1. Large angle simple pendulum

The code is constructed in two separate sections. The first section is the C++ implementation. The C++ code is constructed to be able to use fourth order Runge-Kutta and Euler's method on any set of two coupled ODEs. The program can also generate a sequence of N points from an arbitrary numerical method and arbitrary set of coupled ODEs. The main method utilizes these abstract functions, with the simple pendulum set of equations, and produces the sequence of points up to 10 seconds with dt = 0.01, 0.0001 . The data that is produced is four files, two of which are the undamped system with the two numerical methods. The other two files, are from the damped system with, again, the two numerical methods. The sets of data for the different dt values are held in the same file. The files are then parsed by the second part of the code, the Matlab section. The Matlab code separates the data, where necessary, and produces plots for Angle vs Time, Angular Velocity vs Time, Angle vs Angular Velocity, and Energy vs Time. This is done for each dt in both damped and undamped systems. The two numerical methods are superimposed on each plot. The absolute error, or the absolute difference between the estimated and expected values for angular velocity and energy, are also plotted against time. The error is plotted for the two systems, damped and undamped, and each dt. According to the error plots, the error on the estimations from both numerical methods are sinusoidal. Euler's method had overall a much better performance in the undamped system, while the Runge-Kutta method outperformed Euler's in the damped system.

2. Random Numbers

The code for this section is a simple Matlab implementation for Hit and Miss Monte Carlo integration. The program runs a variable number (100) independent Monte Carlo with N = 10000 test points. Each run uses a separate seed to ensure that runs are independent. The program also produces several plots. The first plots produced, are two sets of histograms that consist of all of the test points, and all of the hits. The numbers generated from the built-in RNG in Matlab look sufficiently uniform, so the mean and rms are expected to both be 0.5, since x, y, and z are generated between 0 and 1. To ensure independence between the variables, the correlation coefficients between each possible set of two variables, for each run, are placed in a histogram. The correlation coefficients appeared to be Gaussian distributed centered at 0, therefore the variables are uncorrelated. The distribution for r = sqrt(x^2 + y^2 + z^2) is also histogrammed with every set of hit points. As r approaches 1 the frequency of points that sum to corresponding r value increase. This is reasonable because as r increase, the volume of the sphere increases, so more sets of points are need to fill this larger volume, hence higher frequency of x,y,z values produce r=1. The estimated value for π is 3.1451 ± 0.0040 whics is gathered from the mean π value over 100 runs with error combined by the total differential for the same 100 runs. To reduce the error by an order of magnitude the number of points need to be increased. If the error for Hit and Miss MC is given by Ve*sqrt(Nh- Nh^2/N)/N where Ve is the Volume enclosed (equal to 8), Nh is number of hits and N is number of points then by roughly estimating Nh is approximately 0.5N. Then the error is approximately (Ve/2)*sqrt(N)/N. This rough approximation yields 0.04 at N = 10000. Solving this equation for an order of magnitude less of error, we find N = 1000000.