Import required packages

In [19]:

```
from portbalance import *
from io import StringIO

## For debugging
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Initialize a portfolio

In [20]:

```
## Load a portfolio
inline_txt = StringIO("""
GOOG, 150, 03/02/2019,1119.92
MSFT, 60, 03/02/2019, 112.53
GM, 90, 03/02/2019, 39.53
""")
my_portfolio = Portfolio(inline_txt)
print(my_portfolio)
```

```
  Ticker  NumShares         Date  BuyPrice  CurrentPrice
0   GOOG        150   03/02/2019   1119.92       1140.99
1   MSFT         60   03/02/2019    112.53        112.53
2     GM         90   03/02/2019     39.53         39.53
```

Calculate historical performance

In [21]:

```
print("Net return on principal:", my_portfolio.calculate_performance())
```

```
Net return on principal: 0.017725991671223573
```

Initialize a target allocation / strategy

In [22]:

```
## Pick a strategy (alternatively, can also write your own as a dict())
strat = {
        'AMZN' : 0.7*0.1765,
        'GOOG' : 0.7*0.1765,
        'FB' : 0.7*0.094,
        'MSFT' : 0.7*0.077,
        'JPM' : 0.7*0.065,
                 }
```

Calculate purchases needed for rebalancing

In [23]:

```
### Run gradient descent for a specified budget
buying_budget = 1000000
my_buys, drift_final, leftover_cash = my_portfolio.find_investing_strategy(buying_b

### Print output
print("Investing strategy:")
for key in my_buys:
    print (key,':',my_buys[key])

print('\nResidual balance: ', leftover_cash)

print('\nResidual drift: ', str(100*drift_final),'%')
```

```
Investing strategy:
JPM : 1838
AMZN : 170
FB : 1330
GOOG : 99
MSFT : 1733

Residual balance:  58.659999910974875

Residual drift:  58.77 %
```

Assumptions

In order to calculate a reasonable strategy, this code makes a few assumtions

No sells.

This code finds the best way to invest additional money into your existing portfolio, but it will not include selling off current assests as part of its strategy

Greedy.

Currently, the algorithm used by the app decides the best stock choice on a buy-by-buy basis.

Allocations.

The "target portfolios" are based entirely on crowdsourced suggestions on public forums like the Bogleheads website. Much better long-term performance likely may be achieved by using a unique portfolio strategy based on professional and individualized research

Risk Assesment

$$\sigma_{portfolio} = \sqrt{w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + 2w_1w_2\rho_{1,2}\sigma_1\sigma_2}$$

Where:

W_1	=	Proportion of the portfolio invested in Asset 1
W_2	=	Proportion of the portfolio invested in Asset 2
σ_1	=	Asset 1 standard deviation of returns
σ_2	=	Asset 2 standard deviation of returns
$\rho_{1,2}$	=	Correlation coefficient between the returns of Asset 1 and Asset 2

In [30]:

```
from pandas_datareader import data as web
import pandas as pd
import numpy as np

assets =  ['GOOG', 'MSFT', 'GM']

df = pd.DataFrame()

for stock in assets:
    df[stock] = web.DataReader(stock, data_source='yahoo',
                               start='2019-1-1' , end='2019-3-3')['Adj Close']

d_returns = df.pct_change()

cov_matrix_d = d_returns.cov()
cov_matrix_a = cov_matrix_d * 250

weights = np.array([0.2, 0.2, 0.2])  # assign equal weights

# calculate the variance and risk of the portfolo
port_variance = np.dot(weights.T, np.dot(cov_matrix_a, weights))
port_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix_a, weights)))

percent_var = str(round(port_variance, 4) * 100) + '%'
percent_vols = str(round(port_volatility, 4) * 100) + '%'

print('Variance of Portfolio is {}, Portfolio Risk is {}'
      .format(percent_var, percent_vols))
```

```
Variance of Portfolio is 1.5699999999999998%, Portfolio Risk is 12.53%
```

$$\begin{aligned}\sigma_p^2 &= \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} \\ \sigma_{2,1} & \sigma_2^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_1\sigma_1^2 + w_2\sigma_{2,1} & w_1\sigma_{1,2} + w_2\sigma_2^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\ &= w_1^2\sigma_1^2 + w_1w_2\sigma_{2,1} + w_1w_2\sigma_{1,2} + w_2^2\sigma_2^2 \\ &= w_1^2\sigma_1^2 + 2w_1w_2\sigma_{1,2} + w_2^2\sigma_2^2\end{aligned}$$

In [ ]: