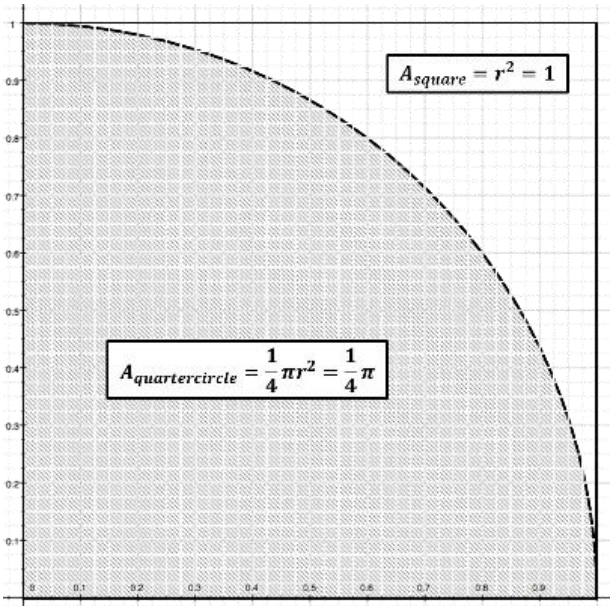
Homework Assignment (Problem Set) 4:

Jerry Zhao

1. Perform Monte Carlo integration using R statistical programming or Python programming to estimate the value of π. To summarize the approach, consider the unit quarter circle illustrated in the figure below:



Generate N pairs of uniform random numbers (x, y), where $x \sim U(0,1)$ and $y \sim U(0,1)$, and each (x, y) pair represents a point in the unit square. To obtain an estimate of π, count the fraction of points that fall inside the unit quarter circle and multiply by 4. Note that the fraction of points that fall inside the quarter circle should tend to the ratio between the area of the unit quarter circle (i.e., ¼ π) as compared to area of the unit square (i.e., 1). We proceed step-by-step:

a) Create a function inside circle that takes two inputs between 0 and 1 and returns 1 if these points fall within the unit circle.

b) Create a function estimate pi that takes a single input N, generates N pairs of uniform random numbers and uses insidecircle to produce an estimate of π as described above. In addition to the estimate of π, estimatepi should also return the standard error of this estimate, and a 95% confidence interval for the estimate.

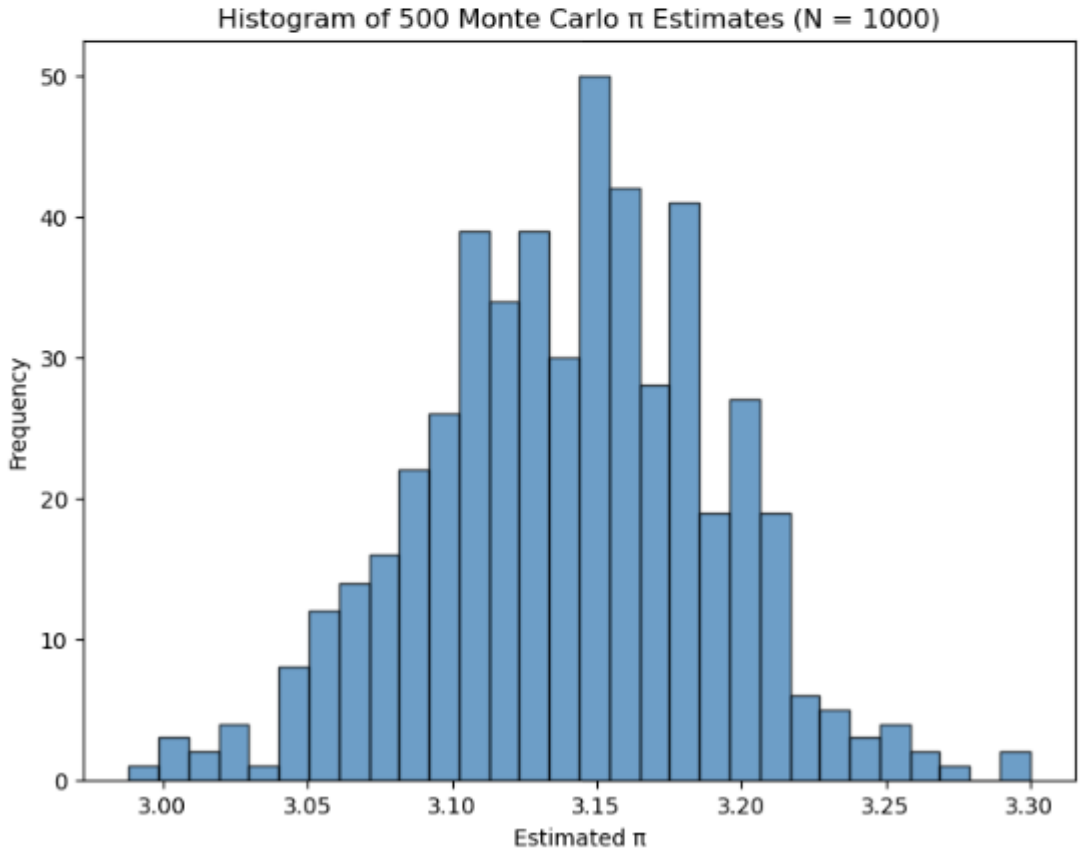
c) Use estimate pi to estimate π for N = 1000 to 10000 in increments of 500 and record the estimate, its standard error and the upper and lower bounds of the 95% CI. How large must N be in order to ensure that your estimate of π is within 0.1 of the true value?

d) Using the value of N you determined in part c), run estimatepi 500 times and collect 500 different estimates of π. Produce a histogram of the estimates and note the shape of this distribution. Calculate the standard deviation of the estimates – does it match the standard error you obtained in part c)? What percentage of the estimates lies within the 95% CI you obtained in part c)?

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import pandas as pd

# Step 1: Function to check if a point is inside the quarter circle
def inside_circle(x, y):
    return x**2 + y**2 <= 1

# Step 2: Function to estimate π
def estimate_pi(N):
    x = np.random.uniform(0, 1, N)
    y = np.random.uniform(0, 1, N)
    inside = np.array([inside_circle(xi, yi) for xi, yi in zip(x, y)])

    pi_estimate = 4 * np.mean(inside)
    std_error = np.sqrt((pi_estimate * (4 - pi_estimate)) / N)

    ci_lower = pi_estimate - 1.96 * std_error
    ci_upper = pi_estimate + 1.96 * std_error

    return pi_estimate, std_error, ci_lower, ci_upper

# Step 3: Estimating π for N = 1000 to 10000 in increments of 500
results = []
N_values = np.arange(1000, 10001, 500)

for N in N_values:
    pi_estimate, std_error, ci_lower, ci_upper = estimate_pi(N)
    results.append([N, pi_estimate, std_error, ci_lower, ci_upper])

df_results = pd.DataFrame(results, columns=['N', 'Pi Estimate', 'Standard Error', 'CI Lower', 'CI Upper'])
display(df_results)

# Step 4: Finding the N value to ensure accuracy within 0.1 of π
required_N = None
for N, pi_estimate, std_error, ci_lower, ci_upper in results:
    if abs(pi_estimate - np.pi) < 0.1:
        required_N = N
        break

# Step 5: Running estimate_pi 500 times for the determined N
N_final = required_N if required_N else 10000
estimates = [estimate_pi(N_final)[0] for _ in range(500)]

# Histogram of π estimates
plt.figure(figsize=(8,6))
plt.hist(estimates, bins=30, edgecolor='black', alpha=0.7)
plt.xlabel("Estimated π")
plt.ylabel("Frequency")
plt.title(f"Histogram of 500 Monte Carlo π Estimates (N = {N_final})")
plt.show()

std_dev_estimates = np.std(estimates)

# Checking percentage of estimates within 95% CI
within_CI = sum(ci_lower <= est <= ci_upper for est in estimates) / 500 * 100

# Displaying final analysis
{
    "N ensuring accuracy within 0.1 of π": N_final,
    "Standard Deviation of Estimates": std_dev_estimates,
    "Percentage within 95% CI": within_CI
}
```

	N	Pi Estimate	Standard Error	CI Lower	CI Upper
0	1000	3.208000	0.050406	3.109205	3.306795
1	1500	3.130667	0.042596	3.047179	3.214154
2	2000	3.142000	0.036714	3.070041	3.213959
3	2500	3.118400	0.033161	3.053404	3.183396
4	3000	3.140000	0.030002	3.081196	3.198804
5	3500	3.129143	0.027903	3.074453	3.183833
6	4000	3.115000	0.026252	3.063545	3.166455
7	4500	3.159111	0.024297	3.111490	3.206732
8	5000	3.107200	0.023555	3.061033	3.153367
9	5500	3.145455	0.022107	3.102125	3.188784
10	6000	3.118667	0.021403	3.076716	3.160617
11	6500	3.168000	0.020137	3.128531	3.207469
12	7000	3.143429	0.019613	3.104988	3.181869
13	7500	3.137067	0.018999	3.099830	3.174304
14	8000	3.159000	0.018223	3.123282	3.194718
15	8500	3.141647	0.017812	3.106736	3.176558
16	9000	3.116000	0.017495	3.081711	3.150289
17	9500	3.136421	0.016885	3.103328	3.169516
18	10000	3.188800	0.016083	3.157277	3.220323




```
{'N ensuring accuracy within 0.1 of π': 1000,
 'Standard Deviation of Estimates': 0.05026639816020243,
 'Percentage within 95% CI': 73.6}
```

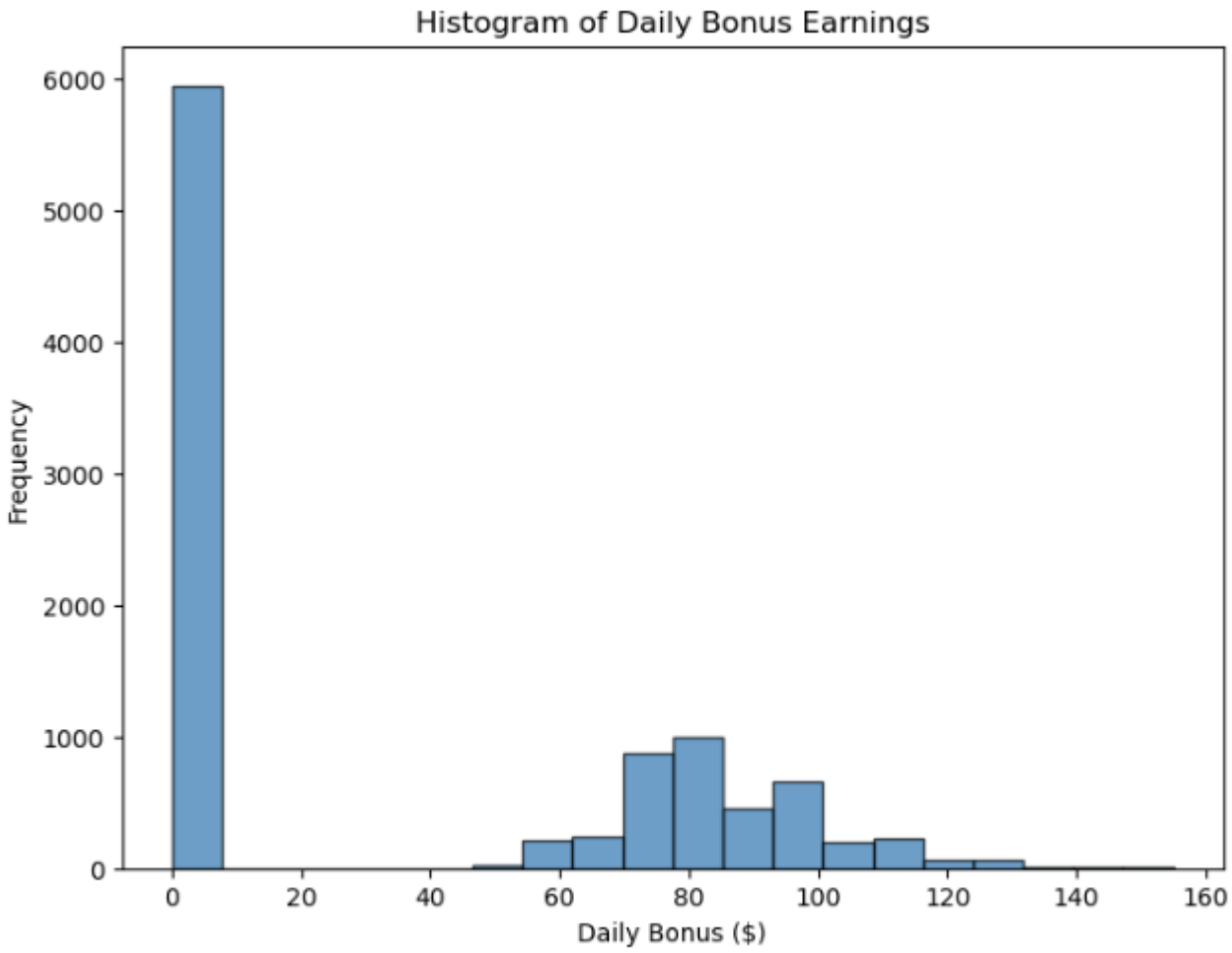
2. A salesperson in a large bicycle shop is paid a bonus if he sells more than 4 bicycles a day. The probability of selling more than 4 bicycles a day is only 0.40. If the number of bicycles sold is greater than 4, the distribution of sales as shown below. The shop has four different models of bicycles. The amount of the bonus paid out varies by type. The bonus for model A is $10; 40% of the bicycles sold are of this type. Model B accounts for 35% of the sales and pays a bonus of $15. Model C has a bonus rating of $20 and makes up 20% of the sales. Finally, a model D pays a bonus of $25 for each sale but accounts for only 5% of the sales. Develop a simulation model to calculate the bonus a salesperson can expect in a day.

Table

Number of Bicycles Sold	Probability
5	0.35
6	0.45
7	0.15
8	0.05

Solution:

```
# Simulation of daily bonus earnings for a salesperson

# Given probabilities for sales above 4 bicycles per day
sales_distribution = {
    5: 0.35,
    6: 0.45,
    7: 0.15,
    8: 0.05
}

# Given probabilities for bicycle model sales
model_distribution = {
    'A': (0.40, 10),
    'B': (0.35, 15),
    'C': (0.20, 20),
    'D': (0.05, 25)
}

# Probability of selling more than 4 bicycles a day
prob_sales_above_4 = 0.40

# Number of simulations
num_simulations = 10000

# Simulate the daily bonus earnings
bonus_earnings = []

for _ in range(num_simulations):
    if np.random.rand() < prob_sales_above_4:
        # Choose number of bicycles sold based on given probabilities
        num_sold = np.random.choice(list(sales_distribution.keys()), p=list(sales_distribution.values()))

        # Assign bike models to sales based on given probabilities
        models_sold = np.random.choice(list(model_distribution.keys()), size=num_sold, p=[v[0] for v in model_distribution.value

        # Calculate total bonus for the day
        total_bonus = sum(model_distribution[model][1] for model in models_sold)
    else:
        total_bonus = 0  # No bonus if selling ≤ 4 bicycles

    bonus_earnings.append(total_bonus)

# Expected daily bonus
expected_bonus = np.mean(bonus_earnings)

# Histogram of bonus earnings
plt.figure(figsize=(8,6))
plt.hist(bonus_earnings, bins=20, edgecolor='black', alpha=0.7)
plt.xlabel("Daily Bonus ($)")
plt.ylabel("Frequency")
plt.title("Histogram of Daily Bonus Earnings")
plt.show()

# Display expected bonus
expected_bonus
```



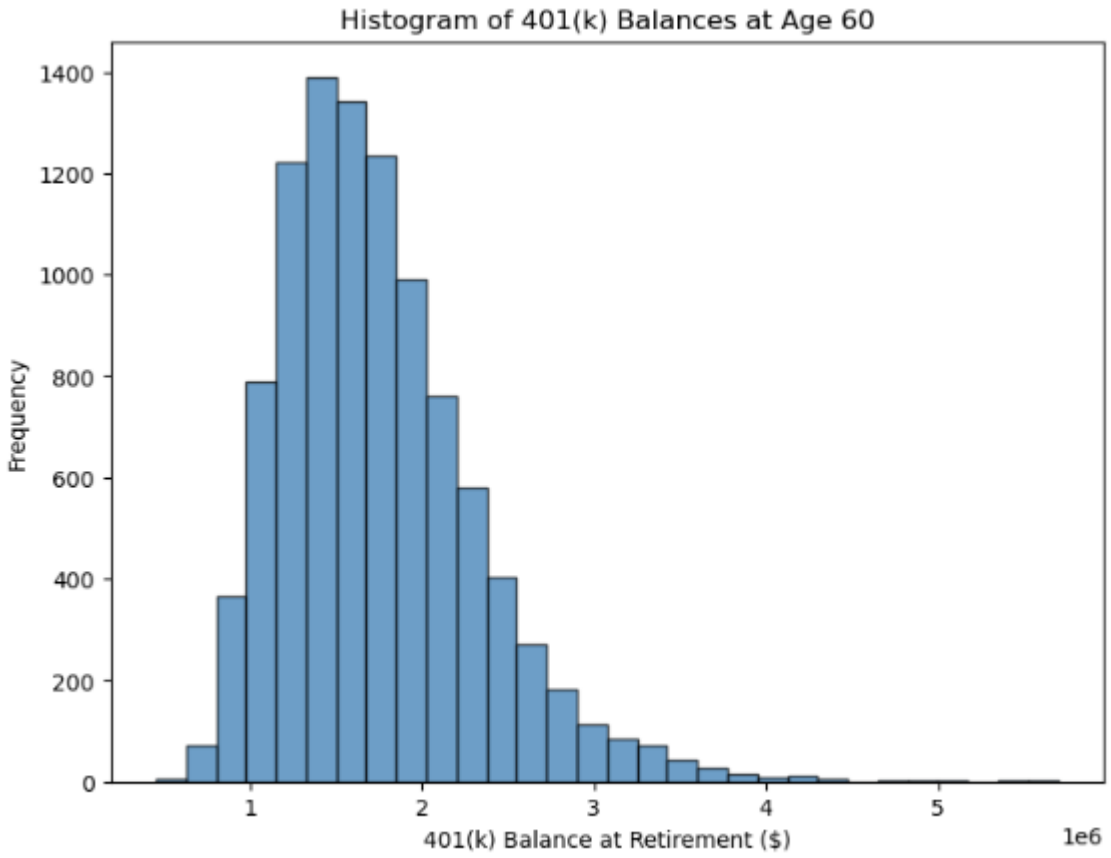

```
expected bonus is  34.726
```

3. An entry-level employee is 23 years old and has a 401(k) plan through their employer. Their employer matches 50% of their contributions up to 6% of their salary. They currently contribute the maximum amount they can (i.e., 6%). In their 401(k), they have three mutual funds and one fixed rate offering. Mutual Fund A is a large-cap index fund with an average annual growth over the past ten years of 6.84% with a standard deviation of 14.36%. Mutual Fund B is a mid-cap index fund with a 10-year average annual growth of 9.59% and a standard deviation of 14.82%. Mutual Fund C is a small-cap Index fund with a 10-year average annual growth rate of 8.07% and a standard deviation of 17.02%. The fixed-rate offering averages an annual growth rate of 5% with no standard deviation. Forty percent of their contribution is directed to Mutual Fund A, 25% to Mutual Fund B, 25% to Mutual Fund C, and 10% to the fixed rate offering. Their current salary is $75,000, and based on a compensation survey, they have an average raise of 2.6% with a standard deviation of 0.3% each year. Develop a Monte Carlo simulation model to predict their 401(k) balance at age 60, which means they expect to work 38 years (from age 23 to 60).

You can assume that the Mutual Funds and salary raises are distributed using the Normal Distribution. It would be best to assume that funds are compounded annually, retirement investments are at the end of the year, and funds are reallocated to the 40%, 25%, 25%, and 10% allocation at the end of each year.

Solution:

```
# Monte Carlo Simulation for 401(k) Balance at Age 60

current_age = 23
retirement_age = 60
years_to_retirement = retirement_age - current_age

initial_salary = 75000
salary_growth_mean = 0.026
salary_growth_std = 0.003

employee_contribution = 0.06
employer_match = 0.5 * employee_contribution

investment_allocation = {
    'A': (0.40, 6.84 / 100, 14.36 / 100),
    'B': (0.25, 9.59 / 100, 14.82 / 100),
    'C': (0.25, 8.07 / 100, 17.02 / 100),
    'Fixed': (0.10, 5.00 / 100, 0.00)
}

num_simulations = 10000

# Simulate 401(k) balance at retirement
final_balances = []

for _ in range(num_simulations):
    salary = initial_salary
    balance = 0

    for _ in range(years_to_retirement):
        # Calculate salary increase
        salary_growth = np.random.normal(salary_growth_mean, salary_growth_std)
        salary *= (1 + salary_growth)

        # Calculate contributions
        annual_contribution = salary * (employee_contribution + employer_match)

        # Simulate returns for each investment type
        total_growth = 0
        for fund, (allocation, mean_return, std_dev) in investment_allocation.items():
            if std_dev == 0:
                annual_return = mean_return  # Fixed-rate return
            else:
                annual_return = np.random.normal(mean_return, std_dev)  # Randomized return
            total_growth += allocation * annual_return

        # Compound the balance
        balance = balance * (1 + total_growth) + annual_contribution

    final_balances.append(balance)

# Expected 401(k) balance at retirement
expected_balance = np.mean(final_balances)

# Histogram of final 401(k) balances
plt.figure(figsize=(8,6))
plt.hist(final_balances, bins=30, edgecolor='black', alpha=0.7)
plt.xlabel("401(k) Balance at Retirement ($)")
plt.ylabel("Frequency")
plt.title("Histogram of 401(k) Balances at Age 60")
plt.show()

print('expected 401k balance age 60 ', round(expected_balance,2))
```




```
expected 401k balance age 60  1740586.79
```