ILLINOIS INSTITUTE OF TECHNOLOGY

MATH548

MATHEMATICAL FINANCE I

Project # 1 Report

Team 5:

Xie Hua Li Da Chen Jielin

Liu Zhaojian Zhao Xiyu

Instructor:

Prof. IGOR CIALENCO

November 11, 2014

Contents

1 Design of Model

First we denote the following notations.

Table 1: Notation

S_0	current stock price
S_t	stock price at time t
K	strike price determined in the options
r	interest rate
T	the maturity of the corresponding option
M	time steps in binomial model
u	up factor in binomial Model
q	R.N. probability in binomial model for price going up
c	theoretical price of the European call/put option
$\bar{c}$	the market price of the European call/put option
σ	the volatility of the underlying stock
σ_m	the implied volatility
$opType$	option type, call or put

1.1 Build binomial model

We used discrete binomial option pricing model for this project.
About how to build the model, please refer to memo and python file in Appendix 1.

1.2 Methods to get u

After building the model, the key problem becomes how can we get this u factor. We used 4 different methods to do this.

1.2.1 up/down factor method

Step1: First find all ATM option data--($S_0, K, r, T, M, opType, c$) based on current stock price.

Step2: define function $F(u) =$ crrModel.payoff$(S_0, K, r, u, T, M, opType) - c$

Step3: use a non-linear solver to solve $F(u) = 0$ For following volatility related methods, we calculate u from volatilities using $u = e^{\sigma\sqrt{T/M}}$.[3]

1.2.2 implied volatilities

Suppose the Black-Scholes Model for option pricing is: $c = f(\sigma, .)$, then implied volatility is given by $\sigma_m = f^{-1}(\bar{c}, .)$. In practice, we find the root of function: $f(\sigma_m, .) - \bar{c} = 0$. We used python package *mibian*for Black-Scholes Model. Please check Part3 from Appendix 2 for this.

1.2.3 volatility based on historical data

This method is to estimate the volatility from historical data. In order to estimate the volatility of the stock price, the stock price at some fixed intervals of time needs to be observed. In this project, $n+1$ is used to denote the number of observations, S_i is used to denote the stock price at the end of i^{th} interval, where i takes values in 0,1,2,…,n., and t is used to denote the length of time interval in years.

Step1: computation for daily volatility The goal of this part is to computing the daily volatility by using the formula $u_i = \log(\frac{S_i}{S_{i-1}})$ The data which is used here is from June 26, 2014 to November 5, 2014. The length of this period is the same as the length of the period from today to the expiration day of the option. So the annual volatility can be estimated in the future by using the historical data from June 26 till now.

Step2: estimating the standard deviation of u_i The estimated s can be computed by using the formula $s = \sqrt{\frac{\sum_{i=1}^{n}(u_i - \bar{u})^2}{n-1}}$

Step3: computation for annual volatility The estimated s, which is the standard deviation of the daily return, has to be converted into the volatility per annum. Assume there are 252 trading days per year, the volatility per annual is $s * \sqrt{252}$.[3]

1.2.4 volatility using GARCH

GARCH is the abbreviation of Generalized AutoRegressive Conditional Heteroskedasticity. The distinctive feature of the model is that it acknowledge that volatilities and correlations are not constants. The model attempts to keep track of the variations in the volatility or correlation

through time. It is often used to analyse financial data. GARCH (1,1) is the most popular of the GARCH models.[2]

We use σ_n denote the volatility of a market on day n, which is estimated at the end of day n-1. The square of the volatility, σ_n^2, on day n is the variance rate. We assume the value of the market variable at the end of the day i is S_i. Also, we use variable u_i denote the continuously compound return during day i.

$$u_i = \frac{S_i - S_{i-1}}{S_{i-1}} \tag{1}$$

In the GARCH (1,1) model, σ_n^2 is calculated from a long-run average variance rate, V_L, as well as from σ_{n-1} and u_{n-1}. The equation for the model is

$$\sigma_n^2 = rV_L + \alpha u_{n-1}^2 + \beta\sigma_{n-1}^2 \tag{2}$$

Then an unbiased estimate of the variance rate per day, σ_n, using the most recent m observations on the market is $\sigma_n^2 = \frac{1}{m-1}\sum_m^{i=1}(u_i - \bar{u})^2$

The term (p, q) of the GARAH (p, q) means that is based on the most recent p observations of and the most recent q estimate of the variance rate. The (1, 1) term is the most common used one in the GARCH models.[4]

Let $\omega = rV_L$, we can rewrite the model as

$$\sigma_n^2 = \omega + \alpha u_{n-1}^2 + \beta\sigma_{n-1}^2 \tag{3}$$

This form is frequently used for estimating the parameters.

Step 1: Choose the sample of our model Three months volatility of the stock price is computed by using the historical data. Three months volatility makes sense here because the period from now to the expiration day of the option is about three months and the forecast value will be more accurate. The close price of GE from November 5, 2010 to November 5, 2014 was chosen as the samples.

Step 2: Compute the historical volatility of every three month of the sample space The volatility of the stock price of every three months can be computed by the samples.

Step 3: Estimate the parameter of GARCH(1,1) The coefficient of GARCH(1,1) model can be estimated by EVIEWS.

Step 4: Forecast the volatility of next three months Forecast the volatility of the stock price of GE in the next three months by using GARCH (1,1) model. Since the outcome of the forecast is the volatility of three months, and the u and d are computed by the annual volatility, so the three months volatility need to convert into annual volatility.

2 Calibration

2.1 Statistics

We use two statistics for calibrating the model, errMean and errStd. They are defined in the following way:

$errMean = \mathbb{E}[\frac{|price_{pred} - price_{market}|}{price_{market}}]$ $\quad$ $errStd = \sqrt{\text{var}[\frac{|price_{pred} - price_{market}|}{price_{market}}]}$

Note this is the relative error of predicted option price.

The first column of the table indicates the method, note there are nine different u to choose for up/down factor methods.

imVola means implied volatility is used(calculated by ourself). imVolaGiven used given implied volatility on yahoo.finance.

hisVola is volatility calculated by historical data. garchVola is volatility get from GRACH method.

From the results we can pick out the best u for up/down factor method, which is $u = 1.007315$ for call option and $u = 1.009611$ for put option. We can also compare these four methods based on this results. Implied volatility seems to perform best among the four, with 5% error for call option and 9% error for put option.

Table 2: error statistics for call options

u	errMean	errStd
1.0084473613	0.232986975094	0.31327168106
1.00961093521	0.352464003312	0.455026566863
1.00731514523	0.188305741618	0.313810653125
1.00755989205	0.189470517888	0.30527422953
1.00683422169	0.201975445117	0.331794778452
1.00804336626	0.207643935028	0.299982554264
1.00709275526	0.192770213663	0.321432952658
1.00819095924	0.214498332689	0.303342310173
1.00725089647	0.189207566397	0.31581231149
imVola	0.0505378834343	0.0843678348481
imVolaGiven	0.0448397508013	0.0723158330639
hisVola	0.193911239052	0.301018401547
garchVola	0.329698783236	0.390333582641

Table 3: error statistics for put options

u	errMean	errStd
1.0084473613	0.60674063871	0.400081408063
1.00961093521	0.564245956315	0.404760883793
1.00731514523	0.65367226676	0.383312211102
1.00755989205	0.644074662439	0.386925369376
1.00683422169	0.671691659253	0.37749829154
1.00804336626	0.624208642415	0.394064409419
1.00709275526	0.662189928282	0.38042940678
1.00819095924	0.617735419942	0.396344616465
1.00725089647	0.656141017274	0.38243926609
imVola	0.0965905974612	0.17889593364
imVolaGiven	0.189867073335	0.274381940521
hisVola	0.638018294857	0.388996733062
garchVola	0.728728309805	0.362709765342

2.2 Plots

We did the plots for all four methods. Both predicted price and market price are plotted against strike price. We did this for all options listed on yahoo.finance.

Figure 1 and 2 are results using up/down factor method.

Figure 3 and 4 are results using historical volatility method.

Figure 5 and 6 are results using GARCH volatility method.

Figure 7 and 8 are results using implied volatility method.

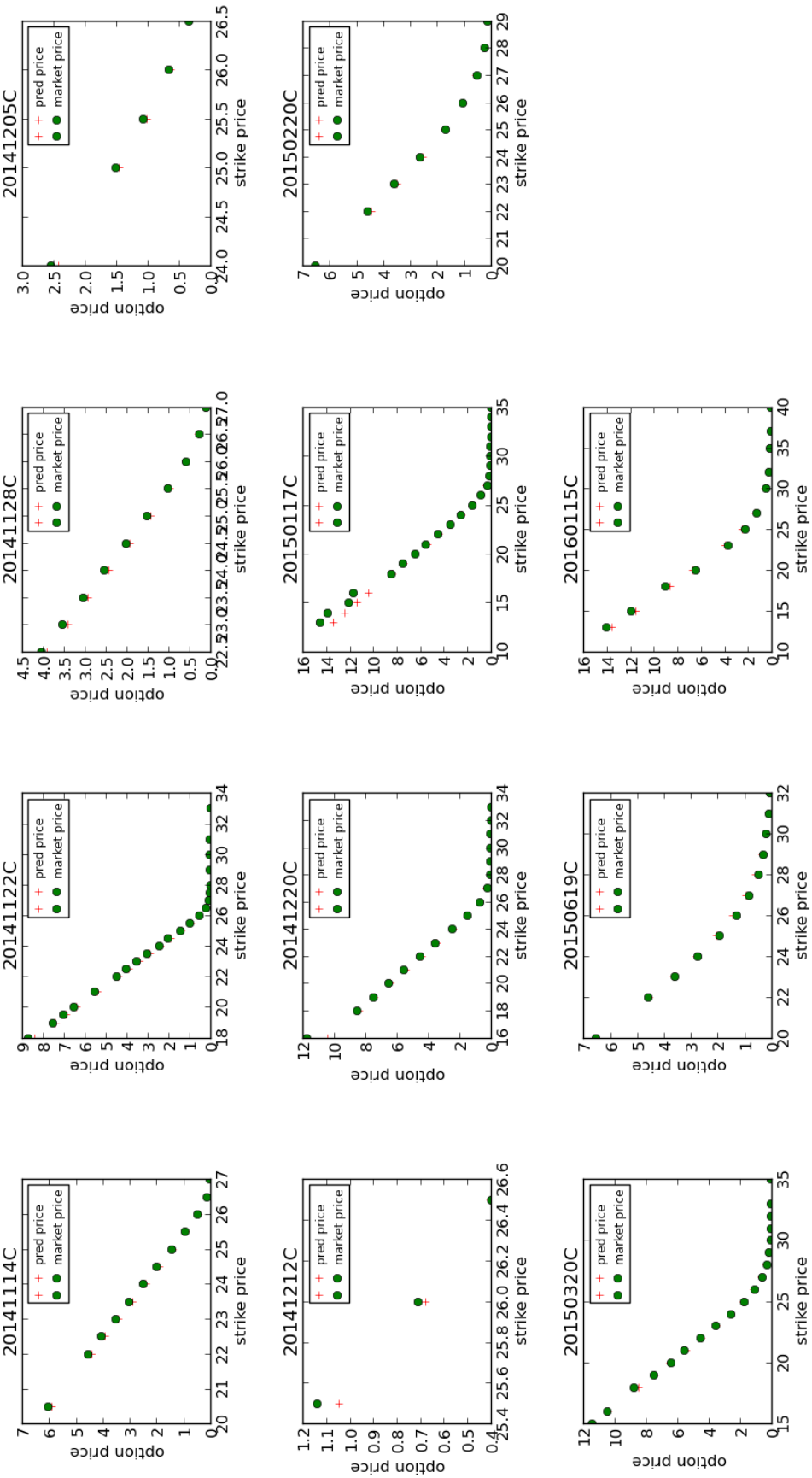

Figure 1: call option price, $u = 1.007315$

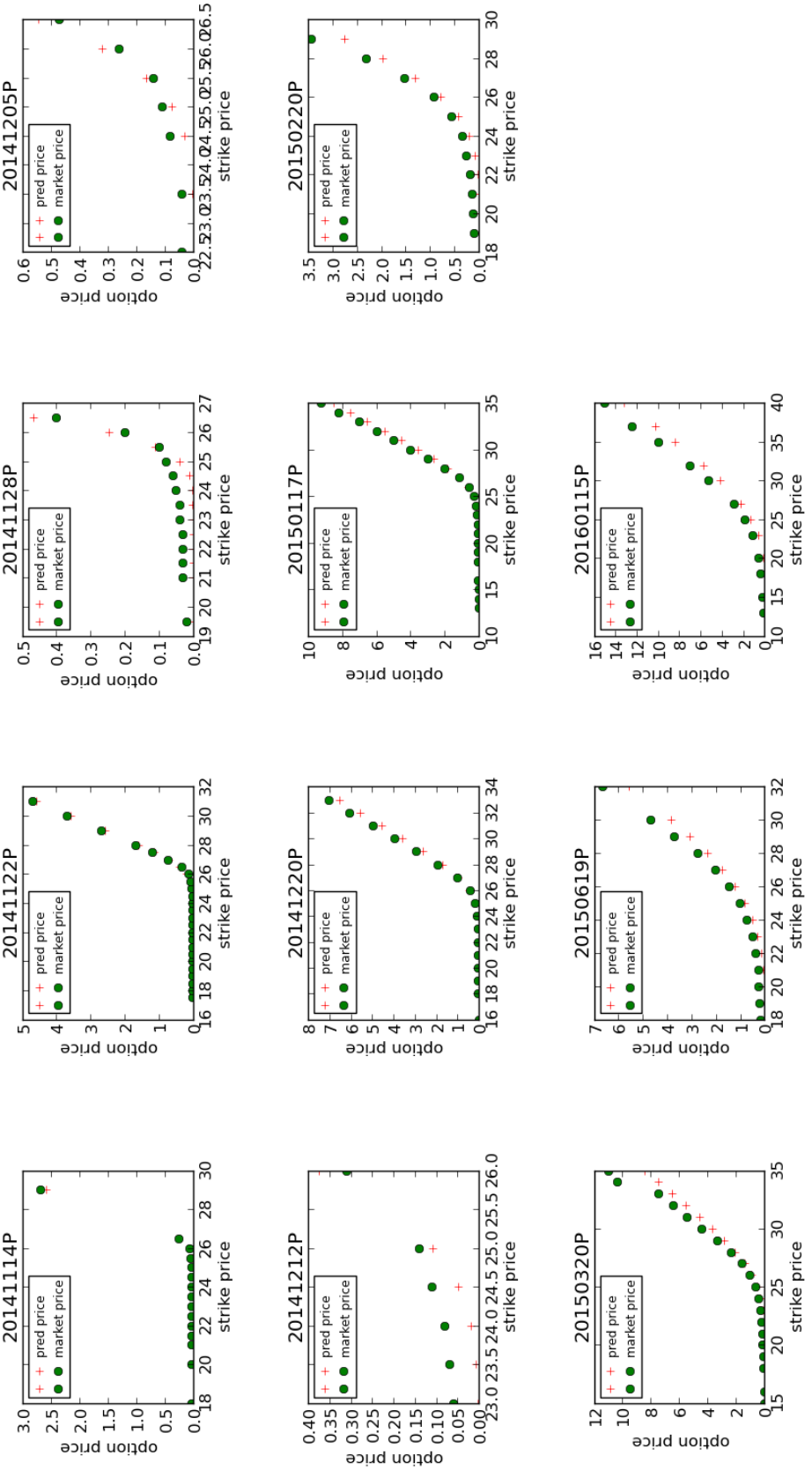

Figure 2: put option price, $u = 1.009611$

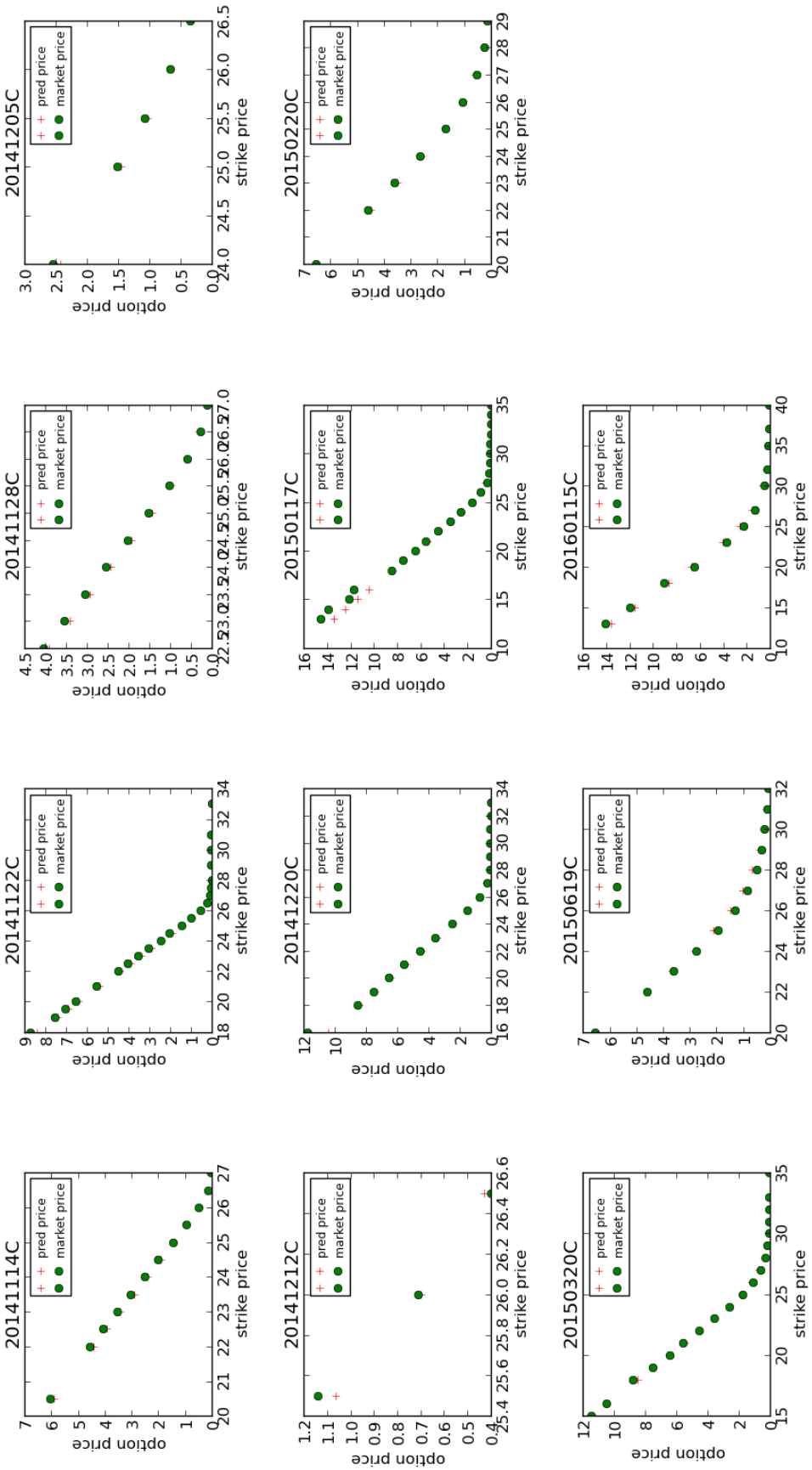

Figure 3: call option price using historical volatility

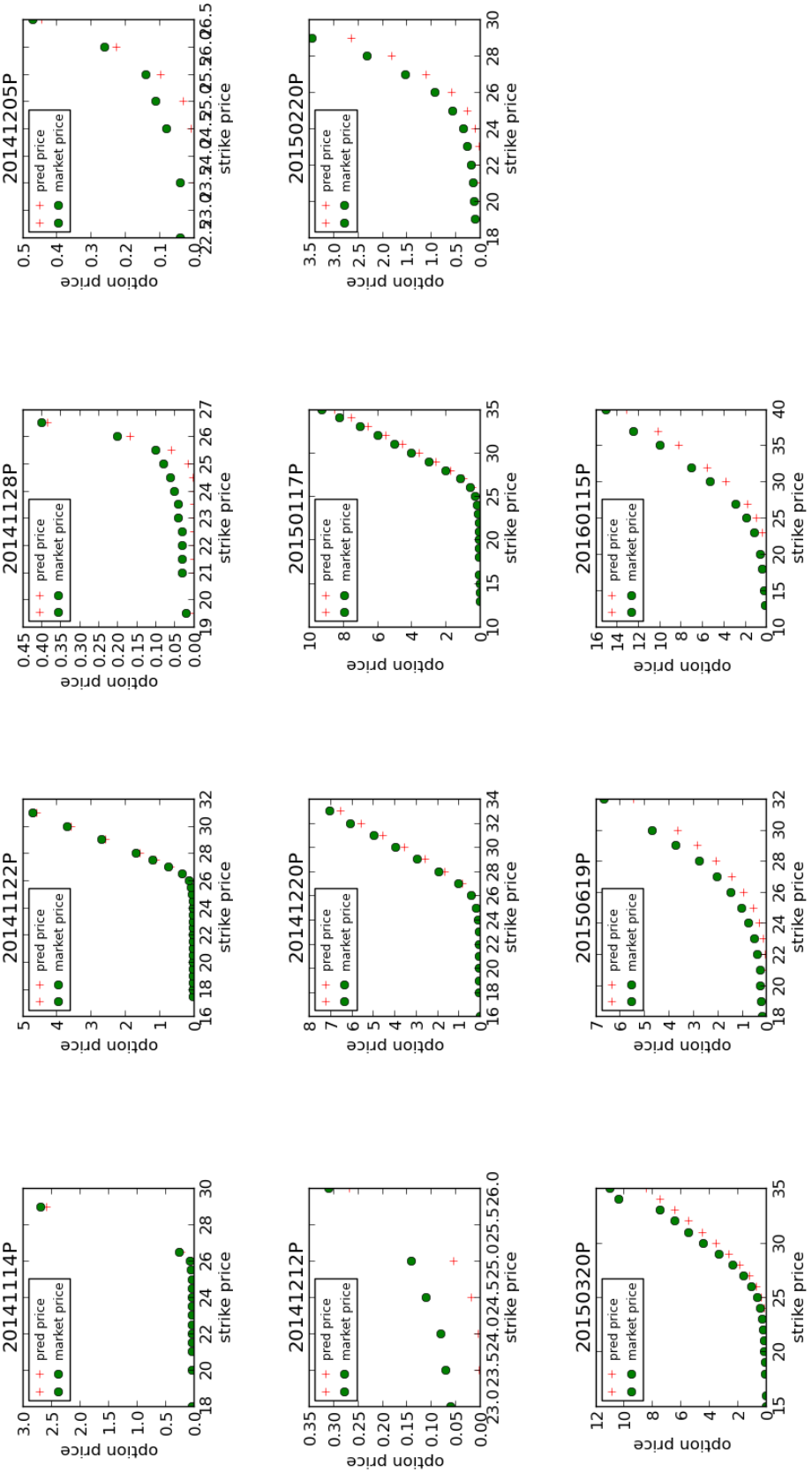

Figure 4: put option price using historical volatility

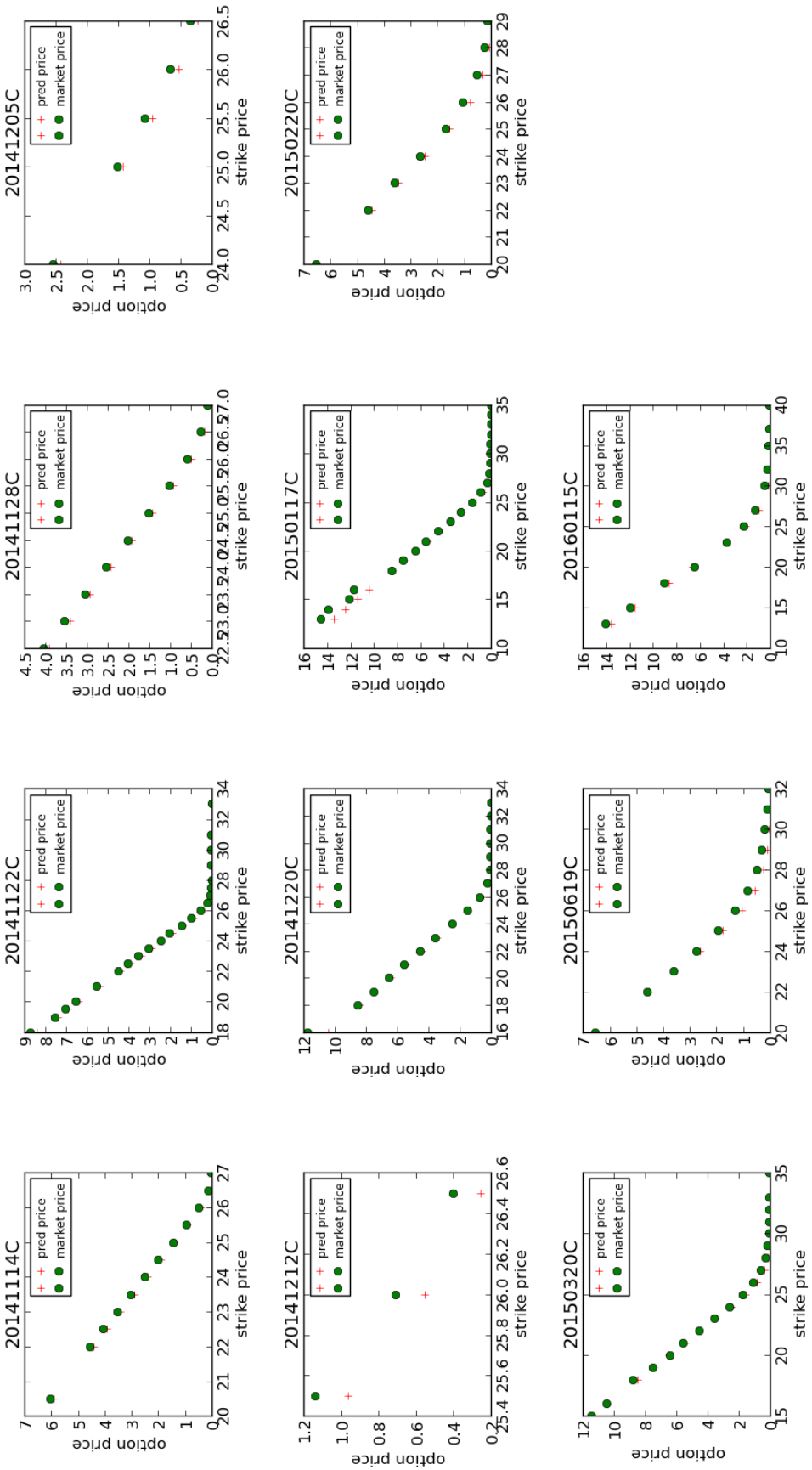

Figure 5: call option price using GARCH volatility

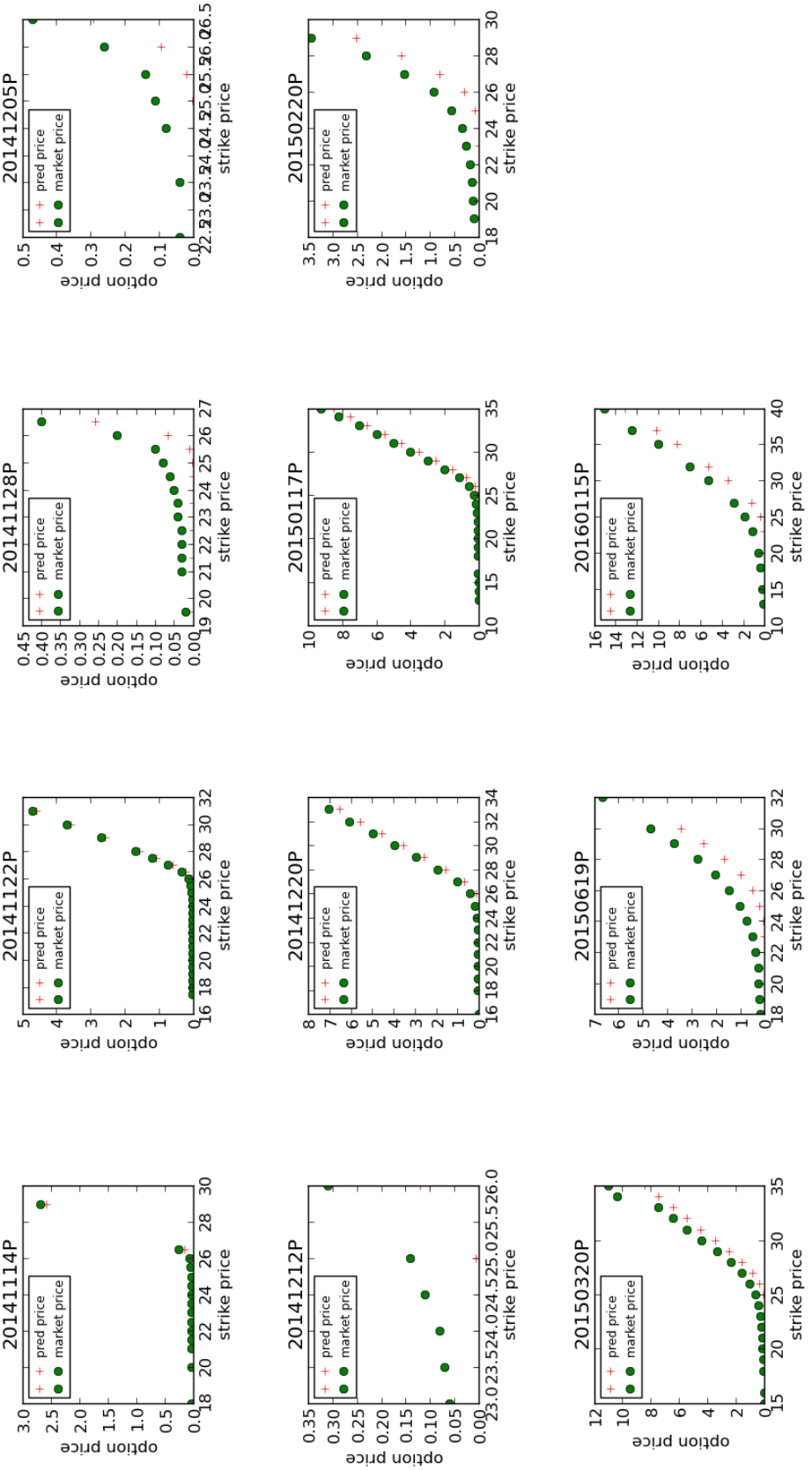

Figure 6: put option price using GARCH volatility

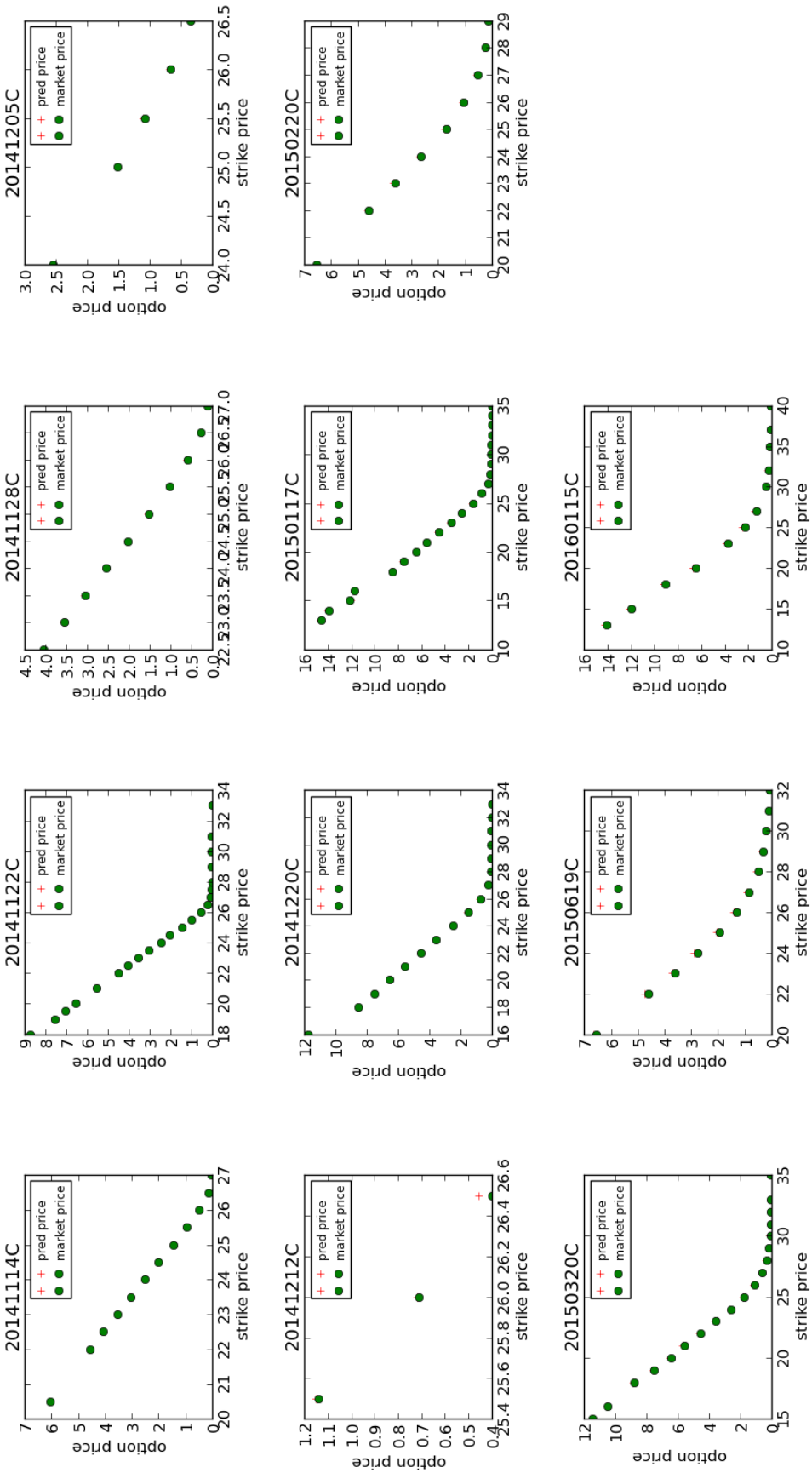

Figure 7: call option price using implied volatility

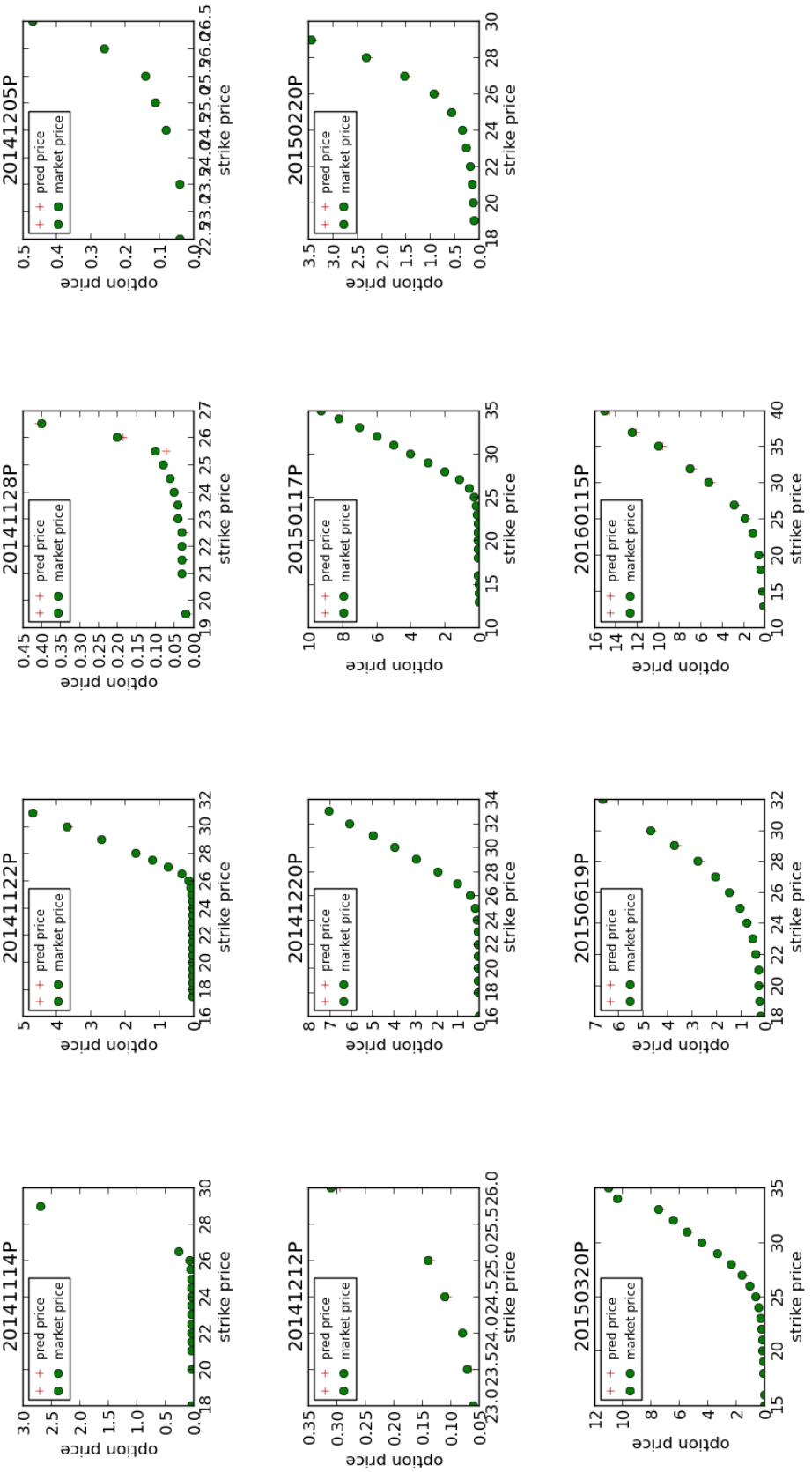

Figure 8: put option price using implied volatility

3 Explanation

From the above results, we can see using a fixed up/down factor method have a big error 20% for calls,60% for puts. Fixed volatility method are even worse, historical are better than GARCH and had a similar error as up/down factor method. This makes sense since we know the volatility has a smile curve and shouldn't be fixed values.

The best method should be implied volatility method, which is also predictable. Because what we did is using Black-Schole Model to compute implied volatility and plug it back to binomial model, and we know with large M, binomial will converge to Black-Schole Model. Hence, we should get a close value since what we did is similar to $F(F^{-1}(c_0)) = c_0$

However, using given implied volatility we get a bigger error than using calculated ones. This means the models we use are different from the model for market quotes. In fact, we found the given implied volatility are flatter than the calculated ones, so we think the model they used may have jumps(for example Levy process).

Finally, there is a big difference of errors for put and call option. We think this may resulted by too many zeros prices for put options.

4 Pricing GE option (maturity March 15 2015)

Based on the previous step, implied volatility method seems to have best performance. However, since there is no option quotes for maturity at March 15 2015 we can't use implied volatility. Instead we use up/down factor method which is second best among the four.
We calculated the price separately, with Table 4 for call options and Table 5 for put options.

Table 4: Mar 15,2015 call option pricing using crr model ($u = 1.007315$)

terDate	opType	strike price	option price
20150320	call	15.0	11.461282001246699
20150320	call	16.0	10.464700801353789
20150320	call	18.0	8.47153864941777
20150320	call	19.0	7.4749642726694931
20150320	call	20.0	6.4784966311719181
20150320	call	21.0	5.4829452187317358
20150320	call	22.0	4.4924554541957704
20150320	call	23.0	3.5213405813841572
20150320	call	24.0	2.6023390848848038
20150320	call	25.0	1.7868116952216715
20150320	call	26.0	1.1244037468974664
20150320	call	27.0	0.63865629860344442
20150320	call	28.0	0.33042940859411701
20150320	call	29.0	0.15175046822042076
20150320	call	30.0	0.063501731314656626
20150320	call	31.0	0.02402151939562441
20150320	call	32.0	0.0081636703833747361
20150320	call	33.0	0.002507714114669486
20150320	call	35.0	0.00018172046078672249

Table 5: Mar 15,2015 put option pricing using crr model ($u = 1.009611$)

terDate	opType	strike price	option price
20150320	put	15.0	9.7867179963705859e-09
20150320	put	16.0	3.6325419023585853e-07
20150320	put	18.0	7.639062809058851e-05
20150320	put	19.0	0.00057746451374074057
20150320	put	20.0	0.0030596500184336423
20150320	put	21.0	0.013060676245928219
20150320	put	22.0	0.041849964937799349
20150320	put	23.0	0.1136019753043295
20150320	put	24.0	0.25774564177442444
20150320	put	25.0	0.50647277601371621
20150320	put	26.0	0.88537267682497467
20150320	put	27.0	1.4046909519551112
20150320	put	28.0	2.0568312590497917
20150320	put	29.0	2.8207176450788012
20150320	put	30.0	3.6696609789588552
20150320	put	31.0	4.5798043461938622
20150320	put	32.0	5.5290513505741883
20150320	put	33.0	6.4999793185178305
20150320	put	34.0	7.4842697437628178
20150320	put	35.0	8.4748829594833452

References

[1] Black, F., and Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. Journal of Political Economy, 81(3), 637.

[2] Gong, H., Thavaneswaran, A., and Singh, J. (2010). A Black-Scholes Model with GARCH Volayility. Mathematical Scientist, 35(1), 37-42.

[3] Hull, J. (2008). Options, Futures, and Other Derivatives (7th ed.). Pearson Education. Molgedey, L. (2000). Historical Volatility Distribution In Gaussian And Garch(1,1) Models. International Journal of Theoretical and Applied Finance, 3(3), 417-417.

[4] Vejendla, A., and Enke, D. (2013). Evaluation of GARCH, RNN and FNN Models for Forecasting Volatility in the Financial Markets. IUP Journal of Financial Risk Management, 10(1), 41-49.

.1 Binomial Model--crrModel.py

```
#! /usr/bin/env python
import numpy as np
import scipy as sp
import pandas as pd
import mibian as mb
import matplotlib.pyplot as plt

def payoff ( S0, K, r, vola , T, M, opType, modelType):
        """
        This function is to calculate the option payoff at present
        using binomial option pricing model

        Inputs:
        ------------------------
        S0: present stock price
        K: strike price
        r: risk-free interest rate
        u: up factor
        T: terminal time(years)
        M: time steps
        opType: 'call'/'put', default value is call
        modelType: if choose 'vola', u is changed with volatility

        temp paras:
        ------------------------
        d: down factor, set as 1/u
        q: R.N. probability
        dt: time increment
        df: discounted factor

        Outputs:
        ------------------------
        discounted payoff of option at t=0
        """

        # para set up
        dt = T / M
        if modelType == 'vola':
                u = np.exp(vola * np.sqrt(dt)) # use volatility
                df = np.exp(-r * dt) # Continuous compounding IR
        else:
                u = vola # do not use volatility
                df = 1/(1+r) * dt # Compounding IR
        d = 1 / u # down factor
        q = (1/df - d) / (u - d) # r.n. probability
```

```
46
47          # intialization for stock price
48          S = np.zeros((M+1,M+1), dtype = np.float64)
49          S[0, 0] = S0
50          z = 0
51          for j in range(1, M + 1, 1):
52                  z += 1
53                  for i in range(z + 1):
54                          S[i, j] = S[0, 0] * (u ** j) * (d ** (i * 2))
55
56          # calculate payoff at each node
57          payoff = np.zeros((M+1, M+1), dtype = np.float64)
58          z = 0
59          if opType == 'call':
60                  for j in range(0, M + 1, 1):
61                          for i in range(z+1):
62                                  payoff[i, j] = max(S[i, j] - K, 0)
63                          z += 1
64          if opType == 'put':
65                  for j in range(0, M + 1, 1):
66                          for i in range(z+1):
67                                  payoff[i, j] = max(K - S[i, j] , 0)
68                          z += 1
69          # valuation discounted payoff by R.N. meansure
70          payoffStar = np.zeros((M+1, M+1), dtype = np.float64)
71          payoffStar[:, M] = payoff[:, M]
72          z = M + 1
73          for j in range(M - 1, -1, -1):
74                  z -=1
75                  for i in range(z):
76                          payoffStar[i, j] = (q * payoffStar[i, j + 1] + (1 - q) * payoffStar[i+1, j+1]) *
                                df
77          return payoffStar[0, 0]
```

.2 Pricing all options--crrPrice.py

```
1  #! /usr/bin/env python
2  import numpy as np
3  import scipy as sp
4  import pandas as pd
5  import mibian as mb
6  import matplotlib.pyplot as plt
7  import crrModel
8
9  """
10 Part1: Reading option,stock Data From file
```

```
"""
# read data from csv data file
opData = pd.read_csv('geOpQuotes.txt')
stData = pd.read_csv('geStPrice.txt')
n = len(opData)  # number of options to price
unPrice = stData['adjClose'][0] # current underlying price of stock
r = 0.01 # risk-free insterest rate


"""
Part2: calculation of u(up factor in crr model)

Step1: select ATM option data
Step2: define function F(u)=crrModel.payoff(.,u)-optionPrice
Step3: use a non-linear solver to solve for u
"""
uData = opData[(opData['strike']==round(unPrice*2)/2)] # find data of ATM option
k = len(uData) # number of u's used for calibration
u = np.zeros(k, dtype=np.float64)
for i,j in zip(uData.index,range(k)):
        def F(u):
                return crrModel.payoff(unPrice,uData['strike'][i],r, u,\
                        uData['deltaT'][i]/365.0, uData['deltaT'][i],\
                        uData['opType'][i],'novola')-uData['ask'][i]

        u[j] = sp.optimize.newton_krylov(F, 1.02)

pd.DataFrame({'u':u, 'terDate':uData['terDate'], 'opType':uData['opType']}).to_csv('geUpFactor.txt')

"""
Part3: calculation of listed option price

opPriceU: use crr model without volatility
opPriceVola1: use crr model with calculated vola (used 'mibian.BS' model)
opPriceVola0: use crr model with given vola
"""
# create temp para
opPriceVola0 = np.zeros( n , dtype = np.float64) # payoff using given implied volatility
opPriceVola1 = np.zeros( n , dtype = np.float64) # payoff using implied volatility
opPriceU = np.zeros( (n,k) , dtype = np.float64) # payoff using u
imVola1 = np.zeros( n, dtype = np.float64) # implied volatility

# calculate implied volatility and option price for each contract
for i in range(n):
                if opData['deltaT'][i] <= 0: continue # incase expired options are still on website
                if opData['opType'][i]=='call':
                        c = mb.BS( [ unPrice, opData['strike'][i], r, opData['deltaT'][i] ],\
```

```
                                        callPrice = opData['ask'][i])
                        imVola1[i] = c.impliedVolatility/100.0
                else:
                        p = mb.BS( [ unPrice, opData['strike'][i], r, opData['deltaT'][i] ],\
                                        putPrice = opData['ask'][i])
                        imVola1[i] = p.impliedVolatility/100.0

                opPriceVola1[i] = crrModel.payoff( unPrice, opData['strike'][i], r, imVola1[i],\
                                opData['deltaT'][i]/365.0,  opData['deltaT'][i],\
                                opData['opType'][i],'vola')
                opPriceVola0[i] = crrModel.payoff( unPrice, opData['strike'][i], r, opData['imVola0'][i
                    ],\
                                opData['deltaT'][i]/365.0,  opData['deltaT'][i],\
                                opData['opType'][i],'vola')

                for j in range(k):
                        opPriceU[i,j] = crrModel.payoff( unPrice, opData['strike'][i], r,u[j] ,\
                                        opData['deltaT'][i]/365.0,  opData['deltaT'][i],\
                                        opData['opType'][i],'novola' )
print 'calculation of option price --- Done!'


# writing data to files
for i in range(k):
        opData['opPriceU%d'%i]=pd.Series(opPriceU[:,i],index=opData.index)
        opData['errPriceU%d'%i]=pd.Series(np.abs(opData['opPriceU%d'%i]-opData['ask'])/opData['ask'],\
                        index=opData.index)
opData['imVola1']=pd.Series(imVola1,index=opData.index)
opData['errVola']=pd.Series(np.abs(opData['imVola1']-opData['imVola0'])/opData['imVola0'],\
                index=opData.index)
opData['opPriceVola1']=pd.Series(opPriceVola1,index=opData.index)
opData['errPriceVola1']=pd.Series(np.abs(opData['opPriceVola1']-opData['ask'])/opData['ask'],\
                index=opData.index)
opData['opPriceVola0']=pd.Series(opPriceVola0,index=opData.index)
opData['errPriceVola0']=pd.Series(np.abs(opData['opPriceVola0']-opData['ask'])/opData['ask'],\
                index=opData.index)

opData.to_csv('geOpPriceOut.txt')
```

.3 Testing methods--crrTesting.py

```
#! /usr/bin/env python
import numpy as np
import scipy as sp
import pandas as pd
import mibian as mb
```

```
import matplotlib.pyplot as plt
import os
ls=os.linesep

"""
Part1: Reading option,stock Data From file
"""
# read data from csv data file
opData=pd.read_csv('geOpPriceOut.txt')
stData = pd.read_csv('geStPrice.txt')
uData = pd.read_csv('geUpFactor.txt')
k = len(uData)  # number of u
cData = opData[(opData['opType']=='call')]
pData = opData[(opData['opType']=='put')]
"""
Part2: Testing results—model:crr with u
"""
f=open('geUcallStats.txt', 'w')
f.write('u'+','+ 'errMean' +',' +'errStd'+ls)
for i in range(k):
        f.write(str(uData['u'][i])+','\
                        +str(sp.mean(cData['errPriceU%d'%i]))+','\
                        +str(sp.std(cData['errPriceU%d'%i]))+ls)
f.write('imVola1'+','\
                +str(sp.mean(cData['errPriceVola1']))+','\
                +str(sp.std(cData['errPriceVola1']))+ls)
f.write('imVola0'+','\
                +str(sp.mean(cData['errPriceVola0']))+','\
                +str(sp.std(cData['errPriceVola0']))+ls)
f.close()

f=open('geUputStats.txt', 'w')
f.write('u'+','+ 'errMean' +',' +'errStd'+ls)
for i in range(k):
        f.write(str(uData['u'][i])+','\
                        +str(sp.mean(pData['errPriceU%d'%i]))+','\
                        +str(sp.std(pData['errPriceU%d'%i]))+ls)
f.write('imVola1'+','\
                +str(sp.mean(pData['errPriceVola1']))+','\
                +str(sp.std(pData['errPriceVola1']))+ls)
f.write('imVola0'+','\
                +str(sp.mean(pData['errPriceVola0']))+','\
                +str(sp.std(pData['errPriceVola0']))+ls)
f.close()


"""
```

```
Part4: Plot
"""
ucOpt=2
upOpt=1
ucOptValue=uData['u'][ucOpt]
upOptValue=uData['u'][upOpt]
col=4
row=np.ceil(cData['terDate'].nunique()/float(col))

# call option plot, model: crr with u
for i,j in zip(cData['terDate'].unique(), range(cData['terDate'].nunique())):
        data = cData[(cData['terDate']==i)]
        plt.subplot(row, col , j+1)
        plt.title('%sC' %i)
        plt.xlabel('strike price')
        plt.ylabel('option price')
        plt.plot(data['strike'],data['opPriceU%d'%ucOpt], 'r+', label='pred price')
        plt.plot(data['strike'],data['ask'], 'go', label='market price')
        plt.legend(prop={'size':9})
plt.tight_layout(pad=0.1, w_pad =0.1, h_pad=0.1)
plt.show()

# put option plot, model: crr with u
for i,j in zip(pData['terDate'].unique(), range(pData['terDate'].nunique())):
        data = pData[(pData['terDate']==i)]
        plt.subplot(row, col , j+1)
        plt.title('%sP' % i)
        plt.xlabel('strike price')
        plt.ylabel('option price')
        plt.plot(data['strike'],data['opPriceU%d'%upOpt], 'r+', label='pred price')
        plt.plot(data['strike'],data['ask'], 'go', label='market price')
        plt.legend(loc=2, prop={'size':9})
plt.tight_layout(pad=0.8, w_pad =0.8, h_pad=1.0)
plt.show()


# call option plot, model: crr with imVola
for i,j in zip(cData['terDate'].unique(), range(cData['terDate'].nunique())):
        data = cData[(cData['terDate']==i)]
        plt.subplot(row, col , j+1)
        plt.title('%sC' %i)
        plt.xlabel('strike price')
        plt.ylabel('option price')
        plt.plot(data['strike'],data['opPriceVola1'], 'r+', label='pred price')
        plt.plot(data['strike'],data['ask'], 'go', label='market price')
        plt.legend(prop={'size':9})
plt.tight_layout(pad=0.1, w_pad =0.1, h_pad=0.1)
```

```
plt.show()

# put option plot, model: crr with imVola
for i,j in zip(pData['terDate'].unique(), range(pData['terDate'].nunique())):
        data = pData[(pData['terDate']==i)]
        plt.subplot(row, col , j+1)
        plt.title('%sP' % i)
        plt.xlabel('strike price')
        plt.ylabel('option price')
        plt.plot(data['strike'],data['opPriceVola1'], 'r+', label='pred price')
        plt.plot(data['strike'],data['ask'], 'go', label='market price')
        plt.legend(loc=2, prop={'size':9})
plt.tight_layout(pad=0.8, w_pad =0.8, h_pad=1.0)
plt.show()
```

.4 Pricing particular options--crrCalc.py

```
#! /usr/bin/env python
import numpy as np
import scipy as sp
import pandas as pd
import mibian as mb
import matplotlib.pyplot as plt
import crrModel


"""
 Calculate Option Price on March 15, 2015
"""
# find closest listed option to get strike price
opData=pd.read_csv('geOpQuotes.txt')
stData=pd.read_csv('geStPrice.txt')
uData=pd.read_csv('geUpFactor.txt')
opData=opData[(opData['terDate']==20150320)]
unPrice=stData['adjClose'][0]
ucOpt=uData['u'][2]
upOpt=uData['u'][1]
r=0.01
n=len(opData)
opPrice=np.zeros(n, dtype=np.float64)
opData.index=[a for a in range(n)]
for i in opData.index:
        if opData['opType'][i]=='call':
                opPrice[i]=crrModel.payoff(unPrice, opData['strike'][i],r,\
                                ucOpt, (opData['deltaT'][i]-5)/365.0,\
                                opData['deltaT'][i]-5,opData['opType'][i],'novola')
```

```
        else:
                opPrice[i]=crrModel.payoff(unPrice, opData['strike'][i], r,\
                                upOpt, (opData['deltaT'][i]-5)/365.0,\
                                opData['deltaT'][i]-5,opData['opType'][i],'novola')


opData['opPrice']=opPrice
opData=opData[['terDate','opType','strike','opPrice']]
opData.to_csv('ge20150315.txt',index=False)
```

.5 Scraping option quotes--scrOpQuotes.py

```
#! /usr/bin/env python
#coding=utf-8
"""
This program is to scrape GE options quotes
from  fiance.yahoo website

Author: Da Li           Nov 2, 2014

"""
import os
import datetime as dt
import time
from calendar import datetime
import urllib2
from bs4 import BeautifulSoup as BS
ls=os.linesep

# creat a txt file  to store data
f=open('geOpQuotes.txt','w')
# write header for data
f.write('terDate'+','\
        +'deltaT'+','\
        +'opType'+',' \
        +'strike'+ ',' \
        +'contract'+',' \
        +'last'+',' \
        +'bid'+',' \
        +'ask'+',' \
        +'change'+',' \
        +'%change'+',' \
        +'volume'+',' \
        +'openInterest'+',' \
        +'imVola0'+ls)
# define url, search for datelist of option quotes
```

```
n= 0
url="http://finance.yahoo.com/q/op?s=GE"
terDate = BS(urllib2.urlopen(url)).find_all('option') # terminal date of op, listed  on website
today = dt.datetime.utcnow().replace(hour=0,minute=0,second=0,microsecond=0)
for t in terDate:
        # create soup obj
        timestamp = timegm(time.strptime(t.text+'GMT', '%B %d, %Y%Z')) # same as terDate
        url= "http://finance.yahoo.com/q/op?s=GE" + '&date=' + str(timestamp)
        data=BS(urllib2.urlopen(url)).find_all('td') # this is  the op data
        i=3
        # write data into file
        while i<=len(data)-9:
                if data[i].text==u'\n\n\n\u2715\n[modify]\n\n':
                        i +=1
                f.write( '20'+data[i+1].text[3:9]+',') # write terDate
                deltaT = dt.datetime.utcfromtimestamp(timestamp) - today
                f.write( str(deltaT.days) + ',' ) # write days to terminial date
                if data[i+1].text[9:10]=='C':
                        f.write('call') # o stand for call
                else:
                        f.write('put') # 1 stand for put

                for j in range(i, i+10, 1):
                        if j == i+9:
                                f.write(','+ str(float(data[j].text[1:-1].strip('%'))/100)  )
                        else:
                                f.write(','+ data[j].text[1:-1]) # write all other data
                i +=10
                f.write(ls)
        print( "scraping GE option on %s ---- Done!"%  t.text)
f.close()
```

.6 Scraping stock prices--scrStPrice.py

```
#! /usr/bin/env python
#coding=utf-8
"""
This program is to scrape GE stock prices(past 3 months)
from  fiance.yahoo website

Author: Da Li           Nov 2, 2014

"""
import os
import datetime as dt
import urllib2
```

```
13 from bs4 import BeautifulSoup as BS
14 ls=os.linesep
15
16 # creat a txt file  to store data
17 f=open('geStPrice.txt','w')
18 # write header for data
19 f.write('Date'+','\
20         +'Open'+','\
21         +'High'+',' \
22         +'Low'+ ',' \
23         +'Close'+',' \
24         +'Volume'+',' \
25         +'adjClose'+ls)
26 # define url, search for stock price data
27 i = 0
28 url="http://finance.yahoo.com/q/hp?s=GE"
29 data = BS(urllib2.urlopen(url)).find_all('td',{'class':'yfnc_tabledata1'})
30
31 # write data into file
32 while i<=len(data)-6:
33         if 'Dividend' in data[i+1].text:
34                 i +=2
35         for j in range(i, i+7, 1):
36                 if j == i: f.write(data[j].text)
37                 else: f.write(','+ data[j].text)
38         i +=7
39         f.write(ls)
40 print "scraping GE stock price --- Done!"
41 f.close()
```