Asset_Pricing_Assignment_20181016

October 16, 2018

Xicheng XIA

In [1]:
```
# import modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# read data
df = pd.read_excel('Industry_Portfolios.xlsx')
df['Date'] = pd.to_datetime(df['Date'], format = '%Y%m')
df = df.set_index('Date')
```

Create a table showing the mean returns and standard deviation of returns for the ten industry portfolios

In [3]:
```
# calculate the mean and varaince of return
Mean_Variance_Each = pd.concat([df.std()**2, df.mean()], axis = 1)

# create a table
Mean_Variance_Each.columns = columns = ['Varaince(%^2)', 'Mean(%)']

Mean_Variance_Each.T
```

Out[3]:
```
                   NoDur      Durbl      Manuf      Enrgy     HiTec  \
Varaince(%^2)  11.193422  69.920577  28.198970  36.984933  28.95722
Mean(%)         0.902833   0.733333   1.012833   1.231167   0.76625

                   Telcm      Shops       Hlth      Utils      Other
Varaince(%^2)  19.787227  16.759084  14.342669  13.703052  31.163771
Mean(%)         0.881417   0.916333   0.783833   0.907167   0.489083
```

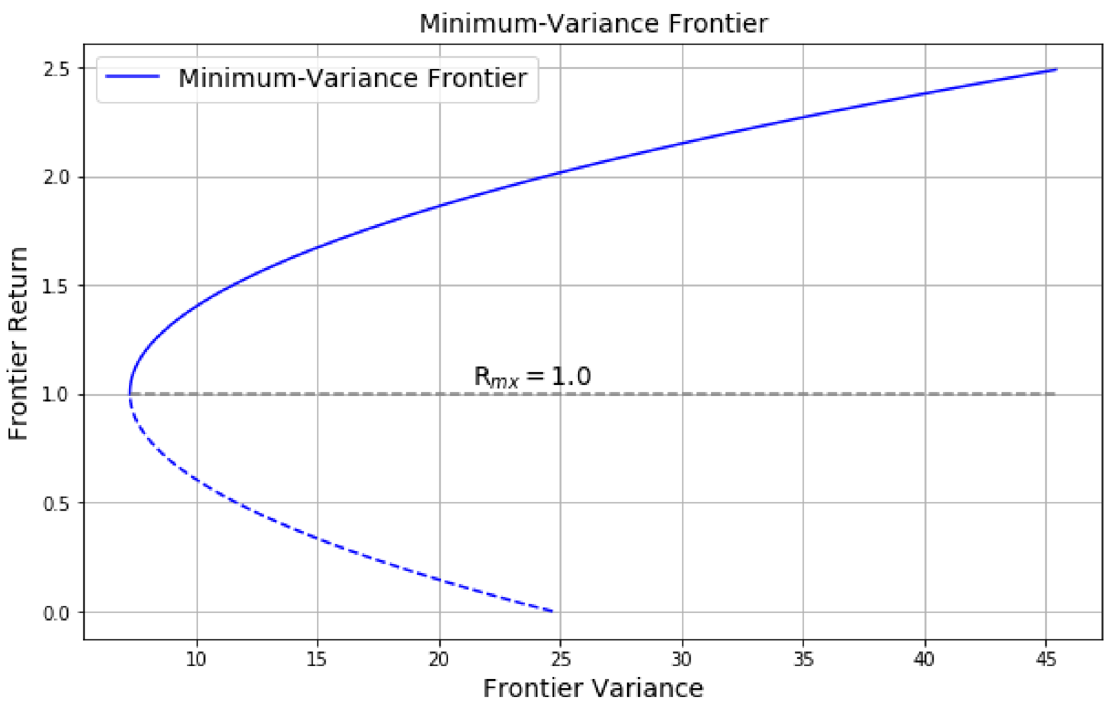
Define the function for calculating the minimum variance of return for each given mean of return.

In [4]:
```
def frontier(V, R, Rp):
    ''' Frontier function
    Calculate the minimum variance of return for a given mean of return.
    Input:
```

```
    --- V:     n by n covariance matrix,
                 numpy.array object or numpy.matrix object;

    --- R:     n by 1 mean return matrix,
                 numpy.array object or numpy.matrix object;

    --- Rp:   Expected return of the frontier portfolio,
                 float type.

    Ouput:
    --- variance: variance of the frontier portfolio,
                        float type.
    --- w: weight of optimal portfolio,
              numpy.array type or numpy.matrix type
    '''
    # reshape return matrix, guarantee the shape of it
    R = R.reshape((max(R.shape), 1))

    assert(V.shape[0] == V.shape[1])
    assert(R.shape[0] == V.shape[0])

    # define e = (1; : : : ; 1)'
    e = np.ones((R.shape[0], 1))

    # calculate alpha, zeta, and sigma
    alpha = np.dot(np.dot(R.T, np.linalg.inv(V)), e)
    zeta = np.dot(np.dot(R.T, np.linalg.inv(V)), R)
    sigma = np.dot(np.dot(e.T, np.linalg.inv(V)), e)

    # variance of return for frontier portfolio
    variance = 1/sigma +sigma/(zeta*sigma - alpha**2)*(Rp - alpha/sigma)**2

    # weight of frontier portfolio
    w = ((sigma*Rp - alpha)/(zeta*sigma - alpha**2))*np.dot(np.linalg.inv(V), R) \
    + ((zeta - alpha*Rp)/(zeta*sigma - alpha**2))*np.dot(np.linalg.inv(V), e)

    return float(variance), w
```

```
In [5]: V = df.cov().values # covariance matrix
        R = df.mean().values # mean return matrix

        # list of given mean of return
        Return_lst = np.arange(0, 2.5, 0.01)

        # list of varaince of return
        Variance_lst = []
        for Rp in Return_lst:
            Variance_lst.append(frontier(V, R, Rp)[0])
```

```
In [6]: z = 2
        fig = plt.figure(figsize = (10, 6))
        ind = Variance_lst.index(min(Variance_lst))

        plt.plot(Variance_lst[ind:], Return_lst[ind:],  color = 'blue', label = 'Minimum-Variar
        plt.plot(Variance_lst[:ind+1], Return_lst[:ind+1],  '--', color = 'blue')
        plt.plot([min(Variance_lst), max(Variance_lst)], [Return_lst[ind]]*2, '--', color = 'g
        plt.text(Variance_lst[10], Return_lst[ind], 'R$%s$'%'_{mx} = '\
                 + str(round(Return_lst[ind], 4)), verticalalignment = 'bottom', fontsize = 7*z

        plt.xlabel('Frontier Variance', fontsize = 7*z)
        plt.ylabel('Frontier Return', fontsize = 7*z)
        plt.title('Minimum-Variance Frontier', fontsize = 7*z)
        plt.legend(loc = 'best', fontsize = 7*z)
        plt.grid(True)
        plt.show()
```



A combinition of efficient frontier and utility line can give you the optimal assets allocation weights. People with different risk prefference may have difference optimal portfolio lying on the frontier.

Any portfolio lying under the efficient frontier is sub-optimal. It doesn't provide as much return as the frontier portfolio while suffering from the same level of risk.

Define the function for calculating the minimum variance of return for a given mean of return and a risk free asset.

```
In [7]: def frontier_rf(V, R, Rp, Rf):
```

```
    ''' Frontier function with risk free asset.
    Calculate the minimum variance of return for a given mean of return and a risk fre

    Input:
    --- V:    n by n covariance matrix,
                 numpy.array object or numpy.matrix object;

    --- R:    n by 1 mean return matix,
                  numpy.array object or numpy.matrix object;

    --- Rp: Expected return of the frontier portfolio,
                 float;

    --- Rf:  Risk free rate,
                  float.

    Ouput:
    --- variance: variance of the frontier portfolio,
                          float type.
    '''
    # reshape return matrix, guarantee the shape of it
    R = R.reshape((max(R.shape), 1))

    assert(V.shape[0] == V.shape[1])
    assert(R.shape[0] == V.shape[0])

    # define e = (1; : : : ; 1)'
    e = np.ones((R.shape[0], 1))

    # calculate alpha, zeta, and sigma
    alpha = np.dot(np.dot(R.T, np.linalg.inv(V)), e)
    zeta = np.dot(np.dot(R.T, np.linalg.inv(V)), R)
    sigma = np.dot(np.dot(e.T, np.linalg.inv(V)), e)

    # variance of return for frontier portfolio
    variance = (Rp - Rf)**2/(zeta - 2*alpha*Rf +sigma*Rf**2)
    return float(variance)
```

Define the function for calculating mean and variance of return for tangency portfolio.

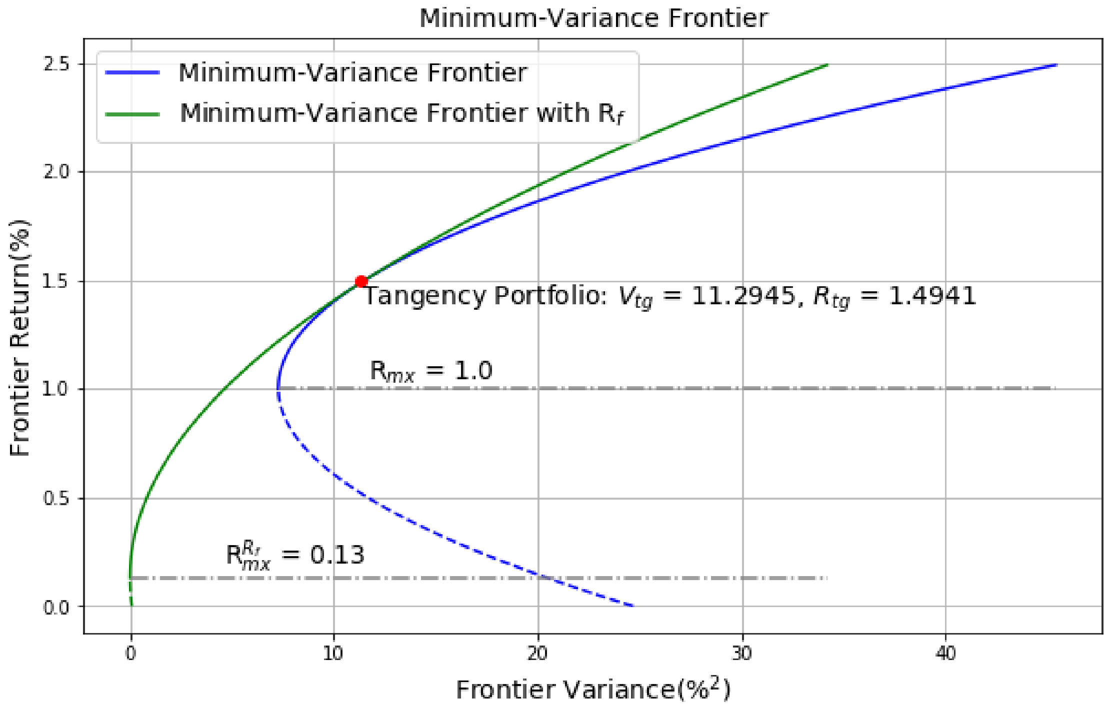
```
In [8]: def tangency_portfolio(V, R, Rf):
    ''' Tangency portfolio function.
    Calculate mean and variance of return for tangency portfolio.

    Input:
    --- V:  n by n covariance matrix,
               numpy.array object or numpy.matrix object;
```

```
        --- R:  n by 1 mean return matrix,
                   numpy.array object or numpy.matrix object;

        --- Rf: Risk free rate,
                    float.

        Ouput:
        --- Rtg: return of the tangency portfolio,
                     float type;

        --- Vtg: variance of the tangency portfolio,
                      float type.
        '''
        # reshape return matrix, guarantee the shape of it
        R = R.reshape((max(R.shape), 1))

        assert(V.shape[0] == V.shape[1])
        assert(R.shape[0] == V.shape[0])

        # define e = (1; : : : ; 1)'
        e = np.ones((R.shape[0], 1))

        # calculate alpha, zeta, and sigma
        alpha = np.dot(np.dot(R.T, np.linalg.inv(V)), e)
        zeta = np.dot(np.dot(R.T, np.linalg.inv(V)), R)
        sigma = np.dot(np.dot(e.T, np.linalg.inv(V)), e)

        # variance of return for frontier portfolio
        Rtg = (alpha*Rf - zeta)/(alpha*Rf - sigma)
        Vtg = 1/sigma*(1 + (zeta*sigma - alpha**2)/(sigma*Rf - alpha)**2)
        return float(Rtg), float(Vtg)
```

```
In [9]: V = df.cov().values # covariance matrix
        R = df.mean().values # mean return matrix
        Rf = 0.13

        # calculate mean and variance of return for tangency portfolio.
        Rtg, Vtg = tangency_portfolio(V, R, Rf)

        # list of given mean of return
        Return_rf_lst = np.arange(0, 2.5, 0.01)

        #calculating the minimum variance of return for a given mean of return and a risk free
        Variance_rf_lst = []
        for Rp in Return_rf_lst:
            Variance_rf_lst.append(frontier_rf(V, R, Rp, Rf))
```

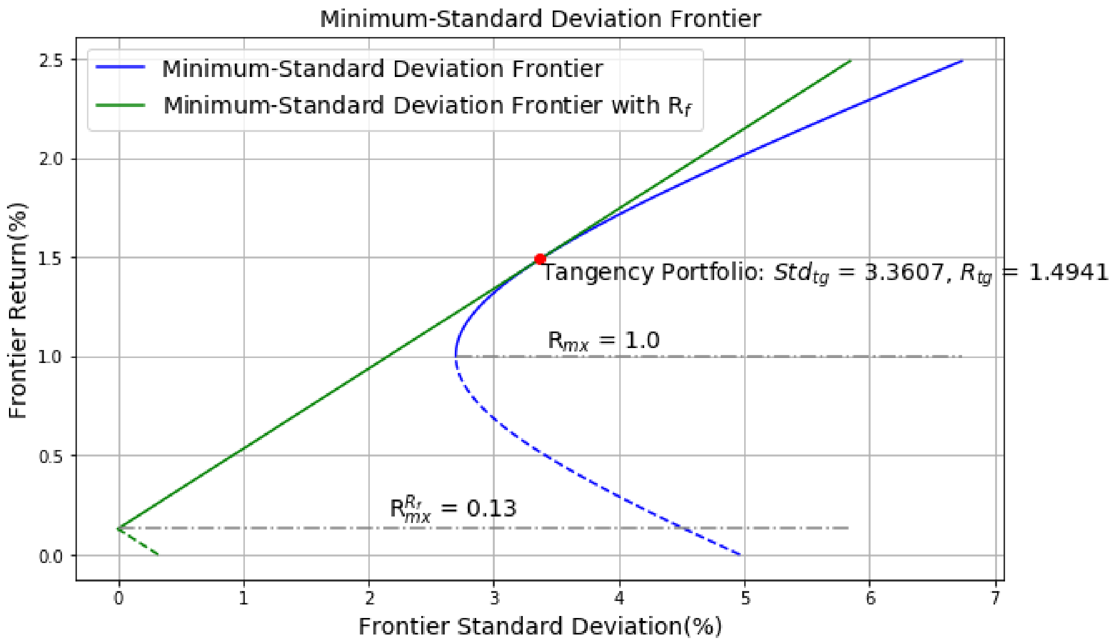
```
In [10]: z = 2
         fig = plt.figure(figsize = (10, 6))
```

```
ind = Variance_lst.index(min(Variance_lst))
ind_rf = Variance_rf_lst.index(min(Variance_rf_lst))

plt.plot(Variance_lst[ind:], Return_lst[ind:],  color = 'blue',
         label = 'Minimum-Variance Frontier')
plt.plot(Variance_lst[:ind+1], Return_lst[:ind+1],  '--', color = 'blue')

plt.plot(Variance_rf_lst[ind_rf:], Return_rf_lst[ind_rf:],  color = 'green',
         label = 'Minimum-Variance Frontier with R$%s$'%'_f')
plt.plot(Variance_rf_lst[:ind_rf+1], Return_rf_lst[:ind_rf+1],  '--', color = 'green'

plt.plot([min(Variance_lst), max(Variance_lst)], [Return_lst[ind]]*2, '-.', color = '
plt.text(Variance_lst[50], Return_lst[ind], 'R$%s$ = '%'_{mx}'+str(Return_lst[ind]),
         verticalalignment='bottom', fontsize = 7*z)

plt.plot([min(Variance_rf_lst), max(Variance_rf_lst)], [Return_rf_lst[ind_rf]]*2, '-.
plt.text(Variance_rf_lst[100], Return_rf_lst[ind_rf], 'R$%s$ = '%'_{mx}^{R_f}'\
         +str(Return_rf_lst[ind_rf]), verticalalignment='bottom', fontsize = 7*z)

plt.plot(Vtg, Rtg, 'o', color = 'red', lw = 2)
plt.text(Vtg, Rtg, 'Tangency Portfolio: $%s$ = '%'V_{tg}'+str(round(Vtg,4)) \
         + ', $%s$ = '%'R_{tg}'+str(round(Rtg, 4)), verticalalignment='top', fontsize

plt.xlabel('Frontier Variance($%s$)'%'\%^2', fontsize = 7*z)
plt.ylabel('Frontier Return(%)', fontsize = 7*z)
plt.title('Minimum-Variance Frontier', fontsize = 7*z)
plt.legend(loc = 'upper left', fontsize = 7*z)
plt.grid(True)
plt.show()
```




```
In [11]: Std_lst = [np.sqrt(V) for V in Variance_lst]
         Std_rf_lst = [np.sqrt(V) for V in Variance_rf_lst]
         z = 2
         fig = plt.figure(figsize = (10, 6))

         ind = Std_lst.index(min(Std_lst))
         ind_rf = Std_rf_lst.index(min(Std_rf_lst))

         plt.plot(Std_lst[ind:], Return_lst[ind:],  color = 'blue',
                  label = 'Minimum-Standard Deviation Frontier')
         plt.plot(Std_lst[:ind+1], Return_lst[:ind+1],  '--', color = 'blue')

         plt.plot(Std_rf_lst[ind_rf:], Return_rf_lst[ind_rf:],  color = 'green',
                  label = 'Minimum-Standard Deviation Frontier with R$%s$'%'_f')
         plt.plot(Std_rf_lst[:ind_rf+1], Return_rf_lst[:ind_rf+1],  '--', color = 'green')

         plt.plot([min(Std_lst), max(Std_lst)], [Return_lst[ind]]*2, '-.', color = 'gray')
         plt.text(Std_lst[50], Return_lst[ind], 'R$%s$ = '%'_{mx}'+str(Return_lst[ind]),
                  verticalalignment='bottom', fontsize = 7*z)

         plt.plot([min(Std_rf_lst), max(Std_rf_lst)], [Return_rf_lst[ind_rf]]*2, '-.', color =
         plt.text(Std_rf_lst[100], Return_rf_lst[ind_rf], 'R$%s$ = '%'_{mx}^{R_f}'\
                  +str(Return_rf_lst[ind_rf]), verticalalignment='bottom', fontsize = 7*z)
```

```
plt.plot(np.sqrt(Vtg), Rtg, 'o', color = 'red', lw = 2)
plt.text(np.sqrt(Vtg), Rtg, 'Tangency Portfolio: $%s$ = '%'Std_{tg}'+str(round(np.sqr
        + ', $%s$ = '%'R_{tg}'+str(round(Rtg, 4)), verticalalignment='top', fontsize

plt.xlabel('Frontier Standard Deviation(%)', fontsize = 7*z)
plt.ylabel('Frontier Return(%)', fontsize = 7*z)
plt.title('Minimum-Standard Deviation Frontier', fontsize = 7*z)
plt.legend(loc = 'upper left', fontsize = 7*z)
plt.grid(True)
plt.show()
```



In [12]:
```
# calculate the weight for tangency portfolio.
weight_of_risky = pd.DataFrame(frontier(V, R, Rtg)[1]).T

weight_of_risky.columns = df.columns

weight_of_risky
```

Out[12]:
```
      NoDur     Durbl     Manuf   Enrgy     HiTec     Telcm     Shops  \
0  0.568368 -0.215766  0.728121  0.1047 -0.368132 -0.094593  0.998165

       Hlth     Utils     Other
0  0.074479  0.129684 -0.925026
```

On the Minimum-Standard Deviation Frontier, relative weights among risky assets remain the same, which means they simply increase or decrease at the same level. So, this curve means with risk free assets available, everyone has the same relative prefference among different risky assets.

With risk free asset available, mean of return for all optimal portfolio can be written as:

$$R_p = \omega R_{tg} + (1 - \omega) R_f$$

And standard deviation of return can be written as:

$$\sigma_p = \omega \sigma_{tg}$$