Statistics Complements to

Modern Applied Statistics with S-Plus

Third edition

by

W. N. Venables and B. D. Ripley

Springer (1999). ISBN 0-387-98825-4

27 May 2000

These complements have been produced to supplement the third edition of MASS. They will be updated from time to time. The definitive source is http://www.stats.ox.ac.uk/pub/MASS3/.

Selectable links are in this colour.
Selectable URLs are in this colour.

Introduction

These complements are made available on-line to supplement the book making use of extensions to S-PLUS in user-contributed library sections.

The general convention is that material here should be thought of as following the material in the chapter in the book, so that new sections are numbered following the last section of the chapter, and figures and equations here are numbered following on from those in the book.

All the libraries mentioned are available for Unix and for Windows. Compiled versions for Windows (for S-PLUS 3.2, 3.3, 4.0, 4.5 and 2000) are available from either of the URLs

```
http://www.stats.ox.ac.uk/pub/SWin/
http://lib.stat.cmu.edu/DOS/S/SWin/
```

Compiled versions of most for S-PLUS 6.0 for Windows are available via

```
http://www.stats.ox.ac.uk/pub/MASS3/Winlibs
```

Most of the Unix sources are available at

```
http://lib.stat.cmu.edu/S/
```

and more specific information is given for the exceptions where these are introduced. In most cases some modifications are needed for use with S-PLUS 5.x and 6.0: try the migration tools.

Contents

Chapter 5

Distributions and Data Summaries

5.6 Density estimation

Spline fitting to log-densities

There are several closely-related proposals[1] to use a univariate density estimator of the form

$$f(y) = \exp g(y; \theta) \tag{5.7}$$

for a parametric family $g(\cdot; \theta)$ of smooth functions, most often splines. The fit criterion is maximum likelihood, possibly with a smoothness penalty. The advantages of (5.7) is that it automatically provides a non-negative density estimate, and that it may be more natural to consider 'smoothness' on a relative rather than absolute scale. It is necessary to ensure that the estimated density has unit mass, and this is most conveniently done by taking

$$f(y) = \exp g(y; \theta) / \int \exp g(y; \theta)\, \mathrm{d}y \tag{5.8}$$

The library `logspline` [2] by Charles Kooperberg implements one variant[3] on this theme by Kooperberg & Stone (1992). This uses a cubic spline for g in (5.8), with smoothness controlled not by a penalty (as in smoothing splines) but by the number of knots selected. There is an AIC-like penalty; the number of the knots is chosen to maximize

$$\sum_{i=1}^{n} g(y_i; \widehat{\theta}) - n \log \int \exp g(y; \widehat{\theta})\, \mathrm{d}y - a \times \text{number of parameters} \tag{5.9}$$

The default value of a is $\log n$ (sometimes known as BIC) but this can be set as an argument of `logspline.fit`. A Newton method is used to maximize the log-likelihood given the knot positions. The initial knots are selected at quantiles of the data and then deleted one at a time using the Wald criterion for significance. Finally, (5.9) is used to choose one of the knot sequences considered.

We first try out our two running examples:

[1] see Simonoff (1996, pp. 67–70, 90–92) for others.

[2] `logsplin` on Windows.

[3] although a later version described in Stone *et al.* (1997) has been long promised to replace it, it has not appeared.

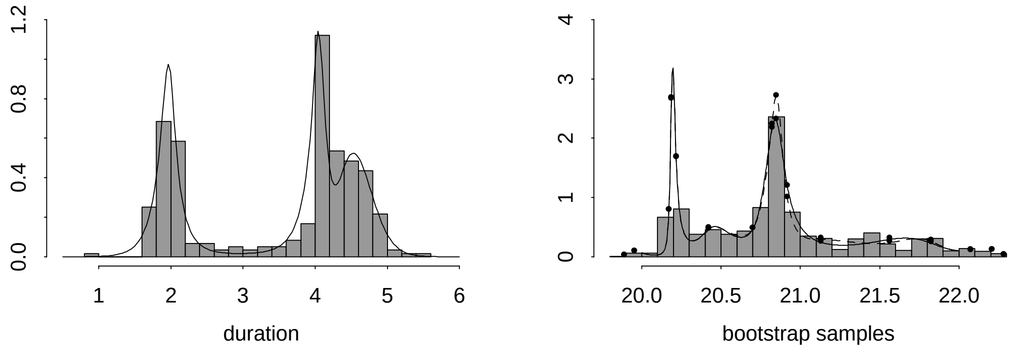

Figure 5.15: Histograms and logspline density plots of (left) the Old Faithful eruptions data and (right) bootstrap samples of the median of the `galaxies` dataset. Compare with Figures 5.9 (on page 136) and Figure 8.5 (page 262).

```
library(logspline) # logsplin on Windows
attach(geyser)
geyser.ls <- logspline.fit(duration, lbound=0)
x <- seq(0.5, 6, len=200)
truehist(duration, nbins=15, xlim=c(0.5,6), ymax=1.2)
lines(x, dlogspline(x, geyser.ls))
detach()

truehist(tperm, xlab="diff")
tperm.ls <- logspline.fit(tperm)
x <- seq(-5, 5, len=200)
lines(x, dlogspline(x, tperm.ls))

sres <- c(sort(tperm), 5); yres <- (0:1024)/1024
plot(sres, yres, type="S", xlab="diff", ylab="cdf")
lines(x, plogspline(x, tperm.ls))

par(pty="s")
x <- c(0.0005, seq(0.001, 0.999, 0.001), 0.9995)
plot( qt(x, 9), qlogspline(x, tperm.ls),
      xlab="Quantiles of t on 9 df", ylab="Fitted quantiles",
      type="l", xlim=c(-5, 5), ylim=c(-5, 5))
points( qt(ppoints(tperm), 9), sort(tperm) )
```

The functions `dlogspline`, `plogspline` and `qlogspline` compute the density, CDF and quantiles of the fitted density, so the final plot is a QQ-plot of the data and the fitted density against the t_9 density. The final plot shows that the t_9 density is a better fit in the tails; the logspline density estimate always has exponential tails. (The function `logspline.plot` will make a simple plot of the density, CDF or hazard estimate.)

We can also explore density plots of the bootstrapped median values from page 142 (which we recall actually has a discrete distribution).

```
truehist(res, nbins=nclass.FD(res), ymax=4,
```

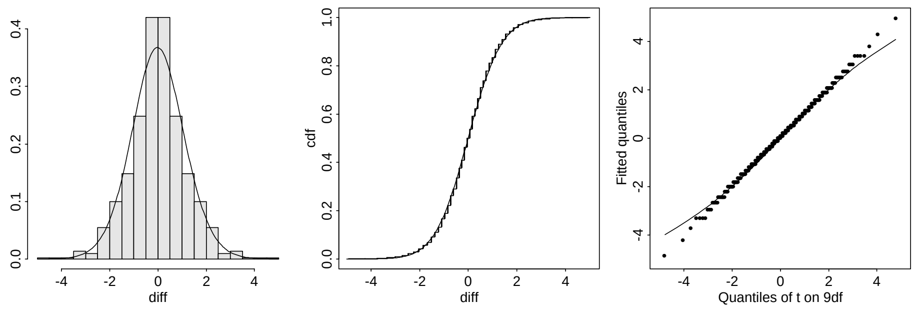

Figure 5.16: Plots of the logspline density estimate of the permutation dataset `tperm`. The three panels show the histogram with superimposed density estimate, the empirical and fitted CDFs and QQ–plots of the data and the fitted density against the conventional t_9 distribution.

```
          xlab="bootstrap samples")
x <- seq(20, 22, length=500)
res.ls <- logspline.fit(res)
lines(x, dlogspline(x, res.ls))
points(res.ls$knots, dlogspline(res.ls$knots, res.ls))
res.ls <- logspline.fit(res, penalty=2)
lines(x, dlogspline(x, res.ls), lty=3)
points(res.ls$knots, dlogspline(res.ls$knots, res.ls))
```

Changing the penalty a to the AIC value of 2 has a small effect. The dots show where the knots have been placed. (The function `logspline.summary` shows details of the selection of the number of knots.)

The results for the `galaxies` data are also instructive (Figure 5.17).

```
x <- seq(8000, 35000, 200)
plot(x, dlogspline(x, logspline.fit(galaxies)), type="l",
     xlab="velocity of galaxy", ylab="density")
lines(density(galaxies, n=200, window="gaussian",
      width=width.SJ(galaxies)), lty=3)
```

Maximum-likelihood methods and hence `logspline.fit` can easily handle censored data (see page 49).

Local polynomial density estimation

The local regression approach of `loess` can be extended to local likelihood estimation and hence used for density estimation. One implementation is the function `locpoly` in library `KernSmooth`[4]. This uses a fine grid of bins on the x axis and applies a local polynomial smoother to the counts of the binned data.

[4] `ksmooth` on Windows. The current Unix sources are at
http://www.biostat.harvard.edu/~mwand

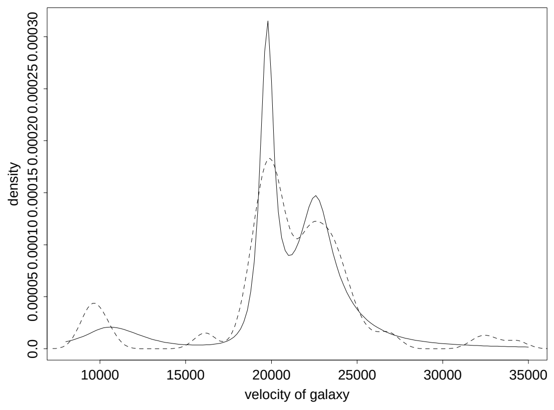

Figure 5.17: Logspline (solid line) and kernel density (dashed) estimates for the `galaxies` data. The bandwidth of the kernel estimate was chosen by `width.SJ`.

Loader (1997) introduces his implementation in the `locfit` package; the theory for density estimation is in Loader (1996). The default is that $\log f(y)$ is fitted by a quadratic polynomial: to estimate the density at x we maximize

$$\sum_{i=1}^{n} K\left(\tfrac{y_i-x}{b}\right) g(y_i;\theta(x)) - n \log \int K\left(\tfrac{y-x}{b}\right) \exp g(y;\theta(x))\,\mathrm{d}y$$

that is, (5.9) localized near x, and with a quadratic polynomial model for $g(y;\theta)$. The function K is controlled by the argument `kern`; by default it is the tricubic function used by `loess`; `kern="gauss"` gives a Gaussian kernel with bandwidth 2.5 times[5] the standard deviation. The documentation with the package is sparse: the Web site

http://cm.bell-labs.com/stat/project/locfit

has the sources but the help there refers to the much older R version, and the various on-line documents have been removed. Unfortunately many of our examples no longer work in the current (June 1999) release of `locfit`, so our remaining examples should be seen as indicative only.

We can use `locfit` on the duration data by

```
library(locfit)
geyser.lf <- locfit(~ duration, data=geyser, flim=c(0.5, 6))
plot(geyser.lf, get.data=T, mpv=200, ylim=c(0,1))
```

where `get.data` adds the rug and `mpv` evaluates at 200 points to ensure a smooth curve. (The `flim` parameter asks for a fit to cover that range of x values.)

As for `loess` we have to choose how much to localize, that is to choose the bandwidth h, possibly as a function of x. This is done in `locfit` by choosing the larger of a nearest-neighbour-based estimate and a fixed bandwidth. Loader (1997) suggests

[5] `density` and hence our account in Chapter 5 uses $4\times$.

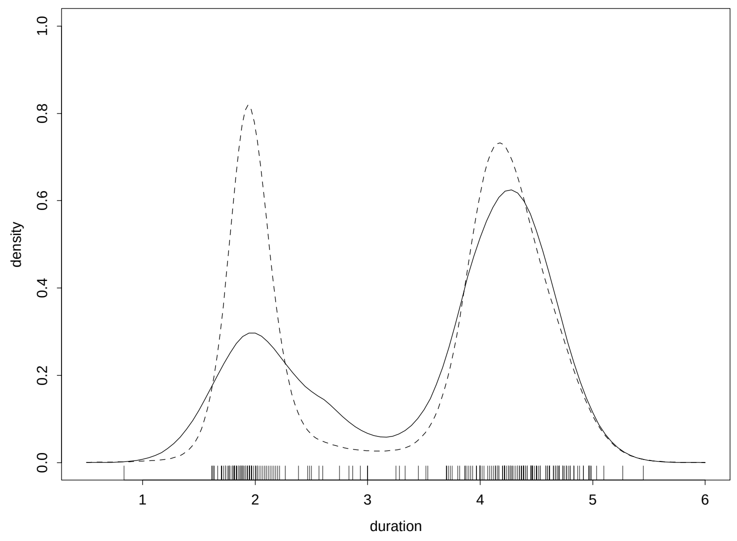

Figure 5.18: `locfit` density estimates for the duration from the `geyser` dataset.The solid line is the default, the dashed line is the adaptive bandwidth chosen by Loader (1997).

```
geyser.lf1 <- locfit(~ duration, data=geyser, flim=c(0.5,6),
                     alpha=c(0.15, 0.9))
lines(geyser.lf1, m=200, lty=3)
```

but without explaining where these numbers came from. (The default is `c(0.7, 0)`. The notes on the Web site had `c(0.1, 0.8)`. Clearly this is not an automated choice!) The first number is equivalent to the `span` parameter of `loess`; set it to zero to remove the adaptive part of the bandwidth choice. The second number is a fixed bandwidth; there is also a third argument related to the penalty in (5.9).

Chapter 6

Linear Statistical Models

6.5 Robust and resistant regression

Median polish

Consider a two-way layout. The additive model is

$$\hat{y}_{ij} = \mu + \alpha_i + \beta_j, \qquad \alpha. = \beta. = 0$$

The least squares fit corresponds to choosing the parameters μ, α_i and β_j so that the row and column sums of the residuals are zero.

Means are not resistant. Suppose we use medians instead. That is, we seek a fit of the same form, but with median $(\alpha_i) = $ median $(\beta_j) = 0$ and median$_i$ $(e_{ij}) = $ median$_j$ $(e_{ij}) = 0$. This is no longer a set of linear restrictions, so there may be many solutions. The median polish algorithm (Mosteller & Tukey, 1977; Emerson & Hoaglin, 1983) is to augment the table with row and column effects as

$$\begin{array}{cccc} e_{11} & \cdots & e_{1c} & a_1 \\ \vdots & \ddots & \vdots & \vdots \\ e_{r1} & \cdots & e_{rc} & a_r \\ b_1 & \cdots & b_r & m \end{array}$$

where initially $e_{ij} = y_{ij}$, $a_i = b_j = m = 0$. At all times we maintain

$$y_{ij} = m + a_i + b_j + e_{ij}$$

In a *row sweep* for each row we subtract the median of columns $1, \ldots, c$ from those columns and add it to the last column. For a *column sweep* for each column we subtract the median of rows $1, \ldots, r$ from those rows and add it to the bottom row.

Median polish operates by alternating row and column sweeps until the changes made become small or zero (or the human computer gets tired!). (Often just two pairs of sweeps are recommended.) The answer may depend on whether rows or columns are tried first and is very resistant to outliers. Using means rather medians will give the least-squares decomposition without iteration.

An example

The table below gives specific volume (*cc/gm*) of rubber at four temperatures ($^\circ C$) and six pressures (*kg/cm*^{2} *above atmo*). These data were published by Wood & Martin (1964, p. 260), and used by Mandel (1969) and Emerson & Wong (1985).

	Pressure					
Temperature	500	400	300	200	100	0
0	1.0637	1.0678	1.0719	1.0763	1.0807	1.0857
10	1.0697	1.0739	1.0782	1.0828	1.0876	1.0927
20	1.0756	1.0801	1.0846	1.0894	1.0944	1.0998
25	1.0786	1.0830	1.0877	1.0926	1.0977	1.1032

The default `trim=0.5` option of `twoway` performs median polish. We have, after multiplying by 10^4,

	Pressure						
Temperature	500	400	300	200	100	0	a_i
0	7.0	4.5	1.5	-1.5	-6.5	-9.0	-96.5
10	3.0	1.5	0.5	-0.5	-1.5	-3.0	-32.5
20	-3.0	-1.5	-0.5	0.5	1.5	3.0	32.5
25	-4.5	-4.0	-1.0	1.0	3.0	5.5	64.0
b_j	-111.0	-67.5	-23.5	23.5	72.5	125.0	$m = 10837.5$

This is interpreted as

$$y_{ij} = m + a_i + b_j + e_{ij}$$

and the body of the table contains the residuals e_{ij}. These have both row medians and column medians zero. Originally the value for temperature 0, pressure 400 was entered as 1.0768; the only change was to increase the residual to 94.5×10^{-4} which was easily spotted.

Note the pattern of residuals in the table; this suggests a need for transformation. Note also how linear the row and column effects are in the factor levels. Emerson & Wong (1985) fit Tukey's 'one degree of freedom for non-additivity' model

$$y_{ij} = m + a_i + b_j + e_{ij} + k a_i b_j \tag{6.11}$$

by plotting the residuals against $a_i b_j / m$ and estimating a power transformation y^λ with $\lambda = 1 - mk$ estimated as -6.81. As this is such an awkward power, they thought it better to retain the model (6.11).

Brownlee's stack loss data

We consider Brownlee's (1965) much-studied stack loss data, given in the S datasets `stack.x` and `stack.loss`. The data are from the operation of a plant for the oxidation of ammonia to nitric acid, measured on 21 consecutive days. There are 3 explanatory variables (air flow to the plant, cooling water inlet temperature, and acid concentration) and the response, 10 times the percentage of ammonia lost.

```
> summary(lm(stack.loss ~ stack.x), cor=T)
Residuals:
   Min    1Q Median   3Q Max
 -7.24 -1.71 -0.455 2.36 5.7

Coefficients:
                    Value Std. Error t value Pr(>|t|)
      (Intercept) -39.920  11.896     -3.356   0.004
  stack.xAir Flow   0.716   0.135      5.307   0.000
stack.xWater Temp   1.295   0.368      3.520   0.003
stack.xAcid Conc.  -0.152   0.156     -0.973   0.344

Residual standard error: 3.24 on 17 degrees of freedom

> lqs(stack.x, stack.loss, method="lms", nsamp="exact")

Coefficients:
 (Intercept) Air Flow Water Temp Acid Conc.
 -34.2         0.714    0.357      0

Scale estimates 0.551 0.48

> summary(lqs(stack.x, stack.loss, method="lms",
              nsamp="exact")$residuals)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
 -7.89   -0.25  0.107 1.08    1.39 9.46

> lqs(stack.x, stack.loss, method="lts", nsamp="exact")
Coefficients:
 (Intercept) Air Flow Water Temp Acid Conc.
 -35.8         0.75     0.333      0

Scale estimates 0.848 0.865
> summary(lqs(stack.x, stack.loss, method="lts",
              nsamp="exact")$residuals)
  Min. 1st Qu. Median  Mean 3rd Qu. Max.
 -8.36  -0.361  0.306 0.976    1.31 9.31
```

Function `lqs` normally uses a random search, but here we can afford an exhaustive search.

Now consider M-estimators:

```
> stack.rl <- rlm(stack.loss ~ stack.x)
> summary(stack.rl, cor=F)
Residuals:
   Min    1Q Median   3Q Max
 -8.92 -1.73 0.0617 1.54 6.5

Coefficients:
                       Value Std. Error t value
      (Intercept) -41.027   9.807     -4.183
  stack.xAir Flow   0.829   0.111      7.460
stack.xWater Temp   0.926   0.303      3.052
stack.xAcid Conc.  -0.128   0.129     -0.992

Residual standard error: 2.44 on 17 degrees of freedom
> round(stack.rl$w, 2)
 [1] 1.00 1.00 0.79 0.50 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
[13] 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.37
> summary(rlm(stack.loss ~ stack.x, method="MM"), cor=F)
Residuals:
   Min    1Q  Median   3Q  Max
 -10.5 -1.44 -0.0908 1.03 7.23

Coefficients:
                       Value Std. Error t value
      (Intercept) -41.523   9.307     -4.461
  stack.xAir Flow   0.939   0.106      8.898
stack.xWater Temp   0.579   0.288      2.012
stack.xAcid Conc.  -0.113   0.122     -0.923

Residual standard error: 1.91 on 17 degrees of freedom
```

The component `w` returned by `rlm` contains the final weights in (6.6). Although all methods seem to agree about observation 21, they differ in their view of the early observations. Atkinson (1985, pp. 129–136, 267–8) discusses this example in some detail, as well as the analyses performed by Daniel & Wood (1980). They argue for a logarithmic transformation, dropping acid concentration and fitting interactions or products of the remaining two regressors. However, the question of outliers and change of model are linked, since most of the evidence for changing the model comes from the possible outliers.

Rather than fit a parametric model we examine the points in the air flow – water temp space, using the robust fitting option of `loess`; see Figure 6.10.

```
x1 <- stack.x[,1]; x2 <- stack.x[,2]
stack.loess <- loess(log(stack.loss) ~ x1*x2, span=0.5,
   family="symmetric")
stack.plt <- expand.grid(x1=seq(50,80,0.5), x2=seq(17,27,0.2))
stack.plt$z <- as.vector(predict(stack.loess, stack.plt))
dupls <- c(2,7,8,11)
contourplot(z ~ x1*x2, stack.plt, aspect=1,
   xlab="Air flow", ylab="Water temp",
   panel = function(x, y, subscripts, ...){
      panel.contourplot(x, y, subscripts, ...)
      panel.xyplot(x1, x2)
      text(x1[-dupls] + par("cxy")[1] ,
           x2[-dupls] + 0.5* par("cxy")[2],
           as.character(seq(x1)[-dupls]), cex=0.7)
   })
```

This shows clearly that the 'outliers' are also outlying in this space. (There are duplicate points; in particular points 1 and 2 are coincident.)

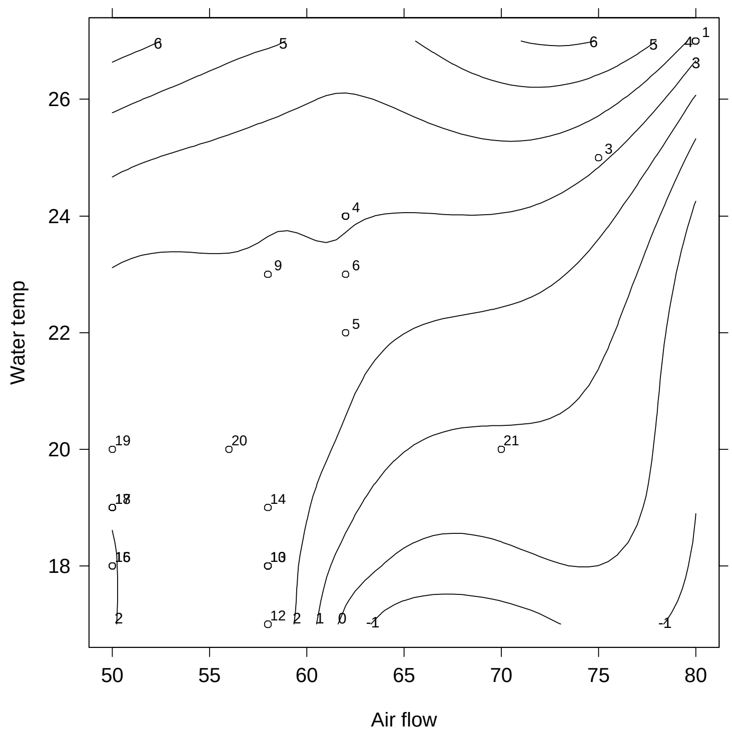

Figure 6.10: Fitted surface for Brownlee's stack loss data on log scale using `loess`.

Chapter 7

Generalized Linear Models

7.6 Over-dispersion in binomial and Poisson GLMs

The role of dispersion parameter φ in the theory and practice of GLMs is often misunderstood. For a Gaussian family with identity link and constant variance function the moment estimator used for φ is the usual unbiased modification of the maximum likelihood estimator (see equations (7.6) and (7.7)). For binomial and Poisson families the theory specifies that $\varphi = 1$, but in some cases we estimate φ as if it were an unknown parameter and use that value in standard error calculations and as a denominator in approximate F-tests rather than use chi-squared tests. This is an *ad hoc* adjustment for over-dispersion (or 'heterogeneity', see Finney (1971) who seems to have proposed the technique originally) but the corresponding likelihood may not correspond to any family of error distributions. (Of course, for the Poisson family the negative binomial family introduced in Section 7.4 provides a parametric alternative way of modelling over-dispersion.) In this section we discuss that *ad hoc* adjustment further.

We begin with a warning. A common way to 'discover' over- or under-dispersion is to notice that the residual deviance is appreciably different from the residual degrees of freedom, since in the usual theory the expected value of the residual deviance should equal the degrees of freedom. *This can be seriously misleading.* The theory is asymptotic, and only applies for large $n_i p_i$ for a binomial and for large μ_i for a Poisson. Figure 7.3 shows the exact expected value, calculated by

```
x <- 0:100
plik <- function(lambda)
  sum(dpois(x, lambda) * 2 *
  ( (lambda - x) + x * log(pmax(1,x)/lambda)))
lambda <- c(0.01, 0.05, seq(0.1, 5, 0.1))
plot(lambda, sapply(lambda, plik), type="l", ylim=c(0, 1.4),
     ylab = "E(deviance)")
abline(h=1)
```

and for a binomial

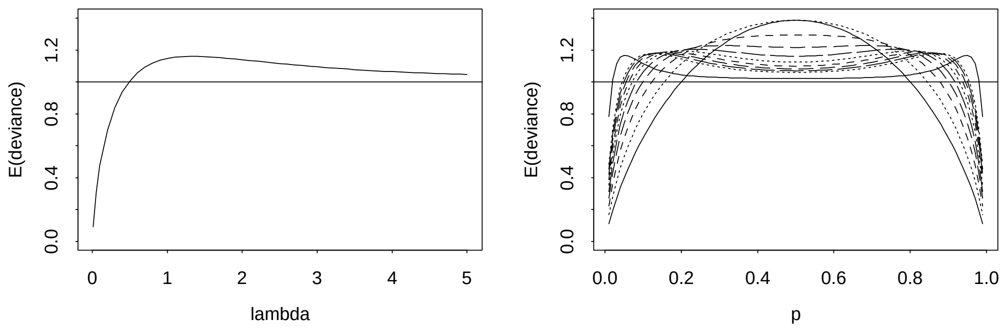

Figure 7.3: Plots of the expected residual deviance against (left) the parameter of a Poisson and (right) the p for a binomial(n, p) for $n = 1, 2, \ldots, 10, 25$.

```
n <- 1
blik <- function(p, n)
{
  y <- (0:n)/n
  devy <- sum(dbinom(1:n, n, p) * y[-1] * log(y[-1])) +
          sum(dbinom(1:n, n, 1-p) * y[-1] * log(y[-1]))
  devmu <- sum(dbinom(0:n, n, p) * (y * log(p) + (1-y)*log(1-p)))
  2 * n * (devy - devmu)
}
p <- seq(0.01, 0.99, 0.01)
plot(p, sapply(p, blik, n=n), type="l",  ylim=c(0, 1.4),
     ylab = "E(deviance)")
for(n in 2:10) lines(p, sapply(p, blik, n=n), type="l",
                     lty= 2 + (n-2)%%4)
lines(p, sapply(p, blik, n=25), type="l")
abline(h=1)
```

The estimate of φ used by `summary.glm` (if allowed to estimate the dispersion) is the (weighted) sum of the squared pearson residuals divided by the residual degrees of freedom (equation (7.8) on page 215). This has much less bias than the other estimator sometimes proposed, namely the deviance (or sum of squared *deviance* residuals) divided by the residual degrees of freedom.

Many authors (for example Finney, 1971; Collett, 1991; Cox & Snell, 1989; McCullagh & Nelder, 1989) discuss over-dispersion in binomial GLMs, and Aitkin *et al.* (1989) also discuss over-dispersion in Poisson GLMs. For binomial GLMs, the accounts all concentrate on sampling regimes that can give rise to over-dispersion in a binomial (n, p) observation Y for $n > 1$. Suppose that p is in fact a random variable θ with mean p: this might arise if there were random effects in a linear logistic model specifying p. Then if we assume that $\operatorname{var}\theta = \phi p(1-p)$ we find that

$$EY = np, \qquad \operatorname{var} Y = np(1-p)[1 + (n-1)\phi]$$

One example occurs if θ has a beta (α, β) distribution, in which case $p = \mathrm{E}\,\theta = \alpha/(\alpha+\beta)$, and $\operatorname{var}\theta = p(1-p)/(\alpha+\beta+1)$.

In the special case that the n_i in a binomial GLM are all equal, we have

$$\operatorname{var} Y = np(1-p)[1+(n-1)\phi] = \varphi np(1-p)$$

say, so this appears to provide an explanation for the *ad hoc* adjustment. However, there are problems with this.

- It is not applicable for $n \equiv 1$, a common circumstance in which to observe over-dispersion.
- There is an upper bound on ϕ and hence φ. The most extreme distribution for θ has $\theta = 1$ with probability p and $\theta = 0$ with probability $1-p$, hence variance $p(1-p)$. Thus $\phi \leqslant 1$ and $\varphi \leqslant n$. Plausible beta-binomial models will lead to much lower bounds, say $n/5$.
- If this model is accepted, the *ad hoc* adjustment of the GLM fit is not maximum likelihood estimation, even for the regression parameters.

McCullagh & Nelder (1989, pp. 125–6) prefer a variation on this model, in which the n data points are assumed to have been sampled from k clusters, and there is independent binomial sampling within the clusters (whose size now varies with n), but the clusters have probabilities drawn independently from a distribution of the same form as before. Then it is easy to show that

$$\mathrm{E}Y = np, \qquad \operatorname{var} Y = np(1-p)[1+(k-1)\phi]$$

This does provide an explanation for the *ad hoc* adjustment model for variable n, but the assumption of the same number of (equally-sized) clusters for each observation seems rather artificial to us.

Asymptotic theory for this model suggests (McCullagh & Nelder, 1989) that changes in deviance and residual deviances *scaled by* φ have asymptotic chi-squared distributions with the appropriate degrees of freedom. Since φ must be estimated, this suggests that F tests are used in place of chi-squared tests in, for example, the analysis of deviance and `addterm` and `dropterm`. At the level of the asymptotics there is no difference between the use of estimators (7.7) and (7.8), but we have seen that (7.8) has much less bias, and it is this that is used by `anova.glm` and `addterm` and `dropterm`.

Another explanation that leads to the same conclusion is to assume that n trials that make up the binomial observations are exchangeable but not necessarily independent. Then the results for any pair of trials might have correlation δ, and this leads to

$$\operatorname{var} Y = np(1-p)[1+(n-1)\delta] = \varphi np(1-p)$$

say. In this model there is no constraint that $\delta \geqslant 0$, but only limited negative correlation is possible. (Indeed, $\operatorname{var} Y \geqslant 0$ implies $\delta \geqslant -1/(n-1)$, and assuming that the trials are part of infinite population does require $\delta \geqslant 0$.)

All these explanations are effectively quasi-likelihood models, in that just the mean and variance of the observations are specified. We believe that they are best handled as `quasi` models. However, one has to be careful as they have been implemented otherwise. For a long time S-PLUS was inconsistent in that the dispersion was (by default) fixed at one for binomial and Poisson models in `summary.glm` *but* estimated in functions such as `predict.glm` that applied `lm`-based methods. Our `MASS` library has for a long time supplied a workaround for `predict.glm` that was incorporated in S-PLUS 2000.

Over-dispersion is handled slightly differently in R as from version 1.1.0. The binomial and Poisson families never allow φ to be estimated, but there are additional families `quasibinomial` and `quasipoisson` for which φ is always estimated. (As `quasi` models have no true likelihood, they have no AIC either, and so `stepAIC` will not work for them.)

7.7 Gamma models

The role of dispersion parameter φ for the Gamma family is rather different. This is a parametric family which can be fitted by maximum likelihood, including its shape parameter α. Elsewhere we have taken its density as

$$\log f(y) = \alpha \log \lambda + (\alpha - 1)\log y - \lambda y - \log \Gamma(\alpha)$$

so the mean is $\mu = \alpha/\lambda$. If we re-parametrize by (μ, α) we obtain

$$\log f(y) = \alpha(-y/\mu - \log \mu) + \alpha \log y + \alpha \log \alpha - \log y - \log \Gamma(\alpha)$$

Comparing this with the general form in equation (7.1) (on page 223) we see that the canonical link is $\theta = -1/\mu$ and $\varphi = 1/\alpha$ is the dispersion parameter. For fixed φ, fitting by `glm` gives the maximum likelihood estimates of the parameters in the linear predictor (which do not depend on the fixed value of φ), but φ is estimated from the sum of squares of the pearson residuals, which may (but need not) approximate the maximum likelihood estimator. Note that $\widehat{\varphi}$ is used to estimate the standard errors for the parameters in the linear predictor, so appreciable differences in the estimate can have practical significance.

Some authors (notably McCullagh & Nelder, 1989, pp. 295–6) have argued against the maximum likelihood estimator of φ. The MLE is the solution to

$$2n\left[\log \alpha - \psi(\alpha)\right] = D$$

where $\psi = \Gamma'/\Gamma$ is the digamma function and D is the residual deviance. Then the customary estimator of $\varphi = 1/\alpha$ is $D/(n-p)$ and the MLE is approximately[1] $\bar{D}(6+\bar{D})/(6+2\bar{D})$ where $\bar{D} = D/n$. Both the customary estimator (7.7) and the MLE are based on the residual deviance

$$D = -2\sum_i \left[\log(y_i/\hat{\mu}_i) - (y_i - \hat{\mu}_i)/\hat{\mu}_i\right]$$

[1] for large $\widehat{\alpha}$

and this is very sensitive to small values of y_i. Another argument is that if the gamma GLM is being used as a model for distributions with a constant coefficient of variation, the MLE is inconsistent for the true coefficient of variation except at the gamma family. These arguments are equally compelling for the customary estimate; McCullagh & Nelder prefer the moment estimator

$$\widehat{\sigma}^2 = \tfrac{1}{n-p} \sum \left[(y_i - \hat{\mu}_i)/\hat{\mu}_i\right]^2 \tag{7.11}$$

for the coefficient of variation σ^2 which equals φ under the gamma model. This coincides with $\widetilde{\varphi}$ as quoted by `summary.glm` (see (7.8) on page 215).

The functions `glm.shape` and `glm.dispersion` in library `MASS` compute the MLEs of α and φ respectively from a fitted Gamma `glm` object. We illustrate these with an example on clotting times of blood taken from McCullagh & Nelder (1989, pp. 300–2).

```
> clotting <- data.frame(
    u = c(5,10,15,20,30,40,60,80,100),
    lot1 = c(118,58,42,35,27,25,21,19,18),
    lot2 = c(69,35,26,21,18,16,13,12,12)  )
> clot1 <- glm(lot1 ~ log(u), data=clotting, family=Gamma)
> summary(clot1, cor=F)
Coefficients:
                Value Std. Error t value
(Intercept) -0.016554 0.00092754 -17.848
     log(u)  0.015343 0.00041496  36.975

(Dispersion Parameter for Gamma family taken to be 0.00245 )

> clot1$deviance/clot1$df.residual
[1] 0.00239
> gamma.dispersion(clot1)
[1] 0.0018583

> clot2 <- glm(lot2 ~ log(u), data=clotting, family=Gamma)
> summary(clot2, cor=F)
Coefficients:
                Value Std. Error t value
(Intercept) -0.023908 0.00132645 -18.024
     log(u)  0.023599 0.00057678  40.915

(Dispersion Parameter for Gamma family taken to be 0.00181 )

> clot2$deviance/clot2$df.residual
[1] 0.0018103
> gamma.dispersion(clot2)
[1] 0.0014076
```

The differences here are enough to affect the standard errors, but the shape parameter of the gamma distribution is so large that we have effectively a normal distribution with constant coefficient of variation.

These functions may also be used for a `quasi` family with variance proportional to mean squared. We illustrate this on the `quine` dataset. We adjust the response slightly, as a response of zero would have a zero variance and the quasi-likelihood would not be properly defined.

```
> gm <- glm(Days + 0.1 ~ Age*Eth*Sex*Lrn,
            quasi(link=log, variance=mu^2), data=quine)
> summary(gm, cor=F)
Coefficients: (4 not defined because of singularities)
                  Value Std. Error   t value
   (Intercept)  3.06105    0.39152  7.818410
         AgeF1 -0.61870    0.52528 -1.177863
         AgeF2 -2.31911    0.87546 -2.649018
         AgeF3 -0.37623    0.47055 -0.799564
   ....

(Dispersion Parameter for Quasi-likelihood family taken
                                         to be 0.61315 )

    Null Deviance: 190.4 on 145 degrees of freedom
Residual Deviance: 128.36 on 118 degrees of freedom

> gamma.shape(gm, verbose=T)
Initial estimate: 1.0603
Iter.  1  Alpha: 1.23840774338543
Iter.  2  Alpha: 1.27699745778205
Iter.  3  Alpha: 1.27834332265501
Iter.  4  Alpha: 1.27834485787226

Alpha: 1.27834
   SE: 0.13452
> summary(gm, dispersion = gamma.dispersion(gm), cor=F)
Coefficients: (4 not defined because of singularities)
                  Value Std. Error   t value
   (Intercept)  3.06105    0.44223  6.921890
         AgeF1 -0.61870    0.59331 -1.042800
         AgeF2 -2.31911    0.98885 -2.345261
         AgeF3 -0.37623    0.53149 -0.707880
    ....
```

In this example the McCullagh–Nelder preferred estimate is given by

```
> sum((residuals(gm, type="resp")/fitted(gm))^2)/gm$df.residual
[1] 0.61347
```

which is the same[2] as the estimate returned by `summary.glm`, whereas (7.7) gives

[2] up to the convergence tolerance: set `epsilon=1e-10` in the call `glm` to get equality to 7 decimal places..

```
> gm$deviance/gm$df.residual
[1] 1.0878
> gamma.dispersion(gm)
[1] 0.78226
```

There will also be differences between deviance tests and the AIC used by `step.glm` and likelihood-ratio tests and the exact AIC. Making the necessary modifications is left as an exercise for the reader.

Chapter 8

Non-linear Models

8.5 Profiles

Measures of local curvature

It is convenient to separate two sources of curvature, that of the solution locus itself, the *intrinsic curvature*, and that of the coordinate system within the solution locus, the *parameter-effects curvature*. The intrinsic curvature is fixed by the data and solution locus, but the parameter-effects curvature additionally depends upon the parametrization.

Summary measures for both kinds of *relative* curvature were proposed by Beale (1960) and elegantly interpreted by Bates & Watts (1980, 1988). (The measures are relative to the estimated standard error of y and hence scale free.) The two measures are denoted by c^θ and c^ι for the parameter-effects and intrinsic root-mean-square curvatures respectively. If F is the $F_{p,n-p}$ critical value, Bates & Watts suggest that a value of $c\sqrt{F} > 0.3$ should be regarded as indicating unacceptably high curvature of either kind. Readers are referred to Bates & Watts (1988) or Seber & Wild (1989, §4.3) for further details.

Calculating curvature measures requires both first and second derivatives of the solution locus with respect to the parameters at each observation. The second derivatives must be supplied as an $n \times p \times p$ array where the ith $p \times p$ "face" provides the symmetric matrix of second partial derivatives $\partial^2 \eta_i(\boldsymbol{\beta})/\partial\beta_j\partial\beta_k$. This may be supplied as a `hessian` attribute of the value of the model function along with the `gradient`. (Unfortunately the `nls` fitting function can make no use of any `hessian` information.)

The function `rms.curv` supplied with our library can be used to calculate and display $c^\theta\sqrt{F}$ and $c^\iota\sqrt{F}$. The only required argument is an `nls` fitted model object, provided the model function has both `gradient` and `hessian` attributes. Consider our weight loss example.

```
> expn3 <- deriv3(~ b0 + b1*2^(-x/th), c("b0","b1","th"),
      function(x, b0, b1, th) {})
> wtloss.he <- nls(Weight ~ expn3(Days, b0, b1, th),
      wtloss, start = coef(wtloss.gr))
> rms.curv(wtloss.he)
```

```
Parameter effects: c^theta x sqrt(F) = 0.1679
        Intrinsic: c^iota  x sqrt(F) = 0.0101
```

Although this result is acceptable, a lower parameter-effects curvature would be preferable (see Exercise 8.4 for a way to achieve this).

Profile traces

Profiles for non-linear regression models are discussed in Sections 8.4 and 8.5. To calculate a profile log-likelihood we hold one parameter fixed and maximize the log-likelihood with respect to all others. If we think of the fixed parameter as the independent variable, the profile log-likelihood is a function of it, but so too are the conditional maximum likelihood estimates of all other parameters. These conditional MLEs as a function of the fixed parameter we call the *profile traces*.

The generic function `profile` generates profile objects from non-linear model objects by varying each parameter up and down from its maximum likelihood value until a suitable cutoff value for the log-likelihood below the maximum is reached on either side. The profile object contains both the profile likelihoods and the traces for each parameter.

The standard S-PLUS library contains `profile` methods for `nls` and `ms` objects and `plot` methods for the objects that shows a particular view of the profile likelihood. The quantity actually plotted is the non-linear t-statistic, $\tau(\theta)$, defined in equation (8.5) on page 251.[1]

In the `MASS` library[2] there is a simple `profile` method for `glm` objects as well as (we claim) a better `plot` method for the objects produced, as well as a `pairs` method for displaying the profile traces.

We will illustrate the tools available for investigating profiles and profile traces using a familiar example: the Stormer data and its non-linear regression model introduced on page 253. The non-linear regression model is written as

$$T = \frac{\beta_1 v}{w - \beta_2} + \varepsilon$$

Note that this can also be written in the form

$$T = \frac{1}{\gamma_1 z_1 + \gamma_1 z_2} + \varepsilon$$

where, say, $\gamma_1 = 1/\beta_1$, $z_1 = w/v$, $\gamma_2 = 1/\beta_2$ and $z_2 = -1/v$. So the model may also be fitted as a generalized linear model, as noted in Exercise 8.3. It is interesting to see how much this non-linear transformation of the parameters affects the parameter effects curvature.

First consider fitting the model as a non-linear regression and displaying both views of the profile object.

[1] Note that this is not a true profile likelihood unless the variance is known.
[2] From some as yet unpublished (but widely used) work of D. M. Bates and WNV.

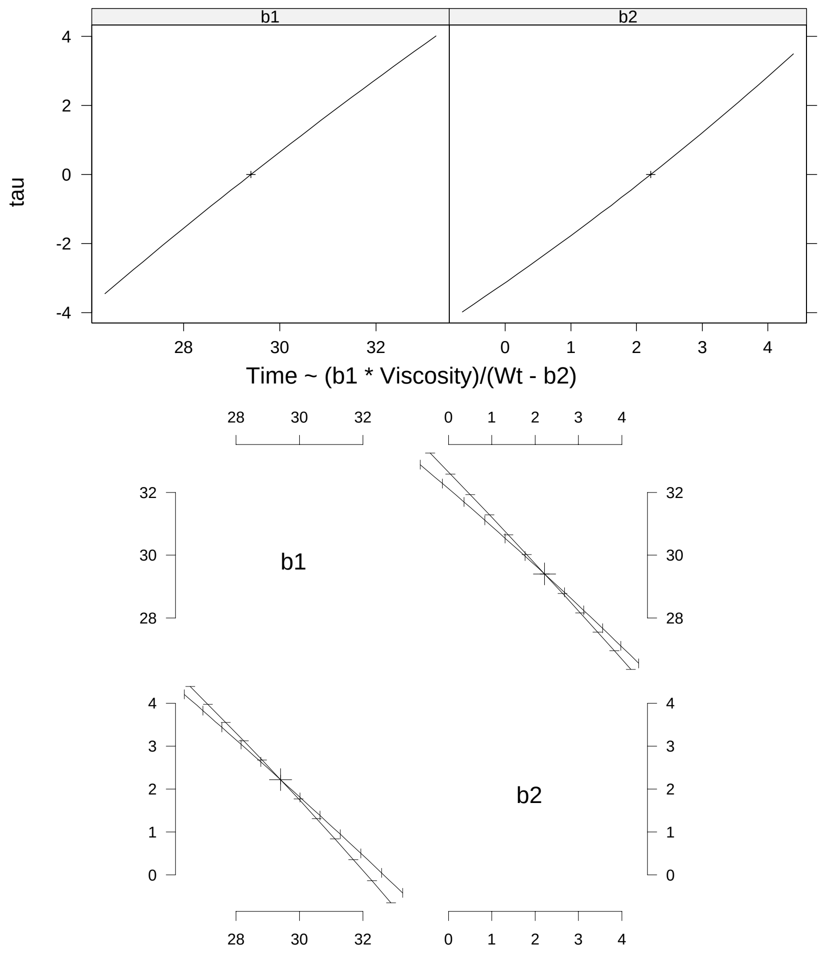

Figure 8.9: Profile and pairs-profile plots for the Stormer data example fitted as a non-linear regression model.

```
> library(MASS, first = T)
> storm.nls <- nls(Time ~ b1*Viscosity/(Wt - b2), stormer,
    start = c(b1=28, b2=2.2), trace = T)
1443.01 : 28 2.2
825.052 : 29.4012 2.21929
825.051 : 29.4013 2.21827
> storm.nls.pro <- profile(storm.nls)
> plot(storm.nls.pro)
> pairs(storm.nls.pro)
```

The results are shown in Figure 8.9. The straight lines in the first display reassure us that the profile likelihood is very nearly quadratic in those directions so the large-sample approximations are probably safe. With the pairs-profile plots note that again the straightness of the lines indicate no serious bivariate departure from normality of the estimates but the narrow angle between them suggests a very high correlation between the estimates, which is certainly the case.

Another interpretation of the profile traces displayed in the pairs-profile plot can be obtained by looking at Figure 8.3 on page 255. The profile traces are the

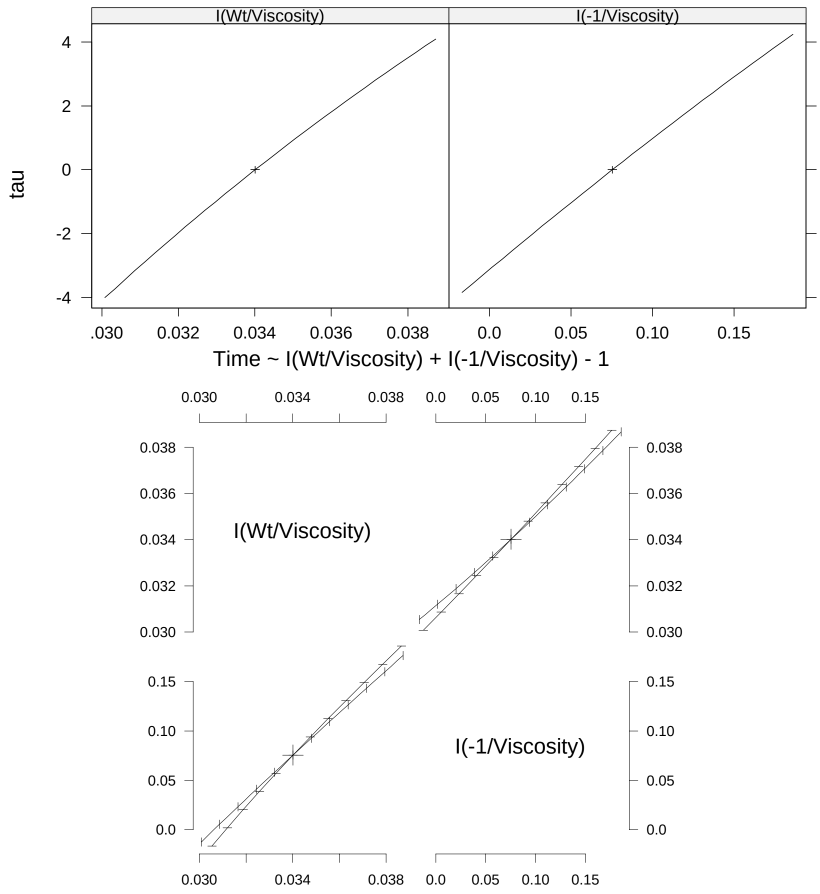

Figure 8.10: Profile and pairs-profile plots for the Stormer data example with the model fitted as a GLM.

lines that would join up the points where the contours have horizontal and vertical tangents respectively, and the fine ‘hairs’ cutting the lines in the pairs plot are an indication of those tangents. In this way the pairs-profile plot gives a hint of how the bivariate region might look, though only through what would be called the conjugate axes of the elliptical contours (if they were indeed exactly elliptical).

The software also has methods for `glm` objects, and after fitting the model as a GLM the procedure is essentially identical. We will turn on the trace when calculating profiles, though, as it shows the discrete steps taken by the algorithm and the way in which the log-likelihood falls below its global maximum value as it does so. (The details are omitted here.)

```
> storm.glm <- glm(Time ~ I(Wt/Viscosity) + I(-1/Viscosity) - 1,
      quasi(link=inverse), stormer, trace = T)
    ....
> storm.nls.pro <- profile(storm.nls)
> storm.glm.pro <- profile(storm.glm, trace=T)
    ....
> plot(storm.glm.pro)
> pairs(storm.glm.pro)
```

The results are shown in Figure 8.10. The non-linear t-statistics plots are again quite straight indicating that even though this is a highly non-linear transformation of the original parameters, for these, too, the assumption of marginal normality of the estimates is probably quite reasonable, leading to symmetric confidence intervals.

Not surprisingly the pairs plot shows us the high correlation between these functions of the original parameters as well, though the sign has changed. Again the lines are quite straight indicating no serious departure from bivariate normality of the estimates, but only in so far as this kind of diagram can indicate.

Curvature questions can be important for GLMs, as we pointed out on page 225, so the `glm` method of `profile` can be a useful exploratory tool.

Chapter 9

Smooth Regression

9.1 Additive models and scatterplot smoothers

Scatterplot smoothing

The methods expounded by Wand & Jones (1995) are implemented in Wand's library `KernSmooth`[1]. We can apply their local polynomial smoother to the simulated motorcycle example by

```
library(KernSmooth)  # ksmooth on Windows < 6.0
attach(mcycle)
plot(times, accel)
lines(locpoly(times, accel, bandwidth=dpill(times,accel)))
lines(locpoly(times, accel, bandwidth=dpill(times,accel),
              degree=2), lty=3)
detach()
```

This applies first a local linear and then a local quadratic fit. The bandwidth is chosen by the method of Ruppert *et al.* (1995).

Fitting additive models

Other ways to fit additive models in S-PLUS are available from the contributions of users. These are generally more ambitious than `gam` and `step.gam` in their choice of terms and the degree of smoothness of each term, and by relying heavily on compiled code can be very substantially faster. All of these methods can fit to multiple responses (by using the total sum of squares as the fit criterion).

Library `mda` of Hastie and Tibshirani provides functions `bruto` and `mars`. The method BRUTO is described in Hastie & Tibshirani (1990); it fits additive models with smooth functions selected by smoothing splines and will choose between a smooth function, a linear term or omitting the variable altogether. The function `mars` implements the MARS method of Friedman (1991) briefly mentioned on page 341 of the book. By default this is an additive method, fitting

[1] `ksmooth` on Windows. The current Unix sources are at http://www.biostat.harvard.edu/~mwand

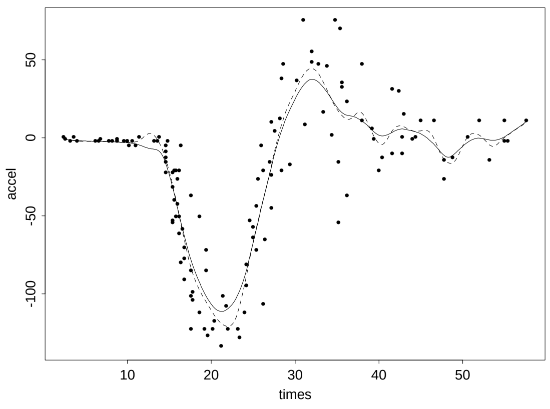

Figure 9.9: Smooths by `locpoly` of the `mcycle` data. The solid line is a locally linear fit and the dashed line a locally quadratic one.

splines of order 1 (piecewise linear functions) to each variable; again the number of pieces is selected by the program so that variables can be entered linearly, non-linearly or not at all.

The library `polymars` of Kooperberg and O'Connor implements a restricted form of MARS (for example, allowing only pairwise interactions) suggested by Kooperberg *et al.* (1997).

An example: the `cpus` data

We consider BRUTO and MARS models. These need matrices (rather than formulae and data frames) as inputs.

```
Xin <- as.matrix(cpus0[samp,1:6])
library(mda)
test2 <- function(fit) {
  Xp <- as.matrix(cpus0[-samp,1:6])
  sqrt(sum((log10(cpus0[-samp, "perf"]) -
            predict(fit, Xp))^2)/109)
}
cpus.bruto <- bruto(Xin, log10(cpus0[samp,7]))
test2(cpus.bruto)
[1] 0.21336

cpus.bruto$type
[1] excluded smooth   linear   smooth   smooth   linear
cpus.bruto$df
 syct   mmin mmax   cach  chmin chmax
    0 1.5191    1 1.0578 1.1698     1

# examine the fitted functions
par(mfrow=c(3,2))
Xp <- matrix(sapply(cpus0[samp, 1:6], mean), 100, 6, byrow=T)
for(i in 1:6) {
  xr <- sapply(cpus0, 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(cpus0)[i], ylab="", type="l")
}
```

The result (not shown) indicates that the non-linear terms have a very slight curvature, as might be expected from the equivalent degrees of freedom that are reported.

We can use `mars` to fit a piecewise linear model with additive terms.

```
cpus.mars <- mars(Xin, log10(cpus0[samp,7]))
showcuts <- function(obj)
{
  tmp <- obj$cuts[obj$sel, ]
  dimnames(tmp) <- list(NULL, dimnames(Xin)[[2]])
  tmp
}
> showcuts(cpus.mars)
     syct   mmin   mmax cach chmin chmax
[1,]    0 0.0000 0.0000    0     0     0
[2,]    0 0.0000 3.6021    0     0     0
[3,]    0 0.0000 3.6021    0     0     0
[4,]    0 3.1761 0.0000    0     0     0
[5,]    0 0.0000 0.0000    0     8     0
[6,]    0 0.0000 0.0000    0     0     0
> test2(cpus.mars)
[1] 0.21366
# examine the fitted functions
Xp <- matrix(sapply(cpus0[samp, 1:6], mean), 100, 6, byrow=T)
for(i in 1:6) {
  xr <- sapply(cpus0, 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(cpus0)[i], ylab="", type="l")
}
> cpus.mars2 <- mars(Xin, log10(cpus0[samp,7]), degree=2)
> showcuts(cpus.mars2)
     syct   mmin   mmax cach chmin chmax
[1,]    0 0.0000 0.0000    0     0     0
[2,]    0 0.0000 3.6021    0     0     0
[3,]    0 1.9823 3.6021    0     0     0
[4,]    0 0.0000 0.0000   16     8     0
[5,]    0 0.0000 0.0000    0     0     0
> test2(cpus.mars2)
[1] 0.21495
> cpus.mars6 <- mars(Xin, log10(cpus0[samp,7]), degree=6)
> showcuts(cpus.mars6)
       syct   mmin   mmax cach chmin chmax
[1,] 0.0000 0.0000 0.0000    0     0     0
```

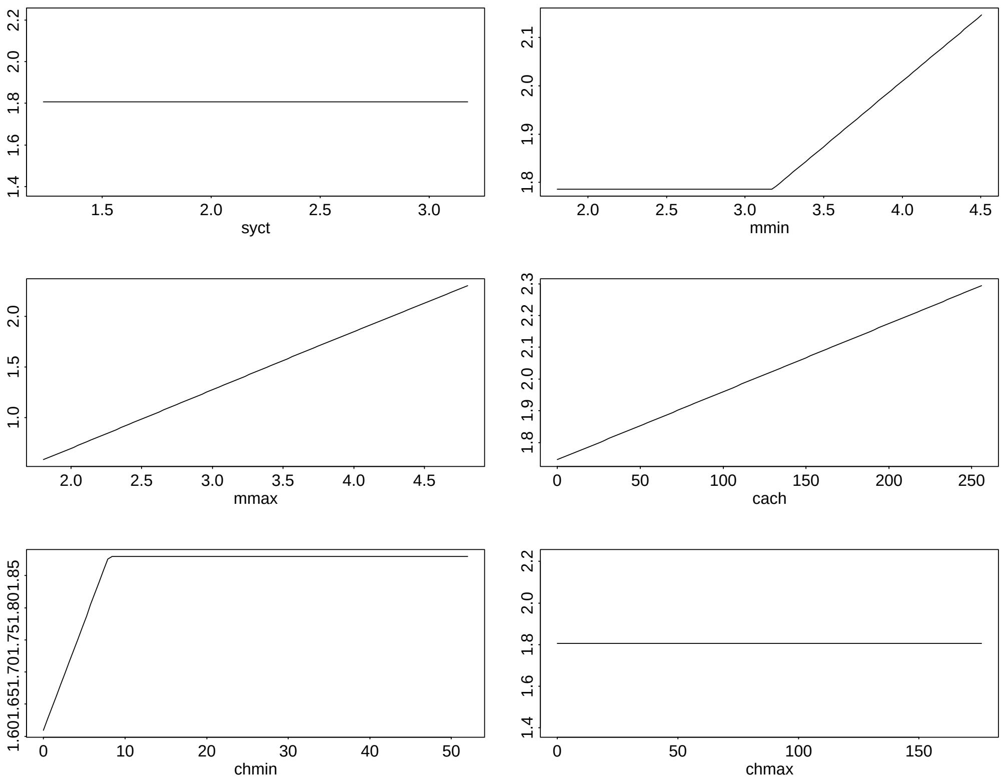

Figure 9.10: Plots of the additive functions used by `cpus.mars`.

```
[2,] 0.0000 1.9823 3.6021    0     0     0
[3,] 0.0000 0.0000 0.0000   16     8     0
[4,] 0.0000 0.0000 0.0000   16     8     0
[5,] 0.0000 0.0000 3.6990    0     8     0
[6,] 2.3979 0.0000 0.0000   16     8     0
[7,] 2.3979 0.0000 3.6990   16     8     0
[8,] 0.0000 0.0000 0.0000    0     0     0
> test2(cpus.mars6)
[1] 0.20604
```

Allowing pairwise interaction terms (by `degree=2`) or allowing arbitrary interactions make little difference to the effectiveness of the predictions.

Local likelihood models

Local likelihood provides a different way to extend models such as GLMs to use smooth functions of the covariates. In the local likelihood approach the prediction at x is made by fitting a fully parametric model to the observations in a neighbourhood of x. More formally, a weighted likelihood is used, where the weight for observation i is a decreasing function of the 'distance' of x_i from x. (We have already seen this approach for density estimation.) Note that in this approach we are compelled to have predictions which are a smooth function of *all* the covariates jointly and so it is only suitable for a small number of covariates,

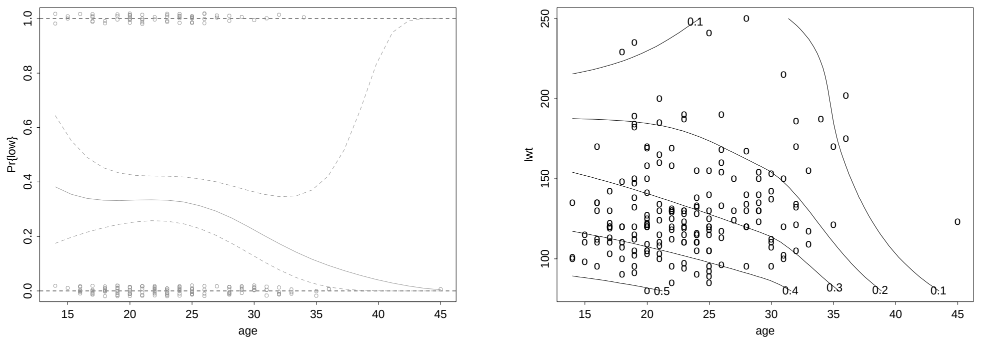

Figure 9.11: Probability of low birthweight in dataset `birthwt`. **Left:** Against mother's age, by `sm.logit`, with pointwise confidence intervals shown by dashed lines. **Right:** Against mother's age and last weight, by `locfit`.

usually not more than two. In principle the computational load will be daunting, but this is reduced (as in `loess`) by evaluating the prediction at a judiciously chosen set of points and interpolating.

The library `sm`[2] of Bowman & Azzalini (1997) implements this approach for a single covariate in functions `sm.logit` (a binomial log-linear model) and `sm.poisson` (a Poisson log-linear model). For example, we can consider the effect of the mother's age on the probability of a low birthweight in the dataset `birthwt` by

```
library(sm)
attach(birthwt)
sm.logit(age, low, h=5, display="se")
detach()
```

Here the bandwidth `h` is the standard deviation of the Gaussian kernel used.

Loader's library `locfit` provides a function `locfit` with much greater flexibility. It can fit Gaussian, binomial, Poisson, gamma and negative binomial GLMs with identity, log, logit, inverse and square root links and one or more (in practice, two or three) covariates. We can try this for the joint response to `age` and `lwt` in `birthwt`.

```
library(locfit, first=T)
bwt.lf <- locfit(low ~ age+lwt, data=birthwt, family="binomial",
                 deg=1, scale=0, alpha=c(0,5))
plot(bwt.lf, get.data=T)
```

Note that the use of `scale=0` is essential as in density estimation. We chose a local linear fit as the data are few and quadratic fitting (the default) has little theoretical advantage over linear fitting.

As a second example, consider the dataset `Pima.tr` of diabetes on 200 Pima Indians. Previous studies (Wahba *et al.*, 1995; Ripley, 1996) have suggested that

[2] available from `http://www.stats.gla.ac.uk/~adrian/sm` and `http://www.stat.unipd.it/dip/homes/azzalini/SW/Splus/sm`.

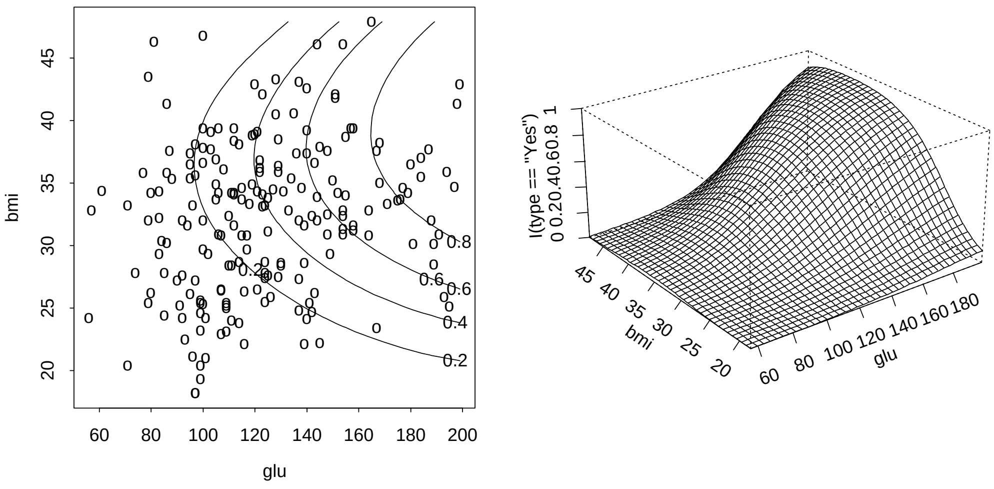

Figure 9.12: Plots of the probability surface fitted to the `Pima.tr` dataset by `locfit` using a local logistic regression.

the two continuous variables `glu` (plasma glucose level) and `bmi` (body mass index) have the most discriminatory effect. We consider a local logistic regression on these two variables

```
pima.lf <- locfit(I(type=="Yes") ~ glu + bmi, data=Pima.tr,
                  family="binomial", scale=0, alpha=c(0,5))
par(mfrow=c(1,2), pty="s")
plot(pima.lf, get.data=T); plot(pima.lf, type="persp")
```

shown in Figure 9.12.

9.4 Neural Networks

Internal details of `nnet.default`

The C code on which `nnet.default` is based is quite general and can in fact be used for networks with an arbitrary pattern of feed-forward connections. Internally the nodes are numbered so that all connections are from lower to higher numbers; the bias unit has number 0, the inputs numbers 1 to m, say, and the output units are the highest-numbered units. The code in `summary.nnet` shows how to 'unpack' the connections. These are stored in vectors, so the weights are stored in a single vector. The connections are sorted by their *destination* so that all connections to unit i precede those to unit $i+1$. The vector `conn` gives the source unit, and `nconn` is an index vector for the first connection to that destination. An example will make this clearer:

```
> rock.nn$nconn
[1]  0  0  0  0  0  4  8 12 19
> rock.nn$conn
 [1] 0 1 2 3 0 1 2 3 0 1 2 3 0 4 5 6 1 2 3
```

```
> summary(rock.nn)
a 3-3-1 network with 19 weights
options were - skip-layer connections  linear output units
decay=0.001
  b->h1 i1->h1 i2->h1 i3->h1
   4.47 -11.16  15.31  -8.78
  b->h2 i1->h2 i2->h2 i3->h2
   9.15 -14.68  18.45 -22.93
  b->h3 i1->h3 i2->h3 i3->h3
   1.22  -9.80   7.10  -3.77
   b->o  h1->o  h2->o  h3->o  i1->o  i2->o  i3->o
   8.78 -16.06   8.63   9.66  -1.99  -4.15   1.65
```

Unit 0 is the bias (`"b"`), units 1 to 3 are the inputs, 4 to 6 the hidden units and 7 the output. The vectors `conn` and `nconn` follow the C indexing convention, starting with zero. Thus unit `h1` (4) has connections from units 0, 1, 2 and 3. The vector `nconn` has a final element giving the total number of connections.

These connection vectors are normally constructed by the function `add.net`; this automatically adds a connection to a bias unit whenever a unit gets its first incoming connection.

Chapter 10

Tree-based Methods

10.4 Tree-structured survival analysis

Survival data are usually continuous, but are characterized by the possibility of censored observations. There have been various approaches to extending regression trees to survival data in which the prediction at each leaf is a survival distribution.

The deviance approach needs a common survival distribution with just one parameter (say the mean) varying between nodes. As the survival distribution has otherwise to be known completely, we would need to take, for example, a Weibull distribution with a specific α. Thus this approach has most often been used with an exponential distribution (it goes back at least to Ciampi *et al.*, 1987 and is expounded in detail by Davis & Anderson, 1989).

Another family of approaches has been via impurity indices, which we recall measure the decrease in impurity on splitting the node under consideration. This can be replaced by a '*goodness-of-split*' criterion measuring the difference in survival distribution in the two candidate daughter nodes. In regression trees the reduction in sum of squares can be seen as a goodness-of-split criterion, but a more natural candidate might be the unpooled (Welch) t-test between the samples passed to the two daughters. Given this change of viewpoint we can replace the t-test by a test which takes censoring into account and is perhaps more appropriate for the typical shape of survival curves. The split selected at a node is then the candidate split with the most significant test of difference.

Library `rpart`

Library `rpart` has two further options selected by its `method` argument:

`"poisson"` in which the response is the number of events N_i in a specified duration t_i of observation. Deviance-based criteria are used to splitting and for pruning, assuming a Poisson-distributed number of events with mean $\lambda_t t_i$ where the rate depends on the node t. The response is specified as either a two-column matrix of (N_i, t_i) or just a vector of N_i (in which case the time intervals are taken to be of unit length for all observations).

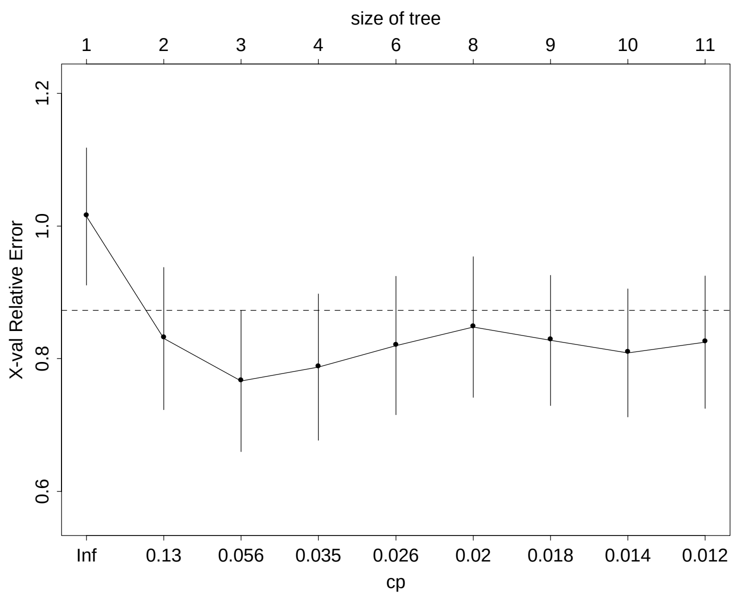

Figure 10.12: Plot by `plotcp` of the `rpart` object `VA.rp`.

`"exp"` A survival tree in which the response must be a survival object, normally generated by `Surv`. This is a variant of the `"poisson"` method. Suppose that an exponential distribution was appropriate for the survival times. Then by the duality between views of a Poisson process the observed number of events (0 or 1) in the duration to censoring or death can be taken to be Poisson distributed, and the `"poisson"` method will give the correct likelihood. In general the exponential distribution is not appropriate, but it can perhaps be made so by non-linearly transforming time by the cumulative hazard function, and this is done estimating the cumulative hazard from the data[1]. This gives a proportional hazards model with the baseline hazard fixed as the estimated marginal hazard.

We use the VA cancer dataset `cancer.vet` considered in Chapter 12 to illustrate a survival example.

```
> set.seed(123)
> VA.rp <- rpart(Surv(stime, status) ~ ., data=VA, minsplit=10)
> plotcp(VA.rp)
> printcp(VA.rp)
    ....
Root node error: 158/137 = 1.15

      CP nsplit rel error xerror   xstd
1 0.1923      0     1.000  1.014 0.1034
2 0.0829      1     0.808  0.830 0.1071
```

[1] Note that this transformation is of the *marginal* distribution of survival times, although an exponential distribution would normally be assumed for the distribution conditional on the covariates. This is the same criticism as we see for the HARE / HEFT methodology in Chapter 12 of these complements. RPart follows LeBlanc & Crowley (1992) in this ‘one-step’ approach.

```
3 0.0380      2     0.725  0.766 0.1067
4 0.0319      3     0.687  0.787 0.1102
5 0.0210      5     0.623  0.820 0.1045
6 0.0189      7     0.581  0.848 0.1060
7 0.0164      8     0.562  0.828 0.0982
8 0.0123      9     0.546  0.809 0.0966
9 0.0110     10     0.533  0.825 0.0999

> print(VA.rp, cp=0.09)
node), split, n, deviance, yval
      * denotes terminal node

1) root 137 160 1.0
  2) Karn>45 99  81 0.8 *
  3) Karn<45 38  46 2.5 *
```

Here `yval` is the relative hazard rate for that node; we have a proportional hazards model and this is the estimated proportional factor.

In our experience it is common for tree-based methods to find little structure in cancer prognosis datasets: what structure there is depends on subtle interactions between covariates.

Library `tssa`

This approach is outlined by Segal (1988), who considers a family of statistics introduced by Tarone & Ware (1977) which includes the log-rank (Mantel-Haenszel) and Gehan tests and Prentice's generalization of the Wilcoxon test. His approach is implemented in the `tssa` library of Segal and Wager. This uses `tssa` as the main function, and generates objects of class `"tssa"` which inherits from class `"tree"`. A member of the family of test statistics is selected by the argument `choice`. Splitting continues until there are `maxnodes` nodes (default 50) or no leaf has as many as `minbuc` cases (default 30) *and* a proportion at least `propn` (default 15%) of uncensored cases.

We consider the VA lung cancer data of Section 12.4. Since `tssa` cannot currently handle multi-level factors, we have to omit the variable `cell`.

```
> library(tssa, first=T)
> VA.tssa <- tssa(stime ~ treat + age  + Karn + diag.time + prior,
                  status, data=VA, minbuc=10)
> VA.tssa
node), split, (n, failures), km-median, split-statistic
      * denotes terminal node, choice is Mantel-Haenzel

 1) root (137,128)  76.5 6.67
   2) Karn<45 (38,37)  19.5 2.71
     4) diag.time<10.5 (28,27)  21.0 2.08
       8) age<62.5 (14,13)  18.0 *
       9) age>62.5 (14,14)  33.0 *
```

```
        5) diag.time>10.5 (10,10)   8.0 *
      3) Karn>45 (99,91) 110.5 2.74
        6) Karn<82.5 (90,84) 104.0 2.22
         12) age<67.5 (74,69) 111.5 1.34
           24) prior<1.5 (50,48) 104.0 1.55
             48) age<59 (24,23) 110.0 1.22
               96) age<46.5 (13,13)  99.0 *
               97) age>46.5 (11,10) 127.0 *
             49) age>59 (26,25)  95.0 0.91
               98) diag.time<3.5 (11,11)  91.0 *
               99) diag.time>3.5 (15,14)  98.5 *
           25) prior>1.5 (24,21) 139.5 1.10
             50) treat<1.5 (14,13) 122.0 *
             51) treat>1.5 (10,8) 145.5 *
         13) age>67.5 (16,15)  72.0 *
        7) Karn>82.5 (9,7) 234.5 *

> summary(VA.tssa)
Survival tree:
tssa(formula = stime ~ treat + age + Karn + diag.time + prior,
        delta = status, data = VA, minbuc = 10)
Number of terminal nodes:  11
> tree.screens()
> plot(VA.tssa)
> text(VA.tssa)
> km.tssa(VA.tssa)
> close.screen(all=T)
```

It can be helpful to examine more than just the mean at each node; the function `km.tssa` will plot the Kaplan-Meier estimates of survival curves for the two daughters of a non-terminal node. Interactive exploration[2] shows that there is very little difference in survival between nodes at (Figure 10.13) or below node 6.

The change from a goodness-of-fit to a goodness-of-split view is not helpful for pruning a tree. Segal (1988) replaced optimizing a measure of the fit of the tree (as in cost-complexity pruning) with a stepwise approach.

(i) Grow a very large tree.

(ii) Assign to each non-terminal node the largest split statistic in the subtree rooted at that node. (This can be done in a single upwards pass on the tree.)

(iii) Obtain a sequence of pruned trees by repeatedly pruning at the remaining node(s) with the smallest assigned values.

(iv) Select one of these trees, perhaps by plotting the minimum assigned value against tree size and selecting the tree at an ‘elbow’.

This is implemented in `prune.tssa`. Like `snip.tree` (and `snip.tssa`), a value is selected by a first click (on the lower screen), and the tree pruned at that value on the second click. For our example we can use

[2] this relies on `erase.screen` which is broken in some versions of S-PLUS 4.x.

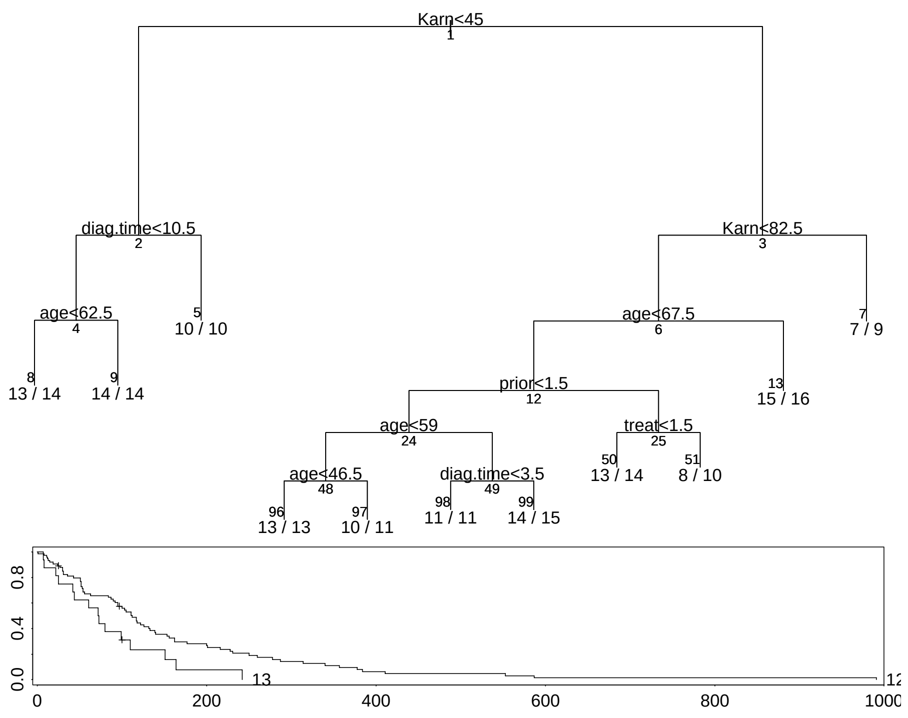

Figure 10.13: Tree fitted by `tssa` to the `cancer.vet` dataset. The bottom screen shows the output from `km.tssa` when node 6 was selected.

```
tree.screens()
plot(VA.tssa)
prune(VA.tssa)
close.screen(all=T)
```

The only clear-cut pruning point (Figure 10.14) is at a single split. There is a function `post.tssa` the equivalent of (and modified from) `post.tree` for `tssa` trees.

Library `survcart`

The library `survcart` [3] is sparsely documented, but appears to implement the strategy of LeBlanc & Crowley (1993). Like Segal, LeBlanc & Crowley use a goodness of split criterion for growing the tree, in this case the log-rank statistic with some adjustment for selecting the maximal statistic over all possible splits of continuous variables. However, the pruning strategy differs from `tssa`. Associate to each non-terminal node the goodness-of-split statistic $G(\ell)$, taking G to be zero at the terminal nodes. Then LeBlanc & Crowley apply cost-complexity pruning to the measure of fit

$$R(T) = -\sum_{\ell \in T} G(\ell)$$

[3] also known as `CART_SD`. Not available for S-PLUS 6 as it uses obselete language features.

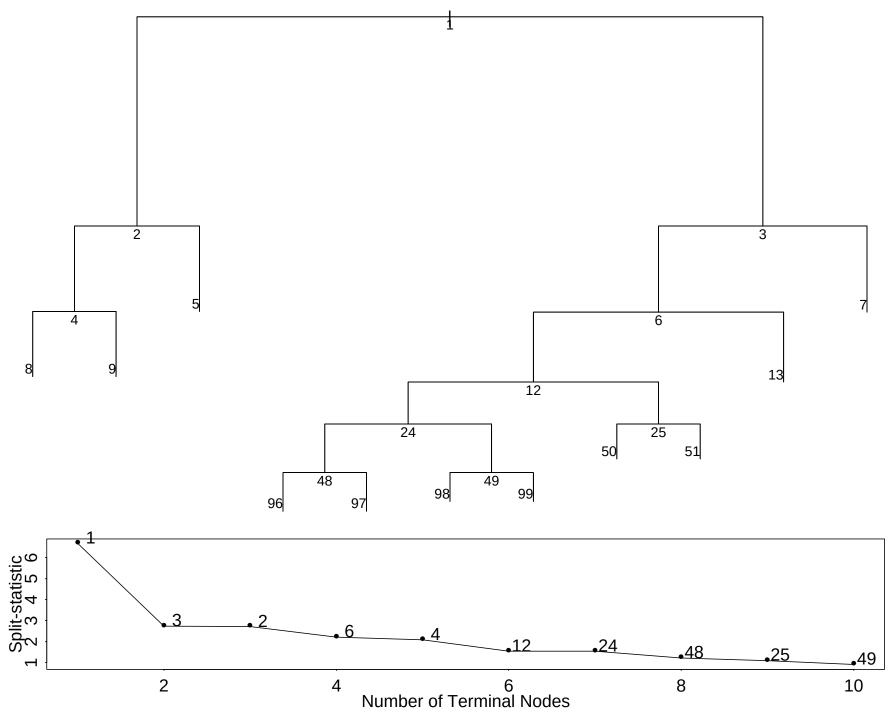

Figure 10.14: Tree fitted by `tssa` to the `cancer.vet` dataset. The bottom screen shows the prune sequence from `prune.tssa`.

This is not a sum over cases, but as it is defined additively over branches the standard pruning algorithm (Breiman *et al.*, 1984; Ripley, 1996) is still justified. (The 'deviance' quoted by `prune.survtree` is $\sum_\ell G(\ell)$.) The measure of fit can be computed on a validation set based down the optimally pruned tree sequence (T_r), but as it is not a measure of performance there is no justification for then choosing the best fit; indeed $R(T)$ decreases monotonically as the tree is grown, since $G(\ell) \geqslant 0$. The suggestion of LeBlanc & Crowley is to choose the pruning minimizing $R_\alpha(T)$ on the validation set for $\alpha \in [2, 4]$. (LeBlanc & Crowley also discuss using bootstrapping to bias-correct $R(T)$ computed on the training set prior to pruning.)

Library `survcart` can be very memory-hungry: it comes with an informative demonstration that needs over 50Mb[4] of virtual memory to run.

We can try our `VA` cancer example by

```
library(survcart, first=T)
VA.st <- survtree(stime ~ treat + age + Karn + diag.time +
                          cell + prior,
                  data=VA, status, fact.flag=c(F,T,T,T,F,F))
plot(prune.survtree(VA.st))
```

The argument `fact.flag` says which variables should be regarded as *not* factors and included in the adjustment of the log-rank statistic for continuous variates

[4] on each of S-PLUS 3.3 for Windows and S-PLUS 3.4 on Sun Solaris; over 100Mb on S-PLUS 4.0 for Windows

(although a factor with many levels will give rise to *very* many more possible splits). The 'deviance' is $-R(T_k) - \alpha_k(|T_k| - 1)$!

We can reserve a validation set and use this for pruning by

```
set.seed(123); tr <- sample(nrow(VA), 90)
VA1 <- VA[tr,]; VA2 <- VA[-tr,]
VA.st1 <- update(VA.st, data=VA1)
VA.st1.pr <- prune.survtree(VA.st1, newdata=VA2,
                            zensor.newdata=VA2$status)
VA.st1.pr
$size:
 [1] 12 11 10  9  8  5  4  3  2  1  0
$dev:
 [1]  36.6986  36.0633  35.1245  24.2267  24.2514  13.5163
 [7]  15.7134  15.5296 -16.7492  -8.2354   0.0000
$k:
 [1]  0.000000  0.033653  0.048377  0.709060  0.733988  2.595874
 [7]  2.692954  3.346168 12.984497 13.469285 19.090138
```

Note that the size is the number of splits, one less than the number of leaves. We need to convert this to a split-complexity measure:

```
attach(VA.st1.pr)
dev <- dev + k*size
> dev - 2*size
 [1] 12.6986 14.4335 15.6082 12.6082 14.1233 16.4956 18.4853
 [8] 19.5681  5.2198  3.2339  0.0000
> dev - 4*size
 [1] -11.3014  -7.5665  -4.3918  -5.3918  -1.8767   6.4956
 [7]  10.4853  13.5681   1.2198   1.2339   0.0000
detach()
```

which suggests a tree with three splits

```
> prune(VA.st1, k=4)

1) root 90 19
  2) cell:2,3 49 13
    4) prior:0 40  0 *
    5) prior:10 9  0 *
  3) cell:1,4 41 13
    6) Karn<45 8  0 *
    7) Karn>45 33  0 *
```

Note how the selection penalty on continuous variables such as `Karn` reduces their prominence.

We can explore the spread of predictions over splits in a manner similar to `km.tssa` by picking values of k in

```
VA.st.tmp <- prune.survtree(VA.st, k=2)
plot(surv.fit(VA$stime, VA$status, factor(VA.st.tmp$where)))
```

This shows the Kaplan-Meier estimates of survival at all the leaves, and by successively reducing k we can see when the range of variation is no longer essentially covered.

The function `graph.survtree` allows various aspects of the tree model to be plotted. The following call plots the median survival by node

```
graph.survtree(prune(VA.st, k=3.5), VA$stime, VA$status,
                    xtile=0.5, interactive=F)
```

but it can also show the survival probability at a fixed time.

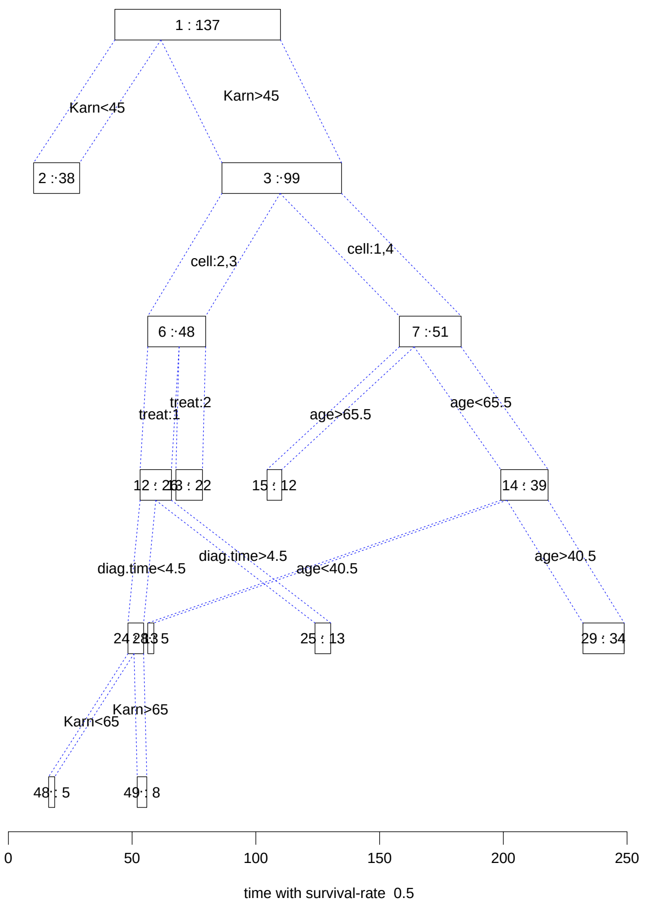

Figure 10.15: Plot of median survival by `graph.survtree`.

Chapter 11

Multivariate Analysis and Pattern Recognition

11.3 Correspondence analysis

Multiple correspondence analysis

Multiple correspondence analysis (MCA) is (confusingly!) a method for visualizing the joint properties of $p \geqslant 2$ categorical variables that does *not* reduce to correspondence analysis (CA) for $p = 2$, although the methods are closely related (see, for example, Gower & Hand, 1996, §10.2).

Suppose we have n observations on the p factors with ℓ total levels. Consider G, the $n \times \ell$ indicator matrix whose rows give the levels of each factor for each observation. Then all the row sums are p. MCA is often (Greenacre, 1992) defined as CA applied to the table G, that is the singular-value decomposition of $D_r^{-1/2}(G/\sum_{ij} g_{ij})D_c^{-1/2} = U\Lambda V^T$. Note that $D_r = pI$ since all the row sums are p, and $\sum_{ij} g_{ij} = np$, so this amounts to the SVD of $p^{-1/2}GD_c^{-1/2}/pn$.[1]

An alternative point of view is that MCA is a principal components analysis of the data matrix $X = G(pD_c)^{-1/2}$; with PCA it is usual to centre the data but it transpires that the largest singular value is one and the corresponding singular vectors account for the means of the variables. Thus a simple plot for MCA is to plot the first two principal components of X. It will not be appropriate to add axes for the columns of X as the possible values are only $\{0, 1\}$, but it is usual to add the positions of 1 on each of these axes, and label these by the factor level. (The 'axis' points are plotted at the appropriate row of $(pD_c)^{-1/2}V$.) The point plotted for each observation is the vector sum of the 'axis' points for the levels taken of each of the factors. Gower and Hand seem to prefer (e.g. their figure 4.2) to rescale the plotted points by p, so they are plotted at the centroid of their levels. This is exactly the asymmetric row plot of the CA of G, apart from an overall scale factor of $p\sqrt{n}$.

We can apply this to the example of Gower & Hand (1996, p. 75) by

[1] Gower & Hand (1996) omit the divisor pn.

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

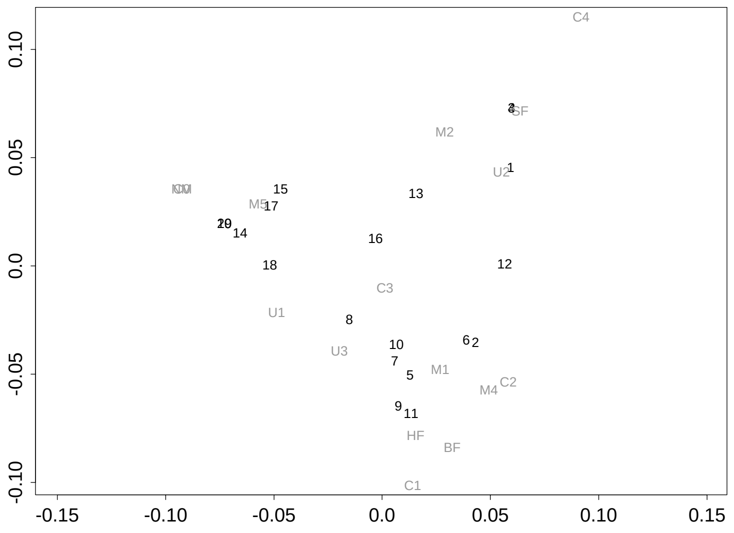

Figure 11.21: Multiple correspondence analysis plot of data on 20 farms on the Dutch island of Terschelling. Numbers represent the farms and labels levels of moisture, grassland usage, manure usage and type of grassland management.

Sometimes it is desired to add rows or factors to an MCA plot. Adding rows is easy: the observations are placed at the centroid of the 'axis' points for levels that are observed. Adding factors (so-called *supplementary variables*) is less obvious. The 'axis' points are plotted at the rows of $(pD_c)^{-1/2}V$. Since $U\Lambda V^T = X = G(pD_c)^{-1/2}$, $V = (pD_c)^{-1/2}G^T U\Lambda^{-1}$ and

$$(pD_c)^{-1/2}V = (pD_c)^{-1}G^T U\Lambda^{-1}$$

This tells us that the 'axis' points can be found by taking the appropriate column of G, scaling to total $1/p$ and then taking inner products with the second and third columns of $U\Lambda^{-1}$. This procedure can be applied to supplementary variables and so provides a way to add them to the plot. The `predict` method for class `"mca"` allows rows or supplementary variables to be added to an MCA plot.

11.10 Factor analysis

We return to discovering structure from the data matrix X alone, without predetermined groups. Factor analysis seeks linear combinations $\boldsymbol{xa}$ of the variables, called *factors*, which represent underlying fundamental quantities of which the observed variables are expressions. The examples tend to be controversial ones such as 'intelligence' and 'social deprivation', the idea being that a small number of factors might explain a large number of measurements in an observational study.

This aim seems close to that of principal component analysis, but the statistical model differs. For a single common factor f we have

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\lambda} f + \boldsymbol{u} \tag{11.6}$$

where $\boldsymbol{\lambda}$ is a vector known as the *loadings* and $\boldsymbol{u}$ is a vector of *unique* (or *specific*) factors for that observational unit. To help make the model identifiable, we assume that the factor f has mean zero and variance one, and that $\boldsymbol{u}$ has mean zero and unknown *diagonal* covariance matrix Ψ. For $k < p$ common factors we have a vector $\boldsymbol{f}$ of common factors and a loadings matrix Λ, and

$$\boldsymbol{x} = \boldsymbol{\mu} + \Lambda \boldsymbol{f} + \boldsymbol{u} \tag{11.7}$$

where the components of $\boldsymbol{f}$ have unit variance and are uncorrelated and $\boldsymbol{f}$ and $\boldsymbol{u}$ are taken to be uncorrelated. Note that *all* the correlations amongst the variables in $\boldsymbol{x}$ must be explained by the common factors; if we assume joint normality the observed variables $\boldsymbol{x}$ will be conditionally independent given $\boldsymbol{f}$.

Principal component analysis also seeks a linear subspace like $\Lambda\boldsymbol{f}$ to explain the data, but measures the lack of fit by the sum of squares of the $\boldsymbol{u}_i$. Since factor analysis allows an arbitrary diagonal covariance matrix Ψ its measure of fit of the $\boldsymbol{u}_i$ depends on the problem and should be independent of the units of measurement of the observed variables. (Changing the units of measurement of the observations does not change the common factors if the loadings and unique factors are re-expressed in the new units.)

Equation (11.7) and the conditions on $\boldsymbol{f}$ express the covariance matrix Σ of the data as

$$\Sigma = \Lambda\Lambda^T + \Psi \tag{11.8}$$

Conversely, if (11.8) holds, there is a k-factor model of the form (11.7). Note that the common factors $G^T\boldsymbol{f}$ and loadings matrix ΛG give rise to the same model for Σ, for any $k \times k$ orthogonal matrix G. Choosing an appropriate G is known as choosing a *rotation*. All we can achieve statistically is to fit the space spanned by the factors, so choosing a rotation is a way to choose an interpretable basis for that space. Note that if

$$s = \tfrac{1}{2}p(p+1) - [p(k+1) - \tfrac{1}{2}k(k-1)] = \tfrac{1}{2}(p-k)^2 - \tfrac{1}{2}(p+k) < 0$$

we would expect an infinity of solutions to (11.8). This value is known as the *degrees of freedom*, and comes from the number of elements in Σ minus the number of parameters in Ψ and Λ (taking account of the rotational freedom in Λ since only $\Lambda\Lambda^T$ is determined). Thus it is usual to assume $s \geqslant 0$; for $s = 0$ there may be a unique solution, no solution or an infinity of solutions (Lawley & Maxwell, 1971, pp. 10–11).

The variances of the original variables are decomposed into two parts, the *communality* $h_i^2 = \sum_j \lambda_{ij}^2$ and *uniqueness* ψ_{ii} which is thought of as the 'noise' variance.

Fitting the factor analysis model (11.7) is performed by the S-PLUS function `factanal`. The default method ('principal factor analysis') dates from the days of limited computational power, and is not intrinsically scale invariant—it should not be used. The preferred method is to maximize the likelihood over Λ and Ψ assuming multivariate normality of the factors $(\boldsymbol{f}, \boldsymbol{u})$, which depends only on the factor space and is scale-invariant. This likelihood can have multiple local maxima; this possibility is usually ignored but `factanal` compares the fit found from five separate starting points. It is possible that the maximum likelihood solution will have some $\widehat{\psi}_{ii} = 0$, so the ith variable lies in the estimated factor space. Opinions differ as to what to do in this case (sometimes known as a *Heywood case*), but often it indicates a lack of data or inadequacy of the factor analysis model. (Bartholomew, 1987, Section 3.6, discusses possible reasons and actions.)

The data matrix X can be specified as the first argument to `factanal` as a matrix or data frame, or via a formula with a null left-hand side. Let us consider the data on Swiss cantons in matrix `swiss.x`.

```
> swiss.FA <- factanal(swiss.x, factors=2, method="mle")
Sums of squares of loadings:
 Factor1 Factor2
  1.9384  1.2923
    ....
Test of the hypothesis that 2 factors are sufficient
versus the alternative that more are required:
The chi square statistic is 2.97 on 1 degree of freedom.
The p-value is 0.0847
    ....
```

The 'Sums of squares of loadings' are the $\sum_i \lambda_{ij}^2$, which do depend on the rotation chosen, although their sum does not. The test statistic is a likelihood ratio test[2] of the fit, and may be used to help select the number of factors; here the fit is marginal with two factors, the maximum possible with five original variables. The `summary` method gives both more and less information:

```
> summary(swiss.FA)
Importance of factors:
               Factor1 Factor2
   SS loadings 1.93843 1.29230
Proportion Var 0.38769 0.25846
Cumulative Var 0.38769 0.64615

The degrees of freedom for the model is 1.

Uniquenesses:
 Agriculture Examination Education   Catholic Infant Mortality
     0.40764     0.19008   0.20264 0.00014068          0.96878
```

[2] with a Bartlett correction: see Bartholomew (1987, p. 46) or Lawley & Maxwell (1971, pp. 35–36). For a Heywood case (as here) Lawley & Maxwell (1971, p. 37) suggest the number of degrees of freedom should be increased by the number of variables with zero uniqueness.

```
Loadings:
                 Factor1 Factor2
     Agriculture -0.713   0.290
     Examination  0.777  -0.453
       Education  0.893
        Catholic -0.161   0.987
Infant Mortality          0.170
```

The function `loadings` gives just the loadings Λ, the smallest numbers in which have been suppressed in the print method. This output is not quite what it appears, as the original variables have been re-scaled to unit variance (with divisor n; equivalently, Σ in (11.8) has been replaced by the correlation matrix), and so the loading Λ and uniquenesses Ψ refer to the rescaled variables. Bartholomew (1987, p. 49) refers to this as the *standard* or *scale-invariant* form of the parameters Λ and Ψ. The component `scale` of the returned object relates[3] the output to the original variables.

The scale-invariant output does show that the `Catholic` variable is very nearly explained by the common factors, and `Infant Mortality` variable is poorly explained. In fact the uniqueness ψ_{ii} for the `Catholic` variable is being estimated as zero as tightening the convergence criteria shows:

```
> factanal(swiss.x, factors=2, method="mle",
    control=list(iter.max=100, unique.tol=1e-20))$uniq
 Agriculture Examination Education   Catholic Infant Mortality
     0.40764     0.19008   0.20264 2.8792e-09          0.96878
```

This confirms that the `Catholic` variable lies in the factor space, so we have a Heywood case. (In this example religion is a plausible candidate for a latent factor.) As the fit is marginal, it is instructive to consider $\Sigma - \widehat{\Lambda}\widehat{\Lambda}^T - \widehat{\Psi}$:

```
> A <- loadings(swiss.FA) %*% t(loadings(swiss.FA)) +
           diag(swiss.FA$uniq)
> round(cor(swiss.x) - A, 3)
            Agriculture Examination Education Catholic Mortality
Agriculture       0.000      -0.001     0.000        0    -0.145
Examination      -0.001       0.000     0.000        0     0.001
  Education       0.000       0.000     0.000        0    -0.054
   Catholic       0.000       0.000     0.000        0     0.000
  Mortality      -0.145       0.001    -0.054        0     0.000
```

Most of the lack of fit comes from just one correlation.

Note that unlike principal components, common factors are not generated one at a time, and the two-factor space will usually not contain the single-factor space. If we ask for one common factor (the default) rather than two we obtain

```
> swiss.FA1 <- factanal(swiss.x, method="mle")
> swiss.FA1
    ....
```

[3] This is a vector x such that original variable j was *divided* by x_j.

```
Test of the hypothesis that 1 factor is sufficient
versus the alternative that more are required:
The chi square statistic is 17.53 on 5 degrees of freedom.
The p-value is 0.00359
    ....
> summary(swiss.FA1)
    ....
Uniquenesses:
 Agriculture Examination Education Catholic Infant Mortality
     0.52866  2.2139e-06   0.51222  0.67184            0.987

Loadings:
                 Factor1
     Agriculture -0.687
     Examination  1.000
       Education  0.698
        Catholic -0.573
Infant Mortality -0.114
```

This time the `Examination` variable is fitted almost exactly. Thus the one-factor solution is the `Examination` variable, and it is easy to check that this is not in the subspace spanned by the two-factor solution.

It is hard to find examples in the literature for which a factor analysis model fits well: many do not give a measure of fit, or have failed to optimize the likelihood well enough and so failed to detect Heywood cases. We consider an example from Smith & Stanley (1983) as quoted by Bartholomew (1987, pp. 61–65)[4]. Six tests were give to 112 individuals, with covariance matrix

```
         general picture  blocks   maze reading   vocab
general   24.641   5.991  33.520  6.023  20.755  29.701
picture    5.991   6.700  18.137  1.782   4.936   7.204
 blocks   33.520  18.137 149.831 19.424  31.430  50.753
   maze    6.023   1.782  19.424 12.711   4.757   9.075
reading   20.755   4.936  31.430  4.757  52.604  66.762
  vocab   29.701   7.204  50.753  9.075  66.762 135.292
```

The tests were of general intelligence, picture completion, block design, mazes, reading comprehension and vocabulary. Both `factanal` and `princomp` can use covariance matrices as input using a `covlist` argument

```
> ability.cl <- list(cov=ability.cov, center=rep(0,6), n.obs=112)
> ability.FA <- factanal(covlist=ability.cl, method="mle")
> ability.FA
    ....
The chi square statistic is 75.18 on 9 degrees of freedom.
    ....
> ability.FA <- update(ability.FA, factors=2)
```

[4] Bartholomew gives both covariance and correlation matrices, but these are inconsistent. Neither are in the original paper.

```
> ability.FA
    ....
The chi square statistic is 6.11 on 4 degrees of freedom.
The p-value is 0.191
    ....
> summary(ability.FA)
    ....
Uniquenesses:
 general picture  blocks    maze  reading   vocab
 0.45523 0.58933 0.21817 0.76942 0.052463 0.33358

Loadings:
        Factor1 Factor2
general 0.501   0.542
picture 0.158   0.621
 blocks 0.208   0.859
   maze 0.110   0.467
reading 0.957   0.179
  vocab 0.785   0.222
```

Remember that the first variable is a composite measure: it seems that the first factor reflects verbal ability, the second spatial reasoning. The main lack of fit is that the correlation 0.308 between `picture` and `maze` is fitted as 0.193.

Factor rotations

There are many criteria for selecting rotations of the factors and loadings matrix; S-PLUS implements 12. There is an auxiliary function `rotate` which will rotate the fitted Λ according to one of these criteria, which is called via the `rotate` argument of `factanal`. The default `varimax` criterion is to maximize

$$\sum_{i,j}(d_{ij} - \overline{d}_{\cdot j})^2 \qquad \text{where} \qquad d_{ij} = \lambda_{ij}^2 / \textstyle\sum_j \lambda_{ij}^2 \tag{11.9}$$

and $\overline{d}_{\cdot j}$ is the mean of the d_{ij}. Thus the varimax criterion maximizes the sum over factors of the variances of the (normalized) squared loadings. The normalizing factors are the communalities which are invariant under orthogonal rotations.

The usual aim of a rotation is to achieve 'simple structure', that is a pattern of loadings which is easy to interpret with a few large and many small coefficients. The effect of normalization is to rescale the variables so the variance explained by the common factors is one for each variable. Normalization makes this rotation criterion scale-invariant; this is not the case for all the criteria, but the S-PLUS functions work with the scale-invariant loadings.

Not all the 'rotations' are orthogonal, for example the `promax` criterion seeks factors (such as arithmetical and verbal reasoning skills in psychology) that might be expected to be correlated. It is constructed by a least-squares fit of Λ to $Q = [|\lambda_{ij}|^4 \mathrm{sign}(\lambda_{ij})]$, and so tends to increase in magnitude large loadings relative to small ones. An initial value of Λ is needed, by default the varimax solution. For our example we have

```
> rotate(swiss.FA, rotation="promax")
Sums of squares of loadings:
[1] 1.9796 1.2511
    ....
Test of the hypothesis that 2 factors are sufficient
versus the alternative that more are required:
The chi square statistic is 2.97 on 1 degree of freedom.
The p-value is 0.0847
    ....
> rotate(loadings(swiss.FA), rotation="promax")
$rmat:
                      [,1]      [,2]
     Agriculture -0.720923  0.269493
     Examination  0.789990 -0.431096
       Education  0.892821  0.015316
        Catholic -0.189352  0.981838
Infant Mortality -0.053304  0.168463
    ....
```

Note that not all rotation methods produce objects of class `loadings` describing the rotated factors (the `rmat` component). so the print method for loadings is not always used, as here. Some care is needed to interpret these *oblique* rotations, as the rotated factors are no longer uncorrelated; for example (11.8) has to modified.

The `oblimin` criterion is another idea to produce oblique rotations: it minimizes the sum over all pairs of factors of the covariance between the squared loadings for those factors. We can illustrate this on the intelligence test data.

```
> loadings(rotate(ability.FA, rotation="oblimin"))
        Factor1 Factor2
general  0.379   0.513
picture          0.640
 blocks          0.887
   maze          0.483
reading  0.946
  vocab  0.757   0.137

Component/Factor Correlations:
        Factor1 Factor2
Factor1 1.000   0.356
Factor2 0.356   1.000
```

We can illustrate the oblique rotation graphically; see Figure 11.22.

```
par(pty="s")
L <- loadings(ability.FA)
eqscplot(L, xlim=c(0,1), ylim=c(0,1))
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])
```

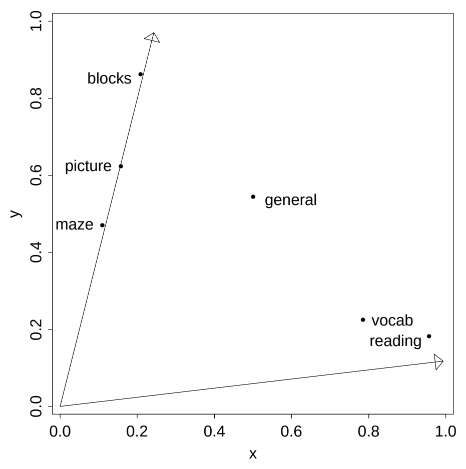

Figure 11.22: The loadings for the intelligence test data after varimax rotation, with the axes for the oblimin rotation shown as arrows.

It is also possible to rotate the loadings from a `princomp` fit, but care is needed as these are not the usual definition (Basilevsky, 1994, p. 258) of loadings for rotation.

Estimating the factor scores

Once factors have been fitted and perhaps interpreted, it may be of interest to estimate the scores of future individuals on the factors. Suppose that the observed vector of observations on a future individual is $\boldsymbol{x}_0$, and the sample mean is $\overline{\boldsymbol{x}}$. Bartlett suggested the use of (weighted) least squares, that is to regress the observations on the fitted loadings treating the $\boldsymbol{u}_i$ as random $N(0, \widehat{\Psi})$ terms and $\boldsymbol{f}$ as the parameters, giving

$$\widehat{\boldsymbol{f}} = [\widehat{\Lambda}^T \widehat{\Psi}^{-1} \widehat{\Lambda}]^{-1} \widehat{\Lambda}^T \widehat{\Psi}^{-1} (\boldsymbol{x}_0 - \overline{\boldsymbol{x}}) \tag{11.10}$$

On the other hand, Thomson noted that if the factors are treated as random variables (as they are in the statistical model),

$$E[\boldsymbol{f} \mid \boldsymbol{x}_0] = \Lambda^T [\Lambda\Lambda^T + \Psi]^{-1} (\boldsymbol{x}_0 - \boldsymbol{\mu}) = \Lambda^T \Sigma^{-1} (\boldsymbol{x}_0 - \overline{\boldsymbol{x}})$$

which suggests the use of

$$\widehat{\boldsymbol{f}} = \widehat{\Lambda}^T \widehat{\Sigma}^{-1} (\boldsymbol{x}_0 - \overline{\boldsymbol{x}}) \tag{11.11}$$

The function `predict.factanal` uses the `type` of `"weighted.ls"` for the Bartlett approach, and `"regression"` for the Thomson approach (its default). The scores for the data are the `scores` component of a `factanal` object, of type specified by the `type` argument to `factanal` (with Thomson scores as the default).

Comparisons with principal component analysis

Despite the many protestations in the literature of a fundamental difference, factor analysis continues to be confused with principal component analysis. Selecting the first k principal components fits the model (11.7) with criterion $\sum \|\boldsymbol{u}_i\|^2 = \sum_{i,j} u_{ij}^2$. By contrast, maximum-likelihood factor analysis uses the criterion

$$-\operatorname{trace} \Sigma^{-1} S + \log |\Sigma^{-1} S|$$

which matches the observed covariances (or correlations) S to $\Sigma = \Lambda\Lambda^T + \Psi$, and there is no assumption that the specific factors $\boldsymbol{u}$ need be small, just uncorrelated.

Nevertheless, we often find that if the variables have been suitably scaled, for example scaled to unit variance, factor analysis chooses Ψ so that either one (or more) $\widehat{\Psi}_{jj} = 0$ or the $\widehat{\Psi}_{jj}$ are fairly similar and quite small. Then either the factor analysis solution is a subspace containing one or more of the variables or it is likely to be rather similar to the subspace spanned by the first k principal components. (Theoretical support is given by Gower, 1966, and Rao, 1955.) Although in theory the interest in factor analysis is in explaining correlations not variances, this is belied by the output of factor analysis functions (`summary.factanal` indicates the importance of the factors by the proportions of variance explained) and by the way case studies are explained. (See, for example, Sections 8.3 and 8.4 of Reyment & Jöreskog, 1993.)

The fundamental difference is that factor analysis chooses the scaling of the variables via $\widehat{\Psi}$ whereas in principal component analysis the scaling must be chosen by the user. If the user chooses well, there may be little difference in the factors found.

Rotation of principal components

The usual aim of both PCA and factor analysis studies is to find an interpretable smaller set of new variables that explain the original variables. Factor rotation is a very appealing way to achieve interpretability, and it can also be applied in the space of the first m principal components. The S-PLUS function `rotate.princomp` applies rotation to the output of a `princomp` analysis. For example, if we varimax rotate the first two principal components of `ir.pca` (page 383 of the text) we find

```
> loadings(rotate(ir.pca, n=2))
        Comp. 1 Comp. 2 Comp. 3 Comp. 4
Sepal.L  0.596   0.324   0.709   0.191
Sepal.W          0.935  -0.331
Petal.L  0.569  -0.102  -0.219  -0.786
Petal.W  0.560          -0.583   0.580
```

Note that only the first two components have been rotated, although all four are displayed.

It is important to consider normalization carefully when applying rotation to a principal component analysis, which is not scale-invariant.

(a) Using argument `cor=T` to `princomp` ensures that the original variables are rescaled to unit variance when the principal components (PCs) are selected.

(b) The 'loadings' matrix given by `princomp` is the orthogonal matrix V which transforms the variables X to the principal components $Z = XV$, so $X = ZV^T$. This is not the usual loadings matrix considered for rotation in principal component analysis (Basilevsky, 1994, p. 258), although it is sometimes used (Jolliffe, 1986, §7.4). The loadings of a factor analysis correspond to a set of factors of unit variance; normalizing the principal components to unit variance corresponds to $X = Z^*A^T$ for $A = V\Lambda$ and $Z^* = Z\Lambda^{-1}$. where (as on page 304) Λ denotes the diagonal matrix of singular values. The matrix A is known as the *correlation loadings*. since A_{ij} is the correlation between the ith variable and the jth PC (provided the variables were normalized to unit variance). Orthogonal rotations of Z^* remain uncorrelated and correspond to orthogonal rotations of the correlation loadings.

(c) The S-PLUS default for rotations such as varimax is to normalize the loadings as at (11.9) so the sum of squares for each row (variable) is one. Thus (standardized) variables which are fitted poorly by the first m PCs are given the same weight as those which are fitted well. This seems undesirable for PCs (Basilevsky, 1994, p. 264), so it seems preferable not to normalize.

Taking these points into account we have

```
    > A <- loadings(ir.pca) %*% diag(ir.pca$sdev)
    > dimnames(A)[[2]] <- names(ir.pca$sdev)
    > B <- rotate(A[, 1:2], normalize=F)$rmat
    > print.loadings(B)
            Comp. 1 Comp. 2
    Sepal.L  0.963
    Sepal.W -0.153   0.981
    Petal.L  0.924  -0.350
    Petal.W  0.910  -0.342
```

which does have a clear interpretation as dividing the variables into two nearly disjoint groups. It does seem that one common use of rotation in both principal component and factor analysis is to cluster the original variables, which can of course also be done by a cluster analysis of X^T.

Chapter 12

Survival Analysis

12.1 Estimators of survival curves

In the text we concentrated on wholly non-parametric estimators of the survivor function S and cumulative hazard H; the resulting estimators were not smooth, indeed discontinuous. There are analogues of density estimation for survival data in which we seek smooth estimates of the survival function S, the density f or (especially) the hazard function h.

Kernel-based approaches

Kernel-based approaches are described by (Wand & Jones, 1995, §6.2.3, 6.3). The code `muhaz`[1] implements an approach by Mueller & Wang (1994). This does not work at all well for small datasets such as `gehan`, but we can apply it to the Australian AIDS dataset `Aids` by

```
attach(Aids2)
plot(muhaz(death-diag+0.9, status=="D"), n.est.grid=250)
```

This is slow (takes 30 seconds) and we had to refine the output grid to produce a fairly smooth result. The result shown in Figure 12.13 is unconvincing.

Likelihood-based approaches

Censoring is easy to incorporate in maximum-likelihood estimation; the likelihood is given by (12.1) on page 368. One approach to using a smooth estimator is to fit a very flexible parametric family and show the density / hazard / survivor function evaluated at the maximum likelihood estimate. This is the approach of the `logspline` library that we considered in Chapter 5 of these complements. Consider the `gehan` dataset.

[1] available on a good day for Unix from `http://odin.mdacc.tmc.edu/anonftp`.

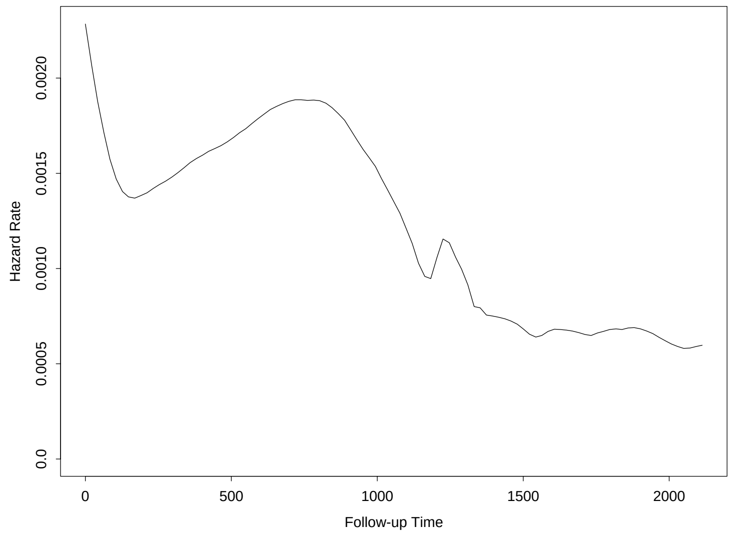

Figure 12.13: Hazard function fitted to the `Aids` dataset by `muhaz`.

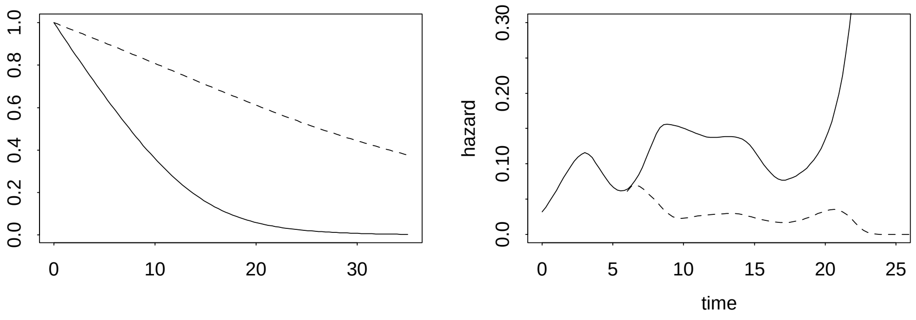

Figure 12.14: Smooth survival (left, by `logspline.fit`) and hazard (right, by `locfit`) fits to the `gehan` dataset. The solid line indicates the control group, the dashed line that receiving 6-MP.

```
library(logspline)  # logsplin on Windows < 6
g1 <- gehan[gehan$treat=="control",]
g2 <- gehan[gehan$treat=="6-MP",]
logspline.plot(
     logspline.fit(uncensored=g1[g1$cens==1,"time"],
                   right=g1[g1$cens==0,"time"], lbound=0),
               what="s", xlim=c(0,35))
g2.ls <- logspline.fit(uncensored=g2[g2$cens==1,"time"],
                       right=g2[g2$cens==0,"time"], lbound=0)
xx <- seq(0, 35, len=100)
lines(xx, 1 - plogspline(xx, g2.ls), lty=3)
```

As there is no function for plotting lines, we have to add the second group by hand. Small changes allow us to plot the density or hazard function.

Once again there is a local likelihood approach (see, for example Hjort, 1997) to hazard estimation, in which the terms are weighted by their proximity to t.

The full log-likelihood is

$$\sum_{t_i:\delta_i=1} \log h(t_i) - \sum_i \int_0^{t_i} h(u)\,\mathrm{d}u$$

and we insert weighting terms as before. This is implemented in Loader's library `locfit`: using a locally polynomial (by default quadratic) hazard.

```
library(locfit)
plot(locfit( ~ time, cens=1-cens, data=g1, family="hazard",
            alpha=0.5, xlim=c(0, 1e10)),
     xlim=c(0, 25), ylim=c(0, 0.3))
lines(locfit( ~ time, cens=1-cens, data=g2, family="hazard",
            alpha=0.5, xlim=c(0, 1e10)), lty=3)
```

The `xlim=c(0, 1e10)` argument sets a lower bound (only) on the support of the density.

Both there approaches can have difficulties in the right tail of the distribution, where uncensored observations may be rare. The right tail of a distribution fitted by `logspline.fit` necessarily is exponential beyond the last observation. In HEFT (Hazard Estimation with Flexible Tails; Kooperberg *et al.*, 1995a). a cubic spline model is used for the log hazard, but with two additional terms $\theta_1 \log t/(t+c)$ and $\theta_2 \log(t+c)$ where c is the upper quartile for the uncensored data. Then the space of fitted hazards includes the functions

$$h(t) = e^{\theta_0} t^{\theta_1} (t+c)^{\theta_2-\theta_1}$$

which includes the Weibull family and the Pareto density

$$f(t) = \frac{bc^b}{(t+c)^{b+1}}$$

for given c. Thus there is some hope that the tail behaviour can be captured within this parametric family. This is implemented in function `heft.fit` in library `heft`.[2] To illustrate this, let us consider the whole of the Australian AIDS dataset `Aids2`.

```
library(heft)
attach(Aids2)
aids.heft <- heft.fit(death-diag+0.9, status=="D")
heft.summary(aids.heft)
par(mfrow=c(2,2))
heft.plot(aids.heft, what="s",  ylim=c(0,1))
heft.plot(aids.heft)
```

This is rather slow (20 seconds). The sharp rise at 0 of the hazard reflects the small number of patients diagnosed at death. Note that this is the *marginal* hazard and its shape need not be at all similar to the hazard fitted in a (parametric or Cox) proportional hazards model.

[2] Not ported to S-PLUS 6.0 on Windows.

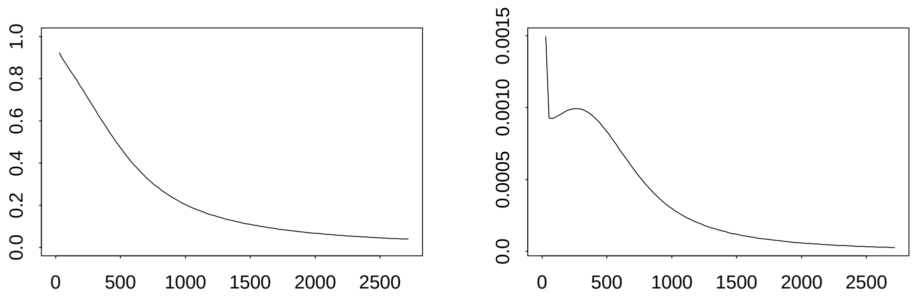

Figure 12.15: Survivor curve and hazard fitted to `Aids` by `heft.fit`.

12.5 Non-parametric models with covariates

There have been a number of approaches to model the effect of covariates on survival without a parametric model. Perhaps the simplest is a localized version of the Kaplan-Meier estimator

$$\hat{S}(t\,|\,x) = \prod_{t_i \leqslant t, \delta_i=1} \left[1 - \frac{w(x_i - x)}{\sum_{j \in R(t_i)} w(x_j - x)}\right]$$

which includes observations with weights depending on the proximity of their covariates to x. This does not smooth the survivor function, but the function `sm.survival` in library `sm` (Bowman & Azzalini, 1997) plots quantiles as a function of x by smoothing the inverse of the survival curve and computing quartiles of the smoothed fit. Following them, we can plot the median survival time after transplantation in the Stanford heart transplant data `heart` by

```
library(sm)
attach(heart[heart$transplant==1,])
sm.survival(age+48, log10(stop - start), event, h=5, p=0.50)
detach()
```

This shows some evidence of a decline with age, which can also be seen in the Cox analysis.

The local likelihood approach easily generalizes to localizing in covariate space too: in `locfit` this is requested by adding covariate terms to the right-hand-side of the formula.

```
library(locfit)
attach(heart[heart$transplant==1,])
td <- stop - start; Age <- age+48
plot(locfit(~ td + Age, cens=1-event, scale=0, alpha=0.5, family="hazard",
              xlim=list(td=c(0,1e10)), flim=list(td=c(0,365))),
     type="persp")
```

Gray (1996, 1994) takes a similar but less formal approach, using `loess` to smooth a discretized version of the problem. This is implemented in his function `hazcov` in library `hazcov`. First the data are grouped on the covariate values,

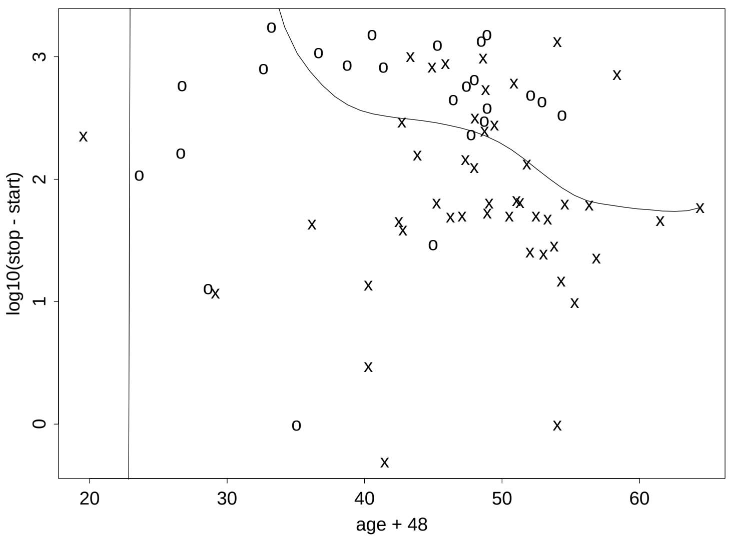

Figure 12.16: Median survival time for the Stanford heart transplant data by `sm.survival`.

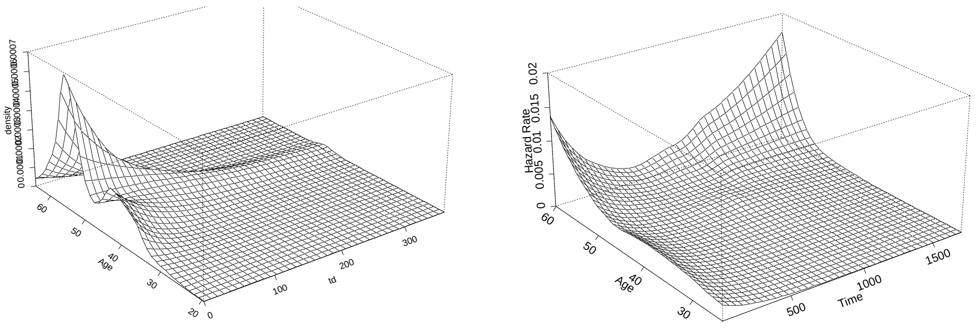

Figure 12.17: Smooth hazard functions (in days) as a function of age post-transplantation in the Stanford heart-transplant study. **Left:** by `locfit` and **right:** by `hazcov` using local scoring.

using quantiles of the marginal distributions or factor levels. Then time is divided into intervals and the number of events and total follow-up time computed for each interval for each covariate combination. In the default method described in the 1996 paper, the numbers of events and the follow-up totals are separately smoothed using `loess` function, and the hazard estimate formed by taking ratios. We can try this by

```
library(hazcov)
heart.hc <- hazcov(Surv(td, event) ~ Age, span=0.5)
plot(heart.hc)
persp.hazcov(Hazard.Rate ~ Time*Age, heart.hc)
```

The `loess` span was chosen by guesswork. Gray describes an approximate version of C_p to help select the span which we can use by

```
heart.50 <- hazcov(Surv(td, event) ~ Age, span=0.5,
                   trace.hat="exact")
for(alpha in seq(0.1, 1, 0.1))
{
 heart.tmp <- hazcov(Surv(td, event) ~ Age, span=alpha,
                     trace.hat="exact")
 print(c(alpha, wcp(heart.tmp, heart.50)))
}
```

This indicates a minimum at $\alpha = 0.2$, but very little difference over the range $[0.2, 0.5]$.

The alternative method (Gray, 1994: 'local scoring' invoked by `ls=T`), the counts are viewed a independent Poisson variates with mean total follow-up times hazard, and a local log-linear Poisson GLM is fitted by IWLS, using `loess` to smooth the log-hazard estimates.

```
heart.hc <- hazcov(Surv(td, event) ~ Age, span=0.5, ls=T)
plot(heart.hc)
persp.hazcov(Hazard.Rate ~ Time*Age, heart.hc)
```

Spline approaches

HARE (HAzard Rate Estimation; Kooperberg *et al.*, 1995a) fits a linear tensor-spline model for the log hazard function conditional on covariates, that is $\log h(t \mid x) = \eta(t, x; \theta)$ is a MARS-like function of (t, x) jointly. The fitting procedure is similar to that for `logspline` and `lspec`: an initial set of knots is chosen, the log-likelihood is maximized given the knots by a Newton algorithm, and knots and terms are added and deleted in a stepwise fashion. Finally, the model returned is that amongst those considered that maximizes a penalized likelihood (by default with penalty $\log n$ times the number of parameters).

It remains to describe just what structures are allowed for $\eta(t, x)$. This is a linear combination of linear spline basis functions and their pairwise products, that is a linear combination of terms like $c, t, (t-c)_+, x_j, (x_j - c)_+, tx_j, (tx_j - c)_+, x_j x_k, (x_j x_k - c)_+$ where the c are generic constants. The product terms are restricted to products of simple terms already in the model, and wherever a non-linear term occurs, that term also occurs with the non-linear term replaced by a linear term in the same variable. Thus this is just a MARS model in the $p+1$ variables restricted to pairwise interactions.

The model for the hazard function will be a proportional hazards model if (and only if) there are no products between t and covariate terms. In any case it has a rather restricted ability to model non-constant hazard functions, and it is recommended to transform time to make the marginal distribution close to exponential (with constant hazard) before applying HARE.

HARE is implemented in library `hare`[3] by function `hare.fit`. The paper contains an analysis of the dataset `cancer.vet` which we can reproduce by

[3] Not ported to S-PLUS 6.0 on Windows.

```
#  VA is constructed on page 363
> attach(VA)
> library(HARE)
> options(contrasts=c("contr.treatment", "contr.poly"))
> VAx <- model.matrix( ~ treat+age+Karn+cell+prior, VA)[,-1]
> VA.hare <- hare.fit(stime, status, VAx)
> hare.summary(VA.hare)
    ....
the present optimal number of dimensions is 9.
penalty(AIC) was the default: BIC=log(samplesize): log(137)=4.92

  dim1           dim2           beta       SE         Wald
Constant                     -9.83e+00  2.26e+00   -4.35
Co-3  linear                  2.50e-01  1.08e-01    2.31
Co-5  linear                  2.43e+00  4.72e-01    5.15
Co-4  linear                 -1.39e+00  6.35e-01   -2.20
Time  1.56e+02 Co-5  linear  -1.25e-02  4.50e-03   -2.77
Time  1.56e+02                2.45e-02  5.84e-03    4.20
Co-3  2.00e+01               -2.60e-01  1.08e-01   -2.41
Co-3  linear   Co-4  linear   3.87e-02  1.12e-02    3.46
Time  1.56e+02 Co-3  linear  -4.33e-04  9.58e-05   -4.52
```

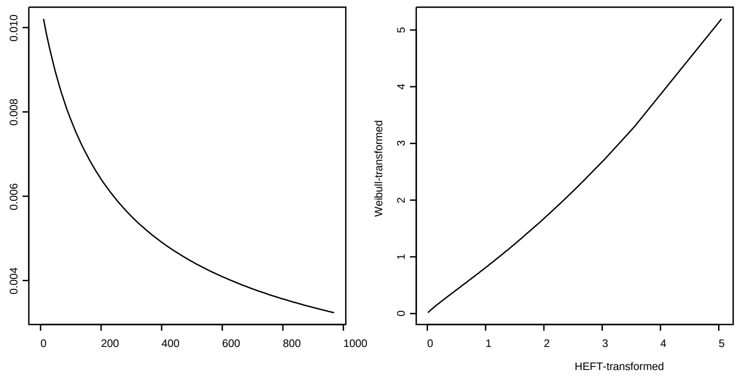

Figure 12.18: The marginal distribution of lifetime in the `cancer.vet` dataset. **Left:** Hazard as fitted by `heft.fit`. **Right:** Time as transformed by the distribution fitted by `heft.fit` and by a fitted Weibull distribution.

We found that an exponential model for the residual hazard was adequate, but Kooperberg *et al.* (1995a) explore the marginal distribution by HEFT and conclude that the time-scale could usefully be transformed. They used

```
library(HEFT)
VA.heft <- heft.fit(stime, status, leftlog=0)
heft.plot(VA.heft, what="h")
nstime <- -log(1 - pheft(stime, VA.heft))
```

In fact the transformation used is close to that from fitting a Weibull distribution

```
survreg(Surv(stime, status) ~ 1, data=VA)
```

```
   ....
Coefficients:
 (Intercept)
      4.7931

Dispersion (scale) = 1.1736

plot(sort(nstime),
     -log(1-pweibull(sort(stime), 1/1.1736, exp(4.9731))),
     type="l", xlab="HEFT-transformed", ylab="Weibull-transformed")
```

It does seem undesirable to ignore the highly significant covariate effects in making such a transformation; this is illustrated in this example by the change in the Weibull shape parameter from 1.1736 to 0.928 (page 389) on fitting linear terms in the survival regression model.

Having transformed time, we can re-fit the model.

```
> VA.hare2 <- hare.fit(nstime, status, VAx)
hare.summary(VA.hare2)
the present optimal number of dimensions is 10.
penalty(AIC) was the default: BIC=log(samplesize): log(137)=4.92

  dim1           dim2            beta       SE        Wald
Constant                      -7.06e+00  2.60e+00   -2.72
Co-3  linear                   2.72e-01  1.10e-01    2.47
Co-5  linear                   5.54e+00  1.15e+00    4.81
Time  2.67e+00                 2.24e+00  6.22e-01    3.60
Time  2.67e+00 Co-5  linear   -2.00e+00  5.40e-01   -3.70
Time  2.67e+00 Co-3  linear   -4.21e-02  9.54e-03   -4.42
Co-4  linear                  -1.16e+00  6.53e-01   -1.77
Co-3  8.50e+01                -2.73e-01  1.17e-01   -2.33
Co-3  linear   Co-4  linear    3.39e-02  1.15e-02    2.94
Co-3  2.00e+01                -2.31e-01  1.08e-01   -2.13
```

Allowing for the time transformation, the fitted model is quite similar. Covariate 3 is the Karnofsky score, and 4 and 5 are the contrasts of cell type adeno and small with squamous. It is not desirable to have a variable selection process that is so dependent on the coding of the factor covariates.

This example was used to illustrate the advantages of HARE / HEFT methodology by their authors, but seems rather to show up its limitations. We have already seen that the *marginal* transformation of time is quite different from that suggested for the *conditional* distribution. In our analysis via Cox proportional hazards models we found support for models with interactions where the main effects are not significant (such models will never be found by a forward selection procedure such as used by HARE) and the suspicion of time-dependence of such interactions (which would need a time cross covariate cross covariate interaction which HARE excludes).

12.6 Expected survival rates

In medical applications we may want to compare survival rates to those of a standard population, perhaps to standardize the experience of the population under study. As the survival experience of the general population changes with calendar time, this must be taken into account.

Unfortunately, there are differences between versions in how calendar time is recorded between the versions of the survival analysis functions: the version in S-PLUS uses modified versions of functions from the `chron` library whereas `survival5` uses the format of Therneau's library `date` (obtainable from `statlib`). Both record dates in days since 1 Jan 1960, but with class `"dates"` and `"date"`) respectively. For the S-PLUS version the easiest way to specify or print calendar dates is the function `dates`; for datasets such as `aids.dat` with numerical day, month and year data the function `julian` may be useful.

For a cohort study, expected survival is often added to a plot of survivor curves. The function `survexp` is usually used with a formula generated by `ratetable`. The optional argument `times` specifies a vector at which to evaluate survival, by default for all follow times. For example, we could add expected survival for 65-year old US white males to the left plot of Figure 12.9 by

```
year <- dates("7/1/91")
expect <- survexp(~ ratetable(sex="male", year=year, age=65*365.25),
    times = seq(0, 1400, 30),  ratetable=survexp.uswhite)
lines(expect$time, expect$surv, lty=4)
```

but as the patients are seriously ill, the comparison is not very useful. As the inbuilt rate tables are in units of days, all of `year`, `age` and `times` must be in days.

Entry and date times can be specified as vectors, when the average survival for the cohort is returned. For individual expected survival, we can use the same form with `cohort=F`, perhaps evaluated at death time.

Some explanation of the averaging used is needed in the cohort case. We can use the cumulative hazard function $H_i(t)$ and survivor function $S_i(t)$ of the exact match (on age and sex) to individual i. There are three possibilities, which differ in the assumptions on what follow-up would have been.

1. The formula has no response. Then the function returns the average of $S_i(t)$. This corresponds to assuming complete follow-up.
2. The death times are given as the response. Then the $H_i(t)$ are averaged over the cases at risk at time t to from a cohort cumulative hazard function and converted to a survivor function.
3. The potential censoring times for each case are given as the response, and `conditional=F`, when the weights in the cohort cumulative hazard function are computed as $S_i(t)I($potentially in study at $t)$. This corresponds to assuming follow-up until the end of the study.

The first is most useful for forecasting, the other two for comparing with the study outcome. Thus to compare the survival in Figure 12.9 to matched males of the same ages we might use

```
expect <- survexp(stop ~ ratetable(sex=1, year=year*365.25,
    age=(age+48)*365.25),  times = seq(0, 1400, 30),
    ratetable = survexp.uswhite, data = heart,
    subset = diff(c(id, 0)) != 0, cohort = T, conditional = T)
lines(expect$time, expect$surv, lty=4)
```

We do need to extract the second record corresponding to transplanted subjects to get the correct death/censoring time for the cohort matching.

It is possible to use the fit from a `coxph` model in place of the inbuilt ratetables to compare the present study to an earlier one.

Chapter 13

Time Series

13.1 Second-order summaries

Spectral analysis

The most common approach to estimating a spectral density is to use a kernel smoother, as implemented by `spectrum`, but there are alternatives, including the use of fitted high-order AR processes (page 448). One promising line is to use some of the alternative methods of estimating a probability density function function, since a spectral density is just a finite multiple of a pdf.

The library `lspec`[1] by Charles Kooperberg implements the logspline approach described in Section 5.6 of these complements. Its application to spectral estimation is described in Kooperberg *et al.* (1995b); note that it is able to estimate mixed spectra that have purely periodic components. We will illustrate this by estimating the spectra of our running examples `lh` and `deaths` as well as the `accdeaths` and `nottem` series.

For `lh` we have

```
> library(lspec)
> lh.ls <- lspec.fit(lh)
> lspec.summary(lh.ls)
 Logspline Spectral Estimation
 =============================
 The fit was obtained by the command:
 lspec.fit(data = lh)
 A spline with 3 knots, was fitted; there were no lines in the model.
 The log-likelihood of the model was 60.25 which corresponds to an
 AIC value of -110.96 .

 The program went though 1 updown cycles, and reached a stable
 solution.  Both penalty (AIC) and minmass were the default
 values. For penalty this was log(n)=log( 24 )= 3.18 (as in BIC)
 and for minmass this was 0.0329.  The locations of the knots were:
 1.178 2.749 3.142
> lspec.plot(lh.ls, log="y")
> lspec.plot(lh.ls, what="p")
```

[1] This is particularly hard to port as it uses calls to ompiled code inconsistently.

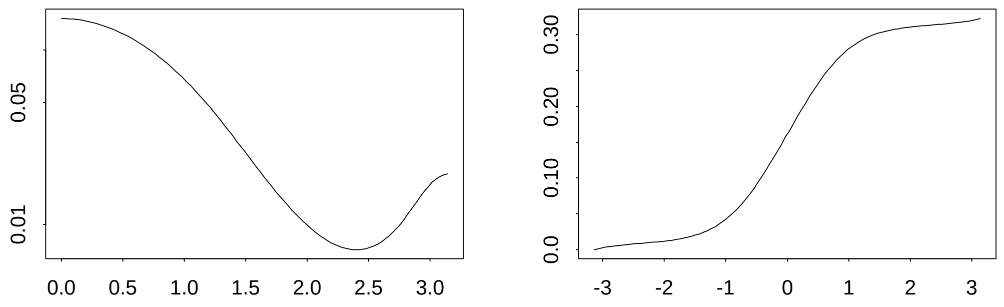

Figure 13.22: Spectral density (left) and cumulative spectral distribution function (right) for the series `lh` computed by library `lspec`.

(Figure 13.22). Note that rather different conventions are used for the spectrum, which is taken to run over $(-\pi, \pi]$ rather than in cycles, and the amplitude is given in the normal units, not decibels. The spectral density and cumulative spectrum can be found by `dlspec` and `plspec` respectively.

```
deaths.ls <- lspec.fit(deaths)
lspec.plot(deaths.ls, log="y", main="deaths")
lspec.plot(deaths.ls, what="p")
accdeaths.ls <- lspec.fit(accdeaths)
lspec.plot(accdeaths.ls, log="y", main="accdeaths")
lspec.plot(accdeaths.ls, what="p")
nott.ls <- lspec.fit(window(nottem, end=c(1936,12)))
lspec.plot(nott.ls, log="y", main="nottem")
lspec.plot(nott.ls, what="p")
```

(Figure 13.23). Note how `lspec.fit` finds the discrete component at frequency $\pi/12$ in all three cases, but is fooled by harmonics in the last two. We can allow `lspec.fit` to fit more discrete components by reducing the value of its argument `minmass` (whose default can be found from `lspec.summary`). In the `accdeaths` example we can pick up all but one of the harmonics by

```
lspec.plot(lspec.fit(accdeaths, minmass=7000), log="y")
lspec.plot(lspec.fit(accdeaths, minmass=1000), log="y")
```

but reducing `minmass` introduces discrete components at non-harmonic frequencies (Figure 13.24).

The functions `clspec` and `rlspec` compute the autocovariance (or autocorrelation) sequence corresponding to the fitted spectrum and simulate a Gaussian time series with the fitted spectrum respectively.

13.7 Multiple time series

The second-order time-domain properties of multiple time series were covered in Section 13.1. The function `ar` will fit AR models to multiple time series, but ARIMA fitting is confined to univariate series. Let $\boldsymbol{X}_t$ denote a multiple time

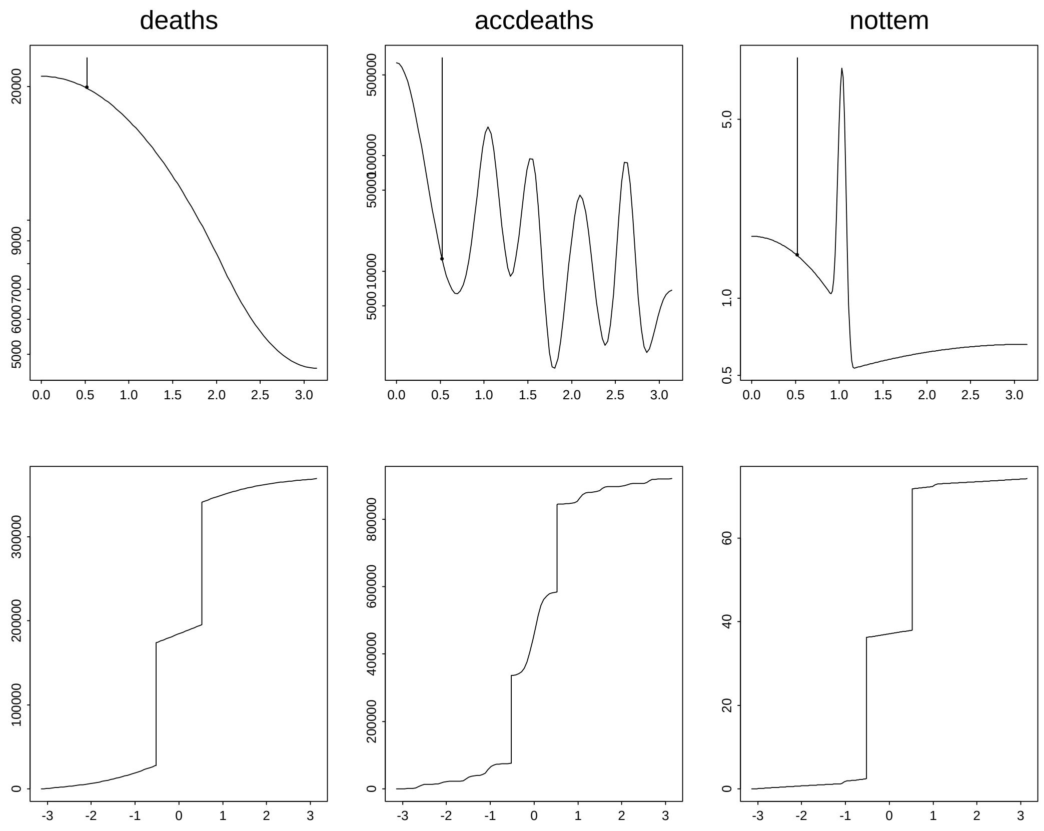

Figure 13.23: Spectral density (top) and cumulative spectral distribution function (bottom) for the series `deaths`, `accdeaths` and `nottem`.

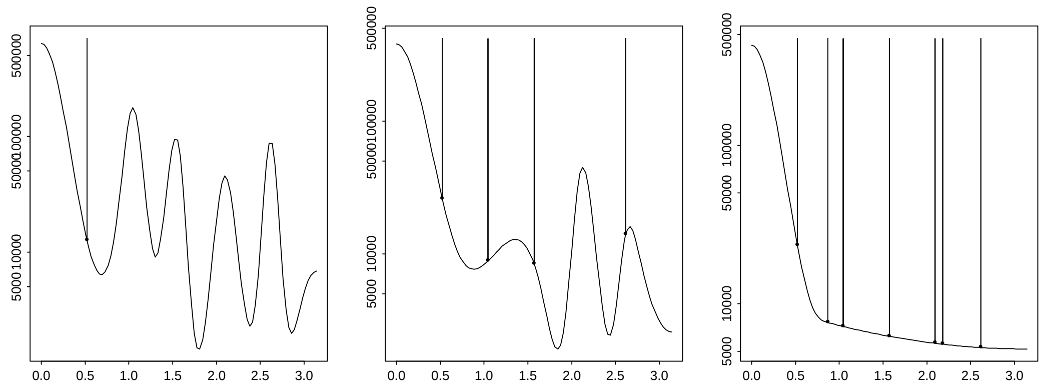

Figure 13.24: Spectra for `nottem` with `minmass` as (left to right) 77 000, 7000 and 1000.

series, and ϵ_t a correlated sequence of identically distributed random variables. Then a vector AR(p) process is of the form

$$\boldsymbol{X}_t = \sum_i^p A_i \boldsymbol{X}_{t-i} + \boldsymbol{\epsilon}_t$$

for matrices A_i. Further, the components of $\boldsymbol{\epsilon}_t$ may be correlated, so we will assume that this has covariance matrix Σ. Again there is a condition on the coefficients, that

$$\text{det}\,[I - \sum_1^p A_i z^i] \neq 0 \text{ for all } |z| \leqslant 1$$

The parameters can be estimated by solving the multiple version of the Yule–Walker equations (Brockwell & Davis, 1991, §11.5), and this is used by `ar.yw`, the function called by `ar`. (The other method, `ar.burg`, also handles multiple series.)

Spectral analysis for multiple time series

The definitions of the spectral density can easily be extended to a pair of series. The cross-covariance is expressed by

$$\gamma_{ij}(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{i\omega t}\, \mathrm{d}F_{ij}(\omega)$$

for a finite complex measure on $(-\pi, \pi]$, which will often have a density f_{ij} so that

$$\gamma_{ij}(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{i\omega t} f_{ij}(\omega)\, \mathrm{d}\omega$$

and

$$f_{ij}(\omega) = \sum_{-\infty}^{\infty} \gamma_{ij}(t) e^{-i\omega t}$$

Note that since $\gamma_{ij}(t)$ is not necessarily symmetric, the sign of the frequency becomes important, and f_{ij} is complex. Conventionally it is written as $c_{ij}(\omega) - i\,q_{ij}(\omega)$ where c is the *co-spectrum* and q is the *quadrature spectrum*. Alternatively we can consider the amplitude $a_{ij}(\omega)$ and phase $\phi_{ij}(\omega)$ of $f_{ij}(\omega)$. Rather than use the amplitude directly, it is usual to work with the *coherence*

$$b_{ij}(\omega) = \frac{a_{ij}(\omega)}{\sqrt{f_{ii}(\omega) f_{jj}(\omega)}}$$

which lies between zero and one.

The *cross-periodogram* is

$$I_{ij}(\omega) = \left[\sum_{s=1}^{n} e^{-i\omega s} X_i(s) \sum_{t=1}^{n} e^{i\omega t} X_j(t) \right] \Big/ n$$

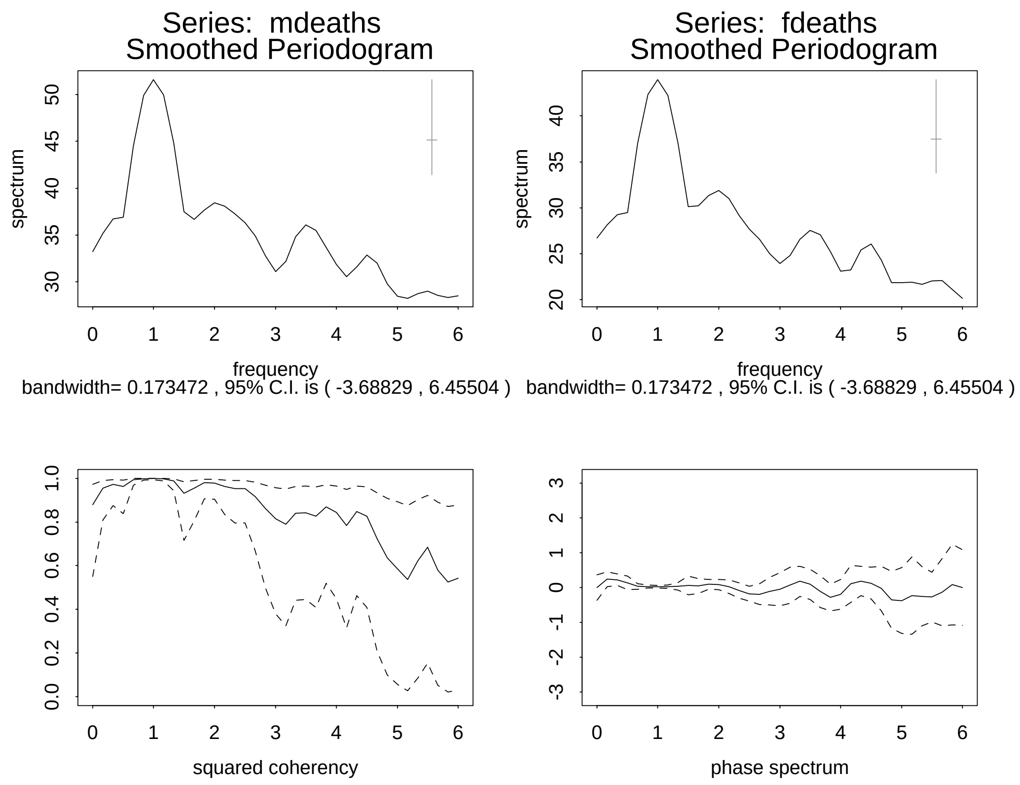

Figure 13.25: Coherence and phase spectra for the two deaths series, with 95% pointwise confidence intervals.

and is a complex quantity. It is useless as an estimator of the amplitude spectrum, since if we define

$$J_i(\omega) = \sum_{s=1}^{n} e^{-i\omega s} X_i(s)$$

then

$$|I_{ij}(\omega)|/\sqrt{I_{ii}(\omega)I_{jj}(\omega)} = |J_i(\omega)J_j(\omega)^*|/|J_i(\omega)|\,|J_j(\omega)| = 1$$

but smoothed versions can provide sensible estimators of both the coherence and phase.

The function `spec.pgram` will compute the coherence and phase spectra given a multiple time series. The results are shown in Figure 13.25.

```
spectrum(mdeaths, spans=c(3,3))
spectrum(fdeaths, spans=c(3,3))
mfdeaths.spc <- spec.pgram(ts.union(mdeaths, fdeaths),
   spans=c(3,3))
plot(mfdeaths.spc$freq, mfdeaths.spc$coh, type="l",
   ylim=c(0,1), xlab="squared coherency", ylab="")
gg <- 2/mfdeaths.spc$df
se <- sqrt(gg/2)
coh <- sqrt(mfdeaths.spc$coh)
```

```
lines(mfdeaths.spc$freq, (tanh(atanh(coh) + 1.96*se))^2, lty=3)
lines(mfdeaths.spc$freq, (pmax(0, tanh(atanh(coh) - 1.96*se)))^2, lty=3)
plot(mfdeaths.spc$freq, mfdeaths.spc$phase, type="l",
   ylim=c(-pi, pi), xlab="phase spectrum", ylab="")
cl <- asin( pmin( 0.9999, qt(0.95, 2/gg-2)*
   sqrt(gg*(coh^{-2} - 1)/(2*(1-gg)) ) ) )
lines(mfdeaths.spc$freq, mfdeaths.spc$phase + cl, lty=3)
lines(mfdeaths.spc$freq, mfdeaths.spc$phase - cl, lty=3)
```

These confidence intervals follow Bloomfield (1976, §8.5). At the frequency of 1/year there is a strong signal common to both series, so the coherence is high and both coherence and phase are determined very precisely. At high frequencies there is little information, and the phase cannot be fixed at all precisely.

It is helpful to consider what happens if the series are not aligned:

```
mfdeaths.spc <- spec.pgram(ts.union(mdeaths, lag(fdeaths, 4)),
   spans=c(3,3))
plot(mfdeaths.spc$freq, mfdeaths.spc$coh, type="l",
   ylim=c(0,1), xlab="coherency", ylab="")
gg <- 2/mfdeaths.spc$df
se <- sqrt(gg/2)
coh <- sqrt(mfdeaths.spc$coh)
lines(mfdeaths.spc$freq, (tanh(atanh(coh) + 1.96*se))^2, lty=3)
lines(mfdeaths.spc$freq, (pmax(0, tanh(atanh(coh) - 1.96*se)))^2, lty=3)
phase <- (mfdeaths.spc$phase + pi)%%(2*pi) - pi
plot(mfdeaths.spc$freq, phase, type="l",
   ylim=c(-pi, pi), xlab="phase spectrum", ylab="")
cl <- asin( pmin( 0.9999, qt(0.95, 2/gg-2)*
   sqrt(gg*(mfdeaths.spc$coh^{-2} - 1)/(2*(1-gg)) ) ) )
lines(mfdeaths.spc$freq, phase + cl, lty=3)
lines(mfdeaths.spc$freq, phase - cl, lty=3)
```

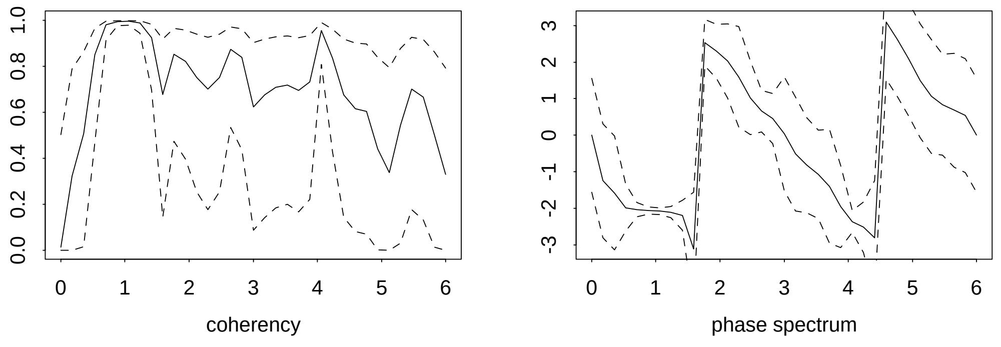

Figure 13.26: Coherence and phase spectra for the re-aligned deaths series, with 95% pointwise confidence intervals.

The results are shown in Figure 13.26. The phase has an added component of slope $2\pi * 4$, since if $X_2(t) = X_1(t-\tau)$,

$$\gamma_{12}(t) = \gamma_{11}(t+\tau), \qquad f_{11}(\omega) = f_{11}(\omega)e^{-i\tau\omega}$$

For more than two series we can consider all the pairwise coherence and phase spectra, which are returned by `spec.pgram`.

13.8 Other time-series functions

S-PLUS has a number of time-series functions which are used less frequently and we have not yet discussed. This section is only cursory.

Many of the other functions implement various aspects of filtering, that is converting one times series into another while emphasising some features and de-emphasising others. A linear filter is of the form

$$Y_t = \sum_j a_j X_{t-j}$$

which is implemented by the function `filter`. The coefficients are supplied, and it is assumed that they are non-zero only for $j \geqslant 0$ (`sides=1`) or $-m \leqslant j \leqslant m$ (`sides=2`, the default). A linear filter affects the spectrum by

$$f_Y(\omega) = \left|\sum a_s e^{-is\omega}\right|^2 f_X(\omega)$$

and filters are often described by aspects of the gain function $|\sum a_s e^{-is\omega}|$. Kernel smoothers such as `ksmooth` are linear filters when applied to regularly-spaced time series.

Another way to define a linear filter is recursively (as in exponential smoothing), and this can be done by `filter`, using

$$Y_t = \sum_{s=1}^{\ell} a_s Y_{t-s}$$

in which case ℓ initial values must be specified by the argument `init`.

Converting an ARIMA process to the innovations process ϵ is one sort of recursive filtering, implemented by the function `arima.filt`.

A large number of smoothing operations such as `lowess` can be regarded as filters, but they are non-linear. The functions `acm.filt`, `acm.ave` and `acm.smo` provide filters resistant to outliers.

Complex demodulation is a technique to extract approximately periodic components from a time series. It is discussed in detail by Bloomfield (1976, Chapter 7) and implemented by the function `demod`.

Some time series exhibit correlations which never decay exponentially, as they would for an ARMA process. One way to model these phenomena is fractional differencing (Brockwell & Davis, 1991, §13.2). Suppose we expand ∇^d by a binomial expansion:

$$\nabla^d = \sum_{j=0}^{\infty} \frac{\Gamma(j-d)}{\Gamma(j+1)\Gamma(-d)} B^j$$

and use the right-hand side as the definition for non-integer d. This will only make sense if the series defining $\nabla^d X_t$ is mean-square convergent. A fractional ARIMA process is defined for $d \in (-0.5, 0.5)$ by the assumption that $\nabla^d X_t$ is an ARMA(p, q) process, so

$$\phi(B)\nabla^d X = \theta(B)\epsilon, \qquad \text{so} \qquad \phi(B)X = \theta(B)\nabla^{-d}\epsilon$$

and we can consider it also as an ARMA(p, q) process with fractionally integrated noise. The spectral density is of the form

$$f(\omega) = \sigma^s \left| \frac{\theta(e^{-i\omega})}{\phi(e^{-i\omega})} \right|^2 \times |1 - e^{-i\omega}|^{-2d}$$

and the behaviour as ω^{-2d} at the origin will help identify the presence of fractional differencing.

The functions `arima.fracdiff` and `arima.fracdiff.sim` implement fractionally-differenced ARIMA processes.

Chapter 14

Spatial Statistics

14.5 Module S+SpatialStats

The first release of the S-PLUS module S+SpatialStats was released in mid-1996. That has a comprehensive manual (published as Kaluzny & Vega, 1997), which we do not aim to duplicate, but rather to show how our examples in Chapter 14 can be done using S+SpatialStats.

The module S+SpatialStats is attached and made operational by

```
module(spatial)
```

which we will assume has been done. Unfortunately the name is the same as our library (as are some of the function names); modules take priority over libraries.

Kriging

The kriging functions use a slight extension of the model formula language. The function `loc` is used to specify the two spatial coordinates of the points, which are used to find the covariance matrix in kriging. Universal kriging is specified by adding other terms to form a linear model. Thus we can specify the model used in the bottom row of Figure 14.5 by

```
> topo.kr <- krige(z ~ loc(x, y) + x + y + x^2 + x*y + y^2,
     data=topo, covfun=exp.cov, range=0.7, sill=770)
> topo.kr
    ....
Coefficients:
 constant       x       y    x^2     xy    y^2
    808.3 -12.896 -64.486 62.137 1.6332 6.3442
    ....
> prsurf <- predict(topo.kr, se.fit = T,
     grid = list(x=c(0, 6.5, 50), y=c(0, 6.5, 50)))
> topo.plt1 <- contourplot(fit ~ x*y, data=prsurf, pretty=F,
     at=seq(700, 1000, 25), aspect=1,
     panel = function(...){
        panel.contourplot(...)
```

```
          points(topo)
       })
    > topo.plt2 <- contourplot(se.fit ~ x*y, data=prsurf, pretty=F,
       at=c(20, 25), aspect=1)
    > print(topo.plt1, split=c(1,1,2,1), more=T)
    > print(topo.plt2, split=c(2,1,2,1))
```

(The `sill` value is explained below.) We can of course obtain a least-squares trend surface by giving a covariance function that drops to zero immediately, for example `exp.cov` with `range = 0`, but there seems no simple way to obtain a trend surface fitted by GLS. The `predict` method for `krige` objects takes either a `newdata` argument or a `grid` argument as used here. The `grid` argument must be a list with two components with names matching those given to `loc` and specifying the minimum, maximum and number of points. (This is passed to `expand.grid` to compute a data frame for `newdata`.)

Analogues of the fits shown in Figure 14.7 may be obtained by

```
topo.kr2 <- krige(z ~ loc(x, y) + x + y + x^2 + x*y + y^2,
     data = topo, covfun = gauss.cov,
     range = 1, sill = 600, nugget = 100)
topo.kr3 <- krige(z ~ loc(x, y), data = topo,
     covfun = gauss.cov, range = 2, sill = 6500, nugget = 100)
```

Various functions are provided to fit variograms and correlograms. We start by fitting a variogram to the original data.

```
topo.var <- variogram(z ~ loc(x, y),  data=topo)
model.variogram(topo.var, gauss.vgram, range=2,
   sill=6500, nugget=100)
```

The function `model.variogram` plots the variogram object (which may also be plotted directly) and draws a theoretical variogram. It then prompts the user to alter the parameters of the variogram to obtain a good fit by eye. It this case `range = 3.5` seems indicated. The parametrization is that `nugget` is the increment at the origin, and `sill` is the change over the range of increase of the variogram. (In geostatistical circles the sum of ‘nugget’ and ‘sill’ is called the sill.) Thus the `alph` of our covariance functions is `nugget/(sill + nugget)`.

There are functions `correlogram` and `covariogram` which can be used in the same way (including with `model.variogram`).

```
topo.cov <- covariogram(z ~ loc(x, y),  data=topo)
model.variogram(topo.cov, gauss.cov, range=2,
   sill=4000, nugget=2000)
```

We can now explain how we chose the the parameters of the exponential covariance in the first plot. An object of class `"krige"` contains residuals, so we can use

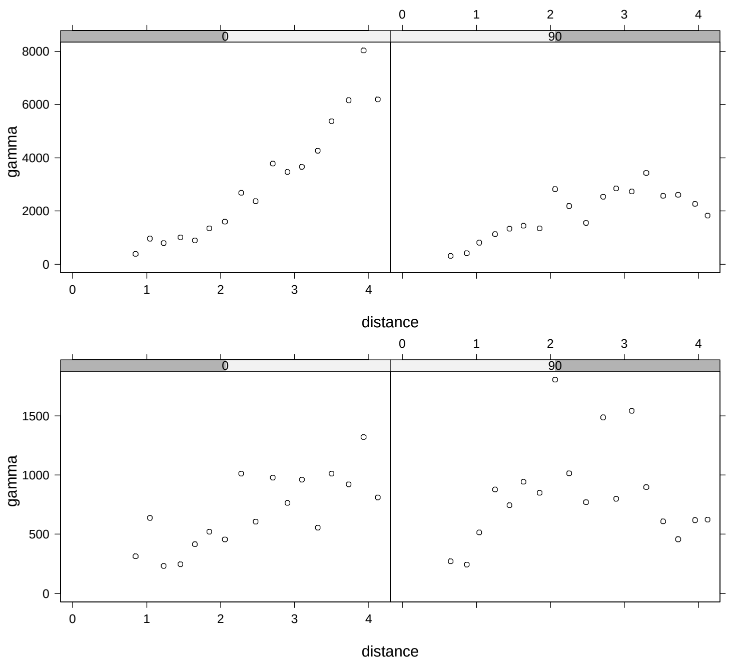

Figure 14.10: Directional variograms for the `topo` dataset. The top pair is for the raw data, the bottom pair of residuals from a quadratic trend surface. The left plots are vertical variograms, the right plots are horizontal ones. (The strip coverage is misleading, only showing the positive part of the angular tolerance.)

```
topo.ls <- krige(z ~ loc(x, y) + x + y + x^2 + x*y + y^2,
     data=topo, covfun=exp.cov, range=0)
topo.res <- residuals(topo.ls)
topo.var <- variogram(topo.res ~ loc(x, y),  data=topo)
model.variogram(topo.var, exp.vgram, range=1, sill=1000)
```

This suggests a sill of about 800. The kriging predictions do not depend on the sill, and our `spatial` library relies on this to work throughout with correlograms and to fit the overall scale factor when plotting the standard errors. Knowledge of our code allowed us to read off the value 770. It would be a good idea to repeat the forming of the residuals, this time from the GLS trend surface. We can choose the covariogram for the Gaussian case in the same way.

```
topo.var <- covariogram(topo.res ~ loc(x, y),  data=topo)
model.variogram(topo.var, gauss.cov, range=1, sill=210,
   nugget=90)
```

Spatial anisotropy

The geostatistical functions in S+SPATIALSTATS have considerable support for studying anisotropy of smooth spatial surfaces, and to correct for geometrical anisotropy (anisotropy which can be removed by ‘squeezing’ the plot in some direction). The function `loc` has two additional parameters `angle` and `ratio` to remove geometrical anisotropy. The functions `variogram`, `correlogram` and `covariogram` all allow multiple plots for pairs of distances in angular sectors. For example

```
plot(variogram(z ~ loc(x, y), data=topo, azimuth = c(0, 90),
   tol.azimuth = 45), aspect=0.7, layout=c(2,1))
plot(variogram(topo.res ~ loc(x, y), data=topo,
   azimuth = c(0, 90), tol.azimuth = 45),
   aspect=0.7, layout=c(2,1))
```

They show vertical and horizontal variograms (for pairs within a tolerance of $\pm 45^\circ$) of the raw `topo` data and then the residuals from the quadratic trend surface. (As these produce *and* print Trellis plots, none of the normal ways to put two plots on one page are possible and Figure 14.10 is assembled from two S-PLUS plots.)

Point process functions

Spatial point patterns are objects of class `"spp"`, with constructor function `spp`. We can convert our `pines.dat` to a `spp` object by

```
library(spatial) # our library, for next line only.
pines <- data.frame(ppinit("pines.dat")[c("x", "y")])
pines <- spp(pines, "x", "y", bbox(c(0,9.6), c(0, 10)), drop=T)
attributes(pines)
$class:
[1] "spp"        "data.frame"
$coords:
[1] "x" "y"
$boundary:
$boundary$x:
[1] 0.0 0.0 9.6 9.6
$boundary$y:
[1] 10  0  0 10
```

An object of class `"spp"` is a data frame with two attributes, `"coords"` declares which columns give the spatial coordinates, and `"boundary"` which gives the boundary of a polygon within which the pattern was observed. (This defaults to the bounding rectangle aligned with the axes, but the use of that is not advisable.)

We can reproduce Figure 14.9 quite closely by

```
par(pty = "s", mfrow=c(2,2))
plot(pines, boundary = T)
Lhat(pines,  maxdist = 5)
Lenv(pines, 25, process = "binomial", maxdist=5)
Lhat(pines,  maxdist  =1.5)
Lenv(pines, 100, process = "Strauss", maxdist = 1.5,
   cpar = 0.2, radius = 0.7)
```

As this code shows, `Lenv` can simulate from several point process models: it does so by calling the function `make.pattern` whose functionality is equivalent to that of our functions `Psim`, `SSI` and `Strauss` plus certain Poisson cluster processes.

There is no way to estimate parameters of point process models in the current release of S+SPATIALSTATS, but it does have functions `Fhat` and `Ghat` to use *nearest neighbour* methods, and function `intensity` to estimate the intensity function of a heterogeneous point process. (This is closely related to bivariate density estimation.)

References

Aitkin, M., Anderson, D., Francis, B. and Hinde, J. (1989) *Statistical Modelling in GLIM*. Oxford: Oxford University Press. [12]

Atkinson, A. C. (1985) *Plots, Transformations and Regression*. Oxford: Oxford University Press. [9]

Bartholomew, D. J. (1987) *Latent Variable Analysis and Factor Analysis*. London: Griffin. [41, 42, 43]

Basilevsky, A. (1994) *Statistical Factor Analysis and Related Methods*. New York: John Wiley and Sons. [46, 48]

Bates, D. M. and Watts, D. G. (1980) Relative curvature measures of nonlinearity (with discussion). *Journal of the Royal Statistical Society, Series B* **42**, 1–25. [18]

Bates, D. M. and Watts, D. G. (1988) *Nonlinear Regression Analysis and Its Applications*. New York: John Wiley and Sons. [18]

Beale, E. M. L. (1960) Confidence intervals in non-linear estimation (with discussion). *Journal of the Royal Statistical Society B* **22**, 41–88. [18]

Bloomfield, P. (1976) *Fourier Analysis of Time Series: An Introduction*. New York: John Wiley and Sons. [64, 65]

Bowman, A. and Azzalini, A. (1997) *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*. Oxford: Oxford University Press. [27, 52]

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*. Monterey: Wadsworth and Brooks/Cole. [35]

Brockwell, P. J. and Davis, R. A. (1991) *Time Series: Theory and Methods*. Second Edition. New York: Springer-Verlag. [62, 66]

Brownlee, K. A. (1965) *Statistical Theory and Methodology in Science and Engineering*. Second Edition. New York: John Wiley and Sons. [8]

Ciampi, A., Chang, C.-H., Hogg, S. and McKinney, S. (1987) Recursive partitioning: A versatile method for exploratory data analysis in biostatistics. In *Biostatistics*, eds I. B. McNeil and G. J. Umphrey, pp. 23–50. New York: Reidel. [30]

Collett, D. (1991) *Modelling Binary Data*. London: Chapman & Hall. [12]

Cox, D. R. and Snell, E. J. (1989) *The Analysis of Binary Data*. Second Edition. London: Chapman & Hall. [12]

Daniel, C. and Wood, F. S. (1980) *Fitting Equations to Data*. Second Edition. New York: John Wiley and Sons. [9]

Davis, R. and Anderson, J. (1989) Exponential survival trees. *Statistics in Medicine* **8**, 947–961. [30]

Emerson, J. D. and Hoaglin, D. C. (1983) Analysis of two-way tables by medians. In *Understanding Robust and Exploratory Data Analysis*, eds D. C. Hoaglin, F. Mosteller and J. W. Tukey, pp. 165–210. New York: John Wiley and Sons. [6]

Emerson, J. D. and Wong, G. Y. (1985) Resistant non-additive fits for two-way tables. In *Exploring Data Tables, Trends and Shapes*, eds D. C. Hoaglin, F. Mosteller and J. W. Tukey, pp. 67–124. New York: John Wiley and Sons. [7]

Finney, D. J. (1971) *Probit analysis*. Third Edition. Cambridge, England: CUP. [11, 12]

Friedman, J. H. (1991) Multivariate adaptive regression splines (with discussion). *Annals of Statistics* **19**, 1–141. [23]

Gower, J. C. (1966) Some distance properties of latent roots and vector methods used in multivariate analysis. *Biometrika* **53**, 325–338. [47]

Gower, J. C. and Hand, D. J. (1996) *Biplots*. London: Chapman & Hall. [38]

Gray, R. J. (1994) Hazard estimation with covariates: algorithms for direct estimation, local scoring and backfitting. Technical Report 784Z, Dana-Farber Cancer Institute, Division of Biostatistics. [Available from `ftp://farber.harvard.edu/stats/gray/784Z.ps.Z`]. [52, 54]

Gray, R. J. (1996) Hazard rate regression using ordinary nonparametric regression smoothers. *J. Comp. Graph. Statist.* **5**, 190–207. [52]

Greenacre, M. (1992) Correspondence analysis in medical research. *Statistical Methods in Medical Research* **1**, 97–117. [38]

Hastie, T. J. and Tibshirani, R. J. (1990) *Generalized Additive Models*. London: Chapman & Hall. [23]

Hjort, N. L. (1997) Dynamic likelihood hazard rate estimation. *Biometrika* **84**, xxx–xxx. [50]

Jolliffe, I. T. (1986) *Principal Component Analysis*. New York: Springer-Verlag. [48]

Kaluzny, S. and Vega, S. C. (1997) *S+SPATIALSTATS*. New York: Springer-Verlag. [67]

Kooperberg, C., Bose, S. and Stone, C. J. (1997) Polychotomous regression. *Journal of the American Statistical Association* **92**, 117–127. [24]

Kooperberg, C. and Stone, C. J. (1992) Logspline density estimation for censored data. *Journal of Computational and Graphical Statistics* **1**, 301–328. [1]

Kooperberg, C., Stone, C. J. and Truong, Y. K. (1995a) Hazard regression. *J. Amer. Statist. Assoc.* **90**, 78–94. [51, 54, 55]

Kooperberg, C., Stone, C. J. and Truong, Y. K. (1995b) Logspline estimation for a possible mixed spectral distribution. *Journal of Time Series Analysis* **16**, 359–388. [59]

Lawley, D. N. and Maxwell, A. E. (1971) *Factor Analysis as a Statistical Method*. Second Edition. London: Butterworths. [40, 41]

LeBlanc, M. and Crowley, J. (1992) Relative risk trees for censored survival data. *Biometrics* **48**, 411–425. [31]

LeBlanc, M. and Crowley, J. (1993) Survival trees by goodness of split. *Journal of the American Statistical Association* **88**, 457–467. [34, 35]

Loader, C. R. (1996) Local likelihood density estimation. *Annals of Statistics* **24**, 1602–1618. [4]

Loader, C. R. (1997) Locfit: An introduction. *Statistical Computing and Graphics Newsletter* [Available from `http://cm.bell-labs.com/stat/project/locfit`]. [3, 4, 5]

Mandel, J. (1969) A method of fitting empirical surfaces to physical or chemical data. *Technometrics* **11**, 411–429. [7]

McCullagh, P. and Nelder, J. A. (1989) *Generalized Linear Models*. Second Edition. London: Chapman & Hall. [12, 13, 14, 15]

Mosteller, F. and Tukey, J. W. (1977) *Data Analysis and Regression*. Reading, MA: Addison-Wesley. [6]

Mueller, H. G. and Wang, J. L. (1994) Hazard rates estimation under random censoring with varying kernels and bandwidths. *Biometrics* **50**, 61–76. [49]

Rao, C. R. (1955) Estimation and tests of significance in factor analysis. *Psychometrika* **20**, 93–111. [47]

Reyment, R. and Jöreskog, K. G. (1993) *Applied Factor Analysis in the Natural Sciences*. Cambridge: Cambridge University Press. [47]

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press. [27, 35]

Ruppert, D., Sheather, S. J. and Wand, M. P. (1995) An effective bandwidth selector for local least squares regression. *Journal of the American Statistical Association* **90**, 1257–1270. [23]

Seber, G. A. F. and Wild, C. J. (1989) *Nonlinear Regression*. New York: John Wiley and Sons. [18]

Segal, M. R. (1988) Regression trees for censored data. *Biometrics* **44**, 35–47. [32, 33, 34]

Simonoff, J. S. (1996) *Smoothing Methods in Statistics*. New York: Springer-Verlag. [1]

Smith, G. A. and Stanley, G. (1983) Clocking g: relating intelligence and measures of timed performance. *Intelligence* **7**, 353–368. [43]

Stone, C. J., Hansen, M., Kooperberg, C. and Truong, Y. K. (1997) Polynomial splines and their tensor products in extended linear modelling. *Annals of Statistics* **25**, 1371–1470. [1]

Tarone, R. E. and Ware, J. (1977) On distribution-free tests for the equality of survival distributions. *Biometrika* **64**, 156–160. [32]

Wahba, G., Gu, C., Wang, Y. and Chappell, R. (1995) Soft classification a.k.a. risk estimation via penalized log likelihood and smoothing spline analysis of variance. In *The Mathematics of Generalization*, ed. D. H. Wolpert, pp. 331–359. Reading, MA: Addison-Wesley. [27]

Wand, M. P. and Jones, M. C. (1995) *Kernel Smoothing*. Chapman & Hall. [23, 49]

Wood, L. A. and Martin, G. M. (1964) Compressibility of natural rubber at pressures below 500 kg/cm ^{2}. *Journal of Research National Bureau of Standards* **68A**, 259–268. [7]

Index

Entries in `this font` are names of S objects.