DSCI 6607– Fall 2024 Assignment 2

2024-10-18

```
# Abdallah Chidjou
# Citation: (source of help: googling in general, stackoverflow, and chatgpt)
```

Question 1

```
# (a). Incorporate bracket broadening into the bisection method. Note that broadening
# is not guaranteed to find x l and x r such that f(x l )f(x r ) <= 0, so you should
# include a limit on the number of times it can be tried.

bisection_broadening <- function(f, xl, xr) {
  # Initialize variables
  iter <- 0
  broaden_count <- 0
# The maximum number of iterations here is 100 and could vary
  while (iter < 100) {
    # Calculate midpoint
    m <- (xl + xr) / 2
    w <- xr - xl

    # Check if the product of f(xl) and f(xr) is less than or equal to zero
    if (f(xl) * f(xr) <= 0) {
      # Bisection process with a tolerance of 1e-5
      if (abs(xr - xl) < 1e-5) {
        return(m)  # Root found within tolerance
      }
      if (f(xl) * f(m) < 0) {
        xr <- m
      } else {
        xl <- m
      }
    } else {
      # Broaden the interval, maximum number of times the interval can
      # be broadened in this case 5; changeable as well
      if (broaden_count < 5) {
        broaden_count <- broaden_count + 1
        xl <- m - w
        xr <- m + w
      } else {
        stop("Bracket broadening limit reached without finding a valid interval.")
      }
    }
```

```
    iter <- iter + 1
  }

  stop("Maximum iterations reached without finding the root.")
}

# Define the function f(x)
f <- function(x) {
  (x - 1)^3 - 2 * x^2 + 10 - sin(x)
}

# (b). Use your modified function to find a root of f(x)
root <- bisection_broadening(f, 1, 2)
print(root)
```

```
## [1] -1.052898
```

Question 2

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

# (a)- Write a python function which takes u1, u2 and simulates x and y
# Box-Muller transformation function
def box_muller(u1, u2):
    # Apply Box-Muller transformation
    x = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    y = np.sqrt(-2 * np.log(u1)) * np.sin(2 * np.pi * u2)
    return x, y

# Example usage with random numbers between [0, 1]
u1, u2 = np.random.uniform(0, 1, 2)
x, y = box_muller(u1, u2)
print(x, y)
```

```
## -0.8047850029190461 -0.5203837306788707
```

```
# b- Generate 1000 observations using the Box-Muller function
n = 500  # Number of pairs to generate
u1 = np.random.uniform(0, 1, n)
u2 = np.random.uniform(0, 1, n)

# Apply Box-Muller transformation to each pair
x_values = []
y_values = []
for i in range(n):
    x, y = box_muller(u1[i], u2[i])
    x_values.append(x)
```

```python
    y_values.append(y)

# Combine the generated x and y values into one list
generated_data = np.array(x_values + y_values)
print()
```

```python
print("The total count of elements in the array:",generated_data.size)
```

```
## The total count of elements in the array: 1000
```

```python
print()
```

```python
print("The first 5 elements of the array:",generated_data[:5])
```

```
## The first 5 elements of the array: [-0.97190992  1.10559803 -0.93856575  0.56825071  1.21910808]
```

```python
print()
```

```python
print()
```

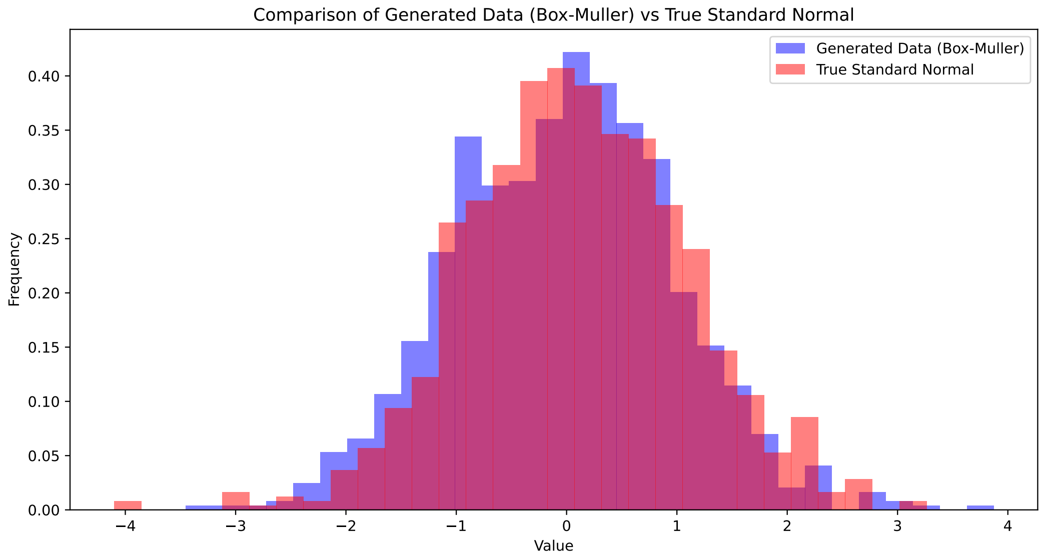
```python
# (c)- Plot the histogram of the generated data and compare with true standard normal
plt.figure(figsize=(12, 6))

# Histogram of the generated data
plt.hist(generated_data, bins=30, density=True, alpha=0.5, color='blue',
        label='Generated Data (Box-Muller)')

# Generate data from standard normal distribution
true_normal_data = np.random.normal(0, 1, 1000)

# Histogram of the true standard normal data
plt.hist(true_normal_data, bins=30, density=True, alpha=0.5, color='red',
        label='True Standard Normal')

# Labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Comparison of Generated Data (Box-Muller) vs True Standard Normal')
plt.legend()
plt.show()
```



Question 3

Write a python function where takes the list and uses only list comprehension and returns the odd values smaller than 23.

```
# Function returns the odd values smaller than 23 given x
def odd_values(x):
  return [i for i in x if i < 23 and i % 2 != 0]

# Given list
x = [3, 8, 13, 18, 108, 25, 23, 17, 203, 11, 23]

# Call the function and print the result
result = odd_values(x)
print(result)
```

```
## [3, 13, 17, 11]
```

Question 4

The problem gives you $X_i \sim N(\mu, \sigma^2)$ for $i = 1, \ldots, 10$, and you need to find the distribution of the statistic:

$$\sum_{i=1}^{10} X_i$$

Since each X_i is normally distributed, and you are summing 10 independent normal random variables, we can determine the distribution of the sum using properties of the normal distribution:

- **Mean**: If $X_i \sim N(\mu, \sigma^2)$, then the mean of the sum $\sum_{i=1}^{10} X_i$ is:

$$E\left(\sum_{i=1}^{10} X_i\right) = \sum_{i=1}^{10} E(X_i) = 10\mu$$

- **Variance**: Similarly, the variance is:

$$Var\left(\sum_{i=1}^{10} X_i\right) = \sum_{i=1}^{10} Var(X_i) = 10\sigma^2$$

Thus, the sum $S = \sum_{i=1}^{10} X_i$ follows a normal distribution:

$$S \sim N(10\mu, 10\sigma^2)$$

Summary

- **Distribution**: The sum of 10 independent normal random variables, each with mean μ and variance σ^2, follows a normal distribution with mean 10μ and variance $10\sigma^2$.
- **Mathematical Explanation**: By the properties of summation of independent normal random variables, the mean and variance scale linearly with the number of variables, resulting in the above distribution for S.

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

mean = 23
sigma_squared = 3.6
sigma = np.sqrt(sigma_squared)
n_samples = 10
n_simulations = 10000

# (2) Generate a sample of size 10 from the Normal distribution
# with parameters mean = 23 and sigma_squared = 3.6 and the compute the sum
# of the generated observations

sample = np.random.normal(mean, sigma, n_samples)
sample_sum = np.sum(sample)
print(f"Sum of a single generated sample of size 10: {sample_sum}")
```

```
## Sum of a single generated sample of size 10: 234.93154078088864
```

```
# print("welcome")

# (3) Simulate 10000 times part (2) and compute the sum
#of the generated samples of size 10
```

```
samples = []
for _ in range(n_simulations):
    sample = np.random.normal(mean, sigma, n_samples)
    samples.append(sample)

# Compute the sums of the generated samples of size 10
sample_sums = [np.sum(sample) for sample in samples]

# Convert list to numpy array
# sample_sums = np.array(sample_sums)

# (4) Plot the histogram of the 10000 observed statistics from part (3).
# Then show the density curve of the theoretical distribution you
# found in part (1) on the histogram.
plt.hist(sample_sums, bins=50, density=True, alpha=0.6, color='g', label='Simulated Sums')
```

```
## (array([0.00010724, 0.        , 0.00010724, 0.        , 0.00010724,
##        0.00053622, 0.00096519, 0.00117968, 0.00139416, 0.00311006,
##        0.00268108, 0.00589838, 0.00836498, 0.01029536, 0.01479957,
##        0.01855309, 0.02530942, 0.03002812, 0.03421061, 0.04032348,
##        0.04611462, 0.05362165, 0.05823311, 0.06327355, 0.06456046,
##        0.0656329 , 0.06552565, 0.06616911, 0.06187938, 0.05909106,
##        0.05147678, 0.04246835, 0.0359265 , 0.03431785, 0.02563115,
##        0.02316455, 0.01608649, 0.01233298, 0.00911568, 0.00600562,
##        0.00514768, 0.00300281, 0.00225211, 0.00128692, 0.0007507 ,
##        0.00064346, 0.00021449, 0.00021449, 0.        , 0.00032173]), array([205.91856753, 206.851026
##        209.64840438, 210.58086359, 211.5133228 , 212.44578201,
##        213.37824122, 214.31070044, 215.24315965, 216.17561886,
##        217.10807807, 218.04053728, 218.97299649, 219.9054557 ,
##        220.83791492, 221.77037413, 222.70283334, 223.63529255,
##        224.56775176, 225.50021097, 226.43267019, 227.3651294 ,
##        228.29758861, 229.23004782, 230.16250703, 231.09496624,
##        232.02742545, 232.95988467, 233.89234388, 234.82480309,
##        235.7572623 , 236.68972151, 237.62218072, 238.55463994,
##        239.48709915, 240.41955836, 241.35201757, 242.28447678,
##        243.21693599, 244.1493952 , 245.08185442, 246.01431363,
##        246.94677284, 247.87923205, 248.81169126, 249.74415047,
##        250.67660968, 251.6090689 , 252.54152811]), <BarContainer object of 50 artists>)
```

```
# Overlay the theoretical density curve
x = np.linspace(min(sample_sums), max(sample_sums), 10000)
th_mean = 10 * mean
th_variance = 10 * sigma_squared
th_std = np.sqrt(th_variance)
th_pdf = stats.norm.pdf(x, th_mean, th_std)
plt.plot(x, th_pdf, 'r-', lw=2, label='Theoretical Density')
```

```
## [<matplotlib.lines.Line2D object at 0x12b528910>]
```

```
# Adding labels and legend
plt.xlabel('Sum of 10 Samples')
```

```
## Text(0.5, 0, 'Sum of 10 Samples')
```

```
plt.ylabel('Density')
```

```
## Text(0, 0.5, 'Density')
```

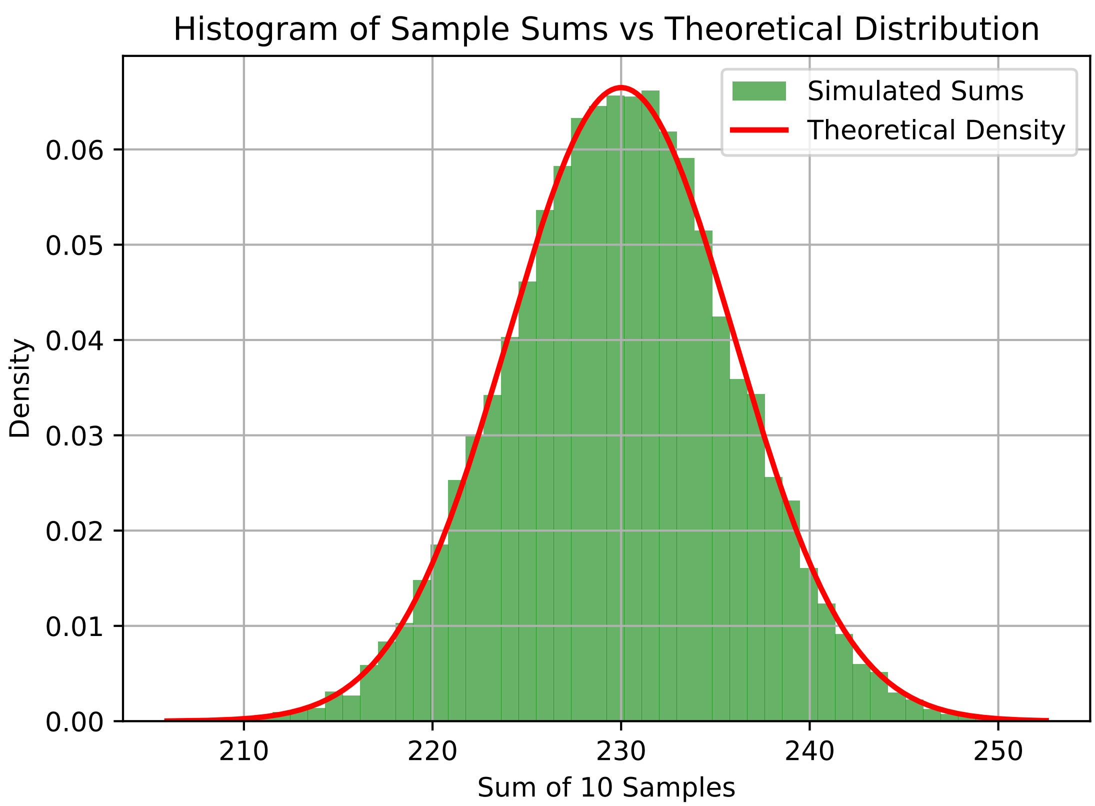
```
plt.title('Histogram of Sample Sums vs Theoretical Distribution')
```

```
## Text(0.5, 1.0, 'Histogram of Sample Sums vs Theoretical Distribution')
```

```
plt.legend()
```

```
## <matplotlib.legend.Legend object at 0x12b8a9ad0>
```

```
plt.grid(True)
plt.show()
```




```
# - We observe that the histogram of the 10,000 sample sums closely
# matches the theoretical normal distribution curve with mean 10 * mu and variance 10 * sigma_squared.
# This indicates that the observed distribution aligns well with the theoretical distribution, as expec
```

Question 6

```
import numpy as np

def summary_statistics(x):
    # Sort the list
    x_sorted = sorted(x)

    # Compute the necessary statistics
    min_val = np.min(x_sorted)
    q1 = np.percentile(x_sorted, 25)
    median = np.median(x_sorted)
    q3 = np.percentile(x_sorted, 75)
    max_val = np.max(x_sorted)
    iqr = q3 - q1

    # Define the boundaries for outliers
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Find outliers
    outliers = [value for value in x_sorted if value < lower_bound or value > upper_bound]

    # Return the statistics in a dictionary
    return {
        "Min": min_val,
        "Q_1": q1,
        "M": median,
        "Q_3": q3,
        "Max": max_val,
        "IQR": iqr,
        "Outliers": outliers
    }

# Apply the function to the provided list
x = [2, 36, 12, 14, 204, 21.6, 22.5, 1, 32.8, 32.1, 13, 10, 88, 3.3, 3.1, 88]
result = summary_statistics(x)

# Print the result
print(result)
```

```
## {'Min': np.float64(1.0), 'Q_1': np.float64(8.325), 'M': np.float64(17.8), 'Q_3': np.float64(33.59999
```

```
print(f"Min : {result['Min']}")
```

```
## Min : 1.0
```

```
print(f"Q_1 : {result['Q_1']}")
```

```
## Q_1 : 8.325
```

```
print(f"M : {result['M']}")
```

```
## M : 17.8
```

```
print(f"Q_3 : {result['Q_3']}")
```

```
## Q_3 : 33.599999999999994
```

```
print(f"Max : {result['Max']}")
```

```
## Max : 204.0
```

```
print(f"IQR : {result['IQR']}")
```

```
## IQR : 25.274999999999995
```

```
print(f"Outliers : {result['Outliers']}")
```

```
## Outliers : [88, 88, 204]
```

Question 6

```
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.utils import resample

# (2). Write a python function which takes X and y
# where X is (n × p) and y is your response vector n × 1.

def leave_one_out_cross_validation(X, y):
    n, p = X.shape
    y_hat = np.zeros(n)

    for i in range(n):
        # Leave one out
        X_train = np.delete(X, i, axis=0)
        y_train = np.delete(y, i)
        x_test = X[i, :]

        # Compute beta coefficients using (X^T * X)^(-1) * X^T * y
        XTX_inv = np.linalg.inv(X_train.T @ X_train)
        beta_hat = XTX_inv @ X_train.T @ y_train

        # Predict the response for the left-out observation
        y_hat[i] = x_test @ beta_hat
```

```
    # Calculate Root MSE
    mse = np.mean((y - y_hat) ** 2)
    root_mse = np.sqrt(mse)

    return root_mse

# (3). Load the diabetes data from sklearn package in python
diabetes = load_diabetes()
X_sample, y_sample = resample(diabetes.data[:, :3], diabetes.target, n_samples=56, random_state=42)

# (4). Apply your function form part 2 to the data set of part 3 and report the root MSE.
root_mse = leave_one_out_cross_validation(X_sample, y_sample)
print(f"Root MSE: {root_mse}")
```

```
## Root MSE: 166.40011704608952
```