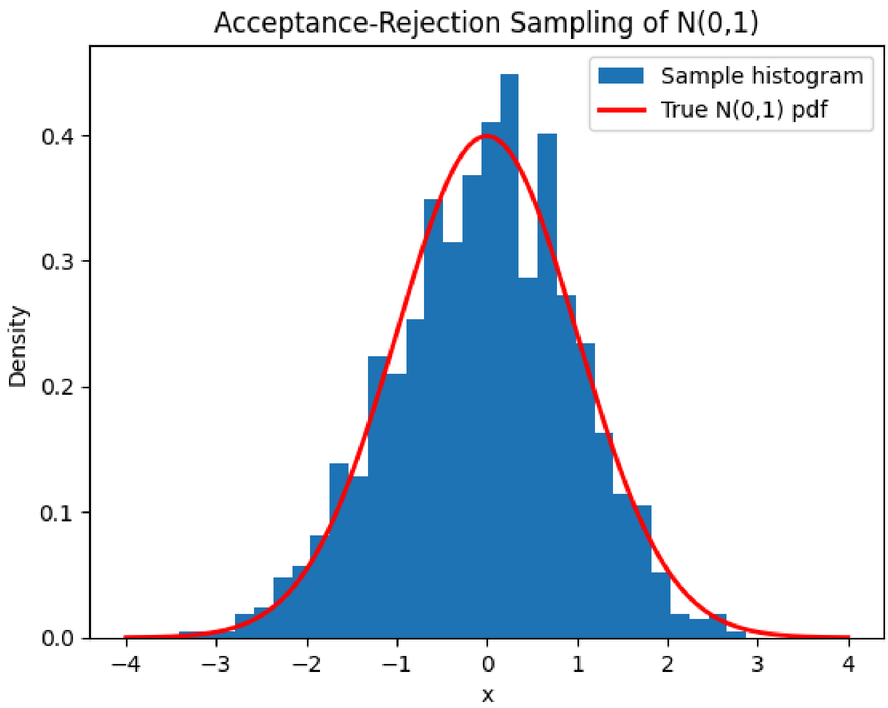
In [4]:
```
def f(x):
    return (1/np.sqrt(2*np.pi)) * np.exp(-x**2/2)

def g(x):
    return 0.5 * np.exp(-np.abs(x))

def ratio(x):
    return f(x) / g(x)

# function to find minimal k which is also the max of f/g
```

```
def find_k(xmin=-10, xmax=10, num_points=200000):
    x_vals = np.linspace(xmin, xmax, num_points)
    ratios = ratio(x_vals)
    return np.max(ratios)

k = find_k()

def sample_laplace(n_samples=1):
    U = np.random.rand(n_samples)
    return np.where(U < 0.5, np.log(2*U), -np.log(2*(1-U)))

def sample_normal(n_samples=1):
    samples = []
    while len(samples) < n_samples:
        y = sample_laplace(1)[0]
        u = np.random.rand()
        if u <= f(y)/(k*g(y)):
            samples.append(y)
    return np.array(samples)


print('Smallest k:',k)
print("Expected uniform draws per accepted sample ≈", k)
```

```
Smallest k: 1.3154892456269678
Expected uniform draws per accepted sample ≈ 1.3154892456269678
```

(c) Generate 1000 samples from your method in part (b), and plot the histogram of the samples. Also report the average and 95% CI for the number of $U[0,1]$ samples needed to generate the 1000 samples.

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

np.random.seed(111)
samples = sample_normal(1000)

x_vals = np.linspace(-4, 4, 300)
pdf_vals = f(x_vals)

plt.hist(samples, bins=30, density=True, label="Sample histogram")
plt.plot(x_vals, pdf_vals, 'r-', linewidth=2, label="True N(0,1) pdf")
plt.xlabel("x")
plt.ylabel("Density")
plt.title("Acceptance-Rejection Sampling of N(0,1)")
plt.legend()
plt.show()
```



In [6]:
```
def sample_normal_ar(n_samples=1):
    samples = []
    U_count = 0
    while len(samples) < n_samples:
        y = sample_laplace(1)[0]
        U_count += 1
        u = np.random.rand()
        U_count += 1

        if u <= f(y)/(k*g(y)):
            samples.append(y)
    return np.array(samples), U_count

np.random.seed(111)
samples, U_used = sample_normal_ar(1000)

avg_U_per_sample = U_used / 1000

trials = []
for _ in range(200):
    _, U_count = sample_normal_ar(200)
    trials.append(U_count/200)
mean_val = np.mean(trials)
se_val = np.std(trials, ddof=1)/np.sqrt(len(trials))
ci_low, ci_high = st.norm.interval(0.95, loc=mean_val, scale=se_val)

print('Average uniforms per accepted sample: ',avg_U_per_sample)
print("95% CI ≈", ci_low, ci_high)
```

```
Average uniforms per accepted sample:  2.68
95% CI ≈ 2.625716449069319 2.649783550930682
```

(d) Now, suppose instead we choose the proposal distribution to be a Cauchy distribution with pdf $g(x) = \frac{1}{\pi(1+x^2)}$. Describe and implement an inversion algorithm to get samples from this distribution, and plot the histogram of 1000 samples from this distribution.

The standard Cauchy has pdf

$$g(x) = \frac{1}{\pi(1+x^2)}, \quad x \in \mathbb{R}.$$

$$G(x) = \frac{1}{\pi}\arctan(x) + \frac{1}{2}.$$

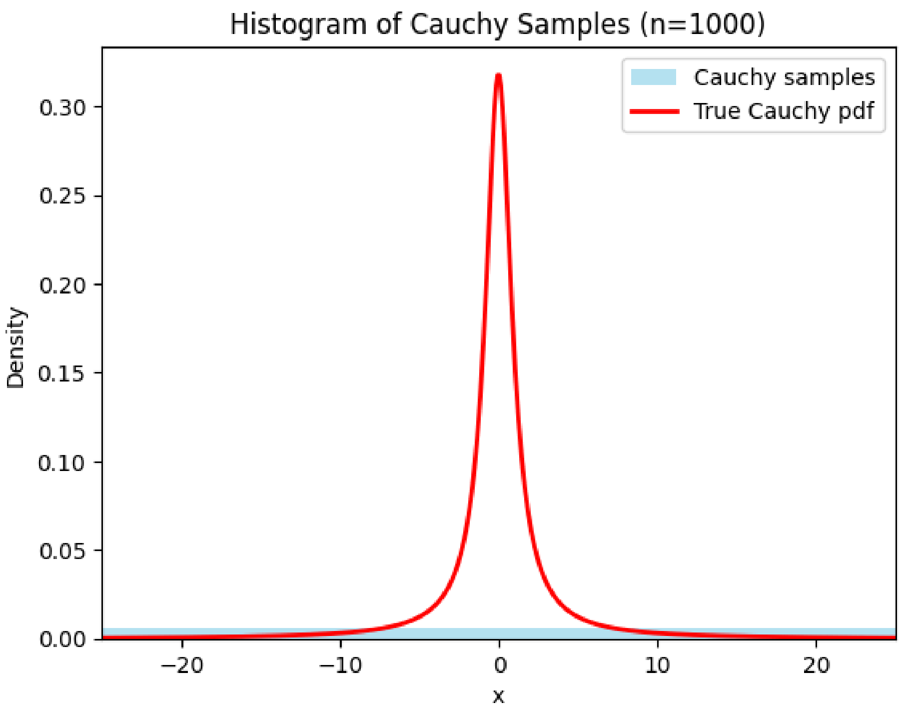
$$G^{-1}(u) = \tan\big(\pi(u - 0.5)\big).$$

To generate a Cauchy random variable:

Generate $U \sim \text{Uniform}(0, 1)$

Set $X = \tan(\pi(U - 0.5))$

In [7]:
```
def sample_cauchy(n_samples=1):
    U = np.random.rand(n_samples)
    X = np.tan(np.pi * (U - 0.5))
    return X

np.random.seed(111)
samples_cauchy = sample_cauchy(1000)

plt.hist(samples_cauchy, bins=50, density=True, alpha=0.6, color='skyblue', label="Cauchy sampl

x_vals = np.linspace(-25, 25, 500)
pdf_vals = 1/(np.pi*(1+x_vals**2))
plt.plot(x_vals, pdf_vals, 'r-', linewidth=2, label="True Cauchy pdf")

plt.xlim(-25,25)
plt.xlabel("x")
plt.ylabel("Density")
plt.title("Histogram of Cauchy Samples (n=1000)")
plt.legend()
plt.show()
```



(e) Repeat parts (b) and (c) for this proposal distribution.

Ans.

In [8]:
```
import numpy as np

def compute_k_and_expected_uniforms(f, g, search_range=(-10, 10), num_points=200000):
    x_vals = np.linspace(search_range[0], search_range[1], num_points)
    ratios = f(x_vals) / g(x_vals)
    k_min = np.max(ratios)

    expected_uniforms = 2 * k_min
```

```
    return k_min, expected_uniforms

def f(x):
    return (1/np.sqrt(2*np.pi)) * np.exp(-x**2/2)

def g(x):
    return 1/(np.pi*(1+x**2))

k_val, exp_uniforms = compute_k_and_expected_uniforms(f, g)
print("Least k =", k_val)
print("Expected uniforms per sample =", exp_uniforms)
```

```
Least k = 1.5203468995269371
Expected uniforms per sample = 3.0406937990538743
```