Project 6

By - Ameya Shete

In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
import bisect
```

Question 1

In [2]:
```
# Initialize parameters
s0, T, r, sigma, x_min, m, x = 50, 1, 0.05, 0.25, 30, 20, 2.5
n = 100
delta = T/n
N = 1000
```

In [3]:
```
# Generate standard normal variables
z = np.random.normal(0, 1, size=(N, n))
```

In [4]:
```
# Create stock matrix
stock_matrix = np.zeros((N, n+1))
stock_matrix[:, 0] = s0
for i in range(n):
    stock_matrix[:, i+1] = stock_matrix[:, i] * np.exp((r - (sigma**2)
```

In [5]:
```
# Calculate arithmetic and geometric averages for the stock prices
arithmetic_avg = np.array([np.mean(stock_matrix[row, :]) for row in ra
geometric_avg = np.array(np.exp([np.mean(np.log(stock_matrix[row, :]))
```

In [6]:
```
# Declare strikes array
strikes = [x_min + i*x for i in range(0, m+1)]
# Calculate Asian option price for arithmetic and geometric approaches
ac_value = np.array([np.mean(np.maximum(0, arithmetic_avg - i)) for i
gc_value = np.array([np.mean(np.maximum(0, geometric_avg - i)) for i i
ac_minus_gc = ac_value - gc_value
ac_gc_option = [np.mean(arithmetic_avg - geometric_avg)] * len(strikes
```

In [7]:
```
data = {'Strike Prices (X)': strikes, 'AC price': ac_value, 'GC price'
        'AC price - GC price': ac_minus_gc, 'A/G price': ac_gc_option}
output = pd.DataFrame(data=data)
output
```

Out[7]:

	Strike Prices (X)	AC price	GC price	AC price - GC price	A/G price
0	30.0	21.382553	21.120143	0.262410	0.26241
1	32.5	18.882553	18.620143	0.262410	0.26241
2	35.0	16.384747	16.123544	0.261203	0.26241
3	37.5	13.905391	13.651473	0.253918	0.26241
4	40.0	11.468385	11.223705	0.244680	0.26241
5	42.5	9.141832	8.914808	0.227024	0.26241
6	45.0	7.017208	6.809964	0.207244	0.26241
7	47.5	5.163589	4.978106	0.185483	0.26241
8	50.0	3.653983	3.487197	0.166786	0.26241
9	52.5	2.458818	2.313630	0.145188	0.26241
10	55.0	1.588841	1.469124	0.119717	0.26241
11	57.5	0.993635	0.895969	0.097666	0.26241
12	60.0	0.582149	0.505247	0.076902	0.26241
13	62.5	0.311049	0.257089	0.053959	0.26241
14	65.0	0.146911	0.110629	0.036282	0.26241
15	67.5	0.064425	0.047307	0.017119	0.26241
16	70.0	0.033763	0.026628	0.007136	0.26241
17	72.5	0.020925	0.016014	0.004911	0.26241
18	75.0	0.013091	0.008514	0.004576	0.26241
19	77.5	0.005591	0.001664	0.003927	0.26241
20	80.0	0.000293	0.000000	0.000293	0.26241

Question 2

In [33]:

```
v0, l0, mu, sigma, gamma, T, r0 = 20000, 22000, -0.1, 0.2, -0.4, 5, 0.
delta, alpha, eta = 0.25, 0.7, 0.95
```

In [119]:

```
def Proj6_2func(lambda_1, lambda_2, T):
    ### First simulate the jump times along each path
    n, paths = T*12, 1000
    h_matrix = []
    for i in range(paths):
        h = [np.random.exponential(1/(lambda_1*T))]
        while np.cumsum(h)[-1] <= T:
            h.append(np.random.exponential(1/(lambda_1*T)))
        h = np.cumsum(h)
        h_matrix.append(h[:-1])
    ### We will now simulate the V process
    z_matrix = np.random.normal(0, 1, size=(N, n))
    delta = T/n
    V_matrix = []
    for i in range(paths):
        V_path = [v0]
        h_path = h_matrix[i].tolist()
        time = np.arange(0, n+1) * delta
        time = time.tolist()
        time.extend(h_path)
        time = sorted(time)
        j = 0
        new_h_needed = True
        while j < n:
            if len(h_path)>0:
                if new_h_needed:
                    h = h_path.pop(0)
                if time[j] < h:
                    V_path.append(V_path[-1] + V_path[-1]*mu*delta + s
                    new_h_needed = False
                elif time[j] == h:
                    V_path.append(V_path[-1]*(1 + gamma))
                    new_h_needed = True
            elif len(h_path)==0:
                V_path.append(V_path[-1] + V_path[-1]*mu*delta + sigma
            j += 1
        V_matrix.append(V_path)
    # Here we will evaluate the Q and S values
    R = r0 + (delta*lambda_2)
    r = R/12
    pmt = (l0 * r)/(1 - (1/(1 +r)**n))
```

```
    a, b, c = pmt/r, pmt/(r * (1 + r)**n), 1+r
    Q_array = []
    payoff, sum_tau, final_payoff = 0, 0, 0
    S_array = np.random.exponential(1/(lambda_2*T), size=paths)
    for i in range(paths):
        for j in range(n):
            time = np.arange(0, n+1) * delta
            time = time.tolist()
            exp = S_array[i]
            index = bisect.bisect(time, exp)
            S = time[index-1]
            loan_value = a - b*c**(12 * delta * j)
            ltv = V_matrix[i][j]/loan_value
            qt = 0.7 + 0.05*(delta * j)
            if ltv <= qt:
                Q = delta * j
                break
            if j == n-1:
                Q = T+1
        tau = min(Q, S)
        sum_tau += tau
        no_default_count = 0
        if (Q > T) and (S > T):
            payoff += 0
            no_default_count += 1
        elif Q < S:
            payoff += np.maximum(0, loan_value - eta*V_matrix[i][j])
        elif Q >= S:
            loan_value = a - b*c**(12 * S)
            V = V_matrix[i][int(S/delta)]
            payoff = abs(loan_value - V)
        final_payoff += np.exp(-r0 * tau) * payoff
    price = final_payoff/paths
    default_prob = ((sum_tau)/(T*paths)) * 100
    expected_exercise_time = sum_tau/paths
    return {"Option price":price, "Default probability":default_prob,
            "Expected exercise time":expected_exercise_time}
```

In [120]:
```
Proj6_2func(0.2, 0.4, 5)
```

Out[120]:
```
{'Option price': 4994.899950696666,
 'Default probability': 6.456666666666666,
 'Expected exercise time': 0.3228333333333333}
```

a)

In [98]:
```
lambda_1_values = np.arange(0.05, 0.45, 0.05)
option_prices = [Proj6_2func(i, 0.4, 5)['Option price'] for i in lambdc
lambda_1_prices = dict(zip(lambda_1_values, option_prices))
lambda_1_prices
```

Out[98]:
```
{0.05: 2967.425883307333,
 0.1: 3691.271355026786,
 0.15000000000000002: 4216.255586093092,
 0.2: 4795.015012102379,
 0.25: 5455.847370898213,
 0.3: 5827.784624678648,
 0.35000000000000003: 7050.219163829208,
 0.4: 7350.540692557653}
```

In [99]:
```
lambda_2_values = np.arange(0.1, 0.9, 0.1)
option_prices = [Proj6_2func(0.2, i, 5)['Option price'] for i in lambdc
lambda_2_prices = dict(zip(lambda_2_values, option_prices))
lambda_2_prices
```

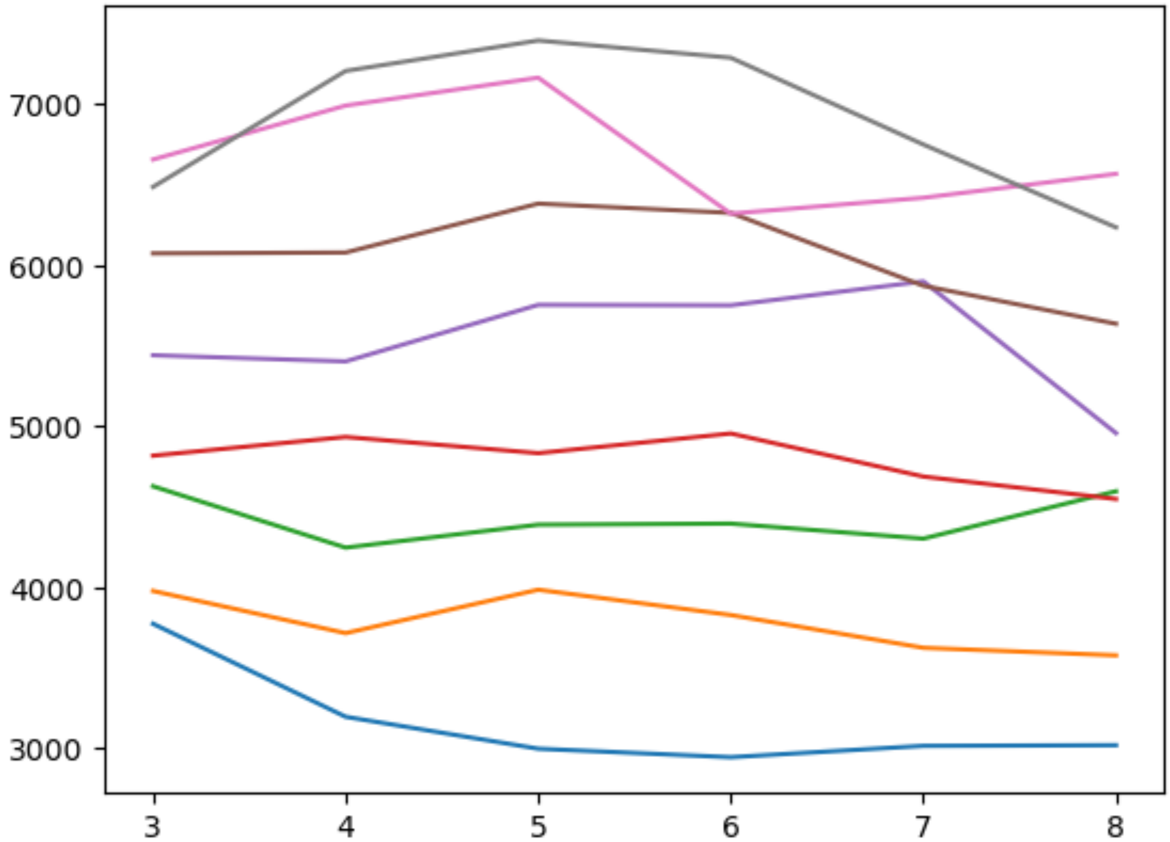
Out[99]:
```
{0.1: 16216.852265765208,
 0.2: 8393.20938349698,
 0.30000000000000004: 6258.249147231628,
 0.4: 4634.221104431177,
 0.5: 4443.419491287902,
 0.6: 4227.233242655481,
 0.7000000000000001: 3502.0256348678954,
 0.8: 3209.1763664705823}
```

In [100]:
```
T_values = np.arange(3, 9, 1)
option_prices = [Proj6_2func(0.2, 0.4, i)['Option price'] for i in T_v
T_prices = dict(zip(T_values, option_prices))
T_prices
```

Out[100]:
```
{3: 4811.403922374227,
 4: 5193.530494247647,
 5: 5249.995932340477,
 6: 5322.575510734497,
 7: 4915.91763586465,
 8: 4931.463802561944}
```

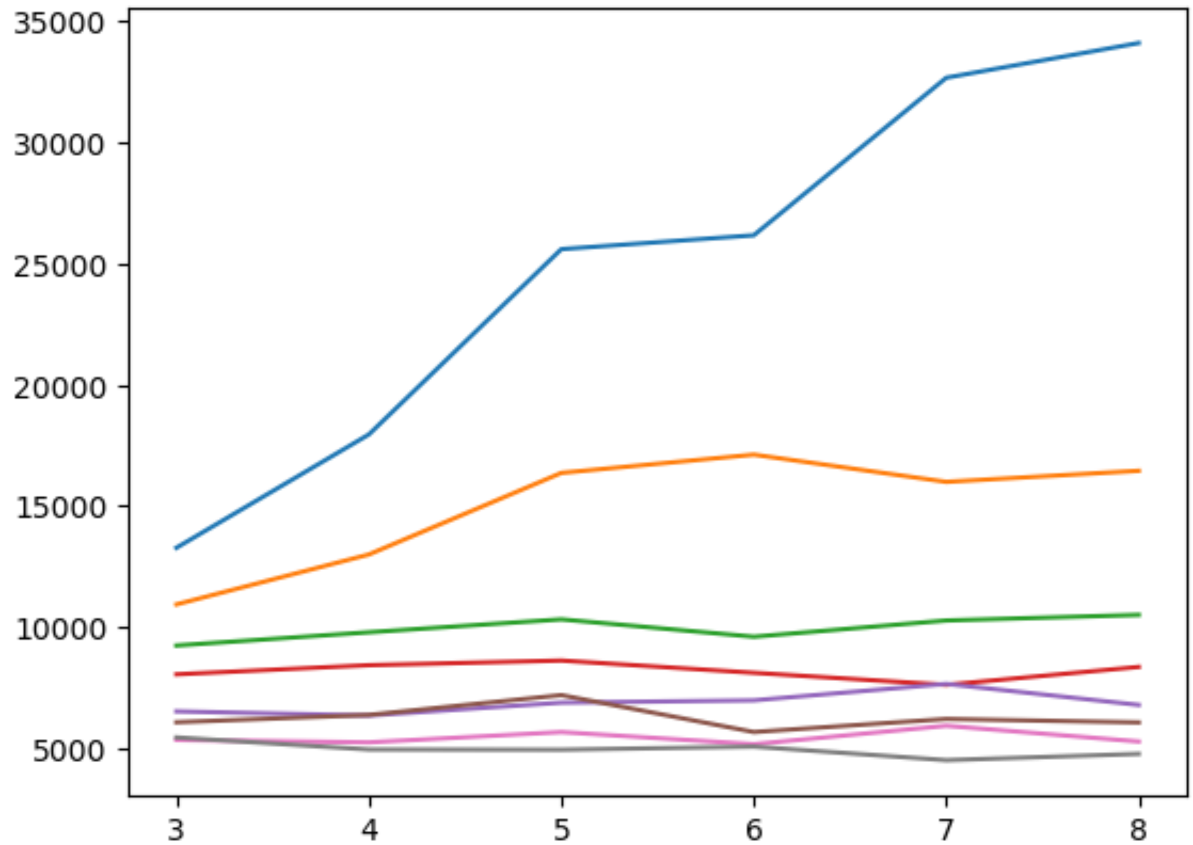
In [101]:
```
lambda_1_values = np.arange(0.05, 0.45, 0.05)
T_values = np.arange(3, 9, 1)
option_prices_lambda_1 = []
for i in lambda_1_values:
    temp_array = []
    for j in T_values:
        temp_array.append(Proj6_2func(i, 0.4, j)['Option price'])
    option_prices_lambda_1.append(temp_array)
```

In [102]:
```
for i in option_prices_lambda_1:
    plt.plot(T_values, i)
```



In [103]:
```
lambda_2_values = np.arange(0.1, 0.9, 0.1)
T_values = np.arange(3, 9, 1)
option_prices_lambda_2 = []
for i in lambda_1_values:
    temp_array = []
    for j in T_values:
        temp_array.append(Proj6_2func(0.2, i, j)['Option price'])
    option_prices_lambda_2.append(temp_array)
```

In [104]:
```
for i in option_prices_lambda_2:
    plt.plot(T_values, i)
```



b)

In [112]:
```
lambda_1_values = np.arange(0.05, 0.45, 0.05)
option_prices = [Proj6_2func(i, 0.4, 5)['Default probability'] for i i
lambda_1_prob = dict(zip(lambda_1_values, option_prices))
lambda_1_prob
```

Out[112]:
```
{0.05: 91.82166666666666,
 0.1: 92.95333333333332,
 0.15000000000000002: 93.23,
 0.2: 93.45666666666666,
 0.25: 93.48166666666667,
 0.3: 94.07666666666668,
 0.35000000000000003: 93.99666666666666,
 0.4: 94.53333333333333}
```

In [113]:
```
lambda_2_values = np.arange(0.1, 0.9, 0.1)
option_prices = [Proj6_2func(0.2, i, 5)['Default probability'] for i i
lambda_2_prob = dict(zip(lambda_2_values, option_prices))
lambda_2_prob
```

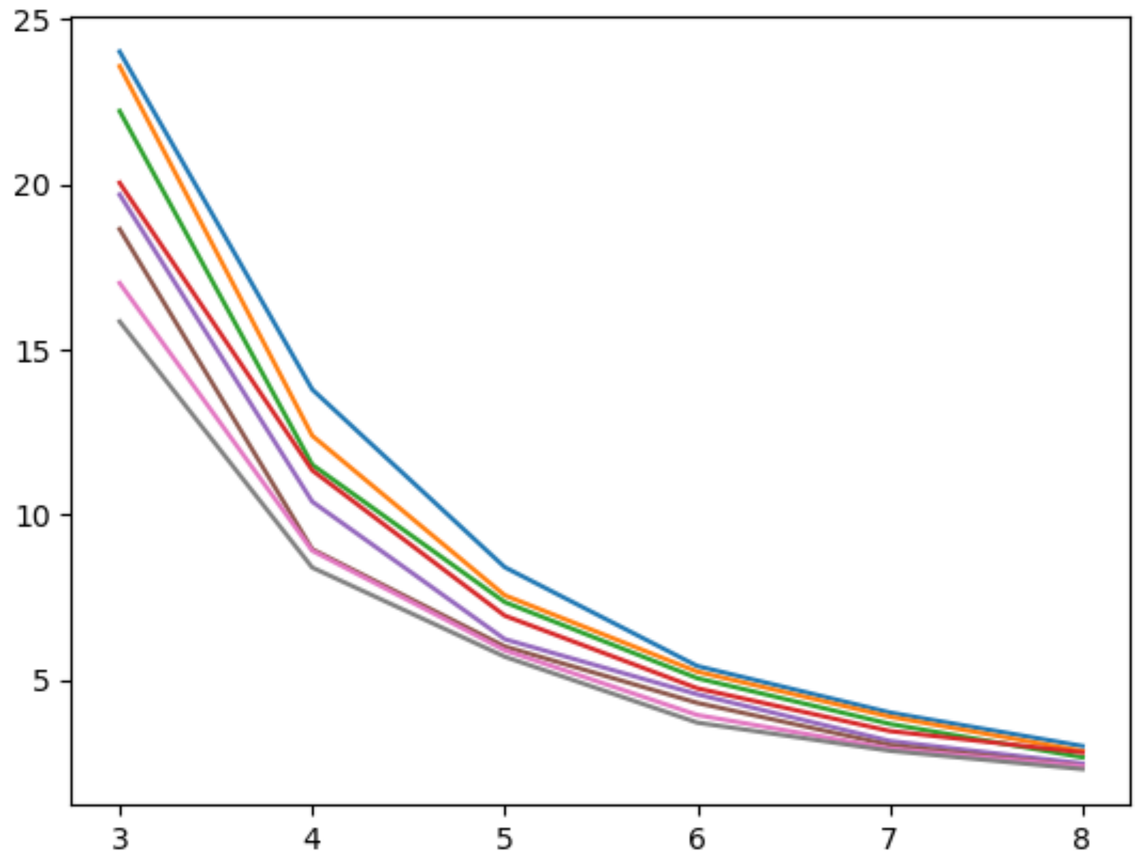
Out[113]:
```
{0.1: 85.075,
 0.2: 89.015,
 0.30000000000000004: 91.94,
 0.4: 93.14666666666666,
 0.5: 94.185,
 0.6: 95.03666666666666,
 0.7000000000000001: 95.84,
 0.8: 96.23333333333333}
```

In [121]:
```
T_values = np.arange(3, 9, 1)
option_prices = [Proj6_2func(0.2, 0.4, i)['Default probability'] for i
T_prob = dict(zip(T_values, option_prices))
T_prob
```

Out[121]:
```
{3: 20.08888888888886,
 4: 10.83333333333333,
 5: 6.325000000000003,
 6: 4.669444444444442,
 7: 3.399999999999994,
 8: 2.548958333333363}
```

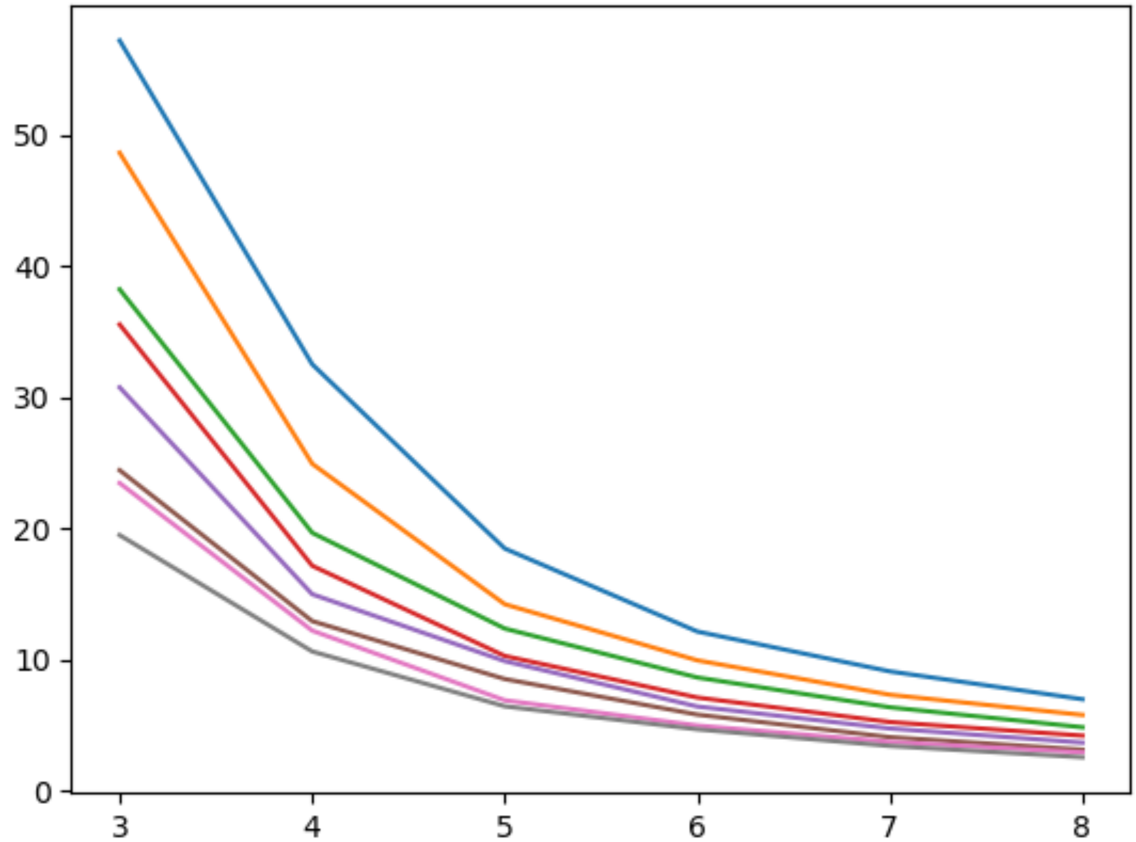
In [122]:
```
lambda_1_values = np.arange(0.05, 0.45, 0.05)
T_values = np.arange(3, 9, 1)
prob_lambda_1 = []
for i in lambda_1_values:
    temp_array = []
    for j in T_values:
        temp_array.append(Proj6_2func(i, 0.4, j)['Default probability'
    prob_lambda_1.append(temp_array)
```

In [123]:
```
for i in prob_lambda_1:
    plt.plot(T_values, i)
```



In [124]:
```
lambda_2_values = np.arange(0.1, 0.9, 0.1)
T_values = np.arange(3, 9, 1)
prob_lambda_2 = []
for i in lambda_1_values:
    temp_array = []
    for j in T_values:
        temp_array.append(Proj6_2func(0.2, i, j)['Default probability'
    prob_lambda_2.append(temp_array)
```

In [125]:
```
for i in prob_lambda_2:
    plt.plot(T_values, i)
```



c)

In [65]:
```
lambda_1_values = np.arange(0.05, 0.45, 0.05)
option_prices = [Proj6_2func(i, 0.4, 5)['Expected exercise time'] for
lambda_1_exercise_time = dict(zip(lambda_1_values, option_prices))
lambda_1_exercise_time
```

Out[65]:
```
{0.05: 0.4090833333333324,
 0.1: 0.3784166666666667,
 0.15000000000000002: 0.3404166666666661,
 0.2: 0.3508333333333328,
 0.25: 0.3040833333333332,
 0.3: 0.2924166666666668,
 0.35000000000000003: 0.2890833333333333,
 0.4: 0.2771666666666656}
```

In [66]:
```
lambda_2_values = np.arange(0.1, 0.9, 0.1)
option_prices = [Proj6_2func(0.2, i, 5)['Expected exercise time'] for
lambda_2_exercise_time = dict(zip(lambda_2_values, option_prices))
lambda_2_exercise_time
```

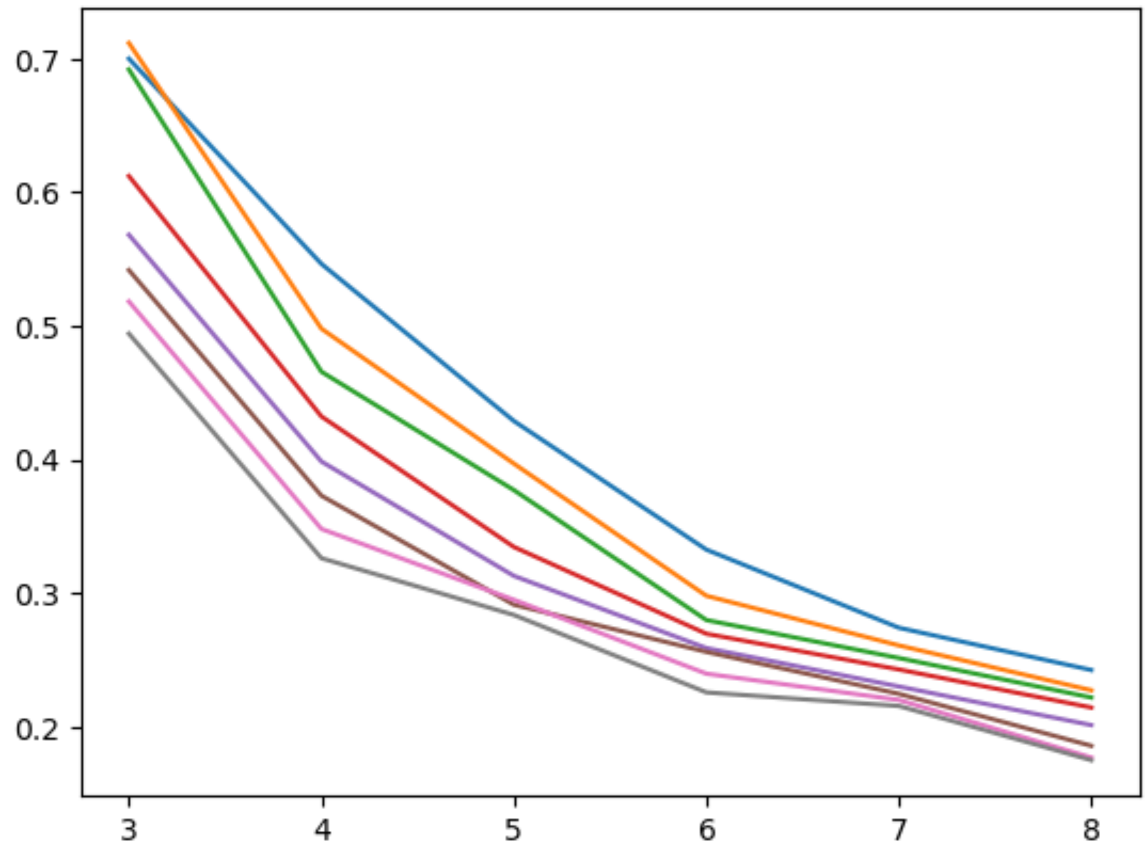
Out[66]:
```
{0.1: 0.6920833333333332,
 0.2: 0.5125833333333332,
 0.30000000000000004: 0.3892500000000004,
 0.4: 0.3480833333333332,
 0.5: 0.2785833333333357,
 0.6: 0.2372499999999999,
 0.7000000000000001: 0.20641666666666703,
 0.8: 0.18916666666666682}
```

In [67]:
```
T_values = np.arange(3, 9, 1)
option_prices = [Proj6_2func(0.2, 0.4, i)['Expected exercise time'] fo
T_exercise_time = dict(zip(T_values, option_prices))
T_exercise_time
```

Out[67]:
```
{3: 0.6401666666666666,
 4: 0.40391666666666726,
 5: 0.3122499999999998,
 6: 0.2576666666666668,
 7: 0.23508333333333337,
 8: 0.2215}
```

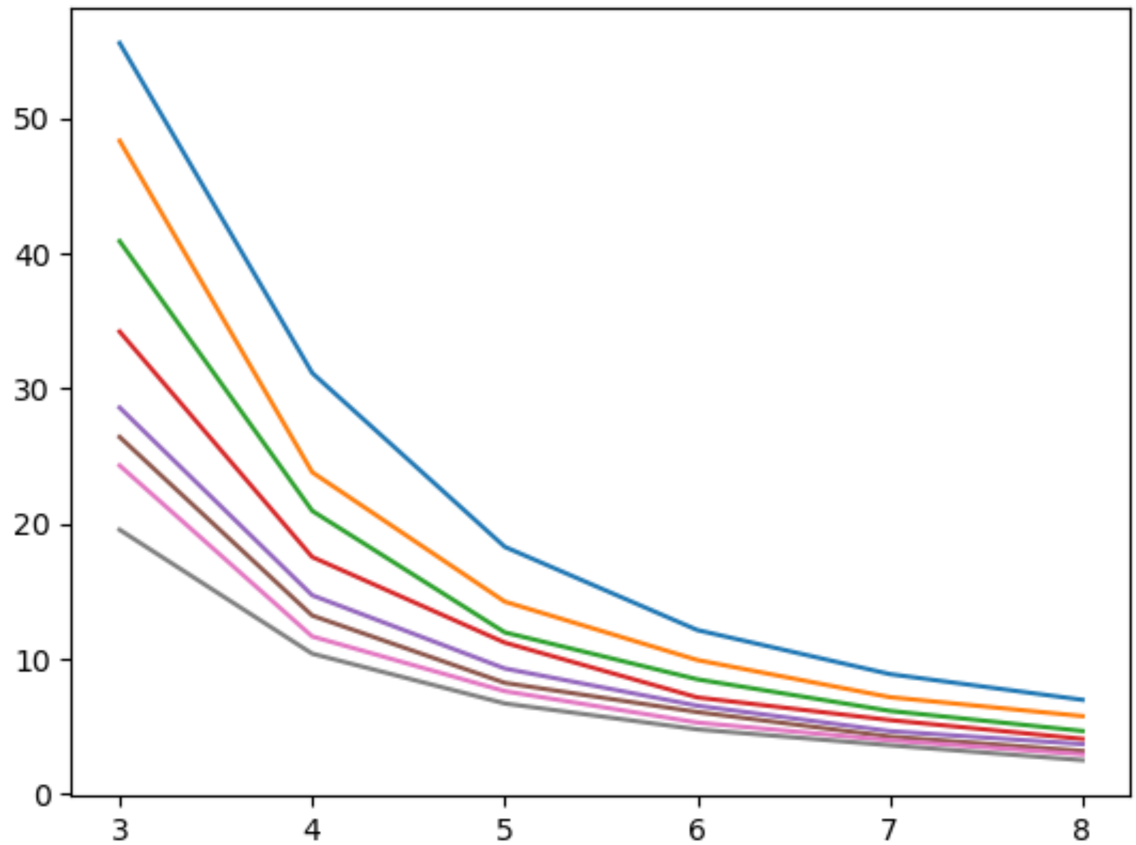
In [82]:
```
lambda_1_values = np.arange(0.05, 0.45, 0.05)
T_values = np.arange(3, 9, 1)
exercise_time_lambda_1 = []
for i in lambda_1_values:
    temp_array = []
    for j in T_values:
        temp_array.append(Proj6_2func(i, 0.4, j)['Expected exercise ti
    exercise_time_lambda_1.append(temp_array)
```

In [83]:
```
for i in exercise_time_lambda_1:
    plt.plot(T_values, i)
```



In [84]:
```
lambda_2_values = np.arange(0.1, 0.9, 0.1)
T_values = np.arange(3, 9, 1)
exercise_time_lambda_2 = []
for i in lambda_1_values:
    temp_array = []
    for j in T_values:
        temp_array.append(Proj6_2func(0.2, i, j)['Default probability'
    exercise_time_lambda_2.append(temp_array)
```

In [85]:
```
for i in exercise_time_lambda_2:
    plt.plot(T_values, i)
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: