

```

#*- R -*

## Script from Fourth Edition of `Modern Applied Statistics with S'

options(echo = T, width=65, digits=5, height=9999)
postscript(file="ch01.ps", width=8, height=6, pointsize=9)

# Chapter 1 Introduction

# 1.1 A quick overview of S

2 + 3
sqrt(3/4)/(1/3 - 2/pi^2)

library(MASS)
mean(chem)
m <- mean(chem); v <- var(chem)/length(chem)
m/sqrt(v)

std.dev <- function(x) sqrt(var(x))
t.test.p <- function(x, mu=0) {
  n <- length(x)
  t <- sqrt(n) * (mean(x) - mu) / std.dev(x)
  2 * (1 - pt(abs(t), n - 1))
}

t.stat <- function(x, mu = 0) {
  n <- length(x)
  t <- sqrt(n) * (mean(x) - mu) / std.dev(x)
  list(t = t, p = 2 * (1 - pt(abs(t), n - 1)))
}
z <- rnorm(300, 1, 2) # generate 300 N(1, 4) variables.
t.stat(z)
unlist(t.stat(z, 1)) # test mu=1, compact result

# 1.4 An introductory session

x <- rnorm(1000)
y <- rnorm(1000)
truehist(c(x,y+3), nbins=25)

# ?truehist

contour(dd <- kde2d(x,y))
image(dd)

x <- seq(1, 20, 0.5)
x
w <- 1 + x/2
y <- x + w*rnorm(x)
dum <- data.frame(x, y, w)
dum
rm(x, y, w)

fm <- lm(y ~ x, data=dum)
summary(fm)
fml <- lm(y ~ x, data = dum, weight = 1/w^2)
summary(fml)

lrf <- loess(y ~ x, dum)

attach(dum)
plot(x, y)
lines(spline(x, fitted(lrf)), col = 2)

abline(0, 1, lty = 3, col = 3)

```

```

abline(fm, col = 4)
abline(fm1, lty = 4, col = 5)

plot(fitted(fm), resid(fm), xlab = "Fitted Values", ylab = "Residuals")

qqnorm(resid(fm))
qqline(resid(fm))

detach()
rm(fm, fm1, lrf, dum)

hills
# S: splom(~ hills)
pairs(hills)

# S: if(interactive()) brush(hills)

attach(hills)
plot(dist, time)
if(interactive()) identify(dist, time, row.names(hills))
abline(lm(time ~ dist))
# library(lqs)
abline(lqs(time ~ dist), lty=3, col=4)
detach()

if(interactive()){
plot(c(0,1), c(0,1), type="n")
xy <- locator(type = "p")
abline(lm(y ~ x, xy), col = 4)
abline(rlm(y ~ x, xy, method = "MM"), lty = 3, col = 3)
abline(lqs(y ~ x, xy), lty = 2, col = 2)
rm(xy)
}

attach(michelson)
search()
plot(Expt, Speed, main="Speed of Light Data", xlab="Experiment No.")
fm <- aov(Speed ~ Run + Expt)
summary(fm)
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
detach()
rm(fm, fm0)

1 - pf(4.3781, 4, 76)
qf(0.95, 4, 76)

# End of ch01
#*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 2 Data Manipulation

library(MASS)
options(echo = T, width=65, digits=5, height=9999)

-2:2

powers.of.pi <- pi^(-2:2)
powers.of.pi
class(powers.of.pi)

print(powers.of.pi)
summary(powers.of.pi)

```

```

# rm(powers.of.pi)

powers.of.pi[5]

names(powers.of.pi) <- -2:2
powers.of.pi
powers.of.pi["2"]
class(powers.of.pi)

as.vector(powers.of.pi)
names(powers.of.pi) <- NULL
powers.of.pi

citizen <- factor(c("uk", "us", "no", "au", "uk", "us", "us"))
citizen

unclass(citizen)

citizen[5:7]

citizen <- factor(c("uk", "us", "no", "au", "uk", "us", "us"),
                 levels = c("us", "fr", "no", "au", "uk"))
citizen

income <- ordered(c("Mid", "Hi", "Lo", "Mid", "Lo", "Hi", "Lo"))
income

as.numeric(income)

inc <- ordered(c("Mid", "Hi", "Lo", "Mid", "Lo", "Hi", "Lo"),
              levels = c("Lo", "Mid", "Hi"))
inc

erupt <- cut(geyser$duration, breaks = 0:6)
erupt <- ordered(erupt, labels=levels(erupt))
erupt

painters
row.names(painters)

summary(painters) # try it!

attach(painters)
School
detach("painters")

mymat <- matrix(1:30, 3, 10)
mymat

myarr <- mymat
dim(myarr) <- c(3, 5, 2)
class(myarr)
myarr
dim(myarr)

dimnames(myarr) <- list(letters[1:3], NULL, c("(i)", "(ii)"))
myarr

newvar <- NA
class(NA)

newvar > 3

x <- c(pi, 4, 5)
x[2] <- NA
x

```

```

class(x)

is.na(x)

1/0

x <- c(-1, 0, 1)/0
x
is.na(x)
x > Inf

x <- c(2.9, 3.1, 3.4, 3.4, 3.7, 3.7, 2.8, 2.5)

letters[1:3]
letters[c(1:3,3:1)]

longitude <- state.center$x
names(longitude) <- state.name
longitude[c("Hawaii", "Alaska")]

myarr[1, 2:4, ]
myarr[1, 2:4, , drop = F]

attach(painters)
painters[Colour >= 17, ]

painters[Colour >= 15 & Composition > 10, ]
painters[Colour >= 15 & School != "D", ]

painters[is.element(School, c("A", "B", "D")), ]
painters[School %in% c("A", "B", "D"), ] ## R only

painters[cbind(1:nrow(painters), ifelse(Colour > Expression, 3, 4))]

painters[grep("io$", row.names(painters)), ]

detach("painters")

m <- 30
fglsub1 <- fgl[sort(sample(1:nrow(fgl), m)), ]

fglsub2 <- fgl[rbinom(nrow(fgl), 1, 0.1) == 1, ]

fglsub3 <- fgl[seq(1, nrow(fgl), by = 10), ]

painters[sort.list(row.names(painters)), ]

lcrabs <- crabs # make a copy
lcrabs[, 4:8] <- log(crabs[, 4:8])

scrabs <- crabs # make a copy
scrabs[, 4:8] <- lapply(scrabs[, 4:8], scale)
## or to just centre the variables
scrabs[, 4:8] <- lapply(scrabs[, 4:8], scale, scale = F)

scrabs <- crabs # make a copy
scrabs[ ] <- lapply(scrabs,
  function(x) {if(is.numeric(x)) scale(x) else x})

sapply(crabs, is.numeric)

by(crabs[, 4:8], list(crabs$sp, crabs$sex), summary)

```

```

aggregate(crabs[, 4:8], by = list(sp=crabs$sp, sex=crabs$sex),
          median)

authors <- data.frame(
  surname = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
  nationality = c("US", "Australia", "US", "UK", "Australia"),
  deceased = c("yes", rep("no", 4)))
books <- data.frame(
  name = c("Tukey", "Venables", "Tierney",
           "Ripley", "Ripley", "McNeil", "R Core"),
  title = c("Exploratory Data Analysis",
            "Modern Applied Statistics ...",
            "LISP-STAT",
            "Spatial Statistics", "Stochastic Simulation",
            "Interactive Data Analysis",
            "An Introduction to R"))

authors
books

merge(authors, books, by.x = "surname", by.y = "name")

attach(quine)
table(Age)
table(Sex, Age)

tab <- xtabs(~ Sex + Age, quine)
unclass(tab)

tapply(Days, Age, mean)

tapply(Days, Age, mean, trim = 0.1)

tapply(Days, list(Sex, Age), mean)

tapply(Days, list(Sex, Age),
       function(x) sqrt(var(x)/length(x)))

quineFO <- quine[sapply(quine, is.factor)]

#tab <- do.call("table", quineFO)
tab <- table(quineFO)

QuineF <- expand.grid(lapply(quineFO, levels))

QuineF$Freq <- as.vector(tab)
QuineF

# End of ch02
#-*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 3   S Language

library(MASS)
options(echo = T, width=65, digits=5, height=9999)

# from Chapter 2

powers.of.pi <- pi^(-2:2)
names(powers.of.pi) <- -2:2
mymat <- matrix(1:30, 3, 10)
myarr <- mymat
dim(myarr) <- c(3, 5, 2)
dimnames(myarr) <- list(letters[1:3], NULL, c("(i)", "(ii)"))

```

```

# 3.1 Language layout

1 - pi + exp(1.7)

a <- 6

b <- a <- 6

(z <- 1 - pi + exp(1.7))

search()

objects()

objects(2)

find("objects")

get("[<-.data.frame", pos = 2)

# hills <- hills # only needed in S-PLUS
hills$speed <- hills$time/hills$dist

# 3.2 More on S objects

length(letters)

Empl <- list(employee = "Anna", spouse = "Fred", children = 3,
             child.ages = c(4, 7, 9))

Empl$employee
Empl$child.ages[2]

x <- "spouse"; Empl[[x]]

unlist(Empl)
unlist(Empl, use.names = F)

attributes(myarr)
attr(myarr, "dim")

Empl <- c(Empl, service = 8)

c(list(x = 1:3, a = 3:6), list(y = 8:23, b = c(3, 8, 39)))

as(powers.of.pi, "vector")
as(powers.of.pi, "numeric")
is(powers.of.pi, "numeric")
as(powers.of.pi, "character")
is(powers.of.pi, "vector")
as(powers.of.pi, "integer")
is(mymat, "array")

# 3.3 Arithmetical expressions

x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
y <- c(x, x)
v <- 2 * x + y + 1

s3 <- seq(-5, 5, by = 0.2)
s4 <- seq(length = 51, from = -5, by = 0.2)

```

```

s5 <- rep(x, times = 5) # repeat whole vector
s5 <- rep(x, each = 5) # repeat element-by-element

x <- 1:4           # puts c(1,2,3,4)           into x
i <- rep(2, 4)    # puts c(2,2,2,2)         into i
y <- rep(x, 2)    # puts c(1,2,3,4,1,2,3,4)  into y
z <- rep(x, i)    # puts c(1,1,2,2,3,3,4,4)  into z
w <- rep(x, x)    # puts c(1,2,2,3,3,3,4,4,4,4) into w

( colc <- rep(1:3, each = 8) )
( rowc <- rep(rep(1:4, each = 2), 3) )

1 + (ceiling(1:24/8) - 1) %% 3 -> colc; colc
1 + (ceiling(1:24/2) - 1) %% 4 -> rowc; rowc
# or
gl(3, 8)
gl(4, 2, 24)

# 3.4 Character vector operations

paste(c("X", "Y"), 1:4)
paste(c("X", "Y"), 1:4, sep = "")

paste(c("X", "Y"), 1:4, sep = "", collapse = " + ")

substring(state.name[44:50], 1, 4)

as.vector(abbreviate(state.name[44:50]))
as.vector(abbreviate(state.name[44:50], use.classes = F))

grep("na$", state.name)
regexpr("na$", state.name)
state.name[regexpr("na$", state.name)> 0]

# 3.5 Formatting and printing

d <- date()
cat("Today's date is:", substring(d, 1, 10),
    substring(d, 25, 28), "\n")

cat(1, 2, 3, 4, 5, 6, fill = 8, labels = letters)

cat(powers.of.pi, "\n")
format(powers.of.pi)
cat(format(powers.of.pi), "\n", sep=" ")

# 3.6 Calling conventions for functions

args(hist.default)

# 3.8 Control structures

yp <- rpois(50, lambda = 1) # full Poisson sample of size 50
table(yp)
y <- yp[yp > 0]           # truncate the zeros; n = 29

ybar <- mean(y); ybar
lam <- ybar
it <- 0                   # iteration count
del <- 1                  # iterative adjustment

```

```

while (abs(del) > 0.0001 && (it <- it + 1) < 10) {
  del <- (lam - ybar*(1 - exp(-lam)))/(1 - ybar*exp(-lam))
  lam <- lam - del
  cat(it, lam, "\n")}

# 3.9 Array and matrix operations

p <- dbinom(0:4, size = 4, prob = 1/3) # an example
CC <- -(p %o% p)
diag(CC) <- p + diag(CC)
structure(3^8 * CC, dimnames = list(0:4, 0:4)) # convenience

apply(iris3, c(2, 3), mean)
apply(iris3, c(2, 3), mean, trim = 0.1)
apply(iris3, 2, mean)

ir.var <- apply(iris3, 3, var)

ir.var <- array(ir.var, dim = dim(iris3)[c(2, 2, 3)],
               dimnames = dimnames(iris3)[c(2, 2, 3)])

matrix(rep(1/50, 50) %**% matrix(iris3, nrow = 50), nrow = 4,
        dimnames = dimnames(iris3)[-1])

ir.means <- colMeans(iris3)
sweep(iris3, c(2, 3), ir.means)
log(sweep(iris3, c(2, 3), ir.means, "/"))

# 3.10 Introduction to classes and methods

methods(summary)

# End of ch03

#*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 4 Graphical Output

library(MASS)
library(lattice)
trellis.device(postscript, file="ch04.ps", width=8, height=6,
               pointsize=9)
options(echo=T, width=65, digits=5)

# 4.2 Basic plotting functions

lung.deaths <- aggregate(ts.union(mdeaths, fdeaths), 1)
barplot(t(lung.deaths), names = dimnames(lung.deaths)[[1]],
        main = "UK deaths from lung disease")
if(interactive())
  legend(locator(1), c("Males", "Females"), fill = c(2, 3))
loc <- barplot(t(lung.deaths), names = dimnames(lung.deaths)[[1]],
              angle = c(45, 135), density = 10, col = 1)
total <- rowSums(lung.deaths)
text(loc, total + par("cxy")[2], total, cex = 0.7, xpd = T)

# S: if(interactive()) brush(hills)

topo.loess <- loess(z ~ x * y, topo, degree = 2, span = 0.25)
topo.mar <- list(x = seq(0, 6.5, 0.2), y=seq(0, 6.5, 0.2))
topo.lo <- predict(topo.loess, expand.grid(topo.mar))

```

```

par(pty = "s")      # square plot
contour(topo.mar$x, topo.mar$y, topo.lo, xlab = "", ylab = "",
  levels = seq(700,1000,25), cex = 0.7)
points(topo$x, topo$y)
par(pty = "m")
topo.lo1 <- cbind(expand.grid(x=topo.mar$x, y=topo.mar$y),
  z=as.vector(topo.lo))
contourplot(z ~ x * y, topo.lo1, aspect = 1,
  at = seq(700, 1000, 25), xlab = "", ylab = "",
  panel = function(x, y, subscripts, ...) {
    panel.levelplot(x, y, subscripts, ...)
    panel.xyplot(topo$x, topo$y, cex = 0.5)
  }
)

# see help(Skye)
# ternary(Skye/100, ord = c(1, 3, 2))

# 4.3 Enhancing plots

attach(wtloss)
oldpar <- par(no.readonly = TRUE)
# alter margin 4; others are default
par(mar = c(5.1, 4.1, 4.1, 4.1))
plot(Days, Weight, type = "p", ylab = "Weight (kg)")
Wt.lbs <- pretty(range(Weight*2.205))
axis(side = 4, at = Wt.lbs/2.205, lab = Wt.lbs, las = 0)
mtext("Weight (lb)", side = 4, line = 3)
detach()
par(oldpar)

x <- 0:100
plik <- function(lambda)
  sum(dpois(x, lambda) * 2 * ((lambda - x) +
    x * log(pmax(1, x)/lambda)))
lambda <- c(1e-8, 0.05, seq(0.1, 5, 0.1))
plot(lambda, sapply(lambda, plik), type = "l", ylim = c(0, 1.4),
  xlab = expression(lambda),
  ylab = expression(paste(E[lambda], "(deviance)")))
abline(h = 1, lty = 3)

# 4.4 Fine control of graphics

## in R just use swiss
# swiss <- data.frame(Fertility = swiss.fertility, swiss.x)
attach(swiss)
qqnorm(Infant.Mortality)
qqline(Infant.Mortality)

samp <- cbind(Infant.Mortality, matrix(rnorm(47*19), 47, 19))
samp <- apply(scale(samp), 2, sort)
rs <- samp[, 1]
xs <- qqnorm(rs, plot = F)$x
env <- t(apply(samp[, -1], 1, range))

matplot(xs, cbind(rs, env), type = "pnn",
  pch = 4, mkh = 0.06, axes = FALSE, xlab = "", ylab = "")

xyul <- par("usr")
smidge <- min(diff(c(xyul[1], xs, xyul[2]))) / 2
segments(xs - smidge, env[, 1], xs + smidge, env[, 1])
segments(xs - smidge, env[, 2], xs + smidge, env[, 2])

xul <- trunc(10*xyul[1:2]) / 10

```

```

axis(1, at=seq(xul[1], xul[2], by=0.1), labels = FALSE, tck=0.01)
xi <- trunc(xyul[1:2])
axis(1, at = seq(xi[1], xi[2], by = 0.5), tck = 0.02)
yul <- trunc(5*xyul[3:4])/5
axis(2, at=seq(yul[1], yul[2], by=0.2), labels = FALSE, tck=0.01)
yi <- trunc(xyul[3:4])
axis(2, at = yi[1]:yi[2], tck = 0.02)

box(bty = "l")          # lower case "L"
# ps.options()$fonts
# R cannot change font family in a plot.
mtext("Quantiles of Standard Normal", side=1, line=2.5, font=3)
mtext(expression(R[i]), side = 2, line = 2, at = yul[2])
detach()

```

4.5 Trellis graphics

```

xyplot(time ~ dist, data = hills,
  panel = function(x, y, ...) {
    panel.xyplot(x, y, ...)
    panel.lmline(x, y, type = "l")
    panel.abline(lqs(y ~ x), lty = 3)
#    identify(x, y, row.names(hills))
  }
)

## note: don't use separate title() call
bwplot(Expt ~ Speed, data = michelson, ylab = "Experiment No.",
  main = "Speed of Light Data")

```

```

splom(~ swiss, aspect = "fill",
  panel = function(x, y, ...) {
    panel.xyplot(x, y, ...); panel.loess(x, y, ...)
  }
)

```

```

sps <- trellis.par.get("superpose.symbol")
sps$pch <- 1:7
trellis.par.set("superpose.symbol", sps)
xyplot(Time ~ Viscosity, data = stormer, groups = Wt,
  panel = panel.superpose, type = "b",
  key = list(columns = 3,
    text = list(paste(c("Weight:  ", "", ""),
      unique(stormer$Wt), "gms")),
    points = Rows(sps, 1:3)
  )
)
rm(sps)

```

```

topo.plt <- expand.grid(topo.mar)
topo.plt$pred <- as.vector(predict(topo.loess, topo.plt))
levelplot(pred ~ x * y, topo.plt, aspect = 1,
  at = seq(690, 960, 10), xlab = "", ylab = "",
  panel = function(x, y, subtitles, ...) {
    panel.levelplot(x, y, subtitles, ...)
    panel.xyplot(topo$x, topo$y, cex = 0.5, col = 1)
  }
)

```

```

wireframe(pred ~ x * y, topo.plt, aspect = c(1, 0.5),
  drape = TRUE, screen = list(z = -150, x = -60),
  colorkey = list(space="right", height=0.6))

```

```

lcrabs.pc <- predict(princomp(log(crabs[,4:8])))

```



```

        if(which == 303 || which == 341) points(1.4, 1.3)
    })
}

Cath <- equal.count(swiss$Catholic, number = 6, overlap = 0.25)
xyplot(Fertility ~ Education | Cath, data = swiss,
       span = 1, layout = c(6, 1), aspect = 1,
       panel = function(x, y, span) {
         panel.xyplot(x, y); panel.loess(x, y, span)
       }
)

Cath2 <- equal.count(swiss$Catholic, number = 2, overlap = 0)
Agr <- equal.count(swiss$Agric, number = 3, overlap = 0.25)
xyplot(Fertility ~ Education | Agr * Cath2, data = swiss,
       span = 1, aspect = "xy",
       panel = function(x, y, span) {
         panel.xyplot(x, y); panel.loess(x, y, span)
       }
)

Cath
levels(Cath)
plot(Cath, aspect = 0.3)

# End of ch04
#*- R -*

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 5  Univariate Statistics

# for later use, from section 5.6
perm.t.test <- function(d) {
# ttest is function(x) mean(x)/sqrt(var(x)/length(x))
  binary.v <- function(x, digits) {
    if(missing(digits)) {
      mx <- max(x)
      digits <- if(mx > 0) 1 + floor(log(mx, base = 2)) else 1
    }
    ans <- 0:(digits - 1)
    lx <- length(x)
    x <- rep(x, rep(digits, lx))
    x <- (x %/% 2^ans) %% 2
    dim(x) <- c(digits, lx)
    x
  }
  digits <- length(d)
  n <- 2^digits
  x <- d * 2 * (binary.v(1:n, digits) - 0.5)
  mx <- matrix(1/digits, 1, digits) %*% x
  s <- matrix(1/(digits - 1), 1, digits)
  vx <- s %*% (x - matrix(mx, digits, n, byrow=T))^2
  as.vector(mx/sqrt(vx/digits))
}

library(MASS)
options(echo = T,width=65, digits=5, height=9999)
library(lattice)
trellis.device(postscript, file="ch05.ps", width=8, height=6, pointsize=9)

rm(A, B) # precautionary clear-out
attach(shoes)
tperm <- perm.t.test(B - A) # see section 5.6
detach()

```

```

# from ch04
if(!exists("fgl.df")) {
fgl0 <- fgl[ , -10] # omit type.
fgl.df <- data.frame(type = rep(fgl$type, 9),
  y = as.vector(as.matrix(fgl0)),
  meas = factor(rep(1:9, each = 214), labels = names(fgl0)))
invisible()
}

# 5.1 Probability distributions

x <- rt(250, df = 9)
par(pty = "s")
qqnorm(x)
qqline(x)
par(pty = "m")

x <- rgamma(100, shape = 5, rate = 0.1)
fitdistr(x, "gamma")
x2 <- rt(250, df = 9)
fitdistr(x2, "t", df = 9)
fitdistr(x2, "t")

# 5.2 Generating random data

contam <- rnorm( 100, 0, (1 + 2*rbinom(100, 1, 0.05)) )

# 5.3 Data summaries

par(mfrow=c(2,3))
hist(geyser$duration, "scott", xlab="duration")
hist(chem, "scott")
hist(tperm, "scott")
hist(geyser$duration, "FD", xlab="duration")
hist(chem, "FD")
hist(tperm, "FD")
par(mfrow=c(1,1))

swiss.fertility <- swiss[, 1]
stem(swiss.fertility)
stem(chem)
stem(abbey)
stem(abbey, scale = 0.4) ## use scale = 0.4 in R

par(mfrow = c(1,2))
boxplot(chem, sub = "chem", range = 0.5)
boxplot(abbey, sub = "abbey")
par(mfrow = c(1,1))

bwplot(type ~ y | meas, data = fgl.df, scales = list(x="free"),
  strip = function(...) strip.default(..., style=1), xlab = "")

# 5.4 Classical univariate statistics

attach(shoes)
t.test(A, mu = 10)
t.test(A)$conf.int

wilcox.test(A, mu = 10)

```

```

var.test(A, B)

t.test(A, B, var.equal = T)

t.test(A, B, var.equal = F)

wilcox.test(A, B)

t.test(A, B, paired = T)

wilcox.test(A, B, paired = T)
detach()

par(mfrow = c(1, 2))
truehist(tperm, xlab = "diff")
x <- seq(-4,4, 0.1)
lines(x, dt(x,9))
#cdf.compare(tperm, distribution = "t", df = 9)
sres <- c(sort(tperm), 4)
yres <- (0:1024)/1024
plot(sres, yres, type="S", xlab="diff", ylab="")
lines(x, pt(x,9), lty=3)

legend(-5, 1.05, c("Permutation dsn","t_9 cdf"), lty = c(1,3))
par(mfrow = c(1, 1))

```

5.5 Robust summaries

```

# Figure 5.7 was obtained by
x <- seq(-10, 10, len=500)
y <- dt(x, 25, log = TRUE)
z <- -diff(y)/diff(x)
plot(x[-1], z, type = "l", xlab = "", ylab = "psi")
y2 <- dt(x, 5, log = TRUE)
z2 <- -diff(y2)/diff(x)
lines(x[-1], z2, lty = 2)

```

```

sort(chem)
mean(chem)
median(chem)
#location.m(chem)
#location.m(chem, psi.fun="huber")
mad(chem)
#scale.tau(chem)
#scale.tau(chem, center=3.68)
unlist(huber(chem))
unlist(hubers(chem))
fitdistr(chem, "t", list(m = 3, s = 0.5), df = 5)

```

```

sort(abbey)
mean(abbey)
median(abbey)
#location.m(abbey)
#location.m(abbey, psi.fun="huber")
unlist(hubers(abbey))
unlist(hubers(abbey, k = 2))
unlist(hubers(abbey, k = 1))
fitdistr(abbey, "t", list(m = 12, s = 5), df = 10)

```

5.6 Density estimation

```

# Figure 5.8
attach(geyser)

```

```

par(mfrow=c(2,3))
truehist(duration, h=0.5, x0=0.0, xlim=c(0, 6), ymax=0.7)
truehist(duration, h=0.5, x0=0.1, xlim=c(0, 6), ymax=0.7)
truehist(duration, h=0.5, x0=0.2, xlim=c(0, 6), ymax=0.7)
truehist(duration, h=0.5, x0=0.3, xlim=c(0, 6), ymax=0.7)
truehist(duration, h=0.5, x0=0.4, xlim=c(0, 6), ymax=0.7)

breaks <- seq(0, 5.9, 0.1)
counts <- numeric(length(breaks))
for(i in (0:4)) counts[i+(1:55)] <- counts[i+(1:55)] +
  rep(hist(duration, breaks=0.1*i + seq(0, 5.5, 0.5),
    prob=TRUE, plot=FALSE)$intensities, rep(5,11))
plot(breaks+0.05, counts/5, type="l", xlab="duration",
  ylab="averaged", bty="n", xlim=c(0, 6), ylim=c(0, 0.7))
detach()

attach(geyser)
truehist(duration, nbins = 15, xlim = c(0.5, 6), ymax = 1.2)
lines(density(duration, width = "nrd"))

truehist(duration, nbins = 15, xlim = c(0.5, 6), ymax = 1.2)
lines(density(duration, width = "SJ", n = 256), lty = 3)
lines(density(duration, n = 256, width = "SJ-dpi"), lty = 1)
detach()

gal <- galaxies/1000
plot(x = c(0, 40), y = c(0, 0.3), type = "n", bty = "l",
  xlab = "velocity of galaxy (1000km/s)", ylab = "density")
rug(gal)
lines(density(gal, width = "SJ-dpi", n = 256), lty = 1)
lines(density(gal, width = "SJ", n = 256), lty = 3)
library(polyspline)
x <- seq(5, 40, length = 500)
lines(x, dologspline(x, oldlogspline(gal)), lty = 2)

geyser2 <- data.frame(as.data.frame(geyser)[-1, ],
  pduration = geyser$duration[-299])
attach(geyser2)
par(mfrow = c(2, 2))
plot(pduration, waiting, xlim = c(0.5, 6), ylim = c(40, 110),
  xlab = "previous duration", ylab = "waiting")
f1 <- kde2d(pduration, waiting, n = 50, lims=c(0.5, 6, 40, 110))
image(f1, zlim = c(0, 0.075),
  xlab = "previous duration", ylab = "waiting")
f2 <- kde2d(pduration, waiting, n = 50, lims=c(0.5, 6, 40, 110),
  h = c(width.SJ(duration), width.SJ(waiting)) )
image(f2, zlim = c(0, 0.075),
  xlab = "previous duration", ylab = "waiting")
persp(f2, phi = 30, theta = 20, d = 5,
  xlab = "previous duration", ylab = "waiting", zlab = "")
detach()

density(gal, n = 1, from = 20.833, to = 20.834, width = "SJ")$y
1/(2 * sqrt(length(gal))) * 0.13)

set.seed(101)
m <- 1000
res <- numeric(m)
for (i in 1:m) res[i] <- median(sample(gal, replace = T))
mean(res - median(gal))
sqrt(var(res))

truehist(res, h = 0.1)
lines(density(res, width = "SJ-dpi", n = 256))

```

```

quantile(res, p = c(0.025, 0.975))
x <- seq(19.5, 22.5, length = 500)
lines(x, doldlogspline(x, oldlogspline(res)), lty = 3)

library(boot)
set.seed(101)
gal.boot <- boot(gal, function(x, i) median(x[i]), R = 1000)
gal.boot

boot.ci(gal.boot, conf = c(0.90, 0.95),
        type = c("norm", "basic", "perc", "bca"))
plot(gal.boot)

if(F){ # bootstrap() is an S-PLUS function
gal.bt <- bootstrap(gal, median, seed = 101, B = 1000)
summary(gal.bt)
plot(gal.bt)
qqnorm(gal.bt)

limits.emp(gal.bt)
limits.bca(gal.bt)
}

sim.gen <- function(data, mle) {
  n <- length(data)
  data[sample(n, replace = T)] + mle*rnorm(n)
}
gal.boot2 <- boot(gal, median, R = 1000,
  sim = "parametric", ran.gen = sim.gen, mle = 0.5)
boot.ci(gal.boot2, conf = c(0.90, 0.95),
        type = c("norm", "basic", "perc"))

attach(shoes)
t.test(B - A)
shoes.boot <- boot(B - A, function(x,i) mean(x[i]), R = 1000)
boot.ci(shoes.boot, type = c("norm", "basic", "perc", "bca"))
mean.fun <- function(d, i) {
  n <- length(i)
  c(mean(d[i]), (n-1)*var(d[i])/n^2)
}
shoes.boot2 <- boot(B - A, mean.fun, R = 1000)
boot.ci(shoes.boot2, type = "stud")
detach()

# End of ch05
#-*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 6 Linear Statistical Models

library(MASS)
library(lattice)
options(echo = T,width=65, digits=5, height=9999)
trellis.device(postscript, file="ch06.ps", width=8, height=6, pointsize=9)
options(contrasts = c("contr.helmert", "contr.poly"))

# 6.1 A linear regression example

xyplot(Gas ~ Temp | Insul, whiteside, panel =
  function(x, y, ...) {
    panel.xyplot(x, y, ...)
    panel.lmline(x, y, ...)
  }, xlab = "Average external temperature (deg. C)",
  ylab = "Gas consumption (1000 cubic feet)", aspect = "xy",

```

```

strip = function(...) strip.default(..., style = 1))

gasB <- lm(Gas ~ Temp, data = whiteside, subset = Insul=="Before")
gasA <- update(gasB, subset = Insul=="After")

summary(gasB)
summary(gasA)

varB <- deviance(gasB)/gasB$df.resid      # direct calculation
varB <- summary(gasB)$sigma^2            # alternative

gasBA <- lm(Gas ~ Insul/Temp - 1, data = whiteside)
summary(gasBA)

gasQ <- lm(Gas ~ Insul/(Temp + I(Temp^2)) - 1, data = whiteside)
summary(gasQ)$coef

# R: options(contrasts = c("contr.helmert", "contr.poly"))
gasPR <- lm(Gas ~ Insul + Temp, data = whiteside)
anova(gasPR, gasBA)

oldcon <- options(contrasts = c("contr.treatment", "contr.poly"))
gasBA1 <- lm(Gas ~ Insul*Temp, data = whiteside)
summary(gasBA1)$coef
options(oldcon)

# 6.2 Model formulae and model matrices

dat <- data.frame(a = factor(rep(1:3, 3)),
                  y = rnorm(9, rep(2:4, 3), 0.1))
obj <- lm(y ~ a, dat)
(alf.star <- coef(obj))
Ca <- contrasts(dat$a)      # contrast matrix for `a`
drop(Ca %*% alf.star[-1])
dummy.coef(obj)

N <- factor(Nlevs <- c(0,1,2,4))
contrasts(N)
contrasts(ordered(N))

N2 <- N
contrasts(N2, 2) <- poly(Nlevs, 2)
N2 <- C(N, poly(Nlevs, 2), 2)      # alternative
contrasts(N2)

fractions(ginv(contr.helmert(n = 4)))

Cp <- diag(-1, 4, 5); Cp[row(Cp) == col(Cp) - 1] <- 1
Cp
fractions(ginv(Cp))

# 6.3 Regression diagnostics

(hills.lm <- lm(time ~ dist + climb, data = hills))
frame()
par(fig = c(0, 0.6, 0, 0.55))
plot(fitted(hills.lm), studres(hills.lm))
abline(h = 0, lty = 2)
# identify(fitted(hills.lm), studres(hills.lm), row.names(hills))
par(fig = c(0.6, 1, 0, 0.55), pty = "s")
qqnorm(studres(hills.lm))
qqline(studres(hills.lm))

```

```

par(pty = "m")
hills.hat <- lm.influence(hills.lm)$hat
cbind(hills, lev = hills.hat)[hills.hat > 3/35, ]
cbind(hills, pred = predict(hills.lm))["Knock Hill", ]
(hills1.lm <- update(hills.lm, subset = -18))
update(hills.lm, subset = -c(7, 18))
summary(hills1.lm)
summary(update(hills1.lm, weights = 1/dist^2))
lm(time ~ -1 + dist + climb, hills[-18, ], weight = 1/dist^2)

# hills <- hills # make a local copy (needed in S-PLUS)
hills$speed <- hills$time/hills$dist
hills$grad <- hills$climb/hills$dist
(hills2.lm <- lm(ispeed ~ grad, data = hills[-18, ]))
frame()
par(fig = c(0, 0.6, 0, 0.55))
plot(hills$grad[-18], studres(hills2.lm), xlab = "grad")
abline(h = 0, lty = 2)
# identify(hills$grad[-18], studres(hills2.lm), row.names(hills)[-18])
par(fig = c(0.6, 1, 0, 0.55), pty = "s")
qqnorm(studres(hills2.lm))
qqline(studres(hills2.lm))
par(pty = "m")
hills2.hat <- lm.influence(hills2.lm)$hat
cbind(hills[-18,], lev = hills2.hat)[hills2.hat > 1.8*2/34, ]

```

6.4 Safe prediction

```

quad1 <- lm(Weight ~ Days + I(Days^2), data = wtloss)
quad2 <- lm(Weight ~ poly(Days, 2), data = wtloss)

new.x <- data.frame(Days = seq(250, 300, 10),
                    row.names = seq(250, 300, 10))

predict(quad1, newdata = new.x)
predict(quad2, newdata = new.x)

# predict.gam(quad2, newdata = new.x) # S-PLUS only

```

6.5 Robust and resistant regression

```

# library(lqs)
phones.lm <- lm(calls ~ year, data = phones)
attach(phones); plot(year, calls); detach()
abline(phones.lm$coef)
abline(rlm(calls ~ year, phones, maxit=50), lty = 2, col = 2)
abline(lqs(calls ~ year, phones), lty = 3, col = 3)
# legend(locator(1), lty = 1:3, col = 1:3,
#        legend = c("least squares", "M-estimate", "LTS"))

summary(lm(calls ~ year, data = phones), cor = F)
summary(rlm(calls ~ year, maxit = 50, data = phones), cor = F)
summary(rlm(calls ~ year, scale.est = "proposal 2",
            data = phones), cor = F)
summary(rlm(calls ~ year, data = phones, psi = psi.bisquare),
        cor = F)

lqs(calls ~ year, data = phones)
lqs(calls ~ year, data = phones, method = "lms")
lqs(calls ~ year, data = phones, method = "S")

summary(rlm(calls ~ year, data = phones, method = "MM"), cor = F)

# library(robust) # S-PLUS only

```

```

# phones.lmr <- lmRob(calls ~ year, data = phones)
# summary(phones.lmr)
# plot(phones.lmr)

hills.lm
hills1.lm # omitting Knock Hill
rlm(time ~ dist + climb, data = hills)
summary(rlm(time ~ dist + climb, data = hills,
             weights = 1/dist^2, method = "MM"), cor = F)
lqs(time ~ dist + climb, data = hills, nsamp = "exact")
summary(hills2.lm) # omitting Knock Hill
summary(rlm(ispeed ~ grad, data = hills), cor = F)
summary(rlm(ispeed ~ grad, data = hills, method="MM"), cor=F)
# summary(lmRob(ispeed ~ grad, data = hills))

lqs(ispeed ~ grad, data = hills)

# 6.6 Bootstrapping linear models

library(boot)
fit <- lm(calls ~ year, data = phones)
ph <- data.frame(phones, res = resid(fit), fitted = fitted(fit))
ph.fun <- function(data, i) {
  d <- data
  d$calls <- d$fitted + d$res[i]
  coef(update(fit, data=d))
}
(ph.lm.boot <- boot(ph, ph.fun, R = 999))

fit <- rlm(calls ~ year, method = "MM", data = phones)
ph <- data.frame(phones, res = resid(fit), fitted = fitted(fit))
(ph.rlm.boot <- boot(ph, ph.fun, R = 999))

# 6.7 Factorial designs and designed experiments

options(contrasts=c("contr.helmert", "contr.poly"))
(npk.aov <- aov(yield ~ block + N*P*K, data = npk))
summary(npk.aov)

alias(npk.aov)
coef(npk.aov)

options(contrasts=c("contr.treatment", "contr.poly"))
npk.aov1 <- aov(yield ~ block + N + K, data = npk)
summary.lm(npk.aov1)
se.contrast(npk.aov1, list(N == "0", N == "1"), data = npk)
model.tables(npk.aov1, type = "means", se = T)

mp <- c("-", "+")
(NPK <- expand.grid(N = mp, P = mp, K = mp))

if(F) {
blocks13 <- fac.design(levels = c(2, 2, 2),
                      factor= list(N=mp, P=mp, K=mp), rep = 3, fraction = 1/2)

blocks46 <- fac.design(levels = c(2, 2, 2),
                      factor = list(N=mp, P=mp, K=mp), rep = 3, fraction = ~ -N:P:K)

NPK <- design(block = factor(rep(1:6, each = 4)),
             rbind(blocks13, blocks46))
i <- order(runif(6)[NPK$block], runif(24))
NPK <- NPK[i,] # Randomized

lev <- rep(2, 7)

```

```
factors <- list(S=mp, D=mp, H=mp, G=mp, R=mp, B=mp, P=mp)
(Bike <- fac.design(lev, factors,
  fraction = ~ S:D:G + S:H:R + D:H:B + S:D:H:P))
replications(~ .^2, data=Bike)
}
```

6.8 An unbalanced four-way layout

```
attach(quine)
table(Lrn, Age, Sex, Eth)

Means <- tapply(Days, list(Eth, Sex, Age, Lrn), mean)
Vars <- tapply(Days, list(Eth, Sex, Age, Lrn), var)
SD <- sqrt(Vars)
par(mfrow = c(1, 2), pty="s")
plot(Means, Vars, xlab = "Cell Means", ylab = "Cell Variances")
plot(Means, SD, xlab = "Cell Means", ylab = "Cell Std Devn.")
detach()

boxcox(Days+1 ~ Eth*Sex*Age*Lrn, data = quine, singular.ok = T,
  lambda = seq(-0.05, 0.45, len = 20))

logtrans(Days ~ Age*Sex*Eth*Lrn, data = quine,
  alpha = seq(0.75, 6.5, len = 20), singular.ok = T)

quine.hi <- aov(log(Days + 2.5) ~ .^4, quine)
quine.nxt <- update(quine.hi, . ~ . - Eth:Sex:Age:Lrn)
dropterm(quine.nxt, test = "F")

quine.lo <- aov(log(Days+2.5) ~ 1, quine)
addterm(quine.lo, quine.hi, test = "F")

quine.stp <- stepAIC(quine.nxt,
  scope = list(upper = ~Eth*Sex*Age*Lrn, lower = ~1),
  trace = F)
quine.stp$anova

dropterm(quine.stp, test = "F")
quine.3 <- update(quine.stp, . ~ . - Eth:Age:Lrn)
dropterm(quine.3, test = "F")
quine.4 <- update(quine.3, . ~ . - Eth:Age)
quine.5 <- update(quine.4, . ~ . - Age:Lrn)
dropterm(quine.5, test = "F")
```

6.9 Predicting computer performance

```
par(mfrow = c(1, 2), pty = "s")
boxcox(perf ~ syct + mmin + mmax + cach + chmin + chmax,
  data = cpus, lambda = seq(0, 1, 0.1))

cpus1 <- cpus
attach(cpus)
for(v in names(cpus)[2:7])
  cpus1[[v]] <- cut(cpus[[v]], unique(quantile(cpus[[v]])),
    include.lowest = T)
detach()
boxcox(perf ~ syct + mmin + mmax + cach + chmin + chmax,
  data = cpus1, lambda = seq(-0.25, 1, 0.1))
par(mfrow = c(1, 1), pty = "m")

set.seed(123)
cpus2 <- cpus[, 2:8] # excludes names, authors' predictions
cpus2[, 1:3] <- log10(cpus2[, 1:3])
#cpus.samp <- sample(1:209, 100)
```

```

cpus.samp <-
c(3, 5, 6, 7, 8, 10, 11, 16, 20, 21, 22, 23, 24, 25, 29, 33, 39, 41, 44, 45,
46, 49, 57, 58, 62, 63, 65, 66, 68, 69, 73, 74, 75, 76, 78, 83, 86,
88, 98, 99, 100, 103, 107, 110, 112, 113, 115, 118, 119, 120, 122,
124, 125, 126, 127, 132, 136, 141, 144, 146, 147, 148, 149, 150, 151,
152, 154, 156, 157, 158, 159, 160, 161, 163, 166, 167, 169, 170, 173,
174, 175, 176, 177, 183, 184, 187, 188, 189, 194, 195, 196, 197, 198,
199, 202, 204, 205, 206, 208, 209)

cpus.lm <- lm(log10(perf) ~ ., data = cpus2[cpus.samp, ])
test.cpus <- function(fit)
  sqrt(sum((log10(cpus2[-cpus.samp, "perf"]) -
    predict(fit, cpus2[-cpus.samp,]))^2)/109)
test.cpus(cpus.lm)
cpus.lm2 <- stepAIC(cpus.lm, trace=F)
cpus.lm2$anova
test.cpus(cpus.lm2)

# 6.10 Multiple comparisons

immer.aov <- aov((Y1 + Y2)/2 ~ Var + Loc, data = immer)
summary(immer.aov)

model.tables(immer.aov, type = "means", se = T, cterms = "Var")

if(F) {
multicomp(immer.aov, plot = T)

oats1 <- aov(Y ~ N + V + B, data = oats)
summary(oats1)
multicomp(oats1, focus = "V")
multicomp(oats1, focus = "N", comparisons = "mcc", control = 1)
lmat <- matrix(c(0,-1,1,rep(0, 11), 0,0,-1,1, rep(0,10),
  0,0,0,-1,1,rep(0,9)),,3,
  dimnames = list(NULL,
    c("0.2cwt-0.0cwt", "0.4cwt-0.2cwt", "0.6cwt-0.4cwt")))
multicomp(oats1, lmat = lmat, bounds = "lower", comparisons = "none")
}

(tk <- TukeyHSD(immer.aov, which = "Var"))
plot(tk)

oats1 <- aov(Y ~ N + V + B, data = oats)
(tk <- TukeyHSD(oats1, which = "V"))
plot(tk)

## An alternative under R is to use package multcomp (which requires mvtnorm)
## This code is for multcomp >= 0.991-1
library(multcomp)
## next is slow:
(tk <- confint(glht(immer.aov, linfct = mcp(Var = "Tukey"))))
plot(tk)

confint(glht(oats1, linfct = mcp(V = "Tukey"))))
lmat <- matrix(c(0,-1,1,rep(0, 11), 0,0,-1,1, rep(0,10),
  0,0,0,-1,1,rep(0,9)),,3,
  dimnames = list(NULL,
    c("0.2cwt-0.0cwt", "0.4cwt-0.2cwt", "0.6cwt-0.4cwt")))
confint(glht(oats1, linfct = mcp(N = t(lmat[2:5, ])), alternative = "greater"))
plot(tk)

# End of ch06
#*- R -*

## Script from Fourth Edition of `Modern Applied Statistics with S'

```

Chapter 7 Generalized Linear Models

```
library(MASS)
options(echo = T,width=65, digits=5, height=9999)
postscript(file="ch07.ps", width=8, height=6, pointsize=9)
options(contrasts = c("contr.treatment", "contr.poly"))

ax.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
            family = gaussian, data = anorexia)
summary(ax.1)

# 7.2 Binomial data

options(contrasts = c("contr.treatment", "contr.poly"))
ldose <- rep(0:5, 2)
numdead <- c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16)
sex <- factor(rep(c("M", "F"), c(6, 6)))
SF <- cbind(numdead, numalive = 20 - numdead)
budworm.lg <- glm(SF ~ sex*ldose, family = binomial)
summary(budworm.lg, cor = F)

plot(c(1,32), c(0,1), type = "n", xlab = "dose",
      ylab = "prob", log = "x")
text(2^ldose, numdead/20, labels = as.character(sex))
ld <- seq(0, 5, 0.1)
lines(2^ld, predict(budworm.lg, data.frame(ldose = ld,
      sex = factor(rep("M", length(ld)), levels = levels(sex))),
      type = "response"), col = 3)
lines(2^ld, predict(budworm.lg, data.frame(ldose = ld,
      sex = factor(rep("F", length(ld)), levels = levels(sex))),
      type = "response"), lty = 2, col = 2)

budworm.lgA <- update(budworm.lg, . ~ sex * I(ldose - 3))
summary(budworm.lgA, cor = F)$coefficients
anova(update(budworm.lg, . ~ . + sex * I(ldose^2)),
      test = "Chisq")

budworm.lg0 <- glm(SF ~ sex + ldose - 1, family = binomial)
summary(budworm.lg0, cor = F)$coefficients

dose.p(budworm.lg0, cf = c(1,3), p = 1:3/4)
dose.p(update(budworm.lg0, family = binomial(link = probit)),
      cf = c(1, 3), p = 1:3/4)

options(contrasts = c("contr.treatment", "contr.poly"))
attach(birthwt)
race <- factor(race, labels = c("white", "black", "other"))
table(ptl)
ptd <- factor(ptl > 0)
table(ftv)
ftv <- factor(ftv)
levels(ftv)[- (1:2)] <- "2+"
table(ftv) # as a check
bwt <- data.frame(low = factor(low), age, lwt, race,
      smoke = (smoke > 0), ptd, ht = (ht > 0), ui = (ui > 0), ftv)
detach(); rm(race, ptd, ftv)

birthwt.glm <- glm(low ~ ., family = binomial, data = bwt)
summary(birthwt.glm, cor = F)
birthwt.step <- stepAIC(birthwt.glm, trace = F)
birthwt.step$anova
birthwt.step2 <- stepAIC(birthwt.glm, ~ .^2 + I(scale(age)^2)
      + I(scale(lwt)^2), trace = F)
birthwt.step2$anova
summary(birthwt.step2, cor = F)$coef
```

```

table(bwt$low, predict(birthwt.step2) > 0)

## R has a similar gam() in package gam and a different gam() in package mgcv
library(gam)
attach(bwt)
age1 <- age*(ftv=="1"); age2 <- age*(ftv=="2+")
birthwt.gam <- gam(low ~ s(age) + s(lwt) + smoke + ptd +
  ht + ui + ftv + s(age1) + s(age2) + smoke:ui, binomial,
  bwt, bf.maxit=25)
summary(birthwt.gam)
table(low, predict(birthwt.gam) > 0)
par(mfrow = c(2, 2))
if(interactive()) plot(birthwt.gam, ask = TRUE, se = TRUE)
par(mfrow = c(1, 1))
detach()

library(mgcv)
attach(bwt)
age1 <- age*(ftv=="1"); age2 <- age*(ftv=="2+")
(birthwt.gam <- gam(low ~ s(age) + s(lwt) + smoke + ptd +
  ht + ui + ftv + s(age1) + s(age2) + smoke:ui, binomial, bwt))
table(low, predict(birthwt.gam) > 0)
par(mfrow = c(2, 2))
plot(birthwt.gam, se = TRUE)
par(mfrow = c(1, 1))
detach()

# 7.3 Poisson models

names(housing)
house.glm0 <- glm(Freq ~ Infl*Type*Cont + Sat,
  family = poisson, data = housing)
summary(house.glm0, cor = F)

addterm(house.glm0, ~. + Sat:(Infl+Type+Cont), test = "Chisq")

house.glm1 <- update(house.glm0, . ~ . + Sat:(Infl+Type+Cont))
summary(house.glm1, cor = F)
1 - pchisq(deviance(house.glm1), house.glm1$df.resid)

dropterm(house.glm1, test = "Chisq")

addterm(house.glm1, ~. + Sat:(Infl+Type+Cont)^2, test = "Chisq")

hnames <- lapply(housing[, -5], levels) # omit Freq
house.pm <- predict(house.glm1, expand.grid(hnames),
  type = "response") # poisson means
house.pm <- matrix(house.pm, ncol = 3, byrow = T,
  dimnames = list(NULL, hnames[[1]]))
house.pr <- house.pm/drop(house.pm %*% rep(1, 3))
cbind(expand.grid(hnames[-1]), round(house.pr, 2))

loglm(Freq ~ Infl*Type*Cont + Sat*(Infl+Type+Cont),
  data = housing)

library(nnet)
(house.mult <- multinom(Sat ~ Infl + Type + Cont,
  weights = Freq, data = housing))
house.mult2 <- multinom(Sat ~ Infl*Type*Cont,
  weights = Freq, data = housing)
anova(house.mult, house.mult2, test = "none")

house.pm <- predict(house.mult, expand.grid(hnames[-1]),
  type = "probs")
cbind(expand.grid(hnames[-1]), round(house.pm, 2))

```

```

house.cpr <- apply(house.pr, 1, cumsum)
logit <- function(x) log(x/(1-x))
house.ld <- logit(house.cpr[2, 1]) - logit(house.cpr[1, 1])
sort(drop(house.ld))
mean(.Last.value)

house.plr <- polr(Sat ~ Infl + Type + Cont,
                 data = housing, weights = Freq)
house.plr

house.pr1 <- predict(house.plr, expand.grid(hnames[-1]),
                    type = "probs")
cbind(expand.grid(hnames[-1]), round(house.pr1, 2))

Fr <- matrix(housing$Freq, ncol = 3, byrow = T)
2 * sum(Fr * log(house.pr/house.pr1))

house.plr2 <- stepAIC(house.plr, ~.^2)
house.plr2$anova

# 7.4 A negative binomial family

glm(Days ~ .^4, family = poisson, data = quine)
quine.nb <- glm(Days ~ .^4, family = negative.binomial(2), data = quine)
quine.nb0 <- update(quine.nb, . ~ Sex/(Age + Eth*Lrn))
anova(quine.nb0, quine.nb, test = "Chisq")

quine.nb <- glm.nb(Days ~ .^4, data = quine)
quine.nb2 <- stepAIC(quine.nb)
quine.nb2$anova
dropterm(quine.nb2, test = "Chisq")
quine.nb3 <-
  update(quine.nb2, . ~ . - Eth:Age:Lrn - Sex:Age:Lrn)
anova(quine.nb2, quine.nb3)
c(theta = quine.nb2$theta, SE = quine.nb2$SE)

par(mfrow = c(2,2), pty = "m")
rs <- resid(quine.nb2, type = "deviance")
plot(predict(quine.nb2), rs, xlab = "Linear predictors",
      ylab = "Deviance residuals")
abline(h = 0, lty = 2)
qqnorm(rs, ylab = "Deviance residuals")
qqline(rs)
par(mfrow = c(1,1))

# End of ch07
#-*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 8 Non-linear and Smooth Regression

library(MASS)
library(lattice)
options(echo = T, width=65, digits=5, height=9999)
trellis.device(postscript, file="ch08.ps", width=8, height=6, pointsize=9)

# From Chapter 6, for comparisons
set.seed(123)
cpus.samp <-
c(3, 5, 6, 7, 8, 10, 11, 16, 20, 21, 22, 23, 24, 25, 29, 33, 39, 41, 44, 45,
46, 49, 57, 58, 62, 63, 65, 66, 68, 69, 73, 74, 75, 76, 78, 83, 86,
88, 98, 99, 100, 103, 107, 110, 112, 113, 115, 118, 119, 120, 122,

```

```
124, 125, 126, 127, 132, 136, 141, 144, 146, 147, 148, 149, 150, 151,
152, 154, 156, 157, 158, 159, 160, 161, 163, 166, 167, 169, 170, 173,
174, 175, 176, 177, 183, 184, 187, 188, 189, 194, 195, 196, 197, 198,
199, 202, 204, 205, 206, 208, 209)
```

```
cpus1 <- cpus
attach(cpus)
for(v in names(cpus)[2:7])
  cpus1[[v]] <- cut(cpus[[v]], unique(quantile(cpus[[v]])),
    include.lowest = T)
detach()
cpus.lm <- lm(log10(perf) ~ ., data=cpus1[cpus.samp, 2:8])
cpus.lm2 <- stepAIC(cpus.lm, trace=F)
res2 <- log10(cpus1[-cpus.samp, "perf"]) -
  predict(cpus.lm2, cpus1[-cpus.samp,])
cpus2 <- cpus[, 2:8] # excludes names, authors' predictions
cpus2[, 1:3] <- log10(cpus2[, 1:3])

test.cpus <- function(fit)
  sqrt(sum((log10(cpus2[-cpus.samp, "perf"]) -
    predict(fit, cpus2[-cpus.samp,]))^2)/109)
```

8.1 An introductory example

```
attach(wtloss)
# alter margin 4; others are default
oldpar <- par(mar = c(5.1, 4.1, 4.1, 4.1))
plot(Days, Weight, type = "p", ylab = "Weight (kg)")
Wt.lbs <- pretty(range(Weight*2.205))
axis(side = 4, at = Wt.lbs/2.205, lab = Wt.lbs, srt = 90)
mtext("Weight (lb)", side = 4, line = 3)
par(oldpar) # restore settings
detach()
```

8.2 Fitting non-linear regression models

```
wtloss.st <- c(b0 = 90, b1 = 95, th = 120)
wtloss.fm <- nls(Weight ~ b0 + b1*2^(-Days/th),
  data = wtloss, start = wtloss.st, trace = T)
wtloss.fm

expn <- function(b0, b1, th, x) {
  temp <- 2^(-x/th)
  model.func <- b0 + b1 * temp
  Z <- cbind(1, temp, (b1 * x * temp * log(2))/th^2)
  dimnames(Z) <- list(NULL, c("b0", "b1", "th"))
  attr(model.func, "gradient") <- Z
  model.func
}

wtloss.gr <- nls(Weight ~ expn(b0, b1, th, Days),
  data = wtloss, start = wtloss.st, trace = T)

expn1 <- deriv(y ~ b0 + b1 * 2^(-x/th), c("b0", "b1", "th"),
  function(b0, b1, th, x) {})

negexp <- selfStart(model = ~ b0 + b1*exp(-x/th),
  initial = negexp.SSival, parameters = c("b0", "b1", "th"),
  template = function(x, b0, b1, th) {})

wtloss.ss <- nls(Weight ~ negexp(Days, B0, B1, theta),
  data = wtloss, trace = T)
```

8.3 Non-linear fitted model objects and method functions

```
summary(wtloss.gr)
deviance(wtloss.gr)
vcov(wtloss.gr)

A <- model.matrix(~ Strip - 1, data = muscle)
rats.nls1 <- nls(log(Length) ~ cbind(A, rho^Conc),
  data = muscle, start = c(rho = 0.1), algorithm = "plinear")
(B <- coef(rats.nls1))

st <- list(alpha = B[2:22], beta = B[23], rho = B[1])
rats.nls2 <- nls(log(Length) ~ alpha[Strip] + beta*rho^Conc,
  data = muscle, start = st)

attach(muscle)
Muscle <- expand.grid(Conc = sort(unique(Conc)),
  Strip = levels(Strip))
Muscle$Yhat <- predict(rats.nls2, Muscle)
Muscle$logLength <- rep(NA, nrow(Muscle))
ind <- match(paste(Strip, Conc),
  paste(Muscle$Strip, Muscle$Conc))
Muscle$logLength[ind] <- log(Length)
detach()

xyplot(Yhat ~ Conc | Strip, Muscle, as.table = TRUE,
  ylim = range(c(Muscle$Yhat, Muscle$logLength), na.rm = TRUE),
  subscripts = T, xlab = "Calcium Chloride concentration (mM)",
  ylab = "log(Length in mm)", panel =
  function(x, y, subscripts, ...) {
    lines(spline(x, y))
    panel.xyplot(x, Muscle$logLength[subscripts], ...)
  })

# 8.5 Confidence intervals for parameters

expn2 <- deriv(~b0 + b1*((w0 - b0)/b1)^(x/d0),
  c("b0","b1","d0"), function(b0, b1, d0, x, w0) {}))

wtloss.init <- function(obj, w0) {
  p <- coef(obj)
  d0 <- - log((w0 - p["b0"])/p["b1"])/log(2) * p["th"]
  c(p[c("b0", "b1")], d0 = as.vector(d0))
}

out <- NULL
w0s <- c(110, 100, 90)
for(w0 in w0s) {
  fm <- nls(Weight ~ expn2(b0, b1, d0, Days, w0),
    wtloss, start = wtloss.init(wtloss.gr, w0))
  out <- rbind(out, c(coef(fm)["d0"], confint(fm, "d0")))
}
dimnames(out)[[1]] <- paste(w0s, "kg:")
out

fm0 <- lm(Wt*Time ~ Viscosity + Time - 1, data = stormer)
b0 <- coef(fm0)
names(b0) <- c("b1", "b2")
b0
storm.fm <- nls(Time ~ b1*Viscosity/(Wt-b2), data = stormer,
  start = b0, trace = T)

bc <- coef(storm.fm)
se <- sqrt(diag(vcov(storm.fm)))
dv <- deviance(storm.fm)
```

```

par(pty = "s")
b1 <- bc[1] + seq(-3*se[1], 3*se[1], length = 51)
b2 <- bc[2] + seq(-3*se[2], 3*se[2], length = 51)
bv <- expand.grid(b1, b2)

attach(stormer)
ssq <- function(b)
  sum((Time - b[1] * Viscosity/(Wt-b[2]))^2)
dbetas <- apply(bv, 1, ssq)

cc <- matrix(Time - rep(bv[,1],rep(23, 2601)) *
  Viscosity/(Wt - rep(bv[,2], rep(23, 2601))), 23)
dbetas <- matrix(drop(rep(1, 23) %*% cc^2), 51)

fstat <- matrix( ((dbetas - dv)/2) / (dv/21), 51, 51)

qf(0.95, 2, 21)

plot(b1, b2, type = "n")
lev <- c(1, 2, 5, 7, 10, 15, 20)
contour(b1, b2, fstat, levels = lev, labex = 0.75, lty = 2, add = T)
contour(b1, b2, fstat, levels = qf(0.95,2,21), add = T, labex = 0)
text(31.6, 0.3, labels = "95% CR", adj = 0, cex = 0.75)
points(bc[1], bc[2], pch = 3, mkh = 0.1)
detach()
par(pty = "m")

library(boot)
storm.fm <- nls(Time ~ b*Viscosity/(Wt - c), stormer,
  start = c(b=29.401, c=2.2183))
summary(storm.fm)$parameters
st <- cbind(stormer, fit=fitted(storm.fm))
storm.bf <- function(rs, i) {
#   st <- st # for S-PLUS
  st$Time <- st$fit + rs[i]
  coef(nls(Time ~ b * Viscosity/(Wt - c), st,
    start = coef(storm.fm)))
}
rs <- scale(resid(storm.fm), scale = FALSE) # remove the mean
(storm.boot <- boot(rs, storm.bf, R = 9999)) ## slow
boot.ci(storm.boot, index = 1,
  type = c("norm", "basic", "perc", "bca"))
boot.ci(storm.boot, index = 2,
  type = c("norm", "basic", "perc", "bca"))

# 8.5 Assessing the linear approximation

opar <- par(pty = "m", mfrow = c(1, 3))
plot(profile(update(wtloss.gr, trace = FALSE)))
par(opar)

# 8.7 One-dimensional curve fitting

attach(GAGurine)
par(mfrow = c(3, 2))
plot(Age, GAG, main = "Degree 6 polynomial")
GAG.lm <- lm(GAG ~ Age + I(Age^2) + I(Age^3) + I(Age^4) +
  I(Age^5) + I(Age^6) + I(Age^7) + I(Age^8))
anova(GAG.lm)
GAG.lm2 <- lm(GAG ~ Age + I(Age^2) + I(Age^3) + I(Age^4) +
  I(Age^5) + I(Age^6))
xx <- seq(0, 17, len = 200)
lines(xx, predict(GAG.lm2, data.frame(Age = xx)))

```

```

library(splines)
plot(Age, GAG, type = "n", main = "Splines")
lines(Age, fitted(lm(GAG ~ ns(Age, df = 5))))
lines(Age, fitted(lm(GAG ~ ns(Age, df = 10))), lty = 3)
lines(Age, fitted(lm(GAG ~ ns(Age, df = 20))), lty = 4)
lines(smooth.spline(Age, GAG), lwd = 3)
legend(12, 50, c("df=5", "df=10", "df=20", "Smoothing"),
      lty = c(1, 3, 4, 1), lwd = c(1,1,1,3), bty = "n")
plot(Age, GAG, type = "n", main = "loess")
lines(loess.smooth(Age, GAG))
plot(Age, GAG, type = "n", main = "supsmu")
lines(supsmu(Age, GAG))
lines(supsmu(Age, GAG, bass = 3), lty = 3)
lines(supsmu(Age, GAG, bass = 10), lty = 4)
legend(12, 50, c("default", "base = 3", "base = 10"),
      lty = c(1, 3, 4), bty = "n")
plot(Age, GAG, type = "n", main = "ksmooth")
lines(ksmooth(Age, GAG, "normal", bandwidth = 1))
lines(ksmooth(Age, GAG, "normal", bandwidth = 5), lty = 3)
legend(12, 50, c("width = 1", "width = 5"), lty = c(1, 3), bty = "n")

```

```

library(KernSmooth)
plot(Age, GAG, type = "n", main = "locpoly")
(h <- dpill(Age, GAG))
lines(locpoly(Age, GAG, degree = 0, bandwidth = h))

lines(locpoly(Age, GAG, degree = 1, bandwidth = h), lty = 3)
lines(locpoly(Age, GAG, degree = 2, bandwidth = h), lty = 4)
legend(12, 50, c("const", "linear", "quadratic"),
      lty = c(1, 3, 4), bty = "n")
detach()

```

8.8 Additive models

```

## R has a different gam() in package mgcv
library(mgcv)
rock.lm <- lm(log(perm) ~ area + peri + shape, data = rock)
summary(rock.lm)
(rock.gam <- gam(log(perm) ~ s(area) + s(peri) + s(shape), data=rock))
#summary(rock.gam)
#anova(rock.lm, rock.gam)
par(mfrow = c(2, 3), pty = "s")
plot(rock.gam, se = TRUE, pages = 0)
rock.gaml <- gam(log(perm) ~ area + peri + s(shape), data = rock)
plot(rock.gaml, se = TRUE)
par(pty="m")
#anova(rock.lm, rock.gaml, rock.gam)

```

```

library(mda)
rock.bruto <- bruto(rock[, -4], rock[, 4])
rock.bruto$type
rock.bruto$df

```

```

Xin <- as.matrix(cpus2[cpus.samp, 1:6])
test2 <- function(fit) {
  Xp <- as.matrix(cpus2[-cpus.samp, 1:6])
  sqrt(sum((log10(cpus2[-cpus.samp, "perf"]) -
             predict(fit, Xp))^2)/109)
}

```

```

cpus.bruto <- bruto(Xin, log10(cpus2[cpus.samp, 7]))
test2(cpus.bruto)
cpus.bruto$type

```

```

cpus.bruto$df
# examine the fitted functions
par(mfrow = c(3, 2))
Xp <- matrix(sapply(cpus2[cpus.samp, 1:6], mean), 100, 6, byrow = T)
for(i in 1:6) {
  xr <- sapply(cpus2, range)
  Xp1 <- Xp; Xp1[, i] <- seq(xr[1, i], xr[2, i], len = 100)
  Xf <- predict(cpus.bruto, Xp1)
  plot(Xp1[, i], Xf, xlab=names(cpus2)[i], ylab= "", type = "l")
}

cpus.mars <- mars(Xin, log10(cpus2[cpus.samp,7]))
showcuts <- function(obj)
{
  tmp <- obj$cuts[obj$sel, ]
  dimnames(tmp) <- list(NULL, dimnames(Xin)[[2]])
  tmp
}
showcuts(cpus.mars)
test2(cpus.mars)
# examine the fitted functions
Xp <- matrix(sapply(cpus2[cpus.samp, 1:6], mean), 100, 6, byrow = T)
for(i in 1:6) {
  xr <- sapply(cpus2, range)
  Xp1 <- Xp; Xp1[, i] <- seq(xr[1, i], xr[2, i], len = 100)
  Xf <- predict(cpus.mars, Xp1)
  plot(Xp1[, i], Xf, xlab = names(cpus2)[i], ylab = "", type = "l")
}

cpus.mars2 <- mars(Xin, log10(cpus2[cpus.samp,7]), degree = 2)
showcuts(cpus.mars2)
test2(cpus.mars2)

cpus.mars6 <- mars(Xin, log10(cpus2[cpus.samp,7]), degree = 6)
showcuts(cpus.mars6)
test2(cpus.mars6)

library(acepack)
attach(cpus2)
cpus.avas <- avas(cpus2[, 1:6], perf)
plot(log10(perf), cpus.avas$ty)
par(mfrow = c(2, 3))
for(i in 1:6) {
  o <- order(cpus2[, i])
  plot(cpus2[o, i], cpus.avas$tx[o, i], type = "l",
       xlab = names(cpus2[i]), ylab = "")
}
detach()

# 8.9 Projection-pursuit regression

attach(rock)
rock1 <- data.frame(area = area/10000, peri = peri/10000,
                   shape = shape, perm = perm)
detach()
(rock.ppr <- ppr(log(perm) ~ area + peri + shape, data = rock1,
               nterms = 2, max.terms = 5))
rock.ppr
summary(rock.ppr)

par(mfrow = c(3, 2))
plot(rock.ppr)
plot(update(rock.ppr, bass = 5))
plot(rock.ppr2 <- update(rock.ppr, sm.method = "gcv", gcvpen = 2))
par(mfrow = c(1, 1))

```

```

summary(rock.ppr2)

summary(rock1) # to find the ranges of the variables
Xp <- expand.grid(area = seq(0.1, 1.2, 0.05),
                 peri = seq(0, 0.5, 0.02), shape = 0.2)
rock.grid <- cbind(Xp, fit = predict(rock.ppr2, Xp))
wireframe(fit ~ area + peri, rock.grid, screen = list(z=160,x=-60),
          aspect = c(1, 0.5), drape = TRUE)
# or
persp(seq(0.1, 1.2, 0.05), seq(0, 0.5, 0.02), matrix(rock.grid$fit, 23),
      d = 5, theta = -160, phi = 30, zlim = c(-1, 15))

(cpus.ppr <- ppr(log10(perf) ~ ., data = cpus2[cpus.samp,],
               nterms = 2, max.terms = 10, bass = 5))
cpus.ppr <- ppr(log10(perf) ~ ., data = cpus2[cpus.samp,],
               nterms = 8, max.terms = 10, bass = 5)
test.cpus(cpus.ppr)

ppr(log10(perf) ~ ., data = cpus2[cpus.samp,],
    nterms = 2, max.terms = 10, sm.method = "spline")
cpus.ppr2 <- ppr(log10(perf) ~ ., data = cpus2[cpus.samp,],
                 nterms = 7, max.terms = 10, sm.method = "spline")
test.cpus(cpus.ppr2)
res3 <- log10(cpus2[-cpus.samp, "perf"]) -
  predict(cpus.ppr, cpus2[-cpus.samp,])
wilcox.test(res2^2, res3^2, paired = T, alternative = "greater")

# 8.10 Neural networks

library(nnet)
attach(rock)
areal <- area/10000; peril <- peri/10000
rock1 <- data.frame(perm, area = areal, peri = peril, shape)
rock.nn <- nnet(log(perm) ~ area + peri + shape, rock1,
               size = 3, decay = 1e-3, linout = T, skip = T,
               maxit = 1000, Hess = T)
sum((log(perm) - predict(rock.nn))^2)
detach()
eigen(rock.nn$Hessian, T)$values # rock.nn$Hessian in R

Xp <- expand.grid(area = seq(0.1, 1.2, 0.05),
                 peri = seq(0, 0.5, 0.02), shape = 0.2)
rock.grid <- cbind(Xp, fit = predict(rock.nn, Xp))
wireframe(fit ~ area + peri, rock.grid, screen = list(z=160, x=-60),
          aspect = c(1, 0.5), drape = TRUE)
# or
persp(seq(0.1, 1.2, 0.05), seq(0, 0.5, 0.02), matrix(rock.grid$fit, 23),
      d = 5, theta = -160, phi = 30, zlim = c(-1, 15))

attach(cpus2)
cpus3 <-
  data.frame(syct= syct-2, mmin=mmin-3, mmax=mmax-4, cach=cach/256,
            chmin=chmin/100, chmax=chmax/100, perf=perf)
detach()

test.cpus <- function(fit)
  sqrt(sum((log10(cpus3[-cpus.samp, "perf"]) -
            predict(fit, cpus3[-cpus.samp,]))^2)/109)
cpus.nn1 <- nnet(log10(perf) ~ ., cpus3[cpus.samp,],
                linout = T, skip = T, size = 0)
test.cpus(cpus.nn1)

cpus.nn2 <- nnet(log10(perf) ~ ., cpus3[cpus.samp,], linout = T,

```

```

        skip = T, size = 4, decay = 0.01, maxit = 1000)
test.cpus(cpus.nn2)
cpus.nn3 <- nnet(log10(perf) ~ ., cpus3[cpus.samp,], linout = T,
               skip = T, size = 10, decay = 0.01, maxit = 1000)
test.cpus(cpus.nn3)
cpus.nn4 <- nnet(log10(perf) ~ ., cpus3[cpus.samp,], linout = T,
               skip = T, size = 25, decay = 0.01, maxit = 1000)
test.cpus(cpus.nn4)

CVnn.cpus <- function(formula, data = cpus3[cpus.samp, ],
                     size = c(0, 4, 4, 10, 10),
                     lambda = c(0, rep(c(0.003, 0.01), 2)),
                     nreps = 5, nifold = 10, ...)
{
  CVnn1 <- function(formula, data, nreps=1, ri, ...)
  {
    truth <- log10(data$perf)
    res <- numeric(length(truth))
    cat("  fold")
    for (i in sort(unique(ri))) {
      cat(" ", i, sep="")
      for(rep in 1:nreps) {
        learn <- nnet(formula, data[ri !=i,], trace=F, ...)
        res[ri == i] <- res[ri == i] +
          predict(learn, data[ri == i,])
      }
    }
    cat("\n")
    sum((truth - res/nreps)^2)
  }
  choice <- numeric(length(lambda))
  ri <- sample(nifold, nrow(data), replace = TRUE)
  for(j in seq(along=lambda)) {
    cat("  size =", size[j], "decay =", lambda[j], "\n")
    choice[j] <- CVnn1(formula, data, nreps=nreps, ri=ri,
                      size=size[j], decay=lambda[j], ...)
  }
  cbind(size=size, decay=lambda, fit=sqrt(choice/100))
}
CVnn.cpus(log10(perf) ~ ., data = cpus3[cpus.samp,],
          linout = T, skip = T, maxit = 1000)

# End of ch08
#-*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 9  Tree-based Methods

library(MASS)
postscript(file="ch09.ps", width=8, height=6, pointsize=9)
options(echo = TRUE, digits=5)

library(rpart)

# Figure 9.3
shuttle.rp <- rpart(use ~ ., data=shuttle, minbucket=0, xval = 0,
                   maxsurrogate = 0, cp = 0, subset = 1:253)
post(shuttle.rp, horizontal = FALSE, height=10, width=8, title = "",
     pointsize = 8, pretty = 0)

# 9.3  Implementation in rpart

set.seed(123)
cpus.rp <- rpart(log10(perf) ~ ., cpus[ , 2:8], cp = 1e-3)

```

```

cpus.rp
print(cpus.rp, cp = 0.01) # default pruning

plot(cpus.rp, uniform = TRUE)
text(cpus.rp, digits = 3)

printcp(cpus.rp)
plotcp(cpus.rp)

cpus.rp1 <- prune(cpus.rp, cp = 0.006)
print(cpus.rp1, digits = 3)
plot(cpus.rp1, branch = 0.4, uniform = TRUE)
text(cpus.rp1, digits = 3)

# for figure 9.2
cpus.rp2 <- prune(cpus.rp, cp = 0.03)
post(cpus.rp2, horizontal = FALSE, title = "", digits=4, pointsize=18)

set.seed(123)
fgl.rp <- rpart(type ~ ., fgl, cp = 0.001)
plotcp(fgl.rp)
printcp(fgl.rp)

fgl.rp2 <- prune(fgl.rp, cp = 0.02)
plot(fgl.rp2, uniform = TRUE)
text(fgl.rp2, use.n = TRUE)
fgl.rp2

summary(fgl.rp2)

set.seed(123)
fgl.rp3 <- rpart(type ~ ., fgl, cp = 0.001,
                parms = list(split="information"))
plotcp(fgl.rp3)
printcp(fgl.rp3)

fgl.rp4 <- prune(fgl.rp3, cp = 0.03)
plot(fgl.rp4, uniform = TRUE); text(fgl.rp4, use.n = TRUE)

plot(cpus.rp, branch = 0.6, compress = TRUE, uniform = TRUE)
text(cpus.rp, digits = 3, all = TRUE, use.n = TRUE)

# 9.3 Implementation in tree

library(tree)
## the stopping criteria differ slightly between R and S-PLUS
cpus.ltr <- tree(log10(perf) ~ ., data = cpus[, 2:8], mindev = 0.005)
summary(cpus.ltr)

cpus.ltr
plot(cpus.ltr, type="u"); text(cpus.ltr)

par(mfrow = c(1, 2), pty = "s")
set.seed(321)
plot(cv.tree(cpus.ltr, , prune.tree))
cpus.ltr1 <- prune.tree(cpus.ltr, best = 10)
plot(cpus.ltr1, type = "u")
text(cpus.ltr1, digits = 3)
par(mfrow = c(1, 1), pty = "m")

fgl.tr <- tree(type ~ ., fgl)
summary(fgl.tr)
plot(fgl.tr)
text(fgl.tr, all = TRUE, cex = 0.5)

```

```

par(mfrow = c(1, 2), pty = "s")
set.seed(123)
fgl.cv <- cv.tree(fgl.tr,, prune.misclass)
for(i in 2:5) fgl.cv$dev <- fgl.cv$dev +
  cv.tree(fgl.tr,, prune.misclass)$dev
fgl.cv$dev <- fgl.cv$dev/5
fgl.cv
plot(fgl.cv)

fgl.tr1 <- prune.misclass(fgl.tr, best = 9)
plot(fgl.tr1, type = "u")
text(fgl.tr1, all = TRUE)

# End of ch09
#*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 10 Random and Mixed Effects

library(MASS)
library(lattice)
trellis.device(postscript, file="ch10.ps", width=8, height=6, pointsize=9)
options(echo=T, width=65, digits=5)
library(nlme)

# 10.1 Linear models

xyplot(Y ~ EP | No, data = petrol,
  xlab = "ASTM end point (deg. F)",
  ylab = "Yield as a percent of crude",
  panel = function(x, y) {
    panel.grid()
    m <- sort.list(x)
    panel.xyplot(x[m], y[m], type = "b", cex = 0.5)
  })

Petrol <- petrol
names(Petrol)
Petrol[, 2:5] <- scale(Petrol[, 2:5], scale = F)
pet1.lm <- lm(Y ~ No/EP - 1, Petrol)
matrix(round(coef(pet1.lm), 2), 2, 10, byrow = T,
  dimnames = list(c("b0", "b1"), levels(Petrol$No)))

pet2.lm <- lm(Y ~ No - 1 + EP, Petrol)
anova(pet2.lm, pet1.lm)

pet3.lm <- lm(Y ~ SG + VP + V10 + EP, Petrol)
anova(pet3.lm, pet2.lm)

pet3.lme <- lme(Y ~ SG + VP + V10 + EP,
  random = ~ 1 | No, data = Petrol)
summary(pet3.lme)

pet3.lme <- update(pet3.lme, method = "ML")
summary(pet3.lme)

anova(pet3.lme, pet3.lm)

pet4.lme <- update(pet3.lme, fixed = Y ~ V10 + EP)
anova(pet4.lme, pet3.lme)
fixed.effects(pet4.lme)
coef(pet4.lme)

pet5.lme <- update(pet4.lme, random = ~ 1 + EP | No)

```

```

anova(pet4.lme, pet5.lme)

nl1 <- nlschools
attach(nl1)
classMeans <- tapply(IQ, class, mean)
nl1$IQave <- classMeans[as.character(class)]
detach()
cen <- c("IQ", "IQave", "SES")
nl1[cen] <- scale(nl1[cen], center = T, scale = F)

options(contrasts = c("contr.treatment", "contr.poly"))
nl.lme <- lme(lang ~ IQ*COMB + IQave + SES,
             random = ~ IQ | class, data = nl1)
summary(nl.lme)

summary(lm(lang ~ IQ*COMB + SES + class, data = nl1,
           singular.ok = T), cor = F)

nl2 <- cbind(aggregate(nl1[c(1,7)], list(class = nl1$class), mean),
            unique(nl1[c("class", "COMB", "GS")]))
summary(lm(lang ~ IQave + COMB, data = nl2, weights = GS),
       cor = F)

sitka.lme <- lme(size ~ treat*ordered(Time),
               random = ~1 | tree, data = Sitka, method = "ML")
Sitka <- Sitka # make a local copy for S-PLUS
attach(Sitka)
Sitka$treatslope <- Time * (treat == "ozone")
detach()
sitka.lme2 <- update(sitka.lme,
                   fixed = size ~ ordered(Time) + treat + treatslope)
anova(sitka.lme, sitka.lme2)

# fitted curves
matrix(fitted(sitka.lme2, level = 0)[c(301:305, 1:5)],
      2, 5, byrow = T,
      dimnames = list(c("control", "ozone"), unique(Sitka$Time)))

# 10.2 Classic nested designs

if(F) {
summary(raov(Conc ~ Lab/Bat, data = coop, subset = Spc=="S1"))

is.random(coop) <- T
is.random(coop$Spc) <- F
is.random(coop)

varcomp(Conc ~ Lab/Bat, data = coop, subset = Spc=="S1")

varcomp(Conc ~ Lab/Bat, data = coop, subset = Spc=="S1",
       method = c("winsor", "minque0"))
}

#oats <- oats # make a local copy: needed in S-PLUS
oats$Nf <- ordered(oats$N, levels = sort(levels(oats$N)))

oats.aov <- aov(Y ~ Nf*V + Error(B/V), data = oats, qr = T)
summary(oats.aov)
summary(oats.aov, split = list(Nf = list(L = 1, Dev = 2:3)))

plot(fitted(oats.aov[[4]]), studres(oats.aov[[4]]))
abline(h = 0, lty = 2)
oats.pr <- proj(oats.aov)
qqnorm(oats.pr[[4]][,"Residuals"], ylab = "Stratum 4 residuals")
qqline(oats.pr[[4]][,"Residuals"])

```

```

oats.aov <- aov(Y ~ N + V + Error(B/V), data = oats, qr = T)
model.tables(oats.aov, type = "means", se = T)
# we can get the unimplemented standard errors from
se.contrast(oats.aov, list(N == "0.0cwt", N == "0.2cwt"), data=oats)
se.contrast(oats.aov, list(V == "Golden.rain", V == "Victory"), data=oats)

# is.random(oats$B) <- T
# varcomp(Y ~ N + V + B/V, data = oats)

lme(Conc ~ 1, random = ~1 | Lab/Bat, data = coop,
    subset = Spc=="S1")

options(contrasts = c("contr.treatment", "contr.poly"))
summary(lme(Y ~ N + V, random = ~1 | B/V, data = oats))

# 10.3 Non-linear mixed effects models

options(contrasts = c("contr.treatment", "contr.poly"))
sitka.nlme <- nlme(size ~ A + B * (1 - exp(-(Time-100)/C)),
    fixed = list(A ~ treat, B ~ treat, C ~ 1),
    random = A + B ~ 1 | tree, data = Sitka,
    start = list(fixed = c(2, 0, 4, 0, 100)), verbose = T)

summary(sitka.nlme)

summary(update(sitka.nlme,
    corr = corCAR1(0.95, ~Time | tree)))

Fpl <- deriv(~ A + (B-A)/(1 + exp((log(d) - ld50)/th)),
    c("A","B","ld50","th"), function(d, A, B, ld50, th) {})

st <- coef(nls(BPchange ~ Fpl(Dose, A, B, ld50, th),
    start = c(A = 25, B = 0, ld50 = 4, th = 0.25),
    data = Rabbit))
Rc.nlme <- nlme(BPchange ~ Fpl(Dose, A, B, ld50, th),
    fixed = list(A ~ 1, B ~ 1, ld50 ~ 1, th ~ 1),
    random = A + ld50 ~ 1 | Animal, data = Rabbit,
    subset = Treatment == "Control",
    start = list(fixed = st))
## The next fails on some R platforms and some versions of nlme
## Rm.nlme <- update(Rc.nlme, subset = Treatment=="MDL")
## so update starting values
st <- coef(nls(BPchange ~ Fpl(Dose, A, B, ld50, th),
    start = c(A = 25, B = 0, ld50 = 4, th = 0.25),
    data = Rabbit, subset = Treatment == "MDL"))
Rm.nlme <- update(Rc.nlme, subset = Treatment=="MDL",
    start = list(fixed = st))

Rc.nlme
Rm.nlme

c1 <- c(28, 1.6, 4.1, 0.27, 0)
R.nlmel <- nlme(BPchange ~ Fpl(Dose, A, B, ld50, th),
    fixed = list(A ~ Treatment, B ~ Treatment,
        ld50 ~ Treatment, th ~ Treatment),
    random = A + ld50 ~ 1 | Animal/Run, data = Rabbit,
    start = list(fixed = c1[c(1, 5, 2, 5, 3, 5, 4, 5)]))
summary(R.nlmel)
R.nlme2 <- update(R.nlmel,
    fixed = list(A ~ 1, B ~ 1, ld50 ~ Treatment, th ~ 1),
    start = list(fixed = c1[c(1:3, 5, 4)]))
anova(R.nlme2, R.nlmel)
summary(R.nlme2)

```

```

xyplot(BPchange ~ log(Dose) | Animal * Treatment, Rabbit,
  xlab = "log(Dose) of Phenylbiguanide",
  ylab = "Change in blood pressure (mm Hg)",
  subscripts = T, aspect = "xy", panel =
    function(x, y, subscripts) {
      panel.grid()
      panel.xyplot(x, y)
      sp <- spline(x, fitted(R.nlm2)[subscripts])
      panel.xyplot(sp$x, sp$y, type = "l")
    })

```

10.4 Generalized linear mixed models

```

# bacteria <- bacteria # needed in S-PLUS
contrasts(bacteria$trt) <- structure(contr.sdif(3),
  dimnames = list(NULL, c("drug", "encourage")))
summary(glm(y ~ trt * week, binomial, data = bacteria),
  cor = F)
summary(glm(y ~ trt + week, binomial, data = bacteria),
  cor = F)

summary(glm(y ~ lbase*trt + lage + V4, family = poisson,
  data = epil), cor = F)

# epil <- epil # needed in S-PLUS
epil2 <- epil[epil$period == 1, ]
epil2["period"] <- rep(0, 59); epil2["y"] <- epil2["base"]
epil["time"] <- 1; epil2["time"] <- 4
epil2 <- rbind(epil, epil2)
epil2$pred <- unclass(epil2$trt) * (epil2$period > 0)
epil2$subject <- factor(epil2$subject)
epil3 <- aggregate(epil2, list(epil2$subject, epil2$period > 0),
  function(x) if(is.numeric(x)) sum(x) else x[1])
epil3$pred <- factor(epil3$pred, labels = c("base", "placebo", "drug"))

contrasts(epil3$pred) <- structure(contr.sdif(3),
  dimnames = list(NULL, c("placebo-base", "drug-placebo")))
summary(glm(y ~ pred + factor(subject) + offset(log(time)),
  family = poisson, data = epil3), cor = F)

glm(y ~ factor(subject), family = poisson, data = epil)

library(survival)
bacteria$Time <- rep(1, nrow(bacteria))
coxph(Surv(Time, unclass(y)) ~ week + strata(ID),
  data = bacteria, method = "exact")
coxph(Surv(Time, unclass(y)) ~ factor(week) + strata(ID),
  data = bacteria, method = "exact")
coxph(Surv(Time, unclass(y)) ~ I(week > 2) + strata(ID),
  data = bacteria, method = "exact")

fit <- glm(y ~ trt + I(week > 2), binomial, data = bacteria)
summary(fit, cor = F)
sum(residuals(fit, type = "pearson")^2)

if(F) { # very slow
library(GLMMGibbs)
# declare a random intercept for each subject
epil$subject <- Ra(data = factor(epil$subject))
glmm(y ~ lbase*trt + lage + V4 + subject, family = poisson,
  data = epil, keep = 100000, thin = 100)

epil3$subject <- Ra(data = factor(epil3$subject))
glmm(y ~ pred + subject, family = poisson,

```

```

    data = epil3, keep = 100000, thin = 100)
}

summary(glmPQL(y ~ trt + I(week > 2), random = ~ 1 | ID,
              family = binomial, data = bacteria))
summary(glmPQL(y ~ lbase*trt + lage + V4,
              random = ~ 1 | subject,
              family = poisson, data = epil))
summary(glmPQL(y ~ pred, random = ~1 | subject,
              family = poisson, data = epil3))

# 10.5 GEE models

## modified for YAGS 3.21-3
library(yags)
attach(bacteria)
yags(y == "y" ~ trt + I(week > 2), family = binomial, alphainit=0,
     id = ID, corstr = "exchangeable")
detach("bacteria")

attach(epil)
yags(y ~ lbase*trt + lage + V4, family = poisson, alphainit=0,
     id = subject, corstr = "exchangeable")
detach("epil")

options(contrasts = c("contr.sum", "contr.poly"))
library(gee)
summary(gee(y ~ pred + factor(subject), family = poisson,
           id = subject, data = epil3, corstr = "exchangeable"))

# End of ch10
#-*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 11 Exploratory Multivariate Analysis

library(MASS)
postscript(file="ch11.ps", width=8, height=6, pointsize=9)
options(echo=T, width=65, digits=5)

# 11.1 Visualization methods

# ir <- rbind(iris[, ,1], iris[, ,2], iris[, ,3])
# ir <- rbind(iris3[, ,1], iris3[, ,2], iris3[, ,3])
# ir.species <- factor(c(rep("s", 50), rep("c", 50), rep("v", 50)))
# (ir.pca <- princomp(log(ir), cor = T))
summary(ir.pca)
plot(ir.pca)
ir.pc <- predict(ir.pca)
eqscplot(ir.pc[, 1:2], type = "n",
         xlab = "first principal component",
         ylab = "second principal component")
text(ir.pc[, 1:2], labels = as.character(ir.species),
     col = 3 + unclass(ir.species))

lcrabs <- log(crabs[, 4:8])
crabs.grp <- factor(c("B", "b", "O", "o")[rep(1:4, each = 50)])
(lcrabs.pca <- princomp(lcrabs))
loadings(lcrabs.pca)
lcrabs.pc <- predict(lcrabs.pca)
dimnames(lcrabs.pc) <- list(NULL, paste("PC", 1:5, sep = ""))

```

```

if(F) { # needs interaction with XGobi
library(xgobi)
xgobi(lcrabs, colors = c("SkyBlue", "SlateBlue", "Orange",
  "Red")[rep(1:4, each = 50)])
xgobi(lcrabs, glyphs = 12 + 5*rep(0:3, each = 50, 4))
}

ir.scal <- cmdscale(dist(ir), k = 2, eig = T)
ir.scal$points[, 2] <- -ir.scal$points[, 2]
eqsplot(ir.scal$points, type = "n")
text(ir.scal$points, labels = as.character(ir.species),
  col = 3 + unclass(ir.species), cex = 0.8)

distp <- dist(ir)
dist2 <- dist(ir.scal$points)
sum((distp - dist2)^2)/sum(distp^2)

ir.sam <- sammon(dist(ir[-143,]))
eqsplot(ir.sam$points, type = "n")
text(ir.sam$points, labels = as.character(ir.species[-143]),
  col = 3 + unclass(ir.species), cex = 0.8)

ir.iso <- isoMDS(dist(ir[-143,]))
eqsplot(ir.iso$points, type = "n")
text(ir.iso$points, labels = as.character(ir.species[-143]),
  col = 3 + unclass(ir.species), cex = 0.8)

cr.scale <- 0.5 * log(crabs$CL * crabs$CW)
slcrabs <- lcrabs - cr.scale
cr.means <- matrix(0, 2, 5)
cr.means[1,] <- colMeans(slcrabs[crabs$sex == "F", ])
cr.means[2,] <- colMeans(slcrabs[crabs$sex == "M", ])
dslcrabs <- slcrabs - cr.means[as.numeric(crabs$sex), ]
lcrabs.sam <- sammon(dist(dslcrabs))
eqsplot(lcrabs.sam$points, type = "n", xlab = "", ylab = "")
text(lcrabs.sam$points, labels = as.character(crabs.grp))

fgl.iso <- isoMDS(dist(as.matrix(fgl[-40, -10])))
eqsplot(fgl.iso$points, type = "n", xlab = "", ylab = "", axes = F)
# either
# for(i in seq(along = levels(fgl$type))) {
#   set <- fgl$type[-40] == levels(fgl$type)[i]
#   points(fgl.iso$points[set,], pch = 18, cex = 0.6, col = 2 + i)}
# key(text = list(levels(fgl$type), col = 3:8))
# or
text(fgl.iso$points,
  labels = c("F", "N", "V", "C", "T", "H")[fgl$type[-40]],
  cex = 0.6)
fgl.iso3 <- isoMDS(dist(as.matrix(fgl[-40, -10])), k = 3)
# S: brush(fgl.iso3$points)
fgl.col <- c("SkyBlue", "SlateBlue", "Orange", "Orchid",
  "Green", "HotPink")[fgl$type]
# xgobi(fgl.iso3$points, colors = fgl.col)

library(class)
gr <- somgrid(topo = "hexagonal")
crabs.som <- batchSOM(lcrabs, gr, c(4, 4, 2, 2, 1, 1, 1, 0, 0))
plot(crabs.som)

bins <- as.numeric(knn1(crabs.som$code, lcrabs, 0:47))
plot(crabs.som$grid, type = "n")
symbols(crabs.som$grid$pts[, 1], crabs.som$grid$pts[, 2],
  circles = rep(0.4, 48), inches = F, add = T)
text(crabs.som$grid$pts[bins, ] + rnorm(400, 0, 0.1),
  as.character(crabs.grp))

```

```

crabs.som2 <- SOM(lcrabs, gr); plot(crabs.som2)

state <- state.x77[, 2:7]; row.names(state) <- state.abb
biplot(princomp(state, cor = T), pc.biplot = T, cex = 0.7,
       expand = 0.8)

library(fastICA)
nICA <- 4
crabs.ica <- fastICA(crabs[, 4:8], nICA)
Z <- crabs.ica$$S
par(mfrow = c(2, nICA))
for(i in 1:nICA) boxplot(Z[, i] ~ crabs.grp)
par(mfrow = c(1, 1))

# S: stars(state.x77[, c(7, 4, 6, 2, 5, 3)], byrow = T)
stars(state.x77[, c(7, 4, 6, 2, 5, 3)])

parcoord(state.x77[, c(7, 4, 6, 2, 5, 3)])
parcoord(log(ir)[, c(3, 4, 2, 1)], col = 1 + (0:149)%/%50)

# 11.2 Cluster analysis

swiss.x <- as.matrix(swiss[,-1])
library(cluster)
# h <- hclust(dist(swiss.x), method = "connected")
h <- hclust(dist(swiss.x), method = "single")
plclust(h)
cutree(h, 3)
# S: plclust( clorder(h, cutree(h, 3) ))

pltree(diana(swiss.x))
par(mfrow = c(1, 1))

h <- hclust(dist(swiss.x), method = "average")
initial <- tapply(swiss.x, list(rep(cutree(h, 3),
  ncol(swiss.x)), col(swiss.x)), mean)
dimnames(initial) <- list(NULL, dimnames(swiss.x)[[2]])
km <- kmeans(swiss.x, initial)
(swiss.pca <- princomp(swiss.x))
swiss.px <- predict(swiss.pca)
dimnames(km$centers)[[2]] <- dimnames(swiss.x)[[2]]
swiss.centers <- predict(swiss.pca, km$centers)
eqsplot(swiss.px[, 1:2], type = "n",
       xlab = "first principal component",
       ylab = "second principal component")
text(swiss.px[, 1:2], labels = km$cluster)
points(swiss.centers[,1:2], pch = 3, cex = 3)
if(interactive()) identify(swiss.px[, 1:2], cex = 0.5)

swiss.pam <- pam(swiss.px, 3)
summary(swiss.pam)
eqsplot(swiss.px[, 1:2], type = "n",
       xlab = "first principal component",
       ylab = "second principal component")
text(swiss.px[,1:2], labels = swiss.pam$clustering)
points(swiss.pam$medoid[,1:2], pch = 3, cex = 3)

fanny(swiss.px, 3)

## From the on-line Errata:
##
## `The authors of mclust have chosen to re-use the name for a
## completely incompatible package. We can no longer recommend its
## use, and the code given in the first printing does not work in R's

```

```

##  mclust-2.x.'
##
library(mclust) # 2.x equivalent commands
h <- hc(modelName = "VVV", swiss.x)
(mh <- as.vector(hclass(h, 3)))
z <- me(modelName = "VVV", swiss.x, z = 0.5*(unmap(mh)+1/3))
eqsplot(swiss.px[, 1:2], type = "n",
        xlab = "first principal component",
        ylab = "second principal component")
text(swiss.px[, 1:2], labels = max.col(z$z))

vals <- EMclust(swiss.x) # all possible models, 0:9 clusters.
(sm <- summary(vals, swiss.x))
eqsplot(swiss.px[, 1:2], type = "n",
        xlab = "first principal component",
        ylab = "second principal component")
text(swiss.px[, 1:2], labels = sm$classification)

# 11.3 Factor analysis

ability.FA <- factanal(covmat = ability.cov, factors = 1)
ability.FA
(ability.FA <- update(ability.FA, factors = 2))
#summary(ability.FA)
round(loadings(ability.FA) %*% t(loadings(ability.FA)) +
      diag(ability.FA$uniq), 3)
# loadings(rotate(ability.FA, rotation = "oblimin"))

if(FALSE) {
par(pty = "s")
L <- loadings(ability.FA)
eqsplot(L, xlim = c(0,1), ylim = c(0,1))
if(interactive()) identify(L, dimnames(L)[[1]])
oblirot <- rotate(loadings(ability.FA), rotation = "oblimin")
naxes <- solve(oblirot$tmat)
arrows(rep(0, 2), rep(0, 2), naxes[,1], naxes[,2])
}

# 11.4 Discrete multivariate analysis

caith <- as.matrix(caith)
names(dimnames(caith)) <- c("eyes", "hair")
mosaicplot(caith, color = T)
House <- xtabs(Freq ~ Type + Infl + Cont + Sat, housing)
mosaicplot(House, color = T)

corresp(caith)

caith2 <- caith
dimnames(caith2)[[2]] <- c("F", "R", "M", "D", "B")
par(mfcol = c(1, 3))
plot(corresp(caith2, nf = 2)); title("symmetric")
plot(corresp(caith2, nf = 2), type = "rows"); title("rows")
plot(corresp(caith2, nf = 2), type = "col"); title("columns")
par(mfrow = c(1, 1))

farms.mca <- mca(farms, abbrev = T) # Use levels as names
plot(farms.mca, cex = rep(0.7, 2))

# End of ch11
#-*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

```

```

# Chapter 12 Classification

library(MASS)
postscript(file="ch12.ps", width=8, height=6, pointsize=9)
options(echo=T, width=65, digits=5)
library(class)
library(nnet)

# 12.1 Discriminant Analysis

ir <- rbind(iris3[, ,1], iris3[, ,2], iris3[, ,3])
ir.species <- factor(c(rep("s", 50), rep("c", 50), rep("v", 50)))

(ir.lda <- lda(log(ir), ir.species))
ir.ld <- predict(ir.lda, dimen = 2)$x
eqsplot(ir.ld, type = "n", xlab = "first linear discriminant",
        ylab = "second linear discriminant")
text(ir.ld, labels = as.character(ir.species[-143]),
     col = 3 + unclass(ir.species), cex = 0.8)

plot(ir.lda, dimen = 1)
plot(ir.lda, type = "density", dimen = 1)

lcrabs <- log(crabs[, 4:8])
crabs.grp <- factor(c("B", "b", "O", "o")[rep(1:4, each = 50)])

(dcrabs.lda <- lda(crabs$sex ~ FL + RW + CL + CW, lcrabs))
table(crabs$sex, predict(dcrabs.lda)$class)

(dcrabs.lda4 <- lda(crabs.grp ~ FL + RW + CL + CW, lcrabs))
dcrabs.pr4 <- predict(dcrabs.lda4, dimen = 2)
dcrabs.pr2 <- dcrabs.pr4$post[, c("B", "O")] %*% c(1, 1)
table(crabs$sex, dcrabs.pr2 > 0.5)

cr.t <- dcrabs.pr4$x[, 1:2]
eqsplot(cr.t, type = "n", xlab = "First LD", ylab = "Second LD")
text(cr.t, labels = as.character(crabs.grp))
perp <- function(x, y) {
  m <- (x+y)/2
  s <- - (x[1] - y[1])/(x[2] - y[2])
  abline(c(m[2] - s*m[1], s))
  invisible()
}
# For R replace @means by $means
cr.m <- lda(cr.t, crabs$sex)$means
points(cr.m, pch = 3, mkh = 0.3)
perp(cr.m[1, ], cr.m[2, ])

cr.lda <- lda(cr.t, crabs.grp)
x <- seq(-6, 6, 0.25)
y <- seq(-2, 2, 0.25)
Xcon <- matrix(c(rep(x, length(y)),
                rep(y, rep(length(x), length(y))))), ,2)
cr.pr <- predict(cr.lda, Xcon)$post[, c("B", "O")] %*% c(1,1)
contour(x, y, matrix(cr.pr, length(x), length(y)),
        levels = 0.5, labex = 0, add = T, lty= 3)

for(i in c("O", "o", "B", "b"))
  print(var(lcrabs[crabs.grp == i, ]))

fgl.ld <- predict(lda(type ~ ., fgl), dimen = 2)$x
eqsplot(fgl.ld, type = "n", xlab = "LD1", ylab = "LD2")
# either
# for(i in seq(along = levels(fgl$type))) {

```

```

# set <- fgl$type[-40] == levels(fgl$type)[i]
# points(fgl.ld[set,], pch = 18, cex = 0.6, col = 2 + i)}
# key(text = list(levels(fgl$type), col = 3:8))
# or
text(fgl.ld, cex = 0.6,
      labels = c("F", "N", "V", "C", "T", "H")[fgl$type[-40]])

fgl.rld <- predict(lda(type ~ ., fgl, method = "t"), dimen = 2)$x
eqsplot(fgl.rld, type = "n", xlab = "LD1", ylab = "LD2")
# either
# for(i in seq(along = levels(fgl$type))) {
# set <- fgl$type[-40] == levels(fgl$type)[i]
# points(fgl.rld[set,], pch = 18, cex = 0.6, col = 2 + i)}
# key(text = list(levels(fgl$type), col = 3:8))
# or
text(fgl.rld, cex = 0.6,
      labels = c("F", "N", "V", "C", "T", "H")[fgl$type[-40]])

# 12.2 Classification theory

#decrease len if you have little memory.
predplot <- function(object, main="", len = 100, ...)
{
  plot(Cushings[,1], Cushings[,2], log="xy", type="n",
        xlab = "Tetrahydrocortisone", ylab = "Pregnanetriol", main = main)
  for(il in 1:4) {
    set <- Cushings$type==levels(Cushings$type)[il]
    text(Cushings[set, 1], Cushings[set, 2],
          labels=as.character(Cushings$type[set]), col = 2 + il) }
  xp <- seq(0.6, 4.0, length=len)
  yp <- seq(-3.25, 2.45, length=len)
  cushT <- expand.grid(Tetrahydrocortisone = xp,
                       Pregnanetriol = yp)
  Z <- predict(object, cushT, ...); zp <- as.numeric(Z$class)
  zp <- Z$post[,3] - pmax(Z$post[,2], Z$post[,1])
  contour(exp(xp), exp(yp), matrix(zp, len),
          add = T, levels = 0, labex = 0)
  zp <- Z$post[,1] - pmax(Z$post[,2], Z$post[,3])
  contour(exp(xp), exp(yp), matrix(zp, len),
          add = T, levels = 0, labex = 0)
  invisible()
}

cushplot <- function(xp, yp, Z)
{
  plot(Cushings[, 1], Cushings[, 2], log = "xy", type = "n",
        xlab = "Tetrahydrocortisone", ylab = "Pregnanetriol")
  for(il in 1:4) {
    set <- Cushings$type==levels(Cushings$type)[il]
    text(Cushings[set, 1], Cushings[set, 2],
          labels = as.character(Cushings$type[set]), col = 2 + il) }
  zp <- Z[, 3] - pmax(Z[, 2], Z[, 1])
  contour(exp(xp), exp(yp), matrix(zp, np),
          add = T, levels = 0, labex = 0)
  zp <- Z[, 1] - pmax(Z[, 2], Z[, 3])
  contour(exp(xp), exp(yp), matrix(zp, np),
          add = T, levels = 0, labex = 0)
  invisible()
}

cush <- log(as.matrix(Cushings[, -3]))
tp <- Cushings$type[1:21, drop = T]
cush.lda <- lda(cush[1:21,], tp); predplot(cush.lda, "LDA")
cush.qda <- qda(cush[1:21,], tp); predplot(cush.qda, "QDA")
predplot(cush.qda, "QDA (predictive)", method = "predictive")

```

```

predplot(cush.qda, "QDA (debiased)", method = "debiased")

Cf <- data.frame(tp = tp,
  Tetrahydrocortisone = log(Cushings[1:21, 1]),
  Pregnanetriol = log(Cushings[1:21, 2]) )
cush.multinom <- multinom(tp ~ Tetrahydrocortisone
  + Pregnanetriol, Cf, maxit = 250)
xp <- seq(0.6, 4.0, length = 100); np <- length(xp)
yp <- seq(-3.25, 2.45, length = 100)
cushT <- expand.grid(Tetrahydrocortisone = xp,
  Pregnanetriol = yp)
Z <- predict(cush.multinom, cushT, type = "probs")
cushplot(xp, yp, Z)

library(tree)
cush.tr <- tree(tp ~ Tetrahydrocortisone + Pregnanetriol, Cf)
plot(cush[, 1], cush[, 2], type = "n",
  xlab = "Tetrahydrocortisone", ylab = "Pregnanetriol")
for(il in 1:4) {
  set <- Cushings$Type==levels(Cushings$Type)[il]
  text(cush[set, 1], cush[set, 2],
    labels = as.character(Cushings$Type[set]), col = 2 + il) }
par(cex = 1.5); partition.tree(cush.tr, add = T); par(cex = 1)

# 12.3 Non-parametric rules

Z <- knn(scale(cush[1:21, ], F, c(3.4, 5.7)),
  scale(cushT, F, c(3.4, 5.7)), tp)
cushplot(xp, yp, class.ind(Z))
Z <- knn(scale(cush[1:21, 1], F, c(3.4, 5.7)),
  scale(cushT, F, c(3.4, 5.7)), tp, k = 3)
cushplot(xp, yp, class.ind(Z))

# 12.4 Neural networks

pltnn <- function(main, ...) {
  plot(Cushings[,1], Cushings[,2], log="xy", type="n",
  xlab="Tetrahydrocortisone", ylab = "Pregnanetriol", main=main, ...)
  for(il in 1:4) {
    set <- Cushings$Type==levels(Cushings$Type)[il]
    text(Cushings[set, 1], Cushings[set, 2],
      as.character(Cushings$Type[set]), col = 2 + il) }
}

plt.bndry <- function(size=0, decay=0, ...)
{
  cush.nn <- nnet(cush, tpi, skip=T, softmax=T, size=size,
  decay=decay, maxit=1000)
  invisible(bl(predict(cush.nn, cushT), ...))
}

bl <- function(Z, ...)
{
  zp <- Z[,3] - pmax(Z[,2], Z[,1])
  contour(exp(xp), exp(yp), matrix(zp, np),
    add=T, levels=0, labex=0, ...)
  zp <- Z[,1] - pmax(Z[,3], Z[,2])
  contour(exp(xp), exp(yp), matrix(zp, np),
    add=T, levels=0, labex=0, ...)
}

cush <- cush[1:21,]; tpi <- class.ind(tp)
# functions pltnn and plt.bndry given in the scripts
par(mfrow = c(2, 2))

```

```

pltnn("Size = 2")
set.seed(1); plt.bndry(size = 2, col = 2)
set.seed(3); plt.bndry(size = 2, col = 3)
plt.bndry(size = 2, col = 4)

pltnn("Size = 2, lambda = 0.001")
set.seed(1); plt.bndry(size = 2, decay = 0.001, col = 2)
set.seed(2); plt.bndry(size = 2, decay = 0.001, col = 4)

pltnn("Size = 2, lambda = 0.01")
set.seed(1); plt.bndry(size = 2, decay = 0.01, col = 2)
set.seed(2); plt.bndry(size = 2, decay = 0.01, col = 4)

pltnn("Size = 5, 20 lambda = 0.01")
set.seed(2); plt.bndry(size = 5, decay = 0.01, col = 1)
set.seed(2); plt.bndry(size = 20, decay = 0.01, col = 2)

# functions pltnn and bl are in the scripts
pltnn("Many local maxima")
Z <- matrix(0, nrow(cushT), ncol(tpi))
for(iter in 1:20) {
  set.seed(iter)
  cush.nn <- nnet(cush, tpi, skip = T, softmax = T, size = 3,
    decay = 0.01, maxit = 1000, trace = F)
  Z <- Z + predict(cush.nn, cushT)
# In R replace @ by $ in next line.
  cat("final value", format(round(cush.nn$value,3)), "\n")
  bl(predict(cush.nn, cushT), col = 2, lwd = 0.5)
}
pltnn("Averaged")
bl(Z, lwd = 3)

# 12.5 Support vector machines

library(e1071)
crabs.svm <- svm(crabs$sp ~ ., data = lcrabs, cost = 100, gamma = 1)
table(true = crabs$sp, predicted = predict(crabs.svm, lcrabs))

svm(crabs$sp ~ ., data = lcrabs, cost = 100, gamma = 1, cross = 10)

# 12.6 Forensic glass example

set.seed(123)
# dump random partition from S-PLUS
rand <- c(9, 6, 7, 10, 8, 8, 2, 2, 10, 1, 5, 2, 3, 8, 6, 8, 2, 6, 4,
4, 6, 1, 3, 2, 5, 5, 5, 3, 1, 9, 10, 2, 8, 2, 1, 6, 2, 7, 7, 8, 4, 1,
9, 5, 5, 1, 4, 6, 8, 6, 5, 7, 9, 2, 1, 1, 10, 9, 7, 6, 4, 7, 4, 8, 9,
9, 1, 8, 9, 5, 3, 3, 4, 8, 8, 6, 6, 9, 3, 10, 3, 10, 6, 6, 5, 10, 10,
2, 10, 6, 1, 4, 7, 8, 9, 10, 7, 10, 8, 4, 6, 8, 9, 10, 1, 9, 10, 6, 8,
4, 10, 8, 2, 10, 2, 3, 10, 1, 5, 9, 4, 4, 8, 2, 7, 6, 4, 8, 10, 4, 8,
10, 6, 10, 4, 9, 4, 1, 6, 5, 3, 2, 4, 1, 3, 4, 8, 4, 3, 7, 2, 5, 4, 5,
10, 7, 4, 2, 6, 3, 2, 2, 8, 4, 10, 8, 10, 2, 10, 6, 5, 2, 3, 2, 6, 2,
7, 7, 8, 9, 7, 10, 8, 6, 7, 9, 7, 10, 3, 2, 7, 5, 6, 1, 3, 9, 7, 7, 1,
8, 7, 8, 8, 8, 10, 4, 5, 9, 4, 6, 9, 6, 10, 2)

con <- function(...)
{
  print(tab <- table(...))
  diag(tab) <- 0
  cat("error rate = ",
    round(100*sum(tab)/length(list(...)[[1]]), 2), "%\n")
  invisible()
}

```

```

CVtest <- function(fitfn, predfn, ...)
{
  res <- fgl$type
  for (i in sort(unique(rand))) {
    cat("fold ", i, "\n", sep = "")
    learn <- fitfn(rand != i, ...)
    res[rand == i] <- predfn(learn, rand == i)
  }
  res
}
res.multinom <- CVtest(
  function(x, ...) multinom(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ], type = "class"),
  maxit = 1000, trace = F )

con(true = fgl$type, predicted = res.multinom)

res.lda <- CVtest(
  function(x, ...) lda(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ])$class )
con(true = fgl$type, predicted = res.lda)

fgl0 <- fgl[ , -10] # drop type
{ res <- fgl$type
  for (i in sort(unique(rand))) {
    cat("fold ", i, "\n", sep = "")
    sub <- rand == i
    res[sub] <- knn(fgl0[!sub, ], fgl0[sub, ], fgl$type[!sub],
                  k = 1)
  }
  res } -> res.knn1
con(true = fgl$type, predicted = res.knn1)

res.lb <- knn(fgl0, fgl0, fgl$type, k = 3, prob = T, use.all = F)
table(attr(res.lb, "prob"))

library(rpart)
res.rpart <- CVtest(
  function(x, ...) {
    tr <- rpart(type ~ ., fgl[x, ], ...)
    cp <- tr$cptable
    r <- cp[ , 4] + cp[ , 5]
    rmin <- min(seq(along = r)[cp[ , 4] < min(r)])
    cp0 <- cp[rmin, 1]
    cat("size chosen was", cp[rmin, 2] + 1, "\n")
    prune(tr, cp = 1.01*cp0)
  },
  function(obj, x)
    predict(obj, fgl[x, ], type = "class"),
  cp = 0.001
)
con(true = fgl$type, predicted = res.rpart)

fgl1 <- fgl
fgl1[1:9] <- lapply(fgl[, 1:9], function(x)
  {r <- range(x); (x - r[1])/diff(r)})

CVnn2 <- function(formula, data,
  size = rep(6,2), lambda = c(0.001, 0.01),
  nreps = 1, nifold = 5, verbose = 99, ...)
{
  CVnn1 <- function(formula, data, nreps=1, ri, verbose, ...)
  {
    truth <- data[,deparse(formula[[2]])]
    res <- matrix(0, nrow(data), length(levels(truth)))
    if(verbose > 20) cat(" inner fold")
  }
}

```

```

for (i in sort(unique(ri))) {
  if(verbose > 20) cat(" ", i, sep="")
  for(rep in 1:nreps) {
    learn <- nnet(formula, data[ri !=i,], trace = F, ...)
    res[ri == i,] <- res[ri == i,] +
      predict(learn, data[ri == i,])
  }
  if(verbose > 20) cat("\n")
  sum(as.numeric(truth) != max.col(res/nreps))
}
truth <- data[,deparse(formula[[2]])]
res <- matrix(0, nrow(data), length(levels(truth)))
choice <- numeric(length(lambda))
for (i in sort(unique(rand))) {
  if(verbose > 0) cat("fold ", i, "\n", sep="")
  ri <- sample(nifold, sum(rand!=i), replace=T)
  for(j in seq(along=lambda)) {
    if(verbose > 10)
      cat(" size =", size[j], "decay =", lambda[j], "\n")
    choice[j] <- CVnn1(formula, data[rand != i,], nreps=nreps,
      ri=ri, size=size[j], decay=lambda[j],
      verbose=verbose, ...)
  }
  decay <- lambda[which.is.max(-choice)]
  csize <- size[which.is.max(-choice)]
  if(verbose > 5) cat(" #errors:", choice, " ") #
  if(verbose > 1) cat("chosen size = ", csize,
    " decay = ", decay, "\n", sep="")
  for(rep in 1:nreps) {
    learn <- nnet(formula, data[rand != i,], trace=F,
      size=csize, decay=decay, ...)
    res[rand == i,] <- res[rand == i,] +
      predict(learn, data[rand == i,])
  }
}
factor(levels(truth)[max.col(res/nreps)], levels = levels(truth))
}

if(F) { # only run this if you have time to wait
res.nn2 <- CVnn2(type ~ ., fgl1, skip = T, maxit = 500, nreps = 10)
con(true = fgl$type, predicted = res.nn2)
}

res.svm <- CVtest(
  function(x, ...) svm(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ]),
  cost = 100, gamma = 1 )
con(true = fgl$type, predicted = res.svm)

svm(type ~ ., data = fgl, cost = 100, gamma = 1, cross = 10)

cd0 <- lvqinit(fgl0, fgl$type, prior = rep(1, 6)/6, k = 3)
cd1 <- olvq1(fgl0, fgl$type, cd0)
con(true = fgl$type, predicted = lvqtest(cd1, fgl0))

CV.lvq <- function()
{
  res <- fgl$type
  for(i in sort(unique(rand))) {
    cat("doing fold", i, "\n")
    cd0 <- lvqinit(fgl0[rand != i,], fgl$type[rand != i,],
      prior = rep(1, 6)/6, k = 3)
    cd1 <- olvq1(fgl0[rand != i,], fgl$type[rand != i,], cd0)
    cd1 <- lvq3(fgl0[rand != i,], fgl$type[rand != i,],

```

```

        cdl, niter = 10000)
    res[rand == i] <- lvqtest(cdl, fgl0[rand == i, ])
  }
  res
}
con(true = fgl$type, predicted = CV.lvq())

# 12.7 Calibration plots

CVprobs <- function(fitfn, predfn, ...)
{
  res <- matrix(, 214, 6)
  for (i in sort(unique(rand))) {
    cat("fold ", i, "\n", sep = "")
    learn <- fitfn(rand != i, ...)
    res[rand == i, ] <- predfn(learn, rand == i)
  }
  res
}
probs.multinom <- CVprobs(
  function(x, ...) multinom(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ], type = "probs"),
  maxit = 1000, trace = F )

probs.yes <- as.vector(class.ind(fgl$type))
probs <- as.vector(probs.multinom)
par(pty = "s")
plot(c(0, 1), c(0, 1), type = "n", xlab = "predicted probability",
      ylab = "", xaxs = "i", yaxs = "i", las = 1)
rug(probs[probs.yes == 0], 0.02, side = 1, lwd = 0.5)
rug(probs[probs.yes == 1], 0.02, side = 3, lwd = 0.5)
abline(0, 1)
newp <- seq(0, 1, length = 100)
lines(newp, predict(loess(probs.yes ~ probs, span = 1), newp))

# End of ch12
#*- R -*

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 13 Survival Analysis

library(MASS)
options(echo=T, width=65, digits=5, height=9999)
options(contrasts=c("contr.treatment", "contr.poly"))
postscript("ch13.ps", width=8, height=6, pointsize=9)

library(survival)

# 13.1 Estimators of survivor curves

plot(survfit(Surv(time) ~ ag, data=leuk), lty = 2:3, col = 2:3)
legend(80, 0.8, c("ag absent", "ag present"), lty = 2:3, col = 2:3)

attach(gehan)
Surv(time, cens)
plot(log(time) ~ pair)
# product-limit estimators with Greenwood's formula for errors:
gehan.surv <- survfit(Surv(time, cens) ~ treat, data = gehan,
                     conf.type = "log-log")
summary(gehan.surv)
plot(gehan.surv, conf.int = T, lty = 3:2, log = T,
      xlab = "time of remission (weeks)", ylab = "survival")
lines(gehan.surv, lty = 3:2, lwd = 2, cex = 2)
legend(25, 0.1 , c("control", "6-MP"), lty = 2:3, lwd = 2)

```

```

detach()

survdifff(Surv(time, cens) ~ treat, data = gehan)
survdifff(Surv(time) ~ ag, data = leuk)

# 13.2 Parametric models

plot(gehan.surv, lty = 3:4, col = 2:3, fun = "cloglog",
     xlab = "time of remission (weeks)", ylab = "log H(t)")
legend(2, 0.5, c("control", "6-MP"), lty = 4:3, col = 3:2)

survreg(Surv(time) ~ ag*log(wbc), leuk, dist = "exponential")
summary(survreg(Surv(time) ~ ag + log(wbc), leuk, dist = "exponential"))
summary(survreg(Surv(time) ~ ag + log(wbc), leuk)) # Weibull
summary(survreg(Surv(time) ~ ag + log(wbc), leuk,
                dist="loglogistic"))
anova(survreg(Surv(time) ~ log(wbc), data = leuk),
      survreg(Surv(time) ~ ag + log(wbc), data = leuk))
summary(survreg(Surv(time) ~ strata(ag) + log(wbc), data=leuk))
leuk.wei <- survreg(Surv(time) ~ ag + log(wbc), leuk)
ntimes <- leuk$time * exp(-leuk.wei$linear.predictors)
plot(survfit(Surv(ntimes)), log = T)

survreg(Surv(time, cens) ~ factor(pair) + treat, gehan,
        dist = "exponential")
summary(survreg(Surv(time, cens) ~ treat, gehan, dist = "exponential"))
summary(survreg(Surv(time, cens) ~ treat, gehan))

plot(survfit(Surv(time, cens) ~ factor(temp), motors),
     conf.int = F)
motor.wei <- survreg(Surv(time, cens) ~ temp, motors)
summary(motor.wei)
unlist(predict(motor.wei, data.frame(temp=130), se.fit = T))

predict(motor.wei, data.frame(temp=130), type = "quantile",
        p = c(0.5, 0.1))
t1 <- predict(motor.wei, data.frame(temp=130),
              type = "uquantile", p = 0.5, se = T)
exp(c(LL=t1$fit - 2*t1$se, UL=t1$fit + 2*t1$se))
t1 <- predict(motor.wei, data.frame(temp=130),
              type = "uquantile", p = 0.1, se = T)
exp(c(LL=t1$fit - 2*t1$se, UL=t1$fit + 2*t1$se))

# summary(censorReg(censor(time, cens) ~ treat, gehan))

# 13.3 Cox proportional hazards model

attach(leuk)
leuk.cox <- coxph(Surv(time) ~ ag + log(wbc), data = leuk)
summary(leuk.cox)
update(leuk.cox, ~ . -ag)

(leuk.coxs <- coxph(Surv(time) ~ strata(ag) + log(wbc), data = leuk))

(leuk.coxs1 <- update(leuk.coxs, . ~ . + ag:log(wbc)))
plot(survfit(Surv(time) ~ ag), lty = 2:3, log = T)
lines(survfit(leuk.coxs), lty = 2:3, lwd = 3)
legend(80, 0.8, c("ag absent", "ag present"), lty = 2:3)
leuk.cox <- coxph(Surv(time) ~ ag, leuk)
detach()

gehan.cox <- coxph(Surv(time, cens) ~ treat, gehan, method = "exact")
summary(gehan.cox)

# The next fit is slow

```

```

coxph(Surv(time, cens) ~ treat + factor(pair), gehan,
      method = "exact")
1 - pchisq(45.5 - 16.2, 20)

(motor.cox <- coxph(Surv(time, cens) ~ temp, motors))
coxph(Surv(time, cens) ~ temp, motors, method = "breslow")
coxph(Surv(time, cens) ~ temp, motors, method = "exact")
plot( survfit(motor.cox, newdata=data.frame(temp=200),
          conf.type = "log-log" ) )
summary( survfit(motor.cox, newdata = data.frame(temp=130)) )

# 13.4 Further examples

# VA.temp <- as.data.frame(cancer.vet)
# dimnames(VA.temp)[[2]] <- c("treat", "cell", "stime",
#   "status", "Karn", "diag.time", "age", "therapy")
# attach(VA.temp)
# VA <- data.frame(stime, status, treat = factor(treat), age,
#   Karn, diag.time, cell = factor(cell), prior = factor(therapy))
# detach(VA.temp)
(VA.cox <- coxph(Surv(stime, status) ~ treat + age + Karn +
  diag.time + cell + prior, data = VA))

(VA.coxs <- coxph(Surv(stime, status) ~ treat + age + Karn +
  diag.time + strata(cell) + prior, data = VA))

par(mfrow=c(1,2), pty="s")
plot(survfit(VA.coxs), log = T, lty = 1:4, col = 2:5)
#legend(locator(1), c("squamous", "small", "adeno", "large"), lty = 1:4, col = 2:5)
plot(survfit(VA.coxs), fun = "cloglog", lty = 1:4, col = 2:5)
cKarn <- factor(cut(VA$Karn, 5))
VA.cox1 <- coxph(Surv(stime, status) ~ strata(cKarn) + cell, data = VA)
plot(survfit(VA.cox1), fun="cloglog")
VA.cox2 <- coxph(Surv(stime, status) ~ Karn + strata(cell), data = VA)
scatter.smooth(VA$Karn, residuals(VA.cox2))

VA.wei <- survreg(Surv(stime, status) ~ treat + age + Karn +
  diag.time + cell + prior, data = VA)
summary(VA.wei, cor = F)

VA.exp <- survreg(Surv(stime, status) ~ Karn + cell,
  data = VA, dist = "exponential")
summary(VA.exp, cor = F)

cox.zph(VA.coxs)

par(mfrow = c(3, 2), pty="m"); plot(cox.zph(VA.coxs))

VA2 <- VA ## needed because VA and stepAIC are both in MASS
VA2$Karnc <- VA2$Karn - 50
VA.coxc <- update(VA.cox, ~ . - Karn + Karnc, data=VA2)
VA.cox2 <- stepAIC(VA.coxc, ~ .^2)
VA.cox2$anova

(VA.cox3 <- update(VA.cox2, ~ treat/Karnc + prior*Karnc
  + treat:prior + cell/diag.time))

cox.zph(VA.cox3)

par(mfrow = c(2, 2))
plot(cox.zph(VA.cox3), var = c(1, 3, 7))
par(mfrow = c(1, 1))

#data(heart) # in package survival
coxph(Surv(start, stop, event) ~ transplant*

```

```

      (age + surgery + year), data = heart)
(stan <- coxph(Surv(start, stop, event) ~ transplant*year +
  age + surgery, data = heart))

stan1 <- coxph(Surv(start, stop, event) ~ strata(transplant) +
  year + year:transplant + age + surgery, heart)
par(mfrow=c(1,2), pty="s")
plot(survfit(stan1), conf.int = T, log = T, lty = c(1, 3), col = 2:3)
#legend(locator(1), c("before", "after"), lty = c(1, 3), col= 2:3)

attach(heart)
plot(year[transplant==0], residuals(stan1, collapse = id),
  xlab = "year", ylab = "martingale residual")
lines(lowess(year[transplant == 0],
  residuals(stan1, collapse = id)))
par(mfrow = c(1,1), pty = "m")
sresid <- resid(stan1, type = "dfbeta", collapse = id)
detach()
-100 * sresid %*% diag(1/stan1$coef)

# Survivor curve for the "average" subject
summary(survfit(stan))
# follow-up for two years
stan2 <- data.frame(start = c(0, 183), stop= c(183, 2*365),
  event = c(0, 0), year = c(4, 4), age = c(50, 50) - 48,
  surgery = c(1, 1), transplant = c(0, 1))
summary(survfit(stan, stan2, individual = T,
  conf.type = "log-log"))

# Aids analysis
time.depend.covar <- function(data) {
  id <- row.names(data); n <- length(id)
  events <- c(0, 10043, 11139, 12053) # julian days
  crit1 <- matrix(events[1:3], n, 3 ,byrow = T)
  crit2 <- matrix(events[2:4], n, 3, byrow = T)
  diag <- matrix(data$diag,n,3); death <- matrix(data$death,n,3)
  incid <- (diag < crit2) & (death >= crit1); incid <- t(incid)
  indr <- col(incid)[incid]; indc <- row(incid)[incid]
  ind <- cbind(indr, indc); idno <- id[indr]
  state <- data$state[indr]; T.categ <- data$T.categ[indr]
  age <- data$age[indr]; sex <- data$sex[indr]
  late <- indc - 1
  start <- t(pmax(crit1 - diag, 0))[incid]
  stop <- t(pmin(crit2, death + 0.9) - diag)[incid]
  status <- matrix(as.numeric(data$status),n,3)-1 # 0/1
  status[death > crit2] <- 0; status <- status[ind]
  levels(state) <- c("NSW", "Other", "QLD", "VIC")
  levels(T.categ) <- c("hs", "hsid", "id", "het", "haem",
    "blood", "mother", "other")
  levels(sex) <- c("F", "M")
  data.frame(idno, zid=factor(late), start, stop, status,
    state, T.categ, age, sex)
}
Aids3 <- time.depend.covar(Aids2)

attach(Aids3)
aids.cox <- coxph(Surv(start, stop, status)
  ~ zid + state + T.categ + sex + age, data = Aids3)
summary(aids.cox)

aids1.cox <- coxph(Surv(start, stop, status)
  ~ zid + strata(state) + T.categ + age, data = Aids3)
(aids1.surv <- survfit(aids1.cox))
plot(aids1.surv, mark.time = F, lty = 1:4, col = 2:5,
  xscale = 365.25/12, xlab = "months since diagnosis")
#legend(locator(1), levels(state), lty = 1:4, col = 2:5)

```

```

aids2.cox <- coxph(Surv(start, stop, status)
  ~ zid + state + strata(T.categ) + age, data = Aids3)
(aids2.surv <- survfit(aids2.cox))

par(mfrow = c(1, 2), pty="s")
plot(aids2.surv[1:4], mark.time = F, lty = 1:4, col = 2:5,
  xscale=365.25/12, xlab="months since diagnosis")
#legend(locator(1), levels(T.categ)[1:4], lty = 1:4, col = 2:5)

plot(aids2.surv[c(1, 5, 6, 8)], mark.time = F, lty = 1:4, col = 2:5,
  xscale=365.25/12, xlab="months since diagnosis")
#legend(locator(1), levels(T.categ)[c(1, 5, 6, 8)], lty = 1:4, col = 2:5)
par(mfrow=c(1,1), pty="m")

cases <- diff(c(0,idno)) != 0
aids.res <- residuals(aids.cox, collapse = idno)
scatter.smooth(age[cases], aids.res, xlab = "age",
  ylab="martingale residual")

age2 <- cut(age, c(-1, 15, 30, 40, 50, 60, 100))
c.age <- factor(as.numeric(age2), labels = c("0-15", "16-30",
  "31-40", "41-50", "51-60", "61+"))
table(c.age)
c.age <- relevel(c.age, "31-40")

summary(coxph(Surv(start, stop, status) ~ zid + state
  + T.categ + age + c.age, data = Aids3))
detach()

make.aidsp <- function(){
  cutoff <- 10043
  btime <- pmin(cutoff, Aids2$death) - pmin(cutoff, Aids2$diag)
  atime <- pmax(cutoff, Aids2$death) - pmax(cutoff, Aids2$diag)
  survtime <- btime + 0.5*atime
  status <- as.numeric(Aids2$status)
  data.frame(survtime, status = status - 1, state = Aids2$state,
    T.categ = Aids2$T.categ, age = Aids2$age, sex = Aids2$sex)
}
Aidsp <- make.aidsp()
aids.wei <- survreg(Surv(survtime + 0.9, status) ~ state
  + T.categ + sex + age, data = Aidsp)
summary(aids.wei, cor = F)

survreg(Surv(survtime + 0.9, status) ~ state + T.categ
  + age, data = Aidsp)

(aids.ps <- survreg(Surv(survtime + 0.9, status) ~ state
  + T.categ + pspline(age, df=6), data = Aidsp))
zz <- predict(aids.ps, data.frame(
  state = factor(rep("NSW", 83), levels = levels(Aidsp$state)),
  T.categ = factor(rep("hs", 83), levels = levels(Aidsp$T.categ)),
  age = 0:82), se = T, type = "linear")
plot(0:82, exp(zz$fit)/365.25, type = "l", ylim = c(0, 2),
  xlab = "age", ylab = "expected lifetime (years)")
lines(0:82, exp(zz$fit+1.96*zz$se.fit)/365.25, lty = 3, col = 2)
lines(0:82, exp(zz$fit-1.96*zz$se.fit)/365.25, lty = 3, col = 2)
rug(Aidsp$age+runif(length(Aidsp$age), -0.5, 0.5), ticksize = 0.015)

# End of ch13
#*- R -*

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 14 Time Series

```

```

library(MASS)
postscript(file="ch14.ps", width=8, height=6, pointsize=9)
options(width=65, digits=5, echo = T)

lh
deaths
#tspar(deaths)
tsp(deaths)
start(deaths)
end(deaths)
frequency(deaths)
cycle(deaths)
ts.plot(lh)
ts.plot(deaths, mdeaths, fdeaths,
        lty = c(1, 3, 4), xlab = "year", ylab = "deaths")

aggregate(deaths, 4, sum)
aggregate(deaths, 1, mean)

# 14.1 Second-order summaries

acf(lh)
acf(lh, type = "covariance")
acf(deaths)
acf(ts.union(mdeaths, fdeaths))

par(mfrow = c(2, 2))
spectrum(lh)
spectrum(deaths)

par(mfrow = c(2, 2))
spectrum(lh)
spectrum(lh, spans = 3)
spectrum(lh, spans = c(3, 3))
spectrum(lh, spans = c(3, 5))

spectrum(deaths)
spectrum(deaths, spans = c(3, 3))
spectrum(deaths, spans = c(3, 5))
spectrum(deaths, spans = c(5, 7))

par(mfrow = c(1, 2))
cpgram(lh)
cpgram(deaths)
par(mfrow = c(1, 1))

# 14.2 ARIMA models

# ts.sim <- arima.sim(list(order = c(1,1,0), ar = 0.7), n = 200)

acf(lh, type = "partial")
acf(deaths, type = "partial")

lh.ar1 <- ar(lh, F, 1)
cpgram(lh.ar1$resid, main = "AR(1) fit to lh")
lh.ar <- ar(lh, order.max = 9)
lh.ar$order
lh.ar$aic
cpgram(lh.ar$resid, main = "AR(3) fit to lh")

(lh.arima1 <- arima(lh, order = c(1,0,0)))
tsdiag(lh.arima1)
(lh.arima3 <- arima(lh, order = c(3,0,0)))

```

```

tsdiag(lh.arima3)
(lh.arima11 <- arima(lh, order = c(1,0,1)))

lh.fore <- predict(lh.arima3, 12)
ts.plot(lh, lh.fore$pred, lh.fore$pred + 2*lh.fore$se,
        lh.fore$pred - 2*lh.fore$se, lty = c(1,2,3,3))

# 14.3 Seasonality

deaths.stl <- stl(deaths, "periodic")
dsd <- deaths.stl$time.series[, "trend"] +
      deaths.stl$time.series[, "remainder"]
#ts.plot(deaths, deaths.stl$sea, deaths.stl$rem)
ts.plot(deaths, deaths.stl$time.series[, "seasonal"], dsd,
        gpars = list(lty = c(1, 3, 2)))

par(mfrow = c(2, 3))
#dsd <- deaths.stl$rem
ts.plot(dsd)

acf(dsd)
acf(dsd, type = "partial")
spectrum(dsd, span = c(3, 3))
cpgram(dsd)
dsd.ar <- ar(dsd)
dsd.ar$order
dsd.ar$aic
dsd.ar$ar
cpgram(dsd.ar$resid, main = "AR(1) residuals")
par(mfrow = c(1, 1))

deaths.diff <- diff(deaths, 12)
acf(deaths.diff, 30)
acf(deaths.diff, 30, type = "partial")
ar(deaths.diff)
# this suggests the seasonal effect is still present.
(deaths.arima1 <- arima(deaths, order = c(2,0,0),
                      seasonal = list(order = c(0,1,0), period = 12)) )
tsdiag(deaths.arima1, gof.lag = 30)
# suggests need a seasonal AR term
(deaths.arima2 <- arima(deaths, order = c(2,0,0),
                      list(order = c(1,0,0), period = 12)) )
tsdiag(deaths.arima2, gof.lag = 30)
cpgram(deaths.arima2$resid)
(deaths.arima3 <- arima(deaths, order = c(2,0,0),
                      list(order = c(1,1,0), period = 12)) )
tsdiag(deaths.arima3, gof.lag = 30)

par(mfrow = c(3, 1))
nott <- window(nottem, end = c(1936, 12))
ts.plot(nott)
nott.stl <- stl(nott, "period")
ts.plot(nott.stl$time.series[, c("remainder", "seasonal")],
        gpars = list(ylim = c(-15, 15), lty = c(1, 3)))
nott.stl <- stl(nott, 5)
ts.plot(nott.stl$time.series[, c("remainder", "seasonal")],
        ylim = c(-15, 15), lty = c(1, 3))

par(mfrow = c(1, 1))
boxplot(split(nott, cycle(nott)), names = month.abb)

nott[110] <- 35
nott.stl <- stl(nott, "period")
nott1 <- nott.stl$time.series[, "trend"] + nott.stl$time.series[, "remainder"]
acf(nott1)

```

```

acf(nott1, type = "partial")
cpgram(nott1)
ar(nott1)$aic
plot(0:23, ar(nott1)$aic, xlab = "order", ylab = "AIC",
     main = "AIC for AR(p)")
(nott1.ar1 <- arima(nott1, order = c(1,0,0)))
nott1.fore <- predict(nott1.ar1, 36)
nott1.fore$pred <- nott1.fore$pred +
  as.vector(nott.stl$time.series[1:36, "seasonal"])
ts.plot(window(nottem, 1937), nott1.fore$pred,
        nott1.fore$pred+2*nott1.fore$se,
        nott1.fore$pred-2*nott1.fore$se, lty = c(3, 1, 2, 2))
title("via Seasonal Decomposition")

```

```

acf(diff(nott,12), 30)
acf(diff(nott,12), 30, type = "partial")
cpgram(diff(nott, 12))
(nott.arima1 <- arima(nott, order = c(1,0,0),
  list(order = c(2,1,0), period = 12)) )
tsdiag(nott.arima1, gof.lag = 30)
(nott.arima2 <- arima(nott, order = c(0,0,2),
  list(order = c(0,1,2), period = 12)) )
tsdiag(nott.arima2, gof.lag = 30)
(nott.arima3 <- arima(nott, order = c(1,0,0),
  list(order = c(0,1,2), period = 12)) )
tsdiag(nott.arima3, gof.lag = 30)

```

```

nott.fore <- predict(nott.arima3, 36)
ts.plot(window(nottem, 1937), nott.fore$pred,
        nott.fore$pred+2*nott.fore$se,
        nott.fore$pred-2*nott.fore$se, lty = c(3, 1, 2, 2))
title("via Seasonal ARIMA model")

```

14.6 Regression with autocorrelated errors

```

attach(beav1)
beav1$hours <- 24*(day-346) + trunc(time/100) + (time%%100)/60
detach()
attach(beav2)
beav2$hours <- 24*(day-307) + trunc(time/100) + (time%%100)/60
detach()
par(mfrow = c(2, 2))
plot(beav1$hours, beav1$temp, type = "l", xlab = "time",
     ylab = "temperature", main = "Beaver 1")
usr <- par("usr"); usr[3:4] <- c(-0.2, 8); par(usr = usr)
lines(beav1$hours, beav1$activ, type = "s", lty = 2)
plot(beav2$hours, beav2$temp, type = "l", xlab = "time",
     ylab = "temperature", main = "Beaver 2")
usr <- par("usr"); usr[3:4] <- c(-0.2, 8); par(usr = usr)
lines(beav2$hours, beav2$activ, type = "s", lty = 2)

```

```

attach(beav2)
temp2 <- ts(temp, start = 8+2/3, frequency = 6)
activ2 <- ts(activ, start = 8+2/3, frequency = 6)
acf(temp2[activ2 == 0])
acf(temp2[activ2 == 1]) # also look at PACFs
acf(temp2[activ2 == 0], type = "partial")
acf(temp2[activ2 == 1], type = "partial")
ar(temp2[activ2 == 0])
ar(temp2[activ2 == 1])
par(mfrow = c(1, 1))
detach()
rm(temp2, activ2)

```

```

library(nlme)

```

```

beav2.gls <- gls(temp ~ activ, data = beav2,
                corr = corAR1(0.8), method = "ML")
summary(beav2.gls)
summary(update(beav2.gls, subset = 6:100))

arima(beav2$temp, c(1,0,0), xreg = beav2$activ)

attach(beav1)
temp1 <- ts(c(temp[1:82], NA, temp[83:114]), start = 9.5, frequency = 6)
activ1 <- ts(c(activ[1:82], NA, activ[83:114]), start = 9.5, frequency = 6)
acf(temp1[1:53])
acf(temp1[1:53], type = "partial")
ar(temp1[1:53])

act <- c(rep(0, 10), activ1)
beav1b <- data.frame(Time = time(temp1), temp = as.vector(temp1),
                    act = act[11:125], act1 = act[10:124],
                    act2 = act[9:123], act3 = act[8:122])
detach()
rm(temp1, activ1)

summary(gls(temp ~ act + act1 + act2 + act3,
            data = beav1b, na.action = na.omit,
            corr = corCAR1(0.82^6, ~Time), method = "ML"))

arima(beav1b$temp, c(1, 0, 0), xreg = beav1b[, 3:6])

# 14.6 Models for financial time series

plot(SP500, type = "l", xlab = "", ylab = "returns (%)", xaxt = "n", las = 1)
axis(1, at = c(0, 254, 507, 761, 1014, 1266, 1518, 1772, 2025, 2277,
              2529, 2781), lab = 1990:2001)

plot(density(SP500, width = "sj", n = 256), type = "l", xlab = "", ylab = "")

par(pty = "s")
qqnorm(SP500)
qqline(SP500)
if(F) {
module(garch)
summary(garch(SP500 ~ 1, ~garch(1,1)))

fit <- garch(SP500 ~ 1, ~garch(1,1), cond.dist = "t")
summary(fit)
plot(fit)

summary(garch(SP500 ~ 1, ~egarch(1,1), cond.dist = "t", leverage = T))
}

library(tseries)
summary(garch(x = SP500 - median(SP500), order = c(1, 1)))

# End of ch14
#*- R -*

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 15 Spatial Statistics

library(MASS)
postscript(file="ch15.ps", width=8, height=8, pointsize=9)
options(echo = TRUE, width=65, digits=5)

library(spacial)

```

15.1 Spatial interpolation and smoothing

```
par(mfrow=c(2,2), pty = "s")
topo.ls <- surf.ls(2, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
eqsplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("Degree=2")
topo.ls <- surf.ls(3, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
eqsplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("Degree=3")
topo.ls <- surf.ls(4, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
eqsplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("Degree=4")
topo.ls <- surf.ls(6, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
eqsplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("Degree=6")

library(lattice)
topo.ls <- surf.ls(4, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
trsurf[c("x", "y")] <- expand.grid(x=trsurf$x, y=trsurf$y)
plt1 <- levelplot(z ~ x * y, trsurf, aspect=1,
  at = seq(650, 1000, 10), xlab = "", ylab = "")
plt2 <- wireframe(z ~ x * y, trsurf, aspect=c(1, 0.5),
  screen = list(z = -30, x = -60))
print(plt1, position = c(0, 0, 0.5, 1), more=T)
print(plt2, position = c(0.45, 0, 1, 1))

par(mfcol = c(2, 2), pty = "s")
topo.loess <- loess(z ~ x * y, topo, degree = 2, span = 0.25,
  normalize = F)
topo.mar <- list(x = seq(0, 6.5, 0.1), y = seq(0, 6.5, 0.1))
topo.lo <- predict(topo.loess, expand.grid(topo.mar), se = T)
eqsplot(topo.mar, xlab = "fit", ylab = "", type = "n")
contour(topo.mar$x, topo.mar$y, topo.lo$fit,
  levels = seq(700, 1000, 25), add = T)
points(topo)
eqsplot(topo.mar, xlab = "standard error", ylab = "", type = "n")
contour(topo.mar$x, topo.mar$y, topo.lo$se.fit,
  levels = seq(5, 25, 5), add = T)
title("Loess degree = 2")
points(topo)

topo.loess <- loess(z ~ x * y, topo, degree = 1, span = 0.25, normalize = F)
topo.lo <- predict(topo.loess, expand.grid(topo.mar), se=T)
eqsplot(topo.mar, xlab = "fit", ylab = "", type = "n")
contour(topo.mar$x, topo.mar$y, topo.lo$fit, levels = seq(700, 1000, 25),
  add = T)
points(topo)
eqsplot(topo.mar, xlab = "standard error", ylab = "", type = "n")
contour(topo.mar$x, topo.mar$y, topo.lo$se.fit, levels = seq(5, 25, 5),
  add = T)
title("Loess degree = 1")
points(topo)
```

```

library(akima)
par(mfrow = c(1, 2), pty= "s")
topo.int <- interp.old(topo$x, topo$y, topo$z)
eqscplot(topo.int, xlab = "interp default", ylab = "", type = "n")
contour(topo.int, levels = seq(600, 1000, 25), add = T)
points(topo)
topo.mar <- list(x = seq(0, 6.5, 0.1), y = seq(0, 6.5, 0.1))
topo.int2 <- interp.old(topo$x, topo$y, topo$z, topo.mar$x, topo.mar$y,
                        ncp = 4, extrap = T)
eqscplot(topo.int2, xlab = "interp", ylab = "", type = "n")
contour(topo.int2, levels = seq(600, 1000, 25), add = T)
points(topo)

```

15.2 Kriging

```

par(mfrow = c(2, 2), pty = "s")
topo.ls <- surf.ls(2, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
eqscplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("LS trend surface")

```

```

topo.gls <- surf.gls(2, expcov, topo, d = 0.7)
trsurf <- trmat(topo.gls, 0, 6.5, 0, 6.5, 30)
eqscplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("GLS trend surface")

```

```

prsurf <- prmat(topo.gls, 0, 6.5, 0, 6.5, 50)
eqscplot(prsurf, , xlab = "", ylab = "", type = "n")
contour(prsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("Kriging prediction")
sesurf <- semat(topo.gls, 0, 6.5, 0, 6.5, 30)
eqscplot(sesurf, , xlab = "", ylab = "", type = "n")
contour(sesurf, levels = c(20, 25), add = T)
points(topo)
title("Kriging s.e.")

```

```

par(mfrow = c(2, 2), pty = "m")
topo.kr <- surf.ls(2, topo)
correlogram(topo.kr, 25)
d <- seq(0, 7, 0.1)
lines(d, expcov(d, 0.7))
variogram(topo.kr, 25)

```

```

## left panel of Figure 15.7
topo.kr <- surf.gls(2, expcov, topo, d=0.7)
correlogram(topo.kr, 25)
lines(d, expcov(d, 0.7))
lines(d, gaucov(d, 1.0, 0.3), lty = 3) # try nugget effect

```

```

## right panel
topo.kr <- surf.ls(0, topo)
correlogram(topo.kr, 25)
lines(d, gaucov(d, 2, 0.05))

```

```

par(mfrow = c(2, 2), pty = "s")
## top row of Figure 15.8
topo.kr <- surf.gls(2, gaucov, topo, d = 1, alph = 0.3)
prsurf <- prmat(topo.kr, 0, 6.5, 0, 6.5, 50)
eqscplot(prsurf, , xlab = "fit", ylab = "", type = "n")

```

```

contour(prsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
sesurf <- semat(topo.kr, 0, 6.5, 0, 6.5, 25)
eqscplot(sesurf, , xlab = "standard error", ylab = "", type = "n")
contour(sesurf, levels = c(15, 20, 25), add = T)
points(topo)

## bottom row of Figure 15.8
topo.kr <- surf.gls(0, gaucov, topo, d = 2, alph = 0.05,
  nx = 10000)
prsurf <- prmat(topo.kr, 0, 6.5, 0, 6.5, 50)
eqscplot(prsurf, , xlab = "fit", ylab = "", type = "n")
contour(prsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
sesurf <- semat(topo.kr, 0, 6.5, 0, 6.5, 25)
eqscplot(sesurf, , xlab = "standard error", ylab = "", type = "n")
contour(sesurf, levels = c(15, 20, 25), add = T)
points(topo)

# 15.3 Point process analysis

library(spatial)
pines <- ppinit("pines.dat")
par(mfrow = c(2, 2), pty = "s")
plot(pines, xlim = c(0, 10), ylim = c(0, 10),
  xlab = "", ylab = "", xaxs = "i", yaxs = "i")
plot(Kfn(pines,5), type = "s", xlab = "distance", ylab = "L(t)")
lims <- Kenvl(5, 100, Psim(72))
lines(lims$x, lims$l, lty = 2)
lines(lims$x, lims$u, lty = 2)

ppregion(pines)
plot(Kfn(pines, 1.5), type = "s",
  xlab = "distance", ylab = "L(t)")
lims <- Kenvl(1.5, 100, Strauss(72, 0.2, 0.7))
lines(lims$x, lims$a, lty = 2)
lines(lims$x, lims$l, lty = 2)
lines(lims$x, lims$u, lty = 2)
pplik(pines, 0.7)
lines(Kaver(1.5, 100, Strauss(72, 0.15, 0.7)), lty = 3)

# End of ch15
#-*- R -*-

## Script from Fourth Edition of `Modern Applied Statistics with S'

# Chapter 16 Optimization and Maximum Likelihood Estimation

library(MASS)
postscript(file="ch16.ps", width=8, height=8, pointsize=9)
options(echo = T, width=65, digits=5)

# 16.3 General optimization

attach(geyser)
truehist(waiting, xlim = c(35, 110), ymax = 0.04, h = 5)
wait.dns <- density(waiting, n = 512, width = "SJ")
lines(wait.dns, lty = 2)

lmix2 <- deriv3(
  ~ -log(p*dnorm((x-u1)/s1)/s1 + (1-p)*dnorm((x-u2)/s2)/s2),
  c("p", "u1", "s1", "u2", "s2"),
  function(x, p, u1, s1, u2, s2) NULL)

```

```

(p0 <- c(p = mean(waiting < 70), u1 = 50, s1 = 5, u2 = 80, s2 = 5))

## using optim

mix.obj <- function(p, x)
{
  e <- p[1] * dnorm((x - p[2])/p[3])/p[3] +
    (1 - p[1]) * dnorm((x - p[4])/p[5])/p[5]
  if(any(e <= 0)) Inf else -sum(log(e))
}
optim(p0, mix.obj, x = waiting)$par # Nelder-Mead

optim(p0, mix.obj, x = waiting, method = "BFGS",
      control = list(parscale= c(0.1, rep(1, 4))))$par

# with derivatives
lmix2a <- deriv(
  ~ -log(p*dnorm((x-u1)/s1)/s1 + (1-p)*dnorm((x-u2)/s2)/s2),
  c("p", "u1", "s1", "u2", "s2"),
  function(x, p, u1, s1, u2, s2) NULL)
mix.gr <- function(p, x) {
  u1 <- p[2]; s1 <- p[3]; u2 <- p[4]; s2 <- p[5]; p <- p[1]
  colSums(attr(lmix2a(x, p, u1, s1, u2, s2), "gradient")) }

optim(p0, mix.obj, mix.gr, x = waiting, method = "BFGS",
      control = list(parscale= c(0.1, rep(1, 4))))$par

mix.nl0 <- optim(p0, mix.obj, mix.gr, method = "L-BFGS-B", hessian = T,
               lower = c(0, -Inf, 0, -Inf, 0),
               upper = c(1, rep(Inf, 4)), x = waiting)
rbind(est = mix.nl0$par, se = sqrt(diag(solve(mix.nl0$hessian))))

dmix2 <- function(x, p, u1, s1, u2, s2)
  p * dnorm(x, u1, s1) + (1-p) * dnorm(x, u2, s2)
attach(as.list(mix.nl0$par))
wait.fdns <- list(x = wait.dns$x,
                 y = dmix2(wait.dns$x, p, u1, s1, u2, s2))
lines(wait.fdns)
par(usr = c(0, 1, 0, 1))
legend(0.1, 0.9, c("Normal mixture", "Nonparametric"),
      lty = c(1, 2), bty = "n")

pmix2 <- deriv(~ p*pnorm((x-u1)/s1) + (1-p)*pnorm((x-u2)/s2),
              "x", function(x, p, u1, s1, u2, s2) {})
pr0 <- (seq(along = waiting) - 0.5)/length(waiting)
x0 <- x1 <- as.vector(sort(waiting)) ; del <- 1; i <- 0
while((i <- 1 + 1) < 10 && abs(del) > 0.0005) {
  pr <- pmix2(x0, p, u1, s1, u2, s2)
  del <- (pr - pr0)/attr(pr, "gradient")
  x0 <- x0 - 0.5*del
  cat(format(del <- max(abs(del))), "\n")
}
detach()
par(pty = "s")
plot(x0, x1, xlim = range(x0, x1), ylim = range(x0, x1),
     xlab = "Model quantiles", ylab = "Waiting time")
abline(0, 1)
par(pty = "m")

mix1.obj <- function(p, x, y)
{
  q <- exp(p[1] + p[2]*y)
  q <- q/(1 + q)
  e <- q * dnorm((x - p[3])/p[4])/p[4] +

```

```

      (1 - q) * dnorm((x - p[5])/p[6])/p[6]
    if(any(e <= 0)) Inf else -sum(log(e))
  }
p1 <- mix.nl0$par; tmp <- as.vector(p1[1])
p2 <- c(a = log(tmp/(1-tmp)), b = 0, p1[-1])
mix.nll <- optim(p2, mixl.obj, method = "L-BFGS-B",
               lower = c(-Inf, -Inf, -Inf, 0, -Inf, 0),
               upper = rep(Inf, 6), hessian = T,
               x = waiting[-1], y = duration[-299])
rbind(est = mix.nll$par, se = sqrt(diag(solve(mix.nll$hessian))))

```

```

if(!exists("bwt")) {
  attach(birthwt)
  race <- factor(race, labels=c("white", "black", "other"))
  ptd <- factor(ptl > 0)
  ftv <- factor(ftv); levels(ftv)[-1:2] <- "2+"
  bwt <- data.frame(low=factor(low), age, lwt, race,
                  smoke=(smoke>0), ptd, ht=(ht>0), ui=(ui>0), ftv)
  detach(); rm(race, ptd, ftv)
}

```

```

logitreg <- function(x, y, wt = rep(1, length(y)),
                    intercept = T, start = rep(0, p), ...)
{
  fmin <- function(beta, X, y, w) {
    p <- plogis(X %*% beta)
    -sum(2 * w * ifelse(y, log(p), log(1-p)))
  }
  gmin <- function(beta, X, y, w) {
    eta <- X %*% beta; p <- plogis(eta)
    -2 * matrix(w *dlogis(eta) * ifelse(y, 1/p, -1/(1-p)), 1) %*% X
  }
  if(is.null(dim(x))) dim(x) <- c(length(x), 1)
  dn <- dimnames(x)[[2]]
  if(!length(dn)) dn <- paste("Var", 1:ncol(x), sep="")
  p <- ncol(x) + intercept
  if(intercept) {x <- cbind(1, x); dn <- c("(Intercept)", dn)}
  if(is.factor(y)) y <- (unclass(y) != 1)
  fit <- optim(start, fmin, gmin, X = x, y = y, w = wt,
              method = "BFGS", ...)
  names(fit$par) <- dn
  cat("\nCoefficients:\n"); print(fit$par)
  # R: use fit$value and fit$convergence
  cat("\nResidual Deviance:", format(fit$value), "\n")
  if(fit$convergence > 0)
    cat("\nConvergence code:", fit$convergence, "\n")
  invisible(fit)
}

```

```

options(contrasts = c("contr.treatment", "contr.poly"))
X <- model.matrix(terms(low ~ ., data=bwt), data = bwt)[, -1]
logitreg(X, bwt$low)

```

```

AIDSfit <- function(y, z, start=rep(mean(y), ncol(z)), ...)
{
  deviance <- function(beta, y, z) {
    mu <- z %*% beta
    2 * sum(mu - y - y*log(mu/y)) }
  grad <- function(beta, y, z) {
    mu <- z %*% beta
    2 * t(1 - y/mu) %*% z }
  optim(start, deviance, grad, lower = 0, y = y, z = z,
        method = "L-BFGS-B", ...)
}

```

```
Y <- scan()
12 14 33 50 67 74 123 141 165 204 253 246 240

library(nnet) # for class.ind
s <- seq(0, 13.999, 0.01); tint <- 1:14
X <- expand.grid(s, tint)
Z <- matrix(pweibull(pmax(X[,2] - X[,1],0), 2.5, 10),length(s))
Z <- Z[,2:14] - Z[,1:13]
Z <- t(Z) %*% class.ind(factor(floor(s/2))) * 0.01
round(AIDSfit(Y, Z)$par)
rm(s, X, Y, Z)

# End of ch16
```