

Homework 2

You Wang

2021-03-12

Problem 1

(a)

For binomial tree model, we set $S = e^x$, $S_u = e^{x+\Delta X_u}$, and $S_d = e^{x+\Delta X_d}$. Assume $\Delta x_u = \Delta x_d$, Δx and probability of up move p_u is given by:

$$\begin{cases} \Delta x = \sqrt{\left(r - \frac{\sigma^2}{2}\right)^2 \Delta t^2 + \sigma^2 \Delta t} \\ p_u = \frac{1}{2} + \frac{1}{2} \frac{\left(r - \frac{\sigma^2}{2}\right) \Delta t}{\Delta x} \end{cases}$$

The `binomial_tree` function will precompute Δx and p_u firstly. And it consists of three subfunctions:

- `tree_construction` : It will return a list containing arrays of stock price at each time period.
- `payoff` : Calculate the payoff of calls and puts.
- `backward` : This function is to calculate the option's payoffs at t_i based on the payoffs at t_{i+1} . i.e. It's just one step in the recursion process.

And at last, we use `tree_construction` to construct a tree, and initialize the payoff at the terminal nodes. Then do the recursive calculation until we get the payoff value at time $t = 0$.

```
def binomial_tree(S, K, T, r, sigma, N, Type, style):  
    """  
    Binomial tree method for option pricing  
    :param S: spot price  
    :param K: strike  
    :param T: time to maturity  
    :param r: risk-free interest  
    :param sigma: volatility of underlying asset  
    :param N: steps of tree  
    :param Type: option type, 'c' or 'p'  
    :param style: option style, 'a' or 'e'  
    :return: price of option interested in
```

```

"""
# check illegal Type and style
if Type not in ['c', 'p']:
    raise TypeError("option type should be 'c' for call,"
                    "'p' for put")
if style not in ['e', 'a']:
    raise TypeError("option style should be 'a' for American option,"
                    "'e' for European option")
# delta_t, delta_x, prob of up and down
t = T / N
delta_x = np.sqrt((r - sigma ** 2 / 2) ** 2 * t ** 2 + sigma ** 2 * t)
p_u = 0.5 + 0.5 * (r - sigma ** 2 / 2) * t / delta_x
p_d = 1 - p_u

# construct stock price tree
def tree_construction():
    s = [np.array([S])]
    for i in range(N):
        temp = np.exp(delta_x) * s[i]
        temp = np.append(temp, s[i][-1] * np.exp(-delta_x))
        s.append(temp)
    return s

# payoff
def payoff(s):
    if Type == 'c':
        return np.maximum(s - K, 0)
    elif Type == 'p':
        return np.maximum(K - s, 0)
    else:
        raise TypeError("Illegal option type!")

# backward calculation
def backward(s, p):
    temp = np.roll(p, -1)
    temp = p * p_u + temp * p_d
    temp = temp * np.exp(-r * t)
    temp = np.delete(temp, -1)
    if style == 'e':
        return temp
    elif style == 'a':
        current_p = payoff(s)
        temp = np.maximum(temp, current_p)
    return temp

```

```

s = tree_construction()
# nodes at maturity
s_t = s[-1]
# print(s_t)
p = payoff(s_t)
while len(p) > 1:
    p = backward(s[len(p) - 2], p)
return float(p)

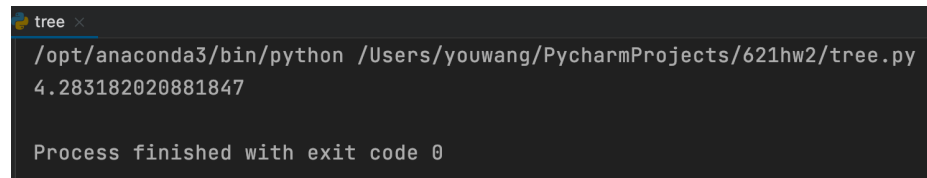
```

Use an example to check if the program works properly: (*Example 21-1* from John Hull's book):

```

if __name__ == '__main__':
    S = 50
    K = 50
    r = 0.1
    sigma = 0.4
    T = 0.4167
    N = 500
    print(binomial_tree(S, K, T, r, sigma, N, 'p', 'a'))

```



```

tree x
/opt/anaconda3/bin/python /Users/youwang/PycharmProjects/621hw2/tree.py
4.283182020881847

Process finished with exit code 0

```

Figure 1: Result of example

It's is exactly the same with the result in the book.

(b)

1. Use DATA2 in **Homework 1**(choose SPY option and matures at 3/24, 4/16, 6/28):

```

import numpy as np
import pandas as pd
from tree import *
data2 = pd.read_pickle('./DATA2.pkl')
r = 0.07/100
spy = data2[1]
spy = [spy[10], spy[13], spy[17]]

```

2. Choose options whose strike prices close to the value at the money:

```

def clean(x):
    info = ['contractSymbol','spotPrice', 'strike', 'bid', 'ask',
            'expiry', 'moneyness', 'delta_t', 'BS_price', 'vol', 'type']
    call = []
    put = []
    for df in x:
        df['moneyness'] = df.spotPrice / df.strike
        df = df[df.moneyness > 0.8]
        df = df[df.moneyness < 1.2]
        df = df[info].reset_index()
        del df['index']
        df = df.dropna()
        y = df[df.type == 'c'].iloc[0:21]
        z = df[df.type == 'p'].iloc[0:21]
        call.append(y)
        put.append(z)
    return call, put

call, put = clean(spy)

```

3. Use `binomial_tree()` function we defined before to calculate price:

```

def tree_price(x, style):
    if style == 'e':
        col_name = 'euro_price'
    elif style == 'a':
        col_name = 'amer_price'
    for df in x:
        df[col_name] = df.apply(lambda x: binomial_tree(x.spotPrice, x.strike, x.delta_t,
                                                         r,x.vol, 500, x.type, style),
                                axis=1)
    tree_price(call, 'e')

tree_price(put, 'e')
tree_price(call, 'a')
tree_price(put, 'a')

```

Here are the tables of options maturing at 3/21:

	contractSymbol	spotPrice	strike	bid	ask	expiry	moneyness	delta_t	BS_price	vol	type	euro_price	amer_price
0	SPY210324C00350000	382.329987	350.0	35.22	35.76	2021-03-24	1.092371	0.033043	35.010089	0.484941	c	35.009908	35.009908
1	SPY210324C00355000	382.329987	355.0	30.77	31.26	2021-03-24	1.076986	0.033043	30.411521	0.454883	c	30.407255	30.407255
2	SPY210324C00360000	382.329987	360.0	26.44	26.88	2021-03-24	1.062028	0.033043	25.932254	0.426139	c	25.934542	25.934542
3	SPY210324C00370000	382.329987	370.0	18.17	18.52	2021-03-24	1.033324	0.033043	17.428597	0.369656	c	17.432639	17.432639
4	SPY210324C00380000	382.329987	380.0	10.85	10.98	2021-03-24	1.006132	0.033043	9.867041	0.312887	c	9.868257	9.868257
5	SPY210324C00385000	382.329987	385.0	7.63	7.75	2021-03-24	0.993065	0.033043	6.636095	0.283810	c	6.639986	6.639986
6	SPY210324C00390000	382.329987	390.0	4.90	5.01	2021-03-24	0.980333	0.033043	3.974826	0.255819	c	3.975945	3.975945
7	SPY210324C00395000	382.329987	395.0	2.80	2.90	2021-03-24	0.967924	0.033043	2.099382	0.233547	c	2.101227	2.101227
8	SPY210324C00400000	382.329987	400.0	1.41	1.49	2021-03-24	0.955825	0.033043	0.960445	0.216491	c	0.959942	0.959942
9	SPY210324C00405000	382.329987	405.0	0.66	0.71	2021-03-24	0.944025	0.033043	0.395667	0.205960	c	0.395186	0.395186
10	SPY210324C00410000	382.329987	410.0	0.30	0.34	2021-03-24	0.932512	0.033043	0.158631	0.201766	c	0.158362	0.158362
11	SPY210324C00415000	382.329987	415.0	0.15	0.18	2021-03-24	0.921277	0.033043	0.067749	0.203075	c	0.067378	0.067378
12	SPY210324C00420000	382.329987	420.0	0.08	0.11	2021-03-24	0.910309	0.033043	0.033556	0.209281	c	0.033395	0.033395
13	SPY210324C00425000	382.329987	425.0	0.05	0.07	2021-03-24	0.899600	0.033043	0.019014	0.218372	c	0.018828	0.018828

Figure 2: call option table

	contractSymbol	spotPrice	strike	bid	ask	expiry	moneyness	delta_t	BS_price	vol	type	euro_price	amer_price
0	SPY210324P00320000	382.329987	320.0	1.07	1.12	2021-03-24	1.194781	0.033043	0.690709	0.579569	p	0.688758	0.688756
1	SPY210324P00325000	382.329987	325.0	1.25	1.31	2021-03-24	1.176400	0.033043	0.813592	0.556167	p	0.810501	0.810512
2	SPY210324P00330000	382.329987	330.0	1.48	1.54	2021-03-24	1.158576	0.033043	0.972759	0.534094	p	0.969776	0.969789
3	SPY210324P00335000	382.329987	335.0	1.75	1.82	2021-03-24	1.141284	0.033043	1.162338	0.511493	p	1.160242	1.160259
4	SPY210324P00340000	382.329987	340.0	2.07	2.15	2021-03-24	1.124500	0.033043	1.406688	0.489958	p	1.406770	1.406791
5	SPY210324P00345000	382.329987	345.0	2.47	2.55	2021-03-24	1.108203	0.033043	1.715585	0.468895	p	1.717110	1.717136
6	SPY210324P00350000	382.329987	350.0	2.95	3.04	2021-03-24	1.092371	0.033043	2.106629	0.448210	p	2.104844	2.104878
7	SPY210324P00355000	382.329987	355.0	3.53	3.62	2021-03-24	1.076986	0.033043	2.597910	0.427557	p	2.599674	2.599722
8	SPY210324P00360000	382.329987	360.0	4.23	4.33	2021-03-24	1.062028	0.033043	3.208412	0.406458	p	3.206562	3.206624
9	SPY210324P00365000	382.329987	365.0	5.07	5.17	2021-03-24	1.047479	0.033043	3.965437	0.384667	p	3.968755	3.968839
10	SPY210324P00370000	382.329987	370.0	6.09	6.19	2021-03-24	1.033324	0.033043	4.898244	0.361709	p	4.902073	4.902191
11	SPY210324P00375000	382.329987	375.0	7.31	7.42	2021-03-24	1.019547	0.033043	6.064039	0.337833	p	6.068295	6.068460
12	SPY210324P00380000	382.329987	380.0	8.81	9.06	2021-03-24	1.006132	0.033043	7.532558	0.313044	p	7.533761	7.533997
13	SPY210324P00385000	382.329987	385.0	10.67	10.92	2021-03-24	0.993065	0.033043	9.420325	0.288277	p	9.424272	9.424623
14	SPY210324P00390000	382.329987	390.0	12.78	13.58	2021-03-24	0.980333	0.033043	11.875811	0.265179	p	11.878736	11.879268
15	SPY210324P00395000	382.329987	395.0	15.74	16.52	2021-03-24	0.967924	0.033043	15.025706	0.246006	p	15.025263	15.026092
16	SPY210324P00400000	382.329987	400.0	18.97	20.44	2021-03-24	0.955825	0.033043	18.921465	0.235862	p	18.921502	18.922722
17	SPY210324P00410000	382.329987	410.0	27.94	29.17	2021-03-24	0.932512	0.033043	28.063612	0.240343	p	28.062666	28.064850
18	SPY210324P00420000	382.329987	420.0	37.58	39.10	2021-03-24	0.910309	0.033043	37.931157	0.282183	p	37.931060	37.933765

Figure 3: put option table

4. Use option matures at 3/24 to do plots:

```
plt.figure(1)
plt.plot(call[0].bid, label='bid')
plt.plot(call[0].ask, label='ask')
plt.plot(call[0].BS_price, label='BS-price')
```

```

plt.plot(call[0].euro_price, label='binomial-tree euro price')
plt.plot(call[0].amer_price, label='binomial-tree amer price')
plt.xlabel('index')
plt.ylabel('price')
plt.title('comparison 1')
plt.legend()

plt.figure(2)
plt.plot(put[0].bid, label='bid')
plt.plot(put[0].ask, label='ask')
plt.plot(put[0].BS_price, label='BS-price')
plt.plot(put[0].euro_price, label='binomial-tree euro price')
plt.plot(put[0].amer_price, label='binomial-tree amer price')
plt.xlabel('index')
plt.ylabel('price')
plt.title('comparison 2')
plt.legend()
plt.show()

```

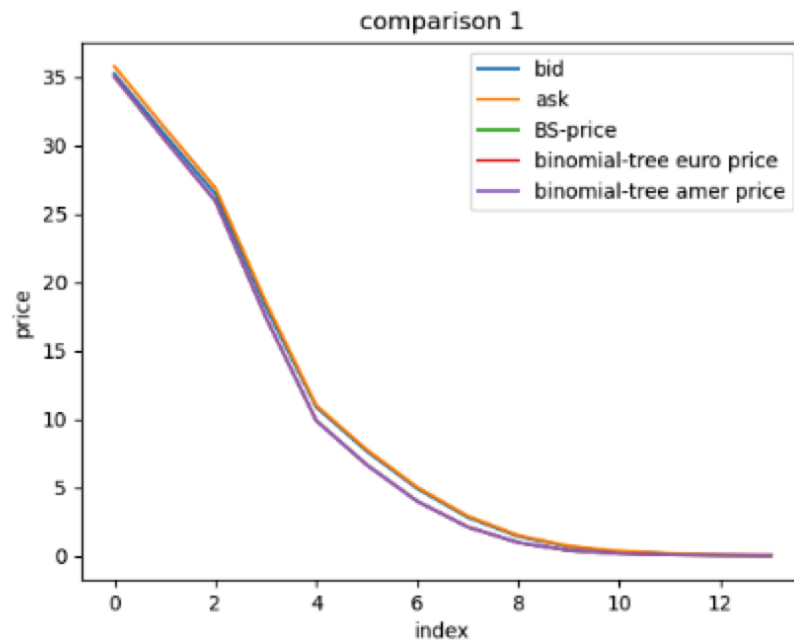


Figure 4: call option comparison plot

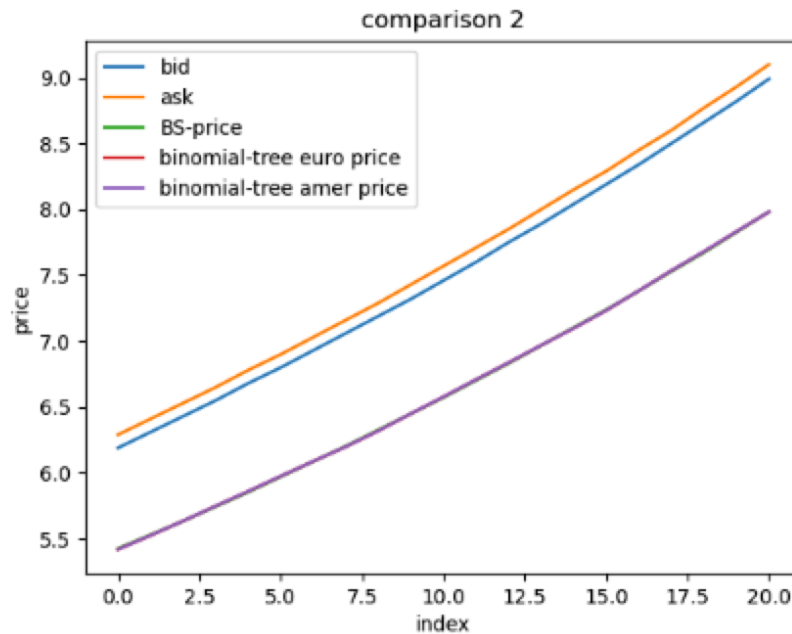


Figure 5: put option comparison plot

(c)

By looking at tables and plots:

- In general, price calculated by binomial tree model are very close to the result of BS model.
- For call option, European option and American option calculated by binomial tree are the same; For put option, American option price is slightly larger than European option price.
- Because in DATA2 we use previous day's implied volatility, both prices of Binomial model and BS model are a little bit smaller than bid/ask value.

(d)

```
S = 50
K = 50
r = 0.1
sigma = 0.4
T = 0.4167
```

```

error = lambda x: abs(BS_formula('c', S, K, T, sigma, r) \
                      - binomial_tree(S, K, T, r, sigma, x, 'c', 'e'))

N = [10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 350, 400]
epsilon = []
for n in N:
    epsilon.append(error(n))
plt.figure(3)
plt.plot(N, epsilon)
plt.xlabel('steps')
plt.ylabel('absolute error')
plt.show()

```

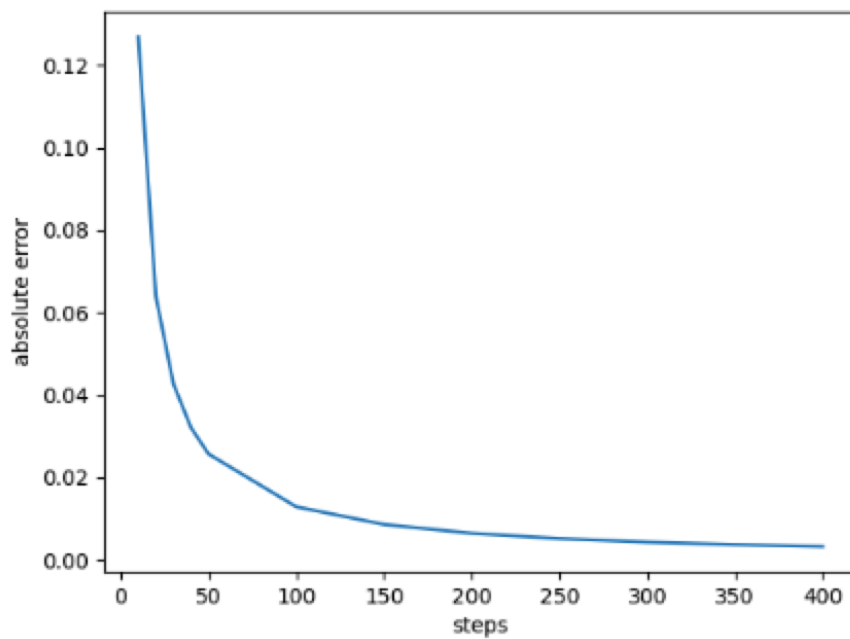


Figure 6: absolute error

We can observe that absolute error decrease dramatically in 100 steps. And the speed becomes slower as steps go larger.

Problem 2

(a)

Similar to binomial tree, the function precomputes Δx and p_u, p_m, p_d . And it also contains three subfunctions which have the same algorithm with the functions in `binomial_tree` function. Here we talk about the `backward_barrier` function specifically:

- Actually the `backward_barrier` only works for knock-out options.(for knock-in option, we use In-Out parity to get price).
- For knock-out options, when we do recursive calculations, on the basis of previous algorithm, we need to check if the stock price touches the barrier and knock out.
- The parameter `barrier` is a list containing 3 elements , elements in it correspond to barrier value, up barrier(U)/lower barrier(D), knock-in/out.

```
def trinomial_tree(S, K, barrier, T, r, sigma, N, Type, style):
    # check illegal Type and style
    if Type not in ['c', 'p']:
        raise TypeError("option type should be 'c' for call,"
                        "'p' for put")
    if style not in ['e', 'a']:
        raise TypeError("option style should be 'a' for American option,"
                        "'e' for European option")

    # initialization
    t = T / N
    D = r - sigma ** 2 / 2
    delta_x = sigma * np.sqrt(3 * t)
    p_u = 0.5 * ((sigma ** 2 * t + D ** 2 * t ** 2) / delta_x ** 2 + D * t / delta_x)
    p_m = 1 - (sigma ** 2 * t + D ** 2 * t ** 2) / delta_x ** 2
    p_d = 0.5 * ((sigma ** 2 * t + D ** 2 * t ** 2) / delta_x ** 2 - D * t / delta_x)

    def tree_construction():
        s = [np.array([S])]
        for i in range(N):
            temp = np.exp(delta_x) * s[i]
            temp = np.append(temp, s[i][-1])
            temp = np.append(temp, s[i][-1] * np.exp(-delta_x))
            s.append(temp)
        return s

    # payoff
    def payoff(s):
        if Type == 'c':
            return np.maximum(s - K, 0)
        elif Type == 'p':
```

```

        return np.maximum(K - s, 0)
    else:
        raise TypeError("Illegal option type!")

def backward(s, p):
    temp1 = np.roll(p, -1)
    temp2 = np.roll(p, -2)
    temp = p * p_u + temp1 * p_m + temp2 * p_d
    temp = temp * np.exp(-r * t)
    temp = np.delete(temp, -1)
    temp = np.delete(temp, -1)
    if style == 'e':
        return temp
    elif style == 'a':
        current_p = payoff(s)
        temp = np.maximum(temp, current_p)
        return temp

def backward_barrier(s, p):
    temp1 = np.roll(p, -1)
    temp2 = np.roll(p, -2)
    temp = p * p_u + temp1 * p_m + temp2 * p_d
    temp = temp * np.exp(-r * t)
    temp = np.delete(temp, -1)
    temp = np.delete(temp, -1)
    if barrier[1] == 'D':
        temp[s <= barrier[0]] = 0
        if style == 'e':
            return temp
        elif style == 'a':
            current_p = payoff(s)
            temp = np.maximum(temp, current_p)
            temp[s <= barrier[0]] = 0
            return temp
    elif barrier[1] == 'U':
        temp[s >= barrier[0]] = 0
        if style == 'e':
            return temp
        elif style == 'a':
            current_p = payoff(s)
            temp = np.maximum(temp, current_p)
            temp[s >= barrier[0]] = 0
            return temp

s = tree_construction()

```

```

s_t = s[-1]
p = payoff(s_t)
# without barrier
if barrier == 0:
    p = payoff(s_t)
    while len(p) > 1:
        p = backward(s[len(p) // 2 - 1], p)
    return float(p)

# with barrier
if barrier[1] == 'D':
    p[s_t <= barrier[0]] = 0
elif barrier[1] == 'U':
    p[s_t >= barrier[0]] = 0
if barrier[2] == '0':
    while len(p) > 1:
        p = backward_barrier(s[len(p) // 2 - 1], p)
    return float(p)
elif barrier[2] == 'I':
    out = trinomial_tree(S, K, [barrier[0], barrier[1], '0'], T, r, sigma, N, Type, style)
    v = trinomial_tree(S, K, 0, T, r, sigma, N, Type, style)
    return v - out

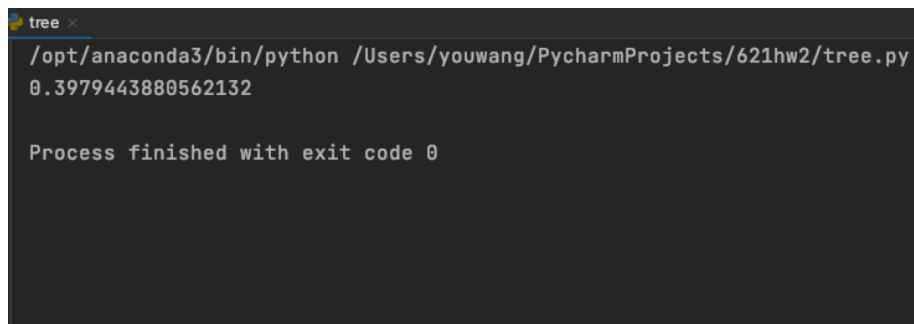
```

Input the parameters given in this problem:

```

if __name__ == '__main__':
    print(trinomial_tree(10, 10, [11, 'U', '0'], 0.3, 0.01, 0.2, 8000, 'c', 'e'))

```



```

tree x
/opt/anaconda3/bin/python /Users/youwang/PycharmProjects/621hw2/tree.py
0.3979443880562132

Process finished with exit code 0

```

Figure 7: Result of Problem2 (a)

Here use 8000 steps to calculate the price, it's \$0.05534.

(b)

For the European Up-and-Out Call option explicit formula is:

$$UO_{BS} = C_{BS}(S, K) - C_{BS}(S, H) - (H - K)e^{-rT}\Phi(d_{BS}(S, H)) \\ - \left(\frac{H}{S}\right)^{\frac{2v}{\sigma^2}} \left\{ C_{BS}\left(\frac{H^2}{S}, K\right) - C_{BS}\left(\frac{H^2}{S}, H\right) - (H - K)e^{-rT}\Phi(d_{BS}(H, S)) \right\}$$

It's straightforward so we can directly define the function of Up-and-Out option:

```
def BS_formula(Type, S, K, T, sigma, r):
    d1 = (np.log(S / K) + (r + sigma ** 2 / 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    if Type == 'c':
        return norm.cdf(d1) * S - norm.cdf(d2) * K * np.exp(-r * T)
    elif Type == 'p':
        return K * np.exp(-r * T) * norm.cdf(-d2) - norm.cdf(-d1) * S
    else:
        raise TypeError("Type must be 'c' for call, 'p' for put")

def UO_call(S, K, T, r, sigma, H):
    nu = r - sigma ** 2 / 2

    def bs(S_, K_):
        return BS_formula('c', S_, K_, T, sigma, r)

    def dbs(S_, K_):
        return (np.log(S_ / K_) + nu * T) / (sigma * np.sqrt(T))

    c1 = bs(S, K)
    c2 = bs(S, H)
    c3 = bs(H ** 2 / S, K)
    c4 = bs(H ** 2 / S, H)
    d1 = dbs(S, H)
    d2 = dbs(H, S)

    temp1 = c1 - c2 - (H - K) * np.exp(-r * T) * norm.cdf(d1)
    temp2 = (H / S) ** (2 * nu / sigma ** 2) * (c3 - c4 - (H - K) * np.exp(-r * T) * norm.cdf(d2))
    return temp1 - temp2
```

Run the following code:

```
S = 10
K = 10
T = 0.3
```

```

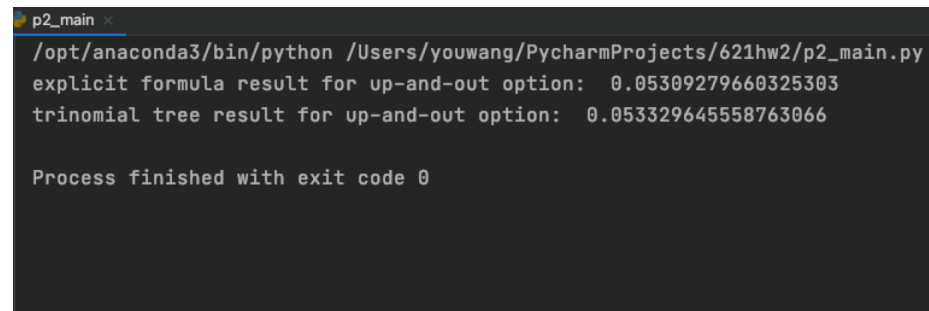
sigma = 0.2
r = 0.01
H = 11

res1 = UO_call(S, K, T, r, sigma, H)
res2= trinomial_tree(S, K, [H, 'U', 'O'], T, r, sigma, 8000, 'c', 'e')

print("explicit formula result for up-and-out option: ", res1)
print("trinomial tree result for up-and-out option: ", res2)

```

The result is:



```

p2_main x
/opt/anaconda3/bin/python /Users/youwang/PycharmProjects/621hw2/p2_main.py
explicit formula result for up-and-out option:  0.05309279660325303
trinomial tree result for up-and-out option:  0.053329645558763066

Process finished with exit code 0

```

Figure 8: Result of Problem2 (b)

Results calculated by two methods are very close to each other.

(c)

First we define the function of explicit formula for Up-and-In call option:

```

def UI_call(S, K, T, r, sigma, H):
    nu = r - sigma ** 2 / 2

    def bsp(S_, K_):
        return BS_formula('p', S_, K_, T, sigma, r)

    def bsc(S_, K_):
        return BS_formula('c', S_, K_, T, sigma, r)

    def dbs(S_, K_):
        return (np.log(S_ / K_) + nu * T) / (sigma * np.sqrt(T))

    p1 = bsp(H ** 2 / S, K)
    p2 = bsp(H ** 2 / S, H)

```

```

d1 = dbs(H, S)
d2 = dbs(S, H)
c1 = bsc(S, H)

temp1 = (H / S) ** (2 * nu / sigma ** 2) * (p1 - p2 + (H - K) * np.exp(-r * T) * norm.cdf(d1))
temp2 = c1 + (H - K) * np.exp(-r * T) * norm.cdf(d2)
return temp1 + temp2

```

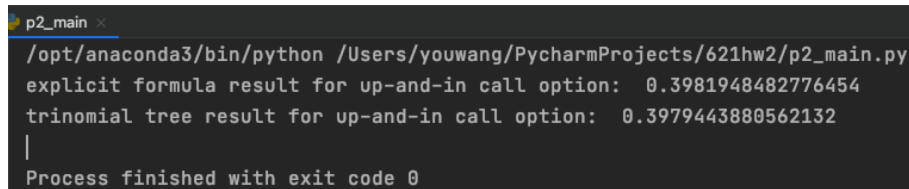
Then compare the result:

```

S = 10
K = 10
T = 0.3
sigma = 0.2
r = 0.01
H = 11

res3 = UI_call(S, K, T, r, sigma, H)
res4 = trinomial_tree(S, K, [H, 'U', 'I'], T, r, sigma, 8000, 'c', 'e')

```



```

p2_main
/opt/anaconda3/bin/python /Users/youwang/PycharmProjects/621hw2/p2_main.py
explicit formula result for up-and-in call option: 0.3981948482776454
trinomial tree result for up-and-in call option: 0.3979443880562132
|
Process finished with exit code 0

```

Figure 9: Result of Problem2 (c)

Obviously the results are still matching.

(d)

Use the same parameter of (b):

```

res5 = trinomial_tree(S, K, [H, 'U', 'I'], T, r, sigma, 8000, 'c', 'a')
print("trinomial tree result for American up-and-in call option: ", res5)

```

The result is:

```
p2_main x
/opt/anaconda3/bin/python /Users/youwang/PycharmProjects/621hw2/p2_main.py
trinomial tree result for American up-and-in call option: 0.025444857423303213

Process finished with exit code 0
```

Figure 10: Result of Problem2 (d)

Problem 3

We need to determine the range of possible strike prices for which early exercise is optimal. Consider when early exercise is optimal, it means that expected American option premium is larger than expected European option premium. So to solve this problem, is to find the strike that makes American option premium larger than European option premium.

(a)

We can write a simple program to find the strike:

```
import numpy as np
from numpy import exp

u = 1.2
d = 0.9

def simple_tree(k):
    r = 0.04
    q = 0.02
    dt = 0.5 / 2
    discount = exp(-r * dt)
    pu = (exp((r - q) * dt) - d) / (u - d)
    pd = 1 - pu
    s0 = 40
    s1 = np.array([48, 36])
    s2 = np.array([57.6, 43.2, 32.4])
    # without early exercise
    payoff1 = np.maximum(s2 - k, 0)
    payoff1 = np.array([payoff1[0] * pu + payoff1[1] * pd,
                        payoff1[1] * pu + payoff1[2] * pd])
```

```

    payoff1 = payoff1 * discount
    payoff2 = np.maximum(payoff1, s1 - k)
    payoff1 = payoff1[0] * pu + payoff1[1] * pd
    payoff1 = discount * payoff1
    payoff2 = payoff2[0] * pu + payoff2[1] * pd
    payoff2 = discount * payoff2
    return payoff2-payoff1

n = np.arange(0, 48, 0.001)
diff1 = []
for i in n:
    diff1.append(simple_tree(i))
plt.plot(n, diff1, label='amer - euro')
plt.legend()
plt.show()

```

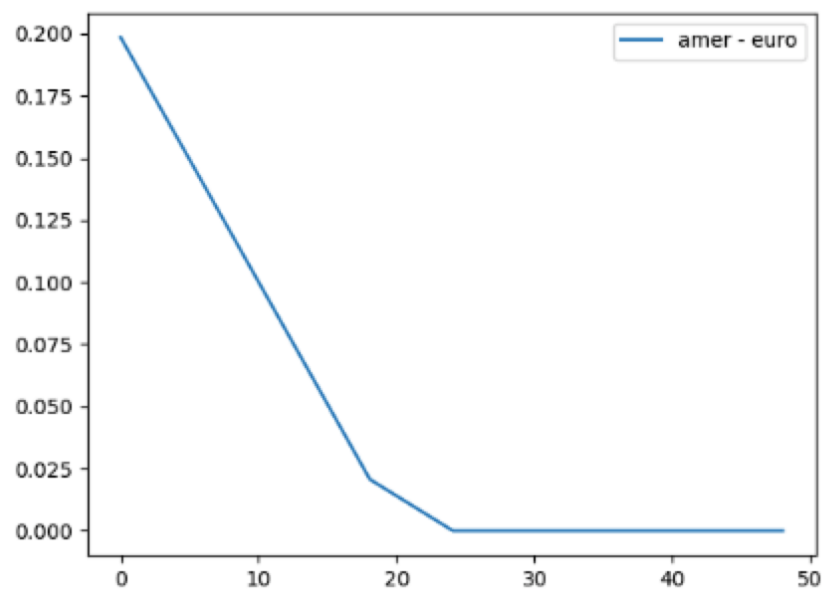


Figure 11: Problem 3 (a)

It reveals that when strike prices is less than 24, early exercise is optimal.

(b)

Do some change on the program before:

```
def simple_tree2(k):
    r = 0.04
    q = 0
    dt = 0.5 / 2
    discount = exp(-r * dt)
    pu = (exp((r - q) * dt) - d) / (u - d)
    pd = 1 - pu
    s0 = 40
    s1 = np.array([46.56, 34.92])
    s2 = np.array([55.872, 41.904, 31.824])
    # without early exercise
    payoff1 = np.maximum(s2 - k, 0)
    payoff1 = np.array([payoff1[0] * pu + payoff1[1] * pd,
                        payoff1[1] * pu + payoff1[2] * pd])
    payoff1 = payoff1 * discount
    payoff2 = np.maximum(payoff1, s1 - k)
    payoff1 = payoff1[0] * pu + payoff1[1] * pd
    payoff1 = discount * payoff1
    payoff2 = payoff2[0] * pu + payoff2[1] * pd
    payoff2 = discount * payoff2
    return payoff2 - payoff1

n = np.arange(0, 48, 0.001)
diff2 = []
for i in n:
    diff2.append(simple_tree2(i))
plt.plot(n, diff2, label='amer - euro')
plt.legend()
plt.show()
```

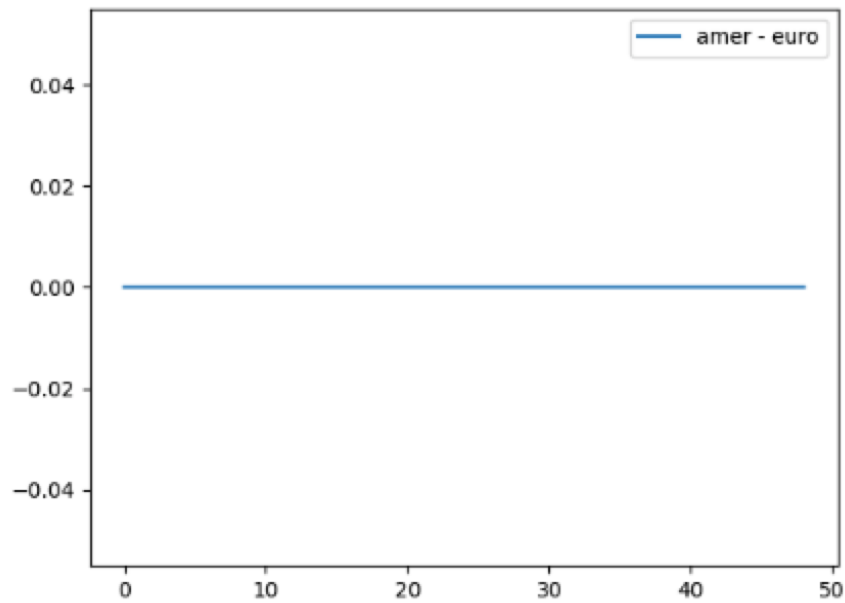


Figure 12: Problem 3 (b)

It reveals that the expected premium of American option is always the same with European option. That's to say, none of strike would satisfy that early exercise is optimal.

Bonus Problem 2

By following the pseudo code in the paper:

```
import numpy as np

h1 = lambda x: np.ones(len(x))
h2 = lambda x: -2 * x
h3 = lambda x: x ** 2
f1 = lambda x: x ** 2
f2 = lambda x: x
f3 = lambda x: np.zeros(len(x))

def trinomial_tree_swap(S, K, T, r, sigma, N, l):
    # initialization
```

```

t = T / N
D = r - sigma ** 2 / 2
delta_x = sigma * np.sqrt(3 * t)
p_u = 0.5 * ((sigma ** 2 * t + D ** 2 * t ** 2) / delta_x ** 2 + D * t / delta_x)
p_m = 1 - (sigma ** 2 * t + D ** 2 * t ** 2) / delta_x ** 2
p_d = 0.5 * ((sigma ** 2 * t + D ** 2 * t ** 2) / delta_x ** 2 - D * t / delta_x)

# construct stock price tree
def tree_construction():
    s = [np.array([S])]
    for i in range(N * l):
        s_temp = np.exp(delta_x) * s[i]
        s_temp = np.append(s_temp, s[i][-1])
        s_temp = np.append(s_temp, s[i][-1] * np.exp(-delta_x))
        s.append(s_temp)
    return s

s = tree_construction()

for i in np.arange(N, -1, -1):
    for j in range(l):
        if i == N and j == 0:
            v = np.zeros(len(s[i * l]))
        elif i > 0 and j == 0:
            v = a * f1(s[i * l]) + b * f2(s[i * l]) + c + f3(s[i * l])
        elif i == 0:
            v = a * f1(s[i * l]) + b * f2(s[i * l]) + c + f3(s[i * l])
            return float(v) / N/l
        if (j == 0 and l > 1) or l == 1:
            temp = s[i * l - j]
            temp1 = np.roll(temp, -1)
            temp2 = np.roll(temp, -2)
            a = h1(temp) * p_u + h1(temp1) * p_m + h1(temp2) * p_d
            b = h2(temp) * p_u + h2(temp1) * p_m + h2(temp2) * p_d
            c = (h3(temp) + v) * p_u + (h3(temp1) + v) * p_m + (h3(temp2) + v) * p_d
            a = np.delete(a, -1)
            a = np.delete(a, -1)
            b = np.delete(b, -1)
            b = np.delete(b, -1)
            c = np.delete(c, -1)
            c = np.delete(c, -1)
        else:
            a1 = np.roll(a, -1)
            a2 = np.roll(a, -2)
            b1 = np.roll(b, -1)

```

```

        b2 = np.roll(b, -2)
        c1 = np.roll(c, -1)
        c2 = np.roll(c, -2)
        a = p_u * a + p_m * a1 + p_d * a2
        b = p_u * b + p_m * b1 + p_d * b2
        c = p_u * c + p_m * c1 + p_d * c2
        a = np.delete(a, -1)
        a = np.delete(a, -1)
        b = np.delete(b, -1)
        b = np.delete(b, -1)
        c = np.delete(c, -1)
        c = np.delete(c, -1)

S = 10
K = 10
r = 0.01
sigma = 0.2
T = 0.3
print(trinomial_tree_swap(S, K, T, r, sigma, 20, 3)*100 - K)
print(trinomial_tree_swap(S, K, T, r, sigma, 30, 2)*100 - K)

```

```

var_swap x
/opt/anaconda3/bin/python /Users/youwang/PycharmProjects/621hw2/var_swap.py
5.4670979129682475
3.158745704245522

Process finished with exit code 0

```

Figure 13: Result of Bonus Problem 2