BS_Hedging

Antoine Jacquier

Title:	BS_Hedging
Author:	Antoine Jacquier
Number of pages:	6
First version:	February 13, 2017
Current version:	February 13, 2017
Revision:	0.0.0

Contents

1 Hedging strategy in the Black-Scholes model

In [26]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
from scipy.optimize import bisect
from math import isnan
```

1.1 Black-Scholes functions

In [27]:
```
def phi(x):  # Gaussian density
    return np.exp(-0.5 * x * x) / np.sqrt(2. * np.pi)

# Black Sholes Function


def BlackScholesCore(CallPutFlag, DF, F, X, T, v):
    vsqrt = v * np.sqrt(T)
    try:
        d1 = (np.log(F / X) + (vsqrt * vsqrt / 2.)) / vsqrt
    except:
        d1 = 0.
    d2 = d1 - vsqrt
    if CallPutFlag:
        return DF * (F * norm.cdf(d1) - X * norm.cdf(d2))
    else:
        return DF * (X * norm.cdf(-d2) - F * norm.cdf(-d1))

# Black Sholes Vega


def BlackScholesVegaCore(DF, F, X, T, v):
    vsqrt = v * np.sqrt(T)
    d1 = (np.log(F / X) + (vsqrt * vsqrt / 2.)) / vsqrt
    return F * phi(d1) * np.sqrt(T) / DF

# Black Sholes Delta


def BlackScholesDelta(CallPutFlag, S, X, T, r, d, v):
    vsqrt = v * np.sqrt(T)
    F = np.exp((r - d) * T) * S
```

```
        try:
            d1 = (np.log(F / X) + (vsqrt * vsqrt / 2.)) / vsqrt
        except:
            d1 = 0.
        d2 = d1 - vsqrt
        if CallPutFlag:
            return norm.cdf(d1)
        else:
            return norm.cdf(d1) - 1.


    def BlackScholes(CallPutFlag, S, X, T, r, d, v):
        return BlackScholesCore(CallPutFlag, np.exp(-r * T), np.exp((r - d) * T) * S, X, T, v)
```

1.1.1 Hedging strategy

```
In [28]: def hedgingStrategy(CallPutFlag, S, X, T, r, d, v, n):
             tt = np.linspace(0, T, n)
             dt = T / (n - 1.)
             expDrift = np.exp((r - 0.5 * v * v) * dt)
             diff = v * np.sqrt(dt)
             SS, bank, strategyS, BSDelta, value, profit = [
             ], [], [], [], [], []
             ##nS = np.random.normal(0., 1., len(tt))

             # initialisation step
             tau = T
             SS.append(S)
             BSValue = BlackScholes(CallPutFlag, S, X, tau, r, d, v)
             BSDelta.append(
                 BlackScholesDelta(CallPutFlag, S, X, tau, r, d, v))
             strategyS.append(BSDelta[-1] * S)
             value.append(BSValue)
             bank.append(BSValue - strategyS[-1])
             profit.append(value[-1] - BSValue)

             # Evolution over time

             for i in arange(1, len(tt)):
                 nS = np.random.normal(0., 1.)
                 tau = T - tt[i]
                 SS.append(SS[-1] * expDrift * np.exp(diff * nS))
                 BSValue = BlackScholes(
                     CallPutFlag, SS[-1], X, tau, r, d, v)
```

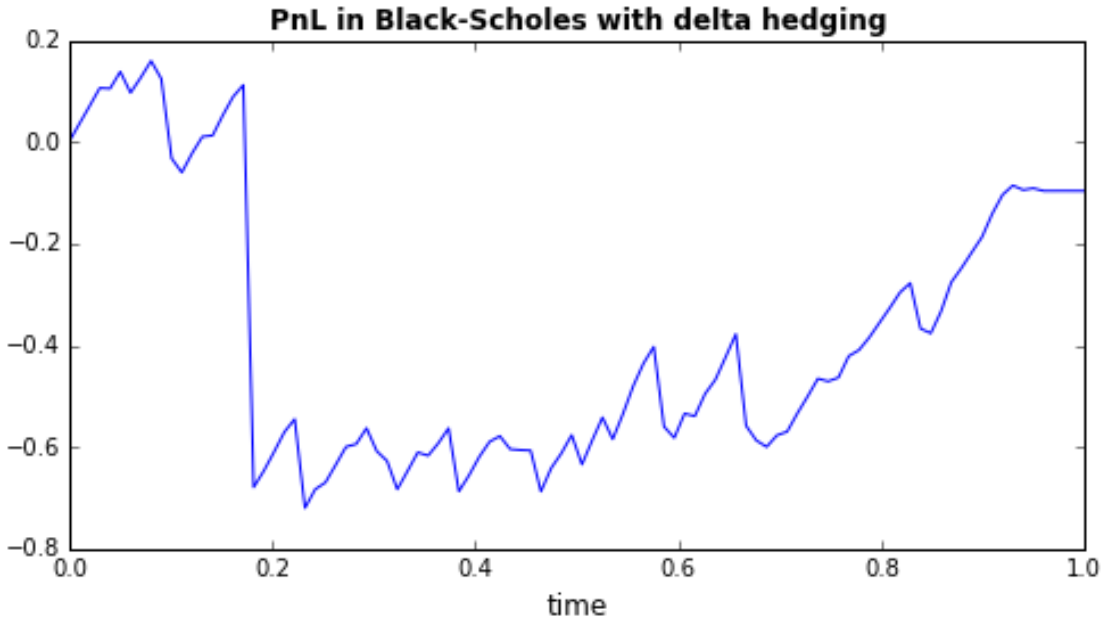
```
            BSDelta.append(
                BlackScholesDelta(CallPutFlag, SS[-1], X, tau, r, d, v))
            strategyS.append(BSDelta[-1] * SS[-1])
            value.append(
                BSDelta[-2] * SS[-1] + np.exp(r * dt) * bank[-1])
            bank.append(value[-1] - strategyS[-1])
            profit.append(value[-1] - BSValue)
        return tt, bank, strategyS, profit
```

1.2 Delta-hedging strategy

1.2.1 One path

In [29]:
```
CallPutFlag, S, X, T, r, d, v, nbSteps = True, 100., 95., 1., 0.03, 0., 0.2, 100
```

In [30]:
```
tt, bank, strategyS, profit = hedgingStrategy(
    CallPutFlag, S, X, T, r, d, v, nbSteps)
fig = plt.figure(figsize=(8, 4))
plt.plot(tt, profit, 'b')
plt.title("PnL in Black-Scholes with delta hedging",
          fontsize=12, fontweight='bold')
plt.xlabel('time', fontsize=12)
plt.show()
```

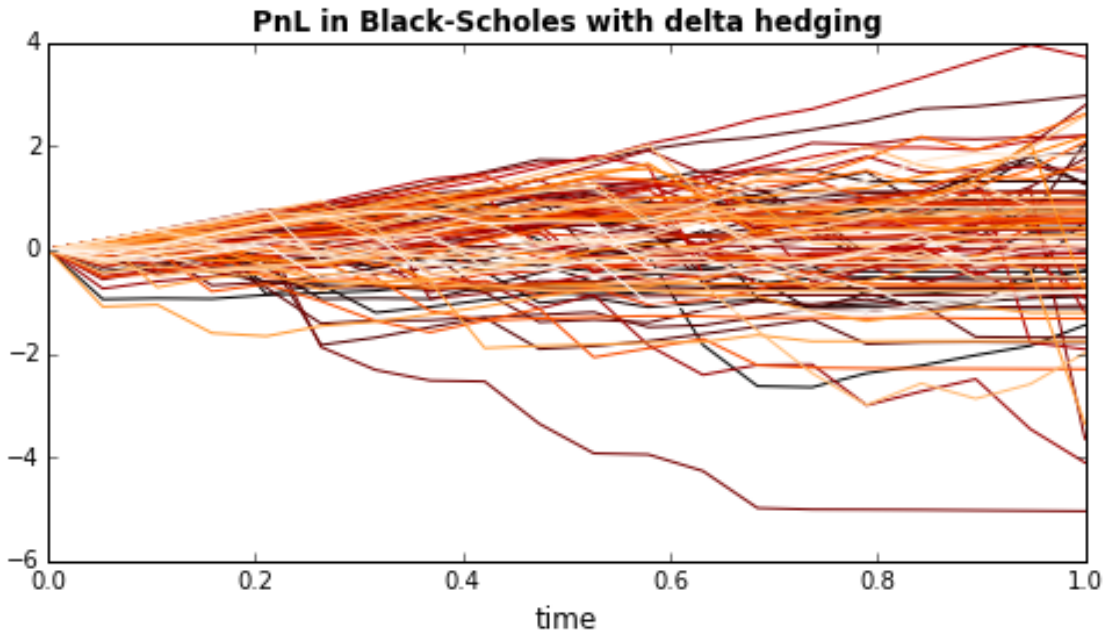


1.2.2 Multiple paths

In [34]:
```
CallPutFlag, S, X, T, r, d, v, nbSteps, nbPaths = True, 100., 95., 1., 0.03, 0., 0.2, 20, 100
```

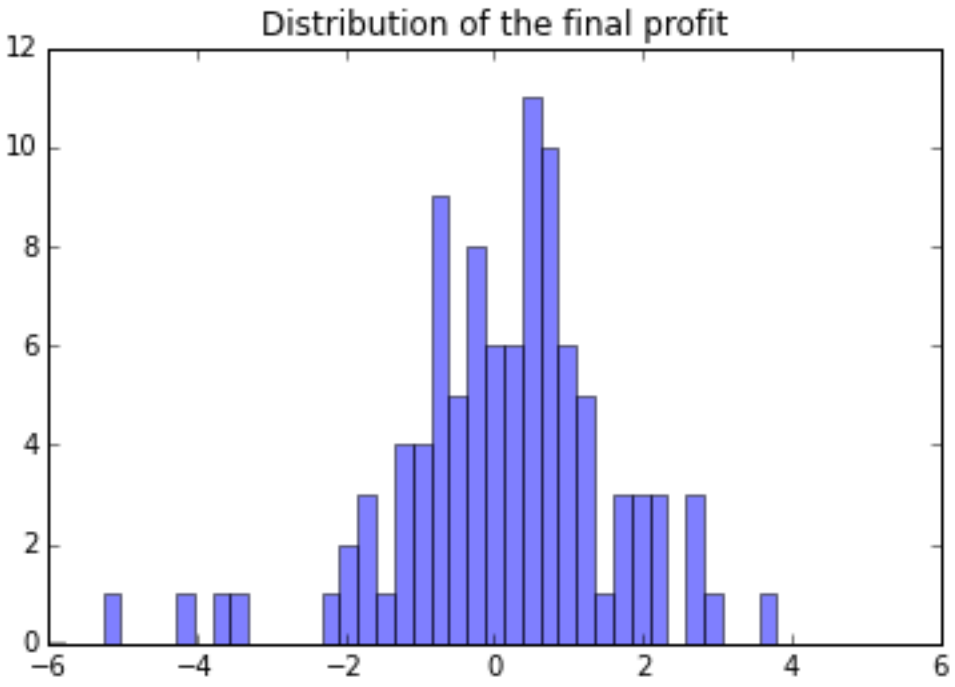
In [37]:
```
ii = arange(nbPaths)
profits, deltas, banks = [], [], []
for i in ii:
    tt, bank, strategyS, prof = hedgingStrategy(
        CallPutFlag, S, X, T, r, d, v, nbSteps)
    profits.append(prof)
    # deltas.append(strategyS)
    # banks.append(bank)

fig = plt.figure(figsize=(8, 4))
for i in ii:
    plt.plot(tt, profits[i], color=get_cmap(
        'gist_heat')(float(i) / len(ii)))
plt.title("PnL in Black-Scholes with delta hedging",
          fontsize=12, fontweight='bold')
plt.xlabel('time', fontsize=12)
plt.show()
```



1.2.2.1 Distribution of the final PnL

In [38]:
```
pp = [profits[i][-1] for i in arange(nbPaths)]
bins = numpy.linspace(-6., 6., 50)
pyplot.hist(pp, bins, alpha=0.5)
pyplot.title("Distribution of the final profit")
pyplot.show()
```



In [33]: