```
%load_ext autoreload
%autoreload 2
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')
```

```
import matplotlib.pyplot as plt
#%matplotlib plt.rcParams['figure.figsize'] = (12.0, 6.0)
#%matplotlib plt.rc('figure', figsize=(20.0, 10.0))

import os, sys

DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/data'
print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")

MODULEPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/nb'
print(f"MODULEPATH:{MODULEPATH} contents:{os.listdir(MODULEPATH)}")

sys.path.append(MODULEPATH)
print(f"sys.path:{sys.path}")

import numpy as np
import pandas as pd

import edhec_risk_kit_111_BBI as erk
```

Lack of Robustness of the Markowitz procedure and the GM

Although the promise of the Markowitz procedure is exciting, in tends to fall apart in practice. The
Expected Returns and Expected Covariance in advance. Our estimates almost certainly contain so
that the procedure is highly sensitive to these errors, which tend to get exaggerated in the portfolio

To see this, let's start by loading up our data as usual.

```
# %load_ext autoreload
# %autoreload 2
# %matplotlib inline
# import edhec_risk_kit_111_BBI as erk

ind = erk.get_ind_returns(DATAPATH)
er = erk.annualize_rets(ind["1996":"2000"], 12)
cov = ind["1996":"2000"].cov()
cov
```

Let's look at a simple 2-asset portfolio and find the optimal weights if we had known what the retu

```
l = ["Food", "Steel"]
import numpy as np
erk.msr(0.1, np.array(er[l]), cov.loc[l,l])
```

Let's look at the returns of the two assets that dictated those weights

```
er[l]
```

Now assume that we had a really good estimator, and we were off by only a fraction of a percent in
estimated a return of 11 and 12 percent respectively for Food and Steel

```
erk.msr(0.1, np.array([.11, .12]), cov.loc[l,l])
```

We see that even a small change in the estimate causes a major change in the weights. What if we
percent in each estimate and estimated 10% and 13% instead of the return of 11.6% and 11.5%?

```
erk.msr(0.1, np.array([.10, .13]), cov.loc[l,l])
```

And if we had made the *same* estimation error, but the error went the other way (13% and 10%)?

```
erk.msr(0.1, np.array([.13, .10]), cov.loc[l,l])
```

Avoiding estimating returns

Let's look at the efficient frontier one more time, and plot the efficient frontier again.

```
import edhec_risk_kit_111_BBI as erk
erk.plot_ef(20, er, cov, show_cml=True, riskfree_rate=0.1)
```

One way to avoid this estimation game is to skip the entire process and just rely on *naive* diversific
with equal weight. We can add the EW portfolio to the plot by enhancing the `plot_ef` function as f

```
    if show_ew:
        n = er.shape[0]
        w_ew = np.repeat(1/n, n)
        r_ew = portfolio_return(w_ew, er)
        vol_ew = portfolio_vol(w_ew, cov)
        # add EW
        ax.plot([vol_ew], [r_ew], color='goldenrod', marker='o', markersize=10)
```

```
1 erk.plot_ef(20, er, cov, show_cml=True, riskfree_rate=0.1, show_ew=True)
```

Researchers have shown that the EW portfolio is a remarkably good portfolio to hold. In fact, there
idea that it is a far better portfolio to hold than a cap-weighted equivalent. We'll examine this in late

EW portfolio is far inside the efficient frontier, but it requires no estimation whatsoever.

However, there is another point on the efficient frontier that is very interesting. This is the *nose* of t
lowest volatility across all possible portfolios. This is called the Minimum Volatility or the Global M

But how do we find the weights of the GMV portfolio?

The interesting thing about it is that if you assume that all returns are the same, the optimizer cann
raising returns, and so it must do so my lowering volatility. This means that if we just skip any retur
returns have the return, we'd get the weights of the GMV portfolio!

```
def gmv(cov):
    """
    Returns the weights of the Global Minimum Volatility portfolio
    given a covariance matrix
    """
    n = cov.shape[0]
    return msr(0, np.repeat(1, n), cov)
```

and we can add that to the plot as follows:

```
    if show_gmv:
        w_gmv = gmv(cov)
        r_gmv = portfolio_return(w_gmv, er)
        vol_gmv = portfolio_vol(w_gmv, cov)
        # add EW
        ax.plot([vol_gmv], [r_gmv], color='midnightblue', marker='o', markersize=10)
```

```
1 erk.plot_ef(20, er, cov, show_cml=True, riskfree_rate=0.1, show_ew=True, show_gm
```

DATAPATH:/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/data cont

```
MODULEPATH:/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/nb cont
sys.path:['', '/env/python', '/usr/lib/python36.zip', '/usr/lib/python3.6', '/
```

```
1
```

```
%load_ext autoreload
%autoreload 2
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')
```

```
import matplotlib.pyplot as plt
#%matplotlib plt.rcParams['figure.figsize'] = (12.0, 6.0)
#%matplotlib plt.rc('figure', figsize=(20.0, 10.0))
```

```
4
5 import os, sys
6
7 DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio
8 print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")
9
10 MODULEPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfol
11 print(f"MODULEPATH:{MODULEPATH} contents:{os.listdir(MODULEPATH)}")
12
13 sys.path.append(MODULEPATH)
14 print(f"sys.path:{sys.path}")
15
16 import numpy as np
17 import pandas as pd
18
19 import edhec_risk_kit_111_BBI as erk
```

Lack of Robustness of the Markowitz procedure and the GMV portfolio

Although the promise of the Markowitz procedure is exciting, in tends to fall apart in practice. The problem is that we rarely know Expected Returns and Expected Covariance in advance. Our estimates almost certainly contain some estimation error, and we'll see that the procedure is highly sensitive to these errors, which tend to get exaggerated in the portfolio.

To see this, let's start by loading up our data as usual.

```
1 # %load_ext autoreload
2 # %autoreload 2
3 # %matplotlib inline
4 # import edhec_risk_kit_111_BBI as erk
5
6 ind = erk.get_ind_returns(DATAPATH)
7 er = erk.annualize_rets(ind["1996":"2000"], 12)
8 cov = ind["1996":"2000"].cov()
```

```
cov
```

Let's look at a simple 2-asset portfolio and find the optimal weights if we had known what the returns would be.

```
l = ["Food", "Steel"]
import numpy as np
erk.msr(0.1, np.array(er[l]), cov.loc[l,l])
```

Let's look at the returns of the two assets that dictated those weights

```
er[l]
```

Now assume that we had a really good estimator, and we were off by only a fraction of a percent in our estimate, and we had estimated a return of 11 and 12 percent respectively for Food and Steel

```
erk.msr(0.1, np.array([.11, .12]), cov.loc[l,l])
```

We see that even a small change in the estimate causes a major change in the weights. What if we were off by around 1% to 2% percent in each estimate and estimated 10% and 13% instead of the return of 11.6% and 11.5%?

```
erk.msr(0.1, np.array([.10, .13]), cov.loc[l,l])
```

And if we had made the *same* estimation error, but the error went the other way (13% and 10%)?

```
erk.msr(0.1, np.array([.13, .10]), cov.loc[l,l])
```

Avoiding estimating returns

Let's look at the efficient frontier one more time, and plot the efficient frontier again.

```
import edhec_risk_kit_111_BBI as erk
erk.plot_ef(20, er, cov, show_cml=True, riskfree_rate=0.1)
```

One way to avoid this estimation game is to skip the entire process and just rely on *naive* diversification, which means hold all stocks with equal weight. We can add the EW portfolio to

the plot by enhancing the `plot_ef` function as follows:

```
    if show_ew:
        n = er.shape[0]
        w_ew = np.repeat(1/n, n)
        r_ew = portfolio_return(w_ew, er)
        vol_ew = portfolio_vol(w_ew, cov)
        # add EW
        ax.plot([vol_ew], [r_ew], color='goldenrod', marker='o', markersize=10)
```

```
1 erk.plot_ef(20, er, cov, show_cml=True, riskfree_rate=0.1, show_ew=True
```

Researchers have shown that the EW portfolio is a remarkably good portfolio to hold. In fact, there is overwhelming siupport for the idea that it is a far better portfolio to hold than a cap-

weighted equivalent. We'll examine this in later sections, but as you can see, the EW portfolio is far inside the efficient frontier, but it requires no estimation whatsoever.

However, there is another point on the efficient frontier that is very interesting. This is the *nose* of the hull, which is the portfolio of lowest volatility across all possible portfolios. This is called the Minimum Volatility or the Global Minimum Volatility or GMV portfolio.

But how do we find the weights of the GMV portfolio?

The interesting thing about it is that if you assume that all returns are the same, the optimizer cannot improve the sharpe ratio through raising returns, and so it must do so my lowering volatility. This means that if we just skip any returns estimation and assume all returns have the return, we'd get the weights of the GMV portfolio!

```
def gmv(cov):
    """
    Returns the weights of the Global Minimum Volatility portfolio
    given a covariance matrix
    """
    n = cov.shape[0]
    return msr(0, np.repeat(1, n), cov)
```

and we can add that to the plot as follows:

```
    if show_gmv:
        w_gmv = gmv(cov)
        r_gmv = portfolio_return(w_gmv, er)
        vol_gmv = portfolio_vol(w_gmv, cov)
        # add EW
        ax.plot([vol_gmv], [r_gmv], color='midnightblue', marker='o', markersize=10)
```

```
1 erk.plot_ef(20, er, cov, show_cml=True, riskfree_rate=0.1, show_ew=True
```

```
1
```