A Package of C and Fortran Routines for Fitting Local Regression Models

William S. Cleveland   Eric Grosse   Ming-Jen Shyu

August 20, 1992

Local regression models provide methods for fitting *regression functions*, or *regression surfaces*, to measurements of two or more variables. One variable is a response, the others are factors, and a function is fitted to the data to explain how the response depends on the factors. Two examples are shown in Figures 1 and 2 In the first figure, E is the factor, NO_x is the response, and the fitted function is

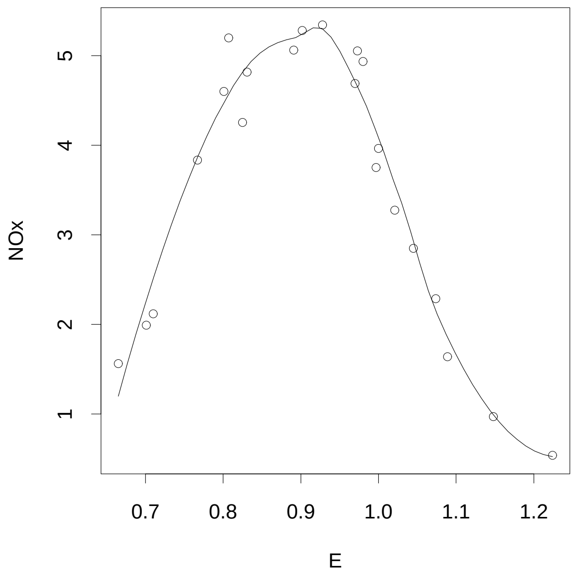

Figure 1: Local regression model with one factor—fitted curve.

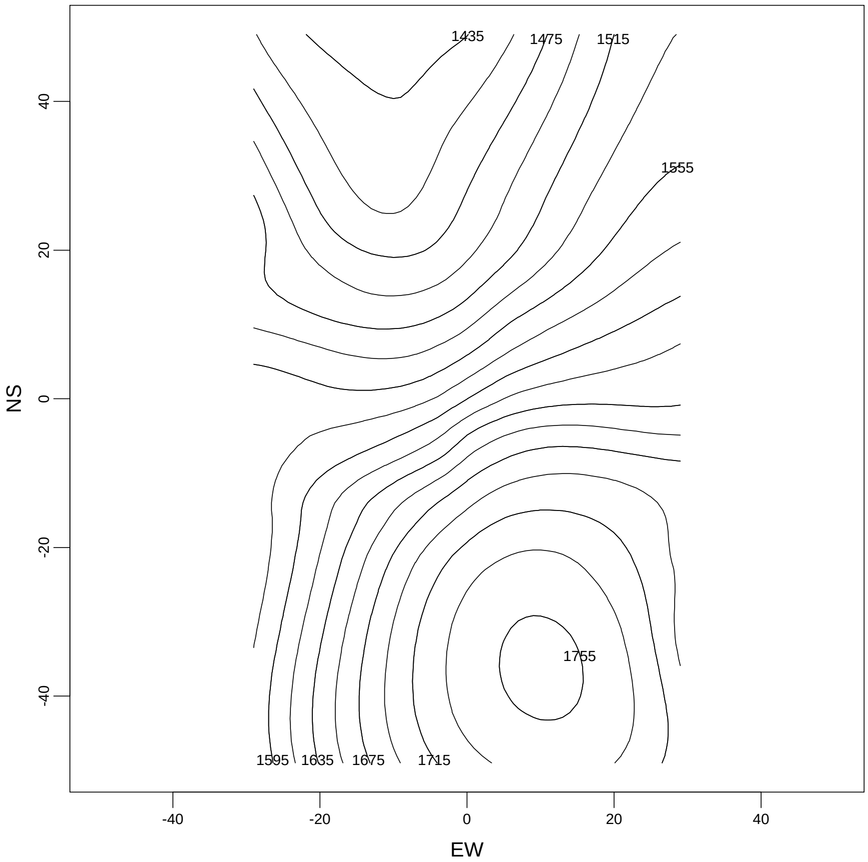

Figure 2: Local regression model with two factors—contours of fitted surface.

shown by the curve. In the second figure, east-west and south-north are the factors, velocity is the response, and the fitted surface is shown by a contour plot. These two examples will be explained in detail later.

This handbook describes a collection of public-domain programs, written in C and Fortran, that carry out such function fitting using *loess*, a method based on local regression. The appendix at the end of this document describes how the programs may be obtained electronically. Most users will want to use the code by writing C programs that access the high-level C routines of loess. This handbook shows how to do that.

Consider any point x in the space of the factors. One basic specification in a local regression model is that there is a neighborhood containing x in which the regression surface is well approximated by a function from a specific parametric class; for the loess implementation described in this handbook, there will be two classes—polynomials of degree 1 or 2. The loess method method consists of fitting polynomials locally, in a moving fashion, and thus amounts to smoothing the response as a function of the factors.

The handbook instructs by doing. Data are analyzed using the code; results of the analyses as well as the code that produces them are described. This shows both how loess works and how the code works. Some of the analysis is graphical, but no graphics code is provided. Users are on their own to interface this code with a graphics package; such tools are essential for assessing and interpreting the functions that are fitted.

1 Statistical Models and Fitting

1.1 Definition of Local Regression Models

Suppose, for each i from 1 to n, that y_i is a measurement of the response and x_i is a corresponding vector of measurements of p factors. In a regression model the response and factors are related by

$$y_i = g(x_i) + \varepsilon_i,$$

where g is the regression surface and the ε_i are random errors. If x is any point in the space of the factors, $g(x)$ is the value of the surface at x; for example, $g(x_i)$ is the expected value of y_i. In the fitting of local regression models we specify properties of the regression surface and the errors; that is, we make assumptions about them. We will now discuss the specifications that are allowable using the C routines and data structures that are described in Section 2.

Specification of the Errors

In all cases, we suppose that the ε_i are independent random variables with mean 0. One of two families of probability distributions can be specified. The first is the Gaussian. The second is symmetric distributions, which allow for the common situation where the errors have a distribution with tails that are stretched out compared with the normal (leptokurtosis), and which lead us to robust methods of estimation.

We can specify properties of the variances of the ε_i in one of two ways. The first is simply that they are a constant, σ^2. The second is that $a_i\varepsilon_i$ has constant variance σ^2, where the *a priori weights*, a_i, are positive and known.

Specification of the Surface

For each x in the space of the factors, we suppose that in a certain neighborhood of x, the regression surface is well approximated by a function from a parametric class. The overall sizes of the neighborhoods are specified by a parameter, α, that is defined in Section 1.2. Size, of course, implies a metric, and we will use Euclidean distance. For two or more factors, the shapes of the neighborhoods are specified by deciding whether to normalize the scales of the factors. We will elaborate on this later.

We will allow the specification of one of two general classes of parametric functions: linear and quadratic polynomials. For example, suppose there are two factors, u and v. If we specify linear, the class consists of three monomials: a constant, u, and v. If we specify quadratic, the class is made up of five monomials: a constant, u, v, uv, u^2, and v^2. We will let λ be a parameter that describes the specification; if $\lambda = 1$, the specification is linear, and if $\lambda = 2$, the specification is quadratic.

Suppose $\lambda = 2$ and there are two or more factors. We can specify that any of the monomials that is a square be dropped from the class. For example, suppose again that the factors are u and v. If we drop the square for u, then the class has four monomials: a constant, u, v, uv, and v^2.

If there are two or more factors we can specify that the surface be *conditionally parametric* in any proper subset of the factors; this means that given the values of the factors not in the subset, the surface is a member of a parametric class as a function of the subset. If we change the conditioning, or given values, the surface is still a function in the same class, although the parameters might change. For example, suppose the factors are u and v. Suppose $\lambda = 1$, and we specify the surface to be conditionally parametric in u. Then given v, the surface is linear in u; this means the general form of the surface is $\beta_0(v) + \beta_1(v)u$. Suppose $\lambda = 2$, and we specify the surface to be conditionally parametric in u. Then given v, the surface is quadratic in u; the general form of the surface in this case is $\beta_0(v)+\beta_1(v)u+\beta_2(v)u^2$. It makes sense to specify a regression surface to be conditionally parametric in one or more variables if exploration of the data or a priori information suggests that the surface is globally a very smooth function of the variables. Making such a specification when it is valid can result in a more parsimonious description of the surface.

Summary of the Choices

Thus, the fitting of local regression models involves making the following choices about the specification of properties of the errors and the regression surface:

- Gaussian or symmetric distribution;
- constant variance or a priori weights;
- locally linear or locally quadratic in the factors;
- neighborhood size;
- normalization of the scales;
- dropping squares;
- conditionally parametric subset.

1.2 Loess: Fitting Local Regression Models

The method we will use to fit local regression models is called *loess*, which is short for *local regression*, and was chosen as the name since a loess is a deposit of fine clay or silt along a river valley, and thus is a surface of sorts. The word comes from the German *löss*, and is pronounced *lṓis*.

Identically Distributed, Gaussian Errors: One Numeric Factor

Let's begin with the classical case of Gaussian errors with constant variance σ^2. Suppose there is just one factor. Let x be any value along the scale of measurement of the variable. The loess fitting procedure is a numerical algorithm that prescribes how $\hat{g}(x)$, the estimate of g at a specific value of x, is computed.

Let $\Delta_i(x) = |x - x_i|$, let $\Delta_{(i)}(x)$ be the values of these distances ordered from smallest to largest, and let

$$T(u;t) = \begin{cases} (1-(u/t)^3)^3, & \text{for } 0 \le u < t \\ 0 & \text{for } u \ge t \end{cases}$$

be the *tricube weight function.*

The smoothness of the loess fit depends on the specification of the neighborhood parameter, $\alpha > 0$. As α increases, $\hat{g}$ becomes smoother. Suppose $\alpha \le 1$. Let q be equal to αn truncated to an integer. We define a weight for (x_i, y_i) by

$$w_i(x) = T(\Delta_i(x); \Delta_{(q)}(x)).$$

For $\alpha > 1$, the $w_i(x)$ are defined in the same manner, but $\Delta_{(q)}(x)$ is replaced by $\Delta_{(n)}(x)\alpha$. The $w_i(x)$, which we will call the *neighborhood weights*, decrease or stay constant as x_i increases in distance from x.

If we have specified the surface to be locally well approximated by a linear polynomial—that is, if λ is 1—then a linear polynomial is fitted to y_i using weighted least squares with the weights $w_i(x)$; the value of this fitted polynomial at x is $\hat{g}(x)$. If λ is 2, a quadratic is fitted. Note that as $\alpha \to \infty$, $\hat{g}(x)$ tends to a linear surface for locally linear fitting or a quadratic surface for locally quadratic fitting.

Identically Distributed, Gaussian Errors: Two or More Numeric Factors

We continue to suppose the errors are identically distributed and Gaussian. The one additional issue that needs to be addressed for p factors with $p > 1$ is the notion of distance in the space of the factors. Suppose x is a value in the space. To define neighborhood weights we need to define the distance, $\Delta_i(x)$, from x to x_i, the ith observation of the factors. We will use Euclidean distance, but the x_i do not have to be the raw measurements. Typically, it makes sense to take x_i to be the raw measurements normalized in some way. We will normalize the factors by dividing them by their 10% trimmed sample standard deviation, and call this the *standard normalization*. There are, however, situations where we might choose not to normalize—for example, if the factors represent position in space.

Armed with the $\Delta_i(x)$, the loess fitting method for $p > 1$ is just an obvious generalization of the one-factor method. For $\alpha < 1$, neighborhood weights, $w_i(x)$, are defined using the same formulas used for one factor; thus, if $\lambda = 1$, we fit a linear polynomial in the factors using weighted least squares, or, if $\lambda = 2$, we fit a quadratic. For $\alpha > 1$, the $w_i(x)$ are defined by the same formula except that $\Delta_{(q)}(x)$ is replaced by $\Delta_{(n)}(x)\alpha^{1/p}$.

Dropping Squares and Conditionally Parametric Fitting for Two or More Factors

Suppose λ has been specified to be 2. Suppose, in addition, that we have specified the squares of certain factors to be dropped. Then those monomials are not used in the local fitting.

Suppose a proper subset of the factors has been specified to be conditionally parametric. Then we simply ignore these factors in computing the Euclidean distances that are used in the definition of the neighborhood weights, $w_i(x)$. It is an easy exercise to show that this results in a conditionally parametric fit.

Symmetric Errors and Robust Fitting

Suppose the ε_i have been specified to have a symmetric distribution. Then we modify the loess fitting procedures to produce a robust estimate; the estimate is not adversely affected if the errors have a long-tailed distribution, but it has high efficiency in the Gaussian case.

The loess robust estimate begins with the Gaussian-error estimate, $\hat{g}(x)$. Then the residuals

$$\hat{\varepsilon}_i = y_i - \hat{g}(x_i)$$

are computed. Let

$$B(u;b) = \begin{cases} (1-(u/b)^2)^2 & \text{for } 0 \le |u| < b \\ 0 & \text{for } |u| \ge b \end{cases}$$

be the *bisquare weight function*. Let

$$m = \text{median}(|\,\hat{\varepsilon}_i\,|)$$

be the median absolute residual. The *robustness weights* are

$$r_i = B(\hat{\varepsilon}_i; 6m).$$

An updated estimate, $\hat{g}(x)$, is computed using the local fitting method, but with the neighborhood weights, $w_i(x)$, replaced by $r_i w_i(x)$; thus, points (x_i, y_i) with large residuals receive reduced weight. Then new residuals are computed and the procedure is repeated. The final robust estimate is the result of updating the initial estimate several times.

Errors with Unequal Scales

Suppose we specify that $a_i\varepsilon_i$ have constant variance σ^2. Then, for the Gaussian-error estimate, the neighborhood weight, $w_i(x)$, is replaced by $a_i w_i(x)$, and for the robust estimate, the weight $r_i w_i(x)$ is replaced by $a_i r_i w_i(x)$.

2 C Functions

This section describes the C functions for local regression modeling. In each subsection we analyze a dataset, illustrating how the C functions are used to fit models. We also use graphics to explore the data and carry out graphical diagnostics to check the specifications of the fitted models. Our goal is to show how the data are analyzed in practice using the C routines and graphics, and how each dataset presents a different challenge. We begin, however, by rapidly running through the C functions for fitting and inference to give an overview; the reader need not understand details at this point.

The basic modeling function is `loess()`. Let's apply it to some madeup data consisting of 100 observations of three variables, one response and two factors. We will fit a Gaussian model with the smoothing parameter, α, equal to 0.5 and the degree, λ, of the locally-fitted polynomial equal to 1:

```
#include <stdio.h>
#include "loess.h"

struct  loess_struct    madeup;
long    n = 100, p = 2;
double  one_two[] = {-0.957581, -2.80955, -0.696511, ...};
double  response[] = {14.4536, 6.62283, 13.6714, ...};

main() {
        loess_setup(one_two, response, n, p, &madeup);
        madeup.model.span = 0.5;
        loess(&madeup);
        loess_summary(&madeup);
}
```

Compiling, linking with the loess library, and executing gives us the output:

Number of Observations: 100
Equivalent Number of Parameters: 14.9
Residual Standard Error: 0.9693

The *equivalent number of parameters*, μ, measures the amount of smoothing, as defined in Section 4.1, and is analogous to the number of parameters in a parametric fit. Also shown is an estimate of σ, the standard error of the residuals.

Notice that `one_two[]`, the vector of the two predictors is of length `(n * p)`. One can think of it as a concatenated vector of all the predictor vectors, in which the j-th coordinate of the i-th point is in `one_two[i+n*j]`, where $0 \leq j < p$, $0 \leq i < n$. We will adhere to this rule of defining the input data structure for both `loess()` and `predict()` in all the subsequent examples (please see the on-line documentation file, `struct.m`, for further detail on the input data structure).

Let's modify the fit by dropping the square of the first factor, making it conditionally parametric, and increase α to 0.8:

```
struct  loess_struct    madeup_new;

loess_setup(one_two, response, n, p, &madeup_new);
madeup_new.model.span = 0.8;
madeup_new.model.drop_square[0] = TRUE;
madeup_new.model.parametric[0] = TRUE;
loess(&madeup_new);
loess_summary(&madeup_new);
```

Number of Observations: 100
Equivalent Number of Parameters: 6.9
Residual Standard Error: 1.4804

The purpose of splitting the invocation into two function calls is now clear. The role of `loess_setup()` is to look at the input data and set a host of internal parameters and options to default values. The analyst assigns to various fields of the `loess_struct` object to override the defaults as desired, then calls `loess()` to perform the main calculation. Until now we have been fitting Gaussian models because the argument that controls this, `madeup_new.model.family`, defaults to `"gaussian"`. Now let us fit a model with the error distribution specified to be symmetric:

```
madeup_new.model.family = "symmetric";
loess(&madeup_new);
loess_summary(&madeup_new);
```

Number of Observations: 100
Equivalent Number of Parameters: 6.9
Residual Scale Estimate: 1.0868

Also, we have been using the standard normalization to normalize the scales of the two factors; this is controlled by the argument `madeup_new.model.normalize`, whose value in the above models has been `1`. Let's now remove the normalization:

```
madeup_new.model.normalize = FALSE;
```

We do not need to see the output this time.

The function `predict()` can be used to evaluate a fitted surface at a set of points in the space of the factors: For the madeup data, the range of the first factor is -2.809549 to 3.451000 and the range of the second factor is -1.885139 to 1.859246. Let us evaluate the current surface at the following values of the two factors: (-2.5, 0), (0,0), and (2.5,0).

```
struct  pred_struct     madeup_pred;
double  newdata1[] = {-2.5, 0, 2.5, 0., 0., 0.};
long    m = 3, se_fit = FALSE;

predict(newdata1, m, &madeup, &madeup_pred, se_fit);
printf("%g %g %g\n", madeup_pred.fit[0],
       madeup_pred.fit[1], madeup_pred.fit[2]);
```

8.15678 14.4936 14.8541

The function `predict()` can also be used to compute information about standard errors, by setting its `se` input argument to `TRUE`. We will evaluate the standard errors at the following two values of the factors: (-0.5, 0) and (0.5, 0).

```
double  newdata2[] = {-0.5, 0.5, 0., 0.};

m = 2;
se_fit = TRUE;
predict(newdata2, m, &madeup, &madeup_pred, se_fit);
printf("%g %g\n", madeup_pred.fit[0], madeup_pred.fit[1]);
printf("%g %g\n", madeup_pred.se_fit[0], madeup_pred.se_fit[1]);
```

```
printf("%g\n", madeup_pred.residual_scale);
printf("%g\n", madeup_pred.df);
```

14.4918 14.3897
0.276746 0.278009
0.969302
81.23189

The components are `fit`, the evaluated surface; `residual_scale`, the estimate of the residual scale; `df`, the degrees of freedom of the t distribution upon which the confidence intervals are based; and `se_fit`, estimates of the standard errors of the fit. Now we can use `pointwise()` to compute upper and lower confidence intervals:

```
struct  ci_struct       madeup_ci;
double  coverage = .99;
int     i;

pointwise(&madeup_pred, m, coverage, &madeup_ci);
for(i = 0; i < m; i++)
        printf("%g ", madeup_ci.upper[i]);
printf("\n");
for(i = 0; i < m; i++)
        printf("%g ", madeup_ci.fit[i]);
printf("\n");
for(i = 0; i < m; i++)
        printf("%g ", madeup_ci.lower[i]);
printf("\n");
```

15.2218 15.1230
14.4918 14.3897
13.7618 13.6564

The computations of `predict()` that produce the standard errors are much more costly than those that evaluate the surface, so the number of points at which standard errors are computed should be modest compared to those at which we do evaluations; this is not a limitation for the practice of local regression modeling since it makes statistical and graphical sense to compute intervals at a limited set of points.

In our first fit to the made-up data we took `span` to be 1/2. Can we increase it and still get a good fit? The best way to check is to use graphical diagnostics, but the analysis of variance can also provide some guidance:

```
struct  loess_struct    madeup2;
struct  anova_struct    madeup_anova;

loess_setup(one_two, response, n, p, &madeup2);
madeup2.model.span = 0.75;
loess(&madeup2);
```

```
anova(&madeup2, &madeup, &madeup_anova);
printf("%g %g %g %g\n", madeup_anova.dfn, madeup_anova.dfd,
       madeup_anova.F_value, madeup_anova.Pr_F);
```

6.45034 81.2319 2.85933 0.0121449

The results suggest that the increase in span has led to a distortion.

2.1 Gas Data

Our first data set is the gas data, 22 observations of two variables from an industrial experiment that studied exhaust from an experimental one-cylinder engine (Brinkman, 1981). The dependent variable, which will be denoted by NO_x, is the concentration of nitric oxide, NO, plus the concentration of nitrogen dioxide, NO_2, normalized by the amount of work of the engine. The units are μg of NO_x per joule. The factor is the equivalence ratio, E, at which the engine was run. E is a measure of the richness of the air and fuel mixture.

Data Exploration

We begin our analysis with an exploration of the data by the scatterplot of NO_x against E in Figure 3. The plot shows that there is substantial curvature as a function of E and that the errors have a small variance compared with the change in the level of NO_x.

Fitting a First Model

Because of the substantial curvature in the overall pattern of the data, we will fit a local regression model using locally quadratic fitting. A reasonable starting point for the smoothing parameter is $\alpha = 2/3$. Also, because variation about the overall pattern shows no unusual behavior, we begin with the hope that an assumption of Gaussian errors is reasonable:

```
struct  loess_struct    gas;
double  E[] = {0.831, 1.045, 1.021, ...},
        NOx[] = {4.818, 2.849, 3.275, ...};
long    n = 22, p = 1;

loess_setup(E, NOx, n, p, &gas);
gas.model.span = 2.0 / 3.0;
loess(&gas);
loess_summary(&gas);
```

Number of Observations: 22
Equivalent Number of Parameters: 5.5
Residual Standard Error: 0.3404

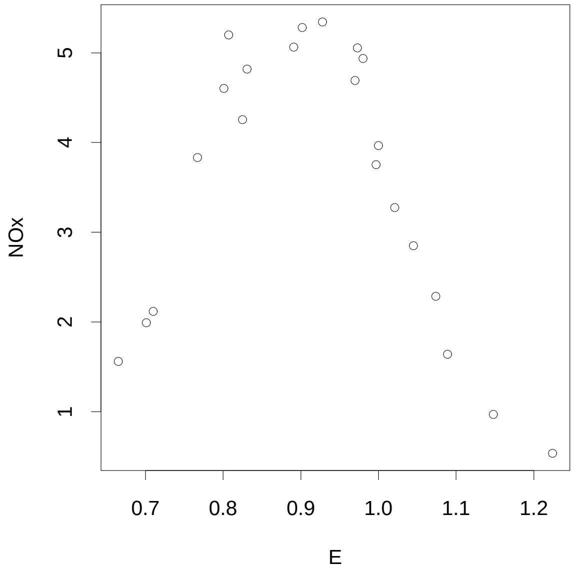

Figure 3: Gas data—NO_x against E.

The equivalent number of parameters of the fit is 5.5. The estimate of the residual variance is 0.3404, but we should not take this estimate seriously before carrying out the diagnostic procedures to come.

Evaluation and Plotting the Curve

Having fitted a model to gas, we will compute $\hat{g}(x)$ at the following values of the factor, E: 0.665 0.949 1.224. These are, respectively, the minimum measurement of E, the median, and the maximum.

```
struct  pred_struct     gas_pred;
double  gas_fit_E[] = {0.665, 0.949, 1.224};
long    m = 3, se_fit = FALSE;
int     i;

predict(gas_fit_E, m, &gas, &gas_pred, se_fit);
for(i = 0; i < m; i++)
        printf("%g ", gas_pred.fit[i]);
printf("\n");
```

1.19641 5.06875 0.523682

We could compute the fitted values, $\hat{y}_i = \hat{g}(x_i)$, by:

```
predict(E, 22,  &gas, &gas_pred, se_fit);
```

However they, as well as the residuals, $y_i - \hat{y}_i$, are stored on the loess output substructure, `gas.out`.

```
gas.out.fitted_values
gas.out.fitted_residuals
```

To study the fitted curve we can evaluate it at 50 equally spaced points from the minimum of E to the maximum and plot it. The result is shown in Figure 4.

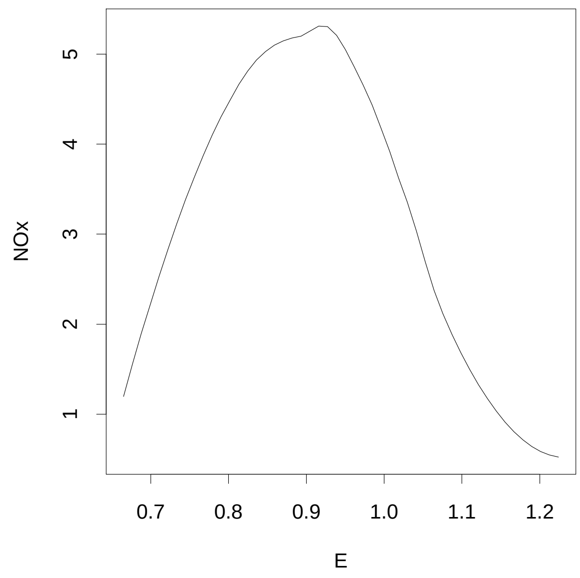

Figure 4: Gas data—local regression fit.

Diagnostic Checking

We turn now to diagnostic checking to accept or reject the specifications of the model we have fitted. To check the properties of $g(x)$ that are specified by the choice of $\alpha = 2/3$ and $\lambda = 2$, we plot the residuals, $\hat{\varepsilon}_i$, against E to look for lack

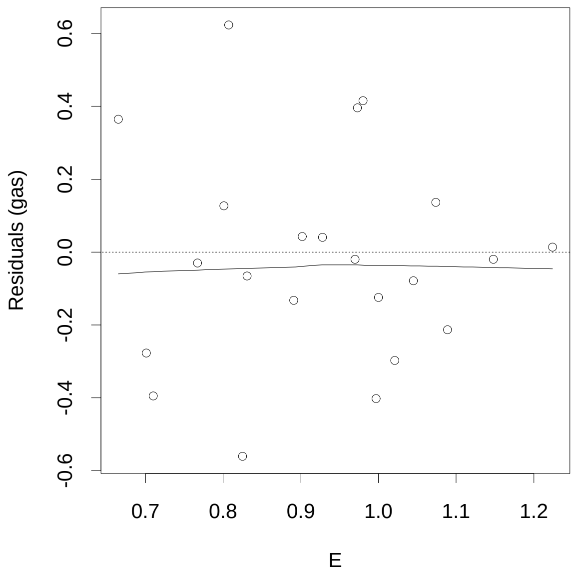

Figure 5: Residuals against E with a scatterplot smoothing—first fit to the gas data.

of fit: The result is shown in Figure 5. No effect appears to be present in the diagnostic plot, so $\alpha = 2/3$ appears to have introduced no lack of fit. But is there surplus of fit, that is, can we get away with a larger α? To check this, we fit a new loess model with $\alpha = 1$:

```
struct  loess_struct    gas_null;

loess_setup(E, NOx, n, p, &gas_null);
gas_null.model.span = 1;
loess(&gas_null);
loess_summary(&gas_null);
```

Number of Observations: 22
Equivalent Number of Parameters: 3.5
Residual Standard Error: 0.5197

The residual plot is shown in Figure 6. There is a strong signal in the residuals—a dependence of the level of the $\hat{\varepsilon}_i$ on E, so $\alpha = 1$ is too large, which suggests that $\alpha = 2/3$ is about as large as we can get away with. Thus, we have verified our specification of the form of $g(x)$ since there appears to be no surplus or lack of fit.

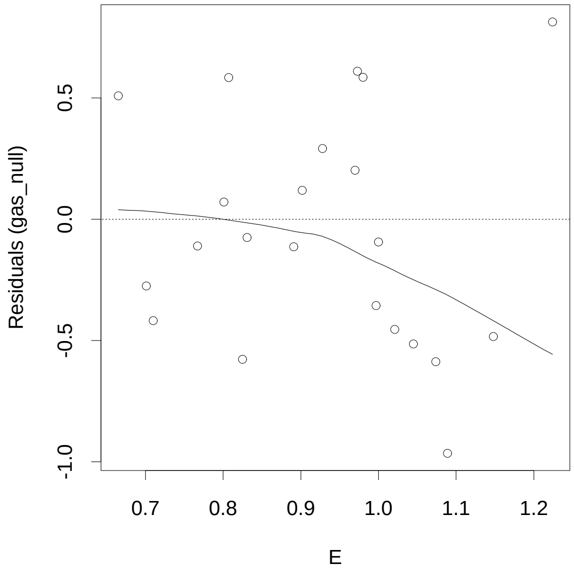

Figure 6: Residuals against E with a scatterplot smoothing—second fit to the gas data.

Next, we check the distributional specifications for the error terms. To see if the scale of the residuals depends on the level of the surface, we plot $\sqrt{|\hat{\varepsilon}_i|}$ against the fitted values, $\hat{y}_i$. Taking the square root tends to symmetrize the distribution of the absolute residuals. For our current example, with its small sample size of 22, we would not expect this method to reliably detect anything but a radical change in scale, but for illustrative purposes we show the plot in Figure 7: The graph does not show any convincing dependence. To check for dependence of the scale on E, a similar graph was made—but against E instead of the fitted values—and, again, no convincing dependence was found. To check the assumption of a Gaussian distribution of the errors, we will make a Gaussian probability plot of the residuals. In order to judge the straightness of the points on such plots, we will draw a line through the lower and upper quartiles. The result, shown in Figure 8, suggests that the Gaussian specification is justified.

Inference

`gas` has passed the diagnostic tests, which allows us to carry out statistical inferences with an assurance of validity. First, we compute 99% pointwise confidence intervals

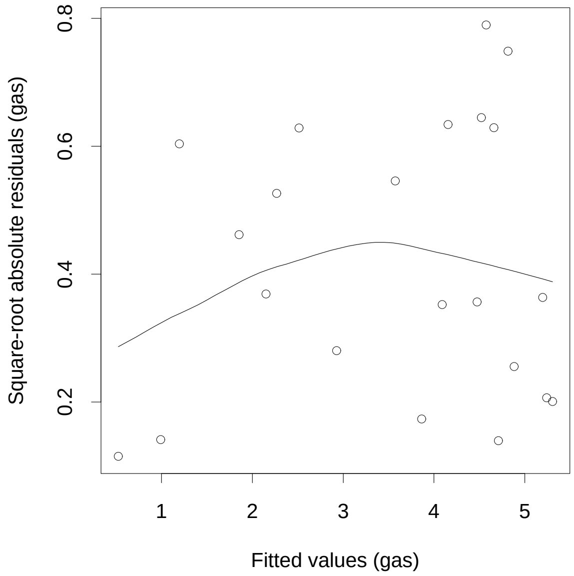

Figure 7: Square-root absolute residuals against fitted values with a scatterplot smoothing.

for $g(x)$ at seven values of E ranging from minimum value of E to the maximum in equal steps: 0.665, 0.758167, 0.851333, 0.9445, 1.03767, 1.13083, and 1.22400.

```
struct  ci_struct       gas_ci;
double  newdata[] = {0.665, 0.758167, 0.851333, 0.9445,
                     1.03767, 1.13083, 1.224},
        coverage = .99;
long    m = 7, se_fit = TRUE;
int     i;

predict(newdata, m, &gas, &gas_pred, se_fit);
pointwise(&gas_pred, m, coverage, &gas_ci);
for(i = 0; i < m; i++)
        printf("%g ", gas_ci.upper[i]);
printf("\n");
for(i = 0; i < m; i++)
        printf("%g ", gas_ci.fit[i]);
printf("\n");
```

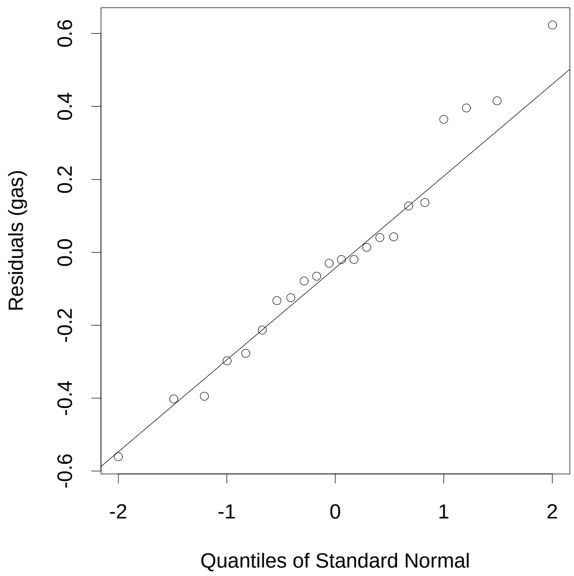

Figure 8: Gaussian quantile plot of residuals with line passing through lower and upper quartiles.

```
for(i = 0; i < m; i++)
        printf("%g ", gas_ci.lower[i]);
printf("\n");
```

1.98562 4.10981 5.48023 5.56651 3.52761 1.71062 1.47205
1.19641 3.67950 5.05571 5.13526 3.14366 1.19693 0.523682
0.407208 3.24919 4.63119 4.70401 2.75970 0.683247 -0.424684

These confidence intervals are added to the graph of the curve in Figure 9:

We know from the diagnostic checking that the second local regression model fitted to the gas data does not fit the pattern of the data. The increase in α for the second fit results in a drop in the equivalent number of parameters, but s, the estimate of σ, increases by a factor of about 1.5. This is to be expected in view of the lack of fit. We will carry out a statistical comparison of the first fit, `gas`, and the second, `gas_null`, by an analysis of variance:

```
struct  anova_struct   gas_anova;

anova(&gas_null, &gas, &gas_anova);
```

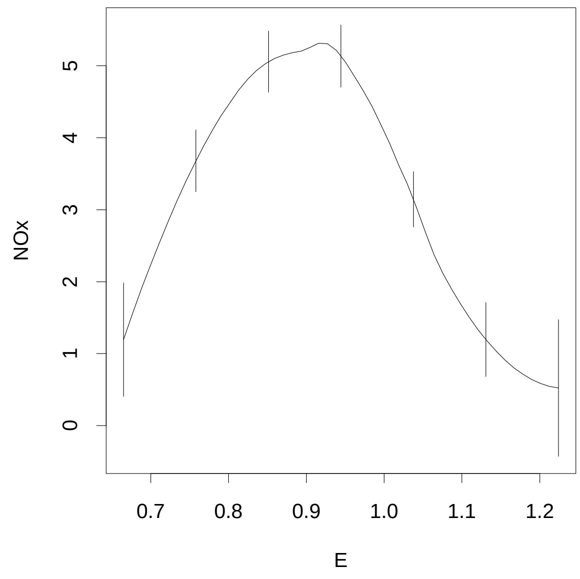

Figure 9: Gas data—local regression fit with 99% pointwise confidence intervals.

```
printf("%g %g %g %g\n", gas_anova.dfn, gas_anova.dfd,
       gas_anova.F_value, gas_anova.Pr_F);
```

2.5531 15.663 10.1397 0.000860102

The result, as expected, is highly significant.

2.2 Ethanol Data

The experiment that produced the gas data that we just analyzed was also run with gasoline replaced by ethanol. There were 88 runs and two factors: E, as before, and C, the compression ratio of the engine.

Exploratory Data Display

An exploratory plot useful for starting an analysis with two or more factors is the scatterplot matrix, shown in Figure 10. We will refer to panels in this and other multipanel displays by column and row, numbering as we would on a graph; thus, the lower left panel is (1,1) and the one to the right of it is (2,1). The (3,3) panel of

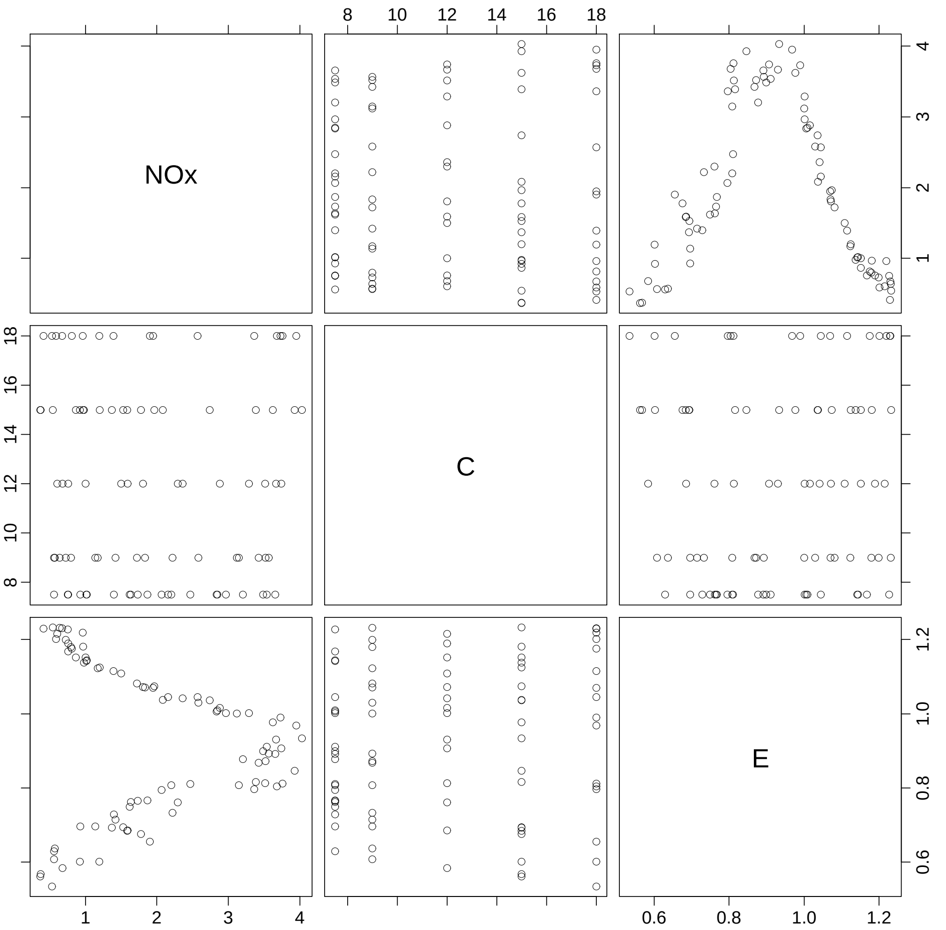

Figure 10: Ethanol data—scatterplot matrix of $\mathrm{NO_x}$, C, and E.

the matrix, a scatterplot of NO_x against E, shows a strong nonlinear dependence with a peak between 0.8 and 1.0. This makes it immediately clear that we need to use locally quadratic fitting. The (2,3) panel of the scatterplot matrix shows no apparent dependence of NO_x on C; however, we should not at this point draw any firm conclusion since it is possible that a dependence is being masked by the strong effect of E. The (1,2) panel, which graphs the configuration of points in the space of the factors, shows that the values of the two variables are nearly uncorrelated and that C takes on one of five values.

Coplots are an essential tool in fitting local regression models. Figure 11 is a coplot of the ethanol data. The dependence panels are the 3×3 array, and the given panel is at the top. On each dependence panel, NO_x is graphed against C for those observations whose values of E lie in an interval; on the panel, we are seeing how NO_x depends on C for E held fixed to the interval. The intervals are shown on the given panel; as we move from left to right through these intervals, we move from left to right and then bottom to top through the dependence panels. The intervals have two properties: approximately the same number of observations lie in each interval and approximately the same number of observations lie in two successive intervals. The data analyst specifies the number of intervals, 9 in Figure 11, and the target fraction of points shared by successive intervals, 1/2 in Figure 11.

Figure 12 is a coplot of NO_x against E given C. Since C takes on five values, we have simply conditioned on each of these five values.

What do we learn from these coplots? First, NO_x does in fact depend on C; for low values of E, NO_x increases with C, and for medium and high values of E, NO_x is constant as a function of C. Thus, there is an interaction between C and E. Second, over the range of values of E and C in the dataset, NO_x undergoes more rapid change as a function of E for C held fixed than as a function of C for E held fixed. Finally, the plots show that the amount of noise—that is, the variance, σ^2, of the ε_i—is small compared with the effect due to E, and is moderate compared with the effect due to C.

Modeling the Ethanol Data

It is quite clear from the exploratory plots that we must specify a locally-quadratic surface—that is, take λ to be 2—because of the substantial curvature as a function of E. Also, we will specify $\alpha = 0.5$ for the first fit:

```
struct  loess_struct    ethanol;
long    n = 88, p = 2;
double  NOx[] = {3.741, 2.295, 1.498, ...};
double  C_E[] = {12, 12, 12, ...};

loess_setup(C_E, NOx, n, p, &ethanol);
ethanol.model.span = 0.5;
loess(&ethanol);
loess_summary(&ethanol);
```

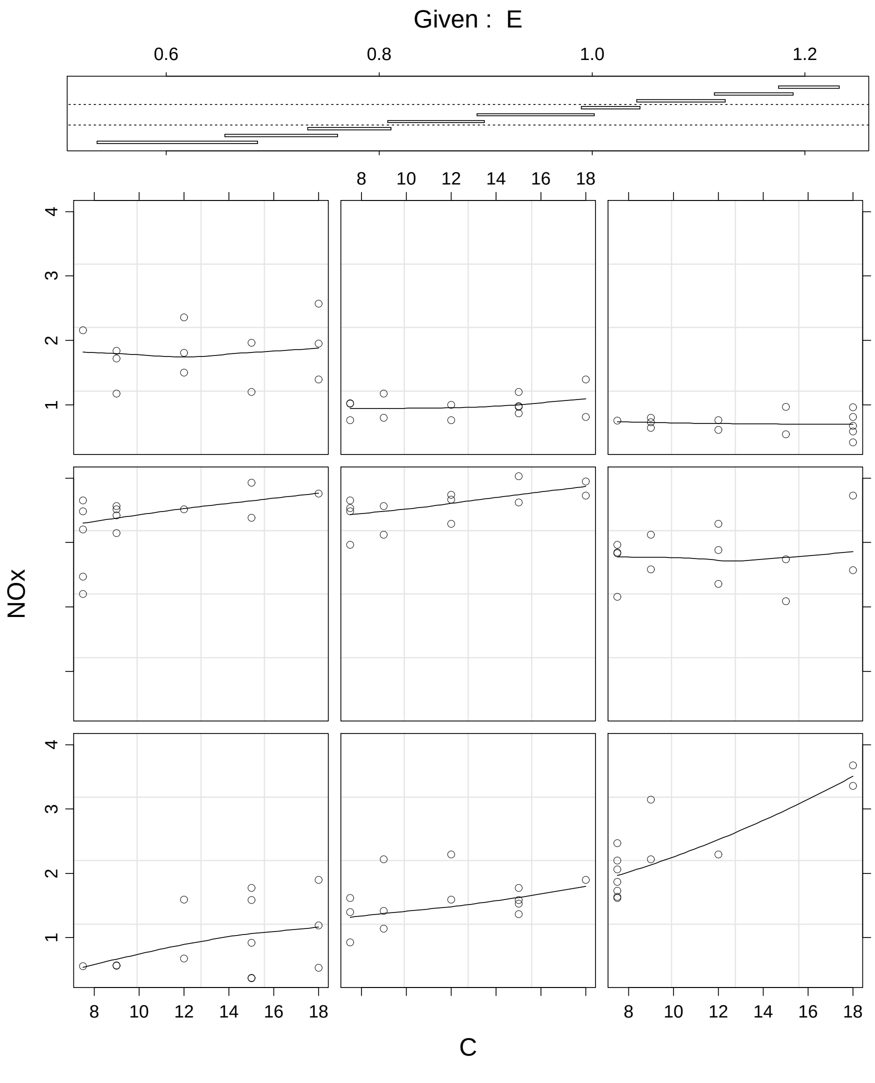

Figure 11: Ethanol data—coplot of NO_x against C given E with scatterplot smoothings.

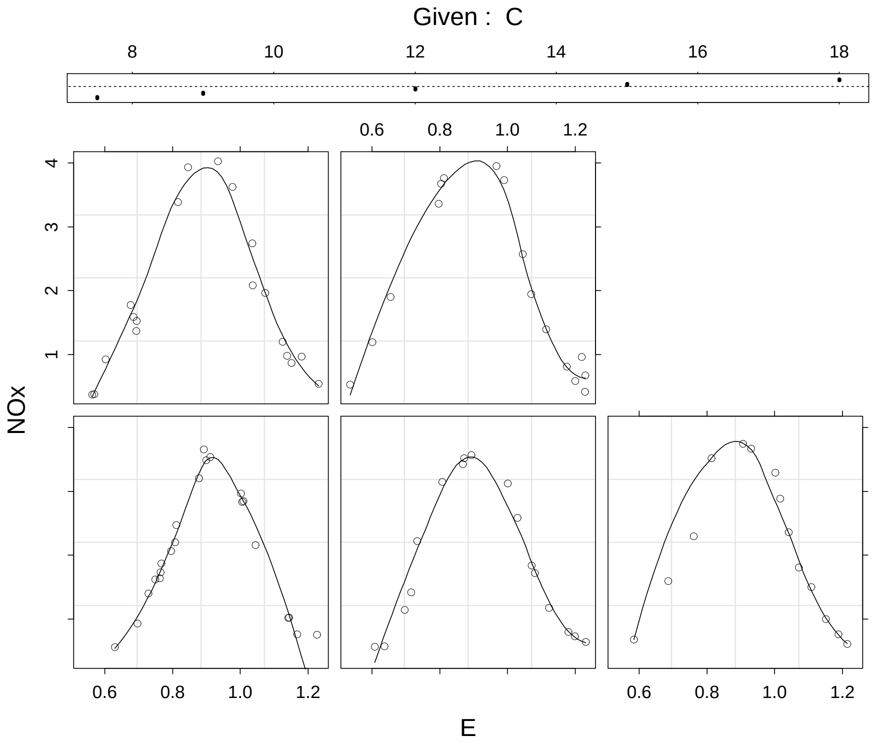

Figure 12: Ethanol data—coplot of NO_x against E given C with scatterplot smoothings.

Number of Observations: 88
Equivalent Number of Parameters: 13.0
Residual Standard Error: 0.2599

We begin a search for lack of fit by plotting the residuals against each of the factors in the top two panels of Figure 13. Clearly there is lack of fit in the right panel. Thus, we drop `span` to 1/4:

```
ethanol.model.span = 0.25;
loess(&ethanol);
loess_summary(&ethanol);
```

Number of Observations: 88
Equivalent Number of Parameters: 21.6
Residual Standard Error: 0.1761

But we must check further; these marginal residual plots can, of course, hide local lack of fit in the (C, E) plane. We check this by the coplots in Figures 14 and 15. There is some suspicious behavior on the (1,2) and (2,2) dependence panels of Figure 14; almost all of the residuals are positive. The detected effect is, however, quite minor, so we will ignore it.

We can check the specifications of the error distribution by the same diagnostic methods used for the gas data—graph $\sqrt{|\hat{\varepsilon}_i|}$ against $\hat{y}_i$, graph $\sqrt{|\hat{\varepsilon}_i|}$ against C and E, and make a Gaussian probability plot of $\hat{\varepsilon}_i$. This was done, and the new fit passed the tests.

Plotting the Surface

For loess fitting with two factors, the fitted surface can also be displayed by coplots. This is done in Figures 16 and 17. Let $\hat{g}(C, E)$ be the fitted surface. Consider a single panel of Figure 16. E has been set to a specific conditioning value, $E = E^*$; then $\hat{g}(C, E^*)$ has been evaluated for 50 equally spaced values of C ranging from the minimum value of C in the data to the maximum, and the surface values have been graphed on the panel against the equally spaced values of C. Also, 99% confidence intervals are drawn at seven equally spaced points from the minimum value of C in the data to the maximum. There are 16 equally spaced conditioning values of E ranging from the minimum value of E in the data to the maximum; the given panel in Figure 16 shows the 16 values. Similarly, Figure 17 shows the dependence of the fitted surface on E for 16 conditioning values of C.

Dropping Squares and Conditionally Parametric Surfaces

The coplot in Figure 16 show that the ethanol fit has an undesirable property: the surface as a function of C for fixed E has unconvincing undulations, especially in the (1,1) dependence panel. Our skepticism comes from two sources. First, in

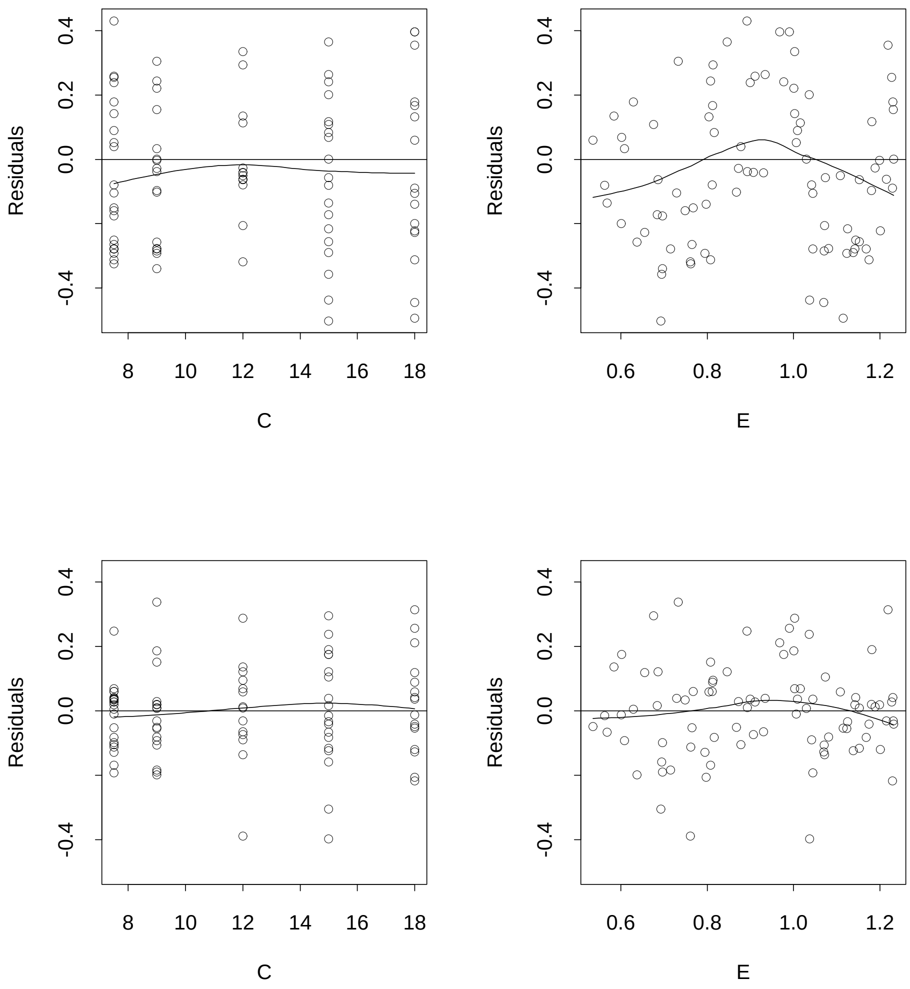

Figure 13: Residuals against E and C with scatterplot smoothings. The top row of plots corresponds to the first fit, the bottom row to the second fit.

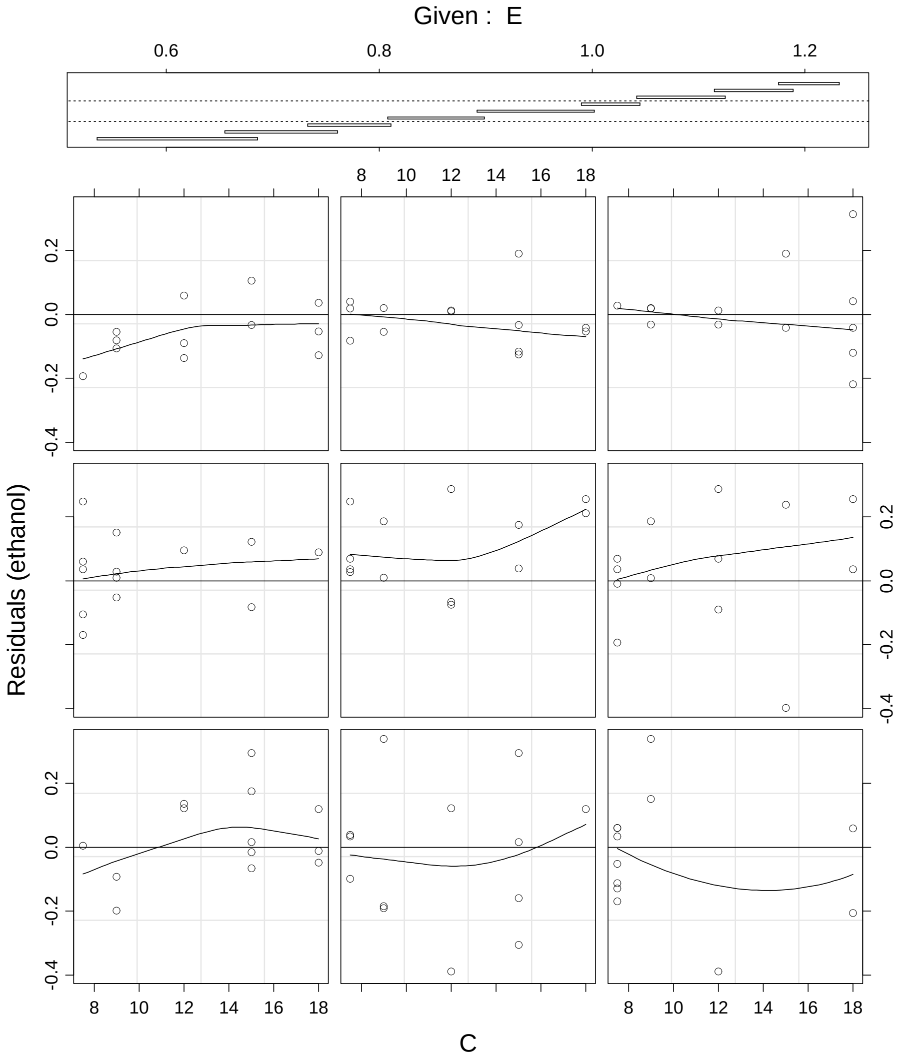

Figure 14: Coplot of residuals against C given E with scatterplot smoothings.

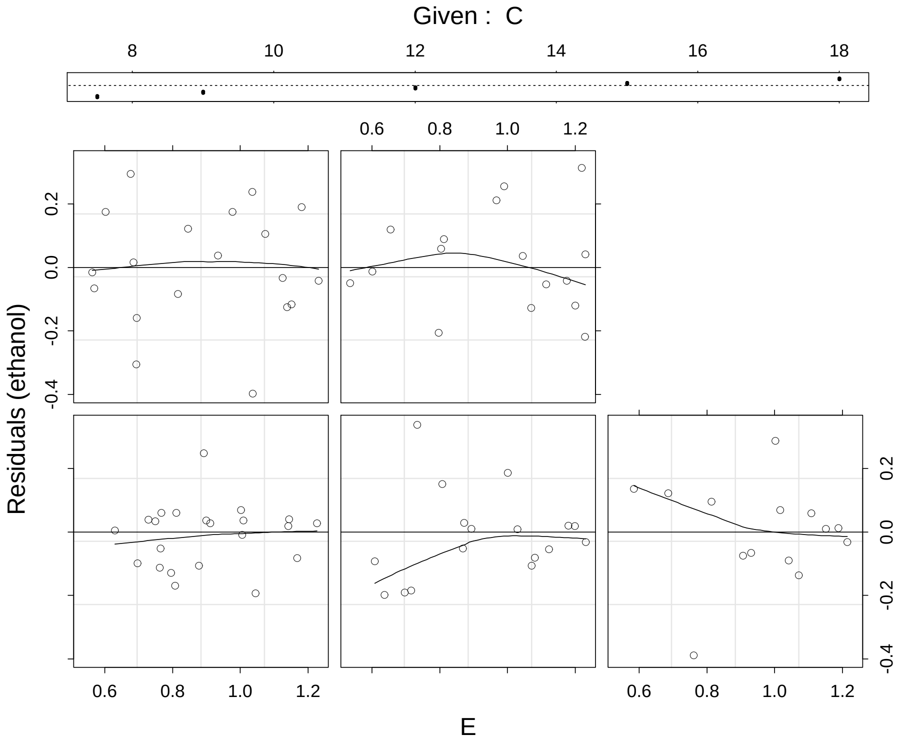

Figure 15: Coplot of residuals against E given C with scatterplot smoothings.

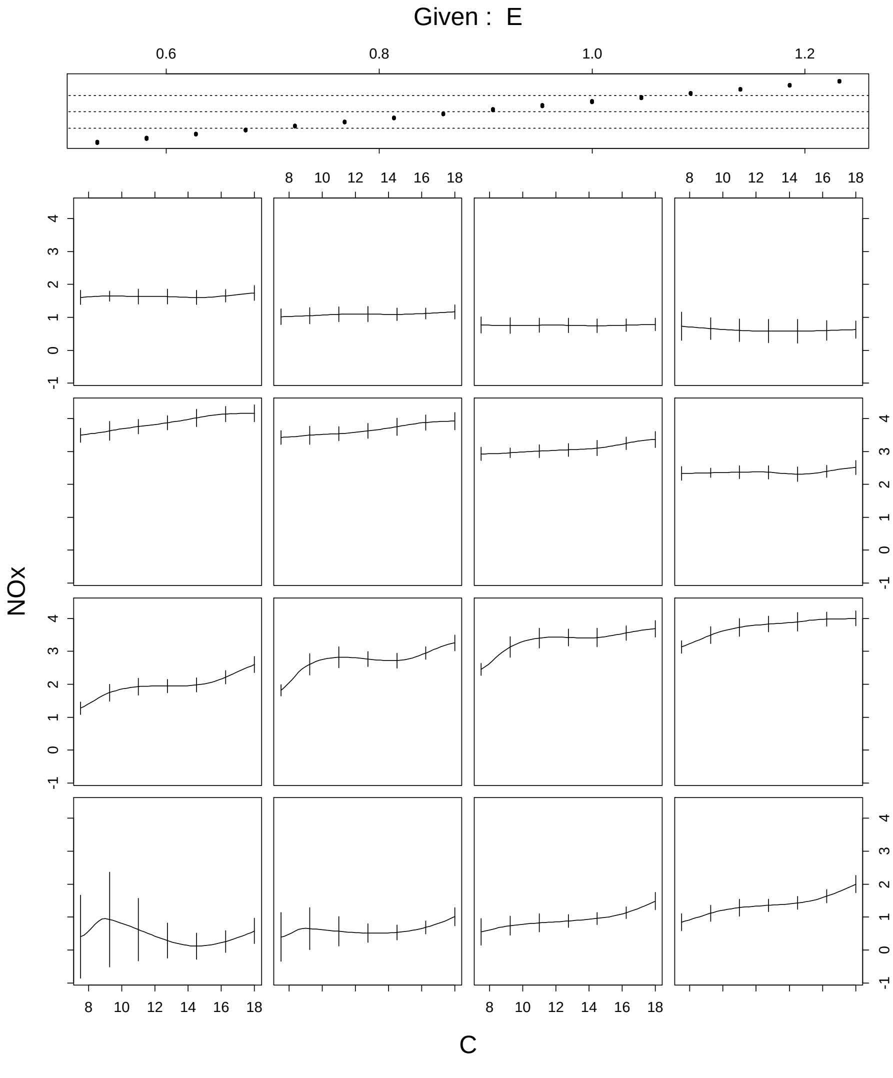

Figure 16: Ethanol data—coplot of the local regression fit with pointwise 99% confidence intervals.

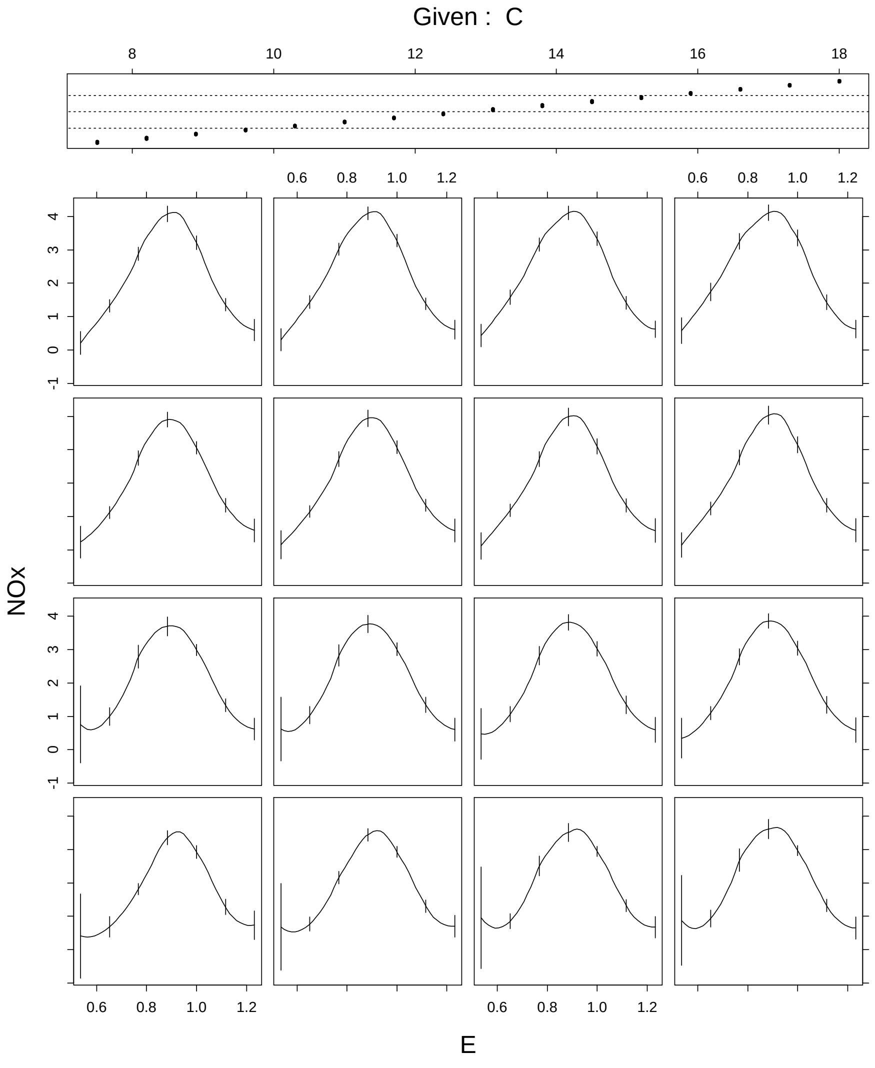

Figure 17: Ethanol data—coplot of the local regression fit with pointwise 99% confidence intervals.

the coplot of the data in Figure 11, NO_x appears to be a very smooth function of C; in fact, the coplot suggests that given E, the dependence is actually linear in C. Second, the undulations in Figure 16 are small compared with the sizes of the confidence intervals.

As we saw from the diagnostic checking, if we increase α and thereby get more smoothness as a function of C, we introduce lack of fit. Instead, we will cut back on the variation of the fit as a function of C by dropping C^2 from the fitting variables; this leaves us with a constant, E, C, EC, and E^2. In addition, we will specify the surface to be conditionally parametric in C; this will result in a fit that is linear in C given E:

```
struct  loess_struct    ethanol_cp;

loess_setup(C_E, NOx, n, p, &ethanol_cp);
ethanol_cp.model.span = 0.25;
ethanol_cp.model.parametric[0] = TRUE;
ethanol_cp.model.drop_square[0] = TRUE;
loess(&ethanol_cp);
```

Let's compare the old fit and the new:

```
loess_summary(&ethanol);
```

Number of Observations: 88
Equivalent Number of Parameters: 21.6
Residual Standard Error: 0.1761

```
loess_summary(&ethanol_cp)
```

Number of Observations: 88
Equivalent Number of Parameters: 18.2
Residual Standard Error: 0.1808

The equivalent number of parameters has dropped by about 15%, the residual standard error has increased insignificantly, and diagnostic plots, not shown here, indicated no lack of fit. But the big gain is that we can now increase `span` to 1/2 without introducing lack of fit:

```
ethanol_cp.model.span = 0.5;
loess(&ethanol_cp);
loess_summary(&ethanol_cp);
```

Number of Observations: 88
Equivalent Number of Parameters: 9.2
Residual Standard Error: 0.1842

In so doing we have driven the equivalent number of parameters to less than half of what it was originally and kept the residual standard error about the same. The coplots in Figures 18 and 19 show the resulting fitted surface.

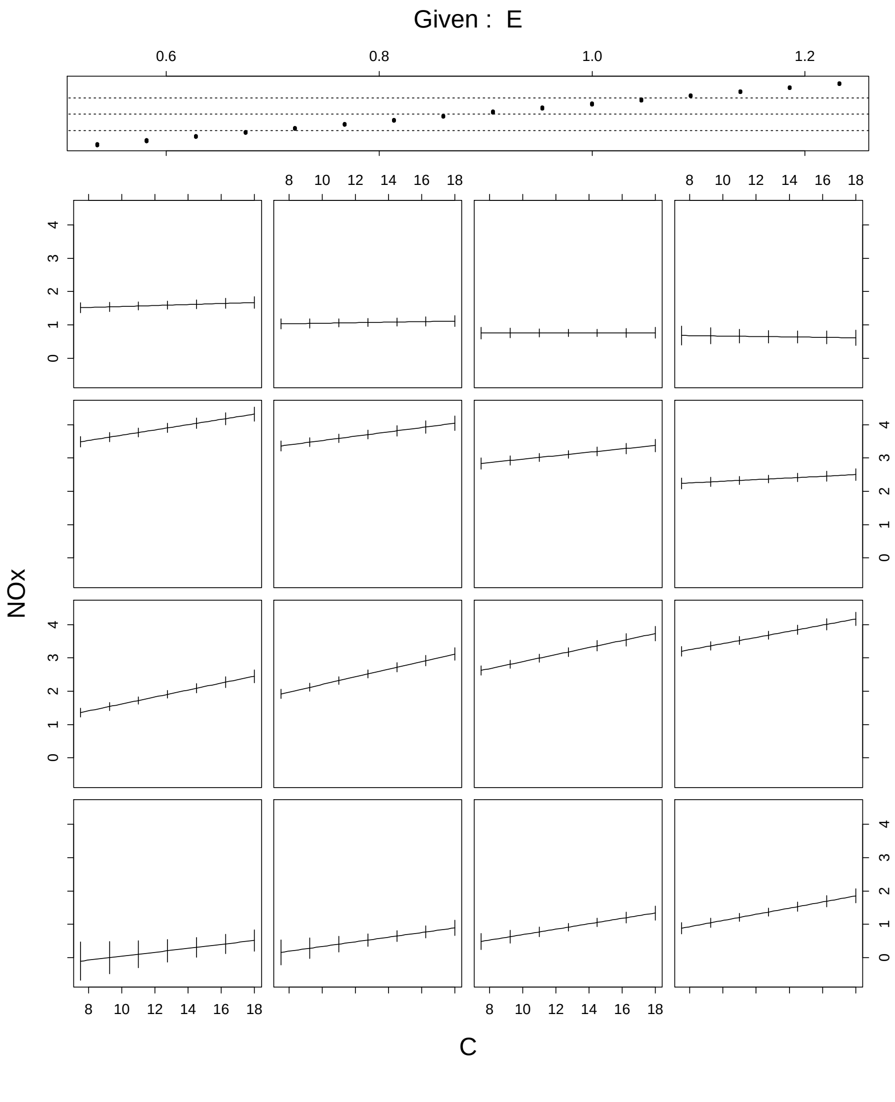

Figure 18: Ethanol data—coplot of conditionally parametric local-regression fit with pointwise 99% confidence intervals.

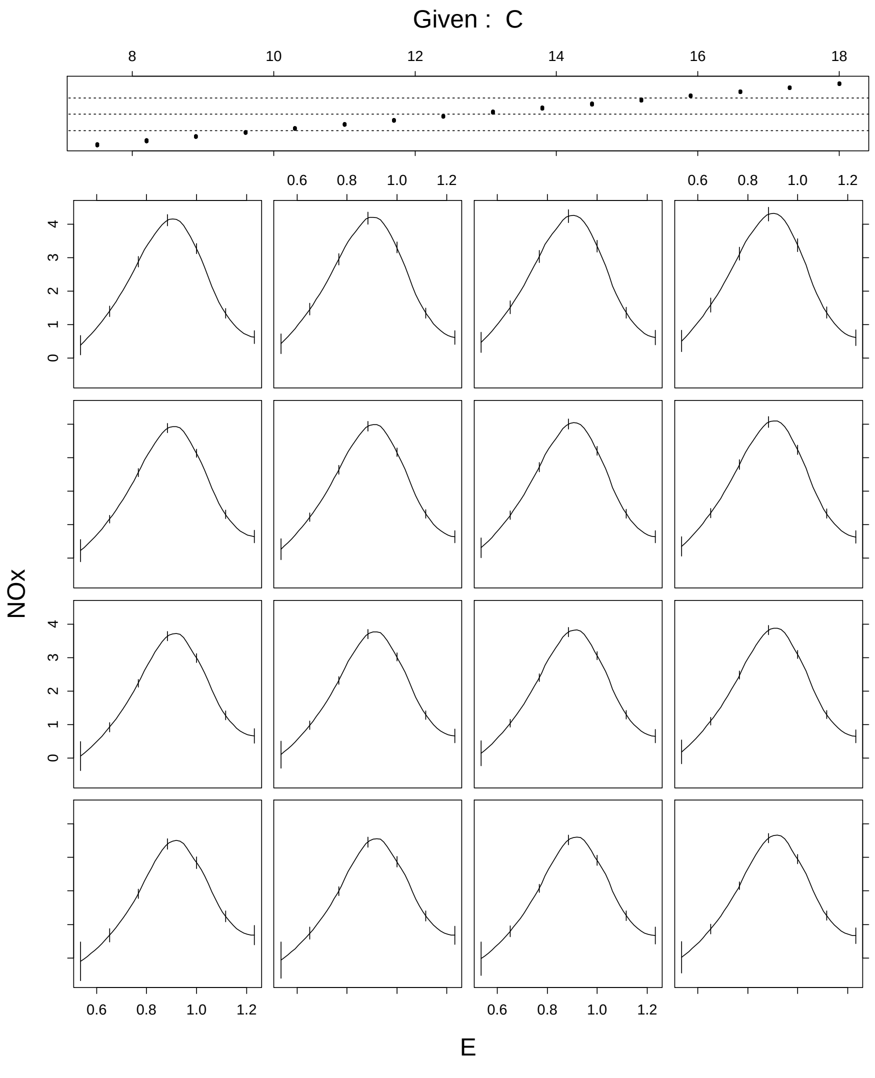

Figure 19: Ethanol data—coplot of conditionally parametric local-regression fit with pointwise 99% confidence intervals.

Computing the Fitted Surface and Confidence Intervals

We turn now to a further discussion of how `predict()` is used to evaluate a fitted surface and to compute information for confidence intervals. Let us evaluate our final fit to the ethanol data at the following values of the two factors: (7.5, 0.6), (9.0, 0.8), (12.0, 1.0), (15.0, 0.8), and (18.0, 0.6).

```
struct  pred_struct     ethanol_pred;
double  newdata[] = {7.5, 9.0, 12.0, 15.0, 18.0, 0.6,
                     0.8, 1.0, 0.8, 0.6};
long    m = 5, se_fit = FALSE;
int     i;

predict(newdata, m, &ethanol_cp, &ethanol_pred, se_fit);
for(i = 0; i < m; i++)
        printf("%g ", ethanol_pred.fit[i]);
printf("\n");
```

0.281582 2.59714 3.06672 3.25558 1.06378

As with one factor, confidence-interval information can be computed at each point of `newdata` by setting `se=1` in the loess output structure, but again we point out that this increases the computational intensity substantially. To get the intervals shown in Figure 19, we define a 7 x 16 evaluation grid that spans `C` and `E`:

```
struct  pred_struct     ethanol_grid;
struct  ci_struct       ethanol_ci;
double  Cmin = 7.5, Cmax = 18.0, Emin = 0.535, Emax = 1.232;
double  Cm[7], Em[16], grid[224];
double  tmp, coverage = .99;
int     i, j, k;

m = 112;
se_fit = TRUE;

tmp = (Cmax - Cmin) / 6;
for(i = 0; i < 7; i++)
        Cm[i] = Cmin + tmp * i;
tmp = (Emax - Emin) / 15;
for(i = 0; i < 16; i++)
        Em[i] = Emin + tmp * i;
for(i = 0; i < 16; i++) {
        k = i * 7;
        for(j = 0; j < 7; j++) {
                grid[k + j] = Cm[j];
                grid[m + k + j] = Em[i];
        }
}
```

```
predict(grid, m, &ethanol_cp, &ethanol_grid, se_fit);
pointwise(&ethanol_pred, m, coverage, &ethanol_ci);
```

2.3 Air Data

We turn now to an application with three factors. The data are from an environmental study that analyzed how the air pollutant ozone depends on three meteorological variables: radiation, wind speed, and temperature (Bruntz et al., 1974). The data are daily measurements of the four variables for 111 days.

For three or more factors, carrying out fitting and inference for local regression models is no more complicated than for two. What gets harder, of course, is graphing the data to explore and diagnose. The coplot idea can be used for three factors since by plotting against one factor, conditioning on two others. Thus, for three factors, we can make three coplots, graphing against each factor conditional on the other two. Figure 20 shows one of the three coplots for the `air` data. We have conditioned on wind and temperature. The dependence panels are the 4×4 matrix of panels. The given panels, one for each conditioning factor, are to the right and top. As we move up a column of dependence panels, the intervals of wind speed increase, and as we move from left to right across a row of dependence panels, the intervals of temperature increase. For example, the points on the (2,3) panel of the coplot are observations for which the temperature measurements are in the second interval and the wind speed measurements are in the third interval. We omit the two remaining coplots, but in the analysis of these data they were carefully studied.

Let's fit a local regression model to the data.

```
struct  loess_struct    air;
double  ozone[] = {3.448217, 3.30193, 2.28943, ...};
double  rad_temp_wind[] = {190, 118, 149, ...};
long    n = 111, p = 3;

loess_setup(rad_temp_wind, ozone, n, p, &air);
air.model.span = 0.8;
loess(&air);
loess_summary(&air);
```

Number of Observations: 111
Equivalent Number of Parameters: 15.6
Residual Standard Error: 0.4329

Diagnostic plots revealed that the specifications of the fit are reasonable assumptions. The fitted surface is displayed by a coplot in in Figure 21.

2.4 Galaxy Velocities

NGC7531 is a spiral galaxy in the Southern Hemisphere with a very bright inner ring. When looked at from the earth, the galaxy takes up a small area on the celestial

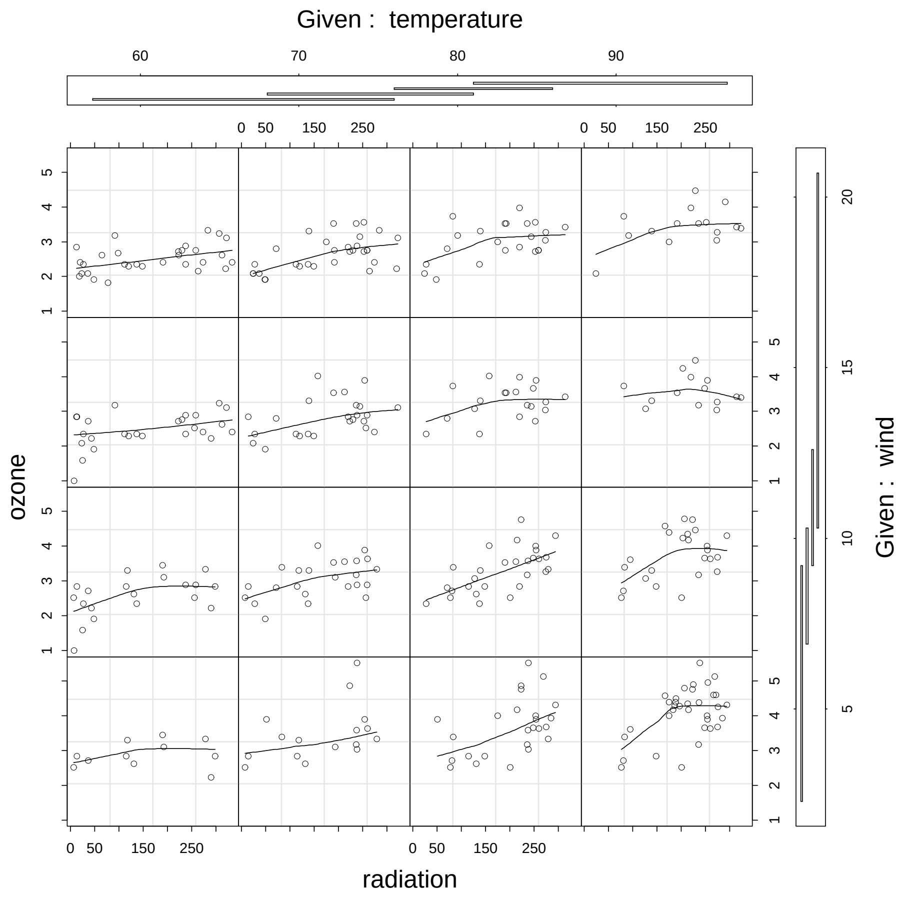

Figure 20: Air data—coplot of ozone against solar radiation given wind speed and temperature with scatterplot smoothings.

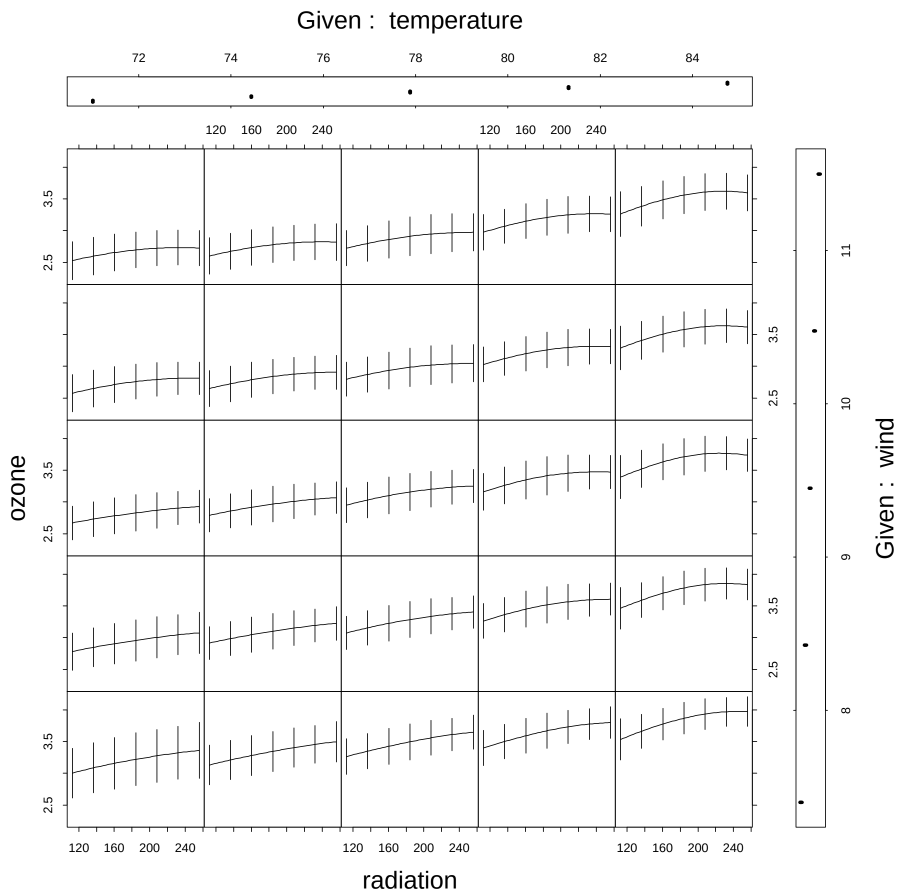

Figure 21: Air data—local regression fit with pointwise 99% confidence intervals.

sphere. Figure 22 shows measurements of the radial velocity of the galaxy at 323 locations in this area (Buta, 1987). The positions have been jittered slightly to reduce overplotting. The horizontal scale of the graph is the east-west coordinate

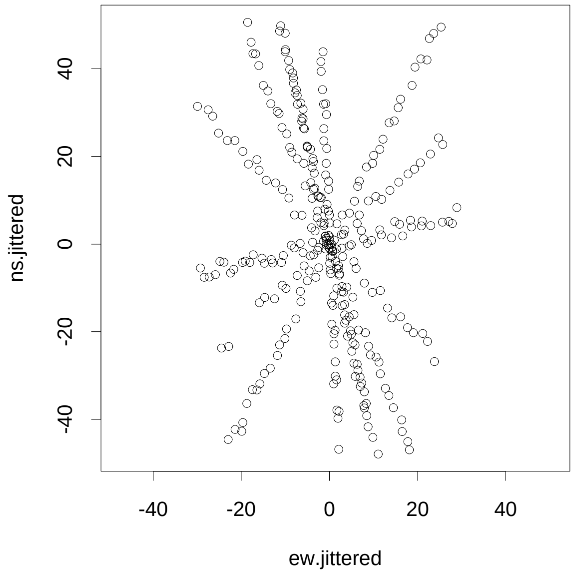

Figure 22: Galaxy data—locations of velocity measurements.

and the vertical scale is the north-south coordinate. Note that north is up and east is to the left because we are looking at the celestial sphere from the inside. Each measurement lies along one of seven slits that nearly intersect at a single point near the origin, (0,0). Suppose the *radial positions* are the signed distances from the origin to the measurement locations; a distance is multiplied by -1 if the east-west coordinate is negative and by 1 if it is positive: Figure 23 graphs against radial position for each slit. The figure shows that it is sensible to approach modeling velocity dependence by a smooth function of the east-west and north-south coordinates with random variation superimposed.

Modeling

The goal in the analysis of these data is to understand how galaxy velocity varies over the measurement region. Thus, velocity is a response and there are two factors: east-west position and south-north position. In Figure 23 the curvature of the

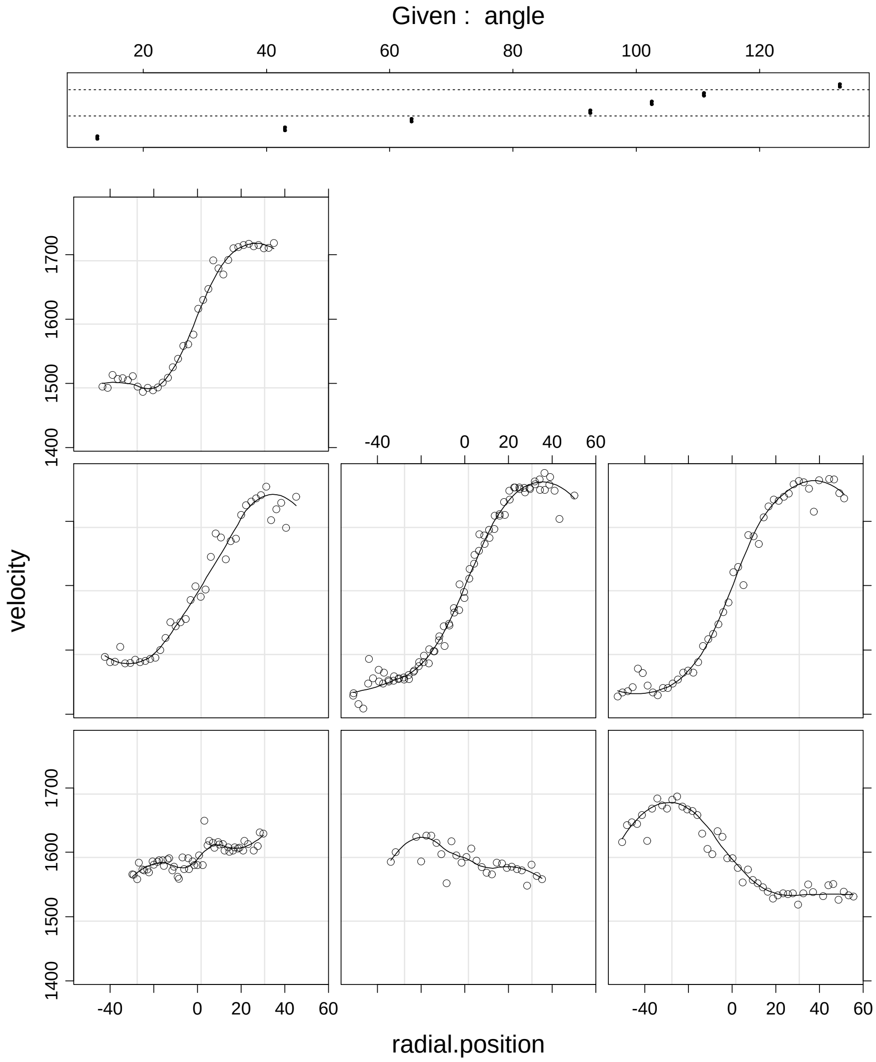

Figure 23: Galaxy data—coplot of velocity against radial position given slit angle.

underlying pattern is substantial; thus we will specify a locally-quadratic surface. Since many points appear to deviate substantially from the overall pattern compared to the deviations of the majority of points, it seems prudent to specify symmetric errors. Finally, it makes sense to preserve the spatial metric of the factors and not normalize the variation in their measurements:

```
struct  loess_struct    galaxy;
double  velocity[] = {1769, 1749, 1749, ...};
double  direction[] = {8.46279, 7.96498, 7.46717, ...};
long    n = 323, p = 2;

loess_setup(direction, velocity, n, p, &galaxy);
galaxy.model.span = 0.35;
galaxy.model.normalize = FALSE;
galaxy.model.family = "symmetric";
loess(&galaxy);
loess_summary(&galaxy);
```

Number of Observations: 323
Equivalent Number of Parameters: 19.6
Residual Scale Estimate: 12.1139

Let's evaluate the surface on a grid and then make a contour plot:

```
struct  pred_struct     galaxy_contour;
double  ew[59], ns[99], grid[11682];
double  tmp;
long    m = 5841, se_fit = FALSE;
int     i, j, k;

tmp = -29.0;
for(i = 0; i < 59; i++)
        ew[i] = tmp++;
tmp = -49.0;
for(i = 0; i < 99; i++)
        ns[i] = tmp++;
for(i = 0; i < 99; i++) {
        k = i * 59;
        for(j = 0; j < 59; j++) {
                grid[k + j] = ew[j];
                grid[m + k + j] = ns[i];
        }
}
predict(grid, m, &galaxy, &galaxy_contour, se_fit);
```

The result is shown in Figure 24. Recall that we studied the fits to `ethanol` and `air` by coplots, but in this application it makes sense to use a contour plot since we want to see the surface as a whole entity—finding peaks, troughs, ridges, steep terrain, and so forth—and are not interested in conditional dependence.

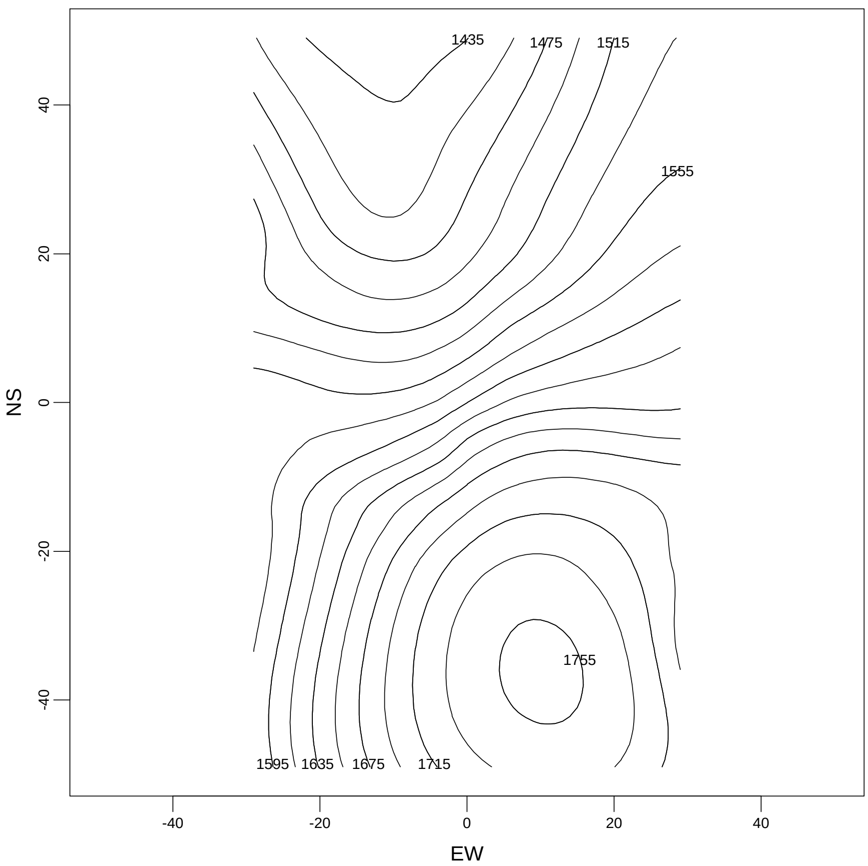

Figure 24: Galaxy data—contour plot of local regression fit.

Diagnostic Checking

Of course, we must carry out diagnostic checking to make sure we have not plotted nonsense in Figure 24. First, in Figure 25, we make a coplot of the residuals, displaying them as we did the original data. The (1,2) dependence panel shows some clear lack of fit. At the left extreme, the distortion is as large as 40 km/sec, which is more than we would like. But since the fraction of observations that are affected is small we push on, but noting that our results are somewhat tainted. Figure 26 is a normal probability plot of the residuals. The distribution of the residuals is symmetric and strikingly leptokurtic. The robust estimation is clearly justified, and we should feel quite smug at having guessed correctly from the exploratory coplot.

Confidence Intervals

Figure 24 shows that the velocity surface has a backbone of sorts. Consider the line in the plane of the factors that goes through the origin and through the position, (10, -37), where the maximum of the surface occurs. The surface is roughly symmetric in directions perpendicular to the line. Also, the line passes close to the minimum of the surface. Let's evaluate the surface at 100 equally-spaced points along this line and compute confidence intervals at 15 selected positions:

```
struct  pred_struct     spine_fit, spine_se;
struct  ci_struct       spine_ci;
double  fit_eval[200], ci_eval[30];
double  range = 98, coverage = .99;

m = 100;
tmp = range / 99;
for(i = 0; i < 100; i++) {
        fit_eval[i + 100] = -49 + tmp * i;
        fit_eval[i] = fit_eval[i + 100] / (-3.7);
}
predict(fit_eval, m, &galaxy, &spine_fit, se_fit);

m = 15;
se_fit = TRUE;
tmp = range / 14;
for(i = 0; i < m; i++) {
        ci_eval[i + m] = -49 + tmp * i;
        ci_eval[i] = fit_eval[i + 100] / (-3.7);
}
predict(ci_eval, m, &galaxy, &spine_se, se_fit);
pointwise(&spine_se, m, coverage, &spine_ci);
```

Figure 27 plots the fit against north-south position, and shows the 99% confidence intervals.

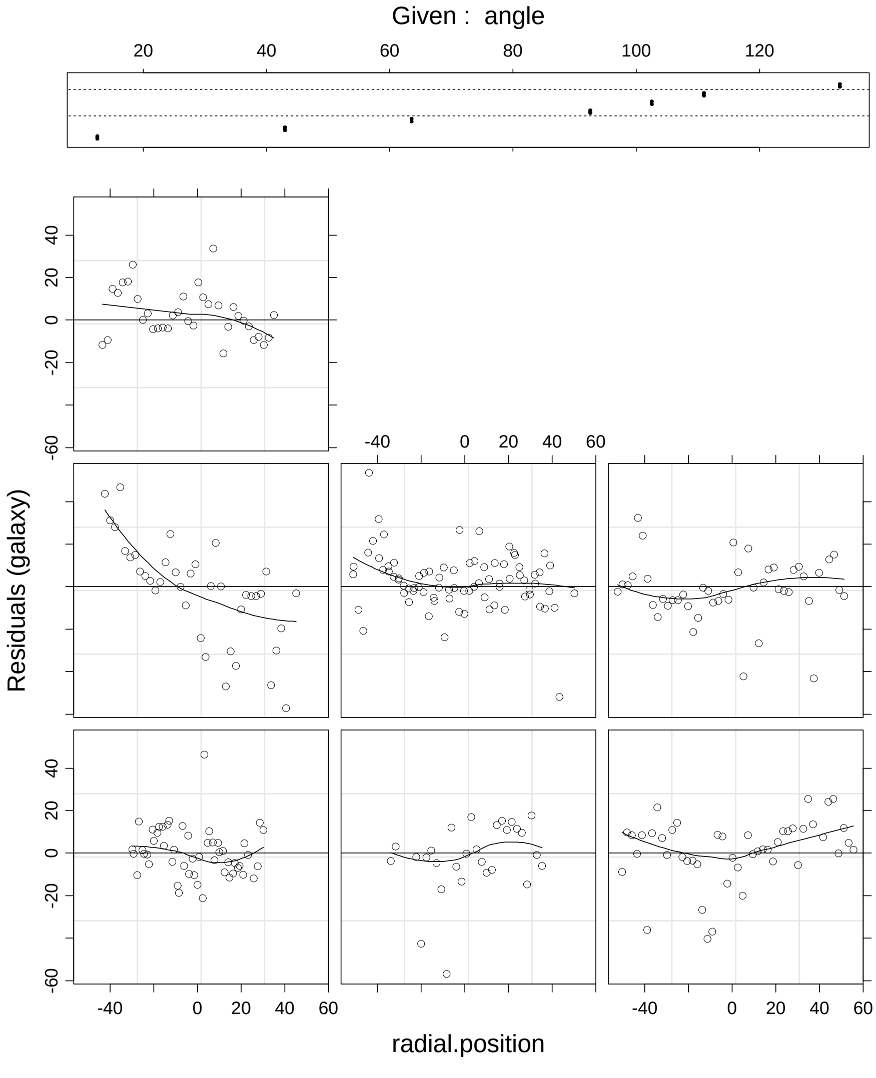

Figure 25: Galaxy data — Coplot of residuals with scatterplot smoothings.

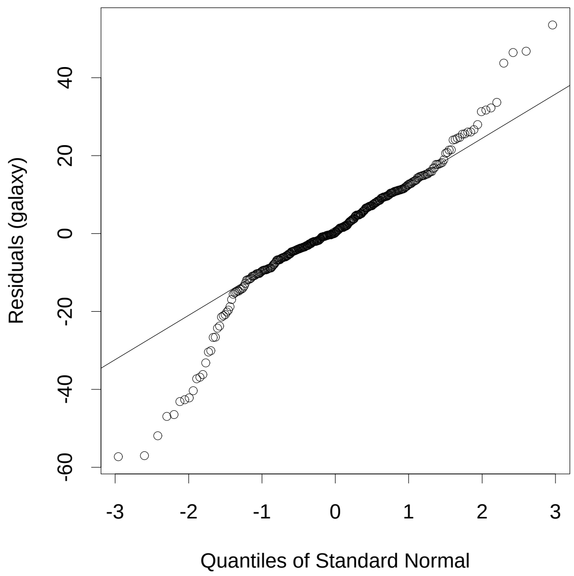

Figure 26: Galaxy data — Gaussian quantile plot of residuals.

3 Specializing and Extending the Computations

3.1 Computation

In the examples of Section 2, the function `predict()` did not use the loess fitting method to compute surfaces directly at every evaluation point. Rather, to get very fast computation, a default algorithm was used that employs interpolation. In this algorithm, a set of points, typically small in number, is selected for direct computation using the loess fitting method, and a surface is evaluated using an interpolation method that is based on *blending functions*. The space of the factors is divided into rectangular cells using an algorithm based on *k-d* trees. The loess fit is evaluated at the cell vertices, and then blending functions do the interpolation. The output data structure structure stores the *k-d* trees and the fits at the vertices. This information is used by `predict()` to carry out the interpolation. Of course, the resulting interpolated surface is not exactly the same as that of a surface computed directly, but the agreement is typically excellent. Even when it is not, the interpolation method is a perfectly logical smoothing method that has a number of desirable properties. This approach is what allows us, for example, to rapidly compute the surface of the galaxy data at a grid of 5841 values. Doing a direct loess evaluation at all of

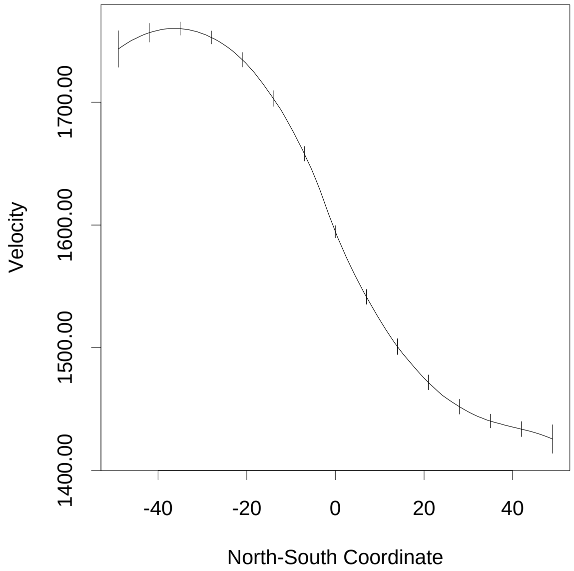

Figure 27: Galaxy data—local regression fit along the backbone with pointwise 99% confidence intervals.

these points would be expensive. The interpolation method, however, results in one restriction: the surface cannot be evaluated outside the range of the data; that is, the value of each numeric variable for an evaluation point must lie within the range of the observations of that variable in the data. This is not the case for the direct computation method, so evaluation can be done anywhere.

The local regression functions produce quantities that express in various ways information about degrees of freedom. `loess()` returns the equivalent number of parameters, `predict()` returns the degrees of freedom of t-intervals, and `anova()` returns the numerator and denominator degrees of freedom of an F-test. In the examples of Section 2, these quantities, which are defined in Section 4, are computed by an approximation method that is described in Section 4. A supercomputer environment (or a user with a great deal of patience) would be needed to routinely compute these statistical quantities exactly.

Most users will not want to use direct computation of surfaces or exact computation of the statistical quantities. However, those who want to explore the computational and statistical methods of loess fitting can change the computational methods by setting the desired values in the `loess_struct.control` structure. We can try this on the gas data:

```
struct  loess_struct    gas_slower;

loess_setup(E, NOx, n, p, &gas_slower);
gas_slower.control.surface = "direct";
gas_slower.control.statistics = "exact";
loess(&gas_slower);    /* runs slower than loess(&gas) */
```

In this example, we have switched the computation of the surface from `"interpolate"` to `"direct"`, and the computation of the statistical quantities from `"approximate"` to `"exact"`; recall that the default values are set by `loess_setup()`.

The structure `loess_struct.control` can also be used to control two other computational matters. When interpolation is used, `loess_struct.control.cell` controls the cell size of the k-d tree. The maximum fraction of points allowed inside a cell is the value of this parameter times the values of α; in the algorithm, a cell is divided if the maximum is exceeded. Also, `loess_struct.control.iterations` specifies the number of iterations of the loess robust estimate.

3.2 Inference

We stressed in Section 2 that it is critical to carry out diagnostic methods to study, among other things, surplus and lack of fit. In some applications, however, a clearly identifiable lack of fit might be acceptable if the identified magnitude of the distortion is judged to be small for the purpose to which the fit is put. For example, we might want a distorted surface if it made communication simpler and the distortion did not interfere with the judgment of salient features. But one problem is that an estimate, s, of σ based on a distorted fit would be biased, and thus a confidence interval based on this estimate would not have the stated coverage. There is a remedy. Suppose we have two loess fits, `fit_biased` and `fit_unbiased`, the first distorted and the second not. We can use the value of s from the undistorted fit to form confidence intervals for the distorted fit. We do this by changing `fit_biased`:

```
fit_biased.out.s = fit_unbiased.out.s;
```

Now giving `fit_biased` to `predict()` and `pointwise()` gives correct confidence intervals. It should be appreciated that the intervals are not for the true surface, but rather for the expected value of the distorted estimate.

4 Statistical and Computational Methods

In this section we discuss computational and statistical methods in the fitting of local regression models. In Section 4.1, we discuss the methods of inference that arise from the loess fitting method. In Section 4.2, we discuss computational methods that underlie loess fitting, and numerical problems that can arise.

4.1 Statistical Inference

Initially, we will suppose that the errors have been specified to be Gaussian and the variances have been specified to be constant.

One important property of a Gaussian-error loess estimate, $\hat{g}(x)$, is that it is linear in y_i—that is,

$$\hat{g}(x) = \sum_{i=1}^{n} l_i(x) y_i$$

where the $l_i(x)$ do not depend on the y_i. This linearity results in distribution properties of the estimate that are very similar to those for classical parametric fitting.

Suppose that the diagnostic methods have been applied and have revealed no lack of fit in $\hat{g}(x)$; we will take this to mean that $E\hat{g}(x) - g(x)$ is small. Suppose further that diagnostic checking has verified the specifications of the error terms in the model.

Estimation of σ

Since $\hat{g}(x)$ is linear in y_i, the fitted value at x_i can be written

$$\hat{y}_i = \sum_{j=1}^{n} l_j(x_i) y_j .$$

Let L be the matrix whose (i, j)th element is $l_j(x_i)$ and let

$$\bar{L} = I - L$$

where I is the $n \times n$ identity matrix. For $k = 1$ and 2, let

$$\delta_k = \mathrm{tr}(\bar{L}'\bar{L})^k .$$

We estimate σ by the scale estimate

$$s = \sqrt{\frac{\sum_{i=1}^{n} \hat{\varepsilon}_i^2}{\delta_1}} .$$

Confidence Intervals for $g(x)$

Since

$$\hat{g}(x) = \sum_{i=1}^{n} l_i(x) y_i ,$$

the standard deviation of $\hat{g}(x)$ is

$$\sigma(x) = \sigma \sqrt{\sum_{i=1}^{n} l_i^2(x)} .$$

We estimate $\sigma(x)$ by

$$s(x) = s\sqrt{\sum_{i=1}^{n} l_i^2(x)}.$$

Let

$$\rho = \frac{\delta_1^2}{\delta_2}.$$

The distribution of

$$\frac{\hat{g}(x) - g(x)}{s(x)}$$

is well approximated by a t distribution with ρ degrees of freedom; we can use this result to form confidence intervals for $g(x)$ based on $\hat{g}(x)$. Notice that the value δ_1 by which we divide the sum-of-squares of residuals is not the same as the value ρ used for the degrees of freedom of the t distribution. For classical parametric fitting, these two values are equal. For loess, they are typically close but not close enough to ignore the difference. We will refer to ρ as the *look-up degrees of freedom* since it is the degrees of freedom of the distribution that we look up to get the confidence interval.

Analysis of Variance for Nested Models

We can use the analysis of variance to test a null local regression model against an alternative one. Let the parameters of the null model be $\alpha^{(n)}$, $\lambda^{(n)}$, $\delta_1^{(n)}$, and $\delta_2^{(n)}$. Let the parameters of the alternative model be α, λ, δ_1, and δ_2. For the test to make sense, the null model should be *nested* in the alternative; we will define this concept shortly. Let rss be the residual sum-of-squares of the alternative model, and let $rss^{(n)}$ be the residual sum-of-squares of the null model.

The test statistic, which is analogous to that for the analysis of variance in the parametric case, is

$$F = \frac{(rss^{(n)} - rss)/(\delta_1^{(n)} - \delta_1)}{rss/\delta_1}.$$

F has a distribution that is well approximated by an F distribution with denominator look-up degrees of freedom ρ, defined earlier, and numerator look-up degrees of freedom

$$\nu = \frac{(\delta_1^{(n)} - \delta_1)^2}{\delta_2^{(n)} - \delta_2}.$$

The concept of a null model being nested in the alternative expresses the idea that the alternative is capable of capturing any effect that the null can capture, but the definition is more precisely a specification of when it makes sense to use the analysis of variance to compare two models. The null is nested in the alternative if the following conditions hold:

(1) $\alpha^{(n)} \geq \alpha$.

(2) $\lambda^{(n)} \leq \lambda$.

(3) If the square of a factor is dropped from the alternative model, then it must not be present in the null model; the converse need not be true.

(4) The models must have the same factors with the following exception: a conditionally parametric factor in the alternative need not be present in the null; if present, though, it must also be conditionally parametric.

Conditions (2) to (4) can be expressed in a different way. To explain, we need to differentiate *neighborhood variables*—the factors used to determine the neighborhoods in the loess fitting—and *fitting variables*—the factors that are fitted locally by weighted least squares. Let's take a specific example. Suppose there are three factors: u, v, and w. Suppose $\lambda = 2$, u is taken to be conditionally parametric, and the square of w is dropped. The neighborhood variables are v and w. The fitting variables are a constant, u, u^2, v, v^2, w, uv, and vw. Now we can reexpress (2) to (4) by the following:

$(2)'$ The null and alternative models have the same neighborhood variables.

$(3)'$ The fitting variables of the null model are a subset of the fitting variables of the alternative model.

The Equivalent Number of Parameters

Let

$$\mu = \mathrm{tr}(L'L).$$

If the $\hat{y}_i$ are the fitted values, then

$$\mu = \frac{\sum_{i=1}^{n} \mathrm{Variance}(\hat{y}_i)}{\sigma^2}.$$

We will call μ the *equivalent number of parameters* since if the $\hat{y}_i$ were the fitted values for a linear model, the right side of the last equation would be the number of estimated parameters. μ is greater than or equal to τ, the number of fitting variables, and approaches τ as α tends to infinity. The equivalent number of parameters is one measure of the amount of smoothing. Strictly speaking, μ depends on α, on the values of the factors, and on the choices of the neighborhood and fitting variables. However, having selected all of these factors except α, we can get, approximately, a desired value μ by taking α to be $1.2\tau/\mu$, where τ is the number of fitting variables.

Symmetric Errors

When the error distribution is specified to be symmetric, inferences are based on *pseudo-values*. Let the robustness weights and the median absolute residual used in the final update of the fit, $\hat{g}(x)$, be r_i and m, respectively, and let $\psi(u;b) = uB(u;b)$. The pseudo-values are

$$\ddot{y}_i = \hat{y}_i + c r_i \hat{\varepsilon}_i$$

where $\hat{y}_i$ are the fitted values, $\hat{\varepsilon}_i$ are the residuals, and

$$c = \frac{n}{\sum_{i=1}^{n} \psi'(\hat{\varepsilon}_i; 6m)}.$$

Inferences are carried out by applying the inference procedures of the Gaussian case but replacing the observations of the response y_i by the pseudo-values $\ddot{y}_i$. For example, suppose we want to compute a confidence interval for $g(x)$ about the robust estimate, $\hat{g}(x)$. Using the pseudo-values as the response, we compute a Gaussian-error estimate, ρ, and $s(x)$ as described above. The confidence interval for $g(x)$ is the $\hat{g}(x)$ plus and minus $s(x)$ times a t value with ρ degrees of freedom. The true coverage using this procedure is well approximated by the nominal coverage. For the analysis of variance, we proceed in a similar fashion using the pseudo-values from the alternative model and carrying out the Gaussian-error procedures. For small samples, the approximation is not as good as for confidence intervals and produces optimistic results, but work is under way to find methods for adjusting degrees of freedom that will improve the approximations.

Errors with Unequal Scales

Suppose we have specified that the random errors ε_i in the model have the property that $a_i \varepsilon_i$ are identically distributed where the *a priori weights*, a_i, are positive and known. Then various modifications are made to the methods of inference.

For the Gaussian-error estimate, the operator matrix L is, of course, different from that in the equal-variance case, but δ_1 and δ_2 are defined in terms of L as before. The estimate of σ becomes

$$s = \sqrt{\frac{\sum_{i=1}^{n} a_i \hat{\varepsilon}_i^2}{\delta_1}}$$

and the estimate of the standard deviation of $\hat{g}(x)$ becomes

$$s(x) = s\sqrt{\sum_{i=1}^{n} l_i^2(x)/a_i}.$$

For the analysis of variance, all residual sum-of-squares are modified by adding the terms a_i, as done above for s.

For the robust estimate, the median absolute residual is defined using the *standardized residuals*

$$\hat{\varepsilon}_i^* = \sqrt{a_i}\hat{\varepsilon}_i$$

That is,

$$m = \text{median}(|\hat{\varepsilon}_i^*|).$$

Similarly, the robustness weights are

$$r_i = B_{6m}(\hat{\varepsilon}_i^*).$$

The pseudo-values are

$$\ddot{y}_i = \hat{y}_i + c r_i \hat{\varepsilon}_i$$

where c is now

$$c = \frac{n}{\sum_{i=1}^n \psi'(\hat{\varepsilon}_i^*; 6m)}.$$

4.2 Computational Methods

Interpolation by k-d Trees and Blending

The *k-d* tree is a particular data structure for partitioning space by recursively cutting cells in half by a hyperplane orthogonal to one of the coordinate axes (Bentley, 1975). For our application, the k in the name refers to the number of neighborhood variables, those factors that are used to define the neighborhoods.

Here is how the *k-d* tree is formed. Start with a rectangular cell just containing the values of the neighborhood variables. Pick the factor whose spread is the greatest and divide the cell in half at the median along the axis of that factor. Recursively apply the same division procedure to each subcell. If a cell contains fewer than βn points, where β is a small fraction, do not refine it. Figure 28 shows a *k-d* tree for two factors, $n = 500$, and $\beta = 0.05$.

Once the *k-d* tree is built, $\hat{g}(x)$ is directly computed at the vertices. By "vertex," we just mean a corner of a cell; "vertex" seems a better term than "corner" because a vertex of one cell typically lies in the middle of a side of an adjacent cell. In addition to computing $\hat{g}(x)$ at a vertex, a derivative of $\hat{g}$ at the vertex is approximated by the derivative of the locally-fitted surface. This derivative is a natural by-product of the least-squares computation and costs nothing extra to obtain.

Typically, the number of vertices, v, will be much smaller than n. This is at least true asymptotically, because the number of cells needed to achieve a certain accuracy of approximation depends on the smoothness of $\hat{g}(x)$, not n. In Figure 28 there are 66 vertices, so we solve 66 least-squares problems instead of one problem per evaluation of $\hat{g}(x)$. (Recall that for the galaxy surface we carried out 5841 evaluations to make a contour plot.) The amount of work in general to construct the *k-d* tree, including vertex coefficients, is $O(v((1.5 + \alpha\tau)n + \tau^3))$. After building the tree, each interpolation costs $O(\log v)$. Since τ is fixed and v is asymptotically bounded, the total running time is linear in the size of the input and output.

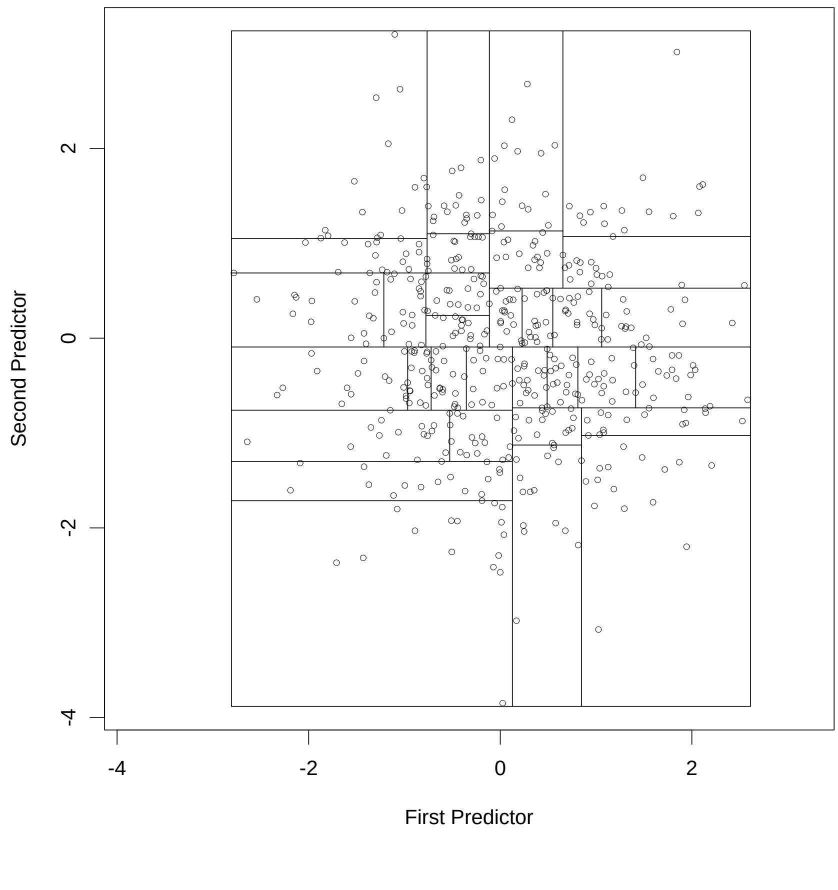

Figure 28: A k-d tree.

Let's turn now to the scheme used to build a piecewise polynomial approximation to $\hat{g}$. To simplify the discussion, we will suppose that there are two neighborhood variables. For our *k-d* tree, the boundary of each rectangular cell is cut into segments by vertices. (There are four sides, some of which will likely contain internal vertices, breaking them into more segments.) On each segment, the surface is interpolated using the unique cubic polynomial determined by the fitted function and derivative values at the vertices. To interpolate in the interior of the cell, we apply blending functions, also known as transfinite interpolants (Cavendish, 1975). This technique, well known in computer-aided design, takes a certain combination of univariate interpolants in each variable separately to build a surface. In effect, each cell is subdivided and on each piece a cubic polynomial in two variables is constructed although the computation is not actually done this way.

For one and two neighborhood variables, the interpolation function is C^1, but for more variables, the present code does not use enough vertices to guarantee a consistent approximation across cell facets. Hence the overall approximation may not be C^1 or even C^0. This defect will be removed in a future implementation.

Computing δ_i

Three statistical quantities are described in Section 4.1 that provide information about degrees of freedom—μ, ρ, and ν. These three quantities are functions of δ_1, δ_2, and n. Straightforward computation of the δ_i is horrendously expensive, so we have developed methods of approximation. First, we generated a large number of datasets, each with a response and one or more factors, and computed the δ_i for each. We discovered, through substantial graphical analysis, that the δ_i could be predicted to within a few percent by the following factors: λ, n, τ, and

$$\zeta = \frac{\sqrt{\tau / \operatorname{tr}(L)} - \sqrt{\tau/n}}{1 - \sqrt{\tau/n}}.$$

The model that was fitted is semiparametric, involving both parametric functions and a local regression model.

Error Messages from the Bowels of Loess

Although loess fitting is based on sound numerical methods, some delicate situations can arise that require the judgment of the user. When problems are detected by the loess routines, messages are transmitted up to the user.

One class of messages involves the smoothing parameter α. In order for the least-squares problem in a direct computation of $\hat{g}(x)$ to be well posed, α must be large enough that there are as many data points in the neighborhood as fitting variables, τ, in the local regression. Moreover, since neighborhood weights drop to 0 at the boundary, at least τ of these points must be strictly inside the neighborhood. If α is too small, the fix is to increase it or reduce τ by lowering λ or dropping squares.

The sample points must be sufficiently well distributed as well as sufficiently numerous. For example, consider locally quadratic fitting in one factor. If, because of multiplicities, there are only two distinct sample locations inside a neighborhood, then a quadratic polynomial is not uniquely determined.

When numerical problems arise because of poor conditioning of the design matrix of the local regression, small eigenvalues are set to zero and a pseudo-inverse message is sent. None of this means the fit has a problem, but a pseudo-inverse message is a caution that extra alertness must be used in examining the diagnostic displays.

Mathematically, $\mathrm{tr}(L)$ is greater than or equal to τ, the number of fitting variables. Numerically, however, if eigenvalues are set to zero, $\mathrm{tr}(L)$ can drop below τ, which causes the method of computing δ_i approximately to abort. If this indicator of an eigenvalue meltdown occurs, the coded message "Chernobyl" is raised.

Finally, when the interpolation method is used, the code must allocate space based on a prediction from the number of observations, the number of factors, and the specification of the surface and errors. If this allocated space is too small, the *k-d* tree division is truncated and a warning message sent up. In some cases the problem is extreme enough that the fit is not carried out; this necessitates increasing the value of α.

5 Bibliographic Notes

Local regression models are treated in detail in a new book by Cleveland and Grosse (forthcoming). But methods of local fitting date back at least to the 1920s. Initial applications were to smooth a time series (Macauley, 1931). An early use of local fitting for the general regression problem was investigated by Watson (1964). The method amounted to fitting a constant locally—in other words, taking the polynomial degree λ to be zero. This came to be known as kernel smoothing. It leads to very interesting theoretical work but is not of use in practice since it is hard to coax the method into following the patterns in most datasets. More serious attempts at local fitting were suggested by McLain (1974), who fitted quadratic polynomials, and Stone (1977), who fitted linear polynomials. The method of fitting used here was described by Cleveland (1979) for one factor. Cleveland and Devlin (1988) extended the method to two or more factors and investigated the sampling properties in the Gaussian case. (Sampling properties in the symmetric case are still under development.) The computational methods described in Sections 3 and 4, which are crucial to local regression being useful in practice, are described in full detail by Cleveland, Devlin, and Grosse (1987) and Cleveland and Grosse (1991).

References

Bentley, J. L. (1975) Multidimensional binary search trees used for associative searching. *Comm. Assoc. Comp. Mach.*, **18**, 509-517.

Brinkman, N. D. (1981) Ethanol fuel — a single-cylinder engine study of efficiency and exhaust emissions. *SAE Transactions*, **90**, No. 810345, 1410-1424.

Bruntz, S. M., Cleveland, W. S., Kleiner, B., and Warner, J. L. (1974) The dependence of ambient ozone on solar radiation, wind, temperature, and mixing height. *Symposium on Atmospheric Diffusion and Air Pollution*, 125–128, Boston: American Meteorological Society.

Buta, R. (1987) The structure and dynamics of ringed galaxies, III: surface photometry and kinematics of the ringed nonbarred spiral NGC7531. *The Astrophysical J. Supplement Ser.*, **64**, 1-37.

Cavendish, J. C., (1975) Local mesh refinement using rectangular blended finite elements. *J. Comp. Physics*, **19**, 211-228.

Cleveland, W. S. (1979) Robust locally-weighted regression and smoothing scatterplots. *J. Am. Statist. Ass.*, **74**, 829-836.

Cleveland, W. S. and Devlin, S. J. (1988) Locally-weighted regression: an approach to regression analysis by local fitting. *J. Am. Statist. Ass.*, **83**, 596-610.

Cleveland, W. S., Devlin, S. J., and Grosse, E. (1988) Regression by local fitting: methods, properties, and computational algorithms. *J. Econometrics*, **37**, 87-114.

Cleveland, W. S. and Grosse, E. (forthcoming) *Fitting Curves and Surfaces to Data.*

Cleveland, W. S. and Grosse, E. (1991) Computational methods for local regression. *Statistics and Computing*, **1**, 47-62.

Macauley, F. R. (1931) *The Smoothing of Time Series* New York: National Bureau of Economic Research.

McLain, D. H. (1974) Drawing contours from arbitrary data points. *Computer J.*, **17**, 318-324.

Stone, C. J. (1977) Consistent nonparametric regression. *Ann. of Stat.*, **5**, 595-620.

Watson, G. S. (1964) Smooth regression analysis. *Sankhya* A, **26**, 359-372.

Appendix: Obtaining the Loess Routines Electronically

The C and Fortran routines, all of which are freely available, may be obtained by sending electronic mail to

```
netlib@research.att.com
```

a mailbox at AT&T Bell Laboratories in Murray Hill, NJ. The message

```
send dloess from a
```

should be sent. The routines are double precsion.

The file `dloess` is a so-called "shell archive" or "bundle". Moreover, in order to send this 172 kilobyte file to you by email, netlib breaks it into pieces which are themselves shell archives. So you'll need to run `sh` once on each piece of mail to reconstruct the file `dloess`, then run `sh dloess` to finally reconstruct all the source files.

Subroutines from linpack, which are called by the Fortan code, are not included. If they are not already on your system, send the message

```
send d1mach dnrm2 dsvdc dqrdc ddot dqrsl idamax from linpack core
```

to the same address. When installing, don't forget to uncomment the appropriate DATA statements in d1mach, as described by the comments in those functions.

The PostScript file for this user manual is also available by email

```
send cloess.ps from a
```

but since it is over half a megabyte, `ftp` is a better choice

```
ftp research.att.com
login: netlib
password: <your email address>
binary
cd a
get cloess.ps.Z
quit
uncompress cloess.ps
```

Bug reports will receive prompt attention. Send electronic mail to

```
shyu@research.att.com
```

or send paper mail to

```
Ming-Jen Shyu
AT&T Bell Laboratories
600 Mountain Avenue, room 2C-263
Murray Hill NJ 07974
USA
```