MathSoft

S-PLUS 2000 Guide to Statistics, Volume I

May 1999

Data Analysis Products Division
MathSoft, Inc.
Seattle, Washington

The correct bibliographical reference for this document is as follows:

S-PLUS 2000 Guide to Statistics, Volume 1, Data Analysis Products Division, MathSoft, Seattle, WA.

Printed in the United States.

Acknowledgments S-PLUS would not exist without the pioneering research of the Bell Labs S team at AT&T (now Lucent Technologies): Richard A. Becker (now at AT&T Laboratories), John M. Chambers, Allan R. Wilks (now at AT&T Laboratories), William S. Cleveland, Trevor Hastie (now at Stanford University), and colleagues.

This release of S-PLUS includes specific work from a number of scientists:

The survival functions were written by Terry Therneau (Mayo Clinic, Rochester, Minnesota).

The mixed-effects modeling functions were written by Doug Bates (University of Wisconsin–Madison) and José Pinheiro (Lucent Technologies).

The discriminant analysis function `discrim` contains code contributed by Brian Ripley (University of Oxford) and William Venables (CSIRO).

The `digamma` function was written by William Venables (CSIRO).

Updates to functions provided to this and earlier releases of S-PLUS were provided by Brian Ripley (University of Oxford), William Venables (CSIRO), and Terry Therneau (Mayo Clinic, Rochester).

CONTENTS

PREFACE

Introduction

Welcome to the *S-PLUS 2000 Guide to Statistics, Volume 1*.

This book is designed as a reference tool for S-PLUS users wanting to use the powerful statistical techniques in S-PLUS. The *Guide to Statistics, Volume 1* covers a wide range of statistical and mathematical modeling; no one user is likely to tap all of these resources since advanced topics such as survival analysis and time series are complete fields of study in themselves.

All examples in this guide are run using input through the **Commands** window–the traditional method of accessing the power of S-PLUS. Many of the functions can also be run through the Statistics menu and dialogs available in the graphical user interface. We hope you will find this book a valuable aid for exploring both the theory and practice of statistical modeling.

Online Version

The *Guide to Statistics, Volume 1* is also available online, through the Online Manuals entry of the main Help menu. It can be viewed using Adobe Acrobat Reader, which is included with S-PLUS.

The online version is identical in content to the printed one but with some particular advantages. First, you can cut-and-paste example S-PLUS code directly into the **Commands** window and run these examples without having to type them. Be careful not to cut-and-paste the ">" prompt character and notice that distinct colors differentiate between command language input and output.

Second, the online text can be searched for any character string. If you wish information on a certain function, for example, you can easily browse through all occurrences of it in the guide.

Also, contents and index entries in the online version are hot-links; click on them to go to the appropriate page.

Evolution of S-PLUS

S-PLUS has evolved considerably from its beginnings as a research tool, and the contents of this guide have grown steadily, and will continue to grow, as the language is improved and expanded. This may mean that some examples in the text do not match your output from S-PLUS in every formatting detail. However, the underlying theory and computations are as described here.

In addition to the huge range of functionality covered in this guide, there are additional modules, libraries, and user-written functions available from a number of sources. Refer to the *User's Guide* for more details.

Companion Guides

The *Guide to Statistics, Volume 1*, together with *Guide to Statistics, Volume 2*, is a companion volume to the *User's Guide* and the *Programmer's Guide*. All four are available both in printed form and online through the help system.

This volume covers the following topics:

- An overview of statistical modeling in S-PLUS
- The S-PLUS statistical modeling framework
- Statistical inference for one, two, and many sample problems, both continuous and discrete
- Cross-classified data and contingency tables
- Regression models
- Robust regression models
- Generalized linear models
- Local regression models
- Classification and regression trees
- Mixed-effects models
- Analysis of variance

The *Guide to Statistics, Volume 2* covers multivariate analysis techniques, cluster analysis, survival analysis, resampling techniques, and mathematical computing.

1 INTRODUCTION TO STATISTICAL ANALYSIS IN S-PLUS

INTRODUCTION

All statistical analysis has, at its heart, a *model* which attempts to describe the structure or relationships in some objects or phenomena on which measurements (the data) are taken. Estimation, hypothesis testing, and inference, in general, are based on the data at hand and a conjectured model which you may define implicitly or explicitly. You specify many types of models in S-PLUS using *formulas*, which express the conjectured relationships between observed variables in a natural way. The power of S-PLUS as a statistical modeling language lies in its convenient and useful way of organizing data, its wide variety of classical and modern modeling techniques, and its way of specifying models.

The goal of this chapter is to give you a feel for data analysis in S-PLUS: examining the data, selecting a model, and displaying and summarizing the fitted model.

DEVELOPING STATISTICAL MODELS

The process of developing a statistical model varies depending on whether you follow a classical, hypothesis-driven approach (confirmatory data analysis) or a more modern, data-driven approach (exploratory data analysis). In many data analysis projects, both approaches are frequently used. For example, in classical regression analysis, you usually examine residuals using exploratory data analytic methods for verifying whether underlying assumptions of the model hold. The goal of either approach is a model which imitates, as closely as possible, in as simple a way as possible, the properties of the objects or phenomena being modeled. Creating a model usually involves the following steps:

1. Determine the variables to observe. In a study involving a classical modeling approach, these variables correspond to the hypothesis being tested. For data-driven modeling, these variables are the link to the phenomena being modeled.
2. Collect and record the data observations.
3. Study graphics and summaries of the collected data to discover and remove mistakes and to reveal low-dimensional relationships between variables.
4. Choose a model describing the important relationships seen or hypothesized in the data.
5. Fit the model using the appropriate modeling technique.
6. Examine the fit using model summaries and diagnostic plots.
7. Repeat steps 4–6 until you are satisfied with the model.

There are a wide range of possible modeling techniques to choose from when developing statistical models in S-PLUS. Among these are linear models (`lm`), analysis of variance models (`aov`), generalized linear models (`glm`), generalized additive models (`gam`), local regression models (`loess`), and tree-based models (`tree`).

DATA USED FOR MODELS

This section provides descriptions of the most common types of data objects used when developing models in S-PLUS. There are also brief descriptions and examples of common S-PLUS functions used for developing and displaying models.

Data Frame Objects

Statistical models allow inferences to be made about objects by modeling associated observational or experimental data, organized by *variables*. A *data frame* is an object that represents a sequence of observations on some chosen set of variables. Data frames are like matrices, with variables as columns and observations as rows. They allow computations where variables can act as *separate objects* and can be referenced simply by naming them. This makes data frames very useful in modeling.

Variables in data frames are generally of three forms:

- Numeric vectors
- Factors and ordered factors
- Numeric matrices

Continuous and Discrete Data

The type of data you have when developing a model is important for deciding which modeling technique best suits your data. *Continuous* data represent quantitative data having a continuous range of values. *Categorical* data, by contrast, represent *qualitative* data and are discrete, meaning they can assume only certain fixed numeric or nonnumeric values.

In S-PLUS, you represent categorical data with *factors*, which keep track of the *levels* or different values contained in the data and the level each data point corresponds to. For example, you might have a factor `gender` in which every element assumed one of the two values `"male"` and `"female"`. You represent continuous data with numeric objects. Numeric objects are vectors, matrices, or arrays of numbers. Numbers can take the form of decimal numbers (such as `11`, `-2.32`, or `14.955`) and exponential numbers expressed in scientific notation (such as `.002` expressed as `2e-3`).

A statistical model expresses a *response* variable as some function of a set of one or more *predictor* variables. The type of model you select depends on whether the response and predictor variables are continuous (numeric) or categorical (factor). For example, the classical regression problem has a continuous response and continuous predictors, but the classical ANOVA problem has a continuous response and categorical predictors.

Summaries and Plots for Examining Data

Before you fit a model, you should examine the data. Plots provide important information on mistakes, outliers, distributions, and relationships between variables. Numerical summaries provide a statistical synopsis of the data in a tabular format.

Among the most common functions to use for generating plots and summaries are the following:

- `summary`: provides a synopsis of an object. The following example displays a summary of the `kyphosis` data frame:

```
> summary(kyphosis)

   Kyphosis         Age              Number
 absent  :64  Min.   :  1.00  Min.   : 2.000
 present :17  1st Qu.: 26.00  1st Qu.: 3.000
              Median : 87.00  Median : 4.000
              Mean   : 83.65  Mean   : 4.049
              3rd Qu.:130.00  3rd Qu.: 5.000
              Max.   :206.00  Max.   :10.000

      Start
 Min.    : 1.00
 1st Qu. : 9.00
 Median  :13.00
 Mean    :11.49
 3rd Qu. :16.00
 Max.    :18.00
```

- `plot`: a generic plotting function, `plot` produces different kinds of plots depending on the data passed to it. In its most common use, it produces a scatter plot of two numeric objects.
- `hist`: creates histograms.

- `qqnorm`: creates quantile-quantile plots.
- `pairs`: creates, for multivariate data, a matrix of scatter plots showing each variable plotted against each of the other variables. To create the pairwise scatter plots for the data in the matrix `longley.x`, use `pairs` as follows:

```
> pairs(longley.x)
```

The resulting plot appears as in Figure 1.1.

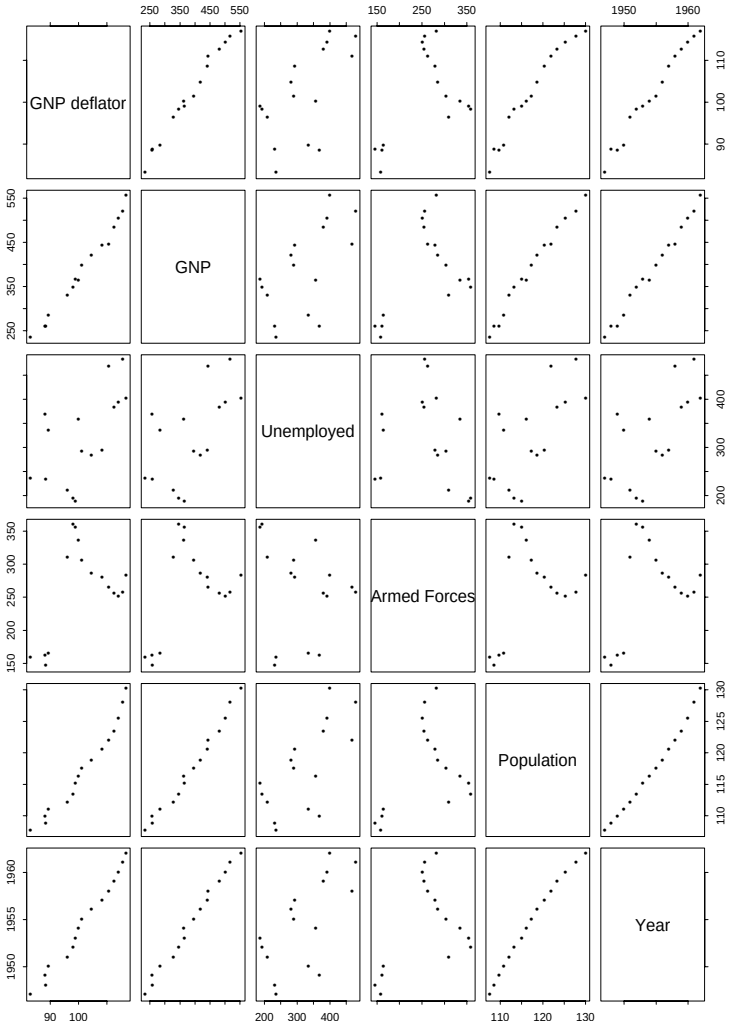

Figure 1.1: *Pairwise scatter plots for* `longley.x`.

- `coplot`: provides a graphical look at cross-sectional relationships, which enable you to assess potential interaction effects. The following example shows the effect of the interaction between `C` and `E` on values of `NOx`. The resulting plots appear as in Figure 1.2.

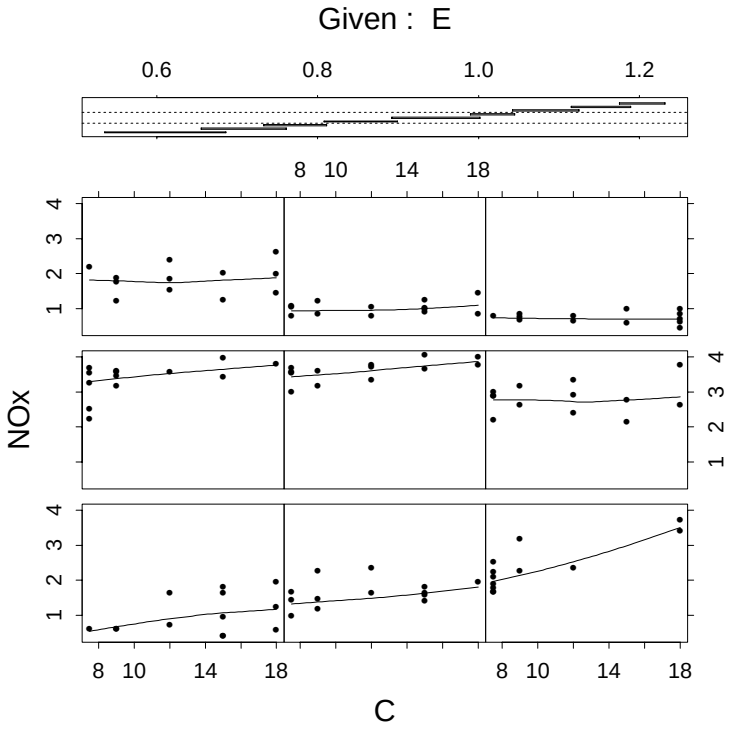

Figure 1.2: *Coplot of response and predictors.*

```
> attach(ethanol)
> E.intervals <- co.intervals(E, 9, 0.25)
> coplot(NOx ~ C | E, given.values = E.intervals,
+ data = ethanol, panel = function(x, y) panel.smooth(x,
+ y, span = 1, degree = 1))
```

STATISTICAL MODELS IN S-PLUS

The development of statistical models is, in many ways, *data dependent*. The choice of the modeling technique you use depends upon the type and structure of your data and what you want the model to test or explain. A model may predict new responses, show general trends, or uncover underlying phenomena. This section gives general selection criteria to help you develop a statistical model.

The fitting procedure for each model is based on a unified modeling paradigm in which:

- A data frame contains the data for the model.
- A *formula* object specifies the relationship between the response and predictor variables.
- The formula and data frame are passed to the fitting function.
- The fitting function returns a fit object.

There is a relatively small number of functions to help you fit and analyze statistical models in S-PLUS.

- Fitting models:
  - `lm`: linear (regression) models.
  - `aov` and `varcomp`: analysis of variance models.
  - `glm`: generalized linear models.
  - `gam`: generalized additive models.
  - `loess`: local regression models.
  - `tree`: tree models.
- Extracting information from a fitted object:
  - `fitted`: returns fitted values.
  - `coefficients` or `coef`: returns the coefficients (if present).
  - `residuals` or `resid`: returns the residuals.
  - `summary`: provides a synopsis of the fit.

    - `anova`: for a single fit object, produces a table with rows corresponding to each of the terms in the object, plus a row for residuals. If two or more fit objects are used as arguments, `anova` returns a table showing the tests for differences between the models, sequentially, from first to last.
- Plotting the fitted object:
    - `plot`: plot a fitted object.
    - `qqnorm`: produces a normal probability plot, frequently used in analysis of residuals.
    - `coplot`: provides a graphical look at cross-sectional relationships for examining interaction effects.
- For minor modifications in a model, use the `update` function (adding and deleting variables, transforming the response, etc.).
- To compute the predicted response from the model, use the `predict` function.

The Unity of Models in Data Analysis

Because there is usually more than one way to model your data, you should learn which type(s) of model are best suited to various types of response and predictor data. When deciding on a modeling technique, it helps to ask: "What do I want the *data* to explain? What hypothesis do I want to test? What am I trying to show?"

Some methods should or should not be used depending on whether the response and predictors are continuous, factors, or a combination of both. Table 1.1 organizes the methods by the type of data they can handle.

Table 1.1: *Criteria for developing models.*

Model	Response	Predictors
`lm`	Continuous	Both
`aov`	Continuous	Factors
`glm`	Both	Both
`gam`	Both	Both
`loess`	Continuous	Both
`tree`	Both	Both

Linear regression models a continuous response variable, y, as a linear combination of predictor variables x_j, for $j = 1, ..., p$. For a single predictor, the data fit by a linear model scatter about a straight line or curve. A linear regression model has the mathematical form

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \varepsilon_i$$

where ε_i, referred to, generally, as the error, is the difference between the ith observation and the model. On average, for given values of the predictors, you predict the response best with the equation

$$y = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

Analysis of variance models are also linear models, but all predictors are categorical, which contrasts with the typically continuous predictors of regression. For designed experiments, use analysis of variance to estimate and test for effects due to the factor predictors. For example, consider the `catalyst` data frame, which contains the data below:

```
  Temp Conc Cat Yield
1  160   20   A    60
2  180   20   A    72
3  160   40   A    54
4  180   40   A    68
5  160   20   B    52
6  180   20   B    83
7  160   40   B    45
8  180   40   B    80
```

Each of the predictor terms, `Temp`, `Conc`, and `Cat`, is a factor with two possible levels, and the response term, `Yield`, contains numeric data. Use analysis of variance to estimate and test for the effect of the predictors on the response.

Linear models produce estimates with good statistical properties when the relationships are, in fact, linear, and the errors are normally distributed. In some cases, when the distribution of the response is skewed, you can transform the response, using, for example, square root, logarithm, or reciprocal transformations, and produce a better fit. In other cases, you may need to include polynomial terms of the predictors in the model. However, if linearity or normality does not hold, or if the variance of the observations is not constant, and transformations of the response and predictors do not help, you should explore other techniques such as generalized linear models, generalized additive models, or classification and regression trees.

Generalized linear models generalize linear models by assuming a *transformation* of the expected (or average) response is a linear function of the predictors, and the variance of the response is a function of the mean response:

$$\eta(E(y)) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

$$\mathrm{VAR}(y) = \phi V(\mu)$$

Generalized linear models, fitted using the `glm` function, allow you to model data with distributions including normal, binomial, Poisson, gamma, and inverse normal, but still require a linear relationship in the parameters.

When the linear fit provided by `glm` does not produce a good fit, an alternative is the generalized additive model, fit with the `gam` function. In contrast to `glm`, `gam` allows you to fit nonlinear data-dependent functions of the predictors. The mathematical form of a generalized additive model is:

$$\eta(E(y)) = \sum_{j=1}^{p} f_j(x_j)$$

where the f_j term represent functions to be estimated from the data. The form of the model assumes a low-dimensional additive structure. That is, the pieces represented by functions, f_i, of each predictor added together predict the response without interaction.

In the presence of interactions, if the response is continuous and the errors about the fit are normally distributed, local regression (or *loess*) models, allow you to fit a multivariate function which include interaction relationships. The form of the model is:

$$\eta_i = g(x_{i1}, x_{i2}, \ldots, x_{ip}) + \varepsilon$$

where g represents the regression surface.

Tree-based models have gained in popularity because of their flexibility in fitting all types of data. Tree models are generally used for exploratory analysis. They allow you to study the structure of data, creating nodes or clusters of data with similar characteristics. The variance of the data within each node is relatively small, since the characteristics of the contained data is similar. The following example displays a tree-based model using the data frame `car.test.frame`:

```
> car.tree <- tree(Mileage ~ Weight, car.test.frame)
> plot(car.tree, type = "u")
> text(car.tree)
> title("Tree-based Model")
```

The resulting plot appears as in Figure 1.3.

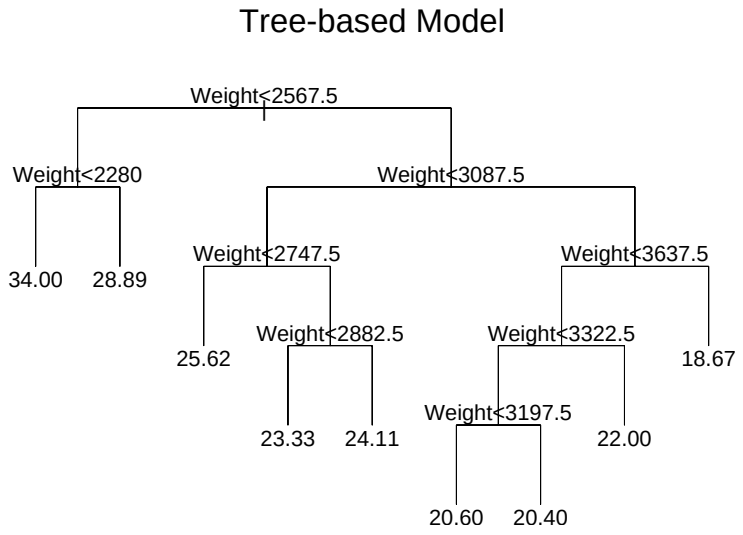

Figure 1.3: *A tree-based model for* `Mileage` *versus* `Weight`.

EXAMPLE OF DATA ANALYSIS

The example that follows describes only one way of analyzing data through the use of statistical modeling. There is no perfect cookbook approach to building models, as different techniques do different things, and not all of them use the same arguments when doing the actual fitting.

The Iterative Process of Model Building

As was discussed at the beginning of this chapter, there are some general steps you can take when building a model:

1. Determine the variables to observe. In a study involving a classical modeling approach, these variables correspond directly to the hypothesis being tested. For data-driven modeling, these variables are the link to the phenomena being modeled.
2. Collect and record the data observations.
3. Study graphics and summaries of the collected data to discover and remove mistakes and to reveal low-dimensional relationships between variables.
4. Choose a model describing the important relationships seen or hypothesized in the data.
5. Fit the model using the appropriate modeling technique.
6. Examine the fit through model summaries and diagnostic plots.
7. Repeat steps 4–6 until you are satisfied with the model.

At any point in the modeling process, you may find that your choice of a model does not appropriately fit the data. In some cases, diagnostic plots may give you clues to improve the fit. Sometimes you may need to try transformed variables or entirely different variables. You may need to try a different modeling technique that will, for example, allow you to fit nonlinear relationships, interactions, or different error structures. At times, all you need to do is remove outlying, influential data, or fit the model robustly. A point to remember is that there is no one answer on how to build good

statistical models. By iteratively fitting, plotting, testing, changing something and then refitting, you will arrive at the best fitting model for your data.

Exploring the Data

The following analysis uses the built-in data set `auto.stats`, which contains a variety of data for car models between the years 1970-1982, including price, miles per gallon, weight, and more. Suppose we want to model the effect that `Weight` has on the gas mileage of a car. The object, `auto.stats`, is not a data frame, so we start by coercing it into a data frame object:

```
> auto.dat <- data.frame(auto.stats)
```

Attach the data frame to treat each variable as a separate object:

```
> attach(auto.dat)
```

Look at the distribution of the data by plotting a histogram of the two variables, `Weight` and `Miles.per.gallon`. First, split the graphics screen into two portions to display both graphs:

```
> par(mfrow = c(1, 2))
```

Plot the histograms:

```
> hist(Weight)
> hist(Miles.per.gallon)
```

The resulting histograms appear as in Figure 1.4.

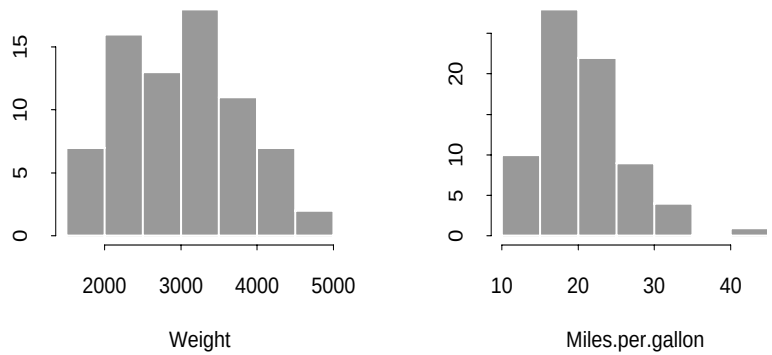

Figure 1.4: *Histograms of* `Weight` *and* `Miles.per.gallon`.

Subsetting (or subscripting) gives you the ability to look at only a portion of the data. For example, type the following to look at only those cars with mileage greater than 34 miles per gallon:

```
> auto.dat[Miles.per.gallon > 34,]

               Price Miles.per.gallon Repair (1978)
    Datsun 210  4589               35             5
        Subaru  3798               35             5
Volk Rabbit(d)  5397               41             5

               Repair (1977) Headroom Rear.Seat Trunk Weight
    Datsun 210             5      2.0      23.5     8   2020
        Subaru             4      2.5      25.5    11   2050
Volk Rabbit(d)             4      3.0      25.5    15   2040
               Length Turning.Circle Displacement Gear.Ratio
    Datsun 210    165             32           85       3.70
        Subaru    164             36           97       3.81
Volk Rabbit(d)    155             35           90       3.78
```

Suppose you want to predict the gas mileage of a particular auto based upon its weight. Create a scatter plot of `Weight` versus `Miles.per.gallon` to examine the relationship between the variables. First, reset the graphics window to display only one graph:

```
> par(mfrow = c(1,1))
```

Plot `Weight` versus `Miles.per.gallon`. The plot appears as in Figure 1.5.

```
> plot(Weight, Miles.per.gallon)
```

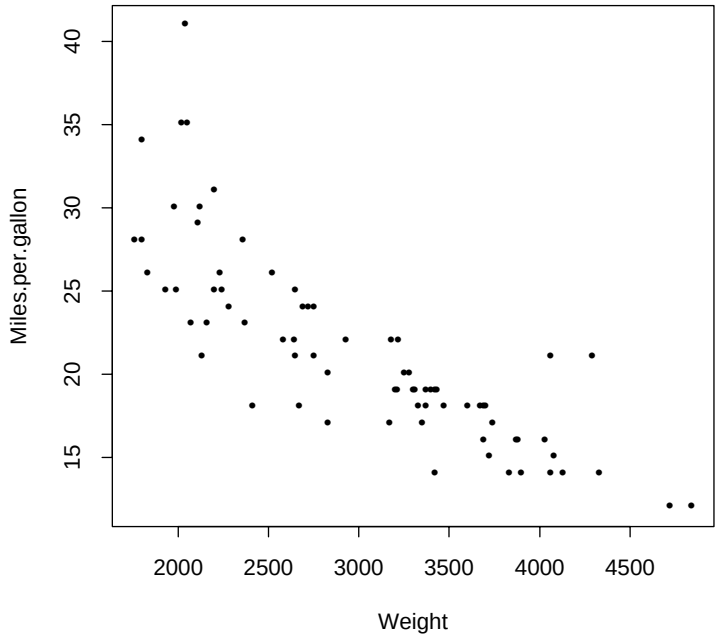

Figure 1.5: *Scatter plot:* `Weight` *versus* `Miles.per.gallon`.

The resulting figure displays a curved scattering of the data. This might suggest a nonlinear relationship. Create a plot from a different perspective, giving gallons per mile (`1/Miles.per.gallon`) as the vertical axis:

```
> plot(Weight, 1/Miles.per.gallon)
```

The resulting scatter plot appears as in Figure 1.6.

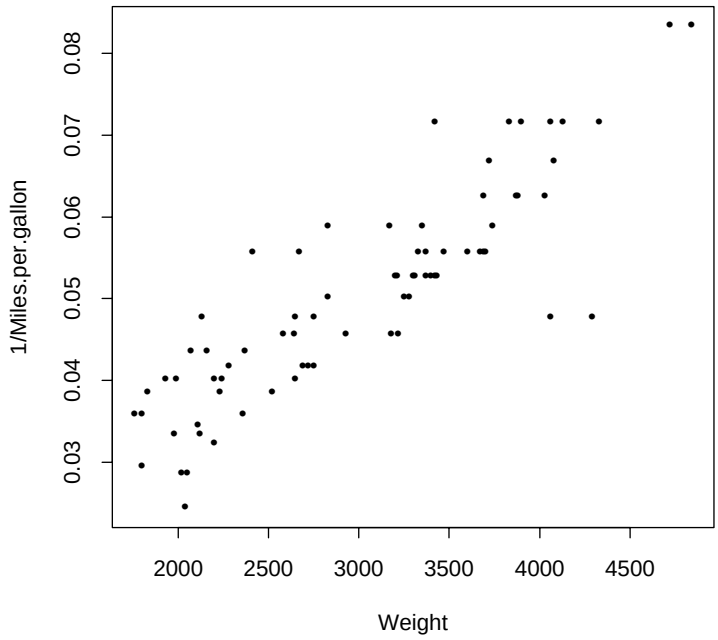

Figure 1.6: *Scatter plot of* `Weight` *versus* `1/Miles.per.gallon`.

Fitting the Model

Gallons per mile is more linear with respect to weight, suggesting that you can fit a linear model to `Weight` and `1/Miles.per.gallon`. The formula `1/Miles.per.gallon ~ Weight` describes this model. Fit the model by using the `lm` function, and name the fitted object `fit1`:

```
> fit1 <- lm(1/Miles.per.gallon ~ Weight)
```

As with any S-PLUS object, when you type the name, `fit1`, S-PLUS prints the object, in this case, using the specific print method for `"lm"` objects:

```
> fit1

Call:
lm(formula = 1/Miles.per.gallon ~ Weight)

Coefficients:
 (Intercept)       Weight
 0.007447302 1.419734e-05
```

```
Degrees of freedom: 74 total; 72 residual
Residual standard error: 0.006363808
```

Plot the regression line to see how well it fits the data. The resulting line appears as in Figure 1.7.

```
> abline(fit1)
```

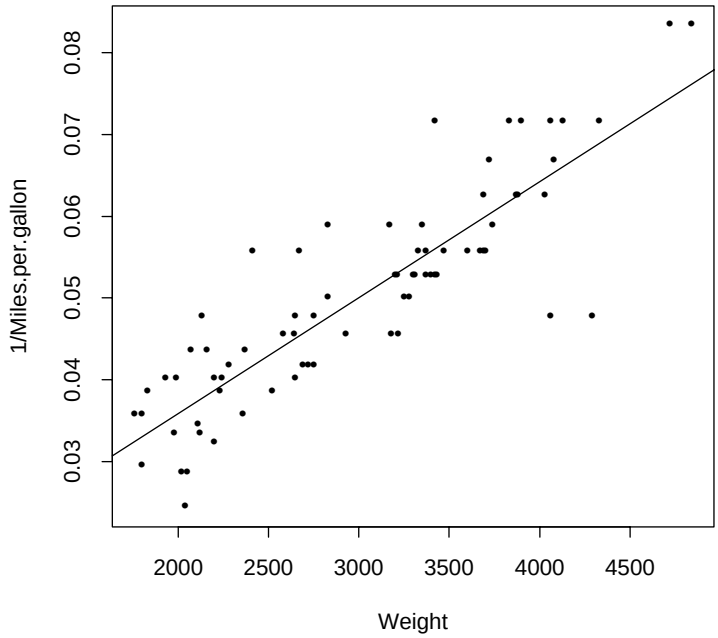

Figure 1.7: *Regression line of* `fit1`.

Judging from Figure 1.7, the regression line does not fit well in the upper range of `Weight`. Plot the residuals versus the fitted values to see more clearly where the model does not fit well.

```
> plot(fitted(fit1), residuals(fit1))
```

The plot appears as in Figure 1.8.

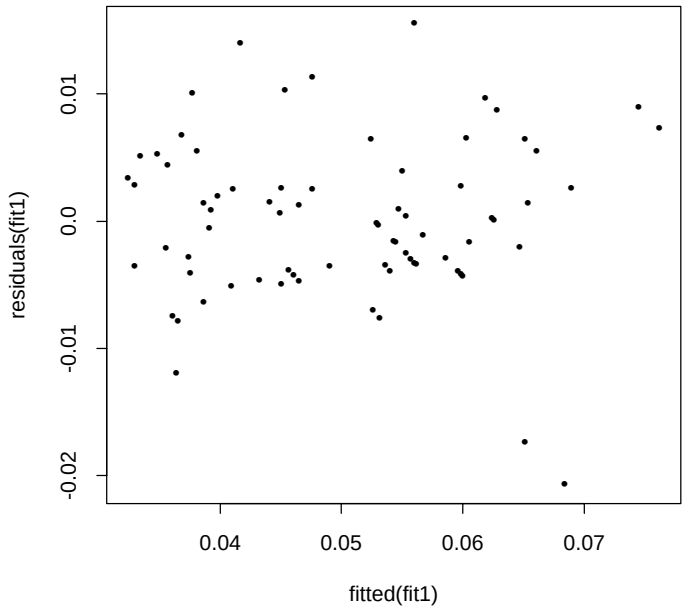

Figure 1.8: *Plot of residuals for* `fit1`.

Note that with the exception of two outliers in the lower right corner, the residuals become more positive as the fitted value increases. You can identify the outliers by typing the following command, then interactively clicking on the outliers with your mouse:

```
> outliers <- identify(fitted(fit1), residuals(fit1),
+ labels = names(Weight))
```

The `identify` function allows you to interactively select the points on the plot. The `labels` argument and `names` function label the points with their names in the object. For more information on the `identify` function, see Chapter 8, Traditional Graphics, in the *Programmer's Guide*. The plot appears as in Figure 1.9.

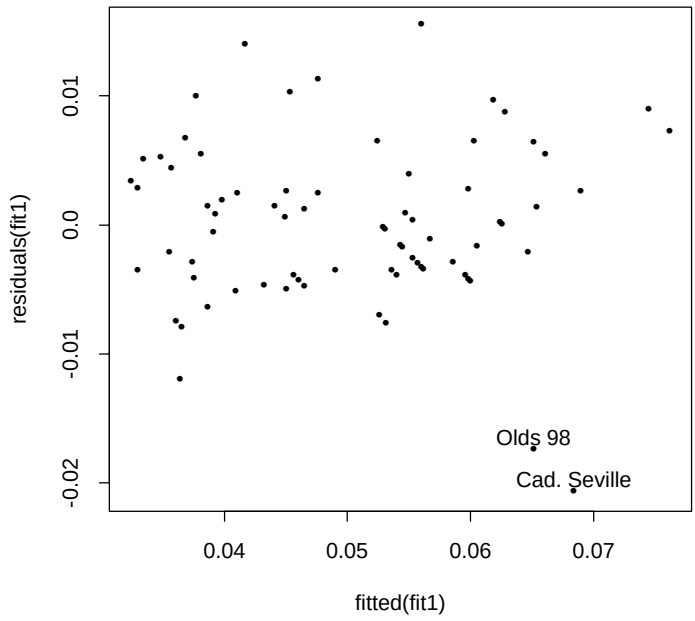

Figure 1.9: *Plot with labeled outliers.*

These outliers correspond to cars with *better* gas mileage than other cars in the study with similar weights. You can remove the outliers using the `subset` argument to `lm`.

```
> fit2 <- lm(1/Miles.per.gallon ~ Weight,
+ subset = -outliers)
```

Plot `Weight` versus `1/Miles.per.gallon`, and also two regression lines, one for the `fit1` object and one for the `fit2` object. Use the `lty=` argument to differentiate between the regression lines:

```
> plot(Weight, 1/Miles.per.gallon)
> abline(fit1, lty=2)
> abline(fit2)
```

The two lines appear with the data in Figure 1.10.

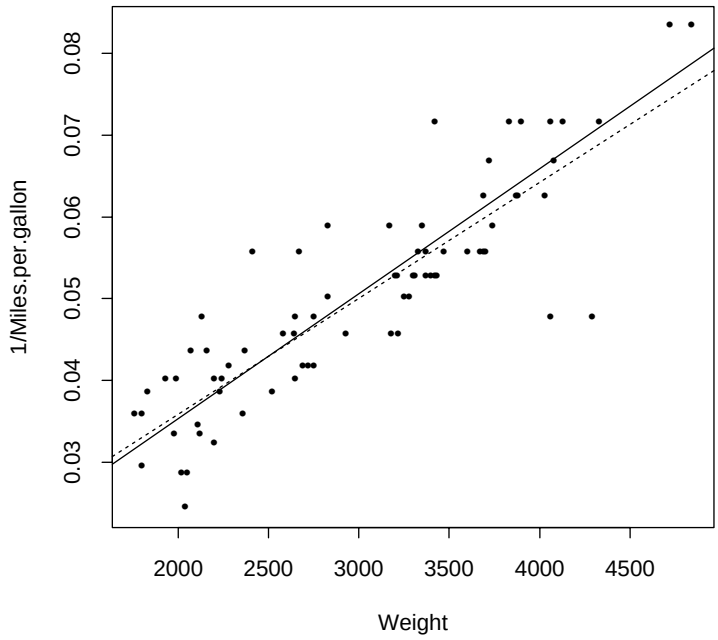

Figure 1.10: *Regression lines of* `fit1` *versus* `fit2`.

A plot of the residuals versus the fitted values shows a better fit. The plot appears as in Figure 1.11.

```
> plot(fitted(fit2), residuals(fit2))
```

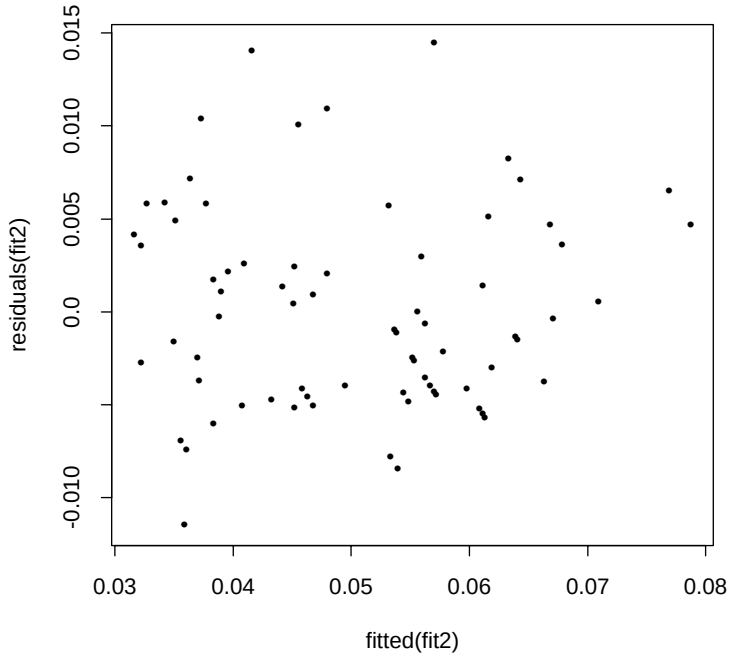

Figure 1.11: *Plot of residuals for* `fit2`.

To see a synopsis of the fit contained in `fit2`, use `summary` as follows:

```
> summary(fit2)

Call: lm(formula = 1/Miles.per.gallon ~ Weight,
subset = - outliers)
Residuals:
      Min        1Q     Median       3Q     Max
 -0.01152 -0.004257 -0.0008586 0.003686 0.01441

Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept)  0.0047     0.0026  1.8103   0.0745
     Weight  0.0000     0.0000 18.0625   0.0000

Residual standard error: 0.00549 on 70 degrees of freedom
Multiple R-squared: 0.8233
F-statistic: 326.3 on 1 and 70 degrees of freedom, the
p-value is 0
```

```
Correlation of Coefficients:
       (Intercept)
Weight -0.9686
```

The summary displays information on the spread of the residuals, coefficients, standard errors, and tests of significance for each of the variables in the model (it includes an intercept by default), and overall regression statistics for the fit. As expected, `Weight` is a very significant predictor of `1/Miles.per.gallon`. The amount of the variability of `1/Miles.per.gallon` explained by `Weight` is about 82%, and the residual standard error is .0055, down about 14% from that of `fit1`.

To see the individual coefficients for `fit2`, use `coef` as follows:

```
> coef(fit2)

 (Intercept)       Weight
0.004713079 1.529348e-05
```

Fitting an Alternative Model

Now consider an alternative approach. Recall the plot in Figure 1.5 showed curvature in the scatter plot of `Weight` versus `Miles.per.gallon`, indicating that a straight line fit is an inappropriate model. You can fit a nonparametric nonlinear model to the data using `gam` using a cubic spline smoother to model the curvature in the data:

```
> fit3 <- gam(Miles.per.gallon ~ s(Weight))
> fit3

Call:
gam(formula = Miles.per.gallon ~ s(Weight))

Degrees of Freedom: 74 total; 69.00244 Residual
Residual Deviance: 704.7922
```

The resulting plot of `fit3` appears as in Figure 1.12.

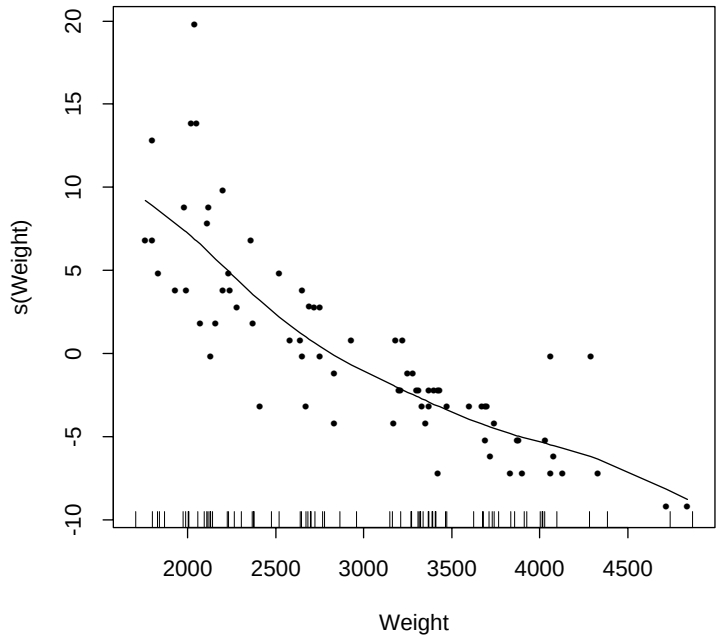

Figure 1.12: *Plot of additive model with smoothed spline term.*

```
> plot(fit3, residuals = T, scale =
+ diff(range(Miles.per.gallon)))
```

The cubic spline smoother in the plot appears to give a good fit to the data. You can check the fit with diagnostic plots of the residuals as we did for the linear models. You should also compare the `gam` model with a linear model using `aov` to produce a statistical test.

Use the `predict` function to make predictions from models. One of the arguments to `predict`, `newdata`, specifies a data frame containing the values at which the predictions are required. If `newdata` is not supplied, the `predict` function will make predictions at the data originally supplied to fit the `gam` model, as in the following example:

```
> predict.fit3 <- predict(fit3)
```

Create a new object `predict.high` and print it to display cars with predicted miles per gallon greater than 30:

```
> predict.high <- predict.fit3[predict.fit3 > 30]
> predict.high

 Ford Fiesta Honda Civic Plym Champ
    30.17946    30.49947   30.17946
```

Conclusions

The previous examples show a few simple methods for taking data and iteratively fitting models until achieving desired results. The chapters that follow discuss the previously mentioned modeling techniques in far greater detail. Before proceeding further, it is good to remember that:

- General formulas define the structure of models.
- Data used in model-fitting are generally in the form of data frames.
- Different methods can be used on the same data.
- A variety of functions are available for diagnostic study of the fitted models.
- The S-PLUS functions, like model-fitting in general, are designed to be very flexible for users. Handling different preferences and procedures in model-fitting are what make S-PLUS very effective for data analysis.

SPECIFYING MODELS IN S-PLUS

2

INTRODUCTION

Models are specified in S-PLUS using *formulas*, which express the conjectured relationships between observed variables in a natural way. Once you begin building models in S-PLUS, you quickly discover that formulas specify models for the wide variety of modeling techniques available in S-PLUS. You can use the same formula to specify a model for linear regression (`lm`), analysis of variance (`aov`), generalized linear modeling (`glm`), generalized additive modeling (`gam`), local regression (`loess`), and tree-based regression (`tree`).

For example, consider the following formula:

```
> mpg ~ weight + displ
```

This formula can specify a least squares regression with `mpg` regressed on two predictors, `weight` and `displ`, or a generalized additive model with purely linear effects.

You can also specify smoothed fits for `weight` and `displ` in the generalized additive model as follows:

```
> mpg ~ s(weight) + s(displ)
```

and compare the resulting fit with the purely linear fit to see if some nonlinear structure must be built into the model.

Thus, formulas provide the means for you to specify models for all modeling techniques: parametric or nonparametric, classical or modern. This chapter provides you with an introduction to the syntax used for specifying statistical models.

The chapters that follow make use of this syntax in a wide variety of specific examples.

BASIC FORMULAS

A formula is an S-PLUS expression that specifies the form of a model in terms of the variables involved. For example, to specify that `mpg` is modeled as a linear and additive model of the two predictors `weight` and `displ`, you use the following formula:

```
> mpg ~ weight + displ
```

The tilde (~) character separates the response variable from the explanatory variables. For something to be interpretable as a variable it must be one of the following:

- numeric vector
- factor or ordered factor
- matrix

For numeric vectors, one coefficient is fit; for matrices, a coefficient for each column is fit; for factors, the *equivalent* of one coefficient is fit for *each* level of the factor.

You can use any acceptable S-PLUS expression in the place of any of the variables, provided the expression evaluates to something *interpretable as one or more variables*. Thus, the formula

```
> log(mpg) ~ weight + poly(displ,2)
```

specifies that the log of `mpg` is modeled as a linear function of `weight` and a quadratic polynomial of `displ`.

Continuous Data

Each continuous variable you provide generates one coefficient in the fitted model. Thus, the formula

```
> mpg ~ weight + displ
```

fits the model

$$\texttt{mpg} = \beta_0 + \beta_1 \texttt{weight} + \beta_2 \texttt{displ} + \varepsilon$$

A formula *always* implicitly includes an intercept term (β_0 in the above formula).

You can, however, remove the intercept term by specifying the model with -1 as an explicit predictor:

```
> mpg ~ -1 + weight + displ
```

Similarly, you can explicitly include an intercept with a + 1.

When you provide a numeric matrix as one term in a formula, each column of the matrix is taken to be a separate variable in the model. Any names associated with the columns are carried along as labels in the subsequent fits.

Categorical Data

When you specify categorical variables (factors, ordered factors, or categories) as predictors in the formulas, the modeling functions fit a coefficient for each level of the variable. For example, to model `salary` as a linear model of `age` (continuous) and `gender` (factor) you specify it as follows:

```
> salary ~ age + gender
```

However, a different parameter is fitted for each of the two levels of gender. This is equivalent to fitting two dummy variables–one for males and one for females. Thus, you need not create and specify dummy variables in the model. (In actuality only one additional parameter is fitted, because the parameters are not independent of the intercept term. More details on over-parameterization and the defining of contrasts between factor levels is provided in the section Contrasts: The Coding of Factors.

General Formula Syntax

This section provides a table summarizing the meanings of the operators in formulas and shows how to create and save formulas.

Table 2.1, based on page 29 of *Statistical Models in S*, summarizes the syntax of formulas.

Table 2.1: *A summary of formula syntax.*

Expression	**Meaning**
T ~ F	T is modeled as F
$F_a + F_b$	Include both F_a and F_b in the model
F_a - F_b	Include all of F_a except what is in F_b in the model
$F_a : F_b$	The interaction between F_a and F_b
F_a * F_b	$F_a + F_b + F_a : F_b$
F_b %in% F_a	F_b is nested within F_a
F_a / F_b	$F_a + F_b$ %in% F_a
F^m	All terms in F crossed to order m

You can create and save formulas as objects using the `formula` function:

```
> form.eg.1 <- formula(Fuel ~ poly(Weight, 2) + Disp. +
+ Type)
> form.eg.1

Fuel ~ poly(Weight, 2) + Disp. + Type
```

INTERACTIONS IN FORMULAS

You can specify interactions for categorical data (e.g., factors), continuous data, or a mixture of the two. In each case, additional parameters are fitted that are appropriate for the different types of variables specified in the model. The syntax for specifying the interaction is the same in each case, but the interpretation varies depending on the data types.

To specify a particular interaction between two or more variables use a colon (`:`) between the variable names. Thus, to specify the interaction between `gender` and `race`, use the following term:

```
gender:race
```

You can use an asterisk (`*`) to specify *all* terms in the model created by the subsets of the variables named along with the `*`. Thus

```
salary ~ age * gender
```

is equivalent to

```
salary ~ age + gender + age:gender
```

You can remove terms with a minus or hyphen (`-`). Thus

```
salary ~ gender*race*education - gender:race:education
```

is equivalent to

```
salary ~ gender + race + education + gender:race +
gender:education + race:education
```

the model consisting of all the terms in the full model except the three-way interaction. A third way to specify this model is by using the power notation to get all terms of order two or less:

```
salary ~ (gender + race + education) ^ 2
```

Categorical Data

For categorical data, interactions add coefficients for each combination of the levels of the named factors. Thus, for two factors, `Opening` and `Mask`, with three and five levels, respectively, the `Opening:Mask` term in a model adds 15 additional parameters to the

model. (In practice, because of dependencies among the parameters, only some of the total number of parameters specified by a model are fitted.)

You can specify, for example, a two-way analysis of variance with the simplified notation as follows:

```
skips ~ Opening * Mask
```

The fitted model is

$$\texttt{skips} = \mu + \texttt{Opening}_i + \texttt{Mask}_j + (\texttt{Opening : Mask})_{ij} + \varepsilon$$

Continuous Data

You can specify interactions between continuous variables in the same way as you do for categorical and a mixture of categorical and continuous variables. However, the interaction specified is multiplicative. Thus

```
mpg ~ weight * displ
```

fits the model

$$\texttt{mpg} = \beta_0 + \beta_1\texttt{weight} + \beta_2\texttt{displ} + \beta_3(\texttt{weight})(\texttt{displ}) + \varepsilon$$

NESTING IN FORMULAS

Nesting arises in models when the levels of one or more factors make sense only *within* the levels of one or more other factors. For example, in sampling the U.S. population, a sample of states is drawn, from which a sample of counties is drawn, from which a sample of cities is drawn, from which a sample of families or households is drawn. Counties are nested within states, cities are nested within counties, and households are nested within cities.

There is special syntax to emphasize the nesting of factors within others. You can write the county within state model as:

```
state + county
```

You can state the model more succinctly with

```
state / county
```

which means "state and then county within state." The slash (/) used for nested models is the counterpart of the asterisk (*) which is used for factorial models.

You can specify the full state-county-city-household example as follows:

```
state / county / city / household
```

INTERACTIONS BETWEEN CATEGORICAL AND CONTINUOUS VARIABLES

For categorical data combined with continuous data, *interactions* add a coefficient for the continuous variable for each level of the categorical variable. So, for example, you can easily fit a model with different slope estimates for different groups where the categorical variables specify the groups.

When you combine categorical and continuous data using the *nesting* syntax, you can specify analysis of covariance models simply. If gender (categorical) and age (continuous) are predictors in a model, you can fit separate slopes for each gender by nesting. First, make gender a factor (that is, `gender <- factor(gender)`). Then the analysis of covariance model is:

```
salary ~ gender / age
```

This fits a model equivalent to:

$$\mu + \texttt{gender}_i + \beta_i \texttt{age}$$

This is also equivalent to `gender * age`. However, the *parameterization* for the two models is different. When you fit the nested model, you get estimates of the individual slopes for each group. When you fit the factorial model, you get an overall slope estimate plus the deviations in the slope for the different group contrasts. For example, in `gender / age`, the formula expands into main effects for `gender` followed by `age` within each level of `gender`. One coefficient is fitted for `age` from each level of `gender`. Another coefficient estimates the contrast between the two levels of `gender`. Thus, the nesting model fits the following type of model:

$$\text{Salary}_M = \mu + \alpha_g + \beta_1 \times \text{age}$$
$$\text{Salary}_F = \mu - \alpha_g + \beta_2 \times \text{age}$$

The intercept is μ, the contrast is α_g, and the model has coefficients β_i for `age` within each level of `gender`. Thus, you have separate slope estimates for each group. Conversely, the factorial model `gender * age` fits the following model:

$$\text{Salary}_M = \mu - \alpha_g + \beta \times \text{age} - \gamma \times \text{age}$$

$$\text{Salary}_F = \mu + \alpha_g + \beta \times \text{age} + \gamma \times \text{age}$$

You get the overall slope estimate β plus the deviations in the slope for the different group contrasts.

You can fit the "equal slope, separate intercept" model by specifying:

```
salary ~ gender + age
```

This fits a model equivalent to:

$$\mu + \text{gender}_i + \beta \times \text{age}$$

USING THE PERIOD OPERATOR IN FORMULAS

A single period (".") operator can act as a default left or right side of a formula. There are numerous ways you can use "." in formulas. To see how "." is used, consider the function `update`, which allows you to modify existing models. The following example uses the data frame `fuel.frame` to display the usage of the single "." in formulas:

```
> fuel.null <- lm(Fuel ~ 1, fuel.frame)
```

If `Weight` is the single best predictor, use `update` to add it to the model:

```
> fuel.wt <- update(fuel.null, . ~ . + Weight)
> fuel.wt

Call:
lm(formula = Fuel ~ Weight, data = fuel.frame)

Coefficients:
 (Intercept)      Weight
   0.3914324 0.00131638
Degrees of freedom: 60 total; 58 residual
Residual standard error: 0.387715
```

The single dots "." in the above example are replaced on the left and right side of the tilde "~" by the left and right sides of the formula used to fit the object `fuel.null`. Two additional methods use "." in reference to data frame objects. In the following example, a linear model is fit using the data frame `fuel.frame`:

```
> lm(Fuel ~ ., data = fuel.frame)
```

Here, the new model includes all the predictors in `fuel.frame`. In the example,

```
> lm(skips ~ .^2, data = solder.balance)
```

all main effects and second-order interactions in `solder.balance` are used to fit the model.

COMBINING FORMULAS WITH FITTING PROCEDURES

Once you specify a model with its associated formula, you can fit it to a given data set by passing the formula and the data to the appropriate fitting procedure. For the following example, you create the data frame `auto.dat` from the data set `auto.stats` by typing,

```
> auto.dat <- data.frame(auto.stats)
```

To fit a linear model to `Miles.per.gallon ~ Weight + Displacement`, when `Miles.per.gallon`, `Weight`, and `Displacement` are columns in a data frame named `auto.dat`, you type:

```
> lm(Miles.per.gallon ~ Weight + Displacement, auto.dat)
```

You could fit a *smoothed* model to the same data with:

```
> loess(Miles.per.gallon ~ s(Weight) + s(Displacement),
+ auto.dat)
```

All the fitting procedures take a formula and an optional data set (actually a data frame) as the first two arguments. If the individual variables are in your search path, or you attached the data frame `auto.dat`, you can omit the data specification and type more simply:

```
> lm(Miles.per.gallon ~ Weight + Displacement)
```

or

```
> loess(Miles.per.gallon ~ s(Weight) + s(Displacement))
```

Warning

If you attach a data frame for fitting models and have objects in your **.Data** directory with names that match those in the data frame, the data frame variables are *masked* and are not used in the actual model fitting.

Composite Terms in Formulas

As was previously mentioned, certain operators have special meaning when used in formula expressions. They must appear only at the top level in the formulas and only on the right side of the "~". However, if the operators appear within arguments to functions within the formula, then they work as they normally do in S-PLUS. In the formula

```
Kyphosis ~ poly(Age, 2) + I((Start > 12) * (Start - 12))
```

the '*' and '-' operators evaluate as they normally do in S-PLUS, without the special meaning they have when used at the top level within the formula because they appear within arguments to the `I` function. The `I` function's sole purpose, in fact, is to protect special operators on the right side of formulas.

You can use any acceptable S-PLUS expression in the place of any variable within the formula, provided the expression evaluates to something *interpretable* as *one or more variables*. The expression must be one of the following:

- Numeric vector
- Factor, ordered factor, or category
- Matrix

Thus, certain composite terms (among them `poly`, `I`, and `bs`) can be used as formula variables. Matrices used in formulas are treated as single terms. You can also use functions that produce factors and categories as formula variables.

CONTRASTS: THE CODING OF FACTORS

A coefficient for each level of a factor cannot usually be estimated because of dependencies among the coefficients of the overall model. An example of this is the sum of all the *dummy* variables for any factor, which is a vector of all ones. This corresponds to the term used for fitting an intercept. Overparametrization induced by dummy variables is removed prior to fitting, by replacing the dummy variables with a set of linear combinations of the dummy variables, which are:

- functionally independent of each other, and
- functionally independent of the *sum* of the dummy variables.

A factor with k levels has $k-1$ possible independent linear combinations. A particular choice of linear combinations of the dummy variables is called a set of *contrasts*. Any choice of contrasts for a factor alters the specific individual coefficients in the model, but *does not change the overall contribution of the term to the fit.*

Built-In Contrasts

S-PLUS provides four different kinds of contrasts as built-in functions:

- *Helmert contrasts*

  The function `contr.helmert` implements Helmert contrasts. The jth linear combination is the difference between the level $j+1$ and the average of the first j. The following example returns a Helmert parametrization based upon four levels:

```
> contr.helmert(4)

  [,1] [,2] [,3]
1   -1   -1   -1
2    1   -1   -1
3    0    2   -1
4    0    0    3
```

- *Orthogonal polynomials*

  The function `contr.poly` implements polynomial contrasts. Individual coefficients represent orthogonal polynomials if the levels of the factor are equally spaced numeric values. In

general, the function produces k - 1 orthogonal contrasts representing polynomials of degree 1 to k - 1. The following example uses four levels:

```
> contr.poly(4)

               L      Q          C
[1,] -0.6708204   0.5 -0.2236068
[2,] -0.2236068  -0.5  0.6708204
[3,]  0.2236068  -0.5 -0.6708204
[4,]  0.6708204   0.5  0.2236068
```

- *Sum*

  The function `contr.sum` implements sum contrasts. This produces contrasts between each of the first k - 1 levels and level k:

```
> contr.sum(4)

  [,1] [,2] [,3]
1    1    0    0
2    0    1    0
3    0    0    1
4   -1   -1   -1
```

- *Treatment*

  The function `contr.treatment` implements treatment contrasts. This is not really a contrast but simply includes each level as a dummy variable excluding the first one. *This generates statistically dependent coefficients even in balanced experiments*.

```
> contr.treatment(4)

  2 3 4
1 0 0 0
2 1 0 0
3 0 1 0
4 0 0 1
```

  This is not a true set of contrasts, for the columns do not sum to zero and thus are not orthogonal to the vector of ones.

Specifying Contrasts

Use the functions `C`, `contrasts`, and `options` to specify contrasts. Use `C` to specify a contrast as you type the formula; it is the simplest way to alter the choice of contrasts. Use `contrasts` to specify a contrast attribute on a factor. Use `options` to specify the default choice of contrasts for all factors.

The C Function

As was previously stated, the `C` function is the simplest way to alter the choice of contrasts. The arguments to the function are `C(object, contr)` where `object` is a factor or ordered factor, and `contr` is the contrast to alter. An optional argument, `how.many`, is for the number of contrasts to assign to the factor. The value returned by `C` is the factor with a `"contrasts"` attribute equal to the specified contrast matrix.

For example, with the soldering experiment contained in `solder.balance`, you could specify `sum` contrasts for `Mask` with `C(Mask, sum)`. You could also have your own contrast function, `special.contrast`, that returns a matrix of the desired dimension with the call `C(Mask, special.contrast)`.

Note

If you create your own contrast function, it must return a matrix with the following properties:

- The number of rows must be equal to the number of levels specified and the number of columns one less than the number of rows.
- The columns must be linearly independent of each other and of the vector of all ones.

You can also specify contrasts by supplying the contrast matrix directly. For example, `quality` is a factor with four levels:

```
> levels(quality)

[1] "tested-low"  "low"       "high"      "tested-high"
```

You can contrast levels 1 and 4 with levels 2 and 3 by including `quality` in the model formula as `C(quality, c(1, -1, -1, 1))`. Two additional contrasts are generated, orthogonal to the one supplied.

To contrast the "low" values versus the "high" values, provide the contrasts as a matrix:

```
> contrast.mat

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

The contrasts Function

Use the `contrasts` function to *set* the contrasts for a particular factor whenever it appears. The `contrasts` function extracts contrasts from a factor and returns them as a matrix. The following sets the contrasts for the `quality` factor:

```
> contrasts(quality) <- contrast.mat
> contrasts(quality)

            [,1] [,2] [,3]
 tested-low    1    1 -0.5
        low   -1    1  0.5
       high   -1   -1 -0.5
tested-high    1   -1  0.5
```

Now `quality` has the `contrast.mat` parametrization by default any time it appears in the formula. To override this new default setting, supply a new contrast specification through the `C` function.

Setting the contrasts Option

Use the contrast `options` function to change the default choice of contrasts for *all* factors, as in the following example:

```
> options()$contrasts

          factor      ordered
 "contr.helmert" "contr.poly"

> options(contrasts = c(factor = "contr.treatment",
+ ordered = "contr.poly"))
> options()$contrasts

[1] "contr.treatment" "contr.poly"
```

In summary, the `options` function sets the default choice of contrasts globally (on all factors); the `contrasts` function sets the default choice of contrasts on a particular factor; and the `C` function overrides the default.

USEFUL FUNCTIONS FOR MODEL FITTING

As model building proceeds, you'll find several functions useful for adding and deleting terms in formulas. The `update` function starts with an existing fit and adds or removes terms as you specify. For example, create a data frame from the data set `fuel.frame` by typing:

```
> fuel.fit <- data.frame(fuel.frame)
```

Suppose you save the result of `lm` as follows:

```
> fuel.lm.fit <- lm(Mileage ~ Weight + Disp., fuel.fit)
```

You can use `update` to change, for example, the response to `Fuel`. Use a period on either side of the tilde to represent the current state of the model in the fit object (`fuel.lm.fit` below).

```
> update(fuel.lm.fit, Fuel ~ . )
```

Recall that the period ("`.`") means to include every predictor that is in `fuel.lm.fit` in the new model. Only the response changes.

You could drop the `Disp.` term, keeping the response the same by:

```
> update(fuel.lm.fit, . ~ . - Disp.)
```

Another useful function is `drop1`, which produces statistics obtained from dropping each term out of the model one at a time. For example:

```
> drop1(fuel.lm.fit)

Single term deletions

Model: Mileage ~ Weight + Disp.
       Df Sum of Sq   RSS    Cp
<none>              380.3 420.3
Weight  1     323.4 703.7 730.4
Disp.   1       0.6 380.8 407.5
```

Each line presents model summary statistics corresponding to dropping the term indicated in the first column. The first line in the table corresponds to the original model, that is, no terms (`<none>`) are deleted.

There is also an `add1` function which adds one term at a time. The second argument to `add1` provides the *scope* for added terms. The scope argument can be a formula or a character vector indicating the terms to be added. The resulting table prints a line for each term indicated by the scope argument.

```
> add1(fuel.lm.fit, c("Type", "Fuel"))

Single term additions

Model: Mileage ~ Weight + Disp.
       Df Sum of Sq     RSS      Cp
<none>              380.271 420.299
  Type  5   119.722 260.549 367.292
  Fuel  1   326.097  54.173 107.545
```

OPTIONAL ARGUMENTS TO MODEL-FITTING FUNCTIONS

In most model-building calls, you will need to specify the data frame to use. You may need arguments that check for missing values in the data frame, or select only particular portions of the data frame to use in the fit. The following list summarizes standard optional arguments for most model-fitting functions (other than nonlinear models) you can use in the model fit:

- `data`: specifies a data frame to interpret the variables named in the formula, or in the `subset` and `weights` arguments. The following example fits a linear model to data in the `fuel.frame` data frame:

  ```
  > fuel.lm <- lm(Fuel ~ Weight + Disp.,
  + data = fuel.frame)
  ```

- `weights`: specifies a vector of observation of weights. If `weights` is supplied, the algorithm fits to minimize the sum of the squared residuals multiplied by the weights:

  $$\sum w_i r_i^2$$

  Negative weights generate an S-PLUS error. We recommend that the weights be strictly positive, since zero weights give no residuals. The following example fits a linear model to the `claims` data frame, and uses `number` with the `weights` argument:

  ```
  > claims.fit <- lm(cost ~ age + type + car.age,
  + claims, weights = number, na.action = na.omit)
  ```

  The `number` in the preceding call corresponds to the number of claims per type of car in the `claims` data frame.

- `subset`: indicates a subset of the rows of the data to be used in the fit. The expression should evaluate to a logical or numeric vector, or a character vector with appropriate row names. The following example removes outliers and fits a linear model to data in the `auto.dat` data frame:

  ```
  > fit <- lm(1/Miles.per.gallon ~ Weight,
  + subset = -outliers)
  ```

- na.action: a missing-data filter function, applied to the model frame, after any subset argument has been used. The following example uses na.omit with the na.action argument to drop any row of the data frame that contains a missing value:

```
> ozone.lm <- lm(ozone ~ temperature + wind,
+ data=air, subset=wind > 8, na.action=na.omit)
```

Each model fitting function has nonstandard optional arguments, not listed above, which you can use to fit the appropriate model. The following chapters describe the available arguments for each model type.

STATISTICAL INFERENCE FOR ONE- AND TWO-SAMPLE PROBLEMS

3

INTRODUCTION

Suppose you have one or two samples of data that are *continuous* in the sense that the individual observations can take on any possible value in an interval. You often want to draw conclusions from your data concerning underlying "population" or distribution model parameters which determine the character of the observed data. The parameters which are most often of interest are the mean and variance in the case of one sample, and the relative means and variances and the correlation coefficient in the case of two samples. This chapter shows you how to use S-PLUS to carry out statistical inference for these parameters.

Often, your samples of data are assumed to come from a distribution that is *normal*, or *Gaussian*. A normal distribution has the familiar bell-shaped population "frequency" curve (or *probability density*) shown by the solid line in Figure 3.1. Another common assumption is that the observations *within* a sample are *serially uncorrelated* with one another. In fact, the data seldom come from an exactly normal distribution. Usually, a more accurate assumption is that the samples are drawn from a *nearly normal* distribution–that is, a nearly bell-shaped curve whose tails do not go to zero in quite the same way as those of the true normal distribution, as shown by the dotted line in Figure 3.1.

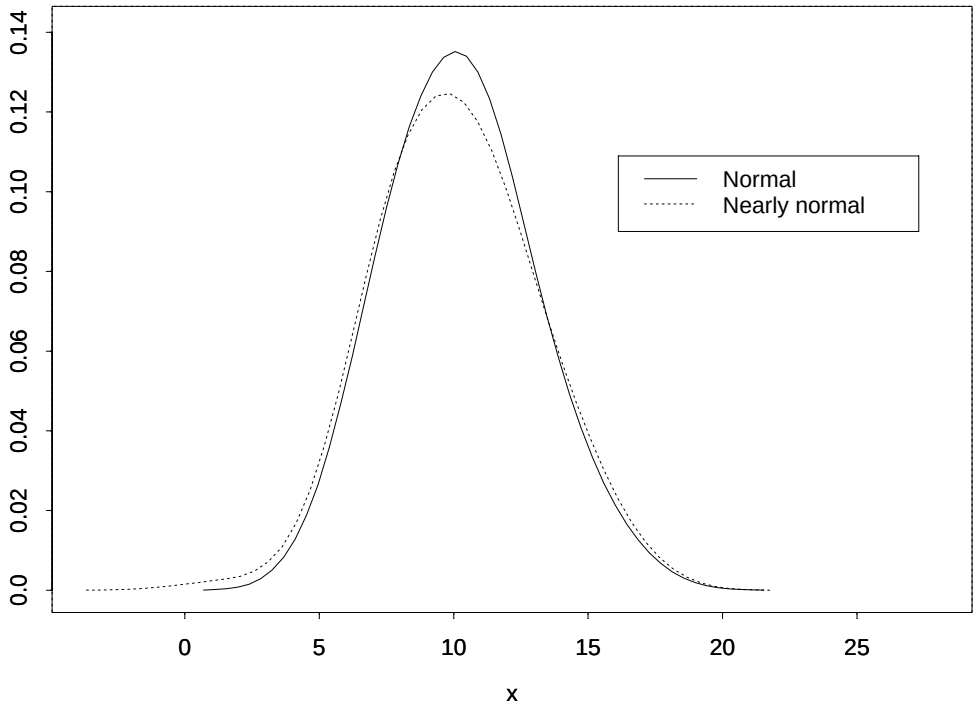

Figure 3.1: *Normal and nearly normal densities.*

It is important that you be aware that nearly normal distributions, which have "heavier tails" than a normal distribution, give rise to *outliers*, that is, unusually aberrant or deviant data values. For example, in Figure 3.1 the left-hand tail of the nearly normal distribution is heavier than the tail of the normal distribution, but the right hand tail is not, and so this nearly normal distribution generates outliers which fall to the left (smaller values than) the bulk of the data.

Even though your data has only a nearly normal distribution, rather than a normal distribution, you can use a normal distribution as a good "nominal" model, as indicated by Figure 3.1. Thus, you are interested in knowing the values of the parameters of a normal distribution (or of two normal distributions in the case of two samples) that provides a good nominal distribution model for your data.

A normal distribution is characterized by two parameters: the *mean* μ and the *variance* σ^2, or, equivalently, the mean and the *standard deviation* σ (the square root of the variance). The mean *locates* the

center of symmetry of the normal distribution, and so the parameter μ is sometimes referred to as the *location*. Similarly, the standard deviation provides a measure of the spread of the distribution, and thus can be thought of as a *scale* parameter.

In the case of two samples, $X_1, X_2, \ldots, X_n$ and $Y_1, Y_2, \ldots, Y_n$, for two variables X and Y, you may also be interested in the value of the *correlation coefficient* ρ. The parameter ρ measures the correlation (or linear dependency) between the variables X and Y. The value of ρ is reflected in the *scatter plot* obtained by plotting Y_i versus X_i for $i = 1, 2, \ldots, n$. A scatterplot of Y_i versus X_i, which has a roughly elliptical shape, with the values of Y_i increasing with increasing values of X_i, corresponds to positive correlation ρ (see, for example, Figure 3.7). An elliptically-shaped scatter plot with the values of Y_i decreasing with increasing values of X_i corresponds to negative correlation ρ. A circular shape to the scatter plot corresponds to a zero value for the correlation coefficient ρ.

Keep in mind that the correlation between two variables X and Y, as just described, is quite distinct from serial correlation between the observations within one or both of the samples when the samples are collected over time. Whereas the former reveals itself in a scatterplot of the Y_i versus the X_i, the latter reveals itself in scatter plots of the observations versus *lagged* values of the observations; for example, a scatter plot of Y_i versus Y_{i+1} or a scatter plot of X_i versus X_{i+1}. If these scatter plots have a circular shape, the data is serially uncorrelated. Otherwise, the data has some serial correlation.

Generally, you must be careful not to assume that data collected over time is serially uncorrelated. You need to check this assumption carefully, because the presence of serial correlation invalidates most of the methods of this chapter.

To summarize: You want to draw conclusions from your data concerning the population mean and variance parameters μ and σ^2 for one sample of data, and you want to draw conclusions from your data concerning the population means μ_1, μ_2, the population variances σ_1^2, σ_2^2 and the population correlation coefficient ρ for two samples of data. You frame your conclusions about the above

parameters in one of the following two types of *statistical inference* statements, illustrated for the case of the population mean μ in a one-sample problem:

- *A CONFIDENCE INTERVAL.* With probability $1 - \alpha$, the mean μ lies within the *confidence interval (L,U).*
- *A HYPOTHESIS TEST.* The computed statistic *T* compares the *null hypothesis* that the mean μ has the specified value μ_0 with the *alternative hypothesis* that $\mu \neq \mu_0$. At any level of significance greater than the reported *p-value* for *T*, we *reject* the null hypothesis in favor of the alternative hypothesis.

A more complete description of confidence intervals and hypothesis tests is provided in the section Statistical Inference on page 59.

Classical methods of statistical inference, such as *Student's t* methods, rely on the assumptions that the data come from a normal distribution and the observations within a sample are serially uncorrelated. If your data contain outliers, or are strongly nonnormal, or if the observations within a sample are serially correlated, the classical methods of statistical inference can give you very misleading results. Fortunately, there are *robust* and *nonparametric* methods which give reliable statistical inference for data that contain outliers or are strongly nonnormal. Special methods are needed for dealing with data that are serially correlated. See, for example, Heidelberger and Welch (1981).

In this chapter, you learn to use S-PLUS functions for making both classical and robust or nonparametric statistical inference statements for the population means and variances for one and two samples, and for the population correlation coefficient for two samples. The basic steps in using S-PLUS functions are essentially the same no matter which of the above parameters you are interested in. They are as follows:

1. *Setting up your data.*

   Before S-PLUS can be used to analyze the data, you must put the data in a form that S-PLUS recognizes.

2. *Exploratory data analysis*, or *EDA*.

   EDA is a graphically-oriented method of data analysis which helps you determine whether the data support the assumptions required for the classical methods of statistical inference: an outlier-free nearly normal distribution and serially uncorrelated observations

3. *Statistical inference*.

   Once you've verified that your sample or samples are nearly normal, outlier-free, and uncorrelated, you can use classical methods of statistical inference which assume a normal distribution and uncorrelated observations, to draw conclusions from your data.

   If your data are not nearly normal and outlier-free, the results of the classical methods of statistical inference may be misleading. Hence, you often need "robust" or "nonparametric" methods, as described in the section Robust and Nonparametric Methods on page 61.

BACKGROUND

This section prepares you for using the S-PLUS functions in the remainder of the chapter by providing brief background information on the following three topics: *exploratory data analysis*, *statistical inference*, and *robust and nonparametric methods*.

Exploratory Data Analysis

The classical methods of statistical inference depend heavily on the assumption that your data is outlier-free and nearly normal, and that your data is serially uncorrelated. *Exploratory data analysis* (EDA) uses graphical displays to help you obtain an understanding of whether or not such assumptions hold. Thus, you should always carry out some graphical exploratory data analysis (EDA) to answer the following questions:

- Do the data come from a nearly normal distribution?
- Do the data contain outliers?
- If the data were collected over time, is there any evidence of serial correlation (correlation between successive values of the data)?

You can get a pretty good picture of the shape of the distribution generating your data, and also detect the presence of outliers, by looking at the following collection of four plots: a *histogram*, a *boxplot*, a *density plot*, and a *normal qq-plot*. Examples of these four plots are provided by Figure 3.2.

Density plots are essentially smooth versions of histograms, which provide smooth estimates of population *frequency*, or *probability density* curves; for example, the normal and nearly normal curves of Figure 3.1. Since the latter are smooth curves, it is both appropriate and more pleasant to look at density plots than at histograms.

A normal qq-plot (or quantile-quantile plot) consists of a plot of the ordered values of your data versus the corresponding quantiles of a *standard* normal distribution; that is, a normal distribution with mean zero and variance one. If the qq-plot is fairly linear, your data are reasonably Gaussian; otherwise, they are not.

Of these four plots, the histogram and density plot give you the best picture of the distribution shape, while the boxplot and normal qq-plot give the clearest display of outliers. The boxplot also gives a clear indication of the median (the solid dot inside the box), and the upper and lower quartiles (the upper and lower ends of the box).

A simple S-PLUS function can create all four suggested distributional shape EDA plots, and displays them all on a single screen or a single hard copy plot. Define the function as follows:

```
> eda.shape <- function(x)
+ {
+       par(mfrow = c(2, 2))
+       hist(x)
+       boxplot(x)
+       iqd <- summary(x)[5] - summary(x)[2]
+       plot(density(x,width=2*iqd), xlab = "x",
+       ylab = "", type = "l")
+       qqnorm(x)
+       qqline(x)
+ }
```

This function is used to make the EDA plots you see in the remainder of this chapter. (The argument `width = 2*iqd` to `density` sets the degree of smoothness of the density plot in a good way. For more details on writing functions, see the *Programmer's Guide*.)

If you have collected your data over time, the data may contain serial correlation. That is, the observations may be correlated with one another at different times. The assessment of whether or not there is any time series correlation in the context of confirmatory data analysis for location and scale parameters (and more generally) is an often-neglected task.

You can check for obvious time series features, such as trends and cycles, by looking at a plot of your data against time, using the function `ts.plot`. You can check for the presence of less obvious

serial correlation by looking at a plot of the autocorrelation function for the data, using the `acf` function. These plots can be created, and displayed one above the other, with the following S-PLUS function:

```
> eda.ts <- function(x)
+ {
+       par(mfrow=c(2,1))
+       ts.plot(x)
+       acf(x)
+       invisible()
+ }
```

This function is used to make the time series EDA plots you find in the remainder of this chapter. See, for example, Figure 3.3. The discussion of Figure 3.3 includes a guideline for interpreting the `acf` plot.

Warning

If either the time series plot or the acf plot suggests the presence of serial correlation, then you can place little credence in the results computed in this chapter, using either the Student's *t*-statistic approach or using the nonparametric Wilcoxon approach! A method for estimating the population mean in the presence of serial correlation is described by Heidelberger and Welch (1981). Seek expert assistance, as needed.

Statistical Inference

Formal methods of *statistical inference* provide probability-based statements about population parameters such as the mean, variance, and correlation coefficient for your data. You may be interested in a simple *(point) estimate* of a population parameter. For example, the sample mean is a point estimate of the population mean. However, a point estimate neither conveys any uncertainty about the value of the estimate, nor indicates whether a hypothesis about the population parameter is to be rejected. To address these two issues, you will usually use one or both of the following methods of statistical inference: *confidence intervals* and *hypothesis tests*.

We define these two methods for you, letting θ represent any one of the parameters you may be interested in; for example, θ may be the mean μ, or the difference between two means $\mu_1 - \mu_2$, or the correlation coefficient ρ.

CONFIDENCE INTERVALS. A $(1 - \alpha)100\%$ confidence interval for the true but unknown parameter θ is any interval of the form *(L,U)*, such that the probability is $1-\alpha$ that *(L,U)* contains θ. The probability a with which the interval *(L,U)* fails to cover q is sometimes called the *error rate* of the interval. The quantity $(1 - \alpha) \times 100\%$ is called the *confidence level* of the confidence interval. Common values of α are $\alpha = .01, .05, .1$, which yield 99%, 95%, and 90% confidence intervals, respectively.

HYPOTHESIS TESTS. A hypothesis test is a probability-based method for making a decision concerning the value of a population parameter θ (for example, the population mean μ or standard deviation σ in a one-sample problem), or the relative values of two population parameters θ_1 and θ_2 (for example, the difference between the population means $\mu_1 - \mu_2$ in a two-sample problem). You begin by forming a *null hypothesis* and an *alternative hypothesis*. For example, in the two-sample problem your null hypothesis is often the hypothesis that $\theta_1 = \theta_2$, and your alternative hypothesis is one of the following:

- The *two-sided* alternative: $\theta_1 \neq \theta_2$
- The *greater-than* alternative: $\theta_1 > \theta_2$
- The *less-than* alternative: $\theta_1 < \theta_2$

Your decision to *accept* the null hypothesis, or to *reject* the null hypothesis in favor of your alternative hypothesis is based on the observed value $T = t_{obs}$ of a suitably chosen test statistic *T*. The probability that the statistic *T* "exceeds" the observed value t_{obs} when your null hypothesis is in fact true, is called the *p-value*.

For example, suppose you are testing the null hypothesis that $\theta = \theta_0$ against the alternative hypothesis that $\theta \neq \theta_0$ in a one-sample problem. The *p*-value is the probability that the absolute value of *T* exceeds the absolute value of t_{obs} for your data, when the null hypothesis is true.

In *formal* hypothesis testing, you proceed by choosing a "good" statistic *T* and specifying a *level of significance*, which is the probability of rejecting a null hypothesis when the null hypothesis is in fact true.

In terms of formal hypothesis testing, your *p*-value has the following interpretation: the *p*-value is the level of significance for which your observed test statistic value t_{obs} lies on the boundary between acceptance and rejection of the null hypothesis. At any significance level greater than the *p*-value, you reject the null hypothesis, and at any significance level less than the *p*-value you accept the null hypothesis. For example, if your *p*-value is .03, you reject the null hypothesis at a significance level of .05, and accept the null hypothesis at a significance level of .01.

Robust and Nonparametric Methods

Two problems frequently complicate your statistical analysis. For example, Student's *t*-test, which is the basis for most statistical inference on the mean-value locations of normal distributions, relies on two critical assumptions:

1. The observations have a common normal (or Gaussian) distribution with mean μ and variance σ^2.
2. The observations are independent.

However, one or both of these assumptions often fail to hold in practice.

For example, if the actual distribution for the observations is an outlier-generating, heavy-tailed deviation from an assumed Gaussian distribution, the confidence level remains quite close to $(1 - \alpha)100\%$, but the average confidence interval length is considerably larger than under normality. The *p*-values based on the Student's *t* test are also heavily influenced by outliers.

In this example, and more generally, you would like to have statistical methods with the property that the conclusions you draw are not much affected if the distribution for the data deviates somewhat from the assumed model; for example, if the assumed model is a normal, or Gaussian distribution, and the actual model for the data is a nearly normal distribution. Such methods are called *robust*. In this chapter you will learn how to use an S-PLUS function to obtain robust point estimates and robust confidence intervals for the population correlation coefficient.

For one and two-sample location parameter problems (among others), there exist *strongly* robust alternatives to classical methods, in the form of *nonparametric* statistics. The term "nonparametric" means that the

methods work even when the actual distribution for the data is far from normal; that is, when the data do not have to have even a nearly normal distribution. In this chapter, you will learn to use one of the best of the nonparametric methods for constructing a hypothesis test *p*-value, namely the Wilcoxon rank method, as implemented in the S-PLUS function `wilcox.test`.

It is important to keep in mind that serial correlation in the data can quickly invalidate the use of both classical methods (such as Student's *t*) and nonparametric methods (such as the Wilcoxon rank method) for computing confidence intervals and *p*-values. For example, a 95% Student's *t* confidence interval can have a much higher error rate than 5% when there is a small amount of positive correlation in the data. Also, most modern robust methods are oriented toward obtaining insensitivity toward outliers generated by heavy-tailed nearly normal distributions, and are not designed to cope with serial correlation. For information on how to construct confidence intervals for the population mean when your data are serially correlated and free of outliers, see Heidelberger and Welch (1981).

ONE SAMPLE: DISTRIBUTION SHAPE, LOCATION, AND SCALE

In 1876, the French physicist Cornu reported a value of 299,990 km/sec for *c*, the speed of light. In 1879, the American physicist A.A. Michelson carried out several experiments to verify and improve on Cornu's value.

Michelson obtained the following 20 measurements of the speed of light:

```
 850  740  900  1070  930  850  950  980  980  880
1000  980  930   650  760  810 1000 1000  960  960
```

To obtain Michelson's actual measurements in km/sec, add 299,000 km/sec to each of the above values.

The twenty observations can be thought of as observed values of twenty random variables with a common but unknown mean-value location μ. If the experimental setup for measuring the speed of light is free of bias, then it is reasonable to assume that μ is the true speed of light.

In evaluating this data, we seek answers to at least five questions:

1. What is the speed of light μ?
2. Has the speed of light changed relative to our best previous value μ_0?
3. What is the uncertainty associated with our answers to (1) and (2)?
4. What is the shape of the distribution of the data?
5. The measurements were taken over time. Is there any evidence of serial correlation?

The first three questions were probably in Michelson's mind when he gathered his data. The last two must be answered to determine which techniques can be used to obtain valid statistical inferences from the data. For example, if the shape of the distribution indicates a nearly normal distribution without outliers, we can use the Student's *t* tests in attempting to answer question (2). If the data contain outliers or are far from normal, we should use a robust method or a nonparametric

method such as the Wilcoxon signed-rank test. On the other hand, if serial correlation exists, neither the Student's *t* nor the Wilcoxon test offers valid conclusions.

In this section, we use S-PLUS to carefully analyze the Michelson data. Identical techniques can be used to explore and analyze any set of one-sample data.

Setting Up the Data

The data form a single, ordered set of observations, so they are appropriately described in S-PLUS as a vector. Use the `scan` function to create the vector `mich`:

```
> mich <- scan()

1: 850 740 900 1070 930
6: 850 950 980 980 880
11: 1000 980 930 650 760
16: 810 1000 1000 960 960
21:
```

Exploratory Data Analysis

To start, we can evaluate the shape of the distribution, by making a set of four EDA plots, using the `eda.shape` function described in the section Exploratory Data Analysis on page 57:

```
> eda.shape(mich)
```

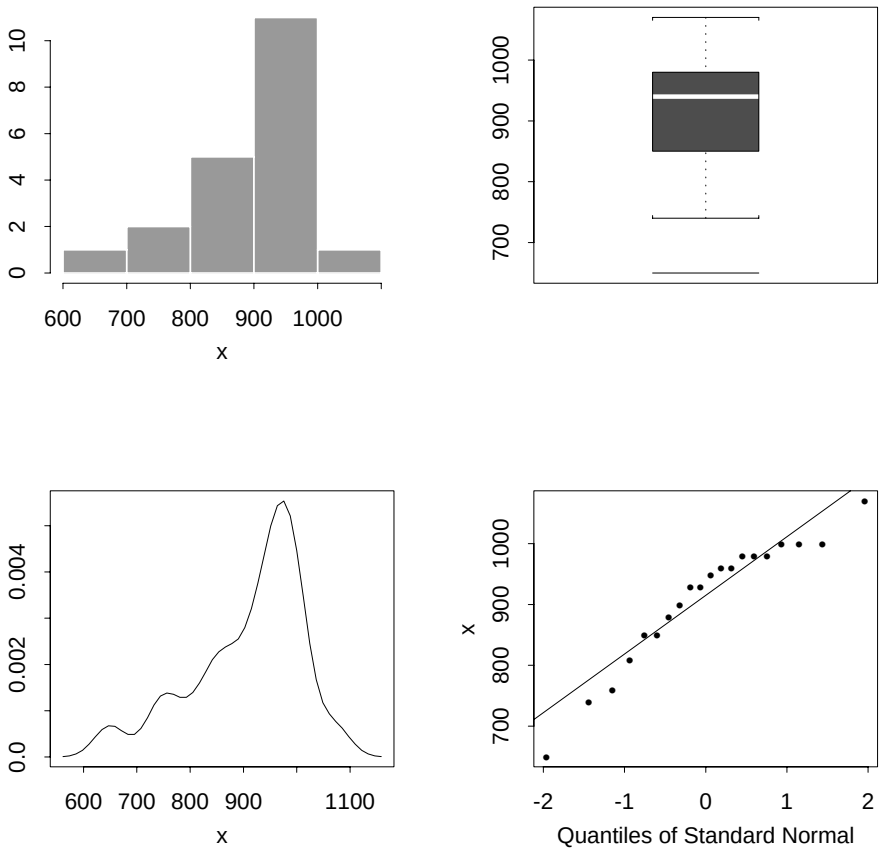

Figure 3.2: *Exploratory data analysis plots.*

The plots in Figure 3.2 reveal a distinctly skewed distribution, skewed toward the left (that is, toward smaller values), but rather normal in the middle region. The distribution is thus not normal, and probably not even "nearly" normal.

The solid horizontal line in the box plot is located at the *median* of the data, and the upper and lower ends of the box are located at the *upper quartile* and *lower quartile* of the data, respectively. To get precise values for the median and quartiles, use the `summary` function:

```
> summary(mich)

 Min. 1st Qu. Median Mean 3rd Qu. Max.
  650     850    940  909     980 1070
```

The summary shows, from left to right, the smallest observation, the first quartile, the median, the mean, the third quartile, and the largest observation. From this summary you can compute the interquartile range, $IQR = 3Q - 1Q$. The interquartile range provides a useful criterion for identifying outliers–any observation which is more than 1.5 x IQR above the third quartile or below the first quartile is a suspected outlier.

To examine possible serial correlation, or dependency, make two plots using the `eda.ts` function defined in the section Exploratory Data Analysis on page 57.

```
> eda.ts(mich)
```

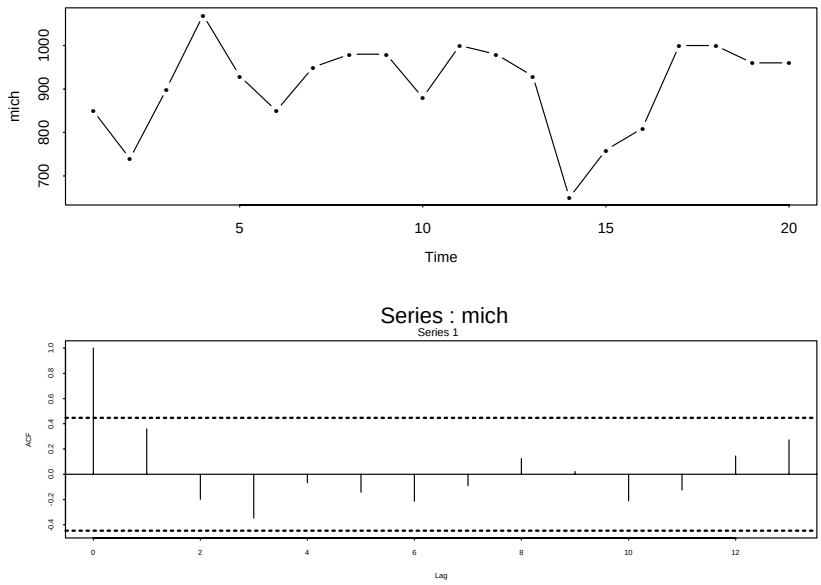

Figure 3.3: *Time series plots.*

The top plot in Figure 3.3 reveals a somewhat unusual excursion at observations 14, 15, 16, and perhaps a slightly unusual oscillation in the first 6 observations. However, the autocorrelation function plot in the lower part of Figure 3.3 reveals no significant serial correlations–all values lie within the horizontal dashed lines for lags greater than 0.

Statistical Inference

Because the Michelson data are not normal, you should probably use the Wilcoxon signed-rank test rather than the Student's *t*-test for your statistical inference. For illustrative purposes, we'll use both.

To compute Student's *t* confidence intervals for the population mean-value location parameter μ, and to compute Student's *t* significance test *p*-values for the parameter μ_0, use the function `t.test`.

To perform the test, you specify the confidence level, the hypothesized mean-value location μ, and the hypothesis being tested, as follows:

- `conf.level=` specifies the confidence level of the confidence interval. Usual values are `0.90`, `0.95`, or `0.99`. The default is `0.95`.
- `mu=` specifies the null hypothesis value μ_0 of μ. The default is $\mu_0 = 0$, which is often inappropriate for one-sample problems. You should choose μ carefully, using either a previously accepted value or a value suggested by the data before sampling.
- `alternative=` specifies the specific hypothesis being tested. There are three options:
- `"two.sided"` tests the hypothesis that the true mean is not equal to μ_0. This is the default alternative.
- `"greater"` tests the hypothesis that the true mean is greater than μ_0.
- `"less"` tests the hypothesis that the true mean is less than μ_0.

For Michelson's data, suppose you want to test the null hypothesis value $\mu_0 = 990$ (plus 299,000) against a two-sided alternative. Then you use `t.test` with the argument `mu=990`:

```
> t.test(mich,mu=990)

        One-sample t-Test

data:  mich
t = -3.4524, df = 19, p-value = 0.0027
alternative hypothesis: true mean is not equal to 990
```

```
95 percent confidence interval:
 859.8931 958.1069
sample estimates:
 mean of x
       909
```

The *p*-value is 0.0027, which is highly significant. S-PLUS returns other useful information besides the *p*-value, including the *t*-statistic value, the degrees of freedom (`df`), the sample mean, and the confidence interval.

Our example used the default confidence level of .95. If you specify a different confidence level, as in the following command:

```
> t.test(mich,conf.level=.90,mu=990)
```

you obtain a new confidence interval of (868,950), which is shorter than before, but nothing else changes in the output from `t.test`.

Wilcoxon Signed Rank Test p-Values

To perform the Wilcoxon signed rank nonparametric test, use the function `wilcox.test`. As with `t.test`, the test is completely determined by the confidence level, the hypothesized mean μ_0, and the hypothesis to be tested. These options are specified for `wilcox.test` exactly as for `t.test`.

For example, to test the hypothesis that $\mu = 990$ (plus 299,000), use `wilcox.test` as follows:

```
> wilcox.test(mich,mu=990)

Wilcoxon signed-rank test

data:  mich
signed-rank normal statistic with correction Z = -3.0715,
p-value = 0.0021
alternative hypothesis: true mu is not equal to 990
Warning messages:
  cannot compute exact p-value with ties in:
  wil.sign.rank(dff, alternative, exact, correct)
```

The *p*-value of .0021 compares with the *t*-test *p*-value of .0027 for testing the same null hypothesis with a two-sided alternative.

Michelson's data have several tied values. Because exact *p*-values cannot be computed if there are tied values (or if the null hypothesis mean is equal to one of the data values), a normal approximation is used and the associated *Z*-statistic value is reported.

TWO SAMPLES: DISTRIBUTION SHAPES, LOCATIONS, AND SCALES

Suppose you are a nutritionist interested in the relative merits of two diets, one featuring high protein, the other low protein. Do the two diets lead to differences in mean weight gain? Consider the data in Table 3.1, which shows the weight gains (in grams) for two lots of female rats, under the two diets. The first lot, consisting of 12 rats, was given the high protein diet, and the second lot, consisting of 7 rats, was given the low protein diet. These data appear in section 6.9 of Snedecor and Cochran (1980).

Table 3.1: *Weight gain data.*

High Protein	Low Protein
134	70
146	118
104	101
119	85
124	107
161	132
107	94
83	

Table 3.1: *Weight gain data. (Continued)*

High Protein	Low Protein
113	
129	
97	
123	

The high protein and low protein samples are presumed to have mean-value location parameters μ_H and μ_L, and standard deviation scale parameters σ_H and σ_L, respectively. While you are primarily interested in whether there is any difference in the μ's, you may also be interested in whether or not the two diets result in different variabilities, as measured by the standard deviations (or their squared values, the variances). This section shows you how to use S-PLUS functions to answer such questions.

Setting Up the Data

In the two-sample case, each sample forms a set of data. Thus, you begin by creating two data vectors, say `gain.high` and `gain.low`, containing, respectively, the first and second columns of data from Table 3.1:

```
> gain.high <- scan()

1: 134 146 104 119 124 161 107 83 113 129 97 123
13:

> gain.low <- scan()

1: 70 118 101 85 107 132 94
8:
```

Exploratory Data Analysis

For each sample, make a set of EDA plots, consisting of a histogram, a boxplot, a density plot and a normal qq-plot, all displayed in a two-by-two plot layout, using the `eda.shape` function defined in the section Exploratory Data Analysis on page 57:

```
> eda.shape(gain.high)
> eda.shape(gain.low)
```

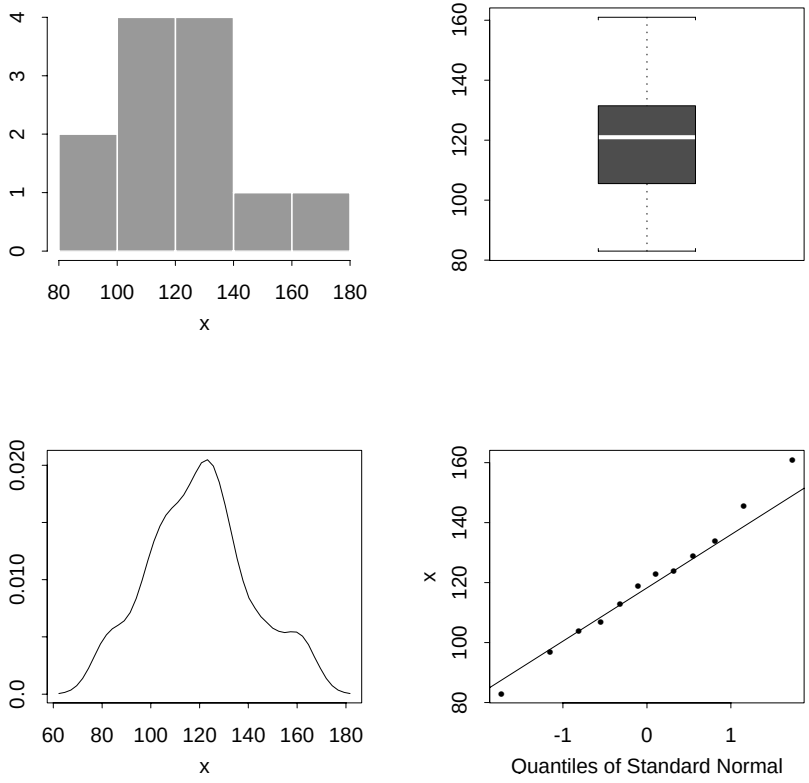

Figure 3.4: *EDA plots for high-protein group.*

The resulting plots for the high-protein group are shown in Figure 3.4. They indicate that the data come from a nearly normal distribution, and there is no indication of outliers. The plots for the low-protein group, which we do not show, support the same conclusions.

Since the data were not collected in any specific time order, you need not make any exploratory time series plots to check for serial correlation.

Statistical Inference

Is the mean weight gain the same for the two groups of rats? Specifically, does the high-protein group show a higher average weight gain? From our exploratory data analysis, we have good reason to believe that Student's *t*-test will provide a valid test of our

hypotheses. As in the one-sample case, you can get confidence intervals and hypothesis test *p*-values for the difference $\mu_1 - \mu_2$ between the two mean-value location parameters μ_1 and μ_2 using the functions `t.test` and `wilcox.test`.

As before, each test is specified by a confidence level, a hypothesized μ_0 (which now refers to the *difference* of the two sample means), and the hypothesis to be tested. However, because of the possibility that the two samples may be from different distributions, you may also specify whether the two samples have equal variances.

You define the test to be performed using the following arguments to `t.test`:

- `conf.level=` specifies the confidence level of the confidence interval. Usual values are `0.90`, `0.95`, or `0.99`. The default is `0.95`.
- `mu=` specifies the null hypothesis value μ_0 of $\mu_{diff} = \mu_H - \mu_L$. The default is $\mu_0 = 0$.
- `alternative=` specifies the hypothesis being tested. There are three options:
- `"two.sided"` tests the hypothesis that the difference of means is not equal to μ_0. This is the default alternative.
- `"greater"` tests the hypothesis that the difference of means is greater than μ_0.
- `"less"` tests the hypothesis that the difference of means is less than μ_0.
- `var.equal=` specifies whether equal variances are assumed for the two samples. The default is `var.equal=TRUE`.

To determine the correct setting for the option `var.equal`, you can either use informal inspection of the EDA boxplots or use the function `var.test` for a more formal test. If the heights of the boxes in the two boxplots are approximately the same, then so are the variances of the two outlier-free samples. The `var.test` function performs the *F* test for variance equality on the vectors representing the two samples. For the weight gain data:

```
> var.test(gain.high,gain.low)
```

```
          F test for variance equality
data:  gain.high and gain.low
F = 1.0755, num df = 11, denom df = 6, p-value = 0.9788
alternative hypothesis: true ratio of variances is not
 equal to 1
95 percent confidence interval:
 0.1988088 4.1737320
sample estimates:
 variance of x variance of y
      457.4545      425.3333
```

The evidence supports the assumption that the variances are the same, so `var.equal=T` is a valid choice.

We are interested in two alternative hypotheses: the two-sided alternative that $\mu_H - \mu_L = 0$ and the one-sided alternative that $\mu_H - \mu_L > 0$. To test these, we run the standard two-sample *t*-test twice, once with the default two-sided alternative and a second time with the one-sided alternative `alt="g"`.

You get both a confidence interval for $\mu_H - \mu_L$, and a two-sided test of the null hypothesis that $\mu_H - \mu_L = 0$, by the following simple use of `t.test`:

```
> t.test(gain.high,gain.low)

          Standard Two-Sample t-Test
data: gain.high and gain.low
t = 1.8914, df = 17, p-value = 0.0757
alternative hypothesis: true difference in means is
 not equal to 0
95 percent confidence interval:
  -2.198905 40.198905
sample estimates:
 mean of x mean of y
       120       101
```

The *p*-value is .0757, so the null hypothesis is rejected at the .10 level, but not at the .05 level. The confidence interval is (-2.2, 40.2).

To test the one-sided alternative that $\mu_H - \mu_L > 0$, use `t.test` again with the argument `alternative="greater"` (abbreviated below for ease of typing):

```
t.test(gain.high,gain.low,alt="g")

         Standard Two-Sample t-Test

data:  gain.high and gain.low
t = 1.8914, df = 17, p-value = 0.0379
alternative hypothesis: true difference in means
 is greater than 0
95 percent confidence interval:
 1.521055       NA
sample estimates:
mean of x mean of y
      120       101
```

In this case, the *p*-value is just half of the *p*-value for the two-sided alternative. This relationship between the *p*-values of the one-sided and two-sided alternatives holds in general. You also see that when you use the `alt="g"` argument, you get a lower confidence bound. This is the natural one-sided confidence interval corresponding to the "greater than" alternative.

Hypothesis Test p-Values Using wilcox.test

To get a two-sided hypothesis test *p*-value for the "two-sided" alternative, based on the Wilcoxon rank sum test statistic, use `wilcox.test`, which takes the same arguments as `t.test`:

```
> wilcox.test(gain.high,gain.low)

             Wilcoxon rank-sum test

data: gain.high and gain.low
rank-sum normal statistic with correction Z = 1.691,
 p-value = 0.0908
alternative hypothesis: true mu is not equal to 0

Warning messages:
  cannot compute exact p-value with ties
```

The above p-value of .0908, based on the normal approximation (used because of ties in the data), is rather close to the t-statistic p-value of .0757.

TWO PAIRED SAMPLES

Often two samples of data are collected in the context of a *comparative* study. A comparative study is designed to determine the *difference* between effects, rather than the individual effects. For example, consider the data in Table 3.2, which gives values of wear for two kinds of shoe sole material, A and B, along with the differences in values.

Table 3.2: *Comparing shoe sole material*

boy	wear.A	wear.B	wear.A-wear.B
1	14.0(R)	13.2(L)	0.8
2	8.8(R)	8.2(L)	0.6
3	11.2(L)	10.9(R)	0.3
4	14,2(R)	14.3(L)	-0.1
5	11.8(L)	10.7(R)	1.1
6	6.4(R)	6.6(L)	-0.2
7	9.8(R)	9.5(L)	0.3
8	11.3(R)	10.8(L)	0.5
9	9.3(L)	8.8(R)	0.5
10	13.6(R)	13.3(L)	0.3

In the table, (L) indicates the material was used on the left sole; (R), that it was used on the right sole.

The experiment leading to this data, described in Box, Hunter, and Hunter (1978), was carried out by taking 10 pairs of shoes and putting a sole of material A on one shoe and a sole of material B on the other

shoe in each pair. Which material type went on each shoe was determined by randomizing, with equal probability that material A was on the right shoe or left shoe. A group of 10 boys then wore the shoes for a period of time, after which the amount of wear was measured. The problem is to determine whether shoe material A or B is longer wearing.

You could treat this problem as a two-sample location problem and use either `t.test` or `wilcox.test`, as described in the section Two Samples: Distribution Shapes, Locations, and Scales on page 70, to test for a difference in the means of wear for material A and material B. However, you will not be very successful with this approach because there is considerable variability in wear of *both* materials types A and B from individual to individual, and this variability tends to mask the difference in wear of material A and B when you use an ordinary two-sample test.

However, the above experiment uses *paired* comparisons. Each boy wears one shoe with material A and one shoe with material B. In general, *pairing* involves selecting similar individuals or things. One often uses *self-pairing* as in the above experiment, in which two procedures, often called *treatments*, are applied to the same individual (either simultaneously or at two closely spaced time intervals) or to similar material. The goal of pairing is to make a comparison more sensitive by measuring experimental outcome differences on each pair, and combining the differences to form a statistical test or confidence interval. When you have paired data, you use `t.test` and `wilcox.test` with the optional argument `paired = T`.

The use of paired versions of `t.test` and `wilcox.test` leads to improved sensitivity over the usual versions when the variability of differences is smaller than the variability of each sample; for example, when the variability of differences of material wear between materials A and B is smaller than the variability in wear of material A and material B.

Setting Up the Data

In paired comparisons you start with two samples of data, just as in the case of ordinary two-sample comparisons. You begin by creating two data vectors, `wear.A` and `wear.B`, containing the first and second columns of Table 3.2:

```
> wear.A <- scan()
1: 14.0 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6
10:
> wear.B <- scan()
1: 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3
10:
```

Exploratory Data Analysis

You can carry out exploratory data analysis on each of the two paired samples $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, as for an ordinary two-sample problem, as described in the section Exploratory Data Analysis on page 71. However, since your analysis is based on differences, it is appropriate to carry out EDA based on a single sample of differences $d_i = x_i - y_i$, $i = 1, \ldots, n$.

In the shoe material wear experiment, you use `eda.shape` on the difference `wear.A-wear.B`:

```
> eda.shape(wear.A - wear.B)
```

The results are displayed in Figure 3.5. The histogram and density indicate some deviation from normality that is difficult to judge because of the small sample size.

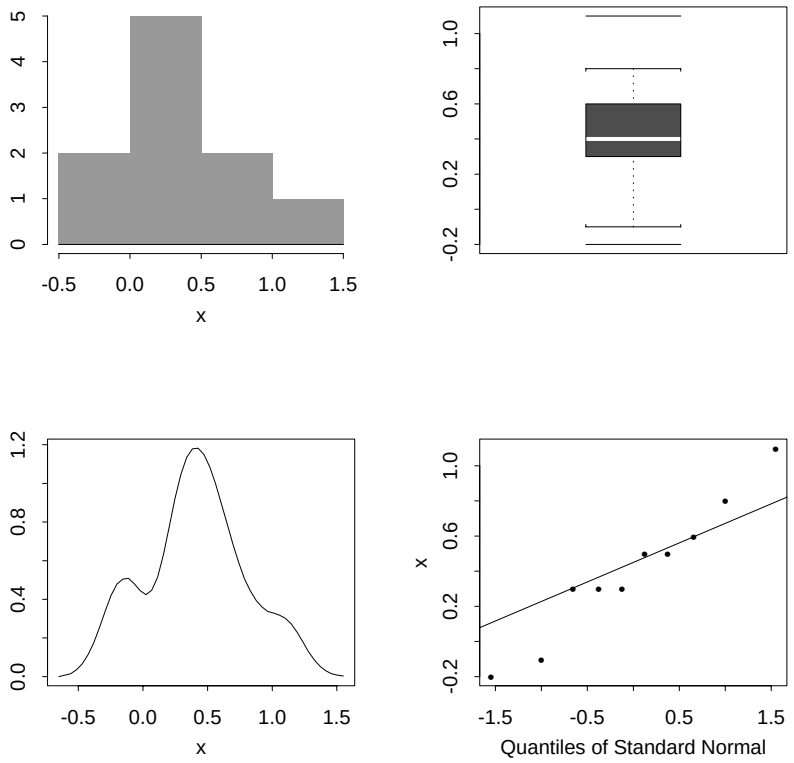

Figure 3.5: *EDA plots for differences in shoe sole material wear.*

You might also want to make a scatter plot of `wear.B` versus `wear.A`, using `plot(wear.A,wear.B)`, as a visual check on correlation between the two variables. Strong correlation is an indication that within-sample variability is considerably larger than the difference in means, and hence that the use of pairing will lead to greater test sensitivity. To obtain the scatter plot of Figure 3.6, use the following S-PLUS expression:

```
> plot(wear.A,wear.B)
```

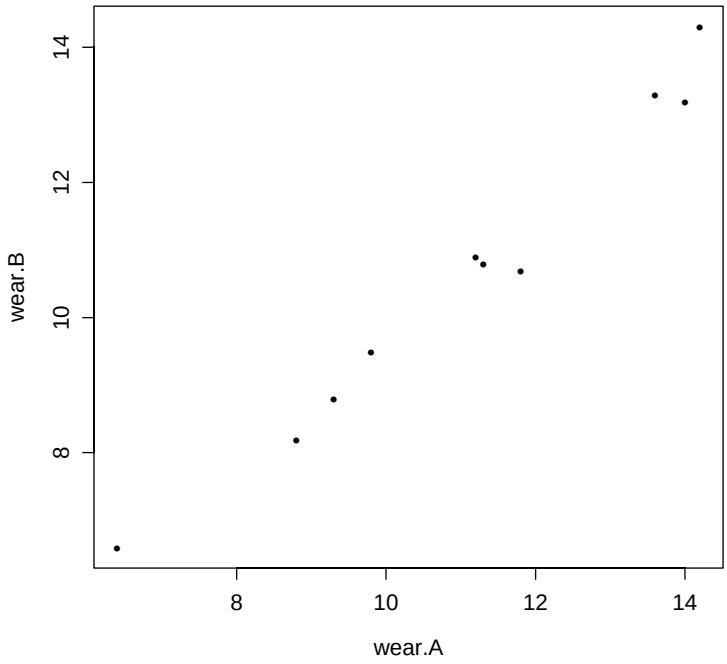

Figure 3.6: *Scatter plot of* `wear.A` *versus* `wear.B`.

Statistical Inference

To perform a paired *t*-test on the shoe material wear data, with the default two-sided alternative use `t.test` with the `paired` argument, as follows:

```
> t.test(wear.A,wear.B,paired=T)

        Paired t-Test

data: wear.A and wear.B
t = 3.3489, df = 9, p-value = 0.0085
alternative hypothesis: true mean of differences is not
 equal to 0
95 percent confidence interval:
 0.1330461 0.6869539
sample estimates:
 mean of x - y
         0.41
```

The *p*-value of .0085 is highly significant for testing the difference in mean wear of materials A and B. You also get the 95% confidence interval (.13,.67) for the difference in mean values. You can control the type of alternative hypothesis with the `alt=` optional argument, and you can control the confidence level with the `conf.level=` optional argument, as usual. To perform a paired Wilcoxon test (often called the *Wilcoxon signed rank test*) on the shoe material data, with the default two-sided alternative use `wilcox.test` with the `paired` argument, as follows:

```
> wilcox.test(wear.A,wear.B,paired=T)

Wilcoxon signed-rank test

data:  wear.A and wear.B
signed-rank normal statistic with correction Z = 2.4495,
 p-value = 0.0143
alternative hypothesis: true mu is not equal to 0

Warning messages:
  cannot compute exact p-value with ties in:
  wil.sign.rank(dff, alternative, exact, correct)
```

The *p*-value of .0143 is highly significant for testing the null hypothesis of equal centers of symmetry for the distributions of `wear.A` and `wear.B`. You can control the type of alternative hypothesis by using the optional argument `alt=` as usual.

CORRELATION

What effect, if any, do housing starts have on the demand for residential telephone service? If there is some useful association, or *correlation*, between the two, you may be able to use housing start data as a predictor of growth in demand for residential phone lines. Consider the data displayed in Table 3.3 (in coded form), which relates to residence telephones in one area of New York City.

Table 3.3: *The phone increase data.*

Diff. HS	Phone Increase
.06	1.135
.13	1.075
.14	1.496
-.07	1.611
-.05	1.654
-.31	1.573
.12	1.689
.23	1.850
-.05	1.587
-.03	1.493

Table 3.3: *The phone increase data. (Continued)*

Diff. HS	Phone Increase
.62	2.049
.29	1.942
-.32	1.482
.71	1.382

The first column of data, labeled “Diff. HS", shows annual first differences in new housing starts over a period of fourteen years. The first differences are calculated as the number of new housing starts in a given year, minus the number of new housing starts in the previous year. The second column of data, labeled “Phone Increase,” shows the annual increase in the number of “main" residence telephone services (excluding extensions), for the same fourteen-year period.

The general setup for analyzing the association between two samples of data such as those above is as follows. You have two samples of observations, of equal sizes n, of the random variables $X_1, X_2, \ldots, X_n$ and $Y_1, Y_2, \ldots, Y_n$. Let’s assume that each of the two-dimensional vector random variables (X_i, Y_i), $i = 1, 2, \ldots, n$, have the same joint distribution.

The most important, and commonly used measure of association between two such random variables is the (population) *correlation coefficient* parameter ρ, defined as

$$\rho = \frac{E(x - \mu_1)(Y - \mu_2)}{\sigma_1 \sigma_2},$$

where μ_1, μ_2 and σ_1, σ_2 are the means and standard deviations, respectively, of the random variables X and Y. The E appearing in the numerator denotes the statistical *expected value*, or *expectation* operator, and the quantity $E(X - \mu_1)(Y - \mu_2)$ is the *covariance* between the random variables X and Y. The value of ρ is always between 1 and -1.

Your main goal is to use the two samples of observed data to determine the value of the correlation coefficient ρ. In the process you want to do sufficient graphical EDA to feel confident that your determination of ρ is reliable.

Setting Up the Data

The data form two distinct data sets, so we create two vectors with the suggestive names `diff.hs` and `phone.gain`:

```
> diff.hs <- scan()

1: .06 .13 .14 -.07 -.05 -.31 .12
8: .23 -.05 -.03 .62 .29 -.32 -.71
15:

> phone.gain <- scan()

1: 1.135 1.075 1.496 1.611 1.654 1.573 1.689
8: 1.850 1.587 1.493 2.049 1.943 1.482 1.382
15:
```

Exploratory Data Analysis

If two variables are strongly correlated, that correlation may appear in a scatter plot of one variable against the other. For example, plot `phone.gain` versus `diff.hs` using the following command:

```
> plot(diff.hs, phone.gain)
```

The results are shown in Figure 3.7. The plot reveals a strong positive correlation, except for two obvious outliers. To identify the observation numbers associated with the outliers in the scatter plot, along with that of a third suspicious point, we used `identify` as follows:

```
> identify(diff.hs, phone.gain, n=3)
```

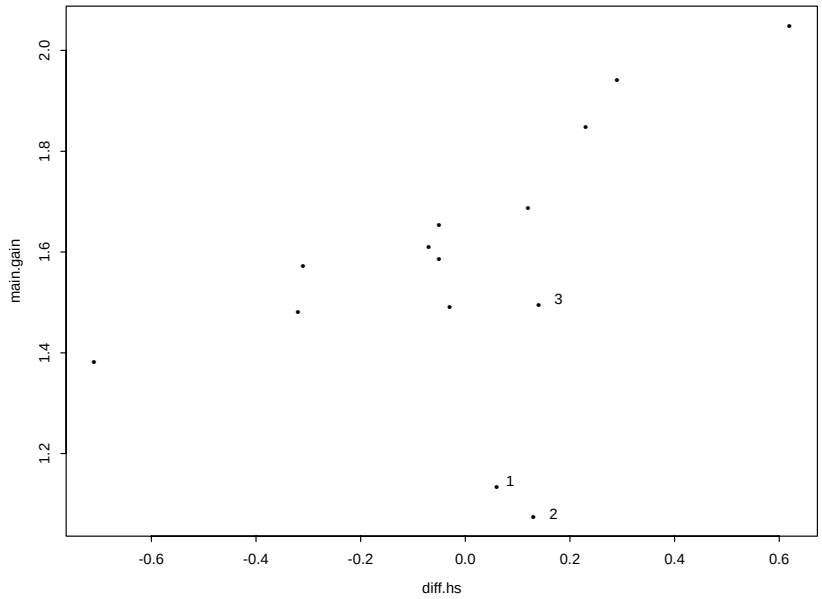

Figure 3.7: *Scatter plot of* `phone.gain` *versus* `diff.hs`.

See the on-line help for a complete discussion of `identify`.

The obvious outliers occur at the first and second observations. In addition, the suspicious point (labeled "3" in the scatter plot) occurs at the third observation time.

Since you have now identified the observation times of the outliers, you can gain further insight by making a time series plot of each series:

```
> plot(diff.hs,type="b")
> plot(phone.gain,type="b")
```

You should also make an autocorrelation plot for each series:

```
> acf(diff.hs)
> acf(phone.gain)
```

The results are shown in Figure 3.8. Except for the first three observations of the two series `phone.gain` and `diff.hs`, there is a strong similarity of shape exhibited in the two time series plots. This accounts for the strong positive correlation between the two variables

`diff.hs` and `phone.gain` shown in Figure 3.7. The dissimilar behavior of the two time series plots for the first three observations produces the two obvious outliers, and the suspicious point, in the scatter plot of `phone.gain` versus `diff.hs`.

The ACF plots show little evidence of serial correlation within each of the individual series.

Statistical Inference

From your exploratory data analysis, two types of questions present themselves for more formal analysis. If the evidence for correlation is inconclusive, you may want to test whether there is correlation between the two variables of interest by testing the null hypothesis that $\rho = 0$. On the other hand, if your EDA convinces you that correlation exists, you might prefer a point estimate $\hat{\rho}$ of the correlation coefficient ρ, or a confidence interval for ρ.

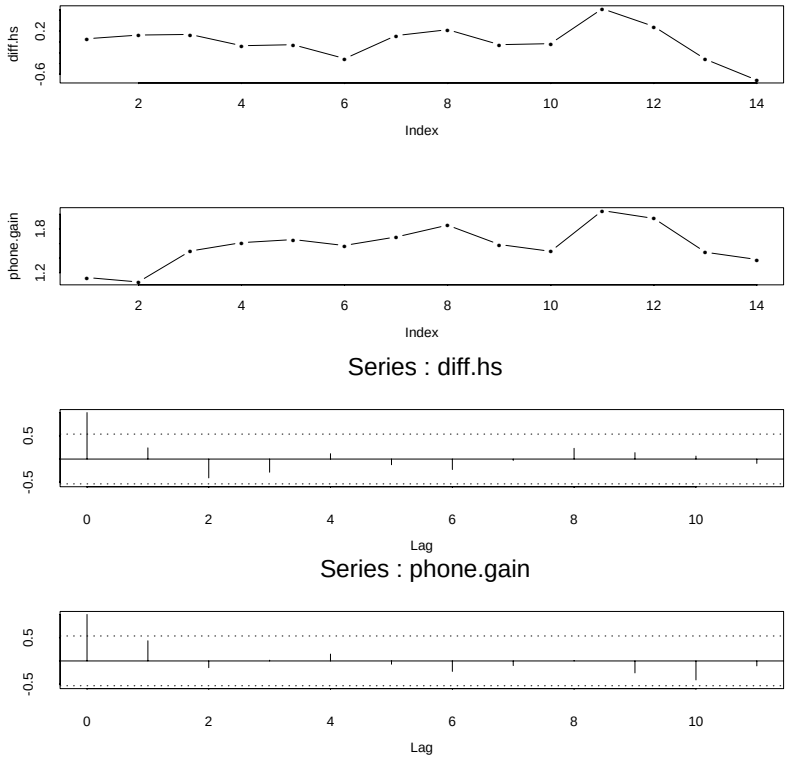

Figure 3.8: *Time series and ACF plots of phone increase data.*

Hypothesis Test p-Values

You can get *p*-values for the null hypothesis that $\rho = 0$ by using the function `cor.test`. To perform this test, you specify the alternative hypothesis to be tested and the test method to use, as follows:

- `alternative=` specifies the alternative hypothesis to be tested. There are three options:
- `"two.sided"` (the default alternative) tests the alternative hypothesis that $\rho \neq 0$.
- `"greater"` tests the alternative hypothesis that $\rho > 0$.
- `"less"` tests the alternative hypothesis that $\rho < 0$.

You can also use the abbreviated forms `alt="g"` and `alt="l"`.

- `method=` specifies which of the following methods is used:
- `"pearson"` (the default) uses the standard Pearson sample correlation coefficient.
- `"kendall"` uses the rank-based Kendall's τ measure of correlation.
- `"spearman"` uses the rank-based Spearman's ρ measure of correlation.

You can abbreviate these methods by using enough of the character string to determine a unique match; here `"p"`, `"k"`, and `"s"` work.

Because both Kendall's τ and Spearman's ρ methods are based on ranks, they are not so sensitive to outliers and nonnormality as the standard Pearson estimate.

Here is a simple use of `cor.test` to test the alternative hypothesis that there is a positive correlation in the phone gain data. We use the default choice of the classical Pearson estimate with the one-sided alternative `alt="g"`:

```
> cor.test(diff.hs,phone.gain,alt="g")

        Pearson product-moment correlation

data:  diff.hs and phone.gain
t = 1.9155, df = 12, p-value = 0.0398
alternative hypothesis: true coef is greater than 0
```

```
sample estimates:
       cor
 0.4839002
```

You get a normal theory *t*-statistic having the modest value of 1.9155, and a *p*-value of .0398. The estimate of ρ is .48, to two decimal places. There are 14 bivariate observations, and since the mean is estimated for each sample under the null hypothesis that $\rho > 0$, the number of degrees of freedom (df) is 12.

Since your EDA plots reveal two obvious bivariate outliers in the phone gain data, the nonparametric alternatives, either Kendall's τ or Spearman's ρ, are preferable in determining *p*-values for this case. Using Kendall's method, we obtain the following results:

```
> cor.test(diff.hs,phone.gain,alt="g",method="k")

         Kendall's rank correlation tau

data: diff.hs and phone.gain
normal-z = 2.0256, p-value = 0.0214
alternative hypothesis: true tau is greater than 0
sample estimates:
       tau
 0.4065934
```

The *p*-value obtained from Kendall's method is smaller than that obtained from the Pearson method. The null hypothesis is rejected at a level of 0.05. Spearman's ρ, by contrast, yields a *p*-value similar to that of the standard Pearson method.

Warning

The values returned for "tau" and "rho" (.407 and .504, respectively, for the phone gain data) do not provide unbiased estimates of the true correlation ρ. Transformations of "tau" and "rho" are required to obtain unbiased estimates of ρ.

Point Estimates and Confidence Intervals for ρ

You may want an estimate $\hat{\rho}$ of ρ, or a confidence interval for ρ. The function `cor.test` gives you the classical sample correlation coefficient estimate *r* of ρ, when you use the default Pearson's method. However, `cor.test` does not provide you with a robust

estimate of ρ, (since neither Kendall's τ or Spearman's ρ provide an *unbiased* estimate of ρ). Furthermore, `cor.test` does not provide any kind of confidence interval for ρ.

To obtain a robust point estimate of ρ, use the function `cor` with a nonzero value for the optional argument `trim=`. You should specify a fraction α between 0 and .5 for the value of this argument. This results in a robust estimate which consists of the ordinary sample correlation coefficient based on the fraction (1 - α) of the data remaining after "trimming" away a fraction α. In most cases, set `trim=.2`. If you think your data contain more than 20% outliers, you should use a larger fraction in place of `.2`. The default value is `trim=0`, which yields the standard Pearson sample correlation coefficient.

Applying `cor` to the phone gain data, you get:

```
> cor(diff.hs,phone.gain,trim=.2)

[1] 0.715215
```

Comparing this robust estimate to our earlier value for ρ obtained using `cor.test`, we see the robust estimate yields a larger estimate of ρ. This is what you expect, since the two outliers cause the standard sample correlation coefficient to have a value smaller than that of the "bulk" of the data.

To obtain a confidence interval for ρ, we'll use the following procedure (as in Snedecor and Cochran (1980)). First, transform ρ using Fisher's "*z*-transform," which consists of taking the inverse hyperbolic tangent transform $z = atanh(\rho)$. Then, construct a confidence interval for the correspondingly transformed true correlation coefficient $\bar{\rho} = \text{atanh}(\rho)$. Finally, transform this interval back to the original scale by transforming each endpoint of this interval with the hyperbolic tangent transformation *tanh*.

To implement the procedure just described as an S-PLUS function, create the function `cor.confint` as follows:

```
> cor.confint <- function(x, y, conf.level = .95, trim = 0)
+ {
+          z <- atanh(cor(x, y, trim))
+          b <- qnorm((1 - conf.level)/2)/(length(x) - 3)^.5
+          ci.z <- c(z - b, z + b)
+          conf.int <- tanh(ci.z)
+          conf.int
+ }
```

You can now use your new function `cor.confint` on the phone gain data:

```
> cor.confint(diff.hs,phone.gain)

[1]  0.80722631 -0.06280418

> cor.confint(diff.hs,phone.gain,trim=.2)

[1] 0.9028239 0.2962303
```

When you use the optional argument `trim = .2`, you are basing the confidence interval on a robust estimate of ρ, and consequently you get a robust confidence interval. Since the robust estimate has the value .72, which is larger than the standard (nonrobust) estimate value of .48, you should be reassured to get an interval which is shifted upward, and is also shorter, than the nonrobust interval you got by using `cor.confint` with the default option `trim = 0`.

REFERENCES

Bishop, Y.M.M., Fienberg, S.J., and Holland, P.W. (1980). *Discrete Multivariate Analysis: Theory and Practice*. The MIT Press, Cambridge, MA.

Box, G.E.P., Hunter, W.G., and Hunter, J.S. (1978). *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building*. John Wiley, New York.

Conover, W.J. (1980). *Practical Nonparametric Statistics, 2nd edition*. John Wiley, New York.

Heidelberger, P. and Welch, P.D. (1981). A Spectral Method for Confidence Interval Generation and Run-length Control in Simulations. *Communications of the ACM*, 24:233-245.

Hogg, R.V. and Craig, A.T. (1970). *Introduction to Mathematical Statistics*, 3rd edition. Macmillan, Toronto, Canada.

Mood, A.M., Graybill, F.A., and Boes, D.C. (1974). *Introduction to the Theory of Statistics*, 3rd edition. McGraw-Hill, New York.

Snedecor, G.W. and Cochran, W.G. (1980). *Statistical Methods*, 7th edition. Iowa State University Press, Ames, IA.

GOODNESS OF FIT TESTS

4

INTRODUCTION

Most S-PLUS functions for hypothesis testing assume a certain distributional form–often normal–and then use data to make conclusions about certain *parameters* of the distribution–often the mean or variance. In Chapter 3, Statistical Inference for One- and Two-Sample Problems, we describe EDA techniques to informally test the assumptions of these procedures. Goodness of fit (GOF) tests are another, more formal, tool to assess the evidence for assuming a certain distribution.

There are two types of GOF problems–corresponding to the number of samples–which ask the following questions:

1. *One sample.* Does the sample arise from a hypothesized distribution?
2. *Two sample.* Do two independent samples arise from the same distribution?

S-PLUS implements the two best known GOF tests:

- Chi-square, in the `chisq.gof` function.
- Kolmogorov-Smirnov, in the `ks.gof` function.

The chi-square test applies only in the one-sample case; Kolmogorov-Smirnov can be used in both the one-sample and the two-sample cases. This chapter describes both tests, together with a graphical function, `cdf.compare`, that can be used in both the one-sample and two-sample cases as an exploratory tool for evaluating goodness of fit.

CUMULATIVE DISTRIBUTION FUNCTIONS

For a random variable X, a cumulative distribution function (cdf), $F(x) = P[X \leq x]$, assigns a measure (between 0 and 1) of the probability that $X \leq x$. If $X_1, \ldots, X_n$ form a random sample from a continuous distribution, with observed values $x_1, \ldots, x_n$, an *empirical distribution function* F_n can be defined for all x, $-\infty < x < \infty$, so that $F_n(x)$ is the proportion of observed values less than or equal to x. The empirical distribution function estimates the unknown cdf. To decide whether two samples arise from the same unknown distribution, a natural procedure is to compare their empirical distribution functions. Likewise, for one sample, you can compare its empirical distribution function with a hypothesized cdf.

A graphical comparison of either one empirical distribution function with a known cdf, or of two empirical distribution functions, can be obtained easily in S-PLUS using the function `cdf.compare`.

For example, consider the plot shown in Figure 4.1. In this example, the empirical distribution function and a hypothetical cdf are quite close. This plot is produced using the `cdf.compare` function as follows:

```
> z <- rnorm(100)
> cdf.compare(z, distribution="normal")
```

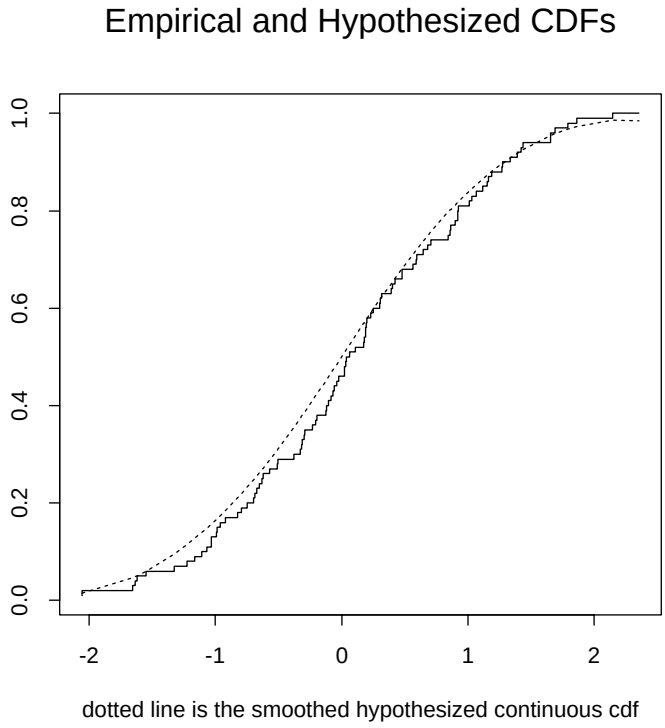

Figure 4.1: *The empirical distribution function of a sample of size 100 generated from a N(0,1) distribution. The dotted line is the smoothed theoretical N(0,1) distribution evaluated at the sample points.*

You may also compare distributions using quantile-quantile plots (qq-plots) generated by either of the following functions:

- `qqnorm` to compare one sample with a normal distribution
- `qqplot` to compare two samples

For our normal sample `z`, `qqnorm(z)` produces the plot shown in Figure 4.2.

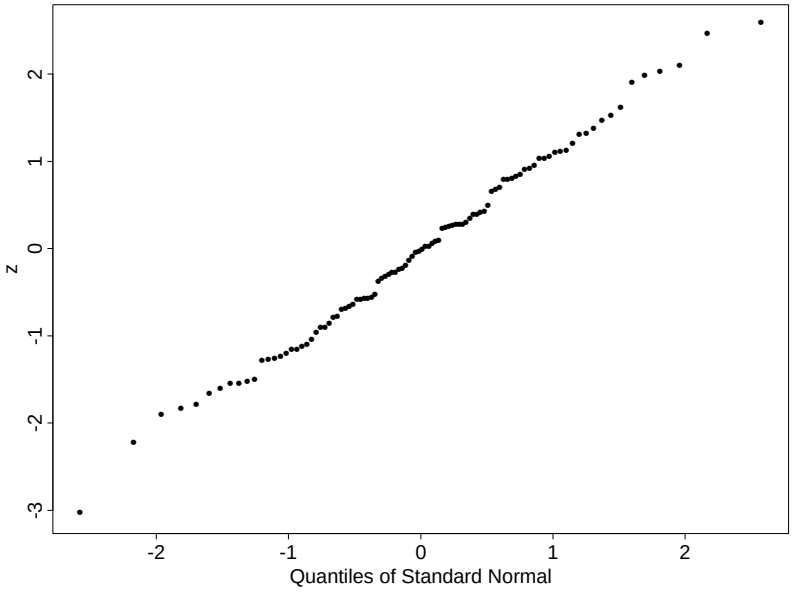

Figure 4.2: *A* `qqnorm` *plot of a sample from a normal distribution.*

Departures from linearity show how the sample differs from the normal, or how the two sample distributions differ. To compare samples with distributions other than the normal, you may produce qq-plots using the function `ppoints`. For more details, see Chapter 8, Traditional Graphics, in the *Programmer's Guide*.

In many cases, the graphical conclusions drawn from `cdf.compare` or the qq-plots make more formal tests such as the chi square or Kolmogorov-Smirnov unnecessary. For example, consider the two empirical distributions compared in Figure 4.3–they clearly have different distribution functions:

```
> x <- rnorm(30)
> y <- runif(30)
> cdf.compare(x,y)
```

THE CHI-SQUARE TEST OF GOODNESS OF FIT

The chi-square test, the oldest and best known goodness-of-fit test, is a one-sample test that examines the frequency distribution of n observations grouped into k classes. The observed counts c_i in each class are compared to the expected counts C_i from the hypothesized distribution. The test statistic, due to Pearson, is

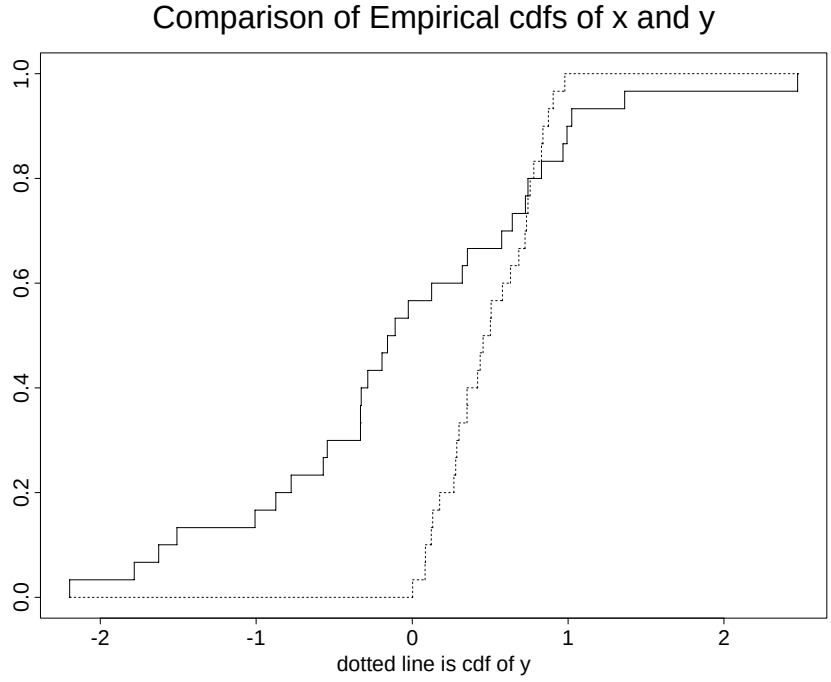

Figure 4.3: *Two clearly different empirical distribution functions.*

$$\hat{\chi}^2 = \sum_{i=1}^{k} \frac{(c_i - C_i)^2}{C_i}$$

Under the null hypothesis that the sample comes from the hypothesized distribution, it has a χ^2 distribution with k - 1 degrees of freedom. For any significance level α, reject the null hypothesis if $\hat{\chi}^2$ is greater than the critical value ν for which $P(\chi^2 > \nu) = \alpha$.

You perform the chi-square goodness of fit test with the `chisq.gof` function. In the simplest case, you specify a test vector and a hypothesized distribution:

```
> z.chisq <- chisq.gof(z, distribution="normal")
> z.chisq

         Chi-square Goodness of Fit Test

data: z

Chi-square = 8.94, df = 12,
p-value = 0.708
alternative hypothesis:
True cdf does not equal the normal Distn.
     for at least one sample point.
```

Since we created z as a random sample from a normal distribution, it is not surprising that we cannot reject the null hypothesis. If we hypothesize a different distribution, we see that the chi-square correctly rejects the hypothesis:

```
> chisq.gof(z, distribution="exponential")

         Chi-square Goodness of Fit Test

data: z

Chi-square = 324.84, df = 12,
p-value = 0
alternative hypothesis:
True cdf does not equal the exponential Distn.
     for at least one sample point.
```

The allowable values for the `distribution` argument are the following strings (the default is `"normal"`):

```
"beta"            "binomial"     "cauchy"    "chisquare"
"exponential"     "f"             "gamma"    "geometric"
"hypergeometric"  "lognormal"  "logistic"  "negbinomial"
"normal"          "poisson"          "t"      "uniform"
"weibull"         "wilcoxon"
```

When the sample being tested is from a continuous distribution, one factor affecting the outcome is the choice of partition for determining the grouping of the observations. This becomes particularly important when the expected count in one or more cells drops below 1, or the average expected cell count drops below five (Snedecor and Cochran (1980), p. 77). You can control the choice of partition using either the `n.classes` or `cut.points` argument to `chisq.gof`. By default, `chisq.gof` uses a default value for `n.classes` due to Moore (1986).

Use the `n.classes` argument to specify the number of equal-width classes:

```
> chisq.gof(z, n.classes=5)
```

Use the `cut.points` argument to specify the end points of the cells; the specified points should span the observed values:

```
> cuts.z <- c(floor(min(z))-1, -1,0,1, ceiling(max(z))+1)
> chisq.gof(z, cut.points=cuts.z)
```

Chi-square tests apply to any type of variable: continuous, discrete, or a combination of these. For large sample sizes ($n \leq 50$), if the hypothesized distribution is discrete, the chi-square is the *only* valid test. In addition, the chi-square test easily adapts to the situation when parameters of a distribution are estimated. However, especially for continuous variables, information is lost by grouping the data.

When the hypothesized distribution is continuous, the *Kolmogorov-Smirnov* test is more likely to reject the null hypothesis when it should; it is *more powerful* than the chi-square test.

THE KOLMOGOROV-SMIRNOV TEST

Suppose F_1 and F_2 are two cdfs. In the one-sample situation, F_1 is the empirical distribution function, and F_2 is a hypothesized cdf. In the two-sample situation, F_1 and F_2 are both empirical distribution functions.

Possible hypotheses and alternatives concerning these cdfs are:

- *Two-sided*:

  H_0: $F_1(x) = F_2(x)$ for all x

  H_A: $F_1(x) \neq F_2(x)$ for at least one value of x

- *One-sided ("less" alternative):*

  H_0: $F_1(x) \geq F_2(x)$ for all x

  H_A: $F_1(x) < F_2(x)$ for at least one value of x.

- *One-sided ("greater" alternative):*

  H_0: $F_1(x) \leq F_2(x)$ for all x

  H_A: $F_1(x) > F_2(x)$ for at least one value of x

The *Kolmogorov-Smirnov* test is a method for testing the above hypotheses. Corresponding to each alternative hypothesis is a Kolmogorov-Smirnov test statistic, as follows:

- *Two-sided Test:* $T = sup_x|F_1(x) - F_2(x)|$
- *Less Alternative:* $T^- = sup_x|F_2(x) - F_1(x)|$
- *Greater Alternative:* $T^+ = sup_x|F_1(x) - F_2(x)|$

If the test statistic is greater than some critical value, the null hypothesis is rejected.

To perform a KS test, use the function `ks.gof`. By default, the one-sample `ks.gof` test compares the sample `x` to a normal with mean `mean(x)` and standard deviation `sqrt(var(x))`:

```
> ks.gof(z)

One sample Kolmogorov-Smirnov Test of Composite Normality
data: z

ks = 0.0457, p-value = 0.5
alternative hypothesis:
True cdf does not equal the normal Distn.
     for at least one sample point.

sample estimates:
   mean of x standard deviation of x
 -0.04593973                1.103777
Warning messages:
  The Dallal-Wilkinson approximation, used to calculate
     the p-value in testing composite normality,
     is most accurate for p-values <= 0.10 .
   The calculated p-value is 0.881
     and so is set to 0.5 . in: dall.wilk(test, nx)
```

In the one-sample case, `ks.gof` can test any of the three alternative hypotheses (`"two-sided"`, `"greater"`, `"less"`). In the two-sample case, `ks.gof` can test only the two-sided hypothesis. The default hypothesized distribution is the normal, as for the `chisq.gof` function. You can specify a different distribution using the `distribution` argument. Allowable values for the `distribution` argument are as follows:

```
"beta"            "binomial"     "cauchy"    "chisquare"
"exponential"     "f"             "gamma"    "geometric"
"hypergeometric"  "lognormal"  "logistic"  "negbinomial"
"normal"          "poisson"          "t"      "uniform"
"weibull"         "wilcoxon"
```

ONE-SAMPLE TESTS

In a real situation, we do *not* know the true source of the data. Suppose, instead, that we think the underlying distribution is chi-square with 2 degrees of freedom. The KS test gives strong evidence against this assumption:

```
> ks.gof(z,alternative="greater",dist="chisquare",df=2)

        One-sample Kolmogorov-Smirnov Test;
                    hypothesized distribution = chisquare

data: z
ks = 0.4741, p-value = 0
alternative hypothesis: True cdf is greater than the
       chisquare Distn. for at least one sample point.
```

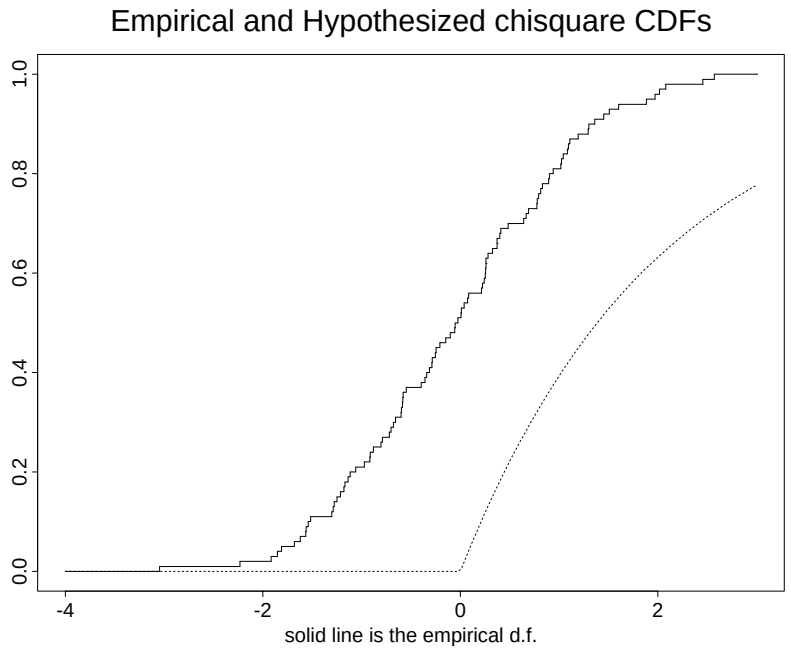

Figure 4.4: *Like the previous figure, except the dotted line shows a chi-square cdf with 2 degrees of freedom.*

Figure 4.4, created as follows, further shows that this assumption is not reasonable:

```
> cdf.compare(z, dist="chisquare", df=2)
```

The `chisq.gof` test gives further confirmation:

```
> chisq.gof(z,dist="chisquare",n.param.est=0,df=2)

         Chi-square Goodness of Fit Test
data: z
Chi-square = 282.98, df = 12, p-value = 0
alternative hypothesis: True cdf does not equal the
         chisquare Distn. for at least one sample point.
```

Note that `chisq.gof` tests only the two sided alternative.

Composite Tests for a Family of Distributions

When the parameters are *estimated* from the sample, rather than specified in advance, the tests described above are no longer adequate. Different tables of critical values are needed. In fact, for the KS test, the tables vary for different distributions, parameters estimated, methods of estimation, and sample sizes. The null hypothesis is now *composite*: rather than hypothesizing that the data have a distribution with specific parameters, we hypothesize only that the distribution belongs to the family of distributions with a certain form, such as normal. This family of distributions results from allowing all possible parameter values.

The two functions `ks.gof` and `chisq.gof` use different strategies to solve composite tests: `ks.gof` explicitly calculates the required parameters in two cases (described below), but otherwise forbids composite hypotheses, while `chisq.gof` requires the user to pass both the number of estimated parameters and the estimates themselves as arguments, then reduces the degrees of freedom for the chi-square by the number of estimated parameters.

The function `ks.gof` estimates parameters in the following two cases:

- Normal, with both mean and variance estimated.
- Exponential, with mean estimated.

As an example, we test whether we can reasonably assume that the Michelson data (see the section section One Sample: Distribution Shape, Location, and Scale on page 63) arises from a normal distribution. We start with an exploratory plot using `cdf.compare` (Figure 4.5) and then use `ks.gof` with estimated mean and variance:

```
> cdf.compare(mich, dist="normal", mean=mean(mich),
+                  sd=sqrt(var(mich)))
> ks.gof(mich,dist="normal")

One-sample Kolmogorov-Smirnov Test of Composite Normality

data: mich
ks = 0.1793, p-value = 0.0914
alternative hypothesis: True cdf does not equal
         the normal Distn. for at least one sample point.
```

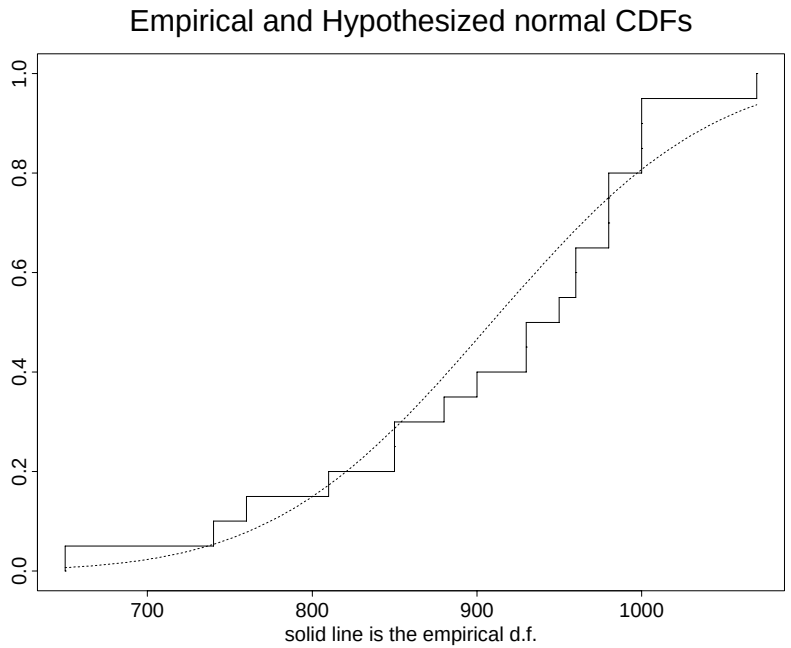

Figure 4.5: *Exploratory plot of cdf of* `mich` *data.*

For the function `chisq.gof`, if parameters are estimated, the degrees of freedom depend on the method of estimation. In practice, you may estimate the parameters from the original (that is, not grouped) data,

and then set the argument `n.param.est` to the number of parameters estimated. The function then subtracts one degree of freedom for each parameter estimated.

In truth, if the parameters are estimated by maximum likelihood, the degrees of freedom are bounded between $(m - 1)$ and $(m - 1 - k)$, where m is the number of cells, and k is the number of parameters estimated. Therefore, especially when the sample size is small, it is important to compare the test statistic to the chi-square distribution with both $(m - 1)$ and $(m - 1 - k)$ degrees of freedom. See Kendall and Stuart (1979), for a more complete discussion.

We again test the normal assumption for the Michelson data using `chisq.gof`:

```
> chisq.gof(mich, dist="normal", n.param.est=2,
+ mean=mean(mich), sd=sqrt(var(mich)))

        Chi-square Goodness of Fit Test
data: mich
Chi-square = 8.7, df = 4, p-value = 0.0691
alternative hypothesis: True cdf does not equal
      the normal Distn. for at least one sample point.

Warning messages:
Expected counts < 5. Chi-square approximation may not
    be appropriate. in: chisq.gof(mich, dist = "normal",
    n.param.est = 2, mean = mean(mich), sd = sqrt( ....
```

Both goodness-of-fit tests return results which make us suspect the null hypothesis, but don't allow us to firmly reject it at 95% or 99% confidence levels.

Note that the distribution theory of the chi-square test is a large sample theory. Therefore when any expected cell counts are small, `chisq.gof` issues a warning message. You should regard p-values with caution in this case.

TWO-SAMPLE TESTS

In the two-sample case, you can use `ks.gof` as for the one-sample case (with the second sample `y` filling in for the hypothesized distribution).

The assumptions of the two-sample KS test are:

- the samples are random samples,
- the two samples are mutually independent, and
- the data are measured on at least an ordinal scale.

In addition, the test gives exact results only if the underlying distributions are continuous.

For example, compare the cdfs of vectors `x` and `y` generated from a normal and exponential distribution, respectively:

```
> x <- rnorm(30)
> y <- rexp(100)
> qqplot(x,y)
> cdf.compare(x,y)
```

Figure 4.6 shows a qq-plot which is not linear and cdfs which are quite different.

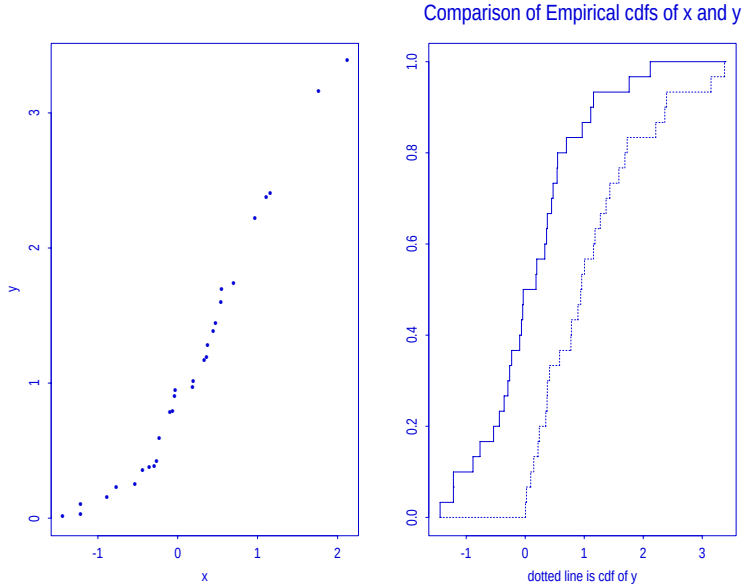

Figure 4.6: *A normal and exponential (dotted line) sample compared.*

This graphical evidence is verified by a formal KS test:

```
> ks.gof(x,y)

        Two-Sample Kolmogorov-Smirnov Test

data: x and y

ks = 0.3667, p-value = 0.0028
alternative hypothesis:
cdf of x does not equal the
             cdf of y for at least one sample point.
```

REFERENCES

Kendall, M.G. and Stuart, A. (1979). *The Advanced Theory of Statistics*, 4th edition, 2:Inference and Relationship. Oxford University Press, New York.

Moore, D.S. (1986). Tests of Chi-Squared Type. In D'Agostino, R.B. and Stevens, M.A., editors, *Goodness-of-Fit Techniques*. Marcel Dekker, New York.

Snedecor, G.W. and Cochran, W.G. (1980). *Statistical Methods*, 7th edition. Iowa State University Press, Ames, Iowa.

STATISTICAL INFERENCE FOR COUNTS AND PROPORTIONS

5

INTRODUCTION

This chapter shows you how to use S-PLUS statistical inference functions for two types of problems that involve *counts* or *proportions*. With these functions, you can obtain confidence intervals for the unknown population parameters and *p*-values for hypothesis tests of the parameter values.

The first type of problem is one where you have one or more samples, or sets of trials, in which the count for each sample represents the number of times that a certain interesting outcome occurs. By common convention, we refer to the occurrence of the outcome of interest as a "success." For example, if you play roulette 100 times at a casino, and you bet on red each time, you are interested in counting the number of times that the color red comes up. This count is a number between 0 and 100. When you divide this count by 100 you get a proportion (that is, a number between 0 and 1). This proportion is a natural estimate of the probability that red comes up on the roulette wheel.

Another example is provided by the famous Salk vaccine trials. These trials involved two groups, one of which received the vaccine and one of which received a placebo. For each group, the count of interest was the number of individuals who contracted polio. The ratio of the number of individuals who contracted polio to the total number of individuals in the group is a proportion that provides a natural estimate of the probability of contracting polio within that group.

The underlying probability model for problems of this first type is the binomial distribution. For each set of trials *i*, this distribution is characterized by the number of trials and the probability *pi* that a success occurs on each trial. This probability is called a *proportion* parameter. Your main interest is in making statistical inference statements concerning the probabilities p_1, p_2, …, p_m of occurrence of the event of interest for each of the m sets of trials.

The second type of problem is one where you have counts on the number of occurrences of several possible outcomes for each of two variables. For example, you may be studying three types of cancer and three types of diet (such as low, medium and high fat diets). The two variables of interest may be "type of cancer" and "type of diet". For a fixed set of individuals, you have counts on the number of

individuals who fall jointly in each of the categories defined by the simultaneous occurrence of a type of cancer and a diet classification. For problems of this kind, the data is arranged in a two-way table called a *contingency table.*

In this second kind of problem, your main interest is to determine whether there is any *association* between the two variables of interest. The probability model for such problems is one of *statistical independence* between the two variables.

The first three sections of this chapter cover the first type of problem described above, for which the proportion parameters are the probabilities of success, p_1, p_2, …, p_m in m sets of binomial trials. The last section covers the second type of problem, where you are interested in testing the null hypothesis of independence between two variables.

PROPORTION PARAMETER FOR ONE SAMPLE

When you play roulette and bet on red, you expect your probability of winning to be close to, but slightly less than, 0.5. You expect this because (in the United States) a roulette wheel has 18 red slots, 18 black slots, and two additional slots labeled "0" and "00", for a total of 38 slots into which the ball can fall. Thus, for a "fair" (that is, perfectly balanced) wheel, you expect the probability of red to be $p_0 = 18/38 = .474$. You hope that the house is not cheating you by altering the roulette wheel so that the probability of red is less than .474.

To test whether a given sample has a particular proportion parameter, use the `binom.test` function.

Setting Up the Data

In the roulette case there is little to do, since the only data are the number n of trials and the number x of successes. These two values can be directly supplied as arguments to `binom.test`, as shown in the examples below.

Hypothesis Testing

You can test the null hypothesis that $p = p_0$ using the function `binom.test`. For example, if you bet on red 100 times and red comes up 42 times, you get a p-value for this null hypothesis against the two-sided alternative that $p \neq .474$ as follows:

```
> binom.test(42,100,p=.474)$p.value

[1] 0.3167881
```

The two-sided alternative is the default alternative for `binom.test`. You can get a p-value for a one-sided alternative by using the optional argument `alt=`. For example, in the roulette wheel example you are concerned that the house might cheat you in some way so that $p < p_0$. Use the following to test the null hypothesis against this one-sided alternative:

```
> binom.test(42,100,p=.474,alt="l")$p.value

[1] 0.1632416
```

Here `alt="l"` specifies the "less than" alternative $p < p_0$. To specify the "greater than" alternative $p > p_0$, use `alt="g"`.

The default for the optional argument `p=`, which specifies the null hypothesis value for p, is `p=.5`. For example, suppose you toss a coin 1000 times, with heads coming up 473 times. To test the coin for "fairness;" that is, to test that the probability of heads equals .5, use the following:

```
> binom.test(473,1000)$p.value

[1] 0.09368729
```

Confidence Intervals

The function `binom.test` does not compute a confidence interval for the probability p of success. You can get a confidence interval for p by using the function `prop.test`. For example, the following shows how to obtain the 95% confidence interval for p:

```
> prop.test(45,100)$conf.int

[1] 0.3514281 0.5524574
attr(, "conf.level"):
[1] 0.95
```

The function `prop.test` uses a normal approximation to the binomial distribution for such computations.

You get different confidence intervals by using the optional argument `conf.level=` with different values. For example, to get a 90% confidence interval:

```
> prop.test(45,100,conf.level=.9)$conf.int

[1] 0.3657761 0.5370170
attr(, "conf.level"):
[1] 0.9
```

PROPORTION PARAMETERS FOR TWO SAMPLES

In the Salk vaccine trials, two large groups were involved in the placebo-control phase of the study. The first group, which received the vaccination, consisted of 200,745 individuals. The second group, which received a placebo, consisted of 201,229 individuals. There were 57 cases of polio in the first group and 142 cases of polio in the second group.

You assume a binomial model for each group, with a probability p_1 of contracting polio in the first group and a probability p_2 of contracting polio in the second group. You are mainly interested in knowing whether or not the vaccine is effective. Thus you are mainly interested in knowing the relationship between p_1 and p_2.

You can use `prop.test` to obtain hypothesis test p-values concerning the values of p_1 and p_2, and to obtain confidence intervals for the difference between the values p_1 and p_2.

Setting Up the Data

The first two arguments to prop.test are vectors containing, respectively, the number of successes and the total number of trials. For consistency with the one-sample case, we name these vectors `x` and `n`. In the case of the Salk vaccine trials, a "success" corresponds to contracting polio (although one hardly thinks of this as a literal success!) Thus, you create the vectors `x` and `n` as follows:

```
> x <- c(57,142)
> n <- c(200745,201229)
```

Hypothesis Testing

For two-group problems, you can use either of two null hypotheses: an equal probabilities null hypothesis that $p_1 = p_2$, with no restriction on the common value of these probabilities other than that they be between 0 and 1, or a completely specified probabilities null hypothesis, where you provide specific probabilities for both p_1 and p_2, and test whether the true probabilities are equal to those hypothesized.

The Equal Probabilities Null Hypothesis

When using the equal probabilities null hypothesis, a common alternative hypothesis is the two-sided alternative $p_1 \neq p_2$. These null and alternative hypotheses are the defaults for `prop.test`.

In the Salk vaccine trials, your null hypothesis is that the vaccine has no effect. For the two-sided alternative that the vaccine has some effect, either positive or negative, you get a *p*-value by extracting the `p.value` component of the list returned by `prop.test`:

```
> prop.test(x,n)$p.value

[1] 2.86606e-09
```

The extremely small *p*-value clearly indicates that the vaccine has some effect.

To test the one-sided alternative that the vaccine has a positive effect; that is, that $p_1 < p_2$, use the argument `alt="l"` to `prop.test`:

```
> prop.test(x,n,alt="l")$p.value

[1] 1.43303e-09
```

Here the *p*-value is even smaller, indicating that the vaccine is highly effective in protecting against the contraction of polio.

Completely Specified Null Hypothesis Probabilities

You can also use `prop.test` to test "completely" specified null hypothesis probabilities. For example, suppose you have some prior belief that the probabilities of contracting polio with and without the Salk vaccine are $p_{01} = .0002$ and $p_{02} = .0006$, respectively. Then you supply these null hypothesis probabilities as a vector object, using the optional argument `p=`. The *p*-value returned is for the joint probability that both probabilities are equal to the hypothesized probabilities; that is, .0002 and .0006 .

```
> prop.test(x,n,p=c(.0002,.0006))$p.value

[1] 0.005997006
```

The above *p*-value is very small and indicates that the null hypothesis that the joint probability that the underlying population probabilities with and without the Salk vaccine are .0002 and .0006 is very unlikely and should be rejected.

Confidence Intervals

You obtain a confidence interval for the difference $p_1 - p_2$ in the probabilities of success for the two samples by extracting the `conf.int` component of `prop.test`. For example, to get a 95% confidence interval for the difference in probabilities for the Salk vaccine trials:

```
> prop.test(x,n)$conf.int

[1] -0.0005641810 -0.0002792617

attr(, "conf.level"):

[1] 0.95
```

The 95% confidence level is the default confidence level for `prop.test`. You get a different confidence level by using the optional argument `conf.level=`. For example, to get a 99% confidence interval, use:

```
> prop.test(x,n,conf.level=.99)$conf.int

[1] -0.0006073705 -0.0002360723

attr(, "conf.level"):

[1] 0.99
```

You get a confidence interval for the difference $p_1 - p_2$ by using `prop.test` only when you use the default null hypothesis that $p_1 = p_2$.

You get all the information provided by `prop.test` as follows:

```
> prop.test(x,n,conf.level=.90)

         2-sample test for equality of proportions with
 continuity correction
data: x out of n
X-squared = 35.2728, df = 1, p-value = 0
alternative hypothesis: two.sided
90 percent confidence interval:
 -0.0005420769 -0.0003013659
sample estimates:
 prop'n in Group 1 prop'n in Group 2
      0.0002839423      0.0007056637
```

PROPORTION PARAMETERS FOR THREE OR MORE SAMPLES

Sometimes you may have three or more samples of subjects, with each subject characterized by the presence or absence of some characteristic. An alternative, but equivalent, terminology is that you have three or more sets of trials, with each trial resulting in a success or failure. For example, consider the data shown in Table 5.1 for four different studies of lung cancer patients, as presented by Fleiss (1981).

Table 5.1: *Smoking status among lung cancer patients in four studies.*

Study	Number of Patients	Number of Smokers
1	86	83
2	93	90
3	136	129
4	82	70

Each study has a certain number of patients, as shown in the second column of the table, and for each study a certain number of the patients were smokers, as shown in the third column of the table. For this data, you are interested in whether the probability of a patient being a smoker is the same in each of the four studies, that is, whether each of the studies involve patients from a homogeneous population.

Setting Up the Data

The first argument to `prop.test` is a vector containing the number of subjects having the characteristic of interest for each of the groups (or the number of successes for each set of trials). The second argument to `prop.test` is a vector containing the number of subjects in each group (or the number of trials for each set of trials). As in the one and two sample cases, we call these vectors `x` and `n`.

For the smokers data in Table 5.1, you create the vectors `x` and `n` as follows:

```
> x <- c(83,90,129,70)
> n <- c(86,93,136,82)
```

Hypothesis Testing

For problems with three or more groups, you can use either an equal probabilities null hypothesis or a completely specified probabilities null hypothesis.

The Equal Probabilities Null Hypothesis

In the lung cancer study, the null hypothesis is that the probability of being a smoker is the same in all groups. Because the default null hypothesis for `prop.test` is that all groups (or sets of trials) have the same probability of success, you get a *p*-value as follows:

```
> prop.test(x,n)$p.value

[1] 0.005585477
```

The *p*-value of .006 is highly significant, so you can not accept the null hypothesis that all groups have the same probability that a patient is a smoker. To see all the results returned by `prop.test`, use:

```
> prop.test(x,n)

         4-sample test for equality of proportions without
 continuity correction

data: x out of n
X-squared = 12.6004, df = 3, p-value = 0.0056
alternative hypothesis: two.sided
sample estimates:
prop'n in Group 1 prop'n in Group 2 prop'n in Group 3
        0.9651163         0.9677419         0.9485294
prop'n in Group 4
        0.8536585
```

Completely Specified Null Hypothesis Probabilities

If you want to test a completely specified set of null hypothesis probabilities, you need to supply the optional argument `p=`, with the value of this argument being a vector of probabilities having the same length as the first two arguments, `x` and `n`.

For example, in the lung cancer study, to test the null hypothesis that the first three groups have a common probability .95 of a patient being a smoker, while the fourth group has a probability .90 of a patient being a smoker, create the vector `p` as follows, the use it as an argument to `prop.test`:

```
> p <- c(.95,.95,.95,.90)
> prop.test(x,n,p)$p.value

[1] 0.5590245
Warning messages:
  Expected counts < 5. Chi-square approximation may not be
appropriate.
```

Alternatively, you could use

```
prop.test(x,n,p=c(.95,.95,.95,.90))$p.value
```

Confidence Intervals

Confidence intervals are not computed by `prop.test` when you have three or more groups (or sets of trials).

CONTINGENCY TABLES AND TESTS FOR INDEPENDENCE

The Salk vaccine trials in the early 1950s resulted in the data presented in Table 5.2.

Table 5.2: *Contingency table of Salk vaccine trials data.*

	No Polio	Non-paralytic Polio	Paralytic Polio	Totals
Vaccinated	200,688	24	33	200,745
Placebo	201,087	27	115	201,229
Totals	401,775	51	148	401,974

There are two categorical variables for the Salk trials: vaccination status, which has the two levels "vaccinated" and "placebo," and polio status, which has the three levels "no polio," "non-paralytic polio," and "paralytic polio." Of 200,745 individuals who were vaccinated, 24 contracted non-paralytic polio, 33 contracted paralytic polio, and the remaining 200,688 did not contract any kind of polio. Of 201,229 individuals who received the placebo, 27 contracted non-paralytic polio, 115 contracted paralytic polio, and the remaining 201,087 did not contract any kind of polio.

Tables such as Table 5.2 are called contingency tables. A contingency table lists the number of counts for the joint occurrence of two levels (or possible outcomes), one level for each of two categorical variables. The levels for one of the categorical variables correspond to the columns of the table, and the levels for the other categorical variable correspond to the rows of the table.

When working with contingency table data, your primary interest is most often determining whether there is any association in the form of statistical dependence between the two categorical variables whose counts are displayed in the table. The null hypothesis is that the two variables are statistically independent. You can test this null hypothesis with the functions `chisq.test` and `fisher.test`. The function `chisq.test` is based on the classic chi-square test statistic, and the associated p-value computation entails some approximations.

The function `fisher.test` computes an exact *p*-value for tables having at most 10 levels for each variable. The function `fisher.test` also entails a statistical conditioning assumption.

For contingency tables involving confounding variables, which are variables related to both variables of interest, you can test for independence using the function `mantelhaen.test`, which performs the Mantel-Haenszel test. For contingency tables involving matched pairs, use the function `mcnemar.test` to perform McNemar's chi-square test.

The functions for testing independence in contingency tables do not compute confidence intervals, only *p*-values and the associated test statistic.

The Chi-Square and Fisher Tests of Independence

The chi-square and Fisher's exact tests are familiar methods for testing independence. The Fisher test is often recommended when expected counts in any cell are below 5, as the chi-square probability computation becomes increasingly inaccurate when the expected counts in any cell are low. (S-PLUS produces a warning message in that case). Other factors may also influence your choice of which test to use, however. Refer to a statistics text for further discussion if you are unsure which test to use.

Setting Up the Data

You can set up your contingency table data in several ways. Which way you choose depends to some extent on the original form of the data and whether the data involves a large number of counts or a small to moderate number of counts.

Two-Column Matrix Objects

If you already have the data in the form of a contingency table in printed form, as in Table 5.2, the easiest thing to do is to put the data in matrix form (excluding the marginal totals, if provided in the original data). For example,

```
> salk.mat <- rbind(c(200688,24,33),c(201087,27,115))
> salk.mat

       [,1] [,2] [,3]
[1,] 200688   24   33
[2,] 201087   27  115
```

You could obtain the same result in a slightly different way as follows:

```
> salk.mat <- matrix(c(200688, 24, 33, 201087, 27, 115),
+ 2, 3, byrow=T)
```

Two Numeric Vector Objects

You may be given the raw data in the form of two equal-length coded vectors, one for each variable. In such cases, the length of the vectors corresponds to the number of individuals, with each entry indicating the level by a numeric coding. For example, suppose you have two variables from a clinical trial of the drug propranolol. (The data was reported by P.J.D. Snow in *Lancet*, (Snow 1965)). The vector `drug` is coded for control or propranolol status, and the vector `status` is coded yes or no indicating whether the patient survived at least 28 days. The raw data is as follows:

```
> drug

 [1] "control" "control" "control" "control" "prop"
 [6] "control" "prop"    "control" "prop"    "control"
[11] "prop"    "prop"    "control" "prop"    "prop"
[16] "control" "control" "prop"    "prop"    "prop"
[21] "prop"    "control" "prop"    "control" "control"
[26] "prop"    "control" "control" "control" "control"
[31] "control" "control" "prop"    "control" "prop"
[36] "control" "prop"    "prop"    "prop"    "control"
[41] "prop"    "control" "prop"    "control" "prop"
[46] "control" "prop"    "control" "control" "prop"
[51] "prop"    "prop"    "control" "prop"    "prop"
[56] "prop"    "control" "control" "control" "prop"
[61] "prop"    "control" "prop"    "control" "prop"
[66] "control" "prop"    "control" "prop"    "control"
[71] "prop"    "control" "prop"    "control" "prop"
[76] "control" "prop"    "control" "prop"    "control"
[81] "prop"    "control" "prop"    "control" "prop"
[86] "control" "prop"    "control" "control" "prop"
[91] "prop"

> status

 [1] "yes" "yes" "yes" "no"  "yes" "yes" "yes" "yes" "yes"
[10] "yes" "yes" "no"  "no"  "yes" "yes" "no"  "no"  "yes"
[19] "yes" "yes" "yes" "no"  "yes" "yes" "no"  "yes" "no"
[28] "yes" "no"  "yes" "no"  "yes" "no"  "yes" "yes" "no"
```

```
[37] "no"  "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
[46] "no"  "yes" "no"  "yes" "yes" "yes" "yes" "yes" "yes"
[55] "yes" "yes" "yes" "yes" "yes" "no"  "yes" "yes" "yes"
[64] "no"  "no"  "no"  "yes" "yes" "yes" "yes" "no"  "no"
[73] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
[82] "yes" "yes" "yes" "yes" "yes" "yes" "no"  "no"  "yes"
[91] "no"
```

To obtain the contingency table (without marginal count totals) use the function `table` with `status` and `drug` as arguments:

```
> table(status,drug)

    control prop
no       17    7
yes      29   38
```

Two Factor Objects

Your data may already be in the form of two factor objects, or you may want to put your data in that form for further analysis in S-PLUS. For example, to put `drug` and `status` into factor form, use the `factor` command as follows:

```
> drug.fac <- factor(drug)
> drug.fac

 [1] control control control control prop    control
 [7] prop    control prop    control prop    prop
[13] control prop    prop    control control prop
[19] prop    prop    prop    control prop    control
[25] control prop    control control control control
[31] control control prop    control prop    control
[37] prop    prop    prop    control prop    control
[43] prop    control prop    control prop    control
[49] control prop    prop    prop    control prop
[55] prop    prop    control control control prop
[61] prop    control prop    control prop    control
[67] prop    control prop    control prop    control
[73] prop    control prop    control prop    control
[79] prop    control prop    control prop    control
[85] prop    control prop    control control prop
[91] prop
```

```
> status.fac <- factor(status)

> status.fac

 [1] yes yes yes no  yes yes yes yes yes yes yes no
[13] no  yes yes no  no  yes yes yes yes no  yes yes
[25] no  yes no  yes no  yes no  yes no  yes yes no
[37] no  yes yes yes yes yes yes yes yes no  yes no
[49] yes yes yes yes yes yes yes yes yes yes yes no
[61] yes yes yes no  no  no  yes yes yes yes no  no
[73] yes yes yes yes yes yes yes yes yes yes yes yes
[85] yes yes yes no  no  yes no
```

Then use `drug.fac` and `status.fac` as arguments to the functions described below.

The Chi-Square Test of Independence

You use the function `chisq.test` to perform a classical chi-square test of the null hypothesis that the categorical variables of interest are independent. For example, using the matrix form of data object `salk.mat` for the Salk vaccine trials

```
> chisq.test(salk.mat)$p.value

[1] 1.369748e-10
```

which yields an exceedingly small *p*-value. This leads to rejection of the null hypothesis of no association between polio status and vaccination status.

To get all the information computed by `chisq.test`, use `chisq.test` without specifying a return component, as usual:

```
> chisq.test(salk.mat)

Pearson's chi-square test without Yates' continuity
correction

data: salk.mat
X-squared = 45.4224, df = 2, p-value = 0
```

You could also use the two factor objects, such as `drug.fac` and `status.fac` as follows:

```
> chisq.test(drug.fac,status.fac)
```

```
        Pearson's chi-square test with Yates' continuity correction

data: drug.fac and status.fac
X-squared = 4.3198, df = 1, p-value = 0.0377
```

The results are the same no matter which way you have set up the data.

Fisher's Exact Test of Independence

You can perform an exact test of indepence by using the S-PLUS function `fisher.test`. You can use any data object type that can be used with `chisq.test`. For example, using the factor object for the propranolol clinical trial:

```
> fisher.test(drug.fac,status.fac)

        Fisher's exact test

data: drug.fac and status.fac
p-value = 0.0314 alternative hypothesis: two.sided
```

When using `fisher.test` you should be aware that the *p*-value is computed conditionally on the fixed marginal counts of the contingency table you are analyzing That is, the inference does not extend to all possible tables that might be obtained by repeating the experiment and getting different marginal counts.

The Mantel-Haenszel Test of Independence

A cancer study produced the data shown in Table 5.3 and Table 5.4, as reported by Rosner (1986). In these tables, "case" refers to an individual who had cancer and "control" refers to an individual who did not have cancer. A "passive" smoker is an individual who lives with a smoker. A smoker can also be a passive smoker if that smoker lives with a spouse who also smokes.

Table 5.3: *Nonsmokers in cancer study.*

Case-Control Status	Passive Smoker	Not a Passive Smoker
case	120	111
control	80	155

Table 5.4: *Smokers in cancer study.*

Case-Control Status	Passive Smoker	Not a Passive Smoker
case	161	117
control	130	124

For each of these tables, you can use `chisq.test` or `fisher.test` to test for independence between cancer status and passive smoking status. The data is presented in separate tables because "smoking status;" that is, being a smoker or not being a smoker, could be a *confounding variable*, because both smoking status and passive smoking status are related to the outcome, cancer status, and because smoking status may be related to the smoking status of the spouse. You would like to be able to combine the information in both tables so as to produce an overall test of independence between cancer status and passive smoking status. You can do so for *two or more two-by-two* tables, by using the function `mantelhaen.test`, which performs the *Mantel-Haenszel* test.

Since the data is now associated with three categorical variables, the two main variables of interest plus a confounding variable, you can prepare your data in any one of the following forms:

- a three-dimensional array which represents the three dimensional contingency table (two-by-two tables stacked on top of one another)
- three numerical vectors representing each of the three categorical variables, two of primary interest and one a confounding variable
- three factor objects for the three categorical variables

Which form you use depends largely on the form in which the data is presented to you. For example, the data in Table 5.3 and Table 5.4 are ideal for use with a three-dimensional array:

```
> x.array <- array(c(120, 80, 111, 155, 161, 130,
+ 117, 124),c(2,2,2))
```

```
> x.array

, , 1
     [,1] [,2]
[1,]  120  111
[2,]   80  155

, , 2
     [,1] [,2]
[1,]  161  117
[2,]  130  124

> mantelhaen.test(x.array)$p.value

[1] 0.0001885083

> mantelhaen.test(x.array)

         Mantel-Haenszel chi-square test with
 continuity correction

data: x.array
Mantel-Haenszel chi-square = 13.9423, df = 1,
p-value = 2e-04
```

McNemar Test for Symmetry Using Matched Pairs

In some experiments with two categorical variables, one of the variables specifies two or more groups of individuals who receive different treatments. In such situations, matching of individuals is often carried out in order to increase the precision of statistical inference. However, when matching is carried out the observations usually are not independent. In such cases, the inference obtained from `chisq.test`, `fisher.test` and `mantelhaen.test` is not valid because these tests all assume independent observations. The function `mcnemar.test` allows you to obtain a valid inference for experiments where matching is carried out.

Consider, for example, the data in Table 5.5, as reported by Rosner (1986). In this table, each entry represents one pair. For instance, the "5" in the lower left cell means that in 5 pairs, the individual with treatment A died, while the individual that that person was paired with, who received treatment B, survived.

Table 5.5: *Matched pair data for cancer study.*

	Survive With Treatment B	**Die With Treatment B**
survive with treatment A	90	16
die with treatment A	5	510

Your interest is in the relative effectiveness of treatments A and B in treating a rare form of cancer. Each count in the table is associated with a matched pair of individuals.

A pair in the table for which one member of a matched pair survives while the other member dies is called a discordant pair. There are 16 discordant pairs in which the individual who received treatment A survives and the individual who received treatment B dies. There are 5 discordant pairs with the reverse situation in which the individual who received treatment A dies and the individual who received treatment B survives.

If both treatments are equally effective, then you expect these two types of discordant pairs to occur with "nearly" equal frequency. Put in terms of probabilities, your null hypothesis is that $p_1 = p_2$, where p_1 is the probability that the first type of discordancy occurs in a matched pair of individuals, and p_2 is the probability that the second type of discordancy occurs.

We illustrate the use of `mcnemar.test` on the above data, putting the data into the form of a matrix object:

```
> x.matched <- cbind(c(90,5),c(16,510))
> x.matched

     [,1] [,2]
[1,]   90   16
[2,]    5  510
```

```
> mcnemar.test(x.matched)$p.value

[1] 0.02909633

> mcnemar.test(x.matched)

        McNemar's chi-square test with continuity
 correction

data: x.matched
McNemar's chi-square = 4.7619, df = 1, p-value = 0.0291
```

You can use `mcnemar.test` with two numeric vector objects, or two factor objects, as the data arguments (just as with the other functions in this section). You can also use `mcnemar.test` with matched pair tables having more than two rows and more than two columns. In such cases, the null hypothesis is symmetry of the probabilities p_{ij} associated with each row and column of the table; that is, the null hypothesis is that $p_{ij} = p_{ji}$ for each combination of i and j.

REFERENCES

Bishop, Y.M.M. and Fienberg, S.J. and Holland, P.W. (1980). *Discrete Multivariate Analysis: Theory and Practice*. The MIT Press, Cambridge, MA.

Conover, W.J. (1980). *Practical Nonparametric Statistics, 2nd edition*. John Wiley, New York.

Fienberg, S.E. (1983). *The Analysis of Cross-Classified Categorical Data*, 2nd edition. The MIT Press, Cambridge, MA.

Fleiss, J.L. (1981). Statistical Methods for Rates and Proportions, 2nd edition. Wiley, New York.

Lehmann, E.L. (1975). *Nonparametrics: Statistical Methods Based on Ranks*. Holden-Day, San Francisco.

Rosner, B. (1986). Fundamentals of Biostatistics. Duxbury Press, Boston.

Snedecor, G.W. and Cochran, W.G. (1980). *Statistical Methods*, 7th edition. Iowa State University Press, Ames, Iowa.

Snow, P.J.D. (1965). Lancet.

CROSS-CLASSIFIED DATA AND CONTINGENCY TABLES

6

INTRODUCTION

Much data of interest is categorical in nature. Did patients receive treatment A, B, or C; did they survive? Do the people in a sample population smoke? Do they have high cholesterol counts? Have they had heart trouble? These data are stored in S-PLUS as *factors*, that is, as vectors where the elements indicate one of a number of *levels*. A useful way of looking at this data is to *cross-classify* it and get a *count* of the number of cases sharing a given combination of levels, and then create a multi-way contingency table (a *cross-tabulation*) showing the levels and the counts.

Consider the data set `claims`. It contains the number of claims for auto insurance received broken down by the following variables: age of claimant, age of car, type of car, and the average cost of the claims. We can disregard the costs for the moment, and consider the question of which groups of claimants generate the most claims. To make the work easier we create an new data frame `claims.src` which does not contain the `cost` variable:

```
> claims.src <- claims[,-4]
> summary(claims.src)
```

```
            age    car.age  type        number
17-20        :16   0-3:32   A:32   Min.   :  0.00
21-24        :16   4-7:32   B:32   1st Qu.:  9.00
25-29        :16   8-9:32   C:32   Median : 35.50
30-34,35-39 :32   10+:32   D:32   Mean   : 69.86
40-49        :16                   3rd Qu.: 96.25
50-59        :16                     Max. :434.00
60+          :16
```

Use the function `crosstabs` to generate tables of *cross-classified* data. The following call to `crosstabs` generates output showing car age vs. car type.

```
> crosstabs(number~car.age+type, claims.src)
```

```
Call:
crosstabs(number ~ car.age + type, claims.src)
8942 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
car.age|type
       |A      |B      |C      |D      |RowTotl|
-------+-------+-------+-------+-------+-------+
0-3    | 391   |1538   |1517   | 688   |4134   |
       |0.0946 |0.3720 |0.3670 |0.1664 |0.462  |
       |0.3081 |0.3956 |0.5598 |0.6400 |       |
       |0.0437 |0.1720 |0.1696 |0.0769 |       |
-------+-------+-------+-------+-------+-------+
4-7    | 538   |1746   | 941   | 324   |3549   |
       |0.1516 |0.4920 |0.2651 |0.0913 |0.397  |
       |0.4240 |0.4491 |0.3472 |0.3014 |       |
       |0.0602 |0.1953 |0.1052 |0.0362 |       |
-------+-------+-------+-------+-------+-------+
8-9    | 187   | 400   | 191   |  44   |822    |
       |0.2275 |0.4866 |0.2324 |0.0535 |0.092  |
       |0.1474 |0.1029 |0.0705 |0.0409 |       |
       |0.0209 |0.0447 |0.0214 |0.0049 |       |
-------+-------+-------+-------+-------+-------+
10+    | 153   | 204   |  61   |  19   |437    |
       |0.3501 |0.4668 |0.1396 |0.0435 |0.049  |
       |0.1206 |0.0525 |0.0225 |0.0177 |       |
       |0.0171 |0.0228 |0.0068 |0.0021 |       |
-------+-------+-------+-------+-------+-------+
ColTotl|1269   |3888   |2710   |1075   |8942   |
       |0.14   |0.43   |0.30   |0.12   |       |
-------+-------+-------+-------+-------+-------+
Test for independence of all factors
        Chi^2 = 588.2952 d.f.=9 (p=0)
        Yates' correction not used
type=A
age    |car.age                                  8942 cases
       |0-3    |4-7    |8-9    |10+    |RowTotl|  in table
-------+-------+-------+-------+-------+-------+ +----------+
17-20  |  8    |  8    |  4    |  1    |21     | |N         |
       |0.38095|0.38095|0.19048|0.04762|0.0165 | |N/RowTotal|
       |0.02046|0.01487|0.02139|0.00654|       | |N/ColTotal|
       |0.00089|0.00089|0.00045|0.00011|       | |N/Total   |
-------+-------+-------+-------+-------+-------+ +----------+
```

```
21-24  | 18     | 31     | 10     |  4     |63      |
       |                                             |
                      .  .  .
       |                                             |
       |                                             |
-------+--------+--------+--------+--------+--------+
35-39  | 43     | 73     | 21     | 14     |151     |
       |0.28477 |0.48344 |0.13907 |0.09272 |0.1190  |
       |0.10997 |0.13569 |0.11230 |0.09150 |        |
       |0.00481 |0.00816 |0.00235 |0.00157 |        |
-------+--------+--------+--------+--------+--------+
40-49  | 90     | 98     | 35     | 22     |245     |
       |0.36735 |0.40000 |0.14286 |0.08980 |0.1931  |
       |0.23018 |0.18216 |0.18717 |0.14379 |        |
       |0.01006 |0.01096 |0.00391 |0.00246 |        |
-------+--------+--------+--------+--------+--------+
50-59  | 69     |120     | 42     | 35     |266     |
       |0.25940 |0.45113 |0.15789 |0.13158 |0.2096  |
       |0.17647 |0.22305 |0.22460 |0.22876 |        |
       |0.00772 |0.01342 |0.00470 |0.00391 |        |
-------+--------+--------+--------+--------+--------+
60+    | 64     |100     | 43     | 53     |260     |
       |0.24615 |0.38462 |0.16538 |0.20385 |0.2049  |
       |0.16368 |0.18587 |0.22995 |0.34641 |        |
       |0.00716 |0.01118 |0.00481 |0.00593 |        |
-------+--------+--------+--------+--------+--------+
ColTotl|391     |538     |187     |153     |1269    |
       |0.308   |0.424   |0.147   |0.121   |        |
-------+--------+--------+--------+--------+--------+
```

The first argument to `crosstabs` is a *formula* that tells which variables to include in the table. The second argument is the data set where the variables are found. The complete call to `crosstabs` is stored in the resulting object as the attribute `"call"` and is printed at the top of the table.

The next item of information is the number of cases, that is, the total count of all the variables considered. In this example, this is the total of the `number` variable; that is, `sum(claims.src$number)`.

Then you get a key which tells you how to interpret the cells of the table. `N` is the count; below it are the proportions of the whole that the count represents: the proportion of the row total, the proportion of the column total and the proportion of the table total. If there are only two terms in the formula, the table total will be the same as the number of cases. A quick look at the counts, and in particular at the

row totals (`4134`, `3549`, `822`, `437`), shows that there are fewer older cars than newer cars; relatively few cars survive to be eight or nine years old, and the number of cars over ten years old is a tenth that of cars three years or newer. It is slightly more surprising to note the four types of cars don't seem to age equally. You can get an inkling of this by comparing the cells near the top of the table with those near the bottom, but if you compare the third figure in each cell, the one the key tells us is `N/ColTotal`, the progression becomes clear. Of cars of type `D`, 64% are no more than three years old, while only 4% are eight or nine, and less than 2% are over 10. Compare this to type `A` cars, where there are slightly more in the four to seven year age group than in the under three year, the proportion between eight and nine is 0.147 and the proportion over ten years is 12. It seems as if the the type of car is related to its age, and if we look below the table where the results of the χ^2 test for independence are written, we see that the *p*-value is so small it appears as 0.

Of course, we must remember these data are from insurance claims forms–this is not a sample of all the cars on the road, just those that got into accidents and had insurance policies with the company that collected the data.

There may also be an interaction between car type/car age and the age of the owner (which seems likely) and between the age of the owner and the likelihood of a automobile accident.

With `crosstabs`, it is possible to tabulate all this data at once, and print the resulting table in a series of layers, each showing two variables. Thus, when we type `crosstabs(number ~ car.age + type + age, claims.src)`, we get a series of 8 layers, one for each factor (age group) in the variable `age`. The variable represented by the first term in the formula to the left of the `~`, `age.car`, is represented by the rows of each layer, the second term, `car.age` is represented by the columns, and each level of the third, `type`, produces a separate layer. If there were more than three variables, there would be one layer for each possible combination of levels in the variables after the first two. Part of the first of these layers is shown above. Note that the number written in the bottom right margin is the sum of the row totals, and is not the same as the number of cases in the entire table, which is still found at the top of the display and which is used to compute `N/Total`, the fourth figure in each cell.

CHOOSING SUITABLE DATA SETS

Cross tabulation is a technique for categorical data. You tabulate the number of cases for each combination of factors between your variables. In the claims data set these numbers were already tabulated. However, when looking at data that has been gathered as a count, you must always keep in mind exactly what is being counted–thus we can tell that of the 40-49 year old car owners who submitted insurance claims, 43% owned cars of type `B`, and of the cars of type `B` whose owners submitted insurance claims, 25% were owned by 40-49 year olds.

The data set `guayule` also has a response variable which is a count, while all the predictor variables are factors. Here, the thing being counted is the number of rubber plants that sprouted from seeds of a number of varieties subjected to a number of treatments. However, this experiment was designed so that the same number of seeds were planted for each possible combination of the factors of the controlling variables. Since we know the exact make-up of the larger population from which our counts are taken, we can observe the relative size of counts with complaisance and draw conclusions with great confidence. The difference between `guayule` and `claims` is that with the former we can view the outcome variable as a binomial response variable ("sprouted"/"didn't sprout") for which we have tabulated one of the outcomes ("sprouted"), and in the `claims` data set we can't.

Another data set in which all the controlling variables are factors is `solder`.

```
> summary(solder)

Opening    Solder      Mask     PadType  Panel       skips
S:300   Thin :450  A1.5:180  L9   : 90  1:300  Min.   : 0.00
M:300   Thick:450  A3  :270  W9   : 90  2:300  1st Qu.: 0.00
L:300              A6  : 90  L8   : 90  3:300  Median : 2.00
                   B3  :180  L7   : 90         Mean   : 5.53
                   B6  :180  D7   : 90         3rd Qu.: 7.00
                             L6   : 90         Max.   :48.00
                          (Other):360
```

The response variable is the number of skips appearing on a finished circuit board. Since any skip on a board renders it unusable, we can easily turn this into a binary response variable:

```
> attach(solder)
> good <- factor(skips==0)
```

Then, when we want to look at the interaction between the variables, `crosstabs` counts up all the cases with like levels among the factors:

```
crosstabs( ~ Opening + Mask + good)
Call:
crosstabs( ~ Opening + Mask + good)
900 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
good=FALSE
Opening|Mask
       |A1.5   |A3     |A6     |B3     |B6     |RowTotl|
-------+-------+-------+-------+-------+-------+-------+
S      |49     |76     |30     |60     |60     |275    |
       |0.1782 |0.2764 |0.1091 |0.2182 |0.2182 |0.447  |
       |0.5326 |0.5033 |0.3371 |0.4444 |0.4054 |       |
       |0.0544 |0.0844 |0.0333 |0.0667 |0.0667 |       |
-------+-------+-------+-------+-------+-------+-------+
M      |22     |35     |59     |39     |51     |206    |
       |0.1068 |0.1699 |0.2864 |0.1893 |0.2476 |0.335  |
       |0.2391 |0.2318 |0.6629 |0.2889 |0.3446 |       |
       |0.0244 |0.0389 |0.0656 |0.0433 |0.0567 |       |
-------+-------+-------+-------+-------+-------+-------+
L      |21     |40     | 0     |36     |37     |134    |
       |0.1567 |0.2985 |0.0000 |0.2687 |0.2761 |0.218  |
       |0.2283 |0.2649 |0.0000 |0.2667 |0.2500 |       |
       |0.0233 |0.0444 |0.0000 |0.0400 |0.0411 |       |
-------+-------+-------+-------+-------+-------+-------+
ColTotl|92     |151    |89     |135    |148    |615    |
       |0.1496 |0.2455 |0.1447 |0.2195 |0.2407 |       |
-------+-------+-------+-------+-------+-------+-------+
```

```
good=TRUE
Opening|Mask
       |A1.5   |A3     |A6     |B3     |B6     |RowTotl|
-------+-------+-------+-------+-------+-------+-------+
S      |11     |14     | 0     | 0     | 0     |25     |
       |0.4400 |0.5600 |0.0000 |0.0000 |0.0000 |0.088  |
       |0.1250 |0.1176 |0.0000 |0.0000 |0.0000 |       |
       |0.0122 |0.0156 |0.0000 |0.0000 |0.0000 |       |
-------+-------+-------+-------+-------+-------+-------+
M      |38     |25     | 1     |21     | 9     |94     |
       |0.4043 |0.2660 |0.0106 |0.2234 |0.0957 |0.330  |
       |0.4318 |0.2101 |1.0000 |0.4667 |0.2812 |       |
       |0.0422 |0.0278 |0.0011 |0.0233 |0.0100 |       |
-------+-------+-------+-------+-------+-------+-------+
L      |39     |80     | 0     |24     |23     |166    |
       |0.2349 |0.4819 |0.0000 |0.1446 |0.1386 |0.582  |
       |0.4432 |0.6723 |0.0000 |0.5333 |0.7188 |       |
       |0.0433 |0.0889 |0.0000 |0.0267 |0.0256 |       |
-------+-------+-------+-------+-------+-------+-------+
ColTotl|88     |119    |1      |45     |32     |285    |
       |0.3088 |0.4175 |0.0035 |0.1579 |0.1123 |       |
-------+-------+-------+-------+-------+-------+-------+
Test for independence of all factors
        Chi^2 = 377.3556 d.f.= 8 (p=0)
        Yates' correction not used
```

In the first example above we specified where to look for the variables `age`, `car.age` and `type` by giving the data frame `claims.src` as the second argument of `crosstabs`. In the second example, we attached the data frame `solder` and let `crosstabs` find the variables in the search list. Both methods work because, when `crosstabs` goes to interpret a term in the formula, it looks first in the data frame specified by the argument data and then in the search list. You can specifiy the data set with the name of a data frame, or a frame number in which to find an attached data frame. Using a frame number gives the advantage of speed that comes from attaching the data frame, while protecting against the possibility of having masked the name of one of the variables with something in your `.Data` directory:

```
> attach(guayule)
```

```
> search()

[1] ".Data"
[2] "guayule" . . .

> rubber <- crosstabs(plants~variety+treatment, data=2)
```

If you specify a data frame and do not give a formula, `crosstabs` uses the formula `~ .`, that is, it will cross classify all the variables in the data frame. Any variable names not found in the specified data frame (which is all of them if you don't specify any) are sought in the search list.

CROSS-TABULATING CONTINUOUS DATA

As was seen in the example of the `solder` data frame above, it is fairly easy to turn a continuous response variable into a binomial response variable. Clearly, we could have used any logical expression that made sense to do so–we could have chosen any cutoff point for acceptable numbers of skips.

A somewhat harder problem is presented by the case where you want a multinomial factor from continuous data. You can make judicious use of the `cut` function to turn the continuous variables into factors, but you need to put care and thought into the points at which to separate the data into ranges. The quartiles given by the function `summary` offer a good starting point. The data frame `kyphosis` represents data on 81 children who have had corrective spinal surgery. The variables here are whether a postoperative deformity (kyphosis) is present, the age of the child in months, the number of vertebrae involved in the operation, and beginning of the range of vertebrae involved.

```
> summary(kyphosis)

  Kyphosis        Age             Number          Start
absent :64  Min.   :  1.00  Min.   : 2.000  Min.   : 1.00
present:17  1st Qu.: 26.00  1st Qu.: 3.000  1st Qu.: 9.00
            Median : 87.00  Median : 4.000  Median :13.00
            Mean   : 83.65  Mean   : 4.049  Mean   :11.49
            3rd Qu.:130.00  3rd Qu.: 5.000  3rd Qu.:16.00
            Max.   :206.00  Max.   :10.000  Max.   :18.00
```

The summary of these variables suggests that two year intervals might be a reasonable division for the age. We use the `cut` function to break the variable `Age` into factors at a sequence of points at 24 month intervals and to label the resulting levels with the appropriate range of years. Since there are at most nine values for `Number` we leave it alone for the moment. Since the mean of the `Start` variable is close to the first quartile, a fairly coarse division of `Start` is probably sufficient. We could require that `cut` simply divide the data into four segments of equal length with the command `cut(Start, 4)`, but the results of this, while mathematically correct, look a bit bizarre–the first level thus created is `"0.830+ thru 5.165"`. The `pretty`

function divides the range of `Start` into equal intervals with whole number end points, and the `cut` function makes them into levels with reasonable names:

```
> attach(kyphosis)
> kyphosis.fac <- data.frame(Kyphosis=Kyphosis,
+  Age =  cut(Age, c( seq(0, 144, by=24), 206),
+    labels=
+      c("0-2","2-4","4-6","6-8","8-10","10-12","12+")),
+  Number = Number,
+  Start = cut(Start, pretty(Start, 4) ) )
> detach(2)
> summary(kyphosis.fac)

   Kyphosis       Age         Number               Start
 absent :64   0-2  :20   Min.   : 2.000    0+ thru  5:13
 present:17   2-4  : 7   1st Qu.: 3.000    5+ thru 10:14
              4-6  : 8   Median : 4.000   10+ thru 15:32
              6-8  : 9   Mean   : 4.049   15+ thru 20:22
              8-10 :11   3rd Qu.: 5.000
              10-12:14   Max.   :10.000
              12+  :12
> attach(kyphosis.fac)
```

The cross-tabulation of this data can then be easily examined:

```
> crosstabs(~Age+Kyphosis, kyphosis.fac)

Call:
crosstabs( ~ Age + Kyphosis, kyphosis.fac)
81 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
```

```
Age     |Kyphosis
        |absent |present|RowTotl|
--------+-------+-------+-------+
0-2     |19     | 1     |20     |
        |0.950  |0.050  |0.247  |
        |0.297  |0.059  |       |
        |0.235  |0.012  |       |
--------+-------+-------+-------+
2-4     | 6     | 1     |7      |
        |                       |
           .  .  .
        |                       |
--------+-------+-------+-------+
10-12   | 9     | 5     |14     |
        |0.643  |0.357  |0.173  |
        |0.141  |0.294  |       |
        |0.111  |0.062  |       |
--------+-------+-------+-------+
12+     |11     | 1     |12     |
        |0.917  |0.083  |0.148  |
        |0.172  |0.059  |       |
        |0.136  |0.012  |       |
--------+-------+-------+-------+
ColTotl|64     |17     |81     |
        |0.79   |0.21   |       |
--------+-------+-------+-------+
Test for independence of all factors
        Chi^2 = 9.588004 d.f.= 6 (p=0.1431089)
        Yates' correction not used
        Some expected values are less than 5,
             don't trust stated p-value
```

CROSS-CLASSIFYING SUBSETS OF DATA FRAMES

There are two ways to subset a data frame for cross-classification. First, the `crosstabs` function will cross-tabulate only those variables specified in the formula. If there is one variable in the data frame in which you are not interested, don't mention it. Second, you can choose which rows you want to consider with the subset argument. You can use anything you would normally use to subscript the rows of a data frame. Thus, the `subset` argument can be an expression that evaluates to a logical vector, or a vector of row numbers or row names.

As an example, recall the `solder` data set. You can look at the relation between the variables without turning `skips` explicitly into a binomial variable by using it to subscript the rows of the data frame:

```
> crosstabs(~Solder+Opening, solder, subset= skips<10)

Call:
crosstabs( ~ Solder+Opening, solder, subset = skips<10)
729 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
Solder |Opening
       |S      |M      |L      |RowTotl|
-------+-------+-------+-------+-------+
Thin   | 50    |133    |140    |323    |
       |0.155  |0.412  |0.433  |0.44   |
       |0.294  |0.494  |0.483  |       |
       |0.069  |0.182  |0.192  |       |
-------+-------+-------+-------+-------+
Thick  |120    |136    |150    |406    |
       |0.296  |0.335  |0.369  |0.56   |
       |0.706  |0.506  |0.517  |       |
       |0.165  |0.187  |0.206  |       |
-------+-------+-------+-------+-------+
```

```
ColTotl|170    |269    |290    |729    |
       |0.23   |0.37   |0.40   |       |
-------+-------+-------+-------+-------+
Test for independence of all factors
        Chi^2 = 20.01129 d.f.= 2 (p=4.514445e-05)
        Yates' correction not used
```

A more common use of the subscript is to look at some of the variables while considering only a subset of the levels of another:

```
> crosstabs( ~ Solder+Opening+good, subset = Panel == "1")

Call:
crosstabs( ~ Solder+Opening+good, subset = Panel == "1")
300 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
good=FALSE
Solder |Opening
       |S      |M      |L      |RowTotl|
-------+-------+-------+-------+-------+
Thin   |49     |33     |31     |113    |
       |0.4336 |0.2920 |0.2743 |0.59   |
       |0.5444 |0.5410 |0.7949 |       |
       |0.1633 |0.1100 |0.1033 |       |
-------+-------+-------+-------+-------+
Thick  |41     |28     | 8     |77     |
       |0.5325 |0.3636 |0.1039 |0.41   |
       |0.4556 |0.4590 |0.2051 |       |
       |0.1367 |0.0933 |0.0267 |       |
-------+-------+-------+-------+-------+
ColTotl|90     |61     |39     |190    |
       |0.474  |0.321  |0.205  |       |
-------+-------+-------+-------+-------+
```

```
good=TRUE
Solder |Opening
       |S      |M      |L      |RowTotl|
-------+-------+-------+-------+-------+
Thin   | 1     |17     |19     |37     |
       |0.0270 |0.4595 |0.5135 |0.34   |
       |0.1000 |0.4359 |0.3115 |       |
       |0.0033 |0.0567 |0.0633 |       |
-------+-------+-------+-------+-------+
Thick  | 9     |22     |42     |73     |
       |0.1233 |0.3014 |0.5753 |0.66   |
       |0.9000 |0.5641 |0.6885 |       |
       |0.0300 |0.0733 |0.1400 |       |
-------+-------+-------+-------+-------+
ColTotl|10     |39     |61     |110    |
       |0.091  |0.355  |0.555  |       |
-------+-------+-------+-------+-------+
Test for independence of all factors
        Chi^2 = 82.96651 d.f.= 2 (p=0)
        Yates' correction not used
```

MANIPULATING AND ANALYZING CROSS-CLASSIFIED DATA

When you apply `crosstabs` to a data frame you get a multidimensional array whose elements are the counts and whose dimensions are the variables involved in the cross-tabulations. The first factor variable is the first, or row dimension, the second is the second, or column dimension, the third is the third dimension, etc. If you wish to do more than tabulate data, say compute means or sums of cross-classified data, you can apply functions to the elements of the array with the function `tapply`.

POWER AND SAMPLE SIZE 7

INTRODUCTION

When contemplating a study, one of the first statistical questions that arises is "How big does my sample need to be?" The required sample size is a function of the alternative hypothesis, the probabilities of Type I and Type II errors, and the variability of the population(s) under study. Two new functions are available for computing power and sample size requirements, `normal.sample.size` and `binomial.sample.size`. Depending on the input, these functions will provide:

- For given power and alternative hypothesis, the required sample size
- For given sample size and power, the detectable difference
- For given sample size and alternative hypothesis, the power to distinguish between the hypotheses

These functions can be applied in one- and two-sample studies, and will produce a table from vectorized input suitable for passing to Trellis graphics.

POWER AND SAMPLE SIZE THEORY

When designing a study, one of the first questions to arise is "How large does my sample size need to be?" Intuitively, we have a sense that this depends on how small a difference we're trying to detect, how much variability is inherent in our data, and how certain we want to be of our results. In a classical hypothesis test of H_0 (null hypothesis) versus H_a (alternative hypothesis), there are four possible outcomes, two of which are erroneous:

- Don't reject H_0 when is H_0 true.
- Reject H_0 when H_0 is false.
- Reject H_0 when H_0 is true (type I error).
- Don't reject H_0 when H_0 is false (type II error).

To construct a test, the distribution of the test statistic under H_0 is used to find a critical region which will ensure the probability of committing a type I error does not exceed some predetermined level. This probability is typically denoted α. The *power* of the test is its ability to *correctly reject* the null hypothesis, or 1 - Pr(type II error), which is based on the distribution of the test statistic under H_a. The required sample size then will be a function of

1. The null and alternative hypotheses.
2. The target α.
3. The desired power to detect H_a.
4. The variability within the population(s) under study.

Our objective is, for a given test, to find a relationship between the above factors and the sample size that will enable us to select a sample size consistent with the desired α and power.

NORMALLY DISTRIBUTED DATA

One-Sample Test of Gaussian Mean

When conducting a one-sample test of a normal mean, we start by writing our assumptions and hypotheses:

$$X_i \sim N(\mu,\ \sigma^2)$$

where $i = 1, \ldots, n$, and σ^2 is known. To perform a two-sided test of equality the hypotheses would be as follows:

$$H_0: \mu = \mu_0$$
$$H_a: \mu = \mu_a$$

Our best estimate of μ is the sample mean, which is normally distributed:

$$\bar{X} \sim N\left(\mu,\ \frac{\sigma^2}{n}\right)$$

and the test statistic is

$$Z = \sqrt{n}(\bar{X} - \mu_0) / \sigma$$
$$Z \sim N(\mu - \mu_0,\ 1)$$
$$Z \sim N(0,\ 1) \text{ for } H_0$$

Reject H_0 if $|Z| > Z_{1-\alpha/2}$, which guarantees a level α test. The power of the test to detect $\mu = \mu_0$ is

$$\text{Power} = \Phi\left(\frac{\sqrt{n}(\mu_0 - \mu_a)}{\sigma} - Z_{1-\alpha/2}\right) + \Phi\left(\frac{\sqrt{n}(\mu_a - \mu_0)}{\sigma} - Z_{1-\alpha/2}\right)$$

We can think of the left side of the sum as the *lower power*, or the power to detect $\mu_a < \mu_0$, and the right side as the *upper power*, or the power to detect $\mu_a > \mu_0$. Solving for n using both upper and lower power would be difficult, but we note that when $\mu_a - \mu_0 < 0$, the upper power is negligible ($< \alpha/2$) and similarly the lower power is small when $\mu_a - \mu_0 > 0$. So the equation can be simplified by using

the absolute value of the difference between μ_a and μ_0 and considering only one side of the sum. This results in the following sample size formula:

$$n = [(\sigma(Z_{1-\alpha/2} + Z_{Power}))/|\mu_a - \mu_o|]^2$$

Comments

- While only one of upper power and lower power is used in deriving the sample size formula, the S-PLUS functions for computing power and sample size uses both the upper and lower power when computing the *power* of a two-tailed test for a given sample size.
- In practice, the variance of the population is seldom known and the test statistic is based on the *t-distribution*. Using the t-distribution to derive sample size requires an iterative approach, since the sample size is needed to specify the degrees of freedom. The difference between the quantile value for the t-distribution versus the standard normal is only significant when small sample sizes are required, so the standard formula based on the normal distribution was chosen. Keep in mind that for samples sizes less than 10, the power of a t-test could be significantly less than the target power.
- The formula for a one-tailed test is derived along similar lines, and is exactly the same as the two-tailed formula with the exception that $Z_{1-\alpha/2}$ is replaced by $Z_{1-\alpha}$.

Examples

The function for computing sample size for normally distributed data is `normal.sample.size`. This function can be used to compute sample size, power, or minimum detectable difference and will automatically chose what to compute based on what information is input. Here are some simple examples:

```
#
# one-sample case, using all the defaults
#
> normal.sample.size(mean.alt = 0.3)

  mean.null sd1 mean.alt delta alpha power n1
1         0   1      0.3   0.3  0.05   0.8 88
```

```
#
# reduce output with summary
#
> summary(normal.sample.size(mean.alt = 0.3))

  delta power n1
1   0.3   0.8 88

#
# upper-tail test recomputing power
#
> normal.sample.size(mean = 100, mean.alt = 105, sd1 = 10,
+ power = c(.8, .9, .95, .99), alt = "greater",
+ recompute.power = T)

  mean.null sd1 mean.alt delta alpha     power n1
1       100  10      105     5  0.05 0.8037649 25
2       100  10      105     5  0.05 0.9054399 35
3       100  10      105     5  0.05 0.9527153 44
4       100  10      105     5  0.05 0.9907423 64

#
# calculate power
#
> normal.sample.size(mean = 100, mean.alt = 105, sd1 = 10,
+ n1 = (1:5)*20)

  mean.null sd1 mean.alt delta alpha     power  n1
1       100  10      105     5  0.05 0.6087795  20
2       100  10      105     5  0.05 0.8853791  40
3       100  10      105     5  0.05 0.9721272  60
4       100  10      105     5  0.05 0.9940005  80
5       100  10      105     5  0.05 0.9988173 100

#
# lower-tail test, minimum detectable difference
#
> summary(normal.sample.size(mean = 100, sd1 = 10,
+ n1 = (1:5)*20, power = .9, alt = "l"))

  mean.alt     delta power  n1
1 93.45636 -6.543641   0.9  20
2 95.37295 -4.627053   0.9  40
3 96.22203 -3.777973   0.9  60
```

```
4 96.72818 -3.271821   0.9  80
5 97.07359 -2.926405   0.9 100
```

See the online help files for `normal.sample.size` and `summary.power.table` for more details.

Comparing Means From Two Samples

Extending this formula to two-sampled tests, is relatively easy. Given two independent samples from normal distributions

$$X_{1,\,i} \sim N(\mu_1,\ \sigma_1^2) \qquad i = 1,\ \ldots,\ n_1$$
$$X_{2,\,j} \sim N(\mu_2,\ \sigma_2^2) \qquad j = 1,\ \ldots,\ n_1$$

where $n_2 = kn_1$, we'll construct a two-sided test of equality of means

$$H_0 : \mu_1 = \mu_2$$
$$H_a : \mu_1 \neq \mu_2$$

which is more conveniently written

$$H_0 : \mu_2 - \mu_1 = 0$$
$$H_a : \mu_2 - \mu_1 \neq 0$$

The difference of the sample means is normally distributed

$$(\bar{X}_2 - \bar{X}_1) \sim N\left(\mu_2 - \mu_1,\ \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}\right) \sim N\left(\mu_2 - \mu_1,\ \frac{1}{n_1}\left(\sigma_1^2 + \frac{\sigma_2^2}{k}\right)\right)$$

which leads to the test statistic

$$Z = \frac{\bar{X}_2 - \bar{X}_1}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

Derivation of the two-sample formulas proceed along the same lines as the one-sample case, producing the following formulas:

$$n_1 = \left(\sigma_1 + \frac{\sigma_2^2}{k}\right)\left[\frac{(Z_{(1-\alpha/\ 2)} + Z_{Power})}{|\mu_2 - \mu_1|}\right]^2$$
$$n_2 = kn_1$$

Examples

For two-sample cases, use `normal.sample.size` with `mean2` instead of `mean.alt`:

```
#
# Don't round sample size
#
> summary(normal.sample.size(mean2 = 0.3, exact.n = T))

  delta power       n1       n2
1   0.3   0.8 174.4195 174.4195

#
# round sample size, then recompute power
#
> summary(normal.sample.size(mean2 = 0.3, recompute = T))

  delta     power  n1  n2
1   0.3 0.8013024 175 175

#
# Unequal sample sizes, lower tail test
#
> normal.sample.size(mean = 100, mean2 = 94, sd1 = 15,
+ prop.n2 = 2, power = 0.9, alt = "less")

  mean1 sd1 mean2 sd2 delta alpha power n1  n2 prop.n2
1   100  15    94  15    -6  0.05   0.9 81 162       2
```

BINOMIAL DATA

One-Sample Test of Binomial Proportion

Another very common test is for a *binomial proportion*. Say we have data sampled from a binomial distribution,

$$X_i \sim B(\pi,\ n),\ i = 1,\ \ldots,\ n$$

Each X_i represents the number of "successes" observed in *n Bernoulli trials*, where Pr(success) = π. The mean and variance of the random variable X is

$$E(X) = n\pi$$
$$Var(X) = n\pi(1-\pi)$$

We wish to test the value of the parameter π, using a two-sided test.

$$H_0: \pi = \pi_0$$
$$H_a: \pi = \pi_a$$

We could use an exact binomial test, but for sufficiently large *n*, and if the distribution is not too skewed (π is not too close to 0 or 1), a normal approximation can be used. A good rule of thumb is that the normal distribution will be a good approximation to the binomial distribution if

$$n\pi(1-\pi) \geq 5$$

When using a continuous distribution to approximate a discrete one, a *continuity correction* is usually recommended; typically, a value of 1/2 is used to extend the range in either direction, so

$$Pr(X_l \leq X \leq X_u)$$

using a binomial distribution, becomes

$$Pr\left(X_l - \frac{1}{2} \leq X \leq X_u + \frac{1}{2}\right)$$

when using a normal approximation. If the continuity correction is temporarily suppressed, the sample size formula is derived very much as in the normal case:

$$n^* = \left[\frac{\sqrt{\pi_0(1-\pi_0)}Z_{1-\alpha/2} + \sqrt{\pi_0(1-\pi_0)}Z_{Power}}{|\pi_a - \pi_0|} \right]^2$$

There have been several suggestions concerning how to best incorporate a continuity correction into the sample-size formula. The one adopted in the S-PLUS function `binomial.sample.size` for a one-sample test is

$$n = n^* + \frac{2}{|\pi_a - \pi_0|}$$

Examples

```
#
# one-sample case, using all the defaults
#
> binomial.sample.size(p.alt = 0.3)

  p.null p.alt delta alpha power n1
1    0.5   0.3  -0.2  0.05   0.8 37

#
# minimal output
#
> summary(binomial.sample.size(p.alt = 0.3))

  delta power n1
1  -0.2   0.8 37

#
# compute power
#
> binomial.sample.size(p = .2, p.alt = .12, n1 = 250)

  p.null p.alt delta alpha     power  n1
1    0.2  0.12 -0.08  0.05 0.8997619 250
```

Comparing Proportions From Two Samples

The two-sample test for proportions is a bit more involved than the others we've looked at. Say we have data sampled from two binomial distributions

$$X_{1,\,i} \sim B(\pi_1,\ n_1),\ i = 1,\ \ldots,\ n_1$$
$$X_{2,\,j} \sim B(\pi_2,\ n_2),\ j = 1,\ \ldots,\ n_2$$

where $n_2 = kn_1$, we'll construct a two-sided test of equality of means

$$H_0: \pi_1 = \pi_2$$
$$H_a: \pi_1 \neq \pi_2$$

which is more conveniently written

$$H_0: \pi_1 - \pi_2 = 0$$
$$H_a: \pi_1 - \pi_2 \neq 0$$

Using our best estimator of the parameter π, we can begin constructing a test statistic:

$$\hat{\pi}_1 = \frac{1}{n_1} \sum_{i=1}^{n_1} X_{1,\,i}$$

$$\hat{\pi}_2 = \frac{1}{n_2} \sum_{j=1}^{n_2} X_{2,\,j}$$

$$\hat{\pi}_2 - \hat{\pi}_1 \sim N\left(\pi_2 - \pi_1,\ \frac{\pi_1(1-\pi_1)}{n_1} + \frac{\pi_2(1-\pi_2)}{n_2}\right)$$

$$\hat{\pi}_2 - \hat{\pi}_1 \sim N\left(\pi_2 - \pi_1,\ \frac{1}{n_1}\left(\pi_1(1-\pi_1) + \frac{\pi_2(1-\pi_2)}{k}\right)\right)$$

In the case where the null hypothesis is true, so $\pi_2 = \pi_1 = \pi$, this can be written as

$$\hat{\pi}_2 - \hat{\pi}_1 \sim N\left(0,\ \frac{\pi(1-\pi)}{n_1}\left(1 + \frac{1}{k}\right)\right)$$

Immediately a problem arises, namely, the variance needed to construct the test statistic depends on the parameters being tested. It seems reasonable to use all of the data available to estimate the variances, and that is exactly what is done. A weighted average of the two estimates for the proportions is used to estimate the variance under H_0. The test statistic then is

$$\bar{\pi} = \frac{n_1\hat{\pi}_1 + n_2\hat{\pi}_2}{n_1 + n_2} = \frac{\hat{\pi}_1 + k\hat{\pi}_2}{1 + k}$$

$$Z = \frac{\hat{\pi}_2 - \hat{\pi}_1}{\sqrt{\bar{\pi}(1 - \bar{\pi})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

If the null hypothesis is true, this gives $Z \sim N(0,\ 1)$. We use this to derive the formula without continuity correction:

$$n_1^* = \left[\frac{\sqrt{\pi_1(1-\pi_1) + \frac{\pi_2(1-\pi_2)}{k}}\,Z_{Power} + \sqrt{\bar{\pi}(1-\bar{\pi})\left(1 + \frac{1}{k}\right)}\,Z_{1-\alpha/2}}{|\pi_2 - \pi_1|}\right]^2$$

Applying the two-sample adjustment for a continuity correction produces the final results

$$n_1 = n_1^* + \frac{k+1}{k|\pi_2 - \pi_1|}$$

$$n_2 = kn_1$$

Examples

```
#
# for two-sample, use p2 instead of p.alt
#
> summary(binomial.sample.size(p2 = 0.3))

  delta power  n1  n2
1  -0.2   0.8 103 103
```

```
#
# Don't round sample size and don't use continuity
# correction
#
> summary(binomial.sample.size(p2 = 0.3, exact.n = T,
+ correct = F))

  delta power       n1       n2
1  -0.2   0.8 92.99884 92.99884

#
# round sample size, then recompute power
#
> summary(binomial.sample.size(p2 = 0.3, recompute = T))

  delta     power  n1  n2
1  -0.2 0.8000056 103 103

#
# Unequal sample sizes, lower tail test
#
> binomial.sample.size(p = .1, p2 = .25, prop.n2 = 2,
+ power = 0.9, alt = "less")

   p1   p2 delta alpha power n1  n2 prop.n2
1 0.1 0.25  0.15  0.05   0.9 92 184       2

#
# Compute minimum detectable difference (delta) given
sample
# size and power.
#
> binomial.sample.size(p = .6, n1 = 500, prop.n2 = .5,
+ power = c(.8, .9, .95))

   p1        p2     delta alpha power  n1  n2 prop.n2
1 0.6 0.7063127 0.1063127  0.05  0.80 500 250     0.5
2 0.6 0.7230069 0.1230069  0.05  0.90 500 250     0.5
3 0.6 0.7367932 0.1367932  0.05  0.95 500 250     0.5
```

```
#
# compute power
#
> binomial.sample.size(p = 0.3, p2 = seq(0.31, 0.35,
+ by=0.01), n1 = 1000, prop.n2 = 0.5)

 p1   p2 delta alpha      power   n1  n2 prop.n2
1 0.3 0.31  0.01  0.05 0.06346465 1000 500     0.5
2 0.3 0.32  0.02  0.05 0.11442940 1000 500     0.5
3 0.3 0.33  0.03  0.05 0.20446778 1000 500     0.5
4 0.3 0.34  0.04  0.05 0.32982868 1000 500     0.5
5 0.3 0.35  0.05  0.05 0.47748335 1000 500     0.5
```

REFERENCES

Rosner, Bernard (1990). *Fundamentals of Biostatistics* (Third Edition). PWS-Kent, Boston.

Fisher, Lloyd D. and Van Belle, Gerald (1993). *Biostatistics.* Wiley, New York.

Fleiss, Joseph L. (1981). *Statistical Methods for Rates and Proportions.* Wiley, New York.

REGRESSION AND SMOOTHING FOR CONTINUOUS RESPONSE DATA

8

INTRODUCTION

Regression is a tool for exploring relationships between variables. *Linear regression* explores relationships that are readily described by straight lines, or their generalization to many dimensions. A surprisingly large number of problems can be analyzed using the techniques of linear regression, and even more can be attacked by means of *transformations* of the original variables that result in linear relationships among the transformed variables. In recent years, the techniques themselves have been extended through the addition of robust methods and generalizations of the classical linear regression techniques. These generalizations allow familiar problems in categorical data analysis such as logistic and Poisson regression to be subsumed under the heading of the *generalized linear model* (GLM), while still further generalizations allow a predictor to be replaced by an arbitrary smooth function of the predictor in building a *generalized additive model* (GAM).

This chapter describes regression and smoothing in the case of a univariate, continuous response. We start with simple regression, that is, regression with a single predictor variable: fitting the model, examining the fitted models, and analyzing the residuals. We then examine multiple regression, varying models by adding and dropping terms as appropriate. Again, we examine the fitted models and analyze the residuals. We then consider the special case of *weighted regression*, which underlies many of the robust techniques and generalized regression methods.

One important reason for performing regression analysis is to get a model useful for prediction. The section Prediction With the Model describes how to use S-PLUS to obtain predictions from your fitted model, and the section Confidence Intervals describes how to obtain pointwise and simultaneous confidence intervals.

The classical linear regression techniques make several strong assumptions about the underlying data, and the data can fail to satisfy these assumptions in different ways–for example, the regression line may be thrown off by one or more outliers or the data may not be fitted well by any straight line. In the first case, we can bring *robust regression* methods into play; these minimize the effects of outliers

while retaining the basic form of the linear model. Conversely, the robust methods are often useful in identifying outliers. We discuss robust regression in detail in a later chapter.

In the second case, we can expand our notion of the linear model, either by adding polynomial terms to our straight line model, or by replacing one or more predictors by an arbitrary smooth function of the predictor, converting the classical linear model into a generalized additive model (GAM).

Scatterplot smoothers are useful tools for fitting arbitrary smooth functions to a scatter plot of data points. The smoother summarizes the trend of the measured response as a function of the predictor variables. We describe several scatterplot smoothers available in S-PLUS, and describe how the smoothed values they return can be incorporated into additive models.

SIMPLE LEAST-SQUARES REGRESSION

Simple regression uses the method of least squares to fit a continuous, univariate response as a linear function of a single predictor variable. In the method of least squares, we fit a line to the data so as to minimize the sum of the squared residuals. Given a set of n observations y_i of the response variable corresponding to a set of values x_i of the predictor and an arbitrary model $\hat{y} = \hat{f}(x)$, the ith *residual* is defined as the difference between the ith observation y_i and the fitted value $\hat{y}_i = \hat{f}(x_i)$, that is, $r_i = y_i - \hat{y}_i$.

To do simple regression with S-PLUS, use the function `lm` (for linear model) with a simple *formula* linking your chosen response variable to the predictor variable. In many cases, both the response and the predictor are components of a single data frame, which can be specified as the `data` argument to `lm`. For example, consider the air pollution data in the built-in data set `air`:

```
> air[,c(1,3)]

      ozone temperature
 1 3.448217          67
 2 3.301927          72
 3 2.289428          74
 4 2.620741          62
 5 2.843867          65
 . . .
```

A scatter plot of the data is shown in Figure 8.1.

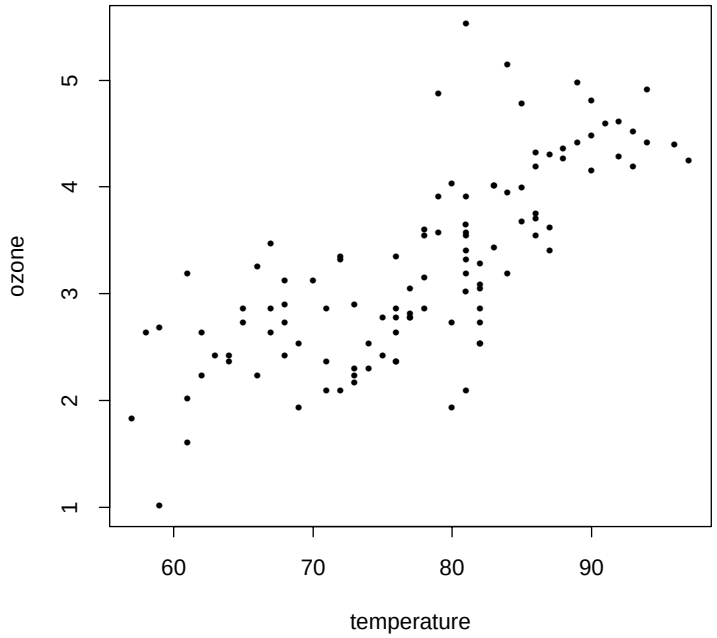

Figure 8.1: *Scatter plot of* `ozone` *against* `temperature`.

From the scatter plot, we hypothesize a linear relationship between temperature and ozone concentration. We choose `ozone` as the response and `temperature` as the single predictor. The choice of response and predictor variables is driven by the subject matter in which the data arise, rather than by statistical considerations.

To fit the model, use `lm` as follows:

```
> ozone.lm <- lm(ozone ~ temperature, data = air)
```

The first argument, `ozone ~ temperature`, is the formula specifying that the variable `ozone` is modeled as a function of `temperature`. The second argument specifies that the data for the linear model is contained in the data frame `air`.

Use the `summary` function to obtain a summary of the fitted model:

```
> summary(ozone.lm)
```

```
Call: lm(formula = ozone ~ temperature)
Residuals:
   Min      1Q  Median      3Q   Max
 -1.49 -0.4258 0.02521 0.3636 2.044

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept)  -2.2260   0.4614    -4.8243   0.0000
temperature   0.0704   0.0059    11.9511   0.0000

Residual standard error: 0.5885 on 109 degrees of freedom
Multiple R-Squared: 0.5672
F-statistic: 142.8 on 1 and 109 degrees of freedom, the
p-value is 0

Correlation of Coefficients:
            (Intercept)
temperature -0.9926
```

The `Value` column under `Coefficients` gives the coefficients of the linear model, allowing us to read off the estimated regression line as follows:

```
ozone = -2.2260 + 0.0704 x temperature
```

The column headed `Std. Error` gives the estimated standard error for each coefficient. The `Multiple R-Squared` term from the `lm` summary tells us that the model explains about 57% of the variation in `ozone`. The `F-statistic` is the ratio of the mean square of the regression to the estimated variance; if there is no relationship between temperature and ozone, this ratio has an F distribution with 1 and 109 degrees of freedom. The ratio here is clearly significant, so the true slope of the regression line is probably not 0.

Diagnostic Plots For Linear Models

Suppose we have the linear model defined as follows:

```
> ozone.lm <- lm(ozone ~ temperature, data=air)
```

How good is the fitted linear regression model? Is temperature an adequate predictor of ozone concentration? Can we do better? Questions such as these are essential any time you try to explain data with a statistical model.

It is not enough to fit a model; you must also assess how well that model fits the data, being ready to modify the model or abandon it altogether if it does not satisfactorily explain the data.

The simplest and most informative method for assessing the fit is to look at the model graphically, using an assortment of plots that, taken together, reveal the strengths and weaknesses of the model. For example, a plot of the response against the fitted values gives a good idea of how well the model has captured the broad outlines of the data, while examining a plot of the residuals against the fitted values often reveals unexplained structure left in the residuals, which in a strong model should appear as nothing but noise. The default plotting method for `lm` objects provides these two plots, along with the following useful plots:

- *Square root of absolute residuals against fitted values.* This plot is useful in identifying outliers and visualizing structure in the residuals.
- *Normal quantile plot of residuals.* This plot provides a visual test of the assumption that the model's errors are normally distributed. If the ordered residuals cluster along the superimposed quantile-quantile line, you have strong evidence that the errors are indeed normal.
- *Residual-Fit spread plot*, or *r-f plot.* This plot compares the spread of the fitted values with the spread of the residuals. Since the model is an attempt to explain the variation in the data, you hope that the spread in the fitted values is *much* greater than that in the residuals.
- *Cook's distance plot.* Cook's distance is a measure of the influence of individual observations on the regression coefficients.

Calling `plot` as follows yields the six plots shown in Figure 8.2:

```
> par(mfrow=c(2,3))
> plot(ozone.lm)
```

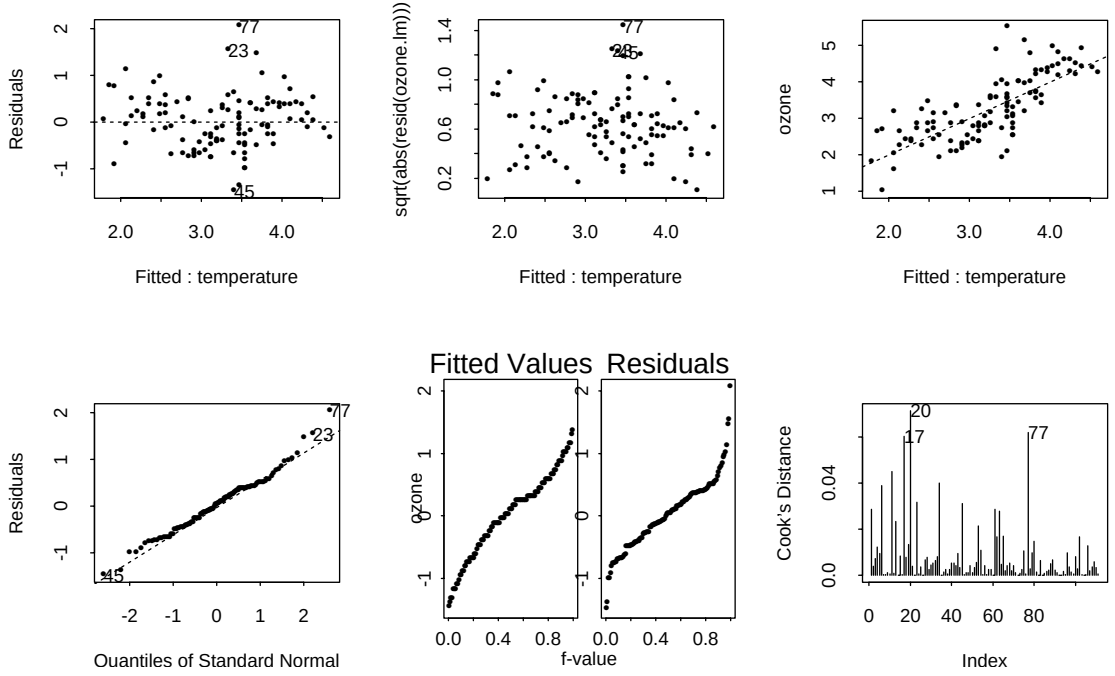

Figure 8.2: *Default plots for* `lm` *objects.*

The line $y = \hat{y}$ is shown as a dashed line in the third plot (far right of top row). In the case of simple regression, this line is visually equivalent to the regression line. The regression line appears to model the trend of the data reasonably well. The residuals plots (left and center, top row) show no obvious pattern, although five observations appear to be outliers. By default, as in Figure 8.2, the three most extreme values are identified in each of the residuals plots and the Cook's distance plot. You can request a different number of points by using the `id.n` argument in the call to plot; for this model, `id.n = 5` is a good choice.

Another useful diagnostic plot is the normal plot of residuals (left plot, bottom row). The normal plot gives no reason to doubt that the residuals are normally distributed.

The r-f plot, on the other hand (middle plot, bottom row), shows a weakness in this model; the spread of the residuals is actually greater than the spread in the original data. However, if we ignore the five outlying residuals, the residuals are more tightly bunched than the original data.

The Cook's distance plot shows four or five heavily influential observations. As the regression line fits the data reasonably well, the regression is significant, and the residuals appear normally distributed, we feel justified in using the regression line as a way to estimate the ozone concentration for a given temperature. One important issue remains–the regression line explains only 57% of the variation in the data. We may be able to do somewhat better by considering the effect of other variables on the ozone concentration. See the section Multiple Regression for this further analysis.

At times, you are not interested in all of the plots created by the default plotting method. To view only those plots of interest to you, call `plot` with the argument `ask = T`. This call brings up a menu listing the available plots:

```
> par(mfrow=c(1,1))
> plot(ozone.lm, id.n=5, ask=T)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Residuals vs Fitted Values
3: plot: Sqrt of abs(Residuals) vs Fitted Values
4: plot: Response vs Fitted Values
5: plot: Normal QQplot of Residuals
6: plot: r-f spread plot
7: plot: Cook's Distances
Selection:
Enter the number of the desired plot.
```

If you want to view all the plots, but want them all to appear in a full graphics window, do not set `par(mfrow = c(2,3))` before calling `plot`, and do not use the `ask = T` argument. Instead, before calling `plot`, call `par(ask = T)`. This tells S-PLUS to prompt you before displaying each additional plot.

MULTIPLE REGRESSION

You can construct linear models involving more than one predictor as easily in S-PLUS as models with a single predictor. In general, each predictor contributes a single *term* in the model formula; a single term may contribute more than one coefficient to the fit.

For example, consider the built-in data sets `stack.loss` and `stack.x`. Together, these data sets contain information on ammonia loss in a manufacturing process. The `stack.x` data set is a matrix with three columns representing three predictors: air flow, water temperature, and acid concentration. The `stack.loss` data set is a vector containing the response. To make our computations easier, combine these two data sets into a single data frame, then attach the data frame:

```
> stack.df <- data.frame(stack.loss, stack.x)
> stack.df

   stack.loss Air.Flow Water.Temp Acid.Conc.
1          42       80         27         89
2          37       80         27         88
3          37       75         25         90
. . .

> attach(stack.df)
```

For multivariate data, it is usually a good idea to view the data as a whole using the pairwise scatter plots generated by the `pairs` function:

```
> pairs(stack.df)
```

The resulting plot is shown in Figure 8.3.

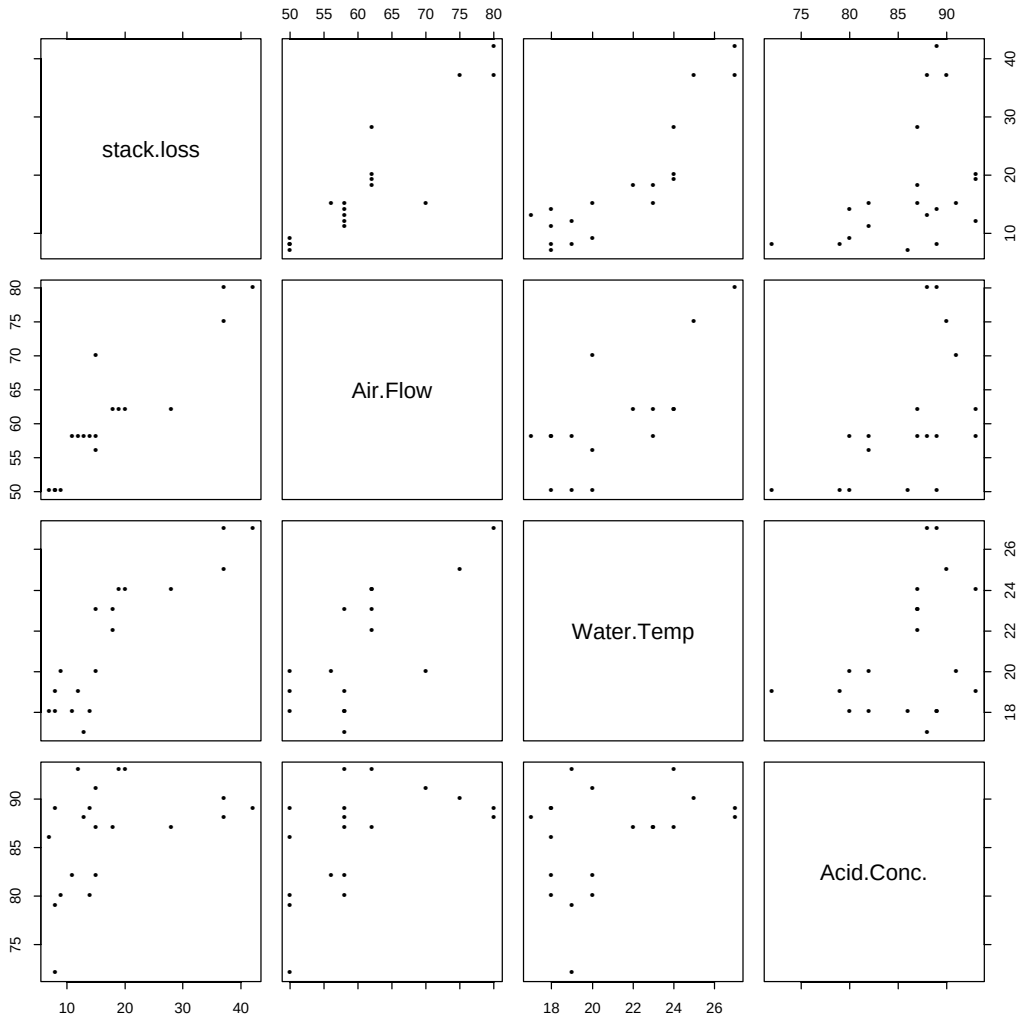

Figure 8.3: *Pairwise scatter plots of stack loss data.*

Call `lm` as follows to model `stack.loss` as a linear function of the three predictors:

```
> stack.lm <- lm(stack.loss ~ Air.Flow + Water.Temp +
+ Acid.Conc.)
```

```
> summary(stack.lm)

Call: lm(formula = stack.loss ~ Air.Flow + Water.Temp +
Acid.Conc.)
Residuals:
   Min     1Q  Median    3Q   Max
-7.238 -1.712 -0.4551 2.361 5.698

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept) -39.9197  11.8960    -3.3557   0.0038
   Air.Flow   0.7156   0.1349     5.3066   0.0001
 Water.Temp   1.2953   0.3680     3.5196   0.0026
 Acid.Conc.  -0.1521   0.1563    -0.9733   0.3440

Residual standard error: 3.243 on 17 degrees of freedom
Multiple R-Squared: 0.9136
F-statistic: 59.9 on 3 and 17 degrees of freedom, the
p-value is 3.016e-09

Correlation of Coefficients:
           (Intercept) Air.Flow Water.Temp
  Air.Flow  0.1793
Water.Temp -0.1489     -0.7356
Acid.Conc. -0.9016     -0.3389   0.0002
```

When the response is the first variable in the data frame, as in `stack.df`, and the desired model includes all the variables in the data frame, the name of the data frame itself can be supplied in place of the `formula` and `data` arguments:

```
> lm(stack.df)

Call:
lm(formula = stack.df)

Coefficients:
 (Intercept)  Air.Flow Water.Temp Acid.Conc.
   -39.91967 0.7156402   1.295286 -0.1521225

Degrees of freedom: 21 total; 17 residual
Residual standard error: 3.243364
```

We examine the default plots to assess the quality of the model (see Figure 8.4):

```
> par(mfrow=c(2,3))
> plot(stack.lm, ask=F)
```

Both the line $y = \hat{y}$ and the residuals plots give support to the model. The multiple R^2 and F statistic also support the model. But would a simpler model suffice?

To find out, let's return to the summary of the `stack.lm` model. From the t values, and the associated p-values, it appears that both `Air.Flow` and `Water.Temp` contribute significantly to the fit. But can we improve the model by dropping the `Acid.Conc.` term? We explore this question further in the section Adding and Dropping Terms From a Linear Model.

ADDING AND DROPPING TERMS FROM A LINEAR MODEL

In the section Multiple Regression, we fitted a linear model with three predictors of which only two appeared to be significant. Can we improve the model `stack.lm` by dropping one or more terms?

The `drop1` function takes a fitted model and returns an ANOVA table showing the effects of dropping in turn each term in the model:

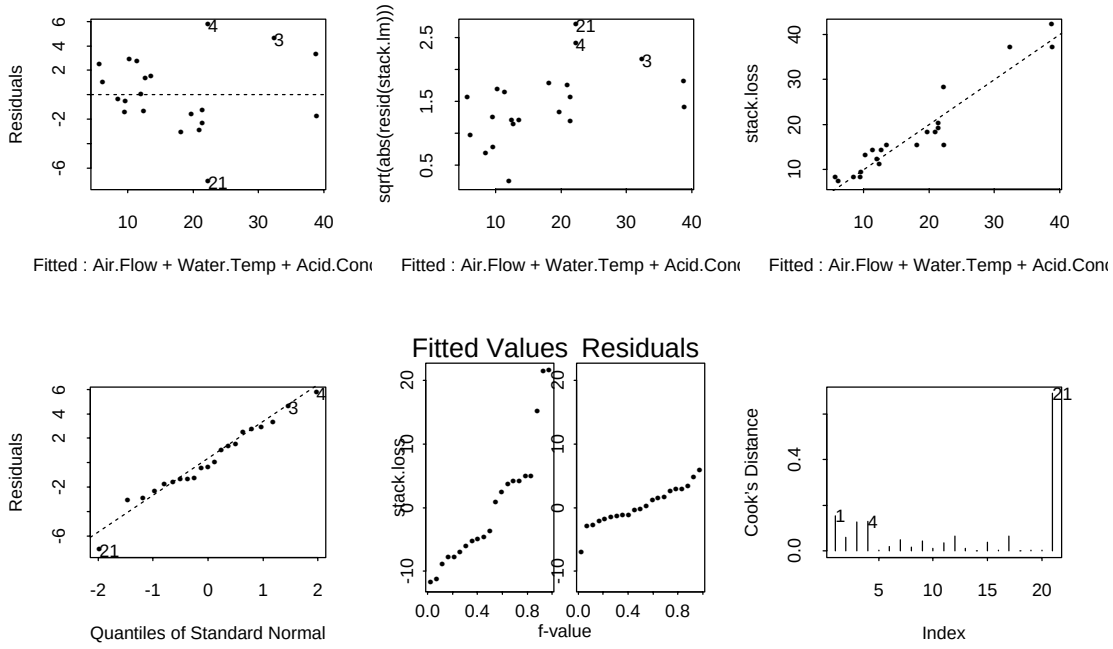

Figure 8.4: *Default plots of fitted model.*

```
> drop1(stack.lm)

Single term deletions
Model:
stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.
           Df Sum of Sq      RSS       Cp
    <none>               178.8300 262.9852
  Air.Flow  1  296.2281 475.0580 538.1745
Water.Temp  1  130.3076 309.1376 372.2541
Acid.Conc.  1    9.9654 188.7953 251.9118
```

The columns of the returned value show the degrees of freedom for each deleted term, the sum of squares corresponding to the deleted term, the residual sum of squares from the resulting model, and the C_p statistic for the terms in the reduced model.

The C_p statistic (actually, what is shown is the AIC statistic, the likelihood version of the C_p statistic–the two are related by the equation $AIC = \hat{\sigma}^2(C_p + n)$) provides a convenient criterion for determining whether a model is improved by dropping a term. If any term has a C_p statistic lower than that of the current model (shown on the line labeled `<none>`), the term with the lowest C_p statistic is dropped. If the current model has the lowest C_p statistic, the model is not improved by dropping any term. The regression literature discusses many other criteria for adding and dropping terms. See, for example, Chapter 8 of Weisberg (1985).

In our example, the C_p statistic shown for `Acid.Conc.` is lower than that for the current model. So it is probably worthwhile dropping that term from the model:

```
> stack2.lm <- lm(stack.loss ~ Air.Flow + Water.Temp)
> stack2.lm

Call:
lm(formula = stack.loss ~ Air.Flow + Water.Temp)

Coefficients:
 (Intercept)  Air.Flow Water.Temp
   -50.35884 0.6711544   1.295351

Degrees of freedom: 21 total; 18 residual
Residual standard error: 3.238615
```

A look at the summary shows that we have retained virtually all the explanatory power of the more complicated model:

```
> summary(stack2.lm)
```

```
Call: lm(formula = stack.loss ~ Air.Flow + Water.Temp)
Residuals:
    Min     1Q Median     3Q    Max
 -7.529 -1.75 0.1894 2.116 5.659

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept) -50.3588   5.1383    -9.8006   0.0000
   Air.Flow   0.6712   0.1267     5.2976   0.0000
 Water.Temp   1.2954   0.3675     3.5249   0.0024

Residual standard error: 3.239 on 18 degrees of freedom
Multiple R-Squared: 0.9088
F-statistic: 89.64 on 2 and 18 degrees of freedom, the
p-value is 4.382e-10

Correlation of Coefficients:
           (Intercept) Air.Flow
  Air.Flow -0.3104
Water.Temp -0.3438     -0.7819
```

The residual standard error has fallen, from 3.243 to 3.239, while the multiple R^2 has decreased only slightly from 0.9136 to 0.9088.

We create the default set of diagnostic plots as follows:

```
> par(mfrow=c(2,3))
> plot(stack2.lm, ask=F)
```

These plots, shown in Figure 8.5, support the simplified model.

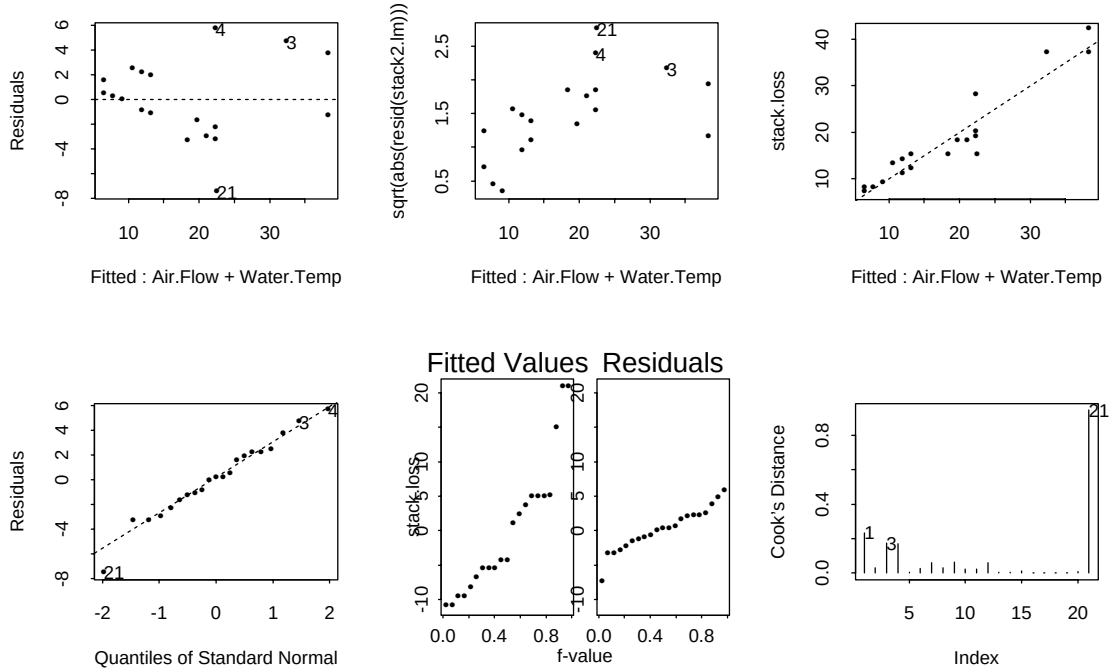

Figure 8.5: *Diagnostic plots for simplified model.*

We turn next to the opposite problem–adding terms to an existing model. Our first linear model hypothesized a relationship between temperature and atmospheric ozone, based on a scatter plot showing an apparent linear relationship between the two variables. The `air` data set containing the two variables `ozone` and `temperature` also includes two other variables, `radiation` and `wind`. Pairwise scatter plots for all the variables can be constructed using `pairs` as follows:

```
> pairs(air)
```

The resulting plot is shown in Figure 8.6. The plot in the top row, third column of Figure 8.6 corresponds to the scatter plot shown in Figure 8.1.

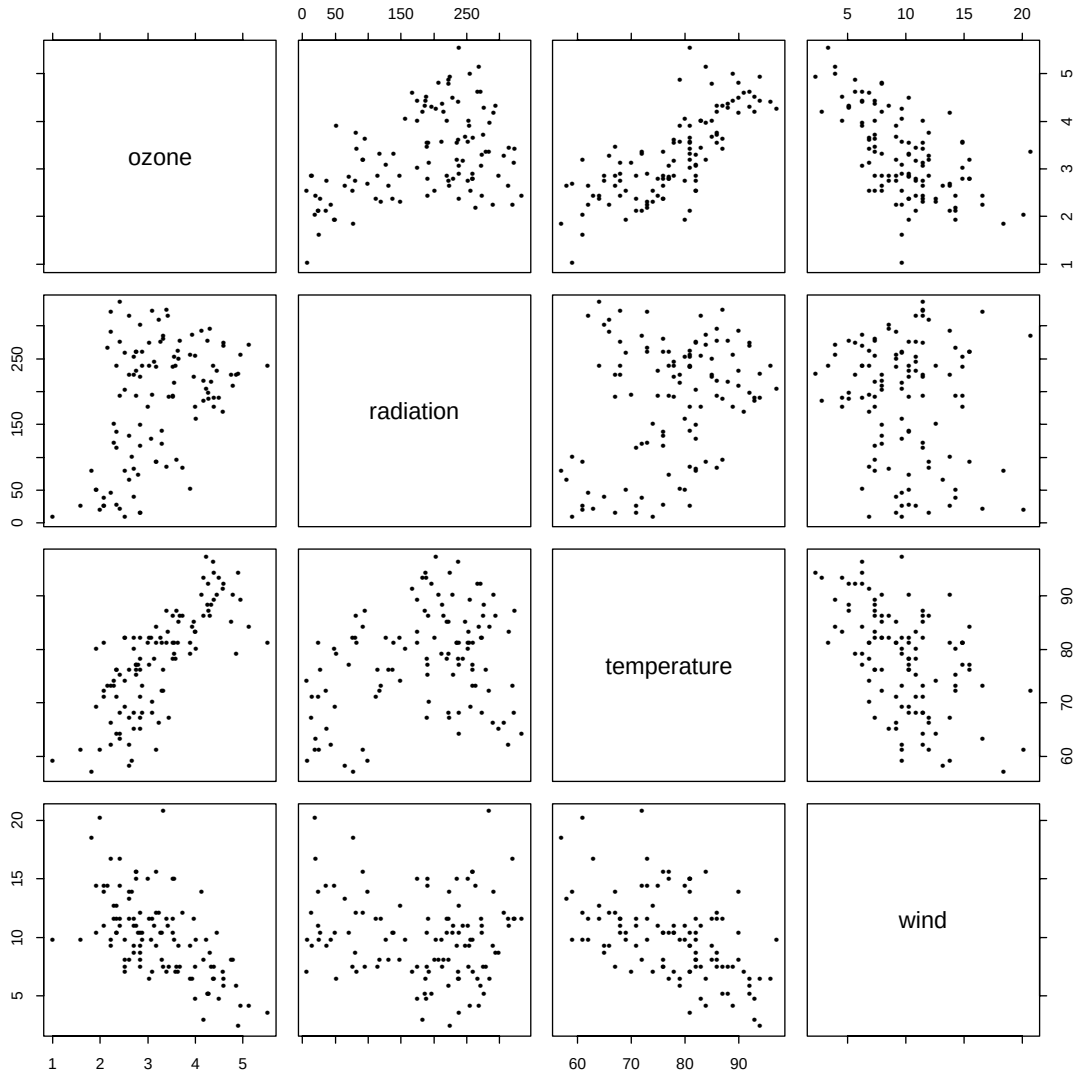

Figure 8.6: *Pairwise scatter plots for* `ozone` *data.*

From the pairwise plots, it appears that the `ozone` varies somewhat linearly with each of the variables `radiation`, `temperature`, and `wind`, and the dependence on `wind` has a negative slope.

We can use the `add1` function to add the terms `wind` and `radiation` in turn to our previously fitted model:

```
> ozone.add1 <- add1(ozone.lm, ~ temperature + wind +
+ radiation)
> ozone.add1

Single term additions

Model:
ozone ~ temperature
           Df Sum of Sq      RSS       Cp
   <none>               37.74698 39.13219
     wind  1  5.839621 31.90736 33.98517
radiation  1  3.839049 33.90793 35.98575
```

The first argument to `add1` is a fitted model object, the second a formula specifying the *scope*, that is, the possible choices of terms to be added to the model. No response need be specified in the formula supplied; the response must be the same as that in the fitted model. The returned object is an ANOVA table like that returned by `drop1`, showing the sum of squares due to the added term, the residual sum of squares of the new model, and the modified C_p statistic for the terms in the augmented model. Each row of the ANOVA table represents the effects of a single term added to the base model. In general, it is worth adding a term if the C_p statistic for that term is lowest among the rows in the table, including the base model term. In our example, we conclude that it is worthwhile adding the `wind` term.

Our choice of `temperature` as the original predictor in the model, however, was completely arbitrary. We can gain a truer picture of the effects of adding terms by starting from a simple intercept model:

```
> ozone0.lm <- lm(ozone ~ 1, data=air)
> ozone0.add1 <- add1(ozone0.lm, ~ temperature + wind +
+ radiation)
```

```
> ozone0.add1

Single term additions

Model:
ozone ~ 1
            Df Sum of Sq      RSS       Cp
     <none>              87.20876 88.79437
temperature  1  49.46178 37.74698 40.91821
       wind  1  31.28305 55.92571 59.09694
  radiation  1  15.53144 71.67732 74.84855
```

The obvious conclusion is that we should start with the `temperature` term, as we did originally.

CHOOSING THE BEST MODEL—STEPWISE SELECTION

Adding and dropping terms using `add1` and `drop1` is a useful method for selecting a model when only a few terms are involved, but it can quickly become tedious. The `step` function provides an automatic procedure for conducting stepwise model selection. Essentially what `step` does is automate the selection process implied in the section Adding and Dropping Terms From a Linear Model–that is, it calculates the C_p statistics for the current model, as well as those for all reduced and augmented models, then adds or drops the term that reduces C_p the most. The `step` function requires an initial model, often constructed explicitly as an intercept-only model, such as the `ozone0.lm` model constructed in the last section. Because `step` calculates augmented models, it requires a `scope` argument, just like `add1`.

For example, suppose we want to find the "best" model involving the stack loss data, we could create an intercept-only model and then call `step` as follows:

```
> stack0.lm <- lm(stack.loss ~ 1, data = stack.df)
> step(stack0.lm, ~ Air.Flow + Water.Temp + Acid.Conc.)

Start:  AIC= 2276.162
 stack.loss ~ 1

Single term additions

Model:
stack.loss ~ 1

scale:  103.4619

            Df Sum of Sq      RSS      Cp
    <none>               2069.238 2276.162
  Air.Flow   1  1750.122  319.116  732.964
Water.Temp   1  1586.087  483.151  896.998
Acid.Conc.   1   330.796 1738.442 2152.290
```

```
Step: AIC= 732.9637
 stack.loss ~ Air.Flow

Single term deletions

Model:
stack.loss ~ Air.Flow

scale:  103.4619

          Df Sum of Sq      RSS       Cp
  <none>               319.116  732.964
Air.Flow  1  1750.122 2069.238 2276.162
Single term additions

Model:
stack.loss ~ Air.Flow

scale:  103.4619

            Df Sum of Sq      RSS       Cp
    <none>               319.1161 732.9637
Water.Temp  1  130.3208 188.7953 809.5668
Acid.Conc.  1    9.9785 309.1376 929.9090
Call:
lm(formula = stack.loss ~ Air.Flow, data = stack.df)

Coefficients:
 (Intercept) Air.Flow
   -44.13202 1.020309

Degrees of freedom: 21 total; 19 residual
Residual standard error (on weighted scale): 4.098242
```

The value returned by `step` is an object of class `lm`, and the final result appears in exactly the same form as the output of `lm`. However, by default, `step` displays the output of each step of the selection process. You can turn off this display by calling `step` with the `trace = F` argument:

```
> step(stack0.lm, ~ Air.Flow + Water.Temp + Acid.Conc.,
+ trace=F)
```

```
Call:
lm(formula = stack.loss ~ Air.Flow, data = stack.df)

Coefficients:
 (Intercept) Air.Flow
   -44.13202 1.020309

Degrees of freedom: 21 total; 19 residual
Residual standard error (on weighted scale): 4.098242
```

UPDATING MODELS

We built our alternate model for the stack loss data by explicitly constructing a second call to `lm`. For models involving only one or two predictors, this is not usually too burdensome. However, if you are looking at many different combinations of many different predictors, constructing the full call repeatedly can be tedious.

The `update` function provides a convenient way for you to fit new models from old models, by specifying an *updated* formula or other arguments. For example, we could create the alternate model `stack2.lm` using `update` as follows:

```
> stack2a.lm <- update(stack.lm, .~. - Acid.Conc.,
+ data=stack.df)
> stack2a.lm

Call:
lm(formula = stack.loss ~ Air.Flow + Water.Temp, data =
stack.df)

Coefficients:
 (Intercept)  Air.Flow Water.Temp
   -50.35884 0.6711544   1.295351

Degrees of freedom: 21 total; 18 residual
Residual standard error: 3.238615
```

The first argument to `update` is always a model object, and additional arguments for `lm` are passed as necessary. The `formula` argument typically makes use of the "`.`" notation on either side of the "~". The "`.`" indicates "as in previous model." The "`-`" and "+" operators are used to delete or add terms. See Chapter 2, Specifying Models in S-PLUS, for more information on formulas with `update`.

WEIGHTED REGRESSION

You can supply weights in fitting any linear model; this can sometimes improve the fit of models with repeated values in the predictor. Weighted regression is the appropriate method in those cases where it is known *a priori* that not all observations contribute equally to the fit.

For example, a software company with a successful training department wanted to estimate the revenue to be generated by an expanded training schedule. For previous courses, the company had the data shown in Table 8.1 concerning the number of courses per month and revenue generated.

Table 8.1: *Revenue and courses per month.*

Courses per Month	Revenue
2	10030
2	7530
2	10801
3	18005
3	15455
3	14986
3	13926
4	16104
4	19166
4	18578
5	27596

We create an S-PLUS data frame from the data in Table 8.1 as follows:

```
> ncourse <- c(2,2,2,3,3,3,3,4,4,4,5)
> revenue <- scan()

 [1] 10030  7530 10801 18005 15455 14986 13926 16104
 [9] 19166 18578 27596

> courserev <- data.frame(revenue, ncourse)
> courserev

   revenue ncourse
 1   10030       2
 2    7530       2
 3   10801       2
 4   18005       3
 5   15455       3
 6   14986       3
 7   13926       3
 8   16104       4
 9   19166       4
10   18578       4
11   27596       5
```

As a first look at the data, we fit a simple regression model as follows:

```
> course.lm <- lm(courserev)
> course.lm

Call:
lm(formula = courserev)

Coefficients:
 (Intercept)  ncourse
   -650.3113 5123.726

Degrees of freedom: 11 total; 9 residual
Residual standard error: 2131.713
```

We then look at the default plots, shown in Figure 8.7:

```
> par(mfrow=c(2,3))
> plot(course.lm)
```

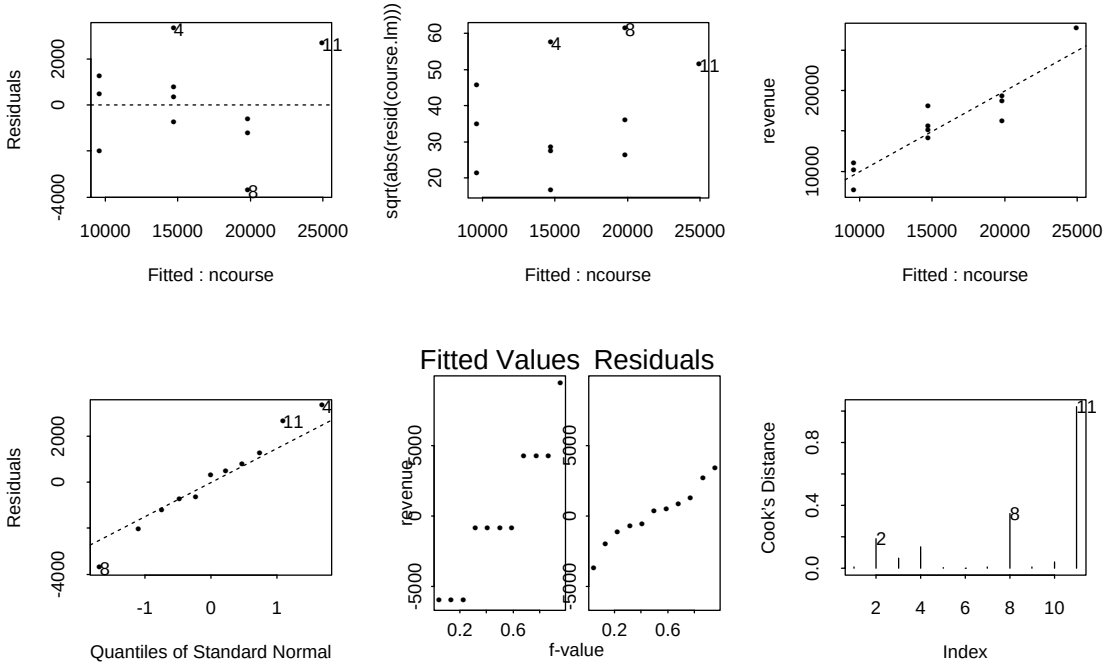

Figure 8.7: *Default diagnostic plots for unweighted course revenue model.*

Overall, the model seems to match the data reasonably well, but the residuals look to be increasing with the predictor. A weighted regression can help in this case.

Weights are specified by supplying a non-negative vector as the `weights` argument to `lm`. We weight with the number of observations for each value of the predictor–this gives a higher weight to the lone observation for 5 courses:

```
> course1.lm <- lm(revenue ~ ncourse, weights=c(1/3, 1/3,
+ 1/3, 1/4, 1/4, 1/4, 1/4, 1/3, 1/3, 1/3, 1))
> course1.lm

Call:
lm(formula = revenue ~ ncourse, weights = c(1/3, 1/3,
1/3, 1/4, 1/4, 1/4, 1/4, 1/3, 1/3, 1/3, 1))
```

```
Coefficients:
 (Intercept)  ncourse
   -2226.167 5678.333

Degrees of freedom: 11 total; 9 residual
Residual standard error (on weighted scale): 1297.942
```

The plots of the weighted regression show again a reasonable fit to the data and a less obvious pattern to the residuals.

PREDICTION WITH THE MODEL

Much of the value of a linear regression model is that, if it accurately models the underlying phenomenon, it can provide reliable *predictions* about the response for a given value of the predictor. The `predict` function takes a fitted model object and a data frame of new data, and returns a vector corresponding to the predicted response. The variable names in the new data must correspond to those of the original predictors; the response may or may not be present, but if present is ignored.

For example, suppose we want to predict the atmospheric ozone concentration from the following vector of temperatures:

```
> newtemp <- c(60, 62, 64, 66, 68, 70, 72)
```

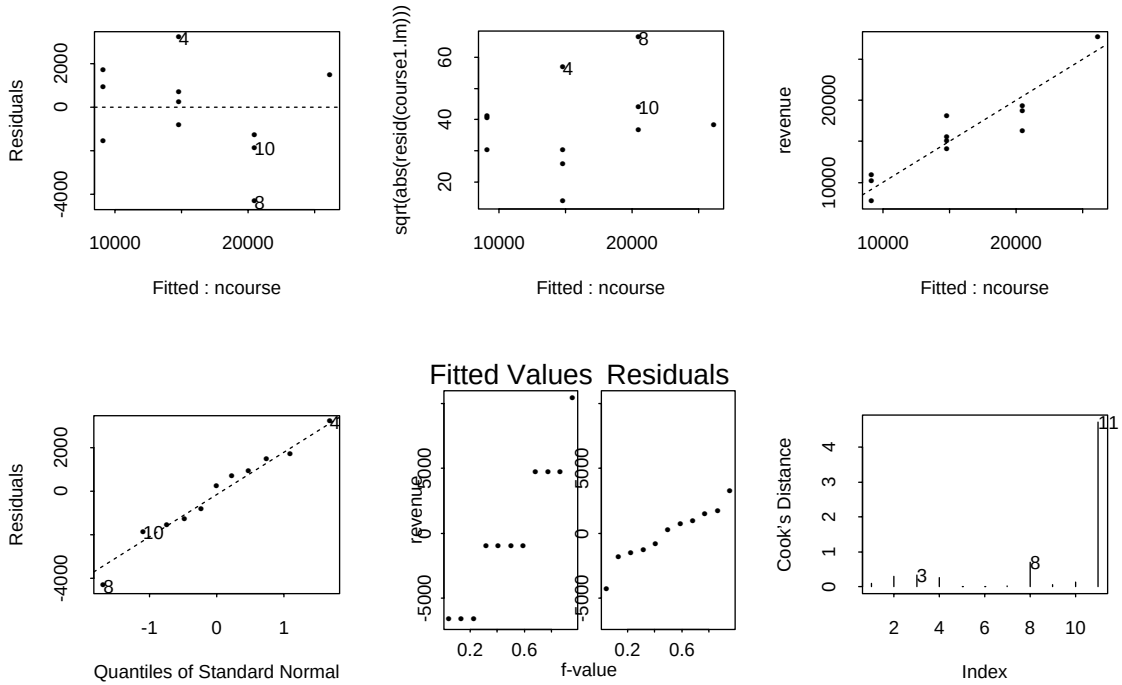

Figure 8.8: *Diagnostic plots for weighted course revenue model.*

We can obtain the desired predictions using `predict` as follows:

```
> predict(ozone.lm, data.frame(temperature=newtemp))

       1        2        3        4        5        6        7
1.995822 2.136549 2.277276 2.418002 2.558729 2.699456 2.840183
```

The predicted values do not stand apart from the original observations.

You can use the `se.fit` argument to `predict` to obtain the standard error of the fitted value at each of the new data points. When `se.fit=T`, the output of `predict` is a list, with a `fit` component containing the predicted values and an `se.fit` component containing the standard errors:

```
> predict(ozone.lm, data.frame(temperature=newtemp),
+ se.fit=T)

$fit:
       1        2        3        4        5        6        7
1.995822 2.136549 2.277276 2.418002 2.558729 2.699456 2.840183

$se.fit:
        1         2         3         4         5         6         7
0.1187178 0.1084689 0.09856156 0.08910993 0.08027508 0.07228355 0.06544499

$residual.scale:
[1] 0.5884748

$df:
[1] 109
```

You can use this output list to compute pointwise and simultaneous confidence intervals for the fitted regression line. See the section Confidence Intervals for details. See the `predict` help file for a description of the remaining components of the return list, `$residual.scale` and `$df`, as well as a description of `predict`'s other arguments.

CONFIDENCE INTERVALS

How reliable is the estimate produced by a simple regression? Provided the standard assumptions hold (that is, normal, identically distributed errors with constant variance σ), we can construct confidence intervals for each point on the fitted regression line based on the *t* distribution, and simultaneous confidence bands for the fitted regression line using the *F* distribution.

In both cases, we need the standard error of the fitted value, `se.fit`, which is computed as follows (Weisberg, 1985, p. 21):

$$\text{se.fit} = \hat{\sigma}\left(\frac{1}{n} + \frac{(x-\bar{x})^2}{\sum_i (x_i-\bar{x})^2}\right)^{\frac{1}{2}}$$

For a fitted object of class `lm`, you can use the `predict` function as follows to calculate `se.fit`:

```
> predict(ozone.lm, se.fit=T)

$se.fit:
         1          2          3         4          5
0.08460301 0.06544499 0.06015393 0.1084689 0.09377002 . . .
```

For any given point *x* in the predictor space, a 1 - α percent confidence interval for the fitted value corresponding to *x* is the set of values *y* such that

$$\hat{y} - t(\alpha,\ n-2) \times \text{se.fit} < y < \hat{y} + t(\alpha,\ n-2) \times \text{se.fi}$$

The `pointwise` function takes the output of `predict` (produced with the `se.fit = T` flag) and returns a list containing three vectors: the vector of lower bounds, the fitted values, and the vector of upper bounds giving the confidence intervals for the fitted values for the predictor:

```
> pointwise(predict(ozone.lm, se.fit=T))
```

```
$upper:
       1        2        3       4        5        6        7
2.710169 3.011759 3.138615 2.42092 2.593475 2.250401 2.363895 . . .

$fit:
       1        2       3         4        5        6        7
2.488366 2.840183 2.98091  2.136549 2.347639 1.925458 2.066185 ...

$lower:
       1        2        3        4        5        6        7
2.266563 2.668607 2.823205 1.852177 2.101803 1.600516 1.768476 ...
```

The output from `pointwise` is suitable, for example, as input for the `error.bar` function. It is tempting to believe that the curves resulting from connecting all the upper points and all the lower points would give a confidence interval for the entire curve. This, however, is not the case; the resulting curve does not have the desired confidence level across its whole range. What is required instead is a *simultaneous* confidence interval, obtained by replacing the t distribution with the F distribution. An S-PLUS function for creating such simultaneous confidence intervals (and by default plotting the result) can be defined as follows:

```
"confint.lm"<-
function(object, alpha = 0.05, plot.it = T, ...)
{
        f <- predict(object, se.fit = T)
        p <- length(coef(object))
        fit <- f$fit
        adjust <- (p * qf(1 - alpha, p, length(fit) -
                p))^0.5 * f$se.fit
        lower <- fit - adjust
        upper <- fit + adjust
        if(plot.it) {
                y <- fit + resid(object)
                plot(fit, y)
                abline(0, 1, lty = 2)
                ord <- order(fit)
                lines(fit[ord], lower[ord])
                lines(fit[ord], upper[ord])
                invisible(list(lower = lower, upper =
                        upper))
        }
        else list(lower = lower, upper = upper)
}
```

To see how it works, let's create a plot of our first model of the ozone data:

```
> confint.lm(ozone.lm)
```

The resulting plot is shown in Figure 8.9.

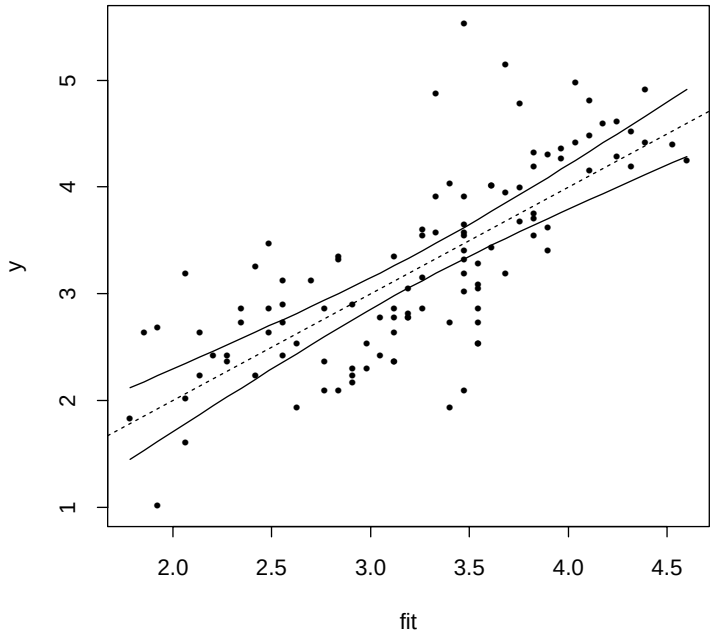

Figure 8.9: *Simultaneous confidence intervals for ozone data.*

POLYNOMIAL REGRESSION

Thus far in this chapter, we've dealt with data sets for which the graphical evidence clearly indicated a linear relationship between the predictors and the response. For such data, the linear model is a natural and elegant choice, providing a simple and easily analyzed description of the data. But what about data that does *not* exhibit a linear dependence? For example, consider the scatter plot shown in Figure 8.10. Clearly, there is *some* functional relationship between the predictor `E` (for Ethanol) and the response `NOx` (for Nitric Oxide), but just as clearly the relationship is not a straight line.

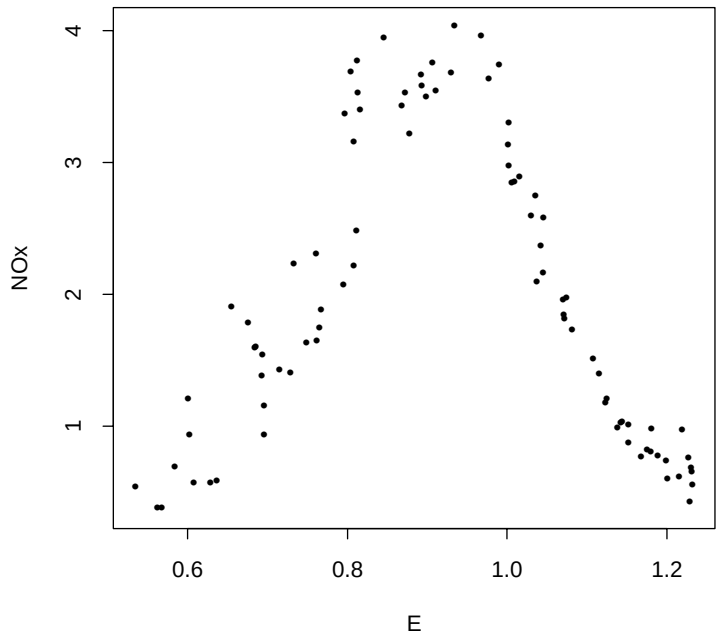

Figure 8.10: *Scatter plot showing nonlinear dependence.*

How should we model such data? One approach is to add polynomial terms to the basic linear model, then use least-squares techniques as before. The classical linear model (with the intercept term represented as the coefficient of a dummy variable X_0 of all 1s) is represented by an equation of the following form:

$$Y = \sum_{k=0}^{n} \beta_k X_k + \varepsilon \tag{8.1}$$

where the predictors X_k enter the equation as linear terms. More generally, classical linear regression techniques apply to any equation of the form

$$Y = \sum_{k=0}^{n} \beta_k Z_k + \varepsilon \tag{8.2}$$

where the Z_k are new variables formed as combinations of the original predictors. For example, consider a cubic polynomial relationship given by the following equation:

$$Y = \sum_{k=0}^{3} \beta_k X^k + \varepsilon \tag{8.3}$$

Taking $X = X_1 = X_2 = X_3$, we can convert this to the desired form by the following assignments:

$$\begin{aligned} X_0 &= Z_0 \\ X_1 &= Z_1 \\ X_2^2 &= Z_2 \\ X_3^2 &= Z_3 \end{aligned}$$

Once these assignments are made, the coefficients β_k can be determined as usual using the classical least-squares techniques.

To perform a polynomial regression in S-PLUS, use `lm` together with the `poly` function. Use `poly` on the right hand side of the `formula` argument to `lm` to specify the independent variable and degree of the polynomial. For example, consider the following made-up data:

```
x <- runif(100, 0, 100)
y <- 50 - 43*x + 31*x^2 - 2*x^3 + rnorm(100)
```

We can fit this as a polynomial regression of degree 3 as follows:

```
> xylm <- lm(y ~ poly(x, 3))
> xylm

Call:
lm(formula = y ~ poly(x, 3))

Coefficients:
(Intercept) poly(x, 3)1 poly(x, 3)2 poly(x, 3)3
  -329798.8    -3681644    -1738826   -333975.4

Degrees of freedom: 100 total; 96 residual
Residual standard error: 0.9463133
```

The coefficients appearing in the object `xylm` are the coefficients for the *orthogonal form* of the polynomial. To recover the simple polynomial form, use the function `poly.transform`:

```
> poly.transform(poly(x,3), coef(xylm))

     x^0       x^1      x^2       x^3
 49.9119 -43.01118 31.00052 -2.000005
```

These coefficients are very close to the exact values used to create `y`.

If the coefficients returned from a regression involving `poly` are so difficult to interpret, why not simply model the polynomial explicitly? That is, why not use the formula `y ~ x + x^2 + x^3` instead of the formula involving `poly`. In our example, there is little difference. However, in problems involving polynomials of higher degree, severe numerical problems can arise in the model matrix. Using `poly` avoids these numerical problems, because `poly` uses an orthogonal set of basis functions to fit the various "powers" of the polynomial.

As a further example of the use of `poly`, let us consider the `ethanol` data we saw at the beginning of this section. From Figure 8.10, we are tempted by a simple quadratic polynomial. However, there is a definite upturn at each end of the data, so we are safer fitting a quartic polynomial, as follows:

```
> ethanol.poly <- lm(NOx ~ poly(E,degree=4))
> summary(ethanol.poly)

Call: lm(formula = NOx ~ poly(E, degree = 4))
Residuals:
     Min       1Q   Median     3Q   Max
 -0.8125 -0.1445 -0.02927 0.1607 1.017

Coefficients:
                       Value Std. Error  t value
         (Intercept)   1.9574   0.0393    49.8407
poly(E, degree = 4)1  -1.0747   0.3684    -2.9170
poly(E, degree = 4)2  -9.2606   0.3684   -25.1367
poly(E, degree = 4)3  -0.4879   0.3684    -1.3243
poly(E, degree = 4)4   3.6341   0.3684     9.8644
                     Pr(>|t|)
         (Intercept)   0.0000
poly(E, degree = 4)1   0.0045
poly(E, degree = 4)2   0.0000
poly(E, degree = 4)3   0.1890
poly(E, degree = 4)4   0.0000
Residual standard error: 0.3684 on 83 degrees of freedom
Multiple R-Squared: 0.8991
F-statistic: 184.9 on 4 and 83 degrees of freedom, the
p-value is 0

Correlation of Coefficients:
(Intercept) poly(E, degree = 4)1
poly(E, degree = 4)1 0
poly(E, degree = 4)2 0           0
poly(E, degree = 4)3 0           0
poly(E, degree = 4)4 0           0
poly(E, degree = 4)2
poly(E, degree = 4)1
poly(E, degree = 4)2
poly(E, degree = 4)3 0
```

```
poly(E, degree = 4)4 0
poly(E, degree = 4)3
poly(E, degree = 4)1
poly(E, degree = 4)2
poly(E, degree = 4)3
poly(E, degree = 4)4 0
```

```
> poly.transform(poly(E,4), coef(ethanol.poly))

      x^0       x^1      x^2       x^3     x^4
 174.3601 -872.2071 1576.735 -1211.219 335.356
```

The summary clearly shows the significance of the 4th order term.

GENERALIZED LEAST SQUARES REGRESSION

Generalized least squares models are regression (or ANOVA) models in which the residuals have a nonstandard covariance structure. Like simple least squares regression, *generalized least squares* regression uses the method of least squares to fit a continuous, univariate response as a linear function of a single predictor variable, but in this case the errors are allowed to be correlated and/or to have unequal variances.

To fit a linear model using generalized least squares with S-PLUS, use the function `gls`. Several arguments are available in `gls`, but typical calls are of the form:

```
gls(model, data, correlation)          # correlated errors
gls(model, data, weights)        # heteroscedastic errors
gls(model, data, correlation, weights)              # both
```

The `model` argument is a two-sided linear formula specifying the model for the expected value of the response just as needed for `lm`, the simple linear model. In many cases, both the response and the predictor are components of a single data frame, which can be specified as the `data` argument to `gls`. The arguments that exemplify the flexibility of `gls` are `correlation` and `weights`.

The optional argument `correlation` is used to specify a correlation structure for the within-group correlation structure if the data are grouped, thus having residuals correlated within these groups. The available correlation structures are organized into a hierarchy of `corStruct` classes, as shown in Table 8.2.

Table 8.2: *Classes of correlation structures.*

Class	Description
corAR1	AR(1)
corARMA	ARMA(p,q)
corCAR1	continuous AR(1)
corCompSymm	compound symmetry
corExp	exponential spatial correlation

Table 8.2: *Classes of correlation structures. (Continued)*

Class	Description
corGaus	Gaussian spatial correlation
corHF	Huyn-Feldt correlation
corLin	linear spatial correlation
corRatio	rational quadratic spatial correlation
corSpher	spherical spatial correlation
corSymm	general correlation matrix

The optional argument `weights` can be used to specify the form of the errors variance-covariance function. The available variance functions are organized into a hierarchy of `varFunc` classes, as shown in Table 8.3.

Table 8.3: *Classes of variance functions.*

Class	Description
varExp	exponential of a variance covariate
varPower	power of a variance covariate
varConstPower	constant plus power of a variance covariate
varIdent	different variances per level of a factor
varFixed	fixed weights, determined by a variance covariate
varComb	combination of variance functions

You can also define your own correlation and variance function classes by specifying appropriate constructor functions and a few method functions. For a new correlation structure, method functions must be defined for at least `corMatrix` and `coef`. For examples of these functions, see the methods for classes `corSymm` and `corAR1`. A

new variance function structure requires methods for at least `coef`, `coef<-`, and `initialize`. For examples of these functions, see the methods for class `varPower`.

Example

The `Ovary` data set has 308 rows and 3 columns giving the number of ovarian follicles detected in different mares at different times in their estrus cycles.

```
> Ovary

Grouped Data: follicles ~ Time | Mare
    Mare        Time follicles
1      1 -0.13636360        20
2      1 -0.09090910        15
3      1 -0.04545455        19
4      1  0.00000000        16
5      1  0.04545455        13
...
305   11  1.00000000         9
306   11  1.05000000         7
307   11  1.10000000         5
308   11  1.15000000         5
```

Biological models suggest that the number of follicles may be modeled as a linear combination of the sin and cosine of `2*pi*Time`.

```
follicles ~ sin(2*pi*Time) + cos(2*pi*Time)
```

Let's fit a simple linear model first and look at the residuals of this fit to demonstrate the need for generalizing the fit to consider their dependences.

```
> Ovary.lm <- lm(follicles ~ sin(2*pi*Time)
+                cos(2*pi*Time), data=Ovary)
> plot(Ovary.lm,which=1)            #Figure 8.11
```

Looking at the plot of the residuals in Figure 8.11 suggests that we try a more general variance-covariance structure for the error term in our model.

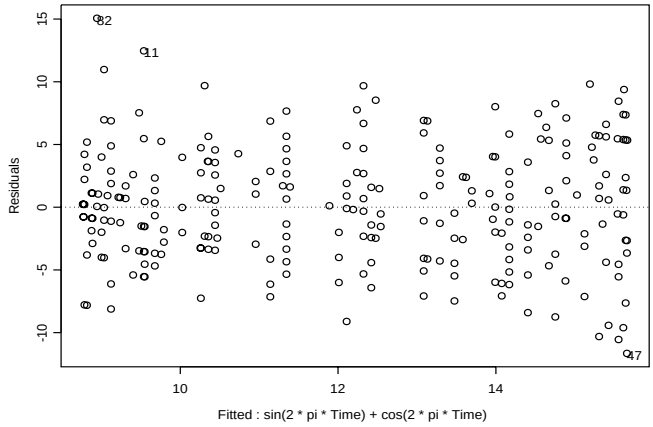

Figure 8.11: *Residuals plot from a simple linear fit to the Ovary dataset.*

We use generalized least squares with a power variance structure (in which variance increases with a power of the absolute fitted values) instead of standard linear regression.

```
> Ovary.fit1 <- gls(
+ follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
+ weights = varPower())
```

Manipulating gls Objects

The fitted objects returned by the `gls` function inherit from class `gls`. A variety of methods are available to display, update, and evaluate the estimation results.

A brief description of the estimation results is returned by the `print` method. For the `Ovary.fit1` object, the results are

```
> print(Ovary.fit1)

Generalized least squares fit by REML
  Model: follicles ~ sin(2*pi*Time) + cos(2*pi*Time)
  Data: Ovary
  Log-restricted-likelihood: -895.8303
```

```
Coefficients:
 (Intercept) sin(2 * pi * Time) cos(2 * pi * Time)
    12.22151          -3.292895          -0.8973728

Variance function:
 Structure: Power of variance covariate
 Formula:  ~ fitted(.)
 Parameter estimates:
     power
 0.4535904
Degrees of freedom: 308 total; 305 residual
Residual standard error: 1.451149
```

A more complete description of the estimation results is returned by the `summary` function.

Diagnostic plots for assessing the quality of the fitted model are obtained using the `plot` method for class `gls`.

```
> plot(Ovary.fit1)                    #Figure 8.12
```

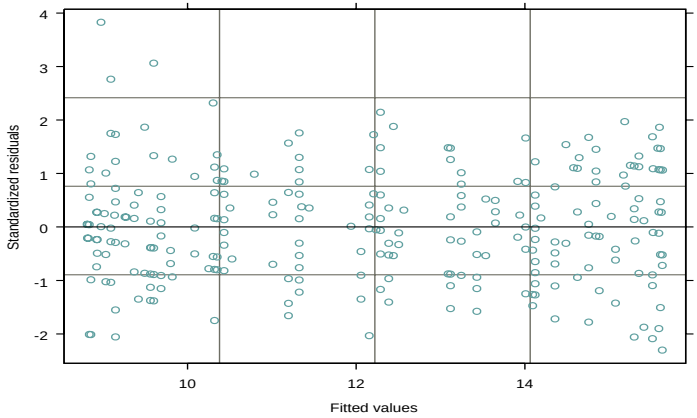

Figure 8.12: *Residuals plot from Generalized Least Squares fit to Ovary data with power variance function.*

The plot in Figure 8.12 seems to still have a lot of extra variation. One possibility, given that `time` is a covariate in the data, is that there is some serial correlation present within the groups. To test this hypothesis, we use the function `ACF` as follows:

```
> ACF(Ovary.fit1)

  lag         ACF
1   0 1.00000000
2   1 0.66042648
3   2 0.55104825
4   3 0.44108949
...
```

These are the corresponding values of the empirical autocorrelation function for the residuals of the `gls` fit. The values are listed for several lags and there appears to be significant autocorrelation at the first few lags. These values can then be plotted with a simple call to the `plot` method for the result of `ACF`.

```
> plot(.Last.value)            #Figure 8.13
```

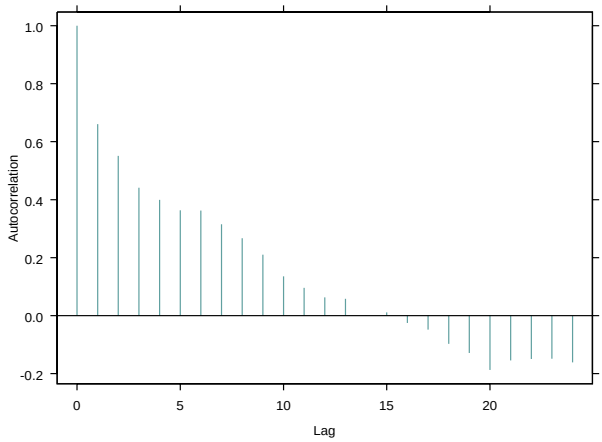

Figure 8.13: *Empirical autocorrelation function corresponding to the standardized residuals of the Ovary.fit1 model object.*

The resulting plot (Figure 8.13) suggests that an Auto Regressive process of order 1 may be adequate to model the serial correlation in the residuals. We use the argument `correlation` to re-fit the model using a correlation structure for the residuals of an AR(1). The value of the first-lag correlation given by ACF is used as an estimate of the autoregressive coefficient.

```
> Ovary.fit2 <- gls(follicles ~ sin(2*pi*Time) +
+ cos(2*pi*Time), Ovary,
+ correlation = corAR1(0.66),
+ weights=varPower())
> plot(Ovary.fit2)
> anova(Ovary.fit1,Ovary.fit2)

           Model df      AIC      BIC    logLik   Test
L.Ratio p-value
Ovary.fit1     1  5 1801.661 1820.262 -895.8303
Ovary.fit2     2  6 1598.496 1620.818 -793.2479 1 vs 2
205.1648  <.0001
```

The residuals plotted in Figure 8.14 look much tighter than for `Ovary.fit1`, indicating that the extra variation we had observed was adequately modeled. The `anova` table comparing the two fits shows great improvement when the serial correlation is considered in the fit.

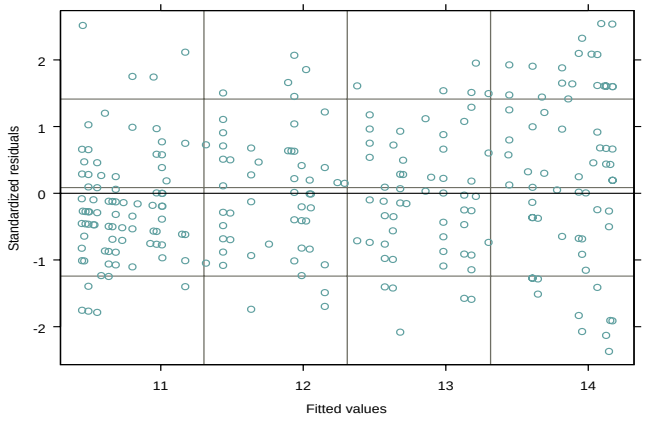

Figure 8.14: *Residuals plot from Generalized Least Squares fit to Ovary data with power variance function and Within-group AR(1) serial correlation.*

The final Generalized Least Squares model for the Ovary data then is:

```
> Ovary.fit2

Generalized least squares fit by REML
  Model: follicles ~ sin(2 * pi * Time) + cos(2 * pi * Time)
  Data: Ovary
  Log-restricted-likelihood: -793.2479

Coefficients:
 (Intercept) sin(2 * pi * Time) cos(2 * pi * Time)
    12.30864          -1.647776          -0.8714635

Correlation Structure: AR(1)
 Parameter estimate(s):
       Phi
 0.7479559
```

```
Variance function:
 Structure: Power of variance covariate
 Formula:  ~ fitted(.)
 Parameter estimates:
      power
 -0.7613254
Degrees of freedom: 308 total; 305 residual
Residual standard error: 32.15024
```

SMOOTHING

Polynomial regression can be useful in many situations. However, the choice of terms is not always obvious, and small effects can be greatly magnified or lost completely by the wrong choice. Another approach to analyzing nonlinear data, attractive because it relies on the data to specify the form of the model, is to fit a curve to the data points *locally*, so that at any point the curve at that point depends only on the observations at that point and some specified neighboring points. Because such a fit produces an estimate of the response that is less variable than the original observed response, the result is called a *smooth*, and procedures for producing such fits are called *scatterplot smoothers*. S-PLUS offers a variety of scatterplot smoothers:

- `loess.smooth` a *locally weighted regression* smoother.
- `smooth.spline` a cubic smoothing spline, with local behavior similar to that of kernel-type smoothers.
- `ksmooth` a kernel-type scatterplot smoother.
- `supsmu` a very fast variable span bivariate smoother.

Halfway between the global parameterization of a polynomial fit and the local, nonparametric fit provided by smoothers are the parametric fits provided by *regression splines*. Regression splines fit a continuous curve to the data by piecing together polynomials fit to different portions of the data. Thus, like smoothers, they are *local* fits. Like polynomials, they provide a parametric fit. In S-PLUS, regression splines can be used to specify the form of a predictor in a linear or more general model, but are not intended for top-level use.

Locally Weighted Regression Smoothing

In locally weighted regression smoothing, we build the smooth function $s(x)$ pointwise as follows:

1. Take a point, say x_0. Find the k nearest neighbors of x_0, which constitute a neighborhood $N(x_0)$. The number of neighbors k is specified as a percentage of the total number of points. This percentage is called the *span*.

2. Calculate the largest distance between x_0 and another point in the neighborhood:

$$\Delta(x_0) = \max_{N(x_0)} |x_0 - x_1|$$

3. Assign weights to each point in $N(x_0)$ using the tri-cube weight function:

$$W\left(\frac{|x_0 - x_1|}{\Delta(x_0)}\right)$$

where

$$W(u) = \begin{cases} (1-u^3)^3 & \text{for } 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

4. Calculate the weighted least squares fit of y on the neighborhood $N(x_0)$. Take the fitted value $\hat{y}_0 = s(x_0)$.

5. Repeat for each predictor value.

Use the `loess.smooth` function to calculate a locally weighted regression smooth. For example, suppose we want to smooth the `ethanol` data. The following expressions produce the plot shown in Figure 8.15:

```
> plot(E,NOx)
> lines(loess.smooth(E,NOx))
```

The plot shown in Figure 8.15 shows the default smoothing, which uses a span of 2/3. For most uses, you will want to specify a smaller span, typically in the range of 0.3 to 0.5.

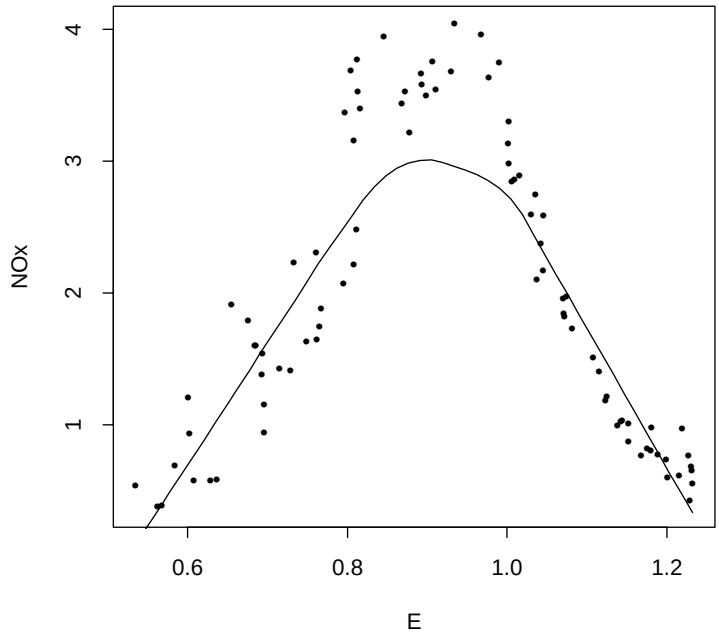

Figure 8.15: *Loess-smoothed* `ethanol` *data.*

Using the Super Smoother

With `loess`, the span is constant over the entire range of predictor values. However, a constant value will not be optimal if either the error variance or the curvature of the underlying function f varies over the range of x. An increase in the error variance requires an increase in the span whereas an increase in the curvature of f requires a decrease. Local cross-validation avoids this problem by choosing a span for the predictor values x_j based on only the leave-one-out residuals whose predictor values x_i are in the neighborhood of x_j. The super smoother, `supsmu`, uses local cross-validation to choose the span. Thus, for one-predictor data, it can be a useful adjunct to `loess`.

For example, Figure 8.16 shows the result of super smoothing the response `NOx` as a function of `E` in the `ethanol` data (dotted line) superimposed on a `loess` smooth. To create the plot, use the following commands:

```
> scatter.smooth(E,NOx, span=1/4)
> lines(supsmu(E,NOx), lty=2)
```

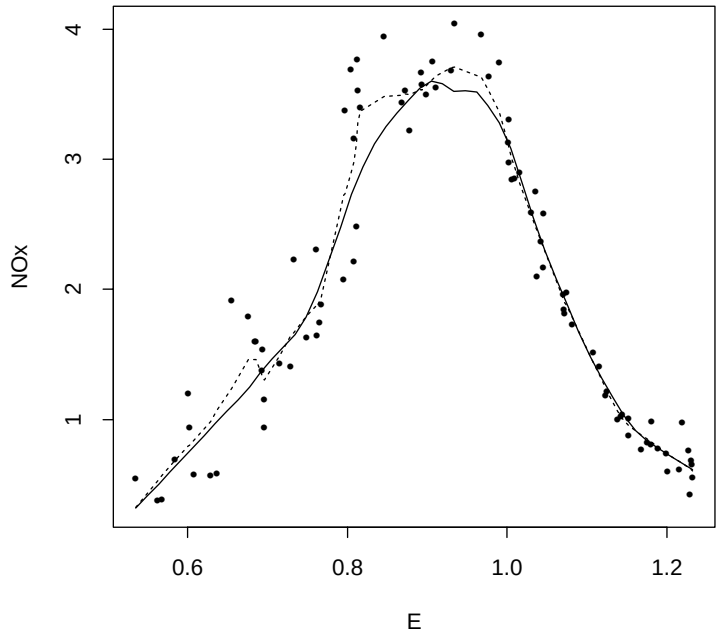

Figure 8.16: *Super smoothed* `ethanol` *data (dotted line).*

Local Cross-Validation

Let $s(x|k)$ denote the linear smoother value at x when span k is used. We wish to choose $k = k(X)$ so as to minimize the mean squared error

$$e^2(k) = E_X Y[Y - s(X|k)]^2$$

where we are considering the joint random variable model for (X, Y). Since

$$E_X Y[Y - s(X|k)]^2 = E_X E_{Y|X}[Y - s(X|k)]^2$$

we would like to choose $k = k(x)$ to minimize

$$\begin{aligned} e_x^2(k) &= E_{Y|X = x}[Y - s(X|k)]^2 \\ &= E_{Y|X = x}[Y - s(x|k)]^2 \end{aligned}.$$

However, we have only the data (x_i, y_i), $i = 1, \ldots, n$, and not the true conditional distribution needed to compute $E_{Y|X=x}$, and so we cannot calculate $e_x^2(k)$. Thus we resort to cross-validation and try to minimize the cross-validation estimate of $e_x^2(k)$:

$$\hat{e}_{CV}^2(k) = \sum_{i=1}^{n} [y_i - s_{(i)}(x_i|k)]^2 .$$

Here $s_{(i)}(x_i|k)$ is the "leave-one-out" smooth at x_i, that is, $s_{(i)}(x_i|k)$ is constructed using all the data (x_j, y_j), $j = 1, \ldots, n$, *except* for (x_i, y_i), and then the resultant local least squares line is evaluated at x_i thereby giving $s_{(i)}(x|k)$. The *leave-one-out residuals*

$$r_{(i)}(k) = y_i - s_{(i)}(x_i|k)$$

are easily obtained from the ordinary residuals

$$r_i(k) = y_i - s(x_i|k)$$

using the standard regression model relation

$$r_{(i)}(k) = \frac{r_i(k)}{h_{ii}} .$$

Here h_{ii}, $i = 1, \ldots, n$, are the diagonals of the so-called "hat" matrix, $H = X(X^T X)^{-1} X^T$, where, for the case at hand of local straight-line regression, X is a 2-column matrix.

Using the Kernel Smoother

A kernel-type smoother is a type of local average smoother that, for each *target point xi* in predictor space, calculates a weighted average $\hat{y}_i$ of the observations in a neighborhood of the target point:

$$\hat{y}_i = \sum_{j=1}^{n} w_{ij} y_j \qquad (8.4)$$

where

$$w_{ij} = \tilde{K}\left(\frac{x_i - x_j}{b}\right) = \frac{K\left(\frac{x_i - x_j}{b}\right)}{\sum_{k=1}^{n} K\left(\frac{x_i - x_k}{b}\right)} .$$

are weights which sum to one:

$$\sum_{j=1}^{n} w_{ij} = 1 .$$

The function K used to calculate the weights is called a *kernel* function, which typically has the following properties:

- (a) $K(t) \geq 0$ for all t
- (b) $\int_{-\infty}^{\infty} K(t)dt = 1$
- (c) $K(-t) = K(t)$ for all t (symmetry)

Note that properties (a) and (b) are those of a probability density function. The parameter b is the *bandwidth* parameter, which determines how large a neighborhood of the target point is used to calculate the local average. A large bandwidth generates a smoother curve, while a small bandwidth generates a wigglier curve. Hastie and Tibshirani (1990), point out that the choice of bandwidth is much more important than the choice of kernel. To perform kernel smoothing in S-PLUS, use the `ksmooth` function. The kernels available in `ksmooth` are shown in Table 8.4.

Table 8.4: *Kernels available for* `ksmooth`.

Kernel	Explicit Form
"box"	$K_{\text{box}}(t) = \begin{cases} 1, & \lvert t\rvert \le 0.5 \\ 0, & \lvert t\rvert > 0.5 \end{cases}$
"triangle"[1]	$K_{\text{tri}}(t) = \begin{cases} 1-\lvert t\rvert/C, & \lvert t\rvert \le \frac{1}{C} \\ 0, & \lvert t\rvert > \frac{1}{C} \end{cases}$
"parzen"[2]	$K_{\text{par}}(t) = \begin{cases} (k_1 - t^2)/k_2, & \lvert t\rvert \le C_1 \\ (t^2/k_3) - k_4\lvert t\rvert + k_5, & C_1 < \lvert t\rvert \le C_2 \\ 0, & C_2 < \lvert t\rvert \end{cases}$
"normal"	$K_{\text{nor}}(t) = (1/\sqrt{2\pi}k_6)\ \exp(-t^2/2k_6^2)$

[1] In convolution form, $K_{\text{tri}}(t) = K_{\text{box}} * K_{\text{box}}(t)$

[2] In convolution form, $K_{\text{par}}(t) = K_{\text{tri}} * K_{\text{box}}(t)$

The constants shown in the explicit forms above are used to scale the resulting kernel so that the upper and lower quartiles occur at ±.25. Also, the is taken to be 1 and the dependence of the kernel on the bandwidth is suppressed.

Of the available kernels, the default "box" kernel gives the crudest smooth. For most data, the other three kernels yield virtually identical smooths. We recommend "triangle" because it is the simplest and fastest to calculate.

The intuitive sense of the kernel estimate $\hat{y}_i$ is clear: Values of y_j such that x_j is close to x_i get relatively heavy weights, while values of y_j such that x_j is far from x_i get small or zero weight. The bandwidth parameter b determines the width of $K(t/b)$, and hence controls the size of the region around x_i for which y_j receives relatively large weights. Since bias increases and variance decreases with increasing bandwidth b, selection of b is a compromise between bias and variance in order to achieve small mean squared error. In practice this is usually done by trial and error. For example, we can compute a kernel smooth for the `ethanol` data as follows:

```
> plot(E,NOx)
> lines(ksmooth(E,NOx, kernel="triangle", bandwidth=.2))
> lines(ksmooth(E,NOx, kernel="triangle", bandwidth=.1),
+ lty=2)
> legend(.54,4.1,c("bandwidth=.2", "bandwidth=.1"),
+ lty=c(1,2))
```

The resulting plot is shown in Figure 8.17.

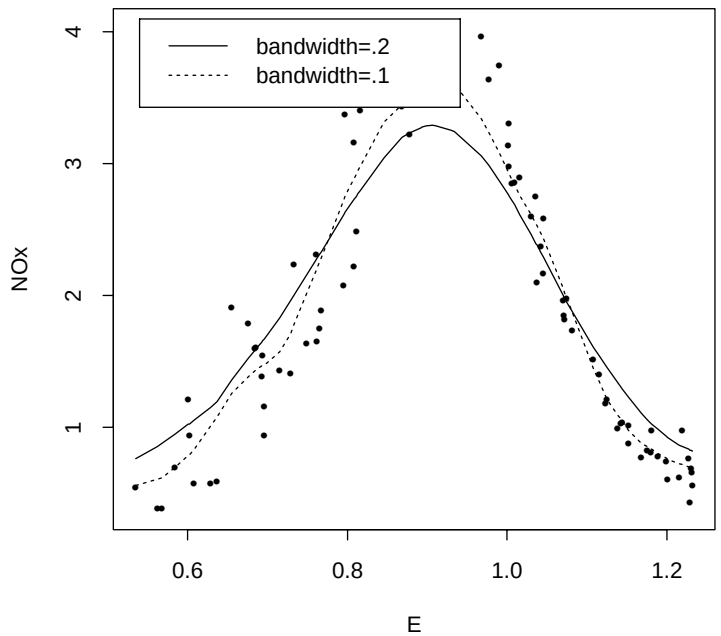

Figure 8.17: *Kernel smooth of* `ethanol` *data for two bandwidths.*

Smoothing Splines

A *cubic smoothing spline* behaves approximately like a kernel smoother, but it arises as the function $\hat{f}$ that minimizes the *penalized residual sum of squares* given by

$$RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int (f''(t))^2 dt$$

over all functions with continuous first and integrable second derivatives. The parameter λ is the smoothing parameter, corresponding to the span in `loess` or `supsmu` or the bandwidth in `ksmooth`.

To generate a cubic smoothing spline in S-PLUS, use the function `smooth.spline` smooth to the input data:

```
> plot(E,NOx)
> lines(smooth.spline(E,NOx))
```

You can specify a different λ using the `spar` argument, although it is not intuitively obvious what a "good" choice of λ might be. When the data is normalized to have a minimum of 0 and a maximum of 1, and when all weights are equal to 1, $\lambda =$ `spar`. More generally, the relationship is given by $\lambda = (\max(x)\text{-}\min(x))^3 \cdot \text{mean}(w) \cdot$ `spar`. You should either let S-PLUS choose the smoothing parameter, using either ordinary or generalized cross-validation, or supply an alternative argument, `df`, which specifies the *degrees of freedom* for the smooth. For example, to add a smooth with approximately 5 degrees of freedom to our previous plot, use the following:

```
> lines(smooth.spline(E,NOx, df=5), lty=2)
```

The resulting plot is shown in Figure 8.18.

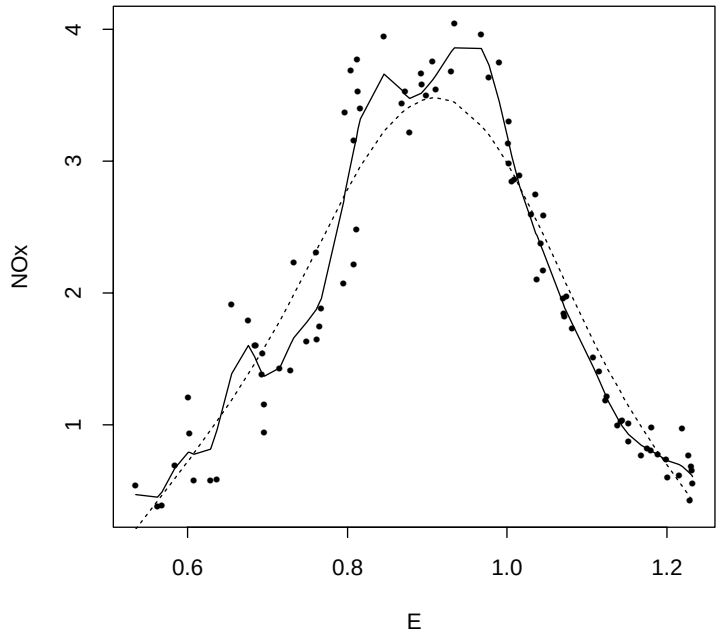

Figure 8.18: *Smoothing spline of* `ethanol` *data with cross-validation (solid line) and pre-specified degrees of freedom.*

Comparing Smoothers

The choice of a smoother is somewhat subjective. All the smoothers discussed in this section can generate reasonably good smooths; you might select one or another based on theoretical considerations or the ease with which one or another of the smoothing criteria can be applied. For a direct comparision of these smoothers, consider the artificial data constructed as follows:

```
> set.seed(14) # set the seed to reproduce this example
> e <- rnorm(200)
> x <- runif(200)
> y <- sin(2*pi*(1-x)2)+x*e
```

A "perfect" smooth would recapture the original signal, $f(x) = \sin(2\pi(1\text{-}x)^2)$, exactly. The following commands sort the input and calculate the *exact* smooth:

```
> sx <- sort(x)
> fx <- sin(2*pi*(1-sx)2)
```

The following commands create a scatter plot of the original data, then superimpose the exact smooth and smooths calculated using each of the smoothers described in this chapter:

```
> plot(x,y)
> lines(sx,fx)
> lines(supsmu(x,y),lty=2)
> lines(ksmooth(x,y),lty=3)
> lines(smooth.spline(x,y),lty=4)
> lines(loess.smooth(x,y),lty=5)
> legend(0,2,c("perfect", "supsmu", "ksmooth",
+ "smooth.spline", "loess"), lty=1:5)
```

The resulting plot is shown in Figure 8.19.

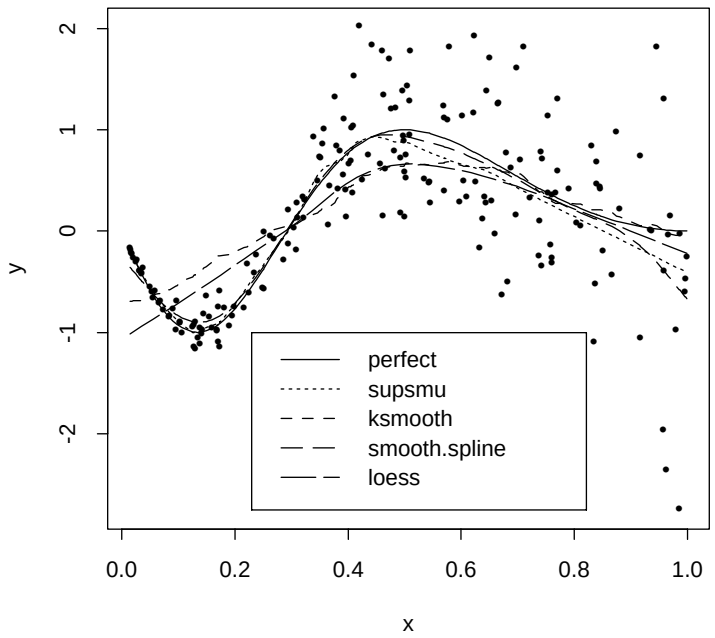

Figure 8.19: *Comparison of S-PLUS smoothers.*

This comparison is crude, at best, because by default each of the smoothers does a different amount of smoothing. A fairer comparison would adjust the smoothing parameters to be roughly equivalent.

ADDITIVE MODELS

An *additive* model extends the notion of a linear model by allowing some or all linear functions of the predictors to be replaced by arbitrary smooth functions of the predictors. Thus, the standard linear model

$$Y = \sum_{i=0}^{n} \beta_i X_i + \varepsilon$$

is replaced by the additive model

$$Y = \alpha + \sum_{i=1}^{n} f_i(X_i) + \varepsilon \quad .$$

The standard linear regression model is a simple case of an additive model. Because the forms of the f_i are generally unknown, they are estimated using some form of scatterplot smoother.

To fit an additive model in S-PLUS, use the `gam` function, where `gam` stands for *generalized additive model*. You provide a formula which may contain ordinary linear terms as well as terms fit using any of the following:

- loess smoothers, using the `lo` function;
- smoothing spline smoothers, using the `s` function;
- natural cubic splines, using the `ns` function;
- *B*-splines, using the `bs` function;
- polynomials, using `poly`.

The three functions `ns`, `bs`, and `poly` result in *parametric* fits; additive models involving only such terms can be analyzed in the classical linear model framework. The `lo` and `s` functions introduce *nonparametric* fitting into the model. For example, the following call takes the `ethanol` data and models the response `NOx` as a function of the loess-smoothed predictor `E`:

```
> attach(ethanol)
```

```
> ethanol.gam <- gam(NOx ~ lo(E, degree=2))
> ethanol.gam

Call:
gam(formula = NOx ~ lo(E, degree = 2))

Degrees of Freedom: 88 total; 81.1184 Residual
Residual Deviance: 9.1378
```

In the call to `lo`, we specify that the smooth is to be locally quadratic by using the argument `degree = 2`. For data that is less obviously nonlinear, we would probably be satisfied with the default, which is locally linear fitting. The printed `gam` object closely resembles a printed `lm` object from linear regression–the call producing the model is shown, followed by the degrees of freedom and the *residual deviance* which serves the same role as the residual sum of squares in the linear model. The deviance is a function of the log-likelihood function, which is related to the probability mass function $f(y_i; \mu_i)$ for the observation y_i given μ_i. The log-likelihood for a sample of n observations is defined as follows:

$$l(m; y) = \sum_{i=1}^{n} \log f(y_i; \mu_i)$$

The deviance $D(y; m)$ is then defined as

$$\frac{D(y; m)}{\phi} = 2l(m^*; y) - 2l(m; y)$$

where μ^* maximizes the log-likelihood over μ unconstrained, and ϕ is the *dispersion parameter*. For a continuous response with normal errors, as in the models we've been considering in this chapter, the dispersion parameter is just the variance σ^2, and the deviance reduces to the residual sum of squares. As with the residual sum of squares, the deviance can be made arbitrarily small by choosing an interpolating solution. As in the linear model case, however, we generally have a desire to keep the model as simple as possible. In the linear case, we try to keep the number of *parameters*, that is, the quantities estimated by the model coefficients, to a minimum.

Additive models are generally *nonparametric*, but we can define for nonparametric models an *equivalent number of parameters*, which we would also like to keep as small as possible.

The equivalent number of parameters for `gam` models is defined in terms of *degrees of freedom*, or `df`. In fitting a parametric model, one degree of freedom is required to estimate each parameter. For an additive model with parametric terms, one degree of freedom is required for each coefficient the term contributes to the model. Thus, for example, consider a model with an intercept, one term fit as a cubic polynomial, and one term fit as a quadratic polynomial. The intercept term contributes one coefficient and requires one degree of freedom, the cubic polynomial contributes three coefficients and thus requires three degrees of freedom, and the quadratic polynomial contributes two coefficients and requires two more degrees of freedom. Thus, the entire model has six parameters, and uses six degrees of freedom. A minimum of six observations is required to fit such a model. Models involving smoothed terms use both *parametric* and *nonparametric* degrees of freedom–parametric degrees of freedom result from fitting a linear (parametric) component for each smooth term, while the nonparametric degrees of freedom result from fitting the smooth after the linear part has been removed. The difference between the number of observations and the degrees of freedom required to fit the model is the *residual degrees of freedom*. Conversely, the difference between the number of observations and the residual degrees of freedom is the degrees of freedom required to fit the model, which is the equivalent number of parameters for the model.

The `summary` method for `gam` objects shows the residual degrees of freedom, the parametric and nonparametric degrees of freedom for each term in the model, together with additional information:

```
> summary(ethanol.gam)

Call: gam(formula = NOx ~ lo(E, degree = 2))
Deviance Residuals:
        Min         1Q      Median        3Q      Max
 -0.6814987 -0.1882066 -0.01673293 0.1741648 0.8479226

(Dispersion Parameter for Gaussian family taken to be
0.1126477 )

Null Deviance: 111.6238 on 87 degrees of freedom
```

```
Residual Deviance: 9.137801 on 81.1184 degrees of freedom

Number of Local Scoring Iterations: 1

DF for Terms and F-values for Nonparametric Effects

                  Df Npar Df   Npar F        Pr(F)
      (Intercept)  1
lo(E, degree = 2)  2     3.9 35.61398 1.110223e-16
```

The `Deviance Residuals` are, for Gaussian models, just the ordinary residuals $y_i - \hat{\mu}_i$. The `Null Deviance` is the deviance of the model consisting solely of the intercept term.

The `ethanol` data set contains a third variable, `C`, which measures the compression ratio of the engine. Figure 8.20 shows pairwise scatter plots for the three variables.

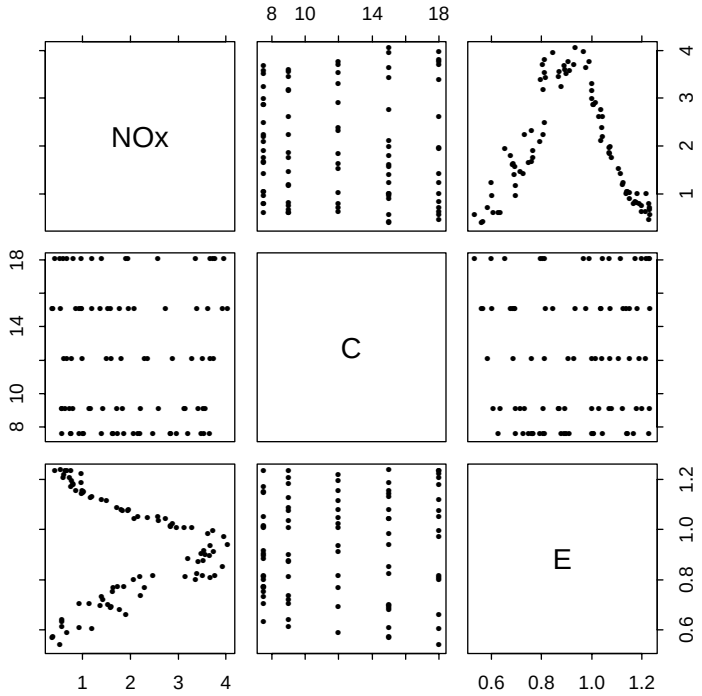

Figure 8.20: *Pairs plot of the* `ethanol` *data.*

Let's incorporate `C` as a linear term in our additive model:

```
> attach(ethanol)
> ethanol2.gam <- gam(NOx ~ C + lo(E, degree = 2))
> ethanol2.gam

Call:
gam(formula = NOx ~ C + lo(E, degree = 2))

Degrees of Freedom: 88 total; 80.1184 Residual
Residual Deviance: 5.16751

> summary(ethanol2.gam)

Call: gam(formula = NOx ~ C + lo(E, degree = 2))
Deviance Residuals:
        Min         1Q     Median         3Q        Max
 -0.6113908 -0.166044 0.0268504 0.1585614 0.4871313

(Dispersion Parameter for Gaussian family taken to be
0.0644985 )

Null Deviance: 111.6238 on 87 degrees of freedom

Residual Deviance: 5.167513 on 80.1184 degrees of freedom

Number of Local Scoring Iterations: 1

DF for Terms and F-values for Nonparametric Effects

                  Df Npar Df   Npar F Pr(F)
      (Intercept)  1
                C  1
lo(E, degree = 2)  2     3.9 57.95895     0
```

We can use the anova function to compare this model with the simpler model involving E only:

```
> anova(ethanol.gam, ethanol2.gam, test="F")

Analysis of Deviance Table

Response: NOx
                   Terms Resid. Df Resid. Dev Test Df
1     lo(E, degree = 2)   81.1184   9.137801
2 C + lo(E, degree = 2)   80.1184   5.167513   +C  1
  Deviance  F Value        Pr(F)
1
2 3.970288 61.55632 1.607059e-11
```

The model involving C is clearly better, since the residual deviance is cut almost in half by expending only one more degree of freedom.

Is the additive model sufficient? Additive models stumble when there are *interactions* among the various terms. In the case of the ethanol data, there is a significant interaction between C and E. In such cases, a full local regression model, fit using the loess function, is often more satisfactory. We discuss the ethanol data more thoroughly in Chapter 11, Local Regression Models.

MORE ON NONPARAMETRIC REGRESSION

The additive models fitted by `gam` in the section Additive Models are simple examples of *nonparametric* regression. The machinery of generalized additive models, proposed by Hastie and Tibshirani (1990), is just one approach to such nonparametric models. S-PLUS includes several other functions for performing nonparametric regression, including the `ace` function, which implements the first proposed technique for nonparametric regression–alternating conditional expectations. S-PLUS also includes AVAS (Additive and VAriance Stabilizing transformations) and projection pursuit regression. This section describes these varieties of nonparametric regression.

Alternating Conditional Expectations

Alternating conditional expectations or `ace`, is an intuitively appealing technique introduced by Breiman and Friedman (1985). The idea is to find nonlinear transformations $\theta(y)$, $\phi_1(x_1)$, $\phi_2(x_2)$, …, $\phi_p(x_p)$ of the response y and predictors x_1, x_2, …, x_p, respectively, such that the *additive model*

$$\theta(y) = \phi_1(x_1) + \phi_2(x_2) + \cdots + \phi_p(x_p) + \varepsilon \tag{8.5}$$

is a good approximation for the data y_i, x_{i1}, …, x_{ip}, $i = 1, \ldots, n$. Let y_i, x_1, x_2, …, x_p be random variables with joint distribution F, and let expectations be taken with respect to F. Consider the *goodness-of-fit* measure

$$e^2 = e^2(\theta, \phi_1, \ldots, \phi_p) = \frac{E\left[\theta(y) - \sum_{k=1}^{p} \phi_k(x_k)\right]^2}{E\theta^2(y)} \tag{8.6}$$

The measure e^2 is the fraction of variance not explained by regressing $\theta(y)$ on $\phi(x_1), \ldots, \phi(x_p)$. The data-based version of e^2 is

$$\hat{e}^2 = \frac{\sum_{i=1}^{n}\left[\hat{\theta}(y_i) - \sum_{k=1}^{p}\hat{\phi}_k(x_{ik})\right]^2}{\sum_{i=1}^{n}\hat{\theta}^2(y_i)} \tag{8.7}$$

where $\hat{\theta}$ and the $\hat{\phi}_j$, estimates of θ and the ϕ_j, are standardized so that $\hat{\theta}(y_i)$ and the $\hat{\phi}_j(x_{ij})$ have mean zero: $\sum_{i=1}^{n}\hat{\theta}(y_i) = 0$ and $\sum_{i=1}^{n}\hat{\phi}_k(x_{ik}) = 0$, $k = 1, \ldots, p$. For the usual linear regression case, where

$$\hat{\theta}(y_i) = y_i - \bar{y}$$

and

$$\hat{\phi}_1(x_{i1} - x_1) = (x_{i1} - x_1)\hat{\beta}_1, \ldots, \hat{\phi}_p(x_{ip} - x_p) = (x_{ip} - x_p)\hat{\beta}_p$$

with $\hat{\beta}_1, \ldots, \hat{\beta}_p$ the least squares regression coefficients, we have

$$\hat{e}^2_{LS} = \frac{RSS}{SSY} \equiv \frac{\sum_{i=1}^{n}\left[(y_i - \bar{y}) - \sum_{k=1}^{p}(x_{ik} - \bar{x}_k)\hat{\beta}_k\right]^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

and the squared multiple correlation coefficient is given by $R^2 = 1 - e_{LS}^2$. The transformations $\hat{\theta}$, $\hat{\phi}_1$, ..., $\hat{\phi}_p$ are chosen to maximize the correlation between $\hat{\theta}(y_i)$ and $\hat{\phi}(x_{i1}) + \cdots + \hat{\phi}(x_{ip})$. Although `ace` is a useful exploratory tool for determining which of the response y and the predictors x_1, ..., x_p are in need of nonlinear transformations and what type of transformation is needed, it can produce anomalous results if errors ε and the $\hat{\phi}_1(x_i)$ fail to satisfy the independence and normality assumptions.

To illustrate the use of `ace`, construct an artificial data set with *additive* errors

$$y_i = e^{1+2x_i} + \varepsilon_i \, i \quad , \text{i=1,...,200}$$

with the ε_i's being $N(0, 10)$ random variables (that is, normal random variables with mean 0 and variance 10), independent of the x_i's, with the x_i's being $U(0, 2)$ random variables (that is, random variables uniformly distributed on the interval from 0 to 2).

```
> set.seed(14) # set the seed to reproduce this example
> x <- 2*runif(200)
> e <- rnorm(200, 0, sqrt(10))
> y <- exp(1+2*x) + e
```

Now use ace:

```
> a <- ace(x,y)
```

Set graphics for 3 x 2 layout of plots:

```
> par(mfrow=c(3,2))
```

Make plots to do the following:

1. Examine original data;
2. Examine transformation of y;
3. Examine transformation of x;
4. Check linearity of the fitted model;
5. Check residuals versus the fit:

The following S-PLUS commands provide the desired plots:

```
> plot(x, y, sub="Original Data")
> plot(x, a$tx, sub="Transformed x vs. x")
> plot(y, a$ty, sub="Transformed y vs. y")
> plot(a$tx, a$ty, sub="Transformed y vs.
+ Continue string: Transformed x")
> plot(a$tx, a$ty - a$tx, xlab="tx",
+ ylab="residuals", sub="Residuals vs. Fit")
```

These plots are displayed in Figure 8.21, where the transformed values $\hat{\theta}(y)$ and $\hat{\phi}(y)$ are denoted by *ty* and *tx*, respectively. The estimated transformation $tx = \hat{\phi}(x)$ seems close to exponential, and except for the small bend at the lower left, the estimated transformation $ty = \hat{\theta}(y)$ seems quite linear. The linearity of the plot of *ty* versus *tx* reveals that a good additive model of the type shown in Equation (8.5) has been achieved. Furthermore, the error variance appears to be relatively constant, except at the very lefthand end. The plot of residuals, $r_i = \hat{\theta}(y_i) - \hat{\phi}(x_i)$ versus the fit $tx = \hat{\phi}(x_i)$ gives a clearer confirmation of the behavior of the residuals' variance.

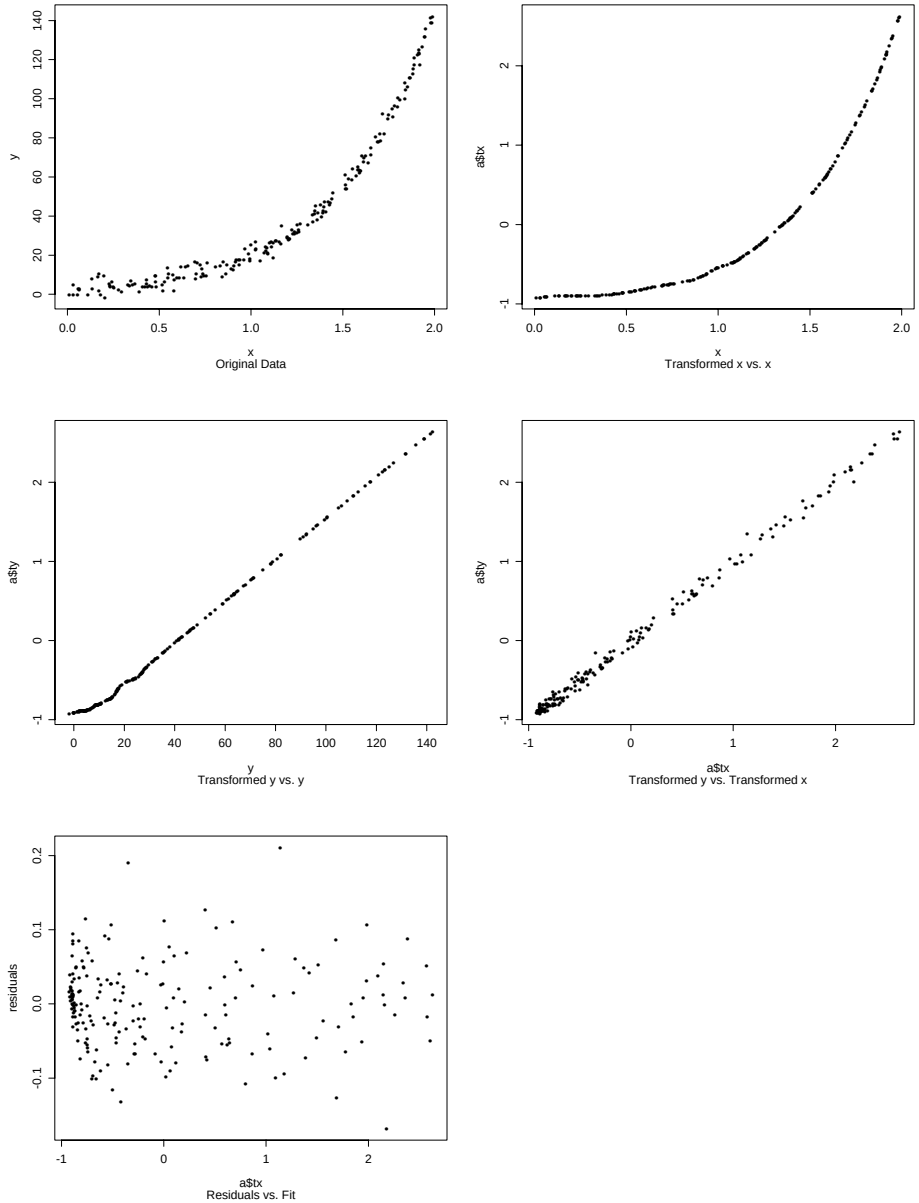

Figure 8.21: `ace` *example with additive errors*

Additive and Variance Stabilizing Transformation

The term "avas" stands for additivity and variance stabilizing transformation. Like ace, avas tries to find transformations $\theta(y)$, $\phi_1(x_1)$, …, $\phi_p(x_p)$ such that

$$\theta(y) = \phi_1(x_1) + \phi_2(x_2) + \cdots + \phi_p(x_p) + \varepsilon \tag{8.8}$$

provides a good *additive* model approximation for the data y_i, x_{i1}, …, x_{ip}, $i = 1, 2, \ldots, n$. However, avas differs from ace in that it chooses $\theta(y)$ to achieve a special *variance stabilizing* feature. In particular the goal of avas is to estimate transformations θ, ϕ_1, …, ϕ_p which have the properties

$$E[\theta(y) | x_1, \ldots, x_p] = \sum_{i=1}^{p} \phi_i(x_i) \tag{8.9}$$

and

$$\text{var}\left[\theta(y) \Big| \sum_{i=1}^{p} \phi_i(x_i)\right] = \text{constant} \tag{8.10}$$

Here $E[z|w]$ is the conditional expectation of z given w. The additivity structure of Equation (8.9) is the same as for ace, and correspondingly the ϕ_i's are calculated by the backfitting algorithm

$$\phi_k(x_k) = E\left[\theta(y) - \sum_{i \neq k} \phi_i(x_i) \Big| x_k\right] \tag{8.11}$$

cycling through $k = 1, 2, \ldots, p$ until convergence. The variance stabilizing aspect comes from Equation (8.9). As in the case of ace, estimates $\hat{\theta}(y_i)$ and $\phi_j(x_{ik})$, $k = 1, 2, \ldots, p$ are computed to approximately satisfy Equation (8.8) through Equation (8.11), with the conditional expectations in Equation (8.8) and Equation (8.11)

estimated using the super smoother scatterplot smoother (see `supsmu` function documentation). The equality (8.9) is approximately achieved by estimating the classic stabilizing transformation.

To illustrate the use of `avas`, construct an artificial data set with *additive* errors

$$y_i = e^{1+2x_i} + \varepsilon_i i, i = 1, ..., 200$$

with the ε_i's being $N(0, 10)$ random variables (that is, normal random variables with mean 0 and variance 10), independent of the x_i's, with the x_i's being $U(0, 2)$ random variables (that is, random variables uniformly distributed on the interval from 0 to 2).

```
> set.seed(14) #set the seed to reproduce this example
> x <- runif(200, 0, 2)
> e <- rnorm(200, 0, sqrt(10))
> y <- exp(1+2*x) + e
```

Now use `avas`:

```
> a <- avas(x, y)
```

Set graphics for a 3 x 2 layout of plots:

```
> par(mfrow=c(3,2))
```

Make plots to: (1) examine original data; (2) examine transformation of x; (3) examine transformation of y; (4) check linearity of the fitted model; (5) check residuals versus the fit:

```
> plot(x, y, sub="Original data")
> plot(x, a$tx, sub="Transformed x vs. x")
> plot(y, a$ty, sub="Transformed y vs. y")
> plot(a$tx, a$ty, sub="Transformed y vs. Transformed x")
> plot(a$tx, a$ty - a$tx, ylab="Residuals",
+ sub="Residuals vs. Fit")
```

These plots are displayed in Figure 8.18 where the transformed values $\hat{\theta}(y)$ and $\hat{\phi}(x)$ are denoted by *ty* and *tx*, respectively. The estimated transformation $tx = \hat{\phi}(x)$ seems close to exponential, and the estimated transformation $ty = \hat{\theta}(y)$ seems linear. The plot of *ty* versus *tx* reveals that a linear additive model holds; that is, we have achieved

a good additive approximation of the type in (8.8). In this plot the error variance appears to be relatively constant. The plot of residuals, $r_i = \hat{\theta}(y_i) - \hat{\phi}(x_i)$, versus the fit $tx = \hat{\phi}(x_i)$ gives further confirmation of this.

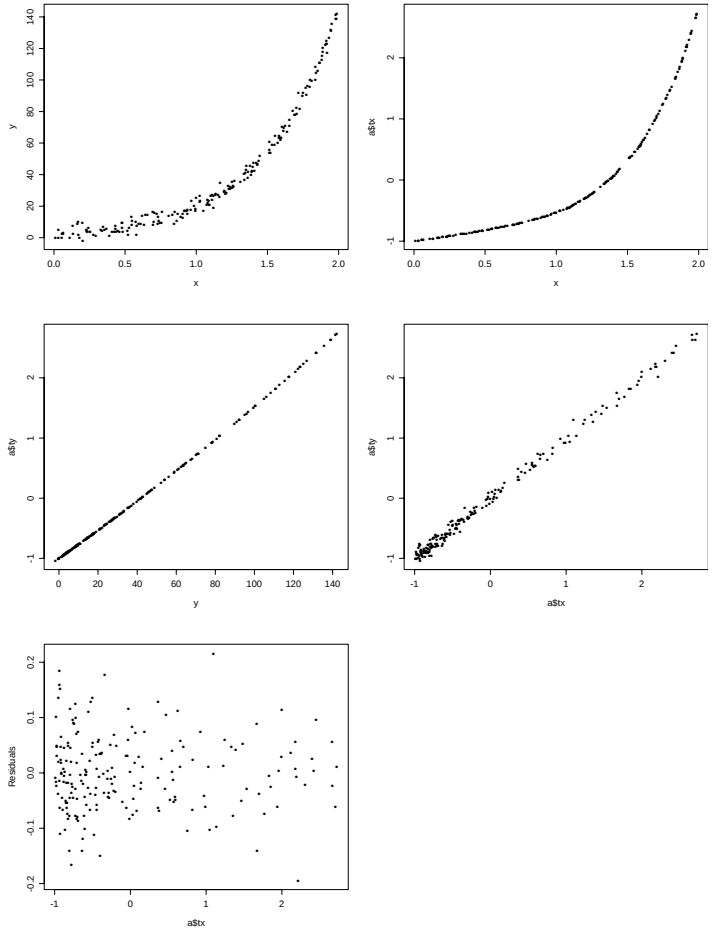

Figure 8.22: `avas` *example with additive errors.*

Key Properties

- Suppose that the true additive model is

$$\theta^0(y) = \sum_{i=1}^{p} \phi_i^0(x_i) + \varepsilon \qquad (8.12)$$

with ε independent of $x_1, x_2, \ldots, x_p$, and var(ε) = constant. Then the iterative `avas` algorithm for (8.9) – (8.11), described below for the data versions of (8.9) – (8.11), yields a sequence of transformations $\theta^{(j)}, \phi_1^{(j)}, \ldots, \phi_p^{(j)}$ which converge to the true transformation $\theta^0, \phi_1^0, \ldots, \phi_p^0$, as the number of iterations j tends to infinity. Correspondingly, the data-based version of this iteration yields a sequence of transformations $\hat{\theta}^{(j)}, \hat{\phi}_1^{(j)}, \ldots, \hat{\phi}_p^{(j)}$, which, at convergence, provide estimates $\hat{\theta}, \hat{\phi}_1, \ldots, \hat{\phi}_p$, of the true model transformations $\theta^0, \phi_1^0, \ldots, \phi_p^0$.

- `avas` appears not to suffer from some of the anomalies of `ace`, for example, not finding good estimates of a true additive model (Equation (8.12)) when normality of ε and joint normality of $\phi_1(x_1), \ldots, \phi_p(x_p)$ fail to hold. See the example below.
- `avas` is a generalization of the Box and Cox (1964) maximum-likelihood procedure for choosing power transformation y^1 of the response. The function `avas` also generalizes the Box and Tidwell (1962) procedure for choosing transformations of the carriers $x_1, x_2, \ldots, x_p$, and is much more convenient than the Box-Tidwell procedure. See also Weisberg (1985).
- $\hat{\theta}(y)$ is a monotone transformation, since it is the integral of a nonnegative function (see the section Further Details on page 240). This is important if one wants to predict y by inverting $\hat{\theta}$: monotone transformations are invertible, and hence we

can predict y with $\hat{y} = \hat{\theta} G^{-1}\left[\sum_{i=1}^{p} \hat{\phi}_i(x_i)\right]$. This predictor has no particular optimality property, but is simply one straightforward way to get a prediction of y once an `avas` model has been fit.

Further Details Let

$$v(u) = \mathrm{VAR}\left[\hat{\theta}(y) \Big| \sum_{i=1}^{p} \phi_i(x_i) = u\right] \tag{8.13}$$

where $\hat{\theta}(y)$ is an arbitrary transformation of y, $\hat{\theta}(y)$ will be the "previous" estimate of $\theta(y)$ in the overall iterative procedure described below. Given the variance function $v(u)$, it is known that

$$\mathrm{VAR}\left[g(\hat{\theta}(y)) \Big| \sum_{i=1}^{p} \phi_i(x_i) = u\right]$$

will be constant if g is computed according to the rule

$$(t) = \int_{c}^{t} \frac{du}{v^{1/2}(u} \tag{8.14}$$

for an appropriate constant c. See Box and Cox (1964).

The detailed steps in the population version of the `avas` algorithm are as follows:

1. *Initialize:*

   Set $(y) = (y - \mathrm{E}y)/(\mathrm{VAR}^{1/2} y$ and backfit on x_1, ..., x_p to get $\hat{\phi}_1$, ..., $\hat{\phi}_p$. (See description of ace for details of "backfitting.")

2. *Get new transformation of y:*

   - Compute variance function:

$$v(u) = \mathrm{VAR}\left[\hat{\theta}(y) \Big| \sum_{i=1}^{p} \hat{\phi}_i(x_i) = u\right]$$

   - Compute variance stabilizing transformation:

$$(t) = \int_c^t \frac{du}{v^{1/2}(u}$$

   - Set $\hat{\theta}(y) - g(\hat{\theta}(y))$ and standardize:

$$\hat{\vartheta}(y) - \frac{\hat{\theta}(y) - \mathrm{E}\hat{\theta}(y)}{\mathrm{VAR}^{1/2}\hat{\theta}(y)}$$

3. *Get new $\hat{\phi}_i$'s:*

   Backfit $\hat{\theta}(y)$ on $x_1, x_2, \ldots, x_p$ to obtain new estimates $\hat{\phi}_1, \ldots, \hat{\phi}_p$.

4. *Iterate steps 2 and 3 until*

$$R^2 = 1 - \hat{e}^2 = 1 - \mathrm{E}\left[\hat{\theta}(y) - \sum_{i=1}^{p} \hat{\phi}_i(x_i)\right]^2 \qquad (8.15)$$

   doesn't change.

Of course the above algorithm is actually carried out using the sample of data $y_i, x_{i1}, \ldots, x_{ip}$, $i = 1, \ldots, n$, with expectations replaced by sample averages, conditional expectations replaced by scatterplot smoothing techniques and VAR's replaced by sample variances.

In particular, super smoother is used in the backfitting step to obtain $\hat{\phi}_1(x_{i1})$, ..., $\hat{\phi}_p(x_{ip})$, $i = 1, ..., n$. An estimate $\hat{v}(u)$ of $v(u)$ is obtained as follows: First the scatter plot of $\log r_i^2 = \log\left[\hat{\theta}(y_i) - \sum_{j=1}^{p}\hat{\phi}_j(x_{ij})\right]^2$ versus $u_i = \sum_{j=1}^{p}\hat{\phi}_j(x_{ij})$ is smoothed using a running straight lines smoother. Then the result is exponentiated. This gives an estimate $\hat{v}(u) \geq 0$, and $\hat{v}(u)$ is truncated below at 10^{-10} to insure positivity and avoid dividing by zero in the integral (8.14). The integration in Equation (8.14) is carried out using a trapezoidal rule.

Projection Pursuit Regression

The basic idea behind projection pursuit regression, `ppreg`, is as follows. Let y and $x = (x_1, x_2, ..., x_p)^T$ denote the response and explanatory vector, respectively. Suppose you have observations y_i and corresponding predictors $x_i = (x_{i1}, x_{i2}, ..., x_{ip})^T$, $i = 1, 2, ..., n$. Let $a_1, a_2, ...,$ denote p-dimensional unit vectors, as "direction" vectors, and let $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. The `ppreg` function allows you to find $M = M_0$, direction vectors $a_1, a_2, ..., a_{M_0}$ and good nonlinear transformations $\phi_1, \phi_2, ..., \phi_{M_0}$ such that

$$y \approx \bar{y} + \sum_{m=1}^{M_0} \beta_m \phi_m(a_m^T x) \tag{8.16}$$

provides a "good" model for the data y_i, x_i, $i = 1, 2, ..., n$. The "projection" part of the term projection pursuit regression indicates that the carrier vector x is *projected* onto the direction vectors $a_1, a_2, ...,$

a_{M_0} to get the lengths $a^T x$, $i = 1, ..., n$ of the projections, and the "pursuit" part indicates that an optimization technique is used to find "good" direction vectors $a_1, a_2, ..., a_{M_0}$.

More formally, y and x are presumed to satisfy the conditional expectation model

$$E[y|x_1, x_2, ..., x_p] = \mu_y + \sum_{m=1}^{M_0} \beta_m \phi_m(a_m^T x) \tag{8.17}$$

where $\mu_y = E(y)$, and the ϕ_m have been standardized to have mean zero and unity variance:

$$E\phi_m(a_m^T x) = 0, \quad E\phi_m^2(a_m^T x) = 0, \quad n = 1, ..., M_0 \tag{8.18}$$

The observations y_i, $x_i = (x_{i1}, ..., x_{ip})^T$, $i = 1, ..., n$, are assumed to be independent and identically distributed random variables like y and x, that is, they satisfy the model in Equation (8.17).

The true model parameters β_m, ϕ_m, a_m, $m = 1, ..., M_0$ in Equation (8.17) minimize the mean squared error

$$E\left[y - \mu_y - \sum_{m=1}^{M_0} \beta_m \phi_m(a_m^T x)\right]^2 \tag{8.19}$$

over all possible β_m, ϕ_m, and a_m.

Equation (8.17) includes the additive `ace` models under the restriction $\theta(y) = y$. This occurs when $M_0 = p$ and $a_1 = (1, 0, ..., 0)^T$, $a_2 = (0, 1, 0, ..., 0)^T$, $a_p = (0, 0, ..., 0, 1)^T$, and the β_m's are absorbed into the ϕ_m's. Furthermore, the ordinary linear model is obtained when $M_0 = 1$,

assuming the predictors x are independent with mean 0 and variance 1. Then $a^T = (b_1, ..., b_p)/\sqrt{b_1^2 + \cdots + b_p^2}$, $\phi_1(t) = t$, and $\beta_1 = \sqrt{b_1^2 + \cdots + b_p^2}$, where the b_j are the regression coefficients.

The projection pursuit model in Equation (8.17) includes the possibility of having *interactions* between the explanatory variables. For example, suppose that

$$E[y|x_1, x_2] = x_1 x_2. \tag{8.20}$$

This is described by (8.17) with $\mu_y = 0$, $M_0 = 2$, $\beta_1 = \beta_2 = \frac{1}{4}$, $a_1^T = (1, 1), a_2^T = (1, -1)$, $\phi_1(t) = t^2$, and $\phi_2(t) = -t^2$. For then

$$\phi_1(a_1^T x) = (x_1 + x_2)^2 = x_1^2 + 2x_1x_2 + x_2^2$$

$$\phi_2(a_2^T x) = -(x_1 - x_2)^2 = -x_1^2 + 2x_1x_2 - x_2^2$$

so that

$$\sum_{m=1}^{2} \beta_m \phi_m(a^T x) = x_1 x_2 .$$

Neither `ace` nor `avas` is able to model interactions. It is this ability to pick up interactions that led to the invention of projection pursuit regression by Friedman and Stuetzle (1981), and it is what makes `ppreg` a useful complement to `ace` and `avas`.

The two variable interactions shown above can be used to illustrate the `ppreg` function. The two predictors, x_1 and x_2 are generated as uniform random variates on the interval -1 to 1. The response, y, is the product of x_1 and x_2 plus a normal error with mean zero and variance 0.04.

```
> set.seed(14)  #set the seed to reproduce this example
> x1 <- runif(400, -1, 1)
> x2 <- runif(400, -1, 1)
> eps <- rnorm(400, 0, .2)
```

```
> y <- x1*x2+eps
> x <- cbind(x1, x2)
```

Now run the projection pursuit regression with `max.term` set at 3, `min.term` set at 2 and with the residuals returned in the `ypred` component (the default if `xpred` is omitted).

```
> p <- ppreg(x, y, 2, 3)
```

Make plots (shown in Figure 8.23) to examine the results of the regression.

```
> par(mfrow=c(3, 2))
> plot(x1, y, sub="Y vs X1")
> plot(x2, y, sub="Y vs X2")
> plot(p$z[,1], p$zhat[,1], sub="1st Term:
+ Continue string: Smooth vs Projection Values z1")
> plot(p$z[,2], p$zhat[,2], sub="2nd Term:
+ Continue string: Smooth vs Projection Values z2")
> plot(y-p$ypred, y, sub="Response vs Fit")
> plot(y-p$ypred, p$ypred, sub="Residuals vs Fit")
```

The first two plots show the response plotted against each of the predictors. It is difficult to hypothesize a function form for the relationship when looking at these plots. The next two plots show the resulting smooth functions from the regression plotted against their respective projection of the carrier variables. Both the plots have a quadratic shape with one being positive and the other negative, the expected result for this type of interaction function. The fifth plot shows clearly a linear relationship between the response and the fitted values. The residuals shown in the last plot do not display any unusual structure.

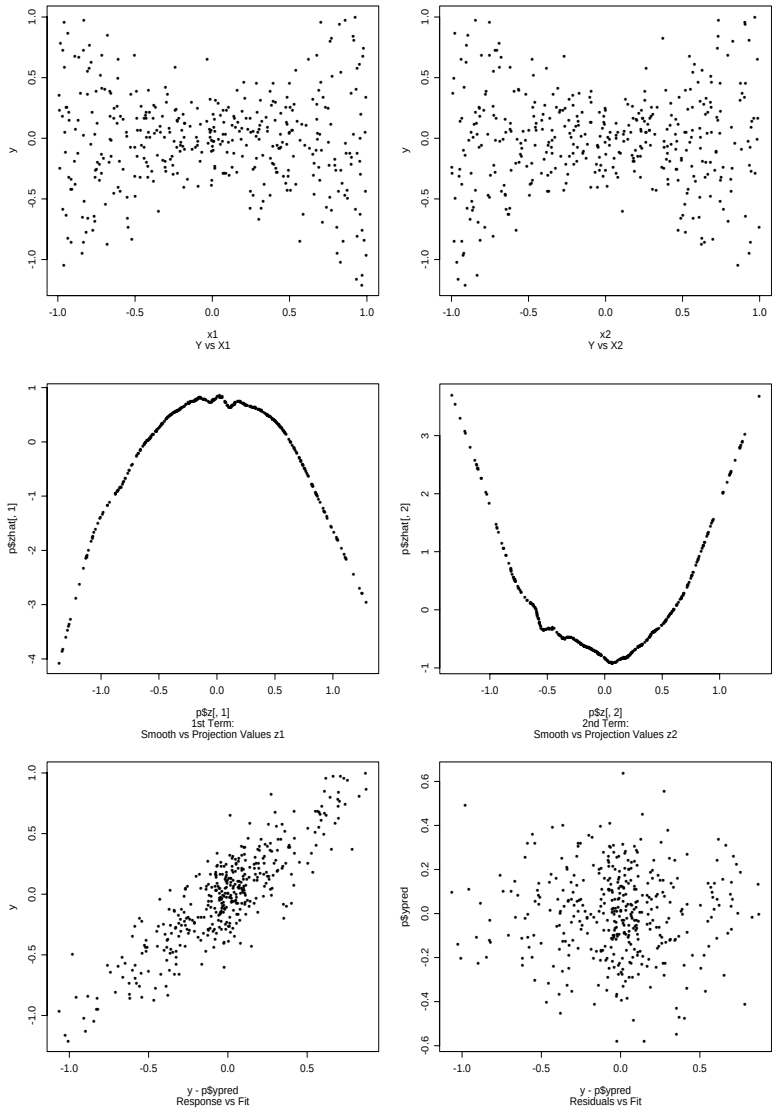

Figure 8.23: *Projection pursuit example.*

Further Details

The forward stepwise procedure

An initial M-term model of the form given by the right-hand side of Equation (8.17), with the constraints of Equation (8.18) and $M > M_0$, is estimated by a forward stepwise procedure, as described by Friedman and Stuetzle (1981).

First, a trial direction a_1 is used to compute the values $z_{i1} = a_1^T x_i$, $i = 1, ..., n$, where $x_i = (x_{i1}, ..., x_{ip})^T$. Then, with $\tilde{y}_i^1 = y_i - \bar{y}$, you have available a scatter plot of data $(\tilde{y}_i, z_{i1})$, $i = 1, ..., n$, which may be smoothed to obtain an estimate $\hat{\phi}_1(z_{i1})$ of the conditional expectation $E[y|z_1] = E[y_i|z_{i1}]$ for the identically distributed random variables y_i, $z_{i1} = a_1^T x_i$. Super Smoother is used for this purpose; see the documentation for `supsmu`. This $\hat{\phi}_1$ depends upon the trial direction vector a_1, so we write $\phi_1 = \phi_{1, a_1}$. Now a_1 is varied to minimize the weighted sum of squares,

$$\sum_{i=1}^{n} w_i [y_i - \hat{\phi}_{1, a_1}(z_{i1})]^2 \tag{8.21}$$

where for each a_1 in the optimization procedure, a new $\hat{\phi}_{1, a_1}$ is computed using super smoother. The weights w_i are user-specified, with the default being all weights unitary: $w_i \equiv 1$. The final results of this optimization will be denoted simply $\hat{a}_1$ and $\hat{\phi}_1$, where $\hat{\phi}_1$ has been standardized according to Equation (8.18) and the corresponding value $\hat{\beta}_1$ is computed. We now have the approximation $y_i \approx \bar{y} + \hat{\beta}_1 \hat{\phi}_1(\hat{a}_1^T x_i)$ $i = 1, ..., n$.

Next we treat $y_i^{(2)} = y_i - \bar{y} - \hat{\beta}_1\hat{\phi}_1(z_{i1})$ as the response, where now $z_{i1} = \hat{a}_1^T x_i$, and fit a second term $\hat{\beta}_2\hat{\phi}_2(z_{i2})$, where $z_{i2} = \hat{a}_2^T x_i$, to this modified response, in exactly the same manner that we fitted $\hat{\beta}_1\hat{\phi}_1(\hat{a}_1^T x_i)$ to $y_i^{(1)}$. This gives the approximation $y_i^{(2)} \approx \hat{\beta}_2\hat{\phi}_2(z_{i2})$ or $y_i \approx \bar{y} + \hat{\beta}_1\hat{\phi}_1(z_{i1}) + \hat{\beta}_2\hat{\phi}_2(z_{i2})$.

Continuing in this fashion we arrive at the forward stepwise estimated model

$$y_i \approx \bar{y} + \sum_{m=1}^{M} \hat{\beta}_m\hat{\phi}_m(z_{im}),\ i = 1, ..., n. \tag{8.22}$$

where $z_{im} = \hat{a}_m^T x_i$, $m = 1, ..., M$.

The backward stepwise procedure

Having fit the M term model in Equation (8.22) in a forward stepwise manner, `ppreg` fits all models of decreasing order $m = M - 1, M - 2, ..., M_{min}$, where M and M_{min} are user-specified. For each term in the model, the weighted sum of squared residuals

$$\mathrm{SSR}(m) = \sum_{i=1}^{n} w_i \left[y_i - \bar{y} - \sum_{l=1}^{m} \beta_l\phi_l(a_l^T x_i) \right]^2 \tag{8.23}$$

is minimized through the choice of β_l, a_l, ϕ_l, $l = 1, ..., m$. The initial values for these parameters, used by the optimization algorithm which minimizes Equation (8.23), are the solution values for the m *most important* out of $m + 1$ terms in the previous order $m + 1$ model. Here *importance* is measured by

$$I_l = \left|\hat{\beta}_l\right| \quad l = 1, ..., m + 1 \tag{8.24}$$

where $\hat{\beta}_l$ are the optimal coefficients for the $m+1$ term model, $m = M - 1, M - 2, \ldots, M_{min}$.

Model selection strategy

In order to determine a "good" number of terms M_0 for the `ppreg` model, proceed as follows. First, run `ppreg` with $M_{min} = 1$ and M set at a value large enough for the data analysis problem at hand. For a relatively small number of variables p, say $p \leq 4$, you might well choose $M \geq p$. For large p, you would probably choose $M < p$, hoping for a parsimonious representation.

For each order m, $1 \leq m \leq M$, `ppreg` will evaluate the fraction of unexplained variance

$$e^2(m) = \frac{\mathrm{SSR}(m)}{\sum_{i=1}^{n} w_i [y_i - \bar{y}]^2}$$

$$= \frac{\sum_{i=1}^{n} w_i \left[y_i - \bar{y} - \sum_{l=1}^{m} \hat{\beta}_l \hat{\phi}_l (\hat{a}_l^T x_i) \right]^2}{\sum_{i=1}^{n} w_i [y_i - \bar{y}]^2}$$

A plot of $e^2(m)$ versus m which is decreasing in m may suggest a good choice of $m = M_0$. Often $e^2(m)$ will decrease relatively rapidly when m is smaller than a good model order M_0 (as the $(\text{bias})^2$ component of prediction mean-squared error is decreasing rapidly), and then tend to flatten out and decrease more slowly for m larger than M_0. You can choose M_0 with this in mind.

The current version of `ppreg` has the feature that when fitting models having $m = M_{min}, M_{min} + 1, \ldots, M$ terms, *all* of the values $\hat{\beta}_l$, $\hat{a}_l$, $\hat{\phi}_l(z_{il})$, $z_{il} = \hat{a}_l^T x_i$, $i = 1, \ldots, n$, $l = 1, \ldots, m$, and $\hat{e}^2(m)$ are returned for $m = M_{min}$, whereas all of these *except* the smoothed values $\hat{\phi}_l(z_{il})$ and their corresponding arguments z_{il} are returned for all $m = M_{min}, \ldots, M$. This feature conserves storage requirements. As a consequence, you must run `ppreg` twice for $m = M_{min}, \ldots, M$, using two different values of M_{min}: The first time $M_{min} = 1$ is used in order to examine $\hat{e}^2(m)$, $m = 1, \ldots, M$ (among other things) and choose a good order M_0. The second time $M_{min} = M_0$ is used in order obtain all output, including $\hat{\phi}_l(z_{il})$ and z_{il} values.

Multivariate response

All of the preceding discussion has been concentrated on the case of a single response y, with observed values $y_1, \ldots, y_n$. In fact, `ppreg` is designed to handle multivariate responses $y_1, \ldots, y_q$ with observed values y_{ij}, $i = 1, \ldots, n, j = 1, \ldots, q$. For this case, `ppreg` allows you to fit a good model

$$y_j \approx \bar{y}_j + \sum_{m=1}^{M_0} \hat{\beta}_{mj} \hat{\phi}_m(\hat{a}_m^T x) \tag{8.25}$$

by minimizing the multivariate response weighted sum of squared residuals

$$\mathrm{SSR}_q(m) = \sum_{j=1}^{q} W_j \sum_{i=1}^{n} w_i \left[y_{ij} - \bar{y}_j - \sum_{l=1}^{m} \hat{\beta}_{lj} \hat{\phi}_l(a_l^T x_i) \right]^2 \tag{8.26}$$

and choosing a good value $m = M_0$.

Here the W_j are user-specified *response* weights (with default $W_j \equiv 1$), the w_i are user-specified *observation* weights (with default $w_i \equiv 1$), and $\bar{y}_j = \frac{1}{n}\sum_{i=1}^{n} y_{ij}$. Note that a single set of $\hat{\phi}_m$'s is used for all responses y_{ij}, $j = 1, \ldots, q$, whereas the different behavior of the different responses is modeled by different linear combinations of the $\hat{\phi}_m$'s by virtue of the different sets of coefficients $\hat{\beta}_j = (\hat{\beta}_{1j}, \ldots, \hat{\beta}_{mj})^T$, $j = 1, \ldots, q$.

The `ppreg` procedure for the multivariate response case is similar to the single response case. For given values of M_{min} and M, `ppreg` first does a forward stepwise fitting starting with a single term ($m = 1$), and ending up with M terms, followed by a backward stepwise procedure stopping with an M_{min}-term model. When passing from an $m + 1$ term model to an m-term model in the multivariate response case, the relative importance of a term is given by

$$I_l = \sum_{j=1}^{q} W_j \left|\hat{\beta}_{jl}\right| \quad l = 1, \ldots, m + 1$$

The most important terms are the ones with the largest I_l, and the corresponding values of $\hat{\beta}_{jl}$, $\hat{\phi}_l$, and a_l are used as initial conditions in the minimization of $SSR_q(m)$. Good model selection; that is, a good choice $m = M_0$, can be made just as in the case of a single response, namely, through examination of the multivariate response fraction of unexplained variation

$$e_q^2(m) = \frac{\mathrm{SSR}_q(m)}{\sum_{j=1}^{q} W_j \sum_{i=1}^{n} w_i [y_{ij} - \bar{y}_j]^2} \tag{8.27}$$

by first using `ppreg` with $M_{min} = 1$ and a suitably large M. Then `ppreg` is run again with $M_{min} = M_0$ and the same large M.

REFERENCES

Box, G.E.P. and Tidwell, P.W. (1962). Transformations of independent variables. *Technometrics*, 4:531-550.

Box, G.E.P. and Cox, D.R. (1964). An analysis of transformations (with discussion). *Journal of the Royal Statistical Society, Series B*, 26:211-246.

Breiman, L. and Friedman, J.H. (1985). Estimating optimal transformations for multiple regression and correlation (with discussion). *Journal of the American Statistical Association*, 80:580-619.

Friedman, J.H. and Stuetzle, W. (1981). Projection pursuit regression. *Journal of the American Statistical Association*, 76:817-823.

Friedman, J.H. (1984). SMART users guide, no. 1. *Laboratory for Computational Statistics, Dept. of Statistics, Stanford University*, Stanford, CA.

Friedman, J.H. (1984). A variable span smoother. Tech. Rept. 5, Laboratory for Computational Statistics, Dept. of Statistics, Stanford University, Stanford, CA.

Friedman, J.H. (1985). Classification and multiple regression through projection pursuit, Tech. Rept. 12, Dept. of Statistics, Stanford University, Stanford, CA.

Hampel, F.R. and Ronchetti, E.M. and Rousseeuw, P.J. and Stahel, W.A. (1986). *Robust Statistics: The Approach Based on Influence Functions.* Wiley, New York.

Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models.* Chapman and Hall, London.

Heiberger, R.M. and Becker, R.A. (1992). Design of an S function for robust regression using iteratively reweighted least squares. *Journal of Computational and Graphical Statistics*, 1:181-196.

Huber, P.J. (1973). *Robust regression: Asymptotics, conjectures, and Monte Carlo.* Annals of Statistics, 1:799-821.

Huber, P.J. (1981). *Robust Statistics.* Wiley, New York.

Rousseeuw, P.J. (1984). Least median of squares regression. *Journal of the American Statistical Association*, 79:871-888.

Rousseeuw, P.J. and Leroy, A.M. (1987). *Robust Regression and Outlier Detection*. Wiley, New York.

Silverman, B.W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London.

Tibshirani, R. (1988). *Estimating transformations for regression via additivity and variance stabilization*. Journal of the American Statistical Association, 83:394-405.

Watson, G.S. (1966). *Smooth regression analysis*. Sankhya, Series A, 26:359-378.

Weisberg, S. (1985). *Applied Linear Regression*, 2nd edition. John Wiley, New York.

ROBUST REGRESSION 9

INTRODUCTION

Robust regression techniques are an important complement to the classical least-squares technique in that they provide answers similar to the least-squares regression when the data are linear with normally distributed errors, but differ significantly from the least-squares fit when the errors don't satisfy the normality conditions or when the data contain significant outliers. S-PLUS includes several robust regression techniques. This chapter focuses on the *Robust MM Regression* technique. We recommend this technique, as it provides both high-quality estimates and a wealth of tools for diagnostics and inference.

Other robust regression techniques available in S-PLUS are least trimmed squares (LTS) regression, least median squares (LMS) regression, least absolute deviations (L1) regression, and M-estimates of regression. These are discussed briefly in the section Other Robust Regression Techniques.

OVERVIEW OF THE ROBUST MM REGRESSION METHOD

This section provides you with an overview of the tools at your disposal for computing a modern robust linear regression model in S-PLUS using robust MM regression, including robust inference for coefficients and robust model selection. You find out how to use the robust regression tools in detail in the sections that follow.

Key Robustness Features of the Method

You will learn how to fit a linear model using a modern robust method that has the following general features:

- In data-oriented terms, the robust fit is minimally influenced by outliers in the independent variables space, in the response (dependent variable) space, or in both.
- In probability-oriented terms, the robust fit minimizes the maximum possible (large sample size) coefficients estimate bias due to a non-Gaussian contamination distribution model which generates outliers, subject to achieving a desired (large sample size) coefficient estimates efficiency when the data has a Gaussian distribution.
- The statistical inference produced by the fit is based on large sample size approximations for such quantities as standard errors and "t-statistics" of coefficients, R-squared values, etc.

For further information, read the section Theoretical Details.

The Essence of the Method: A Special M-Estimate

You are fitting a general linear model of the form

$$y_i = x_i^T \beta + \varepsilon_i, \quad i = 1, \ldots, n$$

with p-dimensional independent predictor (independent) variables x_i and coefficients β, and scalar response (dependent) variable y_i. S-PLUS computes a robust M-estimate $\hat{\beta}$ which minimizes the objective function

$$\sum_{i=1}^{n} \rho\left(\frac{y_i - x_i^T \beta}{\hat{s}}\right)$$

where $\hat{s}$ is a robust scale estimate for the residuals and ρ is a particular optimal symmetric *bounded* loss function, described in the section Theoretical Details. The shape of this optimal function is shown in Figure 9.4.

Alternatively $\hat{\beta}$ is a solution of the estimating equation

$$\sum_{i=1}^{n} x_i \psi\left(\frac{y_i - x_i^T \hat{\beta}}{\hat{s}}\right) = 0$$

where $\psi = \rho'$ is a redescending (nonmonotonic) function.

A key issue is that since ρ is bounded, it is nonconvex, and the minimization above can have many local minima. Correspondingly, the estimating equation above can have multiple solutions. S-PLUS deals with this by computing highly robust inital estimates $\hat{\beta}$ and $\hat{s}$ with breakdown point 0.5, using the S-estimate approach described in the section Theoretical Details, and computes the final estimate $\hat{\beta}$ as the local minimum of the M-estimate objective function nearest to the initial estimate. We refer to an M-estimate of this type and computed in this special way as an MM-estimate, a term introduced by Yohai (1987).

Note

The theory for this new robust method is based on Rousseeuw and Yohai (1984), Yohai, Stahel, and Zamar (1991), and Yohai and Zamar (1998). The code is based on the ROBETH library of Alfio Marazzi, with additional work by R. Douglas Martin, Douglas B. Clarkson, and Jeffrey Wang of MathSoft, partially supported by an SBIR Phase I grant entitled "Usable Robust Methods" funded by the National Institutes of Health.

S-PLUS also provides for an automatic choice between the initial and final estimates based on evaluating the potential bias of the final estimate.

Using the lmRobMM Function to Obtain a Robust Fit

You will compute a robust regression fit using the `lmRobMM` function. The resulting robustly fitted model object is almost identical in structure to a least squares fitted model object returned by `lm`, that is, you will get most of the same fitted model components, such as coefficient standard errors and t-statistics, etc.

Comparison of Least Squares and Robust Fits

In order to facilitate comparison of least squares and robust fits of a linear regression model, you use a special function to create an object with the relevant information from the least squares and robust fits, for example, `t-statistics`, `residuals`, etc. You then use this object as arguments to the usual S-PLUS printing, summarizing and plotting functions to get tabular and graphical displays in a form that makes it easy for you to compare the results of the least squares and robust fits.

Robust Model Selection

It is not enough for you to use a robust linear model fitting method when you are trying to decide which of several alternative models to use, based on alternative sets of predictor variables. You also need a robust model selection criterion. To this end, you may use one of the following three robust model selection criteria: robust F-test, robust Wald test, and robust FPE (RFPE) criterion.

COMPUTING LEAST SQUARES AND ROBUST FITS

Computing a Least Squares Fit

The S-PLUS data frame `oil.df` contains monthly excess returns on the stocks of Oil City Petroleum, Inc. from April 1979 to December 1989 and the monthly excess returns of the market of the same period. "Returns" are defined as the relative change in the price of the stock over a one-month interval, and "excess" means relative to the monthly return at the risk-free rate of a 90-day U.S. Treasury bill.

The scatter plot of the data is shown in Figure 9.1. Obviously there is one big outlier in the data.

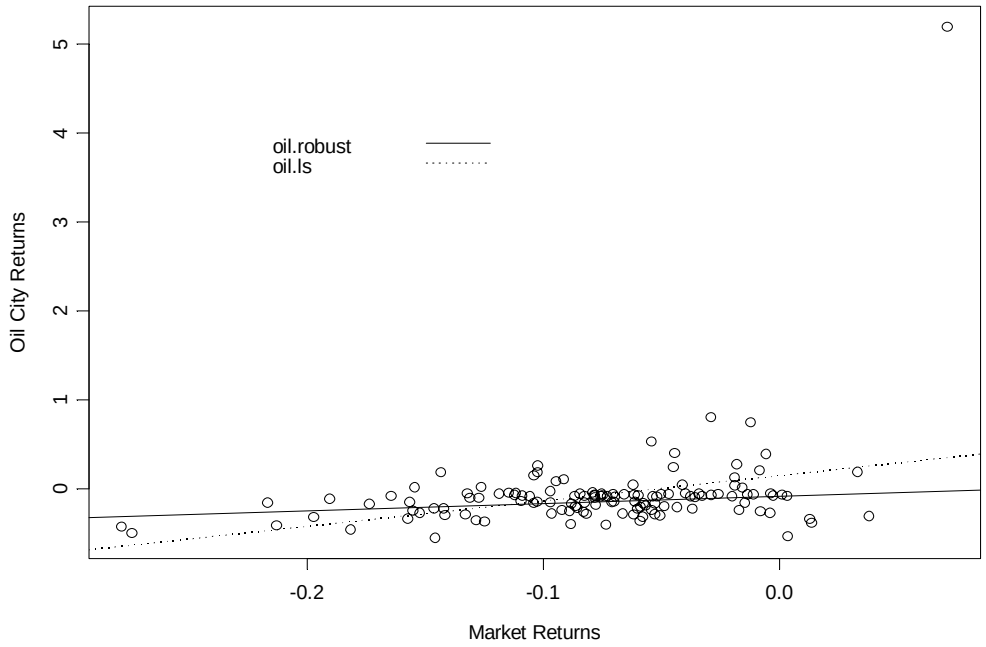

Figure 9.1: *LS fit and robust fit of* `oil.df`.

Financial economists usually use LS to fit a straight line to a particular stock return and the market return, and the estimated coefficient of the market return is called the "beta", which measures the riskiness of the stock in terms of standard deviation and the expected returns. The larger the beta, the more risky the stock is compared with the market, but the larger the expected returns.

For comparison purposes, first fit an LS model to the data as follows:

```
> oil.ls <- lm(Oil~Market,data=oil.ls)
```

and print a short summary of the fitted model:

```
> oil.ls

Call:
lm(formula = Oil ~ Market, data = oil.df)

Coefficients:
 (Intercept)  Market
   0.1474486 2.85674

Degrees of freedom: 129 total; 127 residual
Residual standard error: 0.4866656
```

Computing a Robust Fit

To obtain a robust fit, you use the `lmRobMM` function just like the `lm` function:

```
> oil.robust <- lmRobMM(Oil~Market,data=oil.df)
> oil.robust

Final M-estimates.

Call:
lmRobMM(formula = Oil ~ Market, data = oil.df)

Coefficients:
 (Intercept)    Market
 -0.08395777 0.8288791

Degrees of freedom: 129 total; 127 residual
Residual scale estimate: 0.1446283
```

Obviously, the robust estimate of beta is dramatically different from the LS estimate. According to the LS method, the beta of this stock is 2.857, which implies that the stock is 2.857 times as volatile as the market, and has about 2.857 times the expected return. The robust estimate of beta is 0.829, which implies that the stock has somewhat less volatility and expected return than the market.

Also note that the robust scale estimate is 0.14, whereas the scale estimate from LS is 0.49. The LS scale estimate is based on the sum of squared residuals and thus considerably inflated by the presence of outliers in the data.

Least Squares vs. Robust Fitted Model Objects

The object returned by the `lm` function for LS fit is of class `"lm"`:

```
> class(oil.ls)

[1] "lm"
```

On the other hand, the object returned by `lmRobMM` is of class `"lmRobMM"`:

```
> class(oil.robust)

[1] "lmRobMM"
```

Just as with an object of class "`lm`", you can easily visualize, print and summarize robust fit objects of class "`lmRobMM`" using the generic functions `plot`, `print` and `summary`.

VISUALIZING AND SUMMARIZING THE ROBUST FIT

Visualizing the Fit With the plot Function

For a simple linear regression, you can easily see outliers in the scatter plot, as in the above example. However, in multiple regression it is not so easy to tell if there are some outliers in the data, and what the outliers are. Nonetheless, S-PLUS makes it easy for you to visualize the outliers in a multiple regression. To illustrate this point, let us use the well known "stack loss" data set.

The S-PLUS product includes the stack loss data set which has been analyzed by a large number of statisticians. The stack loss in this data set is the percent loss (times 10) of ammonia during 21 days of operation. The ammonia is lost during the process of producing nitric acid by dissolving the ammonia in water. Three variables–air flow, water temperature, and acid concentration–may influence the loss of ammonia. The stack loss response data is contained in the vector `stack.loss`, and the three independent variables are contained in the matrix `stack.x`.

First, you combine the response and independent variables into a data frame `stack.df`:

```
> stack.df <- data.frame(Loss=stack.loss,stack.x)
```

Then you compute an LS fit object `stack.ls`:

```
> stack.ls <- lm(Loss ~ Air.Flow + Water.Temp +
+ Acid.Conc., data =stack.df)
```

and finally compute a robust fit object `stack.robust`:

```
> stack.robust <- lmRobMM(Loss ~ Air.Flow + Water.Temp +
+ Acid.Conc., data =stack.df)
```

Now you use the `plot` function to visualize the fit:

```
> plot(stack.robust)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Residuals vs Fitted Values
3: plot: Sqrt of abs(Residuals) vs Fitted Values
4: plot: Response vs Fitted Values
```

```
5: plot: Normal QQplot of Residuals
6: plot: r-f spread plot
Selection:
```

Note that Cook's distance is not currently available when a robust method is used.

Now you can compare the plot of residuals versus fitted values for both the LS fit and the robust fit using the following commands:

```
> par(mfrow=c(2,1))
> plot(stack.ls,which.plots=1)
> plot(stack.robust,which.plots=1)
```

Figure 9.2 shows those two plots. As you can see, the robust fit pushes the outliers further away from the majority of the data, so that you can more easily identify the outliers.

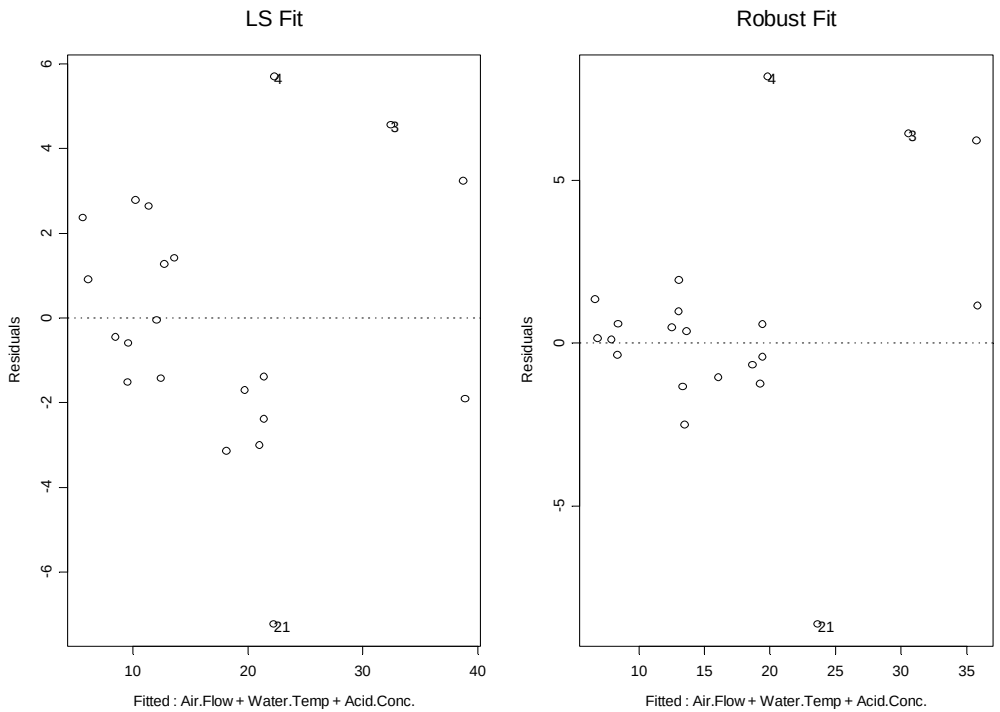

Figure 9.2: *Residuals vs. fitted values:* `stack.loss` *data.*

Statistical Inference With the summary Function

The generic `summary` function provides you with the usual kinds of inference output, for example, t-values and p-values along with some additive and useful information, including tests for bias. For example, to obtain more information about the robust fit `oil.robust`, use `summary` on this object:

```
> summary(oil.robust)

Final M-estimates.

Call: lmRobMM(formula = Oil ~ Market, data = oil.df)

Residuals:
     Min       1Q  Median     3Q    Max
 -0.4566 -0.08875 0.03082 0.1031 5.218

Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept) -0.0840   0.0281    -2.9929  0.0033
     Market  0.8289   0.2834     2.9245  0.0041

Residual scale estimate: 0.1446 on 127 degrees of freedom
Proportion of variation in response explained by model:
0.05261

Test for Bias
            Statistics   P-value
 M-estimate       2.16 0.3398475
LS-estimate      22.39 0.0000138

Correlation of Coefficients:
       (Intercept)
Market 0.8169

The seed parameter is : 1313
```

First note the standard errors, the t-values, and the p-values of the coefficients. The standard errors are computed from the robust covariance matrix of the estimates. For technical details about the computation of robust covariance matrix, refer to Yohai, Stahel and Zamar (1991).

Second, the `summary` method provides another piece of useful information: the "Proportion of variation in response explained by model," usually known as R^2. S-PLUS calculates a robust version of R^2. The details of how the robust R^2 is calculated can be found in the section Theoretical Details.

Finally, there is a "Test for Bias" section in the summary. This section provides the test statistics of the bias of the final M-estimates and the LS estimates against the initial S-estimates. In this case, the test for bias of the final M-estimates yields a p-value of 0.33, which suggests that the bias of the final M-estimates relative to the initial S-estimates is not significant at the usual level. That is why the "Final M-estimates" is reported in the first line of its summary output instead of the initial S-estimates. The test for bias of the LS estimates relative to the S-estimates yields a p-value of 0, which indicates that the LS estimate is highly biased, so you strongly prefer to use the robust MM-estimator.

For technical details about how the tests for bias are calculated, see Yohai, Zamar and Stahel (1991).

COMPARING LEAST SQUARES AND ROBUST FITS

Creating a Comparison Object for LS and Robust Fits

In the section Visualizing the Fit With the plot Function, we compared the residuals vs. fitted values plot for both the LS and robust fits. You might have noted that the two plots do not have the same vertical scale. It would be nice to have the capability of plotting different fits on the same scale for easy visual comparison and also making tabular displays of LS and robust fits which are conveniently aligned for ease of comparing inference results. To this end, S-PLUS provides a function `compare.fits`, for creating a models comparison object, along with appropriate `print`, `plot` and `summary` methods for this class of object.

For example, to compare the results from the two fits `oil.ls` and `oil.robust`, first create the comparison object `oil.cmpr` with the following command:

```
> oil.cmpr <- compare.fits(oil.ls,oil.robust)
```

The object returned by `compare.fits` is of class "`compare.fits`". Now you can print a short summary of the comparison:

```
> oil.cmpr

Calls:
    oil.ls lm(formula = Oil ~ Market, data = oil.df)
oil.robust lmRobMM(formula = Oil ~ Market, data = oil.df)

Coefficients:
            oil.ls oil.robust
(Intercept) 0.1474   -0.08396
     Market 2.8567    0.82888

Residual Scale Estimates:
    oil.ls : 0.4867 on 127 degrees of freedom
oil.robust : 0.1446 on 127 degrees of freedom
```

Visualizing LS vs. Robust Fits

You can easily plot a `compare.fits` object to obtain a visual comparison of the LS and robust fits:

```
> plot(oil.cmpr)

Make a plot selection (or 0 to exit):

1: Normal QQ-Plots of Residuals
2: Estimated Densities of Residuals
3: Residuals vs Fitted Values
4: Response vs Fitted Values
Selection:
```

For example, the normal qq-plot and estimated densities for `oil.cmpr` are shown in Figure 9.3. The densities of residuals are estimated using a kernel type density estimate. For a good model fit, the probability density estimates for the residuals will be centered at zero and nearly as narrow as possible. Figure 9.3 shows that the density of residuals from the LS estimate is shifted to the left of the origin, whereas that of the robust fit is well centered. Furthermore, the outlier bumps in the residual density estimates for the MM-estimator are pushed further from the mode of the density, and thus are a little more pronounced than those for the LS estimates (because there is one big outlier in the data).

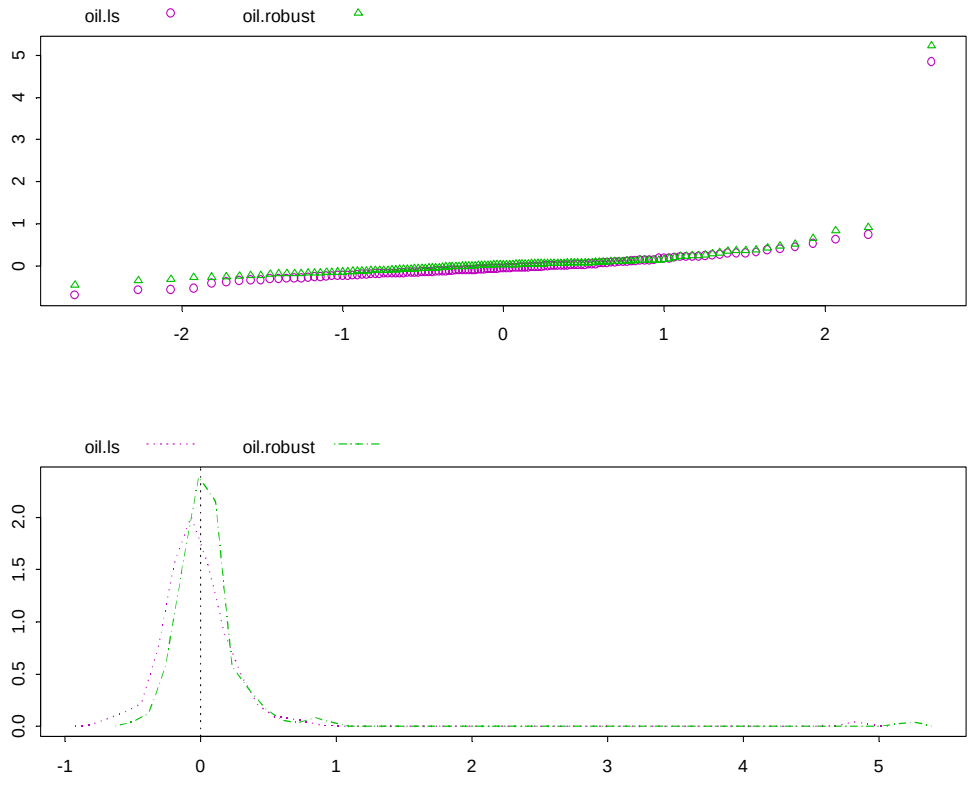

Figure 9.3: *Sample plots of* `oil.cmpr`.

Statistical Inference for LS vs. Robust Fits

A more detailed comparison, particularly comparison of t-values and p-values, can be obtained using the generic `summary` function on a "`compare.fits`" object. For example:

```
> summary(oil.cmpr)

Calls:
    oil.ls lm(formula = Oil ~ Market, data = oil.df)
oil.robust lmRobMM(formula = Oil ~ Market, data = oil.df)

Residual Statistics:
               Min       1Q   Median      3Q   Max
    oil.ls -0.6952 -0.17323 -0.05444 0.08407 4.842
oil.robust -0.4566 -0.08875  0.03082 0.10314 5.218
```

```
Coefficients:
    oil.ls
              Value Std. Error t value  Pr(>|t|)
(Intercept) 0.1474    0.07072   2.085 0.0390860
     Market 2.8567    0.73175   3.904 0.0001528

oil.robust
               Value Std. Error t value Pr(>|t|)
(Intercept) -0.08396    0.02805  -2.993 0.003321
     Market  0.82888    0.28342   2.925 0.004087

Residual Scale Estimates:
    oil.ls : 0.4867 on 127 degrees of freedom
oil.robust : 0.1446 on 127 degrees of freedom

Proportion of variation in response(s) explained by
model(s):
    oil.ls : 0.1071
oil.robust : 0.05261

Correlations:
    oil.ls
                Market
(Intercept) 0.7955736

oil.robust
                Market
(Intercept) 0.8168693
```

Caveat

When the final M-estimate is not used, that is, p-values of test for bias indicates that the final M-estimate is highly biased relative to the initial S-estimates, the asymptotic approximations for the inference may not be very good and you should not trust them very much.

ROBUST MODEL SELECTION

Robust F and Wald Tests

Another important part of statistical inference is hypothesis testing. S-PLUS provides two robust tests for testing whether or not some of the regression coefficients are zero: the robust Wald test and the robust F test. For technical details on how these tests are computed, see the section Theoretical Details below. Before proceeding, create the data frame `simu.dat`:

```
> simu.dat <- gen.data(1:3)
```

where the function `gen.data` is provided in the appendix to this chapter. This function generates a data frame with five columns: y, $x1$, $x2$, $x3$ and $x4$. The variable y is generated according to the following equation:

$$y = b_1 x1 + b_2 x2 + b_3 x3 + u$$

where b_1, b_2, b_3 is given by `1:3` in the above S-PLUS command, and u is sampled from a N(0,3) family with 10% contamination. The term $x4$ is independent of y, $x1$, $x2$, and $x3$. First, you fit a model with $x1$, $x2$, $x3$, and $x4$ as the predictor variables:

```
> simu.mm4 <- lmRobMM(y~x1+x2+x3+x4-1, data=simu.dat)
```

To test the hypothesis that the coefficient of $x4$ is actually zero, you can fit another model with only $x1$, $x2$, and $x3$ as the predictor variables, then use `anova` to test the significance of the coefficient of $x4$:

```
> simu.mm3 <- update(simu.mm4,.~.-x4)
> anova(simu.mm4,simu.mm3)

Response: y
                   Terms     Df       Wald  P(>Wald)
1 x1 + x2 + x3 + x4 - 1
2      x1 + x2 + x3 - 1      1 0.04436687 0.8331725
```

The p-value in this case is greater than 0.8, which leads you to accept the null hypothesis that the fourth coefficient value is zero.

The default test used by `anova` is the Wald test based on robust estimates of the coefficients and covariance matrix (a robust Wald test). To use the robust F test instead, specify the optional argument `test` to `anova`:

```
> anova(simu.mm4,simu.mm3,test="RF")

Response: y
                    Terms     Df    RobustF P(>RobustF)
1 x1 + x2 + x3 + x4 - 1
2      x1 + x2 + x3 - 1      1 0.03374514   0.8507404
```

which gives a quite similar result to that of the robust Wald test.

Robust FPE Criterion

Although many robust estimators have been constructed in the past, the issue of robust model selection has not received its due attention. For robust model selection, S-PLUS provides Robust Final Prediction Errors (RFPE) as a criterion, which is a robust analogue to the classical Final Prediction Errors (FPE) criterion. RFPE is defined as:

$$RFPE = \sum_{i=1}^{n} E\rho\left(\frac{y_i^* - x_i^T\beta^{(1)}}{\rho}\right),$$

where $\beta^{(1)}$ is the final M-estimate of β, y_i^*'s are the values you are trying to predict using $\beta^{(1)}$, and the expectation is taken with respect to both $\beta^{(1)}$ and y_i^*'s. When considering a variety of model choices with respect to different choices of predictor variables, you choose the model with the smallest value of RFPE.

Note that when $\rho(u) = u^2$, RFPE reduces to the classical FPE. RFPE can also be shown to be asymptotically equivalent to the robust version of AIC proposed by Ronchetti (1985). The section Theoretical Details provides a sketch of technical details supporting the use of RFPE.

The RFPE criterion is used as the robust method, invoked by use of the generic functions, of `drop1` and `add1`. For example, use of `drop1` on the robustly fitted model `simu.mm4` in the previous section gives:

```
> drop1(simu.mm4)

Significant test at level  10 %.
  for  x3


Single term deletions

Model:
y ~ x1 + x2 + x3 + x4 - 1
       Df     RFPE
<none>     24.24174
    x1  1  24.46596
    x2  1  52.19800
    x3  1  64.32633
    x4  1  23.95825
```

The output indicates that dropping *x*4 gives a better model.

You can also use `add1` to explore the relevance of other variables. For example, if you fit `simu.mm3` first, you can use the following command to investigate if *x*4 helps predict *y*:

```
> add1(simu.mm3,"x4")

Single term additions

Model:
y ~ x1 + x2 + x3 - 1
       Df     RFPE
<none>    24.10184
    x4  1 24.38769
```

Since addition of *x*4 causes RFPE to increase, addition of *x*4 results in a poor model.

Caveat

If the test for bias of final M-estimates is significant for any of the models considered by drop1 and add1, you should not trust the corresponding RFPE very much.

CONTROLLING OPTIONS FOR ROBUST REGRESSION

In this section, you will learn how to change the default settings of some control parameters for the MM-estimator so as to obtain particular estimates that fit your purpose. Most of the default settings can be changed through the functions `lm.robust.control` and `lm.genetic.control`. Only the commonly used control parameters are introduced in this section. For the default settings of other parameters and how to change them, see the online help file for `lm.robust.control` and `lm.genetic.control`.

Efficiency at Gaussian Model

If the final M-estimates are accepted, they have a default asymptotic efficiency of 85% compared with the LS estimates, when the errors are normally distributed.

Sometimes an asymptotic efficiency of 85% may not be what you exactly want. To change the efficiency of the final M-estimates, the `lmRobMM` optional argument `robust.control` should be generated from `lmRobMM.robust.control` with desired efficiency:

```
> oil.tmp <- lmRobMM(Oil~Market,data=oil.df,
+ robust.control=lmRobMM.robust.control(efficiency=0.95))
> coef(oil.tmp)

 (Intercept)    Market
  -0.07398806  0.8491126
```

Alternative Loss Function

As mentioned in the introduction, the final M-estimates are based on the initial S-estimates of regression coefficients and scale parameter. For both the initial S-estimate and the final M-estimate, S-PLUS uses a loss function for the estimation. Two different loss functions are available in S-PLUS: Tukey's bisquare function and the optimal loss function recently discovered by Yohai and Zamar (1998). Figure 9.4 shows the Tukey bisquare function on the left and the optimal loss function on the right.

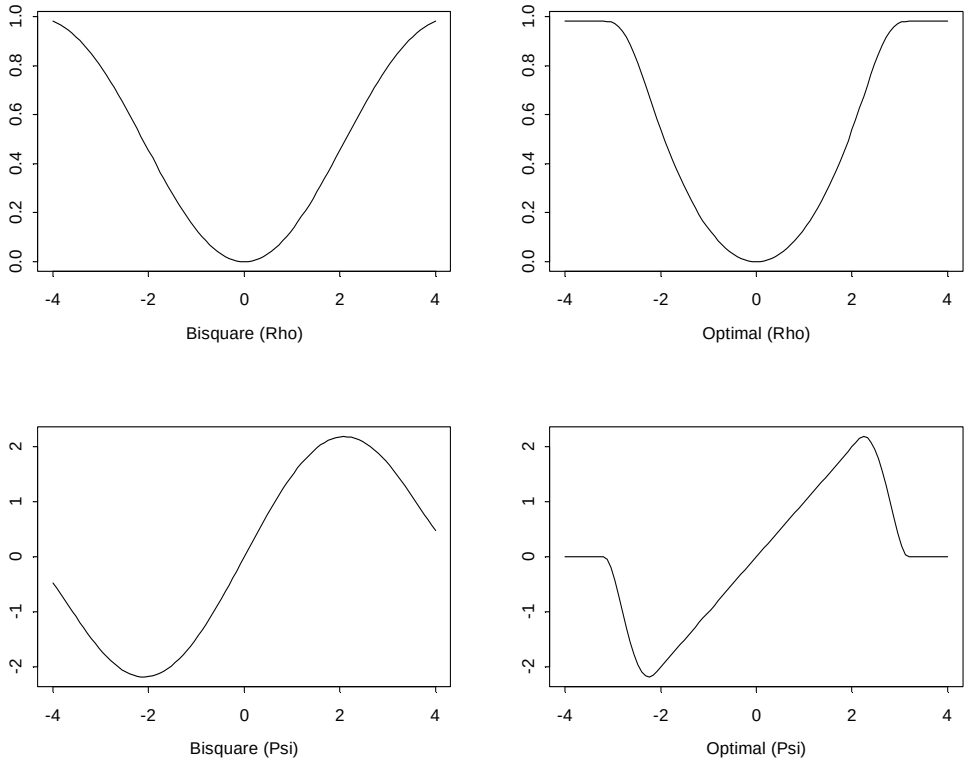

Figure 9.4: *Available loss functions.*

The exact forms of these functions can be found in the section Theoretical Details.

Since the optimal loss function above has better combined Gaussian efficiency and non-Gaussian bias control properties, it is used as the default for robust regression. However, you can choose to use the Tukey bisquare function or a combination of those two functions by controlling the `weight` argument to `lmRobMM.robust.control` as follows:

```
> control <- lmRobMM.robust.control(weight=c("Bisquare",
+ "Optimal"))
> oil.tmp <- lmRobMM(Oil~Market,data=oil.df,
+ robust.control=control)
```

```
> coef(oil.tmp)

 (Intercept)      Market
  -0.08371818  0.8291069
```

In the above commands, the rescaled bisquare function is used for the initial S-estimates, and the optimal loss function is used for the final M-estimates.

Confidence Level of Bias Test

In the `oil.robust` example shown above, the final M-estimates are accepted over the initial S-estimates because the *p*-value of the test for bias is 0.33. The default level of this test is set at 10%, so whenever the *p*-value of the test is greater than 10%, the final M-estimates are returned; otherwise, the initial S-estimates are returned.

To change the level of the test for bias of the final M-estimates to a different value, you should specify the argument `level` for the `lmRobMM.robust.control` function. A higher value of `level` will reject the final M-estimates more often, and a lower value of `level` will reject the final M-estimates less often. For example, you can force the procedure to return the initial S-estimates by using the following commands:

```
> control.s <- lmRobMM.robust.control(level=1)
> oil.s <- lmRobMM(Oil~Market,data=oil.df,
+ robust.control=control.s)

Significant test at level  100 %

> oil.s

Initial S-estimates.

Call:
lmRobMM(formula = Oil ~ Market, data = oil.df,
  robust.control = control.s)

Coefficients:
 (Intercept)      market
  -0.06244374  0.8273216

Degrees of freedom: 129 total; 127 residual
Residual scale estimate: 0.1446283
```

Warning

The bias is high; inference based on final estimates is not recommended; use initial estimates as exploratory tools.

Caveat

The above warning is only relevant when you use levels in the range of 1% to 10%, and the choice of level in this range is a rather subjective choice of the user.

Similarly, using `level=0` forces `lmRobmm` to return the final M-estimates:

```
> control.mm <- lmRobMM.robust.control(level=0)
> oil.mm <- lmRobMM(Oil ~ Market, data = oil.df,
+ robust.control = control.mm)
```

Sometimes you may want to change the level of the test after fitting a robust regression model. For this purpose, you can use the generic function `update`, which has a method for `"lmRobMM"` objects. For example, to change the level of test for bias for `oil.s`, use the following command:

```
> oil.tmp <- update(oil.s,level=0.2)
> oil.tmp

Final M-estimates.

Call:
lmRobMM(formula = Oil ~ Market, data = oil.df,
  robust.control = control.s)

Coefficients:
 (Intercept)      Market
  -0.08395777  0.8288791

Degrees of freedom: 129 total; 127 residual
Residual scale estimate: 0.1478398
```

Now the final M-estimates are returned. Also, if both the `formula` and the `level` arguments are missing for `update`, the function alternates between the initial S-estimates and final M-estimates.

Note: If you only want to compute the S-estimates and do not care about the final M-estimates, you can do so by specifying the `estim` argument to `lmRobMM.robust.control` as follows:

```
> control.s <- lmRobMM.robust.control(estim="S")
> oil.s <- lmRobMM(Oil ~ Market, data = oil.df,
+ robust.control = control.s)
> oil.s

Initial S-estimates.

Call:
lmRobMM(formula = Oil ~ Market, data = oil.df,
    robust.control = control.s)

Coefficients:
 (Intercept)      Market
  -0.06244374  0.8273216

Degrees of freedom: 129 total; 127 residual
Residual scale estimate: 0.1446283
```

Similarly, you can get the final M-estimates if you use `estim="MM"`.

Resampling Algorithms

When computing the initial S-estimates, a resampling scheme is used. S-PLUS provides three resampling algorithms for the initial S-estimates: random resampling, exhaustive resampling and genetic algorithm. These algorithms can be selected by using the `sampling` argument to the function `lmRobMM.robust.control`, for which the valid choices are `"Random"`, `"Exhaustive"` and `"Genetic"`. Note that exhaustive resampling is only used/recommended when the sample size is small and there are less than 10 predictor variables.

Random Resampling Parameters

Random resampling is controlled by two paramemters: a random seed and the number of subsamples to draw. By default, the number of subsamples is set at $\lfloor 4.6 \cdot 2^p \rfloor$, where p is the number of explanatory variables, and $\lfloor\ \rfloor$ denotes the operation of rounding a number to its

closest integer. Note that this number will work fine if you have less than 13 predictor variables. However, if you have more than 13 predictor variables, the default number may be too big for computing in a reasonable time. To choose a different value for the number of subsamples to draw, use the optional argument `nrep` as follows:

```
> oil.tmp <- lmRobMM(Oil~Market,data=oil.df,nrep=10)
```

The seed of the random resampling can be controlled by specifying the argument `seed` to `lmRobMM.robust.control`.

Genetic Algorithm Parameters

If you choose to use the genetic algorithm, the parameters for genetic algorithm can be changed through the `lmRobMM` optional argument `genetic.control`, the default of which is `NULL`. The optional argument `genetic.control` should be a list, usually returned by a call to the function `lmRobMM.genetic.control`. To look at the arguments of the function `lmRobMM.genetic.control`, use the following command:

```
> args(lmRobMM.genetic.control)
function(popsize = NULL, mutate.prob = NULL,
random.n = NULL, births.n = NULL, stock = list(),
maxslen = NULL, stockprob = NULL, nkeep = 1)
```

For an explanation of the various arguments above, you should read the help file for the function `ltsreg.default`.

THEORETICAL DETAILS

Initial Estimate Details

The key to obtaining a good local minimum of the M-estimation objective function when using a bounded, nonconvex loss function is to compute a highly robust initial estimate β^0. S-PLUS does this by using the S-estimate method introduced by Rousseeuw and Yohai (1984), as part of an overall MM-estimate computational strategy proposed by Yohai, Stahel and Zamar (1991), and supported by a number of robustness experts who participated in the 1989 IMA summer conference on "Directions in Robust Statistics and Diagnostics."

The S-estimate approach has as its foundation an M-estimate $\hat{s}$ of an unknown scale parameter for observations $y_1, y_2, ..., y_n$, assumed to be robustly centered (that is, by subtracting a robust location estimate). The M-estimate $\hat{s}$ is obtained by solving the equation

$$\frac{1}{n}\sum_{i=1}^{n}\rho\left(\frac{y_i}{\hat{s}}\right) = .5 \tag{9.1}$$

where ρ is a symmetric, bounded function. It is known that such a scale estimate has a breakdown point of one-half (Huber, 1981), and that one can find min-max bias robust M-estimates of scale (Martin and Zamar, 1989, 1993).

The following regression S-estimate method was introduced by Rousseeuw and Yohai (1984). Consider the linear regression model modification of (3.7):

$$\frac{1}{n-p}\sum_{i=1}^{n}\rho\left(\frac{y_i - x_i^T\beta}{\hat{s}(\beta)}\right) = .5 \tag{9.2}$$

For each value of β, we have a corresponding robust scale estimate $\hat{s}(\beta)$. The regression S-estimate (which stands for "minimizing a robust scale estimate") is the value $\hat{\beta}^0$ that minimizes $\hat{s}(\beta)$:

$$\hat{\beta}^0 = \text{argmin}_\beta \hat{s}(\beta) \tag{9.3}$$

This presents another nonlinear optimization, one for which the solution is traditionally found by a random resampling algorithm, followed by a local search, as described in Yohai, Stahel and Zamar (1991). S-PLUS allows you to use a genetic algorithm in place of the resampling algorithm, and also to use an exhaustive form of sampling algorithm for small problems. Once the initial S-estimate $\hat{\beta}^0$ is computed, the final M-estimate is obtained as the nearest local minimum of the M-estimate objective function.

For details on the numerical algorithms used, see Marazzi (1993), whose algorithms, routines and code were used in creating `lmRobMM`.

Optimal and Bisquare Rho and Psi-Functions

A robust M-estimate of regression coefficient β is obtained by minimizing

$$\sum_{i=1}^{n} \rho\left(\frac{y_i - x_i^T \beta}{\sigma}; c\right),$$

where $\rho(\cdot;c)$ is a convex weight function of the residuals with tuning constant c. The derivative of $\rho(\cdot;c)$ is usually denoted by $\psi(\cdot;c)$. For both the initial S-estimate and the final M-estimate in S-PLUS, two different weight functions can be used: Tukey's bisquare function and an optimal weight function introduced in Yohai and Zamar (1998).

Tukey's bisquare functions $\rho(\cdot;c)$ and $\psi(\cdot;c)$ are as follows:

$$\rho(r;c) = \begin{cases} \left(\frac{r}{c}\right)^6 - 3\left(\frac{r}{c}\right)^4 + 3\left(\frac{r}{c}\right)^2 & \text{if } |r| \le c \\ 1 & \text{if } |r| > c \end{cases}$$

$$\psi(r;c) = \begin{cases} \frac{6}{c}\left(\frac{r}{c}\right) - \frac{12}{c}\left(\frac{r}{c}\right)^3 + \frac{6}{c}\left(\frac{r}{c}\right)^5 & \text{if } |r| \le c \\ 1 & \text{if } |r| > c \end{cases}$$

The Yohai and Zamar optimal functions $\rho(\cdot;c)$ and $\psi(\cdot;c)$ are as follows:

$$\rho(r;c) = \begin{cases} 3.25c^2 & \text{if } \left|\frac{r}{c}\right| > 3 \\ c^2\left[1.792 + h_1\left(\frac{r}{c}\right)^2 + h_2\left(\frac{r}{c}\right)^4 + h_3\left(\frac{r}{c}\right)^6 + h_4\left(\frac{r}{c}\right)^8\right] & \text{if } 2 < \left|\frac{r}{c}\right| \le 3 \\ \frac{r^2}{2} & \text{if } \left|\frac{r}{c}\right| \le 2 \end{cases}$$

$$\psi(r;c) = \begin{cases} 0 & \text{if } \left|\frac{r}{c}\right| > 3 \\ c\left[g_1\frac{r}{c} + g_2\left(\frac{r}{c}\right)^3 + g_3\left(\frac{r}{c}\right)^5 + g_4\left(\frac{r}{c}\right)^7\right] & \text{if } 2 < \left|\frac{r}{c}\right| \le 3 \\ r & \text{if } \left|\frac{r}{c}\right| \le 2 \end{cases}$$

where

$$\begin{aligned} g_1 &= -1.944 & h_1 &= \frac{g_1}{2} \\ g_2 &= 1.728 & h_2 &= \frac{g_2}{4} \\ g_3 &= -0.312 & h_3 &= \frac{g_3}{6} \\ g_4 &= 0.016 & h_4 &= \frac{g_4}{8} \end{aligned}.$$

The Efficient Bias Robust Estimate

Yohai and Zamar (1998) showed that the ρ and ψ functions given above are optimal in the following highly desirable sense: the final M-estimate has a breakdown point of one-half, and minimizes the maximum bias under contamination distributions (locally for small fractions of contamination), subject to achieving a desired efficiency when the data is Gaussian.

Efficiency Control

The Gaussian efficiency of the final M-estimate is controlled by the choice of the tuning constant c. As discussed in the earlier sections, you can specify a desired Gaussian efficiency and S-PLUS will automatically use the correct c for achieving that efficiency.

Robust R-Squared

The robust R^2 is calculated as follows:

- *Initial S-estimator* $\hat{\beta}^0$

  If an intercept term is included in the model, then

  $$R^2 = \frac{(n-1)s_y^2 - (n-p)s_e^2}{(n-1)s_y^2}$$

  where $s_e = \hat{s}^0$ and s_y is the minimized $\hat{s}(\mu)$, for a regression model with only an intercept term with parameter μ. If there is no intercept term, replace $(n-1)s_y^2$ in the above formula with $n\hat{s}(0)^2$.

- *Final M-estimator* $\hat{\beta}^1$

  If an intercept term μ is included in the model, then

  $$R^2 = \frac{\sum \rho\left(\frac{y_i - \hat{\mu}}{\hat{s}^0}\right) - \sum \rho\left(\frac{y_i - x_i^T\hat{\beta}}{\hat{s}^0}\right)}{\sum \rho\left(\frac{y_i - \hat{\mu}}{\hat{s}^0}\right)}$$

where $\hat{\mu}$ is the location M-estimate corresponding to the local minimum of

$$Q_y(\mu) = \sum \rho\left(\frac{y_i - \mu}{\hat{s}^0}\right)$$

such that

$$Q_y(\hat{\mu}) \leq Q_y(\mu^*)$$

where μ^* is the sample median estimate. If there is no intercept, replace $\hat{\mu}$ with zero in the formula.

Robust Deviance

For an M-estimate, the deviance is defined as the optimal value of the objective function on the σ^2-scale; that is:

- *Initial S-estimator* $\hat{\beta}^0$

$$D = \hat{s}^2(\hat{\beta}^0) = (\hat{s}^0)^2$$

- *Final M-estimator* $\hat{\beta}^1$

$$D = 2 \cdot (\hat{s}^0)^2 \cdot \sum \rho\left(\frac{y_i - x_i^T \hat{\beta}^1}{\hat{s}^0}\right)$$

Robust F Test

See Chapter 7 of Hampel, Ronchetti, Rousseeuw, and Stahel (1986), where this test is referred to as the "tau" test.

Robust Wald Test

See Chapter 7 of Hampel, Ronchetti, Rousseeuw, and Stahel (1986).

Robust FPE (RFPE)

Ronchetti (1985) proposed to generalize the Akaike Information Criterion (AIC) to robust model selection. However, the results therein are subject to certain restrictions, such as they only apply to M-estimates with zero breakdown point and the density of the errors

has to be in a certain form. Yohai (1997) proposed the following RFPE criterion which is not subject to the restrictions that apply to Ronchetti's robust version of AIC.

$$RFPE = nE\rho\left(\frac{\varepsilon}{\sigma}\right) + p\frac{A}{2B} \tag{9.4}$$

where

$$A = E\psi^2\left(\frac{\varepsilon}{\sigma}\right) \qquad B = E\psi'\left(\frac{\varepsilon}{\sigma}\right).$$

Since the first term in equation Equation (9.4) can be approximated by

$$\eta E\rho\left(\frac{\varepsilon}{\sigma}\right) \approx \sum_{i=1}^{n}\rho\left(\frac{r_i}{\sigma}\right) + p\frac{A}{2B}$$

where $r_i = y_i - x_i^T\hat{\beta}^1$, Equation (9.4) can be estimated by

$$RFPE \approx \left[\sum_{i=1}^{n}\rho\left(\frac{y_i - x_i^T\hat{\beta}^1}{\hat{s}^0}\right)\right] + p\frac{\hat{A}}{\hat{B}} \tag{9.5}$$

with

$$\hat{A} = \frac{1}{n}\sum_{i=1}^{n}\psi^2\left(\frac{r_i}{\hat{s}^0}\right) \qquad \hat{B} = \frac{1}{n}\sum_{i=1}^{n}\psi'\left(\frac{r_i}{\hat{s}^0}\right)$$

The approximation on the right-hand side of Equation (9.5) is used as our RFPE.

OTHER ROBUST REGRESSION TECHNIQUES

Least Trimmed Squares Regression

Least trimmed squares regression (LTS) regression, introduced by Rousseeuw, 1984, is a highly robust method for fitting a linear regression model. The LTS estimate $\hat{\beta}_{LTS}$ minimizes the sum of the q smallest squared residuals

$$\sum_{i=1}^{q} r_i^2(\beta) \tag{9.6}$$

where $r_i(\beta)$ is the ith ordered residual. The value of q is often set to be slightly larger than half of n.

By contrast, the ordinary least squares estimate $\hat{\beta}_{LS}$ minimizes the sum of all of the squared residuals.

$$\sum_{i=1}^{n} r_i^2(\beta) \tag{9.7}$$

The least squares estimator lacks robustness because a single "observation" (y_i, x_i, T) can cause $\hat{\beta}_{LS}$ to take on *any* value. The same is true of M-estimators of regression, which are discussed in the section M-Estimates of Regression.

To compute the least trimmed squares regression, use the ltsreg function. For example, for the stack loss data, we can compute the LTS estimate as follows:

```
> stack.df <- data.frame(stack.loss, stack.x)
> stack.lts <- ltsreg(stack.loss ~ ., stack.df)
```

```
> stack.lts

Method:
Least Trimmed Squares Robust Regression.

Call:
ltsreg.formula(stack.loss ~ ., stack.df)

Coefficients:
 Intercept Air.Flow Water.Temp Acid.Conc.
 -36.2921    0.7362    0.3691     0.0081

Scale estimate of residuals:  1.038

Total number of observations:  21

Number of observations that determine the LTS estimate:  13
```

Comparing the coefficients to those for the least-squares fit, we observe that the LTS values are noticeably different:

```
> stack.lm <- lm(stack.loss ~ ., stack.df)
> coef(stack.lm)

 (Intercept)  Air.Flow Water.Temp Acid.Conc.
   -39.91967 0.7156402   1.295286 -0.1521225

> coef(stack.lts)

 Intercept  Air Flow Water Temp  Acid Conc.
-36.29212 0.7361645  0.3691111 0.008085206
```

A plot of the residuals versus the fitted values for the two fits is also revealing:

```
> par(mfrow=c(1,2))
> plot(fitted(stack.lm), resid(stack.lm),
+ ylim=range(resid(stack.lts)))
> plot(fitted(stack.lts), resid(stack.lts))
```

The resulting plot is shown in Figure 9.5.

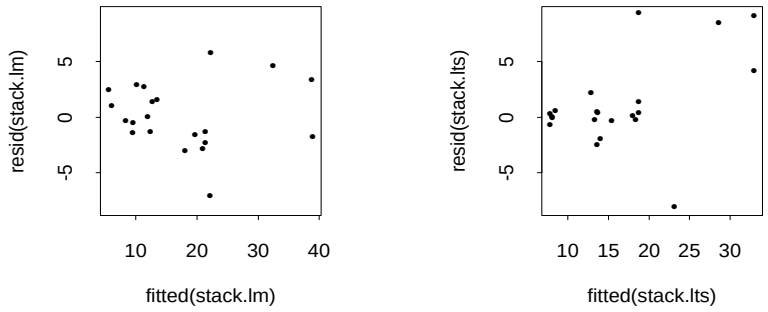

Figure 9.5: *Residual plots for least-squares (left) and least trimmed squares regression.*

The plot on the left shows the residuals scattered with no apparent pattern; the plot on the right shows four clear outliers–three at the top and one at the bottom.

The *LTS* estimator has the highly attractive robustness property that its *breakdown point* is approximately 1/2 (if q is the right fraction of n). The *breakdown point* of a regression estimator is the largest fraction of data which may be replaced by arbitrarily large values without making the Euclidean norm $\|\hat{\beta}\|$ of the resulting estimate tend to ∞. The Euclidean norm is defined as follows:

$$\|\hat{\beta}\|^2 = \sum_{i=1}^{p} \hat{\beta}_i^2 \quad .$$

To illustrate the concept of breakdown point, consider the simple problem of estimating location, where often the estimator is the sample mean $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$. The breakdown point of the mean estimator is 0, since if any single $y_i \to \pm\infty$, then $\bar{y} \to \pm\infty$. On the other hand, the sample median has breakdown point approximately 1/2, since, taking the case of an odd sample size n for convenience, one can move $(n - 1)/2$ of the observations y_i to $\pm\infty$ without taking the median to $\pm\infty$.

Any estimator with breakdown point approximately 1/2 is called a *high breakdown point* estimator. Thus, the *LTS* estimator is a high breakdown point regression estimator.

The high breakdown point of the *LTS* estimator means that the values $x_i^T\hat{\beta}_{LTS}$, $i = 1, ..., n$, fit the bulk of the data well, even when the bulk of the data may consist of only somewhat more than 50% of the data. Correspondingly, the residuals $r_i\hat{\beta}_{LTS} = y_i - x_i^T\beta_{LTS}$ will reveal the outliers quite clearly. Least squares residuals $r_i\hat{\beta}_{LS} = y_i - x_i^T\hat{\beta}_{LS}$ and M-estimate residuals $r_i\hat{\beta}_M = y_i - x_i^T\hat{\beta}_M$ often fail to reveal problems in the data. This can be illustrated as follows.

First construct an artificial data set with 60 percent of the data scattered about the line $y_i = x_i$ and the remaining 40 percent in an outlying cluster centered at (6,2).

```
> set.seed(14) #set the seed to reproduce this example
> x30 <- runif(30, .5, 4.5)
> e30 <- rnorm(30, 0, .2); y30 <- 2+x30+e30
> x20 <- rnorm(20, 6, .5); y20 <- rnorm(20, 2, .5)
> x <- c(x30, x20)
> y <- c(y30, y20)
```

Plot the data, then fit and label 3 different regression lines: the ordinary least squares line, an M-estimate line, and the least trimmed squared residuals line.

```
> plot(x, y)
> abline(lm(y ~ x))
> text(5, 3.4, "LS")
> abline(rreg(x, y))
> text(4, 3.2, "M")
> abline(ltsreg(x, y))
> text(4, 6.5, "LTS")
```

The resulting plot is shown in Figure 9.6.

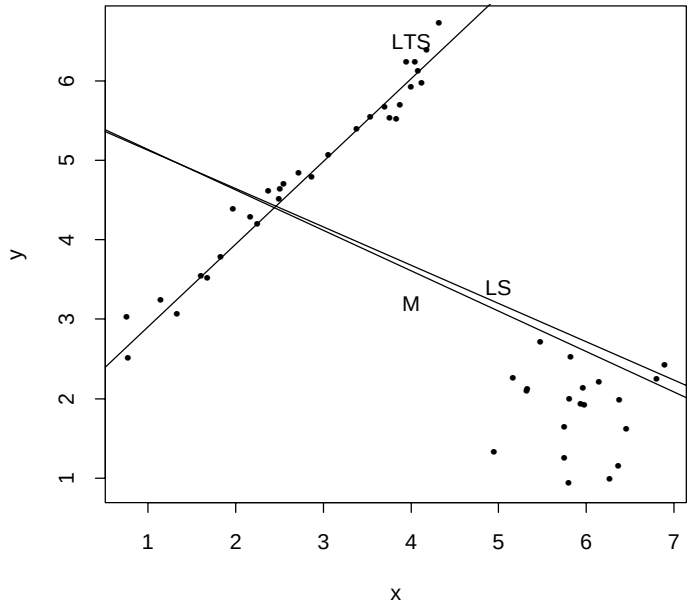

Figure 9.6: *Least trimmed squares regression, compared to least-squares and M-estimates.*

The outlying points pull both the ordinary least squares line and the M-estimate away from the bulk of the data. Neither of these two fitting methods is robust to outliers in the *x* direction. (Such outliers are called *leverage points* in the literature.) The LTS line recovers the linear structure in the bulk of the data and essentially ignores the outlying cluster. In higher dimensions such outlying clusters are very hard to identify using classical regression techniques.

Least Median Squares Regression

An idea quite similar to LTS regression is least median squares or LMS regression. Rather than minimizing the sum of the squared residuals as least squares regression does, least median of squares (Rousseeuw, 1984) minimizes the median of the squared residuals. The `lmsreg` function performs LMS regression.

Least median of squares regression has a very high breakdown point of almost 50%. That is, almost half of the data can be corrupted in an arbitrary fashion and the least median of squares estimates continue

to follow the majority of the data. At the present time this property is virtually unique among the robust regression methods that are publicly available.

However, least median of squares is statistically very inefficient. It is due to this inefficiency that we recommend `lmRobMM` and `ltsreg` over `lmsreg`.

Least Absolute Deviation Regression

The idea of least absolute deviation regression, or L1 regression, is actually older than that of least-squares, but until the development of high-speed computers, it was too cumbersome to have wide applicability. S-PLUS has the function l1fit (note that the second character in the function name is the number "1", not the letter "l") for computing least absolute deviation regression. As its name implies, least absolute deviation regression finds the estimate $\hat{\beta}_{L1}$ that minimizes the sum of the absolute values of the residuals

$$\sum_{i=1}^{n} |r_i \beta| \ .$$

As an example, consider again the stack loss data. We construct the L1 regression using `l1fit` as follows:

```
> stack.l1 <- l1fit(stack.x, stack.loss)
> stack.l1

$coefficients:
 Intercept  Air Flow Water Temp  Acid Conc.
 -39.68984 0.8318844 0.5739114 -0.06086962

$residuals:
 [1]  5.06086922  0.00000000  5.42898512  7.63478470
 [5] -1.21739304 -1.79130435 -1.00000000  0.00000000
 [9] -1.46376634 -0.02029470  0.52752948  0.04057107
[13] -2.89855528 -1.80290544  1.18260598  0.00000000
[17] -0.42608732  0.00000000  0.48695821  1.61739087
[21] -9.48116493
```

Plotting the residuals against fitted values and comparing the plot to the corresponding plot for the least-squares fit shows the outliers clearly, though not so clearly as for the `ltsreg` plot.

```
> plot(fitted(stack.lm), resid(stack.lm))
> plot(stack.loss - resid(stack.l1), resid(stack.l1))
```

The plot is shown in Figure 9.7.

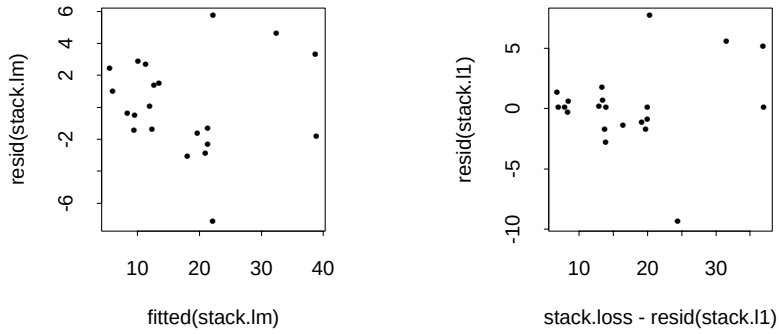

Figure 9.7: *Residual plots for least squares (left) and least absolute deviation regression.*

M-Estimates of Regression

The M-estimator of regression was introduced by Huber in 1973. An M-estimate $\hat{\beta}_M$ of regression is the β which minimizes

$$\sum_{i=1}^{n} \rho\left(\frac{r_i \beta}{\sigma}\right) \tag{9.8}$$

for a given ρ. Least squares corresponds to $\rho(x) = x^2$ and least absolute deviation regression corresponds to $\rho(x) = |x|$. Generally, although not in the two cases mentioned above, the value of $\hat{\beta}_M$ is dependent on the value of σ, which is usually unknown.

Although M-estimates are protected against wild values in the response y, they are susceptible to high leverage points–that is, points which have quite different x values relative to the other observations. In particular, a typographical error in an explanatory variable can have a dramatic affect on an M-estimate, while least trimmed squares

regression handles this situation easily. One advantage of M-estimates is that they can be computed in much less time than least trimmed squares or other high-breakdown estimates.

M-estimates in S-PLUS are calculated using the rreg function to perform an *iteratively reweighted least-squares* fit. What this means is that S-PLUS calculates an initial fit (using traditional weighted least-squares, by default), then calculates a new set of weights based on the results of the initial fit. The new weights are then used in another weighted least-squares fit, new weights are calculated, and so on, iteratively, until either some convergence criteria are satisfied or a specified maximum number of iterations is reached.

The only required arguments to `rreg` are x, the vector or matrix of explanatory variables, and y, the vector response:

```
> stack.M1 <- rreg(stack.x, stack.loss)
> stack.M1

$coefficients:
 (Intercept)  Air Flow Water Temp Acid Conc.
   -42.07438 0.8978265   0.731816 -0.1142602

$residuals:
 [1]   2.65838630  -2.45587390   3.72541082   6.78619020
 [5]  -1.75017776  -2.48199378  -1.52824862  -0.52824862
 [9]  -1.89068795  -0.03142924   0.99691253   0.61446835
[13]  -2.80290885  -1.27786270   2.17952419   0.83674360
[17]  -0.49471517   0.30510621   0.68755039   1.52911203
[21] -10.01211661
$w:
 [1] 0.87721539 0.91831885 0.77235329 0.41742415 0.95387576
 [6] 0.90178786 0.95897484 0.99398847 0.93525890 0.99958817
[11] 0.97640677 0.98691782 0.89529949 0.98052477 0.92540436
[16] 0.98897286 0.99387986 0.99933718 0.99574820 0.96320721
[21] 0.07204303
```

You control the choice of ρ in rreg by specifying a *weight* function as the method argument. There are eight weight functions built into S-PLUS; there is not yet a consensus on which is "best." See the rreg help file for details on the weight functions. The default weight function uses Huber's weight function until convergence, then a

bisquare weight function for two more iterations. The following call to rreg defines a simple weight function that corresponds to the least-squares choice $\rho = x^2$:

```
> stack.MLS <- rreg(stack.x,stack.loss,
+ method=function(u) 2*abs(u),iter=100)

Warning messages:
failed to converge in 100 steps in:
rreg(stack.x, stack.loss, method = function(u) ....

> stack.MLS$coef

 (Intercept)  Air Flow Water Temp Acid Conc.
   -39.70404 0.7165807   1.298218 -0.1561163

> coef(stack.lm)

 (Intercept)  Air.Flow Water.Temp Acid.Conc.
   -39.91967 0.7156402   1.295286 -0.1521225
```

APPENDIX

The function `gen.data` used in the section Robust Model Selection is as follows:

```
> gen.data <- function(coeff, n = 100, eps = 0.1,
+ sig = 3, snr = 1/20, seed = 837)
+ {
+ # coeff : 3 x 1 vector of coefficients
+ # eps   : the contamination ratio, between 0 and 0.5
+ # sig   : standard deviation of most observations
+ # snr   : signal-to-noise ratio, well, not really
+ # Note  : the regressors are generated as: rnorm(n,1),
+ #         rnorm(n,1)^3, exp(rnorm(n,1)). It also
+ #         generates an unused vector x4.
+ set.seed(seed)
+ x <- cbind(rnorm(n, 1), rnorm(n, 1)^3, exp(rnorm(n, 1)))
+ ru <- runif(n)
+ n1 <- sum(ru < eps)
+ u <- numeric(n)
+ u[ru < eps] <- rnorm(n1, sd = sig/snr)
+ u[ru > eps] <- rnorm(n - n1, sd = sig)
+ data.frame(y = x %*% matrix(coeff, ncol = 1) + u,
+ x1 = x[,1], x2 = x[,2], x3 = x[,3],
+ x4 = rnorm(n, 1))
+ }
```

BIBLIOGRAPHY

Hampel, F., Ronchetti, E.M., Rousseeuw, P.J. and Stahel, W.A. (1986): *Robust Statistics: the Approach Based on Influence Functions*, John Wiley & Sons.

Huber, P.J. (1981). *Robust Statistics*. John Wiley & Sons.

Marazzi, A. (1993): *Algorithms, Routines, and S Functions for Robust Statistics*, Wadsworth & Brooks/Cole, Pacific Grove, CA.

Martin, R.D. and Zamar, R.H. (1989). Asymptotically Min-Max Robust M-estimates of Scale for Positive Random Variables. *J. Amer. Statist. Assoc.*, 84, 494-501.

Martin, R.D. and Zamar, R.H. (1993). Bias Robust Estimates of Scale. *Annals of Statistics*.

Ronchetti, E. (1985): Robust Model Selection in Regression, S-PLUS Statistics & Probability Letters, 3, 21-23.

Rousseeuw, P.J. (1984). Least median of squares regression. *Journal of the American Statistical Association*, 79, 871-881.

Rousseeuw, P.J. and Yohai, V. (1984): Robust Regression by Means of S-estimators. In *Robust and Nonlinear Time Series Analysis*, J. Franke, W. Hardle, and R. D. Martin (eds.), Lecture Notes in Statistics, 26, 256-272, Springer-Verlag.

Yohai, V.J. (1987): High Breakdown-Point and High Efficiency Estimates for Regression, *Annals of Statistics*, 15, 642-665.

Yohai, V.J. (1997): A New Robust Model Selection Criterion for Linear Models: RFPE, unpublished note.

Yohai, V., Stahel, W.A. and Zamar, R.H. (1991): A Procedure for Robust Estimation and Inference in Linear Regression, in Stahel, W.A. and Weisberg, S.W., Eds., *Directions in Robust Statistics and Diagnostics, Part II*, Springer-Verlag, New York.

Yohai, V.J. and Zamar (1998). "Optimal locally robust M-estimates of regression", *Jour. of Statist. Inf. and Planning*.

GENERALIZING THE LINEAR MODEL

10

INTRODUCTION

The use of least squares estimation of regression coefficients for linear models dates back to the early nineteenth century. It met with immediate success as a simple way of mathematically summarizing relationships between observed variables of real phenomena. It quickly became and remains one of the most widely used statistical methods of practicing statisticians and scientific researchers.

Because of the simplicity, elegance, and widespread use of the linear model, researchers and practicing statisticians have tried to adapt its methodology to different data configurations. For example, there is no reason conceptually why a categorical response or some transformation of it could not be related to a set of predictor variables in a similar way to the continuous response of the linear model. Although conceptually plausible, developing regression models for categorical responses lacked solid theoretical foundation until the introduction of the generalized linear model by Nelder and Wedderburn (1972).

This chapter focuses on generalized linear models and their generalization, generalized additive models, as they apply to categorical responses. In particular, we focus on logistic and Poisson regressions and also include a brief discussion of the fitting of models when you can't specify an exact likelihood, using the quasi-likelihood method.

LOGISTIC REGRESSION

To fit a logistic regression model, use either the `glm` function or the `gam` function with a formula to specify the model and the `family` argument set to `binomial`. In this case the response variable is necessarily binary or two-valued. As an example, consider the built-in data frame `kyphosis`. A summary of the data frame produces the following:

```
> attach(kyphosis)
> summary(kyphosis)
```

```
  Kyphosis         Age              Number          Start
absent :64     Min. : 1.00     Min. : 2.000    Min. : 1.00
present:17   1st Qu.: 26.00  1st Qu.: 3.000  1st Qu.: 9.00
             Median : 87.00  Median : 4.000  Median :13.00
             Mean   : 83.65    Mean : 4.049    Mean :11.49
             3rd Qu.:130.00  3rd Qu.: 5.000  3rd Qu.:16.00
             Max.   :206.00    Max. :10.000    Max. :18.00
```

The four variables in `kyphosis` are defined as follows:

- `Kyphosis`

  A binary variable indicating the presence/absence of a postoperative spinal deformity called *Kyphosis*.

- `Age`

  The age of the child in months.

- `Number`

  The number of vertebrae involved in the spinal operation.

- `Start`

  The beginning of the range of the vertebrae involved in the operation.

A convenient way of examining the bivariate relationship between each predictor and the binary response, `Kyphosis`, is with a set of boxplots produced by `plot.factor`:

```
> par(mfrow=c(1,3), cex = .7)
> plot.factor(kyphosis)
```

Setting the `mfrow` parameter to `c(1,3)` produces three plots in a row. The character expansion is set to 0.7 times the normal size using the `cex` parameter of the `par` function. Figure 10.1 displays the result.

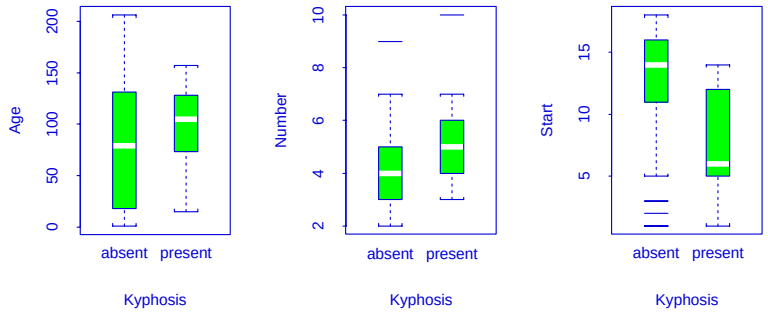

Figure 10.1: *Boxplots of the predictors of kyphosis versus* `Kyphosis`.

Both `Start` and `Number` show strong location shifts with respect to the presence or absence of `Kyphosis`. `Age` does not show such a shift in location.

Fitting a Linear Model

The logistic model we start with relates the probability of developing Kyphosis to the three predictor variables, `Age`, `Number`, and `Start`. We fit the model using `glm` as follows:

```
> kyph.glm.all <- glm(Kyphosis ~ Age + Number + Start,
+ family = binomial, data = kyphosis)
```

The `summary` function produces a summary of the resulting fit:

```
> summary(kyph.glm.all)

Call: glm(formula = Kyphosis ~ Age + Number + Start,
family = binomial, data = kyphosis)
Deviance Residuals:
      Min         1Q     Median         3Q     Max
-2.312363 -0.5484308 -0.3631876 -0.1658653 2.16133

Coefficients:
                  Value Std. Error   t value
(Intercept) -2.03693225 1.44918287 -1.405573
Age          0.01093048 0.00644419  1.696175
Number       0.41060098 0.22478659  1.826626
```

```
Start       -0.20651000 0.06768504 -3.051043

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 83.23447 on 80 degrees of freedom

Residual Deviance: 61.37993 on 77 degrees of freedom

Number of Fisher Scoring Iterations: 5

Correlation of Coefficients:
       (Intercept)        Age     Number
   Age -0.4633715
Number -0.8480574   0.2321004
 Start -0.3784028  -0.2849547  0.1107516
```

The summary includes:

1. a replica of the call that generated the fit,
2. a summary of the deviance residuals (more on these later),
3. a table of estimated regression coefficients, their standard errors, and the partial *t*-test of their significance,
4. estimates of the null and residual deviances (more on these later), and
5. a correlation matrix of the coefficient estimates.

The partial *t*-tests indicate that `Start` is important even after adjusting for `Age` and `Number`, but they provide little information on the other two variables.

You can produce an analysis of deviance for the sequential addition of each variable by using the `anova` function, specifying the chi-square test to test for differences between models:

```
> anova(kyph.glm.all, test = "Chi")

Analysis of Deviance Table

Binomial model

Response: Kyphosis
```

```
Terms added sequentially (first to last)
       Df Deviance Resid. Df Resid. Dev   Pr(Chi)
NULL                     80   83.23447
 Age  1  1.30198         79   81.93249 0.2538510
Number  1 10.30593        78   71.62656 0.0013260
Start   1 10.24663        77   61.37993 0.0013693
```

Here we see that `Number` is important after adjusting for `Age`. We already know that `Number` loses its importance after adjusting for `Age` and `Start`. `Age` does not appear to be important as a linear predictor.

You can examine the bivariate relationships between the probability of Kyphosis and each of the predictors by fitting a "null" mode and then adding each of the terms, one at a time:

```
> kyph.glm.null <- glm(Kyphosis ~ 1, family = binomial,
+ data = kyphosis)
> add1(kyph.glm.null, ~ . + Age + Number + Start)

Single term additions

Model: Kyphosis ~ 1
          Df Sum of Sq      RSS       Cp
<none>                  81.00000 83.02500
Age        1   1.29546 79.70454 83.75454
Number     1  10.55222 70.44778 74.49778
Start      1  16.10805 64.89195 68.94195
```

The `Cp` statistic is used to compare models that are not nested. A small `Cp` corresponds to a better model in the sense of smaller residual deviance penalized by the number of parameters that are estimated in fitting the model.

Clearly `Start` is the best single variable to use in a linear model. These statistical conclusions, however, should be verified by looking at graphical displays of the fitted values and residuals.

The plot method for generalized linear models produces four plots:

1. a plot of deviance residuals versus the fitted values.
2. a plot of the square root of the absolute deviance residuals versus the linear predictor values.
3. a plot of the response versus the fitted values.
4. a Normal quantile plot of the Pearson residuals.

This set of plots is similar to those produced by the plot method for `lm` objects.

Systematic curvature in the residual plots could be indicative of problems in the choice of link, wrong scale of one of the predictors, or omission of a quadratic term in a predictor. Large residuals can be also be detected with these plots. These may be indicative of the need to remove the corresponding observations and re-fit the model. The plot of the absolute residuals against predicted values gives a visual check on the adequacy of the assumed variance function.

The Normal quantile plot is also generated for `glm` objects. This plot could be useful in the detection of extreme observations deviating from a general trend but one should exercise caution in not over-interpreting its shape, which is not necessarily of interest in the nonlinear context.

Figure 10.2 results from simply plotting the fit.

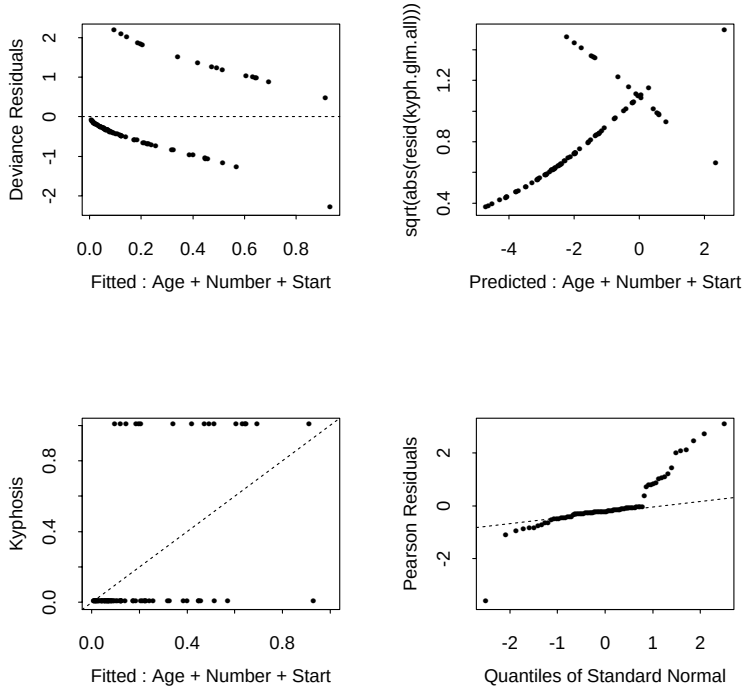

Figure 10.2: *Plots of the generalized linear model of* `Kyphosis` *predicted by* `Age`, `Start`, *and* `Number`.

Residual plots are not useful for binary data, such as `Kyphosis`, because all of the points line on one of two curves depending on whether the response is 0 or 1.

```
> par(mfrow=c(2,2))
> plot(kyph.glm.all, ask=F)
```

A more useful diagnostic plot is produced by `plot.gam`. By default, `plot.gam` plots the estimated relationship between the individual fitted terms and each of the corresponding predictors. You can request that partial residuals be added to the plot by specifying the argument `resid=T`. The scale argument can be used to keep all of the plots on the same scale to ease comparison. Figure 10.3 is produced by `plot.gam`:

```
> par(mfrow=c(1,3))
> plot.gam(kyph.glm.all, resid = T, scale = 6)
```

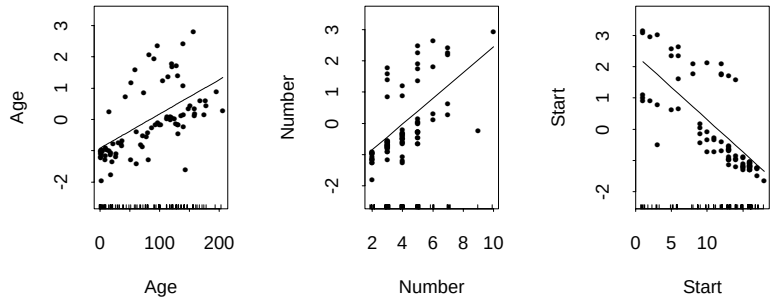

Figure 10.3: *Additional plots of the generalized linear model of* `Kyphosis` *predicted by* `Age`, `Number`, *and* `Start`.

These plots give a quick assessment of how well the model fits the data through examination of the fit of each term in the formula. The plots are of the adjusted relationship for each predictor versus each predictor. When the relationship is specified as *linear* the label on the vertical axis reduces to the variable name. We will see the utility of this plot method and the reason for the labels when we plot additive models produced by `gam`.

Both `plot.glm` (the underlying plotting method for generalized linear models) and `plot.gam` produce multiple plots; you can, however, choose which plots you look at by using the argument

`ask=T` (the default). This produces a menu of available plots; you then select the number of the desired plot. For example, here is the menu of default GLM plots for the function `kyph.glm.all`:

```
> plot(kyph.glm.all, ask=T)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Residuals vs Fitted Values
3: plot: Sqrt of abs(Residuals) vs Predictions
4: plot: Response vs Fitted Values
5: plot: Normal QQplot of Std. Residuals
Selection:
```

Fitting an Additive Model

So far we have examined only *linear* relationships between the predictors and the probability of developing Kyphosis. We can assess the validity of the linear assumption by fitting an *additive* model with relationships estimated by smoothing operations (cubic splines or local regression) and comparing it to the linear fit. We use the `gam` function to fit additive models.

```
> kyph.gam.all <-
+ gam(Kyphosis ~ s(Age) + s(Number) + s(Start),
+ family = binomial, data = kyphosis)
```

Including each variable as an argument to the `s` function instructs `gam` to estimate the "smoothed" relationships with each predictor by using cubic *B*-splines. Alternatively we could have used the `lo` function for local regression smoothing (`loess`). A summary of the fit is:

```
> summary(kyph.gam.all)

Call: gam(formula = Kyphosis ~ s(Age) +s(Number)+ s(Start),
family = binomial, data = kyphosis)
Deviance Residuals:
Min 1Q Median 3Q Max
-1.351358 -0.4439636 -0.1666238 -0.01061843 2.10851

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 83.23447 on 80 degrees of freedom
```

```
Residual Deviance: 40.75732 on 68.1913 degrees of freedom

Number of Local Scoring Iterations: 7

DF for Terms and Chi-squares for Nonparametric Effects

            Df Npar Df Npar Chisq    P(Chi)
(Intercept)  1
     s(Age)  1     2.9   5.782245 0.1161106
  s(Number)  1     3.0   5.649706 0.1289318
   s(Start)  1     2.9   5.802950 0.1139286
```

The summary of a `gam` fit is similar to the summary of a `glm` fit. One noticeable difference is the analysis of deviance table. For the additive fit the tests correspond to *approximate* partial tests for the importance of the smooth for each term in the model. These tests are typically used for screening variables for inclusion in the model. The approximate nature of these tests is discussed in detail in Hastie and Tibshirani (1990). For a single variable in the model, this is equivalent to testing for a difference between a linear fit and a smooth fit which includes a linear term along with the smooth term.

Now let's fit two additional models, adding a smooth of each of `Age` and `Number` to the base model which has a smooth of `Start`.

```
> kyph.gam.start.age <-
+ gam(Kyphosis ~ s(Start) + s(Age),
+ family = binomial, data = kyphosis)
> kyph.gam.start.number <-
+ gam(Kyphosis ~ s(Start) + s(Number),
+ family = binomial, data = kyphosis)
```

We produce the following analysis of deviance tables:

```
> anova(kyph.gam.start, kyph.gam.start.age, test="Chi")

Analysis of Deviance Table

Response: Kyphosis

                Terms Resid. Df Resid. Dev
1           s(Start)  76.24543   59.11262
2 s(Start) + s(Age)  72.09458   48.41713
```

```
     Test        Df  Deviance   Pr(Chi)
1
2 +s(Age)  4.150842  10.69548  0.0336071

> anova(kyph.gam.start, kyph.gam.start.number,
+ test="Chi")

Analysis of Deviance Table

Response: Kyphosis

               Terms Res.Df Res.Dev
1           s(Start) 76.245 59.1126
2 s(Start)+s(Number) 72.180 54.1790
        Test       Df Deviance   Pr(Chi)
1
2 +s(Number) 4.064954 4.933668 0.3023856
```

The indication is that `Age` is important in the model even with `Start` included whereas `Number` isn't important under the same conditions.

You can plot the fit with a smooth on `Age` and `Start` adding partial residuals while maintaining all figures on the same scale as follows:

```
> par(mfrow = c(2,2))
> plot(kyph.gam.start.age, resid = T, scale = 8)
```

Or you can simply plot the fit adding pointwise confidence intervals for the fit.

```
> plot(kyph.gam.start.age, se = T, scale = 10)
```

Figure 10.4 displays the resulting plots produced by `plot.gam`.

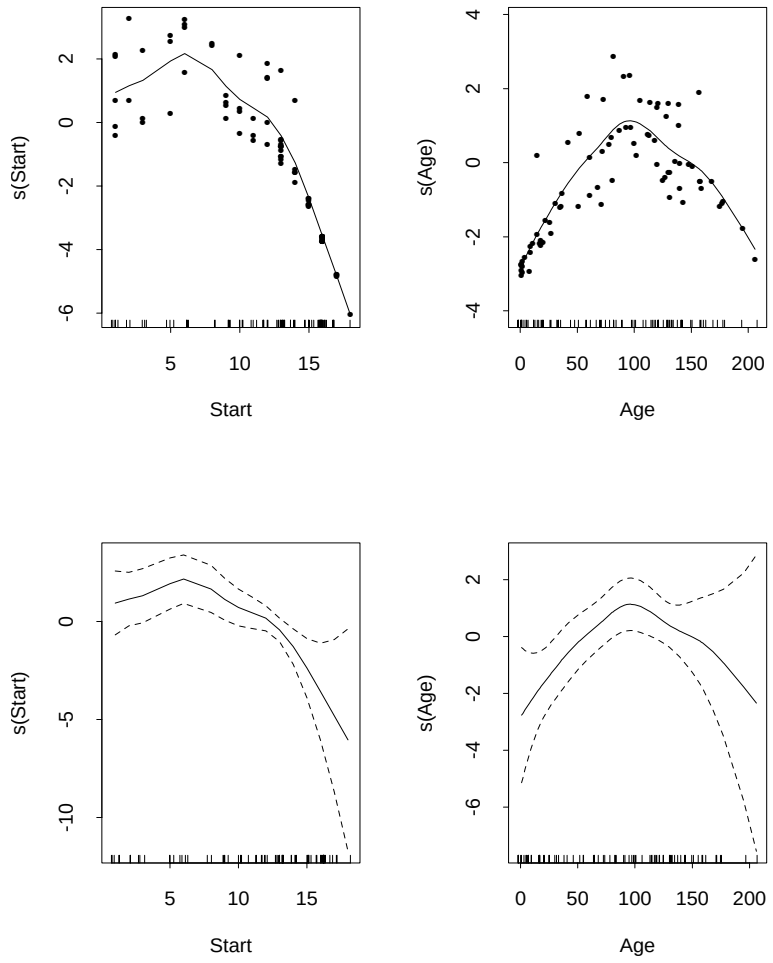

Figure 10.4: *The partial fits for the generalized additive logistic regression model of* `Kyphosis` *with* `Age` *and* `Start` *as predictors.*

Notice the vertical axes labels now. They reflect the smoothing operation included in the modeling.

The summary of the additive fit with smooths of `Age` and `Start` included appears as follows:

```
> summary(kyph.gam.start.age)
```

```
Call: gam(formula = Kyphosis ~ s(Start) + s(Age),
family = binomial, data = kyphosis)
Deviance Residuals:
       Min         1Q     Median          3Q      Max
 -1.694389 -0.4212112 -0.1930565 -0.02753535 2.087434

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 83.23447 on 80 degrees of freedom

Residual Deviance: 48.41713 on 72.09458 degrees of freedom

Number of Local Scoring Iterations: 6

DF for Terms and Chi-squares for Nonparametric Effects
            Df Npar Df Npar Chisq    P(Chi)
(Intercept)  1
   s(Start)  1     2.9   7.729677 0.0497712
     s(Age)  1     3.0   6.100143 0.1039656
```

Returning to the Linear Model

The plots of the fits of the additive model displayed in Figure 10.4 suggest a quadratic relationship for `Age` and a piecewise linear relationship for `Start`. It is useful to fit these suggested relationships as a linear model if the model is further simplified without losing too much precision in predicting the response.

For `Age` we fit a second degree polynomial. For `Start`, recall that its values indicate the beginning of the range of the vertebrae involved in the operation. Values less than or equal to 12 correspond to the thoracic region of the spine and values greater than 12 correspond to the lumbar region. Since the relationship for `Start` is fairly flat for values of `Start` approximately less than or equal to 12, and then drops off linearly for values greater than 12, we will try fitting a linear model with the term `I((Start - 12) * (Start > 12))`. The `I` function is used here to prevent the `"*"` from being used for factor expansion in the formula sense.

Figure 10.5 displays the resulting fit along with the partial residuals as well as the fit along with two standard errors bands.

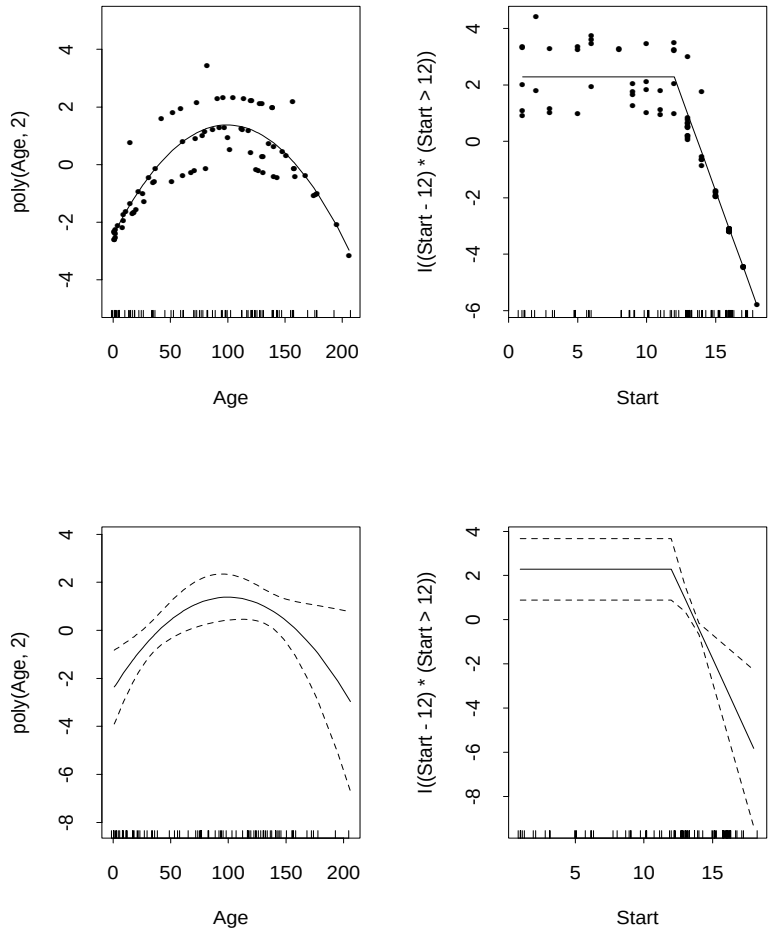

Figure 10.5: *The partial fits for the generalized linear logistic regression model of* `Kyphosis` *with quadratic fit for* `Age` *and piecewise linear fit for* `Start`.

The summary of the fit follows:

```
> summary(kyph.glm.istart.age2)

Call: glm(formula = Kyphosis ~ poly(Age, 2) +
   I((Start - 12) * (Start > 12)), family = binomial,
   data = kyphosis)
```

```
Deviance Residuals:
     Min         1Q     Median          3Q      Max
-1.42301 -0.5014355 -0.1328078 -0.01416602 2.116452
Coefficients:
                              Value Std. Error   t value
             (Intercept)  -0.6849607  0.4570976 -1.498500
           poly(Age, 2)1   5.7719269  4.1315471  1.397038
           poly(Age, 2)2 -10.3247767  4.9540479 -2.084109
I((Start-12)*(Start>12))  -1.3510122  0.5072018 -2.663658

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 83.23447 on 80 degrees of freedom

Residual Deviance: 51.95327 on 77 degrees of freedom

Number of Fisher Scoring Iterations: 6

Correlation of Coefficients:
                       (Intercept) poly(Age,2)1 poly(Age,2)2
           poly(Age, 2)1 -0.1133772
           poly(Age, 2)2  0.5625194  0.0130579
I((Start-12)*(Start>12)) -0.3261937 -0.1507199 -0.0325155
```

Contrasting the summary of this linear fit (`kyph.glm.istart.age2`) with the additive fit with smooths of `Age` and `Start` (`kyph.gam.start.age`) we can see the following important details:

1. The linear fit is more parsimonious; the effective number of parameters being estimated is approximately 5 less than for the additive model with smooths.
2. The residual deviance has increased by only about 3.5 even with a decrease in the effective number of parameters in fitting the linear model by about five. We use the `anova` function to verify that there is no difference between these models.

```
> anova(kyph.glm.istart.age2, kyph.gam.start.age,
+ test="Chi")
```

```
Analysis of Deviance Table

Response: Kyphosis

                                      Terms  Res. Df Res. Dev
1 poly(Age,2)+I((Start-12)*(Start>12)) 77.00000 51.95327
2                    s(Start) + s(Age) 72.09458 48.41713

     Test       Df Deviance   Pr(Chi)
1
2 1 vs. 2 4.905415 3.536134 0.6050618
```

3. Having fit a linear model, we can produce an *analytical* expression for the model, which we can't do for an additive model with smooth fits. This is because for a linear model, coefficients are estimated for a *parametric* relationship whereas for an additive model with smooth fits, the smooths are nonparametric estimates of the relationship. In general, these nonparametric estimates have no analytical form and are based on an iterative computer algorithm. This is an important distinction between linear models and additive models with smooth terms.

POISSON REGRESSION

To fit a Poisson regression model use either the `glm` function or the `gam` function with a formula to specify the model and the `family` argument set to `poisson`. In this case the response variable is discrete taking on non-negative integer values. Count data is frequently modeled as a Poisson distribution. As an example, consider the built-in data frame `solder.balance`. A summary of the data frame produces the following:

```
> attach(solder.balance)
> summary(solder.balance)

Opening   Solder    Mask   PadType  Panel        skips
S:240  Thin :360 A1.5:180  L9 : 72  1:240  Min.   : 0.000
M:240  Thick:360 A3  :180  W9 : 72  2:240  1st Qu.: 0.000
L:240            B3  :180  L8 : 72  3:240  Median : 2.000
                 B6  :180  L7 : 72          Mean   : 4.965
                           D7 : 72          3rd Qu.: 6.000
                           L6 : 72          Max.   :48.000
                        (Other):288
```

The *solder* experiment, contained in `solder.balance`, was designed and implemented in one of AT&T's factories to investigate alternatives in the "wave-soldering" procedure for mounting electronic components on circuit boards. Five different factors were considered as having an effect on the number of solder skips. A brief description of each of the factors follows. For more details, see the paper by Comizzoli, Landwehr, and Sinclair (1990).

- `Opening`: amount of clearance around the mounting pad
- `Solder`: amount of solder
- `Mask`: type and thickness of the material used for the solder mask
- `PadType`: the geometry and size of the mounting pad
- `Panel`: each board was divided into three panels, with three runs on a board
- `skips`: number of visible solder skips on a circuit board.

Two useful preliminary plots of the data are a histogram of `skips`, the response, and plots of the mean response for each level of the predictor. Figure 10.6 and Figure 10.7 display the resulting plots.

```
> par(mfrow=c(1,1))
> hist(skips)
> plot(solder.balance)
```

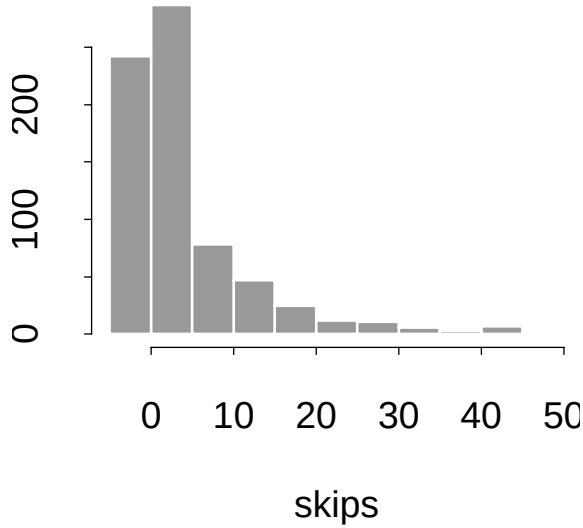

Figure 10.6: *A histogram of* `skips` *for* `solder` *data.*

The histogram of `skips` in Figure 10.6 shows the skewness and long-tailedness typical of count data. We will model this using a Poisson distribution.

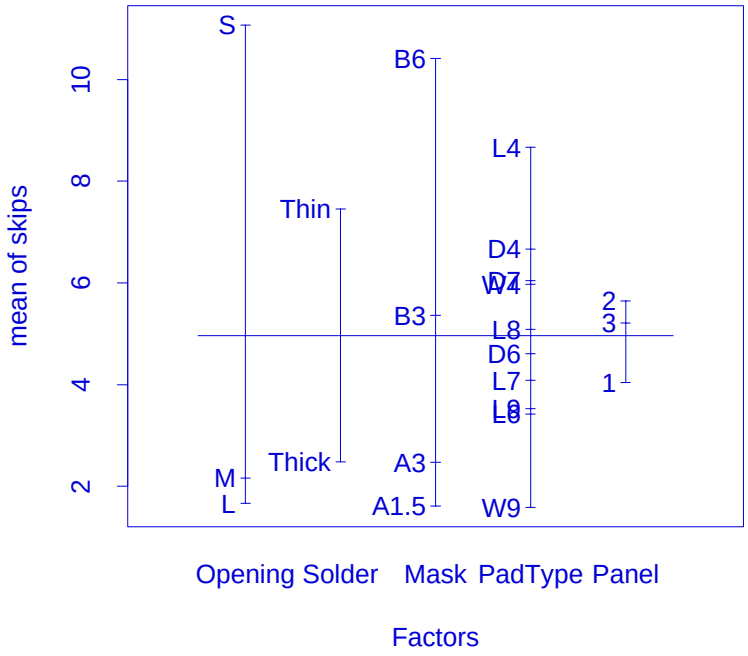

Figure 10.7: *A plot of the mean response for each level of each factor.*

The plot of the mean skips for different levels of the factors displayed in Figure 10.7 shows a very strong effect due to `Opening`. For levels `M` and `L`, only about two skips were seen on average, whereas for level `S`, more then 10 skips were seen. Effects almost as strong were seen for different levels of `Mask`.

If we do boxplots of `skips` for each level of the two factors, `Opening` and `Mask`, we get an idea of the distribution of the data across levels of the factors. Figure 10.8 displays the results of doing "factor" plots on these two factors.

```
> par(mfrow=c(1,2))
> plot.factor(skips ~ Opening + Mask)
```

On examining Figure 10.8, it is clear that the variance of `skips` increases as its mean increases. This is typical of Poisson distributed data.

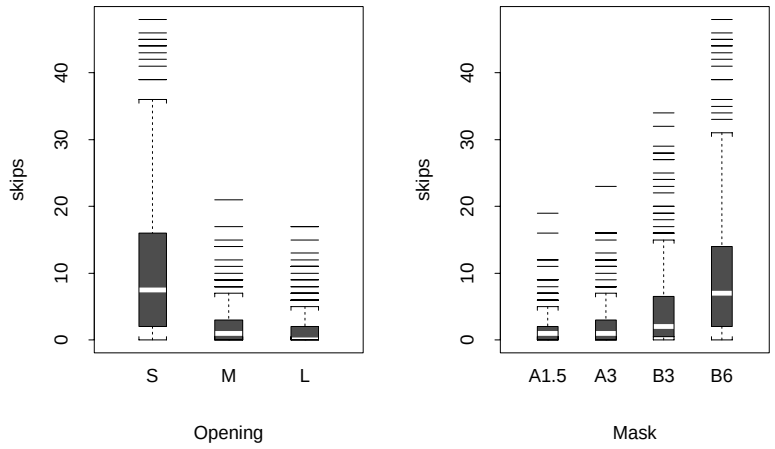

Figure 10.8: *Boxplots for each level of the two factors* `Opening` *and* `Mask`.

We proceed now to model `skips`, using `glm`, as a function of the controlled factors in the experiment declaring `family = poisson`. We start with a simple-effects model for `skips` as follows:

```
> paov <- glm(skips ~ . , family = poisson,
+ data = solder.balance)
> anova(paov, test = "Chi")

Analysis of Deviance Table

Poisson model

Response: skips

Terms added sequentially (first to last)
         Df Deviance Resid. Df Resid. Dev      Pr(Chi)
   NULL                    719   6855.690
Opening   2 2524.562       717   4331.128 0.000000e+00
 Solder   1  936.955        716   3394.173 0.000000e+00
   Mask   3 1653.093        713   1741.080 0.000000e+00
PadType   9  542.463        704   1198.617 0.000000e+00
  Panel   2   68.137        702   1130.480 1.554312e-15
```

The chi-squared test is requested in this case because `glm` assumes $\phi = 1$ (no under- or over-dispersion). We use the quasi-likelihood family, `quasi`, when we want to estimate the scale parameter as part of the model fitting computations for binomial or Poisson families. We could also set the argument `disp` in the `summary` function to 0 to obtain this estimate while summarizing the fitted model.

According to the analysis of deviance, it appears that all of the factors considered have a very significant influence on the number of solder skips. The solder experiment contained in `solder.balance` is balanced, so we need not be concerned with the sequential nature of the analysis of deviance table above; the tests of a sequential analysis are identical to the *partial* tests of a regression analysis when the experiment is balanced.

Now let's fit a second order model. We fit all the simple effects and all the second order terms except those including `Panel` (we have looked ahead and discovered that the interactions with `Panel` are non-significant, marginal, or of less importance than the other interactions). The analysis of deviance table follows:

```
> paov2 <- glm(skips ~ . +
+ (Opening + Solder + Mask + PadType) 2,
+ family = poisson, data = solder.balance)
> anova(paov2, test = "Chi")

Analysis of Deviance Table

Poisson model

Response: skips

Terms added sequentially (first to last)
               Df Deviance Res.Df Resid. Dev      Pr(Chi)
            NULL               719   6855.690
         Opening  2 2524.562   717   4331.128 0.0000000000
          Solder  1  936.955   716   3394.173 0.0000000000
            Mask  3 1653.093   713   1741.080 0.0000000000
         PadType  9  542.463   704   1198.617 0.0000000000
           Panel  2   68.137   702   1130.480 0.0000000000
 Opening:Solder  2   27.978   700   1102.502 0.0000008409
   Opening:Mask  6   70.984   694   1031.519 0.0000000000
Opening:PadType 18   47.419   676    984.100 0.0001836068
```

```
              Solder:Mask  3  59.806    673    924.294 0.0000000000
           Solder:PadType  9  43.431    664    880.863 0.0000017967
             Mask:PadType 27  61.457    637    819.407 0.0001694012
```

All of the interactions estimated in `paov2` are quite significant.

To verify the fit we do several different kinds of plots. The first four result from doing the standard plot for a `"glm"` object.

```
> par(mfrow=c(2,2))
> plot(paov2)
```

The resulting plot is displayed in Figure 10.9.

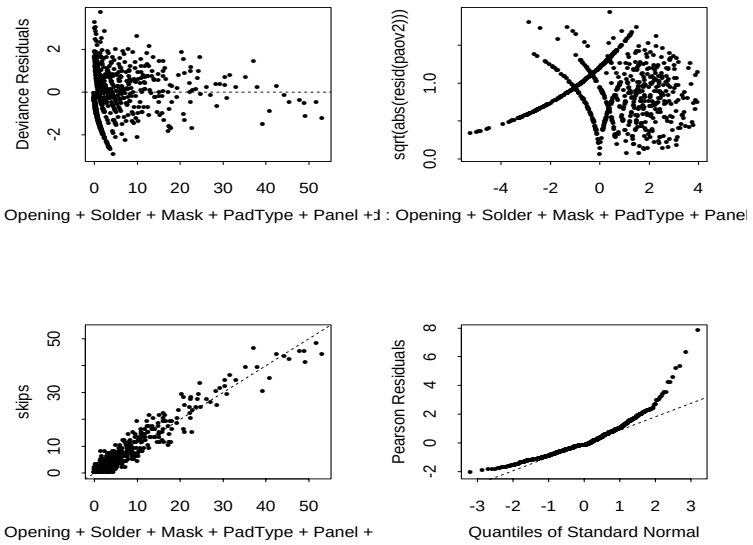

Figure 10.9: *Plots of the second order model of* `skips`.

The plot of the observations versus the fitted values shows no great departures from the model. The plot of the absolute deviance residuals shows striations due to the discrete nature of the data. Otherwise the deviance residual plot doesn't reveal anything to make us uneasy about the fit.

The other set of plots useful for examining the fit is produced by `plot.gam`. These are plots of the adjusted fit with partial residuals overlaid for each predictor variable. Since all the variables are factors, the resulting fit is a step function; a constant is fitted for each level of a factor. Figure 10.10 displays the resulting plots.

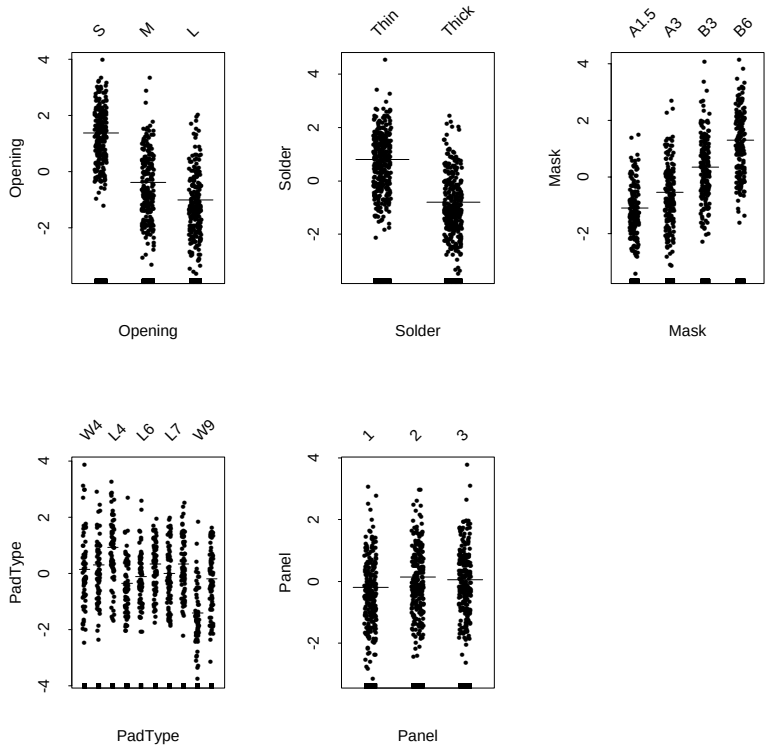

Figure 10.10: *Partial residual plots of the second order model of* `skips`.

```
> par(mfrow=c(2,3))
> plot.gam(paov2, resid = T)
```

The `plot.gam` function adds a bit of random noise to the coded factor levels to spread the plotted points out so that it is easier to see their vertical locations.

Note
The warning message about interaction terms not being saved can be safely ignored here.

These plots also indicate that the data is modeled reasonably well. Please note, however, that the default plots will show only *glaring* lack of fit.

GENERALIZED LINEAR MODELS

The linear model discussed in Chapter 8, Regression and Smoothing For Continuous Response Data, is a special case of the generalized linear model. A linear model provides a way of estimating the response variable, Y, conditional on a linear function of the values, x_1, x_2, ..., x_p, of some set of predictors variables, X_1, X_2, ..., X_p. Mathematically, we write this as:

$$E(Y|x) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i \tag{10.1}$$

For the linear model, the variance of Y is assumed constant and denoted by $\text{var}(Y) = \sigma^2$.

A *generalized* linear model provides a way to estimate a *function* (called the *link* function) of the mean response as a linear function of the values of some set of predictors. This is written as:

$$g(E(Y|x)) = g(\mu) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i = \eta(x) \tag{10.2}$$

where g is the link function. The linear function of the predictors, $\eta(x)$, is called the *linear predictor*. For the generalized linear model, the variance of Y may be a function of the mean response μ:

$$\text{var}(Y) = \phi V(\mu)$$

The logistic regression and Poisson regression examples we have seen are special cases of the generalized linear model. To do a logistic regression we declare the `binomial` family which uses the *logit* link function defined by

$$g(p) = \text{logit}(p) = \log \frac{p}{1-p}$$

and variance function defined by

$$\text{var}(Y) = \phi \frac{p}{1-p}$$

where p is the probability of an event occurring. The parameter p corresponds to the mean response of a binary (0-1) variable. In logistic regression, we model the probability of some event occurring as a linear function of a set of predictors. Usually, for the logistic regression problem ϕ is fixed to be 1 (one).

When we cannot assume that $\phi = 1$ (this is the case of over- or under-dispersion discussed in McCullagh and Nelder (1989)), we must use the `quasi` family for quasi-likelihood estimation. The quasi-likelihood "family" allows us to estimate the parameters in the model without specifying what the distribution function is. In this case the link and variance functions are all that is used for fitting the model. Once these are known, the same iterative procedure that is used for fitting the other families can be used to estimate the model parameters. For more detail, see Chambers and Hastie (1992) and McCullagh and Nelder (1989).

The Poisson regression example declares a `poisson` family with the *log* link function

$$g(\mu) = \log(\mu)$$

and the variance defined by

$$\text{var}(Y) = \phi\mu$$

The binomial and Poisson families are for fitting regression models to categorical response data. For the binomial case, the response is a binary variable indicating whether or not some event has occurred. The most common example of using the binomial family is the logistic regression problem where we try to predict the probability of the event occurring as a function of the predictors. Some examples of a binary response are presence/absence of AIDS, presence/absence of a plant species in a vegetation sample, failure/non-failure of a electronic component in a radio.

The Poisson family is useful for modeling counts which typically follow a Poisson distribution. Our earlier example modeled the number of soldering skips as a function of various controlled factors in the solder experiment.

Other families are available for modeling other kinds of data. For example normal (the linear model special case) and inverse normal distributions are modeled with the `gaussian` and `inverse.gaussian` families. Table 10.1 lists the distribution families available for use with either the `glm` or the `gam` function.

Table 10.1: *Link and variance functions for the generalized linear and generalized additive models.*

Distribution	Family	Link	Variance
Normal/Gaussian	`gaussian`	μ	1
Binomial	`binomial`	$\log(\mu/(1-\mu))$	$\mu(1-\mu)/n$
Poisson	`poisson`	$\log(\mu)$	μ
Gamma	`gamma`	$1/\mu$	μ^2
Inverse Normal/ Gaussian	`inverse.gaussian`	$1/\mu^2$	μ^3
Quasi	`quasi`	$g(\mu)$	$V(\mu)$

Each of these families represents an exponential family of distributions of a particular form. The link function for each family listed in Table 10.1 is referred to as the *canonical* link because it relates the canonical parameter of the distribution family to the linear predictor, $\eta(x)$. For more details, on the parameterization of these distributions, see McCullagh and Nelder (1989).

The estimates of the regression parameters in a `glm` are maximum likelihood estimates, produced by iteratively reweighted least-squares (IRLS). Essentially, the log-likelihood, $l(\beta, y)$, is maximized by solving the *score* equations, defined by:

$$\partial l(\beta, y) / \partial \beta = 0 \tag{10.3}$$

Since the score equations are nonlinear in β, they are solved iteratively. This iterative procedure is what is referred to as IRLS. For more details, see Chambers and Hastie (1992) or McCullagh and Nelder (1989).

GENERALIZED ADDITIVE MODELS

The section Generalized Linear Models discusses an extension of linear models to data with error distributions other than normal or Gaussian. By using `glm`, we can fit data with Gaussian, binomial, Poisson, gamma, or inverse Gaussian errors, which extends dramatically the kind of data for which we can build regression models. The primary restriction of a `glm` is the fact that it is still a *linear* model. The linear predictor is just that, a linear function of the parameters of the model.

The generalized additive model, `gam`, extends the `glm` by fitting nonparametric functions to estimate the relationships between the response and the predictors. The nonparametric functions are estimated from the data using smoothing operations.

The general form of a `gam` is:

$$g(E(Y|x)) = g(\mu) = \alpha + \sum_{i=1}^{p} f_i(x_i) = \eta(x) \qquad (10.4)$$

where g is the link function, α is a constant intercept term, f_i corresponds to the nonparametric function describing the relationship between the transformed mean response (the link transform function) and the ith predictor. In this context, $\eta(x)$ is referred to as the *additive* predictor and is entirely analogous to the *linear* predictor of a `glm` defined in Equation (10.2). As for a `glm`, the variance of Y may be function of the mean response μ:

$$\mathrm{VAR}(Y) = \phi V(\mu)$$

All of the distribution families listed in Table 10.1 are available for `gam`s. Thus fully nonparametric, nonlinear additive regression models can be fit to binomial data (logistic regression) and count data (Poisson regression) as presented in the section Logistic Regression and the section Poisson Regression, as well as to data with error distributions that are modeled by the other families listed in Table 10.1.

Two functions that are useful for fitting a `gam` are `s` and `lo`. Both of these functions are for fitting smooth relationships between the transformed response and the predictors. The `s` function fits cubic B-splines to estimate the smooth and `lo` fits a locally weighted least-squares regression to estimate the smooth. For more detail on using these functions, see their help files.

QUASI-LIKELIHOOD ESTIMATION

Quasi-likelihood estimation allows you to estimate regression relationships without fully knowing the error distribution of the response variable. Essentially, you provide link and variance functions which are used in the estimation of the regression coefficients. Although the link and variance functions are typically associated with a *theoretical* likelihood, the likelihood need not be specified and fewer assumptions are made in estimation and inference.

As a simple analogy, there is a connection between normal-theory regression models and least-squares regression estimates. Least-squares estimation gives identical parameter estimates to those produced from normal-theory models. However, least-squares estimation assumes far less; only second moment assumptions are made by least-squares compared to full distribution assumptions of normal-theory models.

Quasi-likelihood estimation for the distributions of Table 10.1 is analogous to least-squares estimation for the normal distribution. For the Gaussian family, IRLS is equivalent to standard least-squares estimation. Used in this context, quasi-likelihood estimation allows us to estimate the dispersion parameter in under- or over-dispersed regression models. For example, an under- or over-dispersed logistic or Poisson regression model can be estimated by using quasi-likelihood methodology and supplying the appropriate link and variance functions for the binomial and Poisson families, respectively.

However, quasi-likelihood estimation extends beyond the families represented in Table 10.1. Any modeling situation for which suitable link and variance functions can be derived can be modeled using the quasi-likelihood methodology. Several good examples of this kind of application are presented in McCullagh and Nelder (1989).

For our example of quasi-likelihood estimation, let's go back to the the Poisson regression example using the `solder.balance` data frame. Recall that we modeled `skips` as a function of all the factors

plus all the two-way interactions except those including `Panel`. The modeling call was:

```
> glm(formula = skips ~ . +
+ (Opening + Solder + Mask + PadType)^2,
+ family = poisson, data = solder.balance)
```

When we declare the `family` argument to be either `Poisson` or `binomial`, the dispersion parameter is set to a constant equal to one. In many problems this assumption is not valid. We can use quasi-likelihood estimation to force the *estimation* of the dispersion parameter for these families. For the solder experiment we do it as follows:

```
> paov3 <-glm(formula = skips ~ . +
+ (Opening + Solder + Mask + PadType) ^ 2,
+ family = quasi(link = "log", var = "mu"),
+ data = solder.balance)
```

A summary of the fit reveals that the dispersion parameter is estimated to be 1.4, suggesting over-dispersion. We now recompute the ANOVA table, computing *F*-statistics for testing for effects:

```
> anova(paov3, test = "F")

Analysis of Deviance Table

Quasi-likelihood model

Response: skips

Terms added sequentially (first to last)
                Df Deviance R.Df Res. Dev  F Value      Pr(F)
           NULL              719 6855.690
        Opening  2 2524.562  717 4331.128 901.1240 0.00000000
         Solder  1  936.955  716 3394.173 668.8786 0.00000000
           Mask  3 1653.093  713 1741.080 393.3729 0.00000000
        PadType  9  542.463  704 1198.617  43.0285 0.00000000
          Panel  2   68.137  702 1130.480  24.3210 0.00000000
 Opening:Solder  2   27.978  700 1102.502   9.9864 0.00005365
   Opening:Mask  6   70.984  694 1031.519   8.4457 0.00000001
Opening:PadType 18   47.419  676  984.100   1.8806 0.01494805
    Solder:Mask  3   59.806  673  924.294  14.2316 0.00000001
 Solder:PadType  9   43.431  664  880.863   3.4449 0.00036929
   Mask:PadType 27   61.457  637  819.407   1.6249 0.02466031
```

All of the factors and interactions are still significant even when we model the over-dispersion. This gives us more assurance in our previous conclusions.

RESIDUALS

Residuals are our principal tool for assessing how well a model fits the data. For regression models, residuals are used to assess the importance and relationship of a term in the model as well as to search for anomalous values. For generalized models we have the additional task of assessing and verifying the form of the variance as a function of the mean response.

Generalized models require a generalization of the residual which will be applicable to all the distributions which replace the normal or Gaussian distribution and which can be used in the same way as the normal residuals of the linear model. In fact, four different kinds of residuals are defined for use in assessing how well a model fits, in determining the form of the variance function, and in diagnosing problem observations.

- `"deviance"`: Deviance residuals are defined as:

$$r_i^D = \text{sign}(y_i - \hat{\mu}_i)\sqrt{d_i}$$

  where d_i is the contribution of the ith observation to the deviance.

  The deviance itself is $D = \sum_i (r_i^D)^2$. Consequently, these residuals are reasonable for use in detecting observations with unduly large influence in the fitting process, since they reflect the same criterion as used in the fitting.

- `"working"`: Working residuals are the difference between the *working* response and the linear predictor at the final iteration of the IRLS algorithm. They are defined as:

$$r_i^W = (y_i - \hat{\mu}_i)\frac{\partial \hat{\eta}}{\partial \hat{\mu}_i}$$

  These residuals are the ones you get when you extract the `residuals` component directly from the `glm` object.

- "pearson": The Pearson residuals are defined as:

$$r_i^P = \frac{y_i - \hat{\mu}_i}{\sqrt{V(\hat{\mu}_i)}}$$

  Their sum-of-squares

$$X^2 = \sum_{i=1}^{n} \frac{(y_i - \hat{\mu}_i)^2}{V(\hat{\mu}_i)}$$

  is the chi-squared statistic. Pearson residuals are a rescaled version of the working residuals. When proper account is taken of the associated weights, $r_i^P = \sqrt{w_i} r_i^W$.

- "response": The response residuals are simply $y_i - \hat{\mu}_i$.

You compute residuals for "glm" and "gam" objects with the residuals function, abbreviated resid (or resid for short) function. The type argument allows you to specify one of "deviance", "working", "pearson", or "response". By default you get the deviance residuals, so to plot the deviance residuals versus the fitted values of a model you just do:

```
> plot(fitted(glmobj), resid(glmobj))
```

Alternatively, to plot the Pearson residuals versus the fitted values you do:

```
> plot(fitted(glmobj), resid(glmobj, type = "pearson"))
```

Selecting which residual to plot is somewhat a matter of personal preference. The deviance residual is the default because a large deviance residual corresponds to an observation which does not fit the model well in the same sense that a large residual for the linear model doesn't fit well. You can find additional detail on residuals in McCullagh and Nelder (1989).

PREDICTION FROM THE MODEL

Prediction for generalized linear models, `glm`, and generalized additive models, `gam`, is similar to prediction for linear models. The only important point to remember is that for either of the generalized models predictions can be on one of two scales. You can predict:

1. on the scale of the linear predictor, which is the *transformed/* scale after applying the link function, or
2. on the scale of the original response variable.

Since prediction is based on the linear predictor, $\eta(x)$, computing predicted values on the scale of the original response effectively transforms the linear predictor evaluated at the predictor data back to the scale of the response via the inverse link function.

The `type` argument to either `predict.glm` or `predict.gam` allows you to choose one of three options for predictions:

- `"link"`: Computes predictions on the scale of the linear predictor (the *link* scale).
- `"response"`: Computes predictions on the scale of the response.
- `"terms"`: Computes a matrix of predictions on the scale of the linear predictor, one column for each term in the model.

Specifying `type = "terms"` allows you to compute the component of the prediction for each term separately. Summing the columns of the matrix and adding the constant (intercept) term is equivalent to specifying `type = "link"`.

Predicting the Additive Model of Kyphosis

As an example, consider the generalized additive model of Kyphosis modeled as smooths of `Start` and `Age`. Recall the fit was saved as `kyph.gam.start.age`:

```
> kyph.gam.start.age

Call:
gam(formula = Kyphosis ~ s(Start) + s(Age),
family = binomial, data = kyphosis)
Degrees of Freedom: 81 total; 72.09458 Residual
Residual Deviance: 48.41713
```

If we are interested in plotting the prediction surface over the range of the data we start by generating appropriate sequences of values for each predictor and storing them in a data frame with variable labels that correspond to the variables in the model:

```
> attach(kyphosis)
> kyph.margin <-
+ data.frame(Start = seq(from = min(Start),
+ to = max(Start), len = 40), Age =
+ seq(from = min(Age), to = max(Age), len = 40) )
```

Since a `gam` is additive, we need to do predictions only at the margins and then sum them together to form the entire prediction surface. We produce the marginal fits by specifying `type = "terms"`.

```
> margin.fit <- predict(kyph.gam.start.age, kyph.margin,
+ type="terms")
```

Now generate the surface for the marginal fits.

```
> kyph.surf <- outer(margin.fit[,1], margin.fit[,2], "+")
> kyph.surf <- kyph.surf + attr(margin.fit, "constant")
> kyph.surf <- binomial()$inverse(kyph.surf)
```

The first line adds the marginal pieces of the predictions together to create a matrix of surface values, the second line adds in the constant intercept term, and the third line applies the inverse link function to transform the predictions back to the scale of the original response. Now we produce the plot using the `persp` function (or `contour` or `image` if we wish):

```
> persp(kyph.margin[,1], kyph.margin[,2], kyph.surf,
+ xlab = "Start", ylab = "Age", zlab = "Kyphosis")
```

Figure 10.11 displays the resulting plot.

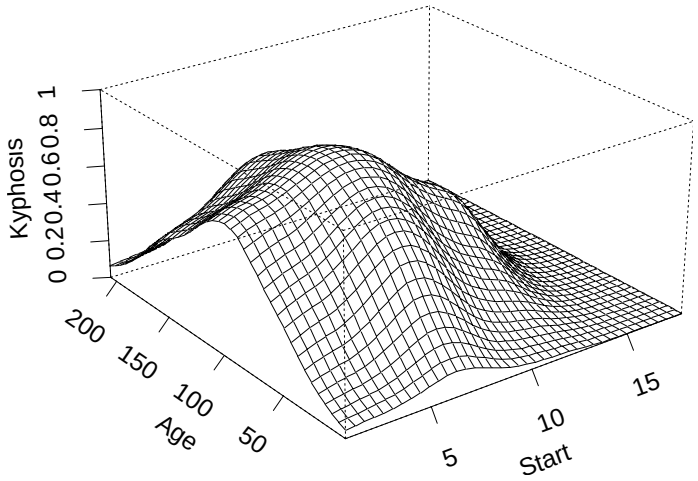

Figure 10.11: *Plot of the probability surface for developing Kyphosis based age in months and start position.*

Safe Prediction

Prediction for linear and generalized linear models is a two-step procedure.

1. Compute a model matrix using the new data where you want predictions.
2. Multiply the model matrix by the coefficients extracted from the fitted model.

This procedure works perfectly fine as long as the model has no *composite* terms which are dependent on some overall summary of a variable such as any of the following:

```
(x - mean(x))/sqrt(var(x))
(x - min(x))/diff(range(x))
poly(x)
bs(x)
ns(x)
```

The reason the procedure doesn't work for such composite terms is that the resulting coefficients are dependent on the summaries used in computing the terms. If the new data are different from the original data used to fit the model (which is more than likely when you provide new data), the coefficients are inappropriate. One way

around this problem is to eliminate such dependencies on data summaries. For example, change `mean(x)` and `var(x)` to their numeric values rather than computing them from the data at the time of fitting the model. For the spline functions, `bs` and `ns`, provide the knots explicity in the call to the function rather than letting the function compute them from the overall data. If the removal of dependencies on the overall data is possible, prediction can be made safe for new data. However, when the dependencies cannot be removed (for example, using `s` or `lo` in a `gam`), there is a function for doing prediction in as safe a way as possible given the need for generality. The function is `predict.gam`, which works as follows when new data is supplied:

1. A new data frame, `both.data`, is constructed by combining the data used to produce the fit, say `old.data`, and the new data in `new.data`.
2. The model frame and model matrix are constructed from the combined data frame `both.data`. The model matrix is separated into two pieces X^0 and X^n corresponding to the old and new data.
3. The parametric part of fit is refit using X^0.
4. The coefficients from this new fit are then applied to X^n to obtain the new predictions.
5. For `"gam"` objects with both parametric and nonparametric components, an additional step is taken to evaluate the fitted nonlinear functions at the new data values.

This procedure works perfectly for terms with `mean` and `var` in them as well as for `poly`. For other kinds of composite therms, it works approximately. For example, for `bs` knots are placed at equally spaced (in terms of percentiles) quantiles of the distribution of the predictor. Because the knots produced by the combined data will, in general, be different from the knots produced by the original data there will be some error in predicting the new data. If the old data and the new data have roughly the same distribution the error in predicting the new data should be small.

REFERENCES

Chambers, J.M. and Hastie, T.J. (1992). *Statistical Models in S.* Wadsworth and Brooks Cole Advanced Books and Software, Pacific Grove, CA.

Comizzoli, R.B. and Landwehr, J.M. and Sinclair, J.D. (1990). *Robust Materials and Processes: Key to Reliability*, 6:113-128.

Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models.* Chapman and Hall, London.

McCullagh, P. and Nelder, J.A. (1989). *Generalized Linear Models*, 2nd edition. Chapman and Hall, London.

Nelder, J.A. and Wedderburn, R.W.M. (1972). *Generalized linear models.* Journal of the Royal Statistical Society, Series A, 135:370-384.

LOCAL REGRESSION MODELS 11

INTRODUCTION

In both Chapter 8, Regression and Smoothing For Continuous Response Data, and Chapter 10, Generalizing the Linear Model, we discuss fitting curves or surfaces to data. In both of these earlier chapters, a significant limitation on the surfaces considered was that the effects of the terms in the model were expected to enter the model *additively*, without interactions between terms.

Local regression models provide much greater flexibility in that the model is fitted as a single smooth function of all the predictors. There are no restrictions on the relationships among the predictors.

Local regression models in S-PLUS are created using the `loess` function, which uses locally weighted regression smoothing, as described in the section Smoothing on page 213. In that section, the focus was on the smoothing function as an estimate of one predictor's contribution to the model. In this chapter, we use locally weighted regression to fit the complete regression surface.

FITTING A SIMPLE MODEL

As a simple example of a local regression model, we return to the `ethanol` data discussed in Chapter 8, Regression and Smoothing For Continuous Response Data. We start by considering only the two variables `NOx` and `E`. We smoothed these data with `loess.smooth` in the section Smoothing on page 213. Now we use `loess` to create a complete local regression model for the data.

We fit an initial model to the ethanol data as follows, using the argument `span = 1/2` to specify that each local neighborhood should contain about half of the observations:

```
> ethanol.loess <- loess(NOx ~ E, data = ethanol,
+ span = 1/2)
> ethanol.loess

Call:
loess(formula = NOx ~ E, data = ethanol, span = 1/2)

Number of Observations:          88
Equivalent Number of Parameters: 6.2
Residual Standard Error:         0.3373
Multiple R-squared:              0.92
Residuals:
     min   1st Q   median  3rd Q    max
-0.6656 -0.1805 -0.02148 0.1855 0.8656
```

The *equivalent number of parameters* gives an estimate of the complexity of the model. The number here, 4.3, indicates that the local regression model is somewhere between a cubic polynomial and a quartic polynomial in complexity. The default print method for `"loess"` objects also includes the residual standard error, multiple R^2, and a five number summary of the residuals.

DIAGNOSTICS: EVALUATING THE FIT

How good is our initial fit? The following function calls plot the `loess` object against a scatter plot of the original data:

```
> attach(ethanol)
> plot(ethanol.loess, xlim=range(E),
+ ylim=range(NOx,fitted(ethanol.loess)))
> points(E, NOx)
```

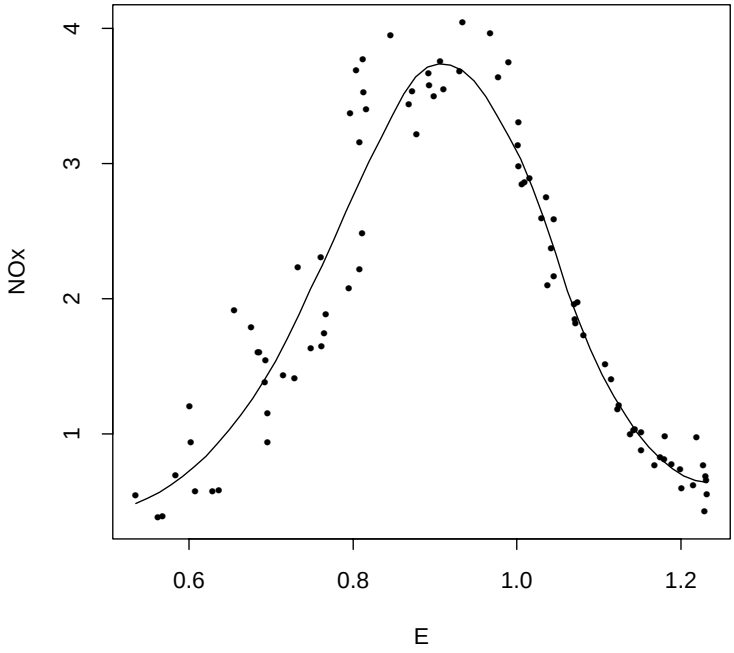

Figure 11.1: *Locally weighted smooth of ethanol data.*

The resulting figure, shown in Figure 11.1, captures the trend reasonably well. The following expressions plot the residuals against the predictor `E` to check for lack of fit:

```
> scatter.smooth(E, resid(ethanol.loess), span=1,
+ degree =1)
> abline(h=0)
```

The resulting plot, shown in Figure 11.2, indicates no lack of fit.

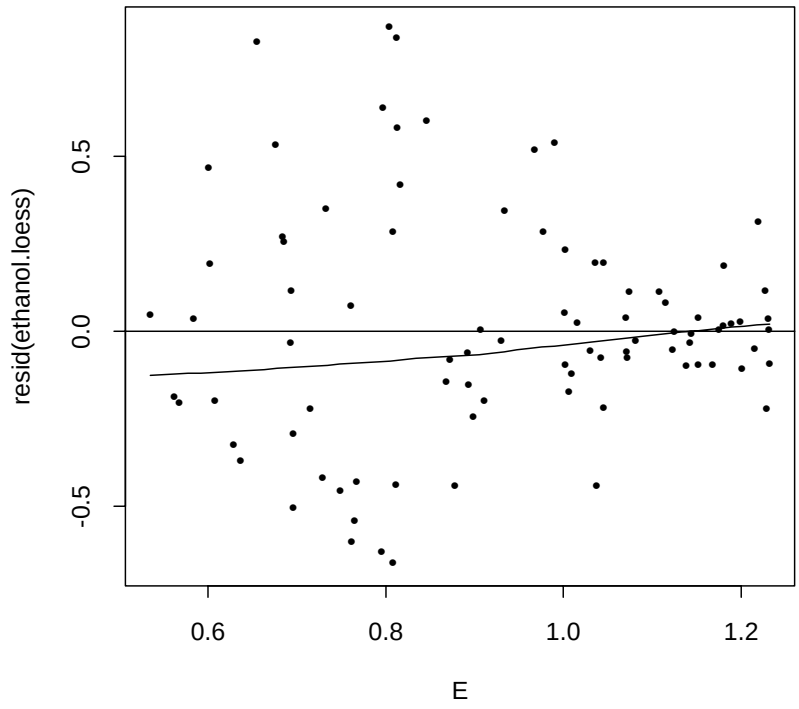

Figure 11.2: *Residual plot for* `loess` *smooth.*

Is there a surplus of fit? That is, can we increase the span of the data and still get a good fit? To see, let's refit our model, using `update`:

```
> ethanol.loess2 <- update(ethanol.loess, span=1)
> ethanol.loess2

Call:
loess(formula = NOx ~ E, data = ethanol, span = 1)
Number of Observations:          88
Equivalent Number of Parameters: 3.5
Residual Standard Error:         0.5126
Multiple R-squared:              0.81
Residuals:
     min   1st Q median  3rd Q    max
-0.9791 -0.4868 -0.064 0.3471 0.9863
```

By increasing the span, we reduce somewhat the equivalent number of parameters; this model is thus more *parsimonious* than our first model. We do seem to have lost some fit and gained some residual error. The diagnostic plots, shown in Figure 11.3, reveal a less satisfying fit in the main plot, and much obvious structure left in the residuals.

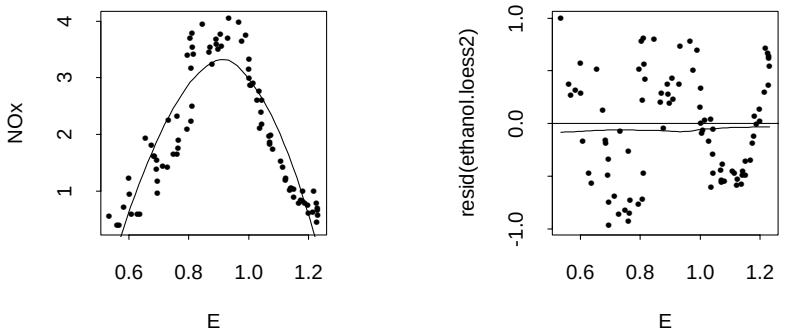

Figure 11.3: *Diagnostic plots for* `loess` *fit with span 1.*

The residuals are also more broadly spread than those of the first model. We confirm this with a call to `anova` as follows:

```
> anova(ethanol.loess2, ethanol.loess)

Model 1:
loess(formula = NOx ~ E, data = ethanol, span = 1)
Model 2:
loess(formula = NOx ~ E, data = ethanol, span = 1/2)
Analysis of Variance Table
      ENP     RSS     Test     F Value      Pr(F)
1     3.5 22.0840   1 vs 2       32.79 8.2157e-15
2     6.2  9.1685
```

The difference between the models is highly significant, so we stick with our original model.

EXPLORING DATA WITH MULTIPLE PREDICTORS

Conditioning Plots

The `ethanol` data set actually has three variables, with the compression ratio, `C`, of the engine as another predictor joining the equivalence ratio `E` and the response, nitric oxide emissions, `NOx`. A summary of the data is shown below:

```
> summary(ethanol)
```

```
      NOx               C                E         
Min.   :0.370   Min.   : 7.500   Min.   :0.5350  
1st Qu.:0.953   1st Qu.: 8.625   1st Qu.:0.7618  
Median :1.754   Median :12.000   Median :0.9320  
Mean   :1.957   Mean   :12.030   Mean   :0.9265  
3rd Qu.:3.003   3rd Qu.:15.000   3rd Qu.:1.1100  
Max.   :4.028   Max.   :18.000   Max.   :1.2320  
```

A good place to start an analysis with two or more predictors is a pairwise scatter plot, as generated by the `pairs` function:

```
> pairs(ethanol)
```

The resulting plot is shown in Figure 11.4. The top row shows the nonlinear dependence of `NOx` on `E`, and no apparent dependence of `NOx` on `C`. The middle plot in the bottom row shows `E` plotted against `C`–this plot reveals no apparent correlation between the predictors, and shows that the compression ratio `C` takes on only 5 distinct values.

Another useful plot for data with two predictors is the perspective plot. This lets us view the response as a surface over the predictor plane.

```
> persp(interp(E, C, NOx))
```

The resulting plot is shown in Figure 11.5.

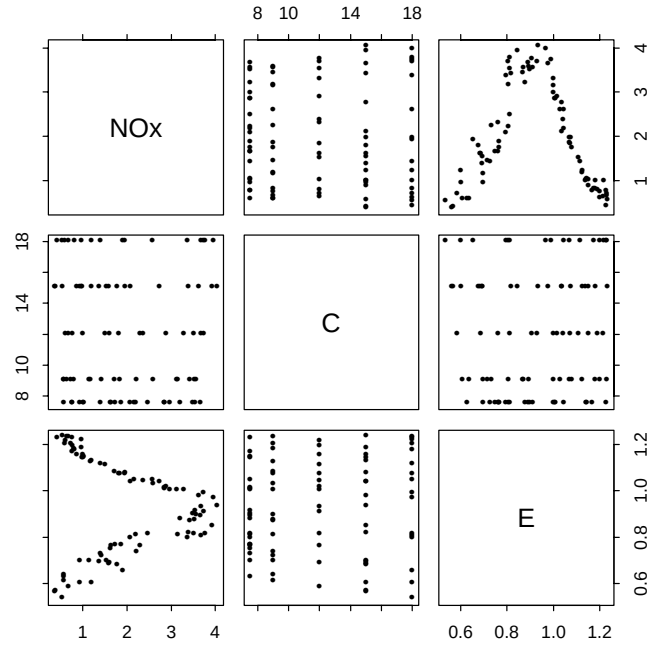

Figure 11.4: *Pairs plot of ethanol data.*

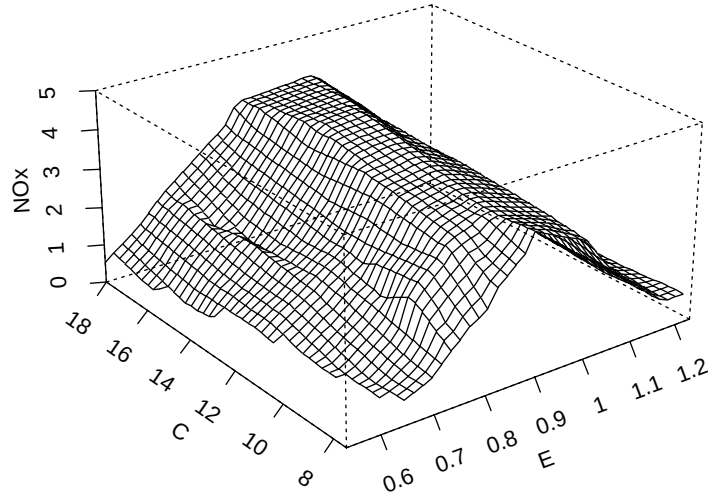

Figure 11.5: *Perspective plot of ethanol data.*

One conclusion we *cannot* draw from the pairwise scatter plot is that there is no effect of `C` on `NOx`. Such an effect might well exist, but be masked by the strong effect of `E`. Another type of plot, the *conditioning plot*, or *coplot*, can reveal such hidden effects.

A coplot shows how a response depends upon a predictor *given* other predictors. Basically, the idea is to create a matrix of *conditioning panels*; each panel graphs the response against the predictor for those observations whose value of the given predictor lie in an interval.

To create a coplot:

1. *(Optional)* Create the conditioning values. The `coplot` function creates default values if conditioning values are omitted, but they are not usually as good as those created specifically for the data at hand.
2. Use the `coplot` function to create the plot.

We discuss these steps in detail in the following subsections.

Creating Conditioning Values

How you create conditioning values depends on the nature of the values taken on by the predictor, whether continuous or discrete.

For continuous data, the conditioning values are intervals, created using the function `co.intervals`. For example, the following call creates nine intervals for the predictor `E`:

```
> E.intervals <- co.intervals(E, number = 9, overlap = 1/4)
```

For data taking on discrete values, the conditioning values are the sorted, unique values. For example, the following call creates the conditioning values for the predictor `C`:

```
> C.points <- sort(unique(C))
```

Constructing a Conditioning Plot

To construct a conditioning plot, use `coplot` using a formula with the special form `A ~ B | C`, where `A` is the response, `B` is the predictor of interest, and `C` is the given predictor. Thus, to see the effect of `C` on `NOx` given `E`, use the formula `NOx ~ C | E`.

In most cases, you also want to specify one or both of the following arguments:

- `given.values`: The conditioning values created above.

- `panel`: A function of `x` and `y` used to determine the method of plotting in the dependence panels. The default is `points`.

To create the conditioning plot shown in Figure 11.6:

```
> coplot(NOx ~ C | E, given.values = E.intervals)
```

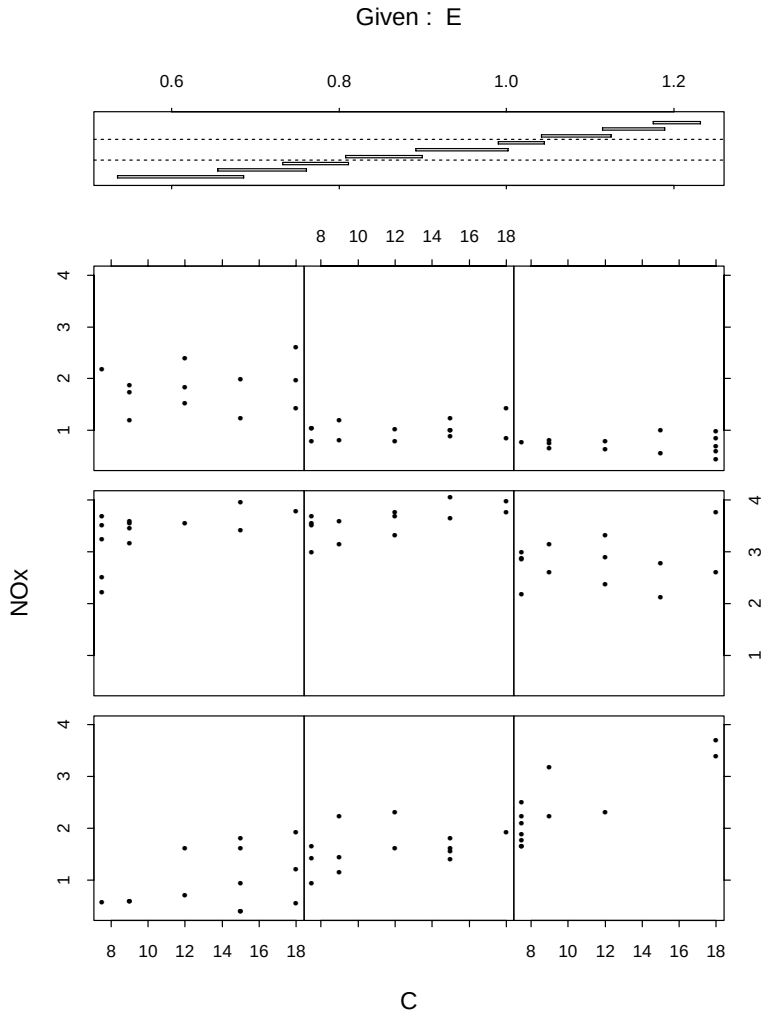

Figure 11.6: *Conditioning plot of ethanol data.*

Analyzing Conditioning Plots

To read the coplot, move from left to right, bottom to top. The scatter plots on the bottom row show an upward trend, while those on the upper two rows show a flat trend. We can more easily see the trend by using a smoothing function inside the conditioning panels, which we can do by specifying the `panel` argument to `coplot` as follows:

```
> coplot(NOx ~ C | E, given.values = E.intervals,
+ panel = function(x, y) panel.smooth(x, y,
+ degree = 1, span = 1))
```

The resulting plot is shown in Figure 11.7.

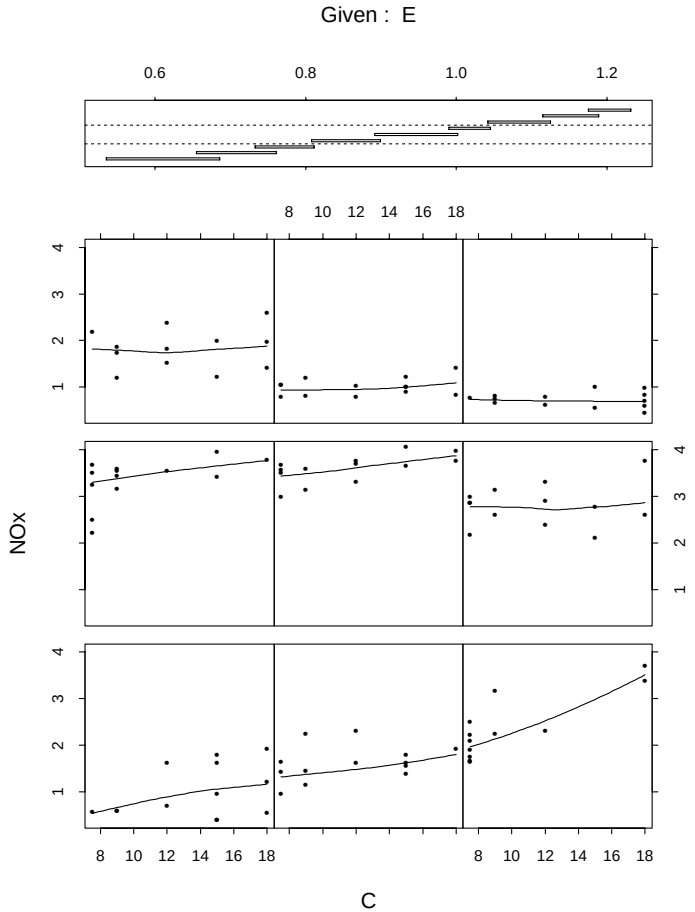

Figure 11.7: *Smooth conditioning plot of ethanol data.*

This plot clearly shows that for low values of E, NOx increases linearly with C, while for higher values of E, NOx remains constant with C.

Conversely, the coplot for the effects of E on NOx given C is created with the following call to coplot, and shown in Figure 11.8:

```
> coplot(NOx ~ E | C, given.values = C.points,
+ panel = function(x, y) panel.smooth(x,y, degree =2,
+ span = 2/3))
```

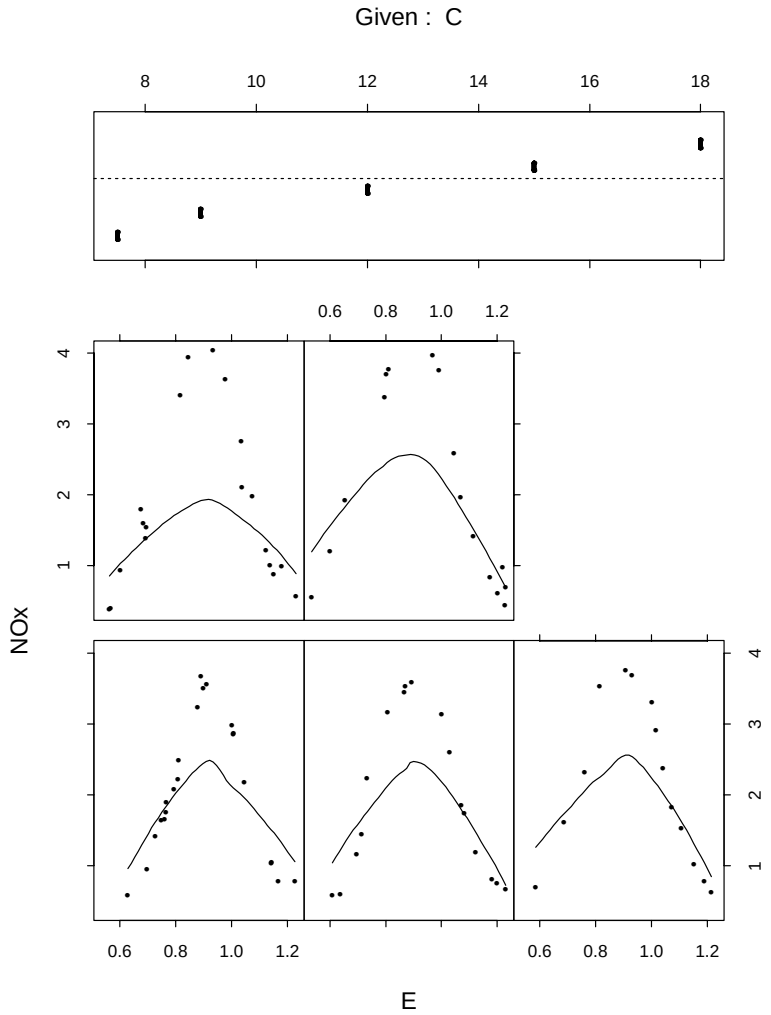

Figure 11.8: *Smooth conditioning plot of ethanol data, conditioned on* C.

Comparing the two coplots, we can see that `NOx` changes more rapidly as a function of `E` with `C` fixed than as a function of `C` with `E` fixed. Also, the variability of the residuals is small compared to the effect of `E`, but noticeable compared to the effect of `C`.

FITTING A MULTIVARIATE LOESS MODEL

We have learned quite a bit about the `ethanol` data without fitting a model: there is a strong nonlinear dependence of `NOx` on `E` and there is an interaction between `C` and `E`. We can use this knowledge to shape our initial local regression model. First, we specify a formula that includes as predictors both `E` and `C`, namely `NOx ~ C * E`. Then, we accept the default of local quadratic fitting to better model the nonlinear dependence.

```
> ethanol.m <- loess(NOx ~ C * E, data = ethanol)
> ethanol.m

Call:
loess(formula = NOx ~ C * E, data = ethanol)

Number of Observations:          88
Equivalent Number of Parameters: 9.4
Residual Standard Error:         0.3611
Multiple R-squared:              0.92
Residuals:
     min   1st Q   median 3rd Q    max
-0.7782 -0.3517 -0.05283 0.195 0.6338
```

We search for lack of fit by plotting the residuals against each of the predictors:

```
> par(mfrow=c(1,2))
> scatter.smooth(C, residuals(ethanol.m),span=1, deg=2)
> abline(h=0)
> scatter.smooth(E, residuals(ethanol.m),span=1, deg=2)
> abline(h=0)
```

The resulting plot is shown in Figure 11.9.

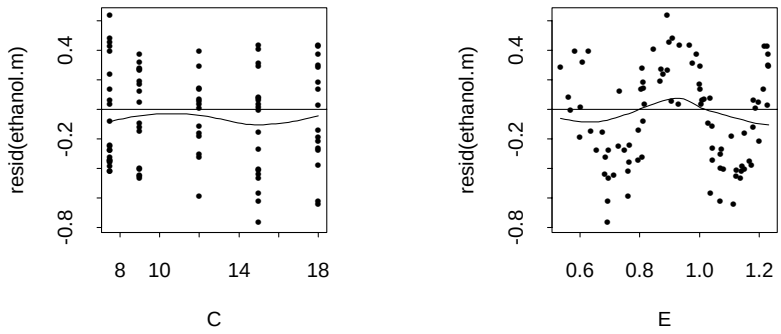

Figure 11.9: *Diagnostic plot for* `loess` *model of ethanol data.*

The right-hand plot shows considerable lack of fit, so we reduce the span from the default `0.75` to `0.4`:

```
> ethanol.m2 <- update(ethanol.m, span = .4)
> ethanol.m2

Call: loess(formula = NOx ~ C * E, data = ethanol,
span = 0.4)

Number of Observations:          88
Equivalent Number of Parameters: 15.3
Residual Standard Error:         0.2241
Multiple R-squared:              0.97
Residuals:
    min   1st Q   median  3rd Q    max
-0.4693 -0.1865 -0.03518 0.1027 0.3739
```

Repeating the commands for generating the diagnostic plots with `ethanol.m2` replacing `ethanol.m` yields the plot shown in Figure 11.10.

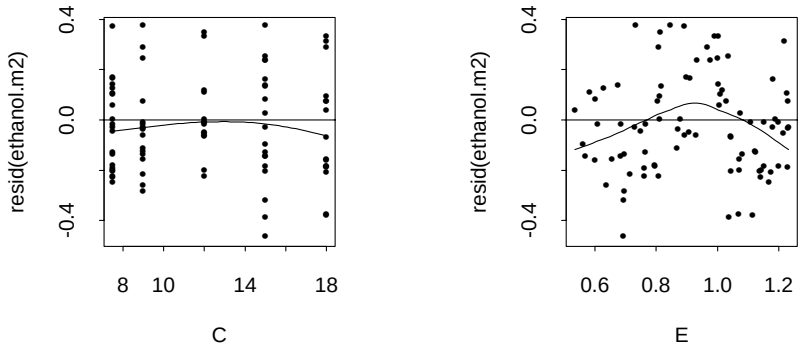

Figure 11.10: *Diagnostic plot for first revised model.*

The right-hand plot looks better but still has some quadratic structure, so we shrink the span still further, and try again:

```
> ethanol.m3 <- update(ethanol.m, span = .25)
> ethanol.m3

Call:
loess(formula = NOx ~ C * E, data = ethanol, span = 0.25)

Number of Observations:          88
Equivalent Number of Parameters: 21.6
Residual Standard Error:         0.1761
Multiple R-squared:              0.98
Residuals:
     min     1st Q  median   3rd Q    max
-0.3975 -0.09077 0.00862 0.06205 0.3382
```

Again, we create the appropriate residuals plots to check for lack of fit. The result is shown in Figure 11.11. This time the fit is much better.

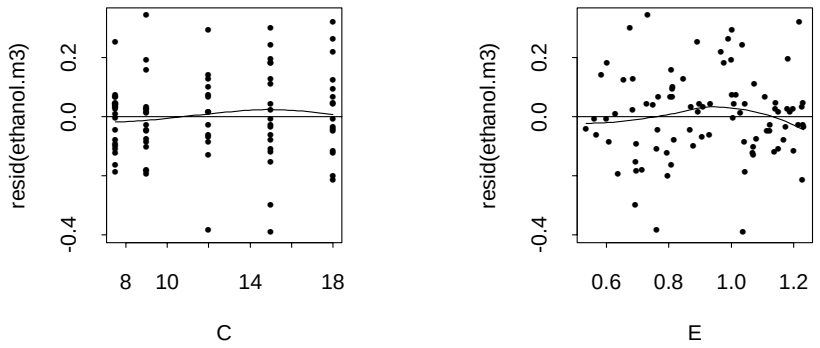

Figure 11.11: *Diagnostic plot for second revised model.*

Another check on the fit is provided by coplots using the residuals as the response variable:

```
> coplot(residuals(ethanol.m3) ~ C | E,
+ given = E.intervals,
+ panel= function(x,y)
+ panel.smooth(x,y, degree=1, span=1, zero.line=TRUE))
> coplot(residuals(ethanol.m3) ~ E | C, given = C.points,
+ panel= function(x,y)
+ panel.smooth(x,y, degree=1, span=1, zero.line=TRUE))
```

The resulting plots are shown in Figure 11.12 and Figure 11.13. The middle row of Figure 11.12 shows some anomalies–the residuals are virtually all positive. However, the effect is small, and limited in scope, so it can probably be ignored.

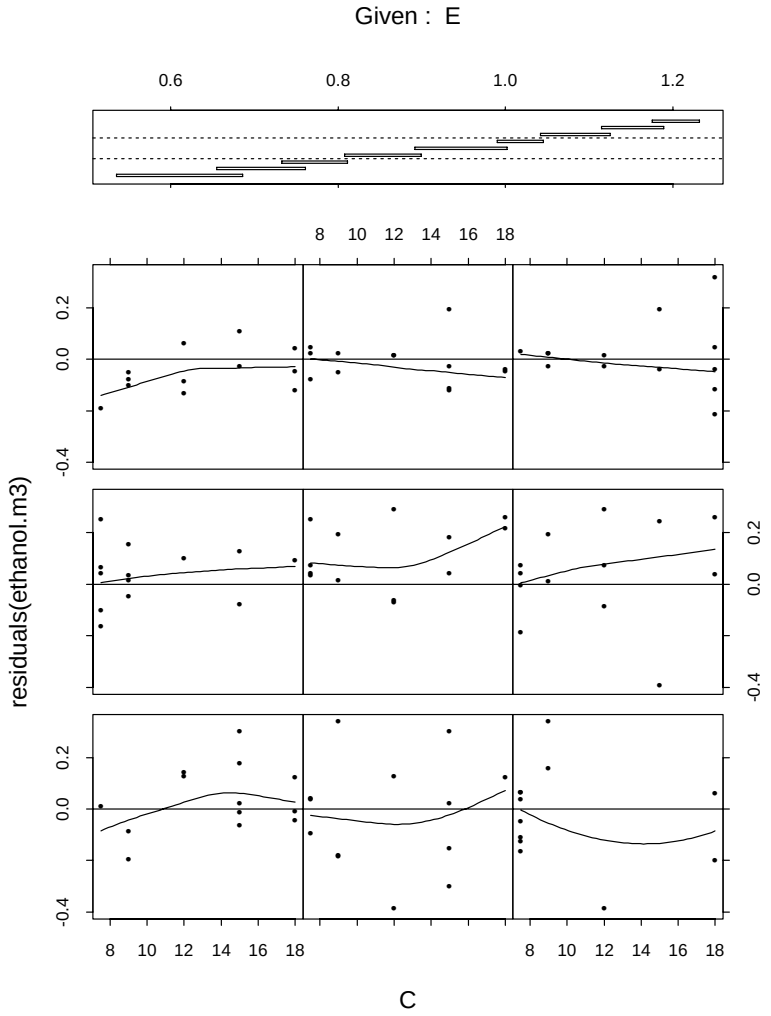

Figure 11.12: *Conditioning plot on* E *for second revised model.*

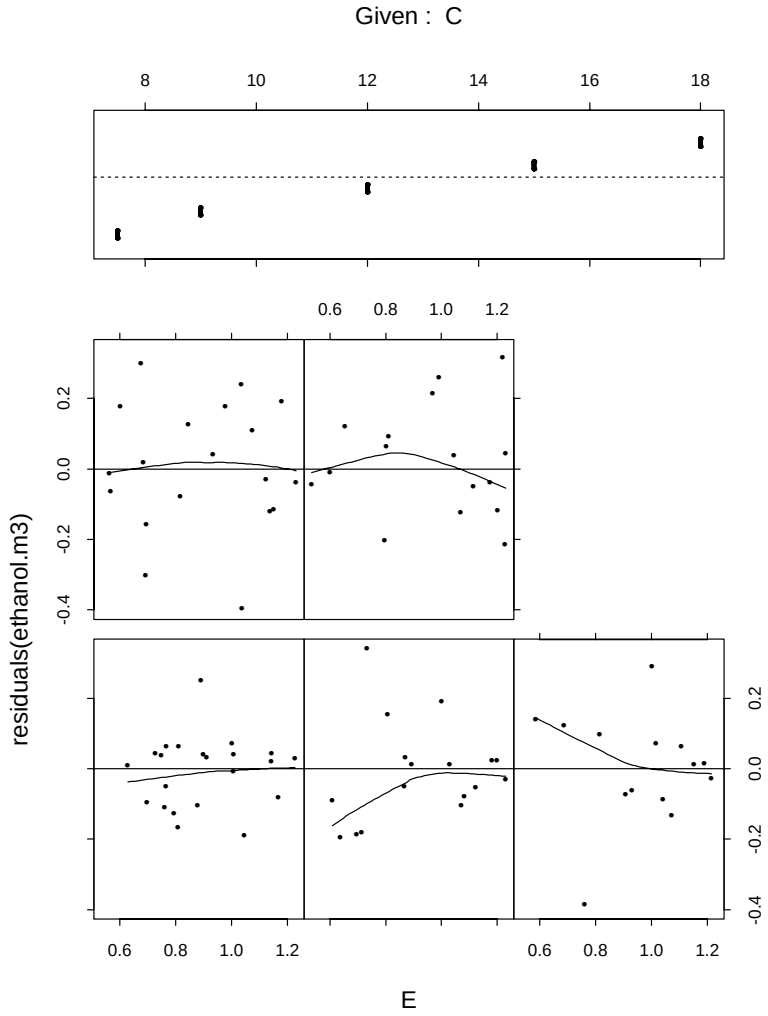

Figure 11.13: *Conditioning plot on* `C` *for second revised model.*

As a final test, we make several more diagnostic plots to check the distribution of the error terms (Figure 11.14):

```
> par(mfrow=c(2,2))
> plot(fitted(ethanol.m3), sqrt(abs(resid(ethanol.m3))))
> plot(C, sqrt(abs(resid(ethanol.m3))))
> plot(E, sqrt(abs(resid(ethanol.m3))))
> qqnorm(resid(ethanol.m3))
```

```
> qqline(resid(ethanol.m3))

NULL
```

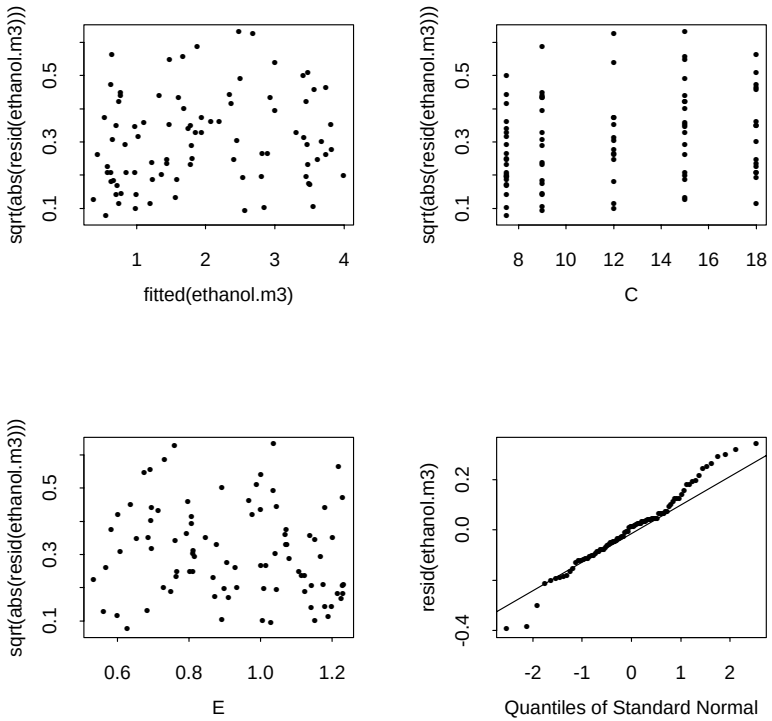

Figure 11.14: *Diagnostic plots for second revised model.*

The model passes these checks–the errors appear to be Gaussian, or nearly so.

LOOKING AT THE FITTED MODEL

Examining the fitted model graphically is no less important than graphically examining the data. One way to test the model is to compare the predicted surface with the data surface shown in Figure 11.5 . We can create the corresponding perspective plot for the model as follows. First, define an evenly-spaced grid of points spanning the range of `E` and `C`:

```
> newC <- seq(from = min(C), to = max(C), length = 40)
> newE <- seq(from = min(E), to = max(E), length = 40)
> new.ethanol <- expand.grid(E = newE, C = newC)
```

The `expand.grid` function returns a data frame with 1600 rows and 2 columns, corresponding to all possible combinations of `newC` and `newE`. We can then use `predict` with the fitted model and these new data points to calculate predicted values for each of these grid points:

```
> eth.surf <- predict(ethanol.m3, new.ethanol)
```

The perspective plot of the surface is then created readily as follows:

```
> persp(newE, newC, eth.surf, xlab = "E",
+ ylab = "C")
```

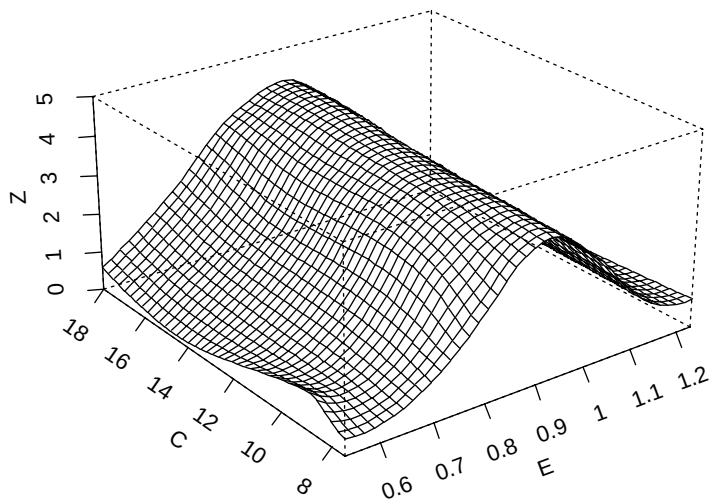

Figure 11.15: *Perspective plot of the model.*

The resulting plot is shown in Figure 11.15. Not surprisingly, the surfaces look quite similar, with the model surface somewhat smoother than the data surface. The data surface has a noticeable wrinkle for `E` ≈ 0.7, `C` ≈ 14. This wrinkle is smoothed out in the model surface. Another graphical view is probably worthwhile.

The default graphical view for `"loess"` objects with multiple predictors is a set of coplots, one per predictor, created using the `plot` function.

```
> par(ask=T)
> plot(ethanol.m3, confidence = 7)
```

The resulting plots are shown in Figure 11.16 and Figure 11.17. One feature that is immediately apparent, and somewhat puzzling, is the curvy form of the bottom row of Figure 11.16. Our preliminary coplots revealed that the dependence of `NOx` on `C` was approximately linear for small values of `E`. Thus, the model as fitted has a noticeable departure from our understanding of the data.

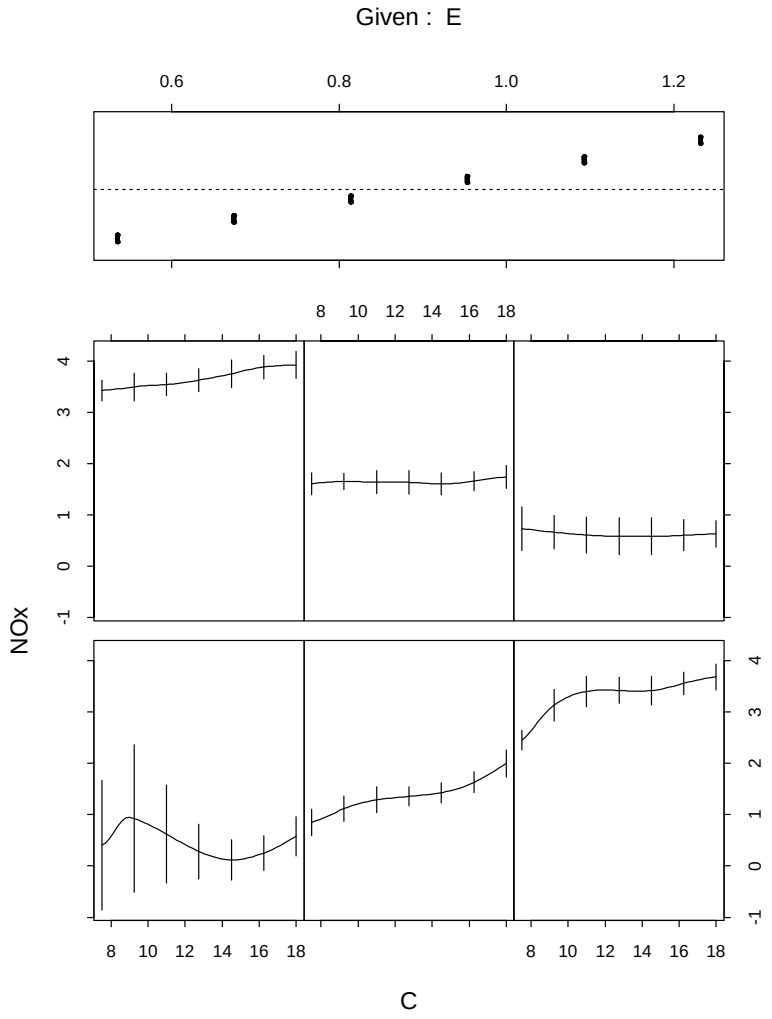

Figure 11.16: *Default conditioning plot of the model, first predictor.*

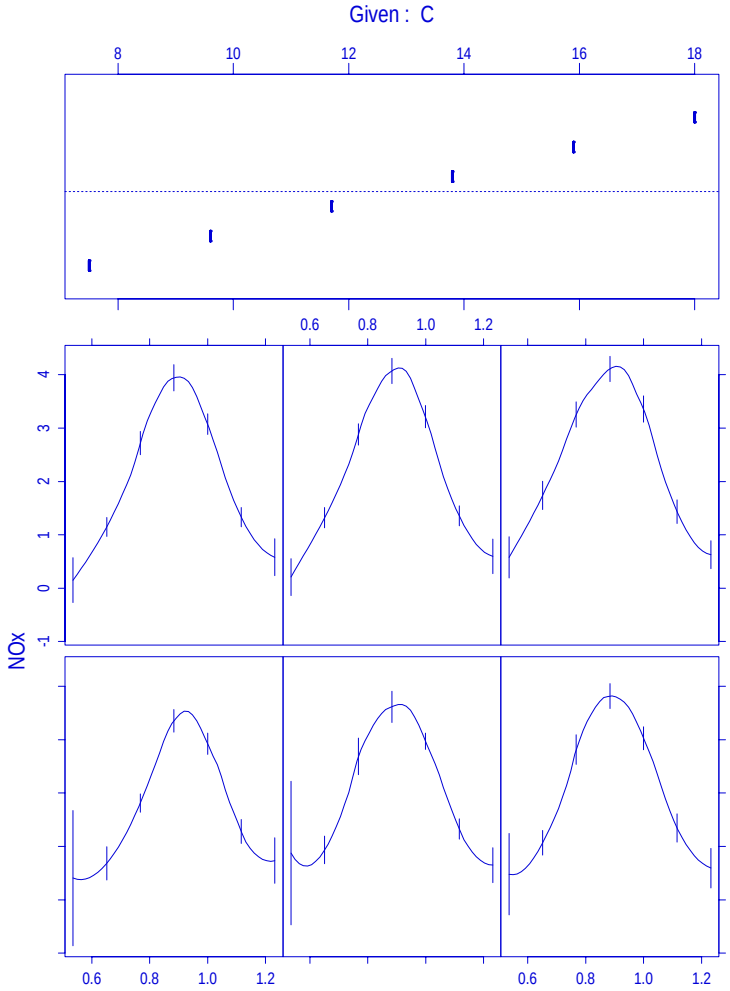

Figure 11.17: *Default conditioning plot of the model, second predictor.*

IMPROVING THE MODEL

The model in `ethanol.m3` is fit using local quadratic fitting for all terms corresponding to `C*E`. This means that the model contains the following fitting variables: a constant, `E`, `C`, `EC`, C^2, and E^2. However, our original look at the data led us to believe that the effect of `C` was piecewise linear; it thus makes sense to fit `C` *parametrically*, and drop the quadratic term. We can make these changes using the `update` function as follows:

```
> ethanol.m4 <- update(ethanol.m3, drop.square="C",
+ parametric = "C")
> ethanol.m4

Call:
loess(formula = NOx ~ C * E, span = 0.25, parametric = "C",
drop.square = "C")

Number of Observations:          88
Equivalent Number of Parameters: 18.2
Residual Standard Error:         0.1808
Multiple R-squared:              0.98
Residuals:
     min     1st Q    median   3rd Q     max
-0.4388 -0.07358 -0.009093 0.06616 0.5485
```

The offending coplot, Figure 11.18 and Figure 11.19, now shows the appropriate linear fit, and we have introduced no lack of fit, as shown by the residuals plots in Figure 11.20.

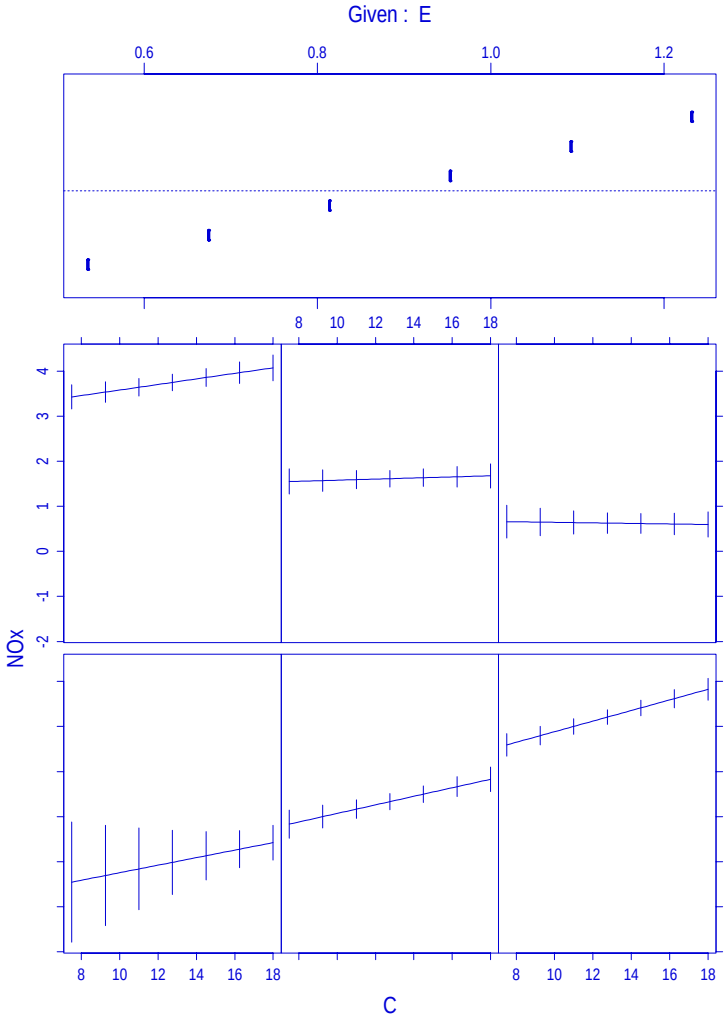

Figure 11.18: *Default conditioning plot of improved model, first predictor.*

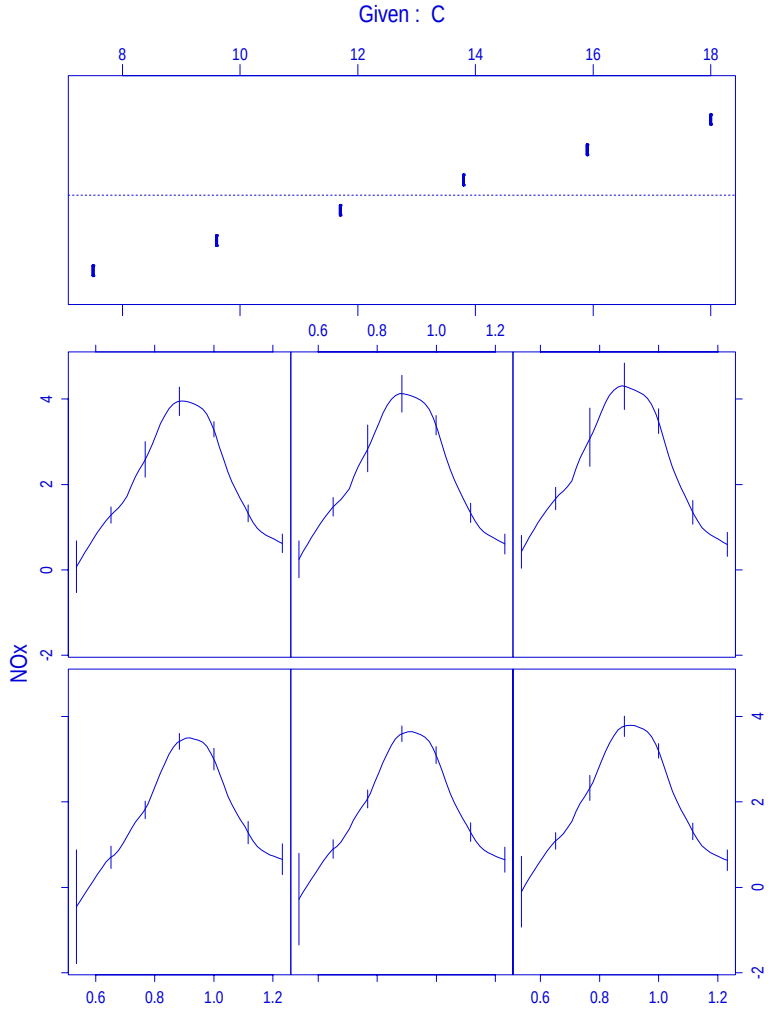

Figure 11.19: *Default conditioning plot of improved model, second predictor.*

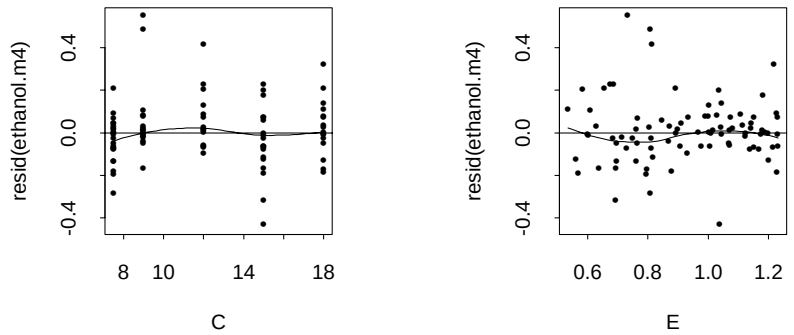

Figure 11.20: *Residual plot of improved model.*

In fact, comparing the plot of residuals against `E` for the latest model with that for `ethanol.m3` (Figure 11.21) indicates we may be able to increase the span for the latest model and not introduce any lack of fit:

```
> ethanol.m5 <- update(ethanol.m4, span = 1/2)
> ethanol.m5

Call:
loess(formula = NOx ~ C * E, span = 1/2, parametric = "C",
drop.square = "C")

Number of Observations:          88
Equivalent Number of Parameters: 9.2
Residual Standard Error:         0.1842
Multiple R-squared:              0.98
Residuals:
     min   1st Q  median   3rd Q    max
-0.5236 -0.0972 0.01386 0.07326 0.5584
```

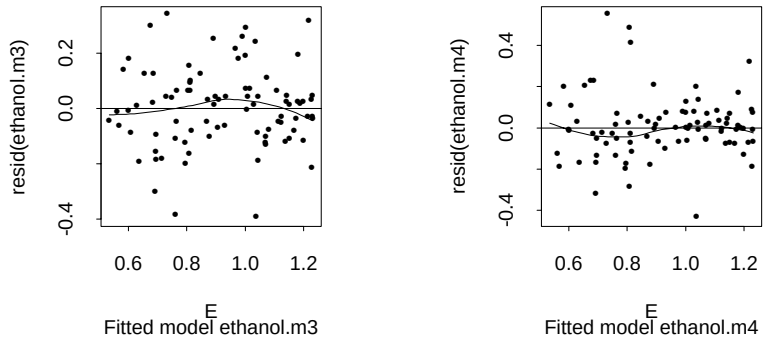

Figure 11.21: *Comparison of residual plots for original and improved models.*

We gain a *much* more parsimonious model–the Equivalent Number of Parameters drop from approximately 18 to about 9. An *F*-test using `anova` shows no significant difference between our first acceptable model and the latest, more parsimonious model:

```
> anova(ethanol.m3,ethanol.m5)

Model 1:
loess(formula = NOx ~ C * E, span = 0.25)
Model 2:
loess(formula = NOx ~ C * E, span = 1/2, parametric = "C",
drop.square = "C")
Analysis of Variance Table
      ENP     RSS     Test      F Value     Pr(F)
1    21.6  1.7999   1 vs 2         1.42   0.16486
2     9.2  2.5433
```

CLASSIFICATION AND REGRESSION TREES

12

INTRODUCTION

Tree-based modeling is an exploratory technique for uncovering structure in data, increasingly used for:

- devising prediction rules that can be rapidly and repeatedly evaluated
- screening variables
- assessing the adequacy of linear models
- summarizing large multivariate datasets

Tree-based models are useful for both classification and regression problems. In these problems, there is a set of classification or predictor variables (x), and a single-response variable (y).

If y is a factor, *classification* rules are of the form:

if $x_1 \leq 2.3$ *and* $x_3 \in \{A, B\}$

then y is most likely to be in level 5.

If y is numeric, *regression* rules for description or prediction are of the form:

if $x_2 \leq 2.3$ *and* $x_9 \in \{C, D, F\}$ *and* $x_5 \leq 3.5$

then the predicted value of y is 4.75.

A classification or regression tree is the collection of many such rules displayed in the form of a binary tree, hence the name. The rules are determined by a procedure known as *recursive partitioning*. Tree-based models provide an alternative to linear and additive models for regression problems, and to linear and additive logistic models for classification problems.

Compared to linear and additive models, tree-based models have the following advantages:

- Easier to interpret when the predictors are a mix of numeric variables and factors.
- Invariant to monotone re-expressions of predictor variables.
- More satisfactorily treat missing values.

- More adept at capturing nonadditive behavior.
- Allow more general (that is, other than of a particular multiplicative form) interactions between predictor variables.
- Can model factor response variables with more than two levels.

GROWING TREES

We describe the tree-growing function `tree` by presenting several examples. The `tree` function generates objects of class `"tree"`. This function automatically decides whether to fit a regression or classification tree, according to whether the response variable is numeric or a factor. We also show two types of displays, generated by generic functions: a tree display produced by `plot` and a table produced by `print`.

In general, the response *y* and predictors *x* may be any combination of numeric or factor types. In fact, the predictors can be a mix of numeric *and* factor. However, no factor predictor can have no more than 32 levels, and no factor response can have more than 128 levels. In both of the examples below, the predictors are all numeric. The numeric response example illustrates a regression tree. The factor response example illustrates a classification tree.

Numeric Response and Predictor

In the first example, we grow a *regression* tree relating the numeric response `Mileage` to the predictor variable `Weight` from the data frame `car.test.frame`. The resulting tree is given the name `auto.tree`, which is then plotted by the generic `plot` function and labeled by the generic `text` function (see Figure 12.1).

```
> attach(car.test.frame)
> auto.tree <- tree(Mileage ~ Weight, car.test.frame)
> plot(auto.tree,type="u")
> text(auto.tree)
> title("A Tree-Based Model\nfor Mileage versus Weight")
```

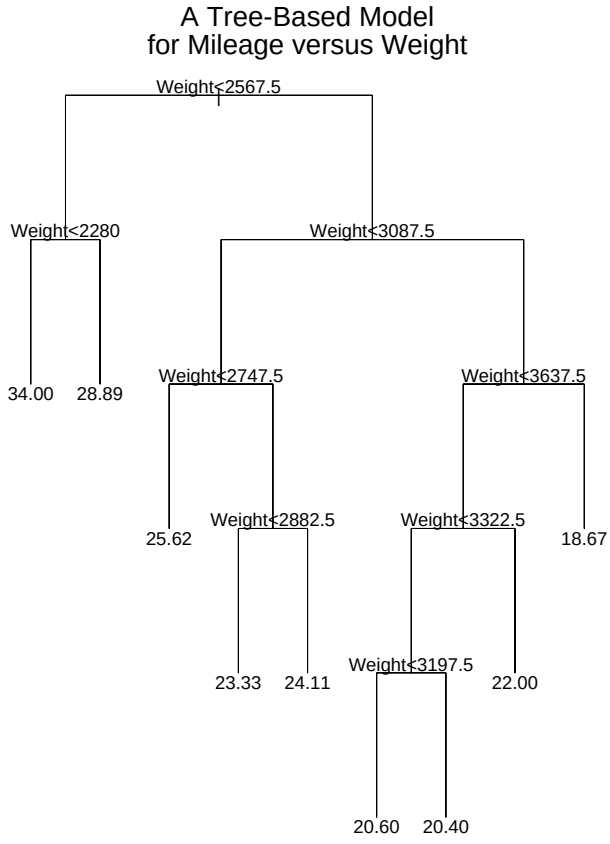

Figure 12.1: *Display of a tree-based model with a numeric response,* `Mileage`*, and one numeric predictor,* `Weight`.

In describing tree-based models, the terminology mimics real trees:

- rootThe top node of the tree
- leafA terminal node of the tree
- splitA rule for creating new branches

In growing a tree, the binary partitioning algorithm recursively splits the data in each node until either the node is homogeneous or the node contains too few observations (≤ 5, by default).

In order to *predict* mileage from weight, one follows the path from the root, to a leaf, according to the splits at the interior nodes. The tree in Figure 12.1 is interpreted in the following way:

- Automobiles are first split according to whether they weigh less than 2567.5 pounds.
- If so, they are again split according to weight being less than 2280 pounds.
- Lighter cars ($<$ 2280 pounds) have a predicted mileage of 34 mpg.
- Heavier cars ($2280 \leq$ `Weight` ≤ 2567.5) have a mileage of 28.9 mpg.
- For those automobiles weighing more than 2567.5 pounds, seven weight classes are formed.
- The predicted mileage ranges from a high of 25.6 mpg to a low of 18.7 mpg.
- Overall, heavier cars get poorer mileage than lighter cars.
- It appears that doubling the weight of an automobile approximately halves its mileage.

Factor Response and Numeric Predictor

In this classification example, we model the probability of developing Kyphosis, using the `kyphosis` data frame with predictors `Age`, `Start`, and `Number`.

First, use boxplots to plot the distributions of the predictor variables as a function of `Kyphosis` in Figure 12.2. `Start` appears to be the single best predictor of `Kyphosis` since Kyphosis is more likely to be present among individuals with Start $\leq$ 12.

```
> kyph.tree <- tree(Kyphosis ~ Age + Number + Start,
+ data = kyphosis)
```

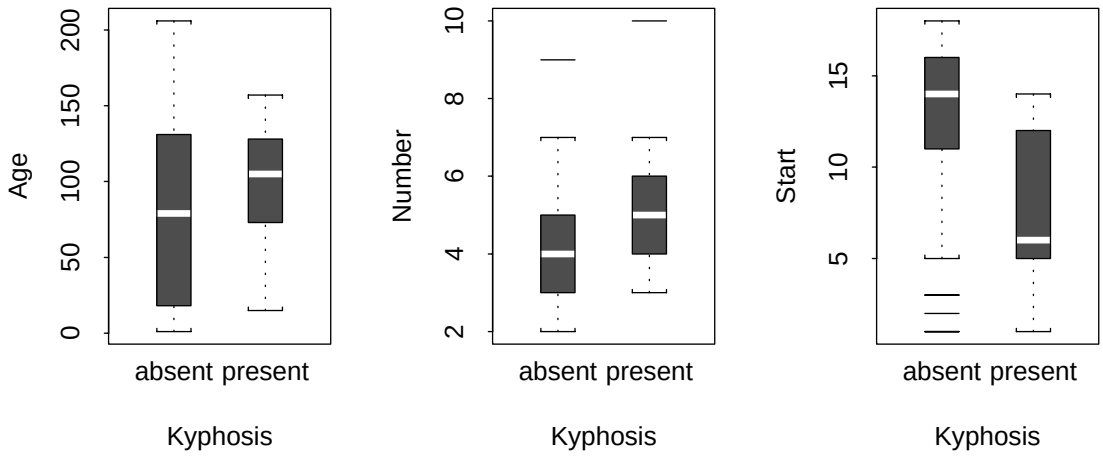

Figure 12.2: *Boxplots of the predictors of* `Kyphosis`.

Since `Kyphosis` is a factor response, the result `kyph.tree` is a *classification* tree.

Either the `formula` or `data` arguments to the `tree` function may be missing. Without the `formula` argument, a tree is constructed from the data frame using the first variable as the response. Hence, the Kyphosis example could have been constructed as follows:

```
> auto.tree <- tree(car.test.frame)
```

Without the `data` argument, the variables named in `formula` are expected to be in the search list. The Kyphosis tree could also have been grown with

```
> attach(car.test.frame)
> auto.tree <- tree(Mileage ~ Weight)
```

The only meaningful *operator* on the right side of a formula is "+". Since tree-based models are invariant to monotone re-expressions of individual predictor variables, functions like `log`, `I`, and `^` have little use. Also, tree-based models capture interactions without explicit specification.

This time, we display the fitted tree using the generic function, `print`, which is called automatically simply by typing the name of the tree object. This tabular representation is most useful when the details of the fitting procedure are of interest. Indentation is added as a key to the underlying structure.

```
> kyph.tree

node), split, n, deviance, yval, (yprob)
      * denotes terminal node
1) root 81 83.230 absent ( 0.7901 0.20990 )
  2) Start<12.5 35 47.800 absent ( 0.5714 0.42860 )
    4) Age<34.5 10 6.502 absent ( 0.9000 0.10000 )
    8) Age<16 5 5.004 absent ( 0.8000 0.20000 ) *
    9) Age>16 5 0.000 absent ( 1.0000 0.00000 ) *
  5) Age>34.5 25 34.300 present ( 0.4400 0.56000 )
   10) Number<4.5 12 16.300 absent ( 0.5833 0.41670 )
     20) Age<127.5 7 8.376 absent ( 0.7143 0.28570 ) *
     21) Age>127.5 5 6.730 present ( 0.4000 0.60000 ) *
   11) Number>4.5 13 16.050 present ( 0.3077 0.69230 )
     22) Start<8.5 8 6.028 present ( 0.1250 0.87500 ) *
     23) Start>8.5 5 6.730 absent ( 0.6000 0.40000 ) *
3) Start>12.5 46 16.450 absent ( 0.9565 0.04348 )
  6) Start<14.5 17 12.320 absent ( 0.8824 0.11760 )
   12) Age<59 5 0.000 absent ( 1.0000 0.00000 ) *
   13) Age>59 12 10.810 absent ( 0.8333 0.16670 )
     26) Age<157.5 7 8.376 absent ( 0.7143 0.28570 ) *
     27) Age>157.5 5 0.000 absent ( 1.0000 0.00000 ) *
  7) Start>14.5 29 0.000 absent ( 1.0000 0.00000 ) *
```

The first number in each row of the output is a node number. The nodes are numbered to index the tree for quick identification. For a *full* binary tree, the nodes at depth d are integers n, $2^d \le n < 2^{d+1}$. Usually, a tree is *not* full, but the numbers of the nodes that *are* present are the same as they would be in a full tree.

In the `print` output, the nodes are ordered according to a depth-first traversal of the tree, printed output.

Let us first examine one row of the output:

```
2) Start<12.5 35 47.800 absent ( 0.5714 0.42860 )
```

This row is for node 2. Following the node number is the split, `Start < 12.5`. This states the the observations in the parent (root) node with `Start < 12.5` were put into node 2.

The next number after the split is the number of observations, 35. The number 47.8 is the *deviance*, the measure of node heterogeneity used in the tree-growing algorithm. A perfectly homogeneous node has deviance zero. The fitted value, `yval`, of the node is `absent`. Finally, the numbers in parentheses (0.5714 0.42860), `yprob`, are the estimated probabilities of the observations in that node not having, and having, kyphosis. Therefore, the observations with `Start < 12.5` have a 0.5714 chance of not having kyphosis under this tree model.

An interpretation of the table follows:

- The split on `Start` partitions the 81 observations into groups of 35 and 46 individuals (nodes 2 and 3) with probability of `Kyphosis` 0.429 and 0.043, respectively.
- The group at node 2 is then partitioned into groups of 10 and 25 individuals (nodes 4 and 5) depending on whether `Age` is less than 34.5 years or not.
- The group at node 4 is divided in half depending on whether `Age` is less than 16 or not. If `Age` $>$ 16 none of the individuals have Kyphosis (probability of Kyphosis is 0). These subgoups are divided no further.
- The group at node 5 is subdivided into groups of size 12 and 13 depending on whether or not Number is less than 4.5. The respective probabilities of Kyphosis for these groups is 0.417 and 0.692.
- The procedure continues, yielding 10 distinct groups with probabilities of Kyphosis ranging from 0.0 to 0.875.
- Asterisks signify *terminal* nodes; that is, those that are not split.

DISPLAYING TREES

The generic functions `print`, `plot`, and `summary` work as expected for `"tree"` objects. We have already encountered the first two functions in the examples above. A further interesting feature of `plot` is that an optional `type` argument controls node placement. The `type` argument can have either of the two values:

- "" produces nonuniform spacing as the default. The more important the parent split, the further the children node pairs are spaced from their parents.
- "u" produces uniform spacing.

In the car mileage example, we used uniform spacing in order to label the tree. However, if the goal is tree simplification, we gain insight into the relative importance of the splits by using the default `type`, that is, nonuniform spacing. This is shown in Figure 12.3.

When you first plot the tree using `plot`, the nodes and splits will be displayed without any text labels. The generic `text` function, described in the *User's Guide*, uses the same arguments to rotate and adjust text in tree plots that it uses with most other types of plots.

The `summary` function has a tree-specific method which indicates the tree type (regression/classification), a record of how the tree was created, the residual mean deviance, and other information.

The residual deviance is the sum, over all the observations, of terms which vary according to type (regression/classification) of tree. The residual mean deviance is then obtained after dividing by the degrees of freedom (number of observations minus the number of terminal nodes).

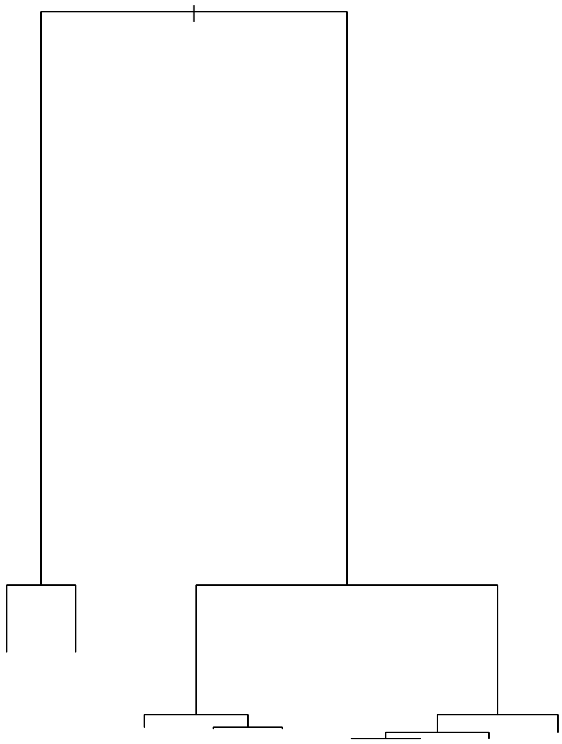

Figure 12.3: *Plot of the car mileage tree with non-uniform node placement.*

The following summary is typical for *regression*:

```
> summary(auto.tree)

Regression tree:
tree(formula = Mileage ~ Weight, data = car.test.frame)
Number of terminal nodes: 9
Residual mean deviance: 4.289 = 218.7 / 51
Distribution of residuals:
   Min. 1st Qu. Median Mean 3rd Qu. Max.
 -3.889 -1.111 0 0 1.083 4.375
```

The regression tree has nine terminal nodes. Under a normal (Gaussian) assumption, the terms in the residual mean deviance are the squared differences between the observations and the predicted

values. See the section Prediction and Residuals for a discussion of prediction and residuals. The `summary` function also summarizes the distribution of residuals.

The following summary is typical for *classification* trees:

```
> summary(kyph.tree)

Classification tree:
tree(formula = Kyphosis ~ Age + Number + Start)
Number of terminal nodes: 10
Residual mean deviance: 0.5809 = 41.24 / 71
Misclassification error rate: 0.1235 = 10 / 81
```

Note that, for classification trees, the `summary` function gives the misclassification error rate instead of distribution of residuals. First, predicted classifications are obtained as described in the section Prediction and Residuals. The error rate is then obtained by counting the number of misclassified observations, and dividing by the number of observations. The terms in the residual mean deviance are based on the multinomial distribution (see Chambers and Hastie (1992)).

PREDICTION AND RESIDUALS

Once a tree is grown, an important use of the fitted tree is to predict the value of the response variable for a set of predictor variables.

For concreteness, consider just one observation x on the predictor variables. In prediction, the splits direct x through the tree. The *prediction* is taken to be the the `yval` at the deepest node reached. Usually this corresponds to a leaf node. However, in certain situations, a prediction may reside in a nonterminal node (Chambers and Hastie (1992)). In particular this may happen if missing values occur in x, and the tree was grown with only complete observations.

The generic function `predict` has a tree-specific method. It takes a tree object and, optionally, a data frame as arguments. If the data frame is not supplied, `predict` returns the fitted values for the data originally used to construct the tree. The function returns predicted values either as a vector (the default) or a tree object (`type = "tree"`).

The residuals can then be obtained either by subtracting the fitted values from the response variable, or directly using the function `residuals`. Figure 12.4 presents a plot of the residuals versus the predicted values and a normal probability of the residuals for the `auto.tree` model.

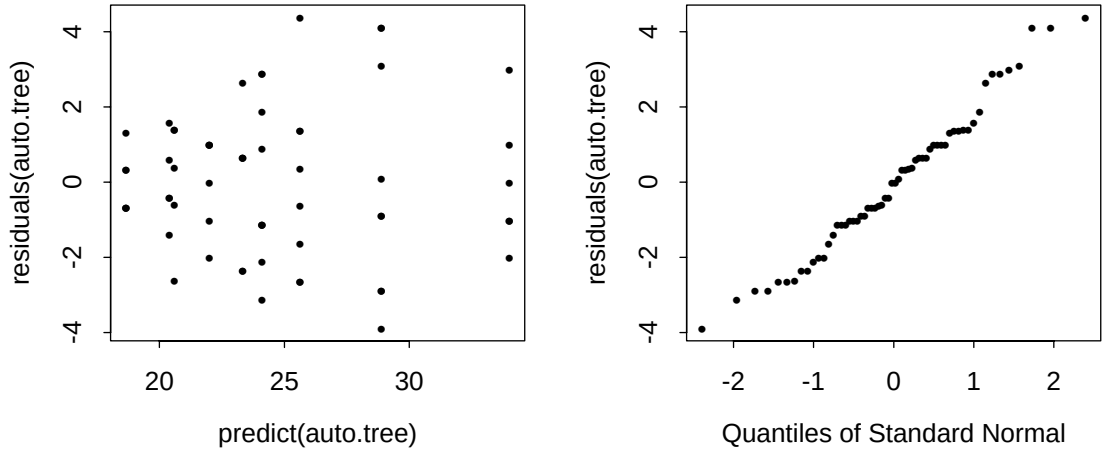

Figure 12.4: *Residuals versus predicted values and a normal probability plot of the residuals for a* `"tree"` *object.*

MISSING DATA

Missing values, NAs, can occur either in data used to build trees, or in a set of predictors for which the value of the response variable is to be predicted.

For data used to build trees, the `tree` function permits NAs only in predictor variables, but only if the argument `na.action = na.tree.replace` or `na.action = na.tree.replace.all`. For any predictor with missing values, the `na.tree.replace` function creates a new factor variable, with an added level named `"NA"` for the `NA`s. However, it leaves numeric predictors alone even if they have `NA`s. The `na.tree.replace.all` function behaves like `na.tree.replace` for factor predictors, and for numeric predictors with `NA`s, it converts them to factors (based on quantiles) and then adds a separate level for `NA`s.

In prediction, suppose an observation is missing a value for the variable *V*. Further, suppose there were no missing values for *V* in the training data. The observation follows its path down the tree until it encounters a node whose split is based upon *V*. The prediction is then taken to be the `yval` at that node. If values of several variables are missing, the observation stops at the first such variable split encountered.

To clarify this, let us return to the automobile example, where some of the data are missing values on the variable `Reliability`. We first fit a tree on the data with *no* missing values. The resulting tree is displayed in Figure 12.5. Notice the split on the variable `Reliability`.

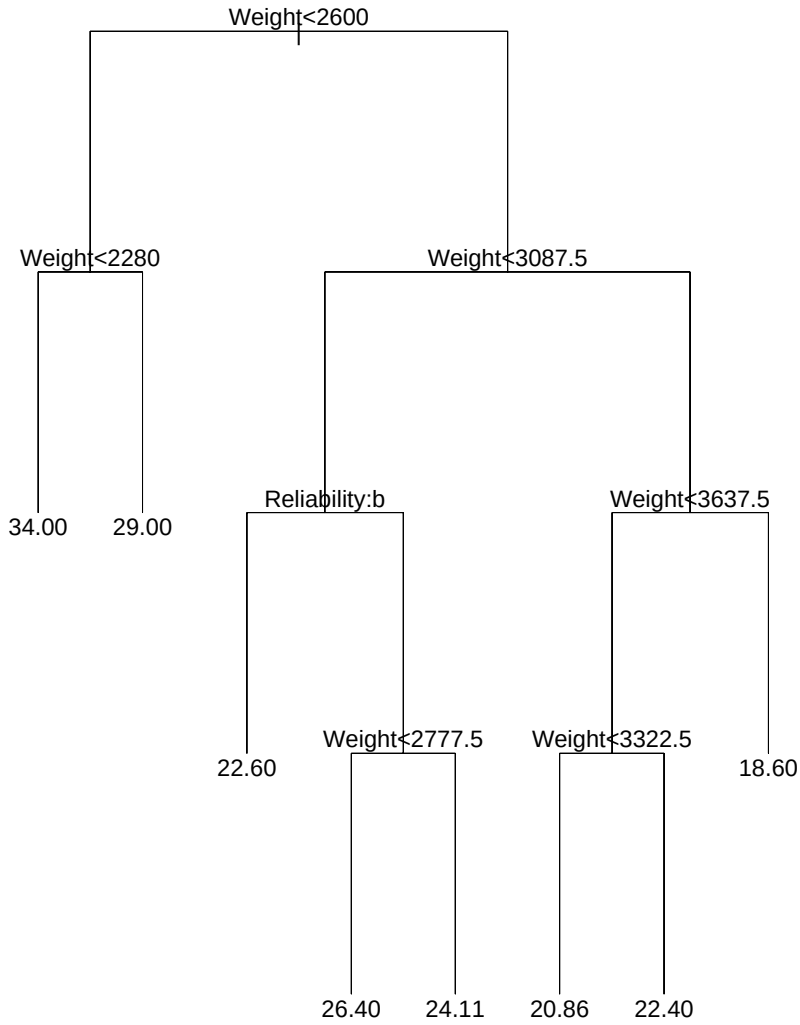

Figure 12.5: *Display of tree relating* `Mileage` *to* `Weight` *and* `Reliability`. *All of the data used to fit the data are complete.*

To create the tree shown in Figure 12.5, first create a new data set from `car.test.frame`, omitting those observations which are missing data for `Reliability`:

```
> car.test.no.miss <-
+ car.test.frame[!is.na(car.test.frame[,3]),]
```

Now grow the tree using the cleansed data:

```
> car.tree <- tree(Mileage ~ Weight + Reliability,
+ car.test.no.miss)
```

Next, we predict the data with values missing on `Reliability`, by extracting those observations that were omitted from `car.test.no.miss`, and then calling `predict` on the resulting data set:

```
> car.test.miss <-
+ car.test.frame[is.na(car.test.frame[,3]),]
> pred.miss <-
+ predict(car.tree,car.test.miss,type="tree")
> pred.miss

node), split, n, deviance, yval
      * denotes terminal node
1) root 11 245.300 24.80
  2) Weight<2600 3 65.940 30.92
    4) Weight<2280 1 0.000 34.00 *
    5) Weight>22figz80 2 26.000 29.00 *
  3) Weight>2600 8 81.060 22.58
    6) Weight<3087.5 3 11.770 24.32
     12) Reliability:2 0 0.000 22.60 *
     13) Reliability:1,3,4,5 0 0.000 24.93
       26) Weight<2777.5 0 0.000 26.40 *
       27) Weight>2777.5 0 0.000 24.11 *
    7) Weight>3087.5 5 10.680 20.65
     14) Weight<3637.5 4 17.000 21.50
       28) Weight<3322.5 3 8.918 20.86 *
       29) Weight>3322.5 1 5.760 22.40 *
     15) Weight>3637.5 1 0.160 18.60 *
```

Notice that there are no observations in the nodes (12, 13, 26, 27) at or below the split on `Reliability`.

PRUNING AND SHRINKING

Since tree size is not limited in the growing process, a tree may be more complex than necessary to describe the data. Two functions assess the degree a tree can be simplified without sacrificing goodness-of-fit. The `prune.tree` function achieves parsimonious description by reducing the nodes on a tree, whereas the `shrink.tree` function *shrinks* each node towards its parent.

Both functions take the following arguments:

- treeFitted model object of class tree.
- kcost complexity parameter (for `prune.tree`); shrinkage parameter (for `shrink.tree`).
- newdataa data frame containing the values at which predictions are required. if missing, the data used to grow the tree are used.

Pruning

Pruning successively snips off the *least important splits*. Importance of a subtree is measured by the cost-complexity measure:

$$D_k(T') = D(T') + k \cdot \text{size}(T')$$

where

$$\begin{aligned} D_k(T') &= \text{the deviance of the subtree } T', \\ \text{size}(T') &= \text{the number of terminal nodes of } T', \\ k &= \text{the cost-complexity parameter.} \end{aligned}$$

Cost-complexity pruning determines the subtree T that minimizes $D_k(T')$ over all subtrees. The larger the k, the fewer nodes there will be.

The `prune.tree` function takes a cost-complexity parameter argument `k`, which can be either a scalar or a vector. A scalar `k` defines one subtree of `tree` whereas a vector `k` defines a sequence of subtrees minimizing the cost-complexity measure. If the `k` argument is not supplied, a nested sequence of subtrees is created by recursively snipping off the least important splits.

Figure 12.6 shows the deviance decreasing as a function of the number of nodes and the cost-complexity parameter `k`.

```
> plot(prune.tree(kyph.tree))
```

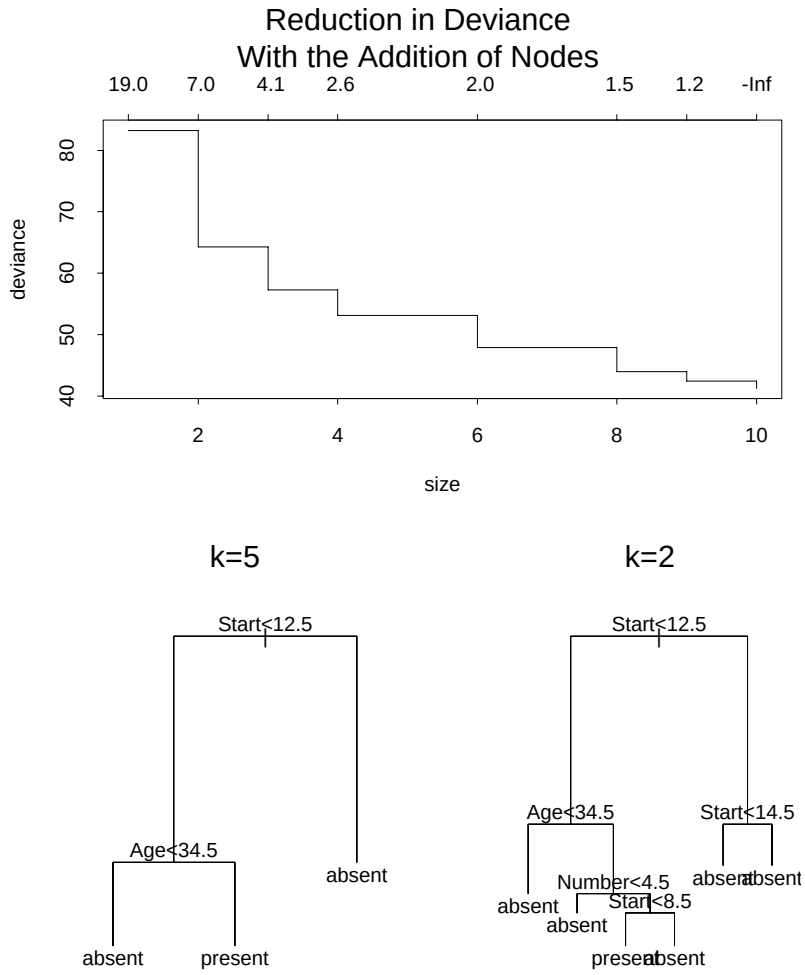

Figure 12.6: *A sequence of plots generated by the* `prune.tree` *function.*

Since over one half of the reduction in deviance is explained by the first three nodes, we limit the tree to three nodes.

```
> plot(prune.tree(kyph.tree, k = 5))
> text(prune.tree(kyph.tree, k = 5))
```

```
> summary(prune.tree(kyph.tree, k = 5))

Classification tree:
prune.tree(tree = kyph.tree, k = 5)
Variables actually used in tree construction:
[1] "Start" "Age"
Number of terminal nodes: 3
Residual mean deviance: 0.734 = 57.25 / 78
Misclassification error rate: 0.1728 = 14 / 81
```

By comparing this to the summary of the full tree in the section Displaying Trees, we see that reducing the number of nodes from 10 to 3 simplifies the model, but at the cost of increased misclassification.

Increasing the complexity of the tree to 6 nodes drops the misclassification to a rate comparable to that of the full tree with 10 nodes:

```
> summary(prune.tree(kyph.tree, k = 2))

Classification tree:
prune.tree(tree = kyph.tree, k = 2)
Number of terminal nodes: 6
Residual mean deviance: 0.6383 = 47.88 / 75
Misclassification error rate: 0.1358 = 11 / 81
```

Figure 12.6 shows `kyph.tree` pruned to 3 and 6 nodes.

Shrinking

Shrinking reduces the number of *effective* nodes by shrinking the fitted value of each node towards its parent node. Shrunken fitted values, for a shrinking parameter k, are computed according to the recursion:

$$\hat{y}(\text{node}) = k \cdot \bar{y}(\text{node}) + (1-k) \cdot \hat{y}(\text{parent})$$

where

$$\bar{y}(\text{node}) = \text{the usual fitted value for a node,}$$
$$\hat{y}(\text{parent}) = \text{the } \textit{shrunken} \text{ fitted value for the node's parent.}$$

The `shrink.tree` function *optimally* shrinks children nodes to their parent, based on the magnitude of the difference between $\bar{y}(\text{node})$ and $\bar{y}(\text{parent})$. The shrinkage parameter argument $(0 < k < 1)$ may be a scalar or a vector. A scalar `k` defines one shrunken version of

tree, whereas a vector k defines a sequence of shrunken trees obtained by optimal shrinking for each value of k. If the k argument is not supplied, a nested sequence of subtrees is created by recursively shrinking the tree for a default sequence of values (roughly .05 to .91) of k.

Figure 12.7 shows the deviance decreasing as a function of the number of *effective* nodes and the shrinkage parameter, k.

```
> plot(shrink.tree(kyph.tree))
```

Limit the tree to three *effective nodes* as done with pruning as follows:

```
> kyph.tree.sh.25 <- shrink.tree(kyph.tree, k = 0.25)
> plot(kyph.tree.sh.25)
> text(kyph.tree.sh.25)
> title("k = 0.25")
> summary(kyph.tree.sh.25)

Classification tree:
shrink.tree(tree = kyph.tree, k = 0.25)
Number of terminal nodes: 10
Effective number of terminal nodes: 2.8
Residual mean deviance: 0.7385 = 57.75 / 78.2
Misclassification error rate: 0.1358 = 11 / 81
```

The lower misclassification rate is maintained even with only three effective nodes.

Expand the tree to three *effective nodes* as follows:

```
> kyph.tree.sh.47 <- shrink.tree(kyph.tree, k = 0.47)
> plot(kyph.tree.sh.47)
> text(kyph.tree.sh.47)
> title("k = 0.47")
> summary(kyph.tree.sh.47)

Classification tree:
shrink.tree(tree = kyph.tree, k = 0.47)
Number of terminal nodes: 10
Effective number of terminal nodes: 6
Residual mean deviance: 0.6281 = 47.11 / 75
Misclassification error rate: 0.1358 = 11 / 81
```

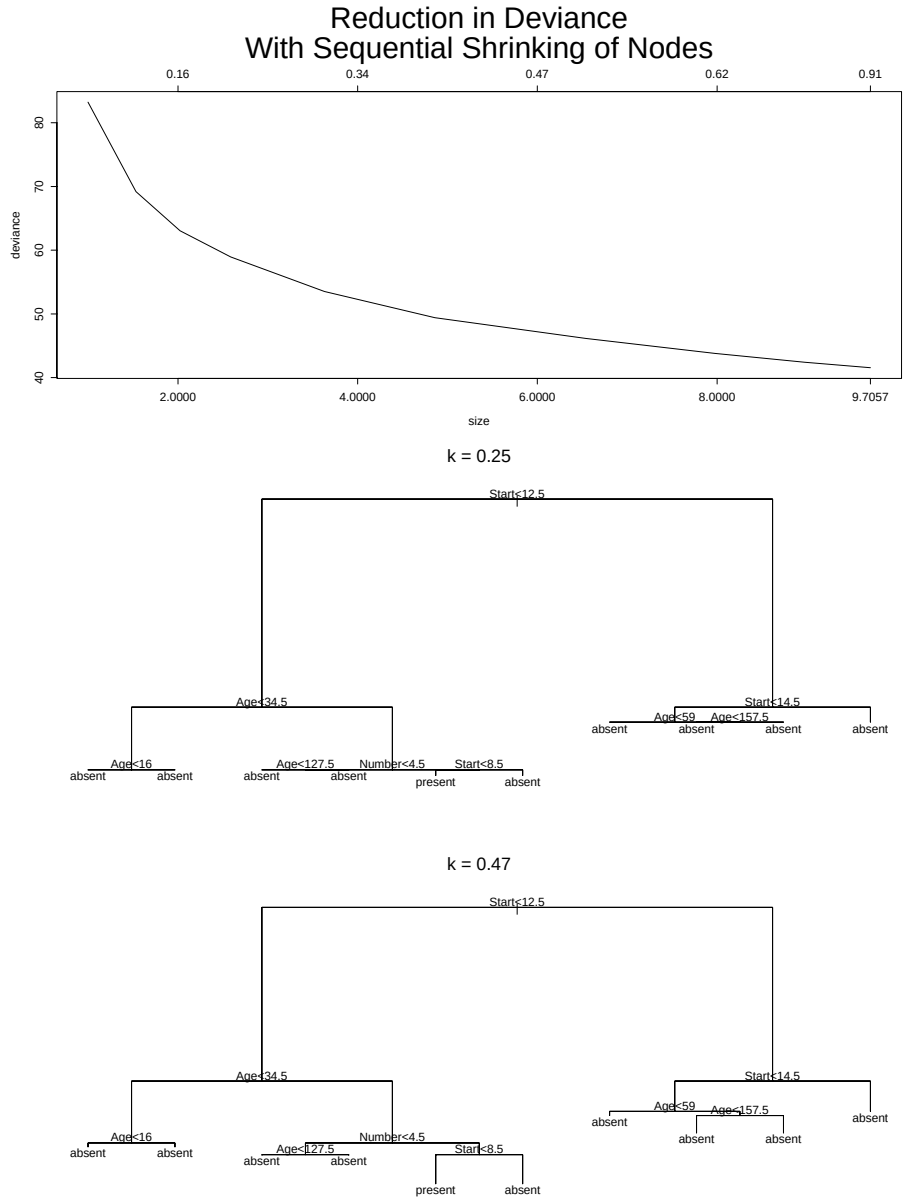

Figure 12.7: *A sequence of plots generated by the* `shrink.tree` *function.*

Note that no change other than a decrease in the residual mean deviance and an increase in the number of effective nodes.

GRAPHICALLY INTERACTING WITH TREES

A number of S-PLUS functions use the tree metaphor to diagnose tree-based model fits. The functions are naturally grouped by components of trees: *subtrees*, *nodes*, *splits*, and *leaves*. Except for the leaves functions, these functions allow you to interact graphically with trees, to perform a *what-if* analysis. You can also use these functions noninteractively by including a list of nodes as an argument. The goal is to better understand the fitted model, examine alternatives, and interpret the data in light of the model.

You can select subtrees from a large tree, and apply a common function (such as a plot) to the *stand* of resulting trees. Similarly, you can snip subtrees from the large tree, in order to gain resolution and label the top of the tree.

You can browse nodes to obtain important information too bulky to be usefully placed on a tree plot. You can obtain the names of observations which occur in a node. By examining the path (that is, the sequence of splits) that lead to a node, you can characterize the observations in that node.

You may compare *optimal* splits (generated by the tree-growing algorithm) to other potential splits. This helps to discover splits on variables that may shed light on the nature of the data. Any split divides the observations in a node into two groups. Therefore, you can compare the distribution of observations of a chosen variable in each of the two groups. This helps characterize the two groups, and also find variables with good discriminating abilities. You may regrow the tree, after designating a different split at a node.

The leaves of the trees represent the most homogeneous partitions of the data. You can investigate the differences across leaves by studying the distribution or summary statistics of chosen variables.

Subtrees

You can select or delete subtrees by *subscripting* the original tree, or by using one of the two functions described below.

The function `snip.tree` function deletes subtrees; that is, it snips branches off a specified tree. One goal may be to gain resolution at the top of the tree so that it can be labeled.

The graphical interface, using a mouse, proceeds as follows:

- first click informs you of the change in tree deviance if that branch is snipped off.
- second click removes the branch from the tree.

Figure 12.8 shows the result of snipping three branches off `kyph.tree`.

```
> par(mfrow=c(3,1))
> plot(kyph.tree)
> plot(kyph.tree)
> kyph.tree.sn <- snip.tree(kyph.tree)

node number: 4
   tree deviance = 41.24
   subtree deviance = 42.74
node number: 10
   tree deviance = 42.74
   subtree deviance = 43.94
node number: 6
   tree deviance = 43.94
   subtree deviance = 47.88

> plot(kyph.tree.sn)
> text(kyph.tree.sn,cex=1)
```

For noninteractive use, we can equivalently supply the node numbers in `snip.tree(kyph.tree,c(4,10,6))`. Negative subscripting is a convenient shorthand: `kyph.tree[-c(4,10,6)]`.

Similarly, the function `select.tree` function selects subtrees of a specified tree. For each node specified in the argument list or selected interactively, the function returns a tree object rooted at that node. These can in turn be plotted, etc.

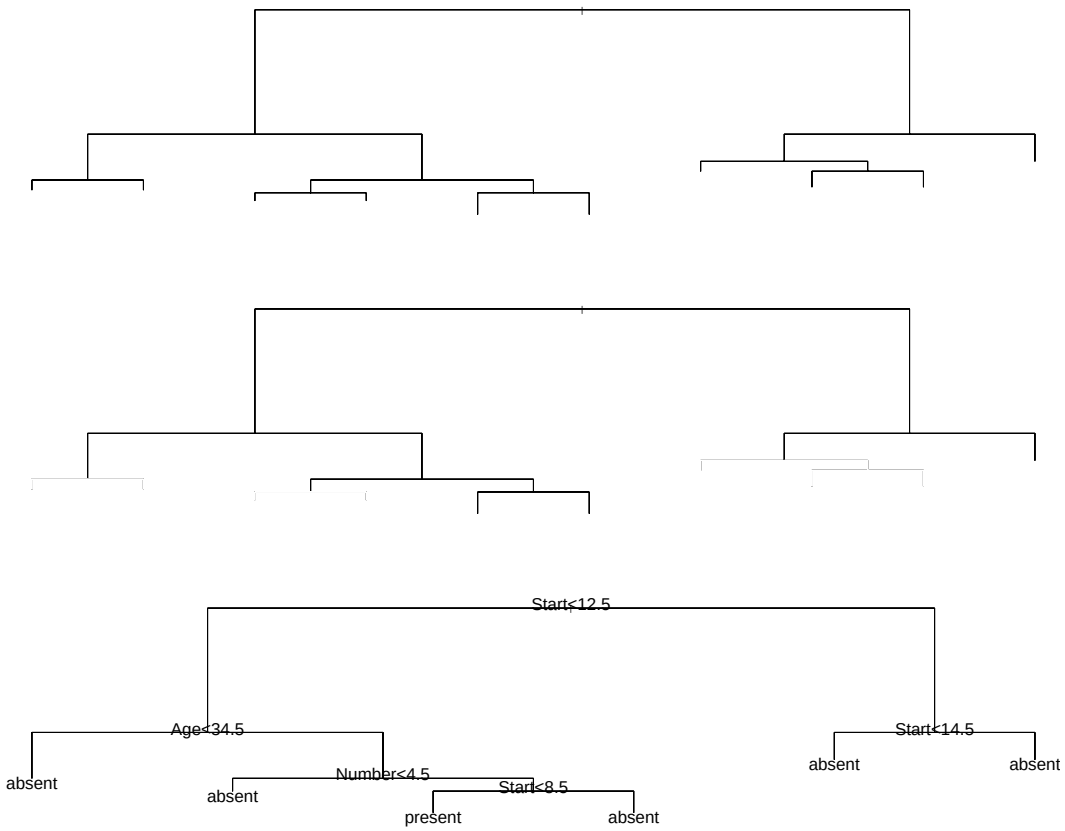

Figure 12.8: *A sequence of plots created by snipping branches from the top tree.*

Nodes

Several S-PLUS functions encourage the user to obtain more detailed information about nodes. Each of them take a tree object as a required argument, and an optional list of nodes. If the node list is omitted, graphical interaction is expected. The functions return a list, with one component for each node.

The `browser` function returns a summary of the information contained in a node. Interactively, you obtain information on the second and fifth nodes of `kyph.tree` by:

```
> browser(kyph.tree)
```

```
node number: 2
 split: Start<12.5
 n: 35
 dev: 47.800
 yval: absent
        absent present
[1,] 0.5714286 0.4285714
node number: 5
 split: Age>34.5
 n: 25
 dev: 34.300
 yval: present
     absent present
[1,]   0.44    0.56
```

Alternatively, provide a list of nodes as an argument:

```
> browser(kyph.tree,c(2,5))

     var  n      dev   yval splits.cutleft splits.cutright
2    Age 35 47.80357 absent          <34.5           >34.5
5 Number 25 34.29649 present           <4.5            >4.5
  yprob.absen yprob.present
2   0.5714286     0.4285714
5   0.4400000     0.5600000
```

The `identify` function is another generic function with a tree-specific method. The following noninteractive call lists the observations in the eighth and ninth nodes of `kyph.tree`:

```
> identify(kyph.tree,nodes=c(8,9))

$"8":
[1] "4" "14" "26" "29" "39"
$"9":
[1] "13" "21" "41" "68" "71"
```

The function `path.tree` returns the *path* (sequence of splits) from the root to any node of a tree. This is useful in those cases where overplotting results if the tree is labeled indiscriminately. As an example, we interactively look at the path to the rightmost terminal node of the kyphosis tree:

```
> path.tree(kyph.tree)
```

```
node number: 27
  root
  Start>12.5
  Start<14.5
  Age>59
  Age>157.5
```

By examining the path, we can determine that the children in this node are more than 157.5 months old, and the beginnings of the range of vertebrae involved are between 12.5 and 14.5.

Splits

The recursive partitioning algorithm underlying the `tree` function chooses the "best" set of splits that partition the predictor variable space into increasingly homogeneous regions. However, it is important to remember that this is just an algorithm. There may be other splits that also help you understand the data. The functions in this section help to examine alternative splits.

Using the `burl.tree` function, you can select a node either interactively or through the argument list, and observe the *goodness-of-split* for each predictor in the model formula. The goodness-of-split criterion is the difference in deviance between the node and its children (defined by the tentative split). Large differences correspond to important splits. Reduction in deviance is plotted against a quantity which depends upon the form of the predictor:

- numeric: each possible cut-point split.
- factor: a decimal equivalent of the binary representation of each possible subset split. The plotting character is a string labeling the left split.

In the following example and Figure 12.9, competing splits are plotted for each of the four predictor variables in the `cu.summary` data frame.

```
> reliab.tree <- tree(Reliability ~
+ Price + Country + Mileage + Type,
+ na.action = na.tree.replace.all, data = cu.summary)
> tree.screens() #establish plotting regions

[1] 1 2

> plot(reliab.tree,type="u")
> text(reliab.tree)
```

```
> burl.tree(reliab.tree) # Now click at the root node
```

The `burl.tree` function returns a list. For each variable there is a component which contains the necessary information for doing each of the plots.

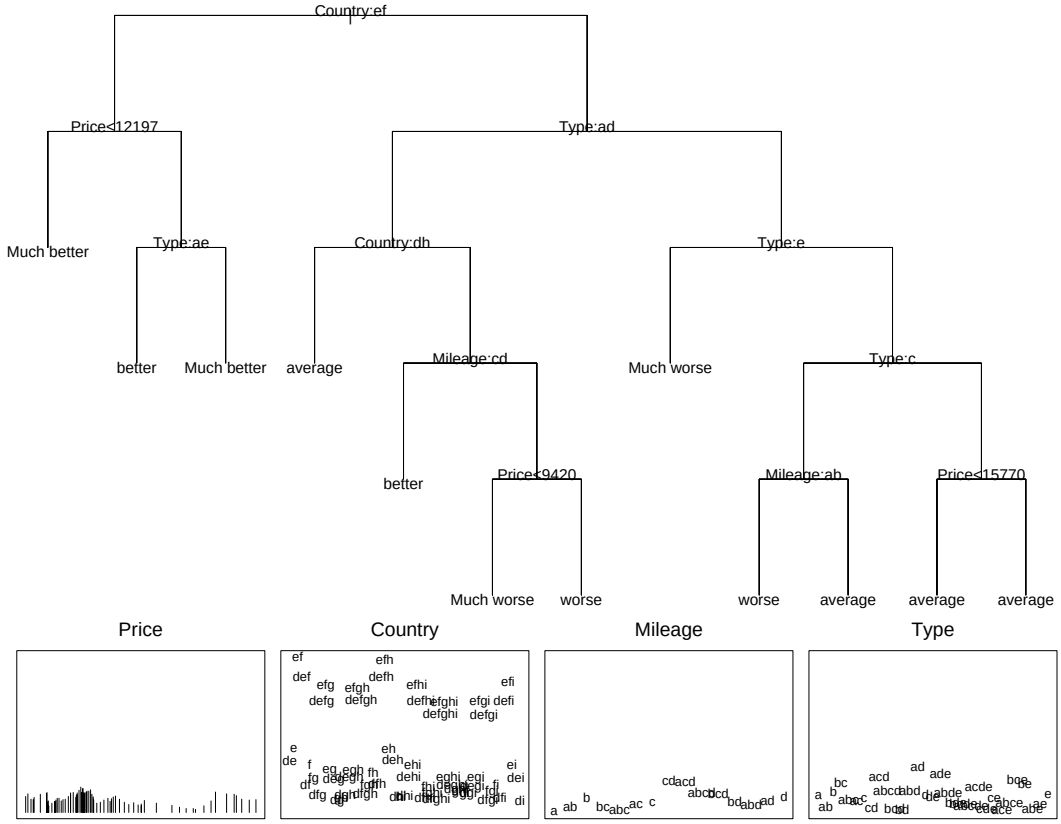

Figure 12.9: *A tree for* `Reliability` *in the* `cu.summary` *data frame with a burl plot of the four predictors for the root node.*

The burl plots show that the most important splits involve the variable `Country`. The candidate splits on this variable divide into two groups; the top group discriminates better than the bottom. The very best split is the one labeled `ef = Japan, Japan/USA`. Moreover, this occurs in all candidate splits in the top group. Therefore, we conclude that this is a meaningful split.

The function `hist.tree` requires a list of variable names, in addition to the tree object (and, optionally, a list of nodes). Unlike `burl.tree`, the variables need not be predictors. For a given node, a side-by-side histogram is plotted for each variable. The histogram on the left displays the distribution of the observations following the left split; similarly the histogram on the right displays the distribution of the observations following the right split.

Figure 12.10 is produced by the following expression:

```
> reliab.tree.2 <- tree(Reliability ~
+ Country + Mileage + Type,
+ na.action = na.tree.replace.all, data = cu.summary)
> tree.screens() #establish plotting regions

[1] 1 2

> plot(reliab.tree.2, type="u")
> text(reliab.tree.2)
> hist.tree(reliab.tree.2, Price, Mileage, nodes=1)
```

The figure shows that Japanese cars manufactured here or abroad tend to be less expensive and more fuel efficient than others. The lower portion of the plot displays a side-by-side histogram for each of the variables `Price` and `Mileage`. Note that it is possible to get a histogram of this variable even though the formula for this tree does not include `Price`.

Manual Splitting and Regrowing

After examining competitor splits at a node, you may wonder what the tree would look like if the node were split differently. You can achieve this by using the `edit.tree` function.

The arguments to `edit.tree` are:

- `object`: Fitted model object of class `"tree"`.
- `node`: Number of the node to edit.
- `var`: Character string naming variable to split on.
- `split`: Left split. Numeric for continuous variables; character string of levels that go left for a factor.
- `split`: Right split. Character string of levels that go right for a factor.

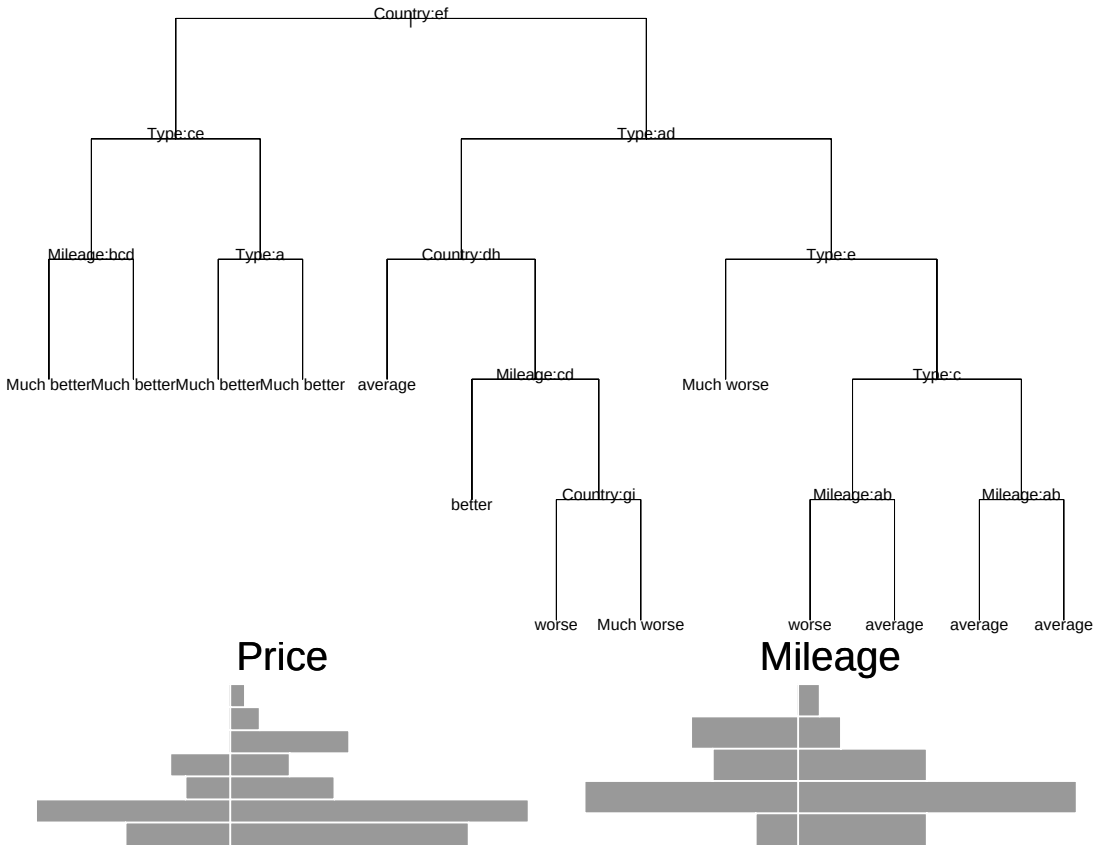

Figure 12.10: *A tree for* `Reliability` *in the* `cu.summary` *data frame.*

As an example, look at a burl of `kyph.tree` at the root node for the variable `Start`.

```
> kyph.burl <- burl.tree(kyph.tree, node = 1)
> kyph.burl$Start
```

```
   Start       dev numl
1    1.5  1.001008    5
2    2.5  1.887080    7
3    4.0  2.173771   10
4    5.5  5.098140   13
5    7.0 11.499747   17
6    8.5 17.946393   19
7    9.5 12.812267   23
8   10.5 12.821041   27
```

```
 9  11.5 10.136948   30
10  12.5 18.977175   35
11  13.5 13.927629   47
12  14.5 17.508746   52
13  15.5 12.378558   59
14  16.5  2.441679   76
```

Use `edit.tree` to regrow the tree with a designated split at `Start = 8.5`. The result is shown in Figure 12.11.

```
> kyph.tree.edited <- edit.tree(kyph.tree, node = 1,
+ var = "Start", splitl = 8.5)
```

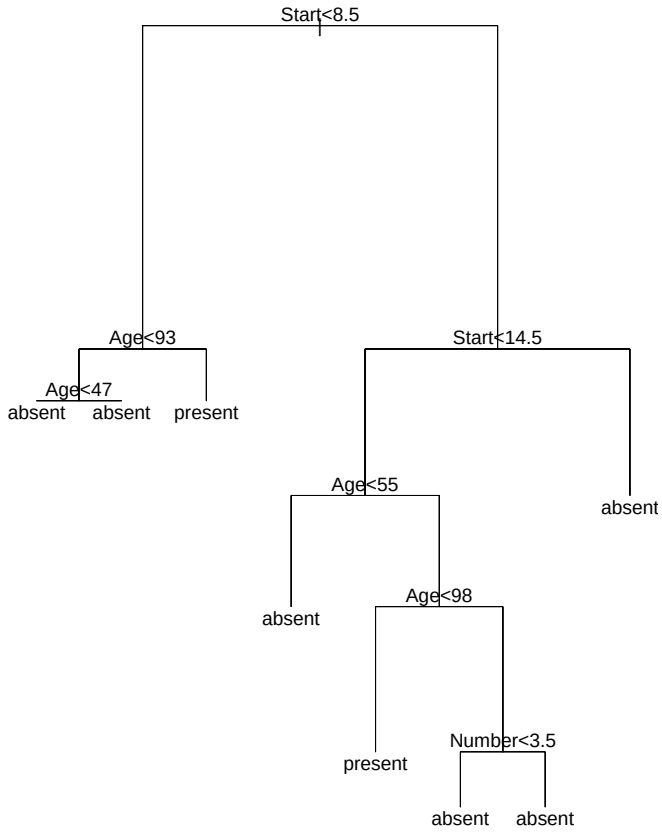

Figure 12.11: `kyph.tree` *regrown at the root node with a split at* `Start = 8.5`.

Leaves

Two noninteractive functions show the distribution of a variable over *all* terminal nodes of a tree.

The function `tile.tree` plots histograms of a specified variable for observations in each leaf. This function can be used, for example, to display class probabilities across the leaves of a tree. Figure 12.12 shows the distribution across leaves for `Kyphosis`.

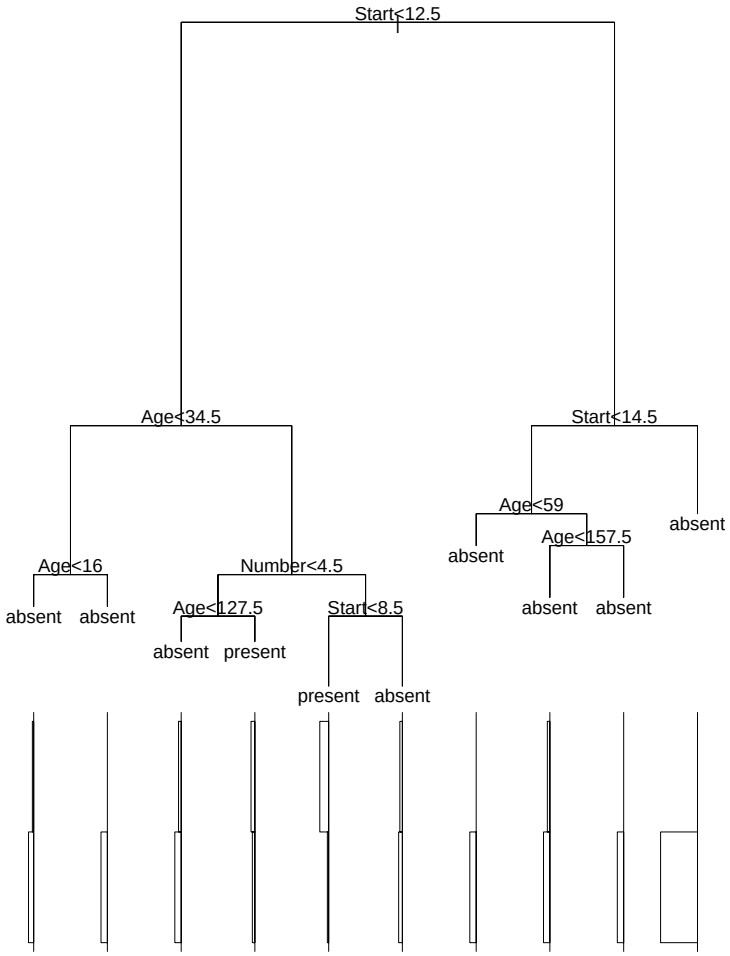

Figure 12.12: *A tree of the* `kyphosis` *data with a tile plot of* `Kyphosis`.

```
> tree.screens() #split plotting screen
> plot(kyph.tree,type="u")
> text(kyph.tree)
```

```
> tile.tree(kyph.tree, Kyphosis)
```

A related function, `rug.tree`, shows the average value of a variable over the leaves of a tree. The optional argument `FUN` allows you to summarize the variable with something other than the mean (for example, trimmed means, medians).

```
> rug.tree(kyph.tree, Start, FUN = median)
```

Figure 12.13 shows the rug plot of medians for `Start`.

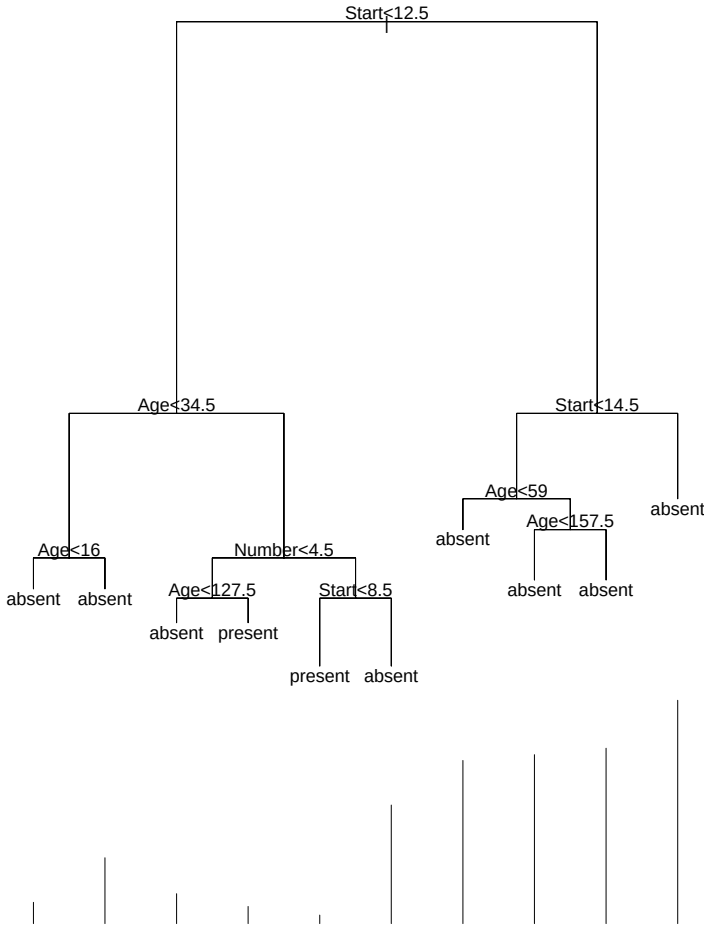

Figure 12.13: *A tree of the* `kyphosis` *data with a rug plot of* `Start`.

REFERENCES

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

Chambers, J.M. and Hastie, T.J. (1992). *Statistical Models in S.* Wadsworth and Brooks Cole Advanced Books and Software, Pacific Grove, CA.

LINEAR AND NONLINEAR MIXED-EFFECTS MODELS 13

INTRODUCTION

Mixed-effects models provide a powerful and flexible tool for analyzing grouped data, that is, data that can be classified according to one or more grouping variables. Mixed-effects models incorporate both fixed and random effects:

- Fixed effects are parameters associated with an entire population, or with repeatable levels of experimental factors.
- Random effects are instead associated with experimental units drawn at random from a population.

Such models typically describe relationships between a response variable and some covariates in data grouped according to one or more classification factors. Common applications are longitudinal data, repeated measures data, multilevel data, and block designs. Mixed-effects models flexibly represent the covariance structure induced by the grouping of the data by associating common random effects to observations sharing the same level of a classification factor.

This chapter describes a set of functions, classes, and methods for the analysis of linear and nonlinear mixed-effects models in S-PLUS. These provide a comprehensive set of tools for analyzing linear and nonlinear mixed-effects models with an arbitrary number of nested grouping levels. They supersede the modeling facilities available in release 3 of S (Chambers and Hastie, 1992) and releases 3.4 (Unix) and 4.5 (Windows) of S-PLUS.

This chapter will teach you:

- How to represent grouped data sets using the `groupedData` class.
- How to fit basic linear mixed-effects models using the `lme` function, and how to manipulate the returned `lme` objects.
- How to fit basic nonlinear mixed-effects models using the `nlme` function, and how to manipulate the returned `nlme` objects.
- How to fit advanced linear and nonlinear mixed-effects models by defining positive-definite matrices, correlation structures, and variance functions.

The analysis of several sample data sets will illustrate many of the available features. A detailed description of all functions, classes, and methods can be found in the online help files.

The code was contributed by Douglas M. Bates, of the University of Wisconsin, and José C. Pinheiro of Bell Laboratories. Their book, *Mixed Effects Models in S,* Springer-Verlag, 1999, contains a careful description of the statistical theory behind mixed-effects models, as well as detailed examples of the software for fitting and displaying them herein introduced.

REPRESENTING GROUPED DATA SETS

The datasets used for fitting mixed-effects models have several characteristics in common. They consist of measurements of a continuous response, at several levels of a covariate (for example, `time`, `dose`, or `treatment`), grouped according to one or more factors. Additional covariates may also be present, some of which may vary within a group (*inner* covariates) and some of which may not (*outer* covariates).

A natural way to represent such data in S-PLUS is as a `data.frame` containing the response, the primary covariate, the grouping factor(s), and any additional factors or continuous covariates. The different roles of the variables in the data frame can be described by a formula of the form

```
response ~ primary | grouping1/grouping2/...
```

which is similar to the display formula in a Trellis plot (Becker, Cleveland, and Shyu, 1996).

The groupedData Class

The formula and the data frame are packaged together in a `groupedData` class. The constructor (the function used to create objects of a given class) for `groupedData` takes a formula and data frame as arguments. The call to the constructor establishes the roles of the variables, converts the grouping factors to ordered factors so panels in plots are ordered in a natural way, and stores descriptive labels for data plots and plots of derived quantities.

By default, the grouping factors are converted to ordered factors with the order determined by a summary function applied to the response split according to the groups, taking into account the nesting order. (The default summary function is the maximum.) Additionally, labels can be given for the response and the primary covariate and their units can be specified as arbitrary strings. The reason for separating the labels and the units is to allow propagation of the units to derived quantities such as the residuals from a fitted model.

When outer factors are present, they are given by formula such as `outer = ~ Sex` or `outer = ~ Treatment * Type`. When multiple grouping factors are present, a list of such formulas must be supplied. Inner factors are described in a similar way. When establishing the

order of the levels of the grouping factor, and hence the order of panels in a plot, re-ordering is only permitted within combinations of levels for the outer factors.

Example: The Orthodont Data Set

As a first example of grouped data, consider the data from an orthodontic study presented in Porthoff and Roy (1964). These data, displayed in Figure 13.1, consist of four measurements of the distance (in millimeters) from the center of the pituitary to the pterygomaxillary fissure made at ages 8, 10, 12, and 14 years on 16 boys and 11 girls.

In the `Orthodont` data set, subjects are classified into two groups by `Sex`, an indicator variable assuming the value "`Male`" for boys and "`Female`" for girls. Each subject has four measures of distance, and the 108 total records are grouped into 27 groups by `Subject`. This is an example of balanced repeated measures data, with a single level of grouping (`Subject`).

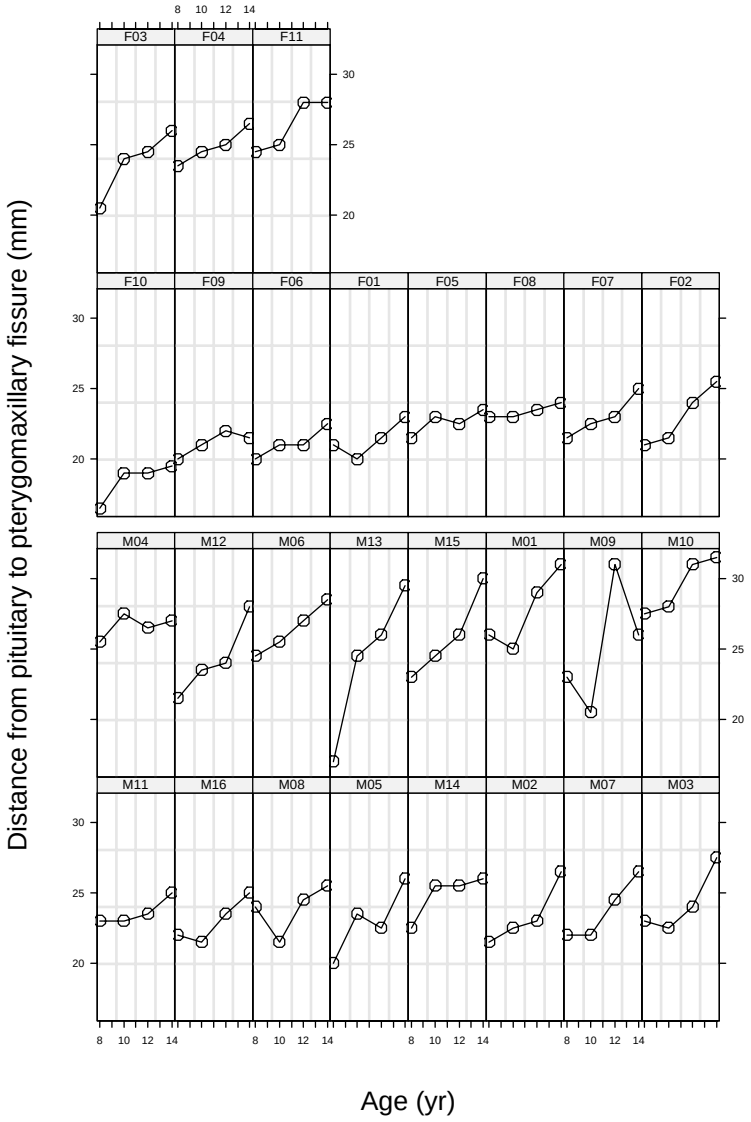

Figure 13.1: *Orthodontic growth patterns in 16 boys (M) and 11 girls (F) between 8 and 14 years of age. Panels within each gender group are ordered by maximum response.*

To create a new `groupedData` object, use the class constructor.

```
> Orthodont <- groupedData(distance ~ age | Subject,
+ data = Orthodont, outer = ~ Sex,
+ labels = list(x = "Age",
+ y="Distance from pituitary to pterygomaxillary fissure"),
+ units = list(x = "(yr)", y = "(mm)"))
```

The `print` method will print the display formula and the data frame associated with a `groupedData` object.

```
> print(Orthodont)

Grouped Data: distance ~ age | Subject
    distance age Subject    Sex
  1     26.0   8     M01   Male
  2     25.0  10     M01   Male
  3     29.0  12     M01   Male
  4     31.0  14     M01   Male
...
105     24.5   8     F11 Female
106     25.0  10     F11 Female
107     28.0  12     F11 Female
108     28.0  14     F11 Female
```

You can also use the `names` and `formula` methods to return the variable names and their roles for an existing `groupedData` object.

```
> names(Orthodont)

[1] "distance" "age" "Subject" "Sex"

> formula(Orthodont)

distance ~ age | Subject
```

An advantage of using a formula to describe the roles of the variables in the `groupedData` class is that this information can be used within the model-fitting functions to make the specification of the model easier. For example, getting preliminary simple linear regression fits by subject for this example is as simple as

```
> Ortho.lis <- lmList(Orthodont)
```

Plot the grouped data with

```
> plot(Orthodont, layout = c(8,4),          #Figure 13.1
+ between = list(y = c(0, 0.5, 0)))
```

When establishing the order of the levels of the grouping factor, and hence the order of panels in a plot, re-ordering is only permitted within combinations of levels for the outer factors. That is why the panels from boys and girls are grouped together in Figure 13.1.

The `plot` method for the `groupedData` class allows an optional argument `outer` which can be given a logical value or a formula. A logical value of `TRUE` (or `T`) indicates that the outer formula stored with the data should be used in the plot. The right hand side of the explicit or inferred formula replaces the grouping factor in the trellis formula. The grouping factor is then used to determine which points to join with lines. For example

```
> plot(Orthodont, outer = T)              #Figure 13.2
```

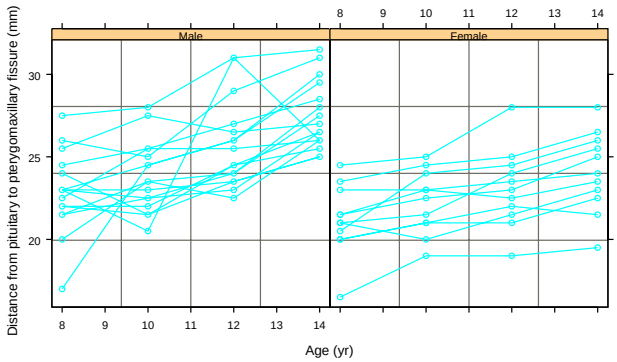

Figure 13.2: *Orthodontic growth patterns in 16 boys (M) and 11 girls (F) between 8 and 14 years of age, with different panels per gender.*

Example: The Pixel Data Set

An example of grouped data with two levels of grouping is given by a study in radiology consisting of repeated measures of mean pixel values from CT scans of the right and the left lymph nodes in the

axillary region of 10 dogs over a period of 14 days after application of a contrast. The purpose of the experiment was to model the mean pixel value as a function of time, in order to estimate the time when the maximum mean pixel value was attained. The data are shown in Figure 13.3.

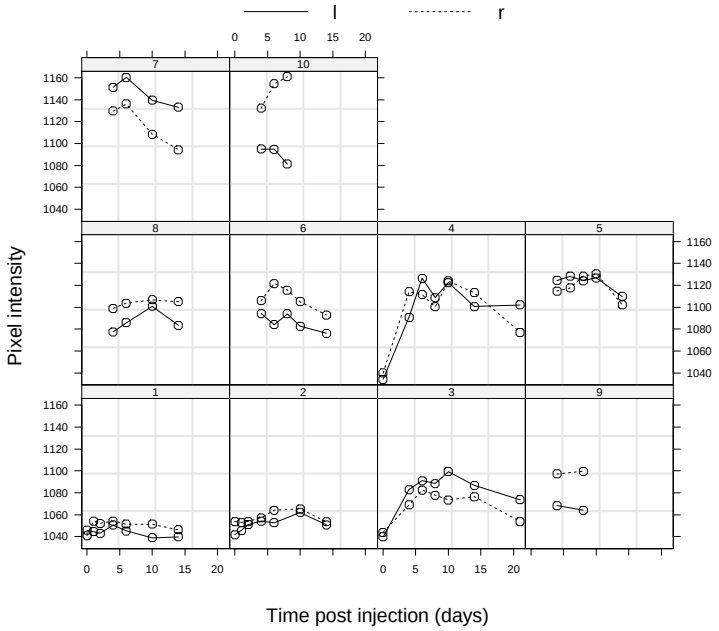

Figure 13.3: *Mean pixel intensity of the right (r) and left (l) lymph nodes in the axillary region versus time from intravenous application of a contrast. The pixel intensities were obtained from CT scans.*

Create a new `groupedData` object using the class constructor like this

```
> Pixel <- groupedData(pixel ~ day | Dog/Side,
+ data = Pixel, labels =
+ list(x="Time post injection",y="Pixel intensity"),
+ units = list(x = "(days)"))
```

Plot the grouped data with

```
> plot(Pixel, display = 1, inner = ~Side)    #Figure 13.3
```

An inner factor is used to determine which points within a panel are joined by lines, as in the plot for the `Pixel` data set above. When multiple levels of grouping are present, the `plot` method allows two

optional arguments: `displayLevel` and `collapseLevel`, specifying the grouping level to be used to determine the panels of the Trellis plot and the grouping level over which to collapse the data.

Example: The CO2 Data Set

As an example of grouped data with a nonlinear response, consider an experiment on the cold tolerance of a C_4 grass species, *Echinochloa crus-galli*, described in Potvin, Lechowicz, and Tardif (1990). A total of twelve four-week-old plants, six from Quebec and six from Mississippi, were divided into two groups: control plants that were kept at 26° C and chilled plants that were subject to 14 h of chilling at 7° C. After 10 h of recovery at 20° C, CO_2 uptake rates (in $\mu mol/m^2 s$) were measured for each plant at seven concentrations of ambient CO_2 (100, 175, 250, 350, 500, 675, 1000 μL/L). Each plant was subjected to the seven concentrations of CO_2 in increasing, consecutive order. The objective of the experiment was to evaluate the effect of plant type and chilling treatment on the CO_2 uptake. The data are shown in Figure 13.4.

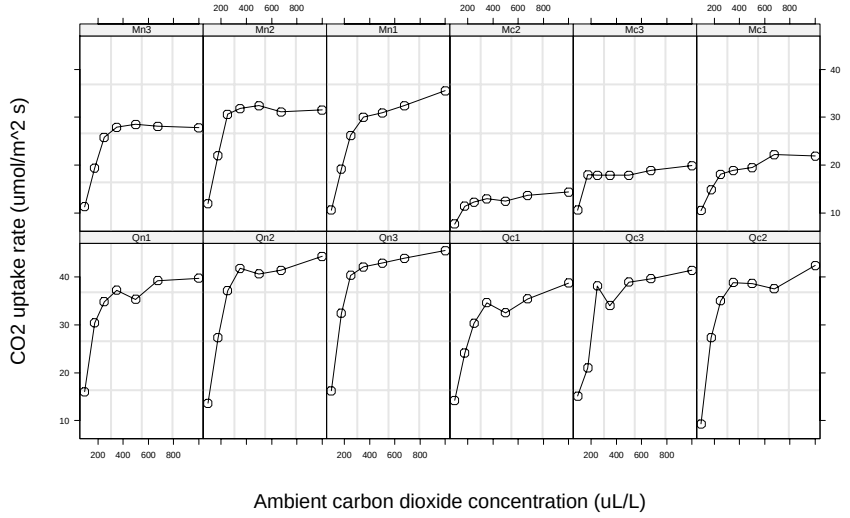

Figure 13.4: *CO_2 uptake versus ambient CO_2 concentration by treatment and type for* Echinochloa crus-galli *plants, six from Quebec and six from Mississippi. Half the plants of each type were chilled overnight before the measurements were taken.*

Create a new `groupedData` object using the class constructor like this

```
> CO2 <- groupedData(uptake ~ conc | Plant, data = CO2,
+ outer = ~ Treatment * Type,
+ labels = list(x = "Ambient carbon dioxide concentration",
+ y = "CO2 uptake rate"),
+ units = list(x = "(uL/L)", y = "(umol/m^2 s)"))
```

Plot the grouped data with

```
> plot(CO2)                                #Figure 13.4
```

As in the `Orthodont` example, you can use the optional argument `outer=T` to indicate that the outer formula stored with the data should be used in the plot. For example

```
> plot(CO2, outer = T)                     #Figure 13.5
```

Trellis parameters can be used to control the graphical presentation of grouped data. See the online help files for the family of functions related to `plot.xxGroupedData` for details.

Extractor functions can be used on `groupedData` objects to obtain the different components of the display formula. Functions such as `getGroups`, `getCovariate`, and `getResponse` can be applied to extract the corresponding model element.

In addition to the usual summarizing functions in S-PLUS, `groupedData` objects can be summarized by *group* using the function `gsummary`, as follows:

```
> gsummary(CO2)

    Plant        Type  Treatment conc   uptake
Qn1   Qn1      Quebec nonchilled  435 33.22857
Qn2   Qn2      Quebec nonchilled  435 35.15714
Qn3   Qn3      Quebec nonchilled  435 37.61429
Qc1   Qc1      Quebec    chilled  435 29.97143
Qc3   Qc3      Quebec    chilled  435 32.58571
Qc2   Qc2      Quebec    chilled  435 32.70000
Mn3   Mn3 Mississippi nonchilled  435 24.11429
Mn2   Mn2 Mississippi nonchilled  435 27.34286
Mn1   Mn1 Mississippi nonchilled  435 26.40000
Mc2   Mc2 Mississippi    chilled  435 12.14286
Mc3   Mc3 Mississippi    chilled  435 17.30000
Mc1   Mc1 Mississippi    chilled  435 18.00000
```

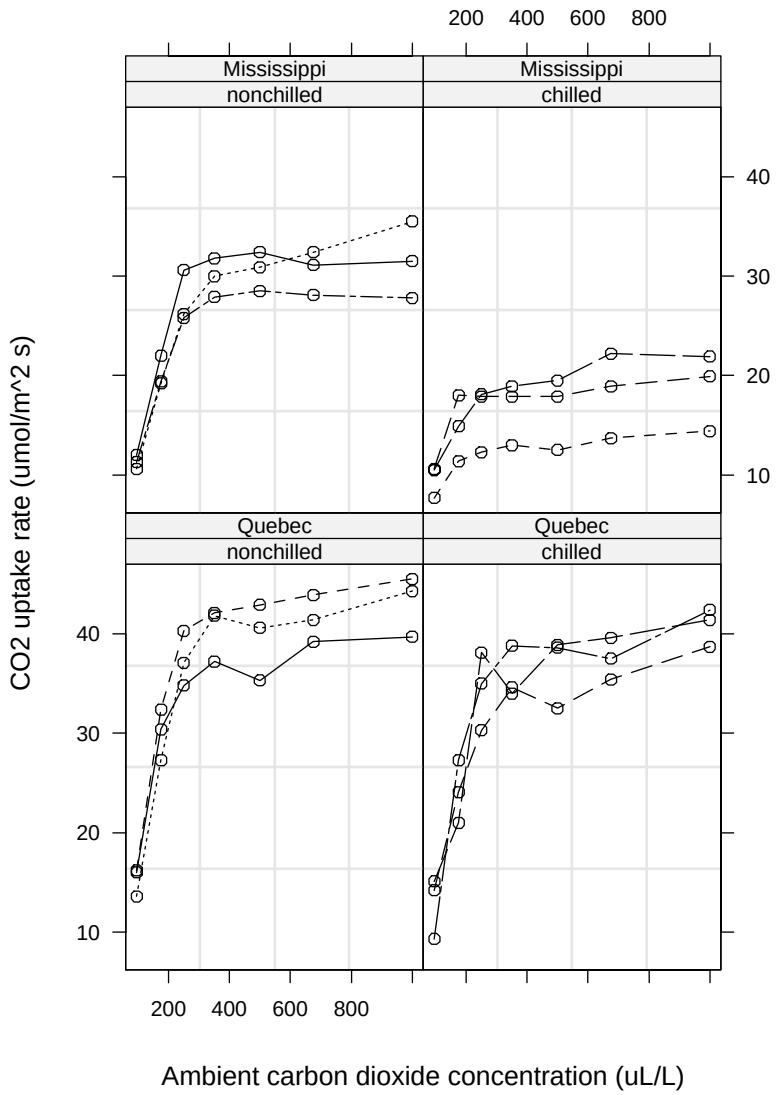

Figure 13.5: *CO_2 uptake versus ambient CO_2 by treatment and type for* Echinochloa crus-galli *plants, six from Quebec and six from Mississippi. Half the plants of each type were chilled overnight before the measurements were taken.*

Example: The Soybean Data Set

The `Soybean` data come from an experiment to compare growth patterns of two genotypes of soybean as described in Davidian and Giltinan (1995). One genotype is a commercial variety, Forrest (F), and the other is an experimental strain, Plant Introduction #416937

(P). The data were collected in the three years from 1988 to 1990. At the beginning of the growing season in each year, 16 plots were planted with seeds; 8 plots with each genotype. Each plot was sampled eight to ten times at approximately weekly intervals. At each sampling time, six plants were randomly selected from each plot, leaves from these plants were weighed, and the average leaf weight per plant was calculated for the plot. Different plots in different sites were used in different years. The data are stored in the data frame `Soybean` shown below.

```
> Soybean

    Plot Variety Year Time    weight
1      1       F 1988   14  0.10600
2      1       F 1988   21  0.26100
3      1       F 1988   28  0.66600
. . .
410   48       P 1990   51  6.131667
411   48       P 1990   64 16.411667
412   48       P 1990   79 16.946667
```

Create a new `groupedData` object using the class constructor like this

```
> Soybean <- groupedData(weight ~ Time | Plot,
+ data = Soybean,
+ outer = ~ Variety * Year
+ labels = list(x = "Time since planting",
+ y = "Leaf weight/plant"),
+ units = list(x = "(days)", y = "(g)"))
```

Plot the grouped data with

```
> plot(Soybean, outer= ~ Year * Variety)     #Figure 13.6
```

The objective is to model the growth pattern in terms of average leaf weight. Davidian and Giltinan (1995) suggest a logistic function as appropriate. Later in the chapter, you will learn to use the S-PLUS function `nlsList` to create a list of logistic fits, one for each group, to test this hypothesis.

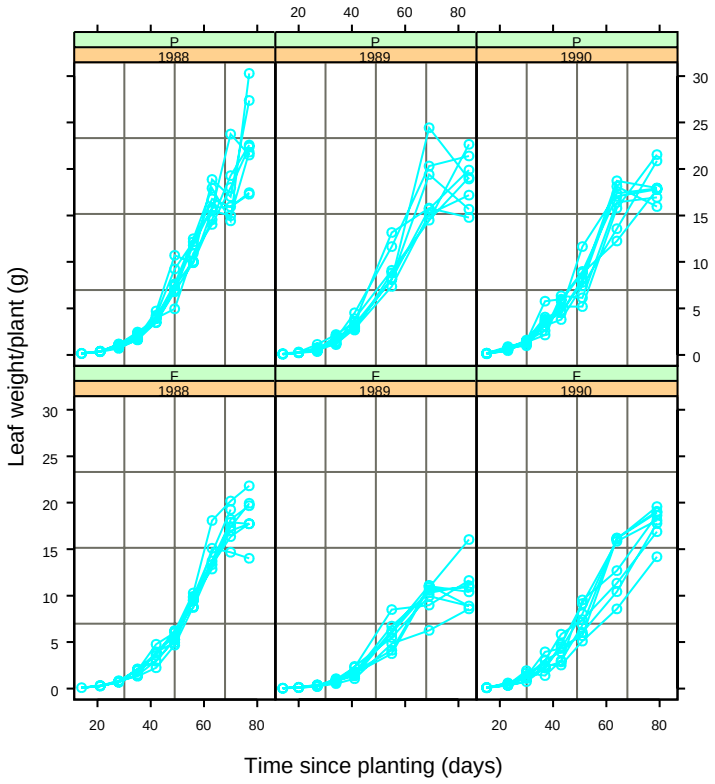

Figure 13.6: *Average leaf weight per plant versus time since planting for plots of soybeans. The plots are from two different years and represent two different genotypes of soybeans.*

FITTING MODELS USING THE LME FUNCTION

The S-PLUS function `lme` fits a linear mixed-effects model (as described in Laird and Ware, 1982), or a multilevel linear mixed-effects model (as described, for example, in Longford, 1993, or Goldstein, 1995), using either maximum likelihood or restricted maximum likelihood. It produces an object of class `lme`.

Model Definitions

The plot of the individual growth curves in Figure 13.1 suggests that a linear model might adequately explain the orthodontic distance as a function of age, but the intercept and the slope seem to vary with the individual. The corresponding linear mixed-effects model is

$$d_{ij} = (\beta_0 + b_{i0}) + (\beta_1 + b_{i1})\text{age}_j + \varepsilon_{ij} \tag{13.1}$$

where d_{ij} represents the distance for the ith individual at age j, β_0 and β_1 are the population average intercept and the population average slope respectively, b_{i0} and b_{i1} are the effects in intercept and slope associated with the ith individual, and ε_{ij} is the within-subject error term. It is assumed that the $\boldsymbol{b}_i = (b_{i0}, b_{i1})^T$ are independent and identically distributed with a $\boldsymbol{N}(0, \sigma^2\boldsymbol{D})$ distribution and the ε_{ij} are independent and identically distributed with a $\boldsymbol{N}(0, \sigma^2)$ distribution, independent of the $\boldsymbol{b}_i$.

One of the questions of interest for these data is whether the curves show significant differences between boys and girls. Model (13.1) can be modified as

$$\begin{aligned} d_{ij} = {} & (\beta_{00} + \beta_{01}\text{sex}_i + b_{i0}) + \\ & (\beta_{10} + \beta_{11}\text{sex}_i + b_{i1})\text{age}_j + \varepsilon_{ij} \end{aligned} \tag{13.2}$$

to test for sex related differences in intercept and slope. In model (13.2), sex_i is an indicator variable assuming the value zero if the ith individual is a boy and one if she is a girl. β_{00} and β_{10} represent the population average intercept and slope for the boys and β_{01} and β_{11} are the changes in population average intercept and slope for girls. Differences between boys and girls can be evaluated by testing whether β_{01} and β_{11} are significantly different from zero. The remaining terms in (13.2) are defined as in (13.1).

In the `Pixel` example, a second order polynomial seems adequate to explain the evolution of pixel intensity with time since the contrast was injected. Preliminary analyses indicated that the intercept varies with dog, as well as with side within dog, and the linear term varies with dog, but not with side.

The corresponding multilevel linear mixed-effects model is

$$y_{ijk} = (\beta_0 + b_{0i} + b_{0i,\,j}) + (\beta_1 + b_{1i})t_{ijk} + \beta_2 t_{ijk}^2 + \varepsilon_{ijk} \tag{13.3}$$

where i refers to the dog number (1 through 10), j to the lymph node side (1 -- right, 2 -- left), and k refers to time; β_0, β_1, and β_2 denote respectively the intercept, the linear term, and the quadratic term fixed effects; b_{0i} denotes the intercept random effect at the dog level, $b_{0i,j}$ denotes the intercept random effect at the side within dog level, and b_{1i} denotes the linear term random effect at the dog level; y denotes the pixel intensity, t denotes the time since contrast injection, and ε_{ijk} denotes the error term. It is assumed that the $\boldsymbol{b}_i = (b_{0i}, b_{1i})^T$ are independent and identically distributed with common distribution $\boldsymbol{N}(0, \sigma^2 \boldsymbol{D}_1)$, the $\boldsymbol{b}_{i,j} = [b_{0i,j}]$ are independent and identically distributed with common distribution $\boldsymbol{N}(0, \sigma^2 \boldsymbol{D}_2)$ and independent of the $\boldsymbol{b}_i$ and the ε_{ijk} are independent and identically distributed with common distribution $\boldsymbol{N}(0, \sigma^2)$ and independent of the $\boldsymbol{b}_i$ and the $\boldsymbol{b}_{i,j}$.

Arguments

The typical call to the `lme` function is of the form

```
lme(fixed, data, random)
```

Only the first argument is required. The arguments `fixed` and `random` are generally given as formulas. Any linear model formula is allowed, giving the model formulation considerable flexibility. For model (13.1) and the `Orthodont` data, these formulas would be written as

```
fixed = distance ~ age, random = ~ age
```

For model (13.2), they would be written as

```
fixed = distance ~ age * Sex, random = ~ age
```

Note that the response variable is given only in the formula for the `fixed` argument and that `random` is usually given as a one-sided linear formula. If the `random` argument is omitted, it is assumed to be the same as the right hand side of the formula given in `fixed`.

Because `Orthodont` is a `groupedData` object, no grouping structure is explicitly given in `random`, as it is extracted from the `groupedData` display formula. Alternatively, the grouping structure can be included in the formula as conditioning expression.

```
random = ~ age | Subject
```

When multiple levels of grouping are present, as in the `Pixel` example, `random` must be given as a list of formulas, as below.

```
fixed = pixel ~ day + day^2
random = list(Dog = ~ day, Side = ~ 1)
```

Note that the names of the elements in the `random` list correspond to the names of the grouping factors and are assumed to be in outermost to innermost order. A model with a single intercept is represented by `~ 1`.

The optional argument `data` specifies the data frame in which the variables used in the model are available.

Examples

A simple call to `lme` to fit model (13.1) is

```
> Ortho.fit1 <- lme(fixed = distance ~ age,
+ data = Orthodont, random = ~ age | Subject)
```

To fit model (13.2), use

```
> ## set contrasts for desired parameterization
> options(contrasts = c("contr.treatment", "contr.poly"))
> Ortho.fit2 <- update(Ortho.fit1,
+ fixed = distance ~ age*Sex)
```

The multilevel model (13.3) is fitted by:

```
> Pixel.fit1 <- lme(fixed = pixel ~ day + day^2,
+ data=Pixel, random = list(Dog = ~ day, Side = ~1))
```

Other arguments of the `lme` function allow for flexible definitions of the within-group correlation and heteroscedasticity structures, subset of the data to be modeled, method to use when fitting the model, and a list of control values for the estimation algorithm. See the `lme` online help file for specific details on each argument.

MANIPULATING LME OBJECTS

A call to the `lme` function returns an object of class `lme`. The online help file for `lmeObject` contains a description of the returned object and each of its components. There are several methods available for `lme` objects, including `print`, `summary`, `anova`, and `plot`. These are described in the following sections.

The print Method

A brief description of the estimation results is returned by the `print` method. It gives estimates of the standard errors and correlations of the random effects, the within-group variance, and the fixed effects. For the `Ortho.fit1` object, the results are

```
> print(Ortho.fit1)

Linear mixed-effects model fit by REML
  Data: Orthodont
  Log-restricted-likelihood: -221.3183
  Fixed: distance ~ age
 (Intercept)       age
    16.76111 0.6601852

Random effects:
 Formula:  ~ age | Subject
 Structure: General positive-definite
              StdDev   Corr
(Intercept) 2.327037 (Inter
        age 0.226427 -0.609
   Residual 1.310040

Number of Observations: 108
Number of Groups: 27
```

The summary Method

A more complete description of the estimation results is returned by the `summary` function.

```
> summary(Ortho.fit2)

Linear mixed-effects model fit by REML
 Data: Orthodont
       AIC      BIC    logLik
  448.5817 469.7368 -216.2908

Random effects:
 Formula:  ~ age | Subject
 Structure: General positive-definite
              StdDev   Corr
(Intercept) 2.4055382 (Inter
        age 0.1803496 -0.668
   Residual 1.3100369

Fixed effects: distance ~ age + Sex + age:Sex
               Value Std.Error DF   t-value p-value
(Intercept)  16.34063  1.018536 79  16.04325  <.0001
        age   0.78438  0.086000 79   9.12065  <.0001
        Sex   1.03210  1.595739 25   0.64679  0.5237
    age:Sex  -0.30483  0.134736 79  -2.26242  0.0264
 Correlation:
        (Intr)    age    Sex
    age -0.880
    Sex -0.638  0.562
age:Sex  0.562 -0.638 -0.880

Standardized Within-Group Residuals:
       Min         Q1        Med        Q3     Max
 -3.168056 -0.3859336 0.00710403 0.4451484 3.84947

Number of Observations: 108
Number of Groups: 27
```

The approximate standard errors for the fixed effects are derived using an algorithm based on the asymptotic theory described in Pinheiro (1994). The results above indicate that the measurement

increases faster in boys than in girls (significant, negative `age:Sex` fixed effect), but the average intercept is common to boys and girls (non-significant `Sex` fixed effect).

To summarize the estimation results for model (13.3) use

```
> summary(Pixel.fit1)

Linear mixed-effects model fit by REML
 Data: Pixel
       AIC      BIC    logLik
  841.2102 861.9712 -412.6051

Random effects:
 Formula:  ~ day | Dog
 Structure: General positive-definite
              StdDev   Corr
(Intercept) 28.36994 (Inter
        day  1.84375 -0.555

 Formula:  ~ 1 | Side %in% Dog
        (Intercept) Residual
StdDev:    16.82424 8.989609

Fixed effects: pixel ~ day + day^2
               Value Std.Error DF   t-value p-value
(Intercept)  1073.339  10.17169 80  105.5222  <.0001
        day     6.130   0.87932 80    6.9708  <.0001
   I(day^2)    -0.367   0.03395 80  -10.8218  <.0001
 Correlation:
         (Intr)    day
     day -0.517
I(day^2)  0.186 -0.668

Standardized Within-Group Residuals:
       Min         Q1        Med       Q3      Max
 -2.829056 -0.4491807 0.02554919 0.557216 2.751964

Number of Observations: 102
Number of Groups:
 Dog Side %in% Dog
  10            20
```

The anova Method

A likelihood ratio test can be used to test the difference between the fixed effects models represented by `Ortho.fit1` and `Ortho.fit2`. The `anova` method provides that capability.

Warning:

Likelihood comparisons between restricted maximum likelihood (REML) fits with different fixed effects structures are not meaningful. Re-fit the objects using maximum likelihood (ML), before calling `anova`.

Use the `update` function to re-fit the two objects using maximum likelihood (ML)

```
> Ortho.fit1.ML <- update(Ortho.fit1, method = "ML")
> Ortho.fit2.ML <- update(Ortho.fit2, method = "ML")
```

then call `anova`

```
> anova(Ortho.fit1.ML, Ortho.fit2.ML)

              Model df      AIC      BIC    logLik
Ortho.fit1.ML     1  6 451.2116 467.3044 -219.6058
Ortho.fit2.ML     2  8 443.8060 465.2630 -213.9030
                Test  L.Ratio p-value
Ortho.fit1.ML
Ortho.fit2.ML 1 vs 2 11.40565  0.0033
```

The likelihood ratio test strongly rejects the null hypothesis of no sex differences. For small sample sizes, likelihood ratio tests tend to be too liberal when comparing models with nested fixed effects structures and should be used with caution. We recommend using the Wald-type tests provided by the `anova` method with a single argument, as these tend to have significance levels close to nominal, even for small samples.

The plot method

Diagnostic plots for assessing the quality of the fitted model are obtained using the `plot` method for class `lme`. This method takes several optional arguments, but a typical call is of the form.

```
plot(object, formula)
```

where the first argument is the `lme` object and the second is a display formula for the Trellis plot to be produced. The fitted object can be referenced by the symbol "`.`" in the formula argument. For example, to produce a plot of the standardized residuals versus fitted values by gender for the `Ortho.fit2` object included in Figure 13.7, use

```
> plot(Ortho.fit2,                                  # Figure 13.7
+ resid(., type = "p") ~ fitted(.) | Sex)
```

The expression above introduces two other common methods: `resid` and `fitted`. The argument `type=` for the method `residuals.lme` accepts the strings "`pearson`" (or "`p`") for standardized residuals, "`normalized`", and "`response`". By default the raw or response residuals (observed - fitted) are calculated, so we must set `type="p"` in the call above.

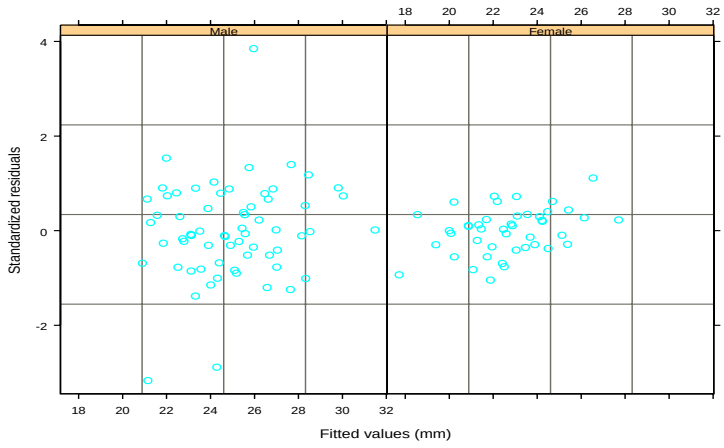

Figure 13.7: *Standardized residuals versus fitted values by gender, for the* `lme` *fit of model (13.2).*

There is evidence that the variability of the orthodontic distance is greater in boys than in girls and that some possible outliers are present in the data. To assess the predictive power of the fitted model, consider the plot of the observed versus fitted values by individual, presented in Figure 13.8 and obtained with.

```
> plot(Ortho.fit2,                                #Figure 13.8
+ distance ~ fitted(.) | Subject, layout = c(4, 7),
+ between = list(y = c(0, 0, 0, 0.5)), aspect = 1.0,
+ abline = c(0,1))
```

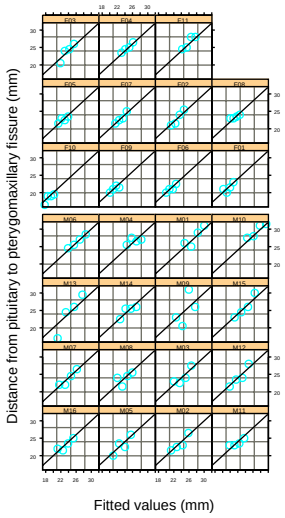

Figure 13.8: *Observed distances versus fitted values by subject, for the* `lme` *fit of model (13.2).*

For most of the subjects, there is very good agreement between the observed and fitted values, indicating that the fit is adequate.

The `formula` argument to the `plot` method gives virtually unlimited flexibility for generating customized diagnostic plots.

As a final example, consider the plot of the standardized residuals (at the side within dog level) for the `Pixel.fit1` object by dog.

```
> plot(Pixel.fit1, Dog~resid(., type="p"))  #Figure 13.9
```

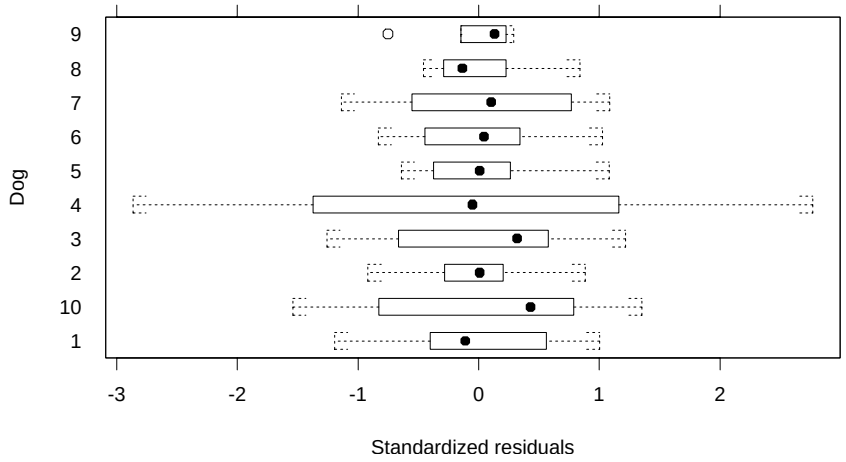

Figure 13.9: *Standardized residuals by dog, for the* `lme` *fit of model (13.3).*

The residuals seem symmetrically scattered around zero, with similar variabilities (except, possibly, for dog number 4).

Other Methods

Standard S-PLUS methods for extracting components of fitted objects, such as `residuals`, `fitted`, and `coefficients`, can be also be used on `lme` objects. In addition, `lme` includes the methods `fixed.effects` and `random.effects` for extracting the fixed effects and the random effects estimates. (Short names for the last two functions are `fixef` and `ranef`, respectively.)

```
> coef(Ortho.fit2)

    (Intercept)       age       Sex    age:Sex
M16    15.55739 0.6957259 1.032102 -0.3048295
M05    14.69527 0.7759020 1.032102 -0.3048295
...
F04    18.00174 0.8125874 1.032102 -0.3048295
F11    18.53691 0.8858564 1.032102 -0.3048295
```

```
> fixef(Pixel.fit1)

 (Intercept)      day   I(day^2)
    1073.339 6.129597 -0.3673503

> ranef(Pixel.fit1, level=1)  #random effects at Dog level

   (Intercept)         day
 1  -24.714229 -1.19537074
10   19.365854 -0.09936872
 2  -23.582059 -0.43243128
 3  -27.080310  2.19475596
 4  -16.658544  3.09597260
 5   25.299771 -0.56127136
 6   10.823243 -1.03699983
 7   49.353938 -2.27445838
 8   -7.053961  0.99025533
 9   -5.753702 -0.68108358
```

Random effects estimates can be visualized by plotting them using the S-PLUS function `plot.ranef.lme`, designed specifically for this purpose. This function offers great flexibility for the display of random effects, its simplest display produces a dotplot of the different coefficients.

Predicted values are returned by the `predict` method. For example, if you are interested in predicting the average measurement for both boys and girls at ages 14, 15, and 16, as well as for subjects `M01` and `F10` at age 13, based on model (13.2), you can create a new data frame, as follows

```
> Orthodont.new <- data.frame(
+ Sex = c("Male", "Male", "Male", "Female", "Female",
+ "Female", "Male", "Female"),
+ age = c(14, 15, 16, 14, 15, 16, 13, 13),
+ Subject = c(NA, NA, NA, NA, NA, NA, "M01", "F10"))
```

and then use

```
> predict(Ortho.fit2, Orthodont.new, level = c(0,1))

   Subject predict.fixed predict.Subject
1               27.32188              NA
2               28.10625              NA
3               28.89063              NA
4               24.08636              NA
5               24.56591              NA
6               25.04545              NA
7      M01      26.53750        29.17264
8      F10      23.60682        19.80758
```

to get the subject-specific and population predictions. The `level` argument is used to define the desired prediction levels, with `0` (zero) referring to the population predictions.

The models considered so far do not assume any special form for the random effects variance-covariance matrix. See the section Advanced Model Fitting in this chapter for a variety of specifications for the structure of this, and the within-group correlation structure. Beyond the available covariance structures, customized structures can also be designed by the user. This topic is also addressed in the section Advanced Model Fitting.

FITTING MODELS USING THE NLME FUNCTION

Nonlinear mixed-effects models, which generalize nonlinear models as well as linear mixed-effects models can be analyzed with the S-PLUS function `nlme`.

There are many advantages to using nonlinear mixed-effects models. For example, the model or expectation function is usually based on sound theory about the mechanism generating the data hence, the model parameters usually have a physical meaning of interest to the investigator.

The `nlme` function is used to fit nonlinear mixed-effects models, as defined in Lindstrom and Bates (1990), using either maximum likelihood or restricted maximum likelihood. These models are of class `nlme` and inherit from the `lme` class, so methods for the `lme` class apply to the `nlme` class as well.

Model Definition

Recall the `CO2` data set introduced before as an example of grouped data with a nonlinear response. The objective of the data collection was to evaluate the effect of plant type and chilling treatment on their CO_2 uptake. The model used in Potvin, *et al.* (1990) is

$$U_{ij} = \phi_{1i}\{1 - \exp[-\phi_{2i}(C_j - \phi_{3i})]\} + \varepsilon_{ij} \qquad (13.4)$$

where U_{ij} denotes the CO_2 uptake rate of the ith plant at the jth CO_2 ambient concentration; ϕ_{1i}, ϕ_{2i}, and ϕ_{3i} denote respectively the asymptotic uptake rate, the uptake growth rate, and the maximum ambient CO_2 concentration at which no uptake is verified for the ith plant; C_j denotes the jth ambient CO_2 level; and the ε_{ij} are independent and identically distributed error terms with distribution $N(0, \sigma^2)$.

Arguments

Several optional arguments can be used with the `nlme` function, but a typical call is

```
nlme(model, data, fixed, random, start)
```

The `model` argument is required and consists of a formula specifying the nonlinear model to be fitted. Any S-PLUS nonlinear formula can be used, giving the function considerable flexibility. For the CO_2 uptake data, we have

```
uptake ~ A * (1 - exp(-B * (conc - C)))
```

from (13.4), where $A = \phi_1$, $B = \phi_2$, and $C = \phi_3$. To enforce the rate parameter ϕ_2 to be positive, while preserving an unrestricted parametrization, you can re-parametrize the model above using *lB = log(B)*

```
uptake ~ A * (1 - exp(-exp(lB) * (conc - C)))
```

Alternatively, you can define an S-PLUS function

```
> CO2.func <-
+ function(conc, A, lB, C) A*(1 - exp(-exp(lB)*(conc - C)))
```

then write the `model` argument as

```
uptake ~ CO2.func(conc, A, lB, C)
```

The advantage of this latter approach is that the analytic derivatives of the model function can be passed to the `nlme` function as the `gradient` attribute of the returned value from `CO2.func` and used in the optimization algorithm. The S-PLUS function `deriv` can be used to create expressions for the derivatives.

```
> CO2.func <-
+ deriv(~ A * ( 1 - exp(-exp(lB) * (conc - C))),
+ c("A", "lB", "C"), function(conc, A, lB, C){})
```

If the value returned by the model function does not have a `gradient` attribute, numerical derivatives are used in the optimization.

The arguments `fixed` and `random` are formulas, or lists of formulas, that define the structures of the fixed and random effects in the model. The first argument is required. In these formulas a `1` on the right hand side of a formula indicates that a single parameter is associated with the effect, but any linear formula in S-PLUS could be used instead. Again, this gives considerable flexibility to the model, as time-dependent parameters can be easily incorporated (for example, when a formula in `fixed` involves a covariate that changes with time).

Usually every parameter in the model will have an associated fixed effect, but it may, or may not, have an associated random effect. Since we assumed that all random effects have mean zero, the inclusion of a random effect without a corresponding fixed effect would be unusual. Note that the `fixed` and `random` formulas could be directly incorporated in the model declaration. The approach used in `nlme` allows for more efficient calculation of derivatives.

For the CO_2 uptake data, if you want to fit a model in which all parameters are random and no covariates are included, use

```
fixed = A + lB + C ~ 1, random = A + lB + C ~ 1
```

By default, random = fixed, so the `random` argument can be omitted. Because `CO2` is a `groupedData` object, no grouping structure need be explicitly given in `random`, as it is extracted from the `groupedData` display formula.

Alternatively, the grouping structure can be included in the formula as a conditioning expression.

```
random = A + lB + C ~ 1 | Plant
```

If you want to estimate the (fixed) effects of plant type and chilling treatment on the parameters in the model, use

```
fixed = A + lB + C ~ Type * Treatment,
random = A + lB + C ~ 1
```

`data` is an optional argument to `nlme` that names a data frame in which the variables in `model`, `fixed`, and `random` are found, and `start` provides a list of starting values for the iterative algorithm. Only the fixed effects starting estimates are required. The default starting estimates for the random effects are zero.

Examples

A simple call to `nlme` to fit model (13.4), without any covariates and with all parameters as mixed effects, is

```
> CO2.fit1 <-
+ nlme(model = uptake ~ CO2.func(conc, A, lB, C),
+ fixed = A + lB + C ~ 1, data = CO2,
+ start = c(30, log(0.01), 50))
```

The initial values for the fixed effects were obtained from Potvin, *et al.* (1990).

MANIPULATING NLME OBJECTS

Objects returned by the `nlme` function are of class `nlme` which inherits from `lme`. All methods described for the `lme` class are also available for the `nlme` class. In fact, with the exception of the `predict` method, all methods are common to both classes. We illustrate their use here with the CO_2 uptake data described above.

The print Method

The `print` method provides a brief description of the estimation results. It gives estimates of the standard errors and correlations of the random effects, of the within-group variance, and of the fixed effects.

```
> print(CO2.fit1)

Nonlinear mixed-effects model fit by maximum likelihood
  Model: uptake ~ CO2.func(conc, A, lB, C)
  Data: CO2
  Log-likelihood: -201.3102
  Fixed: A + lB + C ~ 1
        A        lB        C
 32.47374 -4.636226 43.54071

Random effects:
 Formula: list(A ~ 1, lB ~ 1, C ~ 1)
 Level: Plant
 Structure: General positive-definite
             StdDev   Corr
       A  9.5099518 A      lB
      lB  0.1282905 -0.160
       C 10.4078430  0.999 -0.139
Residual  1.7663862

Number of Observations: 84
Number of Groups: 12
```

Note that there is strong correlation between the A and the C random effects and that these have small correlations with the lB random effect. The scatter plot matrix of the random effects, obtained using the `pairs` method:

```
> pairs(CO2.fit1, ~ranef(.))
```

and shown in Figure 13.10, gives a graphical description of the random effects correlation structure.

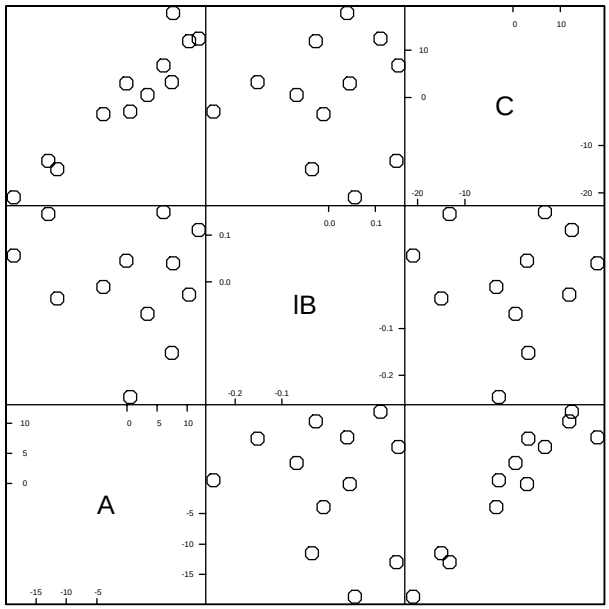

Figure 13.10: *Scatterplot matrix of the estimated random effects in model (13.4).*

The correlation between A and C may be due to the fact that the plant type and the chilling treatment, which were not included in the `CO2.fit1` model, are affecting A and C in the similar ways.

The `plot` method for the `ranef.lme` class can be used to explore the dependence of the individual parameters A, lB, and C in model (13.4) on plant type and chilling factor.

```
> plot(ranef(CO2.fit1, augFrame = T),        #Figure 13.11
+ outer = ~Treatment*Type,
+ layout = c(3,1))
```

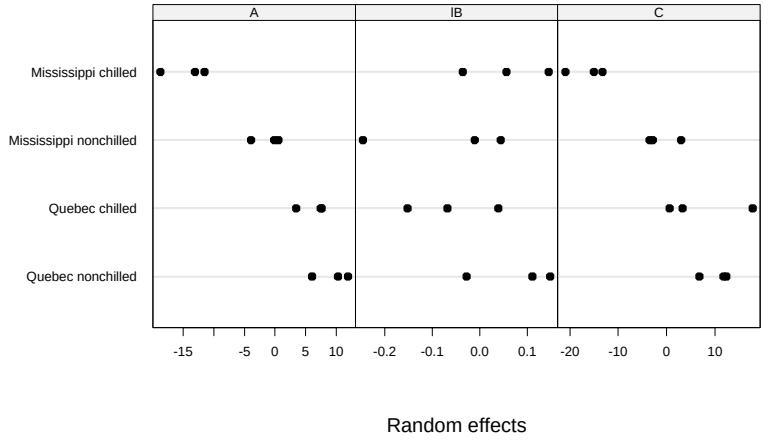

Figure 13.11: *Estimated random effects versus plant type and chilling treatment.*

These plots indicate that chilled plants tend to have smaller values of A and C, but the Mississippi plants seem to be much more affected than the Quebec plants, suggesting an interaction effect between plant type and chilling treatment. There is no clear pattern of dependence between lB and the treatment factors, suggesting that this parameter is not significantly affected by either plant type or chilling treatment.

We can then update the fitted object letting the A and C fixed effects depend on the treatment factors, as below.

```
> CO2.fit2 <- update(CO2.fit1,
+ fixed = list(A+C ~ Treatment * Type, lB ~ 1),
+ start = c(32.55, 0, 0, 0, 41.56, 0, 0, 0, -4.6))
```

The summary Method

The `summary` method provides more detailed information on the new fitted object.

```
> summary(CO2.fit2)

Nonlinear mixed-effects model fit by maximum likelihood
  Model: uptake ~ CO2.func(conc, A, lB, C)
 Data: CO2
       AIC      BIC    logLik
  392.4074 431.3004 -180.2037
```

```
Random effects:
 Formula: list(A ~ 1, lB ~ 1, C ~ 1)
 Level: Plant
 Structure: General positive-definite
                StdDev   Corr
A.(Intercept) 2.3709586 A.(In) lB
           lB 0.1475551 -0.336
C.(Intercept) 8.1632925  0.355  0.761
     Residual 1.7113001

Fixed effects: list(A + C ~ Treatment * Type, lB ~ 1)
                    Value Std.Error DF   t-value
   A.(Intercept)  32.47034  0.787419 64  41.23641
     A.Treatment  -4.23974  0.735246 64  -5.76642
          A.Type  -7.93288  0.737964 64 -10.74969
A.Treatment:Type  -2.39355  0.735683 64  -3.25352
   C.(Intercept)  39.96569  6.411938 64   6.23301
     C.Treatment  -7.83988  4.399918 64  -1.78182
          C.Type -10.75033  4.435898 64  -2.42348
C.Treatment:Type -12.25407  4.420052 64  -2.77238
              lB  -4.65064  0.080102 64 -58.05876
                 p-value
   A.(Intercept)  <.0001
     A.Treatment  <.0001
          A.Type  <.0001
A.Treatment:Type  0.0018
   C.(Intercept)  <.0001
     C.Treatment  0.0795
          C.Type  0.0182
C.Treatment:Type  0.0073
              lB  <.0001
 Correlation:
...
```

The small *p*-values of the t-statistics associated with the `Treatment:Type` effects indicate that both factors have a significant effect on parameters A and C and their joint effect is not just the sum of the individual effects.

The anova Method

You can investigate the joint effect of `Treatment` and `Type` on A and C using the `anova` method.

```
> anova(CO2.fit2,
+ terms = c("A.Treatment", "A.Type", "A.Treatment:Type"))

F-test for: A.Treatment, A.Type, A.Treatment:Type
  numDF denDF  F-value p-value
1     3    64 51.77681  <.0001

> anova(CO2.fit2,
+ terms = c("C.Treatment", "C.Type", "C.Treatment:Type"))

F-test for: C.Treatment, C.Type, C.Treatment:Type
  numDF denDF  F-value p-value
1     3    64 2.939707  0.0397
```

The p-values of the Wald F-tests suggest that `Treatment` and `Type` have a stronger influence on A than on C.

The plot Method

Diagnostic plots can be obtained using the `plot` method, in the same way as for `lme` objects. For example, plots of the standardized residuals versus fitted values broken up by `Treatment` and `Type`, shown in Figure 13.12, are obtained with

```
> plot(CO2.fit2,                              #Figure 13.12
+ resid(., type = "p") ~ fitted(.) | Treatment * Type)
```

The plots do not indicate any departures from the assumptions in the model–no outliers seem to be present and the residuals are symmetrically scattered around the $y = 0$ line, with constant spread for different levels of the fitted values.

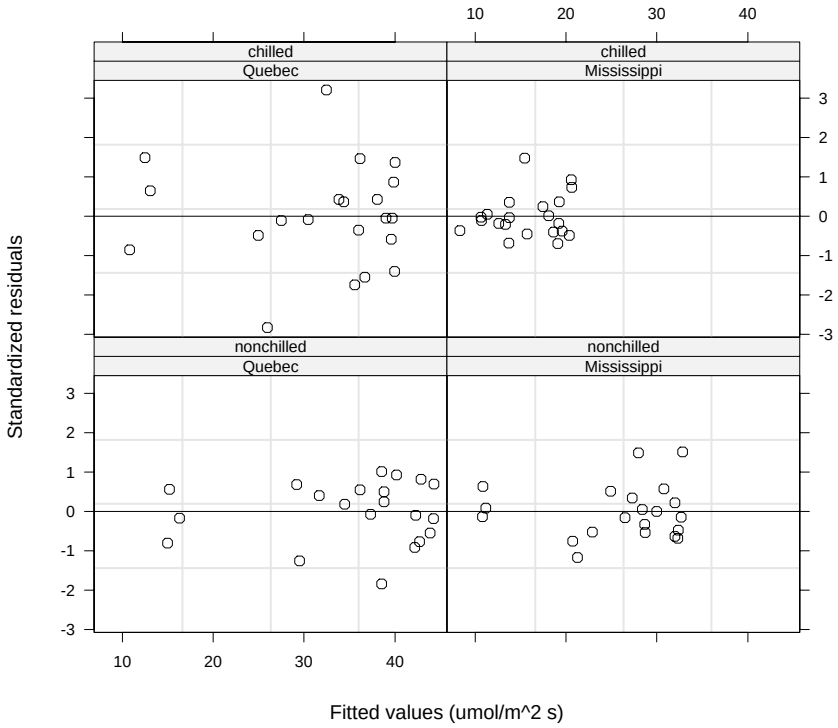

Figure 13.12: *Standardized residuals versus fitted values for the* `CO2.fit2` *fit, by plant type and chilling treatment.*

Other Methods

Predictions are returned by the `predict` method. For example, to obtain the population predictions of the CO_2 uptake rate for Quebec and Mississippi plants under chilling and no chilling, at ambient CO_2 concentrations of 75, 100, 200, and 500 μL/L, first define

```
> CO2.new <- data.frame(
+ Type = rep(c("Quebec","Mississippi"), c(8, 8)),
+ Treatment =rep(rep(c("chilled","nonchilled"),c(4,4)),2),
+ conc = rep(c(75, 100, 200, 500), 4))
```

and then use the following to obtain the predictions:

```
> predict(CO2.fit2, CO2.new, level = 0)

 [1]  6.667665 13.444049 28.898573 38.007578 10.133070
 [6] 16.957681 32.522196 41.696029  8.363814 10.391107
[11] 15.014640 17.739783  6.785150 11.967006 23.784979
[16] 30.750575
attr(, "label"):
[1] "Predicted values (umol/m^2 s)"
```

The `augPred` method can be used for plotting smooth fitted curves by calculating fitted values at closely spaced concentrations. Figure 13.13 presents the individual fitted curves for all twelve plants evaluated at 51 concentrations between 50 and 1000 μL/L, obtained with

```
> plot(augPred(CO2.fit2))                    #Figure 13.13
```

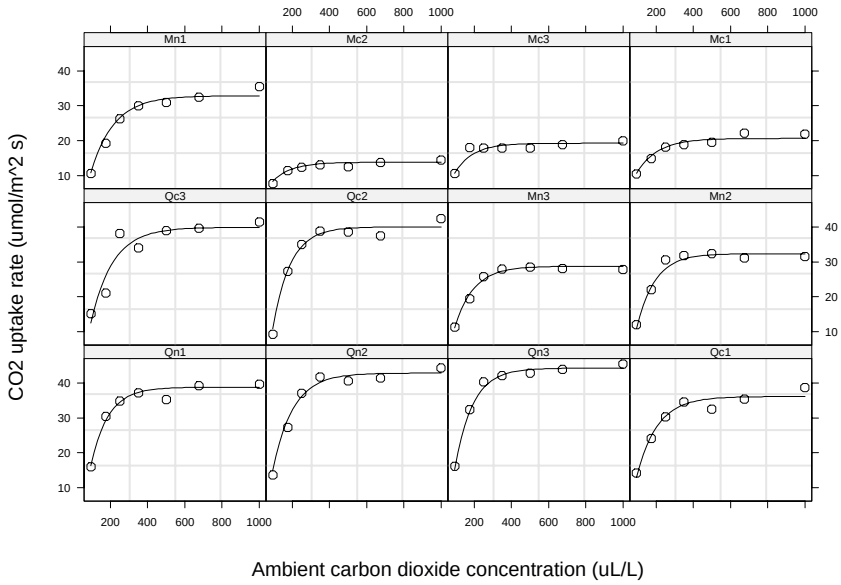

Figure 13.13: *Individual fitted curves for the twelve plants in the CO_2 uptake data based on the* `CO2.fit2` *object.*

The `CO2.fit2` model explains the data reasonably well, as evidenced by the close agreement between its fitted values and the observed uptake rates.

Methods for extracting components from a fitted `nlme` object are also available and parallel those for `lme` objects. Some of the most commonly used are `coef`, `fitted`, `fixef`, `ranef`, and `resid`.

ADVANCED MODEL FITTING

In many practical applications, we want to restrict the random-effects variance-covariance matrix to special forms parameterized by fewer parameters. For example, we may want to assume that the random effects are independent and their variance-covariance matrix is then a diagonal matrix. We may also want to make specific assumptions about the within-group error structure.

Both the `lme` function and the `nlme` function include advanced options for defining positive-definite matrices, correlation structures, and variance functions.

Positive-Definite Matrix Structures

Different positive-definite matrix structures can be used to represent the random effects variance-covariance matrix. These are organized in the code as different `pdMat` classes. Table 13.1 lists the available `pdMat` classes.

Table 13.1: *Classes of positive-definite matrices.*

Class	Description
`pdSymm`	general positive-definite
`pdDiag`	diagonal
`pdIdent`	multiple of an identity
`pdCompSymm`	compound symmetry
`pdBlocked`	block diagonal

By default, the `pdSymm` class is used to represent a random effects covariance matrix. The desired `pdMat` class must be specified with the `random` argument.

You can define your own `pdMat` classes by specifying a constructor function and, at a minimum, methods for the functions `pdConstruct`, `pdMatrix` and `coef`. For examples of these functions, see the methods for classes `pdSymm` and `pdDiag`.

Examples

To fit a model with independent intercept and slope random effects in model (13.2), use

```
>  Ortho.fit3 <- update(Ortho.fit2, random = pdDiag(~ age))
>  Ortho.fit3

Linear mixed-effects model fit by REML
  Data: Orthodont
  Log-restricted-likelihood: -216.5755
  Fixed: distance ~ age + Sex + age:Sex
 (Intercept)      age      Sex    age:Sex
   16.34062 0.784375 1.032102 -0.3048295

Random effects:
 Formula:  ~ age | Subject
 Structure: Diagonal
        (Intercept)        age Residual
StdDev:    1.554633 0.08801454   1.3655

Number of Observations: 108
Number of Groups: 27
```

The grouping structure is inferred from the `groupedData` display formula. Alternatively, `random` could have been passed to the function as

```
random = list(Subject = pdDiag(~ age))
```

To test if the random effects in `CO2.fit2` can be assumed to be independent, use

```
> CO2.fit3 <- update(CO2.fit2, random = pdDiag(A+lB+C~1))
> anova(CO2.fit2, CO2.fit3)

         Model df      AIC      BIC    logLik   Test
CO2.fit2     1 16 392.4074 431.3004 -180.2037
CO2.fit3     2 13 391.3930 422.9936 -182.6965 1 vs 2
          L.Ratio p-value
CO2.fit2
CO2.fit3 4.985649  0.1729
```

Note that because the two models have the same fixed effects structure, the likelihood ratio test based on REML is meaningful.

As evidenced by the large *p*-value for the likelihood ratio test in the `anova` output, the independence between the random effects seems plausible.

Correlation Structures and Variance Functions

The within-group error covariance structure can be flexibly modeled by combining correlation structures and variance functions. Correlation structures are used to model within-group correlations, not captured by the random effects. These are generally associated with temporal or spatial dependencies. The variance function structures are used to model heteroscedasticity in the within-group errors.

Similar to the positive-definite matrix structures described in `pdMat`, the different correlation and variance functions structures are organized into `corStruct` and `varFunc` classes. Table 13.2 and Table 13.3 list the standard classes for each structure.

Table 13.2: *Classes of correlation structures.*

Class	Description
`corAR1`	AR(1)
`corARMA`	ARMA(p,q)
`corCAR1`	continuous AR(1)
`corCompSymm`	compound symmetry
`corExp`	exponential spatial correlation
`corGaus`	Gaussian spatial correlation
`corHF`	Huyn-Feldt correlation
`corLin`	linear spatial correlation
`corRatio`	rational quadratic spatial correlation
`corSpher`	spherical spatial correlation
`corSymm`	general correlation matrix

Table 13.3: *Classes of variance functions.*

Class	Description
`varExp`	exponential of a variance covariate
`varPower`	power of a variance covariate
`varConstPower`	constant plus power of a variance covariate
`varIdent`	different variances per level of a factor
`varFixed`	fixed weights, determined by a variance covariate
`varComb`	combination of variance functions

The optional argument `correlation` is used to specify a correlation structure and the optional argument `weights` is used for variance functions. By default, the within-group errors are assumed to independent and homoscedastic.

You can define your own correlation and variance function classes by specifying appropriate constructor functions and a few method functions. For a new correlation structure, method functions must be defined for at least `corMatrix` and `coef`. For examples of these functions, see the methods for classes `corSymm` and `corAR1`. A new variance function structure requires methods for at least `coef`, `coef<-`, and `initialize`. For examples of these functions, see the methods for class `varPower`.

Examples

The residual versus fitted values plot of the residuals on Figure 13.7 suggests that different variances should be allowed for boys and girls. You can test that by updating the fit using the `varIdent` variance function structure.

```
> Ortho.fit4 <- update(Ortho.fit3,
+ weights = varIdent(form = ~1|Sex))
```

```
> Ortho.fit4

Linear mixed-effects model fit by REML
  Data: Orthodont
  Log-restricted-likelihood: -207.4704
  Fixed: distance ~ age + Sex + age:Sex
 (Intercept)       age       Sex    age:Sex
   16.85668 0.6319602 0.5160511 -0.1524148

Random effects:
 Formula:  ~ age | Subject
 Structure: Diagonal
        (Intercept)       age Residual
StdDev:    1.448708 0.1094044  1.65842

Variance function:
 Structure: Different standard deviations per stratum
 Formula:  ~ 1 | Sex
 Parameter estimates:
 Male   Female
    1 0.425368
Number of Observations: 108
Number of Groups: 27

> anova(Ortho.fit3, Ortho.fit4)

           Model df      AIC      BIC    logLik
Ortho.fit3     1  7 449.9235 468.4343 -217.9618
Ortho.fit4     2  8 430.9407 452.0958 -207.4704
             Test  L.Ratio p-value
Ortho.fit3
Ortho.fit4 1 vs 2 20.98281  <.0001
```

There is strong indication that the orthodontic distance is less variable in girls than in boys.

The fitted object can be referenced in the `form` argument to the `varFunc` constructors through the symbol ".". For example, to use a variance function that is an arbitrary power of the fitted values in model (13.3), update `Pixel.fit1` as below.

```
> Pixel.fit2 <- update(Pixel.fit1,
+ weights = varPower(form=~fitted(.)))
```

```
> Pixel.fit2

Linear mixed-effects model fit by REML
  Data: Pixel
  Log-restricted-likelihood: -412.4592
  Fixed: pixel ~ day + day^2
 (Intercept)      day   I(day^2)
    1073.312 6.101551 -0.3663839

Random effects:
 Formula:  ~ day | Dog
 Structure: General positive-definite
               StdDev   Corr
(Intercept) 28.498576 (Inter
        day  1.871504 -0.565

 Formula:  ~ 1 | Side %in% Dog
        (Intercept)      Residual
StdDev:    16.65769 4.534689e-006

Variance function:
 Structure: Power of variance covariate
 Formula:  ~ fitted(.)
 Parameter estimates:
    power
 2.074139
Number of Observations: 102
Number of Groups:
 Dog Side %in% Dog
  10            20

> anova (Pixel.fit1, Pixel.fit2)

           Model df      AIC      BIC    logLik
Pixel.fit1     1  8 841.2102 861.9712 -412.6051
Pixel.fit2     2  9 842.9184 866.2744 -412.4592
             Test   L.Ratio p-value
Pixel.fit1
Pixel.fit2 1 vs 2 0.2918317   0.589
```

There is no evidence of heteroscedasticity in this case, as evidenced by the large p-value of the likelihood ratio test in the `anova` output. Because the default value for `form` in `varPower` is `~fitted(.)`, it suffices to use `weights = varPower()` in this example.

We can test for the presence of an autocorrelation of lag 1 in the orthodontic growth example by updating `Ortho.fit4` as below.

```
> Ortho.fit5 <- update(Ortho.fit4, corr = corAR1())
> Ortho.fit5

Linear mixed-effects model fit by REML
  Data: Orthodont
  Log-restricted-likelihood: -207.4233
  Fixed: distance ~ age + Sex + age:Sex
(Intercept)       age       Sex    age:Sex
   16.84766 0.6325383 0.5303993 -0.1534489

Random effects:
 Formula:  ~ age | Subject
 Structure: Diagonal
        (Intercept)       age Residual
StdDev:    1.451008 0.1121105 1.630654

Correlation Structure: AR(1)
 Parameter estimate(s):
         Phi
-0.05702521
Variance function:
 Structure: Different standard deviations per stratum
 Formula:  ~ 1 | Sex
 Parameter estimates:
 Male    Female
    1 0.4250633
Number of Observations: 108
Number of Groups: 27
```

```
> anova(Ortho.fit4, Ortho.fit5)

           Model df      AIC      BIC    logLik
Ortho.fit4     1  8 430.9407 452.0958 -207.4704
Ortho.fit5     2  9 432.8467 456.6462 -207.4233
             Test  L.Ratio p-value
Ortho.fit4
Ortho.fit5 1 vs 2 0.094035  0.7591
```

The large *p*-value of the likelihood ratio test indicates that the autocorrelation is not present. Note that the correlation structure is used together with the variance function, representing an heterogeneous AR(1) process (Littel *et al.*, 1996}. Because the two structures are defined and constructed separately, any correlation structure can be combined with any variance function.

As a final example, you can test for the presence of serial correlation in the within-group errors of the nonlinear `CO2` model like this:

```
> CO2.fit4 <- update(CO2.fit3, correlation = corAR1())
> anova(CO2.fit3, CO2.fit4)

         Model df      AIC      BIC    logLik   Test
CO2.fit3     1 13 391.3930 422.9936 -182.6965
CO2.fit4     2 14 393.2968 427.3283 -182.6484 1 vs 2
            L.Ratio p-value
CO2.fit3
CO2.fit4 0.09616825  0.7565
```

There does not appear to be evidence of within-group serial correlation.

Self-Starting Functions

The S-PLUS function `nlsList` can be used to create a list of fits to each group of a `groupedData` object. This function is an extension of the `nls` function, discussed in detail in a later chapter. To call `nlsList`, as well as when using `nlme`, the user must provide either initial estimates for the fixed-effects parameters to be fitted or a self-starting function.

One way to provide initial values to `nls` and hence to `nlsList` is to include them in the data frame as a `parameters` attribute. Both `nlsList` and `nlme` function also have an argument `start` to be used when providing the initial estimates as input. Alternatively, the function to derive initial estimates can be added to the model function

itself as an attribute. This constitutes a `selfStart` function in S-PLUS. When a self-starting function is used in calls to `nlsList` and `nlme`, initial estimates for the parameters are taken directly from the "`initial`" attribute of the self-starting function. A self-starting function is a class of models useful for particular applications. Several self-starting functions are provided with S-PLUS. The following four self-starting functions are useful in Biostatistics.

- Biexponential model: `SSbiexp(input, A1, lrc1, A2, lrc2)`

$$\alpha_1 e^{-e^{\beta_1} t} + \alpha_2 e^{-e^{\beta_2} t},$$

where `input` = t is a covariate, and `A1` = α_1, `A2` = α_2, `lrc1` = β_1, and `lrc2` = β_2 are parameters.

- First Order Compartment model: `SSfol(Dose, input, lCl, lKa, lKe)`

$$\frac{e^{\beta} \cdot e^{\gamma} \cdot (e^{-e^{\gamma} t} - e^{-e^{\cdot}}}{e^{\alpha} \cdot (e^{\beta} - e^{\gamma})},$$

where `Dose` = d is a covariate representing the initial dose, `input` = t is a covariate at which to evaluate the model , and `lCl` = α, `lKa` = β, and `lKe` = γ are parameters.

- Four-Parameter Logistic model: `SSfpl(input, A, B, xmid, scal)`

$$\alpha + \frac{\beta - \alpha}{1 + e^{-(x-\gamma)/\theta}},$$

where `input` = x is a covariate, and `A` = α, `B` = β, `xmid` = γ, and `scal` = θ are parameters.

- Logistic model: `SSlogis(time, Asym, xmid, scal)`

$$\frac{\alpha}{1 + e^{-(t-\beta)/\gamma}},$$

where `time` = t is a covariate, and `Asym` = α, `xmid` = β, and `scal` = γ are parameters.

Other self-starting functions already built-in S-PLUS are listed in Table 13.4 below and details about them can be found in their corresponding online help files. You can define your own self-starting function by using the function `selfStart`.

Table 13.4: *Additional self-starting models in S-PLUS.*

Function	Model
`SSasymp`	Asymptotic Regression
`SSasympOff`	Asymptotic Regression with an Offset
`SSasympOrig`	Asymptotic Regression through the Origin
`SSmicmen`	Michaelis-Menten

Examples

We can apply the self-starting function `SSlogis` to the `Soybean` data introduced above, to verify the hypothesis that a logistic model can be used represent leaf growth. The `nlsList` call is as follows.

```
> Soybean.nlsList <- nlsList(weight ~
+ SSlogis(Time, Asym, xmid, scal) | Plot, data = Soybean)

Error in nls(y ~ 1/(1 + exp((xmid - x)/scal)), data ..:
singular gradient matrix
Dumped
```

The error message indicates that a group could not be fitted by `nls`. The object `Soybean.nlsList` is still created.

Warning:

On occasion `nlsList` will give one or more errors as a result of one or more groups not being fitted adequately with `nls`. The remaining groups are still fitted.

The results in `Soybean.nlsList` below show that one of the groups below (1989P8) could not be fitted appropriately with the logistic model, if the within-group variations are not adjusted.

```
> coef(Soybean.nlsList)

             Asym      xmid      scal
1988F4  15.151338  52.83361  5.176641
1988F2  19.745503  56.57514  8.406720
1988F1  20.338576  57.40265  9.604870
1988F7  19.871706  56.16236  8.069718
1988F5  30.647205  64.12857 11.262351
1989P2  28.294391  67.17185 12.522720
...
1989P8         NA        NA        NA
1990F2  19.459767  66.28652 13.158397
...
1990P5  19.543787  51.14830  7.291976
1990P2  25.787317  62.35974 11.657019
1990P4  26.132712  61.20345 10.973765
```

There exists an `nlme` method for `nlsList` objects. Population parameters and individual random-effects can be fitted to the `Soybean` data even when a group could not be fitted above by using the simple call:

```
> Soybean.fit1 <- nlme(Soybean.nlsList)
```

Again, we use the summary method to obtain more detailed information on the fitted object.

```
> summary(Soybean.fit1)

Nonlinear mixed-effects model fit by maximum likelihood
  Model: weight ~ SSlogis(Time, Asym, xmid, scal)
 Data: Soybean
       AIC      BIC    logLik
  1499.671 1539.881 -739.8353

Random effects:
 Formula: list(Asym ~ 1, xmid ~ 1, scal ~ 1)
 Level: Plot
```

```
 Structure: General positive-definite
           StdDev  Corr
    Asym 5.201130 Asym  xmid
    xmid 4.197413 0.721
    scal 1.404698 0.711 0.958
Residual 1.123465

Fixed effects: list(Asym ~ 1, xmid ~ 1, scal ~ 1)
        Value Std.Error  DF  t-value p-value
Asym 19.25303 0.8031850 362 23.97086  <.0001
xmid 55.01986 0.7272665 362 75.65296  <.0001
scal  8.40334 0.3152861 362 26.65306  <.0001
 Correlation:
      Asym  xmid
xmid 0.724
scal 0.620 0.807

Standardized Within-Group Residuals:
       Min         Q1         Med        Q3     Max
 -6.086891 -0.2216123 -0.03390552 0.2974126 4.84693

Number of Observations: 412
Number of Groups: 48
```

`Soybean.fit1` does not incorporate covariates or within-group errors. Comparing the estimated standard deviations and means of `Asym`, `xmid`, and `scal`, the asymptotic weight `Asym` has the highest coefficient of variation ($5.2/19.25 = 0.27$). Modeling this random-effects parameter is the focus of the following analyses.

We can try to model the asymptotic weight `Asym` as a function of the variety of the genotype and planting year. To model the within-group errors, we will assume the serial correlation follows an AR(1). Given that the observations are not equally spaced in time, we need to use the continuous form of the AR process, and provide the time variable. From Figure 13.14, obtained with a simple call to the plot method for the object `Soybean.fit1`, the within-group variance is assumed to be proportional to some power of the absolute value of the predictions.

The improved `nlme` model is fitted to the `Soybean` data below. In fitting the full model, the results from `Soybean.nlsList` are used to derive initial estimates in the parametrization of `Asym`.

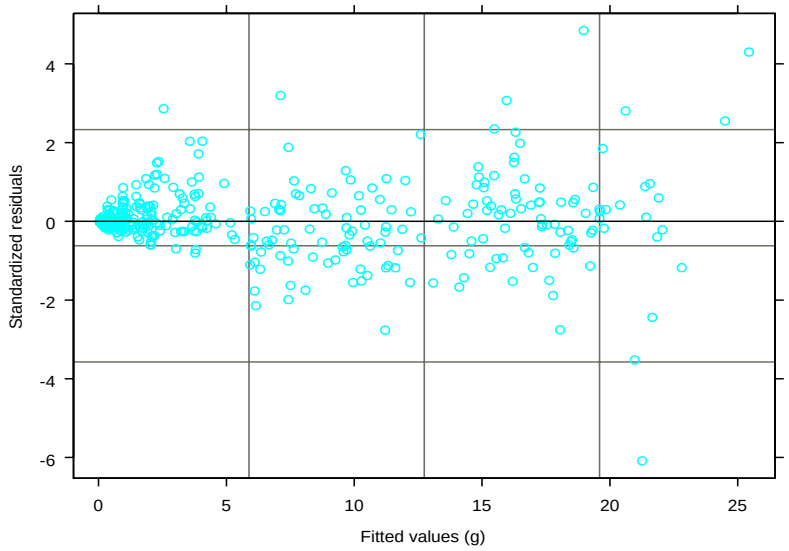

Figure 13.14: *Standardized residuals plot used to diagnose model* `Soybean.fit1`.

```
> Soybean.fit2 <- nlme(weight ~ SSlogis(Time, Asym, xmid,
+ scal), data = Soybean, fixed = list(Asym ~ Variety * Year,
+ xmid ~ 1, scal ~1), random = list(Asym ~ 1, xmid ~ 1,
+ scal ~ 1), start = c(20.08425, 2.03699,-3.785161,
+ 0.3036094, 1.497311,-1.084704, 55.02058, 8.402632),
+ correlation = corCAR1(~Time), weights = varPower())
> plot(Soybean.fit2)              #Figure 13.15
> anova(Soybean.fit1,Soybean.fit2)

             Model df      AIC      BIC    logLik
Soybean.fit1     1 10 1499.671 1539.881 -739.8353
Soybean.fit2     2 17  678.619  746.976 -322.3093

             Test  L.Ratio  p-value

           1 vs 2  835.052  <.0001
```

The `anova` function is used to compare these fits. The progress in each of the log likelihood, AIC, and BIC is tremendous. Figure 13.15 shows the residuals plots.

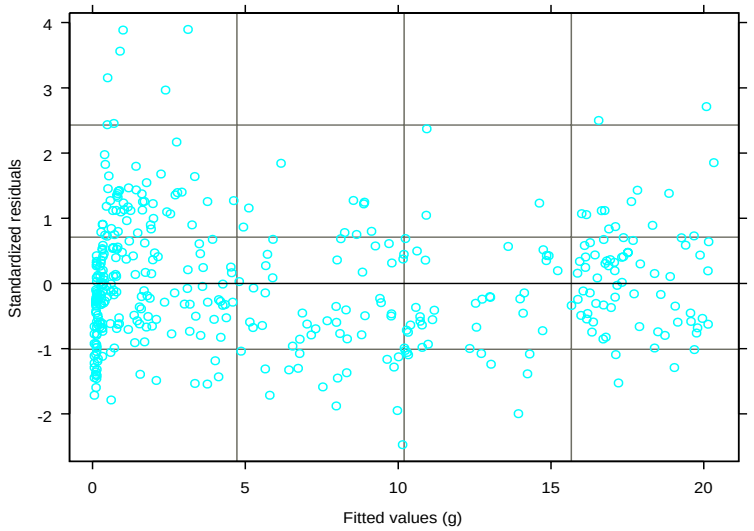

Figure 13.15: *Standardized residuals plot used to diagnose model* `Soybean.fit2`.

These residuals confirm the selection of variance-covariance function. We conclude that the variety of the genotype and the year of planting have a large impact on the limiting leaf weight. The experimental strain gains 2.5 grams in the limit.

Modeling Spatial Dependence

Two main classes of dependence among the within-group errors can be modeled using the tools in the Mixed-effects library of S-PLUS: *temporal* and *spatial*. To model serial correlation (temporal dependence), several correlation structures were already introduced in Table 13.2. To assess and model spatial dependence among the within-group errors we use the `Variogram` function in S-PLUS.

The `Variogram` method for the `lme` class estimates the sample semivariogram from the residuals of the `lme` object. The semivariogram can then be plotted using its corresponding `plot`

method. If the residuals present spatial dependence then you will need to determine a model for this dependence or its correlation structure.

We use the `corSpatial` function to model spatial dependence in the within-group errors. This function is a constructor for the `corSpatial` class, representing a spatial correlation structure. This class is "virtual," having five "real" classes, corresponding to specific spatial correlation structures, associated with it: `corExp`, `corGaus`, `corLin`, `corRatio`, and `corSpher`. The returned object will inherit from one of these "real" classes, determined by the type argument, and from the "virtual" `corSpatial` class. Objects created using this constructor need to be later initialized using the appropriate initialize method.

A typical call to the variogram function for a mixed-effects model would be, for example :

```
> plot(Variogram(Soybean.fit1, form= ~Time))
```

The resulting plot (not shown) does not show a strong pattern in the semivariogram of the residuals from model `Soybean.fit1` in terms of time distance. Refitting the model without the AR(1) term for the within-error correlation shows that `Soybean.fit2` may indeed be over-parameterized and that only the change in the fixed-effects model and the use of weights explain the improvement.

```
> anova(Soybean.fit1,Soybean.fit3,Soybean.fit2)

             Model df      AIC      BIC    logLik
Soybean.fit1     1  10 1499.671 1539.881 -739.8353
Soybean.fit3     2  16  674.669  739.005 -321.3345
Soybean.fit2     3  17  678.619  746.976 -322.3093

                 Test  L.Ratio p-value

                1 vs 2 837.0015  <.0001
                2 vs 3   1.9496  0.1626
```

REFERENCES

Becker, R.A., Cleveland, W.S., and Shyu, M.-J. (1996). The visual design and control of trellis graphics displays. *J. of Computational and Graphical Statistics*, 5(2): 123-156.

Chambers, J.M. and Hastie, T.J. (eds.) (1992). *Statistical Models in S*. Wadsworth, Belmont, CA.

Davidian, M. and Giltinan, D.M. (1995). *Nonlinear Models for Repeated Measurement Data*. Chapman & Hall, London.

Goldstein, H. (1995). *Multilevel Statistical Models*. Halsted Press, New York.

Laird, N.M. and Ware, J.H. (1982). Random-effects models for longitudinal data. *Biometrics*, 38: 963-974.

Lindstrom, M.J. and Bates, D.M. (1990). Nonlinear mixed effects models for repeated measures data. *Biometrics*, 46: 673-687.

Littel, R.C., Milliken, G.A., Stroup, W.W. and Wolfinger, R.D. (1996). *SAS Systems for Mixed Models*. SAS Institute Inc., Cary, NC.

Longford, N.T. (1993). *Random Coefficient Models*. Oxford University Press, New York.

Milliken, G.A. and Johnson, D.E. (1992). *Analysis of Messy Data, Volume 1: Designed Experiments*. Chapman & Hall.

Pinheiro, J.C. (1994). *Topics in Mixed Effect Models*. Unpublished Ph.D. thesis, University of Wisconsin-Madison.

Potthoff, R.F. and Roy, S.N. (1964). A generalized multivariate analysis of variance model useful especially for growth curve problems. *Biometrika*, 51: 313-326.

Potvin, C., Lechowicz, M.J., and Tardif, S. (1990). The statistical analysis of ecophysiological response curves obtained from experiments involving repeated measures. *Ecology*, 71: 1389-1400.

Venables, W.N. and Ripley, B.D. (1997) *Modern Applied Statistics with* S-PLUS*, 2nd Edition*. Springer-Verlag, New York.

NONLINEAR MODELS 14

INTRODUCTION

This chapter covers the fitting of nonlinear models such as in nonlinear regression, likelihood models, and Bayesian estimation. Nonlinear models are more general than the linear models usually discussed. Specifying nonlinear models typically requires one or more of the following: more general formulas, extended data frames, starting values and derivatives.

The two most common fitting criteria for nonlinear models considered are Minimum sum and Minimum sum-of-squares. Minimum sum minimizes the sum of contributions from observations (the maximum likelihood problem). Minimum sum-of-squares minimizes the sum of squared residuals (the nonlinear least-squares regression problem).

The first sections of this chapter summarizes the use of the nonlinear optimization functions. Starting with the section Examples of Nonlinear Models, the use of the `ms` and `nls` functions are examined, along with corresponding examples and theory, in much more detail.

OPTIMIZATION FUNCTIONS

S-PLUS has several functions for finding roots of equations and local maxima and minima of functions, as shown in Table 14.1.

Table 14.1: *The range of S-PLUS functions for finding roots, maxima, and minima.*

Function	Description
`polyroot`	Finds the roots of a complex polynomial equation.
`uniroot`	Finds the root of a univariate real-valued function in a user-supplied interval.
`peaks`	Finds local maxima in a set of discrete points.
`optimize`	Approximates a local optimum of a continuous univariate function within a given interval.
`ms`	Finds a local minimum of a multivariate function.
`nlmin`	Finds a local minimum of a nonlinear function using a general quasi-Newton optimizer.
`nlminb`	Local minimizer for smooth nonlinear functions subject to bound-constrained parameters.
`nls`	Finds a local minimum of the sums of squares of one or more multivariate functions.
`nlregb`	Local minimizer for sums of squares of nonlinear functions subject to bound-constrained parameters.
`nnls`	Finds least-squares solution subject to the constraint that the coefficients be nonnegative.

Finding Roots

The function `polyroot` finds the roots (zeros) of the complex-valued polynomial equation $a_k z^k + \ldots + a_1 z + a_0 = 0$. The input to `polyroot` is the vector of coefficients $c(a_0, \ldots, a_k)$. For example, to solve the equation `z^2+5z+6=0`, use `polyroot` as follows:

```
> polyroot(c(6,5,1))

[1] -2+2.584939e-26i -3-2.584939e-26i
```

The function `uniroot` finds a zero of a continuous, univariate, real-valued function within a user-specified interval for which the function has opposite signs at the endpoints. The input to `uniroot` includes the function, the lower and upper endpoints of the interval, and any additional arguments to the function. For example, suppose you have the function:

```
> my.fcn

function(x, amp=1, per=2*pi, horshft=0, vershft=0)
{
  amp * sin(((2*pi)/per) * (x-horshft)) + vershft
}
```

This is the sine function with amplitude `abs(amp)`, period `abs(per)`, horizontal (phase) shift `horshft` and vertical shift `vershft`. To find a root of the function `my.fcn` in the interval $[\pi/2, 3\pi/2]$ using its default arguments, type:

```
> uniroot(my.fcn, interval = c(pi/2, 3*pi/2))

$root
[1] 3.141593
. . .
```

To find a root of `my.fcn` in the interval $[\pi/4, 3\pi/4]$ with the period set to π, type:

```
> uniroot(my.fcn, interval = c(pi/4, 3*pi/4), per=pi)

$root:
[1] 1.570796
. . .
```

```
> pi/2

[1] 1.570796
```

See the help file for `uniroot` for information on other arguments to this function.

Finding Local Maxima and Minima of Univariate Functions

The `peaks` function takes a data object `x` and returns an object of the same type with logical values: `T` if a point is a local maximum; otherwise, `F`:

```
> peaks(corn.rain)

1890: F T F F F F T F T F T F T F F F F T F F F F T F F T F
1917: T F F F T F F T F T F
```

Use `peaks` on the data object `-x` to find local minima:

```
> peaks(-corn.rain)

1890: F F F F T F F F F F F T F F F T F F F F F T F T F F T
1917: F T F F F T F F T F F
```

To find a local optimum (maximum or minimum) of a continuous univariate function within a particular interval, use the `optimize` function. The input to `optimize` includes the function to optimize, the lower and upper endpoints of the interval, which optimum to look for (maximum versus minimum) and any additional arguments to the function.

```
> optimize(my.fcn, c(0, pi), maximum=T)

$maximum:
[1] 1.570799

$objective:
[1] -1

$nf:
[1] 10

$interval:
[1] 1.570759 1.570840
. . .
```

```
> pi/2

[1] 1.570799

> optimize(my.fcn, c(0, pi), maximum=F, per = pi)

$minimum:
[1] 2.356196

$objective:
[1] -1

$nf:
[1] 9

$interval:
[1] 2.356155 2.356236
. . .

> 3*pi/4

[1] 2.356194
```

See the help file for `optimize` for information on other arguments to this function.

Finding Maxima and Minima of Multivariate Functions

S-PLUS has two functions to find the local minimum of a multivariate function: `nlminb` (Nonlinear Minimization with Box Constraints) and `ms` (Minimize Sums).

The two required arguments to `nlminb` are `objective` (the function f to minimize) and `start` (a vector of starting values for the minimization). The function *f* must take as its first argument a vector of parameters over which the minimization is carried out. By default, there are no boundary constraints on the parameters. The `nlminb` function, however, also takes the optional arguments `lower` and `upper` that specify the bounds on the parameters. (Other arguments to *f* can be passed in the call to `nlminb`.)

1. Example: Using `nlminb` to find a local minimum

```
> my.multvar.fcn

function(xvec, ctr = rep(0, length(xvec)))
{
  if(length(xvec) != length(ctr))
     stop("lengths of xvec and ctr do not match")
  sum((xvec - ctr)^2)
}

> nlminb(start = c(0,0), objective = my.multvar.fcn,
+ ctr = c(1,2))

$parameters:
[1] 1 2

$objective:
[1] 3.019858e-30

$message:
[1] "ABSOLUTE FUNCTION CONVERGENCE"
. . .
```

To find a local maximum of *f*, use `nlminb` on *-f*. Since unary minus cannot be performed on a function, you must define a new function that returns -1 times the value of the function you want to maximize:

2. Example: Using `nlminb` to find a local maximum

```
> fcn.to.maximize

function(xvec)
{
 - xvec[1]^2 + 2 * xvec[1] - xvec[2]^2 + 20 * xvec[2] + 40
}


> fcn.to.minimize

function(xvec)
{
 - fcn.to.maximize(xvec)
}
```

```
> nlminb(start = c(0, 0), objective = fcn.to.minimize)

$parameters:
[1] 1 10

$objective:
[1] -141

$message:
[1] "RELATIVE FUNCTION CONVERGENCE"
. . .
```

See the help file for `nlminb` for information on other arguments to this function. To find the local minimum of a multivariate function subject to constraints, use `nlminb` with the `lower` and/or `upper` arguments.

3. Example: Using `nlminb` to find a constrained minimum

As an example of using `nlminb` to find a constrained minimum, consider the following function `norm.neg.2.ll`, which is (minus a constant) `-2` times the log-likelihood function of a normal (Gaussian) distribution:

```
> norm.neg.2.ll <-
+ function(theta, y)
+ {
+ length(y) * log(theta[2]) +
+ (1/theta[2]) * sum((y - theta[1])^2)
+ }
```

This function assumes that observations from a normal distribution are stored in the vector `y`. The vector `theta` contains the mean (`theta[1]`) and variance (`theta[2]`) of this distribution. To find the maximum likelihood estimates of the mean and variance, we need to find the values of `theta[1]` and `theta[2]` that minimize `norm.neg.2.ll` for a given set of observations stored in `y`. We must use the lower argument to `nlminb` because the estimate of variance (`theta[2]`) must be greater than zero:

```
> set.seed(12)
> my.obs <- rnorm(100, mean = 10, sd = 2)
```

```
> nlminb(start = c(0,1), objective = norm.neg.2.ll,
+ lower = c(-Inf, 0), y = my.obs)

$parameters:
[1] 9.863812 3.477773

$objective:
[1] 224.6392

$message:
[1] "RELATIVE FUNCTION CONVERGENCE"
. . .

> mean(my.obs)

[1] 9.863812

> (99/100) * var(my.obs)

[1] 3.477774
```

The Minimum Sums function `ms` also minimizes a multivariate function, but in the context of the modeling paradigm, so it expects a formula rather than a function as its main argument. Here is the last example redone with `ms` (`mu` is the estimate of the population mean μ, and `ss` is the estimate of the population variance σ^2):

4. Example: Using `ms`

```
> ms( ~length(y) * log(ss) + (1/ss) * sum((y - mu)^2),
+ data = data.frame(y = my.obs),
+ start = list(mu = 0, ss = 1))

value: 224.6392
parameters:
      mu       ss
9.863813 3.477776

formula:  ~length(y) * log(ss) + (1/ss) * sum((y-mu)^2)

1 observations
```

```
call: ms(formula = ~length(y) * log(ss) + (1/ss) *
        sum((y - mu)^2),
data = data.frame(y=my.obs), start=list(mu=0, ss=1))
```

Hint

The `ms` function does not do minimization subject to constraints on the parameters.

If there are multiple solutions to your minimization problem, you may *not* get the answer you want using `ms`. In the above example, the `ms` function tells us we have "1 observations" because the whole vector `y` was used at once in the formula. The Minimum Sum function minimizes the *sum* of contributions to the formula, so we could have gotten the same estimates `mu` and `ss` with the formula shown in example 5.

5. Example: Using `ms` with several observations

```
> ms( ~log(ss) + (1/ss) * (y - mu)^2,
+ data = data.frame(y = my.obs),
+ start = list(mu = 0, ss = 1))

value: 224.6392

parameters:
      mu       ss
9.863813 3.477776

formula:  ~log(ss) + (1/ss) * (y - mu)^2

100 observations
call: ms(formula =  ~log(ss) + (1/ss) * (y - mu)^2,
data = data.frame(y=my.obs), start=list(mu=0,ss=1))
```

If the function you are trying to minimize is fairly complicated, then it is usually easier to write a function to supply as the formula:

6. Example: Using `ms` with a formula function

```
> ms( ~norm.neg.2.ll(theta,y), data=data.frame(y=my.obs),
+ start = list(theta = c(0,1)))

value: 224.6392
```

```
parameters:
  theta1   theta2
9.863813 3.477776

formula:  ~norm.neg.2.ll(theta, y)

1 observations

call: ms(formula = ~norm.neg.2.ll(theta, y), data =
  data.frame(y = my.obs),
start = list(theta = c(0, 1)))
```

Solving Nonnegative Least Squares Problems

Given an $m \times n$ matrix A and a vector b of length m, the linear nonnegative least squares problem is to find the vector x of length n that minimizes $\|Ax - b\|$, subject to the constraint that $x_i \geq 0$ for i in 1, ..., n.

To solve nonnegative least squares problems in S-PLUS, use the `nnls.fit` function. For example, consider the following fit using the `stack` data:

```
$coefficients
  Air Flow Water Temp Acid Conc.
0.2858057  0.05715152          0

$residuals:
[1]  17.59245246 12.59245246 14.13578403
[4]   8.90840973 -0.97728723 -1.03443875
[7]  -0.09159027  0.90840973 -2.89121593
[10] -3.60545832 -3.60545832 -4.54830680
[13] -6.60545832 -5.66260984 -7.31901267
[16] -8.31901267 -7.37616419 -7.37616419
[19] -6.43331572 -2.14814995 -6.14942983

$dual:
    Air Flow   Water Temp Acid Conc.
3.637979e-12 5.400125e-13 -1438.359
$rkappa:
     final    minimum
0.02488167 0.02488167
```

```
$call:
nnls.fit(x = stack.x, y = stack.loss)
```

You can also use `nlregb` to solve the nonnegative least squares problem, since the nonnegativity constraint is just a simple box constraint. To pose the problem to `nlregb`, define two functions (`lin.res` and `lin.jac`) of the form `f(x,params)`, to represent the residual function and the Jacobian of the residual function, respectively:

```
> lin.res <- function(x, b, A) A%*% x - b
> lin.jac <- function(x, A) A
> nlregb(n = length(stack.loss), start = rnorm(3),
+ res = lin.res, jac = lin.jac, lower = 0,
+ A = stack.x, b = stack.loss)

$parameters:
[1] 0.28580571 0.05715152 0.00000000

$objective:
[1] 1196.252
. . .
```

Generally, `nnls.fit` should be preferred to `nlregb` for reasons of efficiency, since `nlregb` is primarily designed for nonlinear problems. However, `nlregb` can solve degenerate problems that can not be handled by `nnls.fit`. You may also want to compare the results of `nnls.fit` with those of `lm`. Remember that `lm` requires a formula, and also that it fits an intercept term by default (which `nnls.fit` does not). Keeping this in mind, you can construct the comparable call to `lm` as follows:

```
> lm(stack.loss ~ stack.x - 1)

Call:
lm(formula = stack.loss ~ stack.x - 1)
Coefficients:
stack.xAir Flow stack.xWater Temp
      0.7967652          1.111422 -0.6249933

Degrees of freedom: 21 total; 18 residual
Residual standard error: 4.063987
```

For the stack loss data, the results of the constrained optimization methods `nnls.fit` and `nlregb` agree completely. The linear model produced by `lm` includes a negative coefficient.

You can use `nnls.fit` to solve the weighted nonnegative least squares problem by providing a vector of weights as the `weights` argument. The weights used by `lm` are the square roots of the weights used by `nnls.fit`; you must keep this in mind if you are trying to solve a problem using both functions.

Solving Nonlinear Least Squares Problems

Two functions, `nls` and `nlregb`, are available for solving the special minimization problem of nonlinear least squares. The function `nls` is used in the context of the modeling paradigm, so it expects a formula rather than a function as its main argument. The function `nlregb` expects a function rather than a formula (the argument name is `residuals`), and, unlike `nls`, it can perform the minimization subject to constraints on the parameters.

1. Example: Using `nls`

In this example, we create 100 observations where the underlying signal is a sine function with an amplitude of 4 and a horizontal (phase) shift of π. Noise is added in the form of normal (Gaussian) random numbers. We then use the `nls` function to estimate the true values of amplitude and horizontal shift.

```
> set.seed(20)
> noise <- rnorm(100, sd = 0.5)
> x <- seq(0, 2*pi, length = 100)
> my.nl.obs <- 4 * sin(x - pi) + noise
> plot(x, my.nl.obs)
> nls(y ~ amp * sin(x - horshft),
+ data = data.frame(y = my.nl.obs, x = x),
+ start = list(amp = 1, horshft = 0))

Residual sum of squares : 20.25668
parameters:
       amp    horshft
 -4.112227 0.01059317
formula: y ~ amp * sin(x - horshft)
100 observations
```

The above example illustrates the importance of finding appropriate starting values. The `nls` function returns an estimate of `amp` close to `-4` and an estimate of `horshft` close to `0` because of the cyclical nature of the sine function: `sin(x - pi) = -sin(x)`. If we start with initial estimates of `amp` and `horshft` closer to their true values, `nls` gives us the estimates we want.

2. Example: Using `nls` with better starting values

```
> nls(y ~ amp * sin(x - horshft),
+ data = data.frame(y = my.nl.obs, x = x),
+ start = list(amp = 3, horshft = pi/2))

Residual sum of squares : 20.25668
parameters:
      amp horshft
 4.112227  -3.131
formula: y ~ amp * sin(x - horshft)
100 observations
```

We could use the `nlregb` function to redo the above example, and specify that the value of `amp` must be greater than 0:

3. Example: Creating `my.new.func` and using `nlregb`

```
> my.new.fcn

function(param, x, y)
{
  amp <- param[1]
  horshft <- param[2]
  y - amp * sin(x - horshft)
}

> nlregb(n = 100, start = c(3,pi/2),
+ residuals = my.new.fcn,
+ lower = c(0, -Inf), x = x, y = my.nl.obs)

$parameters:
[1] 4.112227 3.152186

$objective:
[1] 20.25668
```

```
$message:
[1] "BOTH X AND RELATIVE FUNCTION CONVERGENCE"

$grad.norm:
[1] 5.960581e-09
```

EXAMPLES OF NONLINEAR MODELS

Maximum Likelihood Estimation

Parameters are estimated by maximizing the likelihood function. Suppose n independent observations are distributed with probability densities $p_i(\theta) = p(y_i; \theta)$ where θ is a vector of parameters. The likelihood function is defined as:

$$L(y;\theta) = \prod_{i=1}^{n} p_i(\theta) \tag{14.1}$$

The problem is to find the estimate $\tilde{\theta}$ of that maximizes the likelihood function for the observed data. Maximizing the likelihood is equivalent to minimizing the negative of the log-likelihood:

$$l(\theta) = -\log(L(y;\theta)) = \sum_{i=1}^{n} -\log(p_i(\theta)) \tag{14.2}$$

Example One: Ping-Pong

Each member of the U.S. Table Tennis Association is assigned a rating based on the member's performance in tournaments. Winning a match boosts the winner's rating and lowers the loser's rating some number of points depending on the current ratings of the two players. Using this data, two questions we might like to ask are:

1. Do players with a higher rating tend to win over players with a lower rating?
2. Does a larger difference in rating imply that the higher-rated player is more likely to win?

Assuming a logistic distribution in which `log(p/(1-p))` is proportional to the difference in rating between the winner and loser and the average rating of the two players:

$$p_i = \frac{e^{D_i\alpha + R_i\beta}}{1 + e^{D_i\alpha + R_i\beta}} \tag{14.3}$$

where $D_i = W_i - L_i$ is the difference in rating between the winner and loser and $R_i = 1/2(W_i + L_i)$ is the average rating for the two players.

To fit the model, we need to find α and β which minimize the negative log-likelihood:

$$\sum -\log(p_i) = \sum \left\{ -D_i\alpha - R_i\beta + \log(1 + e^{D_i\alpha + R_i\beta}) \right\} \tag{14.4}$$

Example Two: Wave-Soldering Skips

In a 1988 AT&T wave-soldering experiment, several factors were varied:

Factor	Description
`opening`	amount of clearance around the mounting pad
`solder`	amount of solder
`mask`	type and thickness of the material used for the solder mask
`padtype`	the geometry and size of the mounting pad
`panel`	each board was divided into three panels, with three runs on a board

The results of the experiment gave the number of visible soldering skips (faults) on a board. Physical theory and intuition suggest a model in which the process is in one of two states:

1. A “perfect” state where no defects occur

2. An "imperfect" state where there may or may not be defects

Both the probability of being in the imperfect state and the distribution of skips in that state depend on the factors in the experiment. Assume that some "stress," S, induces the process to be in the imperfect state and also increases the tendency to generate skips when in the imperfect state.

Assume S depends linearly on the levels of the factors, x_j, $j = 1, \ldots, p$:

$$S_i = \sum_{j=1}^{p} x_{ij}\beta_j \tag{14.5}$$

where β is the vector of parameters to be estimated.

Assume the probability P_i of being in the imperfect state is monotonically related to the stress by a logistic distribution:

$$P_i = \frac{1}{1 + e^{(-\tau)S_i}} \tag{14.6}$$

As the stress increases, the above function approaches 1.

Given that the process is in an imperfect state, assume the probability of k_i skips is modeled by the Poisson distribution with mean λ_i:

$$P(k_i) = e^{-\lambda_i} \cdot \frac{\lambda_i^{k_i}}{k_i!} \tag{14.7}$$

The probability of zero skips is the probability of being in the perfect state plus the probability of being in the imperfect state and having zero skips. The probability of one or more skips is the probability of being in the imperfect state and having one or more skips. Mathematically the probabilities may be written as:

$$P(y = y_i) = \begin{cases} \dfrac{e^{(-\tau)S_i}}{1 + e^{(-\tau)S_i}} + \dfrac{e^{-\lambda_i}}{1 + e^{(-\tau)S_i}} & \text{if } y_i = 0 \\ \dfrac{1}{1 + e^{(-\tau)S_i}} e^{-\lambda_i} \dfrac{\lambda_i^{y_i}}{y_i!} & \text{if } y_i > 0 \end{cases} \tag{14.8}$$

The mean skips in the imperfect state is always positive and modeled in terms of the stress by: $\lambda_i = e^{S_i}$. The parameters, τ and β, can be estimated by minimizing the negative log-likelihood. The ith element of the negative log-likelihood can be written to within constants as:

$$_i(\beta, \tau) = \log(1 + e^{(-\tau)S_i}) - \begin{cases} \log\left(e^{(-\tau)S_i} + e^{-e^{S_i}}\right) & \text{if } y_i = 0 \\ y_i S_i - e^{S_i} & \text{if } y_i > 0 \end{cases} \tag{14.9}$$

The model depicted above does not reduce to any simple linear model.

Nonlinear Regression

Parameters are estimated by minimizing the sum of squared residuals. Suppose n independent observations y can be modeled as a nonlinear parametric function f of a vector x of predictor variables and a vector of parameters, β.

$$y = f(x;\beta) + \varepsilon$$

where the errors, ε, are assumed normally distributed. The nonlinear least-squares problem finds parameter estimates $\tilde{\beta}$ that minimize:

$$\sum_{i=1}^{n} (y_i - f(x;\beta))^2 \tag{14.10}$$

Example Three: Puromycin

A biochemical experiment measured reaction velocity in cells with and without treatment by Puromycin. There are three variables in the `Puromycin` data frame.

Variable	Description
`conc`	the substrate concentration
`vel`	the reaction velocity
`state`	indicator of treated or untreated

Assume a Michaelis-Menten relationship between velocity and concentration:

$$V = \frac{V_{max} c}{K + c} + \varepsilon \tag{14.11}$$

where V is the velocity, c is the enzyme concentration, V_{max} is a parameter representing the asymptotic velocity as $c -> \infty$, K is the Michaelis parameter, and ε is experimental error. Assuming the treatment with the drug would change V_{max} but not K, the optimization function is:

$$S(V_{max}, K) = \sum \left(V_i - \frac{(V_{max} + \Delta V_{max} I_{\{treated\}}(state)) c_i}{K + c_i} \right)^2 \tag{14.12}$$

where $I_{\{treated\}}$ is the function indicating if the cell was treated with Puromycin.

INFERENCE FOR NONLINEAR MODELS

Likelihood Models

With likelihood models distributional results are asymptotic. Maximum likelihood estimates tend toward a normal distribution with a mean equal to the true parameter, and a variance matrix given by the inverse of the information matrix, the negative of the second derivatives of the log-likelihood.

Least Squares Models

In least-squares models approximations to quantities such as standard errors or correlations of parameter estimates are used. The approximation proceeds as follows:

1. Replace the nonlinear model with its linear Taylor series approximation at the parameter estimates.
2. Use the methods for *linear* statistical inference on the approximation.

Consequently, the nonlinear inference results are called linear approximation results.

The Fitting Algorithms

Minimum-sum algorithm

This section deals with the general optimization of an objective function modeled as a sum. The algorithm is a version of Newton's method based on a quadratic approximation of the objective function. If both first and second derivatives are supplied, the approximation is a local one using the derivatives. If no derivatives or only the first derivative are supplied, the algorithm approximates the second derivative information. It does this in a way specifically designed for minimization.

The algorithm actually used is taken from the PORT subroutine library which evolved from the published algorithm by Gay (1983). Two key features of this algorithm are:

1. A quasi-Newton approximation for second derivatives.
2. A "trust region" approach controlling the size of the region in which the quadratic approximation is believed to be accurate.

The algorithm is capable of working with user models specifying 0, 1, or 2 orders of derivatives.

Nonlinear least-squares algorithm

The Gauss-Newton algorithm is used with a step factor to ensure that the sum of squares decreases at each iteration. A line-search method is used, as opposed to the trust region employed in the minimum-sum algorithm. The step direction is determined by a quadratic model. The algorithm proceeds as follows:

1. The residuals are calculated, and the gradient is calculated or approximated (depending on the data), at the current parameter values.
2. A linear least-squares fit of the residual on the gradient gives the parameter increment.
3. If applying the full parameter increment increases the sum-of-squares rather than decreasing it, the length of the increment is successively halved until the sum-of-squares is decreased.
4. The step factor is retained between iterations and started at min{2*(*previous step factor*), 1}.

If the gradient is not specified analytically, it is calculated using finite differences with forward differencing. For partially linear models, the increment is calculated using the Golub-Pereyra method (Golub and Pereyra, 1973) as implemented by Bates and Lindstrom (1986).

Specifying Models

Nonlinear models typically require specifying more details than models of other types. The information typically required to fit a nonlinear model, using the S-PLUS functions `ms` or `nls`, is:

1. A formula
2. Data
3. Starting values

Formulas

For nonlinear models a formula is an S-PLUS expression involving data, parameters in the model, and any other relevant quantities. The parameters must be specified in the formula because there is no assumption about where they are to be placed (as in linear models, for example). Formulas are typically specified differently depending on whether you have a minimum-sum problem or nonlinear least-squares problem.

In the puromycin example, you would specify a formula for the simple model (described in Equation (14.11)) by:

```
vel ~ Vm*conc / (K + conc)
```

The parameters `Vm` and `K` are specified along with the data `vel` and `conc`. Since there is no explicit response for minimum-sum models (for example, likelihood models), it is left off in the formula.

In the ping-pong example (ignoring the average rating effect), the formula for Equation (14.4) is:

```
~- DV * alpha + log( 1 + exp( DV * alpha ) )
```

where `DV` is a variable in the data and `alpha` is the parameter to fit. Note that the model here is based only on the difference in ratings, ignoring for the moment the average rating.

Simplifying Formulas

Some models can be organized as a simple expression involving one or more S-PLUS functions that do all the work. Note that `DV*alpha` occurs twice in the formula for the ping-pong model. You can write a general function for the log-likelihood in terms of `DV*alpha`.

```
> lprob <- function(lp) log(1 + exp(lp)) - lp
```

Recall that `lp` is the linear predictor for the GLM. A simpler expression for the model is now:

```
~ lprob( DV * alpha )
```

Having `lprob` now makes it easy to add additional terms or parameters.

Implications of the Formulas

For nonlinear least-squares formulas the response on the left of ~ and the predictor on the right must evaluate to numeric vectors of the same length. The fitting algorithm tries to estimate parameters to minimize the sum of squared differences between response and prediction. If the response is left out the formula is interpreted as a residual vector.

For Minimum-Sum formula, the right of ~ must evaluate to a numeric vector. The fitting algorithm tries to estimate parameters to minimize the sum of this "predictor" vector. The concept here is linked to maximum-likelihood models. The computational form does not depend on an MLE concept. The elements of the vector may be anything and there need not be more than one.

The evaluated formulas can include derivatives with respect to the parameters. The derivatives are supplied as attributes to the vector that results when the predictor side of the formula is evaluated. When explicit derivatives are not supplied, the algorithms use numeric approximations.

Parametrized Data Frames

Relevant data for nonlinear modeling includes:

- Variables
- Initial estimates of parameters
- Fixed values occurring in a model formula

Parametrized data frames allow you to "attach" relevant data to a data frame when the data doesn't occupy an entire column. Information is attached as a `"parameter"` attribute of the data frame. The `parameter` function returns or modifies the entire list of parameters and is analogous to the `attributes` function. Similarly the `param` function returns or modifies one parameter at a time and is analogous to the `attr` function. You could supply values for `Vm` and `K` to the `Puromycin` data frame with:

```
> parameters(Puromycin) <- list(Vm = 200, K = 0.1)
```

The parameter values can be retrieved with

```
> parameters(Puromycin)

$Vm:
[1] 200

$K:
[1] 0.1
```

The class of `Puromycin` is now:

```
> class(Puromycin)

[1] "pframe" "data.frame"
```

Now, when `Puromycin` is attached, the parameters `Vm` and `K` are available when referred to in formulas.

Starting Values; Identifying Parameters

Before the formulas can be evaluated, the fitting functions must know which names in the formula are parameters to be estimated and must have starting values for these parameters. The fitting functions determine this in the following way:

1. If the `start` argument is supplied, its names are the names of the parameters to be estimated, and its values are the corresponding starting values.
2. If `start` is missing, the parameters attribute of the `data` argument defines the parameter names and values.

Hint

Explicitly use the `start` argument to name and initialize parameters.

You can easily see what the starting values are in the `call` component of the fit and you can arrange to keep particular parameters constant when that makes sense.

Derivatives

Supplying derivatives of the predictor side of the formula with respect to the parameters along with the formula can reduce the number of iterations (so speed up the computations), increase numerical accuracy, and improve the chance of convergence. In general derivatives should be used whenever possible.

The fitting algorithms can use both first (the gradient) and second derivatives (the Hessian). The derivatives are supplied to the fitting functions as attributes to the formula. Recall that evaluating the formula gives a vector of n values. Evaluating the first derivative expression should give n values for each of the p parameters, that is an $n \times p$ matrix. Evaluating the second derivative expression should give n values for each of the $p \times p$ partial derivatives, that is, an $n \times p \times p$ array.

First Derivatives

The negative log-likelihood for the simple ping-pong model is:

$$l(\alpha) = \sum[\log(1 + e^{D_i\alpha}) - D_i\alpha] \tag{14.13}$$

Differentiating with respect to α and simplifying gives the gradient:

$$\frac{\partial l}{\partial \alpha} = \sum \left[\frac{-D_i}{(1 + e^{D_i \alpha})} \right] \tag{14.14}$$

The gradient is supplied to the fitting function as the `"gradient"` attribute of the formula:

```
> form.pp <-  ~log(1 + exp( DV*alpha ) ) - DV*alpha
> attr(form.pp, "gradient") <-
+ ~ -DV / ( 1 + exp( DV*alpha ) )
> form.pp

 ~ log(1 + exp(DV * alpha)) - DV * alpha

> attr(form.pp,"gradient")

 ~ - DV/(1 + exp(DV * alpha))
```

When a function is used to simplify a formula, build the gradient into the function. The `lprob` function is used to simplify the formula expression to `~lprob(DV*alpha)`.

```
> lprob

function(lp)
log(1 + exp(lp)) - lp
```

An improved version of `lprob` adds the gradient.

```
> lprob2

function(lp, X)
{
  elp <- exp(lp)
  z <- 1 + elp
  value <- log(z) - lp
  attr(value, "gradient") <- -X/z
  value
}
```

Note `lp` is again the linear predictor and `X` is the data in the linear predictor. With the gradient built into the function, you don't need to add it as an attribute to the formula; it is already an attribute to the object hence used in the formula.

Second Derivatives

The second derivatives may be added as the `"hessian"` attribute of the formula. In the ping-pong example, the second derivative of the negative log-likelihood with respect to α is:

$$\frac{\partial^2 l}{\partial \alpha^2} = \sum \frac{D_i^2 e^{D_i \alpha}}{(1 + e^{D_i \alpha})^2} \tag{14.15}$$

The `lprob2` function is now modified to add the Hessian as follows. The Hessian is added in a general enough form to allow for multiple predictors.

```
> lprob3

function(lp, X)
{
  elp <- exp(lp)
  z <- 1 + elp
  value <- log(z) - lp
  attr(value, "gradient") <- -X/z
  if(length(dx <- dim(X)) == 2)
  {
     n <- dx[1]; p <- dx[2]
  } else
  {
     n <- length(X); p <- 1
  }
  xx <- array(X, c(n, p, p))
  attr(value, "hessian") <- (xx * aperm(xx, c(1, 3, 2)) *
        elp)/z^2
  value
}
```

Interesting points of the added code are:

- The second derivative computations are performed at the time of the assignment of the `"hessian"` attribute.

- The rest of the code (starting with `if(length(...))`) is to make the Hessian general enough for multiple predictors.
- The `aperm` function does the equivalent of a transpose on the second and third dimensions to produce the proper cross products when multiple predictors are in the model.

Symbolic Differentiation

A symbolic differentiation function, `D`, is available to aid in taking derivatives.

Table 14.2: *Arguments to* `D`.

Argument	Purpose
`expr`	Expression to be differentiated.
`name`	Which parameters to differentiate with respect to.

The function `D` is used primarily as a support routine to `deriv`.

Again referring to the ping-pong example, `form` contains the expression of the negative log-likelihood:

```
> form

expression(log((1 + exp(DV * alpha))) - DV * alpha)
```

The first derivative is computed as:

```
> D(form, "alpha")

(exp(DV * alpha) * DV)/(1 + exp(DV * alpha)) - DV
```

And the second derivative is computed as:

```
> D( D(form, "alpha"), "alpha")

(exp(DV * alpha) * DV * DV)/(1 + exp(DV * alpha))
- (exp(DV * alpha) * DV * (exp(DV * alpha) * DV))
/(1 + exp(DV * alpha))^2
```

Improved Derivatives

The `deriv` function takes an expression, computes a derivative, simplifies the result then returns an expression or function for computing the original expression along with its derivative(s).

Table 14.3: *Arguments to* `deriv`

Argument	Purpose
`expr`	Expression to be differentiated, typically a formula, in which case the expression returned computes the right side of the ~ *and* its derivatives.
`namevec`	Character vector of names of parameters.
`function.arg`	Optional argument vector or prototype for a function.
`tag`	Base of the names to be given to intermediate results. Default is `".expr"`.

Periods are used in front of created object names to avoid conflict with user chosen names. The `deriv` function returns an expression in the form expected for nonlinear models.

```
> deriv(form,"alpha")

expression(
{
  .expr1 <- DV * alpha
  .expr2 <- exp(.expr1)
  .expr3 <- 1 + .expr2
  .value <- (log(.expr3)) - .expr1
  .grad <- array(0, c(length(.value), 1), list(NULL,
"alpha"))
  .grad[, "alpha"] <- ((.expr2 * DV)/.expr3) - DV
  attr(.value, "gradient") <- .grad
  .value
})
```

If the `function.arg` argument is supplied, a function is returned.

```
> deriv(form,"alpha",c("DV","alpha"))
```

```
function(DV, alpha)
{ .expr1 <- DV * alpha
  .expr2 <- exp(.expr1)
  .expr3 <- 1 + .expr2
  .value <- (log(.expr3)) - .expr1
  .grad <- array(0, c(length(.value), 1), list(NULL,
"alpha"))
  .grad[, "alpha"] <- ((.expr2 * DV)/.expr3) - DV
  attr(.value, "gradient") <- .grad
  .value
}
```

The `namevec` argument can be a vector.

```
> deriv(vel ~ Vm * (conc/(K + conc)), c("Vm","K"))

expression(
{ .expr1 <- K + conc
  .expr2 <- conc/.expr1
  .value <- Vm * .expr2
  .grad <- array(0, c(length(.value), 2), list(NULL,
c("Vm","K")))
  .grad[, "Vm"] <- .expr2
  .grad[, "K"] <- - (Vm * (conc/(.expr1^2)))
  attr(.value, "gradient") <- .grad
  .value
})
```

The symbolic differentiation interprets each parameter as a scalar. Generalization from scalar to vector parameters (for example, `lprob2`) must be done by hand. Use parentheses to help `deriv` find relevant subexpressions. Without the redundant parentheses around `conc/(K + conc)` the expression `deriv` returns is not as simple as possible.

Fitting Models

There are two different fitting functions for nonlinear models:

- `ms` minimizes the sum of the vector supplied as the right side of the formula.
- `nls` minimizes the sum of squared differences between the left and right sides of the formula.

Table 14.4: *Arguments to* `ms`.

Argument	Purpose
`formula`	The nonlinear model formula (without a left side).
`data`	A data frame in which to do the computations.
`start`	Numeric vector of initial parameter values for the iteration.
`scale`	Parameter scaling.
`control`	List of control values to be used in the iteration.
`trace`	Indicates whether or not intermediate estimates should be printed.

Table 14.5: *Arguments to* `nls`.

Argument	Purpose
`formula`	The nonlinear regression model as a formula.
`data`	A data frame in which to do the computations.
`start`	Numeric vector of initial parameter values for the iteration.
`control`	List of control values to be used in the iteration.
`algorithm`	Which algorithm to use. The default algorithm is a Gauss-Newton algorithm. If `algorithm = "plinear"`, the Golub-Pereyra algorithm for partially linear least-squares models is used.

Table 14.5: *Arguments to* `nls`. *(Continued)*

Argument	Purpose
`trace`	Indicates whether or not intermediate estimates should be printed.

Fitting a Model to the Puromycin Data

Before fitting a model, take a look at the data displayed in Figure 14.1.

```
> attach(Puromycin)
> plot(conc,vel,type="n")
> text(conc,vel,ifelse(state == "treated", "T","U"))
```

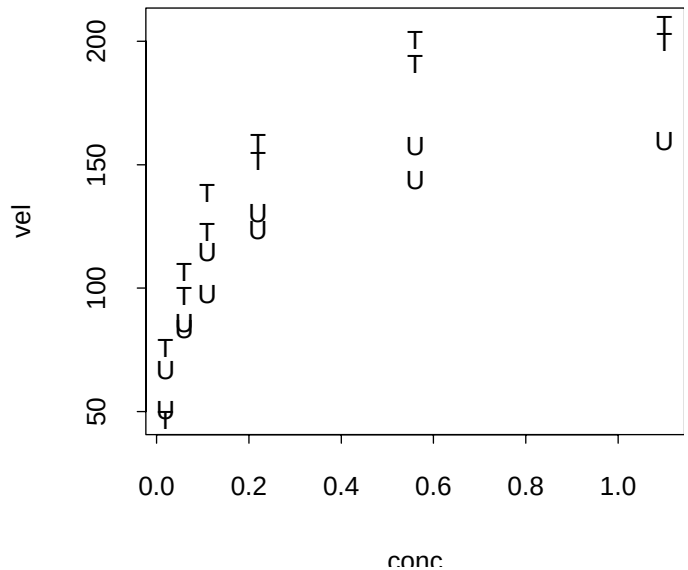

Figure 14.1: `vel` *versus* `conc` *for treated (T) and untreated (U) groups.*

1. Estimating starting values

Obtain an estimate of V_{max} for each group as the maximum value each group attains.

- The treated group has a maximum of about 200.
- The untreated group has a maximum of about 160.

The value of K is the concentration at which V reaches $V_{max}/2$, roughly 0.1 for each group.

2. A simple model

Start by fitting a simple model for the treated group only.

```
> Treated <- Puromycin[Puromycin$state == "treated",]
> Purfit.1 <- nls(vel ~ Vm*conc/(K + conc), data = Treated,
+ start = list(Vm = 200, K = 0.1))
> Purfit.1

residual sum of squares: 1195.449

parameters:
      Vm          K
212.6836 0.06412111

12 observations

gradient.norm:
[1] 0.2781043

RELATIVE FUNCTION CONVERGENCE

formula: vel~(Vm * conc)/(K + conc)
```

Fit a model for the untreated group similarly but with `Vm = 160`.

```
> Purfit.2

residual sum of squares: 859.6043
parameters:
    Vm          K
160.28 0.04770808

11 observations

gradient.norm:
[1] 0.1389438

RELATIVE FUNCTION CONVERGENCE

formula: vel ~ (Vm * conc)/(K + conc)
```

3. A more complicated model

Obtain summaries of the fits with the `summary` function:

```
> summary(Purfit.1)

parameters:
          Value   Std.Error   t value
Vm 212.68362994 6.947148850 30.614520
 K   0.06412111 0.008280931  7.743225
. . .
sigma: 10.93366
df:
[1] 2 10
. . .
correlation:
          Vm         K
Vm 1.0000000 0.7650835
 K 0.7650835 1.0000000

formula:
vel ~ (Vm * conc)/(K + conc)

> summary(Purfit.2)

parameters:
          Value   Std.Error   t value
Vm 160.27997949 6.480240801 24.733646
 K   0.04770808 0.007781862  6.130677
. . .
sigma: 9.773003
df:
[1] 2 9
. . .
correlation:
          Vm         K
Vm 1.0000000 0.7768269
 K 0.7768269 1.0000000

formula:
vel ~ (Vm * conc)/(K + conc)
```

An approximate t-test for the difference in K between the two models suggests there is no difference:

```
> (.06412 - .04771)/sqrt(.00828^2 + .007782^2)

[1] 1.44416
```

The correct test of whether the Ks should be different, and is as follows:

```
> Purboth <- nls(vel ~ (Vm + delV*(state=="treated"))*conc/
+ (K + conc), data=Puromycin,
+ start=list(Vm=160, delV=40, K=0.05))
> summary(Purboth)

parameters:
            Value   Std.Error   t value
  Vm 166.60396617 5.807422147 28.688110
delV  42.02590886 6.272136003  6.700414
   K   0.05797157 0.005910154  9.808809
. . .
sigma: 10.58511

df:
[1] 3 20
. . .
             Vm        delV          K
  Vm  1.0000000 -0.54055820 0.61128147
delV -0.5405582  1.00000000 0.06440645
   K  0.6112815  0.06440645 1.00000000

formula:
vel  ~ ((Vm + delV * (state == "treated")) * conc)/(K +
conc)

> combinedSS <- sum(Purfit.1$res^2) + sum(Purfit.2$res^2)
> Fval <- (sum(Purboth$res^2) - combinedSS)/(combinedSS/19)
> Fval

[1] 1.718169

> 1 - pf(Fval, 1, 19)

[1] 0.2055523
```

Using a single K appears to be reasonable.

Fitting a Model to the Ping-Pong Data

The example here develops a model based only on the difference in ratings, ignoring, for the moment, the average rating. The model to fit is:

```
~ DV * alpha + log( 1 + exp(DV * alpha) )
```

There are four stages to the development of the model.

1. Estimating starting values

A very crude initial estimate for `alpha` can be found as follows:

- Replace all the differences in ratings by $\pm\bar{d}$, where $\bar{d}$ is the mean difference.
- For each match, the probability from the model that the winner had a higher rating satisfies:

$$\bar{d} * a = \log(p/(1-p))$$

- Solve for α by substituting for p the observed frequency with which the player with the higher rating wins.

The computations in S-PLUS proceed as follows:

```
> param(pingpong, "p") <- 0 # make pingpong a "pframe"
> attach(pingpong,1)
> DV <- winner - loser
> p <- sum(winner > loser) /length(winner)
> p

[1] 0.8223401

> alpha <- log(p/(1-p))/mean(DV)
> alpha

[1] 0.007660995

> detach(1, save = "pingpong")
```

2. A simple model

Recall the `lprob` function which calculates the log-likelihood for the ping-pong problem.

```
> lprob

function(lp)
log(1 + exp(lp)) - lp
```

The model is fitted as follows:

```
> attach(pingpong)
> fit.alpha <- ms( ~ lprob( DV * alpha ),
+ start = list(alpha=0.0077))
> fit.alpha

objective: 1127.635

parameters:
     alpha
0.01114251

gradient:
[1] -0.0004497159

RELATIVE FUNCTION CONVERGENCE.
formula:  ~ lprob(DV * alpha)
```

3. Adding the gradient

To fit the model with the gradient added to the formula use `lprob2`.

```
> fit.alpha.2 <- ms( ~ lprob2( DV * alpha, DV),
+ start = list(alpha=0.0077))
> fit.alpha.2

objective: 1127.635

parameters:
     alpha
0.01114251
```

```
gradient:
       alpha
2.938858e-07

RELATIVE FUNCTION CONVERGENCE.
formula:  ~ lprob2(DV * alpha, DV)
```

Even for this simple problem, providing the derivative has decreased the computation time by 20%.

4. Adding the Hessian

To fit the model with the gradient and the Hessian added to the formula use `lprob3`.

```
> fit.alpha.3 <- ms( ~ lprob3(DV*alpha, DV),
+ pingpong, start = list(alpha = .0077))
> fit.alpha.3

objective: 1127.635

parameters:
     alpha
0.01114251

gradient:
       alpha
-0.000218718

BOTH X- AND RELATIVE FUNCTION CONVERGENCE
formula:  ~ lprob3(DV * alpha, DV)
```

Profiling the Objective Function

Profiling provides a more accurate picture of the uncertainty in the parameter estimates than simple standard errors. When there are only two parameters, contours of the objective function can be plotted by generating a grid of values. When there are more than two parameters, examination of the objective function is usually done in one of two ways:

- *slices*: Fixing all but two of the parameters at their *estimated* values and creating a grid of the objective function by varying the remaining two parameters of interest.

- *projections*: Vary two parameters of interest over fixed values, optimizing the objective function over the other parameters.

Two-dimensional projections are often too time consuming to compute. One-dimensional projections are called profiles. Profiles are plots of a t-statistic equivalent called the profile t function for a parameter of interest against a range of values for the parameter.

The Profile t Function

For `nls`, the profile t function for a given parameter, θ_p, is denoted by $\tau(\theta_p)$ and is computed as follows:

$$\tau(\theta_p) = sign(\theta_p - \tilde{\theta}_p)\sqrt{\frac{\tilde{S}(\theta_p) - S(\tilde{\theta})}{s}} \tag{14.16}$$

where $\tilde{\theta}_p$ is the model estimate of θ_p, $\tilde{S}(\theta_p)$ is the sum of squares based on optimizing all parameters except θ_p, which is fixed, and $S(\tilde{\theta})$ is the sum of squares based on optimizing all parameters.

The profile t function is directly related to confidence intervals for the corresponding parameter. $\tau(\theta_p)$ can be shown to be equivalent to the studentized parameter

$$\delta(\theta_p) = \frac{\theta_p - \tilde{\theta}_p}{se(\tilde{\theta}_p)} \tag{14.17}$$

for which a 1 - α confidence interval can be constructed as follows:

$$-t\left(N-P;\frac{a}{2}\right) \leq \delta(\theta_p) \leq t\left(N-P;\frac{a}{2}\right) \tag{14.18}$$

The profile Function in S-PLUS

The profile function produces profiles for "`nls`" and "`ms`" objects. Profiles show confidence intervals for parameters as well as the nonlinearity of the objective function. If the model were linear the

profile would be a straight line through the origin with a slope of one. You can produce the profile plots for the Puromycin fit `Purboth` as follows:

```
> Purboth.prof <- profile(Purboth)
> plot(Purboth.prof)
```

The `"profile"` object returned by `profile` has a component for each parameter containing the evaluations of the profile t function plus some additional attributes. The component for the `Vm` parameter is:

```
> Purboth.prof$Vm
```

	tau	par.vals.Vm	par.vals.delV	par.vals.K
1	-3.9021051	144.6497	54.60190	0.04501306
2	-3.1186052	148.8994	52.07216	0.04725929
3	-2.3346358	153.2273	49.54358	0.04967189
4	-1.5501820	157.6376	47.01846	0.05226722
5	-0.7654516	162.1334	44.50315	0.05506789
6	0.0000000	166.6040	42.02591	0.05797157
7	0.7548910	171.0998	39.57446	0.06103225
8	1.5094670	175.6845	37.12565	0.06431820
9	2.2635410	180.3616	34.67194	0.06783693
10	3.0171065	185.1362	32.20981	0.07160305
11	3.7701349	190.0136	29.73812	0.07563630
12	4.5225948	194.9997	27.25599	0.07995897

Figure 14.2 shows profile plots for the three-parameter Puromycin fit. Each plot shows the profile t function, (τ), when the parameter on the x-axis ranges over the values shown, and the other parameters are optimized. The surface is quite linear with respect to these three parameters.

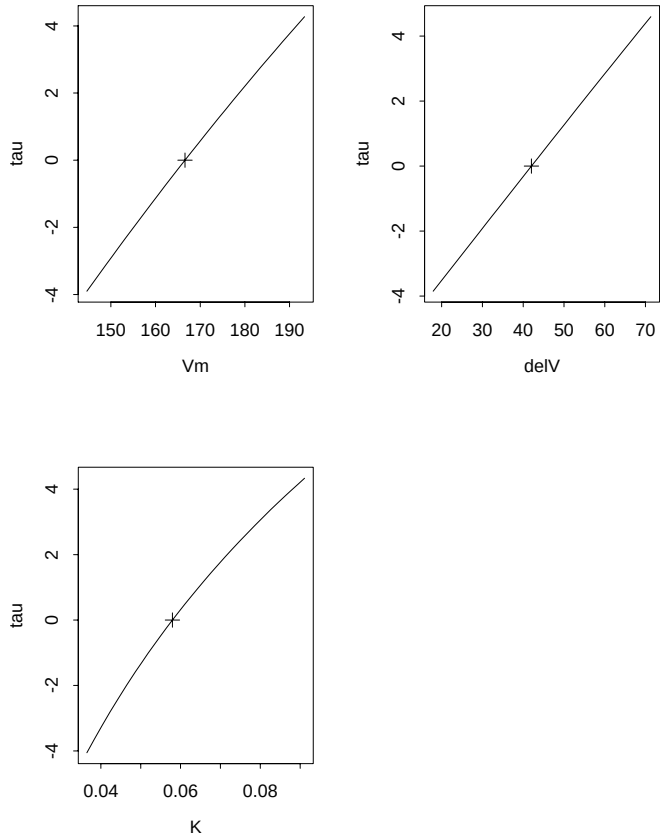

Figure 14.2: *The profile plots for the Puromycin fit.*

Computing Confidence Intervals

An example of a simple function to compute the confidence intervals from the output of `profile` follows:

```
> conf.int <- function(profile.obj, variable.name,
+ confidence.level = 0.95)
+ {if(is.na(match(variable.name, names(profile.obj))))
+ stop(paste("Variable", variable.name,
+ "not in the model"))
+ resid.df <- attr(profile.obj, "summary")[["df"]][2]
+ tstat <- qt(1 - (1 - confidence.level)/2, resid.df)
+ prof <- profile.obj[[variable.name]]
+ approx(prof[, "tau"], prof[, "par.vals"]
```

```
+ [, variable.name],
+ c(-tstat, tstat))[[2]]
+ }
```

The tricky line in `conf.int` is the last one which calls `approx`. `Purboth.prof$Vm` is a data frame with two components (columns). The first component is the vector of τ values which we pick off using `prof[, "tau"]`. The second component named `par.vals` contains a matrix with as many columns as there are parameters in the model. This results in the strange looking subscripting given by `prof[, "par.vals"][, variable.name]`. The first subscript removes the matrix from the `par.vals` component, and the second subscript removes the appropriate column. Three examples using `conf.int` and the profile object `Purboth.prof` follow:

```
> conf.int(Purboth.prof, "delV", conf = .99)

[1] 24.20945 60.03857

> conf.int(Purboth.prof, "Vm", conf = .99)

[1] 150.4079 184.0479

> conf.int(Purboth.prof, "K", conf = .99)

[1] 0.04217613 0.07826822
```

The `conf.int` function can be improved by, for example, doing a cubic spline interpolation rather than the linear interpolation that `approx` does. A marginal confidence interval computed from the profile t function is exact, disregarding any approximations due to interpolation, whereas the marginal confidence intervals produced by using the coefficient and its standard error from the summary of the fit is only a linear approximation.

DESIGNED EXPERIMENTS AND ANALYSIS OF VARIANCE 15

INTRODUCTION

This chapter discusses how to analyze designed experiments. Typically, the data have a numeric response and one or more categorical variables (*factors*) that are under the control of the experimenter. For example, an engineer may measure the yield of some process using each combination of four catalysts and three specific temperatures. This experiment has two factors, catalyst and temperature, and the response is the yield.

Traditionally, the analysis of experiments has centered on the performance of an Analysis of Variance (ANOVA). In more recent years graphics have played an increasingly important role. There is a large literature on the design and analysis of experiments–Box, Hunter, and Hunter (1978) is an example.

This chapter consists of sections which show you how to use S-PLUS to analyze experimental data for each of the following situations:

- Experiments with one factor
- Experiments with two factors and a single replicate
- Experiments with two factors and two or more replicates
- Experiments with many factors at two levels: 2^k designs

Each of these sections stands alone. You can read whichever section is appropriate to your problem, and get the analysis done without having to read the other sections. The examples used in this chapter are from Box, Hunter, and Hunter (1978) and thus is a useful supplement in a course which covers the material of Chapters 6, 7, 9, 10, and 11 of Box, Hunter, and Hunter.

Setting Up the Data Frame

In analyzing experimental data using S-PLUS, the first thing you do is set up an appropriate *data frame* for your experimental data. You may think of the data frame as a matrix, with the columns containing values of the *variables*. Each row of the data frame contains an observed value of the response (or responses), and the corresponding values of the experimental factors.

A First Look at the Data

Use the functions `plot.design`, `plot.factor`, and possibly `interaction.plot` to graphically explore your data.

The Model and Analysis of Variance

It is important that you have a clear understanding of exactly what model is being considered when you carry out the analysis of variance. Use `aov` to carry out the analysis of variance, and use `summary` to display the results.

In using `aov`, you use *formulas* to specify your *model*. The examples in this chapter introduce you to simple uses of formulas. You may supplement your understanding of how to use formulas in S-PLUS by reading Chapter 2, Specifying Models in S-PLUS, or Chapter 2, Statistical Models, and Chapter 5, Analysis of Variance; Designed Experiments, in Chambers and Hastie (1992).

Diagnostic Plots

For each analysis, you should make the following minimal set of plots to convince yourself that the model being entertained is adequate:

- Histogram of residuals (using `hist`)
- Normal qq-plot of residuals (using `qqnorm`)
- Plot of residuals versus fit (using `plot`)

When you know the time order of the observations, you should also make plots of the original data and the residuals in the time order in which the data were collected.

The diagnostic plots may indicate inadequacies in the model from one or more of the following sources: existence of interactions, existence of outliers, and existence of inhomogeneous error variance.

EXPERIMENTS WITH ONE FACTOR

The simplest kind of experiments are those in which a single continuous *response* variable is measured a number of times for each of several *levels* of some experimental *factor*.

For example, consider the data in Table 15.1 (from Box, Hunter, and Hunter (1978)), which consists of numerical values of "blood coagulation times" for each of four diets. Coagulation time is the continuous response variable, and diet is a *qualitative* variable, or *factor*, having four levels: A, B, C, and D. The diets corresponding to the levels A, B, C, and D were determined by the experimenter.

Table 15.1: *Blood coagulation times for four diets.*

Diet			
A	**B**	**C**	**D**
62	63	68	56
60	67	66	62
63	71	71	60
59	64	67	61
	65	68	63
	66	68	64
			63
			59

Your main interest is to see whether or not the factor "diet" has any effect on the mean value of blood coagulation time. The experimental factor, "diet" in this case, is often called the *treatment*.

Formal statistical testing for whether or not the factor level affects the mean is carried out using the method of analysis of variance (ANOVA). This needs to be complemented by exploratory graphics to provide confirmation that the model assumptions are sufficiently correct to validate the formal ANOVA conclusion. S-PLUS provides tools for you to do both the data exploration and formal ANOVA.

Setting Up the Data Frame

In order to analyze the data, you need to get it into a form that S-PLUS can use for the analysis of variance. You do this by setting up a *data frame*. First create a numeric vector `coag`:

```
> coag <- scan()

1: 62 60 63 59
5: 63 67 71 64 65 66
11: 68 66 71 67 68 68
17: 56 62 60 61 63 64 63 59
25:
```

Next, create a factor called `diet`, that corresponds to `coag`:

```
> diet <- factor(rep(LETTERS[1:4],c(4,6,6,8)))
> diet

[1] A A A A B B B B B B C C C C C C D D D D D D D D
```

Now create a data frame with columns `diet` and `coag`:

```
> coag.df <- data.frame(diet,coag)
```

The data frame object `coag.df` is a matrix-like object, so it looks like a matrix when you display it on your screen:

```
> coag.df

   diet coag
 1    A   62
 2    A   60
 3    A   63
    .
    .
    .
23    D   63
24    D   59
```

A First Look at the Data

For each level of the treatment factor, you make an initial graphical exploration of the response data y_{ij} by using the functions `plot.design` and `plot.factor`.

You can make plots of the treatment means and treatment medians for each level of the experimental factor `diet` by using the function `plot.design` twice, as follows:

```
> par(mfrow=c(1,2))
> plot.design(coag.df)
> plot.design(coag.df, fun= median)
> par(mfrow=c(1,1))
```

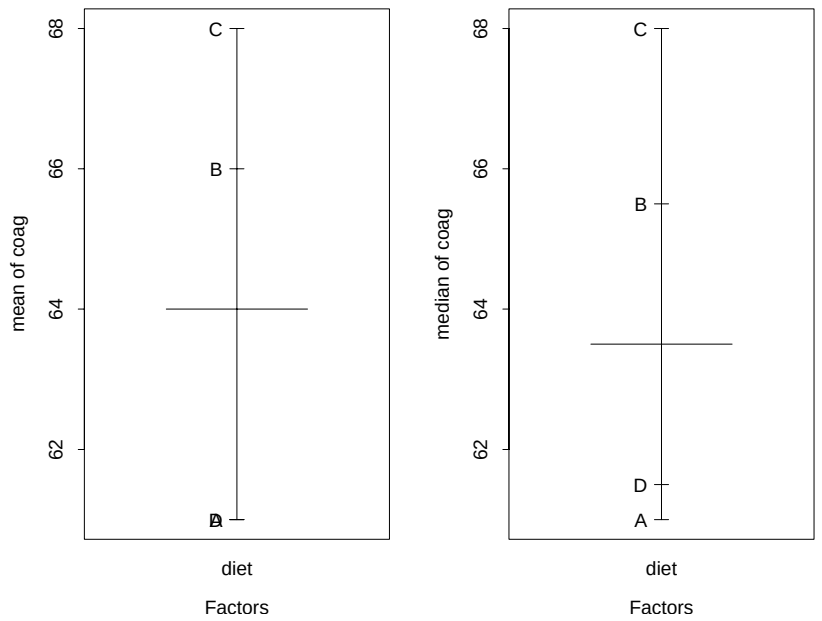

Figure 15.1: *Treatment means and medians.*

The results are shown in the two plots of Figure 15.1. In the left-hand plot, the tick marks on the vertical line are located at the treatment means for the diets A, B, C, and D, respectively. The mean values of coagulation time for diets A and D happen to have the same value, 61, and so the labels A and D are overlaid. The horizontal line, located at 64, indicates the overall mean of all the data.

In the right-hand plot of Figure 15.1, medians rather than means are indicated. There is not much difference between the treatment means and the treatment medians, so you should not be too concerned about adverse effects due to outliers.

The function `plot.factor` produces a boxplot of the response data for each level of the experimental factor:

```
> plot.factor(coag.df)
```

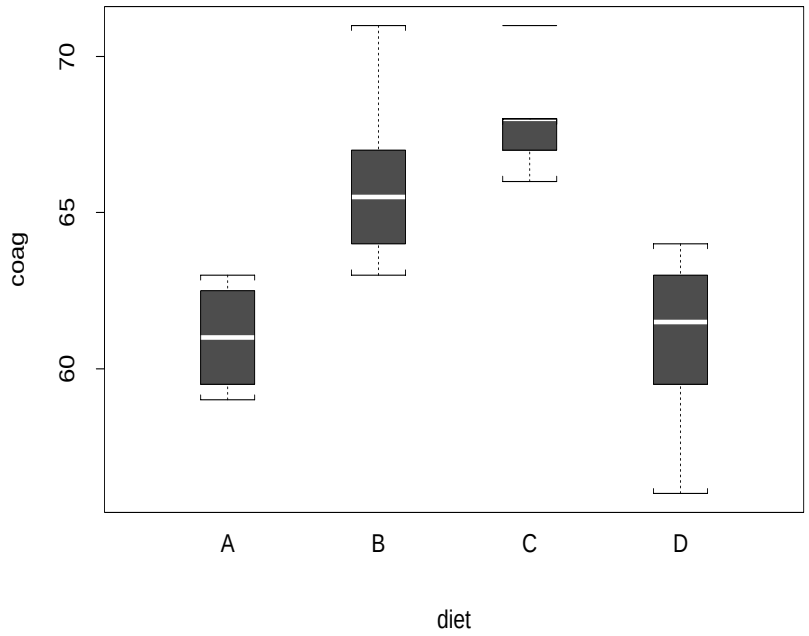

Figure 15.2: *Boxplots for each treatment.*

The resulting plot is shown in Figure 15.2. This plot indicates that the responses for diets A and D are quite similar, while the median responses for diets B and C are considerably larger relative to the variability reflected by the heights of the boxes. Thus, you suspect that diet has an effect on blood coagulation time.

If the exploratory graphical display of the response using `plot.factor` indicates that the interquartile distance of the boxplots depends upon the median, then a transformation to make the error variance constant is called for. The transformation may be selected

with a "spread versus level" plot. See, for example, the section The Two-Way Layout With Replicates or Hoaglin, Mosteller, and Tukey (1983).

The One-Way Layout Model and Analysis of Variance

The classical model for experiments with a single factor is

$$y_{ij} = \mu_i + \varepsilon_{ij} \qquad \begin{matrix} j = 1, \ldots, J \\ i = 1, \ldots, I \end{matrix}$$

where μ_i is the mean value of the response for the ith level of the experimental factor. There are I levels of the experimental factor, and J_i measurements $y_{i1}, y_{i2}, \ldots, y_{iJ}$ are taken on the response variable for level i of the experimental factor.

Using the treatment terminology, there are I treatments, and μ_i is called the ith treatment mean. The above model is often called the *one-way layout* model. For the blood coagulation experiment, there are $I = 4$ diets, and the means μ_1, μ_2, μ_3, and μ_4 correspond to diets A, B, C, and D, respectively. The numbers of observations are $J_A = 4$, $J_B = J_C = 6$, and $J_D = 8$.

You carry out the analysis of variance with the function `aov`:

```
> aov.coag <- aov(coag ~ diet, coag.df)
```

The first argument to `aov` above is the *formula* `coag ~ diet`. This formula is a symbolic representation of the one-way layout model equation; the formula excludes the error term ε_{ij}. The second argument to `aov` is the data frame you created, `coag.df`, which provides the data needed to carry out the ANOVA. The names `diet` and `coag`, used in the formula `coag ~ diet`, need to match the names of the variables in the data frame `coag.df`.

To display the ANOVA table, use `summary`:

```
> summary(aov.coag)

            Df Sum of Sq Mean Sq F Value       Pr(F)
      diet  3       228    76.0 13.5714 4.65847e-05
Residuals 20       112     5.6
```

The p-value is equal to .000047, which is highly significant.

Diagnostic Plots

You obtain the fitted values and residuals using the `fitted.values` and `residuals` functions on the result of `aov`. Thus, for example, you get the fitted values with the following:

```
> fitted.values(aov.coag)
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
61 61 61 61 66 66 66 66 66 66 68 68 68 68 68 68 61 61 61 61 61 61 61 61
```

The `resid` and `fitted` functions are shorter names for `residuals` and `fitted.values`, respectively.

You can check the residuals for distributional shape and outliers by using `hist` and `qqnorm`, with the residuals component of `aov.coag` as argument:

```
> hist(resid(aov.coag))
> qqnorm(resid(aov.coag))
```

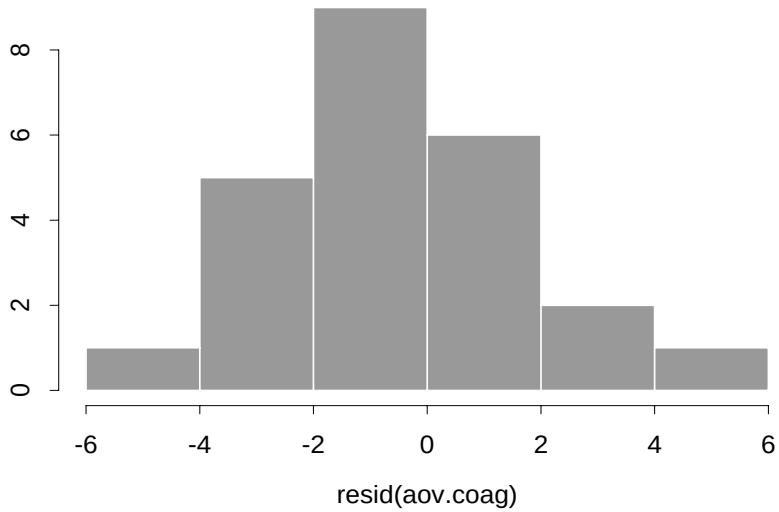

Figure 15.3: *Histogram of residuals.*

Figure 15.3 shows the resulting histogram and Figure 15.4 shows the resulting quantile-quantile plot.

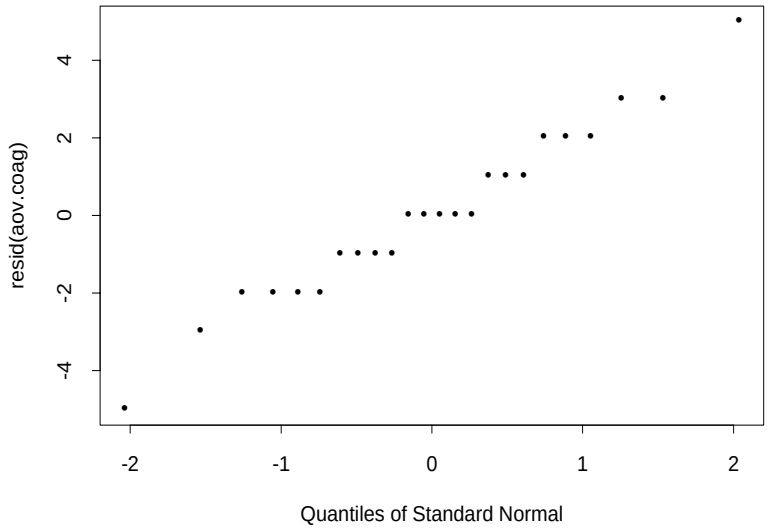

Figure 15.4: *Normal qq-plot of residuals.*

The shape of the histogram, and the linearity of the normal qq-plot, both indicate that the error distribution is quite Gaussian. (The flat sections in the qq-plot are a consequence of tied values in the data.)

You can check for inhomogeneity of error variance and possible outliers by plotting the residuals versus the fit:

```
> plot(fitted(aov.coag), resid(aov.coag))
```

This plot reveals no unusual features and is not shown.

Details

An alternate form of the one-way layout model is the *overall mean* plus *effects* form

$$y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

where μ is the overall mean, and α_i is the effect for level (or treatment) i. The mean μ_i for level (or treatment) i in the first form of the model is related to μ and α_i by

$$\mu_i = \mu + \alpha_i$$

and the effects α_i satisfy the constraint

$$\alpha_1 + \alpha_2 + \ldots + \alpha_I = 0 \ .$$

The function `aov` fits the one-way model in the "overall mean plus effects" form

$$y_{ij} = \hat{\mu} + \hat{\alpha}_i + r_{ij} \ .$$

See the section Model Coefficients and Contrasts on page 553 for more on this.

To obtain the effects, use `model.tables` as follows:

```
> model.tables(aov.coag)

Refitting model to allow projection
Tables of effects

 diet
     A B C  D
    -3 2 4 -3
 rep 4 6 6  8
```

You can get the treatment means as follows:

```
> model.tables(aov.coag, type="means")

Refitting model to allow projection
Tables of means

Grand mean

 64

 diet
     A  B  C  D
    61 66 68 61
rep  4  6  6  8
```

THE UNREPLICATED TWO-WAY LAYOUT

The data in Table 15.2 (used by Box, Hunter, and Hunter (1978)) were collected to determine the effect of treatments A, B, C, and D on the yield of penicillin in a penicillin manufacturing process.

Table 15.2: *Effect of four treatments on penicillin yield.*

	Treatment			
Block	**A**	**B**	**C**	**D**
Blend 1	89	88	97	94
Blend 2	84	77	92	79
Blend 3	81	87	87	85
Blend 4	87	92	89	84
Blend 5	79	81	80	88

The values of the response variable "yield" are the numbers in the table, and the columns of the table correspond to the levels A, B, C, and D of the treatment factor. There was a second factor, namely the blend factor, since a separate blend of the corn-steep liquor had to be made for each application of the treatments.

Your main interest is in determining whether the treatment factor affects yield. The blend factor is of only secondary interest; it is a "blocking" variable introduced to increase the sensitivity of the inference for treatments. The order of the treatments within blocks was chosen at random. Hence, this is a *randomized blocks* experiment.

The methods we use in this section applies equally well to two-factor experiments in which both factors are experimentally controlled and of equal interest.

Setting Up the Data Frame

Table 15.2 is *balanced*–each entry or *cell* of the table (that is, each row and column combination) has the same number of observations (one observation per cell, in the present example)–so you can use `fac.design` to create the data frame.

First, create a list `fnames` with two components named `blend` and `treatment`, where `blend` contains the level names of the blend factor and `treatment` contains the level names of the treatment factor:

```
> fnames <- list(blend=paste("Blend ", 1:5),
+ treatment=LETTERS[1:4])
```

Then use `fac.design` to create the *design* data frame `pen.design`

```
> pen.design <- fac.design(c(5,4), fnames)
```

The first argument, `c(5,4)`, to `fac.design` specifies the design as having two factors because its length is two. The `5` specifies five levels for the first factor, blend, and the `4` specifies four levels for the second factor, treatment. The second argument, `fnames`, specifies the factor names and the labels for their levels.

The design data frame `pen.design` that you just created contains the factors `blend` and `treatment` as its first and second columns, respectively.

Now create `yield` to match `pen.design`:

```
> yield <- scan()

1: 89 84 81 87 79
6: 88 77 87 92 81
11: 97 92 87 89 80
16: 94 79 85 84 88
21:
```

You can now use `data.frame` to combine the design data frame `pen.design` and the response `yield` into the data frame `pen.df`:

```
> pen.df <- data.frame(pen.design,yield)
```

Now look at pen.df:

```
> pen.df

      blend treatment yield
 1 Blend 1         A    89
 2 Blend 2         A    84
 3 Blend 3         A    81
 4 Blend 4         A    87
 5 Blend 5         A    79
 6 Blend 1         B    88
               .
               .
               .
19 Blend 4         D    84
20 Blend 5         D    88
```

Alternatively, you could build the model data frame directly from pen.design as follows:

```
> pen.design[,"yield"] <- yield
```

When you plot the object pen.design, S-PLUS uses the method plot.design, because the object pen.design is of class design. Thus, you obtain the same results as if you called plot.design explicitly on the object pen.df.

A First Look at the Data

You can look at the (comparative) values of the sample means of the data for each level of each factor using plot.design:

```
> plot.design(pen.df)
```

This function produces the plot shown in Figure 15.5. For the blend factor, each tick mark is located at the mean of the corresponding row of Table 15.2. For the treatment factor, each tick mark is located at the mean of the corresponding column of Table 15.2. The horizontal line is located at the sample mean of all the data. Figure 15.5 suggests that the blend has a greater effect on yield than does the treatment.

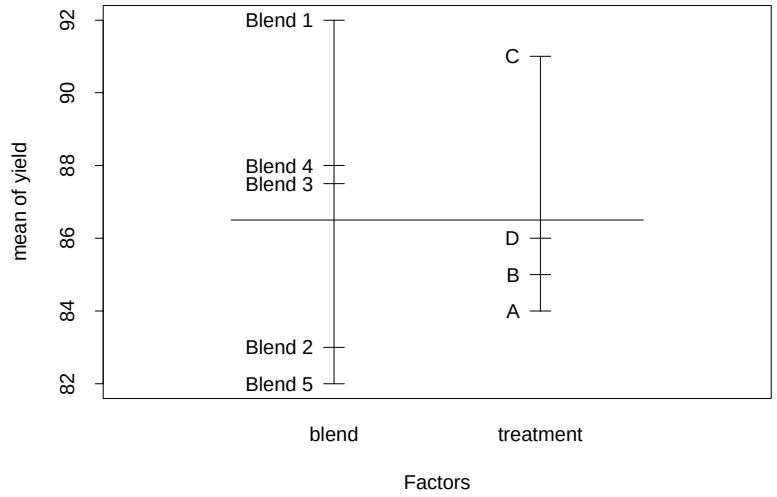

Figure 15.5: *Sample means in penicillin yield experiment.*

Since sample medians are insensitive to outliers, and sample means are not, you may want to make a plot similar to Figure 15.5 using sample medians instead of sample means. You can do this with `plot.design`, using the second argument `fun = median`:

```
> plot.design(pen.df, fun=median)
```

In this case, the plot does not indicate great differences between sample means and sample medians.

Use `plot.factor` to get a more complete exploratory look at the data. But first use `par` to get a one row by two column layout for two plots:

```
> par(mfrow=c(1,2))
> plot.factor(pen.df)
> par(mfrow=c(1,1))
```

This command produces the plot shown in Figure 15.6.

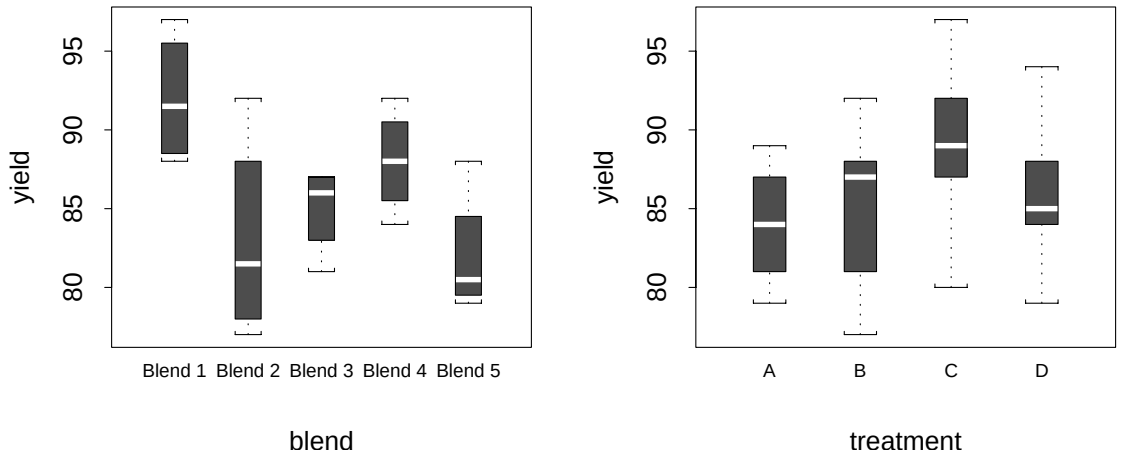

Figure 15.6: *Factor plot for penicillin yield experiment.*

The boxplots for factors, produced by `plot.factor`, give additional information about the data besides the location given by `plot.design`. The boxplots indicate variability, skewness, and outliers in the response, for each fixed level of each factor. For this particular data, the boxplots for both blends and treatments indicate rather constant variability, relatively little overall skewness, and no evidence of outliers.

For two-factor experiments, you should use `interaction.plot` to check for possible interactions (that is, nonadditivity). The `interaction.plot` function does not accept a data frame as an argument. Instead, you must supply appropriate factor names and the response name. To make these factor and response data objects available to `interaction.plot`, you must first "attach" the data frame `pen.df`:

```
> attach(pen.df)
> interaction.plot(treatment,blend,yield)
```

These commands produce the plot shown in Figure 15.7.

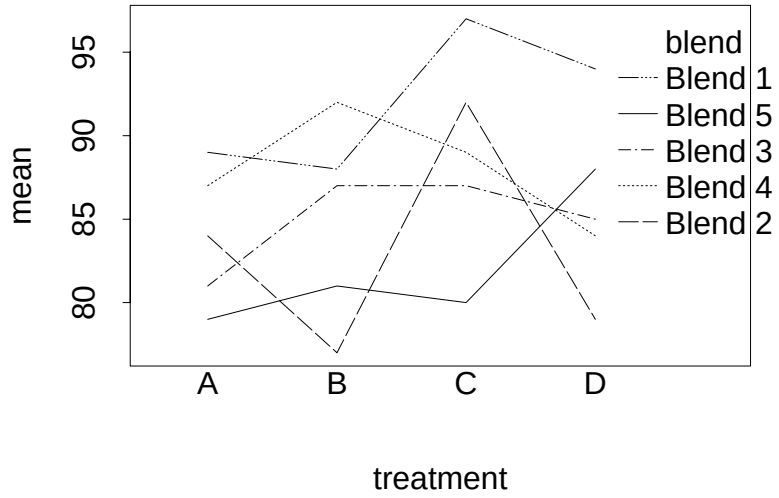

Figure 15.7: *Interaction plot of penicillin experiment.*

The first argument to `interaction.plot` specifies which factor appears along the *x*-axis (in this case, `treatment`). The second argument specifies which factor is associated with each line plot, or "trace" (in this case, `blend`). The third argument is the response variable (in this case, `yield`).

Without replication it is often difficult to interpret an interaction plot since random error tends to dominate. There is nothing striking in this plot.

The Two-Way Model and ANOVA (One Observation Per Cell)

The *additive* model for experiments with two factors, A and B, and one observation per cell is:

$$y_{ij} = \mu + \alpha_i^A + \alpha_i^B + \varepsilon_{ij} \qquad \begin{matrix} i = 1, \ldots, I \\ j = 1, \ldots, J \end{matrix}$$

where μ is the overall mean, α_i^A is the effect of the ith level of factor A and α_j^B is the effect of the jth level of factor B.

For the penicillin data above, factor A is "blend" and factor B is "treatment." Blend has $I = 5$ levels and treatment has $J = 4$ levels.

To estimate the *additive* model, use `aov`:

```
> aov.pen <- aov(yield ~ blend + treatment, pen.df)
```

The formula `yield ~ blend + treatment` specifies that a two factor additive model is fit, with `yield` the response, and `blend` and `treatment` the factors.

Display the analysis of variance table with `summary`:

```
> summary(aov.pen)

            Df Sum of Sq Mean Sq F Value    Pr(F)
    blend    4       264 66.0000 3.50442 0.040746
treatment    3        70 23.3333 1.23894 0.338658
Residuals   12       226 18.8333
```

The p-value for blend is moderately significant, while the p-value for treatment is insignificant.

Diagnostic Plots

Make a histogram of the residuals.

```
> hist(resid(aov.pen))
```

The resulting histogram is shown in Figure 15.8.

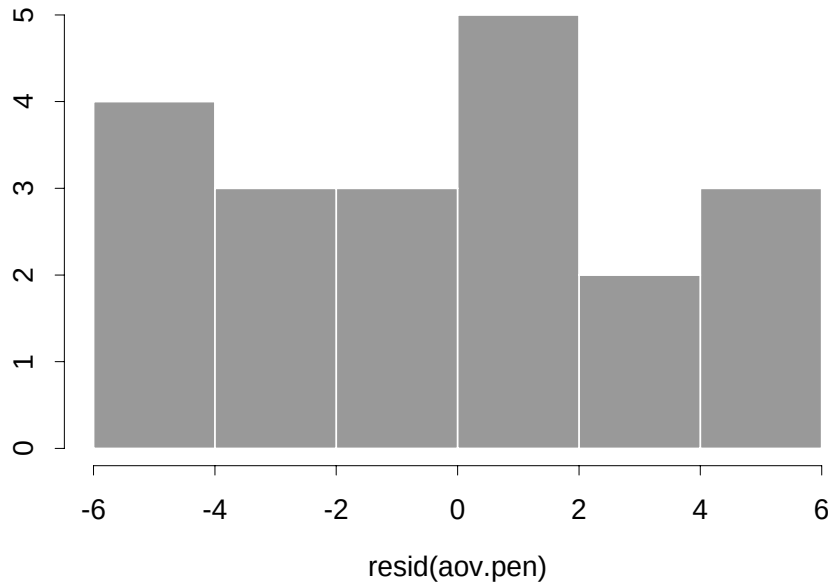

Figure 15.8: *Histogram of residuals for penicillin yield experiment.*

Now make a normal qq-plot of residuals:

```
> qqnorm(resid(aov.pen))
```

The resulting plot is shown in Figure 15.9.

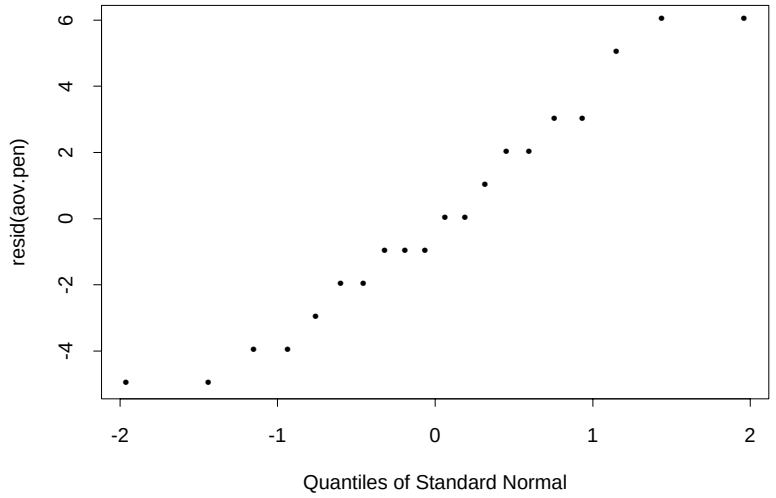

Figure 15.9: *Quantile-quantile plot of residuals for penicillin yield experiment.*

The central four cells of the histogram in Figure 15.8 are consistent with a fairly normal distribution in the middle. The linearity of the normal qq-plot in Figure 15.9, except near the ends, also suggests that the distribution is normal in the middle. The relatively larger values of the outer two cells of the histogram, and the flattening of the normal qq-plot near the ends, both suggest that the error distribution is slightly more short-tailed than a normal distribution. This is not a matter of great concern for the ANOVA F tests.

Make a plot of residuals versus the fit:

```
> plot(fitted(aov.pen), resid(aov.pen))
```

The resulting plot is shown in Figure 15.10.

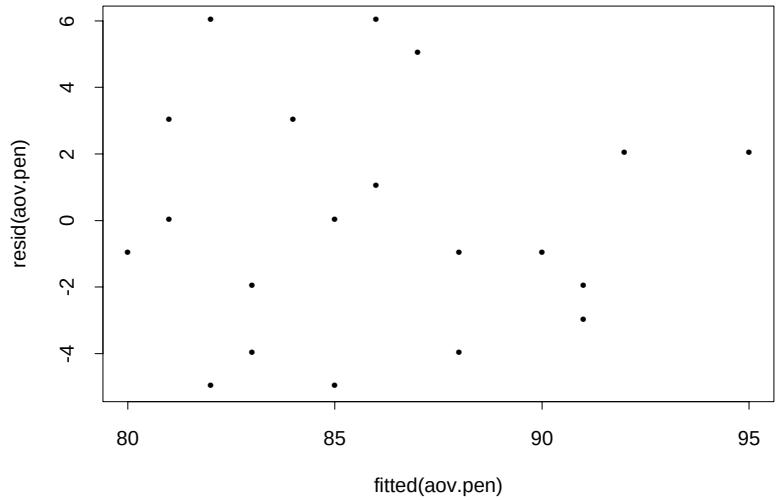

Figure 15.10: *Residuals vs. fitted values for penicillin yield experiment.*

The plot of residuals versus fit gives some slight indication that smaller error variance is associated with larger values of the fit.

Guidance

Since there is some indication of inhomogeneity of error variance, we now consider transforming the response, `yield`.

You may want to test for the existence of a multiplicative interaction, specified by the model

$$y_{ij} = \mu + \alpha_i^A + \alpha_j^B + \theta \alpha_i^A \alpha_j^B + \varepsilon_{ij} .$$

When the unknown parameter θ is not zero, multiplicative interaction exists. A test for the null hypothesis of no interaction may be carried out using the test statistic T_{1df} for Tukey's one degree of freedom for nonadditivity.

An S-PLUS function, `tukey.1`, is provided in the section Details on page 522. You can use it to compute T_{1df} and the p-value. For the penicillin data:

```
> tukey.1(aov.pen, pen.df)
```

```
$T.1df:
[1] 0.09826791

$p.value:
[1] 0.7597822
```

The statistic $T_{1df} = .098$ has a p-value of $p = .76$, which is not significant. Therefore, there is no indication of a multiplicative interaction.

Assuming that the response values are positive, you can find out whether or not the data suggest a specific transformation to remove multiplicative interaction as follows: Plot the residuals r_{ij} for the additive fit versus the *comparison values*

$$c_{ij} = \frac{\hat{\alpha}_i \hat{\alpha}_j}{\hat{\mu}}.$$

If this plot reveals a linear relationship with estimated slope $\hat{\theta}$, then you should analyze the data again, using as new response values the power transformation y_{ij}^{λ} of the original response variables y_{ij}, with exponent

$$\lambda = 1 - \hat{\theta}.$$

(If $\lambda = 0$, use $\log(y_{ij})$.) See Hoaglin, Mosteller, and Tukey (1983) for details.

An S-PLUS function called `comp.plot`, for computing the comparison values c_{ij}, plotting r_{ij} versus c_{ij}, and computing $\hat{\theta}$, is provided in the section Details on page 522 below. Applying `comp.plot` to the penicillin data gives the following result:

```
> comp.plot(aov.pen, pen.df)

$theta.hat:
[1] 4.002165

$std.error:
[1] 9.980428
```

```
$R.squared:
          R2
 0.008854346
```

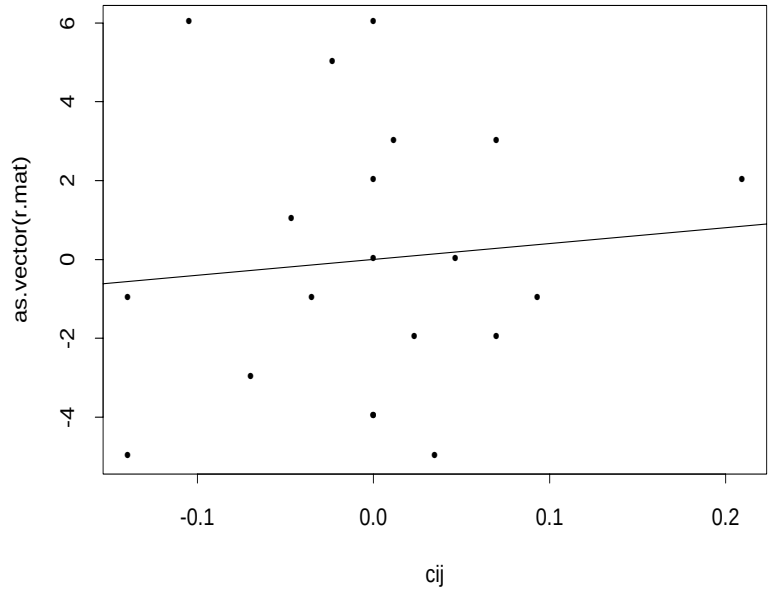

Figure 15.11: *Display from* `comp.plot`.

In this case, the estimated slope is $\hat{\theta} = 4$, which gives $\lambda = -3$. However, this is not a very sensible exponent for a power transformation. The standard deviation of $\hat{\theta}$ is nearly 10 and the R^2 is only .009, which indicates that θ may be zero. Thus, we do not recommend using a power transformation.

Details

The test statistic T_{1df} for Tukey's one degree of freedom is given by:

$$T_{1df} = (IJ - I - J)\frac{SS_{\theta}}{SS_{res.1}}$$

where

$$SS_{\theta} = \frac{\left(\sum_{i=1}^{I} \sum_{j=1}^{J} \hat{\alpha}_i^A \hat{\alpha}_j^B y_{ij} \right)^2}{\sum_{i=1}^{I} (\hat{\alpha}_i^A)^2 \sum_{j=1}^{J} (\hat{\alpha}_j^B)^2}$$

$$SS_{res.1} = SS_{res} - SS_{\theta}$$

$$SS_{res} = \sum_{i=1}^{I} \sum_{j=1}^{J} r_{ij}^2$$

with the $\hat{\alpha}_i^A$, $\hat{\alpha}_j^B$ the additive model estimates of the α_i^A and α_j^B, and r_{ij} the residuals from the additive model fit. The statistic T_{1df} has an $F_{1,IJ-I-J}$ distribution.

Here is a function `tukey.1` to compute the Tukey one degree of freedom for nonadditivity test. You can create your own version of this function by typing `tukey.1 <-` and then the definition of the function.

```
> tukey.1

function(aov.obj, data)
{
        vnames <- names(aov.obj$contrasts)
        if(length(vnames) != 2)
                stop("the model must be two-way")
        vara <- data[, vnames[1]]
        varb <- data[, vnames[2]]
        na <- length(levels(vara))
        nb <- length(levels(varb))
        resp <- data[, as.character(attr(aov.obj$terms,
          "variables")[attr(aov.obj$terms, "response" )])]
        cfs <- coef(aov.obj)
        alpha.A <- aov.obj$contrasts[[vnames[1]]] %*% cfs[
                aov.obj$assign[[vnames[1]]]]
```

```
        alpha.B <- aov.obj$contrasts[[vnames[2]]] %*% cfs[
                aov.obj$assign[[vnames[2]]]]
        r.mat <- matrix(0, nb, na)
        r.mat[cbind(as.vector(unclass(varb)), as.vector(
                unclass(vara)))] <- resp
        SS.theta.num <- sum((alpha.B %*% t(alpha.A)) *
                r.mat)^2
        SS.theta.den <- sum(alpha.A^2) * sum(alpha.B^2)
        SS.theta <- SS.theta.num/SS.theta.den
        SS.res <- sum(resid(aov.obj)^2)
        SS.res.1 <- SS.res - SS.theta
        T.1df <- ((na * nb - na - nb) * SS.theta)/SS.res.1
        p.value <- 1 - pf(T.1df, 1, na * nb - na - nb)
        list(T.1df = T.1df, p.value = p.value)
}
```

Here is a function `comp.plot` for computing a least-squares fit to the plot of residuals versus comparison values:

```
> comp.plot

function(aov.obj, data)
{
        vnames <- names(aov.obj$contrasts)
        if(length(vnames) != 2)
                stop("the model must be two-way")
        vara <- data[, vnames[1]]
        varb <- data[, vnames[2]]
        cfs <- coef(aov.obj)
        alpha.A <- aov.obj$contrasts[[vnames[1]]] %*% cfs[
                aov.obj$assign[[vnames[1]]]]
        alpha.B <- aov.obj$contrasts[[vnames[2]]] %*% cfs[
                aov.obj$assign[[vnames[2]]]]
        cij <- alpha.B %*% t(alpha.A)
        cij <- c(cij)/cfs[aov.obj$assign$"(Intercept)"]
        na <- length(levels(vara))
        nb <- length(levels(varb))
        r.mat <- matrix(NA, nb, na)
        r.mat[cbind(as.vector(unclass(varb)), as.vector(
                unclass(vara)))] <- resid(aov.obj)
        plot(cij, as.vector(r.mat))
        ls.fit <- lsfit(as.vector(cij), as.vector(r.mat))
        abline(ls.fit)
```

```
        output <- ls.print(ls.fit, print.it = F)
        list(theta.hat = output$coef.table[2, 1], std.error
                = output$coef.table[2, 2], R.squared =
                output$summary[2])
}
```

THE TWO-WAY LAYOUT WITH REPLICATES

The data in Table 15.3 (used by Box, Hunter, and Hunter (1978)) displays the survival times, in units of 10 hours, of animals in a 3 x 4 *replicated* factorial experiment. In this experiment, each animal was given one of three poisons, labeled I, II, and III, and one of four treatments, labeled A, B, C, and D. Four animals were used for each combination of poison and treatment, making four *replicates*.

Table 15.3: *A replicated factorial experiment.*

	treatment			
poison	**A**	**B**	**C**	**D**
I	0.31	0.82	0.43	0.45
	0.45	1.10	0.45	0.71
	0.46	0.88	0.63	0.66
	0.43	0.72	0.76	0.62
II	0.36	0.92	0.44	0.56
	0.29	0.61	0.35	1.02
	0.40	0.49	0.31	0.71
	0.23	1.24	0.40	0.38
III	0.22	0.30	0.23	0.30
	0.21	0.37	0.25	0.36
	0.18	0.38	0.24	0.31
	0.23	0.29	0.22	0.33

Setting Up the Data Frame

To set up the data frame, first make a list, `fnames`, with components `treatment` and `poison`, containing the level names of these two factors:

```
> fnames <- list(treatment=LETTERS[1:4],
+ poison=c("I","II","III"))
```

Use `fac.design`, with optional argument `rep = 4`, to create the design data frame `poisons.design`:

```
> poisons.design <- fac.design(c(4,3), fnames, rep=4)
```

Note that since `treatments` is the first factor in the `fnames` list and treatments has 4 levels, 4 is the *first* argument of `c(4,3)`.

You now need to create the vector `surv.time` to match `poisons.design`. Each replicate of the experiment consists of data in three rows of Table 15.3. Rows 1, 5, and 9 make up the first replicate, and so on. The command to get what we want is:

```
> surv.time <- scan()

1: .31 .82 .43 .45
5: .36 .92 .44 .56
9: .22 .30 .23 .30
13: .45 1.10 .45 .71
17: .29 .61 .35 1.02
21: .21 .37 .25 .36
25: .46 .88 .63 .66
29: .40 .49 .31 .71
33: .18 .38 .24 .31
37: .43 .72 .76 .62
41: .23 1.24 .40 .38
45: .23 .29 .22 .33
49:
```

Finally, make the data frame `poisons.df`:

```
> poisons.df <- data.frame(poisons.design, surv.time)
```

A First Look at the Data

Use `plot.design`, `plot.factor`, and `interaction.plot` to get a first look at the data through summary statistics.

Set `par(mfrow = c(3,2))` and use the above three functions to get the three row and two column layout of plots displayed in Figure 15.12:

```
> par(mfrow=c(3,2))
```

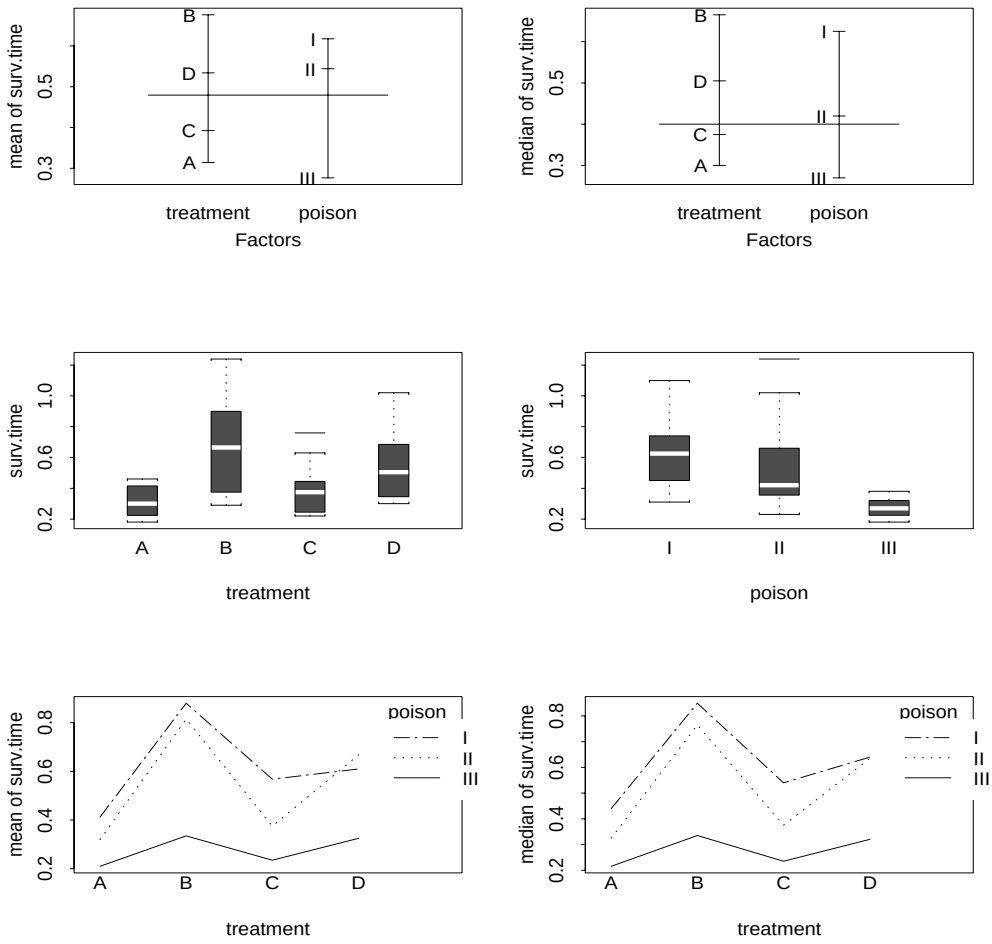

Figure 15.12: *Initial plots of the data.*

To obtain the design plot of sample means shown in the upper left plot of Figure 15.12, use `plot.design` as follows:

```
> plot.design(poisons.df)
```

To obtain the design plot of sample medians shown in the upper right-hand plot of Figure 15.12, use `plot.design` again:

```
> plot.design(poisons.df, fun=median)
```

The two sets of boxplots shown in the middle row of Figure 15.12 are obtained with:

```
> plot.factor(poisons.df)
```

To obtain the bottom row of Figure 15.12, use `interaction.plot`:

```
> attach(poisons.df)
> interaction.plot(treatment,poison,surv.time)
> interaction.plot(treatment,poison,surv.time,fun=median)
```

The main differences between the plots obtained with `plot.design` using means and medians are:

- the difference between the horizontal lines which represents the mean and median, respectively, for all the data,
- the difference between the tick marks for the poison factor at level II.

The boxplots resulting from the use of `plot.factor` indicate a clear tendency for variability to increase with the (median) level of response.

The plots made with `interaction.plot` show stronger treatment effects for the two poisons with large levels than for the lowest level poison–an indication of an interaction.

The Two-Way Model and ANOVA (With Replicates)

When you have replicates, you can consider a model which includes an interaction term α_{ij}^{AB}:

$$y_{ijk} = \mu + \alpha_i^A + \alpha_j^B + \alpha_{ij}^{AB} + \varepsilon_{ijk} \qquad \begin{array}{l} i = 1, \ldots, I \\ j = 1, \ldots, J \\ k = 1, \ldots, K \end{array}$$

You can now carry out an ANOVA for the above model using `aov` as follows:

```
> aov.poisons <- aov(surv.time ~ poison*treatment,
+ poisons.df)
```

The expression `poison*treatment` on the right-hand side of the formula specifies that `aov` fit the above model with interaction. This contrasts with the formula `surv.time ~ poison + treatment`, which tells `aov` to fit an additive model for which α_{ij}^{AB} is assumed to be zero for all levels *i,j*.

You now display the ANOVA table with `summary`:

```
> summary(aov.poisons)

                Df Sum of Sq   Mean Sq  F Value     Pr(F)
poison           2  1.033013 0.5165063 23.22174 0.0000003
treatment        3  0.921206 0.3070688 13.80558 0.0000038
poison:treatment 6  0.250138 0.0416896  1.87433 0.1122506
Residuals       36  0.800725 0.0222424
```

The *p*-values for both poisons and treatment are highly significant, while the *p*-value for interaction is insignificant.

The colon in `poison:treatment` denotes an interaction, in this case the poison-treatment interaction.

Diagnostic Plots

Make a histogram and a normal qq-plot of residuals, arranging the plots side by side in a single figure with `par(mfrow = c(1,2))` before using `hist` and `qqnorm`:

```
> par(mfrow=c(1,2))
> hist(resid(aov.poisons))
> qqnorm(resid(aov.poisons))
> par(mfrow=c(1,1))
```

The call `par(mfrow = c(1,1))`, resets the plot layout to a single plot per figure.

The histogram in the left-hand plot of Figure 15.13 reveals a marked asymmetry, which is reflected in the normal qq-plot in the right-hand side of Figure 15.13. The latter shows a curved departure from

linearity toward the lower left part of the plot, and a break in linearity in the upper right part of the plot. Evidently, all is not well (see the discussion on transforming the data in the Guidance section below).

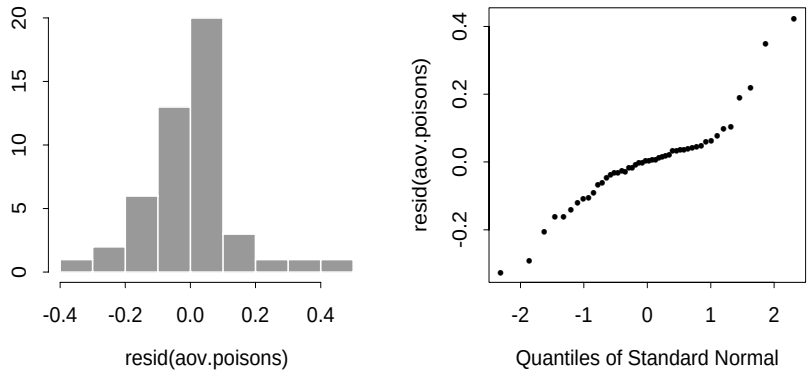

Figure 15.13: *Histogram and normal qq-plot of residuals.*

Make a plot of residuals versus fit:

```
plot(fitted(aov.poisons), resid(aov.poisons))
```

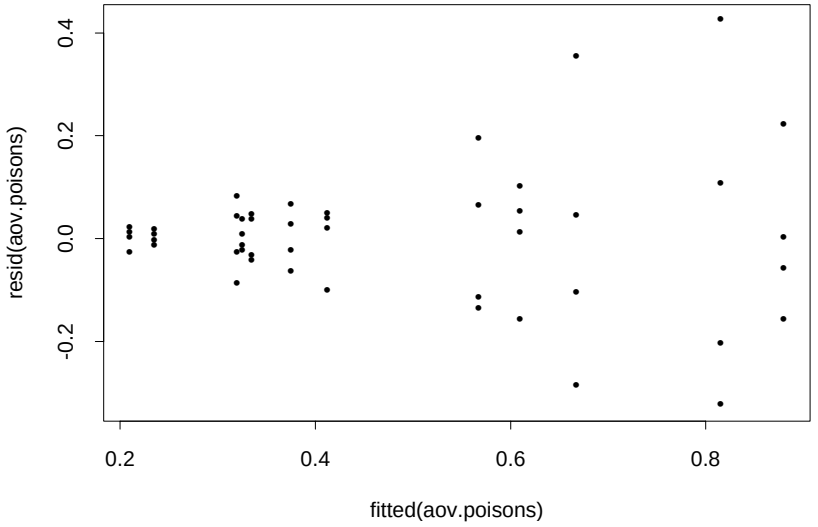

Figure 15.14: *Plot of residuals versus fit.*

The result, displayed in Figure 15.14, clearly reveals a strong relationship between the residuals and the fitted values. The variability of the residuals increases with increasing fitted values. This is another indication that transformation would be useful.

Guidance

When the error variance for an experiment varies with the expected value of the observations, a *variance stabilizing* transformation will often reduce or eliminate such behavior.

We shall show two methods for determining an appropriate variance stabilizing transformation, one which requires replicates and one which does not.

Method for Two-Factor Experiments With Replicates

For two-factor experiments with replicates, you can gain insight into an appropriate variance stabilizing transformation by carrying out the following informal procedure. First, calculate the within-cell standard deviations $\hat{\sigma}_{ij}$ and means $\bar{y}_{ij}$:

```
> std.poison <- tapply(poisons.df$surv.time,
+ list(poisons.df$treatment,
+ poisons.df$poison),var)^.5
> std.poison <- as.vector(std.poison)
> means.poison <- tapply(poisons.df$surv.time,
+ list(poisons.df$treatment,
+ poisons.df$poison),mean)
> means.poison <- as.vector(means.poison)
```

Then plot $\log(\hat{\sigma}_{ij})$ versus $\log(\bar{y}_{ij})$ and use the slope of the regression line to estimate the variance stabilizing transform:

```
> plot(log(means.poison),log(std.poison))
> var.fit <- lsfit(log(means.poison),
+ log(std.poison))
> abline(var.fit)
> theta <- var.fit$coef[2]
> theta

       X
 1.97704
```

Now let $\hat{\lambda} = 1 - \hat{\theta}$ and choose λ to be that value among the set of values $-1,\ -\frac{1}{2},\ 0,\ \frac{1}{2},\ 1$ which is closest to $\hat{\lambda}$. If $\lambda = 0$, then make the transformation $\tilde{y}_{ij} = \log y_{ij}$. Otherwise, make the power transformation $\tilde{y}_{ijk} = y_{ijk}^{\lambda}$. Now you should repeat the complete analysis described in the previous subsections, using the response $\tilde{y}_{ijk}$ in place of y_{ijk}.

Since for the poisons experiment you get $\hat{\theta} \approx 2$, you choose $\lambda = -1$. This gives a reciprocal transformation $\tilde{y}_{ijk} = y_{ijk}^{-1}$, where y_{ijk} are the values you used in the response with `surv.time`. You can think of the new response $\tilde{y}_{ijk}$ as representing the rate of dying.

The model can be refit using the transformed response:

```
> summary(aov(1/surv.time ~ poison*treatment, poisons.df))

                 Df Sum of Sq  Mean Sq  F Value     Pr(F)
poison            2  34.87712 17.43856 72.63475 0.0000000
treatment         3  20.41429  6.80476 28.34307 0.0000000
poison:treatment  6   1.57077  0.26180  1.09042 0.3867329
Residuals        36   8.64308  0.24009
```

With the transformation the p-values for the main effects have decreased while the p-value for the interaction has increased–a more satisfactory fit. The diagnostic plots with the new response are much improved also.

Method for Unreplicated Two-Factor Experiments

An alternative simple method for estimating the variance stabilizing transformation is based on the relationship between the log of the absolute residuals and the log of the fitted values. This method has the advantage that it can be used for unreplicated designs. This method is also often preferred to that of plotting $\log \hat{\sigma}_{ij}$ against $\bar{y}_{ij}$ even for

cases with replication, because $\bar{y}_{ij}$ and $\hat{\sigma}_{ij}$ are not always adequately good estimates of the mean and standard deviation for small values of K ($K < 8$).

This method consists of plotting log of absolute residuals versus log of fitted values, and computing the slope $\hat{\theta}$ of the regression line. You then set $\hat{\lambda} = 1 - \hat{\theta}$. Residuals with very small absolute values should usually be omitted before applying this method. Here is some sample code.

```
> plot(log(abs(fitted(aov.poisons)[
+ abs(resid(aov.poisons))>exp(-10)])),
+ log(abs(resid(aov.poisons)[
+ abs(resid(aov.poisons))>exp(-10)])))
> logrij.fit <- lsfit(
+ log(abs(fitted(aov.poisons)[
+ abs(resid(aov.poisons))>exp(-10)])),
+ log(abs(resid(aov.poisons)[
+ abs(resid(aov.poisons))>exp(-10)])))
> abline(logrij.fit)
> theta <- logrij.fit$coef[2]
> theta

        X
1.930791
```

You get $\hat{\lambda} = 1 - \hat{\theta} \approx -1$.

Note that the two simple methods described above both lead to nearly identical choices of power transformation to stabilize variance.

Details

You will find that a nonconstant standard deviation for observations y_i (y_{ijk} for the two-factor experiment with replicates) is well-explained by a power law relationship in many datasets. In particular, for some constant B and some exponent θ, we have

$$\sigma_y \approx B\eta^{\theta}$$

where σ_y is the standard deviation of the y_i and η is the mean of the y_i. If you then use a power law transformation

$$\tilde{y}_i = y_i^{\lambda}$$

for some fixed exponent λ, it can be shown that the standard deviation $\sigma_{\tilde{y}}$ for the transformed data $\tilde{y}_i$, is given by

$$\sigma_{\tilde{y}} = K\lambda\eta^{\lambda-(1-\theta)} .$$

You can therefore make $\sigma_{\tilde{y}}$ have a constant value, independent of the mean η of the original data y_i (and independent of the approximate mean η^{λ} of the transformed data $\tilde{y}_i$), by choosing

$$\lambda = 1 - \theta .$$

Note that

$$\log \sigma_y \approx \log K + \theta \log \eta$$

Suppose you plot $\log \hat{\sigma}_{ij}$ versus $\log \hat{y}_{ij}$ for a two-factor experiment with replicates and find that this plot results in a fairly good straight line fit with slope $\hat{\theta}$, where $\hat{\sigma}_{ij}$ is an estimate of σ_y and $\hat{y}_{ij}$ is an estimate of η Then the slope $\hat{\theta}$ provides an estimate of θ, and so you set $\hat{\lambda} = 1 - \hat{\theta}$. Since a fractional exponent $\hat{\lambda}$ is not very natural, one often chooses the closest value $\hat{\lambda}$ in the "natural" set:

-1	Reciprocal
$-\frac{1}{2}$	Reciprocal square root
0	Log
$\frac{1}{2}$	Square root
1	No transformation

Alternative Formal Methods

There are two alternative formal approaches to stabilizing the variance. One approach is to select the power transformation that minimizes the residual squared error. This is equivalent to maximizing the log-likelihood function and is sometimes referred to as a Box-Cox analysis (see, for example, Weisberg (1985); Box (1988); Haaland (1989)).

The second approach seeks to stabilize the variance without the use of a transformation, by including the variance function directly in the model. This approach is called generalized least squares/variance function estimation (see, for example, Carroll and Ruppert (1988); Davidian and Haaland (1990)).

Transformations are easy to use and may provide a simpler, more parsimonious model (Box (1988)). On the other hand, modeling the variance function directly allows the analysis to proceed on the original scale and allows more direct insight into the nature of the variance function. In cases when the stability of the variance is critical, either of these methods have better statistical properties than the simple informal graphical methods described above.

MANY FACTORS AT TWO LEVELS: 2^K DESIGNS

The data in Table 15.4 come from an industrial product development experiment in which a response variable called *conversion* is measured (in percent) for each possible combination of two levels of four factors:

- `K`: *catalyst charge* (10 or 15 pounds),
- `Te`: *temperature* (220 or 240° C),
- `P`: *pressure* (50 or 80 pounds per square inch),
- `C`: *concentration* (10% or 12%).

Table 15.4: *Data from product development experiment.*

	Factor					
observation number	**K**	**Te**	**P**	**C**	**conversion(%)**	**run order**
1	–	–	–	–	71	(8)
2	+	–	–	–	61	(2)
3	–	+	–	–	90	(10)
4	+	+	–	–	82	(4)
5	–	–	+	–	68	(15)
6	+	–	+	–	61	(9)
7	–	+	+	–	87	(1)
8	+	+	+	–	80	(13)
9	–	–	–	+	61	(16)

Table 15.4: *Data from product development experiment. (Continued)*

Factor						
observation number	**K**	**Te**	**P**	**C**	**conversion(%)**	**run order**
10	+	−	−	+	50	(5)
11	−	+	−	+	89	(11)
12	+	+	−	+	83	(14)
13	−	−	+	+	59	(3)
14	+	−	+	+	51	(12)
15	−	+	+	+	85	(6)
16	+	+	+	+	78	(7)

The levels are labeled "-" and "+" in the table. All the factors in the experiment are quantitative, so the "-" indicates the "low" level and the "+" indicates the "high" level for each factor. This data set was used by Box, Hunter, and Hunter (1978).

The design for this experiment is called a 2^4 design because there are $2^4 = 16$ possible combinations of two levels for four factors.

Setting Up the Data Frame

To set up the data frame first create a list of the four factor names with the corresponding pairs of levels labels:

```
> fnames <- list(K=c("10","15"), Te=c("220","240"),
+ P=c("50","80"), C=c("10","12"))
```

Now use `fac.design` to create the 2^k design data frame `devel.design`:

```
> devel.design <- fac.design(rep(2,4), fnames)
```

The first argument to `fac.design` is a vector of length four, which specifies that there are four factors. Each entry of the vector is a 2, which specifies that there are two levels for each factor.

Since `devel.design` matches Table 15.4, you can simply scan in the coversion data:

```
> conversion <- scan()

1: 71 61 90 82 68 61 87 80
9: 61 50 89 83 59 51 85 78
17:
```

Finally, create the data frame `devel.df`:

```
> devel.df <- data.frame(devel.design, conversion)
> devel.df

    K  Te  P  C conversion
 1 10 220 50 10         71
 2 15 220 50 10         61
 3 10 240 50 10         90
             .
             .
             .
15 10 240 80 12         85
16 15 240 80 12         78
```

A First Look at the Data

Use `plot.design` and `plot.factor` to make an initial graphical exploration of the data. To see the design plot with sample means, use the following command, which yields the plot shown in Figure 15.15:

```
> plot.design(devel.df)
```

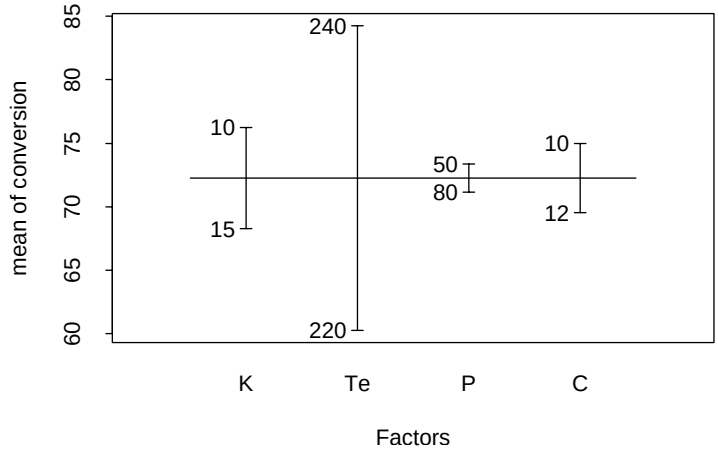

Figure 15.15: *Sample means for product development experiment.*

To see the design plot with sample medians, use:

```
> plot.design(devel.df, fun=median)
```

To see boxplots of the factors, use the following commands, which yield the plots shown in Figure 15.16:

```
> par(mfrow=c(2,2))
> plot.factor(devel.df)
> par(mfrow=c(1,1))
```

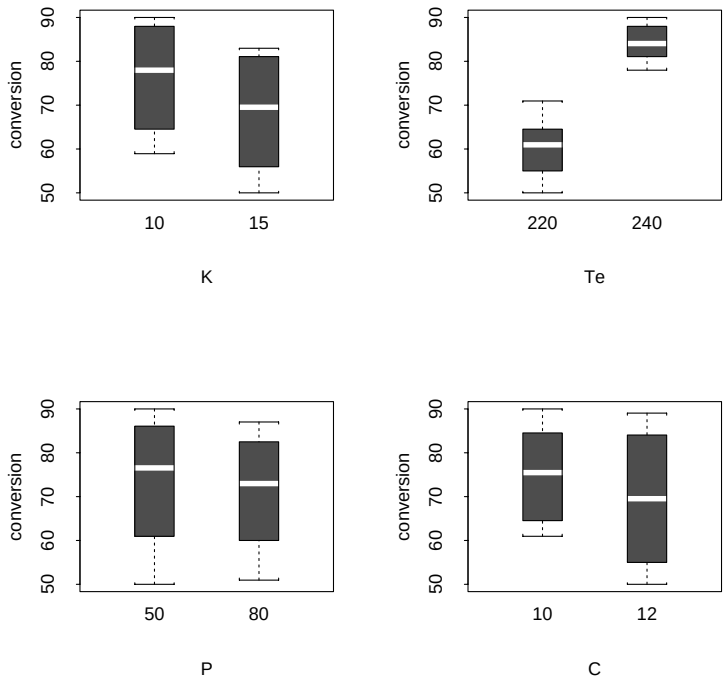

Figure 15.16: *Factor plot for product development experiment.*

Estimating All Effects in the 2^k Model

You can use `aov` to estimate *all* effects (main effects and all interactions), and carry out the analysis of variance. Let's do so, and store the results in `aov.devel`:

```
> aov.devel <- aov(conversion ~ K*Te*P*C, devel.df)
```

The product form `K*Te*P*C` on the right-hand side of the formula tells S-PLUS to fit the above 2^4 design model with *all* main effects and *all* interactions included. You can accomplish the same thing by using the power function `^` to raise the expression `K+Te+P+C` to the 4th power:

```
> aov.devel <- aov(conversion ~ (K+Te+P+C)^4, devel.df)
```

This second method is useful when you want to specify only main effects plus certain low-order interactions. For example, replacing 4 by 2 above results in a model with all main effects and all second-order interactions.

You can obtain the estimated coefficients using the `coef` function on the `aov` output:

```
> coef(aov.devel)

 (Intercept)  K Te      P     C K:Te   K:P   Te:P
K:C
       72.25 -4 12 -1.125 -2.75  0.5 0.375 -0.625 -
5.464379e-17
 Te:C    P:C K:Te:P K:Te:C  K:P:C Te:P:C K:Te:P:C
 2.25 -0.125 -0.375   0.25 -0.125 -0.375   -0.125
```

Notice that colons are used to connect factor names to represent interactions, for example, `K:P:C` is the three factor interaction between the factors `K`, `P`, and `C`.

For more on the relationship between coefficients, contrasts and effects, see the section Experiments With One Factor on page 504 and the section The Unreplicated Two-Way Layout on page 512.

You can get the analysis of variance table with the `summary` command:

```
> summary(aov.devel)

          Df Sum of Sq Mean Sq
K          1    256.00  256.00
Te         1   2304.00 2304.00
P          1     20.25   20.25
C          1    121.00  121.00
K:Te       1      4.00    4.00
K:P        1      2.25    2.25
Te:P       1      6.25    6.25
K:C        1      0.00    0.00
Te:C       1     81.00   81.00
P:C        1      0.25    0.25
K:Te:P     1      2.25    2.25
K:Te:C     1      1.00    1.00
K:P:C      1      0.25    0.25
Te:P:C     1      2.25    2.25
K:Te:P:C   1      0.25    0.25
```

The ANOVA table does not provide any *F* statistics. This is because you have estimated 16 parameters with 16 observations. There are no degrees of freedom left for estimating the error variance, and hence

there is no error mean square to use as the denominator of the F statistics. However, the ANOVA table can give you some idea of which effects are the main contributors to the response variation.

Estimating All Effects in the 2^k Model With Replicates

On some occasions, you may have replicates of a 2^k design. In this case, you can estimate the error variance σ^2 as well as all effects. For example, the data in Table 15.5 is from a replicated 2^3 pilot plant example used by Box, Hunter, and Hunter (1978). The three factors are *temperature* (`Te`), *concentration* (`C`) and *catalyst* (`K`), and the response is *yield.*

Table 15.5: *Replicated pilot plant experiment.*

Te	C	K	rep 1	rep 2
–	–	–	59	61
+	–	–	74	70
–	+	–	50	58
+	+	–	69	67
–	–	+	50	54
+	–	+	81	85
–	+	+	46	44
+	+	+	79	81

To set up the data frame, first make the factor names list:

```
> fnames <- list(Te=c("Tl","Th"), C=c("Cl","Ch"),
+ K=c("Kl","Kh"))
```

Because T is a constant in S-PLUS which stands for the logical value "true," you can not use T as a factor name for temperature. Instead, use Te, or some such alternative abbreviation. Then make the design data frame, pilot.design, with M = 2 replicates, by using fac.design with the optional argument rep = 2:

```
> pilot.design <- fac.design(c(2,2,2), fnames, rep=2)
```

Now, create the response vector pilot.yield as a vector of length 16, with the second replicate values following the first replicate values:

```
> pilot.yield <- scan()

1: 59 74 50 69 50 81 46 79
9: 61 70 58 67 54 85 44 81
17:
```

Finally, use data.frame:

```
> pilot.df <- data.frame(pilot.design, pilot.yield)
```

You can now carry out the ANOVA, and because the observations are replicated, the ANOVA table has an error variance estimate, that is, mean square for error, and *F* statistics:

```
> aov.pilot <- aov(pilot.yield ~ (Te + C + K)^3, pilot.df)
> summary(aov.pilot)

            Df Sum of Sq Mean Sq F Value    Pr(F)
        Te  1      2116    2116 264.500 0.000000
         C  1       100     100  12.500 0.007670
         K  1         9       9   1.125 0.319813
     Te:C  1         9       9   1.125 0.319813
     Te:K  1       400     400  50.000 0.000105
      C:K  1         0       0   0.000 1.000000
   Te:C:K  1         1       1   0.125 0.732810
Residuals  8        64       8
```

Temperature is clearly highly significant, as is the temperature-catalyst interaction, and concentration is quite significant.

Estimating All Small Order Interactions

In cases where you are confident that high-order interactions are unlikely, you can fit a model which includes interactions only up to a fixed order, through the use of the power function ^ with an

appropriate exponent. For example, in the product development experiment of Table 15.4, you may wish to estimate only the main effects and all second-order interactions. In this case, use :

```
> aov.devel.2 <- aov(conversion ~ (K+Te+P+C)^2,devel.df)
```

Now you are using 16 observations to estimate 11 parameters: the mean, the four main effects, and the six two-factor interactions. Since you only use 11 degrees of freedom for the parameters, out of a total of 16, you still have 5 degrees of freedom to estimate the error variance. So the command:

```
> summary(aov.devel.2)
```

will produce an ANOVA table with an error variance estimate and F statistics.

Using Half-Normal Plots to Choose a Model

You are usually treading on thin ice if you assume that higher-order interactions are zero, unless you have extensive first-hand knowledge of the process you are studying with a 2^k design. When you are not sure whether or not higher-order interactions are zero, you should use a half-normal quantile-quantile plot to judge which effects, including interactions of any order, are significant. Use the function `qqnorm` as follows to produce a half-normal plot on which you can identify points:

```
> qqnorm(aov.devel, label=6)
```

The resulting figure, with six points labeled, is shown in Figure 15.17.

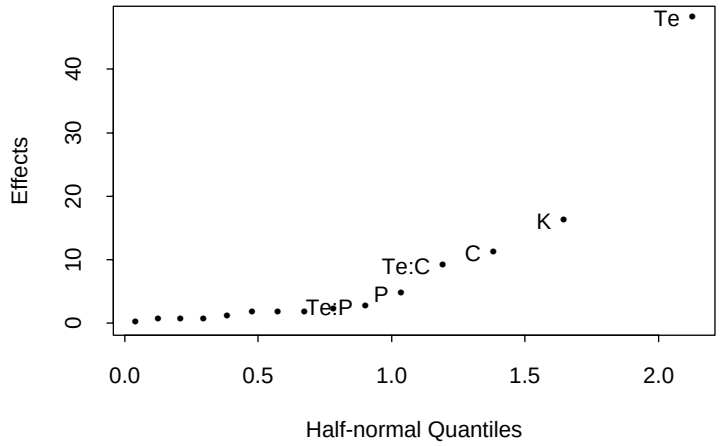

Figure 15.17: *Half-normal plot for product development experiment.*

In general, there are 2^k - 1 points in the half-normal plot, since there are 2^k effects and the estimate of the overall mean is not included in this plot. The *y*-axis positions of the labeled points are the absolute values of the estimated effects. The messages you get from this plot are: You judge the effects for temperature, catalyst, concentration, and temperature by concentration to be *clearly* nonzero. The effect for Pressure is also very likely nonzero. You can examine the marginal effects better by creating a plot with a smaller *y*-range:

```
> qqnorm(aov.devel, label=6, ylim=c(0,20))
```

A full qq-plot of the effects can give you somewhat more information. To get this type of plot, use:

```
> qqnorm(aov.devel, full=T, label=6)
```

Having determined from the half-normal plot which effects are nonzero, now fit a model having terms for the main effects plus the interaction between temperature and concentration:

```
> aov.devel.small <- aov(conversion ~ K+P+Te*C, devel.df)
```

You can now get an ANOVA summary, including an error variance estimate:

```
> summary(aov.devel.small)

            Df Sum of Sq  Mean Sq  F Value       Pr(F)
K            1    256.00  256.000  136.533 0.000000375
P            1     20.25   20.250   10.800 0.008200654
Te           1   2304.00 2304.000 1228.800 0.000000000
C            1    121.00  121.000   64.533 0.000011354
Te:C         1     81.00   81.000   43.200 0.000062906
Residuals 10      18.75    1.875
```

Diagnostic Plots

Once you have tentatively identified a model for a 2^k experiment, you should make the usual graphical checks based on the residuals and fitted values. In the product development example, you should examine the following plots:

```
> hist(resid(aov.devel.small))
> qqnorm(resid(aov.devel.small))
> plot(fitted(aov.devel.small),resid(aov.devel.small))
```

The latter two plots are shown in Figure 15.18 and Figure 15.19.

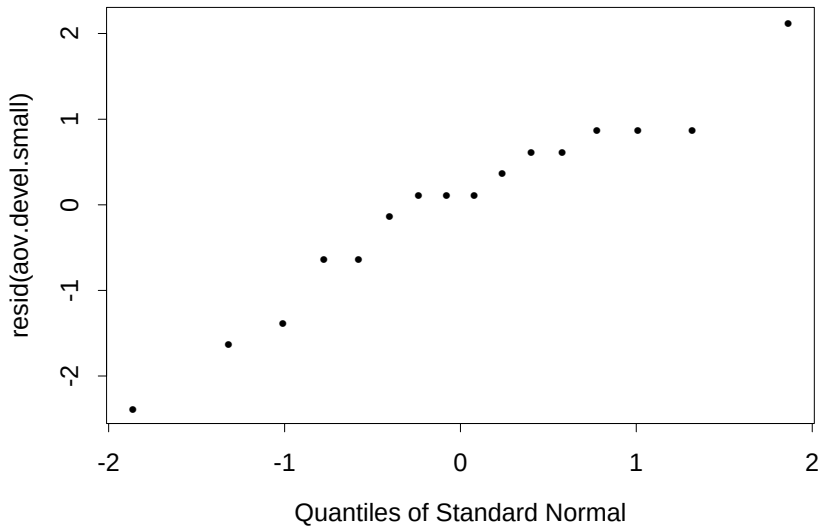

Figure 15.18: *Quantile-quantile plot of residuals, product development example.*

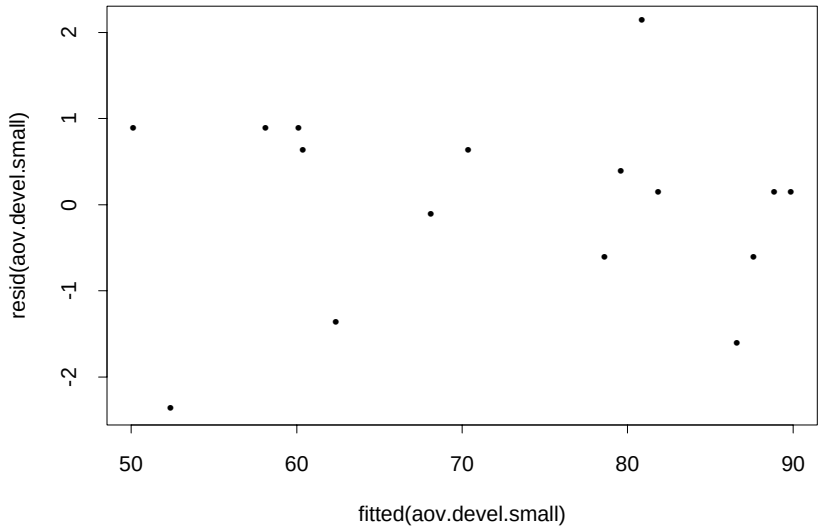

Figure 15.19: *Fitted values vs. residuals, product development example.*

You should also make plots using the time order of the runs:

```
> run.ord <- scan()

1: 8 2 10 4 15 9 1 13 16 5 11 14 3 12 6 7
17:

> plot(run.ord, resid(aov.devel.small))
> plot(run.ord, fitted(aov.devel.small))
```

This gives a slight hint that the first runs were more variable than the latter runs.

Details

The function `aov` returns, by default, coefficients corresponding to the following "usual" ANOVA form for the η:

$$\begin{aligned}\eta = \eta_{i_1 \ldots i_k} = \mu &+ \alpha^{1}_{i_1} + \alpha^{2}_{i_2} + \ldots + \alpha^{k}_{i_k} \\ &+ \alpha^{12}_{i_1 i_2} + \alpha^{13}_{i_1 i_3} + \ldots + \alpha^{k-1,\ k}_{i_{k-1} i_k} \\ &+ \ldots \\ &+ \alpha^{123 \ldots k}_{i_1 i_2 \ldots i_k}\end{aligned}$$

In this form of the 2^k model, each i_m takes on just two values, 1 and 2. There are 2^k values of the k-tuple index $i_1, i_2, ..., i_k$. The parameter μ is the overall mean. The parameters $\alpha^m_{i_m}$, $m = 1, \ldots, k$ correspond to the *main effects*. The parameters $\alpha^{mn}_{i_m i_n}$ correspond to the *two-factor interactions*, the parameters $\alpha^{lmn}_{i_l i_m i_n}$ correspond to the *three-factor interactions*, and the remaining coefficients are the higher-order interactions. The coefficients for the main effects satisfy the constraint $\alpha^i_1 + \alpha^i_2 = 0$, $i = 1, \ldots, k$. All higher-order interactions satisfy the constraint that the sum over any individual subscript index is zero, for example, $\alpha^{12}_{i_1 1} + \alpha^{12}_{i_1 2} = 0$, $\alpha^{124}_{1 i_2 i_4} + \alpha^{124}_{2 i_2 i_4} = 0$, etc.

Because of the constraints on the parameters in this form of the model, it suffices to specify one of the two values for each effect. The function `aov` returns estimates for the "high" levels, for example, $\hat{\alpha}^i_2$, $\hat{\alpha}^{12}_2$.

An estimated effect (in the sense usually used in 2^k models) is equal to the difference between the estimate at the high level minus the estimate at the low level for the ANOVA model form given above:

$$\hat{\alpha}^1 = \hat{\alpha}^1_2 - \hat{\alpha}^1_1$$

and since

$$\hat{\alpha}^1_1 + \hat{\alpha}^1_2 = 0,$$

we have

$$\hat{\alpha}^1 = 2\hat{\alpha}^1_2.$$

REFERENCES

Box, G.E.P. and Hunter, W.G. and Hunter, J.S. (1978). *Statistics for Experimenters: An Introduction to Design, Data Analysis.* John Wiley, New York.

Box, G.E.P. (1988). *Signal-to-noise ratios, performance criteria, and transformations.* Technometrics, 30:1-17.

Carroll, R.J. and Ruppert, D. (1988). *Transformation and Weighting in Regression.* Chapman and Hall, New York.

Chambers, J.M. and Hastie, T.J. (1992). *Statistical Models in S.* Wadsworth and Brooks Cole Advanced Books and Software, Pacific Grove, CA.

Davidian, M. and Haaland, P.D. (1990). *Regression and calibration with non-constant error variance.* Chemometrics and Intelligent Laboratory Systems; 9:231-248.

Haaland, P. (1989). *Experimental Design in Biotechnology.* Marcel Dekker, New York.

Hoaglin, D.C. and Mosteller, F. and Tukey, J.W. (1983). *Understanding Robust and Exploratory Data Analysis.* John Wiley, New York.

Weisberg, S. (1985). *Applied Linear Regression*, 2nd edition. John Wiley, New York.

FURTHER TOPICS IN ANALYSIS OF VARIANCE

16

INTRODUCTION

Chapter 15, Designed Experiments and Analysis of Variance, describes the basic techniques for using S-PLUS for analysis of variance. This chapter extends the concepts to several related topics:

- Multivariate analysis of variance (MANOVA)
- Split-plot designs
- Repeated measures
- Nonparametric tests for one-way and blocked two-way designs
- Variance components models

These topics are preceded by a discussion of model coefficients and contrasts. This information is important in interpreting the available ANOVA summaries.

MODEL COEFFICIENTS AND CONTRASTS

This section explains what the coefficients mean in ANOVA models, and how to get more meaningful coefficients for particular cases.

Suppose we have 5 measurements of a response variable `scores` for each of three treatments, `"A"`, `"B"`, and `"C"`, as shown below:

```
> scores

[1] 4 5 4 5 4 10 7 7 7 7 7 7 8 7 6

> scores.treat

[1] A A A A A B B B B B C C C C C
```

In solving the basic ANOVA problem, we are trying to solve the following simple system of equations:

$$\begin{aligned}\hat{\mu}_A &= \hat{\mu} + \hat{\alpha}_A \\ \hat{\mu}_B &= \hat{\mu} + \hat{\alpha}_B \\ \hat{\mu}_C &= \hat{\mu} + \hat{\alpha}_C\end{aligned}$$

The sample means $\hat{\mu}_A$, $\hat{\mu}_B$, and $\hat{\mu}_C$ can be calculated directly from the data:

$$\begin{aligned}\hat{\mu}_A &= (4 + 5 + 4 + 5 + 4)/\ 5 = 4.4 \\ \hat{\mu}_B &= (10 + 7 + 7 + 7 + 7)/\ 5 = 7.6 \\ \hat{\mu}_C &= (7 + 7 + 8 + 7 + 6)/\ 5 = 7.0\end{aligned}$$

This leaves the following three equations in four unknowns:

$$\begin{aligned}4.4 &= \hat{\mu} + \hat{\alpha}_A \\ 7.6 &= \hat{\mu} + \hat{\alpha}_B \\ 7.0 &= \hat{\mu} + \hat{\alpha}_C\end{aligned}$$

Like all ANOVA models, this system is *overparametrized*, meaning there are more coefficients than can be estimated. We can, however, replace the three variables $\hat{\alpha}_A$, $\hat{\alpha}_B$, and $\hat{\alpha}_C$ with a pair of variables $\hat{\beta}_1$ and $\hat{\beta}_2$ that are functionally independent linear combinations of the original variables, and also independent of $\hat{\mu}$. Such a replacement can be done in more than one way. For unordered factors such as `scores.treat`, the default choice in S-PLUS is the set of *Helmert contrasts* :

$$\hat{\beta}_1 = -\hat{\alpha}_A + \hat{\alpha}_B$$
$$\hat{\beta}_2 = 2\hat{\alpha}_C - (\hat{\alpha}_A + \hat{\alpha}_B)$$

These contrasts, in effect, contrast the ith level with the average of the preceding levels.

More generally, if you have variables α_i, $i = 1, \ldots, k$, you can reparametrize with the k - 1 variables β_j as follows:

$$\beta_j = j\alpha_{j+1} + \sum_{i=1}^{j} \alpha_i \tag{16.1}$$

The transpose of the matrix of coefficients for Equation (16.1) is the following k 3 (k - 1) *contrast matrix*:

$$A = \begin{bmatrix} -1 & -1 & -1 & \ldots & -1 \\ 1 & -1 & -1 & \ldots & -1 \\ 0 & 2 & -1 & \ldots & -1 \\ : & : & : & : & : \\ 0 & 0 & 0 & \ldots & k-1 \end{bmatrix} \tag{16.2}$$

You can recover the original treatment effects α from the reparametrized variables β by matrix multiplication as follows:

$$A\beta = \alpha$$

Returning to our simple example, we can rewrite our original variables in terms of $\hat{\beta}_1$ and $\hat{\beta}_2$ as follows:

$$\begin{aligned}\hat{\alpha}_A &= -\hat{\beta}_1 - \hat{\beta}_2 \\ \hat{\alpha}_B &= \hat{\beta}_1 - \hat{\beta}_2 \\ \hat{\alpha}_C &= 2\hat{\beta}_2\end{aligned}$$

S-PLUS now solves the following system of equations:

$$\begin{aligned}4.4 &= \hat{\mu} - \hat{\beta}_1 - \hat{\beta}_2 \\ 7.6 &= \hat{\mu} + \hat{\beta}_1 - \hat{\beta}_2 \\ 7.0 &= \hat{\mu} + 2\hat{\beta}_2\end{aligned}$$

If we use `aov` as usual to create the `aov` object `scores.aov`, we can use the `coef` function to look at the solved values $\hat{\mu}$, $\hat{\beta}_1$, and $\hat{\beta}_2$:

```
> scores.aov <- aov(scores ~ scores.treat)
> coef(scores.aov)

(Intercept) scores.treat1 scores.treat2
   6.333333           1.6     0.3333333
```

In our example, the contrast matrix is as follows:

$$\begin{pmatrix} -1 & -1 \\ 1 & -1 \\ 0 & 2 \end{pmatrix}$$

You can obtain the contrast matrix for any factor object using the `contrasts` function. For unordered factors such as `scores.treat`, `contrasts` returns the Helmert contrast matrix of the appropriate size:

```
> contrasts(scores.treat)

  [,1] [,2]
A   -1   -1
B    1   -1
C    0    2
```

The contrast matrix, together with the treatment coefficients returned by `coef`, provides an alternative to using `model.tables` to calculate effects:

```
> contrasts(scores.treat) %*% coef(scores.aov)[-1]

        [,1]
A -1.9333333
B  1.2666667
C  0.6666667
```

For *ordered* factors, the Helmert contrasts are replaced, by default, with *polynomial contrasts* that model the response as a polynomial through equally spaced points. For example, suppose we define an ordered factor `water.temp` as follows:

```
> water.temp <- ordered(c(65, 95, 120))
> water.temp

[1] 65  95  120
 65 < 95 < 120
```

The contrast matrix for `water.temp` uses polynomial contrasts:

```
> contrasts(water.temp)

             .L         .Q
 65 -0.7071068  0.4082483
 95  0.0000000 -0.8164966
120  0.7071068  0.4082483
```

For the polynomial contrasts, $\hat{\beta}_1$ represents the linear component of the response, $\hat{\beta}_2$ represents the quadratic component, and so on. When examining ANOVA summaries, you can split a factor's effects into contrast terms to examine each component's contribution to the model. See the section Splitting Treatment Sums of Squares Into Contrast Terms on page 558 for complete details.

At times it is desirable to give particular contrasts to some of the coefficients. In our example, you might be interested in a contrast that has A equal to a weighted average of B and C. This might occur, for instance, if the treatments were really doses. You can add a contrast attribute to the factor using the assignment form of the `contrasts` function:

```
> contrasts(scores.treat) <- c(4,-1,-3)
> contrasts(scores.treat)

  [,1]       [,2]
A    4  0.2264554
B   -1 -0.7925939
C   -3  0.5661385
```

Note that a second contrast was automatically added.

Refitting the model, we now get different coefficients (but the fit remains the same).

```
> scores.aov2 <- aov(scores ~ scores.treat)
> coef(scores.aov2)

 (Intercept) scores.treat1 scores.treat2
    6.333333    -0.4230769      -1.06434
```

More details on working with contrasts can be found in the section Contrasts: The Coding of Factors in Chapter 2.

SUMMARIZING ANOVA RESULTS

Results from an analysis of variance are typically displayed in an *analysis of variance table*, which shows a decomposition of the variation in the response: the total sum of squares of the response is split into sums of squares for each treatment and interaction and a residual sum of squares. You can obtain the ANOVA table, as we have throughout this chapter, by using `summary` on the result of a call to `aov`, such as this overly simple model for the `wafer` data:

```
> attach(wafer)
> wafer.aov <- aov( pre.mean ~ visc.tem + devtime +
+ etchtime)
> summary(wafer.aov)

           Df Sum of Sq   Mean Sq  F Value     Pr(F)
visc.tem    2  1.343361 0.6716807 3.678485 0.0598073
devtime     2  0.280239 0.1401194 0.767369 0.4875574
etchtime    2  0.103323 0.0516617 0.282927 0.7588959
Residuals 11  2.008568 0.1825971
```

Splitting Treatment Sums of Squares Into Contrast Terms

Each treatment sum of squares in the ANOVA table can be further split into terms corresponding to the treatment contrasts. By default, the Helmert contrasts are used for unordered factors and polynomial contrasts for ordered factors. For instance, with ordered factors you can assess whether the response is fairly linear in the factor by listing the polynomial contrasts separately. In the dataset `wafer`, you can examine the linear and quadratic contrasts of `devtime` and `etchtime` by using the `split` argument to the `summary` function:

```
> summary(wafer.aov, split = list(etchtime =
+ list(L = 1, Q = 2),
+ devtime = list(L = 1, Q = 2)))

               Df Sum of Sq   Mean Sq  F Value     Pr(F)
visc.tem        2  1.343361 0.6716807 3.678485 0.0598073
devtime         2  0.280239 0.1401194 0.767369 0.4875574
  devtime: L    1  0.220865 0.2208653 1.209577 0.2949025
  devtime: Q    1  0.059373 0.0593734 0.325161 0.5799830
```

```
etchtime       2  0.103323 0.0516617 0.282927 0.7588959
  etchtime: L  1  0.094519 0.0945188 0.517636 0.4868567
  etchtime: Q  1  0.008805 0.0088047 0.048219 0.8302131
Residuals     11  2.008568 0.1825971
```

Each of the (indented) split terms sum to their overall sum of squares.

The `split` argument can evaluate only the effects of the contrasts used to specify the ANOVA model: if you wish to test a specific contrast, you need to set it explicitly before fitting the model. Thus, if you want to test a polynomial contrast for an unordered factor, you must specify polynomial contrasts for the factor before fitting the model. The same is true for other nondefault contrasts. For instance, the variable `visc.tem` in the `wafer` dataset is a three-level factor constructed by combining two levels of viscosity (204 and 206) with two levels of temperature (90 and 105).

```
> levels(visc.tem)

[1] "204,90"  "206,90"  "204,105"
```

To assess viscosity, supposing temperature has no effect, we define a contrast that takes the difference of the middle and the sum of the first and third levels of `visc.tem`; the contrast matrix is automatically completed:

```
> contrasts(visc.tem) <- c(-1,2,-1)
> contrasts(visc.tem)

        [,1]          [,2]
204,90    -1 -7.071068e-01
206,90     2 -1.110223e-16
204,105   -1  7.071068e-01

> wafer.aov <- aov( pre.mean ~ visc.tem +
+ devtime + etchtime)
```

In this fitted model, the first contrast for `visc.aov` reflects the effect of viscosity:

```
> summary(wafer.aov, split = list(visc.tem =
+ list(visc = 1)))
```

```
                  Df Sum of Sq  Mean Sq  F Value     Pr(F)
visc.tem           2  1.343361 0.671681 3.678485 0.0598073
  visc.tem: visc   1  1.326336 1.326336 7.263730 0.0208372
devtime            2  0.280239 0.140119 0.767369 0.4875574
etchtime           2  0.103323 0.051662 0.282927 0.7588959
Residuals         11  2.008568 0.182597
```

Treatment Means and Standard Errors

Commonly the ANOVA model is written in the form "grand mean plus treatment effects,"

$$y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk}$$

The treatment effects, α_i, β_j, and $(\alpha\beta)_{ij}$, reflect changes in the response due to that combination of treatments. In this parameterization, the effects are constrained, usually to sum to zero.

Unfortunately, the use of the term "effect" in ANOVA is not standardized: in factorial experiments an effect is the difference between treatment levels, in balanced designs it is the difference from the grand mean, and in unbalanced designs there are (at least) two different standardizations that make sense.

The coefficients of an `aov` object returned by `coef(aov.object)` are coefficients for the contrast variables derived by the `aov` function, rather than the grand-mean-plus-effects decomposition. The functions `dummy.coef` and `model.tables` translate the internal coefficients into the more natural treatment effects.

Balanced Designs

In a balanced design, both computing and interpreting effects are straightforward:

```
> gun.aov <- aov(Rounds ~ Method + Physique/Team, gun)
> coef(gun.aov)

 (Intercept)    Method Physique.L  Physique.Q
    19.33333 -4.255556  -1.154941 -0.06123724
 PhysiqueSTeam1 PhysiqueATeam1 PhysiqueHTeam1
         1.9375           0.45          -0.45
 PhysiqueSTeam2 PhysiqueATeam2 PhysiqueHTeam2
        -0.4875    0.008333333     -0.1083333
```

The `dummy.coef` function translates the coefficients into the more natural effects:

```
> dummy.coef(gun.aov)

$"(Intercept)":
 (Intercept)
    19.33333

$Method:
       M1        M2
 4.255556 -4.255556

$Physique:
[1]  0.7916667  0.0500000 -0.8416667

$"Team %in% Physique":
   1T1        2T1       3T1   1T2       2T2
 -1.45 -0.4583333 0.5583333 2.425 0.4416667
       3T2    1T3        2T3        3T3
 -0.3416667 -0.975 0.01666667 -0.2166667
```

For the default contrasts, these effects always sum to zero.

The same information is returned in a tabulated form by `model.tables`. Note that `model.tables` calls `proj`; hence, it is helpful to use `qr = T` in the call to `aov`.

```
> model.tables(gun.aov, se = T)

Refitting model to allow projection
Tables of effects

 Method
    M1     M2
 4.256 -4.256

 Physique
     S    A       H
 0.7917 0.05 -0.8417

 Team %in% Physique
Dim 1 : Physique
Dim 2 : Team
```

```
       T1     T2     T3
S -1.450  2.425 -0.975
A -0.458  0.442  0.017
H  0.558 -0.342 -0.217

Standard errors of effects
     Method Physique Team %in% Physique
     0.3381   0.4141             0.7172
rep 18.0000  12.0000             4.0000
```

Using the first method, the gunners fired on average 4.26 more rounds than the overall mean. The standard errors for the effects are simply the residual standard error scaled by the replication factor, `rep`, the number of observations at each level of the treatment. For instance, the standard error for the `Method` effect is:

$$\text{se(Method)} = \frac{\text{se(Residual)}}{\sqrt{\text{replication(Method)}}} = \frac{1.434}{\sqrt{18}} = 0.3381$$

The `model.tables` function also computes cell means for each of the treatments. This provides a useful summary of the analysis that is more easily related to the original data.

```
> model.tables(gun.aov, type = "means", se = T)

Tables of means

Grand mean
 19.33

 Method
    M1    M2
 23.59 15.08

 Physique
     S     A     H
 20.13 19.38 18.49

 Team %in% Physique
Dim 1 : Physique
Dim 2 : Team
```

```
      T1    T2    T3
S 18.68 22.55 19.15
A 18.93 19.83 19.40
H 19.05 18.15 18.28
Standard errors for differences of means
     Method Physique Team %in% Physique
     0.4782   0.5856              1.014
rep 18.0000  12.0000              4.000
```

The first method had an average firing rate of 23.6 rounds. For the tables of means, standard errors of *differences* between means are given, as these are usually of most interest to the experimenter. For instance the standard error of differences for `Team %in% Physique` is:

$$\text{SED} = \sqrt{2 \times \frac{2.0576}{4}} = 1.014$$

To gauge the statistical significance of the difference between the first and second small physique teams, we can compute the "least significant difference (LSD)" for the `Team %in% Physique` interaction. The validity of the statistical significance is based on the assumption that the model is correct and the residuals are Gaussian. The plots of the residuals indicate these are not unreasonable assumptions for this dataset–you can verify this by creating a histogram and normal qq-plot of the residuals as follows:

```
> hist(resid(gun.aov))
> qqnorm(resid(gun.aov))
```

The LSD at the 95% level is:

$$t(0.975,\ 26) \times \text{SED(Team \%*\% Physique)}$$

We use the *t*-distribution with 26 degrees of freedom because the residual sum of squares has 26 degrees of freedom. In S-PLUS, we type the following:

```
> qt(0.975, 26) * 1.014

[1] 2.084307
```

Since the means of the two teams differ by more than 2.08, the teams are different at the 95% level of significance. From an interaction plot it is clear that the results for teams of small physique are unusually high.

2^k Factorial Designs

In factorial experiments, where each experimental treatment has only two levels, a treatment *effect* is, by convention, the difference between the high and low levels. Interaction effects are half the average difference between paired levels of an interaction. These *factorial effects* are computed when `type = "feffects"` is used in the `model.tables` function:

```
> catalyst.aov <- aov( Yield ~ ., catalyst, qr = T)
> model.tables(catalyst.aov, type = "feffects", se = T)

Table of factorial effects
     Effects    se
Temp    23.0 5.062
Conc    -5.0 5.062
Cat      1.5 5.062
```

Unbalanced Designs

When designs are unbalanced (there are unequal numbers of observations in some cells of the experiment), the effects associated with different treatment levels can be standardized in different ways. For instance, suppose we use only the first 35 observations of the `gun` data set:

```
> gunsmall.aov <- aov(Rounds ~ Method +
+ Physique/Team, gun, subset=1:35, qr = T)
```

The `dummy.coef` function standardizes treatment effects to sum to zero:

```
> dummy.coef(gunsmall.aov)

$"(Intercept)":
 (Intercept)
    19.29177

$Method:
       M1        M2
 4.297115 -4.297115
```

```
$Physique:
[1]  0.83322650  0.09155983 -0.92478632

$"Team %in% Physique":
   1T1        2T1       3T1   1T2       2T2
 -1.45 -0.4583333 0.6830128 2.425 0.4416667

       3T2    1T3        2T3        3T3
 -0.2169872 -0.975 0.01666667  -0.466025
```

The `model.tables` function computes effects that are standardized so the weighted effects sum to zero:

$$\sum_{i=1}^{T} n_i \tau_i = 0,$$

where n_i is the replication of level i and τ_i the effect. The `model.tables` effects are identical to the values of the projection vectors computed by `proj(gunsmall.aov)`:

```
> model.tables(gunsmall.aov)

Tables of effects

 Method
        M1      M2
     4.135  -4.378
rep 18.000  17.000

 Physique
          S        A        H
     0.7923  0.05065  -0.9196
rep 12.0000 12.00000  11.0000

 Team %in% Physique
Dim 1 : Physique
Dim 2 : Team
        T1     T2     T3
S   -1.450  2.425 -0.975
rep  4.000  4.000  4.000
```

```
A    -0.458  0.442  0.017
rep   4.000  4.000  4.000
H     0.639 -0.261 -0.505
rep   4.000  4.000  3.000
```

With this standardization, treatment effects are orthogonal: consequently cell means can be computed by simply adding effects to the grand mean; standard errors are also more readily computed.

```
> model.tables(gunsmall.aov, type="means", se=T)

Standard error information not returned as design is
unbalanced.
Standard errors can be obtained through se.contrast.
Tables of means
Grand mean

 19.45

 Method
       M1    M2
    23.59 15.08
rep 18.00 17.00

 Physique
        S    A     H
    20.25 19.5 18.53
rep 12.00 12.0 11.00

 Team %in% Physique
Dim 1 : Physique
Dim 2 : Team
       T1    T2    T3
S   18.80 22.67 19.27
rep  4.00  4.00  4.00
A   19.05 19.95 19.52
rep  4.00  4.00  4.00
H   19.17 18.27 18.04
rep  4.00  4.00  3.00
```

Note that the `(Intercept)` value returned by `dummy.coef` is not the grand mean of the data, and the coefficients returned are not a decomposition of the cell means. This is a difference that occurs only

with unbalanced designs: in balanced designs the functions `dummy.coef` and `model.tables` return identical values for the effects.

In the unbalanced case, the standard errors for comparing two means depend on the replication factors, hence it could be very complex to tabulate all combinations. Instead, they can be computed directly with `se.contrast`. For instance, to compare the first and third teams of heavy physique:

```
> se.contrast(gunsmall.aov, contrast = list(
+ Physique=="S"  Team == "T1",
+ Physique=="S"  Team == "T3"),
+ data = gun[1:35,])

[1] 1.018648
```

By default, the standard error of the difference of the means specified by `contrast` is computed. Other contrasts are specified by the argument `coef`. For instance, to compute the standard error of the contrast tested in the section Splitting Treatment Sums of Squares Into Contrast Terms on page 558 for the variable `visc.tem`:

```
> attach(wafer)
> se.contrast(wafer.aov, contrast = list(
+ visc.tem ==levels(visc.tem)[1],
+ visc.tem == levels(visc.tem)[2],
+ visc.tem == levels(visc.tem)[3]),
+ coef = c(-1,2,-1), data = wafer)

Refitting model to allow projection
[1] 0.07793052
```

The value of the contrast can be computed from `model.tables(wafer.aov)`. The effects for `visc.tem` are:

```
visc.tem
204,90  206,90 204,105
0.1543 -0.3839  0.2296
```

The contrast is `-0.3839 - mean(c(0.1543, 0.2296)) = -0.5758`. The standard error for testing whether the contrast is zero is 0.0779; clearly, the contrast is nonzero.

Type III Sums of Squares and Adjusted Means

Researchers implementing an experimental design frequently lose experimental units and find themselves with unbalanced, but complete, data. The data is unbalanced in that the number of replications is not constant for each treatment combination; the data is complete in that at least one experimental unit exists for each treatment combination. In this type of circumstance, an experimenter may find the hypotheses tested by Type III sum of squares are of more interest than those tested by Type I (sequential) sum of squares, and the adjusted means of more interest than unadjusted means. New options to the `lm` and `aov` object methods, `anova.lm`, `summary.aov`, and `model.tables.aov` will give the Type III sum of squares and the adjusted (marginal) means. For `anova` and `summary`, the new argument `ssType` can be 1 or 3, with `ssType = 1` as the default; `model.tables` has the new option "`adj.means`", for the existing argument `type`. An example is given to demonstrate the new capabilities of these in an analysis of a designed experiment.

The fat-surfactant example is taken from Milliken and Johnson (1984, p. 166), where they analyze an unbalanced randomized block factorial design. Here, the specific volume of bread loaves baked from dough mixed from each of nine Fat and Surfactant treatment combinations is measured. The experimenters blocked on four flour types. Ten loaves had to be removed from the experiment, but at least one loaf existed for each Fat x Surfactant combination and all marginal means are estimable so the Type III hypotheses are testable.

The overparameterized model is:

$$\mu_{ijk} = \mu + b_i + f_j + s_k + (fs)_{jk}$$

for $i = 1, .., 4$, $j = 1, 2, 3$, and $k = 1, 2, 3$. Because the data are unbalanced the Type III sum of squares for Flour, Fat and Surfactant test a more useful hypothesis than the Type I. Specifically, the Type III hypotheses are that the marginal means are equal:

$$H_{\text{Flour}}: \bar{\mu}_{1..} = \bar{\mu}_{2..} = \bar{\mu}_{3..} = \bar{\mu}_{4..}$$

$$H_{\text{Fat}}: \bar{\mu}_{.1.} = \bar{\mu}_{.2.} = \bar{\mu}_{.3.}$$

$$H_{\text{Surfactant}}: \bar{\mu}_{..1} = \bar{\mu}_{..2} = \bar{\mu}_{..3}$$

where

$$\bar{\mu}_{i..} = \frac{\sum_{j,\ k} \mu_{ikj}}{3 \cdot 3}$$

$$\bar{\mu}_{.j.} = \frac{\sum_{i,\ k} \mu_{ijk}}{4 \cdot 3}$$

$$\bar{\mu}_{..k} = \frac{\sum_{i,\ j} \mu_{ijk}}{4 \cdot 3}$$

The hypotheses tested by the Type I sum of squares are not easily interpreted since they are dependent on the order each term is specified the formula and involve the cell replications (which can be viewed as random variables when there are random drop-outs). Moreover, the hypothesis tested by the blocking term, Flour, involves parameters of the Fat, Flour, and Fat x Flour terms.

ANOVA Tables

The ANOVA tables for both Type I and Type III sum of squares are given below for comparison. Using the Type III sum of squares we see that the block effect, Flour, is significant as is Fat, but Surfactant is not at, say, a test size of $\alpha = 0.05$. However, in the presence of a significant interaction, the test of the marginal means probably has little meaning for Fat and Surfactant.

```
> Baking.aov<-aov(Specific.Vol ~ Flour + Fat * Surfactant,
+ data = Baking, contrasts=list(Flour=contr.sum(4),
+ Fat=contr.sum(3),Surfactant=contr.sum(3)) )
```

```
> anova(Baking.aov)

Analysis of Variance Table
Response: Specific.Vol
Terms added sequentially (first to last)
                Df Sum of Sq  Mean Sq  F Value     Pr(F)
         Flour   3   6.39310 2.131033 12.88269 0.0002587
           Fat   2  10.33042 5.165208 31.22514 0.0000069
    Surfactant   2   0.15725 0.078625  0.47531 0.6313678
Fat:Surfactant   4   5.63876 1.409691  8.52198 0.0010569
     Residuals  14   2.31586 0.165418

> anova(Baking.aov,ssType=3)

Analysis of Variance Table
Response: Specific.Vol
Type III Sum of Squares
                Df Sum.of.Sq  Mean.Sq  F.Value      Pr.F.
         Flour   3   8.69081 2.896937 17.51280 0.00005181
           Fat   2  10.11785 5.058925 30.58263 0.00000778
    Surfactant   2   0.99721 0.498605  3.01421 0.08153989
Fat:Surfactant   4   5.63876 1.409691  8.52198 0.00105692
     Residuals  14   2.31586 0.165418
```

Adjusted Means The adjusted (marginal) means given below estimate the means given in the Type III hypotheses for Flour, Fat, and Surfactant. The means for Flour x Surfactant for the overparameterized model are

$$\bar{\mu}_{.jk} = \frac{\sum_i \mu_{ijk}}{4}$$

Interestingly, these means are still estimable even though not all Flour x Surfactant x Flour combinations were observed.

```
> model.tables(Baking.aov,type="adj.means")

Tables of adjusted means
Grand mean
    6.633281
se  0.084599
 N 26.000000
 Flour
         1      2      3      4
    7.3020 5.7073 6.9815 6.5423
 se 0.1995 0.1467 0.1621 0.1785
rep 5.0000 8.0000 7.0000 6.0000
 Fat
         1      2      3
    5.8502 6.5771 7.4725
 se 0.1365 0.1477 0.1565
rep 9.0000 9.0000 8.0000
 Surfactant
         1      2      3
    6.3960 6.5999 6.9039
 se 0.1502 0.1432 0.1473
rep 8.0000 9.0000 9.0000
Fat:Surfactant
Dim 1 : Fat
Dim 2 : Surfactant
         1      2      3
  1 5.5364 5.8913 6.1229
 se 0.2404 0.2392 0.2414
rep 3.0000 3.0000 3.0000
  2 7.0229 6.7085 6.0000
 se 0.2414 0.3006 0.2034
rep 3.0000 2.0000 4.0000
  3 6.6286 7.2000 8.5889
 se 0.3007 0.2034 0.3001
rep 2.0000 4.0000 2.0000
```

Multiple Comparisons

The F-statistic for the Fat x Surfactant interaction in the Type III ANOVA table is significant so the tests for the marginal means for Fat and Surfactant have little meaning. We can, however, use `multicomp` to find all pairwise comparisons of the mean Fat levels for each level of Surfactant, and those for Surfactant for each level of Fat.

```
> multicomp(Baking.aov,focus="Fat",
+ adjust=list(Surfactant=seq(3)))

95 % simultaneous confidence intervals for specified
linear combinations, by the Sidak method
critical point: 3.2117
response variable: Flour
intervals excluding 0 are flagged by '****'
            Estimate Std.Error Lower Bound Upper Bound
1.adj1-2.adj1   -1.490     0.344      -2.590      -0.381 ****
1.adj1-3.adj1   -1.090     0.377      -2.300       0.120
2.adj1-3.adj1    0.394     0.394      -0.872       1.660
1.adj2-2.adj2   -0.817     0.390      -2.070       0.434
1.adj2-3.adj2   -1.310     0.314      -2.320      -0.300 ****
2.adj2-3.adj2   -0.492     0.363      -1.660       0.674
1.adj3-2.adj3    0.123     0.316      -0.891       1.140
1.adj3-3.adj3   -2.470     0.378      -3.680      -1.250 ****
2.adj3-3.adj3   -2.590     0.363      -3.750      -1.420 ****

> multicomp(Baking.aov,focus="Surfactant",
+ adjust=list(Fat=seq(3)))

95 % simultaneous confidence intervals for specified linear
combinations, by the Sidak method
critical point: 3.2117
response variable: Flour
intervals excluding 0 are flagged by '****'
            Estimate Std.Error Lower Bound Upper Bound
1.adj1-2.adj1   -0.355     0.341    -1.45000       0.740
1.adj1-3.adj1   -0.587     0.344    -1.69000       0.519
2.adj1-3.adj1   -0.232     0.342    -1.33000       0.868
1.adj2-2.adj2    0.314     0.377    -0.89700       1.530
1.adj2-3.adj2    1.020     0.316     0.00922       2.040 ****
2.adj2-3.adj2    0.708     0.363    -0.45700       1.870
1.adj3-2.adj3   -0.571     0.363    -1.74000       0.594
1.adj3-3.adj3   -1.960     0.427    -3.33000      -0.590 ****
2.adj3-3.adj3   -1.390     0.363    -2.55000      -0.225 ****
```

The levels for Fat and Surfactant factors are both labeled 1, 2, and 3 so the row labels in the `multicomp` tables require explanation. For the first table, the label `1.adj1-2.adj1` refers to the difference between levels 1 and 2 of Fat (the `focus` variable) at level 1of Surfactant (the `adjust` variable), whereas for the second table it is the difference between levels 1 and 2 of Surfactant at level 1 of Fat. The reader can verify that the table of differences reported by `multicomp` are the differences in the adjusted means for `Fat:Surfactant` reported by `model.tables`. Significant differences are flagged with `'****'`. As a result of the of Surfactant and Fat interaction, the F test for the equivalence of the Surfactant marginal means is not significant, but there exists significant differences between the mean of Surfactant levels 1-3 at a Fat level of 2 and between the means Surfactant levels 1-3 and 2-3 at a Fat level of 3.

Estimable Functions

The Type I and Type III estimable functions for the overparameterized model show the linear combinations of the overparameterized model parameters tested by each sum of squares. The Type I estimable functions can be obtained by performing row reductions on the cross products of the model matrix, X^tX, that reduce it to upper triangular with each nonzero row divided by its diagonal (SAS Technical Report R-101, 1978).

```
> round(L,4)
```

	L2	L3	L4	L6	L7	L9	L10	L12	L13	L15	L16
(Intercept)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0	0	0	0
Flour.1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0	0	0	0
Flour.2	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0	0	0	0
Flour.3	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0	0	0	0
Flour.4	-1.0000	-1.0000	-1.0000	0.0000	0.0000	0.0000	0.0000	0	0	0	0
Fat.1	0.0667	-0.0833	0.0952	1.0000	0.0000	0.0000	0.0000	0	0	0	0
Fat.2	-0.3000	-0.1250	-0.2143	0.0000	1.0000	0.0000	0.0000	0	0	0	0
Fat.3	0.2333	0.2083	0.1190	-1.0000	-1.0000	0.0000	0.0000	0	0	0	0
Surfactant.1	0.2333	0.2083	0.1190	0.1152	0.1338	1.0000	0.0000	0	0	0	0
Surfactant.2	-0.1000	-0.2500	-0.2143	-0.1966	-0.3235	0.0000	1.0000	0	0	0	0
Surfactant.3	-0.1333	0.0417	0.0952	0.0814	0.1896	-1.0000	-1.0000	0	0	0	0
Fat.1:Surfactant.1	0.2000	0.1250	0.1429	0.3531	0.0359	0.3507	0.0037	1	0	0	0
Fat.2:Surfactant.1	-0.1667	-0.0417	-0.0238	-0.0060	0.3250	0.4242	0.0760	0	1	0	0
Fat.3:Surfactant.1	0.2000	0.1250	0.0000	-0.2319	-0.2271	0.2251	-0.0797	-1	-1	0	0
Fat.1:Surfactant.2	0.0333	-0.1667	-0.0238	0.3167	-0.0060	-0.0149	0.3499	0	0	1	0
Fat.2:Surfactant.2	-0.1667	-0.0417	-0.1667	0.0049	0.2034	0.0190	0.2971	0	0	0	1
Fat.3:Surfactant.2	0.0333	-0.0417	-0.0238	-0.5182	-0.5209	-0.0041	0.3530	0	0	-1	-1
Fat.1:Surfactant.3	-0.1667	-0.0417	-0.0238	0.3302	-0.0299	-0.3358	-0.3536	-1	0	-1	0
Fat.2:Surfactant.3	0.0333	-0.0417	-0.0238	0.0011	0.4716	-0.4432	-0.3731	0	-1	0	-1
Fat.3:Surfactant.3	0.0000	0.1250	0.1429	-0.2499	-0.2520	-0.2210	-0.2733	1	1	1	

The columns labeled L2, L3, and L4 are for the Flour hypothesis; L6 and L7 are for the Fat hypothesis; L9 and L10 are for the Surfactant hypothesis; and L12, L13, L15, and L16 are for the Fat x Surfactant hypothesis. In contrast, the Type III estimable functions can be obtained from the generating set $(X'X)^*(X'X)$, where $(X'X)^*$ is the g-2 inverse of the cross product matrix, (Kennedy and Gentle, 1980, p. 396) and perform the steps outlined in the SAS/STAT User's Guide (1990, pp. 120-121) .

```
> round(L3,4)
```

```
                    L2 L3 L4      L6      L7      L9     L10 L12 L13 L15 L16
          (Intercept)  0  0  0  0.0000  0.0000  0.0000  0.0000   0   0   0   0
              Flour.1  1  0  0  0.0000  0.0000  0.0000  0.0000   0   0   0   0
              Flour.2  0  1  0  0.0000  0.0000  0.0000  0.0000   0   0   0   0
              Flour.3  0  0  1  0.0000  0.0000  0.0000  0.0000   0   0   0   0
              Flour.4 -1 -1 -1  0.0000  0.0000  0.0000  0.0000   0   0   0   0
                Fat.1  0  0  0  1.0000  0.0000  0.0000  0.0000   0   0   0   0
                Fat.2  0  0  0  0.0000  1.0000  0.0000  0.0000   0   0   0   0
                Fat.3  0  0  0 -1.0000 -1.0000  0.0000  0.0000   0   0   0   0
        Surfactant.1  0  0  0  0.0000  0.0000  1.0000  0.0000   0   0   0   0
        Surfactant.2  0  0  0  0.0000  0.0000  0.0000  1.0000   0   0   0   0
        Surfactant.3  0  0  0  0.0000  0.0000 -1.0000 -1.0000   0   0   0   0
  Fat.1:Surfactant.1  0  0  0  0.3333  0.0000  0.3333  0.0000   1   0   0   0
  Fat.2:Surfactant.1  0  0  0  0.0000  0.3333  0.3333  0.0000   0   1   0   0
  Fat.3:Surfactant.1  0  0  0 -0.3333 -0.3333  0.3333  0.0000  -1  -1   0   0
  Fat.1:Surfactant.2  0  0  0  0.3333  0.0000  0.0000  0.3333   0   0   1   0
  Fat.2:Surfactant.2  0  0  0  0.0000  0.3333  0.0000  0.3333   0   0   0   1
  Fat.3:Surfactant.2  0  0  0 -0.3333 -0.3333  0.0000  0.3333   0   0  -1  -1
  Fat.1:Surfactant.3  0  0  0  0.3333  0.0000 -0.3333 -0.3333  -1   0  -1   0
  Fat.2:Surfactant.3  0  0  0  0.0000  0.3333 -0.3333 -0.3333   0  -1   0  -1
  Fat.3:Surfactant.3  0  0  0 -0.3333 -0.3333 -0.3333 -0.3333   1   1   1   1
```

Here we see one of the appealing properties of Type III sum of squares: the hypothesis tested by the Type III sum of squares for Flour only involves parameters of the Flour term, whereas the hypothesis tested by the Type I sum of squares for Flour involves the parameters of Fat, Surfactant and Fat x Surfactant.

The marginal means can also be obtained from `multicomp` using `comparisons = "none"`. Doing so, we obtain the estimable functions for the marginal means for the overparameterized model. For example, the estimable functions for the Fat marginal means are:

```
> Fat.mcomp<-multicomp(Baking.aov,focus="Fat",comp="none")
```

```
> round(Fat.mcomp$lmat,4)

                          1      2      3
       (Intercept) 1.0000 1.0000 1.0000
           Flour.1 0.2500 0.2500 0.2500
           Flour.2 0.2500 0.2500 0.2500
           Flour.3 0.2500 0.2500 0.2500
           Flour.4 0.2500 0.2500 0.2500
             Fat.1 1.0000 0.0000 0.0000
             Fat.2 0.0000 1.0000 0.0000
             Fat.3 0.0000 0.0000 1.0000
      Surfactant.1 0.3333 0.3333 0.3333
      Surfactant.2 0.3333 0.3333 0.3333
      Surfactant.3 0.3333 0.3333 0.3333
Fat.1:Surfactant.1 0.3333 0.0000 0.0000
Fat.2:Surfactant.1 0.0000 0.3333 0.0000
Fat.3:Surfactant.1 0.0000 0.0000 0.3333
Fat.1:Surfactant.2 0.3333 0.0000 0.0000
Fat.2:Surfactant.2 0.0000 0.3333 0.0000
Fat.3:Surfactant.2 0.0000 0.0000 0.3333
Fat.1:Surfactant.3 0.3333 0.0000 0.0000
Fat.2:Surfactant.3 0.0000 0.3333 0.0000
Fat.3:Surfactant.3 0.0000 0.0000 0.3333
```

The reader can verify that the Type III estimable functions for Fat are the differences between columns 1 and 3 and between columns 2 and 3.

Sigma Constrained Parameterization

The function `lm` reparameterizes the linear model in an attempt to make the model matrix full column rank. We will next explore the computation of the adjusted means and the Type III sum of squares for Fat using the sigma constrained linear model. The sigma constraints were used in the `aov` fit above (`aov` calls `lm` with `singular.ok = T`). This was done by specifying `contr.sum` in the

`contrasts` argument. In this setting the adjusted means can be computed with the following estimable functions:

```
> L
```

	Fat.1	Fat.2	Fat.3
(Intercept)	1	1	1
Flour1	0	0	0
Flour2	0	0	0
Flour3	0	0	0
Fat1	1	0	-1
Fat2	0	1	-1
Surfactant1	0	0	0
Surfactant2	0	0	0
Fat1Surfactant1	0	0	0
Fat2Surfactant1	0	0	0
Fat1Surfactant2	0	0	0
Fat2Surfactant2	0	0	0

Some justification to these functions may be in order: The parameterization chosen constrains the sum of the level estimates of each effect to zero. That is,

$$\sum_i b_i = \sum_j f_j = \sum_k s_k = \sum_j (fs)_{jk} = \sum_k (fs)_{jk} = 0$$

Therefore, any effect that we are summing over in the mean estimate vanishes. The intercept in the least squares fit estimates μ and the two coefficients for the Fat effect (labeled in `L` as `Fat1` and `Fat2`) estimate *f1* and *f2*, respectively, and *f3* = -*f1* - *f2*.

We can check that each function is, in fact, estimable by ensuring that they are in the row space of X, then compute the adjusted means.

```
> X<-model.matrix(Baking.aov)
> ls.fit<-lsfit(t(X)%*%X,L,intercept=F)
> apply(abs(ls.fit$residuals),2,max)<0.0001

 Fat.1 Fat.2 Fat.3
     T     T     T

> m<-t(L)%*%Baking.aov$coefficients
```

```
> m

          [,1]
Fat.1 5.850197
Fat.2 6.577131
Fat.3 7.472514
```

Now use the `summary` method for the `lm` object to obtain (X^tX) - 1 and $\hat{\sigma}$ and compute the standard errors of the least squares means.

```
> Baking.sum<-summary.lm(Baking.aov)
> Baking.sum$sigma*sqrt(diag(t(L)%*%
+ Baking.sum$cov.unscaled%*%L))

[1] 0.1364894 0.1477127 0.1564843
```

A set of Type III estimable functions for Fat can be obtained using the contrasts generated by `contr.helmert`.

```
> contr.helmert(3)

  [,1] [,2]
1   -1   -1
2    1   -1
3    0    2
```

We will use this set of orthogonal contrasts to test $\bar{\mu}_{.1.} = \bar{\mu}_{.2.}$ and $\bar{\mu}_{.1.} + \bar{\mu}_{.2.} = 2\bar{\mu}_{.3.}$, which is equivalent to H_{Fat}.

```
> L.typeIII<-L%*%contr.helmert(3)
> L.typeIII

                [,1] [,2]
    (Intercept)    0    0
         Flour1    0    0
         Flour2    0    0
         Flour3    0    0
           Fat1   -1   -3
           Fat2    1   -3
    Surfactant1    0    0
    Surfactant2    0    0
Fat1Surfactant1    0    0
```

```
Fat2Surfactant1    0    0
Fat1Surfactant2    0    0
Fat2Surfactant2    0    0
```

Finally, the Type III sum of squares is computed for Fat.

```
> h.m <- t(contr.helmert(3))%*%m
> t(h.m)%*%solve(t(L.typeIII)%*%Baking.sum$cov.unscaled%*%
+ L.typeIII)%*%h.m

         [,1]
[1,] 10.11785
```

Since we used the sigma-constrained model and the data is complete, we can also use `drop1` to obtain the Type III sum of squares.

```
> drop1(Baking.aov,~.)

Single term deletions
Model:
Specific.Vol ~ Flour + Fat * Surfactant
               Df Sum of Sq      RSS  F Value      Pr(F)
        <none>               2.31586
         Flour  3   8.69081 11.00667 17.51280 0.00005181
           Fat  2  10.11785 12.43371 30.58263 0.00000778
    Surfactant  2   0.99721  3.31307  3.01421 0.08153989
Fat:Surfactant  4   5.63876  7.95462  8.52198 0.00105692
```

For the sigma-constrained model, the hypotheses H_{Fat} and $H_{\text{Surfactant}}$ can also be expressed as

$$H^*_{\text{Fat}}: f_1 = f_2 = 0$$

$$H^*_{\text{Surfactant}}: s_1 = s_2 = s_3 = 0$$

The row for Fat in the `drop1` ANOVA table is the reduction in sum of squares due to Fat given all other terms are in the model. This simultaneously tests that the least squares coefficients $\beta_{\text{Fat1}} = f1$ and $\beta_{\text{Fat2}} = f2$ are zero (and, hence $f3 = -(f1 + f2)$ is zero) (Searle, 1987). The same argument applies to Surfactant. It follows that the following Type III estimable functions for Fat can be used to test H^*_{Fat} (or equivalently H_{Fat}).

```
> L.typeIII

                [,1] [,2]
    (Intercept)    0    0
         Flour1    0    0
         Flour2    0    0
         Flour3    0    0
           Fat1    1    0
           Fat2    0    1
    Surfactant1    0    0
    Surfactant2    0    0
Fat1Surfactant1    0    0
Fat2Surfactant1    0    0
Fat1Surfactant2    0    0
Fat2Surfactant2    0    0

> h.c<-t(L.typeIII)%*%Baking.aov$coef
> t(h.c)%*%solve(t(L.typeIII)%*%Baking.sum$cov.unscaled%*%
+ L.typeIII)%*%h.c

         [,1]
[1,] 10.11785
```

MULTIVARIATE ANALYSIS OF VARIANCE

Multivariate analysis of variance, known as MANOVA, is the extension of analysis of variance techniques to multiple responses. The responses for an observation are considered as one multivariate observation, rather than as a collection of univariate responses.

If the responses are independent, then it is sensible to just perform univariate analyses. However, if the responses are correlated, then MANOVA can be more informative than the univariate analyses as well as less repetitive.

In S-PLUS the `manova` function is used to estimate the model. The formula needs to have a matrix as the response:

```
> wafer.manova <- manova(cbind(pre.mean, post.mean) ~ .,
+ wafer[,c(1:9, 11)])
```

The `manova` function creates an object of class `"manova"`. This class of object has methods specific to it for a few generic functions. The most important function is the `"manova"` method for `summary`, which produces a MANOVA table:

```
> summary(wafer.manova)

           Df Pillai Trace approx. F num df   den df   P-value
  maskdim 1  0.9863       36.00761  2        1        0.11703
 visc.tem 2  1.00879      1.01773   4        4        0.49341
   spinsp 2  1.30002      1.85724   4        4        0.28173
 baketime 2  0.80133      0.66851   4        4        0.64704
 aperture 2  0.96765      0.93733   4        4        0.52425
  exptime 2  1.63457      4.47305   4        4        0.08795
  devtime 2  0.99023      0.98065   4        4        0.50733
 etchtime 2  1.26094      1.70614   4        4        0.30874
Residuals 2
```

There are four common types of test in MANOVA. The example above shows the Pillai-Bartlett trace test, which is the default test in S-PLUS. The last four columns show an approximate F test (since the distributions of the four test statistics are not implemented). The other available tests are Wilks' Lambda, Hotelling-Lawley trace, and Roy's maximum eigenvalue. (By the way, a model with this few residual degrees of freedom is not likely to produce informative tests.)

You can view the results of another test by using the `test` argument. The following command shows you Wilks' lambda test:

```
> summary(wafer.manova, test="wilk")
```

Below is an example of how to see the results of all four of the multivariate tests:

```
> wafer.manova2 <- manova(cbind(pre.mean, post.mean,
+ log(pre.dev), log(post.dev)) ~ maskdim + visc.tem +
+ spinsp, wafer)
> wafer.ms2 <- summary(wafer.manova2)
> for(i in c("p","w","h","r")) print(wafer.ms2, test=i)
```

You can also look at the univariate ANOVA tables for each response with a command like:

```
> summary(wafer.manova, univariate=T)
```

Hand and Taylor (1987) provide a nice introduction to MANOVA. Many books on multivariate statistics contain a chapter on MANOVA. Examples include Mardia, Kent and Bibby (1979), and Seber (1984).

SPLIT-PLOT DESIGNS

A split-plot design contains more than one source of error. This can arise because factors are applied at different scales, as in the guayule example below.

Split-plots are also encountered because of restrictions on the randomization. For example, an experiment involving oven temperature and baking time will probably not randomize the oven temperature totally, but rather only change the temperature after all of the runs for that temperature have been made. This type of design is often mistakenly analyzed as if there were no restrictions on the randomization (an indication of this can be *p*-values that are close to 1). See Hicks (1973) and Daniel (1976).

S-PLUS includes the `guayule` data frame which is also discussed in Chambers and Hastie (1992). This experiment was on eight varieties of guayule (a rubber producing shrub) and four treatments on the seeds. Since a flat (a shallow box for starting seedlings) was not large enough to contain all 32 combinations of variety and treatment, the design was to use only a single variety in each flat and to apply each treatment within each flat. Thus the flats each consist of four sub-plots. This is a split-plot design since flats are the experimental unit for varieties, but the sub-plots are the experimental unit for the treatments. The response is the number of plants that germinated in each sub-plot.

To analyze a split-plot design like this, put the variable that corresponds to the whole plot in an `Error` term in the formula of the `aov` call:

```
> gua.aov1 <- aov(plants ~ variety*treatment +
+ Error(flats), guayule)
```

As usual, you can get an ANOVA table with `summary`:

```
> summary(gua.aov1)

Error: flats
          Df Sum of Sq  Mean Sq  F Value     Pr(F)
variety    7   763.156 109.0223 1.232036 0.3420697
Residuals 16  1415.833  88.4896
```

```
Error: Within
                  Df Sum of Sq  Mean Sq  F Value       Pr(F)
treatment          3  30774.28 10258.09 423.4386 0.00000e+00
variety:treatment 21   2620.14   124.77   5.1502 1.32674e-06
Residuals         48   1162.83    24.23
```

This shows varieties tested with the error from flats, while treatment and its interaction with variety are tested with the within-flat error (which is substantially smaller).

The `guayule` data actually represent an experiment in which the flats were grouped into replicates–making three sources of error, or a split-split-plot design. To model this we put more than one term inside the `Error` term:

```
> gau.aov2 <- aov(plants ~ variety*treatment +
+ Error(reps/flats), guayule)
> summary(gau.aov2)

Error: reps
          Df Sum of Sq  Mean Sq F Value Pr(F)
Residuals  2  38.58333 19.29167

Error: flats %in% reps
          Df Sum of Sq  Mean Sq  F Value     Pr(F)
variety    7   763.156 109.0223 1.108232 0.4099625
Residuals 14  1377.250  98.3750

Error: Within
                  Df Sum of Sq  Mean Sq  F Value       Pr(F)
treatment          3  30774.28 10258.09 423.4386 0.00000e+00
variety:treatment 21   2620.14   124.77   5.1502 1.32674e-06
Residuals         48   1162.83    24.23
```

The `Error` term could also have been specified as `Error(reps + Flats)`. However, the specification `Error(flats + reps)` would not give the desired result (the sequence within the `Error` term is significant); explicitly stating the nesting is preferred. Note that only one `Error` term is allowed.

REPEATED-MEASURES DESIGNS

Repeated-measures designs are those that contain a sequence of observations on each subject–for example, a medical experiment in which each patient is given a drug, and observations are taken at zero, one, two and three weeks after taking the drug. (The above description is a little too simplistic to encompass all repeated-measures designs, but it captures the spirit.)

Repeated-measures designs are similar to split-plot designs in that there is more than one source of error (between subjects and within subjects), but there is correlation in the within-subjects observations. In the example we expect that the observations in week three will be more similar to week two observations than to week zero observations. Because of this, the split-plot analysis (referred to as the univariate approach) is valid only under certain restrictive conditions.

We will use the artificial dataset `drug.mult`, which has the following form:

```
> drug.mult

  subject gender  Y.1  Y.2  Y.3  Y.4
1      S1      F 75.9 74.3 80.0 78.9
2      S2      F 78.3 75.5 79.6 79.2
3      S3      F 80.3 78.2 80.4 76.2
4      S4      M 80.7 77.2 82.0 83.8
5      S5      M 80.3 78.6 81.4 81.5
6      S6      M 80.1 81.1 81.9 86.4
```

The dataset consists of the two factors `subject` and `gender`, and the matrix `Y` which contains 4 columns. The first thing to do is stretch this out into a form suitable for the univariate analysis:

```
> drug.uni <- drug.mult[rep(1:6, rep(4,6)), 1:2]
> ymat <- data.matrix(drug.mult[,paste("Y.",1:4, sep="")])
> drug.uni <- cbind(drug.uni, time=
+ ordered(rep(paste("Week", 0:3,sep=""),6)),
+ y=as.vector(t(ymat)))
```

The univariate analysis treats the data as a split-plot design:

```
> summary(aov(y ~ gender*time + Error(subject), drug.uni))

Error: subject
          Df Sum of Sq  Mean Sq  F Value   Pr(F)
gender     1  60.80167 60.80167 19.32256 0.01173
Residuals  4  12.58667  3.14667

Error: Within
            Df Sum of Sq  Mean Sq  F Value     Pr(F)
time         3  49.10833 16.36944 6.316184 0.0081378
gender:time  3  14.80167  4.93389 1.903751 0.1828514
Residuals   12  31.10000  2.59167
```

Tests in the "Within" stratum are valid only if the data satisfy the "circularity" property, in addition to the usual conditions. Circularity means that the variance of the difference of measures at different times is constant; for example, the variance of the difference between the measures at week 0 and week 3 should be the same as the variance of the difference between week 2 and week 3. We also need the assumption that actual contrasts are used; for example, the `contr.treatment` function should not be used. When circularity does not hold, then the p-values for the tests will be too small.

One approach is to perform tests which are as conservative as possible. Conservative tests are formed by dividing the degrees of freedom in both the numerator and denominator of the F test by the number of repeated measures minus one. In our example there are four repeated measures on each subject, so we divide by 3. The split-plot and the conservative tests are:

```
> 1 - pf(6.316184, 3, 12) # usual univariate test

[1] 0.008137789

> 1 - pf(6.316184, 1, 4) # conservative test

[1] 0.06583211
```

These two tests are telling fairly different tales, so the data analyst would probably move on to one of two alternatives. A Huynh-Feldt adjustment of the degrees of freedom provides a middle ground

between the tests above–see Winer, Brown and Michels (1991), for instance. The multivariate approach, discussed below, substantially relaxes the assumptions.

The univariate test for "time" was really a test on three contrasts. In the multivariate setting we want to do the same thing, so we need to use contrasts in the response:

```
> drug.man <- manova(ymat %*% contr.poly(4) ~ gender,
+ drug.mult)
> summary(drug.man, intercept=T)

           Df Pillai Trace approx. F num df   den df    P-value
(Intercept) 1  0.832005     3.301706  3        2        0.241092
     gender 1  0.694097     1.512671  3        2        0.421731
  Residuals 4
```

The line marked "(Intercept)" corresponds to "time" in the univariate approach, and similarly the "gender" line here corresponds to "gender:time". The p-value of .24 is larger than either of the univariate tests–the price of the multivariate analysis being more generally valid is that quite a lot of power is lost. Although the multivariate approach is preferred when the data do not conform to the required conditions, the univariate approach is preferred when they do (the trick, of course, is knowing which is which).

Let's look at the univariate summaries that this MANOVA produces:

```
> summary(drug.man, intercept=T, univar=T)

Response: .L
            Df Sum of Sq Mean Sq  F Value     Pr(F)
(Intercept)  1    22.188 22.1880 4.327255 0.1059983
gender       1     6.912  6.9120 1.348025 0.3101900
Residuals    4    20.510  5.1275
Response: .Q
            Df Sum of Sq  Mean Sq F Value     Pr(F)
(Intercept)  1  5.415000 5.415000 5.30449 0.0826524
gender       1  4.001667 4.001667 3.92000 0.1188153
Residuals    4  4.083333 1.020833
```

```
Response: .C
            Df Sum of Sq  Mean Sq  F Value     Pr(F)
(Intercept)  1  21.50533 21.50533 13.22049 0.0220425
gender       1   3.88800  3.88800  2.39016 0.1969986
Residuals    4   6.50667  1.62667
```

If you add up the respective degrees of freedom and sums of squares, you will find that the result is the same as the univariate "Within" stratum. For this reason, the univariate test is sometimes referred to as the "average *F* test."

The above discussion has focused on classical inference, which should not be done before graphical exploration of the data.

Many books discuss repeated measures. Some examples are Hand and Taylor (1987), Milliken and Johnson (1984), Crowder and Hand (1990), and Winer, Brown, and Michels (1991).

RANK TESTS FOR ONE-WAY AND TWO-WAY LAYOUTS

This section briefly describes how to use two nonparametric rank tests for ANOVA: the *Kruskal-Wallis* rank sum test for a one-way layout and the *Friedman* test for unreplicated two-way layout with (randomized) blocks.

Since these tests are based on ranks, they are *robust* with regard to the presence of outliers in the data; that is, they are not affected very much by outliers. This is not the case for the classical F tests.

You can find detailed discussions of the Kruskal-Wallis and Friedman rank-based tests in a number of books on nonparametric tests; for example, Lehmann (1975) and Hettmansperger (1984).

The Kruskal-Wallis Rank Sum Test

When you have a one-way layout, as in the section Experiments With One Factor in Chapter 15, you can use the *Kruskal-Wallis rank sum* test `kruskal.test` to test the null hypothesis that all group means are equal.

We illustrate how to use `kruskal.test` for the blood coagulation data of Table 15.1. First you set up your data as for a one-factor experiment (or one-way layout). You create a vector object `coag`, arranged by factor level (or treatment), and you create a factor object `diet` whose levels correspond to the factor levels of vector object `coag`. Then use `kruskal.test`:

```
> kruskal.test(coag,diet)

        Kruskal-Wallis rank sum test

data:  coag and diet
Kruskal-Wallis chi-square = 17.0154, df = 3,
 p-value = 7e-04
alternative hypothesis: two.sided
```

The p-value of $p = .0007$ is highly significant. This p-value is computed using an asymptotic chi-squared approximation. See the help file for more details.

You may find it helpful to note that `kruskal.test` and `friedman.test` return the results of its computations, and associated information, in the same style as the functions in Chapter 3, Statistical Inference for One- and Two-Sample Problems.

The Friedman Rank Sum Test

When you have a two-way layout with one blocking variable and one treatment variable, you can use the *Friedman rank sum* test `friedman.test` to test the null hypothesis that there is no treatment effect.

We illustrate how you use `friedman.test` for the penicillin yield data described in Table 15.1 of Chapter 15. The general form of the usage is

```
friedman.test(y,groups,blocks)
```

where `y` is a numeric vector, `groups` contains the levels of the treatment factor and `block` contains the levels of the blocking factor. Thus, you can do:

```
> attach(pen.df) # make treatment and blend available

> friedman.test(yield, treatment, blend)

        Friedman rank sum test

data:  yield and pen.design[, 2] and pen.design[, 1]
Friedman chi-square = 3.4898, df = 3, p-value = 0.3221
alternative hypothesis: two.sided
```

The p-value is $p = .32$, which is not significant. This p-value is computed using an asymptotic chi-squared approximation. For further details on `friedman.test`, see the help file.

VARIANCE COMPONENTS MODELS

Variance components models are used when there is interest in the variability of one or more variables other than the residual error. For example, manufacturers often run experiments to see which parts of the manufacturing process contribute most to the variability of the final product. In this situation variability is undesirable, and attention is focused on improving those parts of the process that are most variable. Animal breeding is another area in which variance components models are routinely used. Some data, from surveys for example, that have traditionally been analyzed using regression can more profitably be analyzed using variance component models.

Estimating the Model

To estimate a variance component model, you first need to use `is.random` to state which factors in your data are random. A variable that is marked as being random will have a variance component in any models that contain it. Only variables that inherit from class `"factor"` can be declared random. Although `is.random` works on individual factors, it is often more practical to use it on the columns of a data frame. You can see if variables are declared random by using `is.random` on the data frame:

```
> is.random(pigment)

Batch Sample Test
    F      F    F
```

Declare variables to be random by using the assignment form of `is.random`:

```
> is.random(pigment) <- c(T, T, T)
> is.random(pigment)

Batch Sample Test
    T      T    T
```

Because we want all of the factors to be random, we could have simply done the following:

```
> is.random(pigment) <- T
```

The value on the right is replicated to be the length of the number of factors in the data frame.

Once you have declared your random variables, you are ready to estimate the model using the `varcomp` function. This function takes a formula and other arguments very much like `lm` or `aov`. Because the pigment data are from a nested design, the call has the following form:

```
> pigment.vc <- varcomp(Moisture ~ Batch/Sample, pigment)
> pigment.vc

Variances:
    Batch Sample %in% Batch Residuals
 7.127976          28.53333 0.9166667
Call:
varcomp(formula = Moisture ~ Batch/Sample, data = pigment)
```

The result of `varcomp` is an object of class `"varcomp"`. You can use `summary` on `"varcomp"` objects to get more details about the fit, and you can use `plot` to get qq-plots for the normal distribution on the estimated effects for each random term in the model.

Estimation Methods

The `method` argument to `varcomp` allows you to choose the type of variance component estimator. Maximum likelihood and REML (restricted maximum likelihood) are two of the choices. REML is very similar to maximum likelihood but takes the number of fixed effects into account (the usual unbiased estimate of variance in the one-sample model is a REML estimate). See Harville (1977) for more details on these estimators.

The default method is a MINQUE (minimum norm quadratic unbiased estimate); this class of estimator is locally best at a particular spot in the parameter space. The MINQUE option in S-PLUS is locally best if all of the variance components (except that for the residuals) are zero. The MINQUE estimate agrees with REML for balanced data. See Rao (1971) for details. This method was made the default because it is less computationally intense than the other methods, however, it can do significantly worse for severely unbalanced data (Swallow and Monahan (1984)).

You can get robust estimates by using `"method = winsor"`. This method creates new data by moving outlying points or groups of points toward the rest of the data. One of the standard estimators is then applied to this possibly revised data. Burns (1992) gives details of

the algorithm along with simulation results. This method uses much larger amounts of memory than the other methods if there are a large number of random levels, such as in a deeply nested design.

Random Slope Example

We now produce a more complicated example in which there are random slopes and intercepts. The data consist of several pairs of observations on each of several individuals in the study. An example might be that the y values represent the score on a test and the x values are the time at which the test was taken.

Let's start by creating simulated data of this form. We create data for 30 subjects and 10 observations per subject:

```
> subject <- factor(rep(1:30, rep(10,30)))
> set.seed(357) # makes these numbers reproducible
> trueslope <- rnorm(30, mean=1)
> trueint <- rnorm(30, sd=.5)
> times <- rchisq(300, 3)
> scores <- rep(trueint, rep(10,30)) + times *
+ rep(trueslope, rep(10,30)) + rnorm(300)
> test.df <- data.frame(subject, times, scores)
> is.random(test.df) <- T
> is.random(test.df) subject T
```

Even though we want to estimate random slopes and random intercepts, the only variable that is declared random is `subject`. Our model for the data has two coefficients: the mean slope (averaged over subjects) and the mean intercept. It also has three variances: the variance for the slope, the variance for the intercept, and the residual variance.

The following command estimates this model using Maximum Likelihood (the default MINQUE is not recommended for this type of model):

```
> test.vc <- varcomp(scores ~ times * subject,
+ data=test.df, method="ml")
```

This seems very simple. We can see how it works by looking at how the formula get expanded. The right side of the formula is expanded into four terms:

```
scores ~ 1 + times + subject + times:subject
```

The intercept term in the formula, represented by 1, gives the mean intercept. The variable `times` is fixed and produces the mean slope. The `subject` variable is random and produces the variance component for the random intercept. Since any interaction containing a random variable is considered random, the last term, `times:subject`, is also random; this term gives the variance component for the random slope. Finally, there is always a residual variance.

Now we can look at the estimates:

```
> test.vc

Variances:
   subject times:subject Residuals
 0.3162704      1.161243 0.8801149
Message:
[1] "RELATIVE FUNCTION CONVERGENCE"
Call:
varcomp(formula = scores ~ times*subject, data=test.df,
  method = "ml")
```

This shows the three variance components. The variance of the intercept, which has true value .25, is estimated as .32. Next, labeled `times:subject` is the variance of the slope, and finally the residual variance. We can also view the estimates for the coefficients of the model, which have true values of 0 and 1.

```
> coef(test.vc)

 (Intercept)   times
   0.1447211 1.02713
```

REFERENCES

Burns, P.J. (1992). *Winsorized REML estimates of variance components.* Submitted.

Chambers, J.M. and Hastie, T.J. (1992). *Statistical Models in S.* Wadsworth and Brooks Cole Advanced Books and Software, Pacific Grove, CA.

Crowder, M.J. and Hand, D.J. (1990). *Analysis of Repeated Measures.* Chapman and Hall, London.

Daniel, C. (1976). *Applications of Statistics to Industrial Experimentation.* Wiley, New York.

Hand, D.J. and Taylor, C.C. (1987). *Multivariate Analysis of Variance and Repeated Measures.* Chapman and Hall, London.

Harville, D.A. (1977). *Maximum likelihood approaches to variance component estimation and to related problems (with discussion).* Journal of the American Statistical Association, 72:320-340.

Hettmansperger, T.P. (1984). *Statistical Inference Based on Ranks.* John Wiley, New York.

Hicks, C.R. (1973). *Fundamental Concepts in the Design of Experiments.* Holt, Rinehart and Winston, New York.

Kennedy, W.J., Gentle, J.E., (1980), *Statistical Computing.* Marcel Dekker, Inc., New York, p. 396.

Lehmann, E.L. (1975). *Nonparametrics: Statistical Methods Based on Ranks.* Holden-Day, San Francisco.

Mardia, K.V., Kent, J.T., and Bibby, J.M. (1979). *Multivariate Analysis.* Academic Press, London.

Milliken, G.A. and Johnson, D.E., (1984), *Analysis of Messy Data Volume I: Designed Experiments.* Van Norstrand Reinhold Co., New York, 473.

Rao, C.R. (1971). *Estimation of variance and covariance components–MINQUE theory.* Journal of Multivariate Analysis, 1:257-275.

SAS Institute, Inc. (1978). *Tests of Hypotheses in Fixed-Effects Linear Models.* SAS Technical Report R-101. SAS Institute, Inc., Cary, NC.

SAS Institute, Inc. (1990). *SAS/Stat User's Guide*, Fourth Edition. SAS Institute, Inc., Cary, NC, pp. 120-121.

Searle, S.R., (1987), *Linear Models for Unbalanced Data*. John Wiley & Sons, New York, 536.

Seber, G.A.F. (1984). *Multivariate Observations*. Wiley, New York.

Swallow, W.H. and Monahan, J.F. (1984). *Monte Carlo comparison of ANOVA, MIVQUE, REML, and ML estimators of variance components*. Technometrics, 26:47-57.

Winer, B.J., Brown, D.R., and Michels, K.M. (1991). *Statistical Principles in Experimental Design*. McGraw-Hill, New York.

MULTIPLE COMPARISONS 17

INTRODUCTION

This chapter describes the use of the function `multicomp` in the analysis of multiple comparisons. The section Overview describes simple calls to `multicomp` for standard comparisons in one-way layouts. The section Advanced Applications tells how to use `multicomp` for nonstandard designs and comparisons. In the section Capabilities and Limits, the capabilities and limitations of this function are summarized.

OVERVIEW

When an experiment has been carried out in order to compare effects of several treatments, a classical analytical approach is to begin with a test for equality of those effects. Regardless of whether one embraces this classical strategy, and regardless of the outcome of this test, one is usually not finished with the analysis until determining where any differences exist, and how large the differences are (or might be); that is, until one does multiple comparisons of the treatment effects.

As a simple start, consider the built-in S-PLUS data frame on fuel consumption of vehicles, `fuel.frame`. Each row provides the fuel consumption (`Fuel`) in 100*gallons/mile for a vehicle model, as well as the `Type` group of the model: Compact, Large, Medium, Small, Sporty, or Van. There is also information available on the `Weight` and `Displacement` of the vehicle. Figure 17.1 shows a boxplot of fuel consumption, the result of the following commands.

```
> attach(fuel.frame)
> boxplot(split(Fuel,Type))
```

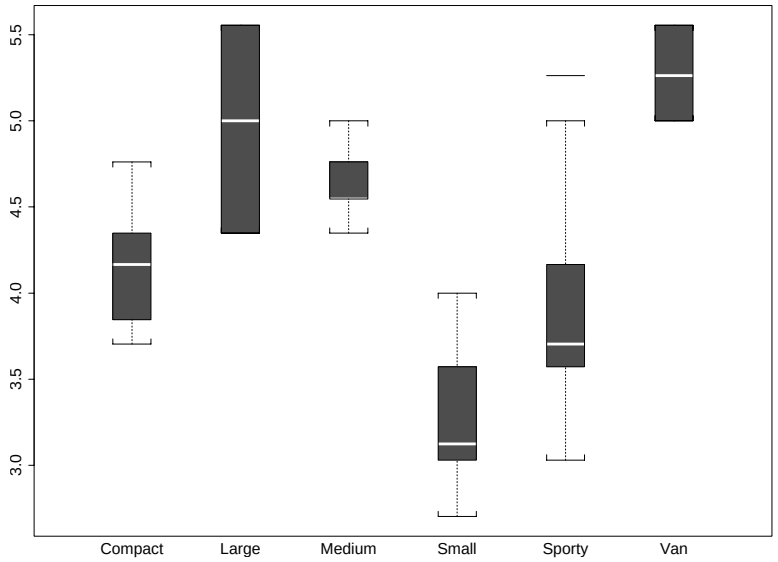

Figure 17.1: *Fuel consumption boxplot.*

Not surprisingly, the plot suggests that there are differences between vehicle types in terms of mean fuel consumption. This is confirmed by a one-factor analysis of variance test of equality obtained by a call to `aov`.

```
> aovout.fuel <- aov( Fuel ~ Type, data = fuel.frame)
> anova(aovout.fuel)

Analysis of Variance Table
Response: Fuel
Terms added sequentially (first to last)

          Df  Sum of Sq    Mean Sq    F Value          Pr(F)
Type       5   24.23960   4.847921   27.22058   1.220135e-13
Residuals 54    9.61727   0.178098
```

The boxplots show some surprising patterns, and inspire some questions. Do Small cars really have lower mean fuel consumption than Compact cars? If so, by what amount? What about Small versus Sporty cars? Vans versus Large cars? Answers to these questions are offered by an analysis of all pairwise differences in mean fuel consumption, which can be obtained from a call to `multicomp`:

```
> mca.fuel <- multicomp(aovout.fuel, focus = "Type")
> plot(mca.fuel)
> mca.fuel

95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method
critical point: 2.9545
response variable: Fuel
intervals excluding 0 are flagged by '****'

                Estimate      Std.    Lower    Upper
                            Error    Bound    Bound
Compact-Large     -0.800     0.267   -1.590  -0.0116  ****
Compact-Medium    -0.434     0.160   -0.906   0.0387
Compact-Small      0.894     0.160    0.422   1.3700  ****
Compact-Sporty     0.210     0.178   -0.316   0.7360
Compact-Van       -1.150     0.193   -1.720  -0.5750  ****
Large-Medium       0.366     0.270   -0.432   1.1600
Large-Small        1.690     0.270    0.896   2.4900  ****
Large-Sporty       1.010     0.281    0.179   1.8400  ****
Large-Van         -0.345     0.291   -1.210   0.5150
Medium-Small       1.330     0.166    0.839   1.8200  ****
Medium-Sporty      0.644     0.183    0.103   1.1800  ****
```

```
Medium-Van          -0.712      0.198    -1.300    -0.1270   ****
Small-Sporty        -0.684      0.183    -1.220    -0.1440   ****
Small-Van           -2.040      0.198    -2.620    -1.4600   ****
Sporty-Van          -1.360      0.213    -1.980    -0.7270   ****
```

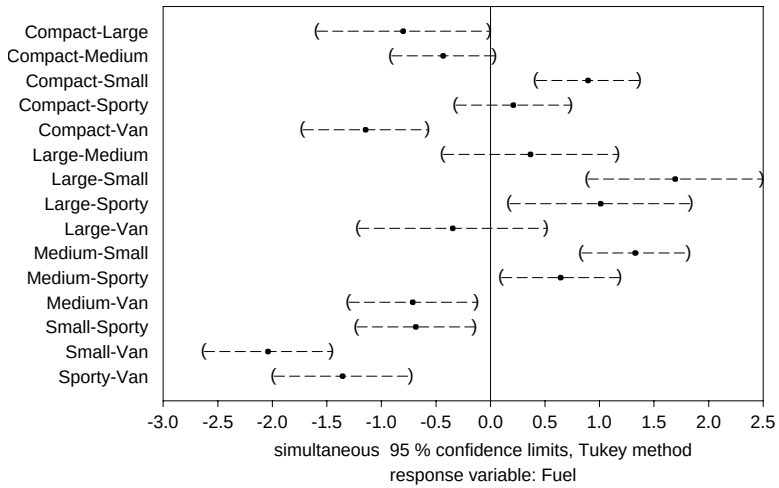

Figure 17.2: *Fuel consumption ANOVA.*

As the output and plot in Figure 17.2 indicate, this default call to `multicomp` has resulted in the calculation of simultaneous 95% confidence intervals for all pairwise differences between vehicle `Fuel` means, based on the levels of `Type`, sometimes referred to as MCA comparisons (Hsu, 1996). The labeling states that Tukey's method (Tukey, unpublished report, Princeton University, 1953) has been used; since group sample sizes are unequal, this is actually equivalent to what is commonly known as the Tukey-Kramer (Kramer, 1956) multiple comparison method.

Honestly Significant Differences

The output indicates via asterisks the confidence intervals which exclude zero; in the plot, these can be identified by noting intervals that do not intersect the vertical reference line at zero. These identified statistically significant comparisons correspond to pairs of (long run) means which can be declared different by Tukey's "HSD" (honestly significant difference) method. Not surprisingly, we can assert that most of the vehicle types have different mean fuel consumption rates. If we require 95% confidence in all of our statements, we cannot claim different mean fuel consumption rates

between the Compact and Medium types, the Compact and Sporty types, the Large and Medium types, and the Large and Van types. Note we should *not* assert that these pairs have *equal* mean consumption rates; for example, the interval for Compact-Medium states that this particular difference in mean fuel consumption is between -0.906 and 0.0387 units. Hence, the Medium vehicle type may have larger mean fuel consumption than the Compact, by as much as 0.9 units. Only an engineer can judge the importance of a difference of this size; if it is considered trivial, then using these intervals we can claim that for all *practical* purposes these two types have equal mean consumption rates; if not, there may still be an important difference between these types, and we would need more data to resolve the question.

The point to the above discussion is that there is more information in these simultaneous intervals than is provided by a collection of significance tests for differences. This is true whether the tests are reported via conclusions "Reject"/"Do not reject", or via *p*-values or adjusted *p*-values. This superior level of information using confidence intervals has been acknowledged by virtually all modern texts on multiple comparisons (Hsu, 1996; Bechhofer *et al*., 1996; Hochberg and Tamhane, 1987; Toothaker, 1993). All multiple comparison analyses using `multicomp` are represented by using confidence intervals or bounds.

Rat Growth Hormone Treatments

If all the intervals are to hold simultaneously with a given confidence level, it is important to calculate intervals only for those comparisons which are truly of interest. For example, consider the summary data in Table 17.1 from Hsu (Hsu, 1996) concerning a study by Juskevich and Guyer (1990) in which rat growth was studied under several growth-hormone treatments.

In this setting, it may only be necessary to compare each hormone treatment's mean growth with that of the placebo (that is, the oral administration with zero dose). These all-to-one comparisons are usually referred to as multiple comparisons with a control (MCC) (Dunnett, 1955). Suppose that the raw data for each rat were available in a data frame `hormone.dfr` with variables `growth` (numeric) and

`treatment` (a factor object) for each rat. Then the following statements would calculate, print, and plot Dunnett's intervals:

```
> aovout.growth <- aov(growth~treatment, data=hormone.dfr)
> multicomp(aovout.growth, focus = "treatment",
+ comparisons = "mcc", control = 1, plot = T)
```

Table 17.1: *Mean weight gain in rats under hormone treatments.*

method/dose	mean growth (g)	std.dev.	sample size
oral, 0	324	39.2	30
inject,1.0	432	60.3	30
oral,0.1	327	39.1	30
oral,0.5	318	53.0	30
oral,5	325	46.3	30
oral,50	328	43.0	30

The results are shown graphically in Figure 17.3. The intervals clearly show that only the injection method is distinguishable from the placebo in terms of long run mean weight gain.

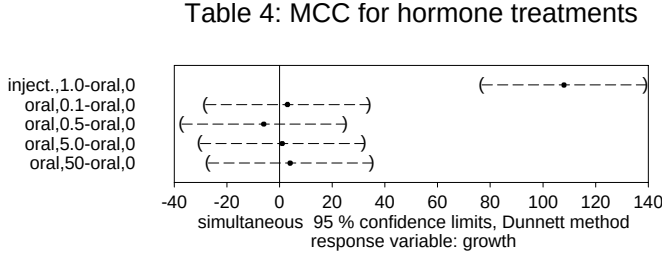

Figure 17.3: *MCC for rat hormone treatments.*

More Detail on multicomp

The first and only required argument to `multicomp` is an `aov` object (or equivalent), the results of a fixed-effects linear model fit by `aov` or a similar model-fitting function. The `focus` argument, when specified, names a factor (a main effect) in the fitted `aov` model. Comparisons will then be calculated on (adjusted) means for levels of the `focus` factor. The `comparisons` argument is an optional argument which can specify a standard family of comparisons for the levels of the `focus` factor. The default is `comparisons = "mca"`, which creates all pairwise comparisons. Setting `comparisons = "mcc"` creates all-to-one comparisons relative to the level specified by the `control` argument. The only other `comparisons` option available is "`none`", which states that the adjusted means themselves are of interest (with no differencing), in which case the default method for interval calculation is known as the studentized maximum modulus method. Other kinds of comparisons and different varieties of adjusted means can be specified through the `lmat` and `adjust` options discussed below.

Upper and Lower Bounds

Confidence intervals provide both upper and lower bounds for each difference or adjusted mean of interest. In some instances, only the lower bounds, or only the upper bounds, may be of interest. For example, in the fuel consumption example earlier, we may only be interested in determining which types of vehicle clearly have greater fuel consumption than compacts, and in calculating lower bounds for the difference. This can be accomplished through lower mcc bounds:

```
> aovout.fuel<-aov(Fuel~Type, data=fuel.frame)
> multicomp(aovout.fuel, focus="Type",comparison="mcc",
+ bounds="lower", control=1, plot=T)

95 % simultaneous confidence bounds for specified
linear combinations, by the Dunnett method

critical point: 2.3332000000000002
response variable: Fuel

bounds excluding 0 are flagged by '****'
```

```
                Estimate Std.Error Lower Bound
 Large-Compact     0.800     0.267      0.1770 ****
Medium-Compact     0.434     0.160      0.0606 ****
 Small-Compact    -0.894     0.160     -1.2700
Sporty-Compact    -0.210     0.178     -0.6250
   Van-Compact     1.150     0.193      0.6950 ****
```

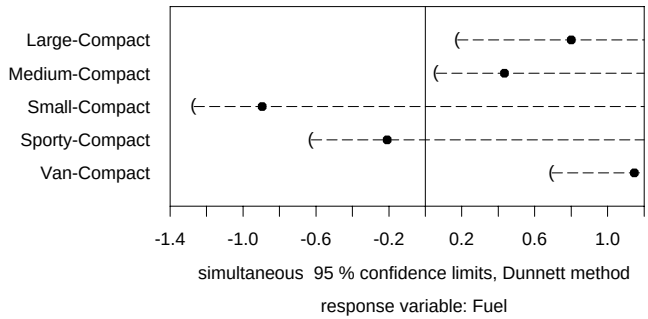

Figure 17.4: *Lower mcc bounds for rat hormone treatments.*

The intervals or bounds computed by `multicomp` are always of the form

$$(\text{estimate}) \pm (\text{critical point}) \times (\text{standard error of estimate})$$

The reader has probably already noticed that the estimates and standard errors are supplied in the output table. The critical point used depends on the specified or implied multiple comparison `method`.

Calculation of Critical Points

The `multicomp` function can calculate critical points for simultaneous intervals or bounds by the following methods:

- Tukey (`method = "tukey"`),
- Dunnett (`method = "dunnett"`),
- Sidak (`method = "sidak"`),
- Bonferroni (`method = "bon"`),
- Scheffé (`method = "scheffe"`)
- Simulation-based (`method = "sim"`).

Non-simultaneous intervals use the ordinary Student's-t critical point, `method = "lsd"`. If the user specifies a `method`, the function will check its validity in view of the model fit and the types of comparisons requested. For example, `method = "dunnett"` will be invalid if `comparisons = "mca"`. If the specified method does not satisfy the validity criterion, the function terminates with a message to that effect. This safety feature can be disabled by specifying the optional argument `valid.check = F`. If no `method` is specified, the function uses the smallest critical point among the valid non-simulation-based methods. If the user specifies `method = "best"`, the function uses the smallest critical point among all valid methods including simulation; this latter method may take a few moments of computer time.

The simulation-based method generates a near-exact critical point via Monte Carlo simulation, as discussed by Edwards and Berry (1987). For nonstandard families of comparisons or unbalanced designs, this method will often be substantially more efficient than other valid methods. The simulation size is set by default to provide a critical point whose actual error rate is within 10% of the nominal α (with 99% confidence). This amounts to simulation sizes in the tens of thousands for most choices of α. The user may directly specify a simulation size via the `simsize` argument to `multicomp`, but smaller simulation sizes than the default are not advisable. It is important to note that if the simulation-based method is used, the critical point (and hence the intervals) will vary slightly over repeated calls; recalculating the intervals repeatedly searching for some desirable outcome will usually be fruitless, and will result in intervals which do not provide the desired confidence level.

Error Rates for Confidence Intervals

Other `multicomp` arguments of interest are the `alpha` argument which specifies the error rate for the intervals or bounds, with default `alpha = .05`. By default, `alpha` is a familywise error rate, that is, the user may be (1 - `alpha`) x 100% confident that *every* calculated bound holds. If the user desires confidence intervals or bounds without simultaneous coverage, specify `error.type = "cwe"`, meaning comparisonwise error rate protection; in this case the user must also

specify `method = "lsd"`. Finally, for users familiar with the Scheffé (1953) method, the critical point is of the form:

```
sqrt(Srank*qf(1-alpha, Srank, df.residual))
```

The numerator degrees of freedom `Srank` may be directly specified as an option. If omitted, it is computed based on the specified comparisons and `aov` object.

ADVANCED APPLICATIONS

In the first example, the Fuel consumption differences found between vehicle types are almost surely attributable to differences in Weight and/or Displacement. Figure 17.5 shows a plot of Fuel versus Weight with plotting symbols identifying the various model types:

```
> plot(Weight,Fuel,type = 'n')
> text(Weight,Fuel,abbreviate(as.character(Type)))
```

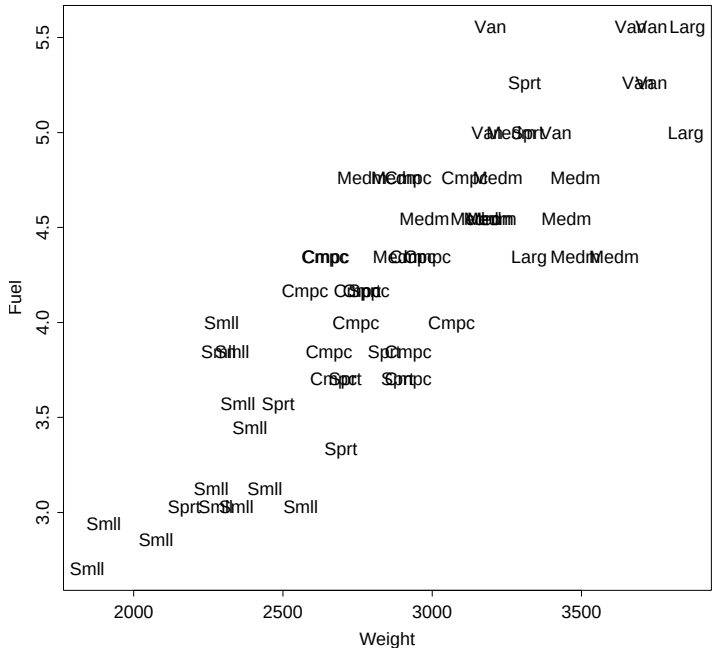

Figure 17.5: *Fuel consumption versus Weight.*

This plot shows a strong, roughly linear relationship between Fuel consumption and Weight, suggesting the addition of Weight as a covariate in the model. Though it may be inappropriate to compare adjusted means for all six vehicle types (see below), for the sake of example the following calls fit this model and calculates simultaneous

confidence intervals for all pairwise differences of adjusted means, requesting the best valid method:

```
> lmout.fuel.ancova <- lm(Fuel ~ Type+Weight,
+ data = fuel.frame)
> multicomp(lmout.fuel.ancova, focus = "Type",
+ method = "best", plot = T)
```

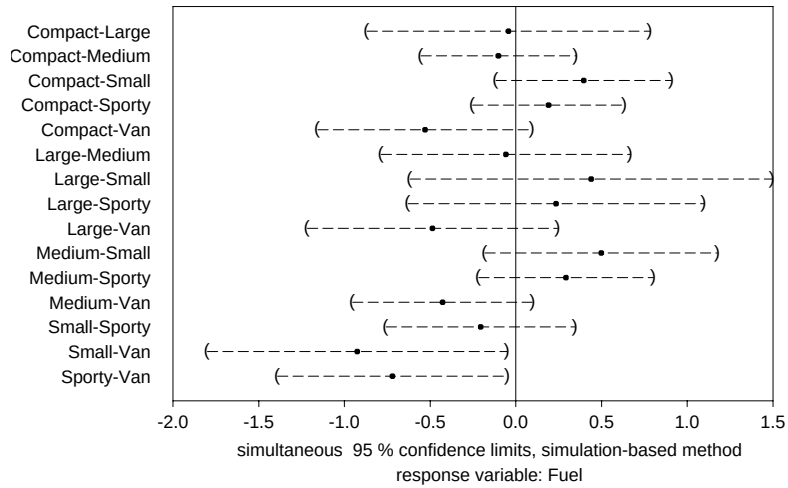

Figure 17.6: *Fuel consumption ANCOVA (adj. for Weight).*

The "best" valid method for this particular setting is the simulation-based method; Tukey's method has not been shown to be valid in the presence of covariates when there are more than three treatments. The intervals show that, adjusting for weight, the mean fuel consumption of the various vehicle types are in most cases within one unit of each other. The most notable exception is the Van type, which is showing higher mean fuel consumption than the Small and Sporty types, and most likely higher than the Compact, Medium and Large types.

Adjustment Schemes

When there is more than one term in the `lm` model, `multicomp` calculates standard adjusted means for levels of the `focus` factor and then takes differences as specified by the `comparisons` argument. Covariates are adjusted to their grand mean value. If there are other factors in the model, the standard adjusted means for levels of the `focus` factor use the average effect over the levels of any other (non-

nested) factors. This adjustment scheme can be changed using the `adjust` argument, which specifies a list of adjustment levels for non-`focus` terms in the model. Any terms excluded from the `adjust` list are adjusted in the standard way. The `adjust` list may include multiple adjustment values for each term; a full set of adjusted means for the `focus` factor is calculated for each combination of values specified by the `adjust` list. Differences (if any) specified by the `comparisons` argument are then calculated for each combination of values specified by the `adjust` list.

Toothaker's Two-Factor Design

Besides allowing the user to specify covariate values for adjustment, the `adjust` argument can be used to calculate"simple effects" comparisons when factors interact, or (analogously) when covariate slopes are different. This is probably best illustrated by an example: Toothaker (1993) discusses a two-factor design, using the data collected by Frank (1984). Subjects are undergraduate females, with response the score on a 20-item multiple choice test over a taped lecture. Factors are cognitive style (*cogstyle*, levels FI = *Field independent* and FD = *Field dependent*) and study technique (*studytech*: NN = no notes, SN = student notes, PO = partial outline supplied, CO = complete outline). The following code fits the model and performs a standard two-factor analysis of variance.

```
> score <- c(13, 13, 10, 16, 14, 11, 13, 13, 11, 16, 15, 16,
+ 10, 15, 19, 19, 17, 19, 17, 20, 17, 18, 17, 18, 18, 19,
+ 19, 18, 17, 19, 17, 19, 17, 19, 17, 15, 18, 17, 15, 15,
+ 19, 16, 17, 19, 15, 20, 16, 19, 16, 19, 19, 18, 11, 14,
+ 11, 10, 15, 10, 16, 16, 17, 11, 16, 11, 10, 12, 16, 16,
+ 17, 16, 16, 16, 14, 14, 16, 15, 15, 15, 18, 15, 15, 14,
+ 15, 18, 19, 18, 18, 16, 16, 18, 16, 18, 19, 15, 16, 19,
+ 18, 19, 19, 18, 17, 16, 17, 15)
> cogstyle <- factor(c(rep("FI",52), rep("FD",52)))
> studytec <- factor(c(rep("NN",13), rep("SN", 13),
+ rep("PO",13), rep("CO",13), rep("NN",13), rep("SN", 13),
+ rep("PO",13), rep("CO",13)))
> interaction.plot(cogstyle,studytec,score)
> aovout.students <- aov( score ~ cogstyle*studytec)
```

```
> anova(lmout.students)

Analysis of Variance Table
Response: score
Terms added sequentially (first to last)
                Df    Sum of Sq   Mean Sq    F Value     Pr(F)
cogstyle        1       25.0096   25.0096   7.78354 0.00635967
studytec        3      320.1827  106.7276  33.21596 0.00000000
cogstyle:studytec 3     27.2596    9.0865   2.82793 0.04259714
Residuals       96     308.4615    3.2131
```

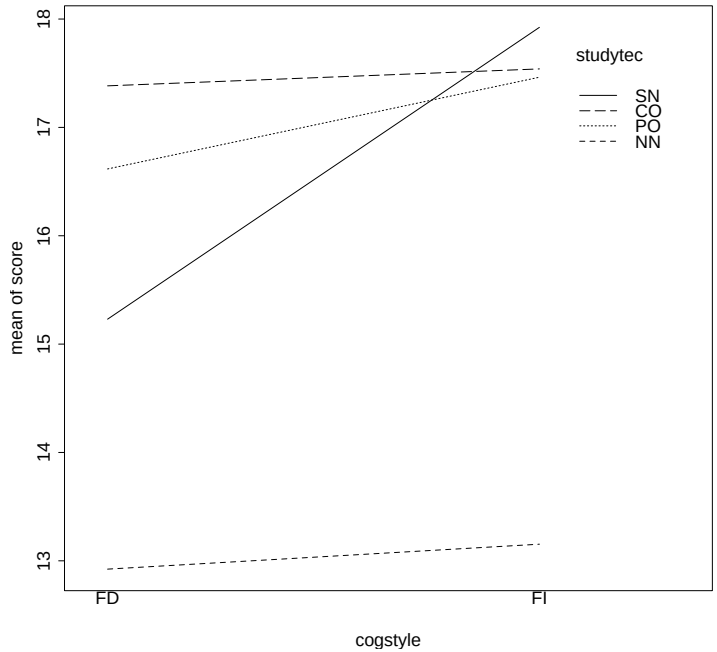

Figure 17.7: *Two-factor design test scores.*

It is apparent from the test for interaction and the profile plot that there is non-negligible interaction between these factors. In such cases it will often be of interest to follow the tests with an analysis of "simple effects," in this case a comparison of the four study techniques performed separately for each cognitive style group. The following

call calculates simultaneous 95% intervals for these differences by the best valid method, which is again simulation.

```
> mcout.students <- multicomp(aovout.students,
+ focus = "studytech", adjust = list(cogstyle =
+ c("FI","FD") ), method = "best")
> plot(mcout.students)
> mcout.students

95 % simultaneous confidence intervals for specified
linear combinations, by the simulation-based method
critical point: 2.8774
response variable: score
simulation size= 12616
                Estimate      Std.    Lower    Upper
                              Error    Bound    Bound
CO-NN.adj1       4.3800       0.703    2.360    6.410  ****
CO-PO.adj1       0.0769       0.703   -1.950    2.100
CO-SN.adj1      -0.3850       0.703   -2.410    1.640
NN-PO.adj1      -4.3100       0.703   -6.330   -2.280  ****
NN-SN.adj1      -4.7700       0.703   -6.790   -2.750  ****
PO-SN.adj1      -0.4620       0.703   -2.480    1.560  ****
CO-NN.adj2       4.4600       0.703    2.440    6.480  ****
CO-PO.adj2       0.7690       0.703   -1.250    2.790
CO-SN.adj2      -2.3100       0.703   -4.330   -0.285  ****
NN-PO.adj2      -3.6900       0.703   -5.720   -1.670  ****
NN-SN.adj2      -2.3100       0.703   -4.330   -0.285  ****
PO-SN.adj2       1.3800       0.703   -0.638   3.410
```

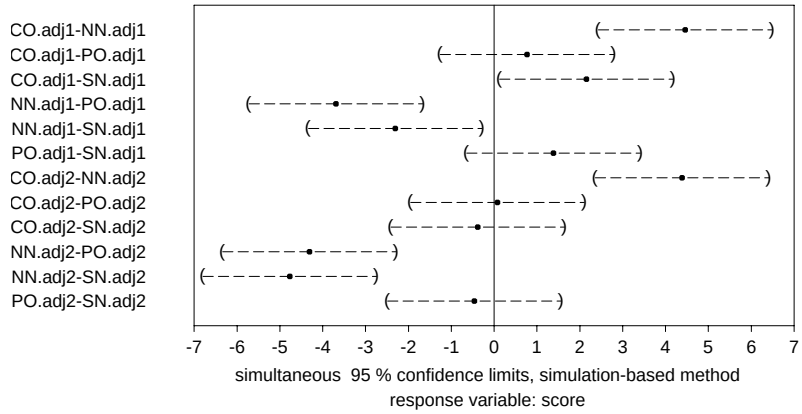

Figure 17.8: *Simple effects for study techniques.*

Setting Linear Combinations of Effects

In many situations, the setting calls for inference on a collection of comparisons or linear combinations other than those available through specifications of the `focus`, `adjust`, and `comparisons` arguments. The `lmat` argument to `multicomp` allows the user to directly specify any collection of linear combinations of the model effects for inference. `lmat` is a matrix (or an expression evaluating to a matrix) whose columns specify linear combinations of the model effects for which confidence intervals or bounds are desired. Specified linear combinations are checked for estimability; if inestimable, the function terminates with a message to that effect. The user may disable this safety feature by specifying the optional argument `est.check = F`. Specification of `lmat` overrides any `focus` or `adjust` arguments; at least one of `lmat` or `focus` must be specified. Differences requested or implied by the `comparisons` argument are taken over the columns of `lmat`. In many instances no such further differencing would be desired, in which case the user should specify `comparisons = "none"`.

Textbook Parameterization

Linear combinations in `lmat` use the "textbook parameterization" of the model. For example, the fuel consumption analysis of covariance model parameterization has eight parameters: an `Intercept`, six coefficients for the factor `Type` (`Compact`, `Large`, `Medium`, `Small`,

`Sporty`, `Van`) and a coefficient for the covariate `Weight`. Note that the levels of the factor object `Type` are listed in alphabetical order in the parameter vector.

In the Fuel consumption problem, many would argue that it is not appropriate to compare, for example, adjusted means of Small vehicles and Large vehicles, since these two groups' weights do not overlap. Inspection of Figure 17.5 shows that, under this consideration, comparisons are probably only appropriate within two weight groups: Small, Sporty, and Compact as a small weight group; Medium, Large, and Van as a large weight group. We can accomplish comparisons within the two `Weight` groups using the following matrix, which is assumed to be pre-typed in a text file `"lmat.fuel"`. Note the column labels, which will be used to identify the intervals in the created figure and plot:

Table 17.2: *The Weight comparison matrix in the file* `lmat.fuel`.

	Com-Sma	**Com-Spo**	**Sma-Spo**	**Lar-Med**	**Lar-Van**	**Med-Van**
Intercept	0	0	0	0	0	0
Compact	1	1	0	0	0	0
Large	0	0	0	1	1	0
Medium	0	0	0	-1	0	1
Small	-1	0	1	0	0	0
Sporty	0	-1	-1	0	0	0
Van	0	0	0	0	-1	-1
Weight	0	0	0	0	0	0

The code below creates the intervals. If we restrict attention to these comparisons only, we cannot assert any differences in adjusted mean fuel consumption.

```
> multicomp.lm(lmout.fuel.ancova, lmat = lmat.fuel,
+ comparisons = "none", method = "best", plot = T)
```

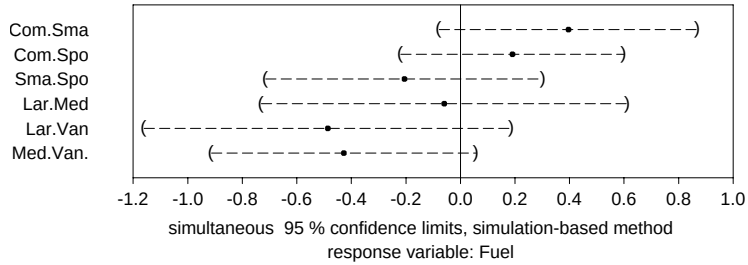

Figure 17.9: *Using* `lmat` *for specialized contrasts.*

The textbook parameterizations for linear models are created according to the following algorithm:

1. An Intercept parameter is included first, if the model contains one.
2. For each "main effect" term in the model (terms of order one), groups of parameters are included in the order the terms are listed in the model specification. If the term is a factor, a parameter is included for each level. If the term is numeric, a parameter is included for each column of its matrix representation.
3. Parameters for terms of order 2 (for example, A:B) are created by "multiplying" the parameters of each main effect in the term, in left-to-right order. For example, if A has levels A1, A2 and B has levels B1, B2, B3, the parameters for A:B are A1B1 A1B2 A1B3 A2B1 A2B2 A2B3.
4. Parameters for higher level terms are created by multiplying the parameterizations of lower level terms two at a time, left to right. For example, the parameters for A:B:C are those of A:B multiplied by C.

Overparameterized Models

The textbook parameterization will often be awkwardly overparameterized. For example, the 2 x 4 factorial model specified in the student study techniques example has the following parameters, in order; note the alphabetical rearrangement of the factor levels:

- Intercept
- FD FI
- CO NN PO SN
- FDCO FDNN FDPO FDSN FICO FINN FIPO FISN

Clearly, care must be taken in creating an `lmat` for factorial designs, especially with crossed and/or nested terms. The flexibility `lmat` provides for creating study-specific linear combinations can be extremely valuable, though. If you are in doubt about the actual "textbook parameterization" of a given linear model, it may help to run a standard analysis and inspect the `lmat` created, which is part of the output list of `multicomp`. For example, for the simple effects analysis of the student test scores of Figure 17.8, the implied `lmat` can be seen using the command:

```
> mcout.students$lmat
```

Multicomp Methods Compared

The function `multicomp.lm`, after checking estimability of specified linear combinations and creating a vector of estimates, a covariance matrix, and degrees of freedom, calls the "base" function `multicomp.default`. The function `multicomp.default` will be directly valuable in many settings. It uses a vector of estimates `bvec` and associated covariance matrix `vmat` as required arguments, with optional degrees of freedom `df.residual` (possibly `Inf`, the default) to calculate confidence intervals on linear combinations of `bvec`. These linear combinations can be specified through an optional `lmat` argument and/or `comparisons` argument; there is neither a `focus` nor an `adjust` argument. Linear combinations of `bvec` defined by columns of `lmat` (if any; the default `lmat` is an identity matrix) are calculated, followed by any differences specified or implied by the `comparisons` argument. The `multicomp.lm` options `method`, `bounds`, `alpha`, `error.type`, `crit.point`, `sim.size`, `Srank`, `valid.check`, and `plot` are also available in `multicomp.default`.

The function `multicomp.default` can be very useful as a means of calculating intervals based on summary data, or using the results of some model-fitting program other than `lm`; `bvec` must be considered as a realization of a multivariate normal vector. If the matrix `vmat` incorporates any estimate of variance considered to be a realized chi-square variable, the degrees of freedom `df.residual` must be specified.

The rat growth data discussed earlier (Table 17.1) provides a simple example of the use of `multicomp.default`. Here, the first few statements create the vector of estimates `bvec` and covariance matrix `vmat` assuming that a single factor analysis of variance model is appropriate for the data, followed by the statement that produced the lower mcc bounds of Figure 17.4:

```
> growth <- c(324, 432, 327, 318, 325, 328)
> stddev <- c(39.2, 60.3, 39.1, 53.0, 46.3, 43.0)
> samp.size <- rep(30,6)
> names(growth) <- c( “oral,0”, ”inject,1.0”, ”oral,0.1”,
+ ”oral,0.5”, ”oral,5”, ”oral,50”)
> mse <- mean(stddev^2)
> vmat <-mse*diag(1/samp.size)
> multicomp.default(growth, vmat, df.residual =
+ sum(samp.size-1), comparisons = “mcc”, bounds = “lower”,
+ control = 1, plot = T)
```

CAPABILITIES AND LIMITS

In summary, the function `multicomp` uses the information in a linear model; that is, a fitted fixed effects linear model. Through some combination of the `focus`, `adjust`, `comparisons` and `lmat` arguments, any collection of estimable linear combinations of the fixed effects may be estimated, and simultaneous or non-simultaneous intervals or bounds computed by any of the applicable methods mentioned above. Specified linear combinations are checked for estimability unless the user specifies `est.check = F`. Specified methods are checked for validity unless the user specifies `valid.check = F`.

The function `multicomp.default` uses a specified vector of parameter estimates `bvec` and a covariance matrix `vmat`, which will usually have some associated degrees of freedom `df.residual` specified. Possibly through some combination of the `comparisons` or `lmat` arguments, any collection of linear combinations of the parameters may be estimated, and simultaneous or non-simultaneous intervals or bounds computed by any of the applicable methods discussed above. Specified methods are checked for validity unless the user specifies `valid.check = F`.

The output from either procedure is an object of class "`multicomp`", a list containing elements `$table` (a matrix of calculated linear combination estimates, standard errors, and lower and/or upper bounds), `$alpha`, `$error.type`, `$method`, `$crit.point`, `$lmat` (the final matrix of linear combinations specified or implied), and other ancillary information pertaining to the intervals. If the argument `plot = T` is specified, the intervals/bounds are plotted on the active device. If not, the created `multicomp` object can be used as an argument to `plot` (see `plot.multicomp`).

The critical points for the methods of Tukey and Dunnett are calculated by numerically using the S-PLUS quantile functions `qtukey`, `qdunnett`, `qmvt`, and `qmvt.sim`, which may be directly useful to advanced users for their own applications.

What the function `multicomp` does *not* do:

1. Any stagewise or multiple range test. The simultaneous testing procedures attributed to Fisher, Tukey, Scheffé, Sidak and Bonferroni are implied by the use of the corresponding

`method` and noting which of the calculated intervals excludes zero. The multiple range tests of Duncan(1955) and Newman(1959)-Keuls(1952) do not provide familywise error protection, and are not very efficient for comparisonwise error protection; modern texts on multiple comparisons recommend uniformly against these two multiple range tests (Hsu, 1996; Hochberg and Tamhane, 1987; Bechofer *et al.*, 1996; Toothaker 1993).

2. Multiple comparisons with the "best" treatment (MCB; Hsu, 1996, chapter 4), or any ranking and selection procedure (Bechofer *et al.*, 1996) other than selection of treatments better than a control implied by Dunnett's one-sided methods. Users familiar with these methods and reasonably proficient at S-PLUS programming will be able to code many of these procedures through creative use of `multicomp` with the `comparisons = "mcc"` option.

REFERENCES

Bechhofer, Robert E., Thomas J. Santner, and David M. Goldsman (1995). *Design and Analysis of Experiments for Statistical Selection, Screening, and Multiple Comparisons.* New York: Wiley.

Duncan, D.B. (1955), "Multiple range and multiple F tests," *Biometrics* 11, 1-42.

Dunnett, C.W. (1955), "A multiple comparison procedure for comparing several treatments with a control," *J.Amer.Stat.Assoc.* 50, 1096-1121.

Edwards, Don and Berry, Jack J. (1987), "The efficiency of simulation-based multiple comparisons," *Biometrics* 43, 913-928.

Frank, B.M. (1984). "Effect of field independence-dependence and study technique on learning from a lecture," *Amer.Educ.Res.J.* 21, 669-678.

Hsu, Jason C. (1996). *Multiple Comparisons: Theory and Methods.* London: Chapman and Hall.

Hochberg, Y. and Tamhane, A.C. (1987). *Multiple Comparison Procedures.* New York: Wiley.

Juskevich, J.C. and Guyer, C.G. (1990). "Bovine growth hormone: human food safety evaluation," *Science* 249, 875-884.

Kramer, C.Y. (1956). "Extension of multiple range tests to group means with unequal numbers of replications," *Biometrics* 12, 309-310.

Keuls, M. (1952). "The use of the 'studentized range' in connection with an analysis of variance," *Euphytica* 1, 112-122.

Newman, D. (1939). "The distribution of the range in samples from a normal population, expressed in terms of an independent estimate of standard deviation," *Biometrika* 35, 16-31.

Scheffé, H. (1953). "A method for judging all contrasts in the analysis of variance," *Biometrika* 40, 87-104.

Sidak, A. (1967). "Rectangular confidence regions for the means of multivariate normal distributions," *J.Amer.Stat.Assoc.* 62, 626-633.

Toothaker, Larry E. (1993). *Multiple Comparison Procedures.* London: Sage publications.

INDEX

Symbols

Numerics

A

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y