```python
import matplotlib.pyplot as plt
import math
import numpy as np
import time
import pandas as pd
from scipy.stats import norm
def lehmer(seed):
    x = [seed]
    a = math.pow(7,5)
    c = 0
    m = math.pow(2,31) - 1
    for i in range(1,10000):
        temp = (a * x[i-1] + c)%m
        x.append(temp)
    uniform = [u / m for u in x]
    return uniform
def question1(seed):
    uniform = lehmer(seed)
    fig, axs = plt.subplots(1, 2, tight_layout=True)
    n_bins=20
    axs[0].hist(uniform, bins=n_bins)
    axs[0].set_title("MinSTD Lehmer")
    axs[0].set_xlabel("Value")
    axs[0].set_ylabel("Count")
    built_in = np.random.uniform(0.0,1.0,10000)
    axs[1].hist(built_in, bins=n_bins)
    axs[1].set_title("Numpy Built In")
    axs[1].set_xlabel("Value")
    axs[1].set_ylabel("Count")
def question2(seed):
    uniform = lehmer(seed)
    pdf = []
    cdf = [0]
    x = [0]
    for i in range(1,10000):
        x.append(i)
        if uniform[i] < 0.3:
            pdf.append(0)
            cdf.append(cdf[i-1])
        elif uniform[i] < 0.75:
            pdf.append(100)
            cdf.append(cdf[i-1]+100)
        else:
            pdf.append(-200)
            cdf.append(cdf[i-1]-200)
    fig, axs = plt.subplots(1, 2, tight_layout=True)
    bins_list = [-200, -150, -100, -50, 0, 50, 100]
    count = axs[0].hist(pdf, bins=bins_list)
    print(count)
    axs[0].set_title("PDF of Financial Asset")
    axs[0].set_xlabel("Change in Price")
    axs[0].set_ylabel("Count")
    axs[0].set_ylim([0,5000])
    axs[1].plot(x,cdf)
    axs[1].set_title("Emperical Distribution")
```

```
    axs[1].set_xlabel("Time")
    axs[1].set_ylabel("Value of Financial Asset")
    axs[1].set_ylim([-50000, 5000])
def question3():
    binomialRV = []
    for i in range(5000):
        bernoulli = np.random.binomial(1, 0.7, 70)
        estBinomial = sum(bernoulli)
        binomialRV.append(estBinomial)
    plt.hist(binomialRV)
    plt.title("Distribution of 5000 Binomial(70,0.7) Random Variables")
    plt.xlabel("Binomial Random Variable Value")
    plt.ylabel("Count")
    numLessThan50 = [x for x in binomialRV if x < 50]
    print(len(numLessThan50)/5000.)
def binomial(success):
    probability = 0
    for i in range(success):
        nchooseX = math.comb(70,i)
        temp = nchooseX * (0.7)**i * (0.3)**(70-i)
        probability = probability + temp
    print(probability)
def question4():
    normal = np.random.normal(0.0,1.0,5000)
    print(normal)
    print(normal.size)
    plt.hist(normal)
    plt.title("Normal Gaussian Distribution")
    plt.xlabel("Value")
    plt.ylabel("Count")
def question_5():
    simple_monte_carlo_estimates = []
    times = []
    confidence_intervals = []
    all_data = []
    trials = [10,100,1000,10000,100000, 1000000]
    for trial in trials:
        start_time = time.time()
        data = []
        runningCount = 0;
        for j in range(trial):
            total = 0
            count = 0
            while total <= 1:
                x = np.random.uniform(0,1)
                count = count + 1
                total = total + x
                data.append(x)
            runningCount = runningCount + count
        estimate = runningCount / trial
        simple_monte_carlo_estimates.append(estimate)
        end_time = time.time()
        time_taken = end_time - start_time
        times.append(time_taken)
        sample_std = np.std(data)
```

```
        (lb, ub) = estimate - 1.96 * sample_std / np.sqrt(trial), estimate + 1.96 * sample_std / np.sqrt(trial)
        confidence_intervals.append((np.round(lb, 3), np.round(ub, 3)))
        all_data.append(data)
    fig, axs = plt.subplots(1, 3, tight_layout=True, figsize=(10, 5))
    ci_lower_bounds = [x for (x,y) in confidence_intervals]
    ci_upper_bounds = [y for (x,y) in confidence_intervals]
    error = [abs(math.exp(1) - x) for x in simple_monte_carlo_estimates]
    axs[0].plot(np.log10(trials), simple_monte_carlo_estimates, label='Estimate')
    axs[0].scatter(np.log10(trials), ci_lower_bounds, c='m', label='CI Lower Bound')
    axs[0].scatter(np.log10(trials), ci_upper_bounds, c='g', label='CI Upper Bound')
    axs[0].set_title('Simple Monte Carlo Estimate')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(trials), times)
    axs[1].set_title('CPU Time')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Time (s)')
    axs[2].plot(np.log10(trials), error)
    axs[2].set_title('Error')
    axs[2].set_xlabel('10^x')
    axs[2].set_ylabel('Absolute Error')
    df = pd.DataFrame(index=trials)
    df.index.name = 'N'
    df['simple_monte_carlo_estimate'] = simple_monte_carlo_estimates
    df['cpu_time'] = times
    df['error'] = error
    df['confidence_intervals'] = confidence_intervals
    return all_data, trials, df
def printData(data,trials):
    for i in range(0,6,3):
        fig, axs = plt.subplots(1, 3, tight_layout=True)
        for j in range(3):
            axs[j].hist(data[i+j])
            axs[j].set_title("x_is | N = {}".format(trials[i+j]))
            axs[j].set_xlabel("Value")
            axs[j].set_ylabel("Count")
def generate_and_plot_question_six_a_data():
    N = np.power(np.full(6, 10), range(1, 7))
    data = []
    for n in N:
        x_is = np.random.normal(0, 1, n)
        data.append(x_is)
    printData(data,[10,100,1000,10000,100000,1000000])
    return data
def generate_and_plot_question_six_b_data():
    N = np.power(np.full(6, 10), range(1, 7))
    data = []
    for n in N:
        x_is = np.random.normal(5,1,n)
        data.append(x_is)
    printData(data,[10,100,1000,10000,100000,1000000])
    return data
def question_six_a(data):
    simple_monte_carlo_estimates = []
```

```
    times = []
    confidence_intervals = []
    N = np.power(np.full(6, 10), range(1, 7))
    idx = range(len(N))
    for i in idx:
        start_time = time.time()
        n = N[i]
        x_is = data[i]
        good_data = len([x for x in x_is if x > 5])
        estimate = good_data/n
        simple_monte_carlo_estimates.append(estimate)
        # time
        end_time = time.time()
        time_taken = end_time - start_time
        times.append(time_taken)
    fig, axs = plt.subplots(1, 3, tight_layout=True, figsize=(10, 5))
    expected = 1 - norm(0,1).cdf(5)
    error = [abs(expected-actual) for actual in simple_monte_carlo_estimates]
    axs[0].plot(np.log10(N), simple_monte_carlo_estimates, label='Estimate')
    axs[0].set_title('Simple Monte Carlo Estimate')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(N), times)
    axs[1].set_title('CPU Time')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Time (s)')
    axs[2].plot(np.log10(N), error)
    axs[2].set_title('Error')
    axs[2].set_xlabel('10^x')
    axs[2].set_ylabel('Absolute Error')
    df = pd.DataFrame(index=N)
    df.index.name = 'N'
    df['simple_monte_carlo_estimate'] = simple_monte_carlo_estimates
    df['cpu_time'] = times
    df['error'] = error
    return df
def question_six_b(data):
    simple_monte_carlo_estimates = []
    times = []
    confidence_intervals = []
    N = np.power(np.full(6, 10), range(1, 7))
    idx = range(len(N))
    def weight(x, shift):
        return 1/math.sqrt(2*math.pi) * math.exp(-0.5*(x-shift)**2)
    for i in idx:
        start_time = time.time()
        n = N[i]
        x_is = data[i]
        good_data = [x for x in x_is if x > 5]
        w = np.vectorize(weight)
        final_data = w(good_data,0)/w(good_data,5)
        estimate = sum(final_data)/n
        simple_monte_carlo_estimates.append(estimate)
        end_time = time.time()
```

```
        time_taken = end_time - start_time
        times.append(time_taken)
    fig, axs = plt.subplots(1, 3, tight_layout=True, figsize=(10, 5))
    expected = 1 - norm(0,1).cdf(5)
    error = [abs(expected-actual)/expected for actual in simple_monte_carlo_estimates]
    axs[0].plot(np.log10(N), simple_monte_carlo_estimates, label='Estimate')
    axs[0].set_title('Simple Monte Carlo Estimate')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(N), times)
    axs[1].set_title('CPU Time')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Time (s)')
    axs[2].plot(np.log10(N), error)
    axs[2].set_title('Error')
    axs[2].set_xlabel('10^x')
    axs[2].set_ylabel('Relative Error')
    df = pd.DataFrame(index=N)
    df.index.name = 'N'
    df['simple_monte_carlo_estimate'] = simple_monte_carlo_estimates
    df['cpu_time'] = times
    df['error'] = error
    return df
def generate_and_plot_question_seven_data():
    N = np.power(np.full(6, 10), range(1, 7))
    data = []
    for n in N:
        x_is = np.random.uniform(0, 1, n)
        y_is = np.random.uniform(0, 1, n)
        data.append((x_is, y_is))
    for i in range(len(data)):
        fig, axs = plt.subplots(1, 2, tight_layout=True)
        data_for_sample = data[i]
        n = N[i]
        axs[0].hist(data_for_sample[0])
        axs[0].set_title("x_is | N = {}".format(n))
        axs[0].set_xlabel("Value")
        axs[0].set_ylabel("Count")
        axs[1].hist(data_for_sample[1])
        axs[1].set_title("y_is | N = {}".format(n))
        axs[1].set_xlabel("Value")
        axs[1].set_ylabel("Count")
    return data
def question_seven_part_a(data):
    simple_monte_carlo_estimates = []
    times = []
    confidence_intervals = []
    N = np.power(np.full(6, 10), range(1, 7))
    idx = range(len(N))
    for i in idx:
        start_time = time.time()
        n = N[i]
        x_is, y_is = data[i] #Estimate
        xis_plus_yis_squared = np.square(x_is + y_is)
```

```
        e_to_the_xis_plus_yis_squared = np.exp(xis_plus_yis_squared)
        estimate = np.mean(e_to_the_xis_plus_yis_squared)
        simple_monte_carlo_estimates.append(estimate)
        end_time = time.time() #Time
        time_taken = end_time - start_time
        times.append(time_taken)
        sample_std = np.std(e_to_the_xis_plus_yis_squared) #CI
        (lb, ub) = estimate - 1.96 * sample_std / np.sqrt(n), estimate + 1.96 * sample_std / np.sqrt(n)
        confidence_intervals.append((np.round(lb, 3), np.round(ub, 3)))
    fig, axs = plt.subplots(1, 3, tight_layout=True, figsize=(10, 5))
    ci_lower_bounds = [x for (x,y) in confidence_intervals]
    ci_upper_bounds = [y for (x,y) in confidence_intervals]
    axs[0].plot(np.log10(N), simple_monte_carlo_estimates, label='Estimate')
    axs[0].scatter(np.log10(N), ci_lower_bounds, c='m', label='CI Lower Bound')
    axs[0].scatter(np.log10(N), ci_upper_bounds, c='g', label='CI Upper Bound')
    axs[0].set_title('Simple Monte Carlo Estimate')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(N), times)
    axs[1].set_title('CPU Time')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Time (s)')
    true_intergral_estimate = 4.89916
    true_intergral_estimate_array = np.full(len(simple_monte_carlo_estimates), true_intergral_estimate)
    error = np.abs(simple_monte_carlo_estimates - true_intergral_estimate_array)
    axs[2].plot(np.log10(N), error)
    axs[2].set_title('Error')
    axs[2].set_xlabel('10^x')
    axs[2].set_ylabel('Time')
    df = pd.DataFrame(index=N)
    df.index.name = 'N'
    df['simple_monte_carlo_estimate'] = simple_monte_carlo_estimates
    df['cpu_time'] = times
    df['error'] = error
    df['confidence_intervals'] = confidence_intervals
    return df
def question_seven_part_b(data):
    antithetic_estimates = []
    times = []
    confidence_intervals = []
    N = np.power(np.full(6, 10), range(1, 7))
    idx = range(len(N))
    for i in idx:
        start_time = time.time()
        n = N[i]
        # mu 1 (xi + yi)
        x_is, y_is = data[i]
        xis_plus_yis_squared = np.square(x_is + y_is)
        e_to_the_xis_plus_yis_squared = np.exp(xis_plus_yis_squared)
        m1_estimate = np.mean(e_to_the_xis_plus_yis_squared)
        # mu 2 (1 - xi + 1 - yi)
        one_minus_xis_plus_yis_squared = np.square(1 - x_is + 1 - y_is)
        e_to_the_one_minus_xis_plus_yis_squared = np.exp(one_minus_xis_plus_yis_squared)
        m2_estimate = np.mean(e_to_the_one_minus_xis_plus_yis_squared)
```

```
        estimate = 0.5 * (m1_estimate + m2_estimate)
        antithetic_estimates.append(estimate)
        end_time = time.time()
        time_taken = end_time - start_time
        times.append(time_taken)
        sample_std = np.std(0.5 * (e_to_the_xis_plus_yis_squared + e_to_the_one_minus_xis_plus_yis_squared))
        (lb, ub) = estimate - 1.96 * sample_std / np.sqrt(n), estimate + 1.96 * sample_std / np.sqrt(n)
        confidence_intervals.append((np.round(lb, 3), np.round(ub, 3)))
    fig, axs = plt.subplots(1, 3, tight_layout=True, figsize=(10, 5))
    ci_lower_bounds = [x for (x,y) in confidence_intervals]
    ci_upper_bounds = [y for (x,y) in confidence_intervals]
    axs[0].plot(np.log10(N), antithetic_estimates, label='Estimate')
    axs[0].scatter(np.log10(N), ci_lower_bounds, c='m', label='CI Lower Bound')
    axs[0].scatter(np.log10(N), ci_upper_bounds, c='g', label='CI Upper Bound')
    axs[0].set_title('Antithetic Variate Method')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(N), times)
    axs[1].set_title('CPU Time')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Time (s)')
    true_intergral_estimate = 4.89916
    true_intergral_estimate_array = np.full(len(antithetic_estimates), true_intergral_estimate)
    error = np.abs(antithetic_estimates - true_intergral_estimate_array)
    axs[2].plot(np.log10(N), error)
    axs[2].set_title('Error')
    axs[2].set_xlabel('10^x')
    axs[2].set_ylabel('Time')
    df = pd.DataFrame(index=N)
    df.index.name = 'N'
    df['antithetic_estimates'] = antithetic_estimates
    df['cpu_time'] = times
    df['error'] = error
    df['confidence_intervals'] = confidence_intervals
    return df
def question_seven_part_c(data):
    control_variate_estimates = []
    times = []
    confidence_intervals = []
    N = np.power(np.full(6, 10), range(1, 7))
    idx = range(len(N))
    for i in idx:
        start_time = time.time()
        n = N[i]
        mu = 1
        x_is, y_is = data[i]
        y_1 = x_is + y_is
        muy_1 = np.mean(y_1)
        y_2 = np.square(x_is + y_is)
        muy_2 = np.mean(y_2)
        sub_y_1 = [abs(i - muy_1) for i in y_1]
        sub_y_2 = [abs(i - muy_2) for i in y_2]
        c = -1 * sum([sub_y_1[i]*sub_y_2[i] for i in range(len(sub_y_1))]) / sum(sub_y_1)**2
        new_x = y_2 + c*(y_1-mu)
```

```
        values = np.exp(new_x)
        estimate = np.mean(values)
        control_variate_estimates.append(estimate)
        # time
        end_time = time.time()
        time_taken = end_time - start_time
        times.append(time_taken)
        #CI
        sample_std = np.std(values)
        (lb, ub) = estimate - 1.96 * sample_std / np.sqrt(n), estimate + 1.96 * sample_std / np.sqrt(n)
        confidence_intervals.append((np.round(lb, 3), np.round(ub, 3)))
    fig, axs = plt.subplots(1, 3, tight_layout=True, figsize=(10, 5))
    ci_lower_bounds = [x for (x,y) in confidence_intervals]
    ci_upper_bounds = [y for (x,y) in confidence_intervals]
    axs[0].plot(np.log10(N), control_variate_estimates, label='Estimate')
    axs[0].scatter(np.log10(N), ci_lower_bounds, c='m', label='CI Lower Bound')
    axs[0].scatter(np.log10(N), ci_upper_bounds, c='g', label='CI Upper Bound')
    axs[0].set_title('Control Variate Method')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(N), times)
    axs[1].set_title('CPU Time')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Time (s)')
    true_intergral_estimate_array = np.full(len(control_variate_estimates), true_intergral_estimate)
    error = np.abs(control_variate_estimates - true_intergral_estimate_array)
    axs[2].plot(np.log10(N), error)
    axs[2].set_title('Error')
    axs[2].set_xlabel('10^x')
    axs[2].set_ylabel('Time')
    df = pd.DataFrame(index=N)
    df.index.name = 'N'
    df['control_variate_estimates'] = control_variate_estimates
    df['cpu_time'] = times
    df['error'] = error
    df['confidence_intervals'] = confidence_intervals
    return df
def question_eight_part_a(r, sigma, s_0, T, K):
    N = np.power(np.full(6, 10), range(1, 7))
    times = []
    option_price_estimates = []
    option_price_estimates_adjusted = []
    fig, axs = plt.subplots(1, 6, tight_layout=True, figsize=(10, 5))
    fig_num = 0
    for n in N:
        start_time = time.time()
        x_is = []
        x_is_adjusted = []
        for i in range(n):
            w_t = np.random.normal(0, np.sqrt(T))
            s_t = s_0 * np.exp(sigma * w_t + (r - np.square(sigma) / 2) * T)
            price = max(s_t - K, 0)
            price_discount_factor_adjusted = np.exp(-r * T) * max(s_t - K, 0)
            x_is.append(price)
```

```
            x_is_adjusted.append(price_discount_factor_adjusted)
        option_price_monte_carlo_estimate = np.mean(x_is)
        option_price_monte_carlo_estimate_adjusted = np.mean(x_is_adjusted)
        option_price_estimates.append(option_price_monte_carlo_estimate)
        option_price_estimates_adjusted.append(option_price_monte_carlo_estimate_adjusted)
        end_time = time.time()
        time_taken = end_time - start_time
        axs[fig_num].hist(x_is)
        axs[fig_num].set_title("x_is | N = {}".format(n))
        axs[fig_num].set_xlabel("Value")
        axs[fig_num].set_ylabel("Count")
        times.append(time_taken)
        fig_num += 1
    fig, axs = plt.subplots(1, 3, tight_layout=True, figsize=(10, 5))
    axs[0].plot(np.log10(N), option_price_estimates, label='Estimate')
    axs[0].set_title('Monte Carlo Option Price Estimate')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(N), option_price_estimates_adjusted, label='Estimate')
    axs[1].set_title('Monte Carlo Option Price Discount Factor Adjusted')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Estimate')
    axs[1].legend(loc='upper right')
    axs[2].plot(np.log10(N), times)
    axs[2].set_title('CPU Time')
    axs[2].set_xlabel('10^x')
    axs[2].set_ylabel('Time (s)')
    df = pd.DataFrame(index=N)
    df.index.name = 'N'
    df['option_price_estimates'] = option_price_estimates
    df['option_price_estimates_adjusted'] = option_price_estimates_adjusted
    df['cpu_time'] = times
    return df
def bs_call(S, K, T, r, sigma):
    norm_cdf = norm.cdf
    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * norm_cdf(d1) - K * np.exp(-r*T)* norm_cdf(d2)
def put_call_parity(C, s_0, K, r, T):
    PV = K / (1 + r)**T
    return C - s_0 + PV
def question_nine_part_c(r, sigma, s_0, T, K):
    N = np.power(np.full(6, 10), range(1, 7))
    times = []
    option_price_estimates = []
    fig, axs = plt.subplots(1, 6, tight_layout=True, figsize=(10, 5))
    fig_num = 0
    for n in N:
        start_time = time.time()
        x_is = []
        for i in range(n):
            w_t = np.random.normal(0, np.sqrt(T))
            s_t1 = s_0 * np.exp(sigma * w_t + (r - np.square(sigma) / 2) * T)
            s_t2 = s_0 * np.exp(sigma * -w_t + (r - np.square(sigma) / 2) * T)
```

```
            price_1 = np.exp(-r * T) * max(s_t1 - K, 0)
            price_2 = np.exp(-r * T) * max(s_t2 - K, 0)
            price_avg = (price_1 + price_2) / 2
            x_is.append(price_avg)
        option_price_monte_carlo_estimate = np.mean(x_is)
        option_price_estimates.append(option_price_monte_carlo_estimate)
        end_time = time.time()
        time_taken = end_time - start_time
        axs[fig_num].hist(x_is)
        axs[fig_num].set_title("x_is | N = {}".format(n))
        axs[fig_num].set_xlabel("Value")
        axs[fig_num].set_ylabel("Count")
        times.append(time_taken)
        fig_num += 1
    fig, axs = plt.subplots(1, 2, tight_layout=True, figsize=(10, 5))
    axs[0].plot(np.log10(N), option_price_estimates, label='Estimate')
    axs[0].set_title('Antithetic Variables Option Price Estimate')
    axs[0].set_xlabel('10^x')
    axs[0].set_ylabel('Estimate')
    axs[0].legend(loc='upper right')
    axs[1].plot(np.log10(N), times)
    axs[1].set_title('CPU Time')
    axs[1].set_xlabel('10^x')
    axs[1].set_ylabel('Time (s)')
    df = pd.DataFrame(index=N)
    df.index.name = 'N'
    df['option_price_estimates'] = option_price_estimates
    df['cpu_time'] = times
    return df
def question_nine_part_d(r, sigma, s_0, T, K):
    N = np.power(np.full(6, 10), range(1, 7))
    times = []
    option_price_estimates = []
    fig, axs = plt.subplots(1, 6, tight_layout=True, figsize=(10, 5))
    fig_num = 0
    for n in N:
        start_time = time.time()
        x_is = []
        for i in range(n):
            w_t = np.random.normal(0, np.sqrt(T))
            s_t1 = s_0 * np.exp(sigma * w_t + (r - np.square(sigma) / 2) * T)
            price = np.exp(-r * T) * max(s_t1 - K, 0)
            call_price = bs_call(s_0,K,T,r,sigma)
            estimate = price + (bs_call(s_0,K,T,r,sigma) - call_price)
            x_is.append(estimate)
        option_price_monte_carlo_estimate = np.mean(x_is)
        option_price_estimates.append(option_price_monte_carlo_estimate)
        end_time = time.time()
        time_taken = end_time - start_time
        axs[fig_num].hist(x_is)
        axs[fig_num].set_title("x_is | N = {}".format(n))
        axs[fig_num].set_xlabel("Value")
        axs[fig_num].set_ylabel("Count")
        times.append(time_taken)
        fig_num += 1
```

```
fig, axs = plt.subplots(1, 2, tight_layout=True, figsize=(10, 5))
axs[0].plot(np.log10(N), option_price_estimates, label='Estimate')
axs[0].set_title('Control Variate Option Price Estimate')
axs[0].set_xlabel('10^x')
axs[0].set_ylabel('Estimate')
axs[0].legend(loc='upper right')
axs[1].plot(np.log10(N), times)
axs[1].set_title('CPU Time')
axs[1].set_xlabel('10^x')
axs[1].set_ylabel('Time (s)')
df = pd.DataFrame(index=N)
df.index.name = 'N'
df['option_price_estimates'] = option_price_estimates
df['cpu_time'] = times
return df
```