Project 7

By - Ameya Shete

In [825]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
```

In [228]:
```
from scipy.stats import norm
def black_scholes(S, K, t, r, sigma):
    d1 = (np.log((S/K))  + ((r + (sigma**2)/2)*t))/(sigma * np.sqrt(t)
    d2 = d1 - (sigma * np.sqrt(t))
    put_price = K*norm.cdf(-1*d2)*np.exp(-r*t) - S*norm.cdf(-1*d1)
    return put_price
```

Question 1

In [620]:
```
def european_put_numerical_solver(stock_range, delta_t, sigma, K, T, F
    '''
    Inputs:
        stock_range: The range over which we want to evaluate the put
        delta_t: Time step
        sigma: volatility
        K: strike
        T: maturity
        FD_method: Explicit Finite Difference, Implicit Finite Differe
        delta_X: The increments for the stock range
        r: annual interest rate
    Output:
        F: The put prices for all the stock prices in the stock range
    '''

    time = np.linspace(0, T, int(T/delta_t))
    if FD_method == 'EFD':
        S_T = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_ra
        payoffs = [np.maximum(0, K - i) for i in S_T]
        N = len(S_T)

        S_second_to_last = S_T[-2]
        S_last = S_T[-1]

        pu = delta_t*(((sigma**2)/(2 * delta_X**2)) + (r - 0.5*sigma**
        pd = delta_t*(((sigma**2)/(2 * delta_X**2)) - (r - 0.5*sigma**
```

```
        pm = 1 - (delta_t*(sigma**2)/delta_X**2) - (r * delta_t)

        A = np.zeros((N-2, N))
        j = 0
        for i in range(N-2):
            if i == j:
                A[i][j], A[i][j+1], A[i][j+2] = pu, pm, pd
            j += 1
        A = A.tolist()

        A.append(A[-1])
        A.insert(0, A[0])

        F = [payoffs]
        for i in np.arange(0, int(T/delta_t)-1)[::-1]:
            next_F = F.pop()
            B = np.zeros(N)
            B[-1] = S_last - S_second_to_last
            F = [np.subtract(np.dot(A, next_F), B)]
        return F[0]

    elif FD_method == 'IFD':
        S_T = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_ra
        payoffs = [np.maximum(0, K - i) for i in S_T]
        N = len(S_T)

        S_second_to_last = S_T[-2]
        S_last = S_T[-1]

        pu = -0.5*delta_t*(((sigma**2)/delta_X**2) + ((r - 0.5*sigma**
        pd = -0.5*delta_t*(((sigma**2)/delta_X**2) - ((r - 0.5*sigma**
        pm = 1 + (delta_t*(sigma**2)/delta_X**2) + (r * delta_t)

        A = np.zeros((N-2, N))
        j = 0
        for i in range(N-2):
            if i == j:
                A[i][j], A[i][j+1], A[i][j+2] = pu, pm, pd
            j += 1
        A = A.tolist()

        A_start, A_end = A[0].copy(), A[-1].copy()
        A_start[0], A_start[1], A_start[2] = 1, -1, 0
        A_end[-1], A_end[-2], A_end[-3] = -1, 1, 0
        A.insert(0, A_start)
        A.append(A_end)
        A_inverse = np.linalg.inv(A)

        B = payoffs
        B[0], B[-1] = 0, S_last - S_second_to_last
```

```
        for i in np.arange(0, int(T/delta_t)-1)[::-1]:
            F = np.dot(A_inverse, B)
            B = F
            B[0], B[-1] = 0, S_last - S_second_to_last
        return F[1:-1]
    elif FD_method == 'CNFD':

        S_T = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_ra
        payoffs = [np.maximum(0, K - i) for i in S_T]
        N = len(S_T)

        S_second_to_last = S_T[-2]
        S_last = S_T[-1]

#         pu_1 = (-0.25*delta_t*(((sigma**2)/delta_X**2) + ((r - 0.5*s
#         pd_1 = (-0.25*delta_t*(((sigma**2)/delta_X**2) - ((r - 0.5*s
#         pm_1 = (1 + (delta_t*(sigma**2)/2*delta_X**2) + (r * delta_t
        pu = -1/4* delta_t*(sigma**2 / delta_X**2 + (r - 0.5*sigma**2)
        pm = 1 + delta_t*sigma**2/(2*delta_X**2) + 0.5*r*delta_t
        pd =  -1/4* delta_t*(sigma**2 / delta_X**2 - (r - 0.5*sigma**2

        A = np.zeros((N-2, N))
        j = 0
        for i in range(N-2):
            if i == j:
                A[i][j], A[i][j+1], A[i][j+2] = pu, pm, pd
            j += 1
        A = A.tolist()

        A_start, A_end = A[0].copy(), A[-1].copy()
        A_start[0], A_start[1], A_start[2] = 1, -1, 0
        A_end[-1], A_end[-2], A_end[-3] = -1, 1, 0
        A.insert(0, A_start)
        A.append(A_end)
        A_inverse = np.linalg.inv(A)

        B, j, prob = [], 0, np.array([-pu, 2-pm, -pd])
        while j <= len(payoffs)-3:
            temp = np.dot(payoffs[j:j+3], prob.T)
            B.append(temp)
            j += 1
        B.insert(0, 0)
        B.append(S_second_to_last - S_last)

        i = int(T/delta_t)-1
        while i > 0:
            F = np.dot(A_inverse, B)
            j =  0
            B = []
            # Improvement: You can slice through the entire array and
            # Zi = -Pu*F[:-2] - (Pm - 2) * F[1:-1] - Pd*F[2:]
```

```
            while j <= len(F)-3:
                temp = np.dot(F[j:j+3], prob.T)
                B.append(temp)
                j += 1
            B.insert(0, 0)
            B.append(S_second_to_last - S_last)
            i -= 1
        return F
```

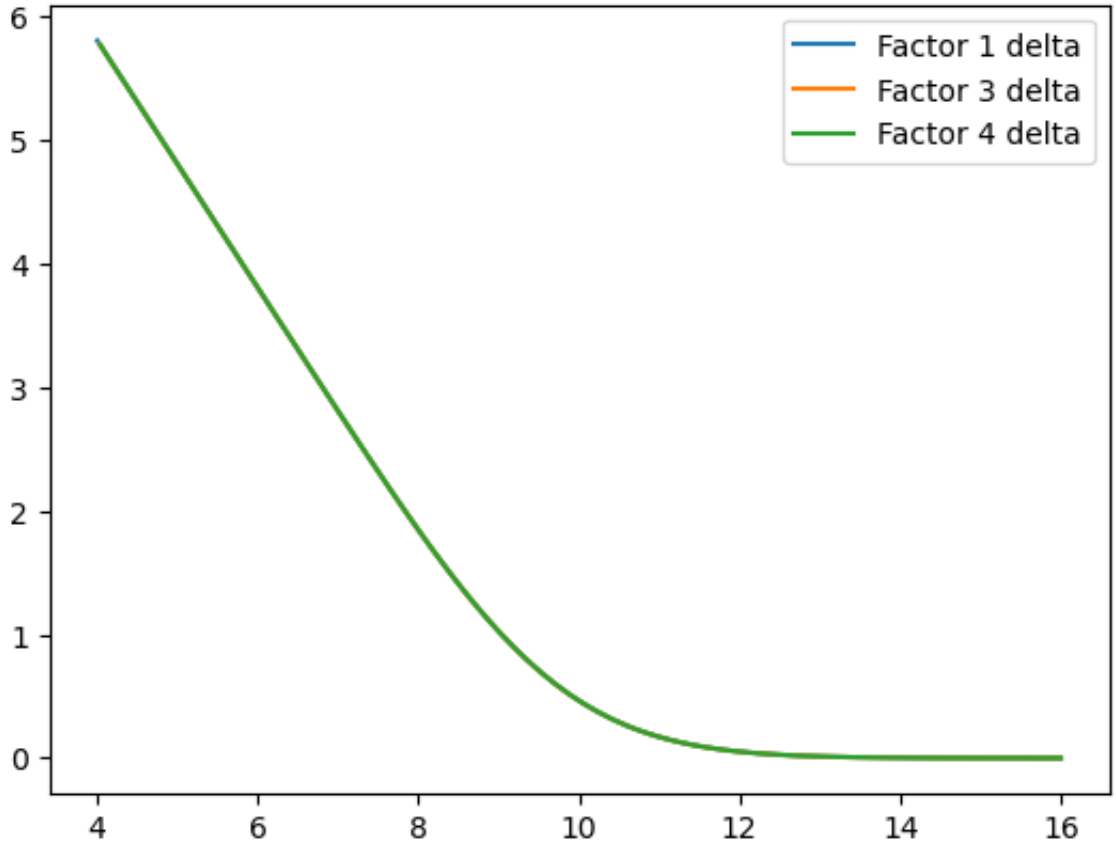
a) Explicit Finite Difference

In [610]:
```
stock_range, sigma, delta_t, K, T, FD_method, r = (4, 16), 0.2, 0.002,
factors = [1, 3, 4]
```

In [611]:
```
factor_1 = european_put_numerical_solver(stock_range, delta_t, sigma,
factor_3 = european_put_numerical_solver(stock_range, delta_t, sigma,
factor_4 = european_put_numerical_solver(stock_range, delta_t, sigma,
```

In [612]:
```
elta_X_1, delta_X_2, delta_X_3 = sigma*np.sqrt(factors[0]*0.002), sigm
_T_1 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]-
_T_3 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]-
_T_4 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]-
lt.plot(S_T_1, factor_1, label='Factor 1 delta')
lt.plot(S_T_3, factor_3, label='Factor 3 delta')
lt.plot(S_T_4, factor_4, label='Factor 4 delta')
lt.legend()
```

Out[612]: <matplotlib.legend.Legend at 0x7fe69411c150>



In [828]:
```
bs_prices = [black_scholes(i, 10, 0.5, 0.04, 0.2) for i in range(4, 17
arr_1 = np.asarray(factor_1)
arr_2 = np.asarray(factor_3)
arr_3 = np.asarray(factor_4)
i_1 = [(np.abs(arr_1 - bs_prices[i])).argmin() for i in range(len(bs_p
i_2 = [(np.abs(arr_2 - bs_prices[i])).argmin() for i in range(len(bs_p
i_3 = [(np.abs(arr_3 - bs_prices[i])).argmin() for i in range(len(bs_p
compare_1 = factor_1[i_1]
compare_2 = factor_3[i_2]
compare_3 = factor_4[i_3]

error_df = pd.DataFrame(data={'Stock Price': np.arange(4, 17), 'Factor
                              'Factor 3': compare_3 - bs_prices})
error_df
```

Out[828]:

	Stock Price	Factor 1	Factor 2	Factor 3
0	4	-0.720919	-0.772581	-0.784321
1	5	-0.015207	-0.006156	0.002102
2	6	0.017717	-0.029722	0.017146
3	7	-0.025663	-0.038127	-0.025587
4	8	-0.031991	0.029814	0.033640
5	9	-0.018730	-0.013847	-0.018735
6	10	0.016283	-0.022325	-0.021073
7	11	0.001360	-0.006542	0.001240
8	12	-0.001478	0.005269	-0.001428
9	13	-0.000582	-0.001525	0.001837
10	14	0.000019	0.000435	-0.000478
11	15	-0.000005	0.000010	-0.000082
12	16	0.000156	0.000198	0.000216

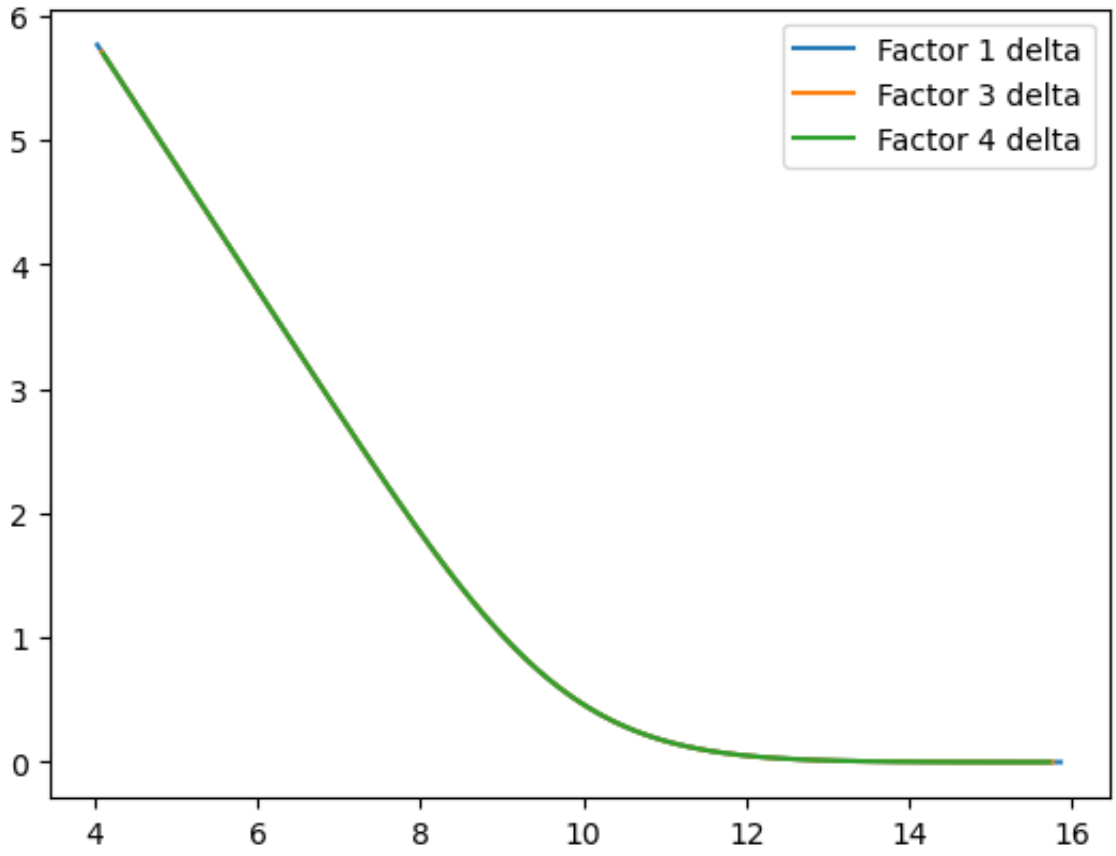
b) Implicit Finite Difference

In [829]:
```
stock_range, sigma, delta_t, K, T, FD_method, r = (4, 16), 0.2, 0.002,
factors = [1, 3, 4]
```

In [830]:
```
factor_1 = european_put_numerical_solver(stock_range, delta_t, sigma,
factor_3 = european_put_numerical_solver(stock_range, delta_t, sigma,
factor_4 = european_put_numerical_solver(stock_range, delta_t, sigma,
```

In [831]:
```
delta_X_1, delta_X_2, delta_X_3 = sigma*np.sqrt(factors[0]*0.002), sig
S_T_1 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]
S_T_3 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]
S_T_4 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]
plt.plot(S_T_1[1:-1], factor_1, label='Factor 1 delta')
plt.plot(S_T_3[1:-1], factor_3, label='Factor 3 delta')
plt.plot(S_T_4[1:-1], factor_4, label='Factor 4 delta')
plt.legend()
```

Out[831]: <matplotlib.legend.Legend at 0x7fe6727f78d0>



In [832]:
```
prices = [black_scholes(i, 10, 0.5, 0.04, 0.2) for i in range(4, 17)]
_1 = np.asarray(factor_1)
_2 = np.asarray(factor_3)
_3 = np.asarray(factor_4)
= [(np.abs(arr_1 - bs_prices[i])).argmin() for i in range(len(bs_price
= [(np.abs(arr_2 - bs_prices[i])).argmin() for i in range(len(bs_price
= [(np.abs(arr_3 - bs_prices[i])).argmin() for i in range(len(bs_price
pare_1 = factor_1[i_1]
pare_2 = factor_3[i_2]
pare_3 = factor_4[i_3]

r_df = pd.DataFrame(data={'Stock Price': np.arange(4, 17), 'Factor 1':
                          'Factor 3': compare_3 - bs_prices})
r_df
```

Out[832]:

	Stock Price	Factor 1	Factor 2	Factor 3
0	4	-0.034873	-0.092294	-0.107717
1	5	-0.001187	-0.005494	-0.001190
2	6	0.019008	-0.028283	0.019011
3	7	-0.025555	-0.038011	-0.025469
4	8	-0.031816	0.029992	0.033818
5	9	-0.018790	-0.013903	-0.018793
6	10	0.016005	-0.022606	-0.021355
7	11	0.001201	-0.006691	0.001081
8	12	-0.001445	0.005290	-0.001394
9	13	-0.000500	-0.001444	0.001919
10	14	0.000074	0.000492	-0.000427
11	15	0.000020	0.000036	-0.000058
12	16	0.000173	0.000217	0.000236

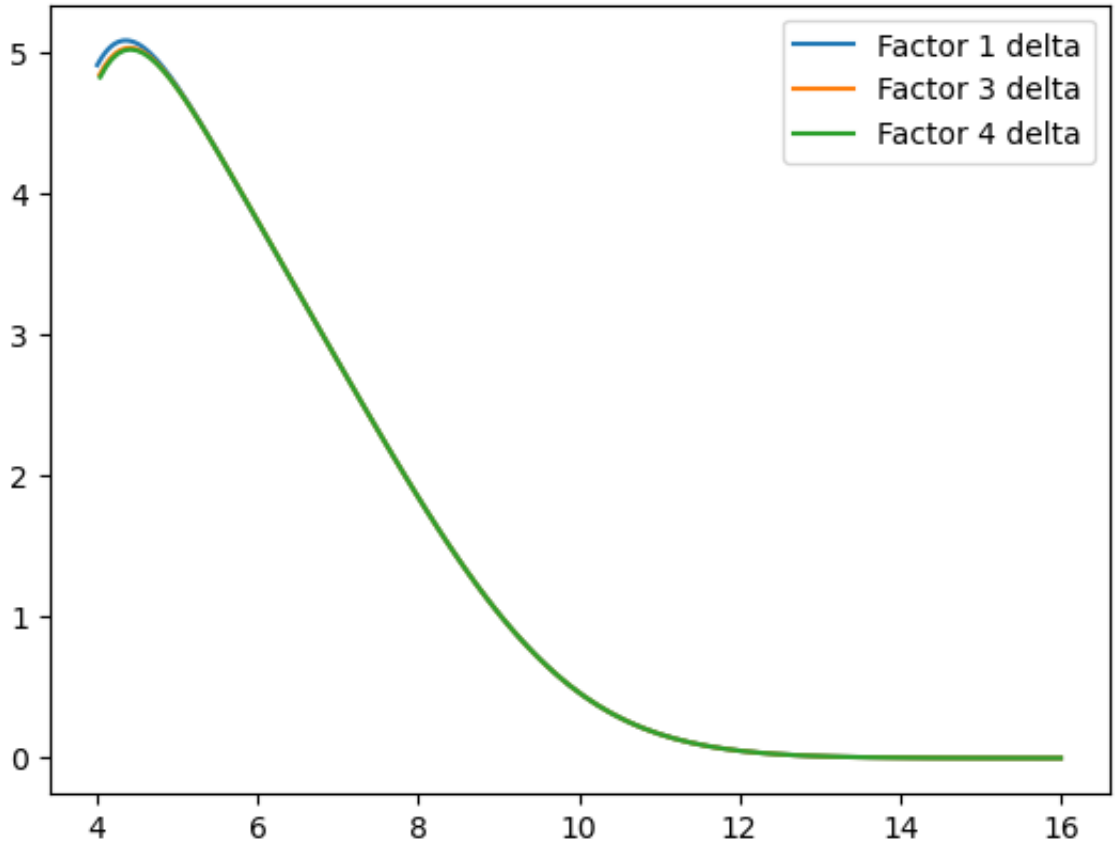
c) Crank Nicholson Finite Difference

In [833]:
```
stock_range, sigma, delta_t, K, T, FD_method, r = (4, 16), 0.2, 0.002,
factors = [1, 3, 4]
```

In [834]:
```
actor_1 = european_put_numerical_solver(stock_range, delta_t, sigma, K,
actor_3 = european_put_numerical_solver(stock_range, delta_t, sigma, K,
actor_4 = european_put_numerical_solver(stock_range, delta_t, sigma, K,
```

In [835]:
```
delta_X_1, delta_X_2, delta_X_3 = sigma*np.sqrt(factors[0]*0.002), sig
S_T_1 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]
S_T_3 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]
S_T_4 = np.exp(np.arange(np.log(stock_range[1]), np.log(stock_range[0]
plt.plot(S_T_1, factor_1, label='Factor 1 delta')
plt.plot(S_T_3, factor_3, label='Factor 3 delta')
plt.plot(S_T_4, factor_4, label='Factor 4 delta')
plt.legend()
```

Out[835]: <matplotlib.legend.Legend at 0x7fe6c7f05b10>



In [836]:
```
bs_prices = [black_scholes(i, 10, 0.5, 0.04, 0.2) for i in range(4, 17
arr_1 = np.asarray(factor_1)
arr_2 = np.asarray(factor_3)
arr_3 = np.asarray(factor_4)
i_1 = [(np.abs(arr_1 - bs_prices[i])).argmin() for i in range(len(bs_p
i_2 = [(np.abs(arr_2 - bs_prices[i])).argmin() for i in range(len(bs_p
i_3 = [(np.abs(arr_3 - bs_prices[i])).argmin() for i in range(len(bs_p
compare_1 = factor_1[i_1]
compare_2 = factor_3[i_2]
compare_3 = factor_4[i_3]

error_df = pd.DataFrame(data={'Stock Price': np.arange(4, 17), 'Factor
                              'Factor 3': compare_3 - bs_prices})
error_df
```

Out[836]:

	Stock Price	Factor 1	Factor 2	Factor 3
0	4	-0.720919	-0.772581	-0.784321
1	5	-0.015207	-0.006156	0.002102
2	6	0.017717	-0.029722	0.017146
3	7	-0.025663	-0.038127	-0.025587
4	8	-0.031991	0.029814	0.033640
5	9	-0.018730	-0.013847	-0.018735
6	10	0.016283	-0.022325	-0.021073
7	11	0.001360	-0.006542	0.001240
8	12	-0.001478	0.005269	-0.001428
9	13	-0.000582	-0.001525	0.001837
10	14	0.000019	0.000435	-0.000478
11	15	-0.000005	0.000010	-0.000082
12	16	0.000156	0.000198	0.000216

Question 2

In [803]:
```
def american_option_numerical_solver(stock_range, delta_t, sigma, K, T
    '''
    Inputs:
        stock_range: The range over which we want to evaluate the put
        delta_t: Time step
        sigma: volatility
        K: strike
        T: maturity
```

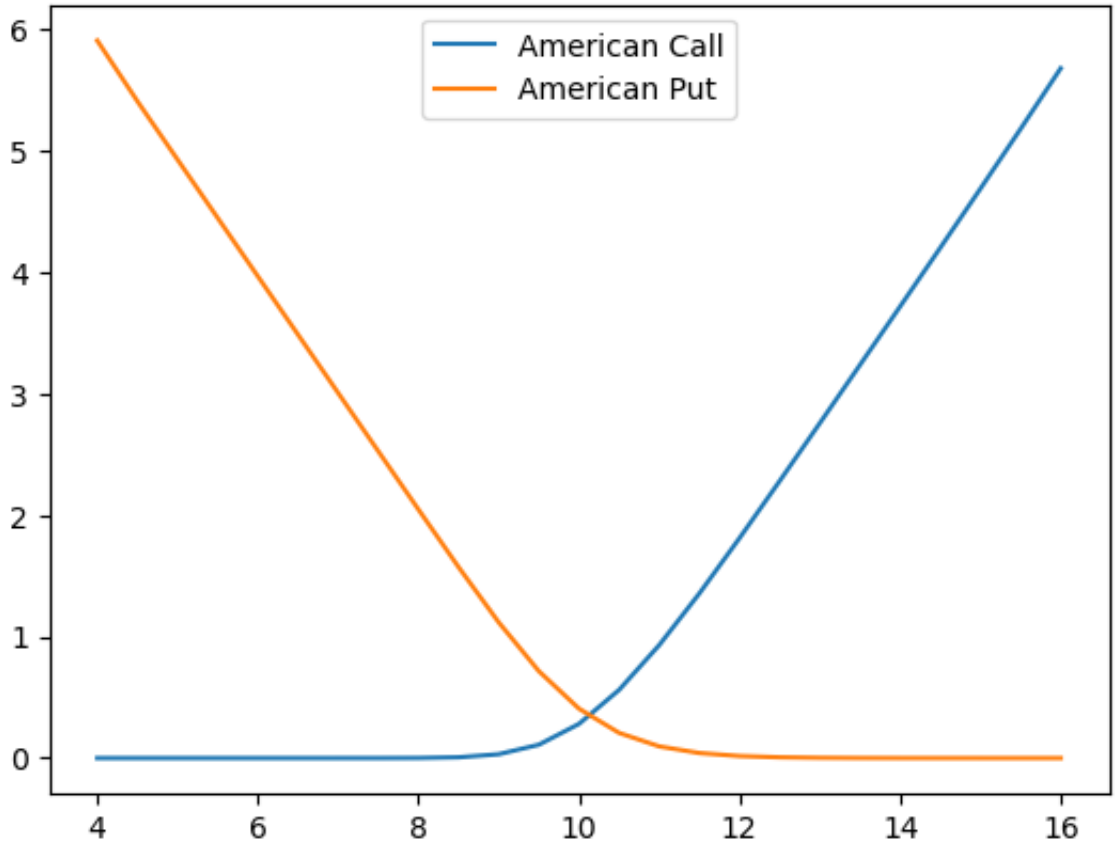
```
    FD_method: Explicit Finite Difference, Implicit Finite Differe
    call_put_flag: 'C' or 'P'
    r: annual interest rate
Output:
    F: The put prices for all the stock prices in the stock range
'''
if FD_method == 'EFD':
    alpha = 1
elif FD_method == 'IFD':
    alpha = 0
elif FD_method == 'CNFD':
    alpha = 0.5

S_T = np.arange(stock_range[1], stock_range[0]-0.5, -0.5)

if call_put_flag == 'C':
    payoffs = [np.maximum(0, i - K) for i in S_T]
    S_second = S_T[1]
    S_first = S_T[0]
elif call_put_flag == 'P':
    payoffs = [np.maximum(0, K - i) for i in S_T]
    S_second_to_last = S_T[-2]
    S_last = S_T[-1]

N = len(S_T)

A, B = np.zeros((N-2, N)), []
j = 0
for i in range(N-2):
    if i == j:
        a1 = ((sigma**2 * (N-1-j)**2 - r * (N-1-j))*(1 - alpha))/2
        a2 = -(1/delta_t) - (sigma**2 * (N-1-j)**2 + r)*(1 - alpha
        a3 = ((sigma**2 * (N-1-j)**2 + r * (N-1-j))*(1 - alpha))/2

        b1 = ((sigma**2 * (N-1-j)**2 - (r * (N-1-j)))*alpha)/2
        b2 = (1/delta_t) - (sigma**2 * (N-1-j)**2 + r)*alpha
        b3 = ((sigma**2 * (N-1-j)**2 + (r * (N-1-j)))*alpha)/2
        B_temp = [-b1, -b2, -b3]
        B.append(B_temp)

        A[i][j] = a1
        A[i][j+1] = a2
        A[i][j+2] = a3
    j += 1
A = A.tolist()


A_start, A_end = A[0].copy(), A[-1].copy()
A_start[0], A_start[1], A_start[2] = 1, -1, 0
A_end[-1], A_end[-2], A_end[-3] = -1, 1, 0
```

```
    A.insert(0, A_start)
    A.append(A_end)
    A_inverse = np.linalg.inv(A)

    j, Z = 0, []
    while j <= len(payoffs)-3:
        temp = np.dot(payoffs[j:j+3], np.array(B[j]).T)
        Z.append(temp)
        j += 1
    if call_put_flag == 'P':
        Z.insert(0, 0)
        Z.append(S_last - S_second_to_last)
    elif call_put_flag == 'C':
        Z.insert(0, S_first - S_second)
        Z.append(0)


    i = int(T/delta_t)-1
    while i > 0:
        F = np.dot(A_inverse, Z)
        j =  0
        Z = []
        while j <= len(F)-3:
            temp = np.dot(F[j:j+3], np.array(B[j]).T)
            Z.append(temp)
            j += 1
        if call_put_flag == 'P':
            Z.insert(0, 0)
            Z.append(S_last - S_second_to_last)
        elif call_put_flag == 'C':
            Z.insert(0, S_first - S_second)
            Z.append(0)
#         Z = [np.maximum(payoffs[i], Z[i]) for i in range(len(Z))]
        i -= 1
    return F
```

a) Explicit Finite Difference

In [804]:
```
F_call = american_option_numerical_solver((4, 16), 0.002, 0.2, 10, 0.5
F_put = american_option_numerical_solver((4, 16), 0.002, 0.2, 10, 0.5,
```

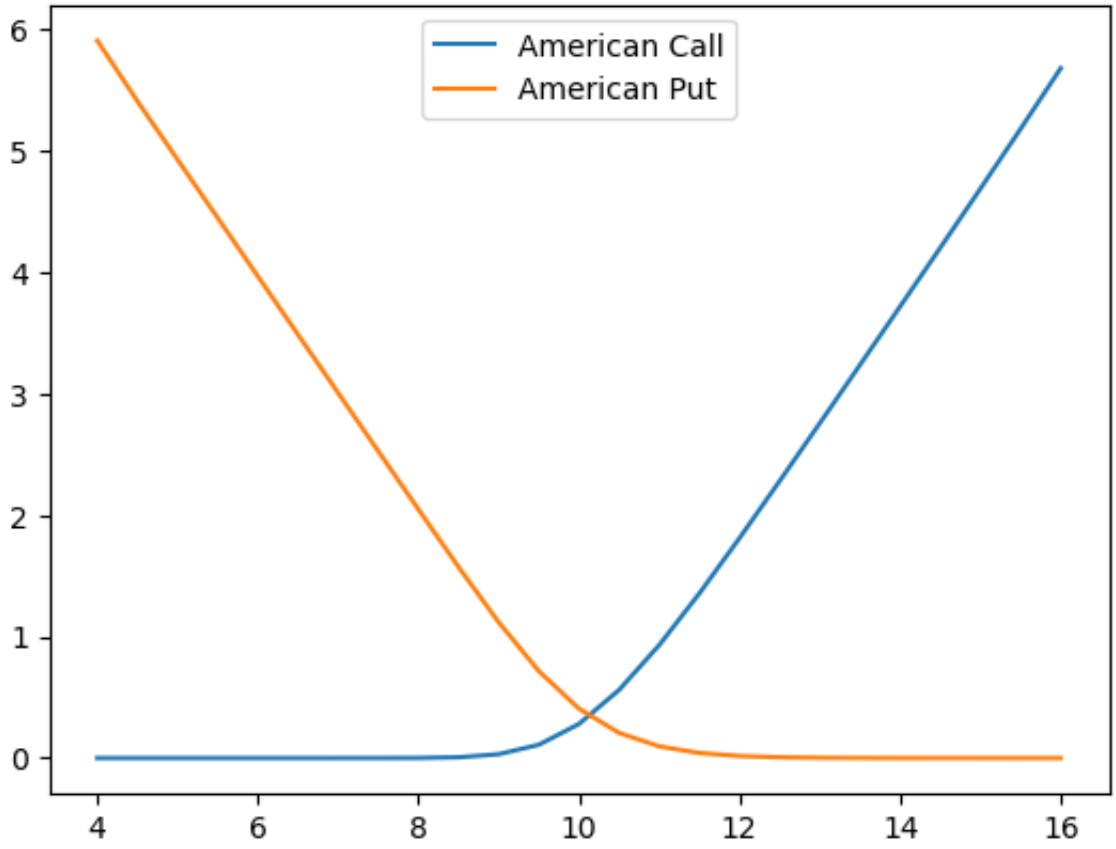
In [805]:
```
S_T = np.arange(16, 4-0.5, -0.5)
plt.plot(S_T, F_call, label='American Call')
plt.plot(S_T, F_put, label='American Put')
plt.legend()
plt.show()
```



b) Implicit Finite Difference

In [806]:
```
F_call = american_option_numerical_solver((4, 16), 0.002, 0.2, 10, 0.5
F_put = american_option_numerical_solver((4, 16), 0.002, 0.2, 10, 0.5,
```

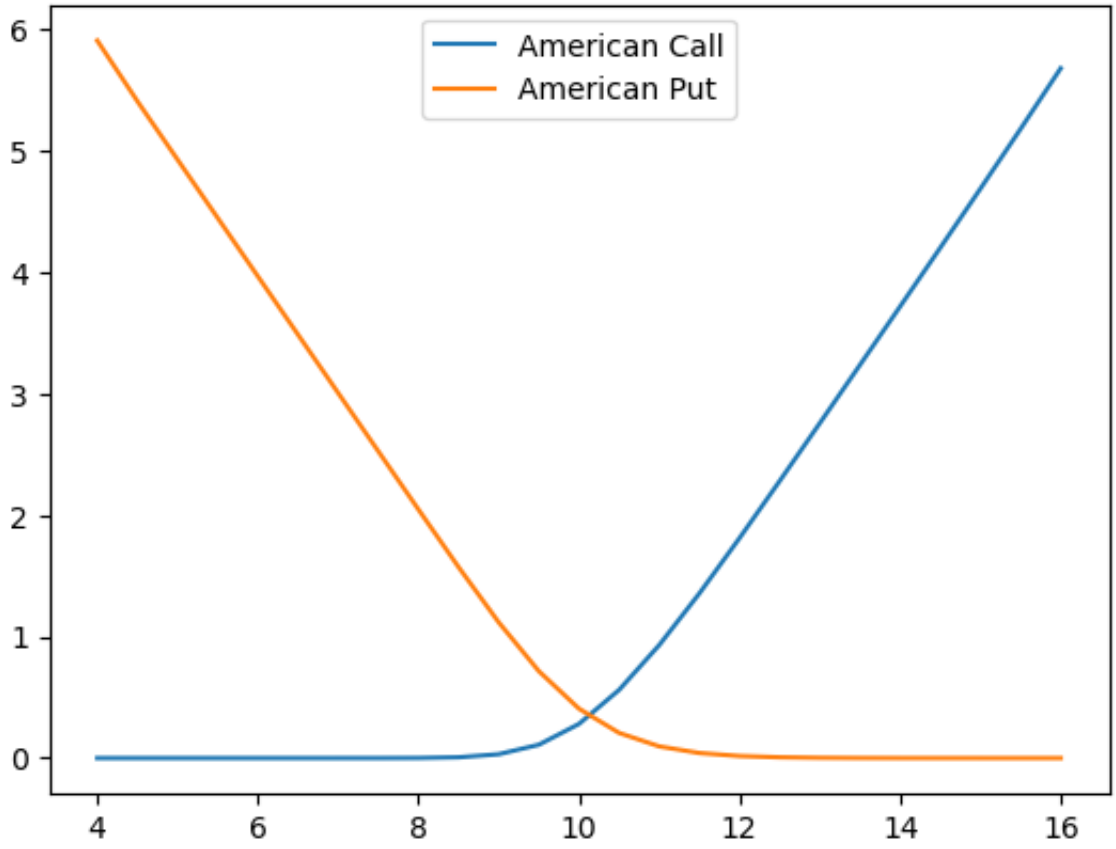
In [807]:
```
S_T = np.arange(16, 4-0.5, -0.5)
plt.plot(S_T, F_call, label='American Call')
plt.plot(S_T, F_put, label='American Put')
plt.legend()
plt.show()
```



c) Crank-Nicholson Finite Difference

In [808]:
```
F_call_CNFD = american_option_numerical_solver((4, 16), 0.002, 0.2, 10
F_put_CNFD = american_option_numerical_solver((4, 16), 0.002, 0.2, 10,
```

In [809]:
```
S_T = np.arange(16, 4-0.5, -0.5)
plt.plot(S_T, F_call_CNFD, label='American Call')
plt.plot(S_T, F_put_CNFD, label='American Put')
plt.legend()
plt.show()
```



In [ ]:
```

```