Extra Topics 2

1 Velocity Example: Non-Linear Regression

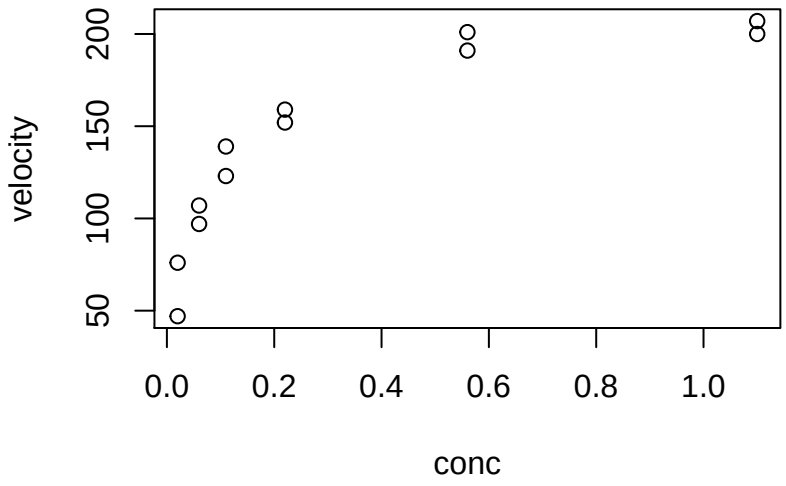
In this example, we model velocity (counts/min) of an enzymatic reaction as a function of substrate concentration (ppm). The Michaelis Menton equation is a well-known model for enzyme kinetics.

We try several different approaches:
1. Quadratic (Model1) and Cubic (Model2) regression
2. Nonlinear regression (Models 3 and 4)
3. Transformation approach (Model5)

```
velocitydat <- read.table("C:/hess/STAT511_FA11/RData/Velocity.txt", header=TRUE)
str(velocitydat)
```

```
## 'data.frame':    12 obs. of  2 variables:
##  $ conc    : num  0.02 0.02 0.06 0.06 0.11 0.11 0.22 0.22 0.56 0.56 ...
##  $ velocity: int  76 47 97 107 123 139 159 152 191 201 ...
```

```
# create a sequence of conc values for plotting:
xconc <- seq(0, 1.1, by=0.005)
plot(velocity ~ conc, data = velocitydat)
```

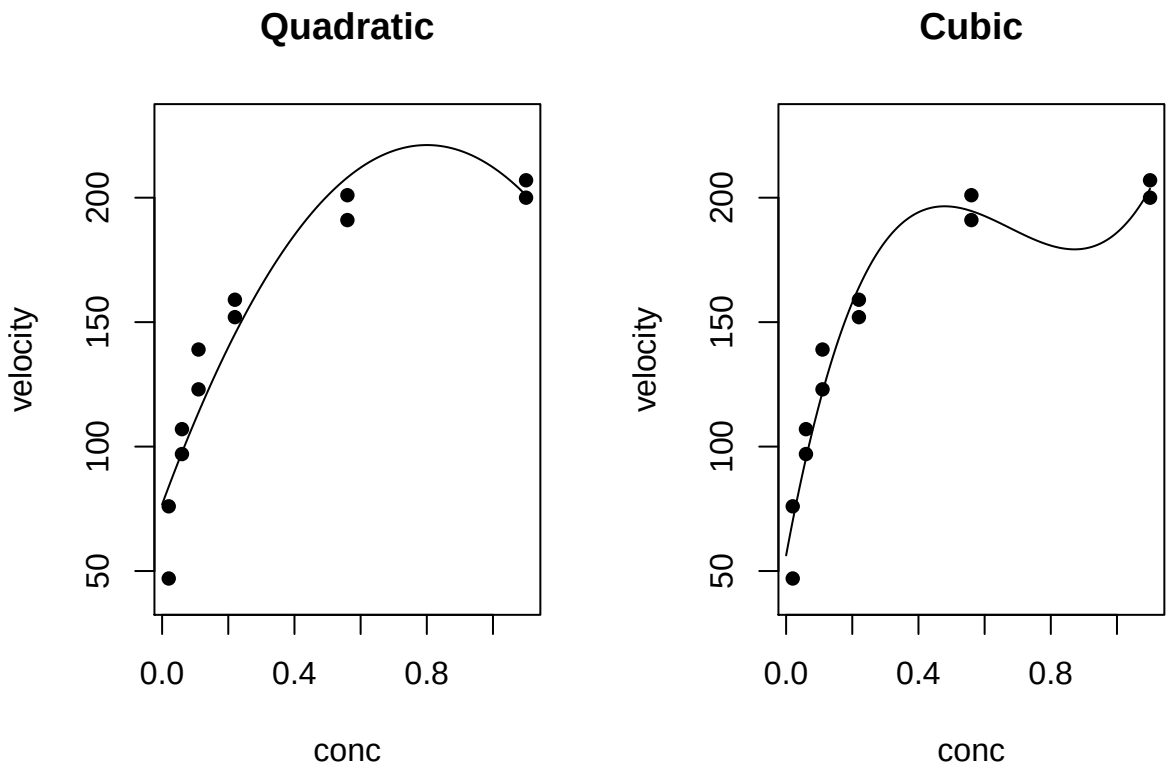


1.1 Polynomial Regression Models

While polynomial regression is reasonable to consider, neither of these models seems appropriate.

```
# Quadratic Fit
model1 <- lm(velocity ~ conc + I(conc^2), data=velocitydat)
p1 <- predict(model1, list(conc=xconc))
# Cubic Fit
```

```r
model2 <- lm(velocity ~ conc + I(conc^2)+I(conc^3), data=velocitydat)
p2 <- predict(model2, list(conc=xconc))
# Plots
par(mfrow =c(1,2))
plot(velocity ~ conc, data=velocitydat, pch=16, ylim=c(40,230), main="Quadratic")
lines(p1 ~ xconc)
plot(velocity ~ conc, data=velocitydat, pch=16, ylim=c(40,230), main="Cubic")
lines(p2 ~ xconc)
```



1.2 Nonlinear Regression Approach1 (Hard Way)

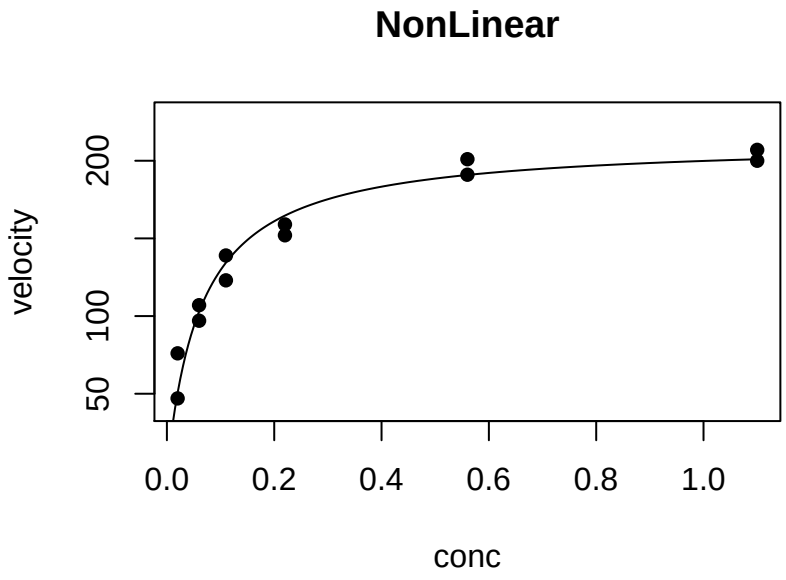
We use the nls() function to fit the Michaelis-Menten model using non-linear regression. Note that for this approach we need to provide starting values.

```r
model3 <- nls(velocity ~ b0*conc/(b1+conc), start=list(b0 = 200, b1 = 0.06), data = velocitydat)
summary(model3)
```

```
##
## Formula: velocity ~ b0 * conc/(b1 + conc)
##
## Parameters:
##     Estimate Std. Error t value Pr(>|t|)
## b0 2.127e+02  6.947e+00  30.615 3.24e-11 ***
## b1 6.412e-02  8.281e-03   7.743 1.57e-05 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.93 on 10 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 9.74e-07
```

```
#Plot
plot(velocity ~ conc, data=velocitydat, pch=16, ylim=c(40,230), main="NonLinear")
p3 <- predict(model3, list(conc=xconc))
lines(p3 ~ xconc)
```




```
#Calculating R2
SSResid <- sum(residuals(model3)^2)
SSTotal <- var(velocitydat$velocity)*(12-1)
R2<- 1-(SSResid/SSTotal)
R2
```

```
## [1] 0.9612608
```

```
# Confidence intervals
confint(model3)
```

```
## Waiting for profiling to be done...
```

```
##            2.5%        97.5%
## b0 197.30212750 229.29006406
## b1   0.04692517   0.08615995
```

1.3 Nonlinear Regression Approach2 (Easy Way)

This time we use the "self-starting" approach which does NOT require starting values. We get identical results to above. Very handy, but not available for all models.

```
model4 <- nls(velocity ~ SSmicmen(conc,a,b), data=velocitydat)
summary(model4)
```

```
##
## Formula: velocity ~ SSmicmen(conc, a, b)
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## a 2.127e+02  6.947e+00  30.615 3.24e-11 ***
## b 6.412e-02  8.281e-03   7.743 1.57e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.93 on 10 degrees of freedom
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 1.937e-06
```

1.4 Transformation Approach

By carefully choosing transformations, the model can be made linear by transformation. We then "back-transform" to the original parameters. The estimated parameters are different from the nonlinear approach.

```
y <- 1/velocitydat$velocity
x <- 1/velocitydat$conc
model5 <- lm(y ~ x)
model5
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##   0.0051072    0.0002472
```

```
B0hat <- 1/coef(model5)[1]
B0hat
```

```
## (Intercept)
##    195.8027
```

```
B1hat <- coef(model5)[2]/coef(model5)[1]
B1hat
```

```
##         x
## 0.04840653
```

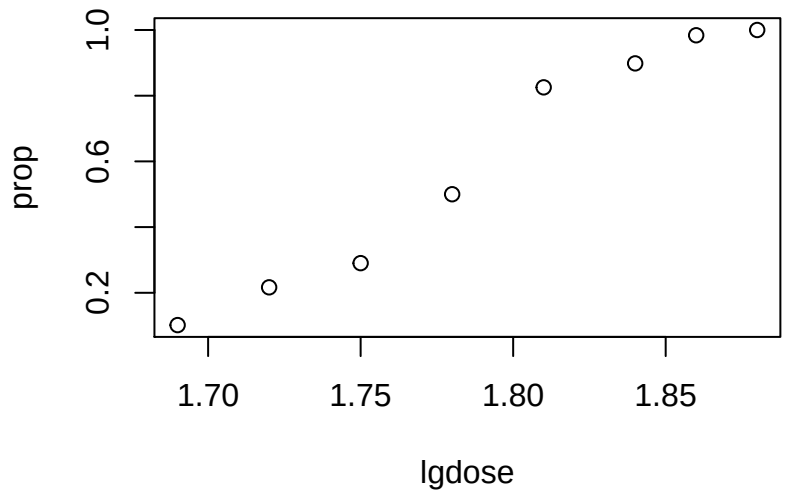
2 Beetle Kill: Logistic Regression

(Simple) logistic regression is used to model the relationship between a binary response variable and a single numerical predictor. In this example, beetle death is considered an event and logdose is the predictor. Note that this data is grouped, meaning that for each of 8 doses we have approximately 60 beetles (either dead or alive) and the data is summarized in only 8 rows (corresponding to 8 doses). The glm function can be used to run logistic regression whether or not the data is grouped, just be careful about the formatting!

```
Beetle <- read.table("c:/hess/STAT511_FA11/RData/Beetles.txt", header = T)
Beetle
```

```
##   lgdose nrtest nrdead
## 1   1.69     59      6
## 2   1.72     60     13
## 3   1.75     62     18
## 4   1.78     56     28
## 5   1.81     63     52
## 6   1.84     59     53
## 7   1.86     62     61
## 8   1.88     60     60
```

```
Beetle$prop <- Beetle$nrdead/Beetle$nrtest
plot(prop ~ lgdose, data = Beetle)
```



2.1 Logistic Regression

With different N at each dose, we supply a 2-column matrix of successes and failures as the "response variable". Note that in order to get the odds ratio, we need to exponentiate the "slope" parameter.

```
model1 <- glm(cbind(nrdead, nrtest-nrdead) ~ lgdose, family = binomial(link = "logit"), data = Beetle)
summary(model1)
```

```
##
```

```
## Call:
## glm(formula = cbind(nrdead, nrtest - nrdead) ~ lgdose, family = binomial(link = "logit"),
##     data = Beetle)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4242  -0.6084   0.7535   1.1080   1.6837
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -59.282      4.995  -11.87   <2e-16 ***
## lgdose        33.519      2.814   11.91   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 284.2024  on 7  degrees of freedom
## Residual deviance:   9.9971  on 6  degrees of freedom
## AIC: 40.195
##
## Number of Fisher Scoring iterations: 4
```

```
#To compute odds ratio estimates and CIs: exponentiate estimates and CI endpoints.
exp(model1$coef)
```

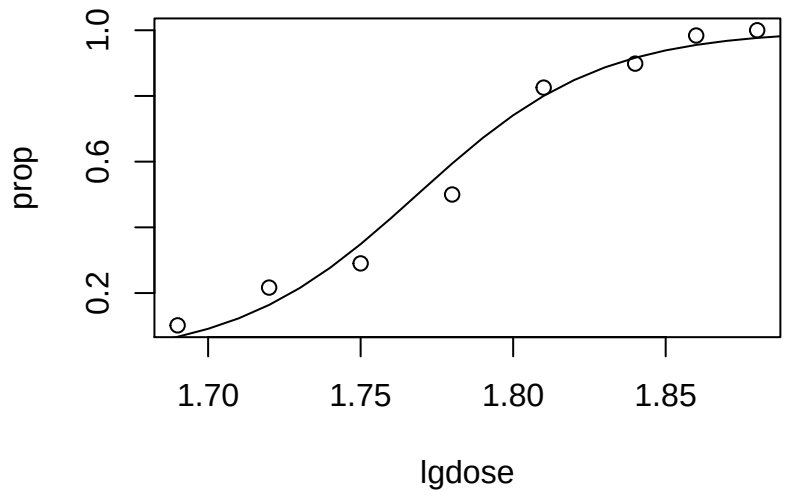
```
##  (Intercept)       lgdose
## 1.794846e-26 3.607265e+14
```

```
exp(confint(model1))
```

```
## Waiting for profiling to be done...
```

```
##                    2.5 %       97.5 %
## (Intercept) 5.847726e-31 1.971502e-22
## lgdose      1.912621e+12 1.219095e+17
```

```
#Plot the fitted curve
plot(prop ~ lgdose, data = Beetle)
lgdosenew <- seq(1.66, 1.9, 0.01)
phat <- predict(model1, list(lgdose = lgdosenew), type = "response")
lines(phat ~ lgdosenew)
```



2.2 Additional Details

dose.p from MASS package computes LD's for various probs. cf=1:2 tells it that coef[1] is the intercept and coef[2] is the slope.

```
library(car)
Anova(model1, type = 3)
```

```
## Analysis of Deviance Table (Type III tests)
##
## Response: cbind(nrdead, nrtest - nrdead)
##        LR Chisq Df Pr(>Chisq)
## lgdose    274.2  1  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
library(MASS)
probs <- seq(0.1, 0.9, 0.1)
ld <- dose.p(model1, cf = 1:2, p = probs)
ld
```

```
##              Dose          SE
## p = 0.1: 1.703059 0.007155075
## p = 0.2: 1.727252 0.005566667
## p = 0.3: 1.743332 0.004703378
## p = 0.4: 1.756514 0.004192013
## p = 0.5: 1.768610 0.003943864
## p = 0.6: 1.780707 0.003949443
## p = 0.7: 1.793888 0.004242038
## p = 0.8: 1.809968 0.004922971
## p = 0.9: 1.834161 0.006363428
```

3 Two-sample Permutation Test

Permutation testing is an alternative to parametric hypothesis testing (for example the two-sample t-test). This example is taken from Introduction to Modern Nonparametric Statistics (2004) by James Higgins. The goal is to compare the mean response for a New vs Traditional training methods.

We try several different approaches: (1) two-sample t-test, (2) two-sample Wilcoxon test, (3) permutation test done "by hand" (for illustration) or using the coin package.

```
library(coin)
PermData <- read.csv("C:/hess/STAT511_FA11/RData/PermTestData.csv")
PermData
```

```
##   Train  Y
## 1   New 37
## 2   New 49
## 3   New 55
## 4   New 57
## 5  Trad 23
## 6  Trad 31
## 7  Trad 46
```

```
#Two-Sample t-test
t.test( Y ~ Train, data=PermData, alternative = "greater", var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Y by Train
## t = 2.0843, df = 5, p-value = 0.04578
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.5372273       Inf
## sample estimates:
##  mean in group New mean in group Trad
##           49.50000           33.33333
```

```
#Two-Sample Wilcoxon test
wilcox_test( Y ~ Train, data=PermData, distribution="exact", alternative = "greater")
```

```
##
##  Exact Wilcoxon-Mann-Whitney Test
##
## data:  Y by Train (New, Trad)
## Z = 1.7678, p-value = 0.05714
## alternative hypothesis: true mu is greater than 0
```

3.1 Permutation Test "by hand" (Hard Way, For Illustration)

```
DiffObs <- mean(PermData$Y[PermData$Train=="New"]) - mean(PermData$Y[PermData$Train=="Trad"])
DiffObs
```

```
## [1] 16.16667
```

```
AllPerms <- combn(7, 4)
AllPerms  <- t(AllPerms)
```

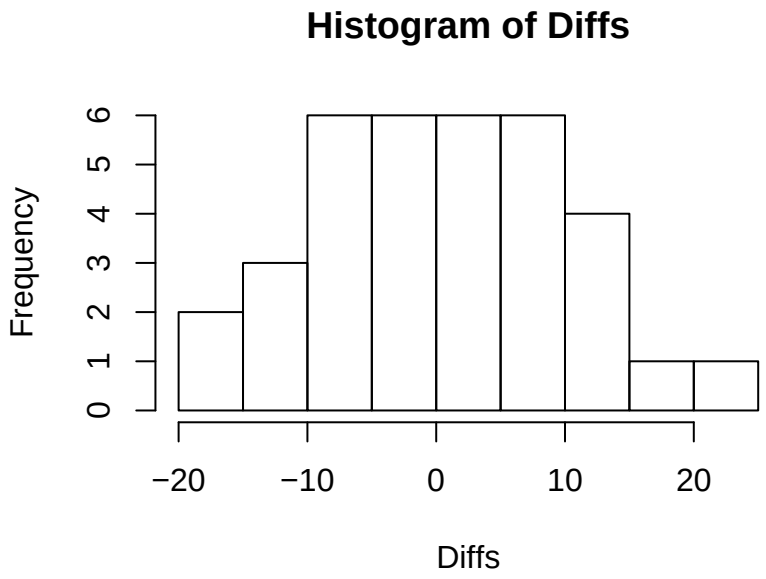
```
dim(AllPerms)
```

```
## [1] 35  4
```

```
head(AllPerms)
```

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

```
Diffs <- c()
for ( i in 1:nrow(AllPerms) )
    {ybar1 <- mean( PermData$Y[AllPerms[i,]] )
     ybar2 <- mean( PermData$Y[-AllPerms[i,]] )
     Diffs <- c(Diffs, ybar1-ybar2)
}
hist(Diffs)
```




```
sort(Diffs)
```

```
##  [1] -19.4166667 -17.6666667 -14.1666667 -13.0000000 -12.4166667
##  [6]  -8.9166667  -8.9166667  -7.7500000  -7.1666667  -6.0000000
## [11]  -5.4166667  -4.2500000  -4.2500000  -3.6666667  -2.5000000
## [16]  -2.5000000  -0.7500000   0.4166667   1.0000000   1.0000000
## [21]   1.5833333   2.1666667   2.7500000   5.6666667   6.2500000
## [26]   6.2500000   7.4166667   8.0000000   9.7500000  10.9166667
## [31]  10.9166667  12.6666667  14.4166667  16.1666667  21.4166667
```

```
Pval <- sum(Diffs >= DiffObs)/length(Diffs)
Pval
```

```
## [1] 0.05714286
```

3.2 Permutation Test using coin package

Note that the p-value for the Wilcoxon test exactly matches the p-value for the permutation test.

```
oneway_test( Y ~ Train, data=PermData, distribution="exact", alternative = "greater")
```

```
##
##  Exact Two-Sample Fisher-Pitman Permutation Test
##
## data:  Y by Train (New, Trad)
## Z = 1.6702, p-value = 0.05714
## alternative hypothesis: true mu is greater than 0
```

4 Survival Analysis

This example uses the Veteran's Administration Lung Cancer data set. Randomized trial of two treatment regimens for lung cancer. Other variables are included in the data set, but we will focus just on trt (1 or 2). Note that status gives the censoring status with 0 representing censored observations. We create a Kaplan-Meier survival curve and run the log-rank test.

```
library(survival)
data(veteran)
str(veteran)
```

```
## 'data.frame':    137 obs. of  8 variables:
##  $ trt     : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ celltype: Factor w/ 4 levels "squamous","smallcell",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ time    : num  72 411 228 126 118 10 82 110 314 100 ...
##  $ status  : num  1 1 1 1 1 1 1 1 1 0 ...
##  $ karno   : num  60 70 60 60 70 20 40 80 50 70 ...
##  $ diagtime: num  7 5 3 9 11 5 10 29 18 6 ...
##  $ age     : num  69 64 38 63 65 49 69 68 43 70 ...
##  $ prior   : num  0 10 0 10 10 0 10 0 0 0 ...
```

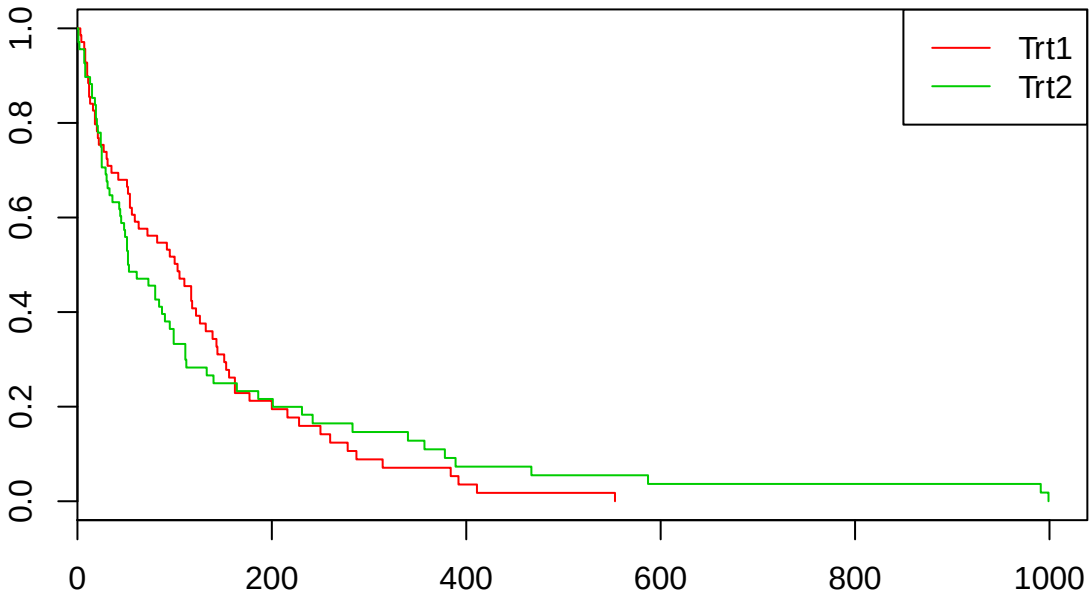
```
survdiff(Surv(time,status) ~ trt, data=veteran)
```

```
## Call:
## survdiff(formula = Surv(time, status) ~ trt, data = veteran)
##
##        N Observed Expected (O-E)^2/E (O-E)^2/V
## trt=1 69       64     64.5   0.00388   0.00823
## trt=2 68       64     63.5   0.00394   0.00823
##
##  Chisq= 0  on 1 degrees of freedom, p= 0.9
```

```
KMfit<-survfit(Surv(time,status) ~ trt, data=veteran)
KMfit
```

```
## Call: survfit(formula = Surv(time, status) ~ trt, data = veteran)
##
##        n events median 0.95LCL 0.95UCL
## trt=1 69     64  103.0      59     132
## trt=2 68     64   52.5      44      95
```

```
plot(KMfit, col=c(2,3))
legend("topright", lty=c(1,1), col=c(2,3), legend=c("Trt1","Trt2"))
```


Trt1
Trt2
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000

5 Multiple Regression: Rice Example

Multiple regression extends the simple linear regression model to include multiple predictor variables. In this example, we consider the yield versus height and tillers for 8 varieties of rice. We use AICc to compare the 3 models.

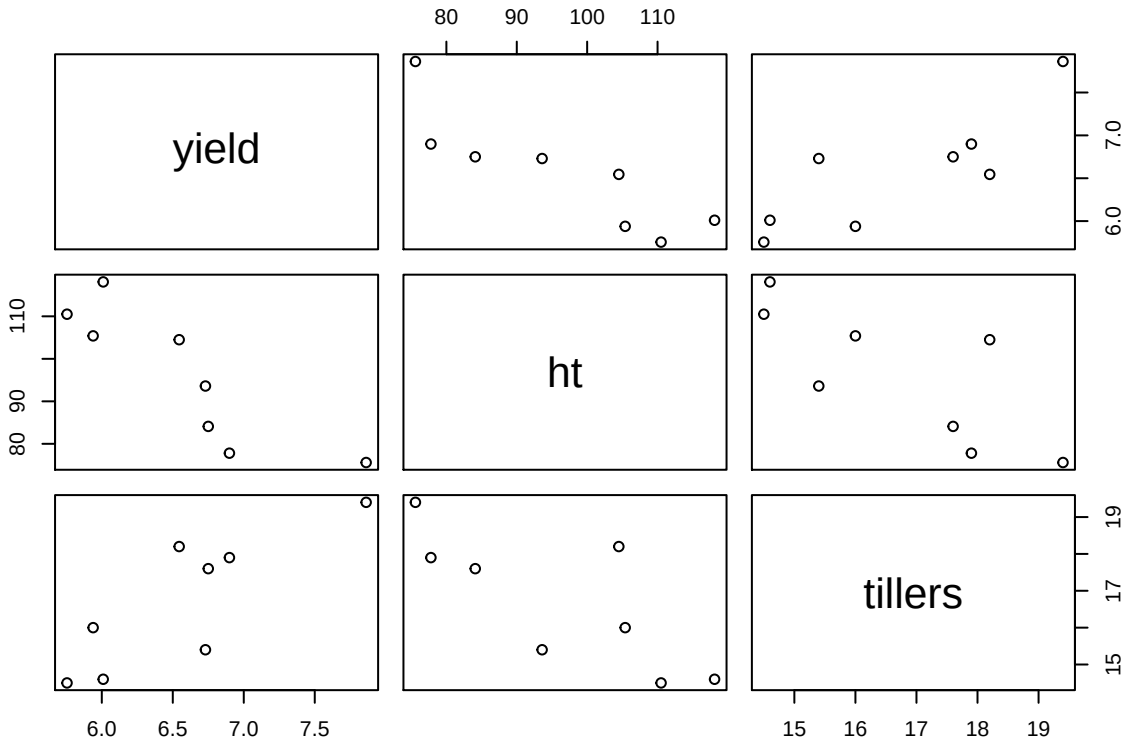
```
ricedata <- read.table("c:/hess/STAT511_FA11/RData/riceheight.txt", header=T, row.names = 1)
ricedata
```

```
##   yield    ht tillers
## 1 5.755 110.5    14.5
## 2 5.939 105.4    16.0
## 3 6.010 118.1    14.6
## 4 6.545 104.5    18.2
## 5 6.730  93.6    15.4
## 6 6.750  84.1    17.6
## 7 6.899  77.8    17.9
## 8 7.862  75.6    19.4
```

```
cor(ricedata)
```

```
##              yield         ht    tillers
## yield    1.0000000 -0.8687070  0.8349761
## ht      -0.8687070  1.0000000 -0.7762814
## tillers  0.8349761 -0.7762814  1.0000000
```

```
pairs(ricedata)
```



5.1 Simple Linear Regressions

```
model1 <- lm(yield ~ ht, data = ricedata)
summary(model1)
```

```
##
## Call:
## lm(formula = yield ~ ht, data = ricedata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.34626 -0.27605 -0.09448  0.27023  0.53495
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.137455   0.842265  12.036    2e-05 ***
## ht          -0.037175   0.008653  -4.296  0.00512 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3624 on 6 degrees of freedom
## Multiple R-squared:  0.7547, Adjusted R-squared:  0.7138
## F-statistic: 18.46 on 1 and 6 DF,  p-value: 0.005116
```

```
model2 <- lm(yield ~ tillers, data = ricedata)
summary(model2)
```

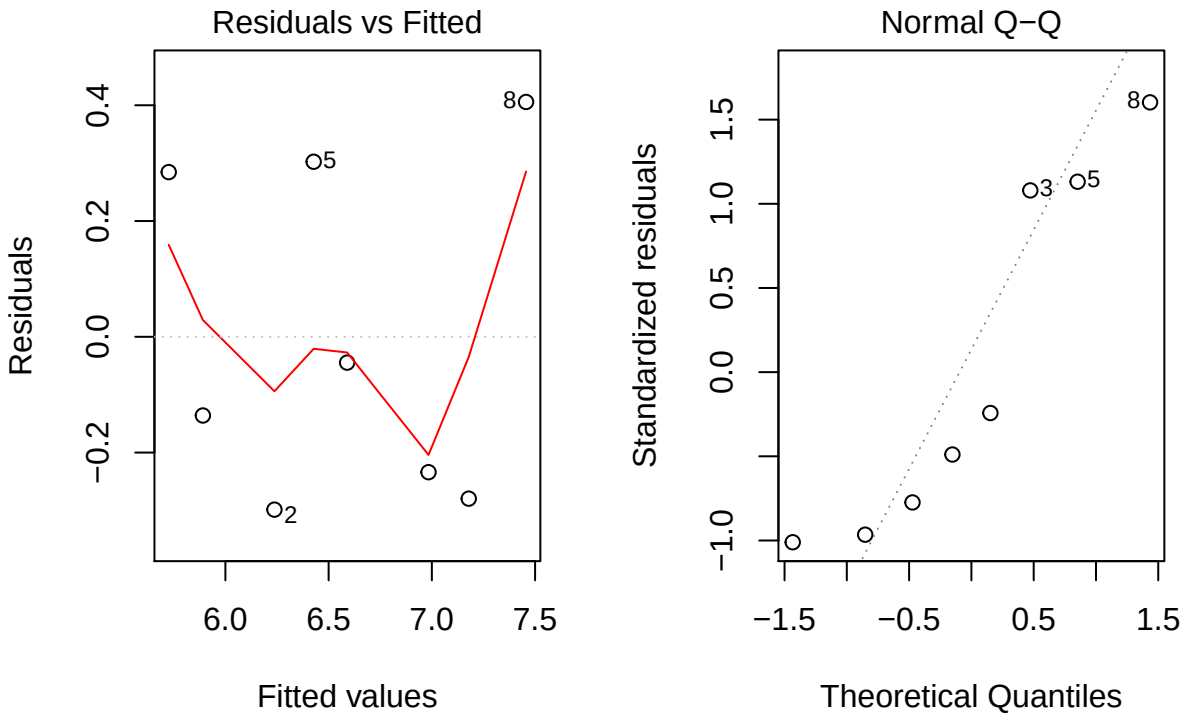
```
##
## Call:
## lm(formula = yield ~ tillers, data = ricedata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4820 -0.1935 -0.0628  0.1912  0.5724
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.37548    1.40249   0.981  0.36460
## tillers      0.31053    0.08355   3.717  0.00989 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4026 on 6 degrees of freedom
## Multiple R-squared:  0.6972, Adjusted R-squared:  0.6467
## F-statistic: 13.81 on 1 and 6 DF,  p-value: 0.009891
```

5.2 Multiple Regression

```
model3 <- lm(yield ~ ht + tillers, data = ricedata)
summary(model3)
```

```
##
## Call:
## lm(formula = yield ~ ht + tillers, data = ricedata)
##
## Residuals:
##        1        2        3        4        5        6        7        8
## -0.13596 -0.29855  0.28449 -0.04461  0.30241 -0.23388 -0.27959  0.40569
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.33560    2.94293   2.153   0.0839 .
## ht          -0.02375    0.01290  -1.842   0.1249
## tillers      0.15031    0.11207   1.341   0.2375
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3404 on 5 degrees of freedom
## Multiple R-squared:  0.8196, Adjusted R-squared:  0.7474
## F-statistic: 11.36 on 2 and 5 DF,  p-value: 0.01383
```

```
par(mfrow=c(1,2))
plot(model3, which =c(1:2))
```



5.3 AICC comparison

Note that model3 is passed to dredge() because it is the largest model considered.

```
library(MuMIn)
options(na.action = "na.fail")
AllSubsets <- dredge(model3, extra=c("R^2"))
AllSubsets
```

```
## Global model call: lm(formula = yield ~ ht + tillers, data = ricedata)
## ---
## Model selection table
##   (Intrc)       ht  tllrs    R^2 df logLik AICc delta weight
## 2  10.140 -0.03717        0.7547  3 -2.080 16.2  0.00  0.657
## 3   1.375          0.3105 0.6972  3 -2.922 17.8  1.68  0.283
## 1   6.561                 0.0000  2 -7.701 21.8  5.64  0.039
## 4   6.336 -0.02375 0.1503 0.8196  4 -0.851 23.0  6.87  0.021
## Models ranked by AICc(x)
```

6 Iris Example: PCA

Principle components analysis (PCA) is a dimension reduction technique, most commonly used for graphing. This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

```
data(iris)
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
cor(iris[,1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

6.1 PCA

Here we use the prcomp() function to do the PCA analysis. The function princomp() also works, but has different format.

summary gives the proportion of variation explained by each component. In this case, the first 2 components explain 96% of variation.

Rotation gives the coefficients (or loadings) of the components.

x gives the principal components (or scores).

```
pcout<-prcomp(iris[,1:4], center=TRUE, scale= TRUE)
summary(pcout)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.7084 0.9560 0.38309 0.14393
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518
## Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

```
pcout$rotation
```

```
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

```
dim(pcout$x)
```

```
## [1] 150   4
```

```
pcout$x[1:5,]
```

```
##            PC1        PC2         PC3         PC4
## [1,] -2.257141 -0.4784238  0.12727962  0.02408751
## [2,] -2.074013  0.6718827  0.23382552  0.10266284
## [3,] -2.356335  0.3407664 -0.04405390  0.02828231
## [4,] -2.291707  0.5953999 -0.09098530 -0.06573534
## [5,] -2.381863 -0.6446757 -0.01568565 -0.03580287
```

```
round(cor(pcout$x), 3)
```

```
##     PC1 PC2 PC3 PC4
## PC1   1   0   0   0
## PC2   0   1   0   0
## PC3   0   0   1   0
## PC4   0   0   0   1
```

6.2 PCA Plot

```
Scores <- data.frame(pcout$x, Species=iris$Species)
plot(Scores[,1:2], pch=c("s ", "c ", "v ")[Scores$Species])
title("First Two Principal Components of Iris Data")
```