n05_mcs_multi_asset_path

May 15, 2017

0.0.1 Introduction to Monte Carlo Simulation in Finance

1 Multiasset Simulation

1.1 Cholesky Decomposition

The **Choleski Decomposition** makes an appearance in Monte Carlo Methods where it is used to simulating systems with correlated variables. Cholesky decomposition is applied to the correlation matrix, providing a lower triangular matrix A, which when applied to a vector of uncorrelated samples, u, produces the covariance vector of the system. Thus it is highly relevant for quantitative trading.

The standard procedure for generating a set of correlated normal random variables is through a linear combination of uncorrelated normal random variables; Assume we have a set of n independent standard normal random variables Z and we want to build a set of n correlated standard normals Z' with correlation matrix Σ

$$Z' = AZ, \qquad AA^t = \Sigma$$

We can find a solution for A in the form of a triangular matrix

$$\begin{pmatrix} A_{11} & 0 & \dots & 0 \\ A_{21} & A_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix}$$

diagonal elements

$$a_{ii} = \sqrt{\Sigma_{ii} - \sum_{k=1}^{i-1} a_{ik}^2}$$

off-diagonal elements

$$a_{ij} = \frac{1}{a_{ii}} \left(\Sigma_{ij} - \sum_{k=1}^{i-1} a_{ik} a_{jk} \right)$$

Using Python, the most efficient method in both development and execution time is to make use of the NumPy/SciPy linear algebra (linalg) library, which has a built in method cholesky to decompose a matrix. The optional lower parameter allows us to determine whether a lower or upper triangular matrix is produced:

```
In [1]: import pprint
        import scipy
        import scipy.linalg   # SciPy Linear Algebra Library

        A = scipy.array([[6, 3, 4, 8], [3, 6, 5, 1], [4, 5, 10, 7], [8, 1, 7, 25]])
        L = scipy.linalg.cholesky(A, lower=True)
        U = scipy.linalg.cholesky(A, lower=False)

        print "A:"
```

```
        pprint.pprint(A)

        print "L:"
        pprint.pprint(L)

        print "U:"
        pprint.pprint(U)
```

```
A:
array([[ 6,  3,  4,  8],
       [ 3,  6,  5,  1],
       [ 4,  5, 10,  7],
       [ 8,  1,  7, 25]])
L:
array([[ 2.44948974,  0.        ,  0.        ,  0.        ],
       [ 1.22474487,  2.12132034,  0.        ,  0.        ],
       [ 1.63299316,  1.41421356,  2.30940108,  0.        ],
       [ 3.26598632, -1.41421356,  1.58771324,  3.13249102]])
U:
array([[ 2.44948974,  1.22474487,  1.63299316,  3.26598632],
       [ 0.        ,  2.12132034,  1.41421356, -1.41421356],
       [ 0.        ,  0.        ,  2.30940108,  1.58771324],
       [ 0.        ,  0.        ,  0.        ,  3.13249102]])
```

For example, for a two-dimension random vector we have simply

$$A = \begin{pmatrix} \sigma_1 & 0 \\ \sigma_2\rho & \sigma_2\sqrt{1-\rho^2} \end{pmatrix}$$

Say one needs to generate two correlated normal variables x_1 and x_2. All one needs to do is to generate two uncorrelated Gaussian random variables z_1 and$ z_2$ and set

$$x_1 = z_1$$

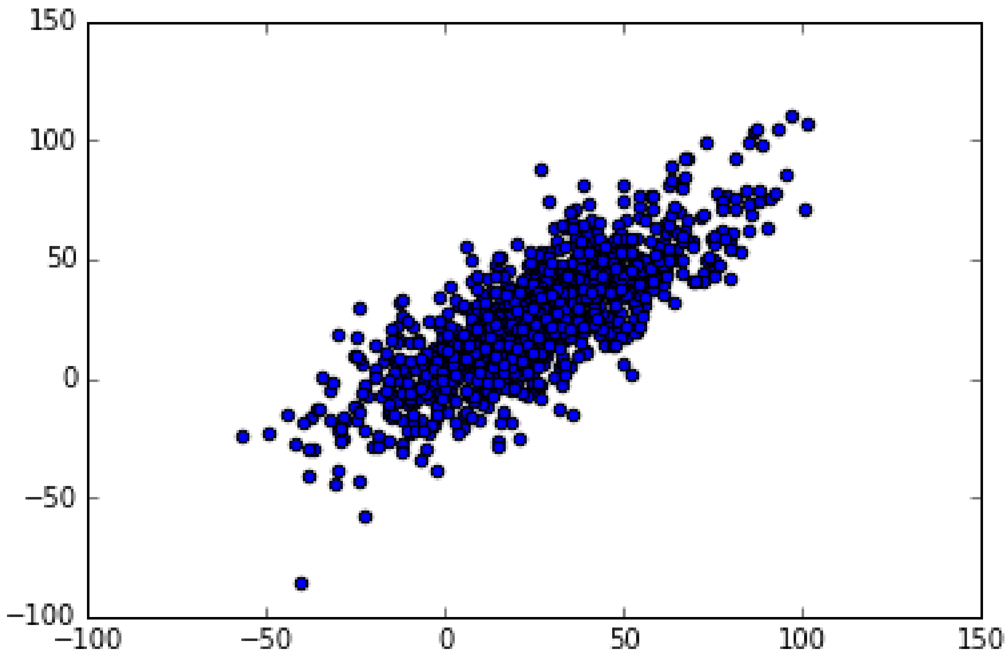
$$x_2 = \rho z_1 + \sqrt{1-\rho^2} z_2$$

In Python everything you need is available in the *numpy* library, as we can see in the next example.

```
In [4]: %matplotlib inline

        import numpy as np
        import scipy as sc

        from math        import sqrt
        from scipy.stats import norm as scnorm
        from pylab       import *
        from matplotlib  import pyplot as pl

        xx = np.array([-0.51, 51.2])
        yy = np.array([0.33, 51.6])
        means = [xx.mean(), yy.mean()]
        stds  = [xx.std()  , yy.std() ]
        corr  = 0.8         # correlation
        covs  = [[stds[0]**2          , stds[0]*stds[1]*corr],
                [stds[0]*stds[1]*corr,           stds[1]**2]]
```

```
m = np.random.multivariate_normal(means, covs, 1000).T
scatter(m[0], m[1])
```

Out[4]: <matplotlib.collections.PathCollection at 0x14b1e550>



1.2 Brownian simulation of correlated assets

When using Monte Carlo methods to price options dependent on a basket of underlying assets (multidimensional stochastic simulations), the correlations between assets should be considered. Here I will show an example of how this can be simulated using pandas.

Download and prepare the data

First we download some data from Yahoo:

In [12]:
```
from pandas_datareader import DataReader
from pandas import Panel, DataFrame

symbols = ['AAPL',     # Apple Inc.
           'GLD',      # SPDR Gold Trust ETF
           'SNP',      # S&P 500 Index
           'MCD']      # McDonald's Corporation
data = dict((symbol, DataReader(symbol, "yahoo", pause=1)) for symbol in symbols)
panel = Panel(data).swapaxes('items', 'minor')
closing = panel['Close'].dropna()
closing.tail()
```

Out[12]:

	AAPL	GLD	MCD	SNP
Date				
2017-05-08	153.009995	116.750000	144.240005	78.589996
2017-05-09	153.990005	116.050003	144.360001	80.250000

```
2017-05-10  153.259995  116.040001  144.520004  80.779999
2017-05-11  153.949997  116.500000  144.210007  80.300003
2017-05-12  156.100006  116.830002  145.360001  80.919998
```

Now we can calculate the log returns:

```
In [14]: rets = log(closing / closing.shift(1)).dropna()
         rets.tail()
```

```
Out[14]:                 AAPL       GLD       MCD       SNP
         Date
         2017-05-08  0.026825 -0.002225  0.001943 -0.000255
         2017-05-09  0.006384 -0.006014  0.000832  0.020902
         2017-05-10 -0.004752 -0.000086  0.001108  0.006583
         2017-05-11  0.004492  0.003956 -0.002147 -0.005960
         2017-05-12  0.013869  0.002829  0.007943  0.007691
```

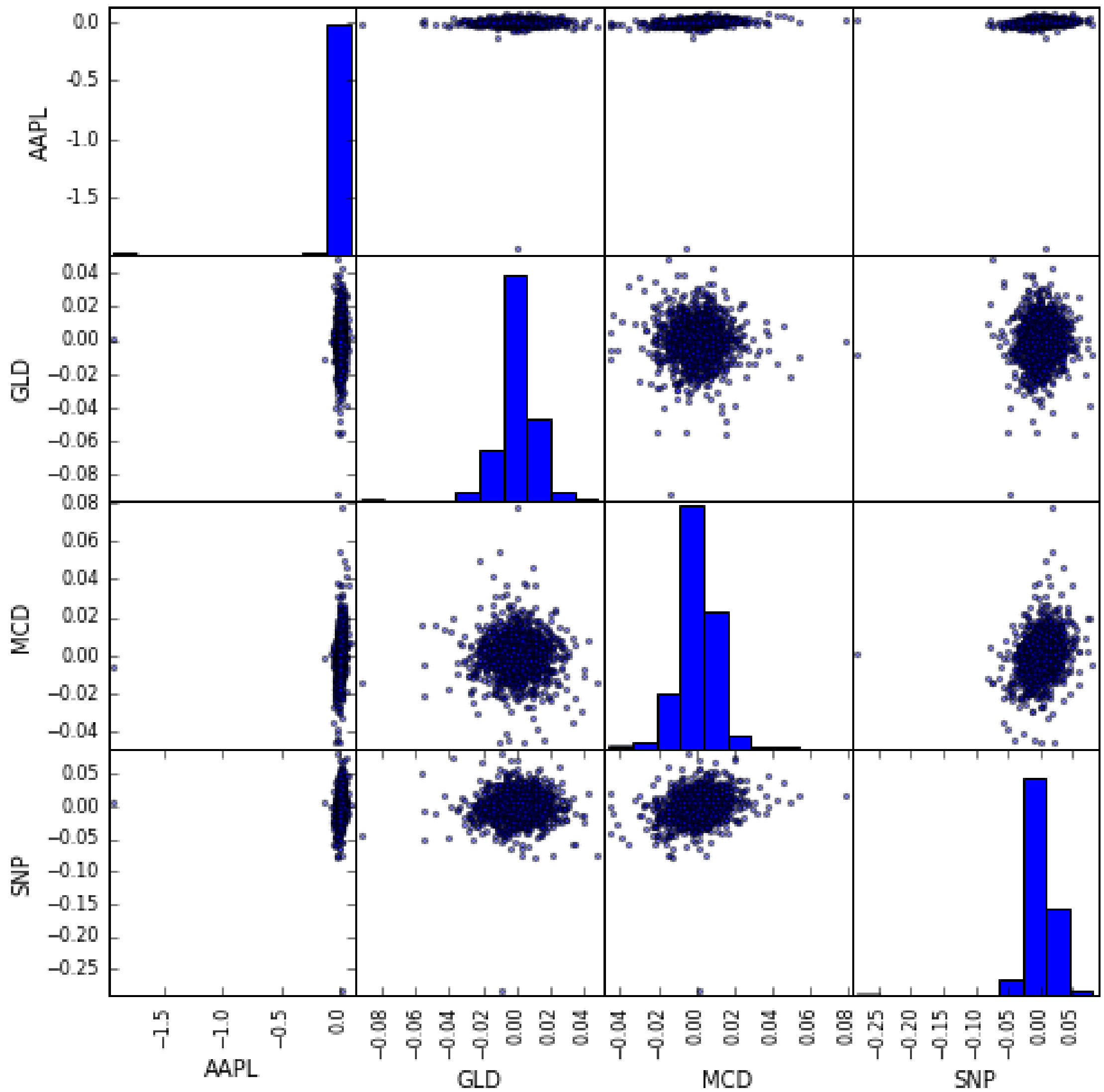
The correlation matrix has information about the historical correlations between stocks in the group. We work under the assumption that this quantity is conserved, so the generated stocks will need to satisfy this condition:

```
In [15]: corr_matrix = rets.corr()
         corr_matrix
```

```
Out[15]:           AAPL       GLD       MCD       SNP
         AAPL  1.000000  0.013416  0.122181  0.087813
         GLD   0.013416  1.000000 -0.029849  0.049627
         MCD   0.122181 -0.029849  1.000000  0.270303
         SNP   0.087813  0.049627  0.270303  1.000000
```

So the most correlated assets are MCD (McDonald's Corporation) and the SPX (S&P 500 Index). Pandas has a nice utility to plot the correlations:

```
In [16]: from pandas.tools.plotting import scatter_matrix

         scatter_matrix(rets, figsize=(8,8));
```



1.2.1 Simulation

The simulation procedure for generating random variables will go like this:

1. Calculate the Cholesky Decomposition matrix, this step will return an upper triangular matrix L^T.
2. Generate random vector $X \sim N(0, 1)$.
3. Obtain a correlated random vector $Z = XL^T$.

As we have previously seen the Cholesky decomposition of the correlation matrix corr_matrix is impemented in scipy:

```
In [17]: from scipy.linalg import cholesky

         upper_cholesky = cholesky(corr_matrix, lower=False)
         upper_cholesky
```

```
Out[17]: array([[ 1.        ,  0.01341619,  0.12218137,  0.08781314],
               [ 0.        ,  0.99991   , -0.03149062,  0.04845349],
               [ 0.        ,  0.        ,  0.99200809,  0.26320354],
               [ 0.        ,  0.        ,  0.        ,  0.9595129 ]])
```

We set up the parameters for the simulation:

```
In [18]: import numpy as np
         from pandas import bdate_range   # business days

         n_days = 21
         dates = bdate_range(start=closing.ix[-1].name, periods=n_days)
         n_assets = len(symbols)
         n_sims = 1000
         dt = 1./252
         mu = rets.mean().values
         sigma = rets.std().values*sqrt(252)
         np.random.seed(1234)            # init random number generator for reproducibility
```

Now we generate the correlated random values X:

```
In [19]: rand_values = np.random.standard_normal(size = (n_days * n_sims, n_assets)) #
         corr_values = rand_values.dot(upper_cholesky)*sigma
         corr_values

Out[19]: array([[ 0.35713951, -0.20102995,  0.22872374,  0.01834271],
               [-0.54588781,  0.14890697,  0.11112759, -0.12214037],
               [ 0.01189091, -0.38053806,  0.18302371,  0.34614481],
               ...,
               [-0.54867396, -0.31335726, -0.16700762, -0.900278  ],
               [ 0.33051825, -0.19508407, -0.10498318, -0.27410942],
               [ 0.48198086,  0.14354678, -0.11908019,  0.11289373]])
```

With everything set up we can start iterating through the time interval. The results for each specific time are saved along the third axis of a pandas Panel.

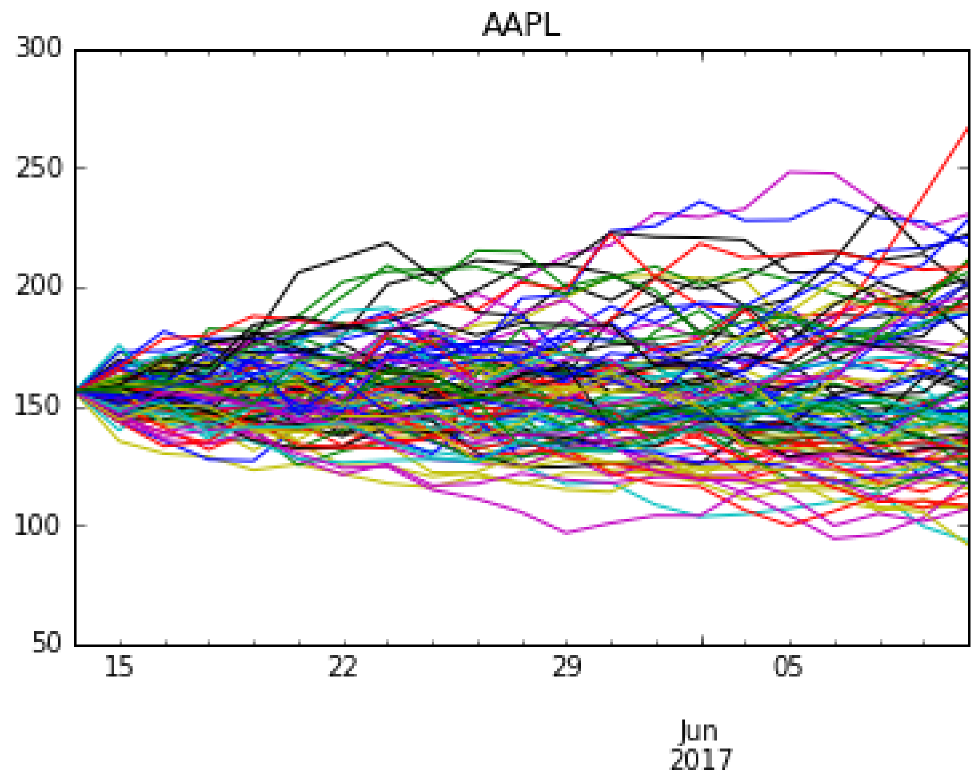
```
In [20]: prices = Panel(items=range(n_sims), minor_axis=symbols, major_axis=dates)
         prices.ix[:, 0, :] = closing.ix[-1].values.repeat(1000).reshape(4,1000).T # set initial values

         for i in range(1,n_days):
             prices.ix[:, i, :] = prices.ix[:, i-1,:] * (exp((mu-0.5*sigma**2)*dt +  sqrt(dt)*corr_valu

         prices.ix[123, :, :].head()   # show random path
```

Out[20]:

	AAPL	GLD	SNP	MCD
2017-05-12	156.100006	116.830002	145.360001	80.919998
2017-05-15	166.181969	117.322889	146.435432	81.309450
2017-05-16	155.136020	116.342457	147.019022	80.702090
2017-05-17	163.772165	117.374372	146.367537	81.071422
2017-05-18	170.116168	117.436894	146.215424	78.741046

And thats all! Now it is time to check our results. First a plot of all random paths for AAPL (Apple Inc.).

```
In [21]: prices.ix[::10, :, 'AAPL'].plot(title='AAPL', legend=False);
```



We can take a look at the statistics for the last day:

```
In [22]: prices.ix[:, -1, :].T.describe()
```

Out[22]:

	AAPL	GLD	SNP	MCD
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	155.583050	116.788898	145.375955	80.892663
std	34.314169	5.650537	6.257044	6.639877
min	79.442820	102.020436	123.501637	58.744554
25%	130.764323	112.768299	141.041126	76.199721
50%	151.124855	116.647954	145.438818	80.820570
75%	176.884535	120.559014	149.263351	85.141468
max	269.846082	134.558858	168.545591	101.942106