Part 1

1. Construct **an additive binomial tree** to calculate the values of the **European Call option** and one for the European **Put** option.

Here is the code of the binomial tree for call and put option (using python):

```
def binom_tree(N,T,S0,sigma,r,K,call=True,array_out=False):
    dt=T/N
    u=np.exp((sigma*np.sqrt(dt)))
    d=1/u
    p=(np.exp(r*dt)-d)/(u-d)

    price_tree=np.zeros([N+1,N+1])

    for i in range(N+1):
        for j in range(i+1):
            price_tree[j,i]=S0*(d**j)*(u**(i-j))

    option=np.zeros([N+1,N+1])
    if call:

        option[:,N]=np.maximum(np.zeros(N+1),price_tree[:,N]-K)
    else:
        option[:,N]=np.maximum(np.zeros(N+1),K-price_tree[:,N])

    for i in np.arange(N-1,-1,-1):
        for j in np.arange(0,i+1):
            option[j,i]=np.exp(-r*dt)*(p*option[j,i+1]+(1-p)*option[j+1,i+1])

    if array_out:
        return [option[0,0],price_tree,option]
    else:
        return option[0,0]
```

2. Download Option prices (you can use the Bloomberg Terminal, Yahoo!Finance, etc.) for an equity, for 3 different maturities (1 month, 2months, and 3 months) and **20 strike prices**. For each strike price in the data, **calculate the corresponding implied volatility**. You may use the data and calculations you have done for Homework 1 (see Problem 1c).

1) I use the data of AMZN with maturities on 3-23, 4-20, 6-15. Part of the options' data is shown as follows:

3-23 call:

	contract_name	last_trade_date	strike_price	last_price	bid_price	ask_price	implied_volatility
0	AMZN180323C01060000	2018-02-02 11:46PM EST	1060.0	385.90	373.70	378.70	0.9212
1	AMZN180323C01070000	2018-02-02 11:46PM EST	1070.0	396.30	363.85	368.85	0.9016
2	AMZN180323C01100000	2018-02-02 11:46PM EST	1100.0	379.70	334.45	339.45	0.8442
3	AMZN180323C01170000	2018-02-02 11:46PM EST	1170.0	261.55	266.90	271.90	0.7189
4	AMZN180323C01190000	2018-02-05 12:22PM EST	1190.0	258.05	226.20	231.20	0.5317
5	AMZN180323C01200000	2018-02-05 9:32AM EST	1200.0	214.75	218.60	223.55	0.5313
6	AMZN180323C01240000	2018-02-05 10:10AM EST	1240.0	212.00	182.80	187.60	0.5016
7	AMZN180323C01250000	2018-02-05 2:58PM EST	1250.0	179.65	174.90	179.90	0.4971
8	AMZN180323C01270000	2018-02-05 2:52PM EST	1270.0	173.70	157.90	162.90	0.4763
9	AMZN180323C01290000	2018-02-02 11:46PM EST	1290.0	178.50	159.85	164.20	0.5458
10	AMZN180323C01310000	2018-02-05 9:43AM EST	1310.0	127.70	128.85	133.65	0.4555
11	AMZN180323C01340000	2018-02-05 1:37PM EST	1340.0	129.75	109.90	113.85	0.4434
12	AMZN180323C01350000	2018-02-05 11:40AM EST	1350.0	122.33	103.65	107.00	0.4365
13	AMZN180323C01360000	2018-02-05 12:34PM EST	1360.0	120.20	97.95	102.30	0.4397
14	AMZN180323C01370000	2018-02-05 1:31PM EST	1370.0	108.10	90.95	95.50	0.4313
15	AMZN180323C01400000	2018-02-05 1:44PM EST	1400.0	90.44	74.65	78.20	0.4165
16	AMZN180323C01415000	2018-02-05 3:05PM EST	1415.0	70.00	67.30	71.25	0.4145

Put:

	contract_name	last_trade_date	strike_price	last_price	bid_price	ask_price	implied_volatility
0	AMZN180323P01060000	2018-02-05 2:18PM EST	1060.0	1.99	2.21	3.85	0.4519
1	AMZN180323P01130000	2018-02-05 11:40AM EST	1130.0	2.80	5.10	6.75	0.4117
2	AMZN180323P01170000	2018-02-05 12:55PM EST	1170.0	4.24	7.45	9.80	0.3950
3	AMZN180323P01180000	2018-02-02 11:48PM EST	1180.0	5.67	5.25	6.20	0.3361
4	AMZN180323P01190000	2018-02-02 11:48PM EST	1190.0	4.50	5.40	6.90	0.3319
5	AMZN180323P01200000	2018-02-05 2:50PM EST	1200.0	8.50	10.20	13.55	0.3891
6	AMZN180323P01240000	2018-02-05 3:06PM EST	1240.0	17.95	16.00	20.30	0.3815
7	AMZN180323P01250000	2018-02-05 2:07PM EST	1250.0	11.00	17.50	22.00	0.3771
8	AMZN180323P01260000	2018-02-05 2:54PM EST	1260.0	17.78	19.20	23.50	0.3705
9	AMZN180323P01270000	2018-02-05 2:24PM EST	1270.0	15.20	21.15	25.95	0.3695
10	AMZN180323P01280000	2018-02-05 9:43AM EST	1280.0	21.82	23.30	28.05	0.3650
11	AMZN180323P01300000	2018-02-05 1:19PM EST	1300.0	20.35	28.15	32.90	0.3574
12	AMZN180323P01310000	2018-02-05 2:22PM EST	1310.0	21.90	30.95	35.70	0.3542
13	AMZN180323P01320000	2018-02-05 2:22PM EST	1320.0	24.55	33.70	38.50	0.3500
14	AMZN180323P01330000	2018-02-05 10:54AM EST	1330.0	20.93	36.90	41.65	0.3467
15	AMZN180323P01340000	2018-02-05 9:43AM EST	1340.0	38.50	40.80	45.15	0.3442
16	AMZN180323P01350000	2018-02-05 2:22PM EST	1350.0	32.92	43.85	48.60	0.3404
17	AMZN180323P01360000	2018-02-05 2:22PM EST	1360.0	36.00	47.85	52.65	0.3385

4-20call:

	contract_name	last_trade_date	strike_price	last_price	bid_price	ask_price	implied_volatility
0	AMZN180420C00620000	2018-01-25 3:11PM EST	620.0	750.00	782.25	786.75	1.1916
1	AMZN180420C00640000	2018-01-16 10:06AM EST	640.0	688.40	656.00	660.70	0.0000
2	AMZN180420C00660000	2018-01-16 3:08PM EST	660.0	653.10	636.00	640.80	0.0000
3	AMZN180420C00680000	2018-02-02 3:12PM EST	680.0	757.35	752.95	757.95	1.4723
4	AMZN180420C00700000	2018-02-05 11:41AM EST	700.0	742.35	706.10	714.90	1.1274
5	AMZN180420C00720000	2018-02-02 3:10PM EST	720.0	711.80	713.25	718.25	1.3799
6	AMZN180420C00740000	2018-02-05 11:41AM EST	740.0	702.60	665.60	670.60	1.0172
7	AMZN180420C00760000	2018-02-02 3:13PM EST	760.0	679.60	673.55	678.55	1.2924
8	AMZN180420C00780000	2018-02-05 11:41AM EST	780.0	662.90	626.40	635.50	0.9833
9	AMZN180420C00800000	2018-02-02 3:12PM EST	800.0	638.05	633.75	638.75	1.2085
10	AMZN180420C00820000	2017-08-25 10:57PM EST	820.0	169.40	183.15	186.90	0.0000
11	AMZN180420C00835000	2017-12-01 9:49AM EST	835.0	345.25	335.60	340.50	0.0000
12	AMZN180420C00840000	2018-01-09 1:51PM EST	840.0	416.05	468.95	472.05	0.0000
13	AMZN180420C00845000	2017-11-28 1:52PM EST	845.0	360.00	324.75	329.50	0.0000
14	AMZN180420C00860000	2018-02-02 9:43AM EST	860.0	623.00	574.35	579.35	1.0921

4-20Put:

	contract_name	last_trade_date	strike_price	last_price	bid_price	ask_price	implied_volatility
0	AMZN180420P00620000	2018-02-01 3:36PM EST	620.0	0.40	0.05	5.00	0.8615
1	AMZN180420P00640000	2018-01-26 3:50PM EST	640.0	0.16	0.00	0.16	0.5635
2	AMZN180420P00660000	2018-01-05 3:16PM EST	660.0	0.44	0.01	0.74	0.6279
3	AMZN180420P00680000	2018-02-05 9:56AM EST	680.0	0.18	0.00	1.87	0.6721
4	AMZN180420P00700000	2018-01-24 3:02PM EST	700.0	0.25	0.00	0.82	0.5872
5	AMZN180420P00720000	2018-02-05 1:54PM EST	720.0	0.31	0.10	1.97	0.6306
6	AMZN180420P00740000	2018-01-12 3:49PM EST	740.0	0.39	0.25	0.89	0.5637
7	AMZN180420P00760000	2018-02-02 12:56PM EST	760.0	0.30	0.00	5.00	0.6614
8	AMZN180420P00780000	2018-02-01 2:37PM EST	780.0	0.81	0.00	2.14	0.5627
9	AMZN180420P00800000	2018-02-05 2:36PM EST	800.0	0.45	0.45	2.30	0.5592
10	AMZN180420P00820000	2018-01-30 1:42PM EST	820.0	0.69	0.00	2.32	0.5242
11	AMZN180420P00835000	2018-02-05 10:21AM EST	835.0	0.43	0.00	2.47	0.5123
12	AMZN180420P00840000	2018-02-02 10:21AM EST	840.0	0.52	0.00	5.00	0.5631
13	AMZN180420P00845000	2018-02-05 10:21AM EST	845.0	0.51	0.19	2.60	0.5103
14	AMZN180420P00860000	2018-01-31 1:22PM EST	860.0	0.86	0.00	5.00	0.5399
15	AMZN180420P00870000	2018-02-02 10:18AM EST	870.0	0.56	0.44	1.58	0.4923
16	AMZN180420P00875000	2018-01-16 12:40PM EST	875.0	1.12	0.86	1.91	0.5010
17	AMZN180420P00880000	2018-01-26 3:11PM EST	880.0	1.20	0.34	1.55	0.4802

6-15call:

	contract_name	last_trade_date	strike_price	last_price	bid_price	ask_price	implied_volatility
0	AMZN180615C00420000	2018-01-11 1:18PM EST	420.0	851.35	887.20	891.30	0.0000
1	AMZN180615C00430000	2018-01-11 1:18PM EST	430.0	841.45	877.25	881.40	0.0000
2	AMZN180615C00440000	2018-02-05 10:10AM EST	440.0	1003.90	953.30	958.30	1.0385
3	AMZN180615C00450000	2018-02-02 9:54AM EST	450.0	1007.60	983.85	988.85	1.6646
4	AMZN180615C00460000	2018-01-10 10:35AM EST	460.0	787.85	847.55	851.70	0.0000
5	AMZN180615C00470000	2018-02-02 9:42AM EST	470.0	1010.85	964.00	969.00	1.6079
6	AMZN180615C00480000	2018-02-05 10:12AM EST	480.0	965.90	913.75	918.75	0.9780
7	AMZN180615C00490000	2017-12-01 11:45PM EST	490.0	686.45	676.00	681.00	0.0000
8	AMZN180615C00500000	2018-01-17 12:07PM EST	500.0	798.85	797.50	802.00	0.0000
9	AMZN180615C00520000	2018-02-02 1:04PM EST	520.0	944.45	914.50	919.50	1.4789
10	AMZN180615C00540000	2018-02-02 9:46AM EST	540.0	932.25	894.70	899.70	1.4312
11	AMZN180615C00560000	2017-12-04 4:26PM EST	560.0	585.10	579.55	584.40	0.0000
12	AMZN180615C00580000	2017-12-04 9:34AM EST	580.0	592.45	560.00	563.90	0.0000
13	AMZN180615C00600000	2018-02-02 3:58PM EST	600.0	838.05	835.65	840.65	1.3022
14	AMZN180615C00620000	2018-01-18 3:37PM EST	620.0	679.37	678.50	683.30	0.0000
15	AMZN180615C00640000	2017-12-04 10:57AM EST	640.0	499.95	501.70	506.60	0.0000
16	AMZN180615C00650000	2017-10-27 10:39AM EST	650.0	449.25	458.65	463.35	0.0000

6-20Put:

	contract_name	last_trade_date	strike_price	last_price	bid_price	ask_price	implied_volatility
0	AMZN180615P00420000	2018-02-05 9:52AM EST	420.0	0.10	0.04	0.20	0.6709
1	AMZN180615P00430000	2018-01-22 10:46AM EST	430.0	0.10	0.00	0.33	0.6777
2	AMZN180615P00440000	2018-01-23 12:07PM EST	440.0	0.05	0.00	0.35	0.6680
3	AMZN180615P00450000	2018-01-16 3:15PM EST	450.0	0.09	0.00	0.35	0.6553
4	AMZN180615P00460000	2018-01-23 12:07PM EST	460.0	0.08	0.00	0.41	0.6528
5	AMZN180615P00470000	2018-02-02 3:57PM EST	470.0	0.25	0.00	0.30	0.6226
6	AMZN180615P00480000	2018-01-23 12:08PM EST	480.0	0.11	0.00	0.49	0.6397
7	AMZN180615P00490000	2018-02-02 3:51PM EST	490.0	0.25	0.00	0.91	0.6687
8	AMZN180615P00500000	2018-02-02 3:51PM EST	500.0	0.30	0.10	0.70	0.6475
9	AMZN180615P00520000	2017-12-20 1:17PM EST	520.0	0.45	0.00	0.74	0.6187
10	AMZN180615P00540000	2018-02-05 9:30AM EST	540.0	0.45	0.00	2.38	0.6823
11	AMZN180615P00560000	2018-01-23 11:58AM EST	560.0	0.40	0.00	0.79	0.5784
12	AMZN180615P00580000	2018-01-12 11:10AM EST	580.0	0.30	0.00	0.59	0.5405
13	AMZN180615P00600000	2018-02-05 10:09AM EST	600.0	0.50	0.05	1.57	0.5825
14	AMZN180615P00620000	2018-01-23 12:15PM EST	620.0	0.45	0.05	0.94	0.5300
15	AMZN180615P00640000	2018-01-12 9:30AM EST	640.0	0.24	0.18	0.78	0.5088
16	AMZN180615P00650000	2018-02-02 9:53AM EST	650.0	0.82	0.36	1.33	0.5335
17	AMZN180615P00660000	2018-01-22 3:08PM EST	660.0	0.68	0.00	1.08	0.5399
18	AMZN180615P00670000	2018-02-05 11:32AM EST	670.0	0.58	0.02	2.77	0.5480

2) Calculate the option prices using the binomial tree:

Here is an example of code of calculating AMZN call option price with a maturity of one month.

```
iv1=np.array(iv1_float)
iv_1=iv1.tolist()
sp1=np.array(data_1['strike_price'])
sp_1=sp1.tolist()
```

```
filename='AMZNcallp1m.csv'
f=open(filename,'w')

headers='price\n'
f.write(headers)

for i,j in zip(iv_1,sp_1):

    K=j
    sigma=i
    callprice=binom_tree(200,0.083,1390,i,0.0149,j,call=True,array_out=False)

    print(callprice)

    f.write(str(callprice)+'\n')

f.close()
```

Result:

```
356.691550682
346.912120642
317.880294041
250.982028508
217.148295682
208.931563291
174.390008927
166.276260508
149.103999645
144.355535175
118.766515216
98.2603204332
91.5451183492
86.4135575867
79.7279720575
62.7227759523
55.7017288787
51.4846483264
49.5815532906
48.4157457596
```

And option prices calculating by BSM:

```
def C(sigma):
    d1=(np.log(S0/K)+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2=d1-sigma*np.sqrt(T)
    C=S0*norm.cdf(d1)-K*norm.cdf(d2)
    return C
```

```
filename='AMZNcallp1m_1.csv'
f=open(filename,'w')

headers='price_1\n'
f.write(headers)

for i,j in zip(iv_1,sp_1):

    K=j
    sigma=i
    callprice_1=C(i)

    print(callprice_1)

    f.write(str(callprice_1)+'\n')

f.close()
```

```
355.625769411
345.887299193
316.792088612
249.905267276
215.944690137
207.721090148
173.164517025
165.097368067
147.915626727
143.238219801
117.694713454
97.3474010101
90.5233842625
85.4641799189
78.8246747464
61.8540967287
55.013862548
50.703495459
48.8396691931
47.7168330258
```

3. Compute and plot the absolute error. Consider cases of N=50, 100, 150, 200, 250, 300, 350, 400.

1) Here is an example of AMZN one-month call option:

First put all the option prices of different cases in one table, including prices from both binomial tree and BSM:

```
for i,j in zip(iv_1,sp_1):

    K=j
    sigma=i
    callprice50=binom_tree_call(50,0.083,1390,i,0.0149,j,array_out=False)
    callprice100=binom_tree_call(100,0.083,1390,i,0.0149,j,array_out=False)
    callprice150=binom_tree_call(150,0.083,1390,i,0.0149,j,array_out=False)
    callprice200=binom_tree_call(200,0.083,1390,i,0.0149,j,array_out=False)
    callprice250=binom_tree_call(250,0.083,1390,i,0.0149,j,array_out=False)
    callprice300=binom_tree_call(300,0.083,1390,i,0.0149,j,array_out=False)
    callprice350=binom_tree_call(350,0.083,1390,i,0.0149,j,array_out=False)
    callprice400=binom_tree_call(400,0.083,1390,i,0.0149,j,array_out=False)


    K=j
    sigma=i
    callprice_1=C(i)
```

	50	100	150	200	250	300	350	400	BS
0	356.916156	356.573495	356.707869	356.691551	356.630315	356.675424	356.726485	356.690574	355.625769
1	347.205894	346.757757	346.927383	346.912121	346.881368	346.978026	346.992222	346.921798	345.887299
2	318.097711	317.875246	317.818385	317.880294	317.927208	317.908692	317.811588	317.908601	316.792089
3	250.736889	250.980113	250.946379	250.982029	251.070783	251.019572	251.024987	251.039317	249.905267
4	217.310141	217.064157	217.158162	217.148296	217.106174	217.164206	217.184057	217.150896	215.944690
5	209.069113	208.930390	208.892918	208.931563	208.959644	208.944444	208.887193	208.947181	207.721090
6	174.383427	174.227020	174.330329	174.390009	174.352969	174.333044	174.365826	174.322246	173.164517
7	166.466943	166.330727	166.334855	166.276261	166.246751	166.293779	166.230561	166.284505	165.097368
8	149.222540	149.122878	148.964633	149.104000	149.061175	149.081091	149.049256	149.083488	147.915627
9	144.546219	144.431670	144.237190	144.355535	144.317285	144.267547	144.333158	144.306283	143.238220
10	119.030917	118.823656	118.793981	118.766515	118.810154	118.731334	118.767570	118.790128	117.694713
11	98.022928	98.482822	98.378719	98.260320	98.363000	98.381470	98.361595	98.323520	97.347401
12	91.534648	91.484720	91.579447	91.545118	91.482143	91.424524	91.481714	91.506360	90.523384
13	86.692085	86.376228	86.315906	86.413558	86.440801	86.440207	86.427529	86.409500	85.464180
14	80.030094	79.874434	79.786187	79.727972	79.685845	79.662624	79.698102	79.719092	78.824675
15	62.752546	62.755526	62.738385	62.722776	62.709814	62.699044	62.689970	62.682207	61.854097
16	56.076717	55.835089	55.709063	55.701729	55.761654	55.785994	55.793173	55.791459	55.013863

Then calculate the absolute error:

```
series1_50=data_1all['50']
series1_100=data_1all['100']
series1_150=data_1all['150']
series1_200=data_1all['200']
series1_250=data_1all['250']
series1_300=data_1all['300']
series1_350=data_1all['350']
series1_400=data_1all['400']

series1_BS=data_1all['BS']
```

```
error50=abs(series1_50-series1_BS)
error100=abs(series1_100-series1_BS)
error150=abs(series1_150-series1_BS)
error200=abs(series1_200-series1_BS)
error250=abs(series1_250-series1_BS)
error300=abs(series1_300-series1_BS)
error350=abs(series1_350-series1_BS)
error400=abs(series1_400-series1_BS)
```

```
data_1all['err50']=error50
data_1all['err100']=error100
data_1all['err150']=error150
data_1all['err200']=error200
data_1all['err250']=error250
data_1all['err300']=error300
data_1all['err350']=error350
data_1all['err400']=error400
```

Put the results in one table:

50	200	250	300	350	400	BS	err50	err100	err150	err200	err250	err300	err350	err400
59	356.691551	356.630315	356.675424	356.726485	356.690574	355.625769	1.290386	0.947725	1.082100	1.065781	1.004545	1.049655	1.100715	1.064805
33	346.912121	346.881368	346.978026	346.992222	346.921798	345.887299	1.318595	0.870458	1.040084	1.024821	0.994069	1.090727	1.104923	1.034499
35	317.880294	317.927208	317.908692	317.811588	317.908601	316.792089	1.305622	1.083157	1.026296	1.088205	1.135119	1.116604	1.019500	1.116512
79	250.982029	251.070783	251.019572	251.024987	251.039317	249.905267	0.831621	1.074845	1.041112	1.076761	1.165516	1.114305	1.119720	1.134050
32	217.148296	217.106174	217.164206	217.184057	217.150896	215.944690	1.365451	1.119467	1.213472	1.203606	1.161484	1.219516	1.239366	1.206206
18	208.931563	208.959644	208.944444	208.887193	208.947181	207.721090	1.348023	1.209299	1.171828	1.210473	1.238554	1.223353	1.166103	1.226091
29	174.390009	174.352969	174.333044	174.365826	174.322246	173.164517	1.218910	1.062503	1.165812	1.225492	1.188452	1.168527	1.201309	1.157729
55	166.276261	166.246751	166.293779	166.230561	166.284505	165.097368	1.369575	1.233359	1.237487	1.178892	1.149383	1.196411	1.133192	1.187137
33	149.104000	149.061175	149.081091	149.049256	149.083488	147.915627	1.306913	1.207251	1.049006	1.188373	1.145548	1.165464	1.133630	1.167862
90	144.355535	144.317285	144.267547	144.333158	144.306283	143.238220	1.307999	1.193450	0.998970	1.117315	1.079065	1.029327	1.094938	1.068063
31	118.766515	118.810154	118.731334	118.767570	118.790128	117.694713	1.336203	1.128942	1.099268	1.071802	1.115441	1.036621	1.072856	1.095415
19	98.260320	98.363000	98.381470	98.361595	98.323520	97.347401	0.675527	1.135421	1.031318	0.912919	1.015599	1.034069	1.014194	0.976119
47	91.545118	91.482143	91.424524	91.481714	91.506360	90.523384	1.011264	0.961336	1.056063	1.021734	0.958758	0.901139	0.958330	0.982976
06	86.413558	86.440801	86.440207	86.427529	86.409500	85.464180	1.227905	0.912048	0.851726	0.949378	0.976621	0.976027	0.963349	0.945320
37	79.727972	79.685845	79.662624	79.698102	79.719092	78.824675	1.205420	1.049759	0.961512	0.903297	0.861170	0.837950	0.873427	0.894417
35	62.722776	62.709814	62.699044	62.689970	62.682207	61.854097	0.898449	0.901430	0.884288	0.868679	0.855717	0.844948	0.835873	0.828110

From the data showing above, it could be observed that as N increases, the option prices from binomial tree get closer to the prices calculating from BSM, indicating that the more steps the binomial tree have, the more accurate the result is.

2) AMZN 2-month call:

	50	100	150	200	250	300	350	400	BS
0	780.182171	780.333533	780.303146	780.358403	780.320458	780.368860	780.375655	780.374436	778.952614
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	750.000000
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	730.000000
3	743.818060	743.699672	743.991504	744.036418	744.023936	743.992011	743.925709	743.936668	742.605803
4	704.871642	704.987969	704.865374	704.976993	704.942184	704.912670	704.909887	704.925302	703.398740
5	704.082217	704.369265	704.354287	704.448749	704.437611	704.382317	704.388885	704.474297	702.997251
6	663.375130	663.402315	663.359436	663.368351	663.289325	663.338177	663.354075	663.353552	661.710754
7	664.299302	664.993892	664.670328	664.814504	664.804749	664.868687	664.953668	664.966094	663.399630
8	625.302857	625.558112	625.599969	625.590781	625.644064	625.649644	625.647707	625.642346	623.905285
9	624.796713	625.504403	625.297684	625.263573	625.327137	625.391829	625.406002	625.347222	623.754128
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	570.000000
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	555.000000
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	550.000000
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	545.000000
14	566.240390	566.390179	566.369271	566.340708	566.334240	566.297564	566.199532	566.276883	564.550838
15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	520.000000
16	551.488340	551.509217	551.531169	551.503957	551.481370	551.421433	551.341367	551.453254	549.677023
17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	510.000000

Absolute error:

	150	200	250	300	350	400	BS	err50	err100	err150	err200	err250	err300	err350	err400
	780.303146	780.358403	780.320458	780.368860	780.375655	780.374436	778.952614	1.229557	1.380920	1.350532	1.405789	1.367844	1.416246	1.423041	1.421822
	NaN	NaN	NaN	NaN	NaN	NaN	750.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	730.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	743.991504	744.036418	744.023936	743.992011	743.925709	743.936668	742.605803	1.212257	1.093868	1.385700	1.430615	1.418133	1.386208	1.319905	1.330864
	704.865374	704.976993	704.942184	704.912670	704.909887	704.925302	703.398740	1.472902	1.589228	1.466634	1.578253	1.543444	1.513930	1.511147	1.526561
	704.354287	704.448749	704.437611	704.382317	704.388885	704.474297	702.997251	1.084966	1.372014	1.357036	1.451498	1.440360	1.385066	1.391634	1.477046
	663.359436	663.368351	663.289325	663.338177	663.354075	663.353552	661.710754	1.664375	1.691561	1.648682	1.657597	1.578571	1.627423	1.643321	1.642798
	664.670328	664.814504	664.804749	664.868687	664.953668	664.966094	663.399630	0.899672	1.594262	1.270698	1.414874	1.405118	1.469057	1.554038	1.566464
	625.599969	625.590781	625.644064	625.649644	625.647707	625.642346	623.905285	1.397572	1.652827	1.694684	1.685496	1.738779	1.744359	1.742422	1.737061
	625.297684	625.263573	625.327137	625.391829	625.406002	625.347222	623.754128	1.042585	1.750275	1.543556	1.509445	1.573009	1.637701	1.651874	1.593093
	NaN	NaN	NaN	NaN	NaN	NaN	570.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	555.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	550.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	545.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	566.369271	566.340708	566.334240	566.297564	566.199532	566.276883	564.550838	1.689552	1.839341	1.818433	1.789870	1.783402	1.746726	1.648694	1.726045
	NaN	NaN	NaN	NaN	NaN	NaN	520.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	551.531169	551.503957	551.481370	551.421433	551.341367	551.453254	549.677023	1.811318	1.832194	1.854146	1.826934	1.804347	1.744410	1.664344	1.776231
	NaN	NaN	NaN	NaN	NaN	NaN	510.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3) AMZN 3-month call:

	50	100	150	200	250	300	350	400	BS
0	793.572992	793.626397	793.660627	793.696929	793.665950	793.659657	793.671233	793.680808	791.688711
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	750.000000
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	730.000000
3	772.675819	772.055448	771.930837	772.105960	772.258683	772.268200	772.101063	772.221590	770.348149
4	721.528104	721.243165	721.555620	721.532269	721.518860	721.519666	721.505024	721.452888	719.327421
5	732.856379	732.515636	732.399373	732.505091	732.565908	732.480467	732.374195	732.525665	730.503475
6	678.540274	678.324581	678.464239	678.393668	678.380008	678.403747	678.425808	678.419558	676.088706
7	692.937637	692.880665	692.773177	692.808562	692.776356	692.595080	692.714457	692.733054	690.619702
8	641.752160	641.866402	641.959482	642.026834	642.016076	641.974268	641.903095	642.005345	639.608033
9	652.830919	653.052351	652.953884	652.920286	652.797079	652.695006	652.862577	652.750955	650.608735
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	570.000000
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	555.000000
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	550.000000
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	545.000000
14	592.779624	593.445604	593.361579	593.218140	593.069845	593.289730	593.218471	593.214024	590.873869
15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	520.000000

	150	200	250	300	350	400	BS	err50	err100	err150	err200	err250	err300	err350	err400
	793.660627	793.696929	793.665950	793.659657	793.671233	793.680808	791.688711	1.884281	1.937686	1.971916	2.008218	1.977238	1.970946	1.982522	1.992096
	NaN	NaN	NaN	NaN	NaN	NaN	750.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	730.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	771.930837	772.105960	772.258683	772.268200	772.101063	772.221590	770.348149	2.327670	1.707300	1.582688	1.757811	1.910534	1.920052	1.752915	1.873442
	721.555620	721.532269	721.518860	721.519666	721.505024	721.452888	719.327421	2.200683	1.915744	2.228200	2.204848	2.191439	2.192245	2.177603	2.125467
	732.399373	732.505091	732.565908	732.480467	732.374195	732.525665	730.503475	2.352904	2.012161	1.895897	2.001616	2.062433	1.976992	1.870720	2.022189
	678.464239	678.393668	678.380008	678.403747	678.425808	678.419558	676.088706	2.451568	2.235875	2.375533	2.304962	2.291302	2.315042	2.337102	2.330852
	692.773177	692.808562	692.776356	692.595080	692.714457	692.733054	690.619702	2.317934	2.260963	2.153475	2.188860	2.156653	1.975378	2.094755	2.113351
	641.959482	642.026834	642.016076	641.974268	641.903095	642.005345	639.608033	2.144127	2.258369	2.351449	2.418801	2.408043	2.366235	2.295062	2.397312
	652.953884	652.920286	652.797079	652.695006	652.862577	652.750955	650.608735	2.222184	2.443616	2.345149	2.311551	2.188344	2.086271	2.253842	2.142220
	NaN	NaN	NaN	NaN	NaN	NaN	570.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	555.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	550.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	545.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	593.361579	593.218140	593.069845	593.289730	593.218471	593.214024	590.873869	1.905754	2.571735	2.487710	2.344271	2.195975	2.415860	2.344602	2.340155
	NaN	NaN	NaN	NaN	NaN	NaN	520.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	578.305589	578.137122	578.120792	578.277161	578.150872	578.224264	575.811220	1.807757	2.574839	2.494368	2.325901	2.309571	2.465941	2.339651	2.413043

4) AMZN one-month put:

	50	100	150	200	250	300	350	400	BS
0	1.019414	1.034670	1.034594	1.037873	1.041254	1.031460	1.042278	1.039290	1.072544
1	2.299188	2.351002	2.356893	2.358073	2.353287	2.349254	2.359495	2.361705	2.424562
2	4.009064	4.020272	4.002801	4.008748	3.982553	3.994829	4.000705	4.000502	4.112518
3	2.138503	2.241503	2.237952	2.217483	2.239650	2.240069	2.234859	2.231547	2.312149
4	2.595572	2.575992	2.579444	2.612461	2.600628	2.595754	2.604532	2.607055	2.692706
5	6.317543	6.301172	6.221063	6.274499	6.281664	6.280147	6.274623	6.263343	6.427680
6	10.975801	10.885661	10.923942	10.916042	10.896779	10.861363	10.905060	10.909251	11.143090
7	12.289936	12.134448	12.106100	12.147337	12.185784	12.194417	12.165591	12.177537	12.445350
8	13.397139	13.427240	13.403125	13.399256	13.376496	13.335523	13.383114	13.365668	13.661991
9	15.022230	15.278478	15.259165	15.201478	15.278114	15.278470	15.240269	15.278959	15.590017
10	17.016558	16.902614	16.969546	17.002487	16.964066	16.967353	16.978553	16.959497	17.323966
11	21.157808	21.084316	20.974881	21.091714	21.038791	21.074745	21.029995	21.071323	21.482394
12	23.294203	23.414705	23.527116	23.439606	23.498461	23.426606	23.488132	23.464284	23.936044
13	25.997791	26.062092	25.944094	26.003810	25.922763	25.990852	25.964241	25.949447	26.474299
14	29.023253	28.827650	28.849054	28.786075	28.844526	28.802943	28.793417	28.825604	29.362683
15	32.158847	32.001344	32.040897	32.054111	31.977413	32.009290	32.033603	32.020764	32.612484
16	35.104180	35.390695	35.279510	35.248879	35.307393	35.306257	35.278909	35.239814	35.929157
17	39.169803	39.061162	39.158695	39.146335	39.107174	39.062141	39.072834	39.099522	39.801013

100	150	200	250	300	350	400	BS	err50	err100	err150	err200	err250	err300	err350	err400
1.034670	1.034594	1.037873	1.041254	1.031460	1.042278	1.039290	1.072544	0.053130	0.037874	0.037950	0.034670	0.031290	0.041084	0.030266	0.033253
2.351002	2.356893	2.358073	2.353287	2.349254	2.359495	2.361705	2.424562	0.125373	0.073559	0.067669	0.066489	0.071275	0.075308	0.065067	0.062857
4.020272	4.002801	4.008748	3.982553	3.994829	4.000705	4.000502	4.112518	0.103454	0.092246	0.109716	0.103770	0.129965	0.117689	0.111813	0.112016
2.241503	2.237952	2.217483	2.239650	2.240069	2.234859	2.231547	2.312149	0.173646	0.070647	0.074197	0.094666	0.072500	0.072081	0.077290	0.080602
2.575992	2.579444	2.612461	2.600628	2.595754	2.604532	2.607055	2.692706	0.097134	0.116714	0.113261	0.080245	0.092078	0.096952	0.088173	0.085651
6.301172	6.221063	6.274499	6.281664	6.280147	6.274623	6.263343	6.427680	0.110138	0.126509	0.206617	0.153182	0.146016	0.147533	0.153057	0.164337
0.885661	10.923942	10.916042	10.896779	10.861363	10.905060	10.909251	11.143090	0.167288	0.257429	0.219148	0.227048	0.246310	0.281727	0.238030	0.233839
2.134448	12.106100	12.147337	12.185784	12.194417	12.165591	12.177537	12.445350	0.155413	0.310902	0.339250	0.298013	0.259566	0.250933	0.279759	0.267813
3.427240	13.403125	13.399256	13.376496	13.335523	13.383114	13.365668	13.661991	0.264853	0.234751	0.258866	0.262736	0.285496	0.326468	0.278877	0.296323
5.278478	15.259165	15.201478	15.278114	15.278470	15.240269	15.278959	15.590017	0.567787	0.311538	0.330851	0.388539	0.311903	0.311547	0.349748	0.311057
5.902614	16.969546	17.002487	16.964066	16.967353	16.978553	16.959497	17.323966	0.307409	0.421352	0.354420	0.321479	0.359900	0.356613	0.345413	0.364470
1.084316	20.974881	21.091714	21.038791	21.074745	21.029995	21.071323	21.482394	0.324586	0.398079	0.507513	0.390680	0.443604	0.407649	0.452399	0.411071
3.414705	23.527116	23.439606	23.498461	23.426606	23.488132	23.464284	23.936044	0.641841	0.521338	0.408927	0.496438	0.437583	0.509437	0.447911	0.471760
3.062092	25.944094	26.003810	25.922763	25.990852	25.964241	25.949447	26.474299	0.476508	0.412207	0.530205	0.470489	0.551537	0.483447	0.510058	0.524852
3.827650	28.849054	28.786075	28.844526	28.802943	28.793417	28.825604	29.362683	0.339430	0.535034	0.513630	0.576608	0.518157	0.559740	0.569267	0.537080
2.001344	32.040897	32.054111	31.977413	32.009290	32.033603	32.020764	32.612484	0.453637	0.611140	0.571587	0.558374	0.635071	0.603195	0.578882	0.591720
5.390695	35.279510	35.248879	35.307393	35.306257	35.278909	35.239814	35.929157	0.824976	0.538462	0.649647	0.680278	0.621764	0.622900	0.650248	0.689343
9.061162	39.158695	39.146335	39.107174	39.062141	39.072834	39.099522	39.801013	0.631209	0.739851	0.642318	0.654678	0.693839	0.738872	0.728179	0.701490

5) AMZN 2-month put:

	50	100	150	200	250	300	350	400	BS
0	1.083065	1.161088	1.141974	1.162360	1.161833	1.166180	1.174122	1.166695	1.205196
1	0.015015	0.017829	0.018956	0.019071	0.019481	0.019403	0.019722	0.019759	0.021159
2	0.109421	0.120168	0.121192	0.123353	0.121828	0.123104	0.123669	0.123357	0.130898
3	0.335300	0.345387	0.356715	0.366607	0.368145	0.368940	0.368887	0.366006	0.383838
4	0.118249	0.133170	0.137138	0.136818	0.138848	0.138781	0.139235	0.139859	0.146721
5	0.364834	0.406614	0.413901	0.408360	0.413946	0.413995	0.415704	0.419044	0.435205
6	0.193455	0.201683	0.203688	0.205815	0.207079	0.207359	0.208109	0.208205	0.218225
7	1.110753	1.153750	1.177862	1.176212	1.183136	1.183645	1.178571	1.185240	1.223407
8	0.401923	0.432409	0.434938	0.435512	0.435081	0.438126	0.440149	0.437661	0.458752
9	0.573911	0.567926	0.584513	0.583185	0.586636	0.587579	0.594052	0.591889	0.618183
10	0.465889	0.477068	0.474558	0.485940	0.487423	0.487252	0.483657	0.488823	0.510001
11	0.508078	0.508508	0.513841	0.520754	0.523336	0.521348	0.525601	0.526959	0.550274
12	1.169275	1.162690	1.198347	1.202131	1.200362	1.192610	1.205000	1.199249	1.247519
13	0.588411	0.584615	0.605856	0.601536	0.604198	0.609328	0.609536	0.607779	0.636652
14	1.181604	1.178674	1.201395	1.206492	1.202616	1.203682	1.209481	1.208551	1.254717
15	0.669557	0.700496	0.700454	0.707302	0.708066	0.704964	0.707518	0.708878	0.740184
16	0.801797	0.871372	0.872744	0.868526	0.878439	0.886118	0.881710	0.886972	0.922742

50	100	150	200	250	300	350	400	BS	err50	err100	err150	err200	err250	err300	err350	err400
83065	1.161088	1.141974	1.162360	1.161833	1.166180	1.174122	1.166695	1.205196	0.122131	0.044108	0.063222	0.042836	0.043362	0.039016	0.031074	0.038501
15015	0.017829	0.018956	0.019071	0.019481	0.019403	0.019722	0.019759	0.021159	0.006145	0.003330	0.002203	0.002088	0.001678	0.001756	0.001437	0.001400
09421	0.120168	0.121192	0.123353	0.121828	0.123104	0.123669	0.123357	0.130898	0.021477	0.010730	0.009706	0.007545	0.009070	0.007794	0.007228	0.007541
35300	0.345387	0.356715	0.366607	0.368145	0.368940	0.368887	0.366006	0.383838	0.048538	0.038451	0.027123	0.017231	0.015693	0.014898	0.014951	0.017832
18249	0.133170	0.137138	0.136818	0.138848	0.138781	0.139235	0.139859	0.146721	0.028473	0.013552	0.009583	0.009903	0.007873	0.007940	0.007487	0.006862
64834	0.406614	0.413901	0.408360	0.413946	0.413995	0.415704	0.419044	0.435205	0.070370	0.028590	0.021304	0.026845	0.021259	0.021209	0.019500	0.016161
93455	0.201683	0.203688	0.205815	0.207079	0.207359	0.208109	0.208205	0.218225	0.024770	0.016542	0.014537	0.012410	0.011146	0.010866	0.010117	0.010021
10753	1.153750	1.177862	1.176212	1.183136	1.183645	1.178571	1.185240	1.223407	0.112654	0.069657	0.045545	0.047195	0.040271	0.039762	0.044836	0.038167
01923	0.432409	0.434938	0.435512	0.435081	0.438126	0.440149	0.437661	0.458752	0.056830	0.026343	0.023814	0.023241	0.023671	0.020626	0.018603	0.021091
73911	0.567926	0.584513	0.583185	0.586636	0.587579	0.594052	0.591889	0.618183	0.044272	0.050256	0.033670	0.034998	0.031547	0.030604	0.024131	0.026294
65889	0.477068	0.474558	0.485940	0.487423	0.487252	0.483657	0.488823	0.510001	0.044112	0.032933	0.035443	0.024061	0.022578	0.022749	0.026344	0.021178
08078	0.508508	0.513841	0.520754	0.523336	0.521348	0.525601	0.526959	0.550274	0.042196	0.041766	0.036433	0.029520	0.026938	0.028925	0.024673	0.023314
69275	1.162690	1.198347	1.202131	1.200362	1.192610	1.205000	1.199249	1.247519	0.078244	0.084828	0.049171	0.045388	0.047156	0.054909	0.042518	0.048269
88411	0.584615	0.605856	0.601536	0.604198	0.609328	0.609536	0.607779	0.636652	0.048241	0.052037	0.030796	0.035116	0.032455	0.027324	0.027117	0.028873
81604	1.178674	1.201395	1.206492	1.202616	1.203682	1.209481	1.208551	1.254717	0.073114	0.076043	0.053322	0.048225	0.052101	0.051035	0.045236	0.046166
69557	0.700496	0.700454	0.707302	0.708066	0.704964	0.707518	0.708878	0.740184	0.070628	0.039688	0.039730	0.032883	0.032118	0.035221	0.032667	0.031306
01797	0.871372	0.872744	0.868526	0.878439	0.886118	0.881710	0.886972	0.922742	0.120945	0.051369	0.049998	0.054216	0.044303	0.036624	0.041032	0.035770

6) AMZN 3-month put:

	50	100	150	200	250	300	350	400	BS
0	0.008174	0.009167	0.009665	0.009955	0.010102	0.010161	0.010361	0.010413	0.011271
1	0.012341	0.014699	0.015222	0.015849	0.015989	0.015784	0.016220	0.016300	0.017532
2	0.013003	0.015589	0.016112	0.016943	0.017039	0.017188	0.017301	0.017496	0.018811
3	0.012964	0.015552	0.016071	0.016915	0.017005	0.017188	0.017269	0.017474	0.018806
4	0.015579	0.019086	0.020710	0.020636	0.021311	0.021551	0.021681	0.021598	0.023401
5	0.011080	0.013087	0.013586	0.013966	0.014148	0.014080	0.014333	0.014288	0.015544
6	0.022961	0.025976	0.026700	0.027301	0.027378	0.027557	0.027485	0.027809	0.029945
7	0.049481	0.060777	0.060470	0.061403	0.063403	0.062502	0.063470	0.063736	0.068082
8	0.043615	0.050315	0.052151	0.052901	0.052406	0.053705	0.054040	0.054174	0.057637
9	0.040101	0.044469	0.047317	0.047967	0.047615	0.048615	0.048637	0.048705	0.052182
10	0.210393	0.220559	0.222028	0.230436	0.228402	0.228130	0.230157	0.232428	0.244274
11	0.042768	0.050589	0.051578	0.052341	0.052722	0.053192	0.053720	0.053889	0.057601
12	0.030879	0.033016	0.034733	0.035127	0.036010	0.035437	0.036159	0.036385	0.039218
13	0.122810	0.136388	0.138845	0.140055	0.140024	0.139111	0.139763	0.141209	0.150418
14	0.064215	0.068815	0.071792	0.072728	0.071918	0.073381	0.073720	0.073874	0.078983
15	0.061910	0.066585	0.069335	0.070277	0.069584	0.070942	0.071246	0.071401	0.076488
16	0.127752	0.144531	0.149386	0.148511	0.151434	0.151647	0.152074	0.152478	0.161888

50	100	150	200	250	300	350	400	BS	err50	err100	err150	err200	err250	err300	err350	err400
08174	0.009167	0.009665	0.009955	0.010102	0.010161	0.010361	0.010413	0.011271	0.003097	0.002104	0.001606	0.001315	0.001169	0.001109	0.000910	0.000858
12341	0.014699	0.015222	0.015849	0.015989	0.015784	0.016220	0.016300	0.017532	0.005190	0.002833	0.002310	0.001683	0.001543	0.001748	0.001312	0.001231
13003	0.015589	0.016112	0.016943	0.017039	0.017188	0.017301	0.017496	0.018811	0.005809	0.003223	0.002699	0.001868	0.001773	0.001624	0.001510	0.001315
12964	0.015552	0.016071	0.016915	0.017005	0.017188	0.017269	0.017474	0.018806	0.005842	0.003254	0.002735	0.001891	0.001801	0.001618	0.001537	0.001332
15579	0.019086	0.020710	0.020636	0.021311	0.021551	0.021681	0.021598	0.023401	0.007822	0.004315	0.002691	0.002764	0.002090	0.001850	0.001720	0.001802
11080	0.013087	0.013586	0.013966	0.014148	0.014080	0.014333	0.014288	0.015544	0.004463	0.002457	0.001958	0.001577	0.001396	0.001464	0.001210	0.001256
22961	0.025976	0.026700	0.027301	0.027378	0.027557	0.027485	0.027809	0.029945	0.006984	0.003969	0.003245	0.002644	0.002567	0.002388	0.002460	0.002136
49481	0.060777	0.060470	0.061403	0.063403	0.062502	0.063470	0.063736	0.068082	0.018600	0.007304	0.007612	0.006679	0.004678	0.005580	0.004611	0.004345
43615	0.050315	0.052151	0.052901	0.052406	0.053705	0.054040	0.054174	0.057637	0.014023	0.007323	0.005486	0.004736	0.005231	0.003933	0.003598	0.003464
40101	0.044469	0.047317	0.047967	0.047615	0.048615	0.048637	0.048705	0.052182	0.012080	0.007713	0.004865	0.004215	0.004567	0.003567	0.003545	0.003477
10393	0.220559	0.222028	0.230436	0.228402	0.228130	0.230157	0.232428	0.244274	0.033881	0.023715	0.022245	0.013838	0.015872	0.016144	0.014117	0.011846
42768	0.050589	0.051578	0.052341	0.052722	0.053192	0.053720	0.053889	0.057601	0.014833	0.007012	0.006022	0.005259	0.004878	0.004409	0.003881	0.003711
30879	0.033016	0.034733	0.035127	0.036010	0.035437	0.036159	0.036385	0.039218	0.008339	0.006201	0.004485	0.004091	0.003208	0.003781	0.003059	0.002833
22810	0.136388	0.138845	0.140055	0.140024	0.139111	0.139763	0.141209	0.150418	0.027608	0.014030	0.011573	0.010364	0.010394	0.011308	0.010656	0.009210
64215	0.068815	0.071792	0.072728	0.071918	0.073381	0.073720	0.073874	0.078983	0.014768	0.010169	0.007191	0.006255	0.007065	0.005602	0.005263	0.005109
61910	0.066585	0.069335	0.070277	0.069584	0.070942	0.071246	0.071401	0.076488	0.014578	0.009902	0.007153	0.006211	0.006904	0.005546	0.005242	0.005087
27752	0.144531	0.149386	0.148511	0.151434	0.151647	0.152074	0.152478	0.161888	0.034136	0.017357	0.012502	0.013377	0.010454	0.010241	0.009814	0.009410

4. Implement an additive binomial tree to calculate the **American option**, both Call and Put. Repeat the steps in part 2) and calculate the respective option prices as if they are American.

Code of pricing American option:

```
import math

class Option(object):

    def __init__(self, s0, sigma, r, t, n, k, type=''):
        self.s0=s0
        self.sigma=sigma
        self.r=r
        self.t=t
        self.n=n
        self.k=k
        self.type=type
```

```
def Am_price(self):
    time=float(self.t)/self.n
    u=math.exp(self.sigma*time**0.5)
    d=math.exp(-self.sigma*time**0.5)
    p=(math.exp(self.r*time)-d)/(u-d)
    pay=[]
    for i in range(self.n+1):
        if self.type=='Call':
            pay.append(max(((self.s0*u**(self.n-2*i))-self.k),0))
        elif self.type=='Put':
            pay.append(max((self.k-(self.s0*u**(self.n-2*i))),0))

    for j in range(self.n-1,-1,-1):
        ppay=[]
        qpay=[]
        for m in range(j+1):
            if self.type=='Call':
                qpay.append((max(((self.s0*u**(j-2*m))-self.k),0)))
            elif self.type=='Put':
                qpay.append((max((self.k-(self.s0*u**(j-2*m))),0)))
            ppay.append((pay[m]*p+pay[m+1]*(1-p))*math.exp(-self.r*time))

            pay[m]=max(qpay[m],ppay[m])
    return pay[0]
```

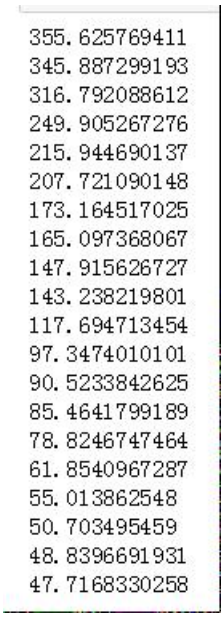
Calculate the option prices:

1) For AMZN one-month call option:

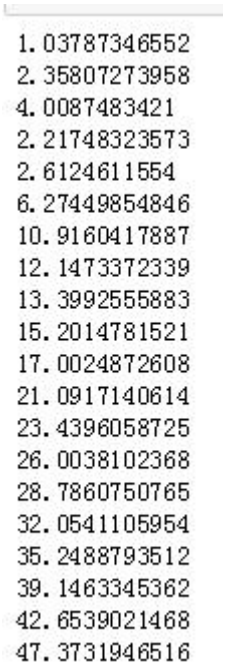
From binomial tree:

	American option
0	356.690574
1	346.921798
2	317.908601
3	251.039317
4	217.150896
5	208.947181
6	174.322246
7	166.284505
8	149.083488
9	144.306283
10	118.790128
11	98.323520
12	91.506360
13	86.409500
14	79.719092
15	62.682207
16	55.791459

From BSM:



2) For put option:
By binomial tree:



From BSM:

```
1.07254372179
2.42456167036
4.1125178604
2.31214925984
2.69270574337
6.42768043473
11.1430898336
12.4453498349
13.6619911086
15.5900167111
17.3239664474
21.4823943218
23.9360436442
26.4742992235
29.3626832685
32.6124842897
35.9291568759
39.8010125776
43.4557626679
48.1333827076
```

5. Create a table which contains the following columns: Bid and Ask values, Black Scholes price, European and American prices calculated using the binomial tree.

1) AMZN 1-month call:

	bid_price	ask_price	price_BS	price_American option	price_Euro option
0	373.70	378.70	355.625769	356.690574	356.691551
1	363.85	368.85	345.887299	346.921798	346.912121
2	334.45	339.45	316.792089	317.908601	317.880294
3	266.90	271.90	249.905267	251.039317	250.982029
4	226.20	231.20	215.944690	217.150896	217.148296
5	218.60	223.55	207.721090	208.947181	208.931563
6	182.80	187.60	173.164517	174.322246	174.390009
7	174.90	179.90	165.097368	166.284505	166.276261
8	157.90	162.90	147.915627	149.083488	149.104000
9	159.85	164.20	143.238220	144.306283	144.355535
10	128.85	133.65	117.694713	118.790128	118.766515
11	109.90	113.85	97.347401	98.323520	98.260320
12	103.65	107.00	90.523384	91.506360	91.545118
13	97.95	102.30	85.464180	86.409500	86.413558
14	90.95	95.50	78.824675	79.719092	79.727972
15	74.65	78.20	61.854097	62.682207	62.722776
16	67.30	71.25	55.013863	55.791459	55.701729
17	62.65	66.80	50.702405	51.380038	51.484648

2) AMZN 1-month put:

	price_BS	bid_price	ask_price	Euro	American
0	1.072544	2.21	3.85	1.037873	1.038634
1	2.424562	5.1	6.75	2.358073	2.359936
2	4.112518	7.45	9.80	4.008748	4.012537
3	2.312149	5.25	6.20	2.217483	2.220065
4	2.692706	5.4	6.90	2.612461	2.615212
5	6.427680	10.2	13.55	6.274499	6.280401
6	11.143090	16	20.30	10.916042	10.927533
7	12.445350	17.5	22.00	12.147337	12.161770
8	13.661991	19.2	23.50	13.399256	13.414623
9	15.590017	21.15	25.95	15.201478	15.220930
10	17.323966	23.3	28.05	17.002487	17.023279
11	21.482394	28.15	32.90	21.091714	21.119449
12	23.936044	30.95	35.70	23.439606	23.471000
13	26.474299	33.7	38.50	26.003810	26.040555
14	29.362683	36.9	41.65	28.786075	28.827553
15	32.612484	40.8	45.15	32.054111	32.102533
16	35.929157	43.85	48.60	35.248879	35.303727
17	39.801013	47.85	52.65	39.146335	39.209658
18	43.455763	51.55	56.30	42.653902	42.726180

3) Then write comments about the observed differences between the various option valuations and how they compare with the actual bid/ask values.
For call:
The option prices of American and European are almost equivalent, while the prices from BSM are lower to some extent. Comparing to the real bid and ask prices, the results calculating from formula and binomial tree are relatively high.
For put:
The real prices are higher than those calculated from BSM and the binomial tree. And the American option value seems to be slightly higher than the European option value.

6. 1) Using the binomial tree for American Calls and Puts, calculate the implied volatility corresponding to the data you have downloaded in part (2).
Use bisection method:
For American call:

```
def f(x):
    if __name__ == '__main__':
        o=Option(s0=1390, sigma=x, r=0.0149, t=0.083, n=200, k=K, type='Call')
        return o.Am_price()-mktprice
```

```
def bisect2(a,b):
    while b - a >= 0.000001:
        mid= (a + b) / 2.
        if f(a)*f(mid) <=0:
            b = mid
        else:
            a = mid

    return (mid)
```

```

```

```
for q,w in zip(avrg_11a, sp_1):
    mktprice=q
    K=w
    print(bisect2(0.000001,2))
```

Results:

```
1.1377996941008566
1.1128477717623708
1.0409121540436748
0.8864284361195564
0.651759821764469
0.65098353125906
0.5956647265210152
0.5900094406523705
0.566497565811634
0.6725747069869042
0.5433766972451209
0.5327871029305457
0.5263478971657752
0.5287301744170189
0.5168435836806297
0.5022695400643348
0.49939898180723186
0.4965970880665779
0.49647501781511305
0.4950750246186256
```

For American put:

```
def f(x):
    if __name__ == '__main__':
        o=Option(s0=1390, sigma=x, r=0.0149, t=0.083, n=200, k=K, type='Put')
        return o.Am_price()-mktprice
```

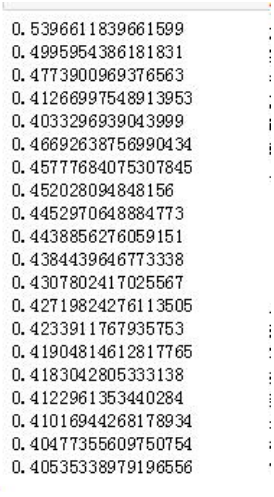
```
def bisect2(a,b):
    while b - a >= 0.000001:
        mid= (a + b) / 2.
        if f(a)*f(mid) <=0:
            b = mid
        else:
            a = mid

    return (mid)
```

```

```

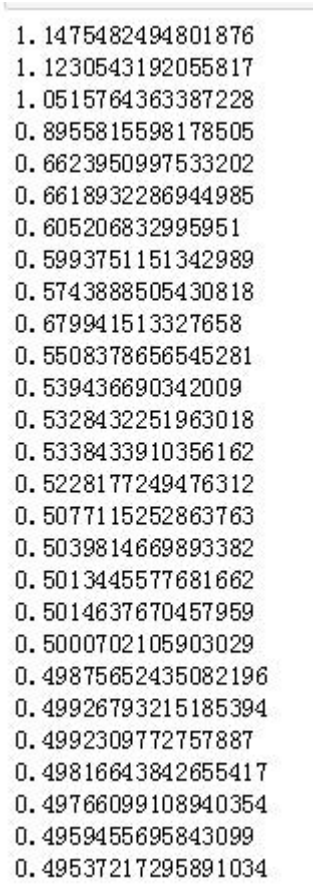
```
for q,w in zip(avrg_1a, sp_4):
    mktprice=q
    K=w
    print(bisect2(0.000001,2))
```



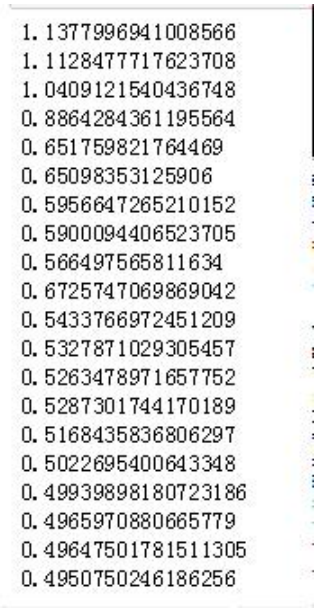
Compare these values of the implied volatility with the volatilities from Homework 1, Problem 1c (or with the ones obtained in part 2 of this problem). Write detailed observations.

For AMZN call:

The IV from H1:



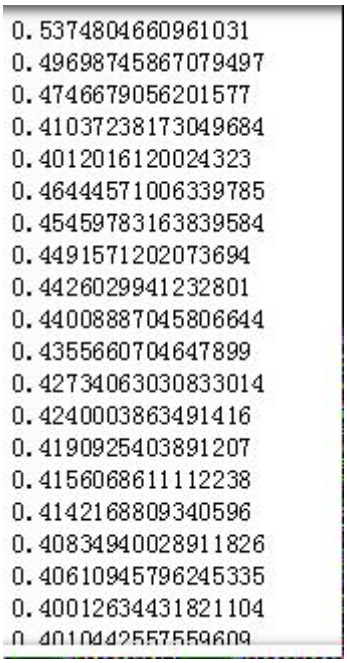
IV from the binomial tree:

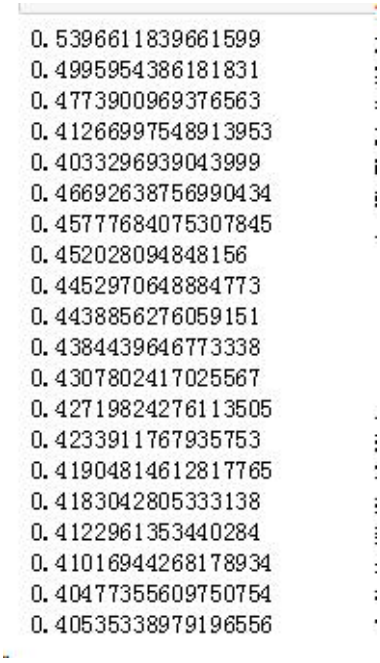


For AMZN put:

From BSM:

From binomial tree:





From the results showing above, for call options, it seems that the volatilities from BSM is higher than that from the binomial tree, while for the put options, the volatilities from the binomial tree is slightly higher than those from BSM.

7. Implement a trinomial tree to price an American Put option.

```
class StockOption(object):
    def __init__(self, S0, K, r, T, N, params):
        self.S0 = S0
        self.K = K
        self.r = r
        self.T = T
        self.N = max(1, N)
        self.STs = None

        self.pu = params.get("pu", 0)
        self.pd = params.get("pd", 0)
        self.div = params.get("div", 0)
        self.sigma = params.get("sigma", 0)
        self.is_call = params.get("is_call", True)
        self.is_european = params.get("is_eu", True)

        self.dt = T/float(N)
        self.df = math.exp(
            -(r-self.div) * self.dt)
```

```
class TrinomialTreeOption(BinomialTreeOption):
    def _setup_parameters_(self):
        self.u = math.exp(self.sigma*math.sqrt(2.*self.dt))
        self.d = 1/self.u
        self.m = 1
        self.qu = ((math.exp((self.r-self.div) *
                             self.dt/2.) -
                    math.exp(-self.sigma *
                             math.sqrt(self.dt/2.))) /
                   (math.exp(self.sigma *
                             math.sqrt(self.dt/2.)) -
                    math.exp(-self.sigma *
                             math.sqrt(self.dt/2.))))**2
        self.qd = ((math.exp(self.sigma *
                             math.sqrt(self.dt/2.)) -
                    math.exp((self.r-self.div) *
                             self.dt/2.)) /
                   (math.exp(self.sigma *
                             math.sqrt(self.dt/2.)) -
                    math.exp(-self.sigma *
                             math.sqrt(self.dt/2.))))**2.

        self.qm = 1 - self.qu - self.qd
```

```
def _initialize_stock_price_tree_(self):
    self.STs = [np.array([self.S0])]
    for i in range(self.N):
        prev_nodes = self.STs[-1]
        self.ST = np.concatenate(
            (prev_nodes*self.u, [prev_nodes[-1]*self.m,
                                 prev_nodes[-1]*self.d]))
        self.STs.append(self.ST)
def _traverse_tree_(self, payoffs):
    for i in reversed(range(self.N)):
        payoffs = (payoffs[:-2] * self.qu +
                   payoffs[1:-1] * self.qm +
                   payoffs[2:] * self.qd) * self.df
        if not self.is_european:
            payoffs = self.__check_early_exercise__(payoffs,
                                                    i)
    return payoffs
```

Use this tree with the data in this problem and compare with the results obtained using the binomial tree.
Choose the AMZN option:

	contract_name	last_trade_date	strike_price	last_price	bid_price	ask_price
0	AMZN180323P01060000	2018-02-05 2:18PM EST	1060.0	1.99	2.21	3.85

Price from binomial tree:

200
1.037873

; from trinomial tree: American put: 1.03994841527.

The price from trinomial tree, which is closer to the real price, is a little higher than it from binomial.

Part 2

1. Construct a binomial tree to calculate the price of an European Up-and-Out call option. Use as many steps in your tree as you think are necessary.

```
def binom_tree(N,T,S0,sigma,r,K,call=True,array_out=False):
    dt=T/N
    u=np.exp((sigma*np.sqrt(dt)))
    d=1/u
    p=(np.exp(r*dt)-d)/(u-d)

    price_tree=np.zeros([N+1,N+1])

    for i in range(N+1):
        for j in range(i+1):
            price_tree[j,i]=S0*(d**j)*(u**(i-j))

    option=np.zeros([N+1,N+1])
    if call:

        option[:,N]=np.maximum(np.zeros(N+1),price_tree[:,N]-K)

    else:
        option[:,N]=np.maximum(np.zeros(N+1),K-price_tree[:,N])
```

```
    for i in np.arange(N-1,-1,-1):
        for j in np.arange(0,i+1):
            if S0*(d**j)*(u**(i-j))<H:
                option[j,i]=np.exp(-r*dt)*(p*option[j,i+1]+(1-p)*option[j+1,i+1])
            else:
                option[j,i]=0.

    if array_out:
        return [option[0,0],price_tree,option]
    else:
        return option[0,0]
```

```
binom_tree(200,0.3,10,0.2,0.01,10,call=True,array_out=False)
```

```
0.062620982567432676
```

2. To price the European Up-and-Out Call option, explicit formulae can be found. Implement the formula (5.2) from [2] and compare your results with part (1). Use the same parameters as before. What can you observe?

```
def C(S0,K):
    d1=(np.log(S0/K)+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2=d1-sigma*np.sqrt(T)
    C=S0*norm.cdf(d1)-K*norm.cdf(d2)
    return C
T=0.3

K=10
sigma=0.2
r=0.01
S=10
H=11
v=r-(sigma**2)/2.
dbs=(np.log(S/H)+v*T)/(sigma*sqrt(T))
dbs2=(np.log(H/S)+v*T)/(sigma*sqrt(T))
UO=C(S,K)-C(S,H)-(H-K)*np.exp(-r*T)*norm.cdf(dbs)-(H/S)**(2*v/(sigma**2))*(C((H**2)/S,K)-C((H**2)/S,H)-(H-K)*np.exp(-r*T)*norm.cdf(dbs2))
```

Result:UO=0.074088676610012158.

The result getting from the formula is higher than that from Q1.

3. Price an European Up-and-In call option, using the same parameters as before.
1)

```
def P(S0,K):
    d1=(np.log(S0/K)+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2=d1-sigma*np.sqrt(T)
    P=K*norm.cdf(-d2)-S0*norm.cdf(-d1)
    return P
UI=(H/S)**(2*v/(sigma**2))*(P(H**2/S,K)-P((H**2)/S,H)+(H-K)*np.exp(-r*T)*norm.cdf(-dbs2))+C(S,H)+(H-K)*np.exp(-r*T)*norm.cdf(dbs)
```

UI=0.36570621724038443

2)

```
#In-Out parity
UI2=C(S,K)-UO
```

UI=0.36254867825684389

4. Construct a binomial tree to calculate the price of an American Up-and-In call option.

```
class Option(object):

    def __init__(self,s0,sigma,r,t,n,k,type=''):
        self.s0=s0
        self.sigma=sigma
        self.r=r
        self.t=t
        self.n=n
        self.k=k
        self.type=type
```

```
def Am_price(self):
    time=float(self.t)/self.n
    u=math.exp(self.sigma*time**0.5)
    d=math.exp(-self.sigma*time**0.5)
    p=(math.exp(self.r*time)-d)/(u-d)
    pay=[]
    for i in range(self.n+1):
        if self.type=='Call':
            if self.s0*u**(self.n-2*i)>11:
                pay.append(max(((self.s0*u**(self.n-2*i))-self.k),0))
            else:
                pay.append(0)
        elif self.type=='Put':
            pay.append(max((self.k-(self.s0*u**(self.n-2*i))),0))

    for j in range(self.n-1,-1,-1):
        ppay=[]
        qpay=[]
        for m in range(j+1):
            if self.type=='Call':
                qpay.append((max(((self.s0*u**(j-2*m))-self.k),0)))
            elif self.type=='Put':
                qpay.append((max((self.k-(self.s0*u**(j-2*m))),0)))
            ppay.append((pay[m]*p+pay[m+1]*(1-p))*math.exp(-self.r*time))

            pay[m]=max(qpay[m],ppay[m])
    return pay[0]
```

```
if __name__ == '__main__':

    o=Option(s0=10,sigma=0.2,r=0.0149,t=0.3,n=200,k=10,type='Call')
    print(o.Am_price())
```

```
0.4578694941186229
```

Part3

1.

```
i=[0,1,2,3,4,5,6,7,8,9,10,11]
i
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
for a in i:
    r=0.05*(1+0.01*a)
    print(r)
```

```
for a in i:
    sigma=0.2*(1+0.005*a)
    print(sigma)
```

```
for i,j in zip(rli,sigli):
    v=i-0.5*(j**2)
    print(v)
```

△t:

```
for i,j in zip(sigli,vli):
    t=1/(2*j**2)*((-i**2)+sqrt(i**4+4*(j**2)*dx**2))
    print(t)

    f.write(str(t)+'\n')

f.close()
```

```
0.06241235570175058
0.061792889198338544
0.061182613722538165
0.060581348070892445
0.059988915487293314
0.059405143532683134
0.05882986395922304
0.05826291258865967
0.05770412919484416
0.05715335739011093
0.05661044451550702
0.05607524153460086
```

P:

```
filename='part3_p.csv'
f=open(filename,'w')

headers='p\n'
f.write(headers)

for i,j in zip(tli,vli):
    p=0.5+j*i/(2.*dx)
    print(p)

    f.write(str(p)+'\n')

f.close()
```

```
0.5187237067105251
0.5187229364626506
0.5187206561468223
0.5187169103932425
0.5187117425187965
0.5187051945698536
0.5186973073635203
0.5186881205273756
0.5186776725377872
0.5186660007568233
0.5186531414678596
0.5186391299098937
```

Table:

	i	r	sigma	v	t	p
0	0	0.0500	0.200	0.030000	0.062412	0.518724
1	1	0.0505	0.201	0.030300	0.061793	0.518723
2	2	0.0510	0.202	0.030598	0.061183	0.518721
3	3	0.0515	0.203	0.030895	0.060581	0.518717
4	4	0.0520	0.204	0.031192	0.059989	0.518712
5	5	0.0525	0.205	0.031488	0.059405	0.518705
6	6	0.0530	0.206	0.031782	0.058830	0.518697
7	7	0.0535	0.207	0.032076	0.058263	0.518688
8	8	0.0540	0.208	0.032368	0.057704	0.518678
9	9	0.0545	0.209	0.032660	0.057153	0.518666
10	10	0.0550	0.210	0.032950	0.056610	0.518653
11	11	0.0555	0.211	0.033240	0.056075	0.518639

2. Using bisection method:

```
import math

def t(x):
    t=0


    for i,j in zip(sigli,vli):
        sig=i
        v=j

        t=(1/(2*j**2))*(-i**2+sqrt(i**4+4*j**2*x**2))+t

    return t-1
```

```
def bisect(a,b):
    while b - a >= 0.000001:
        mid= (a + b) / 2.
        if t(a)*t(mid) <=0 :
            b = mid
        else:
            a = mid

    return (mid)
```

```
bisect(0,1)
```

```
0.059355735778808594
```

BONUS:
Price an European Call Option using the Beliaeva and Nawalkha method.

```
def Int_Function_1(phi, kappa, theta, sigma, rho, v0, r, T, s0, K, type):
    temp = (np.exp(-1*1j*phi*np.log(K))*Int_Function_2(phi, kappa, theta, sigma, rho, v0, r, T, s0, K, type))
    return temp

def Int_Function_2(phi, kappa, theta, sigma, rho, v0, r, T, s0, K, type):
    if type==1:
        u = 0.5
        b = kappa - rho*sigma
    else:
        u = -0.5
        b = kappa
    a = kappa*theta
    x = np.log(s0)
    d = np.sqrt((rho*sigma*phi*1j-b)**2 - sigma**2*(2*u*phi*1j-phi**2))
    g = (b-rho*sigma*phi*1j+d)/(b-rho*sigma*phi*1j-d)
    D = r*phi*1j*T + (a/sigma**2)*((b-rho*sigma*phi*1j+d)*T - 2*np.log((1-g*np.exp(d*T))/(1-g)))
    E = ((b-rho*sigma*phi*1j+d)/sigma**2)*(1-np.exp(d*T))/(1-g*np.exp(d*T))

    return np.exp(D + E*v0 + 1j*phi*x)
```

```
def Heston_Call_Value_Int(kappa, theta, sigma, rho, v0, r, T, s0, K):
    a = s0*Heston_P_Value(kappa, theta, sigma, rho, v0, r, T, s0, K, 1)
    b = K*np.exp(-r*T)*Heston_P_Value(kappa, theta, sigma, rho, v0, r, T, s0, K, 2)
    #print (a,b)
    return a-b

def Heston_P_Value(kappa, theta, sigma, rho, v0, r, T, s0, K, type):
    ifun = lambda phi: Int_Function_1(phi, kappa, theta, sigma, rho, v0, r, T, s0, K, type)
    return 0.5 + (1/np.pi)*spi.quad(ifun, 0, 100)[0]
```