Ronald Christensen
Department of Mathematics and Statistics
University of New Mexico

Extremely Preliminary Version of R Commands for – Topics in Experimental Design

Springer

Contents

Preface

0.1 Basic Information

This online book is an R companion to *Topics in Experimental Design*. It presupposes that the reader is already familiar with the basics of R and in particular with the material in Chapters 1 and 3 of my *Preliminary Version of R Commands for Analysis of Variance, Design, and Regression: Linear Modeling of Unbalanced Data* which is available elsewhere at `http://www.stat.unm.edu/~fletcher/Rcode.pdf`. Since the point is to do *Advanced Design*, it is not surprising that the R commands needed are not intrinsic to R but largely relegated to R libraries. These libraries will be discussed as needed in the chapters but two packages that are of quite general use include are `car` (regression algorithms due to Fox, Weisberg, and associates) and `MASS` (algorithms due to Venables, Ripley, and associates). By google-ing "r library(name)" we can access pdf files that both give information on the packages and give details on who was nice enough to provide these tools.

R has available several overviews to subject matters, a number of which are relevant to *Topics in Experimental Design*. These are available at
`https://cran.r-project.org/web/views/`
and will be mentioned as appropriate.

The R overview for experimental design is `https://cran.r-project.org/web/views/ExperimentalDesign.html`.

Information about all available packages is available by going to the R site: `http://www.r-project.org/` Click on "CRAN", click on a site, then under "software", click on "packages". The terms "package" and "library" are used interchangeably (outside of programming R).

0.2 Organization

The chapter and section structure of this guide mimics the chapter and section structure of *Topics in Experimental Design*.

Chapter 1
Fundamentals

Chapter 2
Confounding And Fractional Replication: 2^n Factorial Systems

Note to self: The figures for this are among the figures for the first edition of ANREG. They are part of `R-FIGscripts` in ANREG2

2.1 Confounding

Yates pea data

```
rm(list = ls())
yates <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab2-3.dat",
   sep="",col.names=c("y", "Blk", "n", "p", "k"))
attach(yates)
yates
#summary(yates)
Blocks=factor(Blk)
N=factor(n)
P=factor(p)
K=factor(k)


yt <- lm(y ~ (Blocks+(N+P+K)^2))
ytp=summary(yt)
ytp
yta=anova(yt)
yta

plot(yt$fit,rstandard(yt),xlab="Fitted",
        ylab="Standardized residuals",main="Residual-Fitted plot")

qqnorm(rstandard(yt),ylab="Standardized residuals")
```

```
rankit=qnorm(ppoints(rstandard(yt),a=I(3/8)))
ys=sort(rstandard(yt))
Wprime=(cor(rankit,ys))^2
Wprime
```

2.1.1 Split Plot Analysis

Using `lm` you have to compute the whole plot *F* statistics on your own.

```
rm(list = ls())
yates <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab2-3a.dat",
   sep="",col.names=c("y", "Blk", "n", "p", "k","rep","npk"))
attach(yates)
yates
#summary(yates)
Blocks=factor(Blk)
N=factor(n)
P=factor(p)
K=factor(k)
Rep=factor(rep)
NPK=factor(npk)


yt <- lm(y ~ ((Rep+NPK)^2+(N+P+K)^2))
ytp=summary(yt)
ytp
yta=anova(yt)
yta
```

More conveniently you could use `aov`.

```
yt <- aov(y ~ Rep+NPK + Error(Rep:NPK) + (N+P+K)^2)
ytp=summary(yt)
ytp
```

2.1.2 Partial Confounding

Except for reading in the data, this is the same code as the first script in this section.

```
rm(list = ls())
yates <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab2-6.dat",
   sep="",col.names=c("Blk", "n", "p", "k","y"))
```

```
attach(yates)
yates
#summary(yates)
Blocks=factor(Blk)
N=factor(n)
P=factor(p)
K=factor(k)



yt <- lm(y ~ (Blocks+(N+P+K)^2))
ytp=summary(yt)
ytp
yta=anova(yt)
yta
```

2.2 Fractional replication

Useful packages `library(DoE.base)` `library(FrF2)` (Groemping 2014)

2.3 Analysis of unreplicated experiments

Useful package `library(unrepx)`

```
hare <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab2-10.dat",
   sep="",col.names=c("Batch","a","b","c","d","e","s_c","s_p"))
attach(hare)
hare
#summary(hare)
A=factor(a)
B=factor(b)
C=factor(c)
D=factor(d)
E=factor(e)



cr <- lm(s_p ~ (A+B+C+D+E)^2)
crp=summary(cr)
crp
cra=anova(cr)
cra
```

```
cra[,2]
SS=sort(cra[,2])
Scores=qchisq(Batch[1:15]/(length(SS)+1),1)
plot(Scores,SS)

plot(Scores[1:12],SS[1:12])

SS=sort(cra[,2])
Scores=qchisq(Batch[1:12]/(length(SS)+1-3),1)
plot(Scores,SS[1:12])



cr <- lm(s_p ~ A+B+C+D+E)
crp=summary(cr)
crp
cra=anova(cr)
cra
```

2.4 More on graphical analysis

```
library(unrepx)
```

2.5 Augmenting designs for factors at two levels

```
library(OBsMD)
```

Chapter 3
p^n Factorial Treatment Structures

3.4 Taguchi's Orthogonal Arrays

```
rm(list = ls())
taguchi <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab3-14.dat",
   sep="",col.names=c("Run", "a", "b", "c", "d","e0f0g0","e0f0g1","e0f1g0","e0
attach(taguchi)
taguchi
#summary(taguchi)
A=factor(a)
B=factor(b)
C=factor(c)
D=factor(d)

# Combine outer array data
out=matrix(c(e0f0g0,e0f0g1,e0f1g0,e0f1g1,e1f0g0,e1f0g1,e1f1g0,e1f1g1),9)
out

# Compute summary measures
mout=1:9
sdout=1:9
SNmax=1:9
for(kk in 1:9)
{
mout[kk]=mean(out[kk,])
sdout[kk]=sd(out[kk,])
z=out[kk,]
SNmax[kk]=-log(mean(1/z^2))
}

# Double check summary data
```

```
ybar=(e0f0g0 + e0f0g1 + e0f1g0 + e0f1g1 + e1f0g0 + e1f0g1 + e1f1g0 + e1f1g1)/8
SNmax2=-log((1/e0f0g0^2 + 1/e0f0g1^2 + 1/e0f1g0^2 + 1/e0f1g1^2 + 1/e1f0g0^2 +
+ 1/e1f1g0^2 + 1/e1f1g1^2)/8)
mout
ybar
SNmax
SNmax2

# Compute sums of squares
cr <- lm(ybar ~ A+B+C+D)
crp=summary(cr)
crp
cra=anova(cr)
cra
cra[,2]
SS=cra[1:4,2]
SS=cra[1:4,2]
ID=list("A","B","C","D")
names(SS) = ID
SS=sort(SS)


# Plot SSs
par(mfrow=c(1,2))
Batch=c(1,2,3,4)
Scores=qchisq(Batch/(length(SS)+1),2)
plot(Scores,SS,main="Chi-squared(2) plots",xlim=c(0,max(Scores)+.3))
text(Scores+.25, SS, labels = names(SS), cex=.8)




# Repeat plot another way
aa=a*a
bb=b*b
cc=c*c
dd=d*d
cr <- lm(ybar ~ a+aa+b+bb+c+cc+d+dd)
crp=summary(cr)
crp
cra=anova(cr)
cra
SS=cra[1:8,2]

ID=list("a","aa","b","bb","c","cc","d","dd")
```

```
names(SS) = ID
SS=sort(SS)
# Plot SSs
Batch=c(1,2,3,4,5,6,7,8)
Scores=qchisq(Batch/(length(SS)+1),1)
plot(Scores,SS,main="Chi-squared(1) plots",xlim=c(0,max(Scores)+.3))
text(Scores+.25, SS, labels = names(SS), cex=.8)
par(mfrow=c(1,1))
# Main-effects plots
level=c(0,1,2)
mA=lm(mout~A-1)
mB=lm(mout~B-1)
mC=lm(mout~C-1)
mD=lm(mout~D-1)
par(mfrow=c(2,2))
plot(level,mA$coef,main="A means",ylim=c(18,21),type="b",ylab="Mean")
plot(level,mB$coef,main="B means",ylim=c(18,21),type="b",ylab="Mean")
plot(level,mC$coef,main="C means",ylim=c(18,21),type="b",ylab="Mean")
plot(level,mD$coef,main="D means",ylim=c(18,21),type="b",ylab="Mean")
par(mfrow=c(1,1))
```

Repeat the plots for log standard deviations.

```
lsdout=log(sdout)
# Compute sums of squares
cr <- lm(lsdout ~ A+B+C+D)
crp=summary(cr)
crp
cra=anova(cr)
cra

SS=cra[1:4,2]
ID=list("A","B","C","D")
names(SS) = ID
SS=sort(SS)


# Plot SSs
par(mfrow=c(1,2))
Batch=c(1,2,3,4)
Scores=qchisq(Batch/(length(SS)+1),2)
plot(Scores,SS,main="Chi-squared(2) plots",xlim=c(0,max(Scores)+.3))
text(Scores+.25, SS, labels = names(SS), cex=.8)
```

```
# Repeat plot another way
aa=a*a
bb=b*b
cc=c*c
dd=d*d
cr <- lm(lsdout ~ a+aa+b+bb+c+cc+d+dd)
crp=summary(cr)
crp
cra=anova(cr)
cra
SS=cra[1:8,2]

ID=list("a","aa","b","bb","c","cc","d","dd")
names(SS) = ID
SS=sort(SS)


# Plot SSs
Batch=c(1,2,3,4,5,6,7,8)
Scores=qchisq(Batch/(length(SS)+1),1)
plot(Scores,SS,main="Chi-squared(1) plots",xlim=c(0,max(Scores)+.3))
text(Scores+.25, SS, labels = names(SS), cex=.8)
par(mfrow=c(1,1))


# Main-effects plots
level=c(0,1,2)
mA=lm(lsdout~A-1)
mB=lm(lsdout~B-1)
mC=lm(lsdout~C-1)
mD=lm(lsdout~D-1)
par(mfrow=c(2,2))
plot(level,mA$coef,main="A means",ylim=c(1.10,1.30),type="b",ylab="Mean")
plot(level,mB$coef,main="B means",ylim=c(1.10,1.30),type="b",ylab="Mean")
plot(level,mC$coef,main="C means",ylim=c(1.10,1.30),type="b",ylab="Mean")
plot(level,mD$coef,main="D means",ylim=c(1.10,1.30),type="b",ylab="Mean")
par(mfrow=c(1,1))
```

Repeat plots for signal-to-noise ratio.

```
# Compute sums of squares
cr <- lm(SNmax ~ A+B+C+D)
crp=summary(cr)
crp
cra=anova(cr)
```

```
cra
SS=cra[1:4,2]
ID=list("A","B","C","D")
names(SS) = ID
SS=sort(SS)


# Plot SSs
par(mfrow=c(1,2))
Batch=c(1,2,3,4)
Scores=qchisq(Batch/(length(SS)+1),2)
plot(Scores,SS,main="Chi-squared(2) plots",xlim=c(0,max(Scores)+.3))
text(Scores+.25, SS, labels = names(SS), cex=.8)




# Repeat plot another way
aa=a*a
bb=b*b
cc=c*c
dd=d*d
cr <- lm(SNmax ~ a+aa+b+bb+c+cc+d+dd)
crp=summary(cr)
crp
cra=anova(cr)
cra
SS=cra[1:8,2]

ID=list("a","aa","b","bb","c","cc","d","dd")
names(SS) = ID
SS=sort(SS)


# Plot SSs
Batch=c(1,2,3,4,5,6,7,8)
Scores=qchisq(Batch/(length(SS)+1),1)
plot(Scores,SS,main="Chi-squared(1) plots",xlim=c(0,max(Scores)+.3))
text(Scores+.25, SS, labels = names(SS), cex=.8)
par(mfrow=c(1,1))


# Main-effects plots
level=c(0,1,2)
mA=lm(SNmax~A-1)
```

```
mB=lm(SNmax~B-1)
mC=lm(SNmax~C-1)
mD=lm(SNmax~D-1)
par(mfrow=c(2,2))
plot(level,mA$coef,main="A means",ylim=c(5.68,6.03),type="b",ylab="Mean")
plot(level,mB$coef,main="B means",ylim=c(5.68,6.03),type="b",ylab="Mean")
plot(level,mC$coef,main="C means",ylim=c(5.68,6.03),type="b",ylab="Mean")
plot(level,mD$coef,main="D means",ylim=c(5.68,6.03),type="b",ylab="Mean")
par(mfrow=c(1,1))
```

3.5 Analysis of a Confounded 3^3

```
rm(list = ls())
kemp <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab3-16.dat",
   sep="",col.names=c("Rep", "d", "s", "n", "y", "blk"))
attach(kemp)
kemp
#summary(kemp)
Blk=factor(blk)
N=factor(n)
S=factor(s)
D=factor(d)
cr <- lm(y ~ Blk + D*S*N)
crp=summary(cr)
crp
cra=anova(cr)
cra
```

3.5.1 The Expanded ANOVA Table

So far I haven't been able to convince myself that it is worth my time to figure out how to program the breakdown into 2 df effects. I'm sure my son could do it in a flash but it would take me quite a while.

3.6 5^f Factorials

Chapter 4
Mixed Factor Levels

4.1 Fractionated Partial Confounding

4.2 More Mixtures of Prime Powers

```
taguchi <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab3-23.dat",
   sep="",col.names=c=("Run", "a", "b", "c", "d","e","f","g","y"))
attach(taguchi)
taguchi
```

4.3 Taguchi's L_9 with an Outer Array

```
rm(list = ls())
taguchi <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab4-2.dat",
   sep="",col.names=c("Run", "a", "b", "c", "d","e","f","g","y"))
attach(taguchi)
taguchi
#summary(taguchi)
A=factor(a)
B=factor(b)
C=factor(c)
D=factor(d)
E=factor(e)
F=factor(f)
G=factor(g)

cr <- lm(y ~ E:F:G+ A+B+C+D)
```

```
#crp=summary(cr)
#crp
cra=anova(cr)
cra
```

If you did not have the inner-outer array structure with the $2^3 \times 3^{4-2}$, you would probably want to try to make sense of

```
cr <- lm(y ~ E*F*G*A*B*C*D)
#crp=summary(cr)
#crp
cra=anova(cr)
cra
```

4.4 Powers of Primes

Chapter 5
Screening Designs

5.1 Screening Designs

5.2 Designs for Two Levels

5.3 Designs for Three Levels

5.3.1 Definitive Screening Designs

```
cn <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\dsd13.dat",
   sep="",col.names=c("Run", "a", "b", "c", "d","e"))
attach(cn)
cn
#summary(cn)
A=factor(a)
B=factor(b)
C=factor(c)
D=factor(d)

dsd=lm(Run ~ a + b + c+ d + e + I(a^2) + I(b^2) + I(c^2) + I(d^2) + I(e^2))
anova(dsd)
summary(dsd)
vcov(dsd)

library(BAYESDEF)
```

Chapter 6
Response Surface Methods

The R overview for experimental design https://cran.r-project.org/web/views/ExperimentalDesign.html includes a number of packages for response surface methods. In particular, Lenth (2009) describes the use of his R package `rsm`.

```
install.packages("rsm") #Do this only once on your computer
library(rsm)
```

The code below does not use `rsm`.

6.1 Approximating Response Functions

6.2 First-Order Models and Steepest Ascent

```
rm(list = ls())
rsm.slr <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab5-1.dat",
          sep="",col.names=c("y","x1","x2","x3","x4","x13","x14","Blks","Ctr")
attach(rsm.slr)
rsm.slr
#summary(rsm.slr)



# Table of coefficients and ANOVA table
rs <- lm(y ~ Blks+x1+x2+x3+x4+x13+x14+Ctr)
rsp=summary(rs)
rsp
anova(rs)
plot(rs$fit,rstandard(rs),xlab="Fitted",
        ylab="Standardized residuals",main="Residual-Fitted plot")

qqnorm(rstandard(rs),ylab="Standardized residuals")

rankit=qnorm(ppoints(rstandard(rs),a=I(3/8)))
ys=sort(rstandard(rs))
Wprime=(cor(rankit,ys))^2
Wprime




# Read the data
rm(list = ls())
rsm.slr <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab5-1.dat",
          sep="",col.names=c("y","x1","x2","x3","x4","x13","x14","Blks","Ctr")
attach(rsm.slr)
rsm.slr
#summary(rsm.slr)



rsm.slr <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab5-1.dat",
          sep="",col.names=c("y",,"x1","x2""x3","x4","x13","x14","Blks","Ctr")
attach(rsm.slr)
rsm.slr
#summary(rsm.slr)

# Table of coefficients and ANOVA table
rs <- lm(y ~ Blks+x1+x2+x3+x4+x13+x14+Ctr)
rsp=summary(rs)
rsp
anova(rs)

# Confidence intervals
confint(rs, level=0.95)

# Line estimation and prediction at x=-16.04
new = data.frame(x=c(-16.04))
predict(rs,new,se.fit=T,interval="confidence")
predict(lm(y~x),new,interval="prediction")
# "rs" and "lm(y~x)" are interchangeable here.

rs <- lm(y ~ x)
rsp=summary(rs)
```

```
# The code incorporates the names "rs" and "rsp".

# Create a table of diagnostic statistics
infv = c(y,rs$fit,hatvalues(rs),rstandard(rs),
          rstudent(rs),cooks.distance(rs))
inf=matrix(infv,I(rsp$df[1]+rsp$df[2]),6,dimnames = list(NULL,
                              c("y", "yhat", "lev","r","t","C")))
inf
# Note: The computation assumes NO missing observations

# Normal and two residual plots:
# Run one plot at a time unless you know how to make matrix plots
qqnorm(rstandard(rs),ylab="Standardized residuals")

plot(rs$fit,rstandard(rs),xlab="Fitted",
        ylab="Standardized residuals",main="Residual-Fitted plot")
plot(x,rstandard(rs),xlab="x",ylab="Standardized residuals",
        main="Residual-Socio plot")

#leverage plot
Leverage=hatvalues(rs)
plot(School,Leverage,main="School-Leverage plot")

# Shapiro-Francia Statistic
rankit=qnorm(ppoints(rstandard(rs),a=I(3/8)))
ys=sort(rstandard(rs))
Wprime=(cor(rankit,ys))^2
Wprime
```

Or, for Shapiro-Francia, if you have installed the package `nortest` as in Subsection 2.5.1, use

```
library(nortest)
sf.test(rstandard(rs))
```

6.3 Fitting Quadratic Models

6.3.1 Factors B and C

```
rm(list = ls())
rsm.slr <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab5-9.dat",
          sep="",col.names=c("y","x1","x2","x3","x4"))
attach(rsm.slr)
```

```
rsm.slr
#summary(rsm.slr)


#What happened to Blks and Ctr????
# Table of coefficients and ANOVA table
#rs <- lm(y ~ x2+x3+I(x2^2)+I(x3^2)+I(x2*x3))
rs <- lm(y ~ I(x2+x3)^2)
rsp=summary(rs)
rsp
anova(rs)


# Figs 6.3 and 6.5
plot(rs$fit,rstandard(rs),xlab="Fitted",
        ylab="Standardized residuals",main="Residual-Fitted plot")

qqnorm(rstandard(rs),ylab="Standardized residuals")

rankit=qnorm(ppoints(rstandard(rs),a=I(3/8)))
ys=sort(rstandard(rs))
Wprime=(cor(rankit,ys))^2
Wprime


rs3 <- lm(y ~ I(x2+x3)^3)
rsp=summary(rs3)
rsp3
anova(rs3)
anova(rs,rs3)
```

6.3.2 All Factors

```
rm(list = ls())
rsm.slr <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab5-9.dat",
          sep="",col.names=c("y","x1","x2","x3","x4"))
attach(rsm.slr)
rsm.slr
#summary(rsm.slr)


# Table of coefficients and ANOVA table
```

```
#rs <- lm(y ~ x1+x2+x3+x4+I(x1^2)+I(x2^2)+I(x3^2)+I(x4^2)+I(x1*x2)+I(x1*x3)+I(
rs <- lm(y ~ I(x1+x2+x3+x4)^2)
rsp=summary(rs)
rsp
anova(rs)

#Figs 6.5 and 6.6
plot(rs$fit,rstandard(rs),xlab="Fitted",
        ylab="Standardized residuals",main="Residual-Fitted plot")

qqnorm(rstandard(rs),ylab="Standardized residuals")

rankit=qnorm(ppoints(rstandard(rs),a=I(3/8)))
ys=sort(rstandard(rs))
Wprime=(cor(rankit,ys))^2
Wprime
```

6.4 Interpreting Quadratic Response Functions

Create $\hat{\beta}_*$ and $\hat{B}$ from earlier output. Compute $\hat{x}_0$ and $\hat{\mu}_0$. Do eigen analysis of $\hat{B}$. Check that P is orthonormal.

EXAMPLE 6.4.1. *Two factor example.*

```
rm(list = ls())
rsm.slr <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab5-9.dat",
          sep="",col.names=c("y","x1","x2","x3","x4"))
attach(rsm.slr)
rsm.slr
#summary(rsm.slr)



# Table of coefficients and ANOVA table
#rs <- lm(y ~ x2+x3+I(x2^2)+I(x3^2)+I(x2*x3))
rs <- lm(y ~ I(x2+x3)^2)
rsp=summary(rs)
rsp
anova(rs)
```

□

EXAMPLE 6.4.2. *Four factors with λ_2, λ_3, λ_4 small.*

□

EXAMPLE 6.4.3. *Four factors with λ_4 small.*

```
rm(list = ls())
rsm.slr <- read.table("C:\\E-drive\\Books\\TopicDesign\\Data\\tab5-9.dat",
          sep="",col.names=c("y","x1","x2","x3","x4"))
attach(rsm.slr)
rsm.slr
#summary(rsm.slr)


# Table of coefficients and ANOVA table
#rs <- lm(y ~ x1+x2+x3+x4+I(x1^2)+I(x2^2)+I(x3^2)+I(x4^2)+I(x1*x2)+I(x1*x3)+I(
rs <- lm(y ~ I(x1+x2+x3+x4)^2)
rsp=summary(rs)
rsp
anova(rs)
```

□

6.4.1 B Singular

EXAMPLE 6.4.4. *Two factors with $\lambda_2 = 0$.*

□

EXAMPLE 6.4.4. *Two factors with $\lambda_2 = 0$.*

□

6.5 C+ Program for Generating Response Surface Data

I'm pretty sure that the idea for this was inspired by (ripped off from?) some article that I read. I have not looked at it much since it was written in 1992.

```
/* Programmer: Changhui Dong;
   Superwiser: Prof. Ron Christensen;

   Dept. of Math. & Stat.
   Univ. of New Mexico

   July 25, 1992
*/

/* This program needs to be compiled under the following command:
   "cc -o filename filename.c -lm"
   since it includes the <math.h>.
*/

#include <stdio.h>
#include <math.h>
#include <string.h>

#define void int

/* The basic type definition for data storage and process in memory.
   It is a one-direction data chain. For each data node, the actual
   experimental yield, the corresponding input values of the six
   factors, the number of the block in which the yield is, the total
   number of fixed factors, the flag array for telling factors to be
   fixed or not and the total number of blocks created are included.
*/

typedef struct x {
 double yield;
 double _value[6];
 int block,fixed,tblock,_level[6];
 struct x * link;
 } datatype;


/* Global variable definitions */

char temp[1];     /* Temporary string to accept nonsense words during input */
char filename[9]; /* To keep file names during saving and retriving. */
int block;        /* The number of the block being processed */
int tblock;       /* Total # of blocks */
int bplot;        /* The plot # within a specific block */
int tplot;        /* Total # of plots */
int Seed=1234;    /* Seed for generating the plot-within-block random effect *
int SeedB;        /* Seed for generating the within-block random effect */
int tellB;        /* Denote the block # just processed */
int fixed;        /* The number of fixed factors */
int level[6];     /* Denote factors to be fixed or not */
```

```
double fvalue[6]; /* Array of factor values */
static char *name[]={" N"," P"," K","Ca","Na","Mg"};

/* Head point, temporary point and tail point to the data chain */
datatype *dataset,*mov,*tail;

FILE *fp;         /* File point used in file operations */

/* Message printed at the beginning of executing this file */
void msg()
{
printf("\n\n\n\n");
printf("Prof. R. Christensen & C. Dong;           \n");
printf("Dept. of Math. & Stat, UNM;               \n");
printf("Version 1.0 - 7/25/1992.                  \n\n");
printf("This  simulation program  generates  data \n");
printf("for  a  Response  Surface  Analysis.      \n\n");
printf("Six factors are available :               \n");
        printf("Nitrogen(N), Phosphorus(P), Potassium(K)  \n");
printf("Sodium(Na), Calcium(Ca), Magnesium(Mg).   \n\n");
printf("Notes:                                    \n");
        printf("1.You  may  stop generating data any time \n");
        printf("  by inputting zero value for any factor. \n");
printf("2.The level values for any factor must be \n");
printf("  positive.                               \n");
printf("3.You  may  fix any factor.   \n");
printf("4.Good luck! You'll need it badly.       \n");
}

/* Main menu */
void menu()
{
printf("\n\n\n");
printf("\tCreate   ------ 1 \n");
        printf("\tModify   ------ 2 \n");
printf("\tDisplay  ------ 3 \n");
printf("\tSave     ------ 4 \n");
printf("\tRetrieve ------ 5 \n");
        printf("\tQuit     ------ 0 \n");
}

/* Initialization when the current data memory is empty */
void init()
{
block=0;      /* Set current block # zero */
```

```
tplot=0;      /* Set total # of blocks zero */
  dataset=NULL; /* Set the head point to the data chain NULL */
tail=NULL;    /* Set the tail point to the data chain NULL */
fixed=0;      /* Set the initial # of fixed factors be zero */
}

/* If f=1, get the approx. normal deviate for within-block effect;
   if f=0, get the approx. normal deviate for plot-within-block effect;
   each deviate with zero mean and unit variance;
   all deviates are pseudo-random numbers scaled to lie in (0,1).
*/
double getX(f)
int f;
{
int i=0,j;
double sum=0;
if (f==1)
       {
if (tellB!=block)
{
   SeedB=Seed;
   tellB=block;
}
j=SeedB+5678;
       }
else j=Seed;
while (i<12)
{
 j=(3125*j)%8388608;
 sum += (double)j ;
 i++;
}
if (f==0) Seed=j;
sum /= (double)12;
sum /= (double)8388608;
return (double)sum;
}

/* Compute the within-block effect if given the plot # in a block */
#define TB sqrt(0.1)
double beft(k)
int k;
{
double y;
int k1;
```

```
if ((k1=k)>32) k1=32;
  y=TB*sqrt((double)k1+16/k1)*getX(1);
return y;
}

/* Compute the plot-within-block effect if given the plot # in a block */
#define T sqrt(0.1)
double pwbeft(k)
int k;
{
double y;
int k1;
if ((k1=k)>32) k1=32;
y=T*sqrt((double)k1+16/k1)*getX(0);
return y;
}

/* Compute the exact experimental yield ignoring all variances */
double  yield(N,P,K,Ca,Na,Mg)
double  N,P,K,Ca,Na,Mg;
{
double y;

  y=(double)1/(0.015+0.0005*N+0.001*P+0.001*K
    +1/((N+5)*(P+2))+0.1/(K+2)+0.02/(Mg+1)
      +0.001*(2+K+0.5*Na)/(Ca+1)
    +0.004*(Ca+1)/(2+K+0.5*Na));
return y;
}

/* Compute the actual experimental yield consisting of variances */
double  getdata()
{
double y,t1,t2;
y=yield(fvalue[0],fvalue[1],fvalue[2],fvalue[3],fvalue[4],fvalue[5]);
t1=beft(bplot);
t2=pwbeft(bplot);
printf("Yield (%d) : %1f \n",++tplot,y+t1+t2);
return (double)(y+t1+t2);
}

/* Make the data chain and fill in with information when creating */
void mchain1()
{
int i;
```

```
mov=(datatype *) malloc(sizeof(datatype));
if (tail!=NULL) tail->link=mov;
else  dataset=mov;
tail=mov;
mov->yield=getdata();
for (i=0;i<=5;i++)
{
  mov->_value[i]=fvalue[i];
  mov->_level[i]=level[i];
}
mov->block=block;
mov->fixed=fixed;
mov->tblock=tblock;
mov->link=NULL;
}

/* When there is an input error, print out the error mesg;
   scan to the end of the input line from the keyboard and clear input buffer.
*/
void iperror()
{
printf("Invalid input, Try again ...\007");
scanf("%[^\n]",temp);
}

/* Create data as a data chain in current memory as desired */
void create()
{
int reply,i,count,count0;
int intp;       /* Flag to interrupt during data creation (1-y,0-n) */
double replyr;
init();
intp=0;
tellB=0;
printf("\n\n\nHow many BLOCKS do you want ?\n");
printf("Answer : ");
do
{
        if (!scanf("%d%*[^\n]",&reply)||(reply<1))
iperror();
  else break;
} while (1);
tblock=reply;
for (i=0;i<=5;i++)
{
  printf("\nIs factor %s fixed ? ( 1 - yes, 0 - no ) : ",name[i]);
  do
  {
  if (!scanf("%d%*[^\n]",&reply)||(reply<0)||(reply>1))
  iperror();
  else break;
  } while (1);
  level[i]=reply;
  if (level[i]==1)
  {
  fixed++;
  printf("Enter the fixed value for factor %s : ",name[i]);
  do
   {
   if (!scanf("%lf%*[^\n]",&replyr)||(replyr<=0))
   iperror();
   else break;
   } while (1);
  fvalue[i]=replyr;
  }
}
printf("\n\n[ *** DATA *** ]\n");
count0=tblock;
do
      {
block++;
printf("\n\n( * Block %d * )\n",block);
printf("\nHow many observations in this block : ");
do
{
  if (!scanf("%d%*[^\n]",&reply)||(reply<0))
  iperror();
  else break;
  } while (1);
if (reply==0)
 {
    tblock=--block;
    printf("\nInterrupted !\n");
    break;
 }
bplot=reply;
count=bplot;
while (count--)
{
  printf("\nProcess #%d \n",bplot-count);
  for (i=0;i<=5;i++)
  if (level[i]!=1)
  {
    printf("Enter a value for factor %s : ",name[i]);
    do
    {
      if (!scanf("%lf%*[^\n]",&replyr)||(replyr<0))
      iperror();
      else break;
    } while (1);
    if (replyr==0)
     {
printf("\nInterrupted !\n");
if (bplot-count==1)  tblock=--block;
else tblock=block;
        intp=1;
break;
     }
    fvalue[i]=replyr;
  }
  if (intp==1) break;
  /* computing the yields and storing in the chain */
  mchain1();
}
if (intp==1) break;
      } while (--count0);
printf("\nNumber of blocks created : %d",tblock);
printf("\nNumber of plots  created : %d",tplot);
printf("\nNumber of fixed factors assumed : %d",fixed);
printf("\n\n\n[ *** DONE *** ]\n");
cblockn();
}

/* Change the total block # stored in data chain to be the current one */
void cblockn()
{
datatype *pp;
pp=dataset;
while (pp!=NULL)
{
 pp->tblock=tblock;
 pp=pp->link;
}
}
```

```
/* To locate the node of the the observation in the data chain;
   "obsv" is the number of the observation in order of creation;
   return the corresponding point of the node.
*/
datatype *locate(obsv)
int obsv;
{
datatype *pp;
pp=dataset;
while ((pp!=NULL)&&(--obsv))
pp=pp->link;
return (pp);
}

/* Decrease the higher block # of some observations if deleting */
void arrblock()
{
datatype *pp;
  pp=dataset;
tblock--;
while (pp!=NULL)
{
  (pp->tblock)--;
  if ((pp->block)>block)
(pp->block)--;
  pp=pp->link;
}
}

/* If there is data chain in current memory, modification is allowed */
void modify()
{
int count,reply,resp,i;
int intp;             /* Mark to interrupt upon inserting a block;
                         (1-y,0-n) */
datatype *pp,*tt,*hh; /* If inserting a block, first generate a single chain,
                         then combine it with the main data chain;
 pp - the temporary point of this single chain;
 tt - the tail point of this single chain;
 hh - the head point of this single chain. */
double replyr;

if (tplot!=0)
{
  do
```

```
{
 printf("\n\nOperations:              \n");
 printf("1 --- Insert  a  block       \n");
 printf("2 --- Delete  an observation \n");
 printf("3 --- Delete  a  block       \n");
 printf("4 --- Display                \n");
 printf("0 --- Quit                   \n");
 printf("\nYour choice : ");
 do
 {
  if (!scanf("%d%*[^\n]",&reply)||(reply<0)||(reply>4))
  iperror();
  else break;
 } while (1);
 if (reply==1)
 {
   printf("\nHow many observations in this block : ");
   do
   {
    if (!scanf("%d%*[^\n]",&resp)||(resp<=0))
    iperror();
else break;
   } while (1);
   bplot=count=resp;
   printf("\nInsert this block after which observation ");
   printf("( 0 - beginning ) : ");
   do
   {
    if (!scanf("%d%*[^\n]",&resp)||(resp<0))
    iperror();
    else break;
   } while (1);
   tblock++;
   hh=NULL;
   tt=NULL;
   intp=0;
   tellB=0;
   if ((resp!=0)&&((mov=locate(resp))==NULL))
   {
tblock--;
printf("\nNo such observation. Try again! \n");
   }
   else
   {
      block=tblock;
```

```
       printf("\n\n[ *** DATA --- Block %d *** ]\n",block);
       while (count--)
      {
printf("\nProcess #%d\n",bplot-count);
for (i=0;i<=5;i++)
  if (level[i]!=1)
{
  printf("Enter a value for factor %s : ",name[i]);
  do
  {
   if (!scanf("%lf%*[^\n]",&replyr)||(replyr<0))
   iperror();
   else
       break;
  } while (1);
  if (replyr==0)
  {
   printf("\n\nInterrupted !\n");
   if (bplot-count==1) tblock=--block;
   intp=1;
   break;
  }
  fvalue[i]=replyr;
}
if (intp==1) break;
pp=(datatype *)malloc(sizeof(datatype));
if (tt!=NULL) tt->link=pp;
else          hh=pp;
        tt=pp;
pp->yield=getdata();
     for (i=0;i<=5;i++)
{
  pp->_value[i]=fvalue[i];
  pp->_level[i]=level[i];
}
pp->block=block;
pp->fixed=fixed;
pp->tblock=tblock;
pp->link=NULL;
      }
      if (hh!=NULL)
     {
      if (resp==0)
      {
tt->link=dataset;
dataset=hh;
      }
      else
      {
tt->link=mov->link;
mov->link=hh;
      }
printf("\n[ * End --- Block %d * ]\n",block);
     cblockn();
     }
              printf("\nNumber of observations generated : %d",bplot-count-1);
    }
  }
  if (reply==2)
  {
    if (tplot==1)
    {
      printf("\nThe last observation is deleted. Memory is empty!\n");
      freenode();
      break;
    }
    else
    {
   printf("\nDelete which observation : ");
do
{
 if (!scanf("%d%*[^\n]",&resp)||((resp<1)||(resp>tplot)))
 iperror();
 else break;
} while (1);
if (resp==1)
{
  pp=dataset;
  dataset=pp->link;
  block=pp->block;
  free(pp);
}
else
{
  mov=locate(resp-1);
  pp=mov->link;
  mov->link=pp->link;
  block=pp->block;
  free(pp);
}
```

```
tplot--;
printf("\n[ * OK *]\n");
pp=dataset;
resp=1;
while (pp!=NULL)
{
  if (pp->block==block)
  {
resp=0;
  break;
  }
      pp=pp->link;
}
if (resp)
arrblock();
    }
  }
  if (reply==3)
  {
    if (tblock==1)
    {
      printf("\nThe last block is deleted. Memory is empty!\n");
      freenode();
      break;
    }
    else
    {
      printf("\nDelete which block : ");
      do
      {
if (!scanf("%d%*[^\n]",&resp)||((resp<1)||(resp>tblock)))
iperror();
else break;
      } while (1);
      block=resp;
      mov=pp=dataset;
      while (pp!=NULL)
      {
if (pp->block==block)
{
 tplot--;
 if (pp==dataset)
 {
   dataset=pp->link;
   free(pp);
```

```
     pp=mov=dataset;
   }
   else
   {
     mov->link=pp->link;
     free(pp);
     pp=mov->link;
   }
 }
 else
 {
      mov=pp;
            pp=mov->link;
         }
       }
       printf("\n[ * OK * ]\n");
       arrblock();
     }
   }
   if (reply==4)
      display();
    } while (reply);
   printf("\n\n\n[ *** DONE *** ]\n");
 }
 else
 printf("\n\nNo data in memory. Sorry!\n");

 }

 /* Display all the information contained in the data chain in current memory *
 void display()
 {
 datatype *pp;
 int i,j=0;
 printf("\n\n");
 pp=dataset;
 if (tplot==0)
 printf("No data in memory. Sorry!\n");
 else
 {
   printf("Plots  Yields      N         P         K");
   printf("        Ca         Na        Mg     Blks\n");
   while (pp!=NULL)
   {
    printf("%-4d ",++j);
    printf("%8.5f ",pp->yield);
    for (i=0;i<=5;i++)
    printf("%9.5f ",pp->_value[i]);
    printf("%2d\n",pp->block);
    pp=pp->link;
     }
  printf("\nNumber of blocks : %d",tblock);
  printf("\nNumber of plots  : %d",tplot);
  printf("\nNumber of fixed factors : %d/6",fixed);
          printf("\n\n\n[ *** DONE *** ]\n");
}
}

/* Save data in the data chain in current memory to a designated file;
   the name inputted should be less than or equal to 8 characters;
   input a name of no suffix, this procedure generates two files, i.e.,
   "dongfile" is given, then "dongfile.RSA" and "dongfile.DAT" are
   formed, the former one is used by this simulation program only,
   the later one is used by users.
*/
void savedata()
{
datatype *pp;
int i,j=0;
char ftname[12]; /* String of file name for temporary use */
  printf("\n\n");
pp=dataset;
if (tplot==0)
printf("No data in memory. Sorry!\n");
else
{
  printf("Save data to file : ");
  scanf("%s",filename);
  scanf("%[^\n]",temp);
  filename[8]=0;
  strcpy(ftname,filename);
  if (NULL!=(fp=fopen(strcat(ftname,".RSA"),"w")))
 { while (pp!=NULL)
  {
    fwrite(pp,sizeof(datatype),1,fp);
    pp=pp->link;
  }
   fclose(fp);
   strcpy(ftname,filename);
   fp=fopen(strcat(ftname,".DAT"),"w");
```

```
     pp=dataset;
     fprintf(fp,"Plots  Yields      N         P         K");
     fprintf(fp,"        Ca        Na        Mg     Blks\n");
     while (pp!=NULL)
    {
      fprintf(fp,"%-4d ",++j);
      fprintf(fp,"%8.5f ",pp->yield);
      for (i=0;i<=5;i++)
      {
  fprintf(fp,"%9.5f ",pp->_value[i]);
      }
      fprintf(fp,"%2d\n",pp->block);
      pp=pp->link;
    }
     fprintf(fp,"\nNumber of blocks : %d",tblock);
     fprintf(fp,"\nNumber of plots  : %d",tplot);
     fprintf(fp,"\nNumber of fixed factors : %d/6 ",fixed);
     fclose(fp);
     printf("\n\n\n[ *** DONE *** ]\n");
   }
    else
      printf("\nTrouble opening the file. Try again! \n");
  }

  }

  /* If clearing the current memory is necessary, just free all nodes;
     after that, initialization will be done.
  */
  void freenode()
  {
  datatype *pp;
  pp=dataset;
  while (pp!=NULL)
  {
    dataset=dataset->link;
    free(pp);
    pp=dataset;
  }
  init();
  }

  /* Retrieve the desired data  from a designated file;
     just input a name of no suffix which is less than or equal to 8 characters;
     this program will automatically search for a file with the name + "RSA";
     if the file exists and there is no reading error, create a data chain;
     fill in every node in the data chain with proper information.
  */
  void retrieve()
  {
  int reply,i;
  datatype *pp;
    printf("\n\n");
  if (tplot!=0)
  {
  printf("Warning: Be aware of losing data in current memory ! \n");
  printf("To confirm retrieval ( 1 - yes, 0 - no ) : ");
  do
   {
     if (!scanf("%d%*[^\n]",&reply)||((reply!=0)&&(reply!=1)))
     iperror();
     else break;
   } while (1);
  printf("\n");
  if (reply==0)
   {
     printf("No retrieval this time !\n");
     return;
   }
  freenode();
  }
  do
  {
  printf("Enter the filename to be retrieved : ");
    scanf("%s",filename);
  scanf("%[^\n]",temp);
  filename[8]=0;
  fp=fopen(strcat(filename,".RSA"),"r");
  if (fp==NULL)
          printf("Failure in searching the file ! Try again ... \n");
  } while (fp==NULL);
  if (feof(fp))
  {
  printf("This file is empty. Check about it !\n");
  return;
  }
    do
  {
    pp=(datatype * ) malloc(sizeof(datatype));
    tplot++;
```

```
    if (fread(pp,sizeof(datatype),1,fp))
    {
     if (tail!=NULL)
       tail->link=pp;
     else
       dataset=pp;
     tail=pp;
     if (pp->link==NULL)
     {
  printf("\n\n\n[ *** DONE *** ]\n");
  tblock=pp->tblock;
          block=tblock;
  fixed=pp->fixed;
  for (i=0;i<=5;i++)
         {
  level[i]=pp->_level[i];
          fvalue[i]=pp->_value[i];
         }
  fclose(fp);
    break;
     }
     pp->link=NULL;
    }
    else
    {
     free(pp);
     printf("Failure in reading the file. Check about it ! \n");
     freenode();
     break;
    }
  } while (1);
  }

  /* Main procedure */
  main()
  {
    char *byemesg="\n\n\n\nBye !\n\n\0";

  int choice;
  int no_quit=1;    /* Flag to denote quiting;
                                (1-n,0-y). */
  msg();
  init();
  while (no_quit)
  {
```

```
  menu();
  do
  {
          printf("\nPlease input your choice : ");
    if (!scanf("%d%*[^\n]",&choice))
       scanf("%[^\n]", temp);
  else
      if (choice==1) { create();   break; }
  else
          if (choice==2) { modify();   break; }
  else
  if (choice==3) { display(); break; }
  else
  if (choice==4) { savedata(); break; }
  else
  if (choice==5) { retrieve(); break; }
  else
  if (choice==0) { no_quit=0;  break; }
  printf("Invalid input !\n");
          } while (1);
}
  printf(byemesg);


}
```

Chapter 7
Recovery of Interblock Information