Charles Fleming

R Programming Tips

April 2013

Contents

Contents i

List of Figures ii

1 Introduction 1

2 parse, deparse, substitute, and expression 3
2.1 Manual Concatenation 3
2.2 Using parse 4
2.3 Concatenating by Means of a Function 4
2.4 Generalizing the Concatenation Function 5
2.5 Writing a Formula in a Graph 6

3 Dates 9
3.1 Varieties of Formats for Dates 9

4 Graphics 13
4.1 identify 13
4.2 las=2 and srt=90 14

5 R Commands for the Examples 21

6 Embedded R Program in a UNIX Script 29

Bibliography 33

List of Figures

2.1 Use of expression to Write Equations in a Graph 8

3.1 The Use of Smoothing in a Plot 12

4.1 Specially Marked Points Using identify 15
4.2 Job Losses in Post WWII Recessions 16
4.3 The Use of las=2 and str=90 19
4.4 The Inclusion of a Level Line 19

Chapter 1

Introduction

Upon the divestiture of AT&T in 1984, the source code for the S language became proprietary, whereas the C language remained in the public domain. The GCC compiler which was developed by Richard Stallman was licensed under the General Public License (GPL), and it paved the way for computer programmers to compile programs which were written in the C language for free. Soon afterwards a compiler was developed to compile FORTRAN programs. The availability of the GCC compiler and the GPL has lead to the worldwide development of Linux and eventually to the development of R Because the owners of the source code for the S language refused to port S to the Linux platform, the R language was developed to imitate the syntax of S, but in such a way as to be independent of the source code of S for copyright reasons. Large number of FORTRAN programs which were in the public domain were incorporated into R , and because of the vigorous beta testing and development of R , has supplanted the S language. In fact, John Chambers who led the development of S at AT&T has since become a core developer of R Because of the robust quality of R ,because of its versatility, and by virtue of being licensed under the GPL, the popularity of R has grown immensely.

Because R is governed by the General Public License, the collection of uncompiled programs which constitute the source code of R must be made freely available to anyone who might want to study the logic of a certain procedure, and to allow someone to correct an error or to modify the code, in order to improve its efficiency. If an improvement to the program is deemed a good one by the core developers of R , then it will be implemented into the official version.

There are many libraries or what are nicknamed *add-on* packages which are collections of procedures which serve a specialized purpose. Some of the standard libraries are invoked automatically when R is initiated. Others, because there are so many of them, are not included in the installation of R and must be invoked manually. For example, the library `foreign` is a set of routines which will allow someone to bring a SAS or SPSS set of data into an R session

(Fleming, 2004) [1].

Given the open nature of the source code for R and the multitude of procedures which are available in R together with the propensity of researchers to write their own customized computer programs, an R program might not be transparent when read by another person instead it might be rather difficult to understand the programmer's logic. As a result, care should be exercised in documenting an R program not only for the benefit of someone else, but also for the programmer himself who might refer to the program later but has since forgotten the rationale his own logic.

It seems that no matter how often some commands or methods are used, some of them are not easy to remember. Those commands involving dates always seem to be troublesome; parsing and deparsing are similarly hard to remember as well as certain nice options for making graphs. It is in that light that the following methods have been chosen. They are useful, but they are not easy to remember.

Commands which appear enclosed in a box may be executed by highlighting and pasting them in an R session. The same commands appear in Chapter 5, and they appear in the ASCII file, `demo_bls.txt`, which accompanies this tutorial.

[1] See page 19

Chapter 2

parse, deparse, substitute, and expression

2.1 Manual Concatenation

The `parse` and `deparse` are useful when referring to the names of objects rather than their contents. Parsing in R occurs in three different variants:

- The read-eval-print loop
- Parsing of text files
- Parsing of character strings

The parsing is an artifact of object oriented programming. In R , what appear to be quantities are called objects. There are various classes of objects like list, date, character, and matrix. Functions are referred to as methods. An object is assigned to a name as in:

```
w1<-c(2,3,4,5)
w2<-c(5,4,3,2)
```

hence the `<-` symbol which signifies assignment. `deparse` will turn unevaluated expressions into character strings.

For example, consider the two objects, `w1` and `w2`.

```
str(w1)
```

shows that `w1` is an object which is numeric.

We can concatenate them row by row as follows:

```
rbind(w1,w2)
```

to get:

```
     [,1] [,2] [,3] [,4]
[1,]    2    3    4    5
[2,]    5    4    3    2
```

2.2 Using parse

Let us define the object:

```
aa<- "w1,w2"
```

`aa` might have been defined in a function, and we want to concatenate `w1` with `w2` using only `aa`.

The commands, `rbind(aa[1],aa[2])` and `rbind(aa)`, will not work. The goal can be achieved by using the `parse` command as in:

```
eval(parse(text=paste("rbind(", aa, ")")))
```

`parse(text=paste("rbind(", aa, ")"))` will produce what would have been typed by hand, that is `"rbind(w1,w2)"`. The `eval` command evaluates the expression which `parse` produced.

2.3 Concatenating by Means of a Function

The method of manually concatenating can be generalized by using a function.

```
B<-function(a1,a2){
A<-rbind(a1,a2)
print(A)}
```

The function is well defined since `B(w1,w2)` produces the correct result. It would be nice to print a statement along with the result such as:

```
"We concatenate vector ", w1, " with vector ", w2, " to get:"
```

```
   [,1] [,2] [,3] [,4]
a1    2    3    4    5
a2    5    4    3    2
```

To do something like that in a function, it is necessary to use both `w1` and `w2` as numeric objects and, also, to use their names. The next function is an attempt to generalize the function B, in order to achieve both uses:

```
C<-function(a1,a2){
A<-rbind(a1,a2)
str<-paste("We concatenate vector ", a1, " with vector ", a2, " to get:", sep="")
print(str)
print(A)
}
```

However, `C(w1,w2)` fails. Though the concatenation is successful, the comment is incorrect. In order to pass the name of the objects to the character string, the `deparse` command will be used with the `substitute` command.

```
C<-function(a1,a2){
A<-rbind(a1,a2)
first<-deparse(substitute(a1))
second<-deparse(substitute(a2))
str<-paste("We concatenate vector ", first, " with vector ", second,
" to get:", sep="")
print(str)
print(A)
}
```

Now, `C(w1,w2)` works as desired.

2.4 Generalizing the Concatenation Function

Instead of only two objects, there might be 200 objects. Rather than type many more lines of code into the function `C`, the function will be generalized to accommodate any number of objects.

Define the following objects:

```
w1<-c(2,3,4,5)     ;   x1<-c(12,3,4,5)     ; xyw1<-c(20,30,40,50)      ;
w2<-c(5,4,3,2)     ;   x2<-c(5,24,3,2)     ; xyw2<-c(50,40,30,20)      ;
w3<-c(-2,-3,-4,-5) ;   x3<-c(-2,-33,-4,-5) ; xyw3<-c(-20,-30,-40,-50)  ;
w4<-c(-5,-4,-3,-2) ;   x4<-c(-5,-4,-34,-2) ; xyw4<-c(-50,-40,-30,-20)  ;
w5<-c(1,1,1,1)     ;   x5<-c(11,12,13,15)  ; xyw5<-c(10,10,10,10)      ;
```

The name of an object begins with a name and the objects are indexed sequentially by a number. Therefore, it is necessary to provide the name and the number of objects under that name.

```
D<-function(u,n){
uu<-deparse(substitute(u))
m0<-c()
for(k in 1:n){
m0<-rbind(m0,eval(parse(text=paste(uu,k,sep=""))))
}
return(as.matrix(m0))
}
```

We see that `D(x,5)` works. Such a function might have been written to make it easy to solve for the parameters of a linear model. For example,

```
z<-t(D(x,5))%*%D(x,5)
```

To test whether or not z is singular, the function, `det`, will evaluate its determinant.

```
det<-function(x)(prod(eigen(x)$values))
```

We wee that `det(z)`$\neq 0$ and that `solve(t(D(x,5))%*%D(x,5))` inverts it.

2.5 Writing a Formula in a Graph

The `parse` command together with the `expression` command are used to write mathematical formulae in a graph as is done in this example.

```
data<-read.table(file=
        "/home/mike/gw/sigstat/bls/bls_dat1.txt",
        header=TRUE,sep=" ")
bls_x<-data$bls_x
bls_y<-data$bls_y
verb+t200<-data$t200
time200<-data$time200
failures<-data$failures
passes<-data$passes
coefalpha<-1611.313
betahat <- 33.08167
center <- 1.357384
fig0<-paste("/home/mike/gw/sigstat/bls/dwell_time.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
par(cex=1.25)
plot(bls_x,bls_y,type="n",xlim=c(5,7),ylim=c(-3,4),
        xlab="Velocity",ylab="Dwell Time at 200g",
        main="Relationship between Dwell Time with Velocity")
```

```
failures<-time200[t200]>=2
passes<-time200[t200]<2
points(bls_x[failures],time200[t200][failures],pch=19,col="red")
points(bls_x[t200][passes],time200[t200][passes],pch=19,col="green")
curve(coefalpha/x^2*log(betahat*x/200)+center,xlim=c(5,7),add=TRUE)
 curve(369.4/x^2*log(38.97143*x/200),xlim=c(5,7),add=TRUE,col="red")
curve(369.4/x^2*log(38.97143*x/200)+.17,xlim=c(5,7),add=TRUE,col="red")
abline(v=5.8,lty=3)
abline(v=6.2,lty=3)
abline(h=0,lty=3)
abline(h=2,lty=3)
text(5.6,3,"v0=5.8")
text(6.4,-1,"v0=6.2")
text(6.0,3.5,"Fail",col="red")
text(6.0,-1.5,"Pass",col="green")
eval(parse(text=paste("text(5.4,-1,
expression(f(x)==frac(alpha,x^2)*log(frac(beta*x,200))))",sep="")))
#dev.off()
```

The comment, #, may be removed, in order to create the Postscript file, `dwell_time.ps`, on the computer, otherwise, with the comment symbol in place, the graph will be displayed on the monitor. In either case, Figure 3.1 shows the equation, $f(x) = \frac{\alpha}{x^2} log(\frac{\beta x}{200})$. Note that the syntax for writing equations in R is not the same as the syntax used in LaTeX .

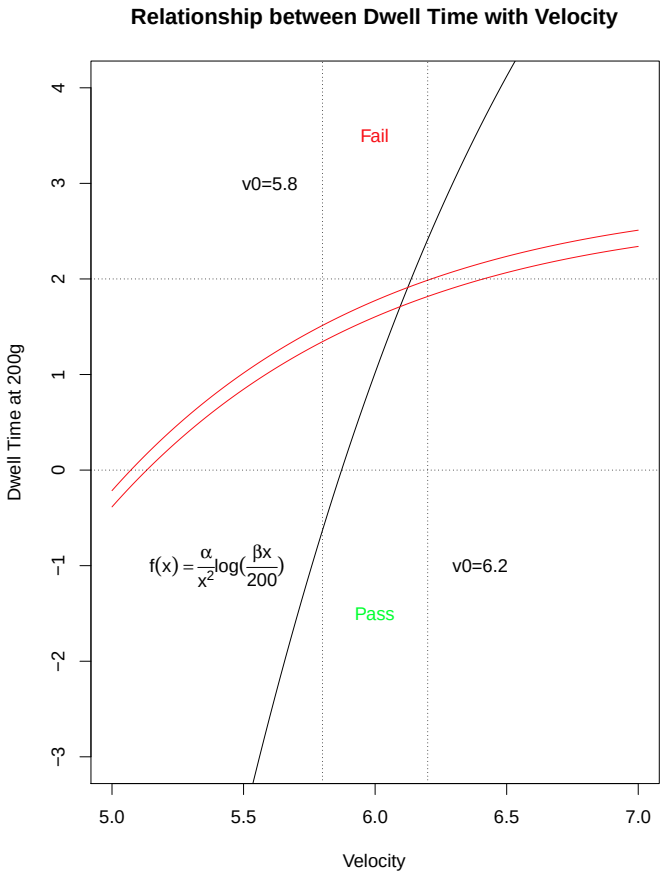

Figure 2.1: Use of expression to Write Equations in a Graph

Chapter 3

Dates

3.1 Varieties of Formats for Dates

Accounting of the variety of formats of date can be a challenge. There are many of them. Once a date has been incorporated in a session of R converting them to Julian dates helps to facilitate their manipulation. In the following example, a beekeeper offered a set of data for analysis. The two most important variables are date and temperature. The beekeeper put four wireless thermometers in one of his hives such that every five minutes the temperatures from all four sensors would be transmitted to his computer for storage. The date and time had the following format:

```
5/29/12 18:11
```

In the R program, that format was converted into a special class of objects meant for dates by means of the `strptime` command. Once an object in such a class, then the R utilities for manipulating dates can be utilized.

```
data_temp<-read.table(file=
        "/home/mike/gw/sigstat/bls/bls_dat2.txt",
        header=TRUE,sep=" ")
date_time<-data_temp$date_time
temp1<-data_temp$temp1
temp2<-data_temp$temp2
temp3<-data_temp$temp3
temp4<-data_temp$temp4
d100<-strptime(as.character(date_time), "%m/%d/%y %H:%M","EDT")
age<-julian(as.POSIXlt(d100))-julian(as.POSIXlt("2012-1-1","EDT"))
```

Notice the specified format used in the `strptime` command. The space, colon, and order of the elements of the date are arranged according to the way the date was read from the set of data into the R session.

After the dates and the temperatures for the four sensors are prepared, then the rest of the program produces a picture of the data. Because the original set of data produces a jagged plot, the `smooth.spline` utility was used to smooth the plot. The
`postscript(horizontal=FALSE, file=fig0)`
and the
`dev.off()`
commands have been made into comments for the sake of producing the graph on the monitor, otherwise, by removing the comments symbol, #, the graph will be made into the Postscript file, `bee_hive.ps`, which will be placed in the directory, `/home/mike/gw/sigstat/bls/`. In the Microsoft system, the path to the directory would be written as:
`C:\home\mike\gw\sigstat\bls`

In the following set of commands which is a continuation of the ones written above, not only is the smoothing performed by means of the method of splines, but the legend command is used as well as a personal convention of putting the name of the output Postscript file for the graph into an object `fig0`. When doing `for` loops, for example, in which the name of the output file must be changed with each new iteration, the `fig0` not only helps to simply the troubleshooting, but it is much easier to create complicated names independently of the `postscript` command. Notice the trick of writing `degrees Centigrade`. The graph, `bee_hive.ps`, appears in Figure 3.1.

```
T1<-cbind(age-age[1],temp1)
T2<-cbind(age-age[1],temp2)
T3<-cbind(age-age[1],temp3)
T4<-cbind(age-age[1],temp4)
fig0<-paste("/home/mike/gw/sigstat/bls/bee_hive.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
ylab0<-expression("Temperature"~(degree*C))
par(cex=1.35)
omit<-which(is.na(T4[,2]))
fig0<-paste("/home/mike/gw/sigstat/bls/bee_hive.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
ylab0<-expression("Temperature"~(degree*C))
par(cex=1.35)
plot(T1,type="n",xlim=c(min(T1[,1]),max(T1[,1])),
        ylim=c(min(T4[-omit,2]),max(T1[,2])),
        main="Temperature in a Frame \n Smoothing = .95",
        xlab="Days",ylab=ylab0)
```

```
gg1<-smooth.spline(T1[-omit,],spar =.95)
f<-splinefun(gg1)
curve(f(x),min(T1[,1]),max(T1[,1]),add=TRUE,col="red")
gg2<-smooth.spline(T2[-omit,],spar =.95)
f<-splinefun(gg2)
curve(f(x),min(T2[,1]),max(T2[,1]),add=TRUE,col="green")
gg3<-smooth.spline(T3[-omit,],spar =.95)
f<-splinefun(gg3)
curve(f(x),min(T3[,1]),max(T3[,1]),add=TRUE,col="blue")
gg4<-smooth.spline(T4[-omit,],spar =.95)
f<-splinefun(gg4)
curve(f(x),min(T4[,1]),max(T4[,1]),add=TRUE,col="cyan")
legend("bottomright",
        legend = c("Sensor 1","Sensor 2","Sensor 3", "Sensor4"),
        text.col = c("red","green","blue","cyan"),
        col=c("red","green","blue","cyan"))
abline(v=8)
text(6,26,"8 Days")
#dev.off()
```

the option, `spar=.95`, indicates the degree to which the smoothing will be made.

Once a program has been sufficiently perfected, it is sometimes expeditious to execute the R program by means of the `source` command:

```
source("/home/mike/gw/sigstat/bls/bls2.fun")
```

In the Microsoft system, the source command would be written as:

```
source("c:\home\mike\gw\sigstat\bls\bls2.fun")
```

and the syntax of referring to directories within `bls2.fun` will have to conform to the Microsoft syntax.

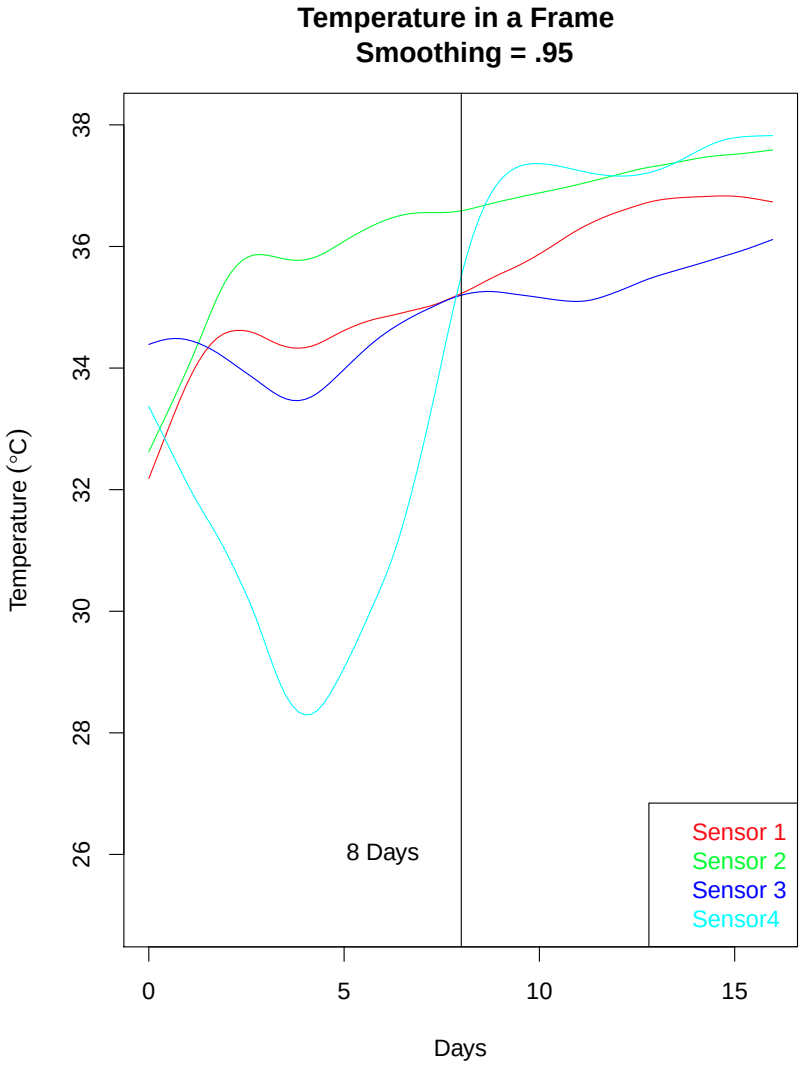

Figure 3.1: The Use of Smoothing in a Plot

Chapter 4

Graphics

4.1 identify

A prominent economist asked for help in his statistical analysis of economic data related to the onset and duration of a recession in conjunction with unemployment rate. He reasoned that the unemployment rate precedes a recession and lags behind it. He summarized his data in terms of correlations which he obtained from the National Bureau of Economic Research where he had worked for many years until he retired in terms of correlations.

There was an intriguing feature of the data which he was not able to explain. It was discovered by looking at a plot of the data. Each point corresponds to a recession which had occurred since WWII.

```
data_jobs<-read.table(file=
        "/home/mike/gw/sigstat/bls/bls_dat3.txt",header=TRUE,sep=" ")
jobs_recess<-data_jobs$jobs_recess
jobs_exp<-data_jobs$jobs_exp
fig0<-paste("/home/mike/gw/sigstat/bls/jobs.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
par(cex=1.35)
plot(jobs\_recess,jobs\_exp,
        main="Jobs Expansion vs Jobs Recession")
```

The `identify` command provided a means for identifying unusual points in the plot of data.

```
identify(jobs_recess,n=10,jobs_exp,labels=1:10,plot=T)
```

In this command, ten points are allocated for identification and a label is place near the point

when it has been identified. These printed labels are then manually entered into the next series of commands for showing their respective points with various symbols and colors as shown in Figure 4.1.

Each of the identified points were plotted again, but with different colors.

```
points(jobs_recess[7],jobs_exp[7],pch=3,col="blue")
points(jobs_recess[c(5,6)],jobs_exp[c(5,6)],pch=5,cex=2)
points(jobs_recess[10],jobs_exp[10],pch=20)
points(jobs_recess[c(3,1,6)],jobs_exp[c(3,1,6)],pch=13,col="green")
points(jobs_recess[2],jobs_exp[2],pch=19,cex=2,col="orange")
```

Lines are drawn to emphasize the two apparently different kinds of data.

```
abline(lsfit(jobs_recess[c(1,4,8,9)],jobs_exp[c(1,4,8,9)]))
abline(lsfit(jobs_recess[c(2,3,7,10)],jobs_exp[c(2,3,7,10)]))
#dev.off()
```

Although Figure 4.2 was not produced by commands in this tutorial, it is presented here to help explain the two parallel lines in Figure 4.1 in that as recessions become longer and more severe, the points in Figure 4.1 might come from two different populations: the short and early ones as opposed to the later and more severe ones.

Figure 4.2 may be found at: http://www.calculatedriskblog.com/2012/12/employment-report-more-positives-than.html

4.2 las=2 and srt=90

In this example, `las=2` is used to place the x labels perpendicular to the x-axis, and `str=90` rotates the names of the presidents to appear vertically.

```
national_debt<-read.table(file=
        "/home/mike/gw/sigstat/bls/national_debt.txt",
        header=TRUE,sep=" ")
x<-national_debt$year
y<-national_debt$debt_ratio

presyr<-c(1940,1945,1953,1961,1963,1969,1974,
        1977,1981,1989,1993,2001,2009)
pres<-c("FDR","Truman","Ike","JFK","LBJ","Nixon",
        "Ford","Carter","Reagan","Bush",
```

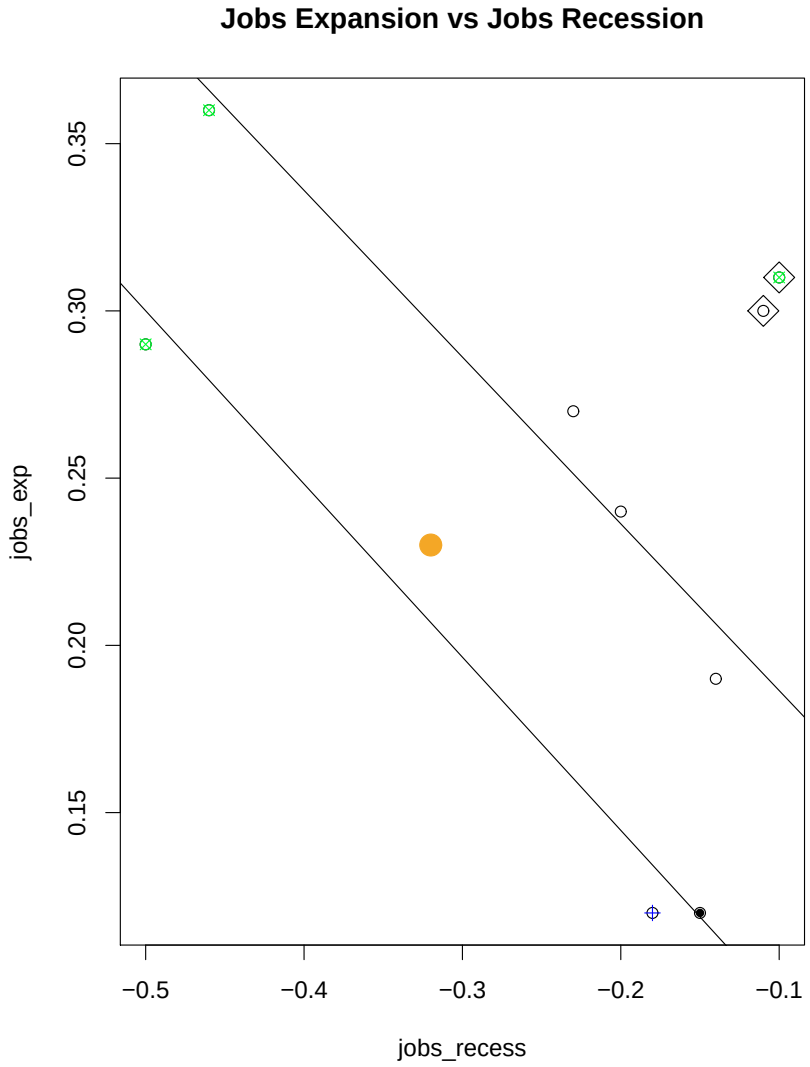

Figure 4.1: Specially Marked Points Using identify

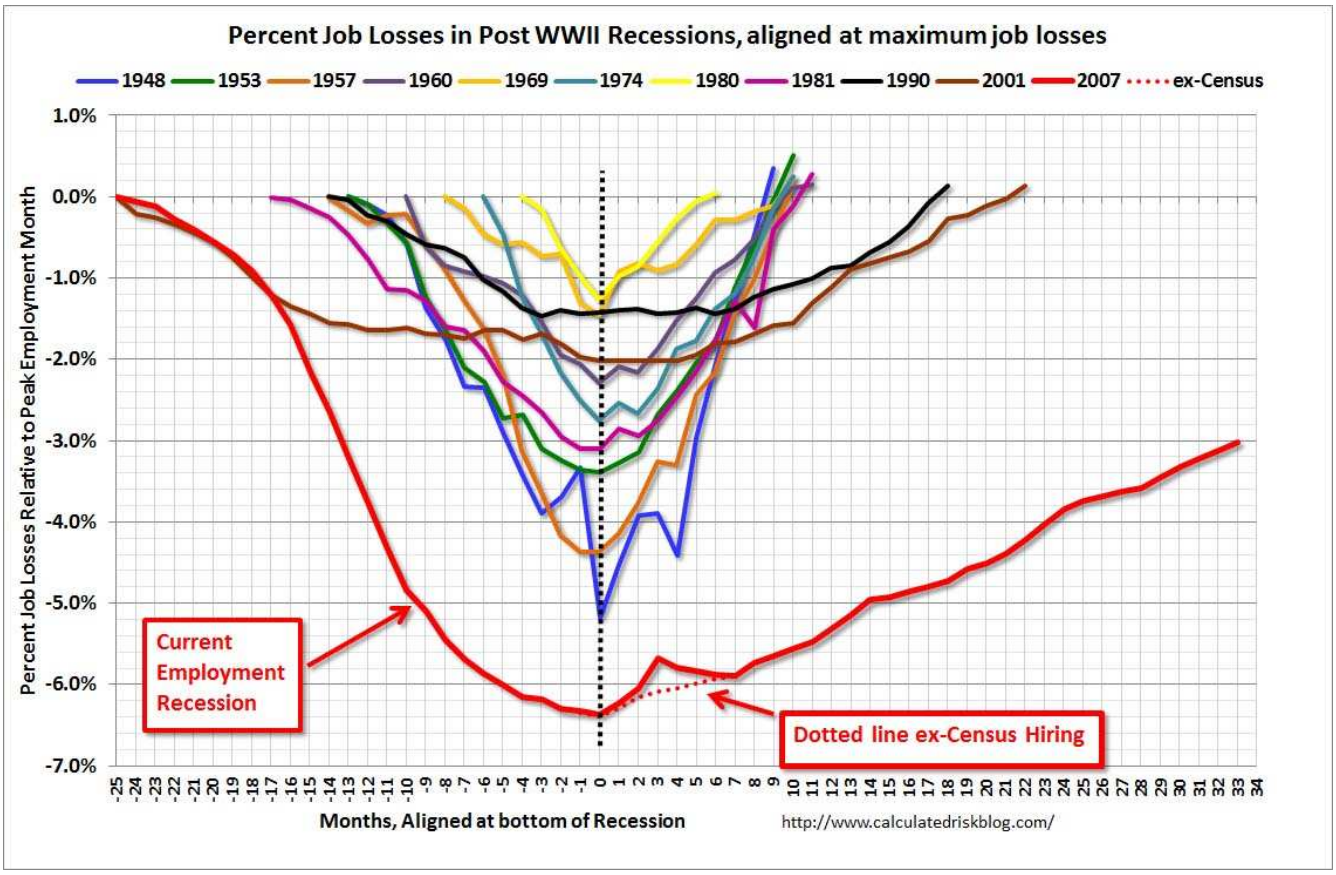

Figure 4.2 Job Losses in Post WWII Recessions

```
        "Clinton","Bush","Obama")

fig0<-paste("/home/mike/gw/sigstat/bls/bls3.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
par(cex=1.5)
plot(x,y,ylim=c(0,130),xaxt="n",type="n",xlab="Year",
        ylab="National Debt in percentage",
        main="National Debt to Gross Domestic \n Product")
axis(1,at=c(presyr), labels=presyr, las=2,cex.axis=.75)
lines(x,y)

for(k in 1:length(presyr)){
lines(c(presyr[k],presyr[k]),c(-5,y[x==presyr[k]]),col="red",lty=3) }

for(k in 1:(length(presyr))){ tt<-paste(pres[k])
text((presyr[k]+presyr[k+1])/2,10,tt,cex=.75,srt=90)
if(k==13){text(2013,10,pres[13],cex=.75,srt=90)} }
#dev.off()
## The end of Figure 4.1

#postscript(horizontal=FALSE,
        file="/home/mike/gw/sigstat/bls/bls4.ps")
par(cex=1.5)
plot(x,y,ylim=c(0,130),xaxt="n",type="n",xlab="Year",
        ylab="National Debt in percentage",
        main="National Debt to Gross Domestic \n Product")
axis(1,at=c(presyr), labels=presyr, las=2,cex.axis=.75)
lines(x,y)
abline(h=y[74])
abline(v=1947)

for(k in 1:length(presyr)){
lines(c(presyr[k],presyr[k]),c(-5,y[x==presyr[k]]),col="red",lty=3) }

for(k in 1:(length(presyr))){ tt<-paste(pres[k])
text((presyr[k]+presyr[k+1])/2,10,tt,cex=.75,srt=90)
if(k==13){text(2013,10,pres[13],cex=.75,srt=90)} }
#dev.off()
## The end of Figure 4.2
```

Whereas the `source` command will facilitate the execution of a set of R commands as was described on page 11, it is easy to execute R code in the `bash` shell on the Linux system. This method of executing a program is especially useful when the program is ready for use in an operational setting. For example, the executable program

```
/home/mike/gw/sigstat/bls/bls3.prg -l "YES"
```

will create the Postscript file, `bls4.ps`, in which the horizontal line marking the height of the national debt as of 2013 is put into the graph.

On the other hand, the following command:

```
/home/mike/gw/sigstat/bls/bls3.prg -l "NO"
```

will produce Postscript file, `bls3.ps`, in which the horizontal line marking the height of the national debt as of 2013 is absent. The UNIX option, `"YES"` or `"NO"`, for specifying the presence of a printed horizontal line is passed from the command line through the UNIX script to the R code.

A copy of `bls3.prg` appears in Chapter 6. The command:

```
/usr/bin/R  --no-save --no-restore --quiet --slave<<+++
```

must be concluded with:

```
+++
```

All commands which appear between `+++` constitute the R program. It can be seen that `bash` shell script variables can be passed into the R program. This property of embedding an R program into a UNIX shell environment significantly extends the versatility of the R program especially in an operational setting. Care should be exercised with respect to referencing UNIX variables such as `\$PRESIDENT` and using the `\$` symbol in the R code as in

```
x<-national\_debt\$year
```

The $ symbol must be preceded by the \ symbol, so that in the program, `bls3.prg`, the object, x, will be defined by

```
x<-national_debt\$year
```

otherwise without the use of \ before the $ symbol, `year` and `debt_ratio` would be interpreted as UNIX variables.

Messages which R generates can be directed via the `sink` command to a file. We see in `bls3.prg` that

```
sink("/home/mike/gw/sigstat/bls/demo.lis",append = FALSE)
```

will direct R messages to the file `demo.lis`. The first `sink` command is paired with another one at the end of program to terminate the process of directing messages to an external file.

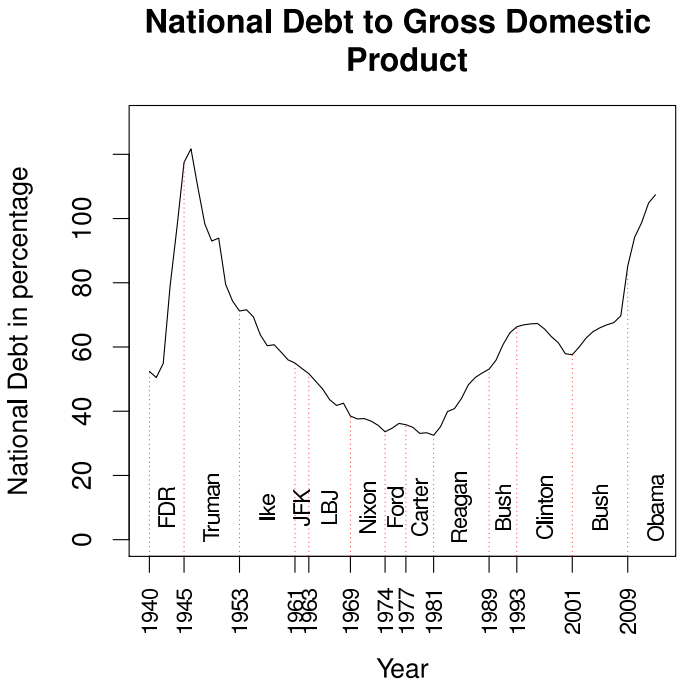

Figure 4.3 The Use of las=2 and str=90

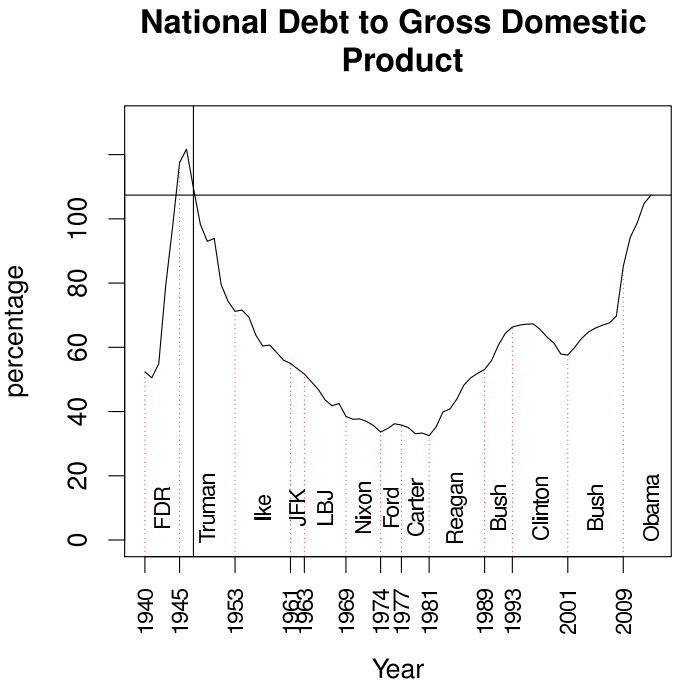

Figure 4.4 The Inclusion of a Level Line

Chapter 5

R Commands for the Examples

```
++++++++++++ page 3

w1<-c(2,3,4,5)
w2<-c(5,4,3,2)

str(w1)

rbind(w1,w2)

++++++++++++ page 4

aa<- "w1,w2"

rbind(aa[1],aa[2]) rbind(aa)

eval(parse(text=paste("rbind(", aa, ")")))

parse(text=paste("rbind(", aa, ")"))

B<-function(a1,a2){ A<-rbind(a1,a2)
print(A)}

B(w1,w2)

++++++++++++ page 5
```

```
C<-function(a1,a2){ A<-rbind(a1,a2)
str<-paste("We concatenate vector ", a1, "with vector ", a2,
        " to get:", sep="")
print(str)
print(A) }

C(w1,w2)

C<-function(a1,a2){
A<-rbind(a1,a2)
first<-deparse(substitute(a1))
second<-deparse(substitute(a2))
str<-paste("We concatenate vector ", first, " with vector ",
        second, " to get:", sep="")
print(str)
print(A)
}


C(w1,w2)


w1<-c(2,3,4,5)    ; x1<-c(12,3,4,5)    ; xyw1<-c(20,30,40,50);
w2<-c(5,4,3,2)    ; x2<-c(5,24,3,2)    ; xyw2<-c(50,40,30,20);
w3<-c(-2,-3,-4,-5); x3<-c(-2,-33,-4,-5); xyw3<-c(-20,-30,-40,-50);
w4<-c(-5,-4,-3,-2); x4<-c(-5,-4,-34,-2); xyw4<-c(-50,-40,-30,-20);
w5<-c(1,1,1,1)    ; x5<-c(11,12,13,15) ; xyw5<-c(10,10,10,10);

++++++++++++ page 6

D<-function(u,n){
uu<-deparse(substitute(u))
m0<-c()
for(k in 1:n){
m0<-rbind(m0,eval(parse(text=paste(uu,k,sep=""))))   }
return(as.matrix(m0))
}

D(x,5)
```

```
z<-t(D(x,5))%*%D(x,5)

det<-function(x)(prod(eigen(x)$values))

det(z)

solve(t(D(x,5))%*%D(x,5))

## Writing a formula in a graph
data<-read.table(file="/home/mike/gw/sigstat/bls/bls_dat1.txt",
        header=TRUE,sep=" ")
bls_x<-data$bls_x
bls_y<-data$bls_y
t200<-data$t200
time200<-data$time200
failures<-data$failures
passes<-data$passes

coefalpha<-1611.313
betahat <- 33.08167
center <- 1.357384

fig0<-paste("/home/mike/gw/sigstat/bls/dwell_time.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
par(cex=1.25)
plot(bls_x,bls_y,type="n",xlim=c(5,7),ylim=c(-3,4),xlab="Velocity",
        ylab="Dwell Time at 200g",main="Relationship between Dwell
        Time with Velocity")

#++++++++++++ page 7

failures<-time200[t200]>=2
passes<-time200[t200]<2
points(bls_x[failures],time200[t200][failures],pch=19,col="red")
points(bls_x[t200][passes],time200[t200][passes],pch=19,col="green")
curve(coefalpha/x^2*log(betahat*x/200)+center,xlim=c(5,7),add=TRUE)
curve(369.4/x^2*log(38.97143*x/200),xlim=c(5,7),add=TRUE,col="red")
curve(369.4/x^2*log(38.97143*x/200)+.17,xlim=c(5,7),add=TRUE,
        col="red")
abline(v=5.8,lty=3)
```

```
abline(v=6.2,lty=3)
abline(h=0,lty=3)
abline(h=2,lty=3)
text(5.6,3,"v0=5.8")
text(6.4,-1,"v0=6.2")
text(6.0,3.5,"Fail",col="red")
text(6.0,-1.5,"Pass",col="green")
eval(parse(text=paste("text(5.4,-1,
        expression(f(x)==frac(alpha,x^2)*log(frac(beta*x,200))))",
        sep="")))
#dev.off()

++++++++++++ page 9

#Same program as bls2.fun
data_temp<-read.table(file=
        "/home/mike/gw/sigstat/bls/bls_dat2.txt",header=TRUE,
        sep=" ")
date_time<-data_temp$date_time
temp1<-data_temp$temp1
temp2<-data_temp$temp2
temp3<-data_temp$temp3
temp4<-data_temp$temp4
d100<-strptime(as.character(date_time), "%m/%d/%y %H:%M","EDT")
age<-julian(as.POSIXlt(d100))-julian(as.POSIXlt("2012-1-1","EDT"))

#++++++++++++ page 10

T1<-cbind(age-age[1],temp1)
T2<-cbind(age-age[1],temp2)
T3<-cbind(age-age[1],temp3)
T4<-cbind(age-age[1],temp4)

omit<-which(is.na(T4[,2]))

fig0<-paste("/home/mike/gw/sigstat/bls/bee_hive.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
ylab0<-expression("Temperature"~(degree*C))
par(cex=1.35)
plot(T1,type="n",xlim=c(min(T1[,1]),max(T1[,1])),
```

```
        ylim=c(min(T4[-omit,2]),max(T1[,2])),
        main="Temperature in a Frame \n Smoothing = .95",
        xlab="Days", ylab=ylab0)

#++++++++++++ page 11

gg1<-smooth.spline(T1[-omit,],spar =.95)
f<-splinefun(gg1)
curve(f(x),min(T1[,1]),max(T1[,1]),add=TRUE,col="red")
gg2<-smooth.spline(T2[-omit,],spar =.95)
f<-splinefun(gg2)
curve(f(x),min(T2[,1]),max(T2[,1]),add=TRUE,col="green")
gg3<-smooth.spline(T3[-omit,],spar =.95)
f<-splinefun(gg3)
curve(f(x),min(T3[,1]),max(T3[,1]),add=TRUE,col="blue")
gg4<-smooth.spline(T4[-omit,],spar =.95)
f<-splinefun(gg4)
curve(f(x),min(T4[,1]),max(T4[,1]),add=TRUE,col="cyan")
legend("bottomright", legend =
        c("Sensor 1","Sensor 2","Sensor 3", "Sensor4"),
        text.col = c("red","green","blue","cyan"),
        col=c("red","green","blue","cyan"))
abline(v=8)
text(6,26,"8 Days")
#dev.off()

## use
source("/home/mike/gw/sigstat/bls/bls2.fun")

++++++++++++ page 13

data_jobs<-read.table(file=
        "/home/mike/gw/sigstat/bls/bls_dat3.txt",header=TRUE,sep=" ")
jobs_recess<-data_jobs$jobs_recess
jobs_exp<-data_jobs$jobs_exp

fig0<-paste("/home/mike/gw/sigstat/bls/jobs.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)

par(cex=1.35)
```

```
plot(jobs_recess,jobs_exp,main="Jobs Expansion vs Jobs Recession")


##Note that 10 points, n=10, are to be identified.
#One could have chosen, n=3.
identify(jobs_recess,n=10,jobs_exp,labels=1:10,plot=T)

#++++++++++++ page 14

points(jobs_recess[7],jobs_exp[7],pch=3,col="blue")
points(jobs_recess[c(5,6)],jobs_exp[c(5,6)],pch=5,cex=2)
points(jobs_recess[10],jobs_exp[10],pch=20)
points(jobs_recess[c(3,1,6)],jobs_exp[c(3,1,6)],pch=13,col="green")
points(jobs_recess[2],jobs_exp[2],pch=19,cex=2,col="orange")

abline(lsfit(jobs_recess[c(1,4,8,9)],jobs_exp[c(1,4,8,9)]))
abline(lsfit(jobs_recess[c(2,3,7,10)],jobs_exp[c(2,3,7,10)]))
#dev.off()



## Use wget http://www.gpo.gov/fdsys/pkg/ERP-2012/xls/
        ERP-2013-table79.xls

national_debt<-read.table(file=
        "/home/mike/gw/sigstat/bls/national_debt.txt",header=TRUE,
        sep=" ")

x<-national_debt$year
y<-national_debt$debt_ratio

presyr<-c(1940,1945,1953,1961,1963,1969,1974,1977,
        1981,1989,1993,2001,2009)
pres<-c("FDR","Truman","Ike","JFK","LBJ","Nixon","Ford",
        "Carter","Reagan","Bush","Clinton","Bush","Obama")

#++++++++++++ page 17

fig0<-paste("/home/mike/gw/sigstat/bls/bls3.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
```

```
par(cex=1.5)
plot(x,y,ylim=c(0,130),xaxt="n",type="n",xlab="Year",
        ylab="National Debt in percentage",
        main="National Debt to Gross Domestic \n Product")
axis(1,at=c(presyr), labels=presyr, las=2,cex.axis=.75)
lines(x,y)

for(k in 1:length(presyr)){
lines(c(presyr[k],presyr[k]),c(-5,y[x==presyr[k]]),
        col="red",lty=3) }

for(k in 1:(length(presyr))){ tt<-paste(pres[k])
text((presyr[k]+presyr[k+1])/2,10,tt,cex=.75,srt=90)
if(k==13){text(2013,10,pres[13],cex=.75,srt=90)} }
#dev.off()

fig0<-paste("/home/mike/gw/sigstat/bls/bls4.ps",sep="")
#postscript(horizontal=FALSE, file=fig0)
par(cex=1.5)
plot(x,y,ylim=c(0,130),xaxt="n",type="n",xlab="Year",
        ylab="National Debt in percentage",
        main="National Debt to Gross Domestic \n Product")
axis(1,at=c(presyr), labels=presyr, las=2,cex.axis=.75)
lines(x,y)
abline(h=y[74])
abline(v=1947)

for(k in 1:length(presyr)){
lines(c(presyr[k],presyr[k]),c(-5,y[x==presyr[k]]),col="red",lty=3) }

for(k in 1:(length(presyr))){ tt<-paste(pres[k])
text((presyr[k]+presyr[k+1])/2,10,tt,cex=.75,srt=90)
if(k==13){text(2013,10,pres[13],cex=.75,srt=90)} }
#dev.off()

#++++++++++++ page 18

## These two programs are executable in the bash shell
/home/mike/gw/sigstat/bls/bls3.prg -l "YES"
```

```
## This tests that the default option is automatically used.
/home/mike/gw/sigstat/bls/bls3.prg
```

Chapter 6

Embedded R Program in a UNIX Script

```
#!/bin/bash
##bls3.prg
##wget http://www.gpo.gov/fdsys/pkg/ERP-2012/xls/ERP-2013-table79.xls

#syntax bls3.prg -l "YES"

standout=` tty `

USAGE () {
         echo Usage is:
         echo '          ' `basename $0`
         echo '                      -l   PRESIDENTS'
         echo '                      -?   this message'
}

while getopts bl: option
do
   case $option in
      b)   SKIPBLANKS=TRUE ;;
      l)   PRESIDENTS=$OPTARG;;
      \?)  USAGE
           exit 2;;
   esac
done
```

```
if [ -z $PRESIDENTS ]; then
PRESIDENTS="NO"
fi
echo "PRESIDENTS = " $PRESIDENTS

echo "bls3.prg"
echo "Begin the R program"

/usr/bin/R  --no-save --no-restore --quiet --slave<<+++
sink("/home/mike/gw/sigstat/bls/bls3.lis",append = FALSE)

national_debt<-read.table(file=
        "/home/mike/gw/sigstat/bls/national_debt.txt",
        header=TRUE,sep=" ")

x<-national_debt\$year
y<-national_debt\$debt_ratio

presyr<-c(1940,1945,1953,1961,1963,1969,1974,1977,
        1981,1989,1993,2001,2009)
pres<-c("FDR","Truman","Ike","JFK","LBJ","Nixon","Ford",
        "Carter","Reagan","Bush","Clinton","Bush","Obama")

fig0<-paste("/home/mike/gw/sigstat/bls/bls3.ps",sep="")
postscript(horizontal=FALSE, file=fig0)
par(cex=1.5)
plot(x,y,ylim=c(0,130),xaxt="n",type="n",xlab="Year",
        ylab="National Debt in percentage",
        main="National Debt to Gross Domestic \n Product")
axis(1,at=c(presyr), labels=presyr, las=2,cex.axis=.75)
lines(x,y)

for(k in 1:length(presyr)){
lines(c(presyr[k],presyr[k]),c(-5,y[x==presyr[k]]),col="red",lty=3)
}

if("${PRESIDENTS}"=="YES"){
for(k in 1:(length(presyr))){
tt<-paste(pres[k])
text((presyr[k]+presyr[k+1])/2,10,tt,cex=.75,srt=90)
```

```
if(k==13){text(2013,10,pres[13],cex=.75,srt=90)}
}
}
dev.off()

postscript(horizontal=FALSE, file=
        "/home/mike/gw/sigstat/bls/bls4.ps")
par(cex=1.5)
plot(x,y,ylim=c(0,130),xaxt="n",type="n",xlab="Year",
        ylab="National Debt in percentage",
        main="National Debt to Gross Domestic \n Product")
axis(1,at=c(presyr), labels=presyr, las=2,cex.axis=.75)
lines(x,y)
abline(h=y[74])
abline(v=1947)

for(k in 1:length(presyr)){
lines(c(presyr[k],presyr[k]),c(-5,y[x==presyr[k]]),col="red",
        lty=3)
}

if("${PRESIDENTS}"=="YES"){
for(k in 1:(length(presyr))){
tt<-paste(pres[k])
text((presyr[k]+presyr[k+1])/2,10,tt,cex=.75,srt=90)
if(k==13){text(2013,10,pres[13],cex=.75,srt=90)}
}
}

dev.off()

sink()

+++
```

Bibliography

John M. Chambers. *Programming with Data A Guide to the S Language*. Springer-Verlag, New York, New York, 1998.

Peter Dalgaard. *Introductory Statistics with R*. Springer-Verlag, New York, New York, 2002.

Charles Fleming. *R Notes*. Unpublished, Washington, D.C., 2004.

Charles Fleming. *Simple Graphs in R*. Unpublished, Washington, D.C., 2005.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-PLUS*. Springer-Verlag, New York, New York, 1999.