Day 3 - Probability

David Carlson

September 30, 2020

Probability distributions in R

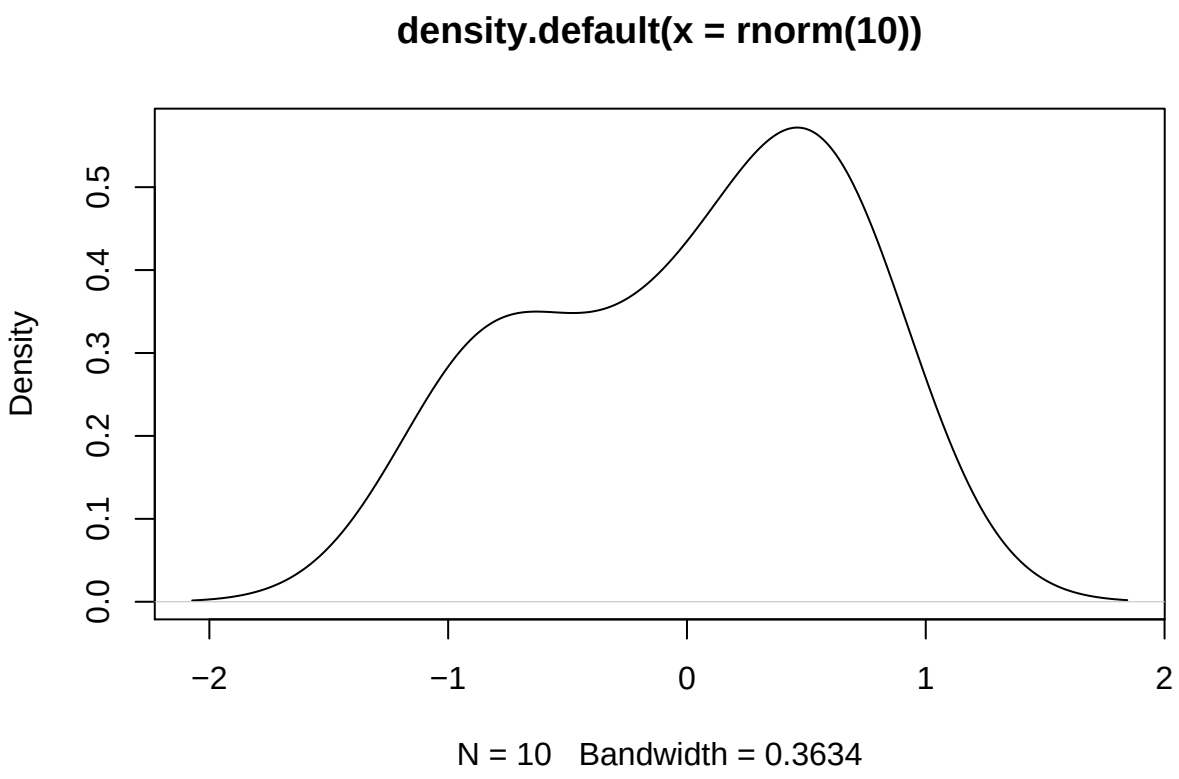
```
#always set a seed when simulating data
set.seed(1454)
#draw from a standard normal
rnorm(1)
```

```
## [1] -0.9848784
```

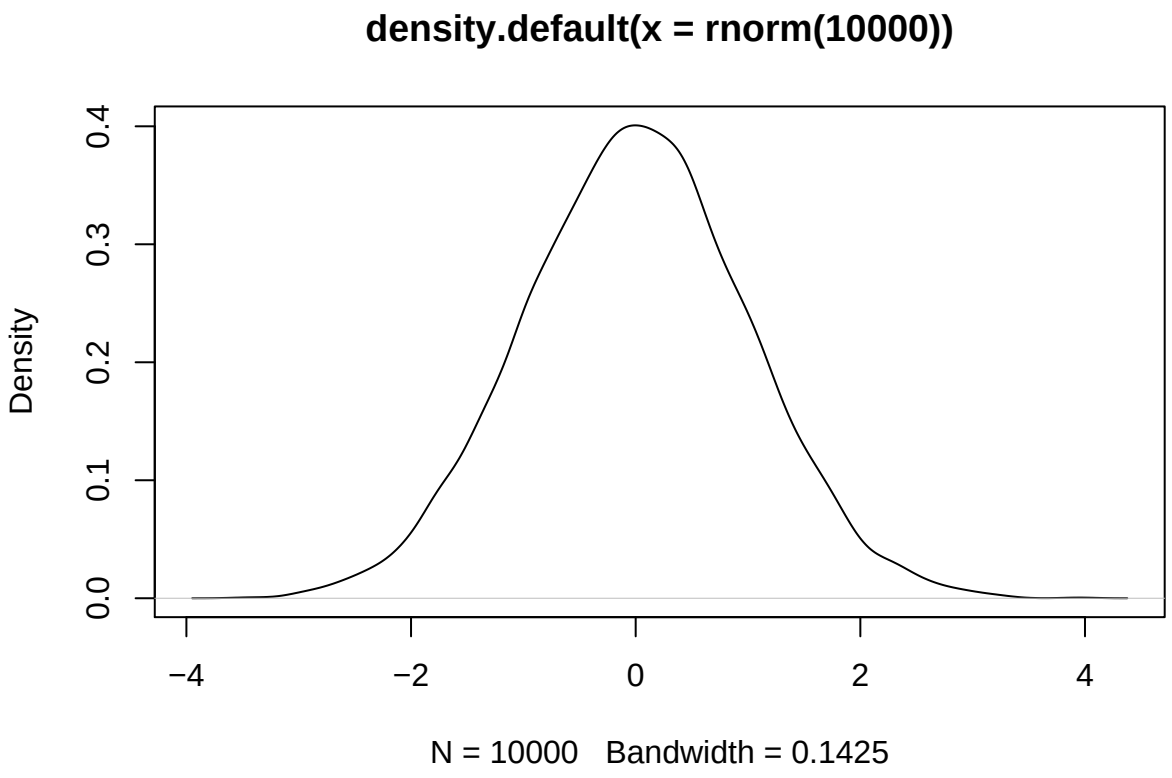
```
#draw 10 from a standard normal
rnorm(10)
```

```
##  [1]  1.3013654  0.4894259 -1.2458497  0.5565725  0.3029823 -0.4876606
##  [7]  0.8460788  1.3303218  1.5572667  0.5571490
```

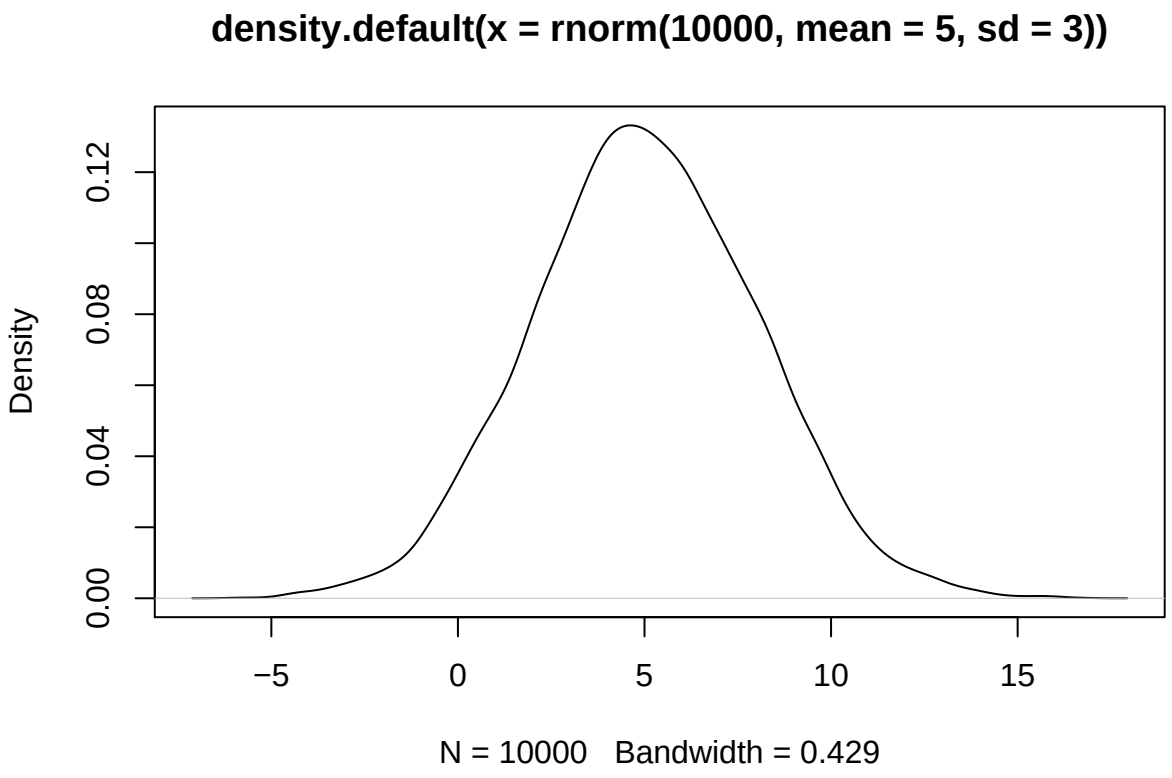
```
#plot a density
plot(density(rnorm(10)))
```




```
plot(density(rnorm(10000)))
```




```
#change the mean and standard deviation
plot(density(rnorm(10000, mean = 5, sd = 3)))
```




```
#normal cumulative density function
pnorm(-Inf)
```

```
## [1] 0
```

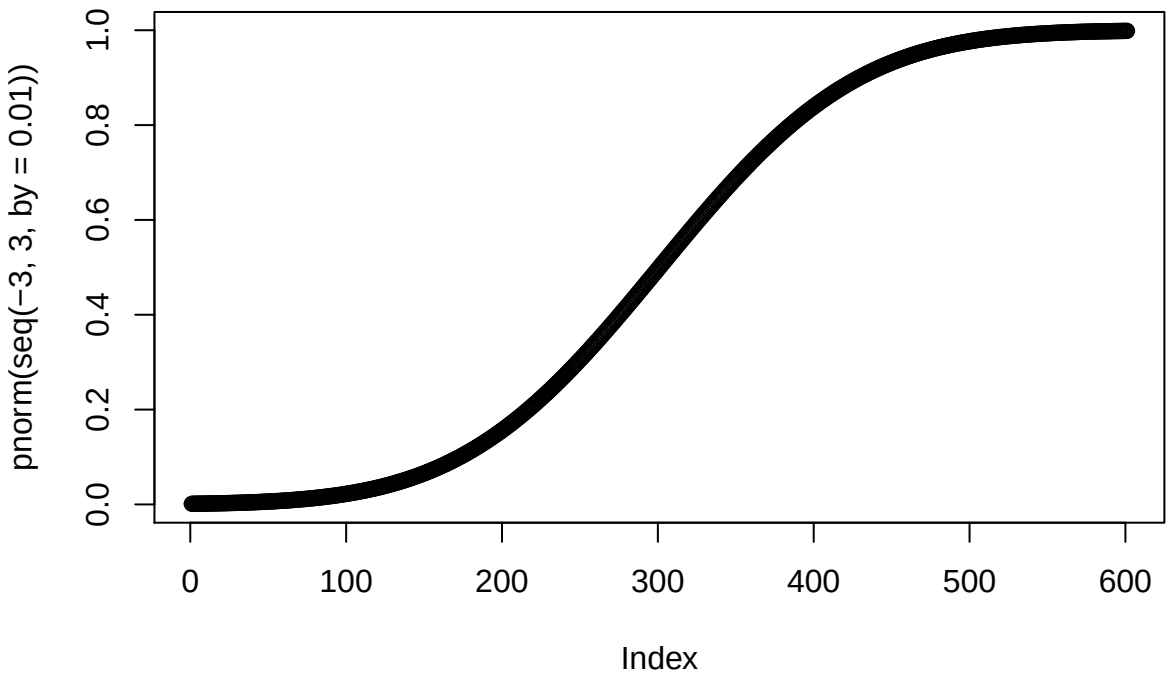
```
pnorm(Inf)
```

```
## [1] 1
```

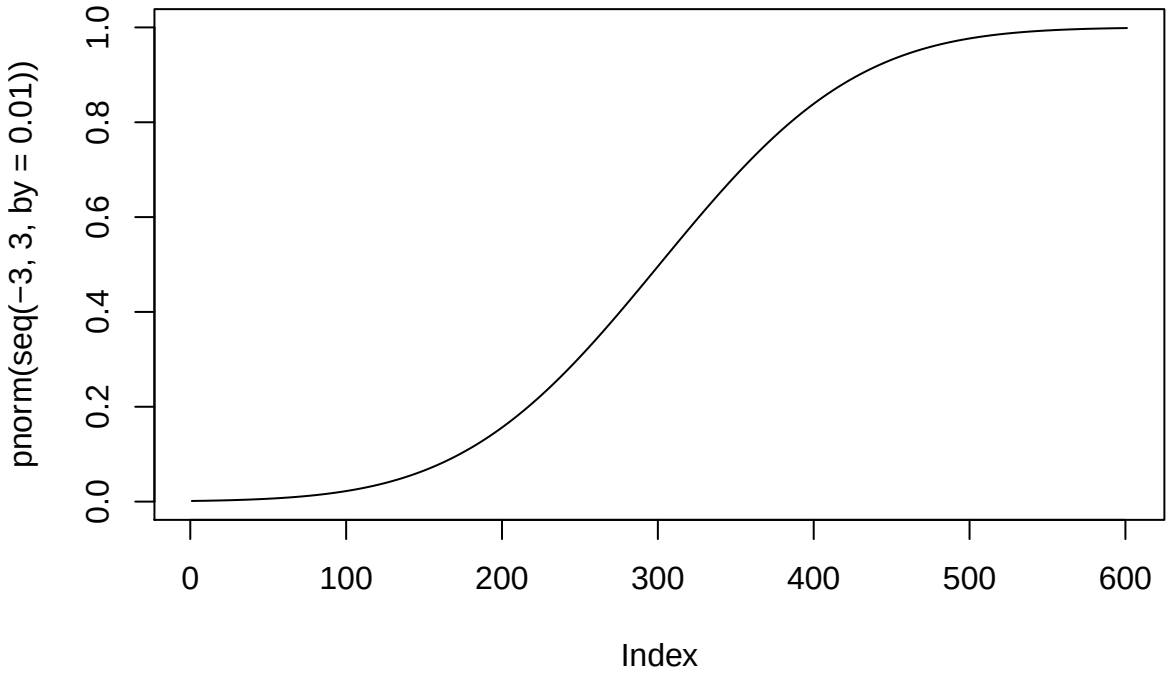
```
pnorm(0)
```

```
## [1] 0.5
```

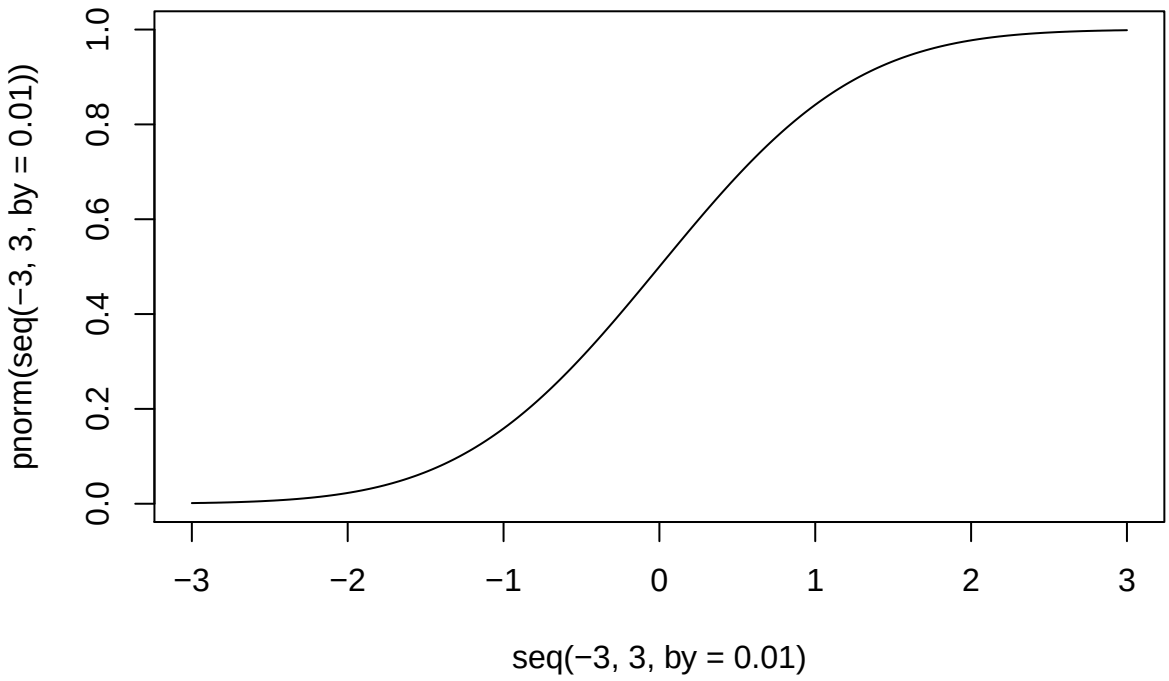
```
plot(pnorm(seq(-3, 3, by = .01)))
```




```
plot(pnorm(seq(-3, 3, by = .01)), type = 'l')
```



```r
plot(seq(-3, 3, by = .01), pnorm(seq(-3, 3, by = .01)), type = 'l')
```




```r
#normal probability density function
dnorm(-Inf)
```

```
## [1] 0
```

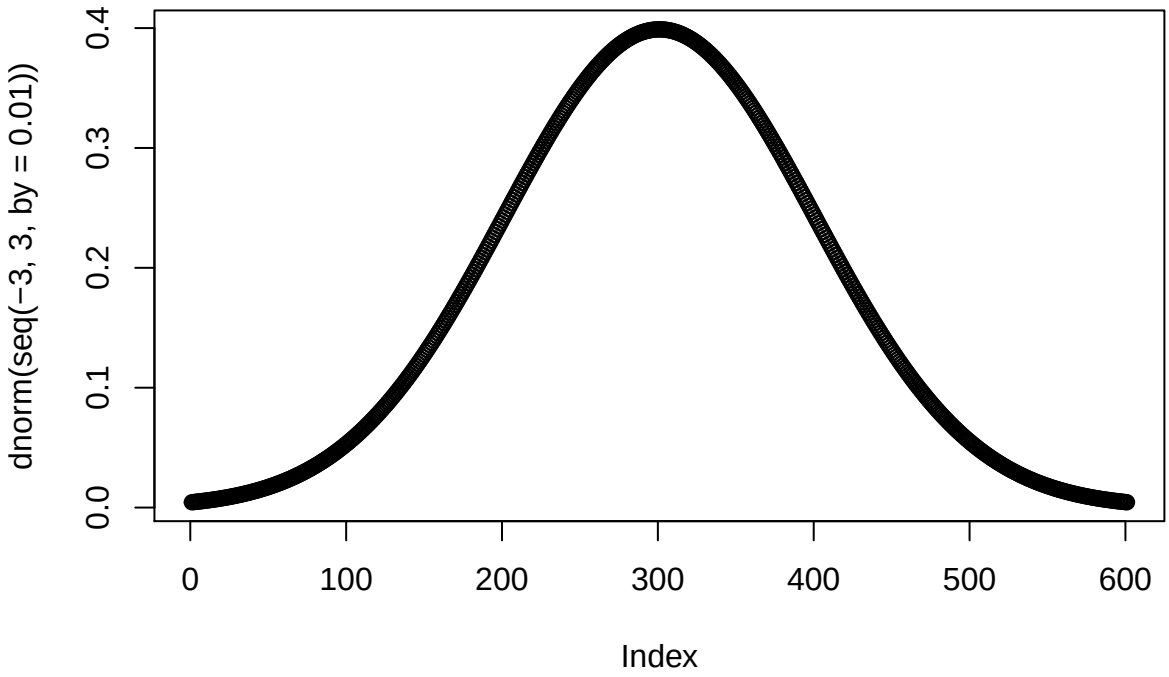
```r
dnorm(Inf)
```

```
## [1] 0
```

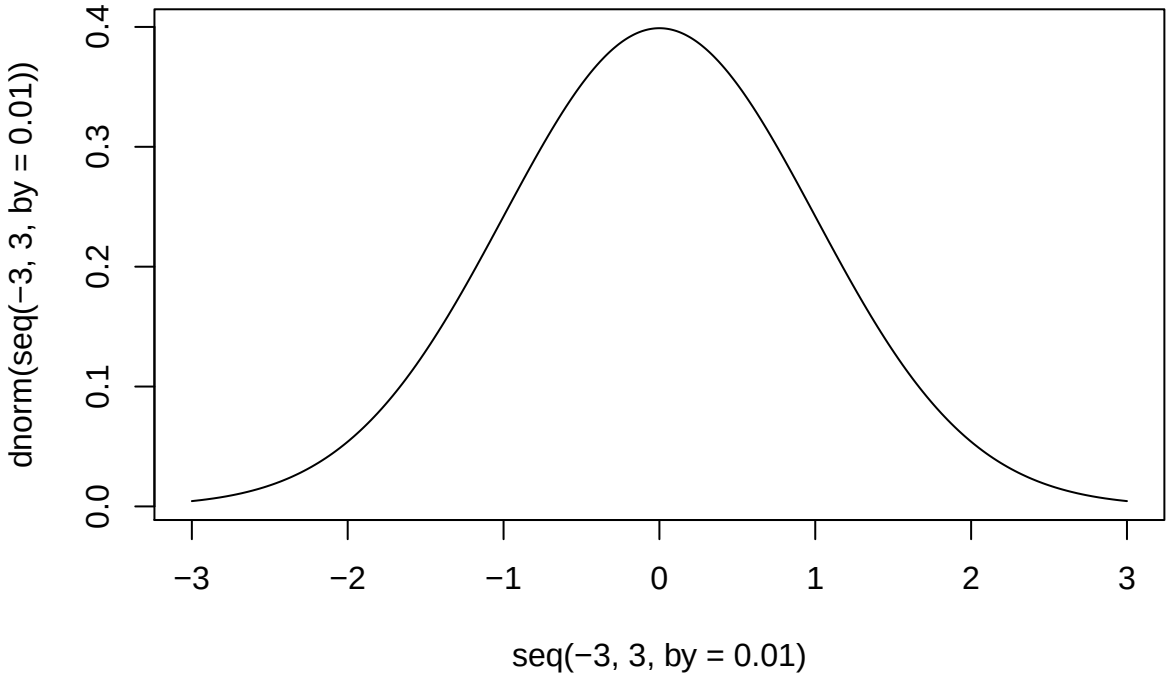
```r
dnorm(0)
```

```
## [1] 0.3989423
```

```r
plot(dnorm(seq(-3, 3, by = .01)))
```




```
plot(seq(-3, 3, by = .01), dnorm(seq(-3, 3, by = .01)), type = 'l')
```




```
#normal inverse CDF
qnorm(.5)
```

```
## [1] 0
```

```
qnorm(.025)
```

```
## [1] -1.959964
```

```r
qnorm(1 - .025)
```

```
## [1] 1.959964
```

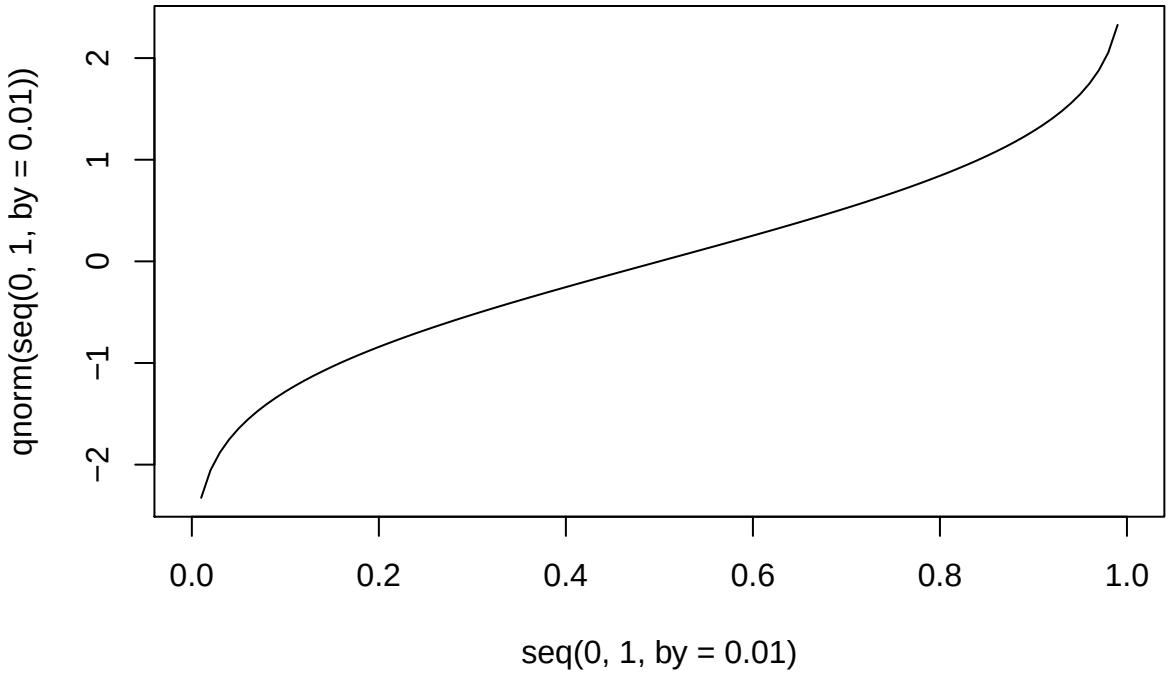
```r
qnorm(.025, lower = TRUE)
```

```
## [1] -1.959964
```

```r
qnorm(.025, lower = FALSE)
```

```
## [1] 1.959964
```

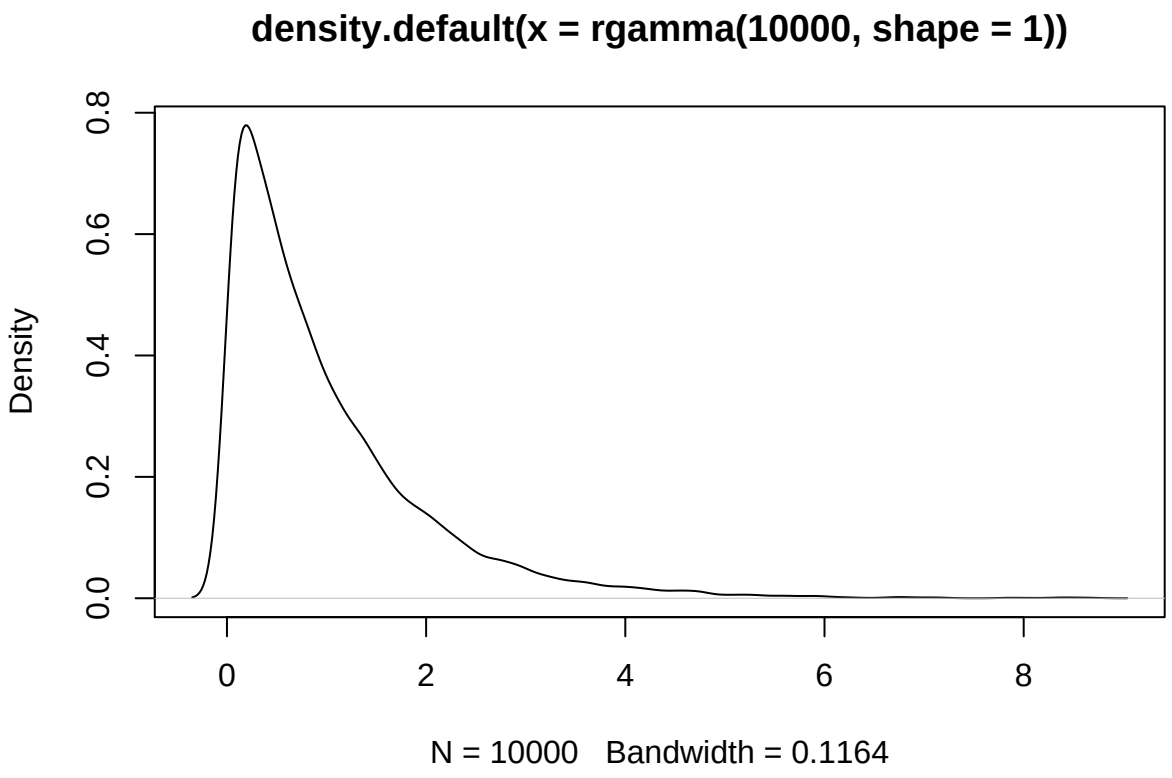
```r
plot(seq(0, 1, by = .01), qnorm(seq(0, 1, by = .01)), type = 'l')
```



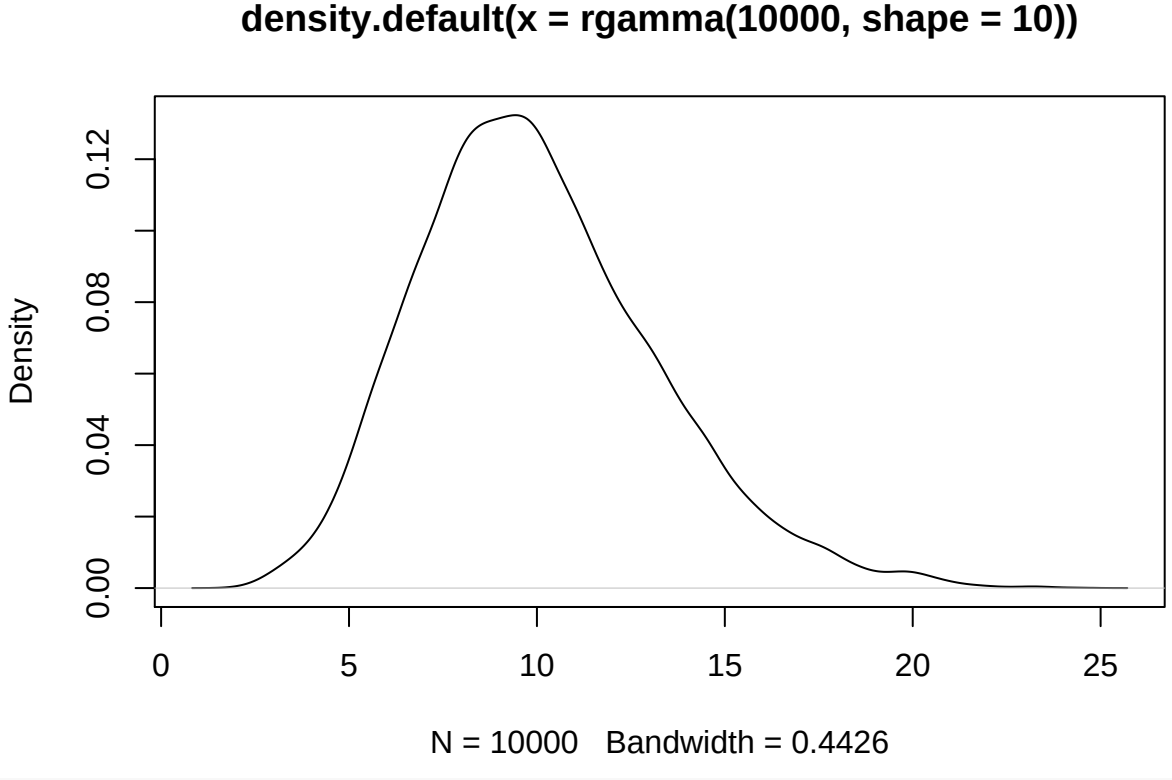

```r
#gamma pdf
rgamma(1, shape = 1)
```

```
## [1] 1.742631
```

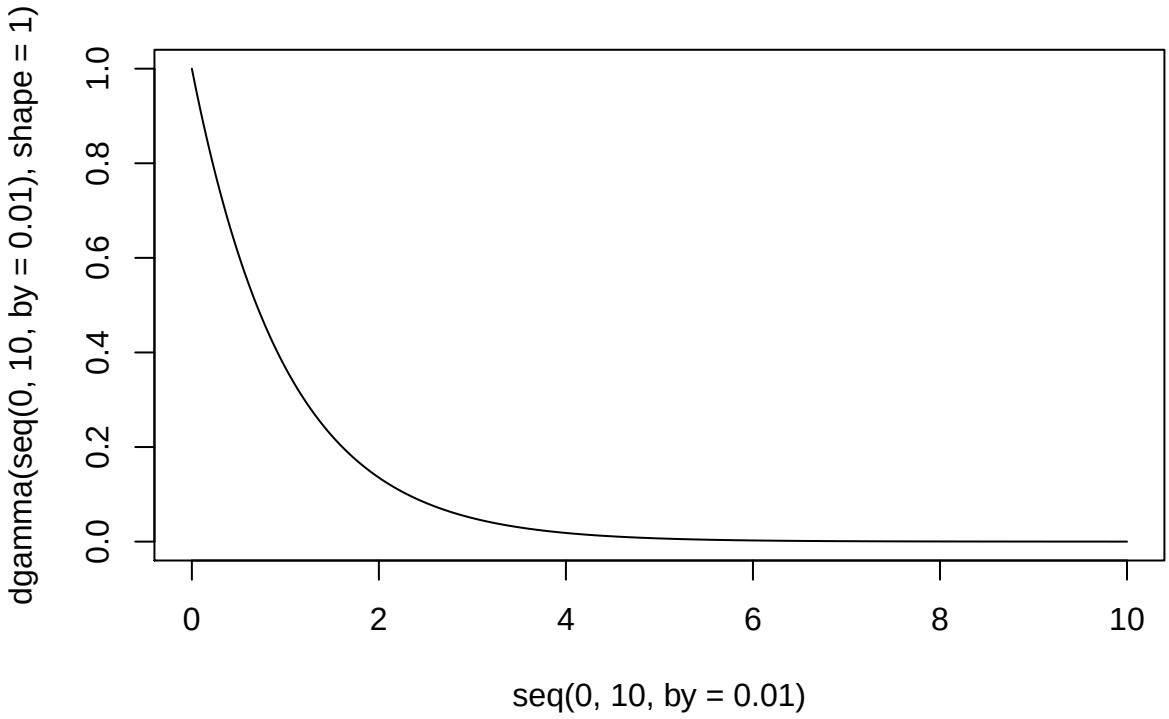
```r
plot(density(rgamma(10000, shape = 1)))
```



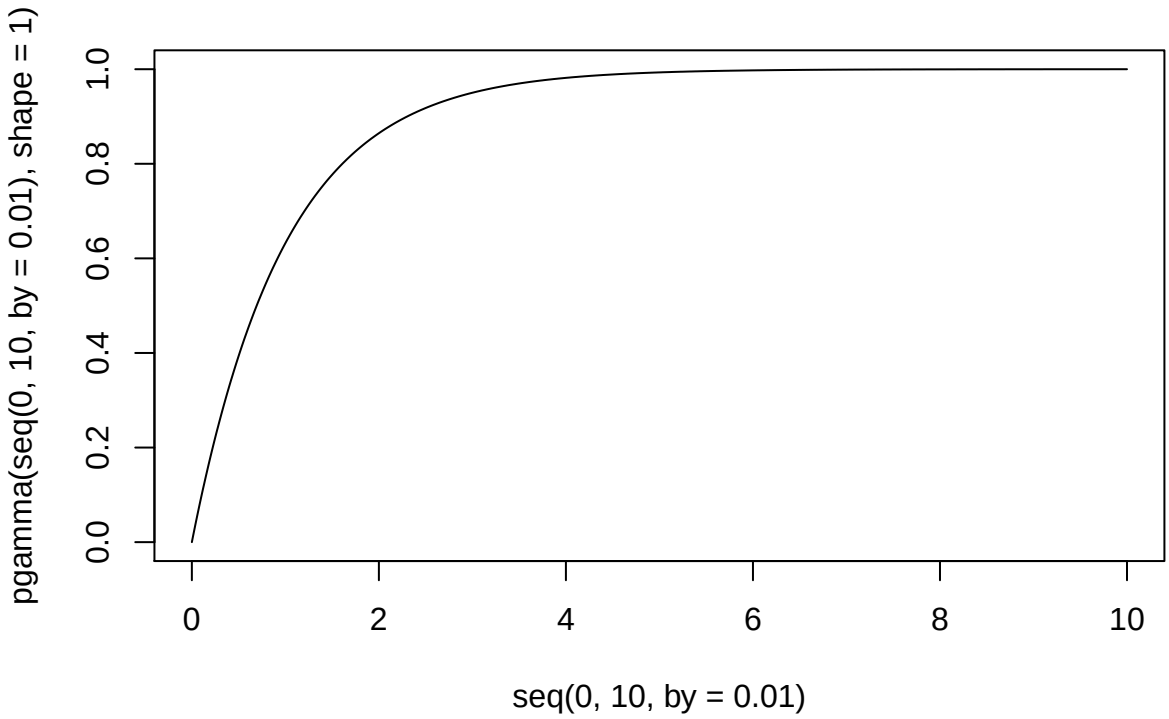

```
plot(density(rgamma(10000, shape = 10)))
```




```
plot(seq(0, 10, by = .01), dgamma(seq(0, 10, by = .01), shape = 1), type = 'l')
```




```
#gamma CDF
plot(seq(0, 10, by = .01), pgamma(seq(0, 10, by = .01), shape = 1), type = 'l')
```




```
#binomial
rbinom(1, 1, .5)
```

```
## [1] 0
```

```r
rbinom(1, 10, .5)
```

```
## [1] 4
```

```r
rbinom(n = 100, size = 10, prob = .5)
```

```
##   [1] 6 4 6 4 6 4 4 7 5 6 4 3 7 5 6 6 6 5 5 6 4 7 4 4 4 5 8 6 7 4 4 6 4 5 7
##  [36] 4 5 9 5 5 6 7 4 5 6 6 8 5 5 5 7 7 4 5 2 5 3 7 6 5 6 5 4 7 3 6 5 8 4 7
##  [71] 4 3 6 6 4 7 6 5 5 4 5 7 0 4 5 7 5 5 5 2 2 9 8 8 4 6 4 4 5 3
```

```r
#sampling
sample(c(0,1), 1, prob = c(.8, .2))
```

```
## [1] 0
```

```r
sample(c(0,1), 100, prob = c(.8, .2), replace = TRUE)
```

```
##   [1] 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 1
##  [36] 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0
##  [71] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0
```

```r
#uniform
runif(100)
```

```
##   [1] 0.24329472 0.83671998 0.19993026 0.67174908 0.04708924 0.49977218
##   [7] 0.39363344 0.60944976 0.48048680 0.30520916 0.65524787 0.94769823
##  [13] 0.88724751 0.01428664 0.24206072 0.28918967 0.90234534 0.73096361
##  [19] 0.28136751 0.60977420 0.95794236 0.98312298 0.30277008 0.51513113
##  [25] 0.28803182 0.22405375 0.99650727 0.11826794 0.83355380 0.36806097
##  [31] 0.60602271 0.15750850 0.04407260 0.41392699 0.18467254 0.98493290
##  [37] 0.92562664 0.89230193 0.02374212 0.90231263 0.99520843 0.47273060
##  [43] 0.84027909 0.52687399 0.19427264 0.06157091 0.72167613 0.11659614
##  [49] 0.99972435 0.37877370 0.52688928 0.18151592 0.37341059 0.02425044
##  [55] 0.09815329 0.60189464 0.76838850 0.28788646 0.27790065 0.87835069
##  [61] 0.49533298 0.92051069 0.91676224 0.87609818 0.97603850 0.13167414
##  [67] 0.96524507 0.02690613 0.65313227 0.71184514 0.16784128 0.69751479
##  [73] 0.40568772 0.61554615 0.61524338 0.27912363 0.52438021 0.70159912
##  [79] 0.36559440 0.40081820 0.43093819 0.06140271 0.66579625 0.63121085
##  [85] 0.94919607 0.05169840 0.68355837 0.72646127 0.42291675 0.66419573
##  [91] 0.54098281 0.42949923 0.98342616 0.28581195 0.99558333 0.07893109
##  [97] 0.65995488 0.56075492 0.65206422 0.74099463
```

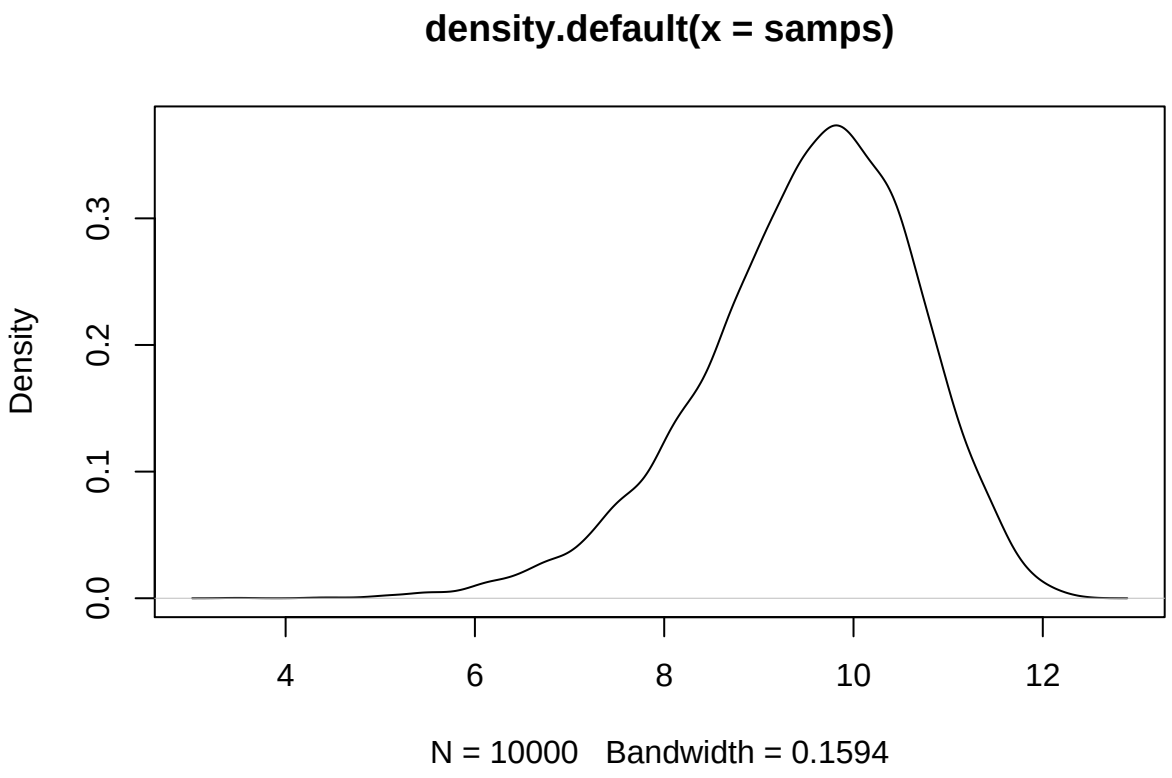
```r
#Weibull
rweibull(1, shape = 1, scale = 1)
```

```
## [1] 0.2590144
```

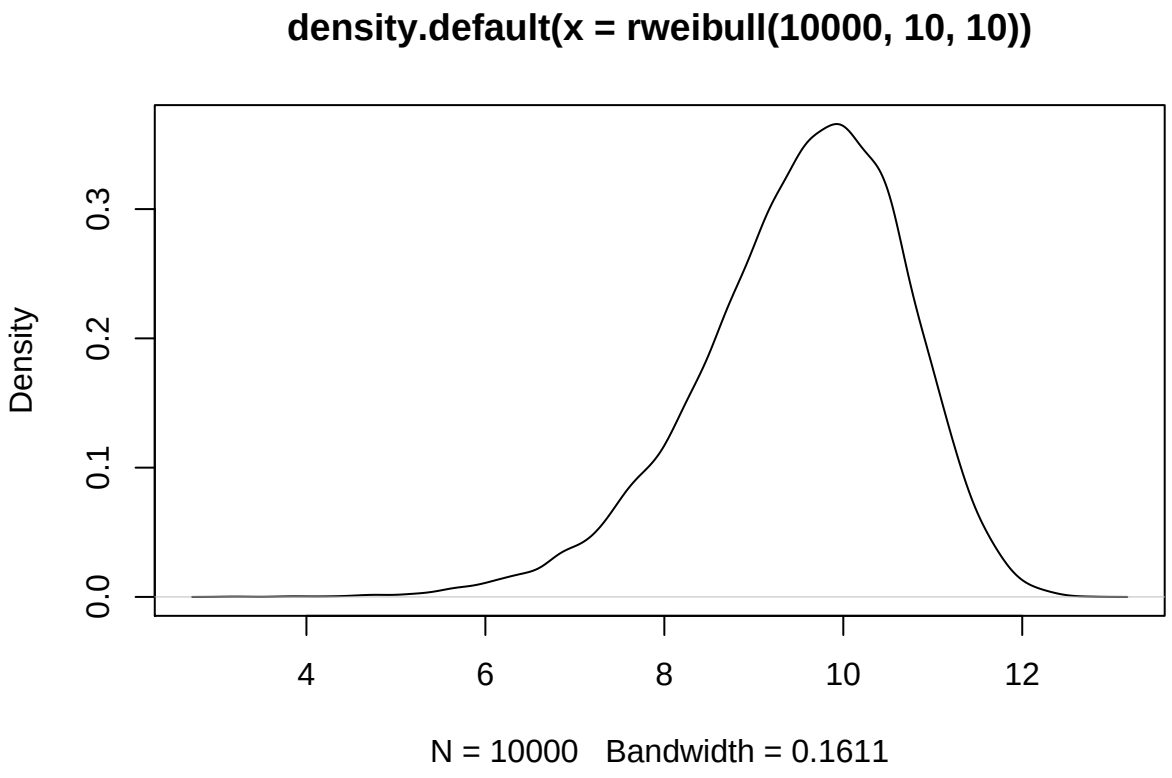
```r
rweibull(1, shape = 10, scale = 10)
```

```
## [1] 9.883106
```

```r
#sample from uniform to generate any distribution using the inverse CDF
u = runif(10000)
samps = qweibull(u, 10, 10)
plot(density(samps))
```




```
plot(density(rweibull(10000, 10, 10)))
```



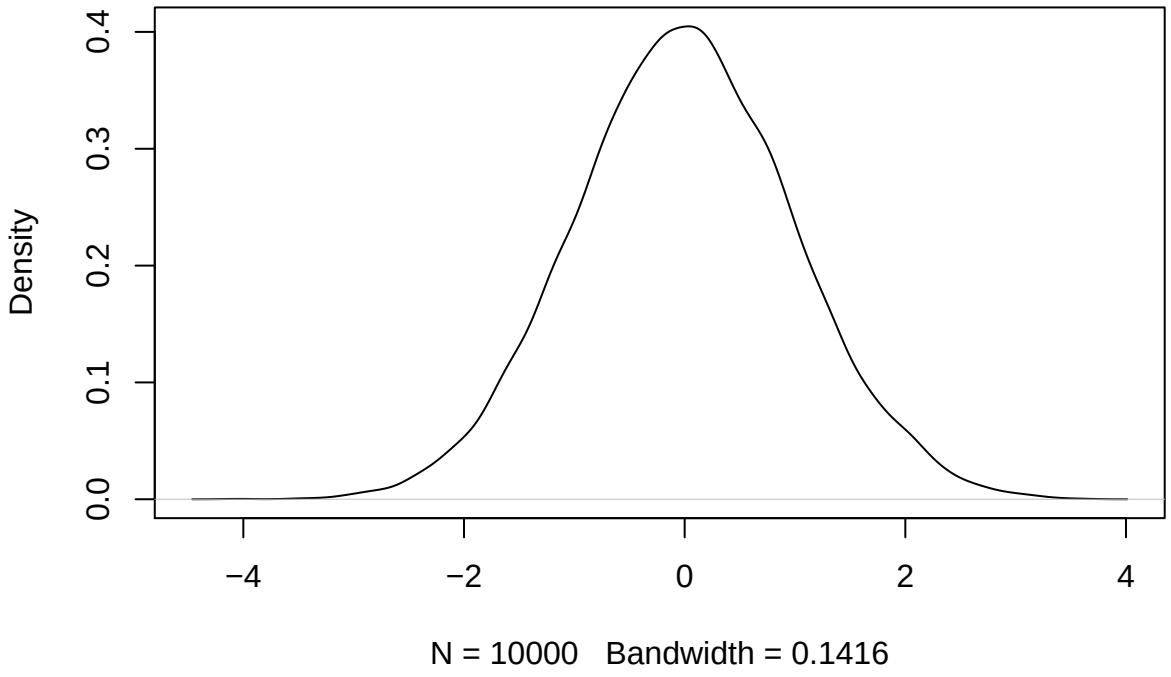

```
plot(density(qnorm(u)))
#generate random variance-covariance matrices
rWishart(1, 10, toeplitz((10:1)/10))
```

```
## , , 1
##
##            [,1]      [,2]      [,3]     [,4]     [,5]      [,6]      [,7]
##  [1,] 11.942067 10.697544 10.189947 7.976558 5.307215  7.000456  5.584695
##  [2,] 10.697544 11.151426 10.526906 7.665823 4.784080  8.357282  7.241122
##  [3,] 10.189947 10.526906 10.890732 8.021565 5.255659  8.673410  8.115783
##  [4,]  7.976558  7.665823  8.021565 7.296160 6.141679  7.960111  6.948293
##  [5,]  5.307215  4.784080  5.255659 6.141679 6.905713  7.603392  6.081731
##  [6,]  7.000456  8.357282  8.673410 7.960111 7.603392 13.463694 12.503391
##  [7,]  5.584695  7.241122  8.115783 6.948293 6.081731 12.503391 13.474270
##  [8,]  2.180164  3.865037  4.996129 4.033035 3.965388  9.353522 10.649708
##  [9,]  2.234241  3.722916  4.591607 2.545943 1.940103  6.929751  9.253341
## [10,]  2.460956  3.995733  5.520931 3.483899 2.756654  8.423246 11.372730
##            [,8]      [,9]     [,10]
##  [1,]  2.180164  2.234241  2.460956
##  [2,]  3.865037  3.722916  3.995733
##  [3,]  4.996129  4.591607  5.520931
##  [4,]  4.033035  2.545943  3.483899
##  [5,]  3.965388  1.940103  2.756654
##  [6,]  9.353522  6.929751  8.423246
##  [7,] 10.649708  9.253341 11.372730
##  [8,]  9.465455  8.919530 10.608118
##  [9,]  8.919530 11.942624 12.384666
## [10,] 10.608118 12.384666 14.164939
```

```
sig = rWishart(1, 10, toeplitz((10:1)/10))
#mutlivariate normal
library(mgcv)
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-33. For overview type 'help("mgcv-package")'.
```




```
samps = rmvn(1000, mu = rep(0, 10), V = sig[,,1])
cov(samps)
```

```
##             [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
##  [1,] 11.7568968 12.047144  9.832708  7.978332  6.616996  5.922498
##  [2,] 12.0471438 14.113757 12.201969  9.919707  8.987500  8.072366
##  [3,]  9.8327076 12.201969 12.023982 10.691221 10.000723  8.820633
##  [4,]  7.9783323  9.919707 10.691221 10.351584  9.705503  8.482473
##  [5,]  6.6169958  8.987500 10.000723  9.705503 10.824088 10.671329
##  [6,]  5.9224983  8.072366  8.820633  8.482473 10.671329 12.078772
##  [7,]  4.3345761  6.508692  7.329562  7.377097 10.608678 12.111232
##  [8,]  2.6675676  4.530043  6.295994  6.693212  9.397595  9.760931
##  [9,]  2.5515520  5.481608  6.808025  6.555238  9.593654  9.818912
## [10,] -0.2179271  1.491228  3.587823  4.400348  6.817972  6.451665
##            [,7]      [,8]      [,9]      [,10]
##  [1,]  4.334576  2.667568  2.551552 -0.2179271
##  [2,]  6.508692  4.530043  5.481608  1.4912280
##  [3,]  7.329562  6.295994  6.808025  3.5878230
##  [4,]  7.377097  6.693212  6.555238  4.4003476
##  [5,] 10.608678  9.397595  9.593654  6.8179720
##  [6,] 12.111232  9.760931  9.818912  6.4516653
##  [7,] 14.123816 11.783561 12.385212  9.3980365
##  [8,] 11.783561 11.667010 12.347519 10.1774685
##  [9,] 12.385212 12.347519 14.802366 10.8732405
## [10,]  9.398037 10.177468 10.873241 11.0224719
```

```
sig
```

```
## , , 1
##
```

```
##             [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
##  [1,] 12.1894752 12.758280 10.613033  8.654694  7.342124  6.511563
##  [2,] 12.7582800 15.058598 13.204693 10.803932  9.991560  8.912461
##  [3,] 10.6130334 13.204693 13.106975 11.679084 11.060006  9.720407
##  [4,]  8.6546943 10.803932 11.679084 11.291337 10.651386  9.288879
##  [5,]  7.3421244  9.991560 11.060006 10.651386 11.786507 11.449417
##  [6,]  6.5115631  8.912461  9.720407  9.288879 11.449417 12.641528
##  [7,]  4.8627449  7.369829  8.186567  8.085278 11.315048 12.631164
##  [8,]  3.2797376  5.498450  7.197200  7.403241 10.112014 10.336295
##  [9,]  3.2565898  6.477994  7.669468  7.215339 10.338392 10.453739
## [10,]  0.3135191  2.336099  4.304234  4.901870  7.355439  6.921761
##             [,7]      [,8]      [,9]      [,10]
##  [1,]  4.862745  3.279738  3.256590  0.3135191
##  [2,]  7.369829  5.498450  6.477994  2.3360992
##  [3,]  8.186567  7.197200  7.669468  4.3042338
##  [4,]  8.085278  7.403241  7.215339  4.9018697
##  [5,] 11.315048 10.112014 10.338392  7.3554390
##  [6,] 12.631164 10.336295 10.453739  6.9217610
##  [7,] 14.608209 12.270304 13.036142  9.7966700
##  [8,] 12.270304 12.040176 12.861630 10.4148549
##  [9,] 13.036142 12.861630 15.427232 11.2580404
## [10,]  9.796670 10.414855 11.258040 11.1452575
```

```
#generate auto-regressive variables
rho = .8
n = 5
sig = diag(1, n)
for(i in 1:n){
  for(k in 1:n){
    sig[i, k] = sig[k, i] = rho^(i-k)
  }
}
sig
```

```
##        [,1]  [,2] [,3]  [,4]   [,5]
## [1,] 1.0000 0.800 0.64 0.512 0.4096
## [2,] 0.8000 1.000 0.80 0.640 0.5120
## [3,] 0.6400 0.800 1.00 0.800 0.6400
## [4,] 0.5120 0.640 0.80 1.000 0.8000
## [5,] 0.4096 0.512 0.64 0.800 1.0000
```

```
samps = rmvn(1000, mu = rep(0, n), V = sig)
cov(samps)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0529910 0.8430885 0.6813850 0.5664003 0.4712110
## [2,] 0.8430885 1.0403215 0.8549665 0.7058342 0.5811866
## [3,] 0.6813850 0.8549665 1.0361815 0.8530715 0.6860905
## [4,] 0.5664003 0.7058342 0.8530715 1.0733213 0.8543329
## [5,] 0.4712110 0.5811866 0.6860905 0.8543329 1.0276702
```

```
#convert to a correlation matrix
library(MBESS)
cov2cor(cov(samps))
```

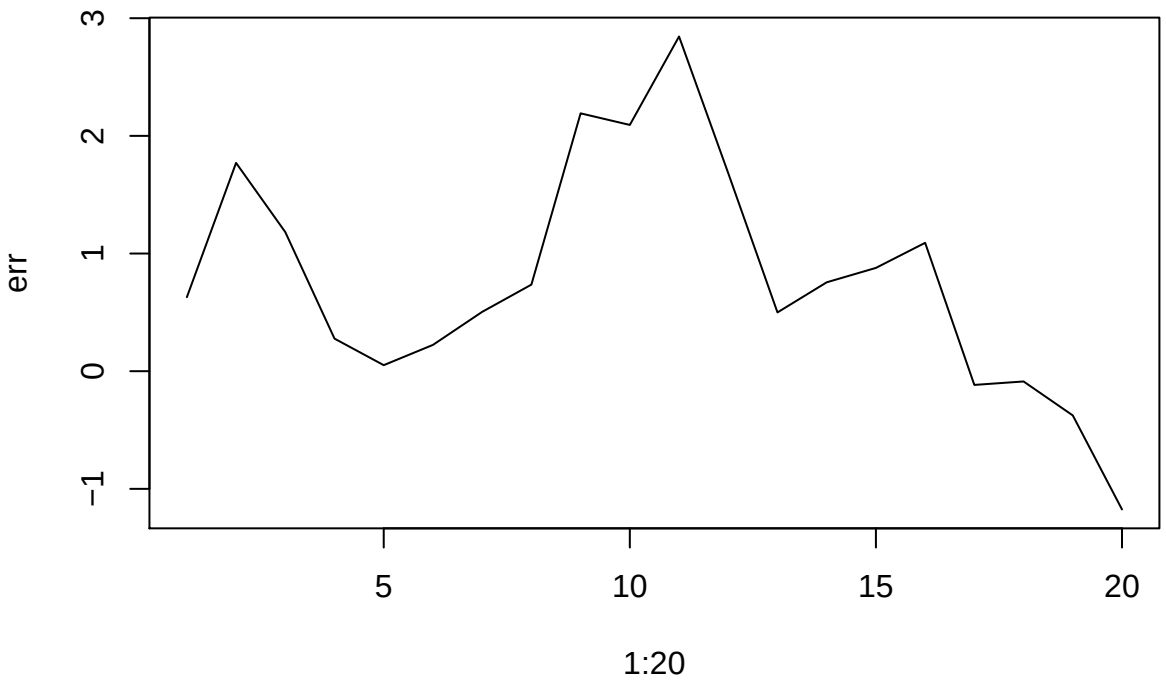
```
##           [,1]      [,2]      [,3]      [,4]      [,5]
```

```
## [1,] 1.0000000 0.8055213 0.6523225 0.5327780 0.4529770
## [2,] 0.8055213 1.0000000 0.8234693 0.6679655 0.5620888
## [3,] 0.6523225 0.8234693 1.0000000 0.8089145 0.6648697
## [4,] 0.5327780 0.6679655 0.8089145 1.0000000 0.8134584
## [5,] 0.4529770 0.5620888 0.6648697 0.8134584 1.0000000
```

```
#generate more explicit correlations
#if x1 and x2 are correlated rho, x1 ~ rho*x2 + sqrt(1-rho^2)*F,
#   where F is the distribution
rho = .6
x1 = rnorm(1000)
x2 = rho * x1 + sqrt(1 - rho^2)*rnorm(1000)
cor(x1, x2)
```

```
## [1] 0.6155677
```

```
#autocorrelation
err = rnorm(1)
n = 20
for(i in 2:n) err[i] = rho * err[i - 1] + sqrt(1 - rho^2)*rnorm(1)
plot(1:20, err, type = 'l')
```



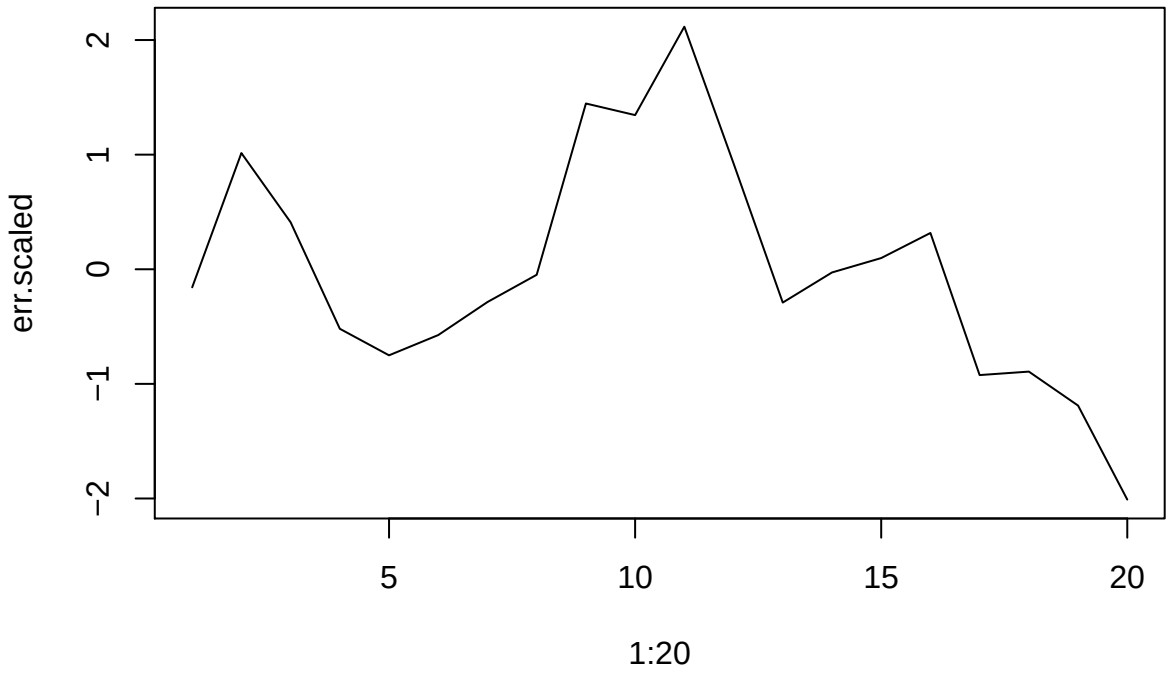

```
mean(err)
```

```
## [1] 0.782722
```

```
sd(err)
```

```
## [1] 0.9741442
```

```
err.scaled = scale(err)
plot(1:20, err.scaled, type = 'l')
```




```
mean(err.scaled)
```

```
## [1] -2.461139e-17
```

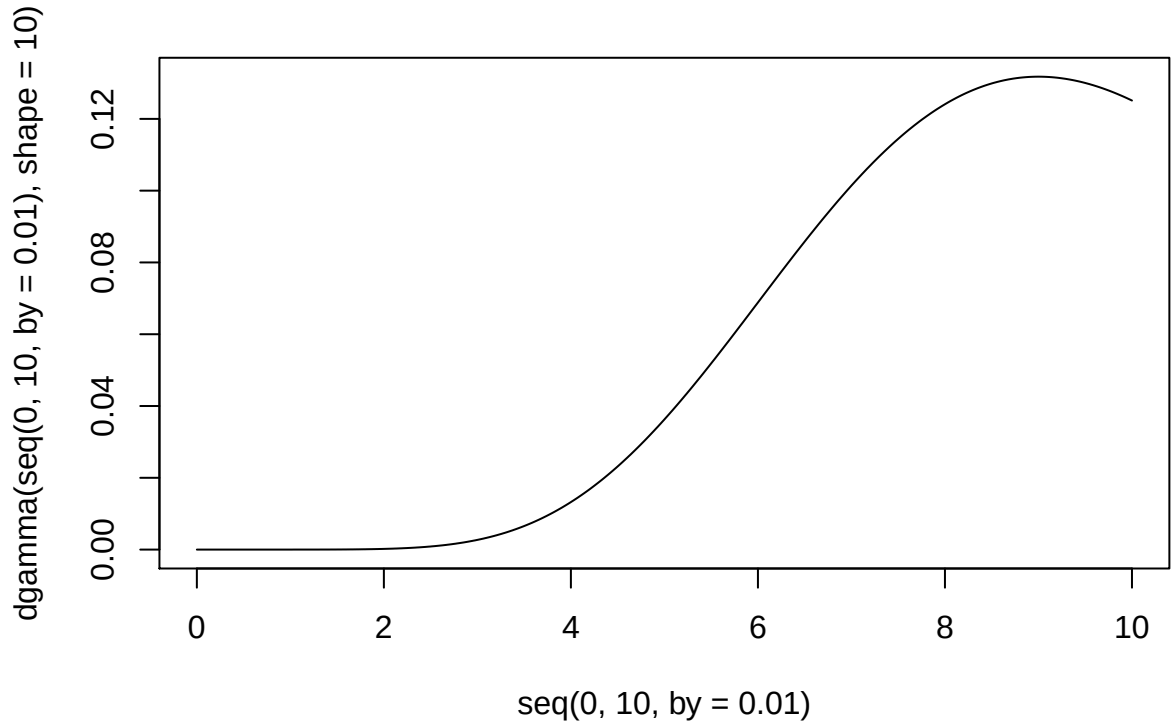
```
sd(err.scaled)
```

```
## [1] 1
```

Central Limit Theorem

- (From Wiki) In probability theory, the central limit theorem (CLT) establishes that, in some situations, when independent random variables are added, their properly normalized sum tends toward a normal distribution (informally a bell curve) even if the original variables themselves are not normally distributed. The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.
- If $X_1, X_2, \ldots, X_n$ are random samples each of size n taken from a population with overall mean μ and finite variance σ^2 and if $\bar{X}$ is the sample mean, the limiting form of the distribution of $Z = \left(\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}}\right)$ as $n \to \infty$, is the standard normal distribution.

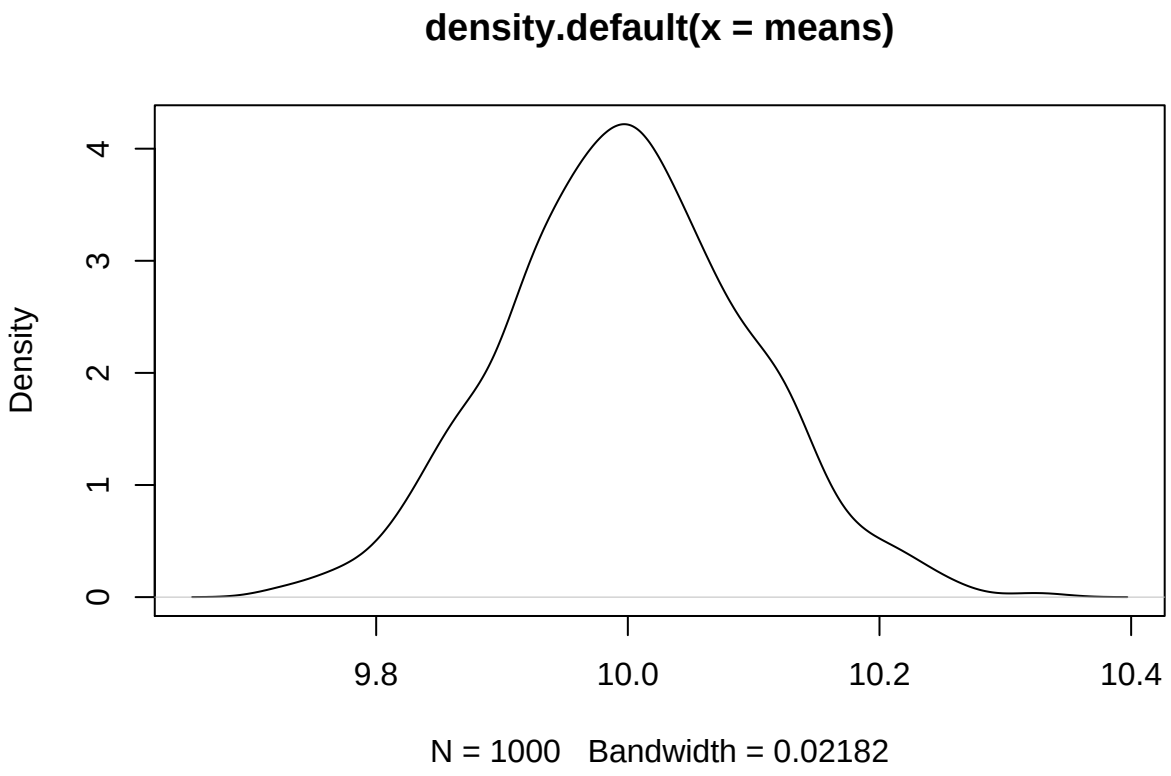
```
#remember the gamma
plot(seq(0, 10, by = .01), dgamma(seq(0, 10, by = .01), shape = 10), type = 'l')
```



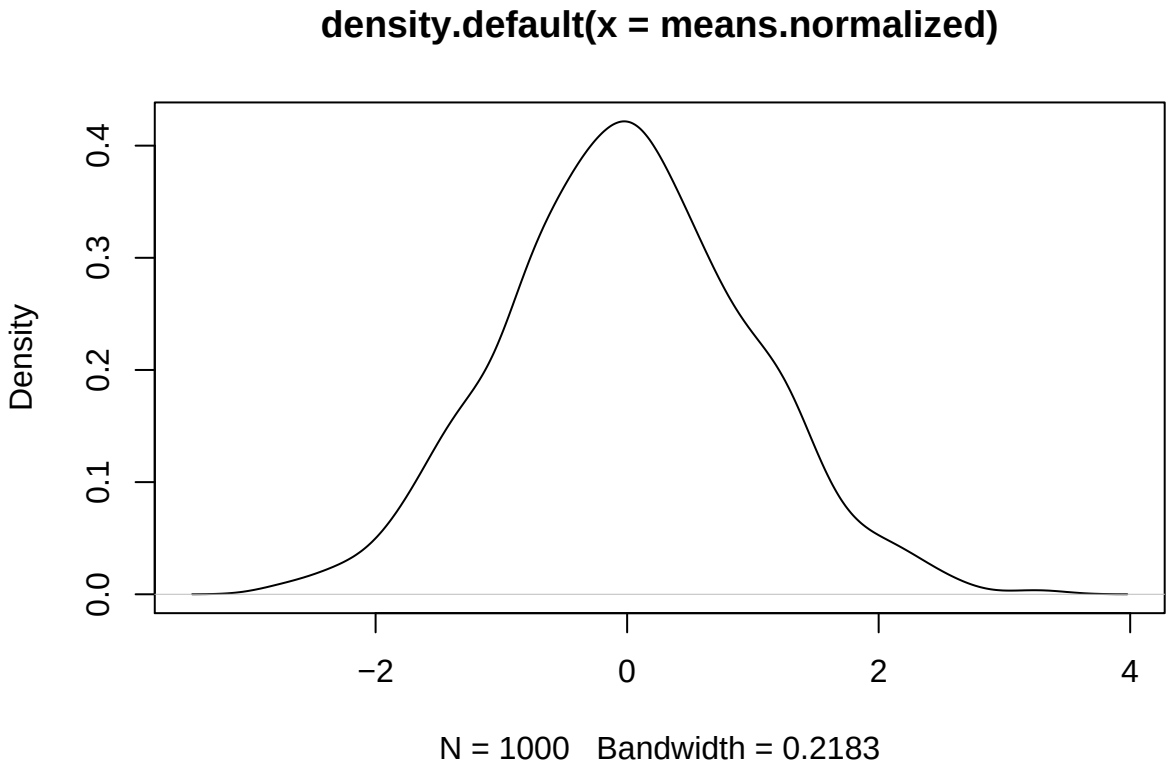

```
samps = lapply(1:1000, function(x) rgamma(1000, shape = 10))
class(samps)
```

```
## [1] "list"
```

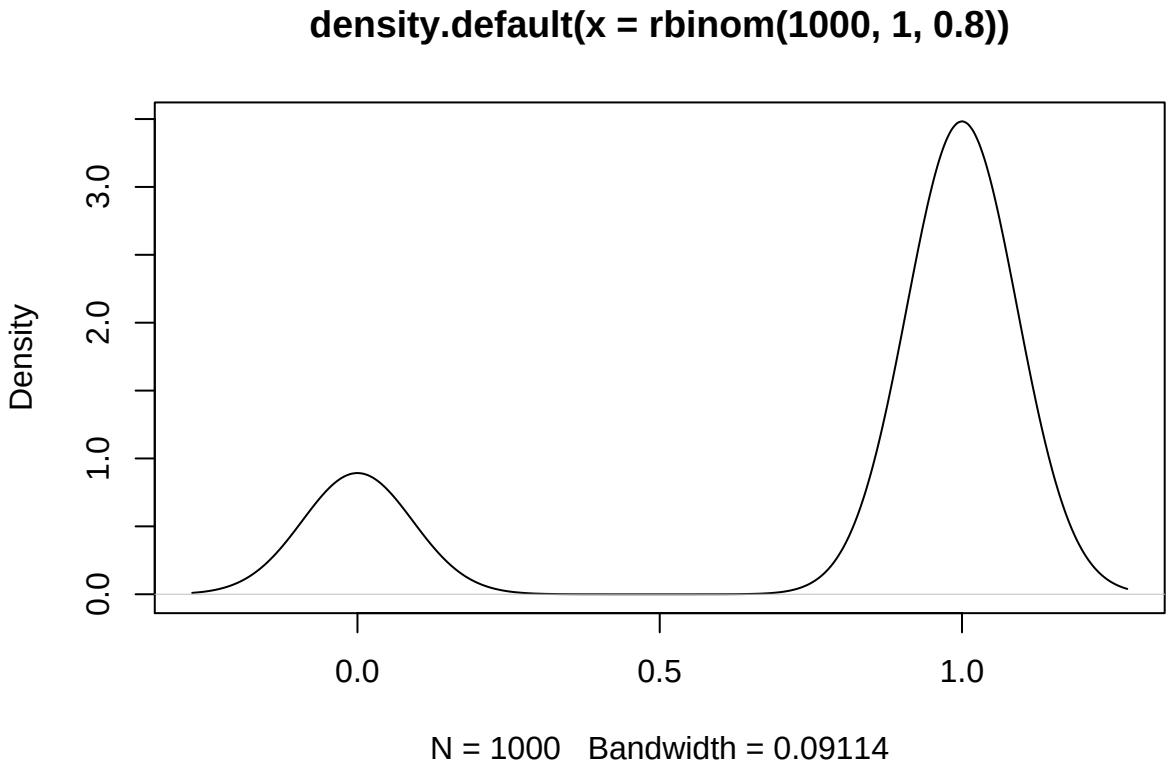
```
means = unlist(lapply(samps, mean))
plot(density(means))
```



```r
samps.bar = mean(unlist(samps))
sig = var(unlist(samps))
means.normalized = (means - samps.bar)/(sqrt(sig / 1000))
plot(density(means.normalized))
```



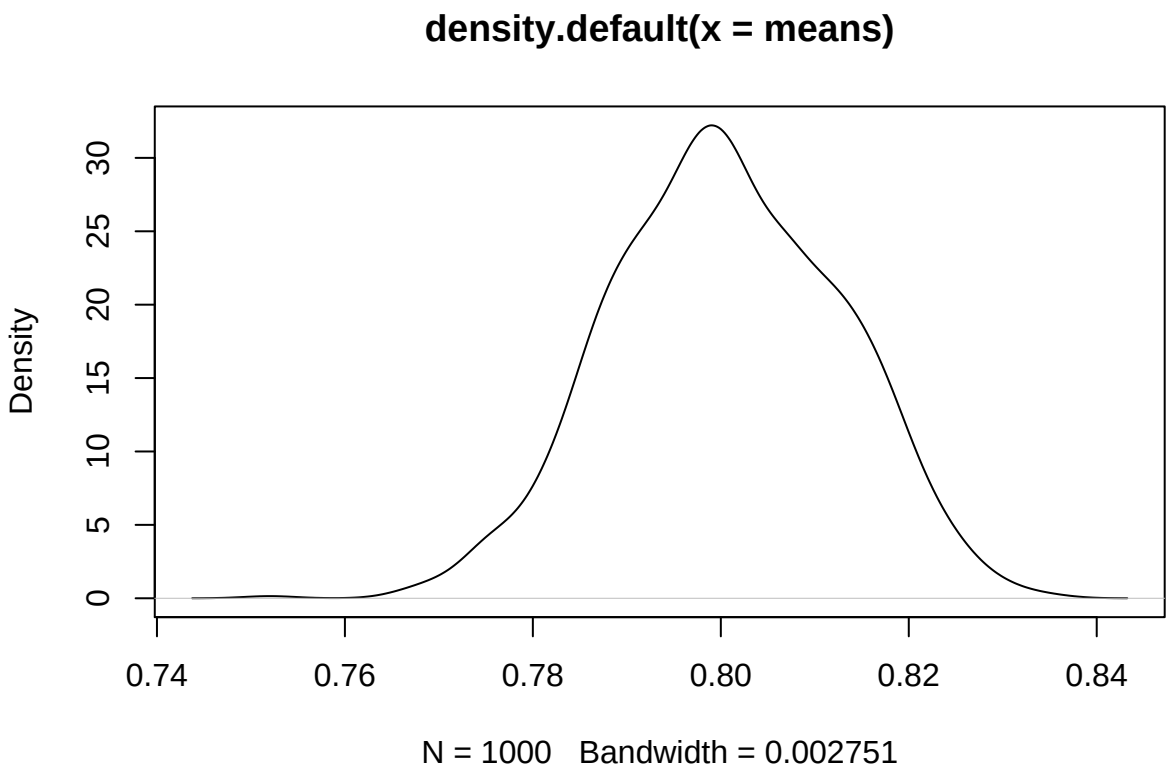

```r
#what about binomial?
plot(density(rbinom(1000, 1, .8)))
```



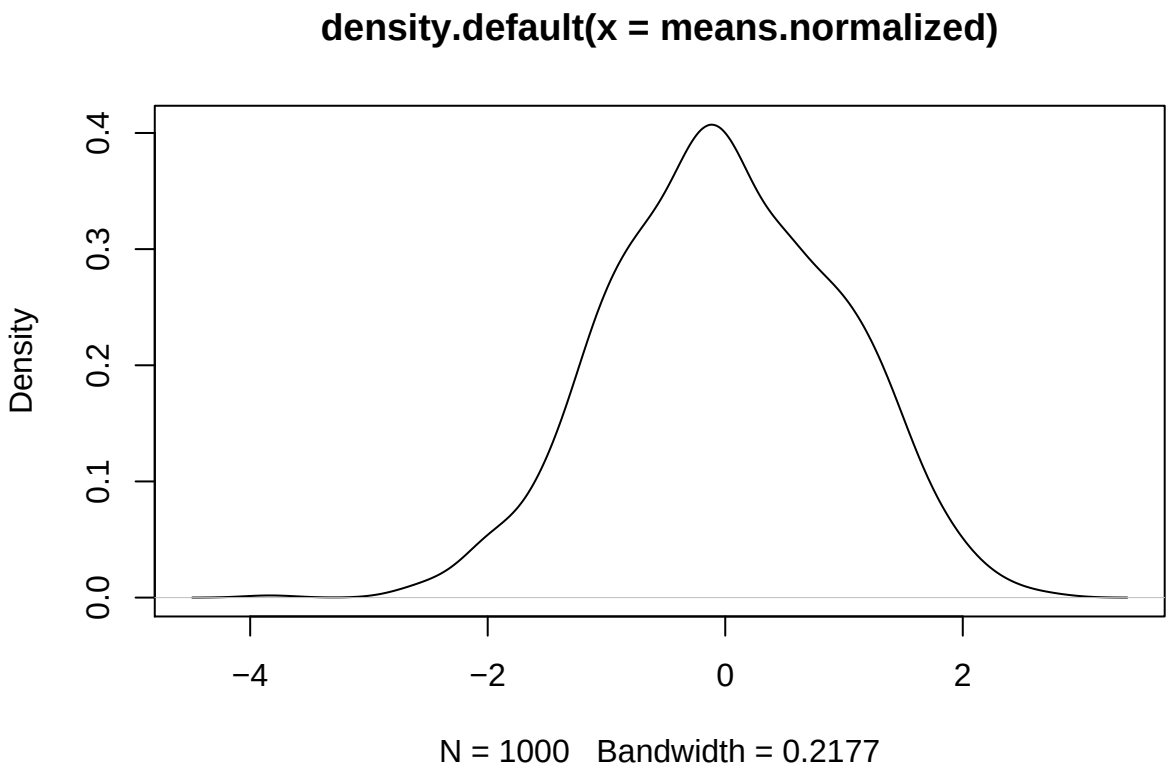

```
samps = lapply(1:1000, function(x) rbinom(1000, size = 1, prob = .8))
class(samps)
```

```
## [1] "list"
```

```
means = unlist(lapply(samps, mean))
plot(density(means))
```




```
samps.bar = mean(unlist(samps))
sig = var(unlist(samps))
means.normalized = (means - samps.bar)/(sqrt(sig / 1000))
plot(density(means.normalized))
```



```r
#are they from a standard normal?
t.test(means.normalized, rnorm(1000))
```

```
##
##  Welch Two Sample t-test
##
## data:  means.normalized and rnorm(1000)
## t = 0.90101, df = 1996.2, p-value = 0.3677
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.04634771  0.12512890
## sample estimates:
##     mean of x     mean of y
## -1.484406e-15 -3.939060e-02
```

```r
shapiro.test(means.normalized)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  means.normalized
## W = 0.99643, p-value = 0.02218
```

```r
#non-parametric
ks.test(means.normalized, rnorm(1000))
```

```
## Warning in ks.test(means.normalized, rnorm(1000)): p-value will be
## approximate in the presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  means.normalized and rnorm(1000)
## D = 0.04, p-value = 0.4005
## alternative hypothesis: two-sided
```

Simulation exercises

- Write a function that takes as an argument n and returns an $n \times 10$ matrix X
- The columns of X should be independently but not identically distributed (get creative)
- Write a second function that takes X and generates an $n \times 1$ outcome y through a linear combination of the variables (only include three in the calculation) plus i.i.d. normally distributed errors
- Write a third function that takes as an argument $n.samps$, generates $n.samps$ X matrices and y outcomes using the above functions, runs a linear regression, and returns coverage probabilities, false positive rates, and false negative rates
- Next, alter the functions to violate assumptions: Create a correlated but unrelated X column, create autocorrelated errors, etc.
- Assess the returns of the functions at various levels of $n.samps$ and n
- Plot the results

```r
set.seed(999)
```