Statistics Complements to

Modern Applied Statistics with S

Fourth edition

by

W. N. Venables and B. D. Ripley

Springer (2002). ISBN 0-387-95457-0

22 April 2002

These complements have been produced to supplement the fourth edition of MASS. They will be updated from time to time. The definitive source is http://www.stats.ox.ac.uk/pub/MASS4/.

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 are available from

http://www.stats.ox.ac.uk/pub/MASS4/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 6.x: try the migration tools.

Contents

Chapter 5

Univariate Statistics

5.2 Generating random data

Details of the pseudo-random number generator

Some users will want to understand the internals of the pseudo-random number generator. Background knowledge on pseudo-random number generators is given by Ripley (1987, Chapter 2).

S-PLUS *pseudo-random number generator*

We describe the generator that was current at the time of writing and which had been in use for many years (with a slight change with S-PLUS 3.2 in 1994); it is of course not guaranteed to remain unchanged. This based on George Marsaglia's "Super-Duper" package from about 1973. The generator produces a 32-bit integer whose top 31 bits are divided by 2^{31} to produce a real number in $[0, 1)$. The 32-bit integer is produced by a bitwise exclusive-or of two 32-bit integers produced by separate generators. The C code for the S random number generator is (effectively)

```
unsigned int congrval, tausval; # assume 32-bit
static double xuni()
{
  unsigned int n, lambda = 69069, res;
  do {
    congrval = congrval * lambda;
    tausval ^= tausval >> 15;
    tausval ^= tausval << 17;
    n = tausval ^ congrval;
    res = (n>>1) & 017777777777;
  } while(res == 0);
  return (res / 2147483648.);
}
```

The integer `congrval` follows the congruential generator

$$X_{i+1} = 69069 X_i \bmod 2^{32}$$

as unsigned integer overflows are (silently) reduced modulo 2^{32}; that is, the overflowing bits are discarded. As this is a multiplicative generator (no additive constant), its period is 2^{30} and the bottom bit must always be odd (including for the seed).

The integer `tausval` follows a 32-bit Tausworthe generator (of period 4 292 868 097 $< 2^{32} - 1$):

$$b_i = b_{i-p} \operatorname{xor} b_{i-(p-q)}, \qquad p = 32, \quad q = 15$$

This follows from a slight modification of Algorithm 2.1 of Ripley (1987, p. 28). (In fact, the period quoted is true only for the vast majority of starting values; for the remaining 2 099 198 non-zero initial values there are shorter cycles.)

For most starting seeds the period of the generator is $2^{30} \times$ 4 292 868 097 $\approx 4.6 \times 10^{18}$, that is quite sufficient for calculations that can be done in a reasonable time in S. The current values of `congrval` and `tausval` are encoded in the vector `.Random.seed`, a vector of 12 integers in the range $0, \ldots, 63$. If x represents `.Random.seed`, we have

$$\texttt{congrval} = \sum_{i=1}^{6} x_i \, 2^{6(i-1)} \quad \text{and} \quad \texttt{tausval} = \sum_{i=1}^{6} x_{i+6} \, 2^{6(i-1)}$$

R *pseudo-random number generators*

R offers a choice of pseudo-random number generators, and also of ways to generate from the normal distribution. Furthermore, user-written code can be substituted for both the uniform and normal generators: `?RNG` and `?Random.user` give details.

The current default pseudo-random number generator is the multicarry generator proposed by Marsaglia (1997). This has two 32-bit integers as the "seed". In C code a single update is $U[0, 1)$ is generated by

```
int I1, I2; # assume 32-bit
static double xuni()
{
    /* 0177777(octal) is the bottom 16 bits*/
    I1 = 36969 * (I1 & 0177777) + (I1 >> 16);
    I2 = 18000 * (I2 & 0177777) + (I2 >> 16);
    return ((I1 << 16)^(I2 & 0177777)) *
           2.328306437080797e-10;
}
```

Marsaglia's original posting said

> I have often been asked to suggest random number generators for use in C. Many good ones are available through the net, but they often require complicated code and various ways to use or initialize through calls to subroutines. The following suggestion has these properties:
>
> - Seems to pass all tests of randomness put to it.

- Has much much longer period, $> 2^{60}$, than most system generators, $< 2^{32}$, and versions can be combined to get periods $> 2^{90}, 2^{120}$, etc.
- Exploits the feature of C that provides segments of in-line code through `#define` statements. Thus random integers or reals can be put directly into expressions, avoiding the cost of calls to subroutines.
- Uses what I view as the most promising new method for random number generation: multiply-with-carry.

Marsaglia & Zaman (1994) is a useful background reference.

Supplied alternatives are

1. The sum modulo 1 of three short-period conguential generators of Wichmann & Hill (1982). This was the original default in R, but is now deprecated.
2. A version of Marsaglia's "Super-Duper" generator, which unlike the S-PLUS version does not skip zeroes.
3. The 'Mersenne-Twister' of Matsumoto & Nishimura (1998), a GFSR with period $2^{19937} - 1$.
4. A GFSR from Knuth (1997) using lagged Fibonacci sequences with subtraction. That is, the recurrence used is

$$X_j = (X_{j-100} - X_{j-37}) \bmod 2^{30}$$

Chapter 6

Linear Statistical Models

6.5 Robust and resistant regression

Median polish

Consider a two-way layout. The additive model is

$$\hat{y}_{ij} = \mu + \alpha_i + \beta_j, \qquad \alpha_\cdot = \beta_\cdot = 0$$

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

In S-PLUS the default `trim = 0.5` option of `twoway` performs median polish, and in R there is function `medpolish` in package `eda`. We have, after multiplying by 10^4, R

	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

$$y_{ij} = m + a_i + b_j + e_{ij} + k a_i b_j \tag{6.8}$$

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 (6.8).

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.

```
> #R: data(stackloss); stack.loss <- stackloss[, 4];
     stack.x <- stackloss[, -4]
> 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

> #R: library(lqs)
> 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.
 -7.8929 -0.2500  0.1071  1.0765  1.3929  9.4643

> 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.
 -8.3611 -0.3611  0.3056  0.9762  1.3056  9.3056
```

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 6.9.

```
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.)

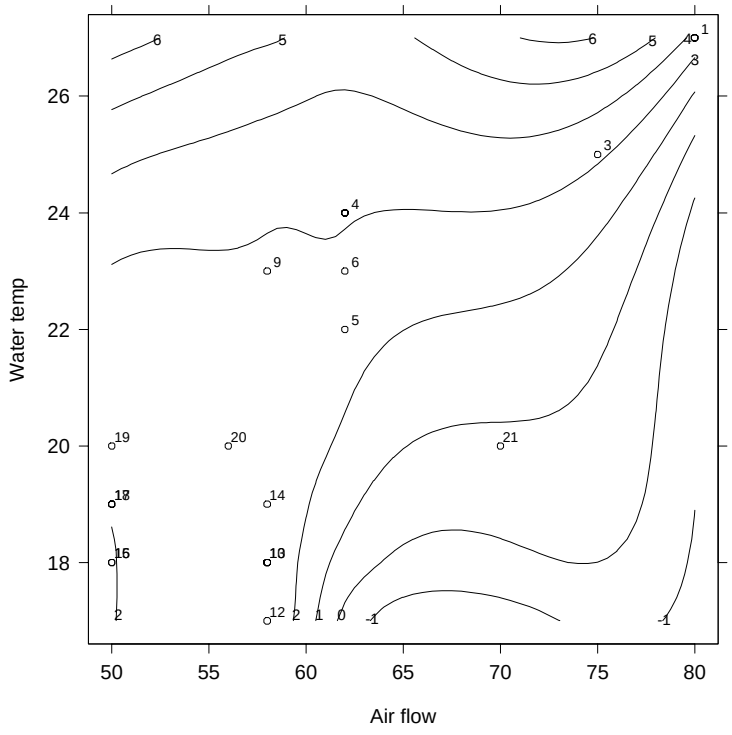

Figure 6.9: Fitted surface for Brownlee's stack loss data on log scale using `loess`.

Chapter 7

Generalized Linear Models

7.6 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 183) 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

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

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) on page 186 and the MLE are based on the residual deviance

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

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

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

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

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

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 186).

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 = "response")/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

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

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

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 220.[1]

In `MASS`[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 222. 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.

```
> 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
```

[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.

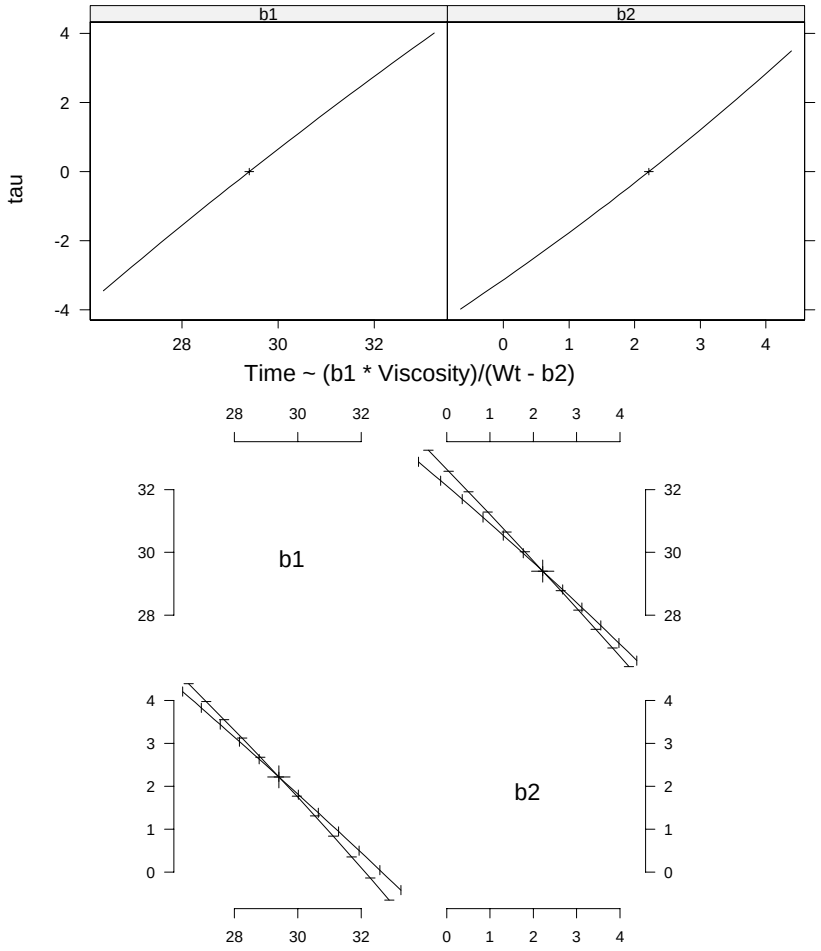

Figure 8.13: Profile and pairs-profile plots for the Stormer data example fitted as a non-linear regression model.

```
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 in S-PLUS[3] are shown in Figure 8.13. 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 224. The profile traces are the lines that would join up the points where the contours have horizontal and vertical

[3] R displays `nls` profiles as the unsigned trace.

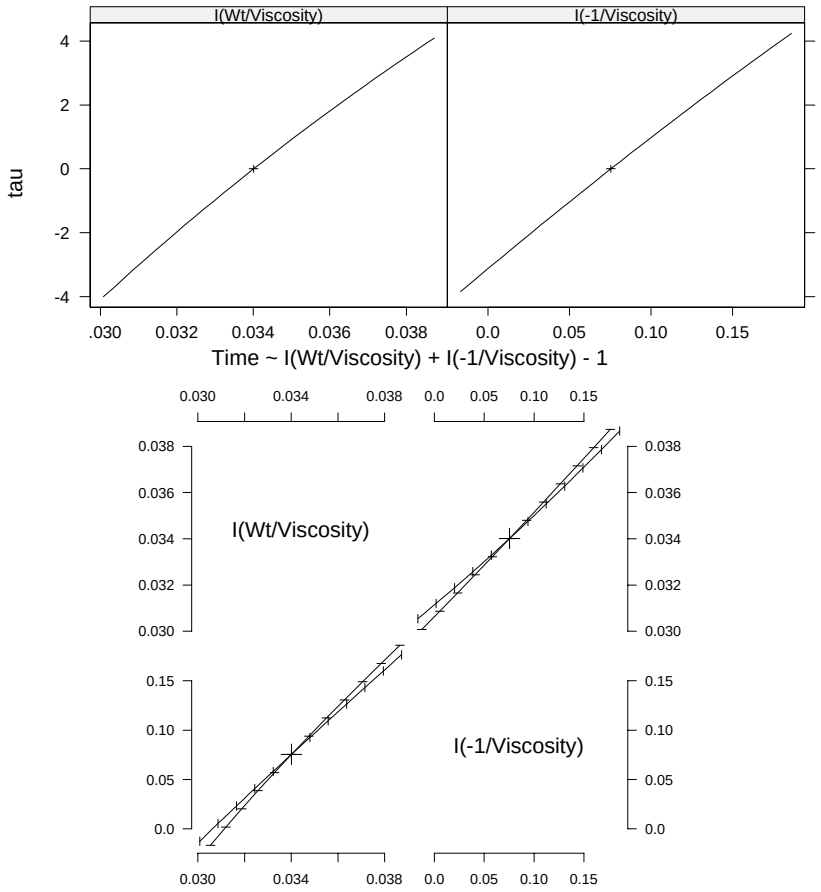

Figure 8.14: Profile and pairs-profile plots for the Stormer data example with the model fitted as a GLM.

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.glm.pro <- profile(storm.glm, trace = T)
    ....
> plot(storm.glm.pro)
> pairs(storm.glm.pro)
```

The results are shown in Figure 8.14. 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 198, so the `glm` method of `profile` can be a useful exploratory tool.

Chapter 13

Survival Analysis

13.1 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 and we had to refine the output grid to produce a fairly smooth result. The result shown in Figure 13.13 is unconvincing.

Likelihood-based approaches

Censoring is easy to incorporate in maximum-likelihood estimation; the likelihood is given by (13.1) on page 352. 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`. Consider the `gehan` dataset.

```
library(logspline)
g1 <- gehan[gehan$treat == "control",]
g2 <- gehan[gehan$treat == "6-MP",]
logspline.plot(
     logspline.fit(uncensored = g1[g1$cens == 1, "time"],
```

[1] available for UNIX from http://odin.mdacc.tmc.edu/anonftp: also an R package on CRAn.

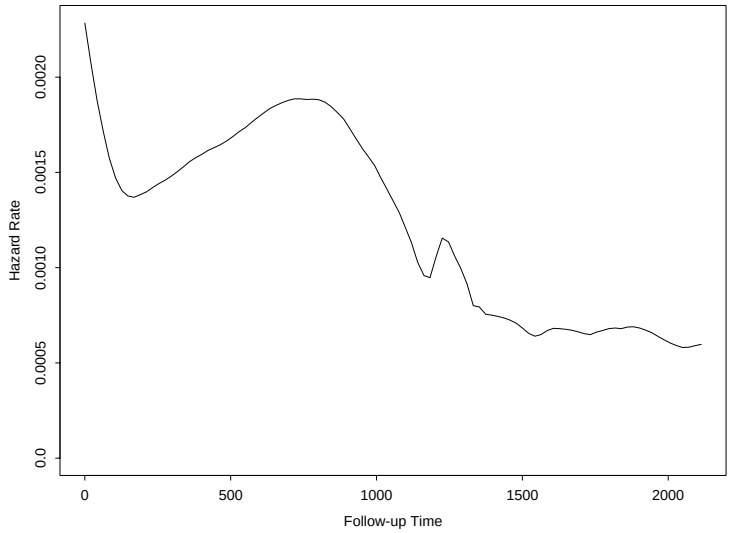

Figure 13.13: Hazard function fitted to the `Aids` dataset by `muhaz`.

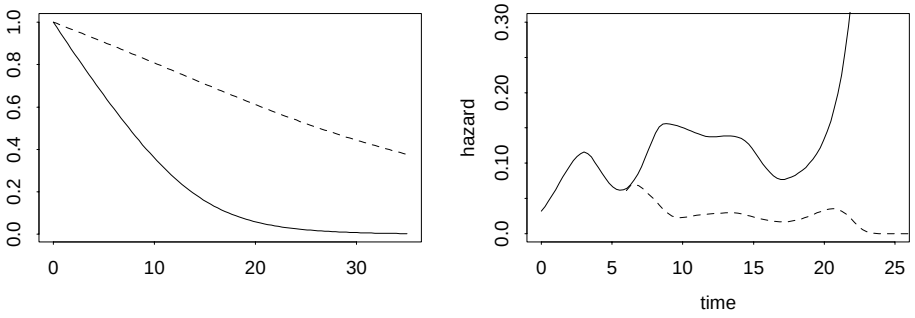

Figure 13.14: Smooth survival (left, by `logspline.fit`) and hazard (right, by `locfit`) fits to the `gehan` dataset. The solid line indicates the control group, the dashed line that receiving 6-MP.

```
                    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.*, 1995). 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. 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 R nor to S-PLUS 6.0 on Windows.

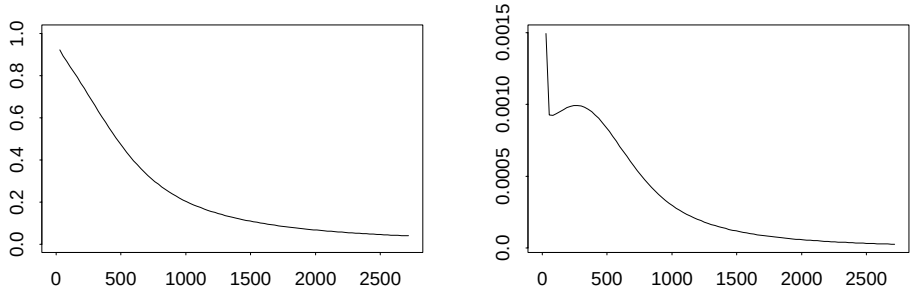

Figure 13.15: Survivor curve and hazard fitted to `Aids` by `heft.fit`.

13.5 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(0,3 65))),
     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,

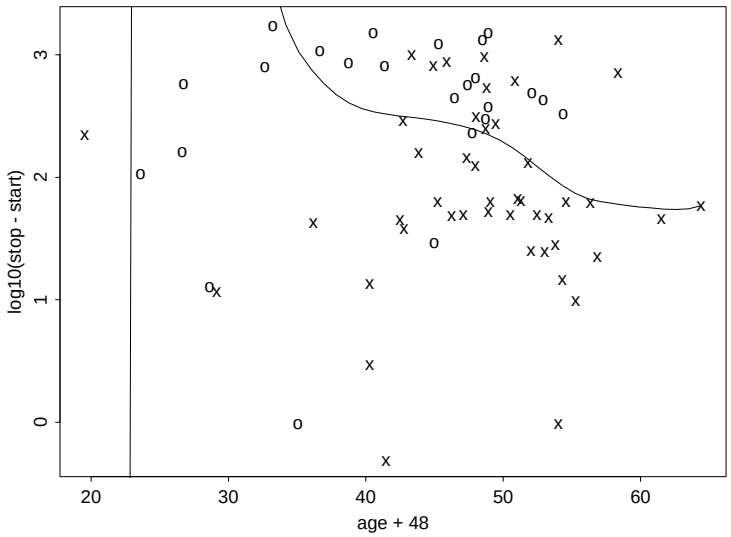

Figure 13.16: Median survival time for the Stanford heart transplant data by `sm.survival`.

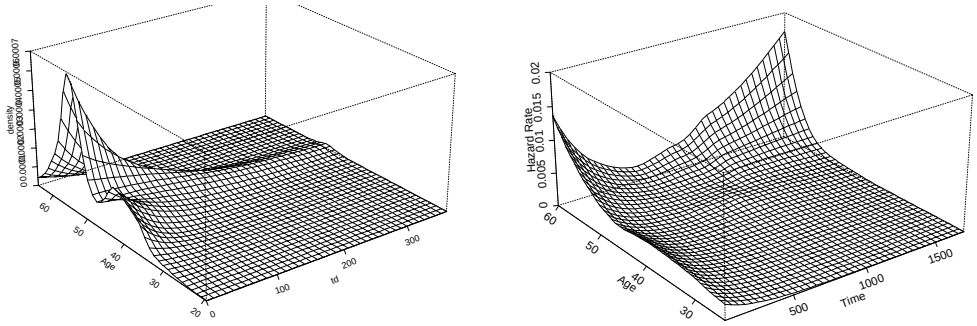

Figure 13.17: Smooth hazard functions (in days) as a function of age post-transplantation in the Stanford heart-transplant study. **Left:** by `locfit` and **right:** by `hazcov` using local scoring.

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.*, 1995) 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

```
#  VA is constructed on page 374
> attach(VA)
```

[3] Not ported to R nor to S-PLUS 6.0 on Windows.

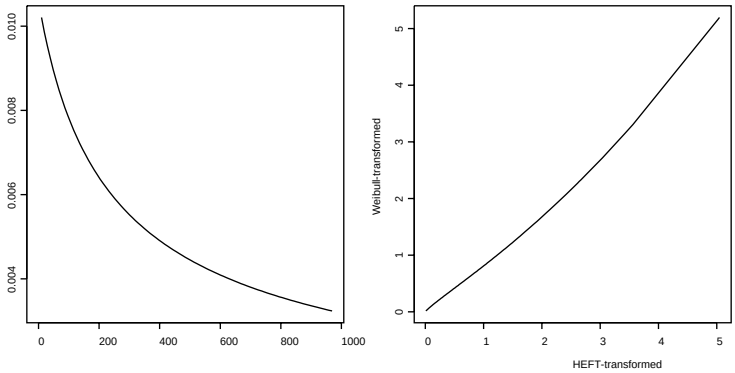

Figure 13.18: The marginal distribution of lifetime in the `cancer.vet` dataset. **Left:** Hazard as fitted by `heft.fit`. **Right:** Time as transformed by the distribution fitted by `heft.fit` and by a fitted Weibull distribution.

```
> 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
```

We found that an exponential model for the residual hazard was adequate, but Kooperberg *et al.* (1995) 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).

13.6 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 13.9 by

```
# S: year <- dates("7/1/91")
# R: data(ratetables); year <- as.date("7/1/91")
     survexp.uswhite <- survexp.usr[,,"white",]
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 $S_i(t)I(\text{potentially in study at } t)$. 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 13.9 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.

13.7 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).

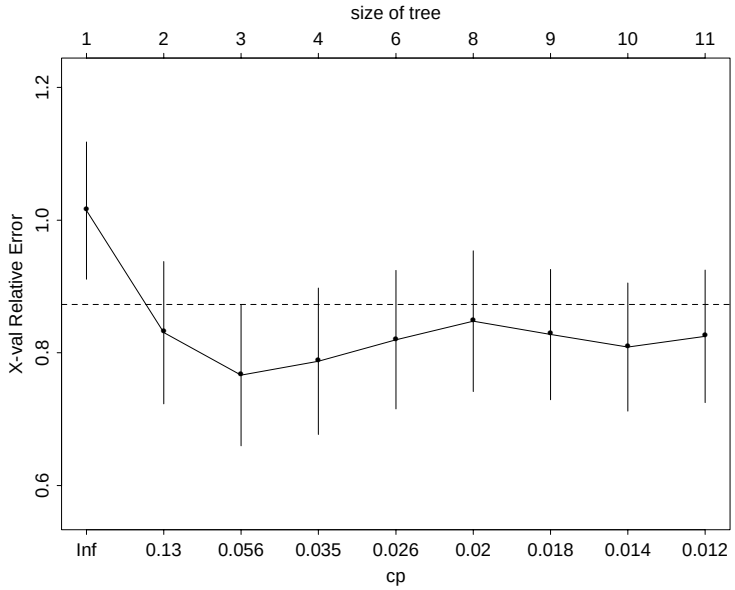

Figure 13.19: Plot by `plotcp` of the `rpart` object `VA.rp`.

`"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[4]. This gives a proportional hazards model with the baseline hazard fixed as the estimated marginal hazard.

We use the VA cancer dataset `cancer.vet` 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
```

[4] 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 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. 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 shows that there is very little difference in survival between nodes at (Figure 13.20) 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

```
tree.screens()
plot(VA.tssa)
```

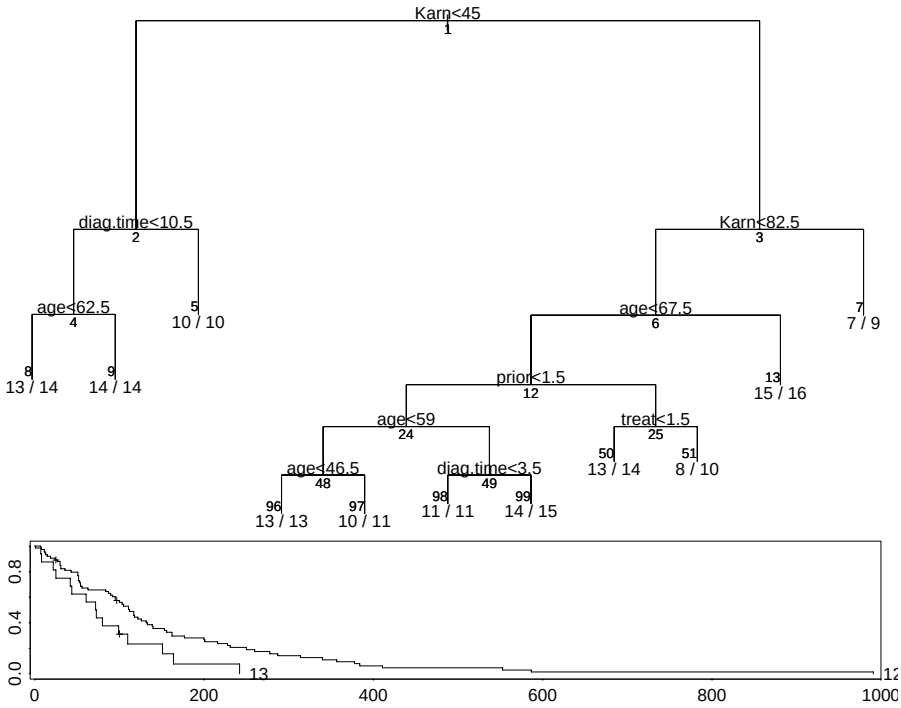

Figure 13.20: Tree fitted by `tssa` to the `cancer.vet` dataset. The bottom screen shows the output from `km.tssa` when node 6 was selected.

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

The only clear-cut pruning point (Figure 13.21) is at a single split. There is a function `post.tssa` the equivalent of (and modified from) `post.tree` for `tssa` trees.

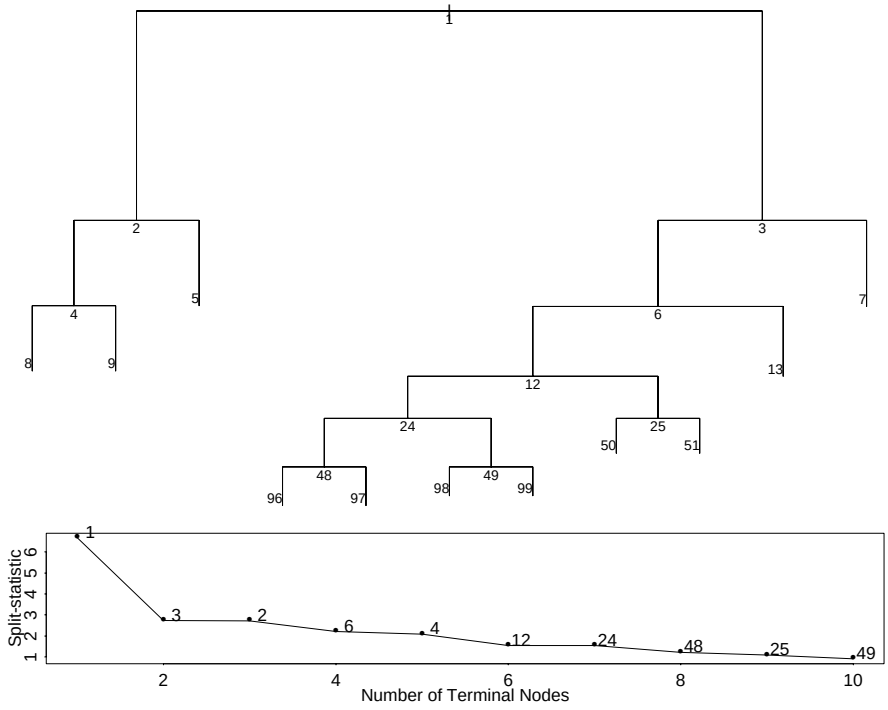

Figure 13.21: Tree fitted by `tssa` to the `cancer.vet` dataset. The bottom screen shows the prune sequence from `prune.tssa`.

Chapter 14

Time Series

14.8 Multiple time series

The second-order time-domain properties of multiple time series were covered in Section 14.1. 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 series, and $\boldsymbol{\epsilon}_t$ 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

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

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

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

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 14.25.

```
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)
```

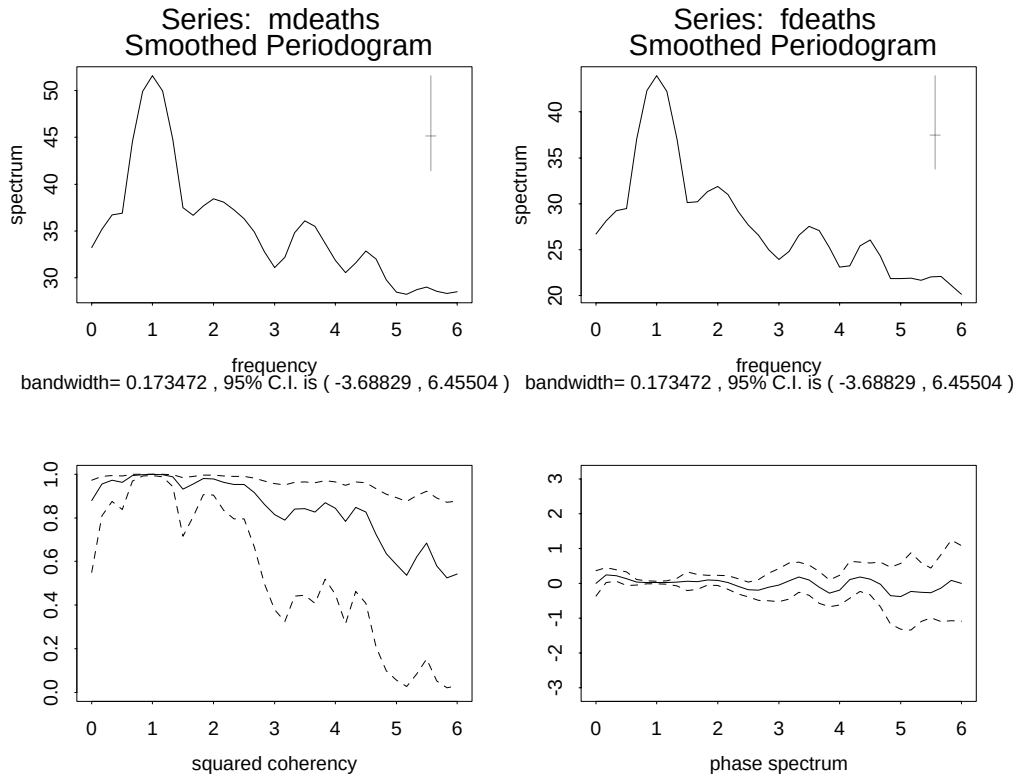

Figure 14.25: Coherence and phase spectra for the two deaths series, with 95% pointwise confidence intervals.

These confidence intervals follow Bloomfield (2000, §9.5). 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)
```

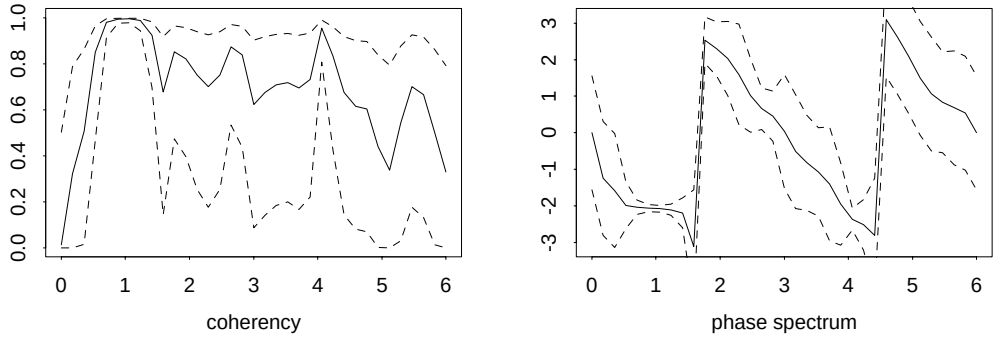

Figure 14.26: Coherence and phase spectra for the re-aligned deaths series, with 95% pointwise confidence intervals.

The results are shown in Figure 14.26. 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`.

14.9 Other time-series functions

S-PLUS and R have a number of time-series functions which are used less frequently and we have not 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 S-PLUS function `arima.filt`.

A large number of smoothing operations such as `lowess` can be regarded as filters, but they are non-linear. The S-PLUS 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 (2000, Chapter 7) and implemented by the S-PLUS 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. Essentially the same functions are `fracdiff` and `fracdiff.sim` in the R package `fracdiff` on CRAN.

Chapter 15

Spatial Statistics

15.5 Module S+SPATIALSTATS

The the S-PLUS module S+SPATIALSTATS 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 15 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 15.5 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 15.7 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

```
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)
```

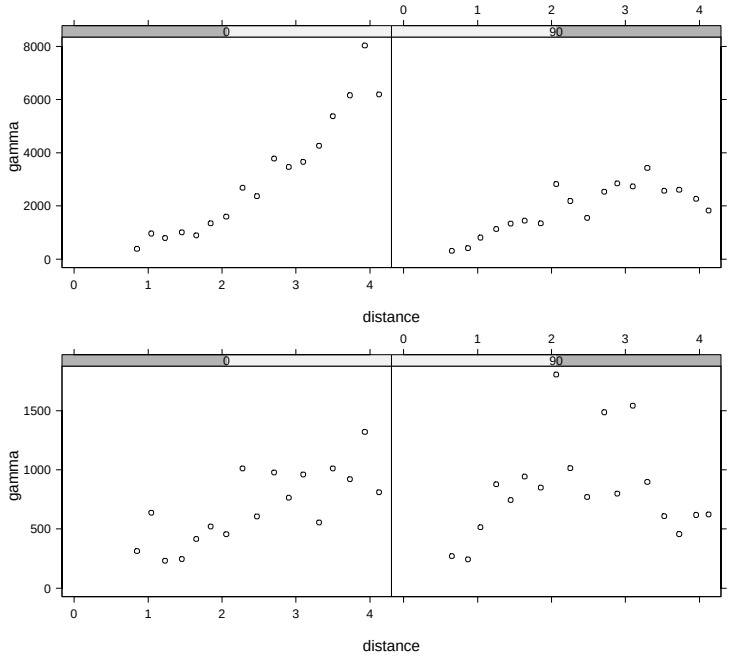

Figure 15.10: Directional variograms for the `topo` dataset. The top pair is for the raw data, the bottom pair of residuals from a quadratic trend surface. The left plots are vertical variograms, the right plots are horizontal ones. (The strip coverage is misleading, only showing the positive part of the angular tolerance.)

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 15.10 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(unclass(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 15.9 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

Atkinson, A. C. (1985) *Plots, Transformations and Regression*. Oxford: Oxford University Press. [7]

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. [13]

Bates, D. M. and Watts, D. G. (1988) *Nonlinear Regression Analysis and Its Applications*. New York: John Wiley and Sons. [13]

Beale, E. M. L. (1960) Confidence intervals in non-linear estimation (with discussion). *Journal of the Royal Statistical Society B* **22**, 41–88. [13]

Bloomfield, P. (2000) *Fourier Analysis of Time Series: An Introduction*. Second Edition. New York: John Wiley and Sons. [35, 37]

Bowman, A. and Azzalini, A. (1997) *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*. Oxford: Oxford University Press. [21]

Brockwell, P. J. and Davis, R. A. (1991) *Time Series: Theory and Methods*. Second Edition. New York: Springer-Verlag. [33, 37]

Brownlee, K. A. (1965) *Statistical Theory and Methodology in Science and Engineering*. Second Edition. New York: John Wiley and Sons. [6]

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. [27]

Daniel, C. and Wood, F. S. (1980) *Fitting Equations to Data*. Second Edition. New York: John Wiley and Sons. [7]

Davis, R. and Anderson, J. (1989) Exponential survival trees. *Statistics in Medicine* **8**, 947–961. [27]

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. [4]

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. [5]

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`]. [21, 23]

Gray, R. J. (1996) Hazard rate regression using ordinary nonparametric regression smoothers. *J. Comp. Graph. Statist.* **5**, 190–207. [21]

Hjort, N. L. (1997) Dynamic likelihood hazard rate estimation. *Biometrika* **84**, xxx–xxx. [19]

Kaluzny, S. and Vega, S. C. (1997) *S*+SPATIALSTATS. New York: Springer-Verlag. [38]

Knuth, D. E. (1997) *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Third Edition. Reading, MA: Addison-Wesley. [3]

Kooperberg, C., Stone, C. J. and Truong, Y. K. (1995) Hazard regression. *J. Amer. Statist. Assoc.* **90**, 78–94. [20, 23, 24]

LeBlanc, M. and Crowley, J. (1992) Relative risk trees for censored survival data. *Biometrics* **48**, 411–425. [28]

Mandel, J. (1969) A method of fitting empirical surfaces to physical or chemical data. *Technometrics* **11**, 411–429. [5]

Marsaglia, G. (1997) *A random number generator for C*. Posting on Usenet newsgroup `sci.stat.math` on September 29, 1997. [2]

Marsaglia, G. and Zaman, A. (1994) Some portable very-long-period random number generators. *omputers in Physics* **8**, 117–121. [3]

Matsumoto, M. and Nishimura, T. (1998) Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* **8**, 3–30. [3]

McCullagh, P. and Nelder, J. A. (1989) *Generalized Linear Models*. Second Edition. London: Chapman & Hall. [9, 10]

Mosteller, F. and Tukey, J. W. (1977) *Data Analysis and Regression*. Reading, MA: Addison-Wesley. [4]

Mueller, H. G. and Wang, J. L. (1994) Hazard rates estimation under random censoring with varying kernels and bandwidths. *Biometrics* **50**, 61–76. [18]

Ripley, B. D. (1987) *Stochastic Simulation*. New York: John Wiley and Sons. [1, 2]

Seber, G. A. F. and Wild, C. J. (1989) *Nonlinear Regression*. New York: John Wiley and Sons. [13]

Segal, M. R. (1988) Regression trees for censored data. *Biometrics* **44**, 35–47. [29, 30]

Tarone, R. E. and Ware, J. (1977) On distribution-free tests for the equality of survival distributions. *Biometrika* **64**, 156–160. [29]

Wand, M. P. and Jones, M. C. (1995) *Kernel Smoothing*. London: Chapman & Hall. [18]

Wichmann, B. A. and Hill, I. D. (1982) Algorithm AS183. an efficient and portable pseudo-random number generator. *Applied Statistics* **31**, 188–190. (Correction **33**, 123.). [3]

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. [5]

Index

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