

Python for Finance

Portfolio theory, Efficient frontier
& PCA analysis

Andras Niedermayer



Outline

① Portfolio optimization and the efficient frontier

② Principal Component Analysis

Preliminaries

- Load daily closing prices on five stocks: Apple, Microsoft, Yahoo, Deutsche Bank, and Goldman Sachs.

```
1 data=DataFrame.from_csv('../L7_Data.csv')
```

- Compute the (log) returns from prices.

```
1 returns=np.log(data/data.shift(1))
```

- Get the number of assets as a variable.

```
1 no_assets=len(returns.columns.tolist())
```

Monte Carlo portfolios

Goal

Plot various portfolios of the five stocks in the *expected return – volatility* space. Let us simulate $N = 1000$ portfolios.

- Create (empty) vectors for returns and volatilities.

```
1 MC_returns = []
2 MC_vols = []
3 N = 1000
```

- In a loop, generate portfolio weights and make sure they add up to 1 (one).

```
1 for p in range(N):
2     weights = np.random.rand(no_assets)
3     weights /= np.sum(weights)
```

Monte Carlo portfolios (c'td)

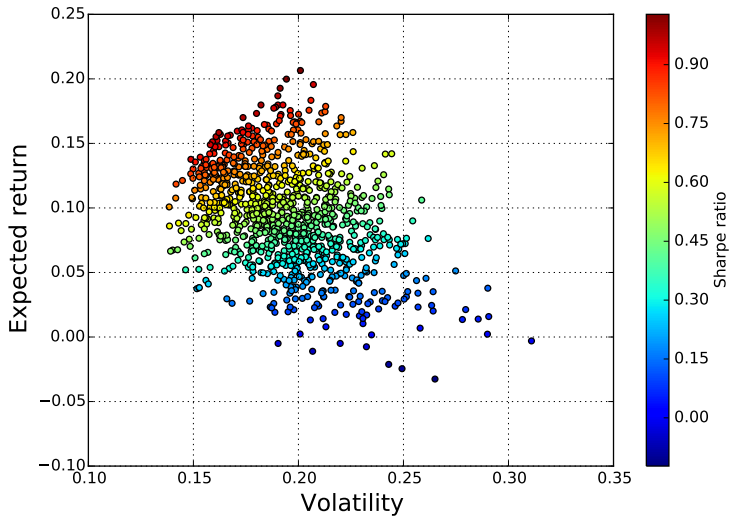
Expected returns and volatility of portfolio i

$$\mathbb{E}R_i = w_{i,j}^T \times \mathbb{E}R_j$$

$$\sigma_i^2 = w_{i,j}^T \times \text{cov}(R_j) \times w_{i,j}$$

- For each random weight vector we picked, we compute the expected return and volatility of the corresponding portfolio (annualised).

```
1 MC_returns.append
2   (np.sum(returns.mean()*weights)*252)
3 MC_vols.append
4   (np.sqrt(np.dot(weights.T,
5     np.dot(returns.cov()*252, weights))))
```



Portfolio summary measures

Objective

Build a function that, for a given vector of portfolio weights w , it returns the portfolio's expected return, volatility, and the Sharpe ratio (as a vector).

```
1 def portfolio(weights):
2     weights=np.array(weights)
3     P_ret=np.sum(returns.mean()*weights)*252
4     P_vol=np.sqrt(np.dot(weights.T,
5         np.dot(returns.cov()*252, weights)))
6     return np.array([P_ret,P_vol, P_ret/P_vol])
```

Maximum Sharpe ratio portfolio

- Define a function for the *negative* Sharpe ratio (we want to maximise it).

```
1 def Sharpe(weights):  
2     return -portfolio(weights)[2]
```

- Set up the constraint that portfolio weights add up to one.

```
1 cons = ({ 'type': 'eq',  
2     'fun': lambda x: np.sum(x) - 1})
```

- Set up boundaries for the portfolio weights (between 0 and 1).

```
1 bnds = tuple((0, 1) for x in range(no_assets))
```

Maximum Sharpe ratio portfolio (c'td)

- Optimisation function.

```
1 import scipy.optimize as sco
2 opt_S=sco.minimize
3   (Sharpe, no_assets*[1.0/no_assets],
4    method='SLSQP', bounds=bnds,
5    constraints=cons)
```

- Print portfolio weights.

```
1 print(opt_S['x'].round(3))
```

- Maximum Sharpe ratio portfolio properties.

```
1 portfolio(opt_S['x'])
```

Application: Minimum variance portfolio

- Define a function for the portfolio variance.

```
1 def Variance(weights):  
2     return portfolio(weights)[1]**2
```

- Set up the constraint that portfolio weights add up to one.

```
1 cons=({'type':'eq',  
2       'fun':lambda x: np.sum(x)-1})
```

- Set up boundaries for the portfolio weights (between 0 and 1).

```
1 bnds=tuple((0,1) for x in range(no_assets))
```

Application: Minimum variance portfolio (c'td)

- Optimisation function.

```
1 import scipy.optimize as sco
2 opt_V=sco.minimize
3   (Variance, no_assets*[1.0/no_assets],
4   method='SLSQP', bounds=bnds,
5   constraints=cons)
```

- Print portfolio weights.

```
1 print(opt_V['x']).round(3)
```

- Maximum Sharpe ratio portfolio properties.

```
1 portfolio(opt_V['x'])
```

Plotting the efficient frontier

Problem

Minimise variance of a portfolio

$$\min_{w_i} w_i^T \Sigma w_i \quad (1)$$

subject to a target return:

$$w_i^T R = R_{\text{target}} \quad (2)$$

- We solve this problem for many levels of the target return.
- Each time we do, we get another point on the efficient frontier.
- We need to specify the constraint in a loop, as the target return is always changing.

Algorithm

- Define a range for target returns.

```
1 TargetRet=np.linspace(0.0,0.25,50)
```

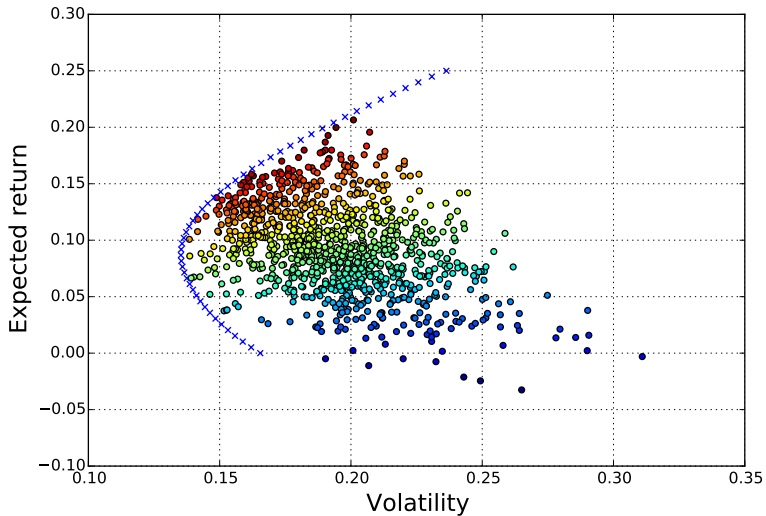
- Define an (empty) vector for the corresponding minimum volatilities.

```
1 MinVols=[]
```

- In a loop of target returns, minimise variance under the constraint that the portfolio return equals the target return.

Algorithm (c'td)

```
1 for tret in TargetRet:
2     cons=({'type':'eq',
3         'fun':lambda x: portfolio(x)[0]-tret},
4         {'type':'eq', 'fun':lambda x: np.sum(x)-1})
5     res=sco.minimize(lambda x: portfolio(x)[1],
6         no_assets*[1.0/no_assets], method='SLSQP',
7         bounds=bnds, constraints=cons)
8
9     MinVols.append(res['fun'])
```



Plotting the Capital Market Line

- 1 Let the risk-free rate, $r_f = 1\%$.
- 2 The CML is a straight line starting from the risk-free rate and tangent to the upper part of the efficient frontier.
- 3 We do not have a closed form for the efficient frontier, so we need to use interpolation.

Efficient frontier (non-dominated)

The trick is to remember that the return vector is sorted in ascending order.

Portfolios with returns above the minimum variance portfolio are non-dominated!

```
1 # minimum variance portfolio index
2 ind=np.argmax(MinVols)
3 evols=MinVols[ind:]
4 erets=TargetRet[ind:]
```


Main idea

- 1 Let $t(x)$ be the capital market line (linear).
- 2 Let $f(x)$ be the efficient frontier function.
- 3 Let x^* be the tangency portfolio.
- 4 At the tangency portfolio, the CML and efficient frontier have equal values and first derivatives.
- 5 This gives a three-equation system with three unknowns.

$$t(x^*) = a + b \times x^* \text{ (Linearity of CML)}$$

$$t(x^*) = f(x^*)$$

$$t'(x^*) = f'(x^*)$$

Numerical approximation of Efficient Frontier

We create numerical approximation via interpolation of

- 1 the efficient frontier
- 2 its first derivative

```
1 import scipy.interpolate as sci
2 tck=sci.splrep(evol,erets)
3 def f(x):
4     return sci.splev(x,tck,der=0)
5 def df(x):
6     return sci.splev(x,tck,der=1)
```

System of equations

- 1 We define a function to solve the system of equations.
- 2 The input is a vector containing
 - the risk-free rate (CML intercept)
 - the maximum Sharpe ratio (CML slope)
 - the tangency portfolio variance.
- 3 The output is the value of the three equations in the system. For the tangency portfolio, they should all be zero!

```
1 def equations(p, rf=0.01):
2     eq1=rf-p[0]
3     eq2=rf+p[1]*p[2]-f(p[2])
4     eq3=p[1]-df(p[2])
5     return eq1, eq2, eq3
```

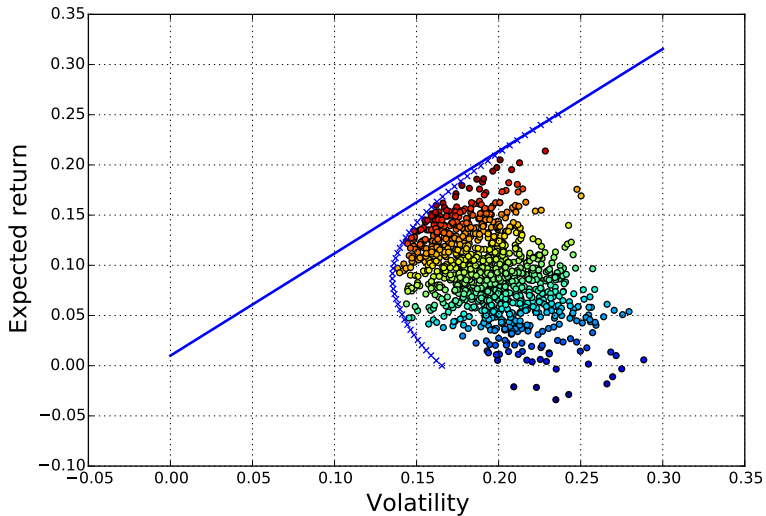
Solve for the CML

- ① Find some plausible starting values for the solver (eye-balling, intuition...)
- ② Use the `fsolve` function to find the CML.

```
1 opt=sco.fsolve(equations, [0.01, 0.5, 0.15])
```

We now have the linear CML function:

- ① `opt[0]` is the risk-free rate.
- ② `opt[1]` is the maximum Sharpe ratio.



Outline

① Portfolio optimization and the efficient frontier

② Principal Component Analysis

Principal Components

- 1 Assume you have N (potentially) correlated time series, e.g., stock prices.
- 2 You want to identify K common factors that drive most of the common variation in the stock prices.
- 3 PCA allows you to find portfolios (linear combinations of stocks) that are orthogonal to each other (zero correlation).
- 4 The first principal component is the portfolio that accounts for most commonality.
- 5 The fraction of the variance a PC accounts for is proportional to its *eigenvalue* (λ).

Application

- ① Let us load data on all CAC 40 stocks, including the index itself.

```
1 DataCAC=read_csv('L7_DataCAC.csv',index_col=0)
```

- ② We separate the index into a different DataFrame, and leave all stocks in DataCAC.

```
1 cac40=DataCAC["^FCHI"]  
2 DataCAC.pop("^FCHI")
```

- ③ PCA is usually performed on normalised data. Let us define a normalisation function:

```
1 def normalize(x): return (x-x.mean())/x.std()
```


Computing PC relative shares

- ① The PCA algorithm is available in:

```
1 from sklearn.decomposition import KernelPCA
```

- ② We fit principal components to normalised stock data:

```
1 pca=KernelPCA().fit(DataCAC.apply(normalize))
```

- ③ How many principal components do we find?

```
1 len(pca.lambdas_)
```

- ④ Let us compute the relative importance (in %) of the 10-largest eigenvalues.

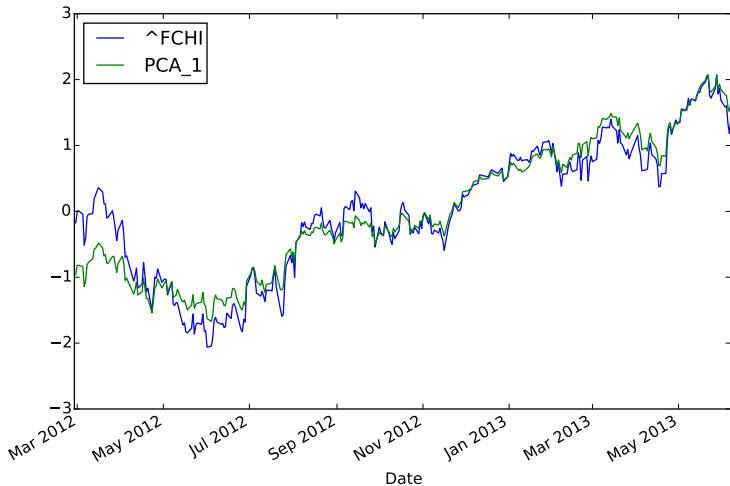
We need first to normalise the eigenvalues to add up to one.

```
1 def fractions(x): return x/x.sum()  
2 var_ratio=fractions(pca.lambdas_)[:10]
```

Constructing a PCA index

- 1 We try to replicate the movement in the index using principal components.
- 2 To do this, we need to transform the data into a single series, using the PCA weights.
- 3 First, let us just look at the first principal component.

```
1 pca=KernelPCA(n_components=1)
2   .fit(DataCAC.apply(normalize))
3 cac40['PCA_1']=pca.transform(-DataCAC)
```



Constructing a PCA index: more than one PC

Application:

- 1 If we want to use more than one PC to replicate the index, we need to weight the principal components.
- 2 A natural weight is of course, the proportion of the variance they explain.

```
1 pca=KernelPCA(n_components=5)
2   .fit(DataCAC.apply(normalize))
3 pca_components=pca.transform(-DataCAC)
4 weights=fractions(pca.lambdas_)
5 cac40['PCA_5']=np.dot(pca_components,weights)
```

