Practical Lab Numerical Computing
Computational Finance
Bachelor-Worksheet 1

Lukas Troska, Ilja Kalmykov

Task 1

The program outputs uniformly distributed random numbers between 0 and 1 in two ways, once by the built-in c++-function rand, and once using gsl.
Removing (double) in the marked line leads to the result being an int (and thus either 0 or 1, mostly 0), since dividing an int by an int in c++ gives an int, so we have to cast one of the variables to double to force a floating point number as the result.
There is a direct function for simulating normally distributed random variables: double gsl_ran_gaussian(const gsl_rng* r, double sigma), which returns a gaussian random variable with mean 0 and variance sigma.

Task 2

For code see task2.cpp.

Task 3

The matlab script plots the data against a unit gaussian probability density function. It first creates a histogram of the data, where it sorts the numbers into 100 different bins, and then counts the amount of numbers in each bin. Then it scales the amount of these numbers to show the relative amount in each bin and plots it against the unit gaussian probability density function.
The problem in fig. 1 is that $[-2, 2]$ is not a good enough bound since the integral comes out to ≈ 0.95, thus we get a lot more variables where $y \leq p(x')$ (since it its more likely for an element of $[-2, 2]$ then say $[-3, 3]$ to fullfill that), and thus the simulated density has a higher mean than it should be.

Task 4

We get a normally distributed random variable if we apply the inverse of the cdf of the gaussian to the uniform random variable.

Proof. Let U be a standard uniform random variable, F like in the problem. Then $P[X <= x] = P[F^{-1}(U) <= x] = P[U <= F(x)] = F(x)$ (2nd equality holds because F is right-continuous, 3rd equality holds because U is uniform on [0,1]) $\square$

Intuitively this is clear, too: We uniformly choose a number between 0 and 1 and interpret that as a proportion of the area under the gaussian curve. Then we return the number that has this proportion of the area on the left of it. Thus, we are unlikely to choose a number in the tails, since there is little area in the tails, so we would have to get pick a number very close to 0, whereas for the other areas we have more "room" so we are more likely to get these.

Task 5

Firstly, the cdf of the gaussian is pointsymmetric w.r.t. the y-intercept, thus we can handle the case $x < 0$ by $1 - NormalCDF(-x)$. If $x > 6$, we know that a normally distributed random variable is guaranteed to be smaller, thus we return 1. For the other 2 cases: for $0 \leq x \leq 1.87$, its easy to see that the term given in the return is very close to the cdf of the gaussian since it behaves almost like a polynomial in that area. For the last case the values of the cdf are very close to 1, and again like above the given term gives us a good approximation of the difference of the cdf's value to 1 in that area.

Task 6

For code see task6.cpp and task6.plot for gnuplot script.

Task 7

Let $g_1(u_1, u_2) = \sqrt{-2\log(u_1)}\cos(2\pi u_2), g_2(u_1, u_2) = \sqrt{-2\log(u_1)}\sin(2\pi u_2)$ be the transformation (denoted as z_1, z_2 in the problem). Solving for u_1, u_2 gives us $u_1 = \exp(-(z_1^2 + z_2^2)/2), u_2 = (1/2\pi)\tan^{-1}(z_2/z_1)$. The joint distribution of z_1, z_2 is described by $f(z_1, z_2) = f_{u_1,u_2}(g_1^{-1}(z_1, z_2), g_2^{-1}(z_1, z_2)) * |J(g)|$ where $J(g)$ is the Jacobion matrix of the transformation. Plugging both in we get $f(z_1, z_2) = (\exp(-({z_1}^2 + {z_2}^2)/2))/(2\pi) = -(\exp(({z_1}^2/2))/\sqrt{2\pi} * \exp(({z_1}^2/2))/\sqrt{2\pi}$. We see that z_1, z_2 are independent and standard normally distributed.

Task 8

The advantage of this algorithm is that it is more precise, we don't have a lot of floating point errors. The variable alpha in the algorithm is the estimated mean. It is the same as the naive computation because: Let x_n be the n-th value, $\hat{x}_n$ be the n-th estimated mean value. Obviously $\hat{x}_0 = x_0$ so the initialization is correct. The computation is correct, too, since in the terms above we have for the variables in the algorithm: $gamma = x_n - \hat{x}_{n-1}$, $alpha = x_{n-1}$ where $n = i+1$, so we have $alpha = alpha + gamma/(i+1) = \hat{x}_{n-1} + (x_n - \hat{x}_{n-1})/n = ((n-1)\hat{x}_{n-1} + x_n)/n = (x_1 + ... + x_2)/n$. This shows that it yields the same result.
!!!!!ADD VARIANCE CORRECT RESULT!!!!!

Task 9

For code see task9.cpp and task9.plot for gnuplot script. !!!!!ADD WHAT DOES PLOT MEAN!!!!!

Task 10

For code see task10.cpp and task10.plot for gnuplot script. The paths look more or less the same, also the mean pricechange is ineed +10 %/year.