STAT 4500 Class Notes

Contents

About

This document is an R markdown file of the class notes from STAT 4500, Non Parametric Statistics for Fall 2016 at UVU. Textbook used was "Applied Nonparametric Statistical Methods" Fourth Edition by Peter Sprent and Nigel C. Smeeton

8/24/16 Notes:

-Defined Random variable
-PMF
-PDF
-Ranks

Binomial Distribution Discussion 8/26/16:

```
# 8/26/16 Non Parametric class notes:
x <- 0:10
dbinom(x, 10, 0.5) #dbinom is the PMF of binomial
```

```
##  [1] 0.0009765625 0.0097656250 0.0439453125 0.1171875000 0.2050781250
##  [6] 0.2460937500 0.2050781250 0.1171875000 0.0439453125 0.0097656250
## [11] 0.0009765625
```

```
pbinom(x,10,0.5, lower.tail = FALSE) # this is for cummulative
```

```
##  [1] 0.9990234375 0.9892578125 0.9453125000 0.8281250000 0.6230468750
##  [6] 0.3769531250 0.1718750000 0.0546875000 0.0107421875 0.0009765625
## [11] 0.0000000000
```

There is a special type of test called "Randomized Test"

read about it on page 27-28 in text book

```
# what is the probability that x = 8 under the null
pbinom(8,10,0.5)
```

```
## [1] 0.9892578
```

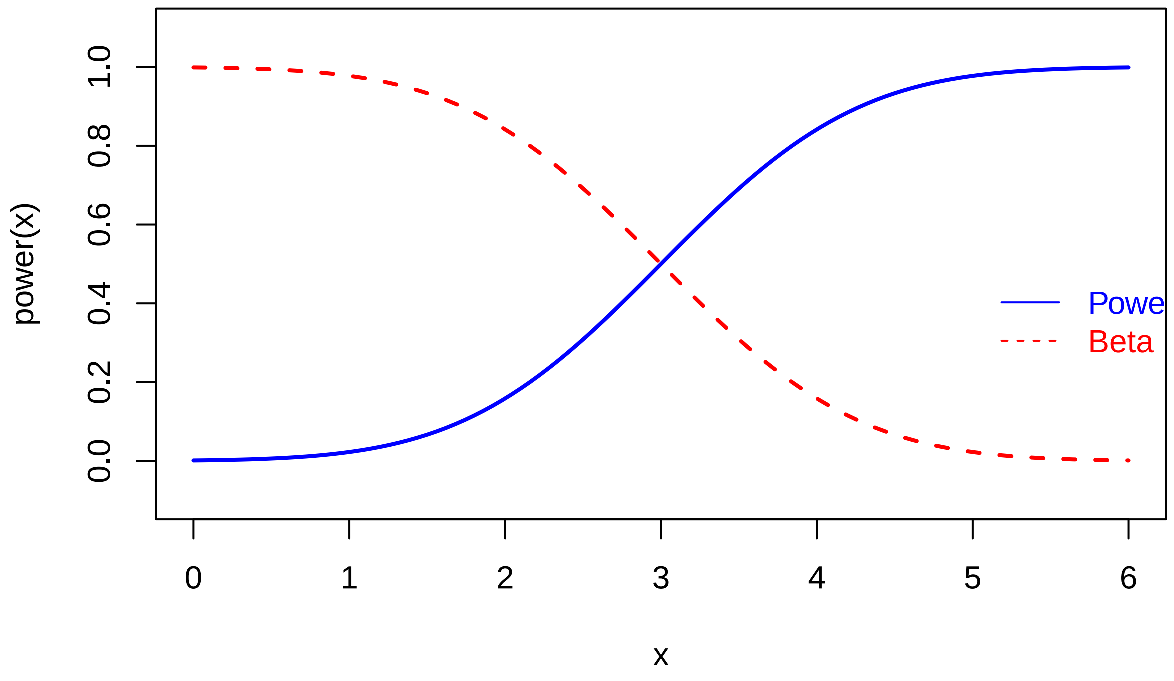
```r
0.0393/pbinom(8,10,0.5)
```

```
## [1] 0.03972675
```

Reject with probability 1 is sum(x_i) > 8
Reject with probability 0.0397 if sum(x_i) = 8
Reject with probability 0 if sum(x_i) < 8

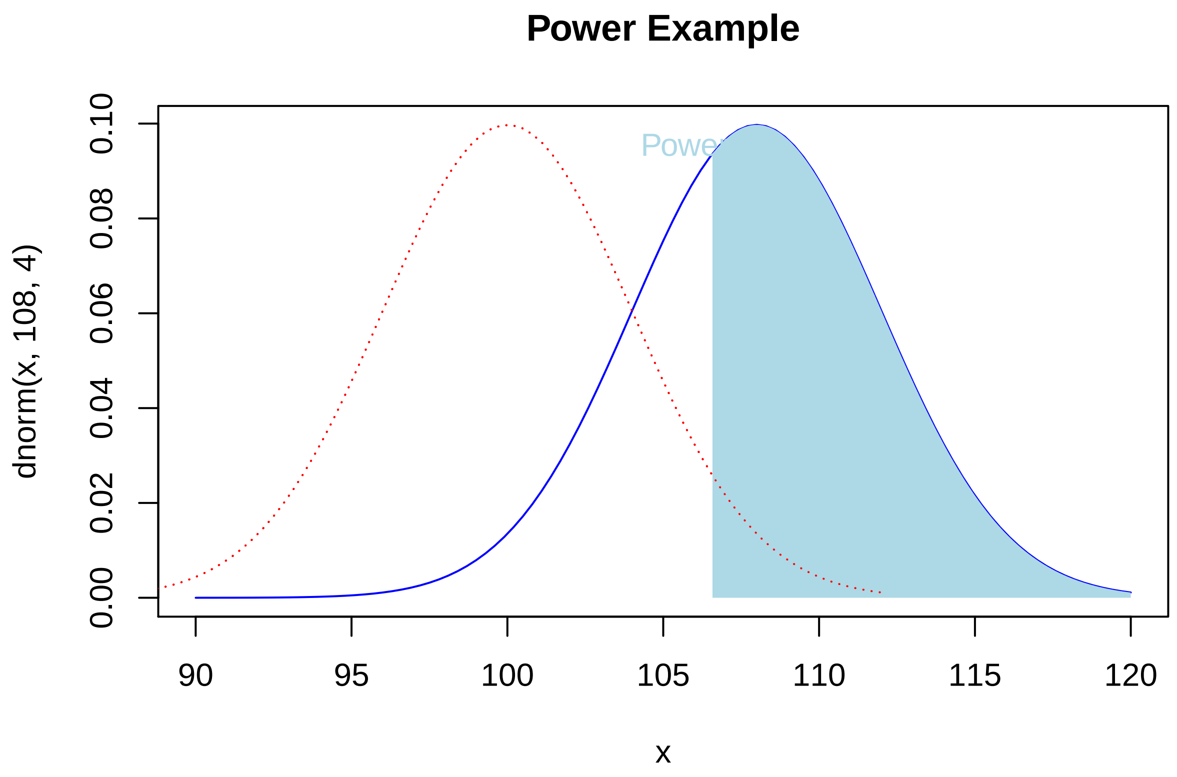
8/31/16 Power Curve Example

```r
# power example from class
power <- function(x){pnorm(3 - x, lower.tail = F)}
beta <- function(x){1-pnorm(3 - x, lower.tail = F)}
curve(power, 0, 6, col="blue", lwd=2, ylim=c(-0.1,1.1), lty=1)
curve(beta, 0, 6, add=T, col="red", lwd=2, lty=2)
legend(5, 0.5, legend = c("Power", "Beta"),lty=c(1,2),
       col =c("blue", "red") , bty = "n", text.col=c("blue", "red"))
```



Power Plot Example:

```r
curve(dnorm(x, 108, 4), 90, 120, col="blue", main="Power Example")
curve(dnorm(x, 100, 4), 80, 112, lty=3, add=T, col="red")
cord.x <- c(106.58, seq(106.58, 120, 0.01), 120)
cord.y <- c(0, dnorm(seq(106.58, 120, 0.01), 108, 4),0)
#abline(v=106.58, lty=2, col = "skyblue")
polygon(cord.x, cord.y,col="lightblue", border = NA)
curve(dnorm(x, 100, 4), 80, 112, lty=3, add=T, col="red")
legend("top", legend = "Power", text.col = "lightblue", bty = "n")
```



Another way of understanding Power of a test. The shaded area is equal to power if the true mean were 108 when the hypothesized mean was 100, when n = 16 and $\sigma = 16$. The area is equal to the value at $\alpha = 0.05$ under the null hypothesis which is

$$Z_\alpha = \frac{x-\mu}{\frac{\sigma}{\sqrt{n}}}$$

and plugging in when $\alpha = 0.05$

$$1.6448536 = \frac{x-100}{\frac{16}{\sqrt{16}}}$$

and solving for x we get 106.5794145.

We now can compute power by finding the area under the normal curve for the "true" mean, 108. This can be done by plugging into the pnorm function in R, pnorm(cv, 108, 16/sqrt(16), lower.tail = F) = 0.63876.

Sign Test

Assumption: f is continuous. θ is the median.
$H_0 : \theta = \theta_0$ vs. $H_1 : \theta > \theta_0$.

Example 2.2 (page 27) from the text: $H_0 : \theta = 200$ vs. $H_1 : \theta \neq 200$

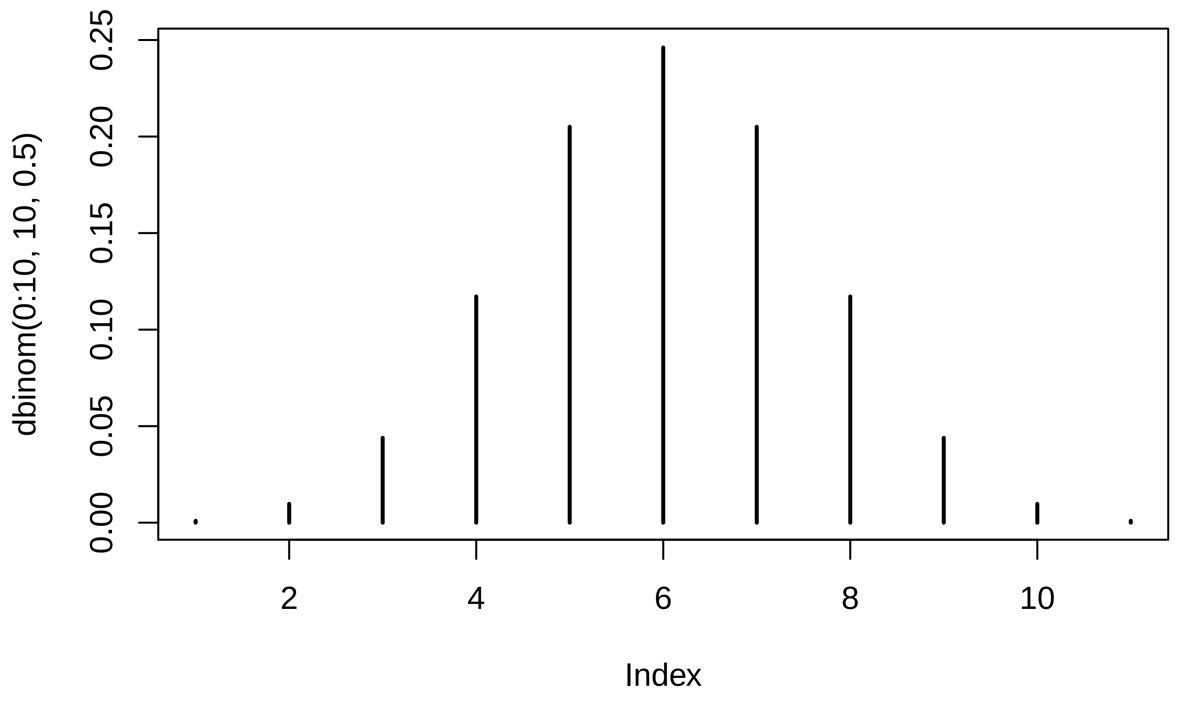
```r
# data from the example
y <- c(49,58,75,110,112,132,151,276,281,362)
# binomial with n = 10, p = 0.5
pbinom(0:10, 10, 0.5)
```

```
##  [1] 0.0009765625 0.0107421875 0.0546875000 0.1718750000 0.3769531250
##  [6] 0.6230468750 0.8281250000 0.9453125000 0.9892578125 0.9990234375
## [11] 1.0000000000
```

```r
1 - pbinom(0:10, 10, 0.5)
```

```
##  [1] 0.9990234375 0.9892578125 0.9453125000 0.8281250000 0.6230468750
##  [6] 0.3769531250 0.1718750000 0.0546875000 0.0107421875 0.0009765625
## [11] 0.0000000000
```

```r
plot(dbinom(0:10, 10, 0.5), type="h", lwd=2)
```



9/2/16

```r
# example from class where H0 = 0.5, H1 < 0.5
p0 <- 0.5
p1 <- 0.25
n <- 20
prob <- pbinom(0:n, n, p0)
prob # critical value is n=1
```

```
##  [1] 9.536743e-07 2.002716e-05 2.012253e-04 1.288414e-03 5.908966e-03
##  [6] 2.069473e-02 5.765915e-02 1.315880e-01 2.517223e-01 4.119015e-01
## [11] 5.880985e-01 7.482777e-01 8.684120e-01 9.423409e-01 9.793053e-01
## [16] 9.940910e-01 9.987116e-01 9.997988e-01 9.999800e-01 9.999990e-01
## [21] 1.000000e+00
```

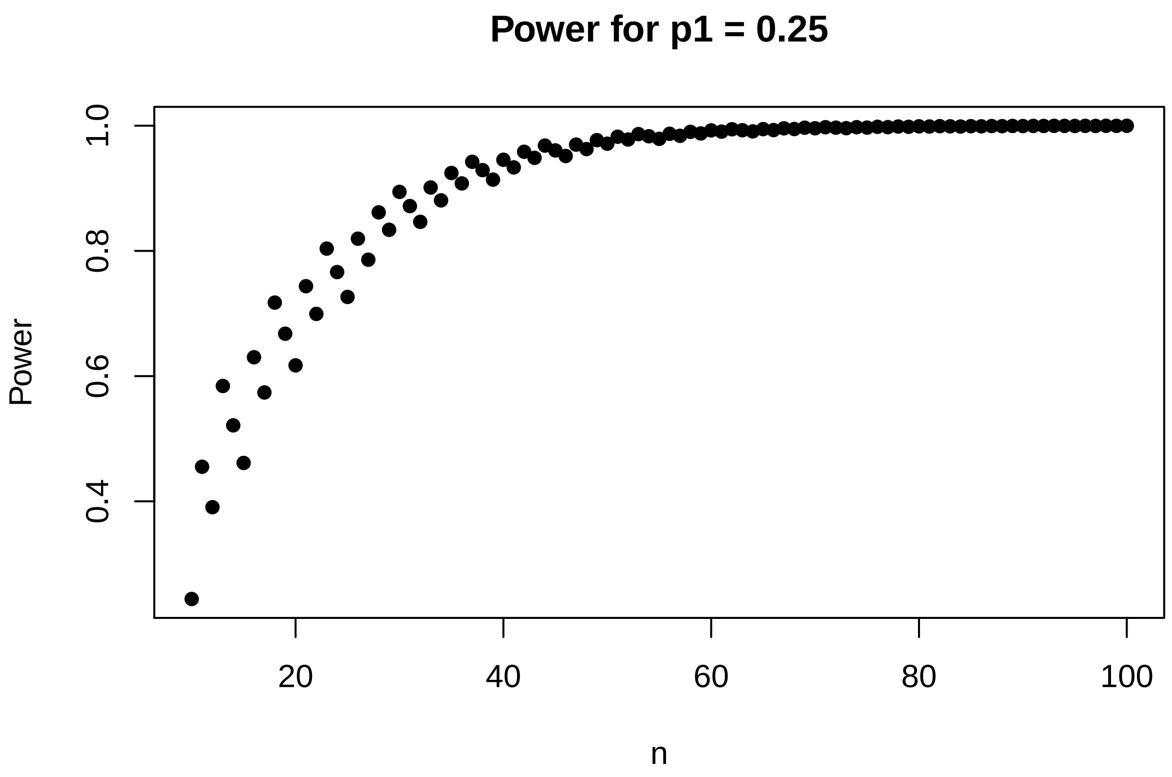
```r
cv <- max(which(prob <= 0.05) - 1) # gives us critical value of x
cv
```

```
## [1] 5
```

```
# what's the power of this test?
pow <- pbinom(cv, n, p1)
pow
```

```
## [1] 0.6171727
```

```
# let's create a function called power
power <- function(n, p0=0.5, p1){
  prob <- pbinom(0:n, n, p0)
  cv <- max(which(prob <= 0.05)-1)
  pow <- pbinom(cv,n,p1)
  return(pow)
} # this function needs an n, which needs to be a scalar not a vector, and p0, p1
# try to plot this
x <- vector()
for(i in 10:100){
  x[i] <- power(i, p0=0.5, p1=0.25)
}
x <- x[-(1:9)] # get rid of NAs
plot(10:100, x, main="Power for p1 = 0.25", xlab="n", ylab="Power", pch=16)
```



Relative Efficiency

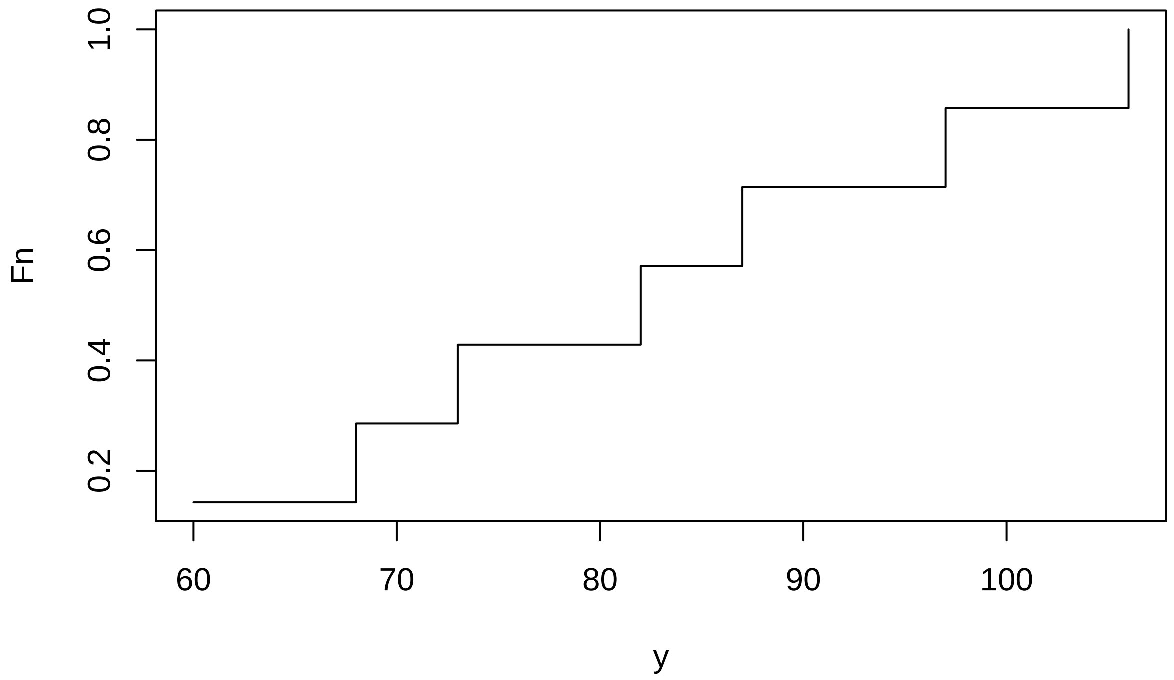
T1 and T2. At some fixed level α

For T1, let n1 be the sample size required to achieve a type II error β.
For T2, let n2 be the sample size required to achieve the same type II error β. Relative efficiency of T2 with respect to T1 is $\frac{n_1}{n_2}$.

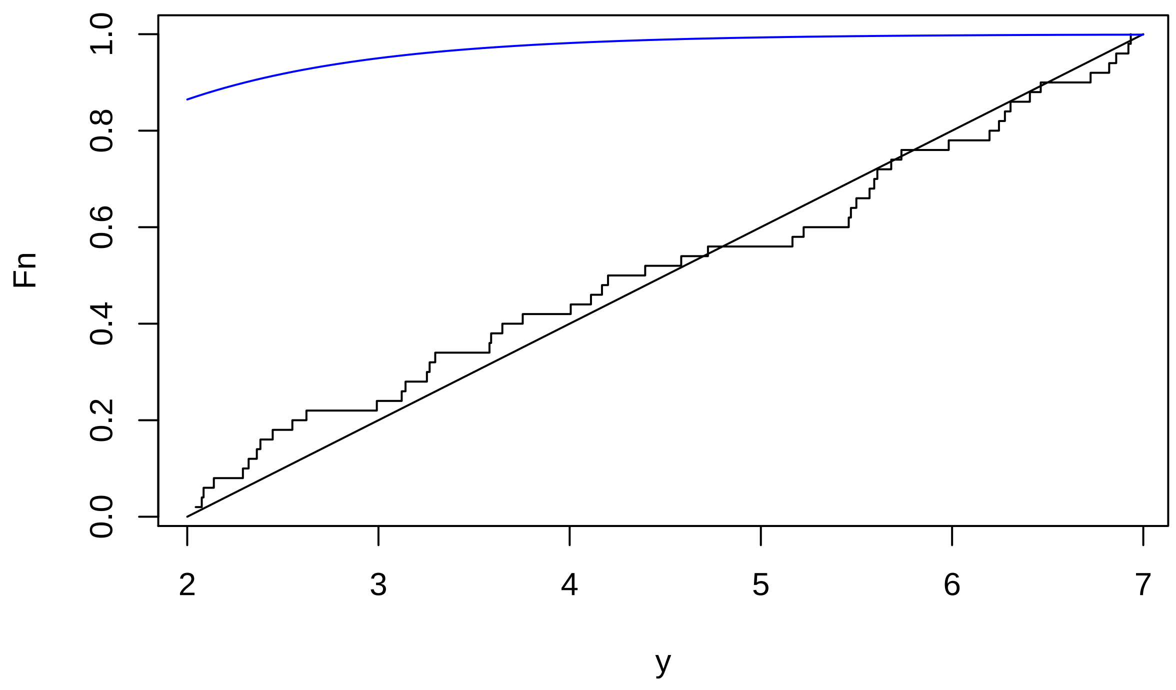
Sign test with respect to t test

$$\frac{2}{\pi} < 1$$

```r
# data from page 48
x <- c(73,82,87,68,106,60,97)
# let's sort
y <- x[order(x)]
# Empical distribution from class
n <- length(x)
Fn <- NULL
for(i in 1:n){
  Fn[i] <- i/n
}
plot(y, Fn, type="s")
```



```r
# another example
# generate data from the uniform distribution
x <- runif(50,2,7)
y <- x[order(x)]
# Empical CDF
n <- length(x)
Fn <- 1:n/n
plot(y, Fn, type="s") # Empiracle distribution
curve((x-2)/5,2,7,add=T) # actual distribution
curve(1 - exp(-x),2,7,add = TRUE, col="blue") # not sure what this one is
```



```
# These are all aspects of exploratory data analysis
```

Tests for Median 9/7/16

(i) Sign test
(ii) Wilcoxon signed rank test

- Data is continuous
- Data is symmetric about the median Difference between these two tests is that Wilcoxon test takes into account BOTH the sign AND the rank. Sign test considers only the sign.

Hypotheses for Wilcoxon test:

$$H_0 : \theta = \theta_0 \ vs \ H_1 \theta \neq \theta_0$$
$$H_0 : \theta = \theta_0 \ vs \ H_1 \theta > \theta_0$$
$$H_0 : \theta = \theta_0 \ vs \ H_1 \theta < \theta_0$$

We have data ordered, from smallest to largest:

$$X_1, X_2, X_3, ..., X_n$$

Calculate d_i which is equal to $|X_i - \theta_0|$

Heart rate example again:

```
x <- c(73,82,87,68,106,60,97)
# how many are less than or equal to 69?
length(which(x <= 69))
```

```
## [1] 2
```

```
# p of x
length(which(x <= 69))/length(x)
```

```
## [1] 0.2857143
```

Using R to assign ranks and signs to a vector:

```
# using the vector x from above:
x <- c(73,82,87,68,106,60,97)
sign <- ifelse(x > 70, 1, -1) # if H0 was 70
d <- rank(abs(x - 70))
rs <- sign*d
cbind(x, sign, d, rs)
```

```
##        x sign d rs
## [1,]  73    1 2  2
## [2,]  82    1 4  4
## [3,]  87    1 5  5
## [4,]  68   -1 1 -1
## [5,] 106    1 7  7
## [6,]  60   -1 3 -3
## [7,]  97    1 6  6
```

```
sum(ifelse(rs>0, rs, 0))
```

```
## [1] 24
```

Notes on page 47-48, developing the sign/rank test

```
library(gtools)
sum(1:7) # sum of 1 to 7
```

```
## [1] 28
```

```
2^7 # possible assignents of -1 or 1 to seven samples
```

```
## [1] 128
```

```
A <- permutations(2, 7, v=c(-1,1), repeats.allowed = T) # all possible combos
head(A) # look at what A is now
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]   -1   -1   -1   -1   -1   -1   -1
## [2,]   -1   -1   -1   -1   -1   -1    1
## [3,]   -1   -1   -1   -1   -1    1   -1
## [4,]   -1   -1   -1   -1   -1    1    1
## [5,]   -1   -1   -1   -1    1   -1   -1
## [6,]   -1   -1   -1   -1    1   -1    1
```

```r
A <- t( t(A) * 1:7) # better than a loop :)
head(A) # now what does A look like
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]   -1   -2   -3   -4   -5   -6   -7
## [2,]   -1   -2   -3   -4   -5   -6    7
## [3,]   -1   -2   -3   -4   -5    6   -7
## [4,]   -1   -2   -3   -4   -5    6    7
## [5,]   -1   -2   -3   -4    5   -6   -7
## [6,]   -1   -2   -3   -4    5   -6    7
```

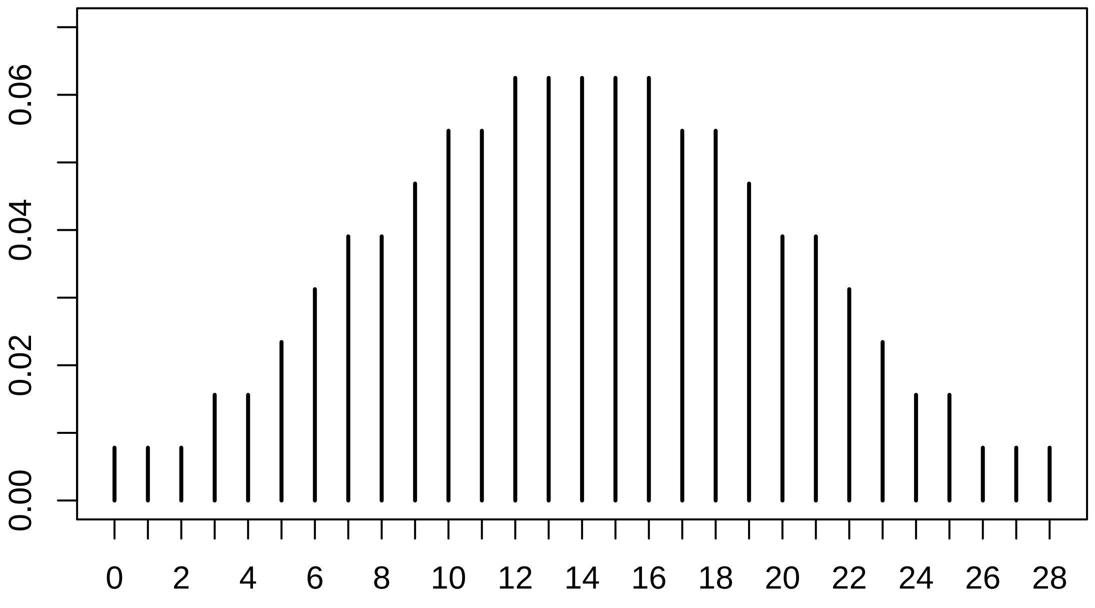
```r
# If our hypothesis involved the positives...
A <- ifelse(A > 0, A, 0) # make all negatives = 0
S <- prop.table(table(apply(A,1,sum))) # probability of each sum
S # same as table 3.1 on pg 48 in textbook
```

```
##
##         0         1         2         3         4         5         6
## 0.0078125 0.0078125 0.0078125 0.0156250 0.0156250 0.0234375 0.0312500
##         7         8         9        10        11        12        13
## 0.0390625 0.0390625 0.0468750 0.0546875 0.0546875 0.0625000 0.0625000
##        14        15        16        17        18        19        20
## 0.0625000 0.0625000 0.0625000 0.0546875 0.0546875 0.0468750 0.0390625
##        21        22        23        24        25        26        27
## 0.0390625 0.0312500 0.0234375 0.0156250 0.0156250 0.0078125 0.0078125
##        28
## 0.0078125
```

```r
# pg 47 - Pr(S+ <= 3)
sum(S[1:4]) # same as 5/128
```

```
## [1] 0.0390625
```

```r
plot(S, ylim = c(0,0.07), ylab="")
```



Writing a function for the sign/rank test:

```
# writing a function that replicates the book example:
# from page 48 Example 3.1
# note: this result is a one sided test.  If you want a two sided
# p value then multiply your result by two.
test <- function(x, theta){
  if(length(x) >= 20){stop("n is too large for this function")}
  sign <- ifelse(x > theta, 1, -1)
  d <- rank(abs(x - theta))
  rs <- sign*d
  t <- sum(ifelse(rs>0, rs, 0))
  A <- gtools::permutations(2, length(x), v=c(-1,1), repeats.allowed = T)
  A <- t( t(A) * 1:length(x))
  A <- ifelse(A > 0, A, 0)
  S <- table(apply(A,1,sum))
  if(t < S[(length(S)/2) + 1]){
    sum(S[1:(t+1)])/2^length(x)
  } else{
    sum(S[(t+1):length(S)])/2^length(x)
  }
}
wilcox.test(x, mu=70, alternative = "greater")$p.value
```

```
## [1] 0.0546875
```

```
test(x,70) # same result :)
```

```
## [1] 0.0546875
```

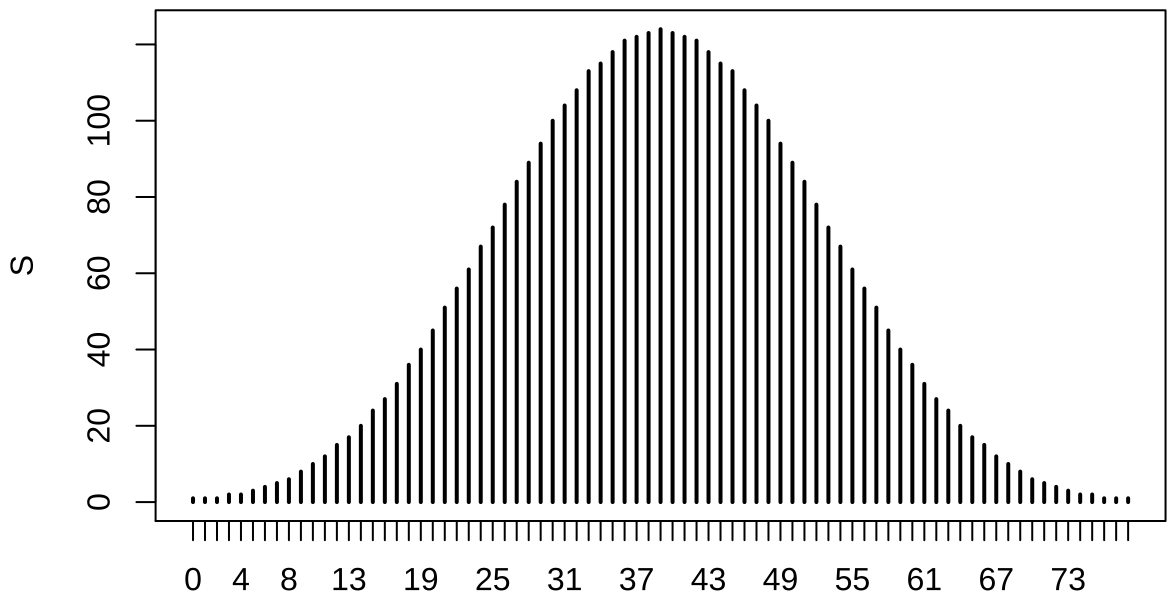
```
#page 51 example
x1 <- c(-2,4,8,25,-5,16,3,1,12,17,20,9)
# using the function above:
test(x1, 15) * 2 # test median is 15, 2x for two sided
```

```
## [1] 0.05224609
```

And we get the same result as the book.

By way of reproducing the plot on page 51 in the text book:

```
x <- x1; theta <- 15
sign <- ifelse(x > theta, 1, -1)
d <- rank(abs(x - theta))
rs <- sign*d
t <- sum(ifelse(rs>0, rs, 0))
A <- gtools::permutations(2, length(x), v=c(-1,1), repeats.allowed = T)
A <- t( t(A) * 1:length(x))
A <- ifelse(A > 0, A, 0)
S <- table(apply(A,1,sum))
plot(S)
```



```r
Sp <- prop.table(S)[1:40] # same as page 50 table 3.2
Sp
```

```
            0            1            2            3            4
0.0002441406 0.0002441406 0.0002441406 0.0004882812 0.0004882812
            5            6            7            8            9
0.0007324219 0.0009765625 0.0012207031 0.0014648438 0.0019531250
           10           11           12           13           14
0.0024414062 0.0029296875 0.0036621094 0.0041503906 0.0048828125
           15           16           17           18           19
0.0058593750 0.0065917969 0.0075683594 0.0087890625 0.0097656250
           20           21           22           23           24
0.0109863281 0.0124511719 0.0136718750 0.0148925781 0.0163574219
           25           26           27           28           29
0.0175781250 0.0190429688 0.0205078125 0.0217285156 0.0229492188
           30           31           32           33           34
0.0244140625 0.0253906250 0.0263671875 0.0275878906 0.0280761719
           35           36           37           38           39
0.0288085938 0.0295410156 0.0297851562 0.0300292969 0.0302734375
```

```r
library(exactRankTests)
# Example 3.2
x <- c(-2,4,8,25,-5,16,3,1,12,17,20,9)
d <- x -15
si <- sign(d)
r <- rank(abs(d))
sum(r[si > 0]); sum(r[si < 0]) # S+ and S-
```

```
## [1] 14
```

```
## [1] 64
```

```
p <- wilcox.test(x, mu=15)$p.value
p
```

```
## [1] 0.05224609
```

```
psignrank(14, 12) * 2 # same p value, two-sided
```

```
## [1] 0.05224609
```

Programming the Walsh averages:

```
x <- c(-2,4,8,25,-5,16,3,1,12,17,20,9)
y <- sort(x)
# creating Table 3.3 on page 54:
w <- outer(y,y,"+")/2
row.names(w) <- y; colnames(w) <- y
w
```

```
##      -5   -2    1    3    4    8    9   12   16   17   20   25
## -5 -5.0 -3.5 -2.0 -1.0 -0.5  1.5  2.0  3.5  5.5  6.0  7.5 10.0
## -2 -3.5 -2.0 -0.5  0.5  1.0  3.0  3.5  5.0  7.0  7.5  9.0 11.5
## 1  -2.0 -0.5  1.0  2.0  2.5  4.5  5.0  6.5  8.5  9.0 10.5 13.0
## 3  -1.0  0.5  2.0  3.0  3.5  5.5  6.0  7.5  9.5 10.0 11.5 14.0
## 4  -0.5  1.0  2.5  3.5  4.0  6.0  6.5  8.0 10.0 10.5 12.0 14.5
## 8   1.5  3.0  4.5  5.5  6.0  8.0  8.5 10.0 12.0 12.5 14.0 16.5
## 9   2.0  3.5  5.0  6.0  6.5  8.5  9.0 10.5 12.5 13.0 14.5 17.0
## 12  3.5  5.0  6.5  7.5  8.0 10.0 10.5 12.0 14.0 14.5 16.0 18.5
## 16  5.5  7.0  8.5  9.5 10.0 12.0 12.5 14.0 16.0 16.5 18.0 20.5
## 17  6.0  7.5  9.0 10.0 10.5 12.5 13.0 14.5 16.5 17.0 18.5 21.0
## 20  7.5  9.0 10.5 11.5 12.0 14.0 14.5 16.0 18.0 18.5 20.0 22.5
## 25 10.0 11.5 13.0 14.0 14.5 16.5 17.0 18.5 20.5 21.0 22.5 25.0
```

```
# ?SignRank, ?dsignrank
par(mfrow = c(2,2))
for(n in c(4,5,10,40)) {
  x <- seq(0, n*(n+1)/2, length = 501)
  plot(x, dsignrank(x, n = n), type = "l",
       main = paste0("dsignrank(x, n = ", n, ")"))
}
```

```
# running the van Waerden test in R with one-sample
install.packages("snpar")
library(snpar)
ns.test(x, q=theta, alternative = "greater")
```

9/16/16

```r
before <- c(51.2,46.5,24.1,10.2,65.3,92.1,30.3,49.2)
after <- c(45.8,41.3,15.8,11.1,58.5,70.3,31.6,35.4)
d <- after - before
wilcox.test(d, conf.int = T, alternative = "less")
```

```
##
##  Wilcoxon signed rank test
##
## data:  d
## V = 3, p-value = 0.01953
## alternative hypothesis: true location is less than 0
## 95 percent confidence interval:
##   -Inf -2.15
## sample estimates:
## (pseudo)median
##           -6.6
```

```r
r <- rank(abs(d)); s <- sign(d)
sum(r[s>0])
```

```
## [1] 3
```

```r
psignrank(sum(r[s>0]), length(r), lower.tail = T) # same p value
```

```
## [1] 0.01953125
```

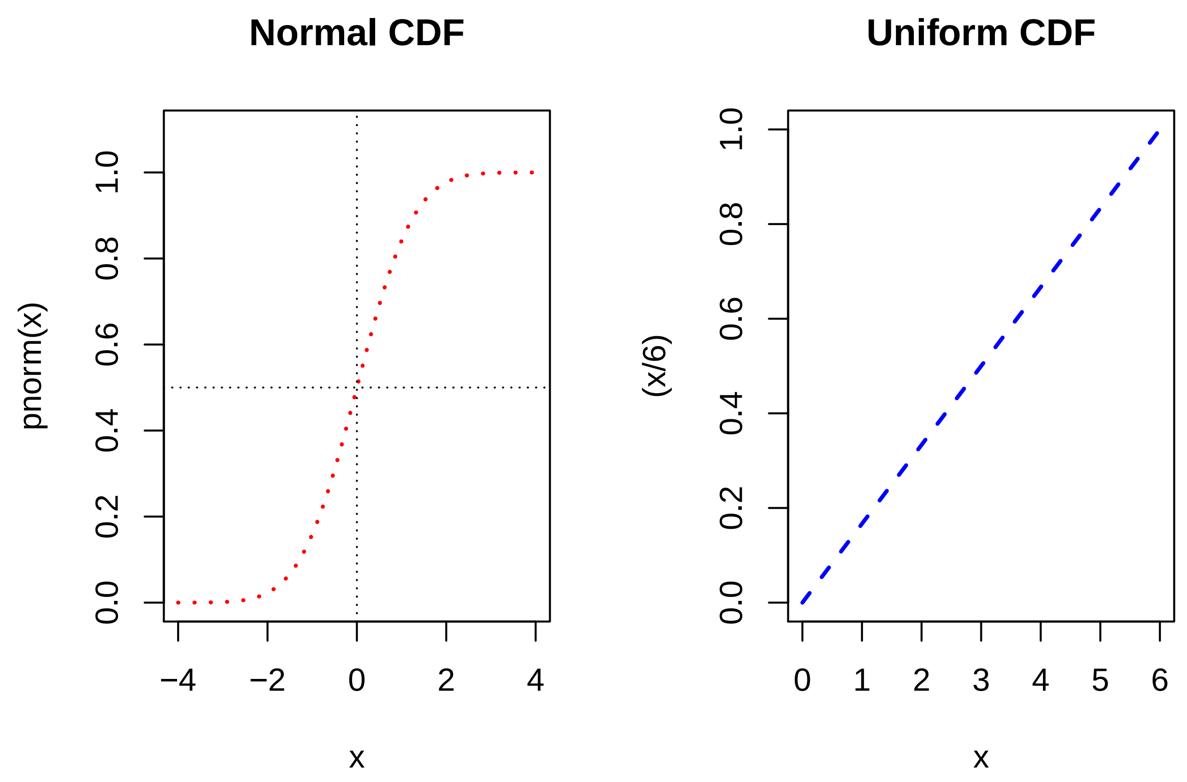
```r
wilcox.test(d, conf.int = T, alternative = "less")$p.value
```

```
## [1] 0.01953125
```

Chapter 4

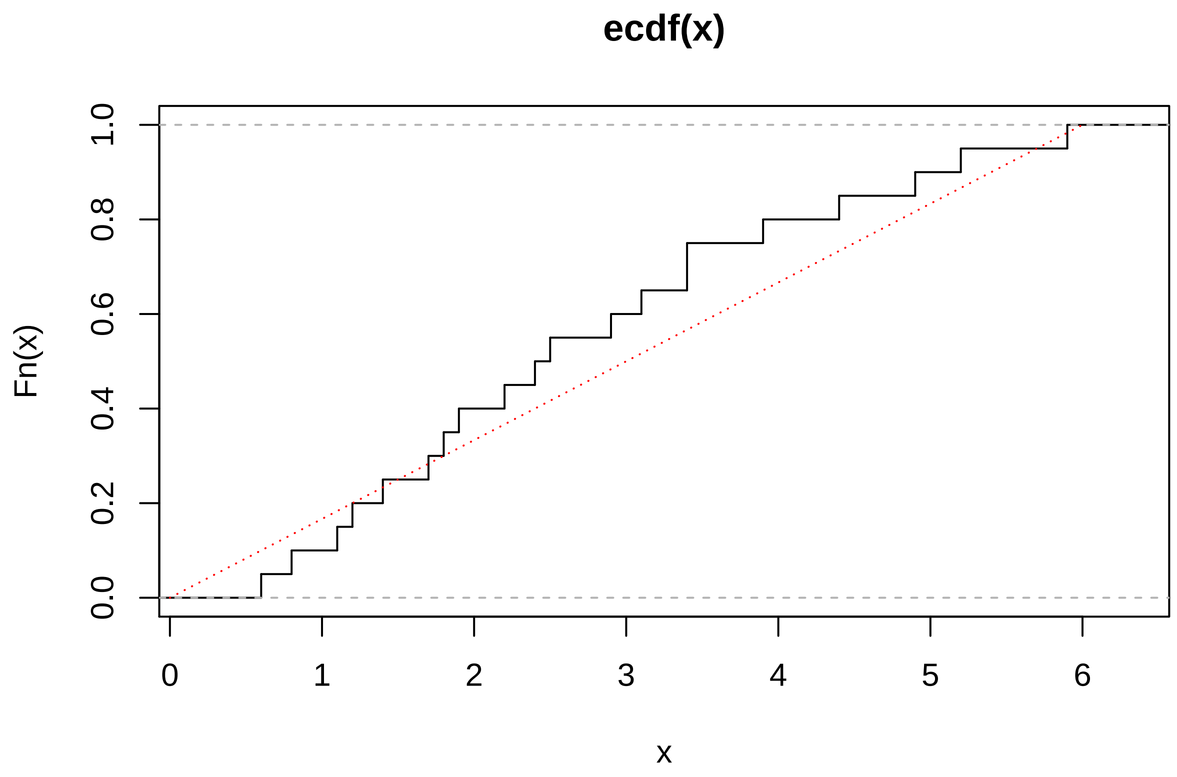
Cumulative Distribution Function (CDF)

```r
par(mfrow=c(1,2))
# uniform CDF for a=0 and b=6
curve(pnorm(x), lty=3, col = "red", lwd=2, from = -4,to=4,
      xlim=c(-4,4), ylim=c(0,1.1), main="Normal CDF")
abline(v=0, h=0.5, lty=3)
curve((x/6), 0, 6, col="blue", lty=2, lwd=2, main="Uniform CDF") #Uniform curve
```



Example 4.1 on page 86 of the textbook

```
x <- c(0.6,0.8,1.1,1.2,1.4,1.7,1.8,1.9,2.2,2.4,2.5,2.9,3.1,3.4,3.4,
       3.9,4.4,4.9,5.2,5.9)
# plot the empircal CDF of our observed data
plot(ecdf(x), verticals = T, pch="")
# overlay theoretical cdf of uniform dist
C <- curve((x/6), 0, 6, col="red", lty=3, add=T)
```



Is the uniform distribution a good "fit" for the data?

Example 4.2

```r
f <- function(x){
  i <- 1:length(x)
  return(i/length(x))
}
x <- sort(x)
Fx <- punif(x, 0, 6)
Sx <- (1:length(x))/length(x)
df <- data.frame(x, Fx = punif(x, 0, 6), Sx, Diff = Fx-f(x),
                 Diff.1 = Fx - lag(Sx) + Sx[1])
knitr::kable(df) #knit the table to appear "nice"
```

x	Fx	Sx	Diff	Diff.1
0.6	0.1000000	0.05	0.0500000	0.1000000
0.8	0.1333333	0.10	0.0333333	0.0833333
1.1	0.1833333	0.15	0.0333333	0.0833333
1.2	0.2000000	0.20	0.0000000	0.0500000
1.4	0.2333333	0.25	-0.0166667	0.0333333
1.7	0.2833333	0.30	-0.0166667	0.0333333
1.8	0.3000000	0.35	-0.0500000	0.0000000
1.9	0.3166667	0.40	-0.0833333	-0.0333333
2.2	0.3666667	0.45	-0.0833333	-0.0333333
2.4	0.4000000	0.50	-0.1000000	-0.0500000

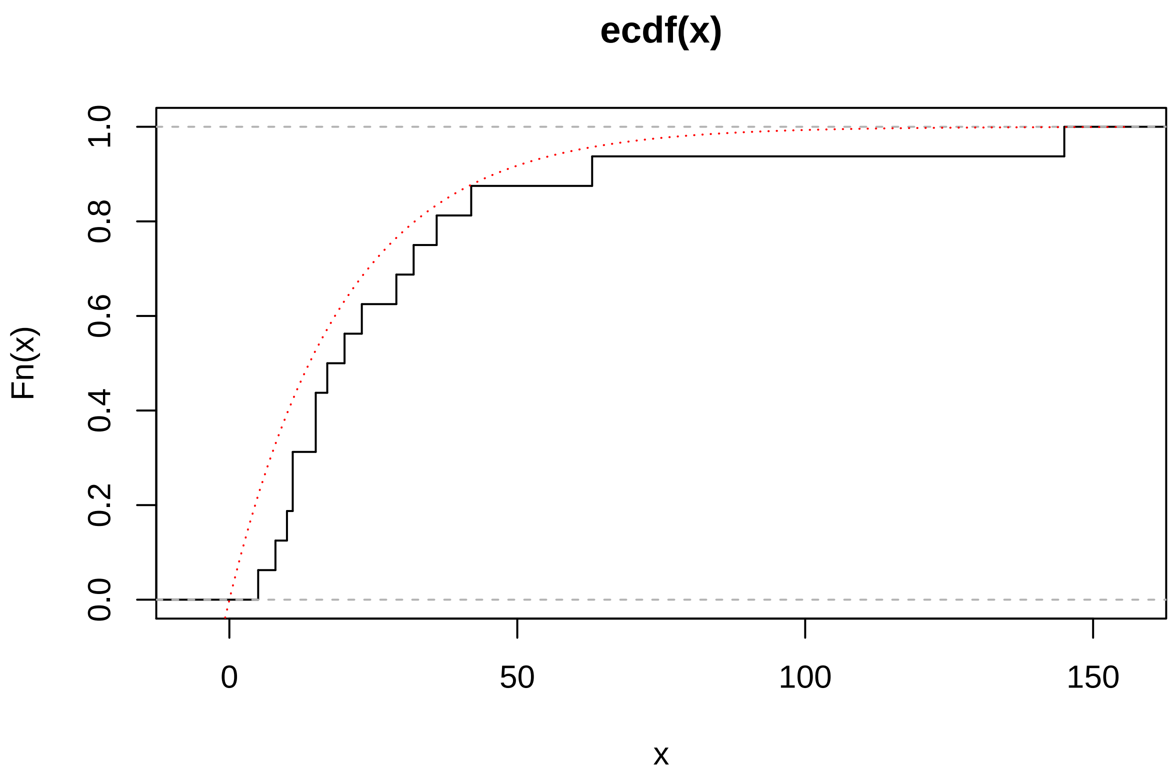
x	Fx	Sx	Diff	Diff.1
2.5	0.4166667	0.55	-0.1333333	-0.0833333
2.9	0.4833333	0.60	-0.1166667	-0.0666667
3.1	0.5166667	0.65	-0.1333333	-0.0833333
3.4	0.5666667	0.70	-0.1333333	-0.0833333
3.4	0.5666667	0.75	-0.1833333	-0.1333333
3.9	0.6500000	0.80	-0.1500000	-0.1000000
4.4	0.7333333	0.85	-0.1166667	-0.0666667
4.9	0.8166667	0.90	-0.0833333	-0.0333333
5.2	0.8666667	0.95	-0.0833333	-0.0333333
5.9	0.9833333	1.00	-0.0166667	0.0333333

```r
max(abs(f(x) - Fx)) # max diff
```

```
## [1] 0.1833333
```

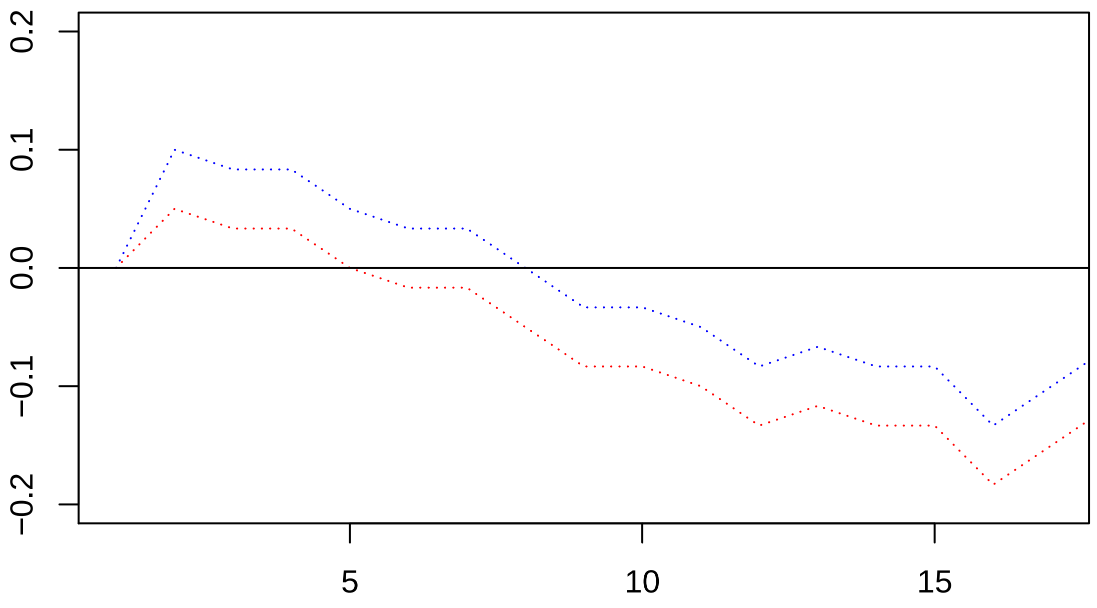
In Class 9/21/16

```r
# Exercise 4.2
x <- c(10,42,29,11,63,145,11,8,23,17,5,20,15,36,32,15)
plot(ecdf(x), pch = "", verticals = TRUE)
curve((1 - exp(-(x/20))), lty=3, col="red", add=T) #exponential
```



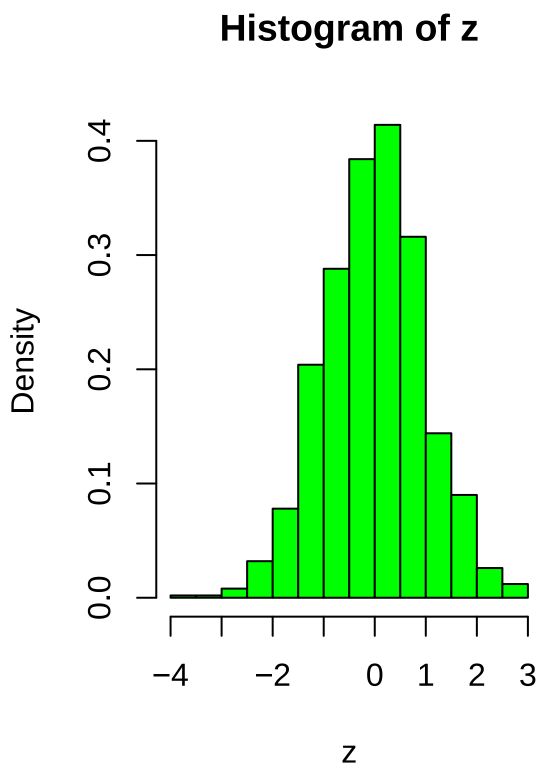
Reproducing figure 4.6 on page 91:

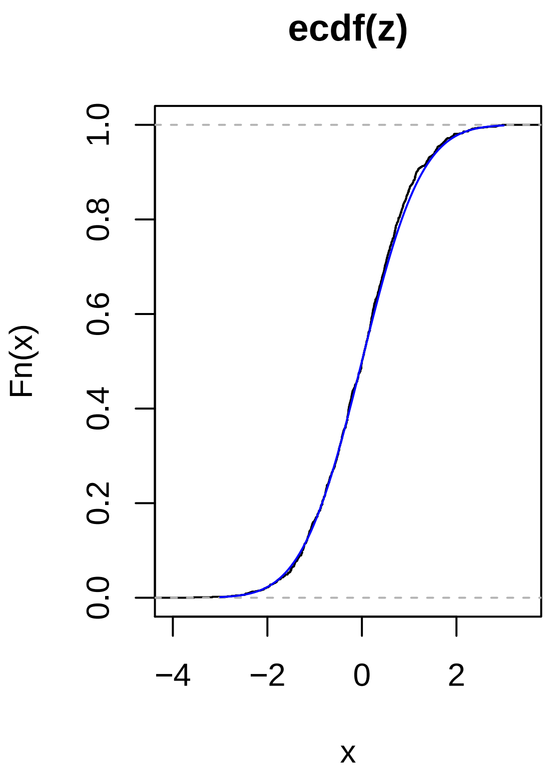
```r
f <- function(x){
  if(x>0 & x<=3){
    y <- x/5}
  y <- 0.2 + 4*x/30
  return(x)
}
plot(c(0,df$Diff), type="l", col="red", lty=3, xlim=c(1, length(x)+1),
     ylim=c(-.2, .2), xlab = "", ylab = "")
lines(c(0,df$Diff.1), lty = 3, col = "blue")
abline(h=0)
```



Relation between CDF and Uniform Distribution

```r
# generate data from standard normal, this is z
z <- rnorm(1000)
# plot the histogram and the CDF:
par(mfrow=c(1,2))
hist(z, freq = F, col="green")
plot(ecdf(z))
curve(pnorm(x), -3, 3, add=T, col = "blue")
```

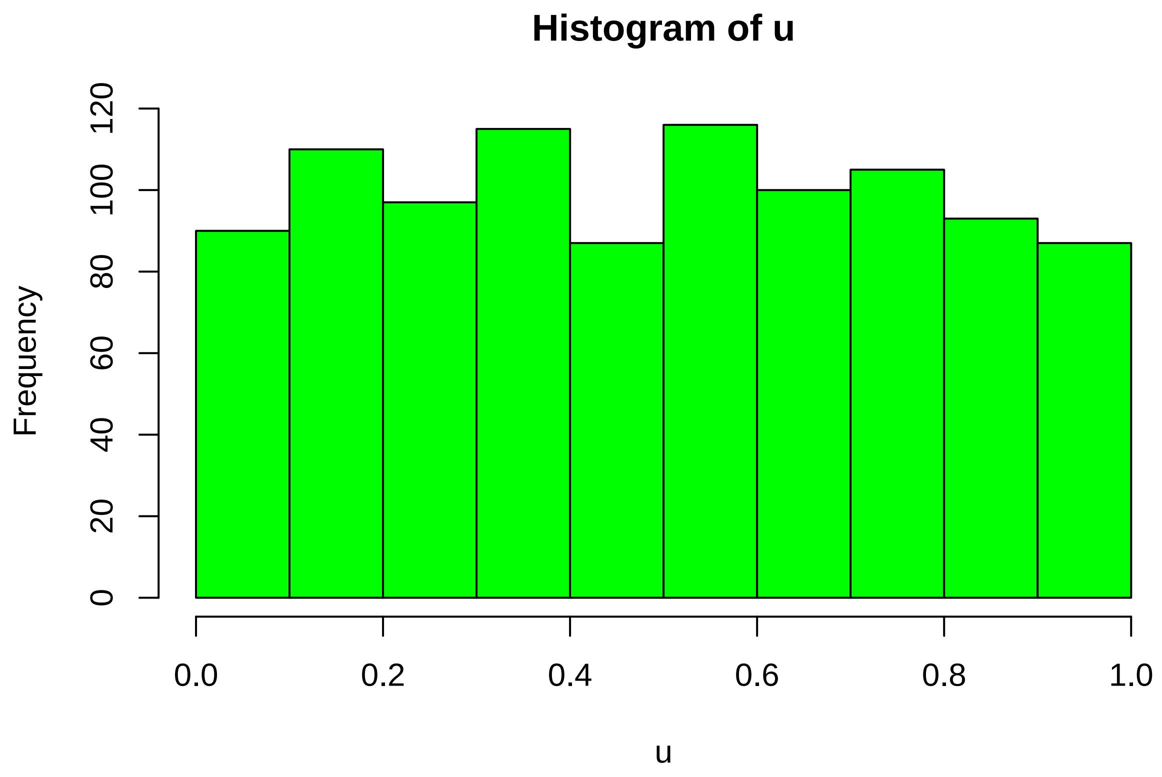




Notice how closely the empirical CDF of our random normal data follows that of the theoretical CDF of the standard normal.

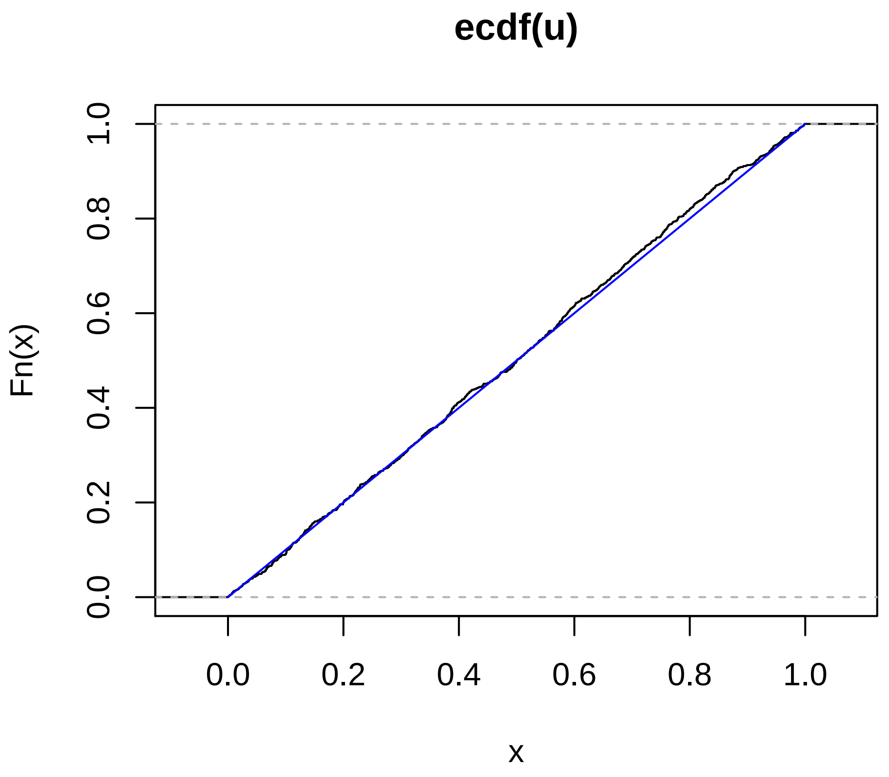
Now to look at $\Phi(z) = u$

```r
u <- pnorm(z) # using R's built in functions to compute u
hist(u, col="green")
```



who’s historgram appears to have the same shape as that of a uniform distribution from 0 to 1. Now to plot the CDF of u with the theoretical CDF of the uniform distribution from 0 to 1.

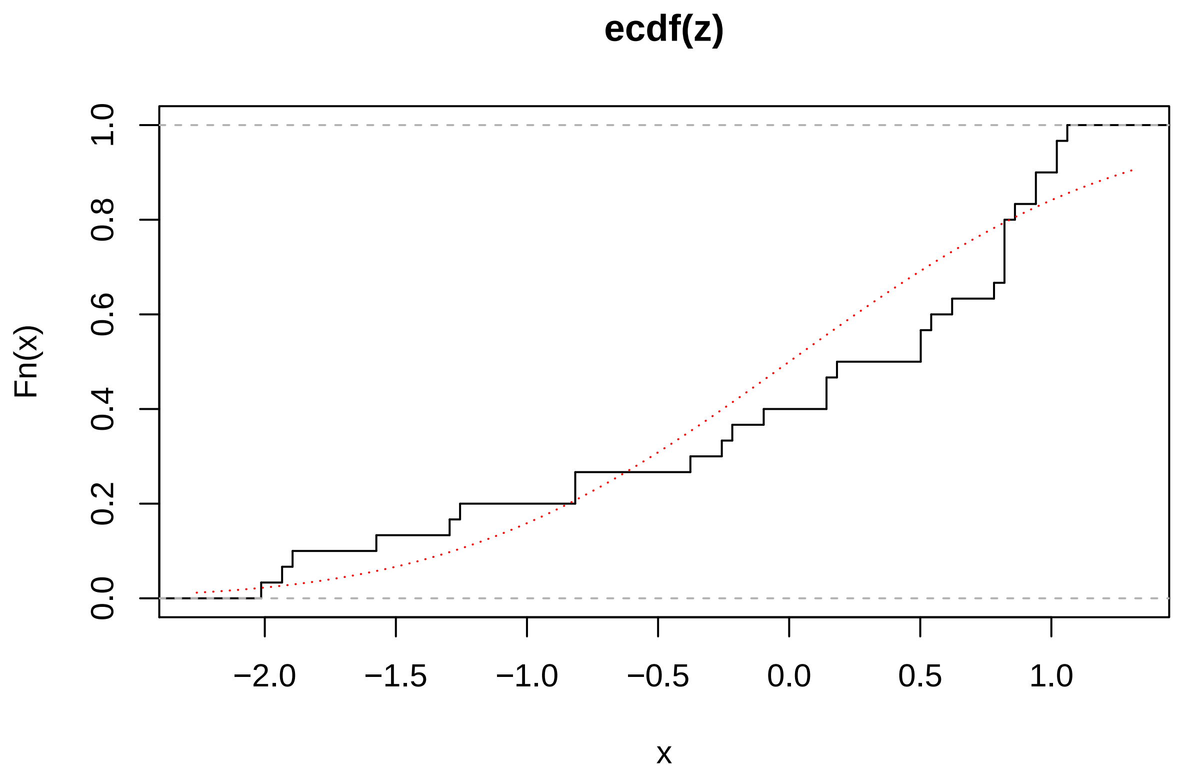
```r
plot(ecdf(u))
curve(punif(x), 0, 1, add=T, col = "blue")
```



And we can see how closely u follows a uniform distribution from 0 to 1 •

Example 4.4 (page 94)

```
x <- c(11,13,14,22,29,30,41,41,52,55,56,59,65,65,66,74,74,75,77,
       81,82,82,82,82,83,85,85,87,87,88)
# are these data from a normal distribution?
# is age at death normally distributed?
v <- ecdf(x)
Sx <- v(x)
z <- (x - mean(x))/sd(x)
plot(ecdf(z), pch="", verticals = TRUE)
curve(pnorm(x), add=TRUE, col="red", lty=3)
```



```
Pz <- pnorm(z)
df <- data.frame(x, z, Pz, Sx, Diff = Pz - Sx,
                 Diff.1 = Pz - c(0,Sx[-length(Sx)]))
test <- shapiro.test(x)
knitr::kable(df)
```

x	z	Pz	Sx	Diff	Diff.1
11	-2.0138715	0.0220115	0.0333333	-0.0113218	0.0220115
13	-1.9340088	0.0265560	0.0666667	-0.0401106	-0.0067773
14	-1.8940774	0.0291074	0.1000000	-0.0708926	-0.0375593
22	-1.5746266	0.0576713	0.1333333	-0.0756620	-0.0423287
29	-1.2951071	0.0976416	0.1666667	-0.0690250	-0.0356917

x	z	Pz	Sx	Diff	Diff.1
30	-1.2551757	0.1047075	0.2000000	-0.0952925	-0.0619592
41	-0.8159308	0.2072699	0.2666667	-0.0593968	0.0072699
41	-0.8159308	0.2072699	0.2666667	-0.0593968	-0.0593968
52	-0.3766858	0.3532036	0.3000000	0.0532036	0.0865369
55	-0.2568917	0.3986312	0.3333333	0.0652978	0.0986312
56	-0.2169604	0.4141196	0.3666667	0.0474529	0.0807863
59	-0.0971663	0.4612972	0.4000000	0.0612972	0.0946305
65	0.1424218	0.5566266	0.4666667	0.0899599	0.1566266
65	0.1424218	0.5566266	0.4666667	0.0899599	0.0899599
66	0.1823532	0.5723472	0.5000000	0.0723472	0.1056806
74	0.5018041	0.6920973	0.5666667	0.1254307	0.1920973
74	0.5018041	0.6920973	0.5666667	0.1254307	0.1254307
75	0.5417354	0.7059996	0.6000000	0.1059996	0.1393329
77	0.6215982	0.7328969	0.6333333	0.0995636	0.1328969
81	0.7813236	0.7826939	0.6666667	0.1160272	0.1493606
82	0.8212549	0.7942495	0.8000000	-0.0057505	0.1275828
82	0.8212549	0.7942495	0.8000000	-0.0057505	-0.0057505
82	0.8212549	0.7942495	0.8000000	-0.0057505	-0.0057505
82	0.8212549	0.7942495	0.8000000	-0.0057505	-0.0057505
83	0.8611863	0.8054323	0.8333333	-0.0279011	0.0054323
85	0.9410490	0.8266601	0.9000000	-0.0733399	-0.0066732
85	0.9410490	0.8266601	0.9000000	-0.0733399	-0.0733399
87	1.0209117	0.8463519	0.9666667	-0.1203148	-0.0536481
87	1.0209117	0.8463519	0.9666667	-0.1203148	-0.1203148
88	1.0608431	0.8556194	1.0000000	-0.1443806	-0.1110473

```r
# removing repeated values like the books table:
knitr::kable(df[!duplicated(df$x), ])
```

	x	z	Pz	Sx	Diff	Diff.1
1	11	-2.0138715	0.0220115	0.0333333	-0.0113218	0.0220115
2	13	-1.9340088	0.0265560	0.0666667	-0.0401106	-0.0067773
3	14	-1.8940774	0.0291074	0.1000000	-0.0708926	-0.0375593
4	22	-1.5746266	0.0576713	0.1333333	-0.0756620	-0.0423287
5	29	-1.2951071	0.0976416	0.1666667	-0.0690250	-0.0356917
6	30	-1.2551757	0.1047075	0.2000000	-0.0952925	-0.0619592
7	41	-0.8159308	0.2072699	0.2666667	-0.0593968	0.0072699
9	52	-0.3766858	0.3532036	0.3000000	0.0532036	0.0865369
10	55	-0.2568917	0.3986312	0.3333333	0.0652978	0.0986312
11	56	-0.2169604	0.4141196	0.3666667	0.0474529	0.0807863
12	59	-0.0971663	0.4612972	0.4000000	0.0612972	0.0946305
13	65	0.1424218	0.5566266	0.4666667	0.0899599	0.1566266
15	66	0.1823532	0.5723472	0.5000000	0.0723472	0.1056806
16	74	0.5018041	0.6920973	0.5666667	0.1254307	0.1920973
18	75	0.5417354	0.7059996	0.6000000	0.1059996	0.1393329
19	77	0.6215982	0.7328969	0.6333333	0.0995636	0.1328969
20	81	0.7813236	0.7826939	0.6666667	0.1160272	0.1493606
21	82	0.8212549	0.7942495	0.8000000	-0.0057505	0.1275828
25	83	0.8611863	0.8054323	0.8333333	-0.0279011	0.0054323
26	85	0.9410490	0.8266601	0.9000000	-0.0733399	-0.0066732
28	87	1.0209117	0.8463519	0.9666667	-0.1203148	-0.0536481

	x	z	Pz	Sx	Diff	Diff.1
30	88	1.0608431	0.8556194	1.0000000	-0.1443806	-0.1110473

```r
# the test statistic:
max(c(df$Diff, df$Diff.1)) #lillies test statistic
```

```
## [1] 0.1920973
```

```r
library(nortest)
lillie.test(x) # using a function for lillies test found on CRAN
```

```
##
##  Lilliefors (Kolmogorov-Smirnov) normality test
##
## data:  x
## D = 0.1921, p-value = 0.006246
```

As can be seen above, at least our test statistics match. Since the book doesn't go into great detail about p-values for this test I do not know where to begin in computing the significance of our statistic. lillie.test() from the nortest package computes this value for us. How "correct" it is I do not know. However, it does reject normality for our sample, which agrees with shapiro-wilk and intuition.

I'm not sure what to think of the textbook at this point. It provides very little detail about how to perform tests, except to say "Stat Exact provides. . . ..". Since we do not have accesss to "Stat Exact" we cannot verify the same results. Having said that, for the values above, I do not get the same p-value when running the Shapiro-Wilk test as the book does. Perhaps it's a typo? The p-value it gives for the liilie test is also not very small and would not lead to rejection of the null hypothesis that the data is from the normal distribution.

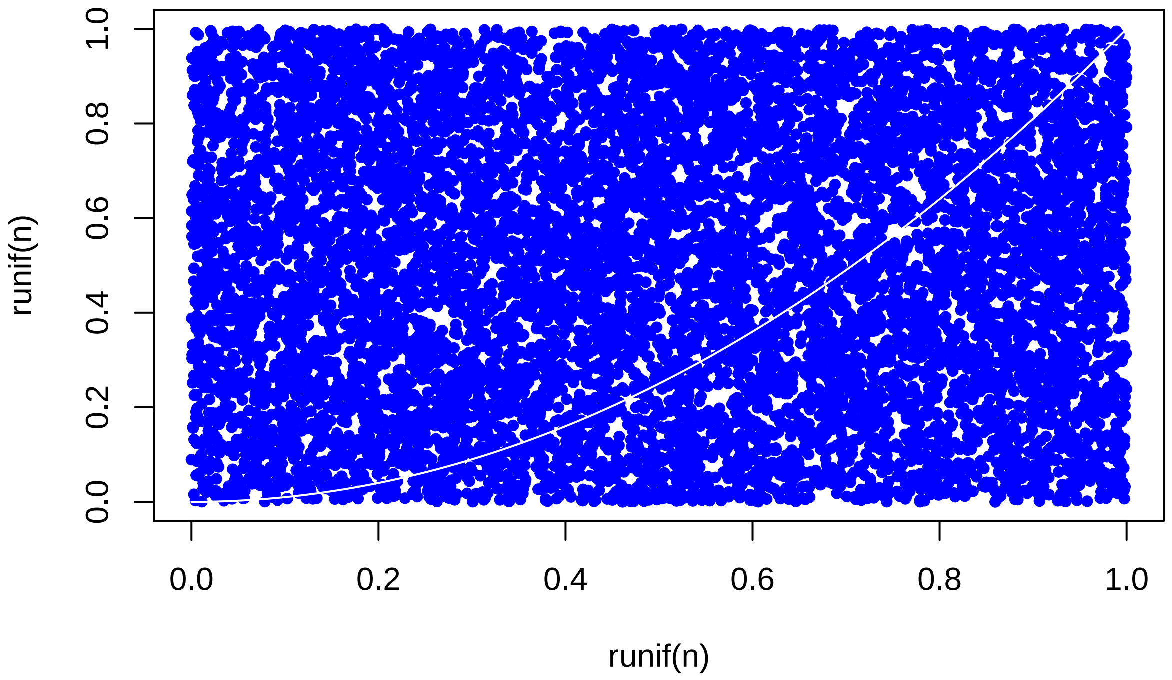
```r
shapiro.test(x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.86294, p-value = 0.001171
```

Monte Carlo approximations:

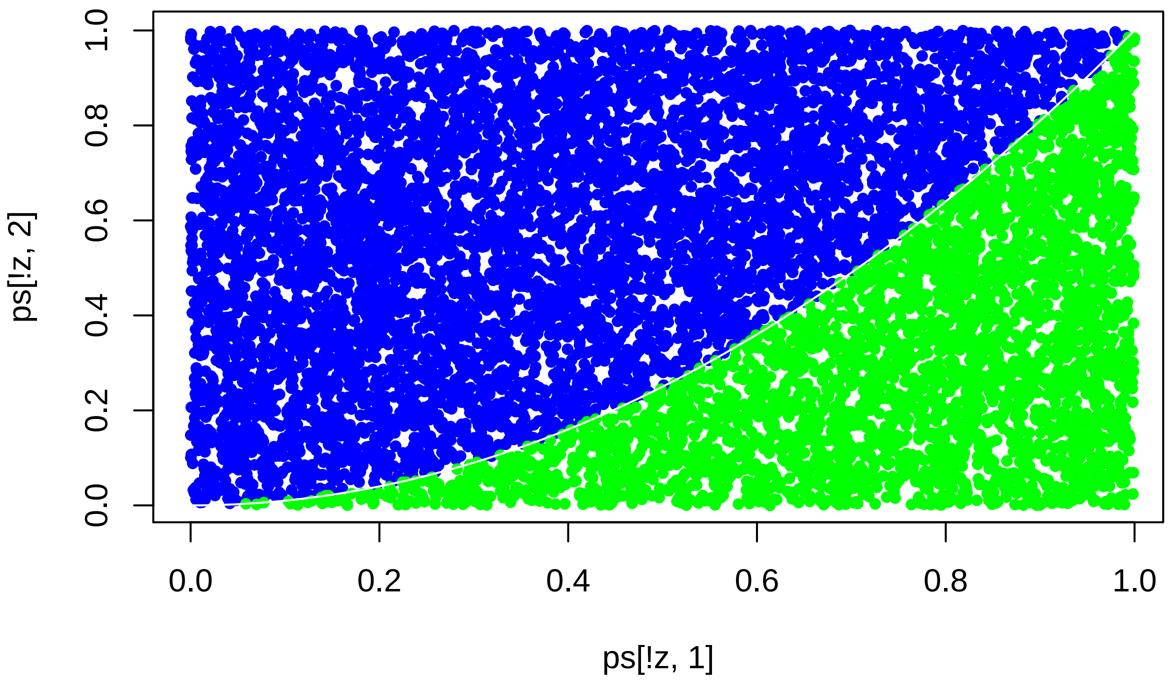
Random points from 0 to 1 with the function $y = x^2$ represented by the white curve. Here, we are approximating the integral of $y = x^2$ by computing the probability of a point falling under the white curve. All points have equal probability in the box pictured below.

```r
n <- 10000
f <- function(x) x^2
plot(runif(n), runif(n), col='blue', pch=20)
curve(f, 0,1, n=100, col='white', add=TRUE)
```



The points under the curve are the ones's we care about:

```
ps <- matrix(runif(2*n), ncol=2) #2x1 matrix with random uniforms
g <- function(x,y) y <= x^2 # logical function
z <- g(ps[,1], ps[,2]) # logical vector
plot(ps[!z,1], ps[!z,2], col='blue', pch=20) # plot above curve
points(ps[z,1], ps[z,2], col='green', pch=20) # plot below curve
curve(f, 0,1, n=100, col='white', add=TRUE) # white curve
```



How many green dots are there as a proportion of total dots?

```r
length(z[z])/n
```

```
## [1] 0.335
```

This is an estimation of $\int_0^1 x^2 dx$ that gets more and more exact to n places as we increase n in the monte carlo simulation.

```r
# actual integral of y = x^2
integrate(f = function(x){x^2}, 0, 1)
```

```
## 0.3333333 with absolute error < 3.7e-15
```

This is pretty cool. However, I have not yet figured out how to do this for a sample, or a small-ish data set where I don't have some continuous funtion of x. All I can estimate is x's CDF and I have not yet figured out how to do this same estimate for a p-value from a CDF.

Tests for Randomness

```r
library(randtests)
# H = 1, T = 0
x <- c(rep(1, 5), rep(0, 10), rep(1, 5))
runs.test(x)
```

```
##
##  Runs Test
##
## data:  x
## statistic = -3.6757, runs = 3, n1 = 10, n2 = 10, n = 20, p-value =
## 0.0002372
## alternative hypothesis: nonrandomness
```

```r
y <- rep(c(1,0), 10)
runs.test(y)
```

```
##
##  Runs Test
##
## data:  y
## statistic = 4.1352, runs = 20, n1 = 10, n2 = 10, n = 20, p-value =
## 3.546e-05
## alternative hypothesis: nonrandomness
```

```r
z <- c(1,1,0,1,0,0,0,1,0,1,1,0,1,0,1,1,1,0,0,1)
runs.test(z)
```

```
##
##  Runs Test
##
## data:  z
## statistic = NaN, runs = 1, n1 = 0, n2 = 9, n = 9, p-value = NA
## alternative hypothesis: nonrandomness
```

```
# example 4.15 on page 113
x <- c("A","A","A","A","B","A","C","A","A","A","A","A","D","D","A","A")
x <- c(1,1,1,1,2,1,3,1,1,1,1,1,4,4,1,1)
k <- 4; N <- 17; n1 <- 12; n2 <- 1; n3 <- 2; n4 <- 2; R <-7
p <- c(n1,n2,n3,n4)/N
eR <- N*(1 - sum(p^2)) + 1
vR <- N*(sum(p^2 - 2*p^3) + sum(p^2)^2)
z <- (R - eR)/sqrt(vR)
pnorm(z) #Asymptotic normal p value
```

```
## [1] 0.061889
```

Chapter 6

10/5/16

Wilcoxon-Mann-Whitney Test (WMW)

$$H_0 : F_x(x) = F_y(x) \quad vs \quad F_x(x) \neq F_y(x)$$

```
before <- c(12.6,11.4,13.2,11.2,9.4,12)
after <- c(16.4,14.1,13.4,15.4,14,11.3)
xi <- c(before, after)
ix <- 1:length(before)
r <- rank(xi)
sum(r[ix])
```

```
## [1] 25
```

```
sum(r[-ix])
```

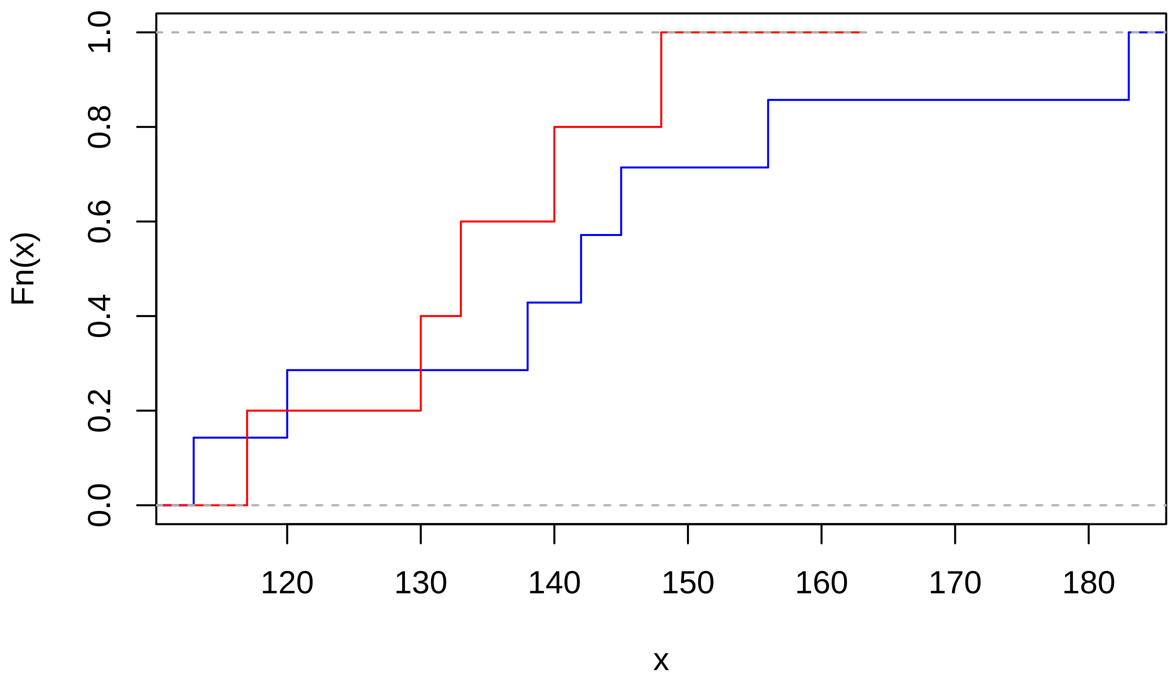
```
## [1] 53
```

```
wilcox.test(before, after, alternative = "less")
```

```
##
##  Wilcoxon rank sum test
##
## data:  before and after
## W = 4, p-value = 0.01299
## alternative hypothesis: true location shift is less than 0
```

10/7/16 Wilcoxon Summed Rank Test

```r
a <- c(156,183,120,113,138,145,142); n <- length(a)
b <- c(130,148,117,133,140); m <- length(b)
df <- data.frame(x = c(a,b), diet = c(rep("a", n), rep("b", m)))
df$rank <- rank(df$x)
wa <- sum(df$rank[df$diet == "a"])
wb <- sum(df$rank[df$diet == "b"])
W <- wb - (m*(m+1))/2
plot(ecdf(a), verticals = TRUE, pch="", xlim=c(min(c(a,b)), max(c(a,b))),
     col="blue", main="")
lines(ecdf(b), verticals=TRUE, pch="", col="red")
```



```r
x <- c(1,6,7); m <- length(x)
y <- c(2,4,9,10,12); n <- length(y)
df <- data.frame(x = c(x,y), diet = c(rep("x", m), rep("y", n)))
df$rank <- rank(df$x)
wx <- sum(df$rank[df$diet == "x"])
wy <- sum(df$rank[df$diet == "y"])
W <- wx - (m*(m+1))/2
pwilcox(W,m,n)*2 # for two sided
```

```
## [1] 0.3928571
```

```r
wilcox.test(x,y, conf.int = TRUE, conf.level = 0.9) # they agree
```

```
##
##  Wilcoxon rank sum test
##
## data:  x and y
## W = 4, p-value = 0.3929
## alternative hypothesis: true location shift is not equal to 0
## 90 percent confidence interval:
##  -9  4
## sample estimates:
## difference in location
##                     -3
```

$$E(W) = \frac{n(m+n+1)}{2}$$
$$V(W) = \frac{mn(m+n+1)}{12}$$

Normal approximation:

$$Z = \frac{W_{mn} - E(W)}{\sqrt{V(W)}}$$

Example from page 154

```r
x <- c(17,18,19,22,23,25,26,29,31,33)
y <- c(21,24,27,28,30,32,34,35,36,39,41)
m <- length(x); n <- length(y)
Mx <- outer(x, y, "<")
Ux <- sum(apply(Mx, 1, sum))
My <- outer(y, x, "<")
Uy <- sum(apply(My, 1, sum))
min(c(Ux,Uy)) # test statistic
```

```
## [1] 21
```

```r
pwilcox(min(c(Uy,Ux)), m, n)*2 # same p-value as in the book
```

```
## [1] 0.01587113
```

```r
test <- wilcox.test(x,y, conf.int = TRUE)
test # same result
```

```
##
##  Wilcoxon rank sum test
##
## data:  x and y
## W = 21, p-value = 0.01587
## alternative hypothesis: true location shift is not equal to 0
## 95 percent confidence interval:
##  -13  -2
## sample estimates:
## difference in location
##                   -7.5
```

Setting up the walsh table to create a confidence interval (page 155-156)

```r
# data from example 6.1, which we already have from the above examle
M <- t(outer(sort(x), sort(y), "-"))
row.names(M) <- sort(y); colnames(M) <- sort(x)
M
```

```
##     17  18  19  22  23  25  26  29  31 33
## 21  -4  -3  -2   1   2   4   5   8  10 12
## 24  -7  -6  -5  -2  -1   1   2   5   7  9
## 27 -10  -9  -8  -5  -4  -2  -1   2   4  6
## 28 -11 -10  -9  -6  -5  -3  -2   1   3  5
## 30 -13 -12 -11  -8  -7  -5  -4  -1   1  3
## 32 -15 -14 -13 -10  -9  -7  -6  -3  -1  1
## 34 -17 -16 -15 -12 -11  -9  -8  -5  -3 -1
## 35 -18 -17 -16 -13 -12 -10  -9  -6  -4 -2
## 36 -19 -18 -17 -14 -13 -11 -10  -7  -5 -3
## 39 -22 -21 -20 -17 -16 -14 -13 -10  -8 -6
## 41 -24 -23 -22 -19 -18 -16 -15 -12 -10 -8
```

```r
# note:  We could also get our test stat from this matrix...
s <- (sign(M))
sum(s < 0)
```

```
## [1] 89
```

```r
sum(s > 0)
```

```
## [1] 21
```

```r
# values for our confidence interval.  Note, 27 = 26 + 1. 26 is the determined
# critical value for alpha at least equal to 0.05
cv <- sort(as.vector(M))
cv[27]; sort(cv, decreasing = T)[27]
```

```
## [1] -13
```

```
## [1] -2
```

```r
# comparison to the t test confidence interval:
t.test(x,y)$conf.int
```

```
## [1] -12.606857  -1.884052
## attr(,"conf.level")
## [1] 0.95
```

6.4.1

```
x <- c(177,200,227,230,232,268,272,297)
y <- c(47,105,126,142,158,172,197,220,225,230,262,270)
x <- x - (median(x) - median(y)) #shift x
data <- data.frame(c(x, y), rep(c(0, 1), c(length(x), length(y))))
names(data) = c("x", "y")
sort.x <- sort(data$x)
sort.id <- data$y[order(data$x)]
x <- data$x
y <- data$y
data.matrix <- data.frame(sort.x, sort.id)
base1 <- c(1, 4)
iterator1 <- matrix(seq(from = 1, to = length(x), by = 4)) - 1
rank1 <- apply(iterator1, 1, function(x) x + base1)
iterator2 <- matrix(seq(from = 2, to = length(x), by = 4))
base2 <- c(0, 1)
rank2 <- apply(iterator2, 1, function(x) x + base2)
if (length(rank1) == length(rank2)) {
    rank <- c(rank1[1:floor(length(x)/2)],
              rev(rank2[1:ceiling(length(x)/2)]))
  } else {
    rank <- c(rank1[1:ceiling(length(x)/2)],
              rev(rank2[1:floor(length(x)/2)]))
  }

unique.ranks <- tapply(rank, sort.x, mean)
unique.ranks # same as table 6.3 on page 173
```

```
##    47   105   126 130.5   142 153.5   158   172 180.5 183.5 185.5   197
##     1     4     5     8     9    12    13    16    17    20    19    18
##   220 221.5   225 225.5   230 250.5   262   270
##    15    14    11    10     7     6     3     2
```

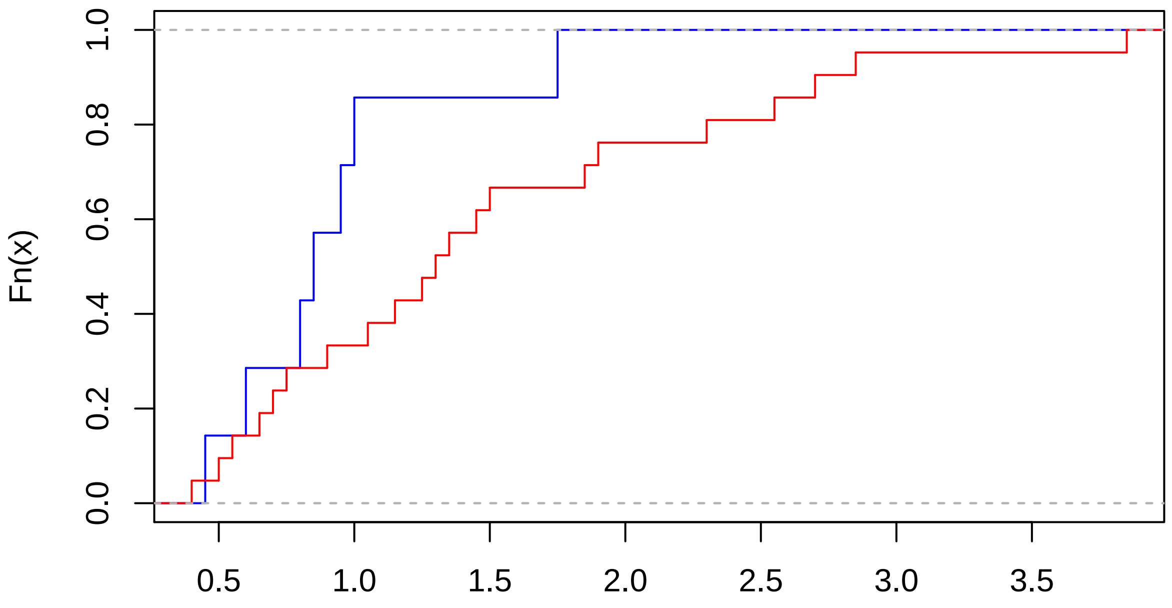
6.5

Test for a common distribution:

```
female <- c(.45,.6,.8,.85,.95,1,1.75)
male <- c(.4,.5,.55,.65,.7,.75,.9,1.05,1.15,1.25,1.3,1.35,1.45,1.5,
          1.85,1.9,2.3,2.55,2.7,2.85,3.85)
test <- ks.test(female, male)
test
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  female and male
## D = 0.52381, p-value = 0.09296
## alternative hypothesis: two-sided
```

```r
x <- ecdf(female)
y <- ecdf(male)
m <- outer(female, male, "<")
plot(ecdf(female), verticals = TRUE, pch="",
     xlim=c(min(c(female,male)), max(c(female,male))),
     col="blue", main="", xlab = "")
lines(ecdf(male), verticals=TRUE, pch="", col="red")
```



10-14-16

Test on 10/16/16 in class will be on:

- Tests of randomness

This is a Runs test. What are some of the alternative hypothesis? One could be clustering, reject if $R < c$. Another could be too much mixing, reject if $R > c$. Another is a trend. Is there a trend among the data, increasing or decreasing? Does a small value always follow a large value? and vice versa. If we have n observations and n-1 plus or minus signs then we would reject the null. Under the null we expect an equal number of plus and minus signs, thus becoming a sign test. Another is runs up and down.

- Test for distribution
  K-S test.

- Two sample median

- Two sample variance

10/19/16

Rest of the semester we will cover chapters 7, 10, 11, 12, and 13.
And maybe a little bit of chapter 14.

10/26/16

Jonckheere-Terpstra Test

```
one <- c(74,58,68,60,69)
two <- c(70,72,75,80,71)
three <- c(73,78,88,85,76)
x <- c(one,two,three)
g <- ordered(factor(c(rep(1,5), rep(2,5),rep(3,5))))
#install.packages("DescTools")
library(DescTools)
JonckheereTerpstraTest(x, g)
```

```
##
##  Jonckheere-Terpstra test
##
## data:  x and g
## JT = 67, p-value = 0.001052
## alternative hypothesis: two.sided
```

```
library(clinfun)
jonckheere.test(x,g) # same result
```

```
##
##  Jonckheere-Terpstra test
##
## data:
## JT = 67, p-value = 0.001052
## alternative hypothesis: two.sided
```

```
# another problem:
x <- c(48,33,59,48,56,60,101,67,85,107)
g <- ordered(c(rep(20,1), rep(25,4), rep(30,3), rep(35, 2)))
JonckheereTerpstraTest(x,g)
```

```
##
##  Jonckheere-Terpstra test
##
## data:  x and g
## JT = 32.5, p-value = 0.004526
## alternative hypothesis: two.sided
```

```
jonckheere.test(x,g)
```

```
##
##  Jonckheere-Terpstra test
##
## data:
## JT = 32.5, p-value = 0.004526
## alternative hypothesis: two.sided
```

10/28/16

```r
x <- c(13,27,26,22,26,43,35,47,32,31,37,33,37,33,26,44,33,54)
g <- factor(c(rep("A", 5), rep("B",6), rep("C", 7)))
test <- kruskal.test(x,g)
test
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  x and g
## Kruskal-Wallis chi-squared = 9.1458, df = 2, p-value = 0.01033
```

Now, nonparametric blocking design. . . .

The model is:

$$y_{ij} = \mu + \theta_i + \beta_j + \epsilon_{ij}$$

and the null hypothesis is That the θ_is are all equal, meaning there is no difference between treatments.

7.2

Example 7.1

```r
x <- c(139,145,171,151,163,188,197,199,250,360)
g <- factor(c(rep("A", 3), rep("B", 4), rep("C", 3)))
kruskal.test(x,g)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  x and g
## Kruskal-Wallis chi-squared = 6.7455, df = 2, p-value = 0.0343
```

```r
# another approach.....almost exact :)
library(coin)
kruskal_test(x ~ g, distribution = approximate(B = 10000))
```

```
##
##  Approximative Kruskal-Wallis Test
##
## data:  x by g (A, B, C)
## chi-squared = 6.7455, p-value = 0.0096
```

```r
# note, this is using monte carlo simulation...the p-value will
# change each time this is run...but only slightly.
```

7.3 The Friedman, Quade and Page Tests

Example 7.6

```
y <- c(72,120,76,96,120,95,88,132,104,92,120,96,74,101,84,76,96,72,82,112,76)
time <- factor(rep(1:3, 7)) # blocking factor
student <- gl(7, 3)
df <- data.frame(y, student, time)
fit <- aov(y ~ time + Error(student), df)
fit.f <- friedman_test(y ~ time | student, data = df,
                       distribution = approximate(B = 100000))
fit.f3 <- friedman.test(y, time, student)
# compare the monte carlo approximation with the chi-squared
fit.f # monte carlo approximation
```

```
##
##  Approximative Friedman Test
##
## data:  y by
##   time (1, 2, 3)
##   stratified by student
## chi-squared = 10.571, p-value = 0.0027
```

```
fit.f3 # chi-squared
```

```
##
##  Friedman rank sum test
##
## data:  y, time and student
## Friedman chi-squared = 10.571, df = 2, p-value = 0.005063
```

```
summary(fit) # parametric with student blocked with Error(student)
```

```
##
## Error: student
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals  6   1860     310
##
## Error: Within
##           Df Sum Sq Mean Sq F value   Pr(>F)
## time       2   4218  2108.9   50.66 1.41e-06 ***
## Residuals 12    500    41.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Chapter 10

Spearman's Correlation Coefficient 11/4/16