

An introduction to **R**

Short course agenda

Day 1 (9:00 – 12:00)

Introductions and course overview

- What is **R** and where can I find it?

The **R** working environment

- Command window
- Workspace

Getting started

- Current R session
- Working directory
- Saving workspace

Executing simple commands

R object types

- Vector, matrix, array, data frame, functions, lists

Accessing data from objects

Data queries in **R**

Exercise 1

Day 1 (1:30 – 5:00)

Importing data from Excel (or a text file)

- Tips on ‘cleaning up your data’ prior to importing

Exporting data from **R**

Introduction to **R** functions

Introduction to loops

User-defined functions

Exercise 2 part 1

Sub-setting data

- The subset function

Exercise 2 part 2

Introduction to graphing in **R**

Plotting commands

- High-level commands
- Low-level commands
- Exercise 3

Day 2 (9:00 – 12:00)

Graphing continued

Lower-level commands

Exercise 4

Introduction to statistical analyses using **R**

- Descriptive
- Univariate
- Multivariate

The basic model structure in **R**

- Simple linear regression example

Linear regression

- Model diagnostics
- Subset data
- Running models through loops-an illustration

Day 2 (1:30 – 5:00)

Analysis of variance

- Categorical explanatory variables
- Multiple comparisons

Exercise 5

Nonlinear regression

- Specifying models
- Starting values
- Von Bertalanffy example (least-squares estimates)
- Plotting fitted lines

Exercise 6

#Quantitative Fisheries Center

#R Short course

#R object types 1 (vectors, arrays, data frames)

#Vector

v1<-c(12, 5, 6, 8, 24) #numeric

v2<-c("Yellow perch", "Largemouth bass", "Rainbow trout", "Lake whitefish",
"Northern pike") #character

years<-c(1990:2007) #Obtain a sequence of numbers in a vector

#Print the vectors

v1

v2

years

#Matrix

#Create a 4 row by 5 column matrix using the array function

m1<-array(1:20, dim=c(4,5)) #The dim statement gives the dimensions of the
matrix or array

m1

#Create a 4 row by 5 column matrix using the matrix function

m2<-matrix(1:20, ncol=5, nrow=4)

m2

#Create a matrix using the cbind (column) and rbind (row) commands (vectors
must be of equal length)

m3<-cbind(v1, v2)

m3

m4<-rbind(v1, v2)

m4

#Create a data frame using vectors 1 and 2

df1<-data.frame(v1, v2)

#Create a data frame using matrix 1

df2<-data.frame(m1)

df2

#Alternatively, you can type vectors into a data frame

df3<-data.frame(x1=c(1,2,3), x2=c(4,5,6))

df3

#Edit data frame 1 in a spreadsheet-like view

df2<-edit(df1)

#Quantitative Fisheries Center

#R Short course

#R object types 2 (vectors, arrays, data frames)

```
#Vectors
v1<-c(1:7)
v2<-c(6:12)
v3<-seq(length=7, from=10, by=3)
v4<-seq(length=7, from=-5, by=2)

#create a matrix
mat1<-cbind(v1, v2, v3, v4)
mat1

#select the element in the 2nd row and 4th column
select1<-mat1[2,4]
select1

#select the 2nd through fourth columns
select2<-mat1[,2:4]
select2

#OR
select3<-mat1[,c(2,3,4)]
select3

#select the first row
select4<-mat1[1,]
select4

#select the first 3 rows and the last 2 columns
select5<-mat1[1:3, 3:4]
select5

#Get length of first vector in mat1
length1<-length(mat1[,1])
length1

#Get dimensions of matrix
dim(mat1)

#Create data frame from vectors
df1<-data.frame(v1,v2,v3,v4)
df1

#query out the last 3 columns of the data frame df1
query<-df1[,2:4]
query

#query all rows with a value of v3>20
query1<-df1$v3 > 20
#print query--prints all the columns for the rows that meet the criteria in
query1
df1[query1,]
#print only those values >20 in vector (column) 3
df1[query1,3]
```

```
#Quantitative Fisheries Center  
#R Short course  
#R functions 1
```

```
#####Generate data#####  
  
#Set the dimensions of the matrix  
row.num=20  
col.num=3  
  
#Create a matrix to fill with data  
data<-matrix(0, ncol=col.num, nrow=row.num)  
data  
  
#Use two loops to fill matrix data  
  
for(i in 1:row.num){ #for row 1 to row.num  
  for(j in 1:col.num){ #for column 1 to col.num  
    data[i,1]=rnorm(1, mean=100, sd=20) #Fill column 1 with a random  
number from a normal distribution with a mean of 100 and sd of 20  
    data[i,2]=runif(1, min=0, max=1) #Fill column 2 with a random 0,1  
uniform number  
    data[i,3]=data[i,1]*data[i,2] #Fill row i of column 3 with the  
product of row i in column 1 and 2  
  }  
}  
data  
  
#rename columns of matrix data  
colnames(data) <- c("x","y","z")  
  
#####  
  
#convert data to a data frame  
df<-data.frame(data)  
df  
  
#retrieve the names of data frame df  
var.names<-names(df)  
var.names  
  
#Calculate the mean of variable x in data frame df  
mean.x<-mean(df$x)  
mean.x  
  
#Calculate the variance of the variable x in data frame df  
var.x<-var(df$x)  
var.x  
  
#Calculate the standard deviation of x  
sd.x<-sd(df$x)  
sd.x  
  
#OR  
  
sd.x2<-sqrt(var.x)
```

```
sd.x2
```

```
#Get summary of variable x  
sum1<-summary(df$x)  
sum1
```

```
#Get summary of contents of data frame df  
sum2<-summary(df)  
sum2
```

```
#Quantitative Fisheries Center  
#R Short course  
#R functions 2
```

```
#Read in data table from text file
```

```
lake.data<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\2_R Functions\\Water_qual_data.txt", na.strings="NA",  
header=T)
```

```
#attach data frame  
attach(lake.data)
```

```
#Print column names  
names(lake.data)
```

```
#Print first few rows of the data frame lake.data  
head(lake.data)
```

```
#Obtain summary of data frame  
summary(lake.data)
```

```
#Obtain the mean for total phosphorus (tp.ugL)  
mean.tp1<-mean(tp.ugL)  
mean.tp1
```

```
#Now obtain the mean after removing the missing values (NA)  
mean.tp2<-mean(tp.ugL, na.rm = TRUE)  
mean.tp2
```

```
#Obtain column means of variables in the data frame  
sapply(lake.data, FUN="mean", na.rm=T)
```

```
#sapply and tapply are similar to "looping through" columns  
loop.mean<-numeric(11)  
for (i in 1:11){  
  loop.mean[i]<-mean(lake.data[,i+2], na.rm=T)  
}  
loop.mean
```

```
#Obtain column standard deviations of variables in the data frame  
sapply(lake.data, FUN="sd", na.rm=T)
```

```
#Obtain column variance estimates of variables in the data frame  
sapply(lake.data, FUN="var", na.rm=T)
```

```
#Obtain mean tp.ugl by group  
tapply(tp.ugL, list(county), mean)
```

```
#Use the by() function to obtain means of select columns  
county.means<-by(lake.data[,3:6], county, summary, na.rm=TRUE)  
county.means
```

```
#Obtain the levels of character variables  
levels<-levels(county)  
levels
```

```
#Export data
write.table(loop.mean, file="C:\\Documents and
Settings\\wagnerty.DOBIESZN\\My Documents\\R-SHORT\\Programs\\2_R
Functions\\export1.txt", sep=",", col.names=NA)
```



```
#Quantitative Fisheries Center  
#R Short course  
#R functions 3
```

```
#Read in data table from text file
```

```
lake.data<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\2_R Functions\\Water_qual_data.txt", na.strings="NA",  
header=T)
```

```
#attach data frame  
attach(lake.data)
```

```
#Create a simple function that squares a number
```

```
sqr<-function(x){  
    x*x  
}
```

```
sqr(tp.ugL)
```

```
#Quantitative Fisheries Center  
#R Short course  
#R subset a data frame
```

```
#Import data frame  
fish<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\2_R Functions\\Exercise 2_Fish_stream_example_TW.txt",  
na.strings="NA", header=T)
```

```
attach(fish)
```

```
#Subset rows using the subset function
```

```
sub1<-subset(fish, no.fish > 50)  
sub1
```

```
#Subset rows using multiple conditions
```

```
sub2<-subset(fish, no.fish>50 & position=="Below")  
sub2
```

```
#Select specific columns in a data frame  
sub3<-subset(fish, select=c(stream, site, no.fish))  
sub3
```

```
#Quantitative Fisheries Center  
#R Short course  
#Graphing Basics 1
```

```
#####Generate data#####  
#generate random numbers to plot  
x<-rnorm(100, mean=20, sd=3)  
y<-(x^2 + runif(100))+ rnorm(100, mean=20, sd=15)  
#Create a chracter column  
group<-ifelse(x>20, "A", "B")  
#####  
  
#Create a data frame  
df<-data.frame(x,y,group)  
  
#Plot x  
plot(x)  
  
#scatter plot of x versus y  
plot(df$x,df$y)  
  
#Use the plot function to form a boxplot  
plot(df$group,df$x)  
  
#histogram of x  
hist(x)  
  
#barplot of x  
barplot(x)  
  
#boxplot of x  
boxplot(x)  
  
#boxplot of x and y  
boxplot(x,y)  
boxplot(df$x~df$group)  
  
#Combine the columns of x and y and use scatter plot matrix  
z<-cbind(x, y)  
pairs(z)  
  
pairs(df[,1:2])
```

```
#Quantitative Fisheries Center  
#R Short course  
#Graphing Basics 2
```

```
#generate random numbers to plot  
x<-rnorm(100, mean=20, sd=3)  
y<-(x^2 + runif(100))+ rnorm(100, mean=20, sd=15)  
  
#Sort x and y  
x1<-sort(x)  
y1<-sort(y)  
plot(x1,y1, type="l", main="Line type l") #type = "l" plots a line graph  
plot(x1,y1, type="o", main="Line type 0") #type = "o" overlays line and  
points  
plot(x1,y1, type="h", main="Line type h") #type = "h" plots a histogram-like  
graph  
  
#Add axis labels to plot  
plot(x1,y1, xlab="x-label here", ylab="y-label here")  
  
#Add main title to graph  
plot(x1,y1, xlab="x-label here", ylab="y-label here", main="Practice Graph")  
  
#Change plotting character in graph using pch=  
plot(x1,y1, xlab="x-label here", ylab="y-label here", main="Practice Graph",  
pch=16)  
plot(x1,y1, xlab="x-label here", ylab="y-label here", main="Practice Graph",  
pch="+")  
  
#Change size of character using cex=  
plot(x1,y1, xlab="x-label here", ylab="y-label here", main="Practice Graph",  
pch=16, cex=2)  
  
#Change color of axis labels and main title using col.lab=, and col.main=  
plot(x1,y1, xlab="x-label here", ylab="y-label here", main="Practice Graph",  
pch=16, cex=2, col.lab="blue", col.main="purple")  
  
#Change the x and y axis scales using the xlim= and ylim= statements  
plot(x1,y1, xlab="x-label here", ylab="y-label here", main="Practice Graph",  
pch=16, cex=2, cex.main=3, col.lab="blue", col.main="purple", xlim=c(0,40),  
ylim=c(0,800))
```

```
#Quantitative Fisheries Center  
#R Short course  
#Graphing basics 3
```

```
chinook<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\3_Graphing Basics\\Chinook salmon data graphing  
example.txt", na.strings="NA",header=T)
```

```
#Plot-showing mean, mean+sd, and median of fish length  
#xlab='' makes x-axis label blank  
plot(chinook$length, xlab='', ylab="Chinook length (mm)")
```

```
#Add lines to graph  
abline(h=mean(chinook$length, na.rm=T), lty=1)  
abline(h=mean(chinook$length, na.rm=T)+ sd(chinook$length, na.rm=T), lty=2)  
abline(h=median(chinook$length, na.rm=T), lty=3)
```

```
#Add text to graph, coordinates are (x,y)  
text(9,130,"Mean", cex=0.8)  
text(9, 125, "Median", cex=0.8)  
text(15, 138, "Mean + 1 SD", cex=0.8)
```

```
#Scatter plot identifying points by hatchery of origin. This is accomplished  
using the points() function  
plot(chinook$length,chinook$wgt, xlab="Length (mm)", ylab="Weight (g)")  
points(chinook$length[chinook$hatchery=="DWOR"],  
chinook$wgt[chinook$hatchery=="DWOR"], col="red", pch=19)  
points(chinook$length[chinook$hatchery=="MCCA"],  
chinook$wgt[chinook$hatchery=="MCCA"], col="blue", pch=9)  
points(chinook$length[chinook$hatchery=="RAPH"],  
chinook$wgt[chinook$hatchery=="RAPH"], col="black", pch=16)
```

```
#Quantitative Fisheries Center  
#R Short course  
#Graphing pch and cex illustration
```

```
x<-1:28  
y<-1:28  
  
pch<-c(1:25)  
plot(x,y,pch=1, cex=0, xlab="", ylab="")  
text(10,25, "pch = symbol types", font=4, cex=1.9)  
text(10,23, "col = color types", font=4, cex=1.9)  
for(i in 1:length(pch)){  
  points(i,i, pch=i, cex=1.7, col=i)  
  text(i-.5,i+2,i, cex=1.5, font=4)  
}
```

```
#Quantitative Fisheries Center
#R Short course
#Graphing example
```

```
chinook<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-
SHORT\\Programs\\3_Graphing Basics\\Chinook salmon data graphing
example.txt", na.strings="NA",header=T)
```

```
#Set the size of a multiple figure array using the par and mfrow functions
par(mfrow=c(2,2))
```

```
#HISTOGRAM
```

```
hist(chinook$length, prob=T, main="Length (mm) histogram", xlab="Chinook
slamon lengths", cex.lab=1, col="lightgrey", cex.axis=1)
lines(density(chinook$length, na.rm=T), col="blue")
#rug and jitter show data points as lines on axis
rug(jitter(chinook$length))
#Boxplot of length
boxplot(chinook$length, ylab="Length (mm)", xlab='', col="orange3", lty=1,
main="Boxplot of length")
```

```
#BOXPLOT
```

```
#boxwex is a scale factor for the box width, lty=line type, h=horizontal
boxplot(chinook$trigly~chinook$hatchery, boxwex=0.15, ylab="Triglycerides
(mg/dL)", cex.lab=1, cex.axis=1, col=c("lightyellow", "red", "blue"), lty=1,
main="Chinook triglyceride levels for three hatcheries")
abline(h=mean(chinook$trigly, na.rm=T), lty=3)
```

```
#SCATTER PLOT
```

```
plot(chinook$length, chinook$wgt, xlab="Length (mm)", ylab="Weight (g)",
cex.lab=1, cex.axis=1, main="Scatter plot of length-weight")
points(chinook$length[chinook$hatchery=="DWOR"],
chinook$wgt[chinook$hatchery=="DWOR"], col="red", pch=19)
points(chinook$length[chinook$hatchery=="MCCA"],
chinook$wgt[chinook$hatchery=="MCCA"], col="blue", pch=9)
points(chinook$length[chinook$hatchery=="RAPH"],
chinook$wgt[chinook$hatchery=="RAPH"], col="black", pch=16)
legend(110,50, c("DWOR","MCCA","RAPH"), pch=c(19,9,16),
col=c("red","blue","black"), bty="n")
arrows(165,35,165,50,0.5,angle=45,lty=1,length=0.25, code=2)
arrows(172,35,172,50,0.5,angle=45,lty=1,length=0.25, code=2)
text(168,33, "Big Fish")
```

```
#Quantitative Fisheries Center  
#R Short course  
#correlations and t-tests
```

```
ex1<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\4_corr and t_test\\Chinook salmon data.txt",  
na.strings="NA",header=T)
```

```
#Obtain correlation matrix for chinook salmon data  
chinook.cor<-cor(ex1[,2:5])  
chinook.cor
```

```
#T-test for examining differences in mean length between Dworshak and McCall  
hatchery fish.
```

```
#Select out Dworshak and Rapid River hatcheries using the subset command ( |  
means logical OR)
```

```
dwor.raph.data<-subset(ex1, hatchery=="DWOR" | hatchery=="RAPH")
```

```
dwor.raph.t.test<-t.test(dwor.raph.data$length~dwor.raph.data$hatchery)  
dwor.raph.t.test
```

```
#OR
```

```
#Query out desired hatcheries within the t.test function  
dwor.raph.t.test2<-t.test(ex1$length[ex1$hatchery=="DWOR"],  
ex1$length[ex1$hatchery=="RAPH"])  
dwor.raph.t.test2
```



```
#Quantitative Fisheries Center
#R Short course
#Simple linear regression 1
```

```
#Linear regression example 1
int<-c(1,1,1,1,1)
x<-c(1,2,3,4,5)
y<-c(1,1.6,3.3,3.7,5.4)
#Create design matrix
design<-cbind(int,x)
design

#Estimate coefficients for linear regression of x and y, including intercept
#Betas = (t(x)*x)^-1 * t(x)*y

#Regression with slope and intercept
m1<-(t(design)%*%design)
#take inverse of m1
m2<-solve(m1)
betas<-m2%*%t(design)%*%y
betas

#Regression with slope only (intercept through origin)
m1a<-(t(x)%*%x)
#take inverse of m1a
m2a<-solve(m1a)
beta<-m2a%*%t(x)%*%y
beta

#Linear regression of y~x with an intercept using the lm() function
reg.example1<-lm(y~x)
reg.example1

#Look at design matrix for the above model
model.matrix(y~x)
reg.example1

#OR

reg.example2<-lm(y~1+x)
reg.example2

#Linear model without an intercept
reg.example3<-lm(y~0+x)
reg.example3

#Look at design matrix
model.matrix(y~0+x)
reg.example3

plot(y~x, pch=16, cex=1.2)
abline(lm(y~1+x), lty=1)
abline(lm(y~0+x), lty=2)
text(2,3, "No intercept model (line through origin)", cex=1.2)
arrows(1.5,2.8,1.5,1.6, angle=30,lty=2,length=0.25, code=2)
text(2.5,1.5, "Intercept model", cex=1.2)
```

```
arrows(2.5,1.6,2.5,2.3, angle=30, lty=1, length=0.25, code=2)
```

```
#Quantitative Fisheries Center  
#R Short course  
#Simple linear regression 2
```

```
#Read in data  
reg.data<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\5_Linear regression\\Chinook salmon data.txt",  
na.strings="NA",header=T)
```

```
#Plot length-weight data  
plot(reg.data$length~reg.data$wgt, xlab="Weight (g)", ylab="Length (mm)",  
pch=16)
```

```
modell<-lm(length~wgt, data=reg.data)  
#obtain summary of fit  
summary(modell)  
#Obtain ANOVA table  
aov(modell)  
#Obtain coefficients for modell  
model.coefs<-coef(modell)  
model.coefs  
#Obtain residuals for modell  
model.resids<-residuals(modell)  
model.resids  
#Obtain fitted values for modell  
model.fit.info<-fitted(modell)  
model.fit.info  
#Obtain AIC fpr modell  
AIC(modell)  
#Log likelihood of modell  
logLik(modell)
```

```
#Plot model fit  
plot(length~wgt, data=reg.data, pch=16, xlab="Weight (g)", ylab="Length  
(mm)")  
abline(lm(length~wgt, data=reg.data))
```

```
#Obtain diagnostics for modell  
#Make it so all four graphs can be placed on one page  
par(mfrow=c(2,2))  
plot(modell)
```

```
#Perform a log-log regression length vs weight  
model2<-lm(log(length)~log(wgt), data=reg.data)  
summary(model2)  
plot(log(length)~log(wgt), data=reg.data)  
abline(lm(log(length)~log(wgt), data=reg.data))  
par(mfrow=c(2,2))  
plot(model2)
```

```
#Perform a quadratic regression. The I represents multiplication and the ^  
means to the power (you have to insulate powers of numeric vectors (using  
I()))  
model3<-lm(length~wgt+I(wgt^2), data=reg.data)
```

```
#Look at desing matrix
model.matrix(length~wgt+I(wgt^2), data=reg.data)
summary(model3)
plot(length~wgt, data=reg.data)
abline(lm(length~wgt+I(wgt^2), data=reg.data))

#Plot both model fits on same page
par(mfrow=c(2,1))
plot(length~wgt, data=reg.data)
abline(lm(length~wgt, data=reg.data))
plot(log(length)~log(wgt), data=reg.data)
abline(lm(log(length)~log(wgt), data=reg.data))

#Multiple regression
model4<-lm(trigly ~ length + chol, data=reg.data)
summary(model4)
```

```
#Quantitative Fisheries Center  
#R Short course  
#Simple linear regression 3
```

```
#Read in data  
reg.data<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\5_Linear regression\\Chinook salmon data.txt",  
na.strings="NA",header=T)
```

```
#Perform regression only on Dworshak hatchery fish  
dwor.reg <- lm(length ~ wgt, subset=hatchery=="DWOR", data=reg.data)  
summary(dwor.reg)  
anova.lm(dwor.reg)
```

```
#Perform regression on those fish less than 140 mm  
length.reg <- lm(length ~ wgt, subset=length < 140, data=reg.data)  
summary(length.reg)  
#Plot the model fit  
plot(length~wgt, subset=length < 140, data=reg.data)  
abline(lm(length ~ wgt, subset=length < 140, data=reg.data))
```

```
##### EXAMPLE of running a model through a loop #####  
#Regressions by grouping factors, using a loop  
#Create a new variable called hatch and set to zero  
reg.data$hatch<-0  
#Re-code the three hatcheries into a numeric  
reg.data[1:30,6]<-1  
reg.data[31:60,6]<-2  
reg.data[61:90,6]<-3
```

```
#Or we can use an ifelse statement  
#ifelse syntax is: ifelse(test, yes, no)  
reg.data$hatch2<-ifelse(reg.data$hatchery=="DWOR", 1,  
ifelse(reg.data$hatchery=="MCCA", 2,3))
```

```
#Create an empty list to fill during the loop  
model<-list()  
#Loop through hatcheries. Note, we use double brackets to index a list  
for(i in 1:3){  
model[[i]]<-lm(length~wgt, subset=hatch==i, data=reg.data)  
print("Model")  
print(i)  
print(summary(model[[i]]))  
}
```

```
#Quantitative Fisheries Center  
#R Short course  
#Analysis of variance 1
```

```
#Read in data  
chinook<-read.table("C:\\Documents and Settings\\wagnerty\\My Documents\\R-  
SHORT\\Programs\\6_ANOVA\\Chinook salmon data.txt", na.strings="NA",header=T)  
  
#Compare mean plasma triglycerides between hatcheries  
  
anova.example<-aov(chinook$trigly~chinook$hatchery)  
summary(anova.example)  
summary.lm(anova.example)  
  
boxplot(chinook$trigly~chinook$hatchery, xlab="Hatchery", ylab="Triglycerides  
(mg/dL)", col=c("blue", "red", "yellow"))
```

```
#Quantitative Fisheries Center  
#R short-course  
#Nonlinear regression example
```

```
#These data are from a spring gill net survey of lake trout in the Great  
Lakes conducted by the Michigan Department of Natural Resources
```

```
length.age<-read.table("C:\\Documents and Settings\\wagnerty\\My  
Documents\\R-SHORT\\Programs\\7_NonlinearRegression\\non linear reg data-  
length-weight-age.txt", header=TRUE)
```

```
#Nonlinear regression - Model for length at age  
#Fit a von Bertalanffy Growth Model
```

```
vonB1<-nls(length~Linf*(1-exp(-k*(age-to))), data=length.age,  
start=list(Linf=1000, k=0.05, to=-2))  
summary(vonB1)
```

```
#Create a sequence of ages from 0 to 16 by 1  
age2<-seq(0,16,1)  
#Predict the lengths using estimated parameters  
y1<-1247.537*(1-exp(-0.06164*(age2--4.6745)))  
#Plot original data points  
plot(length~age, data=length.age, xlim=c(0,17), pch=16, xlab="Age (yrs)",  
ylab="Length (mm)",cex.lab=1.5, cex.axis=1.5)  
#Overlay predicted values form models  
lines(age2,y1)
```

What is R?

R is a programming environment for data analysis and graphing. The R language evolved from the S language, which was developed at AT&T's Bell Laboratories by Rick Becker, John Chambers, and Allan Wilks. A commercial product called S-PLUS was then developed from S. Finally, statisticians (Ross Ihaka and Robert Gentleman) from the University of Auckland, New Zealand, decided to write a version of S for teaching purposes and they called this program 'R'. The code for R was released in 1995 under a General Public License. R is an Open Source implementation of S-PLUS that can be freely downloaded.

Web resources

R home page: <http://www.r-project.org/>

R Archive (download R): <http://cran.r-project.org/>

R FAQ (frequently asked questions about R): <http://cran.r-project.org/doc/FAQ/R-FAQ.html>

R manuals: <http://cran.r-project.org/manuals.html>

The R environment

The R command window (Console) or graphical user interface (RGui) is a window for entering commands for data manipulations, statistical analyses, and graphing. This window can also be used as a sophisticated calculator.

Although commands can be readily entered into the R console, scripts are an easy way of organizing your R programs. To create new scripts use the File menu and select New script. Commands can now be entered into the script and the commands are executed by highlighting the command and hitting <Ctrl> <R>, or by going to the Edit menu and selecting Run all. The results from executing your commands are written to the R console. Scripts can be saved after you have completed your session in R.

The R workspace

The R workspace is your current working environment. This environment consists primarily of user-defined objects, such as variables, datasets, and functions. Saving the current workspace is done while in the R Console by using the File menu and selecting Save Workspace...

When you open your previously saved workspace it will contain the objects from the previous session.

To see what objects are in your workspace type *ls()* or *objects()* in the R console. It is often a good idea to clean up your workspace when you are finished with a session. You can remove objects by typing *rm(object1, object2, etc)* or to remove all objects type *rm(list=ls())*. However, be certain you want to delete all objects.

Some tips for getting started in R

1. Create a new folder on your hard drive for your current R session
2. Open R and set the working directory to that folder (File→Change dir...)

3. Save the workspace with a descriptive name and date
4. Open a new script and save the script with a descriptive name and date

Executing simple commands in R

In R objects (e.g., variables, vectors, matrices, etc) are assigned values by using the assignment operator `<-` which is a composite symbol consisting of the less than symbol (`<`) and the minus sign (`-`).

For example:

`x <- 5` assigns the value of 5 to the variable `x`

`y <- 2*x` assigns the value of 2 times `x` (`2*5`, 10 in this case) to the variable `y`

`r <- 4`

`area.circle <- pi*r^2`

NOTE: R is case-sensitive, the variable `y` \neq `Y`

R object types

Vector: a one-dimensional array, all elements of a vector must be of the same type (numerical, character, etc)

Matrix: a two-dimensional array with rows and columns

Array: as a matrix, but of arbitrary dimension; again all elements must be of the same type

Data frame: a set of data organized similarly to a matrix. Each column of the data frame may contain its own type of data (numeric, character, etc)

Function: Built in and user-created functions (e.g., `min`, `mean`, `var`, etc)

List: a collection of R objects

Entering data into R

Vectors

The easiest way to enter data directly into R is by using the `"c"` command to combine or concatenate data. Data entered using the `"c"` command can be either numeric or character. For example, if we want to enter the numbers 12, 5, 6, 8, 24 into a vector called `v1`, we would enter:

```
v1<-c(12, 5, 6, 8, 24)
```

If we wanted to enter the names of the five fish species into a vector called `v2` we would enter:

```
v2<-c("Yellow perch", "Largemouth bass", "Rainbow trout", "Lake whitefish",  
"Northern pike")
```

It is very easy to generate a sequence of numbers in R. If we wanted to generate a vector of years from 1990 to 2007 into a vector called `years` we would enter:

```
years<-c(1990:2007)
```

A sequence of numbers can also be generated in reverse order (e.g., the construction of `30:1` generates a sequence of `30 - 1`).

NOTE: When working in the R console, you can toggle through previously written code by using the up and down arrow keys (which saves retyping really long commands).

Arrays

Arrays or matrices can easily be generated by using the array or matrix function. To create a 4 row by 5 column matrix called m1 using the array function, enter:

```
m1<-array(1:20, dim=c(4,5))
```

The dim= c(4,5) statement gives the dimensions of the matrix and the 1:20 statement populates the matrix with the number 1 – 20. Note that this will fill the columns before the rows, so the resulting matrix will be:

```
1  5  9 13 17
2  6 10 14 18
...
...
```

NOT

```
1  2  3  4  5
6  7  8  9 10
...
...
```

To create the same matrix using the matrix function, and naming it m2, use:

```
m2<-matrix(1:20, ncol=5, nrow=4)
```

Matrices can also be created by combining vectors using the rbind (combines vectors as rows) and cbind (combines vectors as columns). To combine two vectors v1 and v2 as rows or columns into a matrix we would use:

```
m3<-cbind(v1, v2)
```

```
m4<-rbind(v1, v2)
```

Data frames

Vectors can be combined to form data frames and matrices can be converted to data frames using the data.frame function. Creating a data frame using vectors v1 and v2 and by using matrix m1 are as follows:

```
df1<-data.frame(v1, v2) - this is similar to the cbind function above
```

```
df2<-data.frame(m1)
```

Data frames can also be edited in a spreadsheet-like view by using the edit function.

```
df2<-edit(df1)
```

The columns in data frames are referred to as variables.

Example using vectors and matrices and data frames

(Script: *QFC R short course R object types_1_vectors and arrays.R*)

Accessing data from an array, vector, or data frame

Extracting data (elements) from an object in R is a relatively easy task and is accomplished through the use of subscripts. Subscripts describe the location of desired data within a matrix or data frame. Subscripts appear in square brackets. For example, `x[3]` is the third element in the vector called `x`. When accessing data from a matrix or data frame, the subscripts refer to the row and column, respectively. Thus, `y[3,9]` is referring the element in the 3rd row and 9th column of a matrix called `y`. When we do not specify a row or column using a subscript, this implies that we want all the rows or columns in the query. For example, `y[,2:6]` means that we want all the rows for columns 2 through 6 from the object `y`. Alternatively, `z[1:20,]` means that we want all the columns for rows 1 through 20 from the object `z`.

Placing variables in the R search path

When variables in a data frame are used in R, the data frame name followed by a \$ sign and then the variable name is required. For example, if you want to query the values of variable `v3` in data frame `df1` that are greater than 20, the code is as follows:

```
query1<-df1$v3 > 20
```

Alternatively, the `attach()` command allows the variables in the data frame to be called directly. For example, if we attach data frame `df1`, `attach(df1)`, then the above query can be coded as:

```
query1<-v3 > 20
```

Once data frame `df1` is no longer wanted in the search path, the data frame can be removed from the search path using the `detach()` function, `detach(df1)`. It is often a good idea to immediately detach an object after it is no longer needed to ensure that you don't accidentally use a variable from an attached object. Alternatively, the safest approach is not to attach objects.

We can also use logical tests on one or more variables during a query. For example:

```
query1<-df1$v3 > 20
```

`Query1` is a query for the values of variable `v3` located in data frame `df1` that are greater than 20. To print the results of this query we must use:

```
df1[query1,]
```

The above code is asking for the rows and all the columns that meet the criteria of query1, specifically those rows that have a value of $v3 > 20$.

Multiple logical tests can also be used as follows:

```
Query2<-df1$v3 > 20 & df1$v4 < 30
```

Example querying from a matrix or data frame

(Script: *QFC R short course R object types_2_query arrays data frames.R*)

EXERCISE 1: Vectors, arrays, and data frames

Importing data from Excel (or any other database management program)

Data are typically stored and managed in a data management or manipulation program such as Microsoft Excel or Access. Once data have been entered into a program such as Excel, it is then easy to save the data (e.g., the spreadsheet) as a tab delimited text (.txt) file. This text file can then be read into R using the *read.table()* function. Here are some useful tips to remember before saving your spreadsheet as a text file.

1. You do not want any spaces in your heading names, so either make all heading names one word, or replace the space with a period (this can be easily done using the find-replace tool in Excel).
2. If you have character fields that contain, for example lake names consisting of multiple words, replace the space between words with a period.
3. If there are missing data in your database, replace the missing values with NA. Other values such as -9999 can also be used, but NA is often used in R.

The basic syntax for the *read.table()* function is as follows:

```
data.frame.name<-read.table("file path", na.strings="NA", header=TRUE)
```

```
df1<-read.table("C:\\R\\Example\\datafile1.txt", na.strings="NA", header=TRUE)
```

Note that the path name uses double backslashes. In R, the single backslash (\) is an escape character. The *na.strings="NA"* tells R that missing values are designated with NA. The *header=TRUE* indicates that the first row contains the names of the variables. When the text file is imported, character vectors are automatically converted to factors.

Note: If you set the directory in your R workspace to where your data file (e.g., text file) is found, then you do not need to write the path to your file as above. Instead, you can simply write the name of your data file in parentheses as follows:

```
df<-read.table("datafile1.txt", na.strings="NA", header=TRUE)
```

*This assumes you have set your working directory and your data file is located in that directory.

Exporting data from R

Once analyses have been performed it is often of interest to export data frames or objects from R. This can be accomplished using the `write.table()` function. The syntax for writing a CSV file from a data frame called `df` is:

```
write.table(df, file = "Path Name\\file_name.csv", sep = ",", col.names = NA)
```

Note that other file formats can be exported as well, see R help for details.

Introduction to R functions

R has many built-in functions and many more that can be downloaded from CRAN sites. User-defined functions can also be created, but here we will focus primarily on commonly used R functions that are available in the *base* package. There are many R functions for performing descriptive statistics on data sets (e.g., calculate mean, standard deviation, min, max, etc). Most are fairly intuitive and easy to implement. An example of commonly used functions to obtain summaries of objects and to calculate basic descriptive statistics is as follows.

To obtain the variable names of a data frame

```
var.names <- names(df)
```

To obtain the mean of variable `x` in data frame `df`

```
mean.x <- mean(df$x)
```

To obtain the variance of variable `x` in data frame `df`

```
var.x<-var(df$x)
```

To obtain the standard deviation of variable `x` in data frame `df`

```
sd.x<-sd(df$x)
```

To obtain a summary (min, 1st quantile, median, 3rd quantile, and max) of variable `x` in data frame `df`

```
sum1<-summary(df$x)
```

To obtain a summary of all the variables in data frame `df`

```
sum2<-summary(df)
```

Example using R functions 1

(Script: *QFC R short course R functions_1.R*)

R functions on a ‘real’ data set

The above examples deal with data frames or objects that have no missing data. Most functions in R have to be “told” to exclude missing values during the calculation of desired quantities. For example, if we want the mean value of a vector called `tp.ugL` (i.e., total phosphorus in ug/L) and the vector contains some missing values (e.g., some cells are NA), then the following command will return NA as below:

```
mean.tp1<-mean(tp.ugL)
```

```
mean.tp1  
[1] NA
```

The NA is returned because R can't use missing values (NA) to estimate the mean. You need to specify that R is to exclude missing values, the *mean* function needs to be modified as follows:

```
mean.tp2<-mean(tp.ugL, na.rm = TRUE)
```

The function now states to remove missing values (na.rm=TRUE). The na.rm= statement can be used in many other functions where missing data needs to be excluded.

To obtain column-wise summaries the *sapply()* function can be used as follows:

```
sapply(data, FUN="mean", na.rm=T)
```

The FUN= statement describes the function to apply to each column (e.g., mean, sd, var).

To obtain mean by groups the *tapply()* function can be used as follows:

```
tapply(variable, list(group1, group2), mean)
```

The variable is the variable to calculate means for, the list statement contains the grouping variables.

The *by()* function can also be used to calculate summaries by a grouping variable. For example we can obtain summary statistics of columns 3:6 by county:

```
county.means<-by(lake.data[,3:6], county, summary, na.rm=TRUE)
```

The above code produced summaries for columns 3:6 from the data frame lake.data by county.

To obtain the different levels of a character vector, the *levels()* function can be used as follows. Suppose we have a character variable called 'county' that is composed of several county names and we want to print the names of each county (or level of this character vector). We do the following:

```
county.levels<-levels(county)
```

Example using R functions 2

(Script: *QFC R short course R functions_2.R*)

Simple user-defined functions

There are times when a function is not available in R, so the user must create one. Here we will just briefly discuss how to create a very simple function using the *function()* command in R. In general the function statement will require the name of the function, followed by the *function()*

command, followed the code describing what the function will do (enclosed in braces). To create a function that squares a number, we can do the following:

```
sqr<-function(x){  
  x*x}
```

Example using R functions

(Script: *QFC R short course R user defined functions_3.R*)

EXERCISE 2 (Part 1): Importing text files, summarizing data using basic R functions

R functions part 2: subset data

Subsetting data is an essential part of data management. There are several ways to subset data in R. One approach is to use the *subset()* function to subset rows.

```
sub1<-subset(fish, no.fish > 50)
```

The above code returns the sites with a number of fish (no.fish) greater than 50 from the data frame fish.

Multiple criteria can also be used with the subset function.

```
sub2<-subset(fish, no.fish>50 & position=="Below")
```

The above code returns sites with a number of fish greater than 50 and those that are in the position "Below".

The subset function can also be used to subset the data by selecting specific columns of data.

```
sub3<-subset(fish, select=c(stream, site, no.fish))
```

The above code subsets the data frame fish into a data frame called sub3 that consists of the columns stream, site, and no.fish.

(Script: *QFC R short course R subset_4.R*)

EXERCISE 2 (Part 2): Importing text files, summarizing data using basic R functions

Introduction to basic graphing

R provides a powerful and flexible graphing environment. Plotting commands can be broken down into three main types.

1. High-level functions: These functions create a new plot on the graphics device.
2. Low-level functions: These functions add more information to an already existing plot, such as extra points, lines, and labels.
3. Interactive graphing functions: These functions allow you to interactively add information to a graph.

We will focus on high-level functions for this section. Some of the basic high-level plotting functions include:

plot(): A generic function that produces a type of plot that is dependent on the type of the first argument

hist(): Creates a histogram of frequencies

barplot(): Creates a histogram of values

boxplot(): Creates a boxplot

pairs(): Creates a scatter plot matrix

To plot the values of x simply use the *plot()* function as follows:

```
plot(x)
```

To create a scatter plot of variables x and y, the plot function can be used with two variables.

```
plot(x,y)
```

The *plot()* function can also be used to generate box plots if the first variable is a factor and the second variable is numeric.

```
plot(group,x)
```

To plot a histogram and barplot of x, simply use:

```
hist(x)
```

```
barplot(x)
```

To create a boxplot using the *boxplot()* function, you need to use a formula using a tilde (~) to separate the numeric and character variables.

```
boxplot(x~group)
```

A scatter plot matrix can be useful for examining several bivariate relationships and is implemented using:

```
pairs(z)
```

Where z is a data frame containing the numeric variables you wish to plot.

(Scripts: *QFC R course Graphing Basics_1.R*)

EXERCISE 3: Graphing basics

DAY 2

Graphing: using lower-level functions

R provides a very flexible graphing environment, and we will focus on just a few useful lower-level functions that allow the user to add items to a graph and make them publication quality. Common functions are those that allow the user to modify axes, labels, and add points or lines to an existing graph. R uses variable names to label the x and y axes of a graph. This is often undesirable and more descriptive labels need to be added. Labels can be added using the following:

```
plot(x1,y1, xlab="x-label here", ylab="y-label here")
```

When adding text to a graph, such as labels, use quotation marks (single or double are ok).

To add a main title to a graph, simply use the `main=` statement.

```
plot(x1,y1, xlab="x-label here", ylab="y-label here", main="Practice Graph")
```

To change the symbol shape and color of a graph, use the `pch=` and `col=`, respectively. See the R program “QFC short course graph symbol and color example.R” for details of shapes and colors. The size of the symbol is changed using the `cex=` statement for character expansion. The `cex` statement also changes the font size of titles and axis labels using the `cex.main=` and `cex.lab=` statements.

The scales of graphs are set by R by default to correspond with the range of the data. To change the x and y-axis scales use the `xlim=` and `ylim=` statements. These statements are followed by the `c` command and the lower and upper limit of the axis. For example:

```
plot(x1,y1, xlim=c(0,50), ylim=c(0,100))
```

If the user wants to add lines, points, or text to an already existing graph, these statements must follow the higher-level function that created the graph. To add lines to a graph use the `abline()` function.

```
plot(x)
abline(h=mean(x, na.rm=T))
```

The above commands plot the variable `x` and then adds a horizontal line (the `h` can be replaced with a `v` for a vertical line) at the mean value of `x`. The `abline()` function can be used to add a line of a desired slope and intercept to a graph using the format of:

```
abline(a,b)
```

This code places a line with intercept `a` and slope `b` on a graph. The line style can be changed by using the `lty=` command: 1 = solid line, 2 = dashed, 3 = dotted.

The `text()` function is used to add text to a graph. The text is placed on a graph using `x,y` coordinates. For example,

```
text(9,130,"Mean", cex=0.8)
```

The above code places the text “Mean” on a graph at the x,y coordinates of 9,130 and decreases the font size from default by using the cex=0.8 option.

Points are added to a graph using a similar format, by using x,y coordinates. For example,

```
points(20,150, cex=2, pch=5, col="Blue")
```

The above code places a point at the coordinates 20,150, increases the size from default, changes the symbol character to a diamond, and changes the color to blue.

(Scripts: *QFC R course Graphing Basics_2.R*, *QFC R course Graphing Basics_3.R*)

EXERCISE 4: Graphing basics 2

Introduction to statistical analyses

R provides many functions for analyzing data, from descriptive statistics to inferential statistics. As we have seen, obtaining descriptive statistics is fairly straightforward in R. For instance we can obtain a correlation matrix using the *cor()* function as follows:

```
cor(df1[,2:6])
```

The above code provides a correlation matrix of the variables in columns 2-6 that are located in data frame df1.

For more complex analyses the user must become familiar with how models are specified in R, because R is not a “point-and-click” programming environment. The basic structure of models in R is as follows:

response variable ~ predictor variable(s)

The response variable and predictor variable are separated by the tilde (~) symbol. The right side of the tilde shows the predictor variables, any interactions between predictor variables, and non-linear terms in the predictor variables.

Symbols in model statements are used differently compared to arithmetic expressions.

Symbol	Meaning
+	Indicates inclusion of a predictor variable, not addition
-	Indicates the deletion of a predictor variable, not subtraction
*	Indicates inclusion of a predictor variable and an interaction, not multiplication
/	Indicates nesting of predictor variables, not division
	Indicates conditioning
:	Indicates an interaction (e.g., A:B is a two-way interaction between A and B)

Note that both the response variable and predictor variable(s) can appear as transformations, or as powers, or polynomials.

For a simple example of using a formula, we can perform a t-test to compare the means of two groups using the `t.test()` function as follows:

```
t.test(length~hatchery)
```

The above code compares the mean lengths of fish in two hatcheries.

Simple linear regression:

Model specification in R is simplified if one has some understanding of how models are specified in matrix notation. We will use simple linear regression as an example:

Suppose you have n observations of y and associated observed values for x and you wish to fit a linear function:

$$y_i = \beta_0 + \beta_1 x_i + e_i \quad i = 1, 2, \dots, n$$

where β_0 is the intercept, β_1 is the slope, and e_i is the error for the i^{th} data point. Using matrix notation we can re-express the above equation as:

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \beta_0 \begin{pmatrix} 1 \\ 1 \\ \dots \\ 1 \end{pmatrix} + \beta_1 \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + e_i \begin{pmatrix} e_1 \\ e_2 \\ \dots \\ e_n \end{pmatrix}, = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}, \text{ where } \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{pmatrix} \text{ is the design matrix } X.$$

The model can be rewritten as $Y = X\beta + e$, and we can solve for the parameters β_0 and β_1 by solving the equation:

$$\beta = (X^T X)^{-1} X^T Y$$

(Scripts: *QFC R short course simple linear regression example 1.R*)

In R, a linear model can be fit by either implicitly or explicitly including the intercept (the column of 1's in the design matrix) in the model by:

y~x or

y~1+x

A regression can be fit through the origin by:

$y \sim 0 + x$

Other examples of models include:

$y \sim A$

where A is a categorical variable as in an ANOVA

$y \sim A + x$

where x is a covariate and A is a factor

$y \sim A * B$ or

$y \sim A + B + A : B$

are ANOVAs with interactions.

All the above models can be fit using the linear model *lm()* function in R. Models variables can be specified after attaching the data frame, using the \$ (component selection) syntax, or by using the data= statement in the model statement as follows:

```
modell1<-lm(length~wgt, data=reg1)
```

To obtain a summary of the fit of modell use:

```
summary(modell1)
```

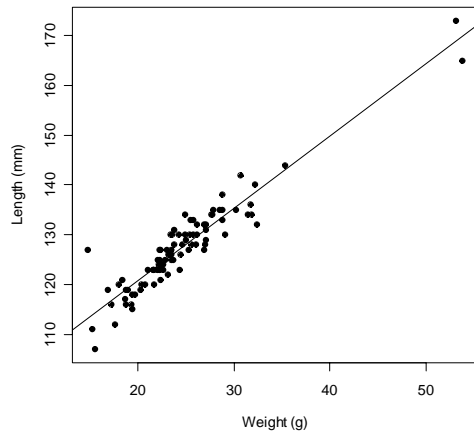
Residuals and fitted values can be obtained easily by using:

```
residuals(modell1)
```

```
fitted(modell1)
```

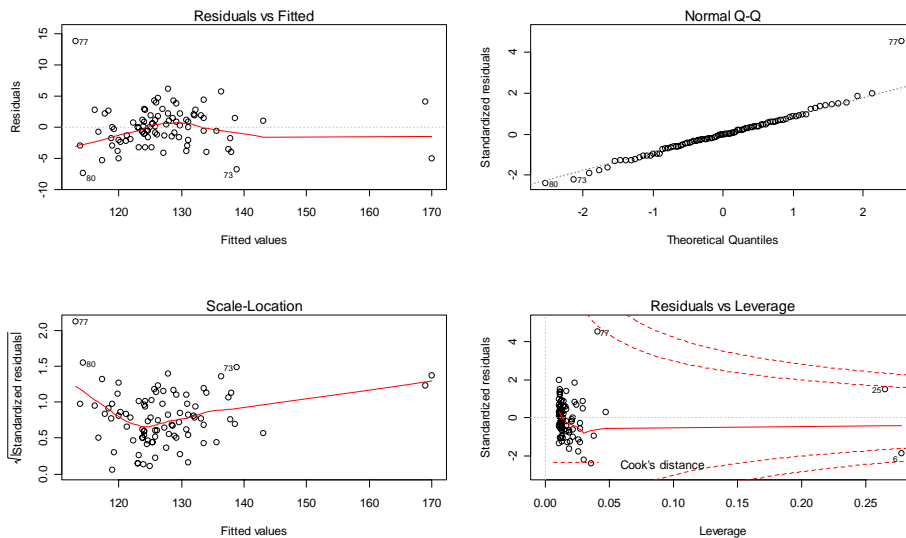
Once parameters have been estimated, it is often desirable to plot the data along with the fitted regression line. This can be accomplished by using the *plot()* and *abline()* functions to produce the following plot.

```
plot(length~wgt, data=reg1)  
abline(lm(length~wgt, data=reg1))
```



Diagnostic plots are obtained for a model by using the `plot()` function with the model name as follows:

```
plot(model1)
```



It is also common to transform variables prior to performing analyses. There are several ways to accomplish this in R. First, one could create a new variable consisting of, for example, the log a variable:

```
df$log.x <- log(df$x)
```

This new variables `log.x` can then be used in the model statement. Alternatively, you can specify the log of a variable (or other transformations) directly within the model statement such as:

```
model2<-lm(log(length)~log(wgt), data=reg1)
```

Plots can be obtained the same way.

```
plot(log(length)~log(wgt), data=reg1)
abline(lm(log(length)~log(wgt), data=reg1))
```

Multiple regression is accomplished by adding additional explanatory variables to the model statement as follows:

```
model4<-lm(trigly ~ length + chol, data=reg1)
```

It is often the case where you will need to perform an analysis on a subset of data. We have already discussed how to subset data, however, this can also be accomplished within the model statement.

```
lm(length ~ wgt, subset=hatchery=="DWOR", data=reg1)
```

The above code performs a linear regression on only Dworshak hatchery fish (DWOR). A subset can also be performed on continuous variables as follows:

```
lm(length ~ wgt, subset=length < 140, data=reg1)
```

(Scripts: QFC R short course simple linear regression example_2.R, QFC R short course simple linear regression example_3.R)

Analysis of variance:

When explanatory variables are categorical, it is often of interest to compare the means in each group or category. In R the *aov()* function is used for analysis of variance. The syntax is very similar to that of *lm()*. To compare means in two or more groups, simply construct a model statement as previously described. To compare the mean triglycerides (trigly) between the three hatcheries, simply write:

```
anova1<-aov(anova$trigly~anova$hatchery)
summary(anova1)
summary.lm(anova1)
```

The summary and summary.lm provide summaries of the output from the aov statement. Means of these groups can also be plotted as previously described.

(Script: QFC short course ANOVA 1.R)

EXERCISE 5: Linear regression and ANOVA

Nonlinear regression

Non-linear regression is commonly used in ecology. In R, the *nls()* function can be used to fit nonlinear least squares regression models. A difference between linear and nonlinear regressions in R, is that for nonlinear regression models the user must specify the exact equation as part of the model statement. The user must also specify initial guesses as to the value of the parameters that are being estimated.

As an example of fitting a nonlinear regression model, we will focus on the Von Bertalanffy (VB) growth model. The VB growth model is a model of growth in length used commonly in fisheries. The model is a three parameter model (von Bertalanffy 1938).

$$L_t = L_{\infty} \left(1 - e^{-k[t-t_0]} \right) + error$$

Where L_t is the length at age t , L_{∞} is the asymptotic average maximum length, k is the growth rate coefficient that determines how quickly the maximum size is attained, and t_0 is the hypothetical age which the species has zero length.

The model in R can be specified as follows:

```
vonB1<-nls(length~Linf*(1-exp(-k*(age-to))), data=length.age,  
start=list(Linf=1000, k=0.05, to=-2))
```

Note, that the entire VB equation is specified in the model statement, along with initial values for the parameters Linf, k, and to. Once parameters are estimated they can be obtained as with linear regression by using the *summary()* function.

Graphing the fitted regression line, however, is a little more complicated compared to simple linear regression. One approach to graphing a fitted nonlinear regression fit is to predict values using the estimated parameters from the model fit and then overlaying the predictions as a line on the original scatter plot.

1. Create a sequence of ages (x-values) from 0 to 16 by 1 (a vector called age2)

```
age2<-seq(0,16,1)
```

2. Predict the lengths using estimated parameters and x-values in age2

```
y1<-1247.537*(1-exp(-0.06164*(age2--4.6745)))
```

3. Plot original data points

```
plot(length~age, data=length.age, xlim=c(0,17), pch=16)
```

4. Overlay predicted values from models

```
lines(age2,y1)
```

(Script: QFC R course Von Bertalanffy Nonlinear regression 6 Oct 2006.R)

EXERCISE 6: Nonlinear regression

Useful R references (descriptions are from the R website)

Richard A. Becker, John M. Chambers, and Allan R. Wilks. *The New S Language*. Chapman & Hall, London, 1988.

This book is often called the “*Blue Book*”, and introduced what is now known as S version 2.

John M. Chambers and Trevor J. Hastie. *Statistical Models in S*. Chapman & Hall, London, 1992.

This is also called the “*White Book*”, and introduced S version 3, which added structures to facilitate statistical modeling in S.

John M. Chambers. *Programming with Data*. Springer, New York, 1998. ISBN 0-387-98503-4. <http://cm.bell-labs.com/cm/ms/departments/sia/Sbook/>]

This “*Green Book*” describes version 4 of S, a major revision of S designed by John Chambers to improve its usefulness at every stage of the programming process.

William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S. Fourth Edition*. Springer, New York, 2002. ISBN 0-387-95457-0. <http://www.stats.ox.ac.uk/pub/MASS4/>]

A highly recommended book on how to do statistical data analysis using R or S-Plus. In the first chapters it gives an introduction to the S language. Then it covers a wide range of statistical methodology, including linear and generalized linear models, non-linear and smooth regression, tree-based methods, random and mixed effects, exploratory multivariate analysis, classification, survival analysis, time series analysis, spatial statistics, and optimization. The ‘on-line complements’ available at the books homepage provide updates of the book, as well as further details of technical material.

William N. Venables and Brian D. Ripley. *S Programming*. Springer, New York, 2000. ISBN 0-387-98966-8. <http://www.stats.ox.ac.uk/pub/MASS3/Sprog/>]

This provides an in-depth guide to writing software in the S language which forms the basis of both the commercial S-Plus and the Open Source R data analysis software systems.

Jose C. Pinheiro and Douglas M. Bates. *Mixed-Effects Models in S and S-Plus*. Springer, 2000. ISBN 0-387-98957-0.

A comprehensive guide to the use of the ‘nlme’ package for linear and nonlinear mixed-effects models.

John Fox. *An R and S-Plus Companion to Applied Regression*. Sage Publications, Thousand Oaks, CA, USA, 2002. ISBN 0-761-92279-2.

<http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/index.html>]

A companion book to a text or course on applied regression (such as “Applied Regression, Linear Models, and Related Methods” by the same author). It introduces S, and concentrates on how to use linear and generalized-linear models in S while assuming familiarity with the statistical methodology.

Peter Dalgaard. *Introductory Statistics with R*. Springer, 2002. ISBN 0-387-95475-9.

<http://www.biostat.ku.dk/~pd/ISwR.html>]

John Maindonald and John Braun. *Data Analysis and Graphics Using R*. Cambridge University Press, Cambridge, 2003. ISBN 0-521-81336-0. <http://www.maths.anu.edu.au/~johnm/r-book.html>

Sylvie Huet, Annie Bouvier, Marie-Anne Gruet, and Emmanuel Jolivet. *Statistical Tools for Nonlinear Regression*. Springer, New York, 2003. ISBN 0-387-40081-8.

Richard M. Heiberger and Burt Holland. *Statistical Analysis and Data Display: An Intermediate Course with Examples in S-Plus, R, and SAS*. Springer Texts in Statistics. Springer, 2004. ISBN 0-387-40270-5. <http://astro.temple.edu/~rmh/HH>]

A contemporary presentation of statistical methods featuring 200 graphical displays for exploring data and displaying analyses. Many of the displays appear here for the first time. Discusses construction and interpretation of graphs, principles of graphical design, and relation between graphs and traditional tabular results. Can serve as a graduate-level standalone statistics text and as a reference book for researchers. In-depth discussions of regression analysis, analysis of variance, and design of experiments are followed by introductions to analysis of discrete bivariate data, nonparametrics, logistic regression, and ARIMA time series modeling. Concepts and techniques are illustrated with a variety of case studies. S-Plus, R, and SAS executable functions are provided and discussed. S functions are provided for each new graphical display format. All code, transcript and figure files are provided for readers to use as templates for their own analyses.

John Verzani. *Using R for Introductory Statistics*. Chapman & Hall/CRC, Boca Raton, FL, 2005. ISBN 1-584-88450-9. <http://wiener.math.csi.cuny.edu/UsingR/>]

There are few books covering introductory statistics using R, and this book fills a gap as a true “beginner” book. With emphasis on data analysis and practical examples, ‘Using R for Introductory Statistics’ encourages understanding rather than focusing on learning the underlying theory. It includes a large collection of exercises and numerous practical examples from a broad range of scientific disciplines. It comes complete with an online resource containing datasets, R functions, selected solutions to exercises, and updates to the latest features. A full solutions manual is available from Chapman & Hall/CRC.

Paul Murrell. *R Graphics*. Chapman & Hall/CRC, Boca Raton, FL, 2005. ISBN 1-584-88486-X. <http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html>]

A description of the core graphics features of R including: a brief introduction to R; an introduction to general R graphics features. The base graphics system of R: traditional S graphics. The power and flexibility of grid graphics. Building on top of the base or grid graphics: Trellis graphics and developing new graphics functions.

Michael J. Crawley. *Statistics: An Introduction using R*. Wiley, 2005. ISBN 0-470-02297-3. <http://www.bio.ic.ac.uk/research/crawley/statistics/>]

The book is primarily aimed at undergraduate students in medicine, engineering, economics and biology - but will also appeal to postgraduates who have not previously covered this area, or wish to switch to using R.

Brian S. Everitt. *An R and S-Plus Companion to Multivariate Analysis*. Springer, 2005. ISBN 1-85233-882-2. <http://biostatistics.iop.kcl.ac.uk/publications/everitt/>]

In this book the core multivariate methodology is covered along with some basic theory for each method described. The necessary R and S-Plus code is given for each analysis in the book, with any differences between the two highlighted.

R QUICK REFERENCE SHEET

Website: <http://www.r-project.org/>

Getting started

- Open R by double clicking on the R icon.
- Commands are entered into the “R Console” window (the command window within the graphical user interface (RGui) or into R scripts.
- Command can also be copied and pasted into the R console or scripts.
- Close R by typing q() in the R console or via the File menu, or by clicking the close button on the top right of the screen.
- When you close R you will be prompted to “Save workspace image?”. This will save all the currently defined objects and is automatically restored when R is started.
- Parentheses () are for functions.
- Brackets [] are for indicating the position of items in a vector or matrix.
- R commands are case sensitive
- You can use the keystroke <Ctrl> <L> to clear the R console

General commands

<u>Command</u>	<u>Function</u>
#	Add a comment
<-	Assign
(less than and minus sign)	
q()	Quit
Help(mean)	Get help on a function called mean
?mean	Get help on a function called mean

Managing objects

<u>Command</u>	<u>Function</u>
objects() or ls()	Display a list of all objects currently stored within R
rm(x, growth)	Remove objects x and growth
dim(matrix1)	Returns the dimensions of a matrix called matrix1
dimnames(matrix1)	Returns the names of dimensions of a matrix called matrix1
length(date)	Returns the length of the vector called date
rep(x, n)	Repeat the vector x n times
cbind(x, y, z)	Combine columns into a matrix
rbind(x2, y2, z2)	Combines rows into a matrix
t(x)	Switch rows and columns (transpose)
data.frame(matrix1)	Create a data frame from a matrix
attach(fish.length)	Make variables in object fish.length accessible by name within R (put variables in search path)
detach(fish.length)	Remove variables in object fish.length from search path
names(fish.length)	Returns a list of variable names in object fish.length

<code>merge(df1, df2)</code>	Merge data two data frames
<code>levels(site.name)</code>	Returns a list of the names of the different categories of site.name
<code>\$</code>	Designate the object a variable is coming from. For example, <code>fish.growth\$length</code> refers to the variable length in the object (e.g., data frame) <code>fish.growth</code>
<code>head(df1)</code>	Show the first few lines of a data frame or matrix
<code>tail(df1)</code>	Show the last few lines of a data frame or matrix
<code>page(x)</code>	Show the structure of a data frame or matrix
<code>is.factor(x1)</code>	Returns TRUE/FALSE if x1 is a factor
<code>is.matrix(x)</code>	Returns TRUE/FALSE if x is a matrix
<code>as.factor(x1)</code>	Encode a vector as a factor
<code>as.matrix(x)</code>	Creates a matrix from a given set of values
<code>read.table</code>	Read a text file into R
<code>write.tb1e</code>	Export a data frame or matrix out of R

Conditional statements

Command

`which(x>y)`
`which(x1==y | x2=="z")`
`ifelse(test, yes, no)`
 If else

Function

Identifies which x variables are greater than y
 Identify variables where x1 equals y OR where x2 = "z"
 Returns the value for yes if the test is true, no otherwise

Logical arguments

Command

`=`
`==`
`!x`
`x & y`
`x && y`
`x | y`
`x || y`

Function

Logical equals
 Logical equals
 Indicates logical negation (not), not x
 Logical and, x and y
 Logical and
 Logical or, x or y
 Logical or

The shorter form of `&` and `|`, perform element-wise comparisons in much the same way as arithmetic operators. The longer form (`&&`, `||`) evaluates left to right examining only the first element of each vector. Evaluation proceeds only until the result is determined. The longer form is appropriate for programming control-flow

Descriptive Statistics

Command

`max(x)`
`min(x)`
`mean(x)`

Function

Max of x
 Min of x
 Mean of x

median(x)	Median of x
sum(x)	Sum of x
var(x)	Variance of x
sd(x)	Standard deviation of x
quantile(x)	Generates multiple quintiles of x
log(x, base)	Computes logarithm of x with base <i>base</i>
colSum(df)	Column-wise sums for a data frame
colMean(df)	Column-wise means of a data frame
apply(x, 2, function)	Column-wise estimation of function (e.g., median, mean, etc) for matrix x
sapply(df, FUN="")	Column-wise calculation of a function
lapply(df, FUN="")	Column-wise calculation of a function
ave(x1, group)	Average of x1 by group
by(df[,1:2], group, mean)	Average of column 1 and 2 in a data frame by a group name

High-level plotting functions (create a new plot on the graphics device)

<u>Command</u>	<u>Function</u>
plot(x,y)	Plot x versus y (generic function, see R help)
hist(x)	Histogram of frequencies of x
barplot(x)	Histogram of values of x
boxplot(x)	Boxplot of x
pairs(df)	Scatter plot matrix of variables in a data frame
For more graphing options see the Lattice library for the mplementation of Trellis Graphics in R.	

Common axis and label functions

<u>Command</u>	<u>Function</u>
xlim=	Specify x-axis lower and upper limits e.g., xlim=c(0,100)
ylim=	Specify y-axis lower and upper limits
xlab=	Specify x-axis label
ylab=	Specify y-axis label
main=	Specify main graph title
sub=	Specify sub-title
add=TRUE	Superimposes a plot on a previous plot

Some low-level plotting functions (add more information to an existing plot)

<u>Command</u>	<u>Function</u>
pch=	Change plotting character
cex=	Change size of character
cex.axis=	Change font size of axes

<code>cex.lab=</code>	Change font size of axis labels
<code>cex.main=</code>	Change size of main title
<code>col.main=</code>	Change color of main title
<code>col.lab=</code>	Change color of axis labels
<code>col.axis=</code>	Change font color of axes
<code>col=</code>	Change color of symbols
<code>abline(a,b)</code>	Draws a line of slope b and intercept a
<code>abline(h=y)</code>	Draws a horizontal line at ordinate y
<code>abline(v=x)</code>	Draws a vertical line at abscissa x
<code>points(x,y)</code>	Adds points
<code>lty=</code>	Changes the line type
<code>lwd=</code>	Change the width of a line
<code>font=</code>	1=normal; 2=italics; 3=bold; 4=bold italics
<code>colors()</code>	List all named colors in R
<code>expression()</code>	Used to create mathematical symbols, superscript, and subscript axis labels or added text to a graph

Commonly used statistical analyses

lm: fits a linear model with normal errors and constant variance. It can be used to carry out single stratum analysis of variance and analysis of covariance (although 'aov' may provide a more convenient interface for these).

aov: fits an analysis of variance model by a call to 'lm' for each stratum. aov fits analysis of variance with normal errors, constant variance, and the identity link.

glm: used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution (e.g., Poisson for count data or binomial for proportion data).

gam: used to fit generalized additive models, specified by giving a symbolic description of the additive predictor and a description of the error distribution. 'gam' uses the backfitting algorithm to combine different smoothing or fitting methods. The methods currently supported are local regression and smoothing splines. (gam package).

lme: fits a linear mixed-effects models with specified mixtures of fixed and random effects and allows for the specification of correlation structure among the explanatory variables and autocorrelation of the response variable (package nlme).

nls: determine the nonlinear (weighted) least-squares estimates of the parameters of a nonlinear model.

nlme: this generic function fits a nonlinear mixed-effects model in the formulation described in Lindstrom and Bates (1990) but allowing for nested random effects. The within-group errors are allowed to be correlated and/or have unequal variances (package nlme).

loess: fit a polynomial surface determined by one or more numerical predictors, using local fitting.

tree: fits a regression tree model. A tree is grown by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side. (package tree)

Functions used to obtain information about statistical models

summary: produces parameter estimates and standard errors from lm and ANOVA tables from aov.

plot: produces diagnostic plots for model checking.

anova: compares different models and produces ANOVA tables.

update: used to modify the latest model fit.

coef: returns the estimated coefficients and sometimes their standard errors

fitted: returns the fitted values

resid: returns residuals

predict: prediction from a model

AIC: computes the Akaike information criterion

logLik: computes the logarithm of the likelihood and the number of parameters

Symbols in model statements are used differently compared to arithmetic expressions.

Symbol	Meaning
+	Indicates inclusion of a predictor variable, not addition
-	Indicates the deletion of a predictor variable, not subtraction
*	Indicates inclusion of a predictor variable and an interaction, not multiplication
/	Indicates nesting of predictor variables, not division
	Indicates conditioning
:	Indicates an interaction (e.g., A:B is a two-way interaction between A and B)

Examples of model formulae (From Crawley 2005)

Model	Model formula	Comments
Null	$y \sim 1$	1 is the intercept in regression models, but here it is the overall mean
Regression	$y \sim x$	x is a continuous explanatory variable

One-way ANOVA	$y \sim \text{gender}$	Gender is a two-level categorical variable
Two-way ANOVA	$y \sim \text{gender} + \text{genotype}$	Genotype is a four-level categorical variable
Factorial ANOVA	$y \sim N * P * K$	N, P, and K are two-level factors to be fit along with all their interactions
Three-way ANOVA	$y \sim N * P * K - N:P:K$	As above, but don't fit the three-way interaction
Analysis of covariance	$y \sim x + \text{gender}$	A common slope for y against x but with two intercepts, one for each gender
Analysis of covariance	$y \sim x * \text{gender}$	Two slopes and two intercepts
Nested ANOVA	$y \sim a / b / c$	Factor c nested within factor b within factor a
Split-plot ANOVA	$y \sim a * b * c + \text{Error}(a/b/c)$	A factorial experiment but with three plot sizes and three different error variances, one for each plot size
Multiple regression	$y \sim x + z$	Two continuous explanatory variables
Multiple regression	$y \sim x * z$	Fit an interaction term as well ($x+z+x:z$)
Multiple regression	$y \sim x + I(x^2) + z + I(z^2)$	Fit a quadratic term for both x and z
Multiple regression	$y \sim \text{poly}(x,2) + z$	Fit a quadratic polynomial for x and linear z
Non-parametric model	$y \sim s(x) + \text{lo}(z)$	y is a function of a smoothed x and loess z
Transformed response and explanatory variables	$\log(y) \sim I(1/x) + \text{sqrt}(z)$	All three variables are transformed in the model

Mathematical annotation in R

Syntax	Meaning
$x + y$	x plus y
$x - y$	x minus y
x^*y	juxtapose x and y
x/y	x forwardslash y
$x \%+\% y$	x plus or minus y
$x \%/ \% y$	x divided by y
$x \%*\% y$	x times y
$x[i]$	x subscript i
x^2	x superscript 2
$\text{paste}(x, y, z)$	juxtapose x, y, and z
$\text{sqrt}(x)$	square root of x
$\text{sqrt}(x, y)$	yth root of x
$x == y$	x equals y
$x != y$	x is not equal to y
$x < y$	x is less than y
$x <= y$	x is less than or equal to y
$x > y$	x is greater than y
$x >= y$	x is greater than or equal to y
$x \% \sim \% y$	x is approximately equal to y
$x \% \equiv \% y$	x and y are congruent
$x \% == \% y$	x is defined as y
$x \% \text{prop} \% y$	x is proportional to y
$\text{plain}(x)$	draw x in normal font
$\text{bold}(x)$	draw x in bold font
$\text{italic}(x)$	draw x in italic font
$\text{bolditalic}(x)$	draw x in bolditalic font
$\text{list}(x, y, z)$	comma-separated list
\dots	ellipsis (height varies)
cdots	ellipsis (vertically centred)
ldots	ellipsis (at baseline)
$x \% \text{subset} \% y$	x is a proper subset of y
$x \% \text{subsetq} \% y$	x is a subset of y
$x \% \text{notsubset} \% y$	x is not a subset of y
$x \% \text{supset} \% y$	x is a proper superset of y
$x \% \text{supsetq} \% y$	x is a superset of y
$x \% \text{in} \% y$	x is an element of y
$x \% \text{notin} \% y$	x is not an element of y
$\text{hat}(x)$	x with a circumflex
$\text{tilde}(x)$	x with a tilde
$\text{dot}(x)$	x with a dot
$\text{ring}(x)$	x with a ring
$\text{bar}(xy)$	xy with bar
$\text{widehat}(xy)$	xy with a wide circumflex
$\text{widetilde}(xy)$	xy with a wide tilde
$x \% \leftrightarrow \% y$	x double-arrow y

$x \rightarrow y$	x right-arrow y
$x \leftarrow y$	x left-arrow y
$x \uparrow y$	x up-arrow y
$x \downarrow y$	x down-arrow y
$x \Leftrightarrow y$	x is equivalent to y
$x \Rightarrow y$	x implies y
$x \Leftarrow y$	y implies x
$x \Updownarrow y$	x double-up-arrow y
$x \Downarrow y$	x double-down-arrow y
alpha – omega	Greek symbols
Alpha – Omega	uppercase Greek symbols
infinity	infinity symbol
partialdiff	partial differential symbol
32*degree	32 degrees
60*minute	60 minutes of angle
30*second	30 seconds of angle
displaystyle(x)	draw x in normal size (extra spacing)
textstyle(x)	draw x in normal size
scriptstyle(x)	draw x in small size
scriptscriptstyle(x)	draw x in very small size
underline(x)	draw x underlined
$x \sim y$	put extra space between x and y
$x + \text{phantom}(0) + y$	leave gap for "0", but don't draw it
$x + \overline{}$	leave vertical gap for "0" (don't draw)
frac(x, y)	x over y
over(x, y)	x over y
atop(x, y)	x over y (no horizontal bar)
sum(x[i], i==1, n)	sum x[i] for i equals 1 to n
prod(plain(P)(X==x), x)	product of P(X=x) for all values of x
integral(f(x)*dx, a, b)	definite integral of f(x) wrt x
union(A[i], i==1, n)	union of A[i] for i equals 1 to n
intersect(A[i], i==1, n)	intersection of A[i]
lim(f(x), x %>% 0)	limit of f(x) as x tends to 0
min(g(x), x > 0)	minimum of g(x) for x greater than 0
inf(S)	infimum of S
sup(S)	supremum of S
$x^y + z$	normal operator precedence
$x^{(y + z)}$	visible grouping of operands
$x^{\{y + z\}}$	invisible grouping of operands
group("(", list(a, b), ")")	specify left and right delimiters
bgroup("(", atop(x, y), ")")	use scalable delimiters
group(lceil, x, rceil)	special delimiters

The following annotations can be used to add equations to a chart as follows (from plotmath demo):

```
plot(1:10, 1:10)
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)",
      cex = .8)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
text(4, 6.4, "expression(bar(x) == sum(frac(x[i], n), i==1, n))",
      cex = .8)
text(8, 5, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
                             plain(e)^{frac(-(x-mu)^2, 2*sigma^2)})),
      cex = 1.2)
```

