Sampling from a Normal distribution

Most (if not all) programming languages have random numbers generator functions that produce *uniformly* distributed values. What about random numbers from an arbiutrary, *non-unform* distribution?

In this post we'll explore several methods to generate random numbers from a *Normal* distribution. To simplify things a bit we'll work with a *standard* Normal distribution. This distrubtion has a mean $\mu = 0$, and standard deviation $\sigma = 1$. This choice doesn't limit us in any way because there's a one-to-one mapping between the standard Normal distrubtion and any other normal distribution wiht the mean of μ and variance of σ:

$$x = x_{starndard} * \sigma + \mu$$

Method 1: Central Limit Theorem

Central limit theorem states that the mean of a *sum* of samples from *any* distribution approaches *Normal* as the size of the sample increases. This is really cool. Regardless how our original values are distributed, if we sum them up, we'll get an (approximate) Normal distribution!

As little as ten elements in a sample is sufficient to approximate the normal distrubution quite closely. Here's the plan:

1. Generate 10 samples from our uniform random numbers generator, and add them up.
2. Repeate this process 10,000 times to obtain 10,000 *normally* distributed samples.
3. Plot our distribution and compare it to the theoretical Normal disbribution to see if get the right result.

Below is the annotated code in **R** language. You can easily re-implement this in any other language.

The number of samples per iteration is 10:

```
sample_n = 10 # We will draw samples form the uniform distribution
```

The number of values from a Normal distibution:

```
n = 10000 # The number of normal samples
```

Generate 10 x 10,000 total random numbers from a *uniform* distribution $[0, 1]$:

```
uniform_samples = runif(n = sample_n * n, min = 0, max = 1)
```

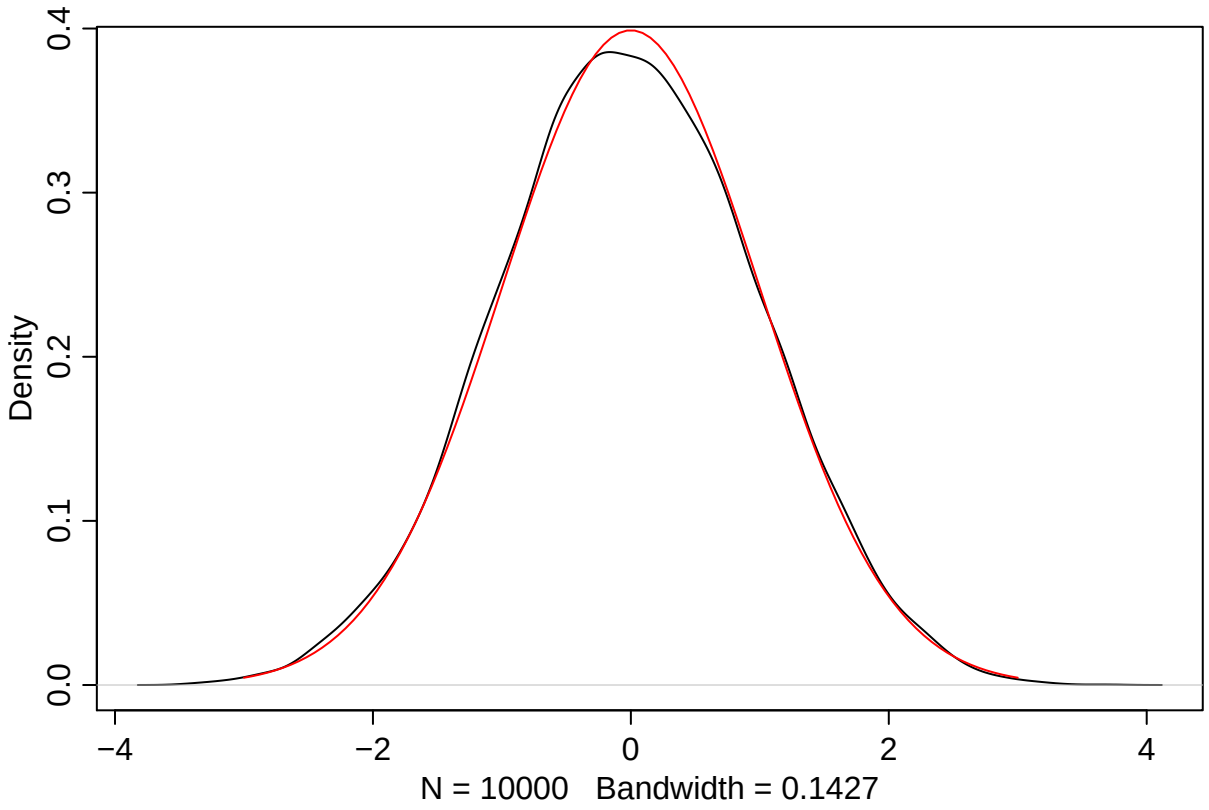
Reshape our array into a matrix of 10,000 rows with 10 elements in each row:

```
samples_matrix <- matrix(uniform_samples, nrow=n, byrow = T)
head(samples_matrix, n = 2)
```

```
##             [,1]      [,2]      [,3]       [,4]      [,5]      [,6]
## [1,] 0.08644357 0.5305361 0.2785668 0.25620633 0.1178065 0.6261371
## [2,] 0.65626758 0.1563880 0.6503585 0.01151149 0.5689057 0.2366750
##            [,7]      [,8]      [,9]      [,10]
## [1,] 0.9804603 0.1937004 0.6418638 0.09810426
## [2,] 0.9238834 0.4781249 0.3475145 0.72908542
```

```
x <- rowMeans(samples_matrix)
# dens(samples)
mu = 0.5
sigma = sd(uniform_samples) / sqrt(sample_n)
normal_x = (x - mu) / sigma
```

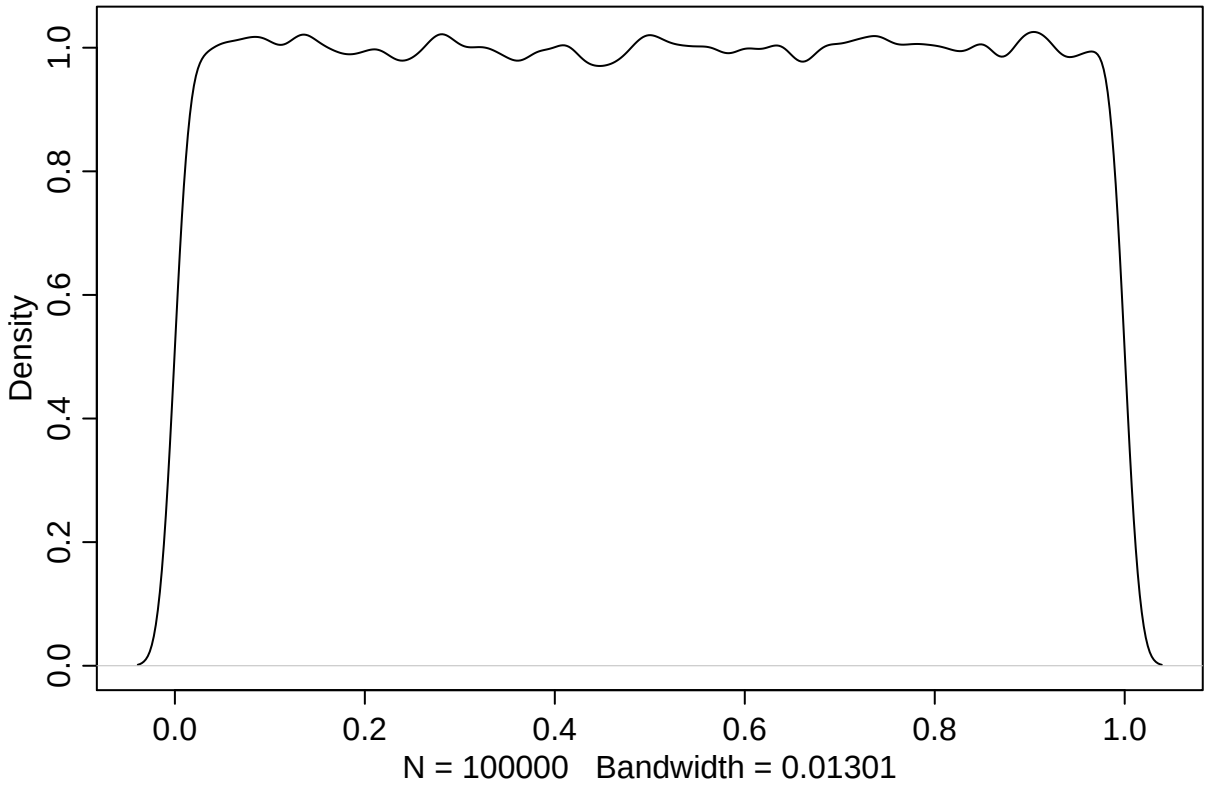
```r
dens(normal_x, adj = 1)
true_normal.x <- seq(-3, 3, length.out = 100)
true_normal.y <- dnorm(true_normal.x, mean=0, sd=1)
lines(true_normal.x, true_normal.y, col="red")
```



Inverse Transform Sampling

Inverst transform sampling relies on the fact, that if X a random variable, F is its cumulative distrubtion function (i.e. $F(x)$ is the probability of the value $\leq$ x) then $F(x)$ is distributed *uniformly*. Let's test this statement:

```r
inverse_transform <- function () {
  samples <- rnorm(n = 100000, mean=0, sd=1)
  cdf <- pnorm(samples)
  dens(cdf)
}
inverse_transform()
```



Indeed, a comulative distrition function is approximately uniform. So, if we know a CDF F for a *Normal* distribution, we can generate sample from a Normal distribution via the following steps:

1. Generate samples u from the *uniform* distribution
2. Calculate $x = F^{-1}(u)$, where F is the CDF for the Normal distribution. Then x will be normally distriubted.

However, this method is not very efficient for continuous cases where the CDF doens't have an analytic integral. Normal distribution is such an example. Because of this, other methods are more popular.

Box-Muller Transform

```
eps = .Machine$double.eps
two_pi = 2 * 3.141592653589793
n = 10000
uniform_samples = runif(n = n * 2, min = 0, max = 1)
samples_matrix <- matrix(uniform_samples, nrow=n, byrow = T)
head(samples_matrix, n = 2)
```

```
##            [,1]      [,2]
## [1,] 0.97254172 0.9363079
## [2,] 0.01197872 0.8092341
```

```
z0 = sqrt(-2 * log(samples_matrix[,1])) * cos(two_pi * samples_matrix[,2])
# z1 = sqrt(-2 * log(uniform_samples[,1])) * sin(two_pi * uniform_samples[,2])
```

```r
dens(z0, adj = 1)
true_normal.x <- seq(-3, 3, length.out = 100)
true_normal.y <- dnorm(true_normal.x, mean=0, sd=1)
lines(true_normal.x, true_normal.y, col="red")
```