Multivariate Normal Monte Carlo

The Monte Carlo Method is something that is used widely throughout risk management. Often we are dealing with functions of random variables, from which the final distribution of outcomes is unknown. The Monte Carlo method gives us a way to approximate that final distribution.

Drawing univariate random numbers is easy, most software packages have a random function and many will allow you to specify a distribution.

Sometimes, only the uniform distribution is available from the random number generator, defined on the interval (0,1). Using the quantile function, inverse CDF, we can use that to give us random draws from a distribution.

Quantile Function is the inverse of the CDF. It is the value of x such that F(X) = p

$$F^{-1}(p)$$

Remember the CDF function is

$$F(x) = P(X <= x) = \int_{lb}^{x} f(t)dt \text{ , where lb is the lower bound of the distribution}$$

The CDF is monotonic and defined between (0,1), so the Quantile function gives a 1:1 mapping from (0,1) into the range of the distribution.

$$x = F^{-1}(random\ uniform)$$

Often we need to simulate a system of correlated values. Most commonly, we use a multivariate normal distribution.

Recall last week we stated if X was multivariate normal

$$X \sim N(\mu, \Sigma)$$

There exists

$$\mu \in \mathbb{R}^n,\ L \in \mathbb{R}^{nxm} \text{ such that } X = LZ + \mu$$

Where

$$Z_i \sim N(0,1), \quad i \in [1,m]$$

$$\Sigma = LL'$$

This provides a path for the generation of multivariate random normals.

Multivariate Draw Algorithm: To simulation K random draws from a N variables distributed multivariate normal, we need:

1. Factor Σinto Lwhich is (NxM)
2. Generate $K * M$random standard normals, Z, shaped (MxK)
3. $X = (LZ + \mu)^T$, with shape (KxN)
    a. Each series is on the row without the transpose. We tend to think of tables with columns of values, so we use the transpose to put the matrix into a data table format.

A lot of packages have a multivariate normal random number generator. However, with the some of the data problems we will encounter, knowing how to generate your own will be necessary.

Cholesky Factorization

The most used factorization of $\Sigma$is the Cholesky Root. The Cholesky factorization can be applied to positive definite (PD) matrices. The factor $L$is a lower triangular matrix. This is sometimes referred to as the Cholesky root, as the factor is analogous of the square root.

The Cholesky root is roughly 2x more efficient to calculate than the LUfactorization. For this reason it is more often used.

If the matrix is only positive semi-definite (PSD), the Cholesky root can still be calculated allowing for a column of 0 values. The number of non-zero columns aligns with the number of positive eigenvalues (the rank) of the matrix.

Cholesky Algorithm:

Wikipedia has an example of a 3x3 matrix.

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix}$$

$$= \begin{pmatrix} L_{11}^2 & & (\text{symmetric}) \\ L_{21}L_{11} & L_{21}^2 + L_{22}^2 & \\ L_{31}L_{11} & L_{31}L_{21} + L_{32}L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix},$$

$$\mathbf{L} = \begin{pmatrix} \sqrt{A_{11}} & 0 & 0 \\ A_{21}/L_{11} & \sqrt{A_{22} - L_{21}^2} & 0 \\ A_{31}/L_{11} & \left(A_{32} - L_{31}L_{21}\right)/L_{22} & \sqrt{A_{33} - L_{31}^2 - L_{32}^2} \end{pmatrix}$$

This is solved either across the rows or down the columns.

Down the columns:

1. Column *j* , start on the diagonal element
2. Subtract the sum of the squares of the values on the root matrix for row *j* from the value on the input matrix on the diagonal.
3. Update the root matrix at position (*j,j*) with the square root of #2
4. Moving down the column, row *i*
    a. Calculate the dot product of sub matrix vector [*i, 1:(j-1)*] and [*j, 1:(j-1)*]
    b. Subtract a. from the (*i,j*) element of the input matrix.
    c. Divide b. by the *j* diagonal element of the root matrix
    d. Store that value in element (*i,j*) of the root matrix.
5. Repeat for the next column.

Most Linear Algebra packages use this algorithm on blocks on the matrix. The dot products become matrix multiplications, matrix inverse for the division, and the square root is a single threaded Cholesky call. The block size is chosen based on CPU cache size and vector operations are used for the matrix multiplications.
http://www.netlib.org/utk/papers/factor/node9.html

If the matrix is PSD instead of PD, the algorithm above breaks down. At least one of the root diagonal elements will be 0. The calculation of the column then fails when we attempt to divide by 0. In this case, we set all the values of the column to 0.

Missing Data

Missing values are very common in finance. Not all markets are open at the same time on the same days. A holiday in one market is not necessarily a holiday in another, even in the same country.

Imagine 3 Assets with different trading schedules. The table below shows the days in which a market is open and the asset trades

Date	Asset A	Asset B	Asset C
1	X	X	
2	X	X	X
3		X	X
4			
5	X		
6	X	X	X
7	X	X	X
8	X	X	
9	X	X	X
10		X	X

2 common way to calculate (we will talk about some others later)

1. Only use the days on which all markets are open.
    a. This limits the information.
    b. Only days 2, 6, 7, and 9 (40%) are open at the same time
    c. Information about asset behavior on the other days are ignored
2. Use pairwise calculations. Find the matching rows for each pair, and build the covariance matrix piece by piece.
    a. We use more information, but not all
    b. Assets A and B would use days 1, 2, 6, 7, 8, & 9.
    c. While we are building a symmetric matrix, we do not guarantee the matrix is PSD

Covariance Matrix Generation Techniques

There are many different techniques people use in practice for creating covariance matrices.

1. Default covariance function. With perfect data, and an assumption of normality, then the covariance as we described last class is the best estimator.
2. If there are missing values that do not fully overlap, but enough history that does, then the method of using only records with overlapping data works well (#1 above). It might not have the full information, but because you have a lot of data, the matrix should be OK.
3. If you are missing values and do not have a lot of fully overlapping observations, then pair wise estimation can be used (#2 above). This will often result in a non-PSD matrix which we will discuss shortly.
4. Financial data are time variant. Often we use a weight function to give more weight to recent observations – usually with weights decreasing exponentially. This is sometimes called the Risk Metrics covariance, after the firm that popularized it in the mid 1990s.
    a. We will detail the calculation later.
5. Another popular method is using an EM algorithm to estimate the covariance. Expectation Maximization is an iterative process using the likelihood function. This can deal with the missing data problem. However, as the proportion of missing values as a percent of the total observations increase, the results can be unstable. It is best suited to a large number of observations so that convergence can occur.
    a. We won't go into detail, but I would highly recommend looking into this.
6. When dealing with high frequency data, the issue of overlapping observations becomes even harder. With time intervals measured in milliseconds, it is impossible to see updated prices on all assets. Lots of work has gone into this area recently. There are a few packages "highfrequency" in R and "HighFrequencyCovariance" in Julia.
    a. If you are going into the field of automated trading and will be dealing with very short time interval data, start by reading the paper for "HighFrequencyCovariance" in Julia and follow the references. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3786912
    b. If someone knows of an analogous package in Python, please let me know and I will add it to the course notes.

7. Because the issue of rank of the matrix comes from the correlation estimation, we will use any of the methods in this list to calculate correlations, but use the full history of each variable for the variance calculation.

When we break the assumption of normality and we have what look like outliers to the normal distribution, the Pearson Coefficient may not be the best estimator of correlation. We will discuss this in a few weeks.

Exponentially Weighted Covariance

JP Morgan was one of the first banks to use exponentially weighted covariance matrices in their risk analytics starting in the late 1980s. Their risk management was considered some of the best in the industry, and they spun out a software and consulting company called "Risk Metrics."

$$(1)\ \sigma_t^2 = \lambda\sigma_{t-1}^2 + (1 - \lambda)\left(x_{t-1} - \bar{x}\right)^2$$

This is a simple exponential smoothing model. The variance for tomorrow is updated based on today's forecast variance and today's deviation from the mean.

Back substitutions and we get

$$(2)\ \sigma_t^2 = (1 - \lambda)\sum_{i=1}^{\infty}\lambda^i\left(x_{t-i} - \bar{x}\right)^2$$

Recognize that

$$(3)\ \lim_{n\to\infty}\ (1 - \lambda)\sum_{i=1}^{n}\lambda^i = 1$$

Setting the weight of a prior observation

$$w_{t-i} = (1 - \lambda)\,\lambda^i$$

Will give us a weighted estimator over an infinite horizon. Obviously we don't have an infinite horizon, so weights are normalized so that they sum to 1:

$$\widehat{w_{t-i}} = \frac{w_{t-i}}{\sum_{j=1}^{n} w_{t-j}}$$

Variance and Covariance Estimators are then

$$\widehat{\sigma_t^2} = \sum_{=1}^{n} w_{t-i}\left(x_{t-i} - \bar{x}\right)^2$$

$$\widehat{cov(x,y)} = \sum_{i=1}^{n} w_{t-i}\left(x_{t-i} - \bar{x}\right)^2\left(y_{t-i} - \bar{y}\right)^2$$

Equal weighting the time series reduces to the MLE values.

Risk Metrics research found that across a number of asset classes, $\lambda = 0.97$ was the best value.

Remember the GARCH(p,q) model from last week

The GARCH(p,q) model:

$$\sigma_t^2 = \omega + \sum_{i=1}^{p}\left(\alpha\epsilon_{t-i}^2\right) + \sum_{i=1}^{q}\left(\beta\sigma_{t-i}^2\right)$$

If we set $(p, q) = (1, 1)$then we can see Risk Metrics' Exponentially weighted methodology is a GARCH(1,1) process with $\alpha = (1 - \lambda)$and $\beta = \lambda$.

We also said that is $\alpha + \beta = 1$ that the long term variance is undefined. Do not use long time horizon simulation where you update the covariance matrix for each future time period using this methodology!

Dealing with Non-PSD Matrices

2 main reasons matrices are non-PSD.

1. Floating point error.
2. Missing Data

Floating point error and rounding can become a problem when calculating the Cholesky on a PSD matrix, sometimes you have a true PSD matrix but you end up with a slightly negative eigenvalue. That will cause one of the diagonals on the root matrix to be complex. Generally it is OK to check for and set to 0 slightly negative values before you attempt the square root. Values above well below a floating point error need to be dealt with in a different manner.

Rebonato and Jackel (1999 https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1969689) present 2 methods for finding an acceptable PSD matrix. The method using a spectral decomposition on the correlation matrix is relatively straightforward.

First, decompose the correlation matrix:

$$C\,S = \Lambda\,S \text{ with } \Lambda = diag(\lambda_i)$$

This is the eigenvalue decomposition of C into its eigenvectors $S$and a diagonal matrix with its eigenvalues, Λ.

Second, construct a new Λ'such that all eigenvalues are greater than or equal to 0

$$\Lambda' : \lambda_i' = max(\lambda_i, 0)$$

Third, construct the diagonal scaling matrix

$$T : t_i = \left[\sum_{j=1}^{n} s_{i,j}^2 \lambda_j'\right]^{-1}, \; T = diag(t_i)$$

$s_{i,j}$ is the (i,j) element of the eigenvector matrix

Now let:

$$B = \sqrt{T}\,S\,\sqrt{\Lambda'}$$

Where $\sqrt{\ }$ of a matrix denotes the element wise square root.

$$BB^T = \hat{C} \approx C$$

This matrix is not necessarily the closest matrix (by whatever method of “closest” you choose) to the original. It is, however, a good starting point and oftentimes, good enough.

Rebonato and Jackel also present an optimization algorithm in their paper. Which we will not discuss here.

Higham (2002 https://www.maths.manchester.ac.uk/~higham/narep/narep369.pdf) presents an iterative algorithm for finding the nearest PSD correlation matrix. This method is used widely.

The goal is to find A such that

$$\gamma(A) = ||A - C||$$

Is minimized, such that A is PSD

Generally, the norm used is the Frobenius Norm

$$||A||_F = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}^2$$

Weights can be applied by 2 methods, if desired. See the paper for more information.

We will assume our weights are a diagonal matrix, W. Unweighted, we can set $W = I$

$$||A||_W = \left|\left|W^{\frac{1}{2}} A\, W^{\frac{1}{2}}\right|\right|_F$$

Higham proposes Algorithm 3.3 (page 9 of the pdf, 337 of the journal). This method alternates projections until the norm converges.

The first projection,

$$P_U(A) = A - W^{-1} diag(\theta_i) W^{-1}$$

Where θ solves the equation

$$\left(W^{-1} \# W^{-1}\right)\theta = diag(A - I)$$

is element wise multiplication

When $W$is diagonal (which we are assuming), then

$$P_U(A) = (p_{i,j})$$

$$p_{i,j} = a_{i,j} \; \forall \; i \neq j$$

$$p_{i,j} = 1 \; \forall \; i = j$$

Second projection is similar to the Spectral Decomposition we performed earlier.

$$P_S(A) = W^{\frac{-1}{2}} \left(\left(W^{\frac{1}{2}} A \, W^{\frac{1}{2}} \right)_+ \right) W^{\frac{-1}{2}}$$

$$(A)_+ = S \, diag(max(\lambda_i, 0)) \, S^T$$

Ie the Eigenvalue system with negative eigenvalues set to 0.

The algorithm proceeds

$$\Delta S_0 = 0, \; Y_0 = A, \; \gamma_0 = max \, Float$$
$$Loop \; k \in 1 \ldots max \, Iterations$$
$$R_k = Y_{k-1} - \Delta S_{k-1}$$
$$X_k = P_S(R_k)$$
$$\Delta S_k = X_k - R_k$$
$$Y_k = P_U(X_k)$$
$$\gamma_k = \gamma(Y_k)$$
$$if \; \gamma_{k-1} - \gamma_k < tol \; then \; break$$

We use the alternating projections to update the Ymatrix until we max out our iterations, or the weighted norm converges to a set tolerance.

In practice, you need to check the smallest eigenvalue from the result. If it is significantly negative, you need to run the algorithm on the result. You need a result that can be processed by your Cholesky function.

Principal Component Analysis (PCA)

PCA provides us another way to simulate a system. The Rebonato and Jackel methodology used to adjust a non-psd matrix is simply PCA, removing the negative eigenvalues. PCA is often used to reduce the dimensionality of the problem.

PCA projects the system into an orthogonal space. It gives us a linear transform of independent random variables into a correlated multivariate distribution.

$$C = S \Lambda S^T$$

Where S are the eigenvectors and Λ is a diagonal matrix of eigenvalues.

If we take the square root of the eigenvalues and store those in a diagonal matrix, $\Lambda^{\frac{1}{2}}$, we can write

$$C = \left[S \Lambda^{\frac{1}{2}} \right] \left[\Lambda^{\frac{1}{2}} S^T \right]$$

$$B = \left[S \Lambda^{\frac{1}{2}} \right] \Rightarrow C = BB^T$$

This provides an alternative to the Cholesky factorization

$$X \sim N(\mu, \Sigma)$$

There exists

$$\mu \in \mathbb{R}^n,\ L \in \mathbb{R}^{nxm}\ such\ that\ X = LZ + \mu$$

Where

$$Z_i \sim N(0,1),\ \ i \in [1, m]$$

$$\Sigma = LL'$$

The Cholesky is preferred for a full rank system. Because it is a triangular matrix, less floating point operations have to be performed.

If the matrix is PSD then there will be m non-zero eigenvalues where $m \leq n$.

If the eigenvalue is 0, then we can remove it from the Λ matrix, along with the corresponding column in the eigenvector matrix S.

Now to simulate N values, we only have to generate M random numbers, effectively reducing the size of the problem.

We can further reduce the dimensionality by removing additional non-zero eigenvalues. The larger the eigenvalue, the more impact it has on the total system.

Take the vector of m non-zero eigenvalues. Sort them from largest to smallest (most eigenvalue functions do this for you) to for the vector

$$L = \left[\lambda_1, \lambda_2, \dots, \lambda_m\right] \forall \lambda_i \geq \lambda_j,\ i > j\ with\ i, j \in [1, 2, \dots, m,]$$

Then the % of variance explained by the first K eigenvalues is

$$Pct\ Explained\ (K) = \frac{\sum_{i=1}^{K} L_i}{\sum_{j=i}^{m} L_j}$$

Simulating from Models

Often we have a model for an asset, this could be OLS, MLE, ARMA, etc. At the heart of most modeling is to get to an error variable that is independent and identically distributed (iid).

Assume we have n variables, X, that we are interested in. There is some information matrix, Ω, that we use for modeling, and the error terms, ϵ, are all assumed to be normally distributed.

Models for variable at time t, $x_{i,t} \in X$ as given as a function:

$$x_{i,t} = f_i\left(\Omega_t, \epsilon_{i,t}\right)$$

Because we assume $\epsilon_i \sim N(0, \sigma_i^2)\ iid$, then we can assume the system of errors is jointly normal

$$\epsilon \sim N(0, \Sigma)$$

If we know Ω_t then we can use the set of functions, f, to simulate X.

1. Simulate m draws for n error terms
2. For each draw, evaluate the set of functions to create the X simulated variables.

If we do not know all elements of Ω_t then we need to

1. Simulate them through their own model
2. Assume a value for them

Often some x_i will depend on other elements in X. In that case we need to order the models so that the independent models are evaluated first.

1. Order models without dependencies on X first.
2. Order remaining models such that dependent models are evaluated earlier.
3. Simulate m draws for n error terms
4. Evaluate the models in order.

If you have a circular pattern, i.e. A depends on B, which depends on C, which depends on A, then you need an iterative solver. We will not deal with it in this class.