IT'S-A ME! NINTENDO!

Harsha Kankanamge

THIS CODE PULLS ALL STOCK DATA FROM YAHOO FINANCE

```
import pandas_datareader as pdr
import matplotlib.pyplot as plt
import datetime

ntdoy = pdr.get_data_yahoo('NTDOY',
                          start=datetime.datetime(1996, 11, 18),
                          end=datetime.datetime(2018, 11, 27))
Current_price=ntdoy['Adj Close'][-1:]
Current_price=int(Current_price)
ntdoy[:10] # Showing first 10 data rows
ntdoy[-10:] # Showing last 10 data rows
```

DATASET

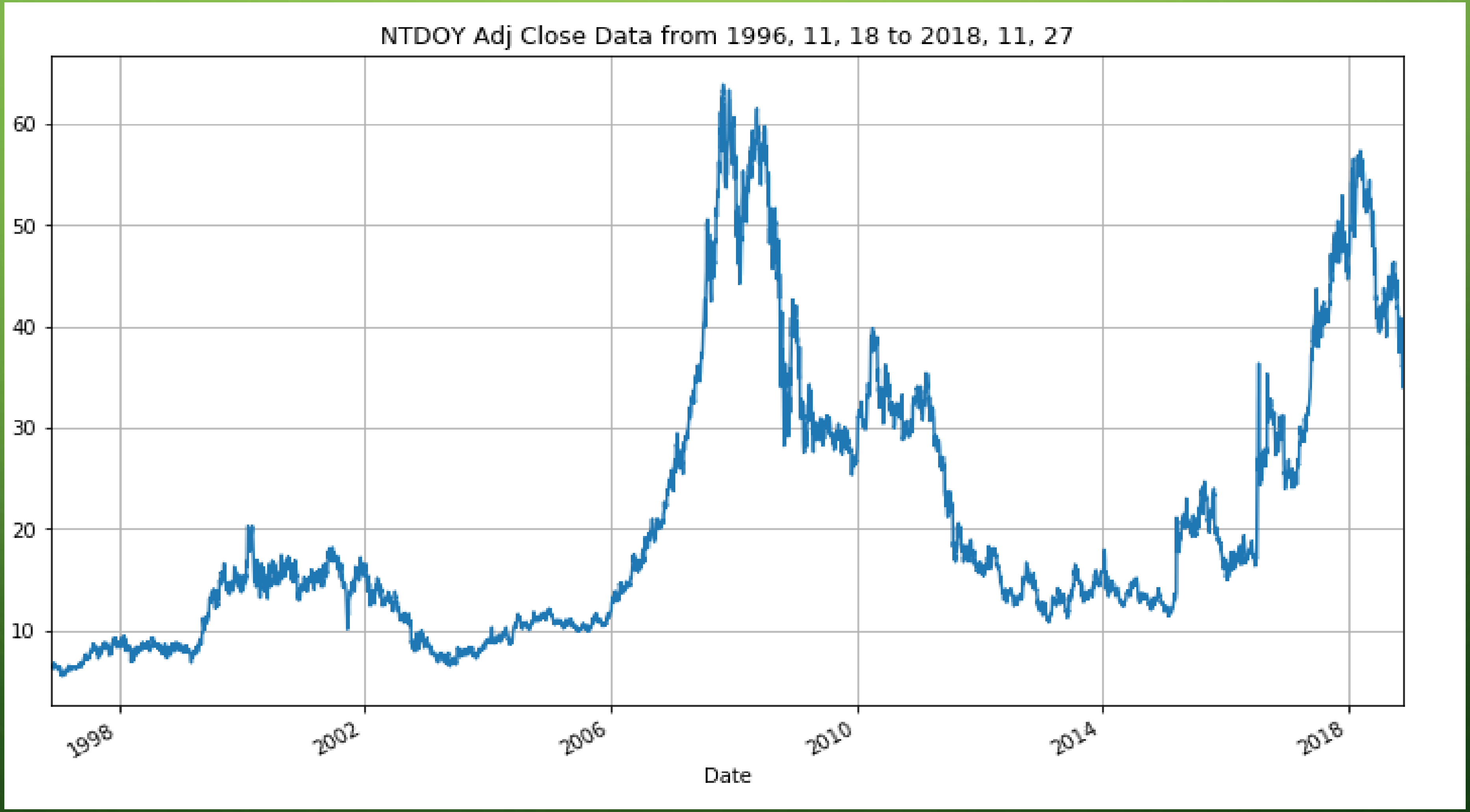
First 10 Rows:

Date	High	Low	Open	Close	Volume	Adj Close
1996-11-18	9.000	8.6250	9.0000	9.0000	81300.0	6.336266
1996-11-19	9.000	8.6250	9.0000	9.0000	138100.0	6.336266
1996-11-20	9.250	8.5000	9.0625	9.0625	56000.0	6.380269
1996-11-21	9.125	8.5625	9.0000	9.0000	91700.0	6.336266
1996-11-22	9.250	8.7500	9.1250	9.1250	200100.0	6.424270
1996-11-25	9.250	8.7500	9.1250	9.1250	90600.0	6.424270
1996-11-26	9.250	8.7500	9.2500	9.2500	76200.0	6.512273
1996-11-27	9.250	8.7500	9.2500	9.2500	75000.0	6.512273
1996-11-29	9.250	8.7500	9.0625	9.0625	48000.0	6.380269
1996-12-02	9.250	8.5000	8.7500	8.7500	130600.0	6.160260

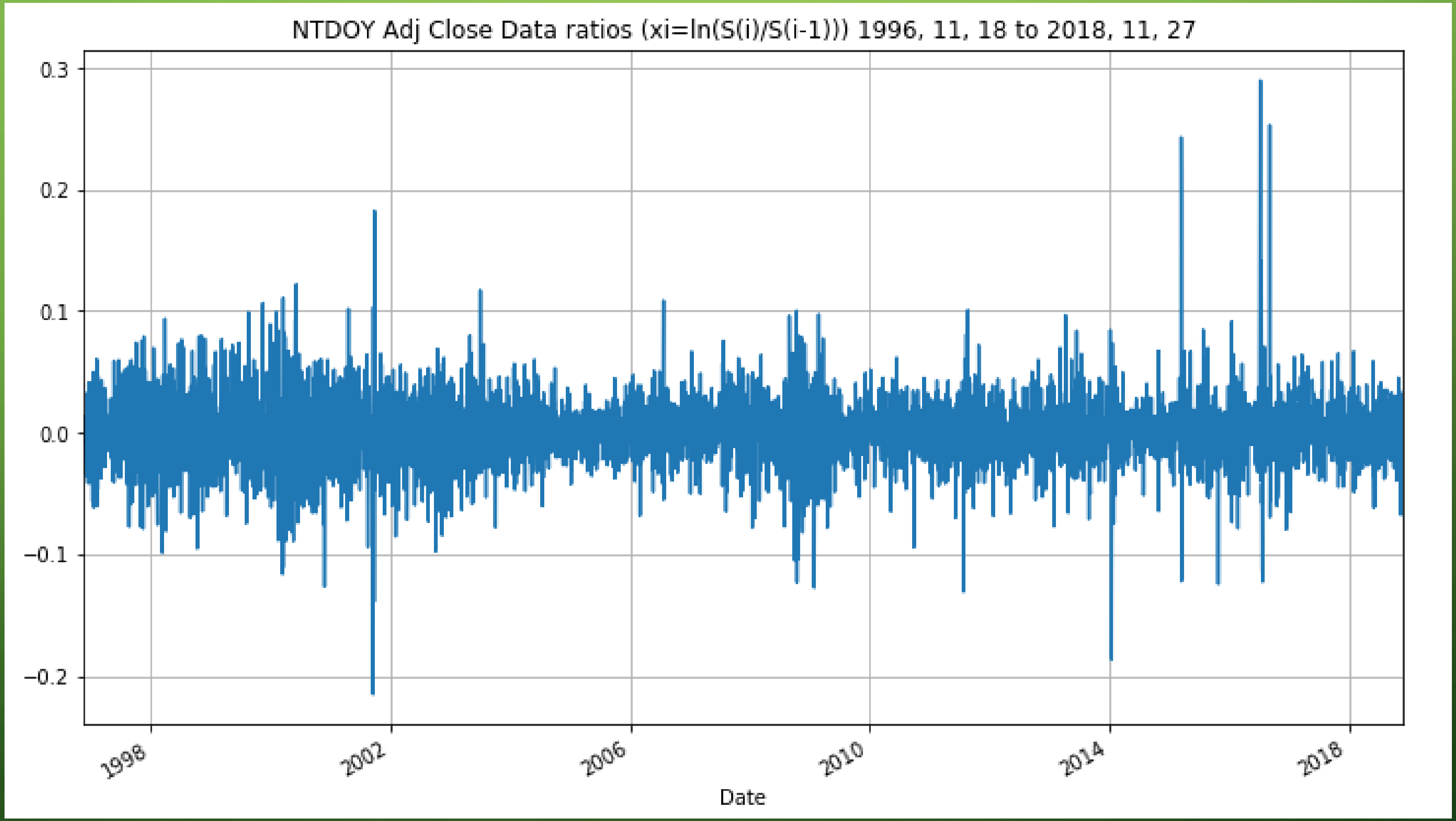
Last 10 Rows:

Date	High	Low	Open	Close	Volume	Adj Close
2018-11-14	39.110001	38.419998	39.110001	38.669998	196500.0	38.669998
2018-11-15	38.750000	38.110001	38.299999	38.520000	424100.0	38.520000
2018-11-16	36.299999	34.959999	35.150002	36.000000	611600.0	36.000000
2018-11-19	36.779999	36.000000	36.779999	36.139999	373000.0	36.139999
2018-11-20	34.400002	33.900002	34.250000	33.950001	564400.0	33.950001
2018-11-21	35.029999	34.110001	34.509998	34.820000	545500.0	34.820000
2018-11-23	35.139999	34.599998	34.950001	35.029999	184800.0	35.029999
2018-11-26	36.349998	35.939999	36.110001	36.189999	270100.0	36.189999
2018-11-27	36.209999	35.810001	36.209999	36.099998	182500.0	36.099998
2018-11-28	37.500000	37.000000	37.099998	37.500000	519000.0	37.500000

NINTENDO ADJUSTED CLOSE: DATA VS YEARS



NINTENDO ADJUSTED CLOSE DATA RATIOS



VOLATILITY CALCULATION

```python
class stock_vol:

    def __init__(self, tk, start, end):
        self.tk = tk
        self.start = start
        self.end = end
        all_data = pdr.get_data_yahoo(self.tk, start=self.start, end=self.end)
        self.stock_data = pd.DataFrame(all_data['Adj Close'], columns=['Adj Close'])
        self.stock_data["log"] = np.log(self.stock_data)-np.log(self.stock_data.shift(1))

    def mean_sigma(self):
        st = self.stock_data["log"].dropna().ewm(span=252).std()
        sigma = st.iloc[-1]
        return sigma


if __name__ == "__main__":
    vol = stock_vol("NTDOY", start="1996-11-18", end="2018-11-27")
    test = vol.stock_data["log"].dropna()
    print(test)
```

```python
import math as m
xi=[]
for i in range(0,5543):
    xi.append(test[i])
x1=0
for i in xi:
    x1=x1+i**2
x2=x1/len(xi)
mean=sum(xi)/len(xi)
mean=mean**2
volatility=m.sqrt(252)*m.sqrt(len(xi)*(x2-mean)/(len(xi)-1))
print('Volatility =', volatility)
```

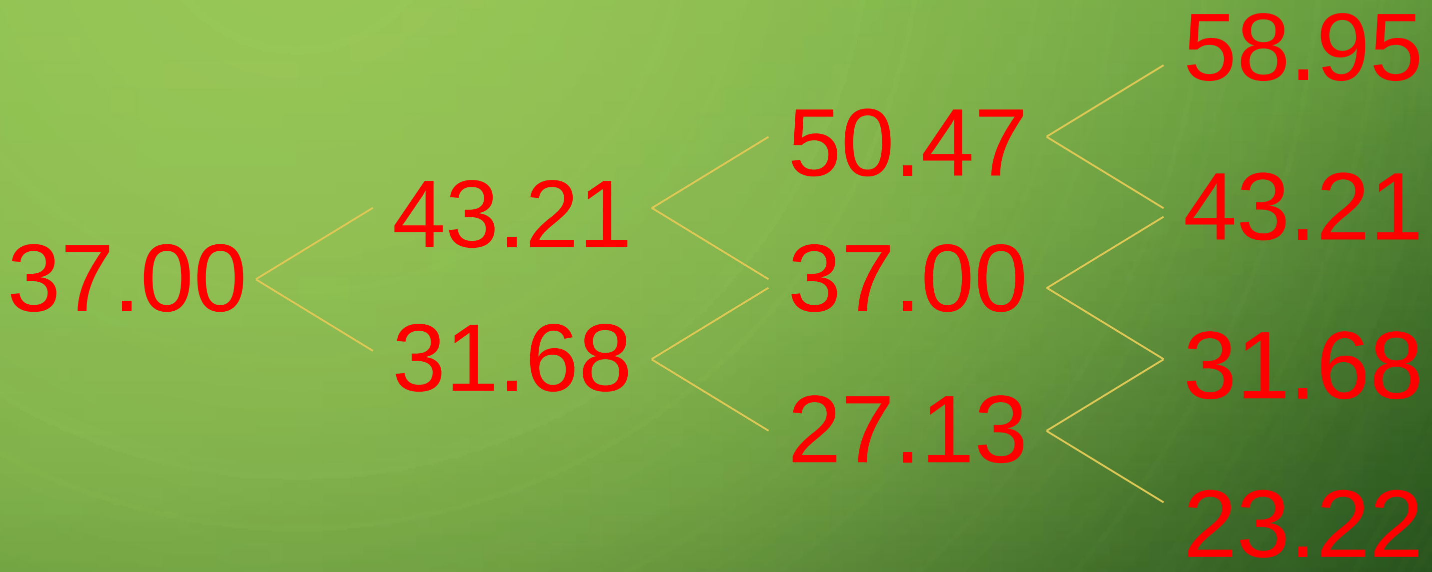
Volatility = 0.4390591586486805

COX-ROSS-RUBINSTEIN TREE

```python
import sys
import numpy as np
from math import pow
import math
# create the Binomial Tree

def build_stock_tree(S,T,N,sig):
    dt=T/N;
    u=math.exp(sig*math.sqrt(dt));
    d=math.exp(-sig*math.sqrt(dt));
    tree=np.zeros((N+1,N+1))
    for i in range(N+1):
        for j in range(i+1):
            tree[i][j]=S*pow(u,j)*pow(d,i-j)


    return(tree)
```




```python
s = [[str(e) for e in row] for row in build_stock_tree(Current_price,0.5,4,volatility)]
lens = [max(map(len, col)) for col in zip(*s)]
fmt = '\t'.join('{{:{}}}'.format(x) for x in lens)
table = [fmt.format(*row) for row in s]
print('\n'.join(table))
```

BUILDING THE COX-ROSS-RUBINSTEIN TREE

```
import numpy as np
def CRRTree(type,S0, K, r, sigma,Ldelta, T, N):
    #calculate delta T
    deltaT = float(T) / N
    # up and down factor will be constant for the tree so we calculate outside the loop
    u = np.exp(sigma * np.sqrt(deltaT))
    d = np.exp(-sigma * np.sqrt(deltaT))
    #to work with vector we need to init the arrays using numpy
    fs =  np.asarray([0.0 for i in range(N + 1)])
    #we need the stock tree for calculations of expiration values
    fs2 = np.asarray([(S0 * u**j * d**(N - j)) for j in range(N + 1)])
    #we vectorize the strikes as well so the expiration check will be faster
    fs3 =np.asarray( [float(K) for i in range(N + 1)])
    a = np.exp(r * deltaT)
    p=((np.exp((r-Ldelta)*deltaT))-d)/(u-d);
    oneMinusP = 1.0 - p
    if type =="C":
        fs[:] = np.maximum(fs2-fs3, 0.0)
    else:
        fs[:] = np.maximum(-fs2+fs3, 0.0)

    #calculate backward the option prices
    for i in range(N-1, -1, -1):
       fs[:-1]=np.exp(-r * deltaT) * (p * fs[1:] + oneMinusP * fs[:-1])
       fs2[:]=fs2[:]*u

    # print fs
    return fs[0]
```

PUT OPTION PRICES USING COX-ROSS-RUBINSTEIN

```
for i in range(1,10):
    print(CRRTree('p',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

5.550985732987845
3.912526904981438
4.8074254771544185
4.155890706828922
4.654228769685962
4.243637755686113
4.588847540998152
4.288644450868489
4.552686527038348

```
for i in range(100000,100010):
    print(CRRTree('P',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

4.427562475493936
4.4275849231754885
4.427562475728777
4.427584922662847
4.427562476047598
4.427584922537167
4.427562476022619
4.427584922495112
4.427562476532915
4.427584922184095

CALL OPTION PRICES USING COX-ROSS-RUBINSTEIN

```
for i in range(1,10):
    print(CRRTree('C',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

5.75015329548223
4.111694467475819
5.006593039648809
4.355058269323318
4.853396332180331
4.442805318180507
4.788015103492518
4.487812013362878
4.751854089532722

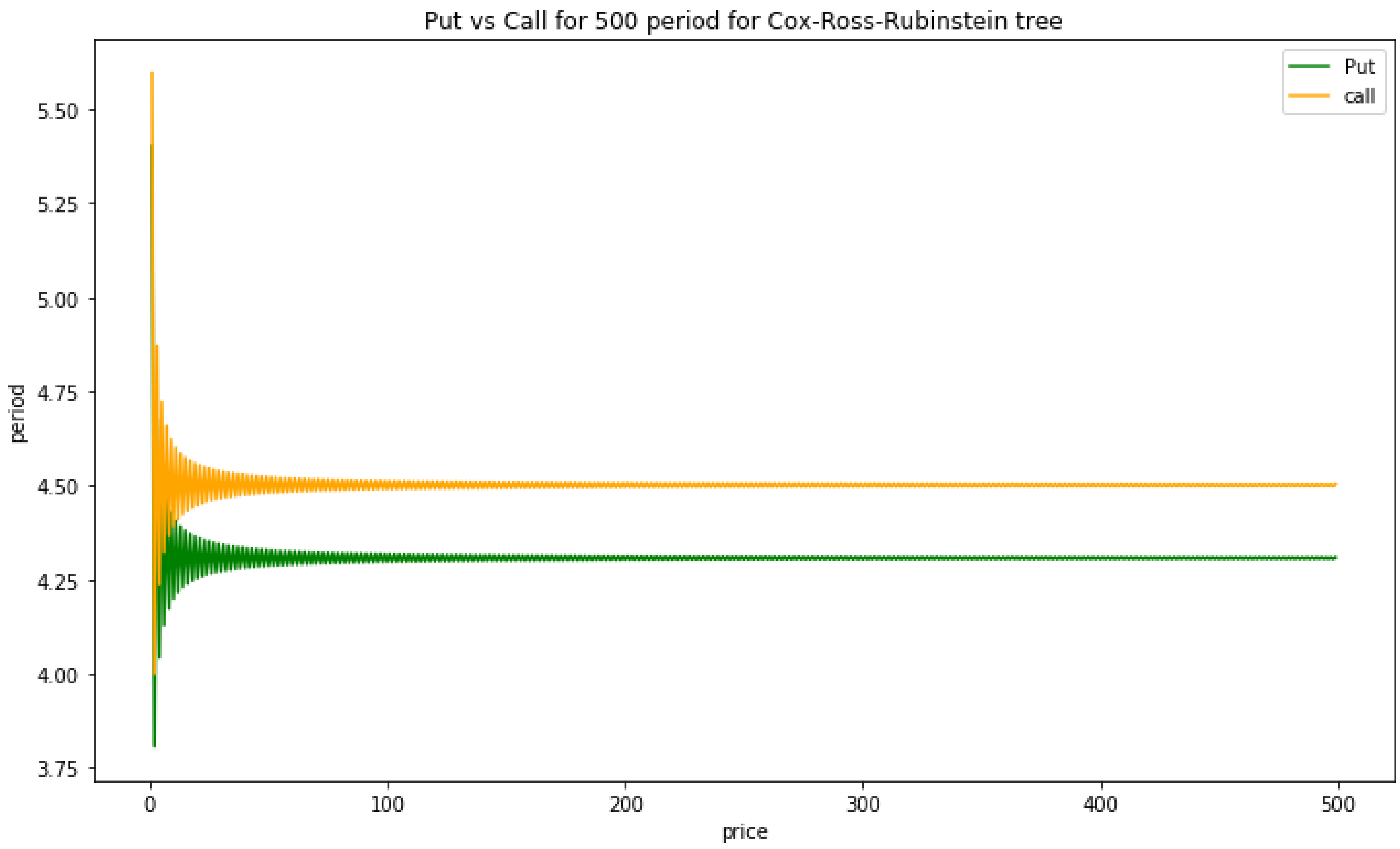
```
for i in range(100000,100010):
    print(CRRTree('C',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

4.6267300380144585
4.6267524852964295
4.626730038228505
4.6267524855004325
4.626730038366811
4.626752485113659
4.626730038816596
4.626752484651023
4.626730038829164
4.626752484577381

PUT VS CALL FOR 500 PERIOD FOR COX-ROSS-RUBINSTEIN TREE

```
%matplotlib inline
import matplotlib.pyplot as plt
CRR_p=[]
CRR_c=[]
for i in range(1,500):
    CRR_p.append(CRRTree('p',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
for i in range(1,500):
    CRR_c.append(CRRTree('C',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

```
plt.plot(range(1,500), CRR_p, c='g', label='Put')
plt.plot(range(1,500), CRR_c, c='orange', label='call')
plt.xlabel('price')
plt.ylabel('period')
plt.title('Put vs Call for 500 period for Cox-Ross-Rubinstein tree')
plt.legend()
plt.show()
```



Put vs Call for 500 period for Cox-Ross-Rubinstein tree
Put
call
period
price
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
0
100
200
300
400
500

BUILDING THE LOG NORMAL TREE

```python
def build_stock_tree_for_log_normal(S,T,N,sigma,Ldelta,r):
    dt=T/N;
    u = np.exp((r-Ldelta-(1/2)*sigma*sigma)*dt+sigma*np.sqrt(dt));
    d = np.exp((r-Ldelta-(1/2)*sigma*sigma)*dt-sigma*np.sqrt(dt));
    tree=np.zeros((N+1,N+1))
    for i in range(N+1):
        for j in range(i+1):
            tree[i][j]=S*pow(u,j)*pow(d,i-j)

    return(tree)

s = [[str(e) for e in row] for row in build_stock_tree_for_log_normal(Current_price,0.5,4,volatility,Ldelta,r)]
lens = [max(map(len, col)) for col in zip(*s)]
fmt = '\t'.join('{{:{}}}'.format(x) for x in lens)
table = [fmt.format(*row) for row in s]
print('\n'.join(table))
```

65.96
57.08
49.40
48.36
42.75
41.85
37.00
36.22
35.45
31.34
30.68
26.55
25.99
22.49
19.05

PRICE TREE FOR PUT AND CALL (LOG NORMAL)

```
import numpy as np
def LogNormalTree(type,S0, K, r, sigma,Ldelta, T, N):

    #calculate delta T
    deltaT = float(T) / N
    # up and down factor will be constant for the tree so we calculate outside the loop
    u = np.exp((r-Ldelta-(1/2)*sigma*sigma)*deltaT+sigma*np.sqrt(deltaT));
    d = np.exp((r-Ldelta-(1/2)*sigma*sigma)*deltaT-sigma*np.sqrt(deltaT));
    #to work with vector we need to init the arrays using numpy
    fs =  np.asarray([0.0 for i in range(N + 1)])
    #we need the stock tree for calculations of expiration values
    fs2 = np.asarray([(S0 * u**j * d**(N - j)) for j in range(N + 1)])
    #we vectorize the strikes as well so the expiration check will be faster
    fs3 =np.asarray( [float(K) for i in range(N + 1)])
    a = np.exp(r * deltaT)
    p=((np.exp((r-Ldelta)*deltaT))-d)/(u-d);
    oneMinusP = 1.0 - p
    if type =="C":
        fs[:] = np.maximum(fs2-fs3, 0.0)
    else:
        fs[:] = np.maximum(-fs2+fs3, 0.0)

    #calculate backward the option prices
    for i in range(N-1, -1, -1):
       fs[:-1]=np.exp(-r * deltaT) * (p * fs[1:] + oneMinusP * fs[:-1])
       fs2[:]=fs2[:]*u

    # print fs
    return fs[0]
```

PUT OPTION PRICES USING LOG NORMAL TREE

```
for i in range(1,10):
    print(LogNormalTree('p',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

5.4311071819904395
4.25760415424375
4.742222911588577
4.403133730123513
4.598560494055691
4.442254621233452
4.537110552681642
4.456954521878634
4.503092720593405

```
for i in range(100000,100010):
    print(LogNormalTree('p',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

4.4275773968652965
4.427580074757706
4.427577385880092
4.427580082922271
4.427577374946007
4.427580091538743
4.42757736366065
4.427580100155761
4.427577352876826
4.427580108519727

CALL OPTION PRICES USING LOG NORMAL TREE

```
for i in range(1,10):
    print(LogNormalTree('C',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

```
for i in range(100000,100010):
    print(LogNormalTree('C',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

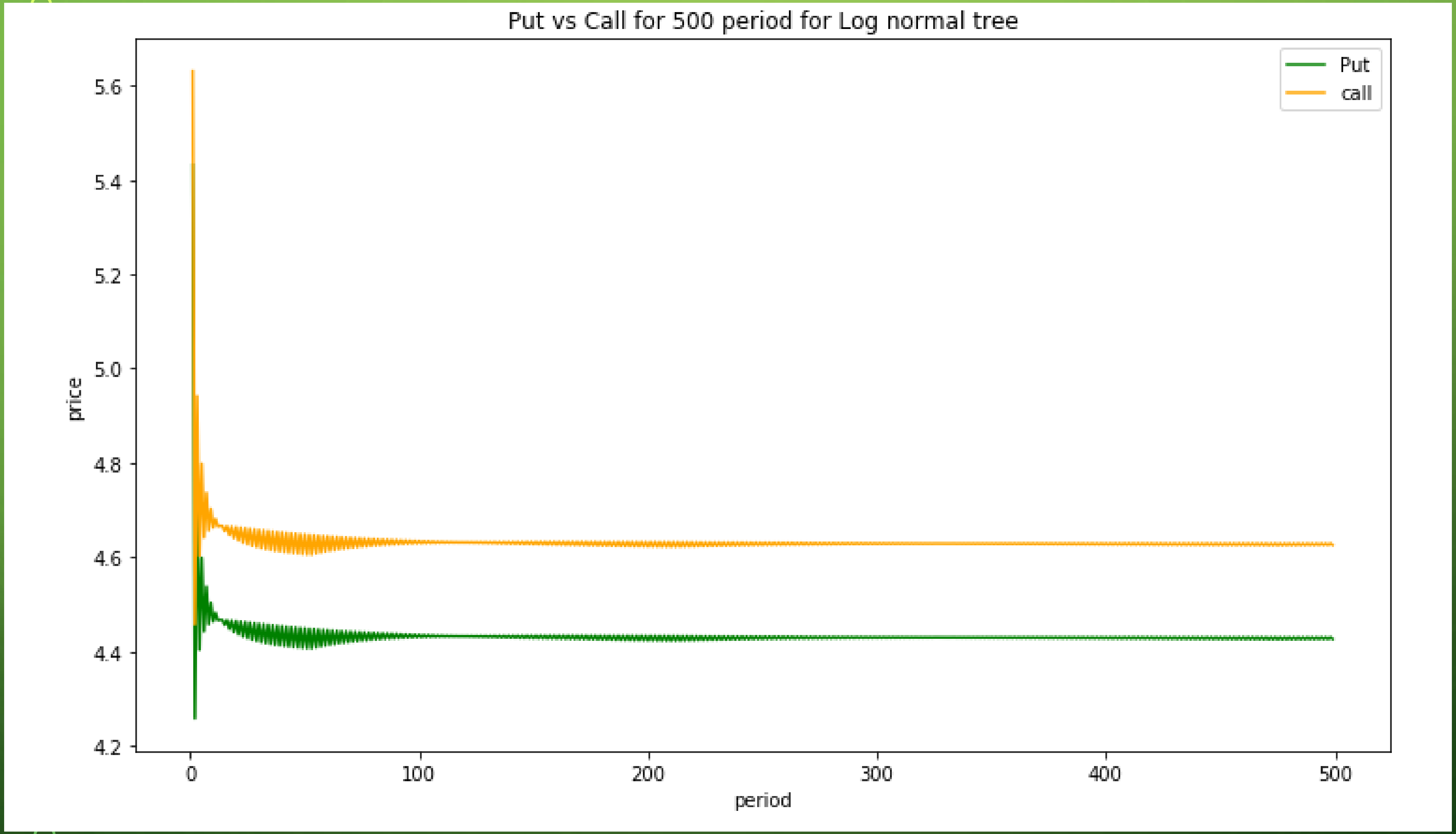
5.630274744484825
4.456771716738132
4.9413904740829695
4.602301292617906
4.797728056550064
4.641422183727843
4.736278115176005
4.656122084373023
4.702260283087781

4.6267449593851735
4.62674763687875
4.626744948379529
4.626747645759001
4.626744937265348
4.626747654116349
4.62674492645451
4.62674766231179
4.626744915172341
4.626747670913005

PUT VS CALL FOR 500 PERIOD FOR LOG NORMAL TREE

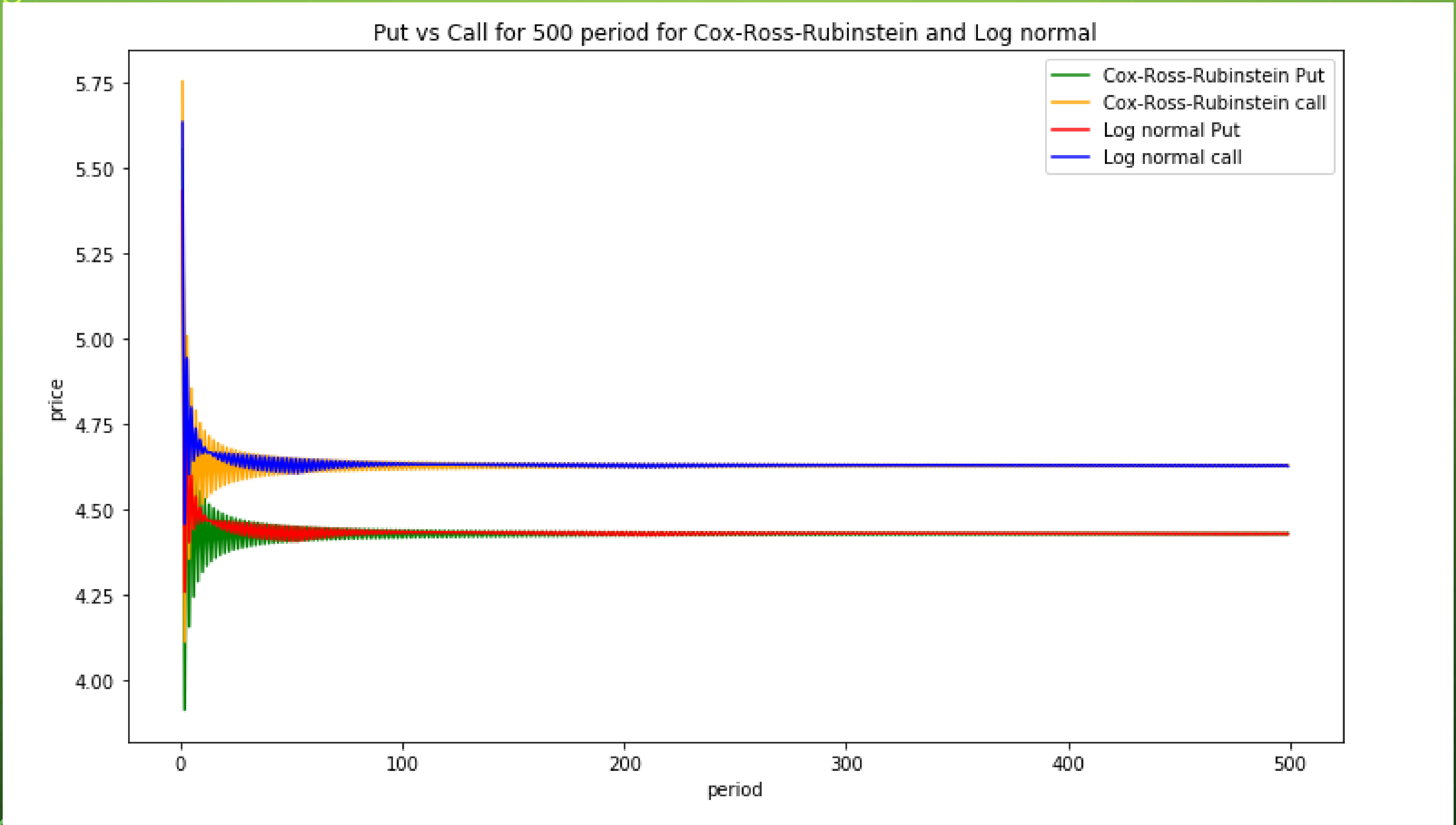
```
Ln_p=[]
Ln_c=[]
for i in range(1,500):
    Ln_p.append(LogNormalTree('p',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
for i in range(1,500):
    Ln_c.append(LogNormalTree('C',Current_price, Current_price, r,volatility,Ldelta, 0.5, N=i))
```

```
plt.plot(range(1,500), Ln_p, c='g', label='Put')
plt.plot(range(1,500), Ln_c, c='orange', label='call')
plt.xlabel('period')
plt.ylabel('price')
plt.title('Put vs Call for 500 period for Log normal tree')
plt.legend()
plt.show()
```


Put vs Call for 500 period for Log normal tree
Put
call
price
period
5.6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
0
100
200
300
400
500

PUT VS CALL FOR 500 PERIOD FOR COX-ROSS-RUBINSTEIN AND LOG NORMAL

```
plt.plot(range(1,500), CRR_p, c='g', label='Cox-Ross-Rubinstein Put')
plt.plot(range(1,500), CRR_c, c='orange', label='Cox-Ross-Rubinstein call')
plt.plot(range(1,500), Ln_p, c='red', label='Log normal Put')
plt.plot(range(1,500), Ln_c, c='blue', label='Log normal call')
plt.xlabel('period')
plt.ylabel('price')
plt.title('Put vs Call for 500 period for Cox-Ross-Rubinstein and Log normal')
plt.legend()
plt.show()
```


Put vs Call for 500 period for Cox-Ross-Rubinstein and Log normal
Cox-Ross-Rubinstein Put
Cox-Ross-Rubinstein call
Log normal Put
Log normal call
price
period
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
0
100
200
300
400
500

THANK YOU