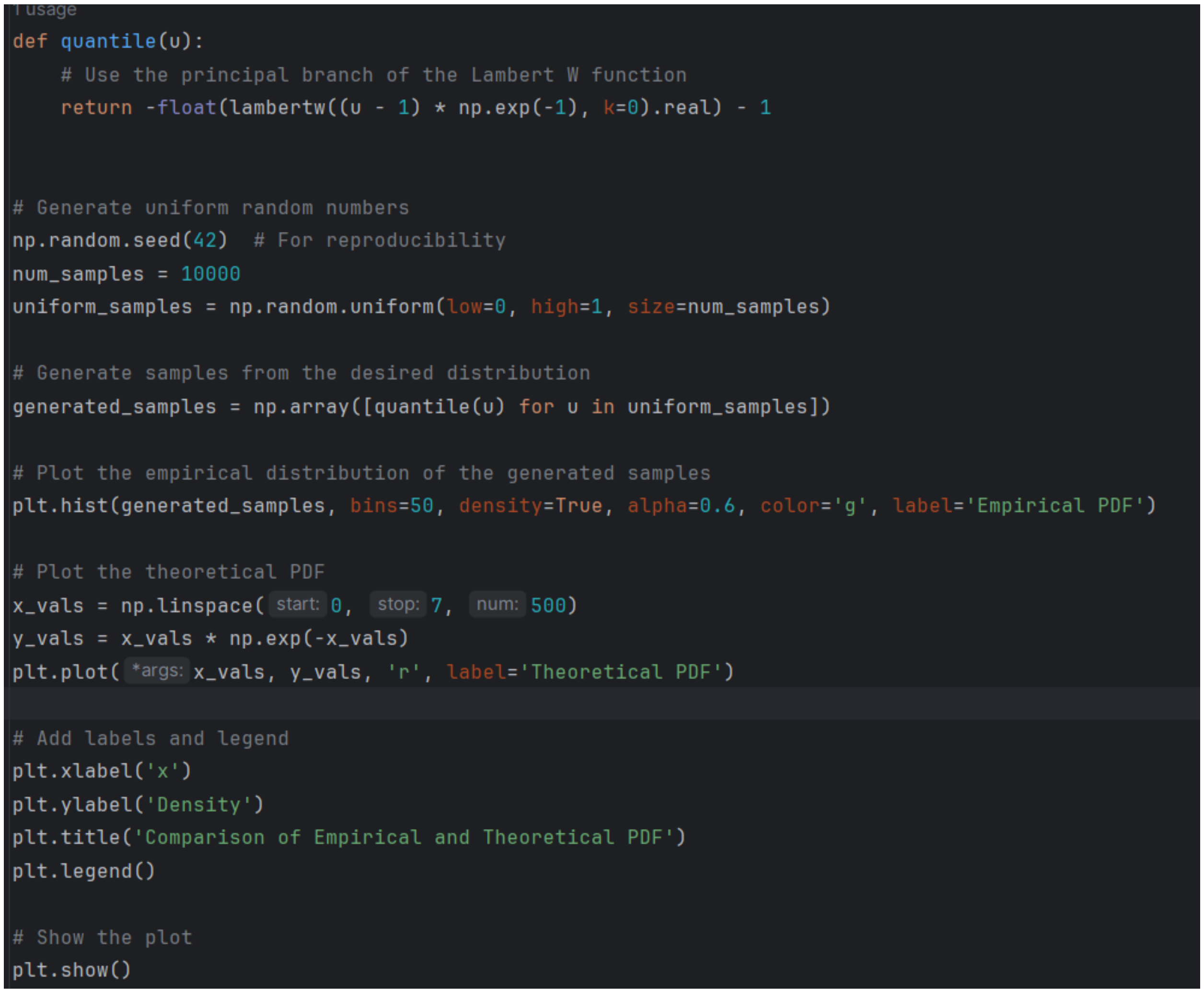
Probability and statistics

Project 2

Description

Mohammad Mahdi Daghighi

9926163



```
1 usage
def quantile(u):
    # Use the principal branch of the Lambert W function
    return -float(lambertw((u - 1) * np.exp(-1), k=0).real) - 1


# Generate uniform random numbers
np.random.seed(42)  # For reproducibility
num_samples = 10000
uniform_samples = np.random.uniform(low=0, high=1, size=num_samples)

# Generate samples from the desired distribution
generated_samples = np.array([quantile(u) for u in uniform_samples])

# Plot the empirical distribution of the generated samples
plt.hist(generated_samples, bins=50, density=True, alpha=0.6, color='g', label='Empirical PDF')

# Plot the theoretical PDF
x_vals = np.linspace( start: 0,  stop: 7,  num: 500)
y_vals = x_vals * np.exp(-x_vals)
plt.plot( *args: x_vals, y_vals, 'r', label='Theoretical PDF')

# Add labels and legend
plt.xlabel('x')
plt.ylabel('Density')
plt.title('Comparison of Empirical and Theoretical PDF')
plt.legend()

# Show the plot
plt.show()
```

This code is an example of using inverse transform sampling to generate samples from a specific probability distribution and then comparing the empirical distribution of the generated samples with the theoretical probability density function (PDF). Here's a detailed explanation of each part of the code:

1. Importing Libraries:

   - `numpy`: A library for numerical computations in Python.

   - `scipy.special`: A module in SciPy that includes many special functions, including the Lambert W function.

   - `matplotlib.pyplot`: A library for creating static, interactive, and animated visualizations in Python.

2. Defining the Quantile Function:

   - `quantile(u)`: This function computes the quantile (inverse of the cumulative distribution function) for a given uniform random variable `u`. It uses the Lambert W function to transform a uniform distribution into a target distribution.

   - `lambertw((u - 1) * np.exp(-1), k=0)`: The Lambert W function, specifically the principal branch (indicated by `k=0`), is applied to `(u - 1) * np.exp(-1)`. The real part of the result is extracted and used in the calculation.

3. Generating Uniform Random Numbers:

   - `np.random.seed(42)`: Sets the seed for NumPy's random number generator to make the output reproducible.

- `np.random.uniform(...)`: Generates `num_samples` (10,000) random numbers uniformly distributed between 0 and 1.

4. Generating Samples from the Target Distribution:

   - `generated_samples = np.array([...])`: For each uniform random number `u`, the `quantile` function is applied to generate a sample from the target distribution.

5. Plotting the Empirical Distribution:

   - `plt.hist(...)`: Plots a histogram of the generated samples to represent the empirical PDF. The `density=True` parameter normalizes the histogram so that it represents a probability density function.

6. Plotting the Theoretical PDF:

   - `x_vals = np.linspace(0, 7, 500)`: Generates 500 points evenly spaced between 0 and 7.

   - `y_vals = x_vals * np.exp(-x_vals)`: Computes the theoretical PDF for the target distribution. The PDF appears to be for an exponential-type distribution.

   - `plt.plot(...)`: Plots the theoretical PDF as a red line.

7. Adding Labels and Legend:

   - The plot is labeled with x-axis and y-axis labels, a title, and a legend to differentiate between the empirical and theoretical PDFs.

8. Displaying the Plot:

   - `plt.show()`: Displays the plot.

In summary, this code demonstrates a method to generate random samples from a non-standard distribution using inverse transform sampling, and then visually compares the empirical distribution of these samples with the theoretical PDF. This approach is valuable in statistics and simulations where direct sampling from the desired distribution is not straightforward.