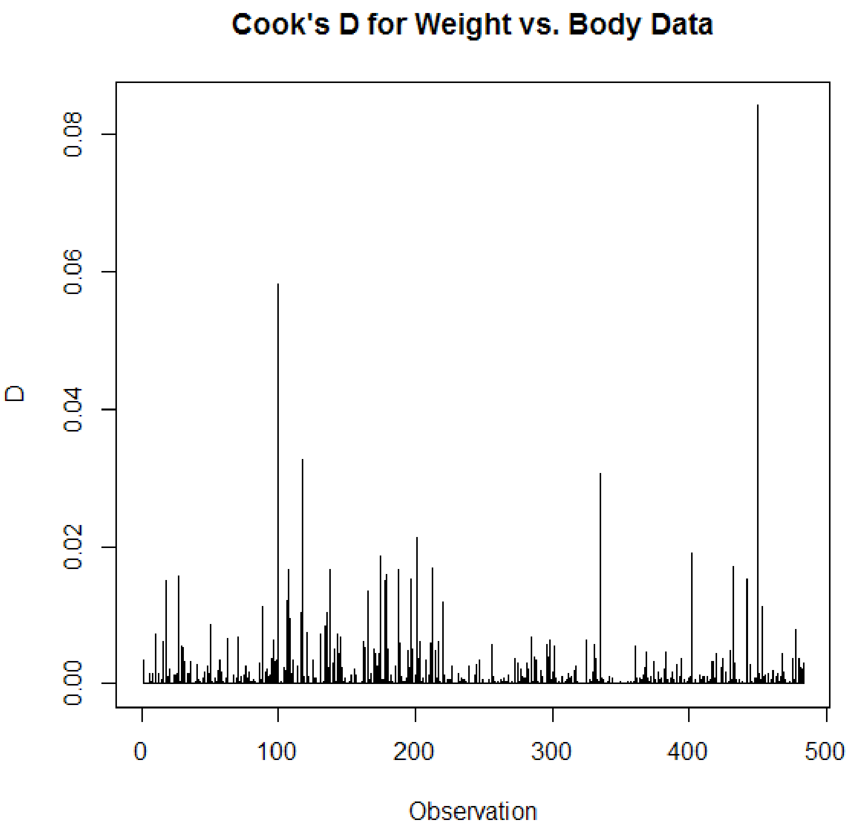
R Odds and Ends

Cook's D

```
# Regression Diagnostics

# Cook's D calculation for each x,y in dataset

# get data
library(RODBC)
db=odbcConnect('st512',uid='sa',pw='')
d=sqlQuery(db,'select * from BodyData')
odbcClose(db)

# create data frame for calculated D's
CooksD=data.frame(observation=integer(0), D=single(0))

# configure design matrix and response vector
# note the column of 1s for intercept
X=as.matrix(cbind(rep(1,nrow(d)), d$age, d$height, d$gender, d$bicep))
Y=as.matrix(d$weight)
n=nrow(X)
p=ncol(X)

# calculate regression parameter and y estimates using all observations
yestFull=X%*%solve(t(X)%*%X)%*%t(X)%*%Y

# save full model MSE, we'll need it later
MSE=t(Y-yestFull)%*%(Y-yestFull)/(n-p-1)

# calculate D index for each observation
# effectively, this is the ratio of
# [the sum of squared deviations of full model y-estimates
#  and those from y estimates using partial model (an estimate for each observation
#  but using the model generated without ith observation)]
# and
# [MSE * the number of predictors]
i=1
while(i<=n)
  { if(i==1)
      { # omit first observation
        X2=as.matrix(X[2:n,])
        Y2=as.matrix(Y[2:n])
      }
    else if(i<n)
      { # omit ith row
        X2=as.matrix(rbind(X[1:(i-1),], X[(i+1):n,]))
        #X2[i:(n-1),1:p]=X[(i+1):n,1:p]
        #rbind append colums (here it returns a 2Xn matrix)
        #Y2=as.matrix(rbind(Y[1:(i-1)], Y[(i+1):n]))
        Y2=as.matrix(Y[1:(i-1)])
        Y2[i:(n-1)]=as.matrix(Y[(i+1):n])
      }
    else
      { # omit last row
        X2=as.matrix(X[1:(n-1),])
        Y2=as.matrix(Y[1:(n-1)])
      }
    # calculate regression parameters without ith observation
    # calculate y estimates with regression estimates from ith-less model
    yestPartial=X%*%solve(t(X2)%*%X2)%*%t(X2)%*%Y2
    # calculate D
    CooksD[i,1]=i
    CooksD[i,2]=t(yestFull-yestPartial)%*%(yestFull-yestPartial)/MSE/p
    i=i+1
  }

# graph D by observation number, type "h" draws vertical from x axis
plot(CooksD$observation, CooksD$D, type="h",
main="Cook's D for Weight vs. Body Data", xlab="Observation", ylab="D")
```

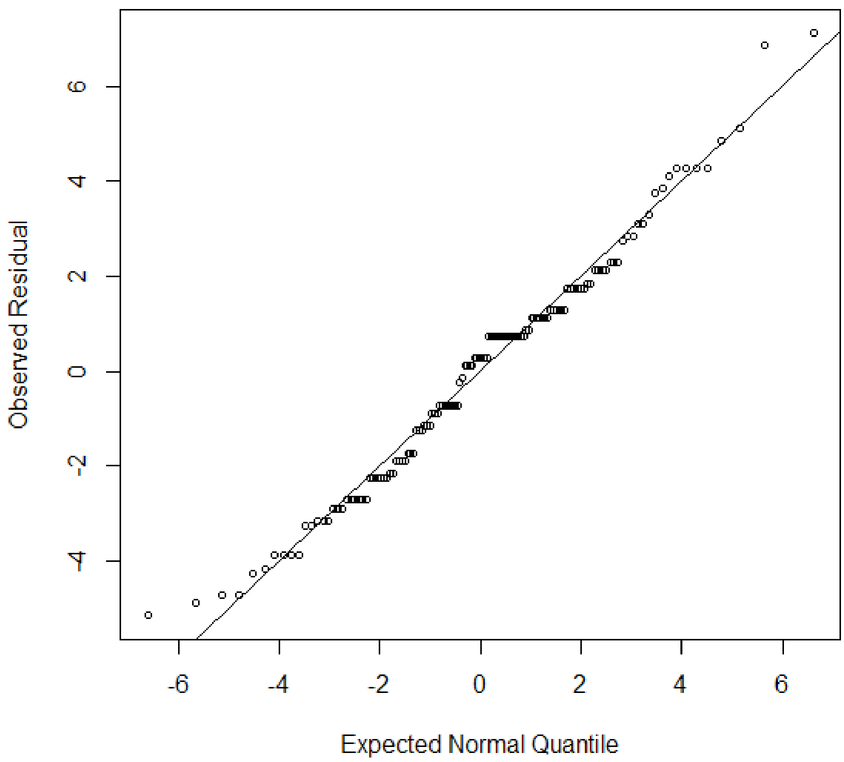


Normality Plot

```
# normality plot

# get data
library(RODBC)
db<-odbcConnect('st512',uid='sa',pw='')
d<-sqlQuery(db,'select * from SingerData')
odbcClose(db)

# generate model
mod=lm(Height~Voice, d)

# results (note that a normality plot is included here free of charge)
plot(mod)
summary(mod)

# linear model contents
attributes(mod)

# residuals (observation vs. fitted values)
mod$residuals
plot(mod$residuals~mod$fitted.values, main='Residual Plot', xlab='Predicted Value',
ylab='Residual')

# rm(resid)

# copy residuals
```

```
resid=as.data.frame(sort(mod$residuals))
names(resid)[1]='resid'

# calculate distribution statistics
n=nrow(resid)
m=sum(resid$resid)/n  # note, this should be 0 since our data are residuals
s=sqrt((sum(resid$resid*resid$resid)-n*m*m)/(n-1))

# calculate normal cmf quantile for observed cumulative mass by rank
# subtract .5 from rank for centering (first quantile is .5/n, last is (n-.5)/n)
i=1
while(i<=n)
  {
    resid$npdquantile[i]=m+s*qnorm((i-.5)/n)
    i=i+1
  }

# plot observed vs. npd residuals
plot(resid$resid~resid$npdquantile, main='Normality Plot', xlab='Expected Normal Quantile',
     ylab='Observed Residual', cex=.75)
# add line (mean 0, slope 1 since normally distributed points would fall on such a line)
abline(0,1)

# verify calculated parameter estimates and estimated response values

# setup design matrix, column of 1s for intercept plus indicator variables for voices
# note the use of 'alto' as reference group (same as lm, it sorts classes then uses top one)
X=as.matrix(cbind(rep(1,nrow(d)),rep(NA,nrow(d)),rep(NA,nrow(d)),rep(NA,nrow(d))))
i=1
while(i<=nrow(d))
{
  X[i,2]=switch(tolower(d$Voice[i]), 'soprano'=1, 0)
  X[i,3]=switch(tolower(d$Voice[i]), 'tenor'=1, 0)
  X[i,4]=switch(tolower(d$Voice[i]), 'bass'=1, 0)
  i=i+1
}
Y=as.matrix(d$Height)

# calculate model parameters
b=solve(t(X)%*%X)%*%t(X)%*%Y
b  # these are identical to lm estimates, no surprise

# calculate residuals
resid=as.data.frame(sort(Y-X%*%b))
names(resid)[1]='resid'
```

Normality Plot



NPD - IBP

```
# NPD CDF Estimation Using Sum of Diminishing Terms from Integration by Parts Solution

# Although there is no known closed form solution to the CDF, integration by parts (IBP)
# gives an infinite series with relatively simple terms

# Note that a CDF is evaluated on the interval(-infinity,z), which requires evaluation of the
# series at z and at -infinity (and the difference taken)
# The IBP series results in negative sums for all negative CDF endpoints (which violates the very
# definition of a CDF), so the series is valid only for positive endpoints and is the cumulative
# density from 0 to the endpoint
# Since the standard NPD function is symmetric about 0, series evaluation at the absolute value
# of supplied endpoints produces CDF areas as a diffential from the centerpoint (where cumulative
# density=.5)
# Therefore, for negative supplied endpoints, subtract the series result from .5, for positive
# endpoints, add to .5

# The series terms do contain increasing factorial like denominators, but each
# term also has an increasing exponent of z (the CDF endpoint of interest)
# So, term-wise calculation of z^(term exponent) divided by term-factorial-multiplier
# remains tame and when multiplied by the previous term yields the proper result
# for the current term, and diminishes due to the implicit factorial-like denominator

NCDF <- function(z) {
  # first term and CDF sum
  s <- z
  F <- z
  i <- 1
  while(abs(s)>.0000001) {
    # calculate next term in series and include in CDF sum
    s <- s*z*z/(2*i+1)
    F <- F+s
    # print(F)
    i <- i+1
  }
  # multiply constant factor and add to centerpoint cumulative density (.5)
  F <- exp(-z*z/2)*F/sqrt(8*atan(1))+.5
  F
}

for(z in(-10:10)) {
 print(NCDF(z/4))
}
```

Geometric pdf

```
first=1
plist=seq(.05,.95,.05)
xmax=0
for(p0 in plist)
  {
    x=matrix(NA, ncol=1)
    y=matrix(NA, ncol=1)
    k=1
    p=p0
    while(p>.025)
      {
        x[k]=k
        p=p0*(1-p0)**(k-1)
        y[k]=p
        k=k+1
      }
    if(first==1)
      {
        # omit axes for now, wait until max values known
        plot(y~x,type="l",main="Geometric pdf",sub=paste("p = ",paste(plist,collapse=", ")),
        axes=FALSE,ylim=c(0,max(plist)),xlab="k (trials to success)",ylab="pdf")
        first=0
```

```
      }
    else
      lines(y~x)
    # save max x and y values for axes later
    if(max(x)>xmax)
      xmax=max(x)
  }
# add axes
axis(1, at=seq(0,xmax,1), labels=TRUE, tick=TRUE)
axis(2, at=seq(0,max(plist)+.1,.1), labels=TRUE, tick=TRUE)
```