# Assignment 3.

Given October 27, due November 10.

**Objective:** Basics of Monte Carlo Sampling

**1.** Write a program that uses the Box Muller algorithm to generate samples from the standard normal distribution. Check that you have correctly programmed the algorithm in two ways:

($i$) The histogram method: choose a small, but not too small step size, $\Delta x$, and make 'bins", $B_j = [x_j - \Delta x/2, x_j + \Delta x/2]$, where the $x_j$ are uniformly spaced a distance $\Delta x$ apart. If independent random samples, $X_1, \cdots, X_N$ are taken from the probability density $f(x)$, then the number of samples in $B_j$ should be approximately $NP_j$, where $P_j$ is the probability of any particular sample landing in $B_j$:

$$P_j = \int_{B_j} f(x)dx = \Delta x f(x_j) + \frac{f''(x_j)}{24}\Delta x^3 + O(\Delta x^5) = \Delta x f(x_j) + O(\Delta x^3) \ .$$

The first approximation is more accurate but will be necessary only if the sample size is very large and the bins are also pretty large. More often, the statistical error in the number of samples landing in $B_j$ will be much larger than the error in approximating $P_j$. Make a large number of samples and plot $N_j$ and $NP_j$ on the same graph to see that they agree for large $N$. It is important to choose the parameters $N$ and $\Delta x$ so that a reasonably large number of samples fall into the most popular bins. Try to find error bars that estimate the order of magnitude of the errors between individual bin counts and the expected bin counts.

($ii$) Computing moments: If $X$ is a Gaussian random variable with mean zero, then its odd moments are all zero, while the even moments are given by

$$E[X^2 p] = (2p-1) \cdot (2p-3) \cdot \cdots \cdot 3 \cdot E[X^2]^p \ .$$

That is, all of the moments are determined by the second moment (the variance). Use your standard normals to make Monte Carlo estimates of a few nontrivial moments and see that they agree with the expected values to within expected statistical errors. Note: accurate estimates of higher moments require larger sample size. Make sure that your data illustrate this. All Monte Computations have error bars.

The Box Muller algorithm has the drawback that it generates samples two at a time, while most applications use samples one at a time. One solution to this is just to use one of the two samples. This is a good solution if the time to use the sample is much larger than the time to generate it. Another solution is to generate a large array of gaussians at the same time, and then to use samples from this array as needed, rather than calling the Box Muller routine for individual samples. This usually makes the code faster because it avoids the overhead of many procedure calls. It also makes the code more complicated: you have to make sure, whenever you enter a loop that uses samples, that there are enough samples in the array that have not yet been used. This forces the programmer to add extra code before each loop that uses samples. The efficiency gain can range from undetectable to a factor of ten, depending on the application and the platform (hardware, compiler, ...). Make sure the gaussian random number generator you end up with is modular, you will use it in most assignments from now on.

**2.** Write a program that computes the Choleski decomposition of a symmetric positive definite matrix. This program should return an error flag if the matrix is not positive definite. To test it, you need to multiply the factors and see whether you get the original matrix.

**3.** Write a program (or pair of programs) that takes as input a symmetric positive definite matrix and produces multidimensional gaussian random variables with that covariance matrix. The algorithm is to take the Choleski decomposition of the covariance matrix and use Box Muller to make independent samples. Your routine should return an error message if the covariance matrix given to it is not positive definite. The Choleski factorization should be done only once. You can test this routine by checking that the individual components of the output samples are gaussian random variables, or even by testing that pairs are jointly gaussian, by a two dimensional binning procedure in which the bins are little squares instead of little intervals. A more serious test is the multidimensional version of test (*ii*) above, which is the identity

$$E(X_1X_2X_3X_4) = E(X_1X_2)E(X_3X_4) + E(X_1X_3)E(X_2X_4) + E(X_1X_4)E(X_2X_3) \ .$$

This identity is true whenever the $X_j$ are components of a multivariate normal with mean 0. If $X_1 = X_2 = X_3 = X_4$, it is the old identity for univariate gaussians: $E[X^4] = 3E[X^2]^2$. Try your program on several matrices of various dimensions, including the one for part 4.

**4.** Now use the above to analyze a portfolio of puts. There are $n$ assets whose prices at some time in the future will have a joint lognormal distribution. This means that the asset prices are

$$S_j = e^{X_j} \ ,$$

Where the $X_j$ are components of a multidimensional gaussian. The data set for this assignment has the information about the portfolio to analyze. Generate a large number of sample stock prices and deduce the probability distribution of $V$, the total value of the portfolio, by making a histogram of the sample values. Indicate error bars. The program should read the data from a file (or several files).