In [1]:
```
import numpy as np
import pandas as pd
from scipy import stats
import math
import pandoc
```

In [2]:
```
mu = 0.168904
sigma_t = 0.2066
T = 0.376
daysToExp = T*250
S0 = 56.47
K = 55
delta = 1/250
```

Black Scholes function is from -- https://github.com/jmiedwards/Python---Black-Scholes-Pricing-calculator-/blob/master/Black-Scholes%20Calculator%20Dividend.py

In [3]:
```
def black_scholes(s, k , t, v, rf = 0.0084):
        """ Price a call using the Black-Scholes model.
        s: initial stock price (56)
        k: strike price(55)
        t: expiration time ( annual- decimal)
        v: volatility ( in decimal30%-0.3)
        rf: risk-free rate (0.0084)
        """

        d1 = (math.log(s/k)+(rf+0.5*math.pow(v,2))*t)/(v*math.sqrt(t))
        d2 = d1 - v*math.sqrt(t)

        optprice = (s*math.exp(-t)*
                    stats.norm.cdf(d1)) - (k*math.exp(-
rf*t)*stats.norm.cdf(d2))
        return optprice
```

The delta of the black

In [4]:
```
def BS_delta (s, k, t, v, rf):
    d1 = (math.log(s/k)+(rf+0.5*math.pow(v,2))*t)/(v*math.sqrt(t))
    delta = stats.norm.cdf(d1)
    return delta
```

In [5]:
```
ht = BS_delta(S0, K, T, sigma_t, 0 )
CallPrice_t = black_scholes(S0, K, T, sigma_t, 0 )
Vt = ht*S0 - CallPrice_t
```

In [6]:
```
Xt_5Delta = pd.Series([-0.6, -0.4, -0.2, 0.2, 0.4, 0.6])
St_5Delta = S0*np.exp(Xt_5Delta)
sigma_5Delta = sigma_t*pd.Series([0.5, 0.75, 1.25, 1.5, 1.75, 2])
Loss_Table = pd.DataFrame({'Loss' : 0.0,
                           'Return' : Xt_5Delta,
                           'Volatility' : sigma_5Delta},
                          index = range(0,len(Xt_5Delta)))
```

In [7]:
```
dataframe_index = 0
for stock_index in range(0,len(St_5Delta)):
    for sigma_index in range(0,len(sigma_5Delta)):
        CallPrice_5Delta = black_scholes(St_5Delta[stock_index]
                                         , K, T, sigma_5Delta[sigma_index], 0
)
        Vt_5Delta = ht*St_5Delta[stock_index] - CallPrice_5Delta
        Lt_5Delta = -(Vt_5Delta - Vt)
        Loss_Table.loc[dataframe_index] = [Lt_5Delta, Xt_5Delta[stock_index],
                                           sigma_5Delta[sigma_index]]
        dataframe_index = dataframe_index+1
```

```
  File "<ipython-input-7-50fe989ce175>", line 4
    CallPrice_5Delta =
                      ^
SyntaxError: invalid syntax
```

In [ ]:
```
Loss_Table.sort_values(by= 'Loss', ascending = False).head()
```

Finding the worst case sceinario risk measure

$$\rho(L_{t+5\Delta}) = max\{l_n|n = 1, \ldots, 36\}$$

In [ ]:
```
Loss_Table.loc[Loss_Table['Loss'].idxmax()]
```

In [ ]:
```
def return_weight_label (row):
    if np.absolute(row['Return']) == 0.6:
        return 0.5
    if np.absolute(row['Return']) == 0.4:
        return 0.75
    if np.absolute(row['Return']) == 0.2:
        return 1
```

In [ ]:
```
def vol_weight_label (row, sigma_t):
    if np.absolute(row['Volatility']) == 0.5*sigma_t:
        return 0.5
    if np.absolute(row['Volatility']) == 0.75*sigma_t:
        return 0.75
    if np.absolute(row['Volatility']) == 1.25*sigma_t:
        return 1
    if np.absolute(row['Volatility']) == 2*sigma_t:
        return 0.5
    if np.absolute(row['Volatility']) == 1.75*sigma_t:
        return 0.75
    if np.absolute(row['Volatility']) == 1.5*sigma_t:
        return 1
```

In [ ]:
```
Loss_Table['ReturnWeights'] = Loss_Table.apply(lambda row:
                                               return_weight_label(row), axis=

Loss_Table['VolWeights'] = Loss_Table.apply(lambda row:
                                            vol_weight_label(row, sigma_t), ax
is=1)
Loss_Table['TotalWeight']=Loss_Table['ReturnWeights']*Loss_Table['VolWeights']
Loss_Table['TWeight*Loss']=Loss_Table['TotalWeight']*Loss_Table['Loss']
```

In [ ]:
```
Loss_Table.head()
```

In [ ]:
```
Loss_Table.loc[Loss_Table['TWeight*Loss'].idxmax()]
```