REGRESSION AND SMOOTHING FOR CONTINUOUS RESPONSE DATA

7

A large number of problems can be analyzed using linear regression, and even more with simple transformations.

7.1 Simple Least-Squares Regression **124**
  Diagnostic Plots for Linear Models 126
7.2 Multiple Regression **129**
7.3 Adding and Dropping Terms from a Linear Model **131**
7.4 Choosing the Best Model—Stepwise Selection **137**
7.5 Updating Models **139**
7.6 Weighted Regression **139**
7.7 Prediction with the Model **142**
7.8 Confidence Intervals **144**
7.9 Robust Regression **146**
  Least Trimmed Squares Regression 147
  Least Absolute Deviation Regression 151
  M-estimates of Regression 152
7.10 Polynomial Regression **155**
7.11 Smoothing **158**
  Locally Weighted Regression Smoothing 159
  Using the Super Smoother 160
  Using the Kernel Smoother 162
  Smoothing Splines 165

  Comparing Smoothers 166
7.12 Additive Models **167**
7.13 More on Nonparametric Regression **173**
  Alternating Conditional Expectations 173
  Additive and Variance Stabilizing Transformation 177
  Projection Pursuit Regression 183
7.14 References **191**

REGRESSION AND SMOOTHING FOR CONTINUOUS RESPONSE DATA

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 linear 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. 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.

7.1 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 *l*inear *m*odel) 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 7.1. 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

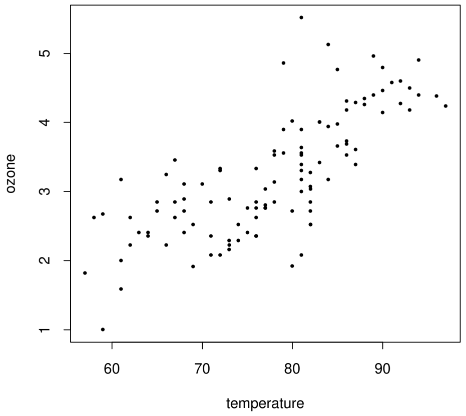

Figure 7.1: *Scatter plot of* `ozone` *against* `temperature`.

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 × `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 7.2:

```
> par(mfrow=c(2,3))
> plot(ozone.lm)
```

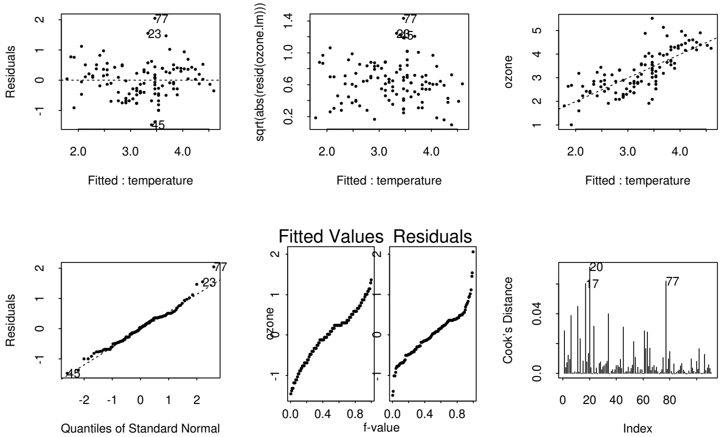

Figure 7.2: *Default plots for* `lm` *objects.*

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 7.2, 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.

7.2 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 7.3.

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
```

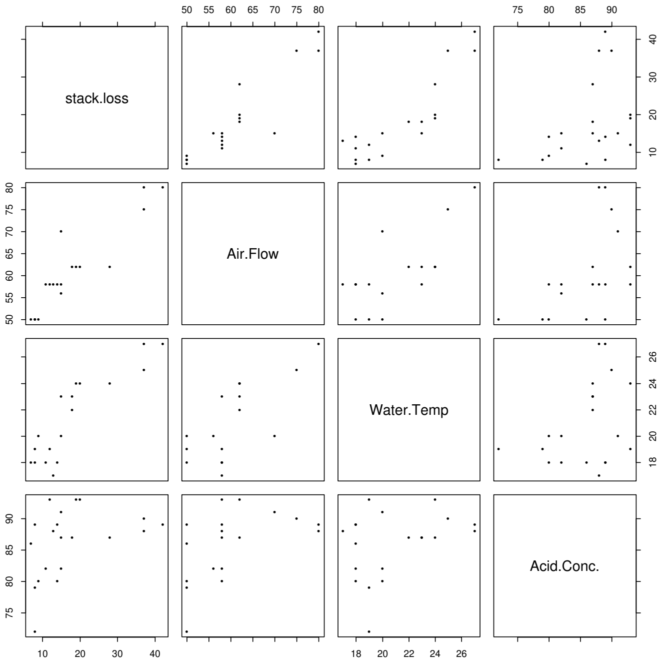

Figure 7.3: *Pairwise scatter plots of stack loss data.*

```
  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 7.4):

```
> 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 section 7.3, Adding and Dropping Terms from a Linear Model.

7.3 ADDING AND DROPPING TERMS FROM A LINEAR MODEL

In section 7.2, 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:

```
> drop1(stack.lm)
Single term deletions
Model:
stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.
```

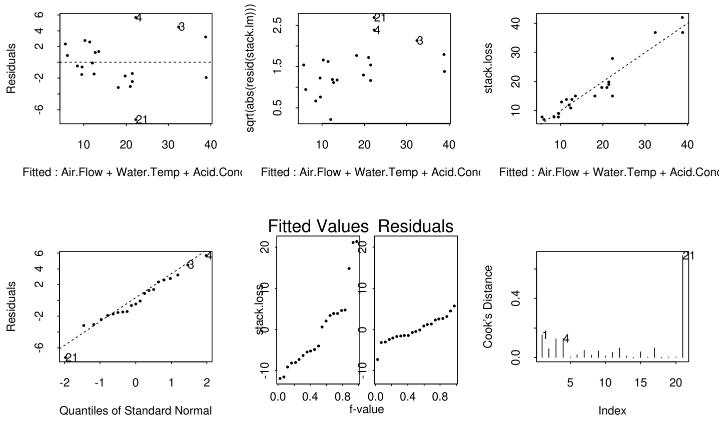

Figure 7.4: *Default plots of fitted model.*

```
           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 7.5, support the simplified model.

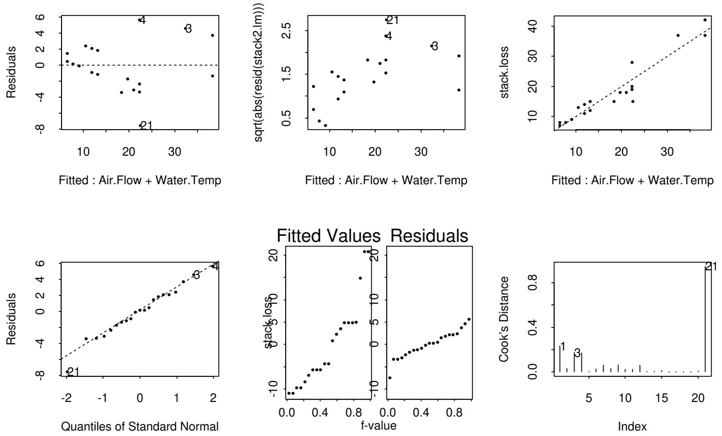

Figure 7.5: *Diagnostic plots for simplified model.*

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 7.6. The plot in the top row, third column of figure 7.6 corresponds to the scatter plot shown in figure 7.1.

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.

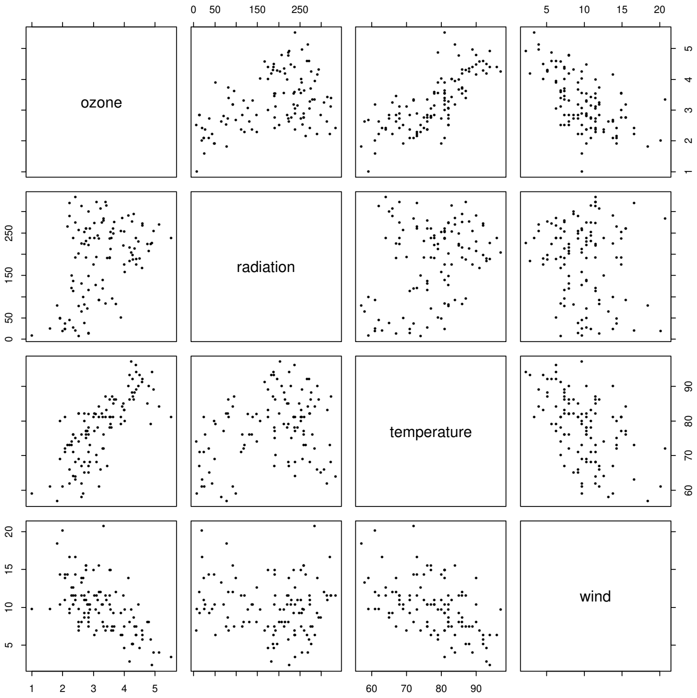

Figure 7.6: *Pairwise scatter plots for* `ozone` *data.*

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.

7.4 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 section 7.3—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 section 7.3. 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
```

7.5 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`.

7.6 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 7.1 concerning the number of courses per month and revenue generated.

Table 7.1: *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 7.1 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 7.7:

```
> par(mfrow=c(2,3))
> plot(course.lm)
```

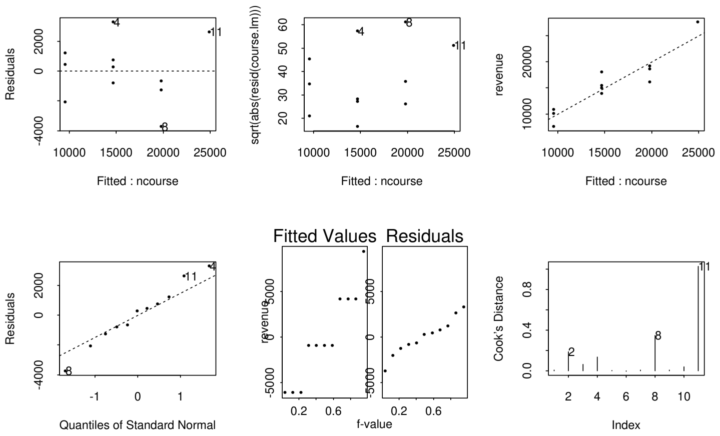

Figure 7.7: *Default diagnostic plots for unweighted course revenue model.*

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.

7.7 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)
```

We can obtain the desired predictions using `predict` as follows:

```
> predict(ozone.lm, data.frame(temperature=newtemp))
        1        2        3        4        5        6
 1.995822 2.136549 2.277276 2.418002 2.558729 2.699456
        7
 2.840183
```

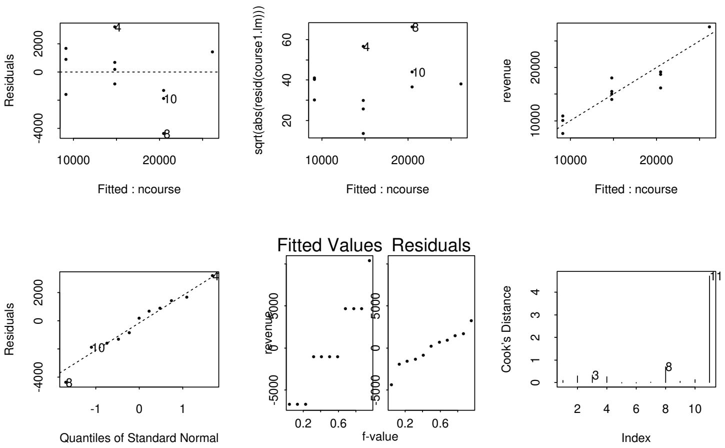

Figure 7.8: *Diagnostic plots for weighted course revenue model.*

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
 1.995822 2.136549 2.277276 2.418002 2.558729 2.699456
        7
 2.840183

$se.fit:
         1         2          3          4          5
 0.1187178 0.1084689 0.09856156 0.08910993 0.08027508
          6          7
 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 section 7.8, 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.

7.8 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):

$$\texttt{se.fit} = \hat{\sigma}\left(\frac{1}{n} + \frac{(x-\bar{x})^2}{\sum_i (x_i-\bar{x})^2}\right)^{\frac{1}{2}}$$

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
 0.08460301 0.06544499 0.06015393 0.1084689
          5
 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

$$\hat{y} - t(\alpha, n-2) \times \texttt{se.fit} < y < \hat{y} + t(\alpha, n-2) \times \texttt{se.fit}$$

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
 2.710169 3.011759 3.138615 2.42092 2.593475 2.250401
        7
 2.363895 . . .

$fit:
        1        2       3        4        5        6
 2.488366 2.840183 2.98091 2.136549 2.347639 1.925458
        7
 2.066185 . . .

$lower:
        1        2        3        4        5       6
 2.266563 2.668607 2.823205 1.852177 2.101803 1.600516
        7
 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 7.9.

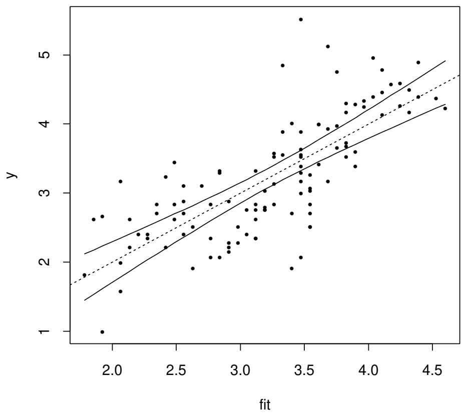

Figure 7.9: *Simultaneous confidence intervals for ozone data.*

7.9 ROBUST REGRESSION

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. These robust techniques are not yet seamlessly integrated into the standard model structure, so using them is somewhat different from fitting a classical model with lm. This section describes three of these robust techniques; we recommend the first, least trimmed squares regression, because it is robust not only with respect to outliers but also to high leverage points, which are points in the model matrix with excessive influence on the result.

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 $\hat{\boldsymbol{\beta}}_{LTS}$ minimizes the sum of the q smallest squared residuals

$$\sum_{i=1}^{q} r_{(i)}^2(\boldsymbol{\beta}) \tag{7.1}$$

where $r_{(i)}(\boldsymbol{\beta})$ 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 $\hat{\boldsymbol{\beta}}_{LS}$ minimizes the sum of all of the squared residuals.

$$\sum_{i=1}^{n} r_i^2(\boldsymbol{\beta}) \tag{7.2}$$

The least squares estimator lacks robustness because a single "observation" (y_i, $\mathbf{x}_i^T$) can cause $\hat{\boldsymbol{\beta}}_{LS}$ to take on *any* value. The same is true of M-estimators of regression, which are discussed in section 7.9.

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.lts <- ltsreg(stack.x, stack.loss)
> stack.lts
$coefficients:
 (Intercept)  Air Flow Water Temp  Acid Conc.
   -35.66978 0.7301577  0.3566785 0.006565819
```

```
$residuals:
 [1]  9.04248471  4.04905053  8.40006437  9.26849051
 [5] -0.01815256 -0.37483103  0.22909559  1.22909559
 [9] -0.45420019  0.37515288  0.31606051 -0.32069520
[13] -2.63797876 -2.06688124  0.15732218 -0.82298037
[17] -0.08773737 -0.13369810  0.50305761  2.10897972
[21] -8.17232057

$fitted.values:
 [1] 32.957515 32.950949 28.599936 18.731509 18.018153
 [6] 18.374831 18.770904 18.770904 15.454200 13.624847
[11] 13.683939 13.320695 13.637979 14.066881 7.842678
[16]  7.822980  8.087737  8.133698  8.496942 12.891020
[21] 23.172321

$objective:
[1] 0.1436481

$stock:
$stock[[1]]:
[1] 17 10 5 17 5 12 15

$births.n:
[1] 440
```

(You will probably get a slightly different answer each time you call `ltsreg`. Because the objective is hard to compute exactly, `ltsreg` uses a random algorithm.) Comparing the coefficients to those for the least-squares fit, we observe that the LTS values are noticeably different:

```
> 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.
   -35.66978 0.7301577  0.3566785 0.006565819
```

A plot of the residuals versus the fitted values for the two fits is also revealing:

```
> motif()
> par(mfrow=c(1,2))
> plot(fitted(stack.lm), resid(stack.lm),
+ ylim=range(resid(stack.lts)))
```

```
> plot(fitted(stack.lts), resid(stack.lts))
```

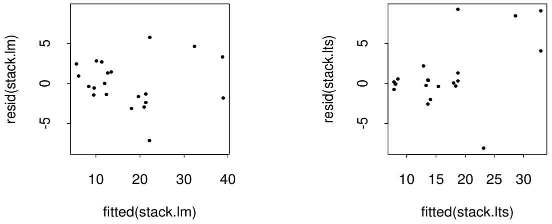

Figure 7.10: Residual plots for least-squares (*left*) and least trimmed squares regression.

The resulting plot is shown in figure 7.10. 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 $\|\hat{\boldsymbol\beta}\|$ of the resulting estimate tend to ∞. The Euclidean norm is defined as follows: $\|\hat{\boldsymbol\beta}\|^2 = \sum_{i=1}^{p} \hat{\boldsymbol\beta}_i^2$.

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 ±∞ without taking the median to ±∞.

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 $\mathbf{x}_i^T \hat{\boldsymbol\beta}_{LTS}$, $i = 1, \ldots, n$, 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 $r_i(\hat{\boldsymbol\beta}_{LTS}) = y_i - \mathbf{x}_i^T \hat{\boldsymbol\beta}_{LTS}$ will reveal the outliers quite clearly. Least squares residuals $r_i(\hat{\boldsymbol\beta}_{LS}) = y_i - \mathbf{x}_i^T \hat{\boldsymbol\beta}_{LS}$ and M-estimate residuals $r_i(\hat{\boldsymbol\beta}_M) = y_i - \mathbf{x}_i^T \hat{\boldsymbol\beta}_M$ 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 7.11.

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.

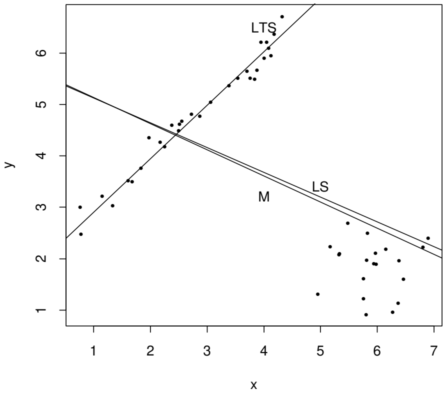

Figure 7.11: *Least trimmed squares regression, compared to least-squares and M-estimates.*

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 $\hat{\boldsymbol{\beta}}_{L1}$ that minimizes the sum of the absolute values of the residuals

$$\sum_{i=1}^{n} |r_i(\boldsymbol{\beta})| \ .$$

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))
```

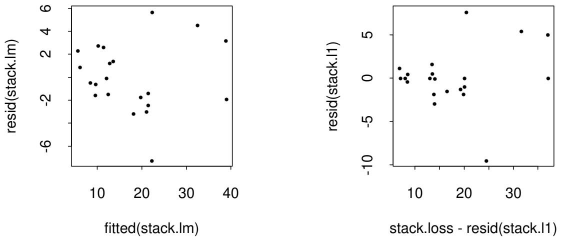

Figure 7.12: Residual plots for least-squares (*left*) and least absolute deviation regression.

The plot is shown in figure 7.12.

M-estimates of Regression

The M-estimator of regression was introduced by Huber, 1973. An M-estimate $\hat{\boldsymbol{\beta}}_M$ of regression is the $\boldsymbol{\beta}$ which minimizes

$$\sum_{i=1}^{n} \rho\left(\frac{r_i(\boldsymbol{\beta})}{\sigma}\right) \qquad (7.1)$$

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 $\hat{\boldsymbol{\beta}}_M$ 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
```

7.10 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 7.13. 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.

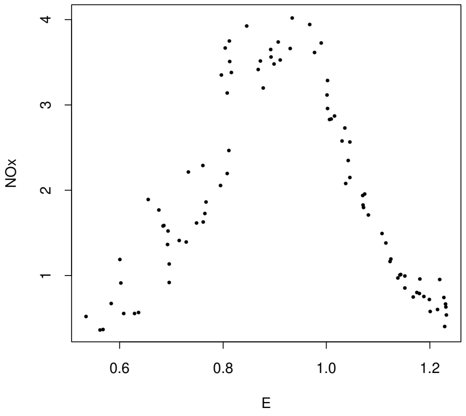

Figure 7.13: *Scatter plot showing nonlinear dependence.*

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:

$$Y = \sum_{k=0}^{n} \beta_k X_k + \varepsilon \tag{7.1}$$

where the predictors X_k enter the equation as linear terms. More generally, classical linear regression techniques apply to any equation of the form

$$Y = \sum_{k=0}^{n} \beta_k Z_k + \varepsilon \tag{7.2}$$

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:

$$Y = \sum_{k=0}^{3} \beta_k X^k + \varepsilon \tag{7.3}$$

Taking $X = X_1 = X_2 = X_3$, we can convert this to the desired form by the following assignments:

$$X_0 = Z_0$$
$$X_1 = Z_1$$
$$X_2^2 = Z_2$$
$$X_3^2 = Z_3$$

Once these assignments are made, the coefficients βk 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 7.13, 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.

7.11 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 7.14:

```
> plot(E,NOx)
> lines(loess.smooth(E,NOx))
```

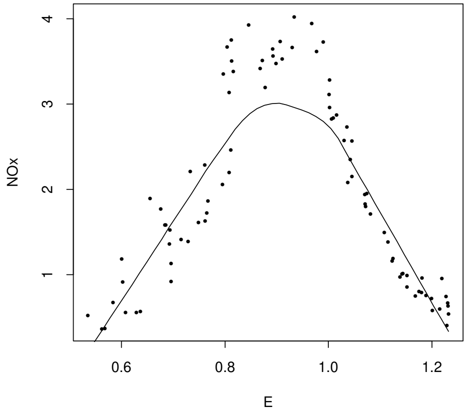

Figure 7.14: *Loess-smoothed ethanol data.*

The plot shown in figure 7.14 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.

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 7.15 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)
```

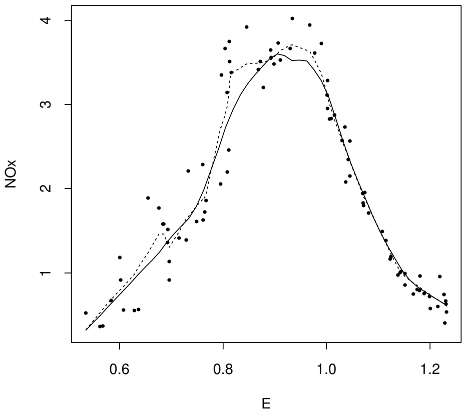

Figure 7.15: *Super Smoothed ethanol data (dotted line).*

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* x_i in predictor space, calculates a weighted average $\hat{y}_i$ of the observations in a neighborhood of the target point:

$$\hat{y}_i = \sum_{j=1}^{n} w_{ij} y_j \tag{7.4}$$

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 7.1.

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:

Table 7.1: *Kernels available for* `ksmooth`.

Kernel	Explicit form
"box"	$K_{\text{box}}(t) = \begin{cases} 1, & \lvert t\rvert \leq 0.5 \\ 0, & \lvert t\rvert > 0.5 \end{cases}$
"triangle"[1]	$K_{\text{tri}}(t) = \begin{cases} 1-\lvert t\rvert/C, & \lvert t\rvert \leq \frac{1}{C} \\ 0, & \lvert t\rvert > \frac{1}{C} \end{cases}$
"parzen"[2]	$K_{\text{par}}(t) = \begin{cases} (k_1 - t^2)/k_2, & \lvert t\rvert \leq C_1 \\ (t^2/k_3) - k_4\lvert t\rvert + k_5, & C_1 < \lvert t\rvert \leq C_2 \\ 0, & C_2 < \lvert t\rvert \end{cases}$
"normal"	$K_{\text{nor}}(t) = (1/\sqrt{2\pi}k_6)\exp(-t^2/2k_6^2)$
In convolution form, $K_{\text{tri}}(t) = K_{\text{box}} * K_{\text{box}}(t)$ 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.	

```
> 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 7.16.

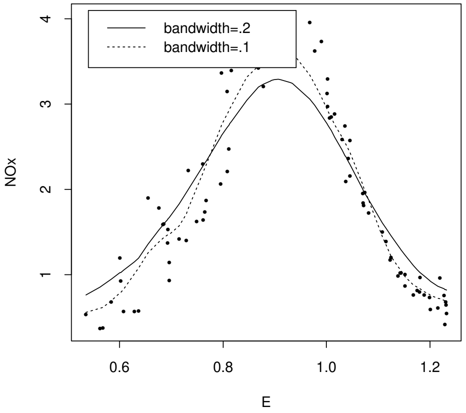

Figure 7.16: *Kernel smooth of ethanol data for two bandwidths.*

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

$$PRSS = \sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda\int(f''(t))^2dt$$

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, λ=`spar`. More generally, the relationship is given by λ=(max(x)-min(x))^3·mean(w)·`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)
```

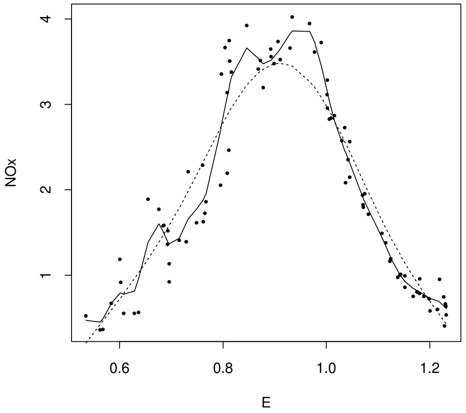

Figure 7.17: *Smoothing spline of ethanol data with cross-validation (solid line) and pre-specified degrees of freedom.*

The resulting plot is shown in figure 7.17.

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 7.18. 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.

7.12 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.

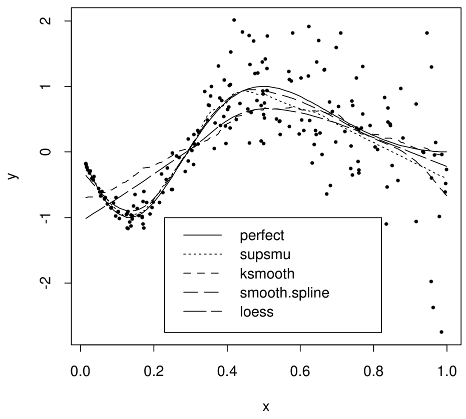

Figure 7.18: *Comparison of* S-PLUS *smoothers.*

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:

$$l(\boldsymbol{\mu}; \boldsymbol{y}) = \sum_{i=1}^{n} \log f(y_i; \mu_i)$$

The deviance $D(\boldsymbol{y}; \boldsymbol{\mu})$ is then defined as

$$\frac{D(\boldsymbol{y}; \boldsymbol{\mu})}{\phi} = 2l(\boldsymbol{\mu}^*; \boldsymbol{y}) - 2l(\boldsymbol{\mu}; \boldsymbol{y})$$

where $\boldsymbol{\mu}^*$ maximizes the log-likelihood over $\boldsymbol{\mu}$ 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 7.19 shows pairwise scatter plots for the three variables. Let's incorporate `C` as a linear term in our additive model:

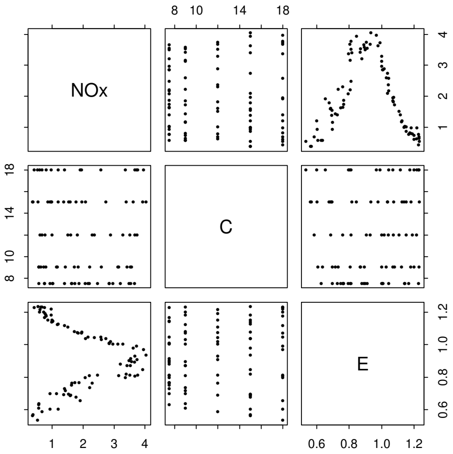

Figure 7.19: *Pairs plot of the* `ethanol` *data.*

```
> 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 the chapter Local Regression Models.

7.13 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 (*A*dditive and *VA*riance *S*tabilizing 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 $x_1, x_2,\ldots, x_p$, respectively, such that the *additive model*

$$\theta(y) = \phi_1(x_1) + \phi_2(x_2) + \cdots + \phi_p(x_p) + \varepsilon \tag{7.5}$$

is a good approximation for the data $y_i, x_{i1},\ldots, x_{ip}$, $i = 1,\ldots, n$. Let y_i, x_1, $x_2,\ldots, x_p$ be random variables with joint distribution F, and let expectations be taken with respect to F. Consider the *goodness-of-fit* measure

$$e^2 = e^2(\theta, \phi_1, \ldots,\phi_p) = \frac{E\left[\theta(y) - \sum_{k=1}^{p} \phi_k(x_k)\right]^2}{E\theta^2(y)} \tag{7.6}$$

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

$$\hat{e}^2 = \frac{\sum_{i=1}^{n}\left[\hat{\theta}(y_i) - \sum_{k=1}^{p} \hat{\phi}_k(x_{ik})\right]^2}{\sum_{i=1}^{n} \hat{\theta}^2(y_i)} \tag{7.7}$$

where $\hat{\theta}$ and the $\hat{\hat{\phi}}_j$, 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

$$\hat{\phi}_1(x_{i1} - \bar{x}_1) = (x_{i1} - \bar{x}_1)\hat{\beta}_1 ,\ldots, \hat{\phi}_p(x_{ip} - \bar{x}_p) = (x_{ip} - \bar{x}_p)\hat{\beta}_p$$

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 $R^2 = 1 - e^2_{LS}$. The transformations $\hat{\theta}$, $\hat{\phi}_1 ,\ldots, \hat{\phi}_p$ 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 $x_1,\ldots,x_p$ 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

$$y_i = e^{1 + 2x_i} + \varepsilon_i i \ , i=1,\ldots,200$$

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 × 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 7.20, 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 (7.5) 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.

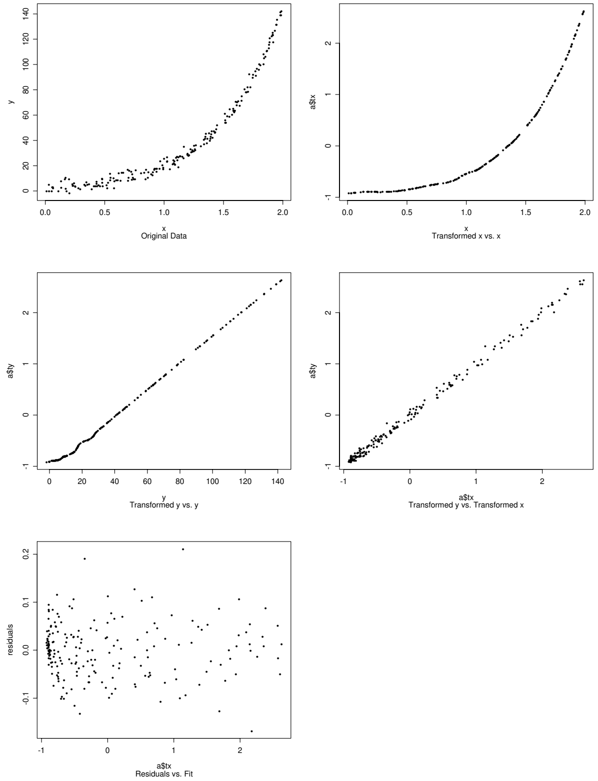

Figure 7.20: `ace` *example with additive errors*

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

$$\theta(y) = \phi_1(x_1) + \phi_2(x_2) + \cdots + \phi_p(x_p) + \varepsilon \qquad (7.8)$$

provides a good *additive* model approximation for the data y_i, x_{i1},…, x_{ip}, i=1, 2,…,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

$$E[\theta(y)|x_1, \ldots, x_p] = \sum_{i=1}^{p} \phi_i(x_i) \qquad (7.9)$$

and

$$\mathrm{var}\left[\theta(y)|\sum_{i=1}^{p} \phi_i(x_i)\right] = \text{constant} \qquad (7.10)$$

Here $E[z|w]$ is the conditional expectation of z given w. The additivity structure 7.9 is the same as for `ace`, and correspondingly the ϕ_i’s are calculated by the backfitting algorithm

$$\phi_k(x_k) = E\left[\theta(y) - \sum_{i \neq k} \phi_i(x_i)|x_k\right] \qquad (7.11)$$

cycling through k = 1, 2,…, p until convergence. The variance stabilizing aspect comes from equation (7.9). As in the case of `ace`, estimates $\hat{\theta}(y_i)$ and $\phi_j(x_{ik})$, k = 1, 2,…, p are computed to approximately satisfy equations 7.8 through 7.11, with the conditional expectations in 7.8 and 7.11 estimated using the super smoother scatterplot smoother (see `supsmu` function documentation). The equality 7.9 is approximately achieved by estimating the classic stabilizing transformation.

To illustrate the use of `avas`, construct an artificial data set with *additive* errors

$$y_i = e^{1+2x_i} + \varepsilon_i i \ , \ i=1, \ldots, 200$$

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 × 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 7.9 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 7.8. 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.

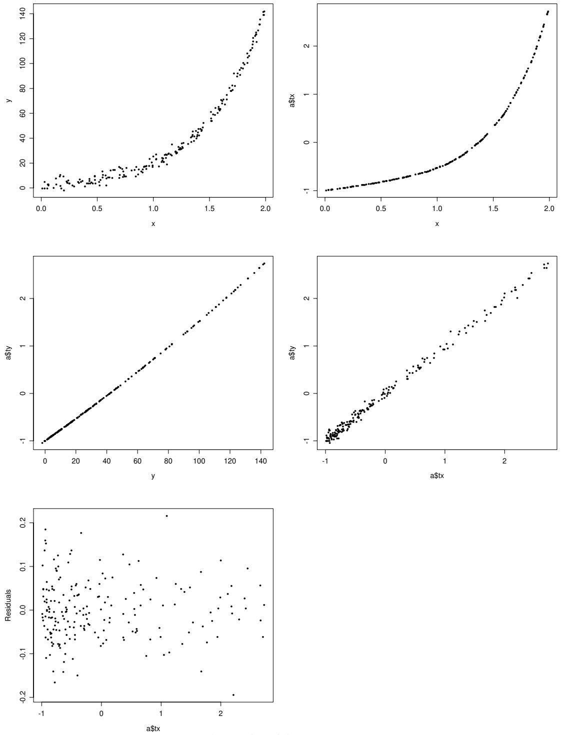

Figure 7.21: `avas` *example with additive errors.*

KEY PROPERTIES

- Suppose that the true additive model is

$$\theta^0(y) = \sum_{i=1}^{p} \phi_i^0(x_i) + \varepsilon \qquad (7.12)$$

with ε independent of $x_1, x_2, \ldots, x_p$, and var(ε) = constant. Then the iterative `avas` algorithm for 7.9 – 7.11, described below for the data versions of 7.9 – 7.11, yields a sequence of transformations $\theta^{(j)}$, $\phi_1^{(j)}$, ..., $\phi_p^{(j)}$ which converge to the true transformation θ^0, ϕ_1^0, ..., ϕ_p^0, as the number of iterations j tends to infinity. Correspondingly, the data-based version of this iteration yields a sequence of transformations $\hat{\theta}^{(j)}$, $\hat{\phi}_1^{(j)}$, ..., $\hat{\phi}_p^{(j)}$, which, at convergence, provide estimates $\hat{\theta}$, $\hat{\phi}_1$, ..., $\hat{\phi}_p$, of the true model transformations θ^0, ϕ_1^0, ..., ϕ_p^0.

- `avas` appears not to suffer from some of the anomalies of `ace`, e.g., not finding good estimates of a true additive model (equation 7.12) when normality of ε and joint normality of $\phi_1(x_1)$, …, $\phi_p(x_p)$ 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^λ of the response. The function `avas` also generalizes the Box and Tidwell (1962) procedure for choosing transformations of the carriers x_1, x_2, ..., 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 Further Details, below). 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

$$v(u) = \mathrm{VAR}\left[\hat{\theta}(y) \,\middle|\, \sum_{i=1}^{p} \phi_i(x_i) = u\right] \tag{7.13}$$

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

$$\mathrm{VAR}\left[g(\hat{\theta}(y)) \,\middle|\, \sum_{i=1}^{p} \phi_i(x_i) = u\right]$$

will be constant if g is computed according to the rule

$$g(t) = \int_c^t \frac{du}{v^{1/2}(u)} \tag{7.14}$$

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 $\hat{\theta}(y) = (y - \mathrm{E}y)/(\mathrm{VAR}^{1/2}y)$ and backfit on $x_1, \ldots, x_p$ to get $\hat{\phi}_1, \ldots, \hat{\phi}_p$. (See description of ace for details of "backfitting".)

2. *Get new transformation* of y:

   a) Compute variance function

   $$v(u) = \mathrm{VAR}\left[\hat{\theta}(y) \,\middle|\, \sum_{i=1}^{p} \hat{\phi}_i(x_i) = u\right]$$

   b) Compute variance stabilizing transformation

   $$g(t) = \int_c^t \frac{du}{v^{1/2}(u)}$$

   c) Set $\hat{\theta}(y) - g(\hat{\theta}(y))$ and standardize

   $$\hat{\theta}(y) - \frac{\hat{\theta}(y) - \mathrm{E}\hat{\theta}(y)}{\mathrm{VAR}^{1/2}\hat{\theta}(y)}$$

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*

   $$R^2 = 1 - \hat{e}^2 = 1 - \mathrm{E}\left[\hat{\theta}(y) - \sum_{i=1}^{p} \hat{\phi}_i(x_i)\right]^2 \tag{7.15}$$

   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 $\hat{\phi}_1(x_{i1}), \ldots, \hat{\phi}_p(x_{ip})$, $i = 1, \ldots, n$. 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 7.14. The integration in equation 7.14 is carried out using a trapezoidal rule.

Projection Pursuit Regression

The basic idea behind projection pursuit regression, `ppreg`, is as follows. Let y and $\mathbf{x} = (x_1, x_2, \ldots, x_p)^T$ denote the response and explanatory vector, respectively. Suppose you have observations y_i and corresponding predictors $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$, $i = 1, 2, \ldots, n$. Let $\mathbf{a}_1, \mathbf{a}_2, \ldots,$ 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 $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{M_0}$ and good nonlinear transformations $\phi_1, \phi_2, \ldots, \phi_{M_0}$ such that

$$y \approx \bar{y} + \sum_{m=1}^{M_0} \beta_m \phi_m(\mathbf{a}_m^T \mathbf{x}) \tag{7.16}$$

provides a "good" model for the data y_i, $\mathbf{x}_i$, $i = 1, 2, \ldots, n$. The "projection" part of the term projection pursuit regression indicates that the carrier vector x is *projected* onto the direction vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{M_0}$ to get the lengths $\mathbf{a}^T\mathbf{x}$, $i = 1, \ldots, n$ of the projections, and the "pursuit" part indicates that an optimization technique is used to find "good" direction vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{M_0}$.

More formally, y and $\mathbf{x}$ are presumed to satisfy the conditional expectation model

$$\mathrm{E}[y \mid x_1, x_2, \ldots, x_p] = \mu_y + \sum_{m=1}^{M_0} \beta_m \phi_m(\mathbf{a}_m^T \mathbf{x}) \tag{7.17}$$

where $\mu_y = E(y)$, and the ϕ_m have been standardized to have mean zero and unity variance:

$$\mathrm{E}\phi_m(\mathbf{a}_m^T \mathbf{x}) = 0, \quad \mathrm{E}\phi_m^2(\mathbf{a}_m^T \mathbf{x}) = 1, \quad m = 1, \ldots, M_0. \tag{7.18}$$

The observations y_i, $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^T$, $i = 1, \ldots, n$, are assumed to be independent and identically distributed random variables like y and $\mathbf{x}$, i.e., they satisfy the model in equation 7.17.

The true model parameters β_m, ϕ_m, $\mathbf{a}_m$, $m = 1, \ldots, M_0$ in equation 7.17 minimize the mean squared error

$$\mathrm{E}\left[y - \mu_y - \sum_{m=1}^{M_0} \beta_m \phi_m(\mathbf{a}_m^T \mathbf{x})\right]^2 \tag{7.19}$$

over all possible β_m, ϕ_m, and $\mathbf{a}_m$.

Equation 7.17 includes the additive `ace` models under the restriction $\theta(y) = y$. This occurs when $M_0 = p$ and $\mathbf{a}_1 = (1, 0, \ldots, 0)^T$, $\mathbf{a}_2 = (0, 1, 0, \ldots, 0)^T$, $\mathbf{a}_p = (0, 0, \ldots, 0, 1)^T$, 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 $\mathbf{x}$ are independent with mean 0 and variance 1. Then $\mathbf{a}^T = (b_1, \ldots, b_p)/\sqrt{b_1^2 + \cdots + b_p^2}$, $\phi_1(t) = t$, 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 7.17 includes the possibility of having *interactions* between the explanatory variables. For example, suppose that

$$E[y \mid x_1, x_2] = x_1 x_2. \tag{7.20}$$

This is described by 7.17 with $\mu_y = 0$, $M_0 = 2$, $\beta_1 = \beta_2 = \frac{1}{4}$, $\mathbf{a}_1^T = (1, 1)$, $\mathbf{a}_2^T = (1, -1)$, $\phi_1(t) = t^2$, and $\phi_2(t) = -t^2$. For then

$$\phi_1(\mathbf{a}_1^T \mathbf{x}) = (x_1 + x_2)^2 = x_1^2 + 2x_1x_2 + x_2^2$$

$$\phi_2(\mathbf{a}_2^T \mathbf{x}) = -(x_1 - x_2)^2 = -x_1^2 + 2x_1x_2 - x_2^2$$

so that

$$\sum_{m=1}^{2} \beta_m \phi_m(\mathbf{a}^T \mathbf{x}) = x_1 x_2.$$

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 7.22) 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.

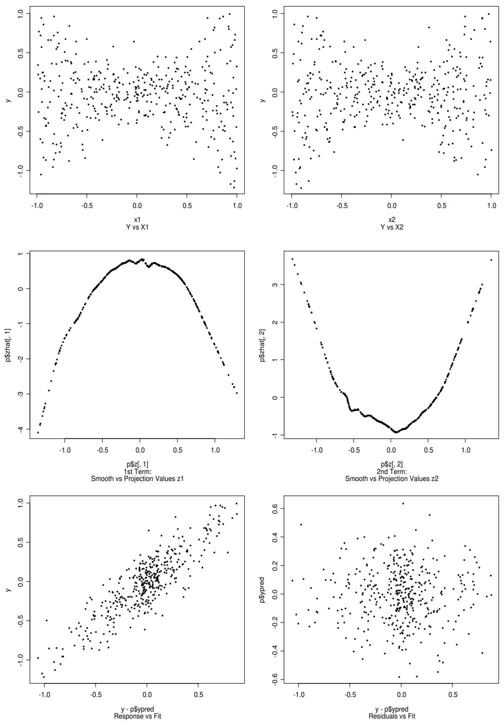

Figure 7.22: *Projection pursuit example.*

FURTHER DETAILS

The Forward Stepwise Procedure

An initial M-term model of the form given by the right-hand side of equation 7.17, with the constraints of equation 7.18 and $M > M_0$, is estimated by a forward stepwise procedure, as described by Friedman and Stuetzle (1981).

First, a trial direction $\mathbf{a}_1$ is used to compute the values $z_{i1} = \mathbf{a}_1^T\mathbf{x}_i$, $i = 1, \ldots, n$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^T$. Then, with $\tilde{y}_i^1 = y_i - \bar{y}$, you have available a scatter plot of data $(\tilde{y}_i, z_{i1})$, $i = 1, \ldots, n$, 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, $z_{i1} = \mathbf{a}_1^T\mathbf{x}_i$. Super Smoother is used for this purpose; see documentation for `supsmu`. This $\hat{\phi}_1$ depends upon the trial direction vector $\mathbf{a}_1$, so we write $\phi_1 = \phi_{1,\mathbf{a}_1}$. Now $\mathbf{a}_1$ is varied to minimize the weighted sum of squares,

$$\sum_{i=1}^{n} w_i[y_i - \hat{\phi}_{1,\mathbf{a}_1}(z_{i1})]^2 \qquad (7.21)$$

where for each $\mathbf{a}_1$ in the optimization procedure, a new $\hat{\phi}_{1,\mathbf{a}_1}$ 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 $\hat{\mathbf{a}}_1$ and $\hat{\phi}_1$, where $\hat{\phi}_1$ has been standardized according to equation 7.18 and the corresponding value $\hat{\beta}_1$ is computed. We now have the approximation

$$y_i \approx \bar{y} + \hat{\beta}_1\hat{\phi}_1(\hat{\mathbf{a}}_1^T\mathbf{x}_i) \quad i = 1, \ldots, n.$$

Next we treat $y_i^{(2)} = y_i - \bar{y} - \hat{\beta}_1\hat{\phi}_1(z_{i1})$ as the response, where now $z_{i1} = \hat{\mathbf{a}}_1^T\mathbf{x}_i$, and fit a second term $\hat{\beta}_2\hat{\phi}_2(\mathbf{z}_{i2})$, where $z_{i2} = \hat{\mathbf{a}}_2^T\mathbf{x}_i$, to this modified response, in exactly the same manner that we fitted $\hat{\beta}_1\hat{\phi}_1(\hat{\mathbf{a}}_1^T\mathbf{x}_i)$ to $\mathbf{y}_i^{(1)}$. 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

$$y_i \approx \bar{y} + \sum_{m=1}^{M} \hat{\beta}_m\hat{\phi}_m(z_{im}),\ i = 1, \ldots, n\,. \qquad (7.22)$$

where $z_{im} = \hat{\mathbf{a}}_m^T\mathbf{x}_i$, $m = 1, \ldots, M$.

The Backward Stepwise Procedure

Having fit the M term model in equation 7.22 in a forward stepwise manner, `ppreg` fits all models of decreasing order $m = M - 1, M - 2, \ldots, M_{min}$, where M and M_{min} are user-specified. For each term in the model, the weighted sum of squared residuals

$$\mathrm{SSR}(m) = \sum_{i=1}^{n} w_i\left[y_i - \bar{y} - \sum_{l=1}^{m} \beta_l\phi_l(\mathbf{a}_l^T\mathbf{x}_i)\right]^2 \qquad (7.23)$$

is minimized through the choice of β_l, $\mathbf{a}_l$, ϕ_l, $l = 1, \ldots, m$. The initial values for these parameters, used by the optimization algorithm which minimizes equation 7.23, 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

$$\boldsymbol{I}_l = |\hat{\beta}_l| \quad l = 1, \ldots, m + 1 \qquad (7.24)$$

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 $p \le 4$, you might well choose $M \ge p$. For large p, you would probably choose $M < p$, hoping for a parsimonious representation.

For each order m, $1 \le m \le M$, `ppreg` will evaluate the fraction of unexplained variance

$$e^2(m) = \frac{\mathrm{SSR}(m)}{\sum_{i=1}^{n} w_i[y_i - \bar{y}]^2}$$

$$= \frac{\sum_{i=1}^{n} w_i\left[y_i - \bar{y} - \sum_{l=1}^{m} \hat{\beta}_l \hat{\phi}_l(\hat{\mathbf{a}}_l^T \mathbf{x}_i)\right]^2}{\sum_{i=1}^{n} w_i[y_i - \bar{y}]^2}$$

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 (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 $\hat{\beta}_l$, $\hat{\mathbf{a}}_l$, $\hat{\phi}_l(z_{il})$, $z_{il} = \hat{\mathbf{a}}_l^T \mathbf{x}_i$, $i = 1, \ldots, n$, $l = 1, \ldots, m$, and $e^2(m)$ 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 $e^2(m)$, $m = 1, \ldots, M$ (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

$$y_j \approx \bar{y}_j + \sum_{m=1}^{M_0} \hat{\beta}_{mj} \hat{\phi}_m(\hat{\mathbf{a}}_m^T \mathbf{x}) \qquad (7.25)$$

by minimizing the multivariate response weighted sum of squared residuals

$$\mathrm{SSR}_q(m) = \sum_{j=1}^{q} W_j \sum_{i=1}^{n} w_i\left[y_{ij} - \bar{y}_j - \sum_{l=1}^{m} \hat{\beta}_{lj} \hat{\phi}_l(\mathbf{a}_l^T \mathbf{x}_i)\right]^2 \qquad (7.26)$$

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 $\hat{\beta}_j = (\hat{\beta}_{ij}, \ldots, \hat{\beta}_{mj})^T$, $j = 1, \ldots, q$.

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

$$I_l = \sum_{j=1}^{q} W_j |\hat{\beta}_{jl}| \qquad l = 1, \ldots, m + 1$$

The most important terms are the ones with the largest I_l, and the corresponding values of $\hat{\beta}_{jl}$, $\hat{\phi}_l$, and $\mathbf{a}_l$ 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

$$e_q^2(m) = \frac{\mathrm{SSR}_q(m)}{\sum_{j=1}^{q} W_j \sum_{i=1}^{n} w_i [y_{ij} - \bar{y}_j]^2} \qquad (7.27)$$

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.

7.14 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.