UROPS Project Presentation 2

Wang Zexin

Chpter 11 Statistics
of Python for Finance

Variance Reduction Techniques
of Monte Carlo methods in Financial Engineering

Quantitative Finance
National University of Singapore

January 27, 2017

Today's Agenda

1. Statistics
   - Normality test
   - Portfolio Optimization
   - Principal Componenet Analysis
   - Bayesian Regression

2. Variance Reduction Techniques
   - Control Variates

Changes due to different Python version

We are using Python 3.6 while the version in the book is Python 2.7
So here is a list of items to change

- print x now becomes print(x)
- dict.iteritems() now becomes dict.items()
- xrange now becomes range
- lambda (k, v) : (v, k) is no longer available
- instead we can only use: lambda x : (x[1], x[0])
- x / 2 is float division, while x // 2 is integer division

Installation requirements - pandas_datareader

We are going to install pandas_datareader instead of pandas.io.data

Reason being that: ImportError: The pandas.io.data module is moved to a separate package (pandas-datareader).

Also, Anaconda does not support direct installation of this new package.

Hence, here is one line of code which you can type in Anaconda Prompt.

conda install -c https://conda.anaconda.org/anaconda pandas-datareader

We shall go through these useful methods

- Normality test
- Portfolio theory
- Principal component analysis
- Bayesian Regression

Normality test - wrapper functions for np arrays

```
def normality_tests(arr):
    ''' Tests for normality distribution of given data set.
    Parameters
    ==========
    array: ndarray
    object to generate statistics on
    '''
    print("Skew of data set %14.3f" % scs.skew(arr))
    print("Skew test p-value %14.3f" % scs.skewtest(arr)[1])
    print("Kurt of data set %14.3f" % scs.kurtosis(arr))
    print("Kurt test p-value %14.3f" % scs.kurtosistest(arr)[1])
    print("Norm test p-value %14.3f" % scs.normaltest(arr)[1])
```

Portfolio Optimization

Assumptions we made are:

- Stock prices follow CAPM mostly
- Use only close price of the stock (adj close)
- Use csv.reader instead of pandas in this section

Portfolio Optimization

We are going to optimize based on these:

- higher order statistics function
- maximization of Sharpe ratio - MVE
- minimization of portfolio variance - MVP
- Efficient Frontier
- Capital Market Line

Read stock data from CSV files

```
def readDataFromCSV(filename, noa):
    infile = open(filename, newline = '')
    reader = csv.reader(infile)

    dates = []
    stocks = [[] for x in range(noa)];
    for row in reader:
        if row[0] != "Date":
            dates.append(row[0])
            for index in range(noa):
                stocks[index].append(float(row[index+1]))

    infile.close()
    return stocks
```

Convert stock prices to returns

There is a bug in the code provided in the book.
nparray.mean() will only calculate one average for the entire 2D array

```
def calculateReturnData(dataset):
    for stock in dataset:
        stock.reverse()
    returnData = []
    returnMean = []
    for stock in dataset:
        returns = []
        for i in range(1,len(stock)):
            returns.append(stock[i] / stock[i-1] - 1)
        returnData.append(returns)
        returnMean.append(np.mean(returns))
    returnData = np.array(returnData, np.float64)
    returnMean = np.array(returnMean, np.float64)
    return returnData, returnMean
```

Generate statistics function

We can create a high order function to generate statistics function for a given set of return data

```
def generate_statistics(rets, retsMean):
    def statistics(weights):
        ''' Returns portfolio statistics.
        Parameters
        ==========
        weights : array-like
        weights for different securities in portfolio
        Returns
        =======
        pret : float
        expected portfolio return
        pvol : float
        expected portfolio volatility
        pret / pvol : float
        Sharpe ratio for rf=0
        '''
        weights = np.array(weights)
        pret = np.sum(retsMean * weights) * 252
        pvol = np.sqrt(np.dot(weights.T, np.dot(np.cov(rets, bias=True) * 252, weights)))
        return np.array([pret, pvol, pret / pvol])
    return statistics

returnData, returnMean = calculateReturnData(readDataFromCSV("Close Prices Data.csv"),5)
stat = generate_statistics(returnData, returnMean)
```

Optimize portfolio to maximise Sharpe ratio

Make use of a new library scipy.optimize
Maximsie Sharpe ratio by minimizing negative Sharpe ratio

```
import scipy.optimize as sco

def optimize_portfolio_max_SR(statistics, noa):
    min_func_sharpe = lambda weights : -statistics(weights)[2]
    cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bnds = tuple((0, 1) for x in range(noa))
    EWP = noa * [1. / noa,]
    return sco.minimize(min_func_sharpe, EWP, method='SLSQP',
                        bounds=bnds, constraints=cons)
```

Return value is a dictionary.
w = optimize_portfolio_max_SR(stat, 5)
w['x'] is the optimal portfolio

Optimize to minimise portfolio variance

$$\text{Portfolio variance} = w^T \Sigma w$$

```
def optimize_portfolio_min_Var(statistics, noa):
    min_func_var = lambda weights : statistics(weights)[1]
    cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bnds = tuple((0, 1) for x in range(noa))
    EWP = noa * [1. / noa,]
    return sco.minimize(min_func_var, EWP,
        method='SLSQP',bounds=bnds, constraints=cons)
```

Return value is a dictionary.

w = optimize_portfolio_min_Var(stat, 5)

w['x'] is the optimal portfolio

Generate Efficient Frontier

We can set certain level of return to generate points on frontier.

```
def portfolio_efficient_frontier(statistics, noa, tret):
    min_func_var = lambda weights : statistics(weights)[1]
    cons = ({'type': 'eq', 'fun': lambda x: statistics(x)[0] - tret},
        {'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bnds = tuple((0, 1) for x in range(noa))
    EWP = noa * [1. / noa,]
    return sco.minimize(min_func_var, EWP,
                        method='SLSQP',bounds=bnds, constraints=cons)

def create_efficient_frontier(statistics, noa):
    targetReturns = np.linspace(0.0, 0.25, 50)
    points = []
    for targetReturn in targetReturns:
        opts = portfolio_efficient_frontier(statistics, noa, targetReturn)
        w = opts['x']
        points.append(statistics(w))
    return points
```

Capital Market Line

We can solve a system of equations to obtain this line.

$$
\begin{aligned}
a &= r_f \\
a + bx &= f(x) \\
b &= f'(x)
\end{aligned}
$$

```python
import scipy.optimize as sco
import scipy.interpolate as sci

def f(x):
    ''' Efficient frontier function (splines approximation).'''
    return sci.splev(x, tck, der=0)
def df(x):
    ''' First derivative of efficient frontier function.'''
    return sci.splev(x, tck, der=1)

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

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

Principal Componenet Analysis

- Obtain data using pandas (initialization)
- Determine minimum number of components
- Create PCA Index

PCA - initialization

```python
from sklearn.decomposition import KernelPCA
import pandas_datareader.data as web
import numpy as np
import pandas as pd
def initialization():
    symbols = ['ADS.DE', 'ALV.DE', 'BAS.DE', 'BAYN.DE', 'BEI.DE',
        'BMW.DE', 'CBK.DE', 'CON.DE', 'DAI.DE', 'DB1.DE',
        'DBK.DE', 'DPW.DE', 'DTE.DE', 'EOAN.DE', 'FME.DE',
        'FRE.DE', 'HEI.DE', 'HEN3.DE', 'IFX.DE', 'LHA.DE',
        'LIN.DE', 'LXS.DE', 'MRK.DE', 'MUV2.DE', 'RWE.DE',
        'SAP.DE', 'SDF.DE', 'SIE.DE', 'TKA.DE', 'VOW3.DE',
        '^GDAXI']
    data = pd.DataFrame()
    for sym in symbols:
        data[sym] = web.DataReader(sym, data_source='yahoo')['Close']
    data = data.dropna()
    dax = pd.DataFrame(data.pop('^GDAXI'))
```

PCA - Obtain stock data

```
In [5]: data[data.columns[:6]].head()
Out[5]:
            ADS.DE  ALV.DE  BAS.DE  BAYN.DE  BEI.DE  BMW.DE
Date
2010-01-04  38.505   88.54  44.850    56.40  46.445  32.050
2010-01-05  39.720   88.81  44.170    55.37  46.200  32.310
2010-01-06  39.400   89.50  44.450    55.02  46.165  32.810
2010-01-07  39.745   88.47  44.155    54.30  45.700  33.100
2010-01-08  39.600   87.99  44.020    53.82  44.380  32.655
```

PCA - Determine number of components required

```
In [10]: len(pca.lambdas_)
Out[10]: 881

In [11]: pca.lambdas_[:10].round()
Out[11]:
array([ 34079.,   5990.,   5360.,   2825.,   2018.,    848.,    756.,
          531.,    309.,    226.])

In [12]: get_we(pca.lambdas_)[:10]
Out[12]:
array([ 0.63250034,  0.11117324,  0.0994747 ,  0.05242408,  0.03744675,
        0.01572997,  0.01403163,  0.00985717,  0.00573016,  0.00418554])

In [13]: get_we(pca.lambdas_)[:5].sum()
Out[13]: 0.93301911384645353
```

More then 93.3% of the variation is explained by the first five components!

PCA - create index

```python
def create_PCA_index(data, dax):
    scale_function = lambda x: (x - x.mean()) / x.std() # convenience function
    get_we = lambda x: x / x.sum() # convenience function
    pca = KernelPCA().fit(data.apply(scale_function)) # multiple components

    pca = KernelPCA(n_components=1).fit(data.apply(scale_function)) # single component
    dax['PCA_1'] = pca.transform(-data)

    pca = KernelPCA(n_components=5).fit(data.apply(scale_function)) # five components
    pca_components = pca.transform(-data)
    weights = get_we(pca.lambdas_)
    dax['PCA_5'] = np.dot(pca_components, weights)

    cut_date = '2011/7/1'
    early_pca = dax[dax.index < cut_date]['PCA_5']
    early_reg = np.polyval(np.polyfit(early_pca,
    dax['^GDAXI'][dax.index < cut_date], 1),early_pca)

    late_pca = dax[dax.index >= cut_date]['PCA_5']
    late_reg = np.polyval(np.polyfit(late_pca,
    dax['^GDAXI'][dax.index >= cut_date], 1),late_pca)
```

Bayesian Regression

- Trouble shooting for package Pymc3
- Building of model
- Gaussian random walk
- Uniform model
- Optimization

Trouble shooting for package Pymc3

If this ValueError happens, go to the file font_manager.py

```
path = _getfullpathname(path)

ValueError: _getfullpathname: embedded null character
```

and add in one line of code in the function win32InstalledFonts

```
        try:
            for j in range(winreg.QueryInfoKey(local)[1]):
                try:
                    key, direc, any = winreg.EnumValue( local, j)
                    if not is_string_like(direc):
                        continue
                    if not os.path.dirname(direc):
                        direc = os.path.join(directory, direc)
                    direc = direc.split('\0', 1)[0]
                    direc = os.path.abspath(direc).lower()
                    if os.path.splitext(direc)[1][1:] in fontext:
                        items[direc] = 1
                except EnvironmentError:
                    continue
                except WindowsError:
                    continue
                except MemoryError:
                    continue
            return list(six.iterkeys(items))
        finally:
            winreg.CloseKey(local)
```

Building model

If you can run this without having AttributeError,

```
import pymc as pm
import numpy as np
np.random.seed(1000)

x = np.linspace(0, 10, 500)
y = 4 + 2 * x + np.random.standard_normal(len(x)) * 2

reg = np.polyfit(x, y, 1)

with pm.Model() as model:
    # model specifications in PyMC3
    # are wrapped in a with statement
    # define priors
    alpha = pm.Normal('alpha', mu=0, sd=20)
    beta = pm.Normal('beta', mu=0, sd=20)
    sigma = pm.Uniform('sigma', lower=0, upper=10)
    # define linear regression
    y_est = alpha + beta * x
    # define likelihood
    likelihood = pm.Normal('y', mu=y_est, sd=sigma, observed=y)
    # inference
    start = pm.find_MAP()
    # find starting value by optimization
    step = pm.NUTS(state=start)
    # instantiate MCMC sampling algorithm
    trace = pm.sample(100, step, start=start, progressbar=False)
    # draw 100 posterior samples using NUTS sampling
```

that means your version of Pymc3 is correctly installed.

Trouble shooting for package Pymc3

If this AttributeError (__exit__) happens,

```
with pm.Model() as model:

AttributeError: __exit__
```

that means you are using a version of Pymc3 in which with statement is not incorporated yet. You should go to Anaconda prompt and type: conda install -c conda-forge pymc3

Trouble shooting for package Pymc3

If this AttributeError (TransformedVar) happens,

```
    model_randomwalk.TransformedVar('sigma_alpha',

AttributeError: 'Model' object has no attribute 'TransformedVar'
```

when you are running the model below

```
model_randomwalk = pm.Model()
with model_randomwalk:
    # std of random walk best sampled in log space
    sigma_alpha, log_sigma_alpha = \
        model_randomwalk.TransformedVar('sigma_alpha',
        pm.Exponential.dist(1. / .02, testval=.1),
        pm.logtransform)
    sigma_beta, log_sigma_beta = \
        model_randomwalk.TransformedVar('sigma_beta',
        pm.Exponential.dist(1. / .02, testval=.1),
        pm.logtransform)
```

the reason is that the attribute TransformedVar is removed from Pymc3.

Trouble shooting for package Pymc3

If the AttributeError (TransformVar) happens, one fix is to use Exponential attribute of the package itself.

```
model_randomwalk = pm.Model()
with model_randomwalk:
    # std of random walk best sampled in log space
    sigma = pm.Exponential('sigma', 1./.02, testval = .1)
```

Trouble shooting for package Zipline

This package seems to be conflicting with python 3.5.0 and above

```
with pm.Model() as model:

AttributeError: __exit__
```

so it is not feasible for us to use zipline,
instead we can use pandas just like in the previous section

Gaussian walk using pandas data

This is an example of gaussian walk model using pandas library data

```
import pandas_datareader.data as web
import pandas as pd
import numpy as np
import pymc3 as pm

symbols = ['GLD', 'GDX']
data = pd.DataFrame()
for sym in symbols:
    data[sym] = web.DataReader(sym, data_source='yahoo')['Close']
data = data.dropna()

#from pymc3.distributions.timeseries import GaussianRandomWalk
# to make the model simpler, we will apply the same coefficients
# to 50 data points at a time
subsample_alpha = 50
subsample_beta = 50
model_randomwalk = pm.Model()
with model_randomwalk:
    # std of random walk best sampled in log space
    sigma_alpha = pm.Exponential('sigma_alpha', 1./.02, testval = .1)
    sigma_beta = pm.Exponential('sigma_beta', 1./.02, testval = .1)

with model_randomwalk:
    alpha = pm.GaussianRandomWalk('alpha', sigma_alpha**-2,
                                  shape= len(data) // subsample_alpha)
    beta = pm.GaussianRandomWalk('beta', sigma_beta**-2,
                                 shape= len(data) // subsample_beta)
    # make coefficients have the same length as prices
    alpha_r = np.repeat(alpha, subsample_alpha)
    beta_r = np.repeat(beta, subsample_beta)
```

Add pm. in front of GaussianRandomWalk

Uniform model - errorneous

There is error occurred when we try to multiply two vectors with different lengths.

```
with model_randomwalk:
    # define regression
    regression = alpha_r + beta_r * data.GDX.values[:1950]
    # assume prices are normally distributed
    # the mean comes from the regression
    sd = pm.Uniform('sd', 0, 20)
    likelihood = pm.Normal('GLD',
            mu=regression,
            sd=sd,
            observed=data.GLD.values[:1950])
```

This error has not been solved.

Bayesian regression - optimization

```
import scipy.optimize as sco
with model_randomwalk:
    # first optimize random walk
    start = pm.find_MAP(vars=[alpha, beta], fmin=sco.fmin_l_bfgs_b)
    # sampling
    step = pm.NUTS(scaling=start)
    trace_rw = pm.sample(100, step, start=start, progressbar=False)
```

Control Variates - Mathematics

Our aim is to estimate $\mathrm{E}[Y_i]$
Under simulation, we use :

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

(unbiased estimate)
Now we introduce a new variable X, with realizations X_i, for i from 1 to n
Define $Y_i(\lambda) = Y_i - \lambda(X_i - \mathrm{E}(X))$

$$\bar{Y}(\lambda) = \bar{Y} - \lambda(\bar{X} - \mathrm{E}(X)) = \frac{1}{n}\sum_{i=1}^{n}[Y_i - \lambda(X_i - \mathrm{E}(X))]$$

Control Variates - Unbiasedness and Consistency

The new estimate $\bar{Y}(\lambda)$ is unbiased and consistent.

Unbiasedness:

$$\mathrm{E}(\bar{Y}(\lambda)) = \mathrm{E}[\bar{Y} - \lambda(\bar{X} - \mathrm{E}(X))] = \mathrm{E}(\bar{Y}) - \lambda(\mathrm{E}(\bar{X}) - \mathrm{E}(X)) = \mathrm{E}(Y)$$

$$\begin{aligned} \textit{Consistency}: \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} Y_i(\lambda) &= \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}[Y_i - \lambda(X_i - \mathrm{E}(X))] \\ &= \mathrm{E}[Y - \lambda(X - \mathrm{E}(X))] \\ &= \mathrm{E}(Y) \end{aligned}$$

Control Variates - Controlling Variance of estimate

$$
\begin{aligned}
\mathrm{Var}[Y_i(\lambda)] &= \mathrm{Var}[Y_i - \lambda(X_i - \mathrm{E}(X))] \\
&= \mathrm{Var}[Y_i - \lambda X_i] \\
&= \mathrm{Var}(Y_i) + \lambda^2 \mathrm{Var}(X_i) - 2\lambda \mathrm{Cov}(X_i, Y_i) \\
&= \sigma_Y^2 + \lambda^2 \sigma_X^2 - 2\lambda \sigma_X \sigma_Y \rho_{XY}
\end{aligned}
$$

In order to find the minimum variance by varying λ

Set $\frac{\partial \mathrm{Var}[Y_i(\lambda)]}{\partial \lambda} = 2\lambda\sigma_X^2 - 2\sigma_X\sigma_Y\rho_{XY}$ to 0:

$$
\lambda^* = \frac{2\sigma_X\sigma_Y\rho_{XY}}{2\sigma_X^2} = \frac{\sigma_X\sigma_Y\rho_{XY}}{\sigma_X^2} = \frac{\mathrm{Cov}(X, Y)}{\mathrm{Var}(X)}
$$

Control Variates - Controlling Variance of estimate

Compare the new variance with the old:

$$
\begin{aligned}
\frac{\mathrm{Var}[Y_i - \lambda^*(X_i - \mathrm{E}(X))]}{\mathrm{Var}(Y)} &= \frac{\sigma_Y^2 + \lambda^{*2}\sigma_X^2 - 2\lambda^*\sigma_X\sigma_Y\rho_{XY}}{\sigma_Y^2} \\
&= 1 + \frac{\frac{\mathrm{Var}(X)(\mathrm{Cov}(X,Y))^2}{(\mathrm{Var}(X))^2} - \frac{2(\mathrm{Cov}(X,Y))^2}{\mathrm{Var}(X)}}{\sigma_Y^2} \\
&= 1 + \frac{\frac{\sigma_X^4\sigma_Y^2\rho_{XY}^2}{\sigma_X^4} - \frac{2\sigma_X^2\sigma_Y^2\rho_{XY}^2}{\sigma_X^2}}{\sigma_Y^2} \\
&= 1 + \frac{\sigma_Y^2\rho_{XY}^2 - 2\sigma_Y^2\rho_{XY}^2}{\sigma_Y^2} \\
&= 1 - \rho_{XY}^2
\end{aligned}
$$

Conclusion from the theoretical results:

The stronger the correlation, the better the reduction in variance.

Choices of Control Variates

We can use several different random variables as control variates.

- Underlying asset prices
- Tractable options
- Bond prices
- Tractable dynamics

Control Variates - Underlying asset prices

The underlying asset prices provide a source of control variates for its natural correlation with the derivative payoff.
The control variate estimator is formed like this:

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \lambda[S_i(T) - e^{rT}S(0)])$$

For European Call option, $Y = e^{-rT}(S(T) - K)^+$. However, one problem arises: as the strike price goes higher, the correlation of the underlying asset prices and the payoff decreases, this diminishes the effect of control variates upon variance reduction.

Thank You

E0007424@u.nus.edu