R Users Guide to Stat 201: Chapter 6

Michael Shyne, 2017

Chapter 6: Normal Probability Distributions

Chapter 6 continues with probability distributions, now focusing on continuous probability distributions. In particular, the uniform and normal distributions are examined, as well as sampling distributions. From an R perspective, these will be handled in a very similar manner to the discrete distributions.

Continuous uniform distribution

R provides an analogous set of functions to handle uniform distributions as we saw previously with other distributions. They are `runif()`, `punif()`, etc. One difference to be aware of with continuous distributions is that we are no longer concerned with the difference between "less than" and "less than or equal to". Since $P(X = x) = 0$ for any x in a continuous distribution, there is no difference between $P(X < x)$ and $P(X \leq x)$.

```
# 10 random numbers (default range is (0,1))
runif(10)
```

```
##  [1] 0.849398684 0.005685813 0.679796802 0.484304396 0.682065186
##  [6] 0.865327292 0.747732786 0.710413145 0.020878707 0.406338693
```

```
# P(X < 2) in U(0,6)
punif(2, min=0, max=6)
```

```
## [1] 0.3333333
```

```
# P(2 < X < 3.5) in U(0,6)
punif(3.5, min=0, max=6) - punif(2, min=0, max=6)
```

```
## [1] 0.25
```

```
# Value x such that P(X > x) = .05 in U(-1,1)
qunif(0.05, min=-1, max=1, lower=F)
```

```
## [1] 0.9
```

Normal distribution

For normal distributions, we will use the functions `rnorm()`, `pnorm()`, etc. The parameters for a normal are the mean μ and standard deviation σ. The default values are 0 and 1 respectively. Thus, the normal distribution used, without supplying parameters, is the standard normal or z distribution.

```
# 10 random numbers (default is standard normal N(0,1))
rnorm(10)
```

```
##  [1] -0.79268783 -1.25032197  1.10254295 -0.45908392 -2.38772711
##  [6]  1.89081270 -0.09989451 -0.58572965 -1.76596373 -0.65166046
```

```
# P(X < 2) in N(3,4)
pnorm(2, mean=3, sd=4)
```

```
## [1] 0.4012937
```

```r
# P(2 < X < 3.5) in N(3,4)
pnorm(3.5, mean=3, sd=4) - pnorm(2, mean=3, sd=4)
```

```
## [1] 0.1484446
```

```r
# Value x such that P(X > x) = .05 in N(42,6)
qnorm(0.05, mean=42, sd=6, lower=F)
```

```
## [1] 51.86912
```

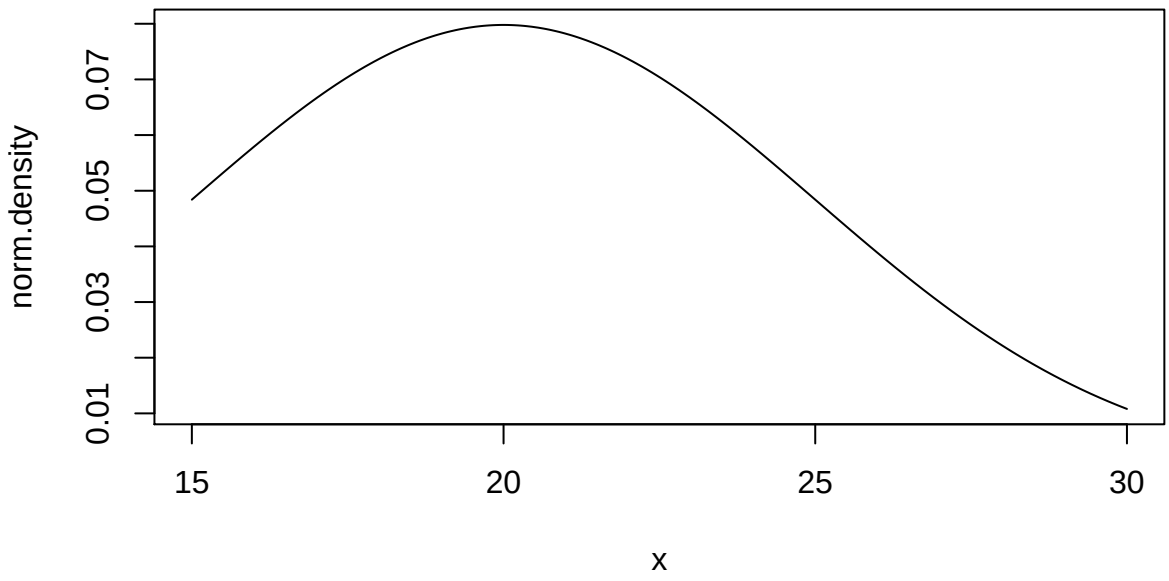
R density functions

As mentioned in a previous guide, density functions for continuous variables (`dunif()`, `dnorm()`) have a slightly different meaning than for discrete variables. Recall, the probability of a single value in a continuous random variable is essentiallyy zero and probabilities of ranges of values are defined as the area under the density curve corresponding to the range. The density function then gives the height of the density curve for a single value, but that should not be interpretted as a probability. However, it can be used to plot density curves, as we did when we added a normal curve to a histogram. Here is an example of how it works.

```r
# define a sequence of x values for which we want to plot a desnity curve
x <- seq(15, 30, length=1000)    # 1000 is the number of values in our sequence.
                                 # It is many more than we need.

# Find the curve height of a normal distribution for each x value
norm.density <- dnorm(x, mean=20, sd= 5)    # Since we passed in a vector
                                            # of x value, we will get back
                                            # a vector of density values

plot(x, norm.density, type='l')    # type = l draws lines instead of points
```



Samples

In order to work with sampling distributions and the Central Limit Theorem, simply use the normal distribution functions with the appropriate mean and standard deviation ($\mu_{\bar{x}} = \mu$ and $\sigma_{\bar{x}} = \sigma/\sqrt{n}$).

I created a demonstration of the Central Limit Theorem (https://seighin.shinyapps.io/clt_demo/) using Shiny, a simple framework for creating web apps running R code. A fuller discussion of Shiny is beyond the scope of these guides, but more information can be found at https://shiny.rstudio.com/.

In order to create random samples from a population of values, I used the `sample()` command. Given a vector, `sample()` will return a random sample of that vector of specified size.

```
# Create a "population" of values
x <- rnorm(1000)

# Get a sample from the "population"
sample(x, 5)
```

```
## [1] -0.4501400 -0.9067998 -1.0130725 -0.4229216 -1.5280317
```

If `sample()` is given an integer (n) instead of a vector, it will create a sample from the range `1:n`. This can be used to generate a vector of indices for non-vector data such as data frames.

```
# Create a data frame
df <- data.frame(x=1:10, x.sqr=(1:10)^2)
df
```

```
##     x x.sqr
## 1   1     1
## 2   2     4
## 3   3     9
## 4   4    16
## 5   5    25
## 6   6    36
## 7   7    49
## 8   8    64
## 9   9    81
## 10 10   100
```

```
# Get a sample from integer list same length as data frame
sam.idx <- sample(nrow(df), 3)

# These are index values
sam.idx
```

```
## [1] 5 8 2
```

```
# Get sample from data frame
df[sam.idx, ]
```

```
##   x x.sqr
## 5 5    25
## 8 8    64
## 2 2     4
```

By default, the `sample()` function samples without replacement. Samples with replacement can be created using the optional parameter `replace=TRUE`. Also, since the samples are randomly generated, to create reproducible samples, be sure to set the random seed prior to calling the function.

License