Option Pricing Project

Cong Cao

Our Incentives

In the derivatives analytics field, every market-maker in the financial market is concerned about the price of those options on a daily basis. Given prices of liquidly traded options, those investors try to parametrize models in a way that replicate the observed option prices as well as possible. This activity is generally referred to as model calibration.

The Black-Scholes model for example, made some unrealistic assumptions:

- Constant Volatility
- Risk Free Rate Unchanged
- Time Continuity

Model Description

Our option pricing model combine the stochastic volatility and jump-diffusion part, we can get the SDE for underlying asset with following form:

First part, stochastic volatility By Heston(1993)

SDE form:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dz_{1t}$$

$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dz_{2t}$$

Second Part, Jump Diffusion BSM Model (1976)

SDE form:

$$dS_t = (r - r_J)S_t dt + \sigma S_t dW_t + J_t S_t dN_t$$

Combining this two parts:

$$dS_t = (r - r_J)S_t dt + \sqrt{v_t} S_t dZ_t^1 + J_t S_t dN_t$$

$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t}\, dZ_t^2$$

S_t : underlying price at date t;

r : constant risk-free rate;

r_J : $r_J \equiv \lambda \cdot \left(e^{\mu_J + \frac{\delta^2}{2}} - 1\right)$,drift correlation for jump;

v_t : variance at date t;

κ : speed of adjustment of v_t;

θ: the long-term average of the variance;

σ : volatility coefficient;

Z_t^n $(\mathrm{n} = 1, 2)$: standard Brownian motions:

$$dZ_t^1 dZ_t^2 \equiv \rho dt;$$

J_t : jump diffusion at date t with:

$$log(1 + J_t) \approx N\left(log(1 + \mu_J) - \frac{\delta^2}{2}, \delta^2\right)$$

N_t : Poisson process with intensity λ

PDE Approach for European Call Option Price Ct

$$dC_t = \frac{\partial C_t}{\partial S_t}(m_t dt + v_t dZ_t^1 + j_t dN_t) + \frac{\partial C_t}{\partial v_t}(\bar{m}_t dt + \bar{v}_t dZ_t^2) + \frac{\partial^2 C_t}{\partial S_t \partial v_t} v_t \bar{v}_t \rho dt$$

$$+ \frac{1}{2}\left(\frac{\partial^2 C_t}{\partial S_t^2} v_t^2 + \frac{\partial^2 C_t}{\partial v_t^2} \bar{v}_t^2 + \frac{\partial C_t}{\partial t}\right) dt$$

$$= \left(\frac{\partial C_t}{\partial S_t} m_t + \frac{\partial C_t}{\partial v_t} \bar{m}_t + \frac{1}{2}\frac{\partial^2 C_t}{\partial S_t^2} v_t^2 + \frac{1}{2}\frac{\partial^2 C_t}{\partial v_t^2} \bar{v}_t^2 + \frac{\partial^2 C_t}{\partial S_t \partial v_t} v_t \bar{v}_t \rho + \frac{\partial C_t}{\partial t}\right) dt$$

$$+ \frac{\partial C_t}{\partial S_t} v_t dZ_t^1 + \frac{\partial C_t}{\partial v_t} \bar{v}_t dZ_t^2$$

$$m_t = (r - r_J) S_t$$

$$\bar{m}_t = \kappa(\theta - v_t)$$

$$v_t = \sqrt{v_t} S_t$$

$$\bar{v}_t = \sigma \sqrt{v_t}$$

$$j_t = J_t S_t$$

finally, we can get a solution to this PDE:

$$C_t(K, T, S_t, v_t, r, t) = S_t \cdot \Pi_1(T; S_t, v_t, r, t) - e^{-r(T-t)} \cdot K \cdot \Pi_2(T; S_t, v_t, r, t)$$

$$\Pi_j = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re\left[\frac{e^{-iu\ln K}\varphi_j(T; u)}{iu}\right] du, \qquad j = 1,2$$

$$\varphi_1(T; u) = \frac{\varphi(T; u - i)}{\varphi(T; -i)}, \quad \varphi_2(T; u) = \varphi(T; u)$$

Data Source

We are using call options prices from April 2017

1. Around the spot prices.
2. Maturity
3. Bid-Ask

Finally we get 161 calls option prices daily for calibration !

```python
def extract_train(file_list=file_list):
    start = datetime.strftime(datetime.strptime('2017-4-15', '%Y-%m-%d'), '%m/%d/%Y')
    end = datetime.strftime(datetime.strptime('2017-06-01', '%Y-%m-%d'), '%m/%d/%Y')
    for file_name in file_list:
        reader = pd.read_csv('%s\%s' % (data_path, file_name), index_col='OptionSymbol',
                             usecols=['UnderlyingSymbol', 'UnderlyingPrice', 'OptionSymbol', 'Type',
                                      'Expiration', 'DataDate', 'Strike', 'Bid', 'Ask'])
        train_file = reader.loc[reader['UnderlyingSymbol'] == 'XLB']
        spot = train_file.head(1).loc[:, 'UnderlyingPrice'].tolist()[0]
        upper = 1.1 * spot
        lower = 0.9 * spot
        train_data = train_file.loc[train_file['Strike'] <= upper]
        train_data = train_data.loc[train_data['Strike'] >= lower]
        train_data['Bid'] = train_data['Bid'].replace(0, np.nan)
        train_data = train_data.dropna(how='any')
        mid = (train_data['Bid'] + train_data['Ask'])/2
        train_data = pd.concat([train_data, mid], axis=1)
        col_name = train_data.columns.tolist()
        col_name[-1] = 'Mid'
        train_data.columns = col_name
        train_data = train_data.loc[train_data['Expiration'] <= end]
        train_data = train_data.loc[train_data['Expiration'] >= start]
        train_data.to_csv('%s\XLB_%s_train.csv' % (data_path, file_name[11:19]))
        print('read %s' % file_name)
```

```python
def extract_contract(contract_list=contract_list, file_list=file_list):
    for contract in contract_list:
        price = []
        time = []
        expire = datetime.strftime(datetime.strptime(reader1.loc[contract, 'Expiration'], '%m/%d/%Y'), '%Y%m%d')
        strike = reader1.loc[contract, 'Strike']
        for file_name in file_list:
            if expire >= file_name[11:19]:
                reader = pd.read_csv('%s\%s' % (data_path, file_name), index_col='OptionSymbol',
                                     usecols=['UnderlyingSymbol', 'OptionSymbol', 'Type',
                                              'Expiration', 'DataDate', 'Strike', 'Bid', 'Ask'])
                cur_time = datetime.strftime(datetime.strptime(reader.loc[contract, 'DataDate'], '%m/%d/%Y'), '%Y-%m-%d')
                price.append((reader.loc[contract, 'Bid']+reader.loc[contract, 'Ask'])/2)
                time.append(cur_time)
            print('read %s' % file_name)
        expire_col = [datetime.strftime(datetime.strptime(expire, '%Y%m%d'), '%Y-%m-%d')]*len(price)
        strike_col = [strike]*len(price)
        df = pd.DataFrame([expire_col, price, strike_col], columns=time, index=['Expiration', 'Mid', 'Strike'])
        df = df.T
        df.to_csv('%s.csv' % contract)
```

Calibration

Brute and fmin

How we did the calibration?

```python
def calibration_short():

    # first run with brute force
    # (scan sensible regions)
    global local_opt, opt1
    opt1 = 0.0
    local_opt = True
    opt1 = brute(BCC_error_function,
                 ((0.0, 0.51, 0.1), # lambda
                  (-0.5, -0.11, 0.1), # mu
                  (0.0, 0.51, 0.25)), # delta
                  finish=None)
    # second run with local, convex minimization
    # (dig deeper where promising)
    opt2 = fmin(BCC_error_function, opt1,
                xtol=0.0000001, ftol=0.0000001,
                maxiter=550, maxfun=750)

    return opt2
```

```python
def error_function(p0):
    '''

    Parameters
    ==========
    kappa_v: float
        mean-reversion factor
    theta_v: float
        long-run mean of variance
    sigma_v: float
        volatility of variance
    rho: float
        correlation between variance and stock/index level
    v0: float
        initial, instantaneous variance

    Returns
    =======
    MSE: float
        mean squared error
    '''
    global i1, min_MSE1
    kappa_v, theta_v, sigma_v, rho, v0 = p0
    if kappa_v < 0.0 or theta_v < 0.005 or sigma_v < 0.0 or rho < -1.0 or rho > 1.0:
        return 500.0
    if 2 * kappa_v * theta_v < sigma_v ** 2:
        return 500.0
    se = []
    for option in options:
        model_value = option.opt_H93_value(kappa_v, theta_v, sigma_v, rho, v0)
        se.append((model_value - option.close) ** 2)
    MSE = sum(se) / len(se)
    min_MSE1 = min(min_MSE1, MSE)
    if i1 % 25 == 0:
        print('%4d |' % i1, np.array(p0), '| %11.7f | %11.7f' % (MSE, min_MSE1))
    i1 += 1
    return MSE
```

Delta Hedging

Think as a Trading Strategy.

The first day:

sell delta shares of stock and buy a call option at market price at the end of the day, and the delta is calculated by the the parameters that we calibrated.

The next day:

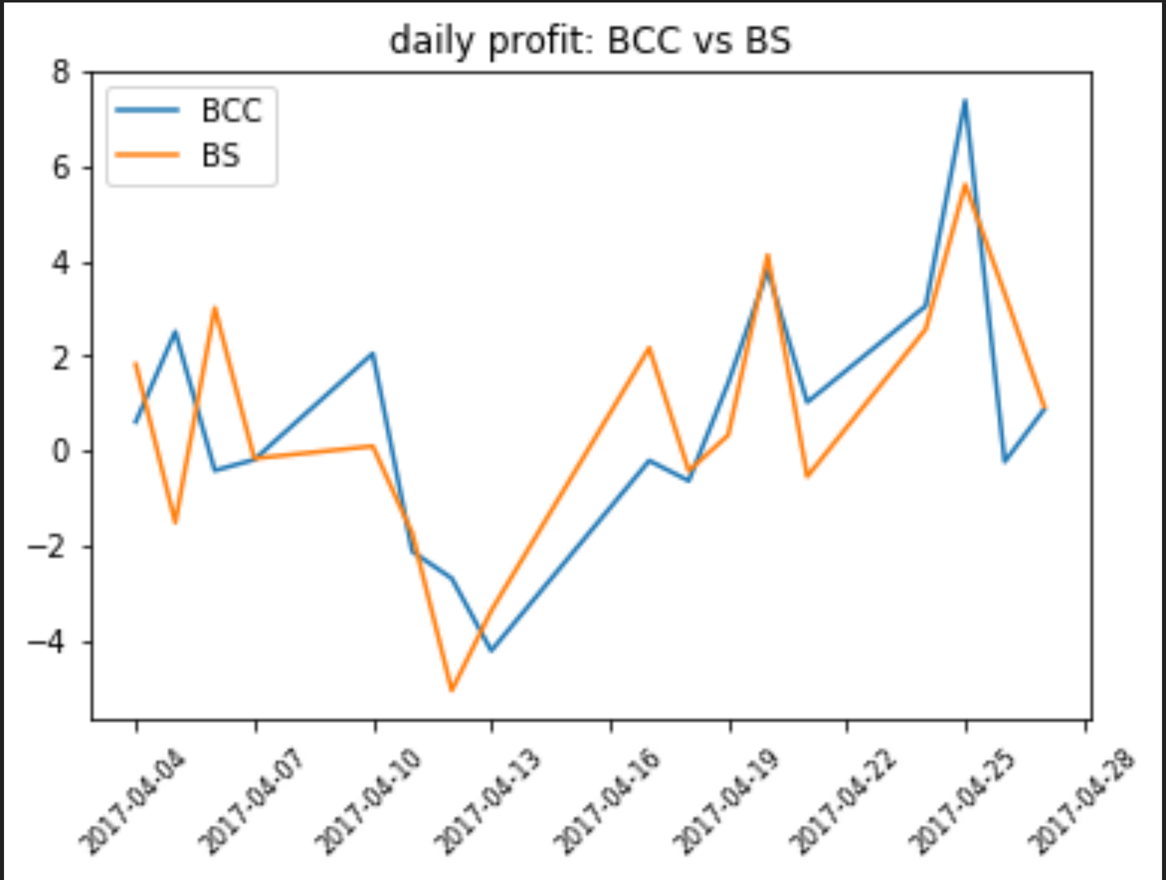
Take the adjustment of the delta by the parameters we calibrated, using the stock price and call option price at that day.

Results in Graphs

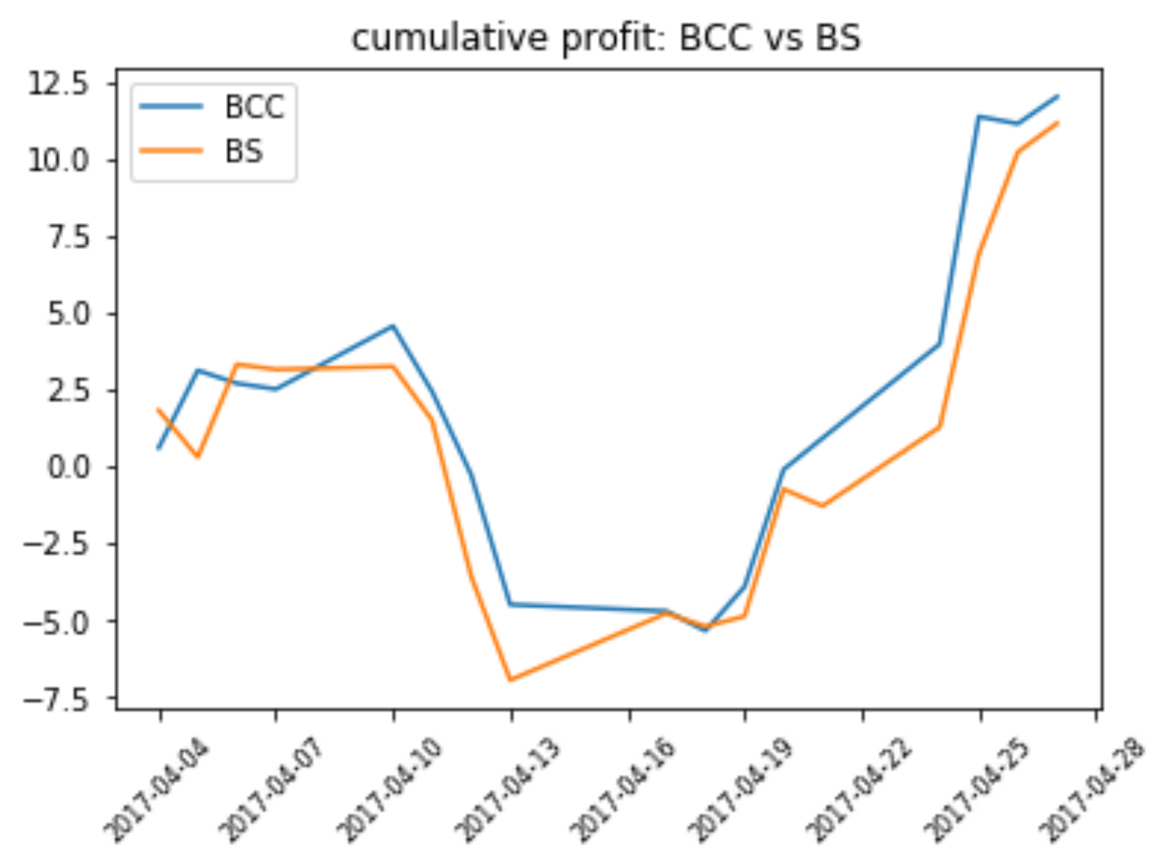
- Daily Profit Comparison
- Cumulative Profit Comparison
- Call Price Comparison between BS and our model

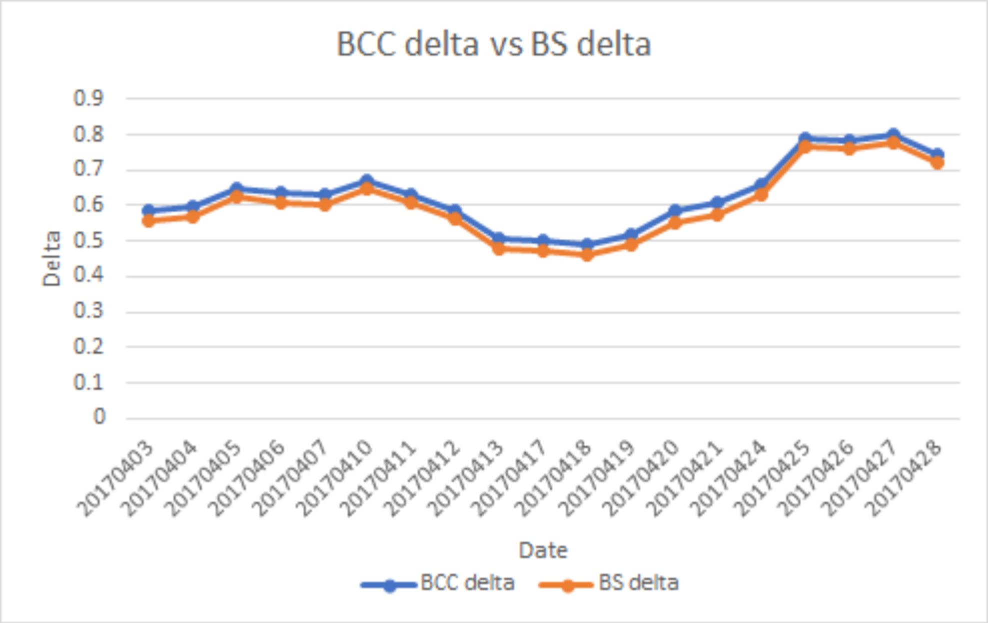
Results in Graphs

Not obvious

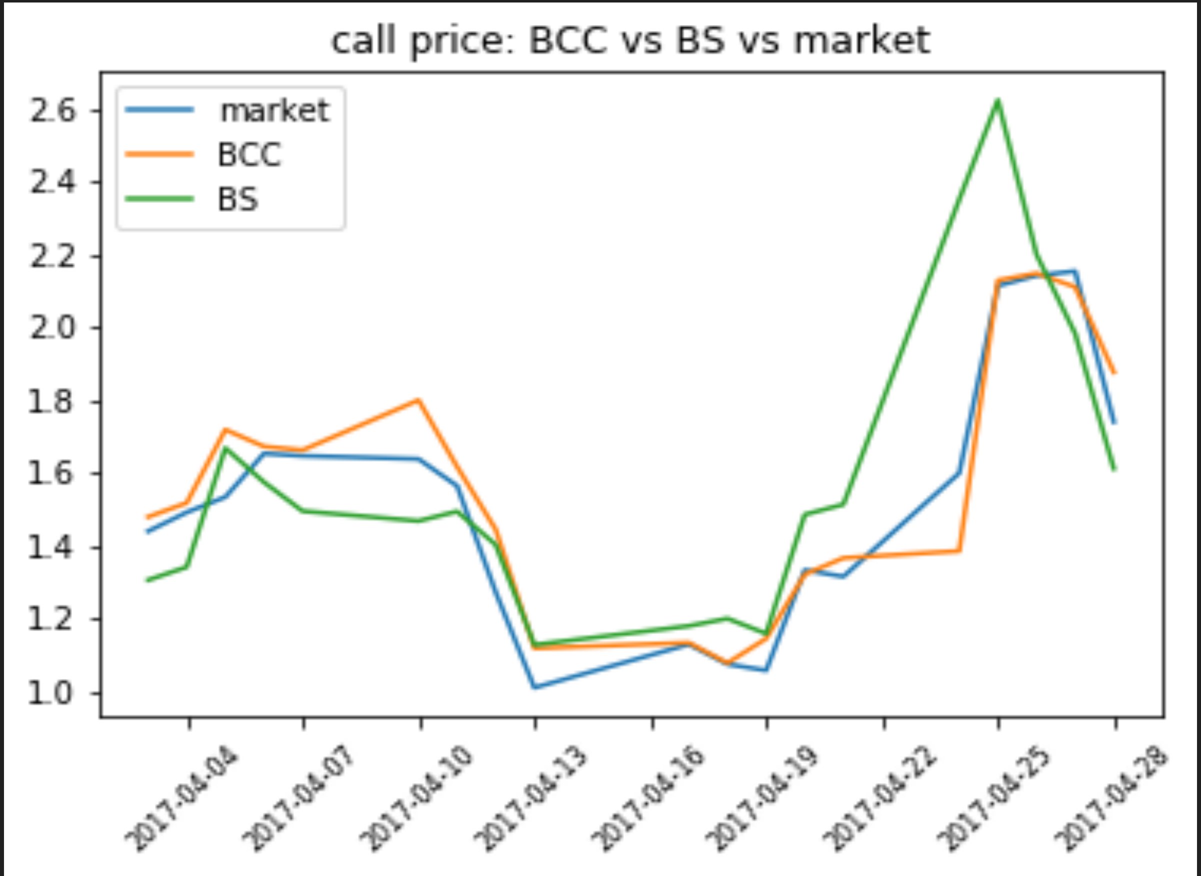


Results in Graphs




BCC delta vs BS delta
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Delta
20170403
20170404
20170405
20170406
20170407
20170410
20170411
20170412
20170413
20170417
20170418
20170419
20170420
20170421
20170424
20170425
20170426
20170427
20170428
Date
BCC delta
BS delta

Results in Graphs



Potential Drawbacks and Improvement

- More Complicated than BS model.
- Overfitting parameters.
- We used daily data. What if monthly?
- Constant risk-free rate (same as BS).