In [7]:
```
import numpy as np
import pandas as pd
from pandas_datareader import data as wb
from scipy.stats import norm
import matplotlib.pyplot as plt
%matplotlib inline
```

In [29]:
```
#call data of a stock

ticker = 'MSFT'
data = pd.DataFrame()
data[ticker] = wb.DataReader(ticker, data_source='yahoo', start='2016-1-1', end='
2019-7-29')['Adj Close']
```

In [30]:
```
#We will use the Euler Discretization formula
```

$$S_t = S_{t-1} \cdot exp((r - 0.5 \cdot stdev^2) \cdot delta_t + stdev \cdot delta_t^2 \cdot Z_t)$$

In [31]:
```
log_returns = np.log(1 + data.pct_change())
```

In [32]:
```
r = 0.025
```

In [35]:
```
# s = standard deviation
s = log_returns.std()* 250**0.5
s
```

Out[35]:
```
MSFT    0.221814
dtype: float64
```

In [36]:
```
type(s)
```

Out[36]:
```
pandas.core.series.Series
```

In [37]:
```
s = s.values
s
```

Out[37]:
```
array([0.22181383])
```

In [38]:
```
# need to set up number of trading days
T = 1.0
t_intervals = 250
delta_t = T / t_intervals

iterations = 1000
```

In [39]:
```
# create the random conponent Z
Z = np.random.standard_normal((t_intervals + 1, iterations))
S = np.zeros_like(Z)
S0 = data.iloc[-1]
S[0] = S0
```

$$S_t = S_{t-1} \cdot exp((r - 0.5 \cdot stdev^2) \cdot delta_t + stdev \cdot delta_t^2 \cdot Z_t)$$

In [40]:
```
for t in range(1, t_intervals + 1):
    S[t] = S[t-1] * np.exp((r - 0.5 * s ** 2) * delta_t + s * delta_t ** 0.5 * Z
[t])
```

In [41]:
```
S
```

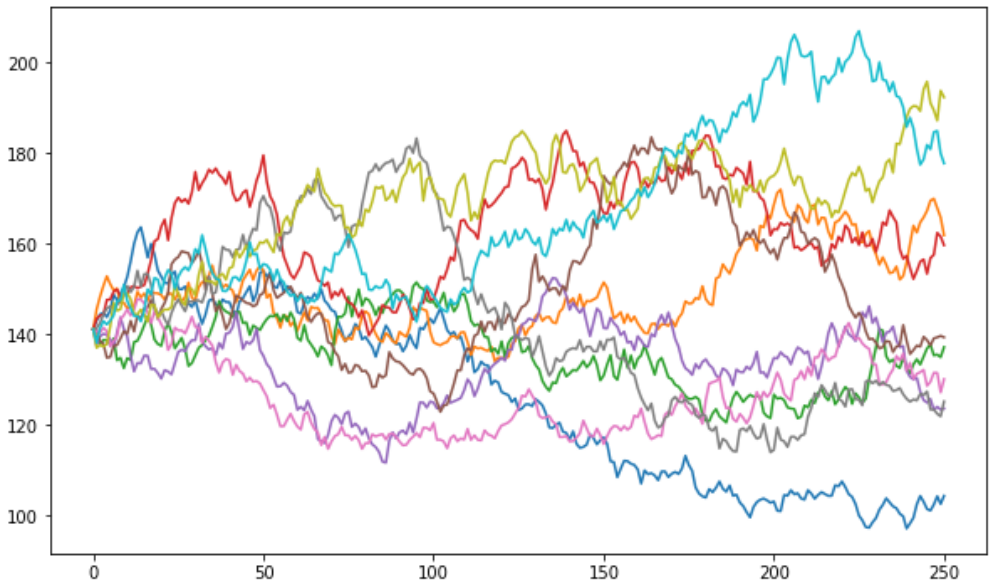
Out[41]:
```
array([[141.02999878, 141.02999878, 141.02999878, ..., 141.02999878,
        141.02999878, 141.02999878],
       [142.97883299, 145.44813519, 138.95029685, ..., 144.9835317 ,
        141.87452558, 141.31241941],
       [144.34321729, 148.26895812, 137.82158793, ..., 144.26659899,
        145.96154259, 145.48744174],
       ...,
       [104.16348308, 167.80097915, 135.47263471, ..., 191.97097438,
        210.54432277, 144.65879632],
       [102.4496181 , 165.79228077, 134.88544001, ..., 191.66750615,
        210.64187254, 145.11645734],
       [104.24794002, 161.82624332, 137.10988594, ..., 195.11724861,
        211.43711755, 147.86587484]])
```

In [43]:
```
S.shape
```

Out[43]:
```
(251, 1000)
```

In [45]:
```
#plot ten iterations
plt.figure(figsize=(10,6))
#plot first 10 iterations
plt.plot(S[:,:10])
```

Out[45]:
```
[<matplotlib.lines.Line2D at 0x1a25c48f98>,
 <matplotlib.lines.Line2D at 0x1a26c6f320>,
 <matplotlib.lines.Line2D at 0x1a26c6f898>,
 <matplotlib.lines.Line2D at 0x1a26c6f048>,
 <matplotlib.lines.Line2D at 0x1a26c6f5f8>,
 <matplotlib.lines.Line2D at 0x1a26c6f710>,
 <matplotlib.lines.Line2D at 0x1a26c6f780>,
 <matplotlib.lines.Line2D at 0x1a26c70da0>,
 <matplotlib.lines.Line2D at 0x1a26c70b00>,
 <matplotlib.lines.Line2D at 0x1a26c70400>]
```



In [56]:
```
# strike price 160
p = np.maximum(S[-1] - 160, 0)
```

In [57]:
```
p
```

Out[57]:
```
array([0.00000000e+00, 1.82624332e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       3.23115792e+01, 1.77263180e+01, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 7.50206024e+00, 0.00000000e+00,
       0.00000000e+00, 1.43165072e+01, 0.00000000e+00, 1.90095482e+01,
       0.00000000e+00, 0.00000000e+00, 1.37741690e+01, 0.00000000e+00,
       1.00010951e+02, 3.52302516e+01, 0.00000000e+00, 0.00000000e+00,
       4.80332550e+01, 4.07280388e+01, 0.00000000e+00, 1.39522246e+01,
       0.00000000e+00, 9.77111868e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 4.20357792e+01, 0.00000000e+00, 0.00000000e+00,
       3.67559793e+01, 0.00000000e+00, 0.00000000e+00, 2.81023519e+01,
       0.00000000e+00, 0.00000000e+00, 2.25823059e+01, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.58865234e+00, 0.00000000e+00,
       2.50498353e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       7.93684803e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       2.22752908e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.10240213e+01, 0.00000000e+00, 2.36331309e+01, 5.16179748e+01,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 6.01873023e+01,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.22268891e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.55353895e+01,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.91940947e+01,
       1.71563881e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.20331853e+00, 5.65795311e+01, 4.63940824e+01, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.04831738e+01, 6.13966183e+00, 0.00000000e+00, 0.00000000e+00,
       2.17950912e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.73398277e+01,
       3.47122302e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 6.24396472e+00, 3.98222945e+01,
       0.00000000e+00, 0.00000000e+00, 1.74461334e+01, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.43557194e+01,
       7.55945606e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       4.43011883e+01, 1.82582410e+00, 0.00000000e+00, 0.00000000e+00,
       5.37957623e+01, 0.00000000e+00, 7.54480802e+01, 4.08858522e+01,
       0.00000000e+00, 0.00000000e+00, 2.81356899e+01, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 2.51700011e+01, 0.00000000e+00,
       1.70299765e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.97844346e+01, 3.20940033e+01,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       3.78892501e+01, 2.28028224e+01, 5.27891829e+00, 1.41444995e+01,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.10196623e+01,
       1.36832226e+00, 1.14401752e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.74393652e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.66085436e+01, 0.00000000e+00, 1.31458948e+01, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.47676978e+01,
       9.26220626e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       2.69326207e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       2.48235096e+01, 0.00000000e+00, 2.46251892e+01, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.16882757e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 4.13207056e+00, 0.00000000e+00, 4.46560194e+01,
       0.00000000e+00, 3.41655605e+00, 3.82575413e+00, 0.00000000e+00,
       0.00000000e+00, 4.34547461e+01, 0.00000000e+00, 2.67572616e+01,
       3.50995339e+00, 0.00000000e+00, 6.50372530e+01, 0.00000000e+00,
       0.00000000e+00, 5.83747893e-01, 0.00000000e+00, 5.47874503e+01,
```

In [54]:
```
p.shape
```

Out[54]:
```
(1000,)
```

Forcast the cost of stock options</h2>

$$C = \frac{exp(-r \cdot T) \cdot \sum p_i}{iterations}$$

In [58]:
```
C = np.exp(-r * T) * np.sum(p) / iterations
C
```

Out[58]:
```
7.949057129735547
```

Run the code multiple times as each time you may get a different result. To verify if your formual is work reffer to the Black Scholes formula as the result should be similar.

When I run a strike price of 160 on the black and scholes formula I get a price of 6.94 and I get a price of 7.94 with this forcast. Average is 7.44

Current ask today July 30 2019 is at 13:03 cental time is 19.50 so this would not be a good buy.

In [ ]:
```

```