Lab5_sol

October 3, 2017

1 Sample solutions of Lab 5 exercises

In this lab the information about bid prices of 3.5-months call options (expiration on January 19,2018) on Apple stock was used. The information (as of 10:15 on October 3, 2017) was as follows:

```
Strike  Price   Change  Bid Ask Volume  Open Int
130.00  24.90   -0.10   24.95   25.30   50  44576
135.00  20.70   +0.13   20.60   20.95   24  31958
140.00  16.85   0.00    16.65   16.90   75  70576
145.00  13.20   -0.14   13.05   13.25   73  23506
150.00  10.00   -0.09   9.95    10.05   431 52565
155.00  7.45    0.00    7.35    7.45    1263    36208
160.00  5.40    +0.05   5.30    5.35    1913    48943
165.00  3.75    0.00    3.65    3.80    391 35171
170.00  2.57    0.00    2.52    2.57    1231    55875
175.00  1.75    0.00    1.69    1.74    2412    82194
current price of stock: 153.81
```

1.1 Exercise 1

(Implied volatility) Let us assume that the Black-Scholes market model with constant volatility holds, then call option prices can be computed by Black-Scholes formula. Assume $r = 0.02$ and $D = 0$, then the only unknown parameter is σ. Since we have only one unknown parameter, only one equation is needed to determine the value of σ and if the assumption about the market model is correct, then every known option price should give the same value of σ. Use the equation solver `fsolve` from `scipy.optimize` to find a value of σ for each of 10 actual option prices corresponding to 10 strike prices closest to the current share price. Show the dependence of found σ values on the exercise price on a graph. In order to do this, for each value of the exercise price define a function of one argument σ that computes the difference of the corresponding theoretical call option price and the observed price of the option and use this function as an input for the command `optimize.fsolve` together with a suitable initial guess for the parameter σ.

Solution Assume that the data is in the file `apple_call.txt` in the directory `h:/compfin_labs/`. We need the pairs (exercise_price,option_price) and we are using ask prices as current option prices (so data in the first and fifth column). Note that the columns are separated by tabulators and that there is a header line (with column names) before the actual data.

```
In [1]: import numpy as np
        from scipy import optimize
```

```
        #read in the data
        E,V=np.loadtxt("h:/compfin_labs/apple_call.txt",delimiter="\t",usecols=(0,4),skiprows=
```

Define variables with the current market data:

```
In [2]: T=3.5/12
        r=0.02
        D=0
        S0=153.81
```

We need the function that computes Call option prices according to Black-Scholes formula. You can use the file t'BSformulas.pyt' created during lab 2. Importing the function for the Call option can be done as follows:

```
In [3]: import sys
        sys.path.append("h:/compfin_labs") #the location of the file
        import BSformulas as bs
```

In order to use the price for a given Call option for a given Exercises price E and exercise time T, we have only one unknown parameter σ. By the implied volatility approach, we use a known pair (exercise_price,option_price) to find such σ that the value given by BS formula equals to the given option price. As the function `fsolve` in the `optimize` subpackage of the `scipy` package finds a value x for which a given function f is zero (solves the equation f(x)=0), we'll define a function that computes the difference of the value coming from the BS formula and the observed option price and use `fsolve` to find σ, for which the difference is 0.

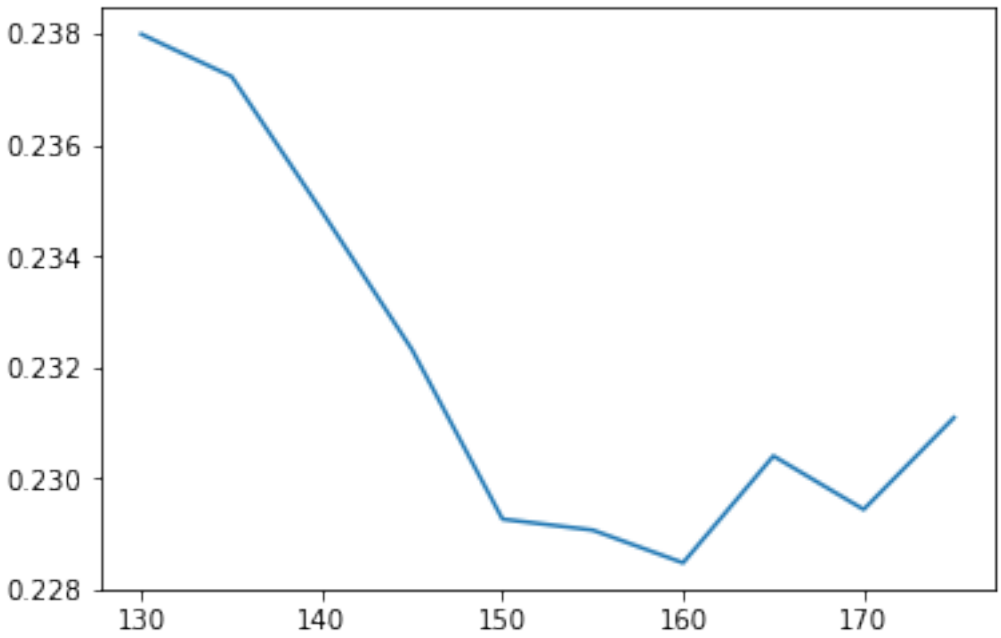
```
In [4]: i=0 #consider the first pair of observed exercise price and corresponding option price
        def f(sigma):
            return bs.Call(S=S0,r=r,D=D,sigma=sigma,T=T,E=E[i])-V[i]
        sigma=optimize.fsolve(f,0.2)
        print(sigma)
```

```
[ 0.23799412]
```

Let us do it for all pairs of (exercise price, option price) and save the results in a vector.

```
In [5]: m=np.size(E)
        sigma_values=np.zeros(m)
        for i in range(m):
            sigma_values[i]=optimize.fsolve(f,0.2)
        print(sigma_values)
        import pylab as pl
        pl.plot(E,sigma_values)
        pl.show()
```

```
[ 0.23799412  0.23723279  0.23480393  0.23231091  0.22926645  0.22906879
  0.22847974  0.2304032   0.22943979  0.23109393]
```



If the BS market model with constant volatility holds, then all implied volatilities should be the same (all option prices should correspond to the same market model). As we see, the implied volatilities are not constant, but actually they change only a little. So, based on the data, we can say that the BS model with constant volatility does not hold (and thus the option prices can not be computed exactly by BS fromulas), but we may still get quit reasonable approximate prices by using the simplified market model.

1.2 Exercise 2

Define a function that for a given value of σ computes the sum of squares of differences of the theoretical and observed option prices and use a minimizer from `scipy.optimize` to find the least squares estimate of σ. For this σ, find the largest difference between the theoretical and observed option prices.

Solution

```
In [6]: def F(sigma):
            return 1/2*np.sum((bs.Call(S=S0,sigma=sigma,D=D,r=r,E=E,T=T)-V)**2)
        best_sigma=optimize.fmin(F,0.5)
        print("The overall best sigma value:",best_sigma)
        print("Maximal absolute difference:",np.max(np.abs(bs.Call(S=S0,sigma=best_sigma,D=D,r=
```

```
Optimization terminated successfully.
         Current function value: 0.021187
         Iterations: 14
         Function evaluations: 28
The overall best sigma value: [ 0.23066406]
Maximal absolute difference: 0.113424887799
```

1.3 Exercise 3

Consider Black-Scholes market model with the non-constant volatility

$$\sigma(s,t) = |\theta_0 + \theta_1 \arctan(0.02 \cdot (s - 152))|.$$

From the course web page you can download a module `lab5solver` that contains a function `lab5solver(theta,E,S0)` that for a given values of E and $S0$ and for a given parameter vector θ computes the theoretical price of a call option. Use the function and the option price data to find suitable values of θ_0 and θ_1.

Solution Assume that the file `lab5solver.pyc` is located at the same place as the file `BSformulas`. Then we can import a function from the module without additional commands (showing the location of the file).

```
In [7]: import lab5solver as l5
```

As this function does not work with vectors, we have to use for cycle to compute the sum of the squared differences

```
In [8]: def F2(theta):
            result=0
            for i in np.arange(m):
               result=result+1/2*(l5.lab5solver(theta,E[i],S0)-V[i])**2
            return result
        theta=optimize.fmin(F2,[0.2,0])
        print(theta)

Optimization terminated successfully.
         Current function value: 0.013500
         Iterations: 39
         Function evaluations: 73
[ 0.17568656  0.02560255]
```

As the value of the second parameter is very small, the fitted volatility function is practically equal to a constant value and so the prices obtained from this model are probably not much more accurate than those corresponding to the simple model with constant volatility.