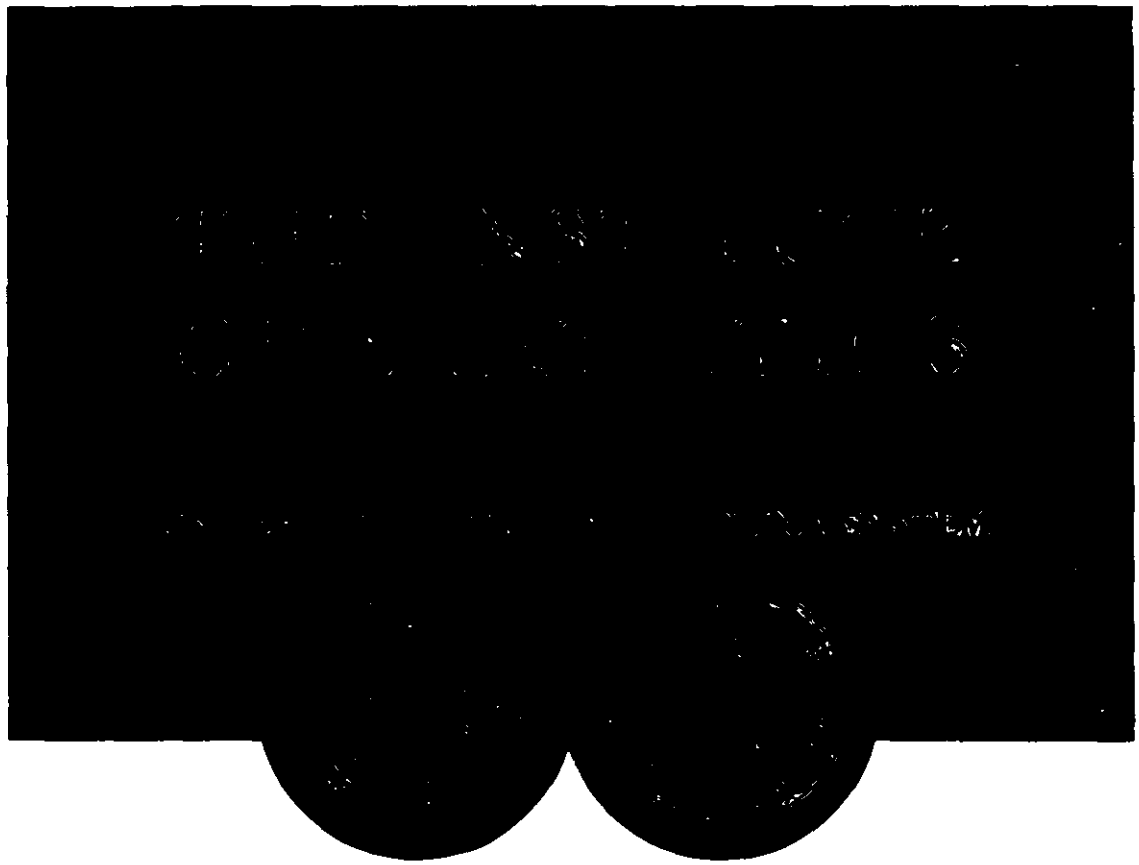
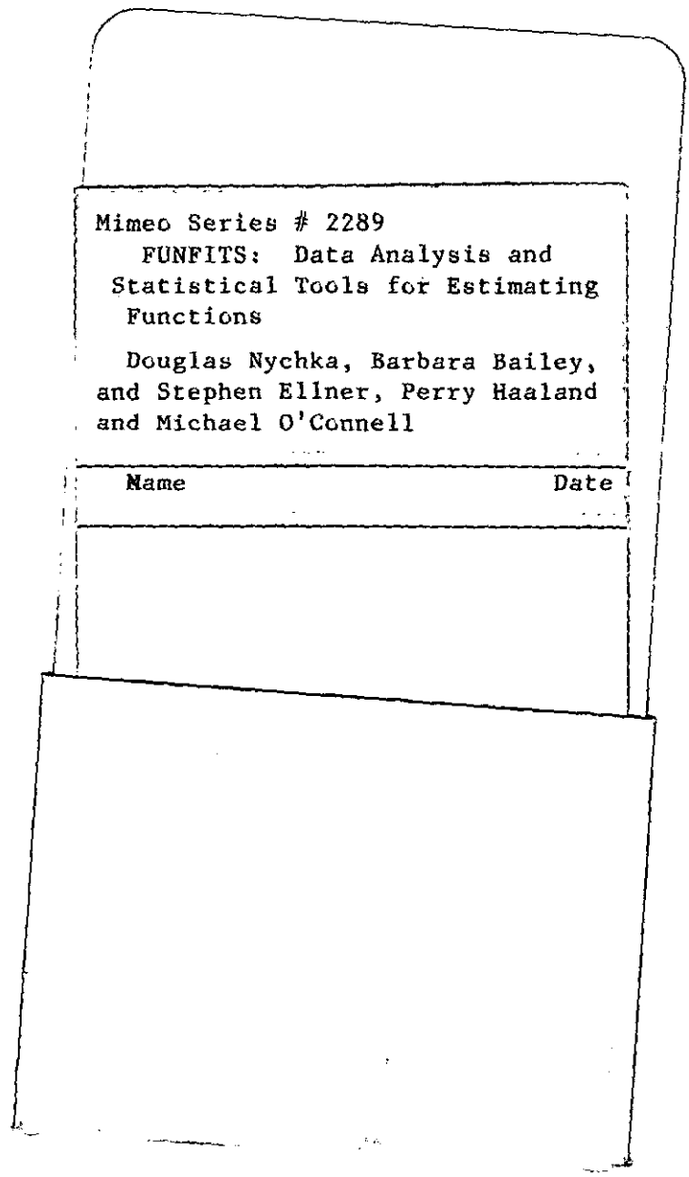


FUNFITS: Data Analysis and Statistical Tools for Estimating Functions

Douglas Nychka, Barbara Bailey and Stephen Ellner
Perry Haaland and Michael O'Connell

Institute of Statistics Mimeo Series #2289

NORTH CAROLINA STATE UNIVERSITY
Raleigh, North Carolina



Mimeo Series # 2289

FUNFITS: Data Analysis and Statistical Tools for Estimating Functions

Douglas Nychka, Barbara Bailey, and Stephen Ellner, Perry Haaland and Michael O'Connell

Name Date

FUNFITS: data analysis and statistical tools for estimating functions

Douglas Nychka, Barbara Bailey and Stephen Ellner*
North Carolina State University

Perry Haaland and Michael O'Connell
Becton Dickinson Research Center

Abstract

A module of functions has been created to simplify and extend the fitting of curves and surfaces to data. Some important applications include estimates of the mean value of a measured response over some space of interest, response surface methods for designed experiments and the modeling of nonlinear dynamic systems. The core methods that are implemented in FUNFITS include prediction, plot and summary functions that help the user interpret results and visualize the estimate. FUNFITS is written mainly in the S programming language with some support from FORTRAN subroutines. All source is available and easy to modify.

*Contact Address: Department of Statistics Box 8203, North Carolina State University, Raleigh, NC, 27695-8203. Please visit the FUNFITS homepage to learn of revisions to this manual (Go to the NCSU statistics department `http://www.stat.ncsu.edu/` then to Douglas Nychka's home page under faculty. You will find FUNFITS as an item on this home page.) This work was supported in part by National Science Foundation Grant DMS-9217866-02, Becton Dickinson Research Center and the Environmental Protection Agency. We would also like to thank Jeffrey Andrew Royle, Qing Yang and Jerry Davis for their help on the spatial statistics functions.

Contents

1 Introduction

FUNFITS is a suite of functions that enhance the S-PLUS^{TM} statistical package and facilitate curve and surface fitting. This software grew up as a project to augment S-PLUS with additional nonparametric fitting strategies. Part of this effort was to focus on three areas rich in applications of function estimation and inference: spatial statistics, response surface methodology and nonlinear time series/dynamical systems. This activity has also lead to methods for spatial sampling and design of experiments when the response is expected to be a complex surface. Implementation of these methods with an object oriented approach gives a high level of integration within the FUNFITS package. This allows the models to be easily fit, summarized and checked. The main methodological contributions of FUNFITS are

- Thin plate splines
- Spatial process models
- Space filling designs
- Neural network regression
- Response surface methodology using nonparametric regression
- Nonlinear autoregressive process models and the analysis of nonlinear dynamic systems

Where possible the statistical methods are supported by generic functions that provide predicted values and derivatives, diagnostics plots and simple ways to visualize the functions using contour and surface plots. In addition there are generic functions to optimize the estimated function and to find a path of steepest ascent. Some discipline has been kept to implement the methods in the S language, however a few of the numerically intensive steps are completed using FORTRAN subroutines. All the source code is included in the distribution so it is possible to examine the algorithms when needed. The emphasis on S code and the organization of the FORTRAN objects helps to make parts of the package easy to modify.

This manual gives an overview of the FUNFITS package and provides some background to the statistical models and the computations. Besides the general treatment given herein the user is referred to help files for each of the functions that provide more detailed information. At the end of the manual are two sections that list the README file for installing the FUNFITS package and an selected of FUNFITS functions and data sets.

2 What's so special about FUNFITS?

This project was started with an interest in implementing surface and curve fitting methods that provide a global representation for the estimated functions. This form is complementary to local methods such as `loess` already supported by S-PLUS. By a global method we mean that the function has a specific (closed form) formula depending on a set of parameters. For example

$$\hat{f}(x) = \sum_{k=1}^{M} c_k \psi_k(x)$$

where $\{\psi_k\}$ are a set of basis functions and $\{c_k\}$ are estimated parameters. Of course the nonparametric flavor of this estimate means that M is similar in size to the number of data points and the estimates of the parameters are constrained in such a way that the resulting estimate is smooth. Global methods are especially useful for smaller sample sizes, low noise situations or applications where some inferences are required.

Although the use of this package requires some knowledge of S-PLUS, many of the FUNFITS functions are easy to use and require a minimum of S-PLUS commands. Also, there are additional functions in FUNFITS that help to round the corners of what we perceive to be rough edges in S-PLUS. Faced with the unpleasant task of learning a new programming/data analysis language, the reader should keep in mind that no system will make function and curve fitting a trivial exercise. The data structures can be fairly complicated and some scrutiny and interaction with the data is always required to prevent nonsensical results. Because S-PLUS already supports many statistical functions and has very flexible graphics it is a natural platform for FUNFITS. For a gradual introduction to S-PLUS the reader might take a look at S-Lab (Nychka and Boos, 1994), a series of labs teaching data analysis and some of the some of basic functions of S-PLUS incrementally. The overview by Ripley and Venables (1994) is also helpful.

2.1 An example

Another project goal is to use the object oriented features of the S language to make the functions easier to use. To illustrate these ideas here is an example of fitting a thin-plate spline surface to the summer 1987 ozone averages for 20 stations in the Chicago urban area. The data is in the S data set, **ozone** with the components: **x** a 20×2 matrix with approximate Cartesian coordinates[1] for the station locations, a vector **y** with the average ozone values for each station, and **lon.lat**, a matrix with the geographic locations of the stations. The text following the S-PLUS prompt: > is what has been typed in (comments follow the pound sign).

```
> ozone  # print the ozone data to the screen
> plot(ozone$lon.lat) #plot locations of stations
> US(add=T)     # with medium resolution US map overlaid
> ozone.tps<- tps(ozone$x, ozone$y)   # thin plate spline fit to data
```

The results from fitting the thin plate surface to the 20 mean ozone values are now stored in the output data set **ozone.tps**. Since a smoothing parameter has not been specified the default is to estimate the amount of smoothness in the surface by cross validation.

The next step is to look at a summary of the fit:

```
Call:
tps(x = ozone$x, y = ozone$y)

 Number of Observations:                          20
 Degree of polynomial null space ( base model): 1
 Number of parameters in the null space           3
```

[1] To make this example simpler and the smoothing comparable in the north-south and east-west directions, the geographic coordinates for each station have been converted to (approximate) X-Y rectangular locations in miles. See the section on spatial process estimates for an analysis of these data using lon-lat coordinates.

```
Effective degrees of freedom:                    4.5
Residual degrees of freedom:                     15.5
Root Mean Square Error:                          4.047
Pure Error:                                      NA
RMSE (PRESS):                                    4.4594
GCV (Prediction error)                           21.17
Mutiple R-squared:                               0.3521
Adjusted R-squared:                              0.07126
Q-squared:                                       -0.1868
PRESS:                                           397.7244
Log10(lambda):                                   -0.034
m, power, d, cost:                               2, 2, 4, 1
Residuals:
   min  1st Q  median 3rd Q   max
-6.712 -1.426 -0.4725 1.326 7.912
Method for specifying the smoothing parameter is  GCV
```

This summary indicates some basic features of the regression. With approximately 4.5 degrees of freedom estimated for the surface one can expect that the fit is similar to a quadratic fit (5 parameters) but must have more structure than just fitting a linear function. To assess the quality of the fit one can look at residual plots and the GCV function:

```
> plot(ozone.tps)
```

This graphical output is given in Figure 1. Based on the two plots of fitted values and residuals, we see that the estimate has some problems with over-fitting ozone at the lower values. Good models tend to have small values of the cross-validation function, and we see on the lower plots that models with higher effective numbers of parameters (or equivalently smaller values of the smoothing parameter lambda) also do not predict as well. One alternative in this case is to use the variances based on the daily ozone values as a way to determine the appropriate amount of smoothness.

One can also examine the fitted surface (see Figure 2):

```
> surface(ozone.tps,'Fitted ozone surface')
```

As expected from the summary we see that the estimated surface is similar to a quadratic function in shape. The surface has been restricted to the convex hull of the stations and so, for example, it is not extrapolated over Lake Michigan (the upper right corner of the region).

Besides a plot of the fitted surface, it is also easy to get some rough measure of uncertainty in the estimate. This involves two steps, first evaluating the standard errors at a grid of locations and then passing these evaluations to one of the surface plotting programs in S. Here is an example that gives the image plot in Figure 3.

```
> ozone.se <- predict.surface.se(ozone.tps)
> image(ozone.se, 'SE of fit')  # or use contour  or   persp
> points( ozone$x) # add in the station locations
> contour( predict.surface( ozone.tps), add=T) # add contours of the
>                                          #  predicted spline fit
```

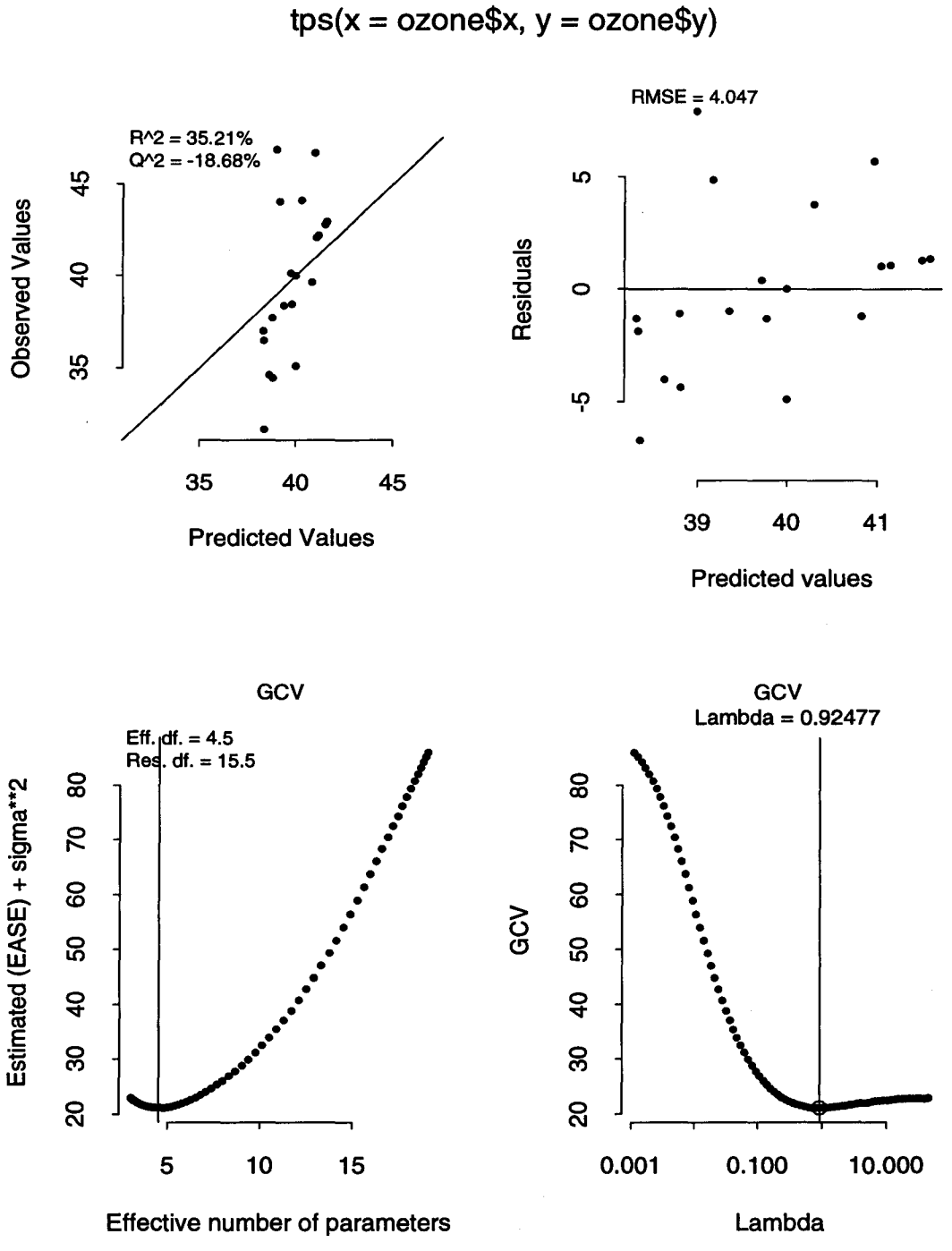


Figure 1: Summary plot of tps fit to the average daily ozone for 20 Chicago monitoring stations

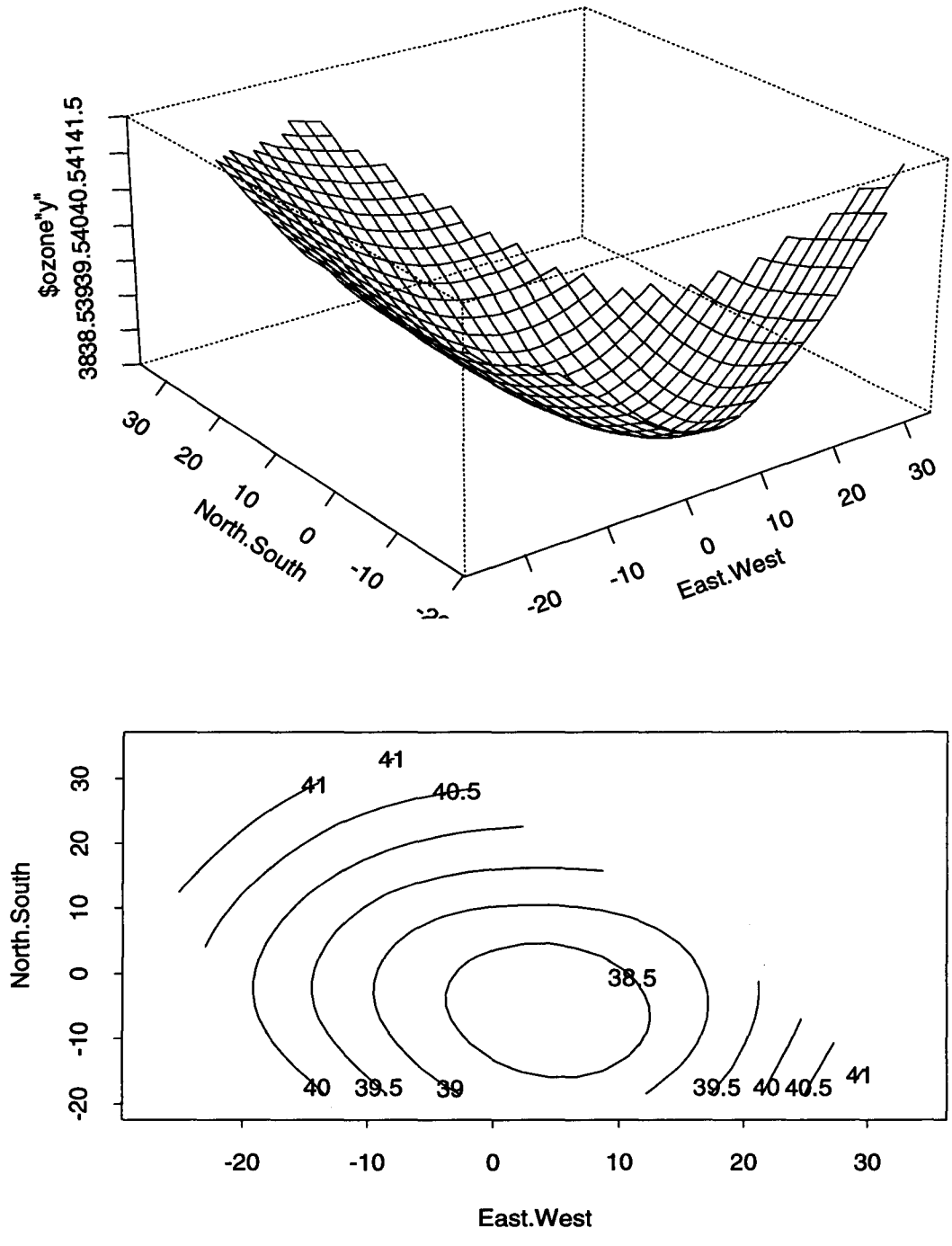


Figure 2: Average daily ozone surface for 20 Chicago monitoring stations

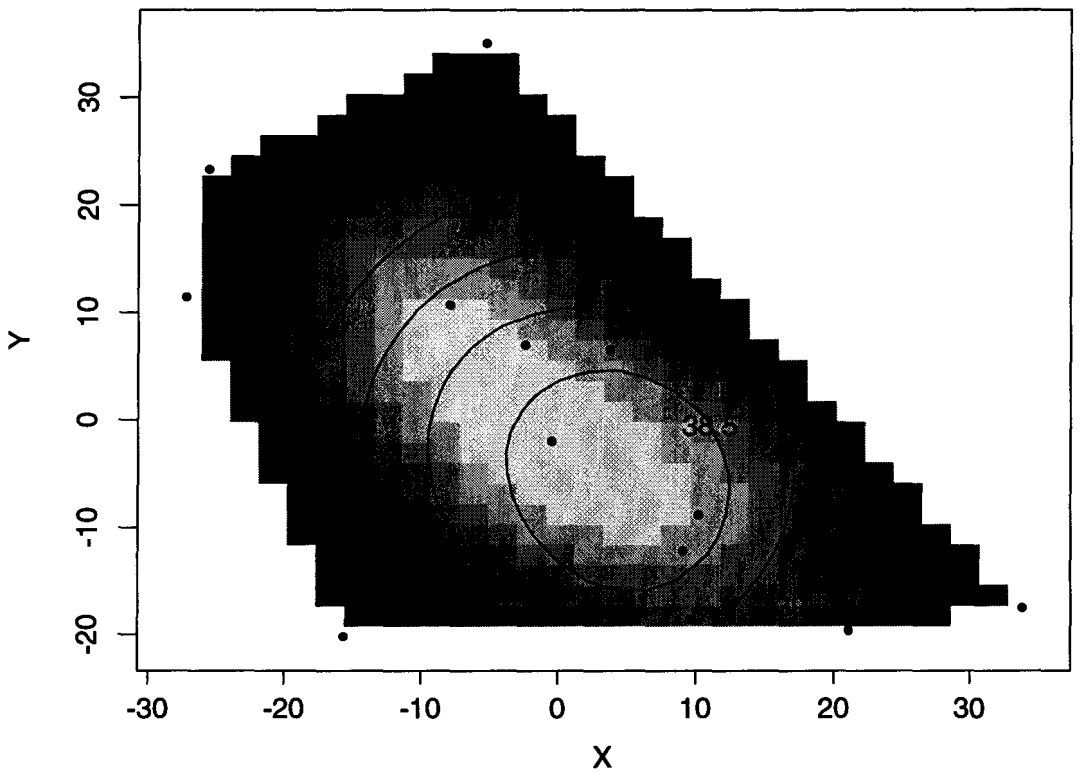


Figure 3: Estimated standard errors for mean ozone surface based on the GCV thin plate spline (grey levels). Estimated mean levels for ozone are given by contour lines and the points are station locations.

Thus with a minimal amount of work it is possible to fit, check and interpret a fairly complicated nonparametric regression method. The main intent of FUNFITS is that functions such as **surface** or **predict** should work with a variety of methods. The burden is on the software to sort out salient differences between the objects that are passed to the function and still produce sensible results. In this example, the **ozone.tps** object contains the description of the estimated spline function but the **ozone.se** object consists of standard errors evaluated on a (default) grid of points. In either case a perspective or contour plot of the function makes sense and thus the **surface** function is designed to handle both types of objects. An another important principle in this example is that the rich set of S-PLUS graphics functions are being leveraged to interpret the spline estimates found by a FUNFITS function. In this way the FUNFITS module is small and takes advantage of the many standard functions in S-PLUS.

3 A Basic model for regression

The suite of FUNFITS regression functions assume observations from the additive model:

$$Y_i = f(\boldsymbol{x}_i) + \epsilon_i \quad \text{for } 1 \leq i \leq n \tag{1}$$

where Y_i is the observed response at the ith combination of design variables $\boldsymbol{x}_i \in \Re^d$ and f is the function of interest. The random components, $\{\epsilon_i\}$, usually associated with the measurement errors, are assumed be independent, zero-mean random variables with variances, $\{\sigma^2/W_i\}$.

One efficient strategy for representing f is as the sum of a low order polynomial and a smooth function:

$$f(\boldsymbol{x}) = p(\boldsymbol{x}) + h(\boldsymbol{x})$$

In the case of thin plate splines (Wahba 1990, Silverman and Green 1994) the degree of p implies a specific roughness penalty on h based on integrated squared (partial) derivatives. Under the assumption that f is a realization of a spatial process, p can be identified as the "spatial trend" or "drift" and h is modeled as a mean zero Gaussian process (Cressie 1991).

Spline and spatial process models are sensitive to the dimension of $\boldsymbol{x}$ because they attempt to represent all possible interactions among the different variables. This feature is the well known "curse of dimensionality" and can be tied to an exponential increase in the number of interactions as the dimension of $\boldsymbol{x}$ increases. Neural network regression (Cheng and Titterington 1994, Ripley and Venables 1994) is less sensitive to fitting high dimensional functions. For regression problems, a useful form is a single hidden layer, feed forward network

$$f(\boldsymbol{x}) = \beta_0 + \sum_{k=1}^{M} \beta_k \phi(\mu_k + \gamma_k^T \boldsymbol{x}) \tag{2}$$

where

$$\phi(u) = \frac{e^u}{1+e^u}.$$

and β, μ and γ_k , $1 \leq k \leq M$ are parameters to be estimated. In this form the dimension reduction is achieved by only considering a small number of projections (M).

Another way to simplify the structure of f is by assuming that it is additive in the variables:

$$f(\boldsymbol{x}) = \sum_{k=1}^{d} f_k(\boldsymbol{x}_k) \tag{3}$$

In FUNFITS the full interactive model (1) can be estimated by the functions **tps** and **krig**, the neural net model (2) is fit by **nnreg** and the additive model (3) by **addreg**. In addition there is a fast, one dimensional cubic spline function, **sreg**. It should be noted that as part of S-PLUS there exist excellent functions for fitting additive models and one-dimensional functions using splines (**gam , smooth.spline**). The main advantage of the FUNFITS version of these methods are efficiency and access to the all of the source code.

4 Thin plate splines: tps

This section gives some back ground for the two surface fitting functions, **tps** and **tpsreg**. Both of these fit thin plate splines the main difference being **tps** is written essentially all in the S language and **tpsreg** has less options and executes a FORTRAN program outside of S-PLUS.

Splines are usually defined implicitly as functions that solve a variational (minimization) problem. The estimator of f is the minimizer of the penalized sum of squares

$$S_\lambda(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}_i))^2 + \lambda J_m(f)$$

for $\lambda > 0$. The thin plate spline is a generalization of the usual cubic smoothing spline with a "roughness" penalty function $J_m(f)$ of the form

$$J_m(f) = \int_{\Re^d} \sum \frac{m!}{\alpha_1!\ldots\alpha_d!}\left(\frac{\partial^m f}{\partial x_1^{\alpha_1}\ldots\partial x_d^{\alpha_d}}\right)^2 d\boldsymbol{x}$$

with the sum in the integrand being over all non-negative integer vectors, α, such that $\sum\alpha_1 + \cdots + \alpha_d = m$, and with $2m > d$. Note that for one dimension and $m = 2$, $J_2(f) = \int \left(f^{(\prime\prime)}(x)\right)^2 dx$, giving the standard cubic smoothing spline roughness penalty.

An important feature of a spline estimate is the part of the model that is unaffected by the roughness penalty. The set of functions where the roughness penalty is zero is termed the *null space*, and for J_m, consists of all polynomials with degree less than or equal to $m-1$.

The minimization of S_λ results in a function that tracks the data but is also constrained to be smooth. A more Bayesian viewpoint is to interpret S_λ as the negative of a posterior density for f given the data Y. The roughness penalty then is equivalent to a prior distribution for f as a realization from a smooth (almost m differentiable) Gaussian process. This is the basis for the connection between splines and spatial process estimates.

4.1 Determining the smoothing parameter

The discussion in the preceding section is given under the assumption the smoothing parameter, λ is known. In most applications it is important to explore estimates of λ derived from the data. One standard way of determining λ is by generalized cross-validation (GCV). Although not obvious at this point, thin-plate splines are linear functions of the observed

dependent variable. Let $A(\lambda)$ denote the $n \times n$ smoothing or "hat" matrix that maps the Y vector into the spline predicted values.

$$(\hat{f}(\boldsymbol{x}_1), ..., \hat{f}(\boldsymbol{x}_n))^T = A(\lambda)Y$$

Given this representation, the trace of $A(\lambda)$ is interpreted as a measure of the number of effective parameters in the spline representation. The residuals are given by $(I - A(\lambda))Y$ and thus $n - trA(\lambda)$ is the degrees of freedom associated with the residuals. The estimate of the smoothing parameter can be found by minimizing the GCV function

$$V(\lambda) = \frac{\frac{1}{n}Y^T(I - A(\lambda)^T W(I - A(\lambda))Y}{(1 - trA(\lambda)/n)^2}$$

One variation on this criterion is to the replace the denominator by

$$(1 - (\mathcal{C}(trA(\lambda) - t) + t)/n)^2.$$

Here t is the number of polynomial functions that span the null space of J_m and $\mathcal{C}$ is a cost parameter that can give more (or less) weight to the effective number of parameters beyond the base polynomial model (null space of the smoother). Once $\hat{\lambda}$ is found, σ^2 is estimated by

$$\hat{\sigma}^2 = \frac{Y^T(I - A(\hat{\lambda}))^T W(I - A(\hat{\lambda}))Y}{n - tr(A(\hat{\lambda}))}.$$

and is analogous to the classical estimate of the residual variance from linear regression.

4.2 Approximate splines for large data sets

The formal solution of the thin-plate spline has as many basis functions as unique $\boldsymbol{x}$ vectors. Except in one-dimension, this makes computation and storage difficult when the sample size exceeds several hundred. The basic problem is that the spline computation includes a singular value decomposition of a matrix with dimensions comparable to the number of basis functions. A solution to this problem in FUNFITS is to use a reduced set of basis functions, described by a sequence of "knot" locations possibly different from the observed $\boldsymbol{x}$ values. Given below is a short example that goes through the steps of using a reduced basis.

The data set **flame** is an experiment to relate the intensity of light from ionization of Lithium ions to characteristics of the ionizing flame. The dependent variables are the flow rate of the fuel and oxygen supplied to the burner used to ionize the sample and data consists of 89 points. Here is the fit to these data using a spline where the base model is a cubic polynomial :

```
flame.full.fit<- tps( flame$x, flame$y, m=4)
```

To reducing the number of basis functions, one could construct a grid of knots that are different from the observed values and smaller in number. The first step is to create the new grid of knots (a 6×6 grid in the example below) and then refit the spline with these knots included in the call to **tps**.

```
> g.list <- list(Fuel=seq(3,28,,6), O2=seq(15,39,,6))
> knots.flame <- make.surface.grid(g.list)
> flame.tps <- tps(flame$x,flame$y, knots=knots.flame, m=4)
```

Here the grid list and the function **make.surface.grid** have been used to construct the two dimensional grid. As an alternative, one could consider a random sample of the observed X values, `knots.flame<- flame$x[sample(1:89,36),]` or chose a subset based on a coverage design, `knots.flame<- cover.design( flame$x, 36)`. Of course in either of these cases the number of points, 36, is flexible and is based on choosing the number large enough to accurately represent the underlying function but small enough to reduce the computational burden.

4.3 Standard Errors

Deriving valid confidence intervals for nonparametric regression estimates is still an active area of research and it appears that a comprehensive solution will involve some form of smoothing that can vary at different locations. To get some idea of the precision of the estimate, in FUNFITS, a standard error of the estimate is computed based on the assumption that the bias is zero. This is not correct and such confidence intervals may be misleading at points on the surface where there is fine structure such as sharp minima or maxima. However, empirical evidence suggests that such intervals provide a reasonable measure of the estimate's average precision if the smoothing parameter is chosen to minimize the mean squared prediction error. The standard error can be adjusted (inflated) by the factor $\sqrt{tr(A(\lambda))/tr(A(\lambda)^2)}$. This makes the average squared standard errors across the design points approximately equal to the average expected squared error. Some theoretical justification for this seemingly *ad hoc* fix is given in Nychka (1988). Operationally one would follow the steps:

```
> ozone.tps<- tps(ozone$x, ozone$y)
> ozone.se <- predict.surface.se(ozone.tps)
```

To adjust, the $z component of the surface object is multiplied by this inflation factor.

```
> inflate <- sqrt(ozone.tps$trace/ozone.tps$trA2)
> ozone.se$z <- ozone.se$z*inflate
> contour( ozone.se) # now plot adjusted standard errors
```

In some extreme cases the actual coverage may drop from the 95% nominal level to 75%. However in most situations such intervals are useful for assessing the order of magnitude of the estimation error.

5 Spatial process models: krig

The spatial process model used in this section assumes that $h(\boldsymbol{x})$ is a mean zero Gaussian process. Let

$$Cov(h(\boldsymbol{x}), h(\boldsymbol{x}')) = \rho k(\boldsymbol{x}, \boldsymbol{x}')$$

be a covariance function where k is a known function and ρ is an unknown multiplier. Letting $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ the covariance matrix assumed for Y is $\rho K + \sigma^2 W^{-1}$ and under the assumption that ρ and σ^2 are known one can estimate $f(\boldsymbol{x})$ as the best linear, unbiased estimate for f based on Y (Cressie 1991, Welch et al. 1992). Recall that the basic model assumes that the measurement errors are uncorrelated with the i^{th} observation having a variance given by σ^2/W_i. In spatial statistics, the variance of the errors is often referred to as the nugget effect and includes not only random variation due to measurement error but

also fine scale variability in the random surface in addition to the larger scale covariance structure modeled by the covariance function.

To make the connection with ridge regression explicit it is helpful to reparametrize the parameter ρ as $\rho = \lambda\sigma^2$. The estimation procedures in **krig** actually determine λ and σ^2 directly and then these estimates are used to derive the estimate for ρ.

Measures of uncertainty for $\hat{f}(\boldsymbol{x})$ are found by evaluating

$$E[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2)]$$

under the assumption that ρ and σ are known and k is the correct covariance function. Because $\hat{f}(\boldsymbol{x})$ is a linear function of Y, it is a straight forward exercise to derive the prediction variance in terms of matrices that have been used to estimate the function. Of course, in practice, one must estimate the parameters of the covariance function, and in FUNFITS the variance estimate is obtained by substituting these estimated values into the variance formula. The user should keep in mind that this is an approximation, underestimating the true variability in the estimate.

5.1 Specifying the Covariance Function

The strength of the **krig** function is in the flexibility to handle very general covariance models The key idea is that in the call to **krig** one must specify an covariance function written in S-PLUS. This function should compute the cross covariance between two sets of points. A basic form for the arguments is **my.cov(x1,x2)** where **x1** and **x2** are matrices giving two sets of locations and the returned result is the matrix of *cross* covariances. Specifically if **x1** is $n_1 \times d$ and **x2** is $n_2 \times d$ then the returned matrix will be $n_1 \times n_2$. where the i, j element of this matrix is the covariance of the field between the location in the i^{th} row of **x1** and the j^{th} row of **x2**. Here is an example using the exponential covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = e^{-||\boldsymbol{x}-\boldsymbol{x}'||/\theta}$$

where the default value for the range parameter, θ, is set to 1.0.

```
> my.cov<- function(x1,x2,theta=1.0) { exp( -rdist(x1,x2)/theta) }
```

Here **rdist** is a handy function that finds the Euclidean distance between all the points in **x1** with those in **x2** and returns the results in matrix form. For geographic coordinates **rdist.earth** gives great circle distances for locations reported in longitude and latitude.

A function that has a few more options is the FUNFITS version **exp.cov** and this could be used as a template to make other covariance functions. At a more complex level is **EXP.cov** that makes use of FORTRAN routines to multiply the covariance function with a vector. Although more difficult to read, these modifications decrease the amount of storage required for evaluating the kriging or thin plate spline estimate at a large number of points.

Most covariance functions also require a range parameter, eg. θ in the case of the exponential. The parameters that appear in a nonlinear fashion in k are *not* estimated by the **krig** function and must be specified. The parameter values can be given in two ways, either by modifying the covariance function directly or changing the default value for a parameter when **krig** is called. Suppose that one wanted to use an exponential function with $\theta = 50$. One could either modify **my.cov** so that the default was **theta= 50** or in the call to **krig** include the **theta** argument:

```
> ozone.krig <- krig(ozone$x, ozone$y, my.cov, theta=50)
```

In general, any extra arguments given to **krig** are assumed to be parameters for the covariance function. Thus, **krig** makes a copy of the covariance function, substitutes these new defaults for the parameters and includes this modified function is part of the output object in the **cov.function** component. For example, looking at the **krig** object created above,

```
> ozone.krig$cov.function
function(x1, x2, theta = 50)
{
        exp( - rdist(x1, x2)/theta)
}
```

This is identical to the orignal version of the function except the default value for **theta** has been changed! In this way subsequent computation based on the krig object will use the correct parameter value.

There are well established methodologies for estimating the parameters of the covariance function and perhaps the most straightforward is nonlinear least squares fitting of the variogram. FUNFITS has a variogram function, **vgram** and many estimation methods are based on estimating parameters from the variogram using nonlinear least squares. Because S already provides support for nonlinear least squares fitting it is not neccesary to develop specialized variogram fitting routintes within FUNFITS. The example in the next section fits a variogram in this way using the S-PLUS function **nls**.

In contrast to the nonlinear parameters of the covariance function, the default for **krig** is to estimate the parameter λ (σ^2/ρ) based on cross-validation[2]. This difference is based on the fact that the qualitative aspects of the estimated surface are usually much more sensitive to the value of the smoothing parameter, λ, than the others. Also, this parameter appears in a linear fashion in the matrix expressions for the estimate, and so it is efficient to evaluate the estimate at different choices of this parameter. Parameters such as the range do not have a similar property, and the matrix decompositions for the estimates must be recalculated for different values of these parameters.

5.2 Some examples of spatial process estimates.

The data set **ozone2** contains the daily, 8-hour average ozone measurements during the summer of 1987 for approximately 150 locations in the Midwest (Figure 4). The component **y** in this dataset is a matrix of ozone values with rows indexing the different days and columns being the different stations. The locations can be plotted by

```
US( xlim= range( ozone2$lon.lat[,1]), ylim= range( ozone2$lon.lat[,2]))
points( ozone2$lon.lat, pch="o")
```

Most of the work in this example is identifying a reasonable covariance function for these data. Here is an example of computing and plotting the variogram for day 16.

```
y16<- c(ozone2$y[16,]
vgram.day16<- vgram( ozone2$lon.lat, y16, lon.lat=T)
plot( vgram.day16, xlim=c(0,150))
```

[2] In the next version we hope to add a maximum likelihood method for determining λ.

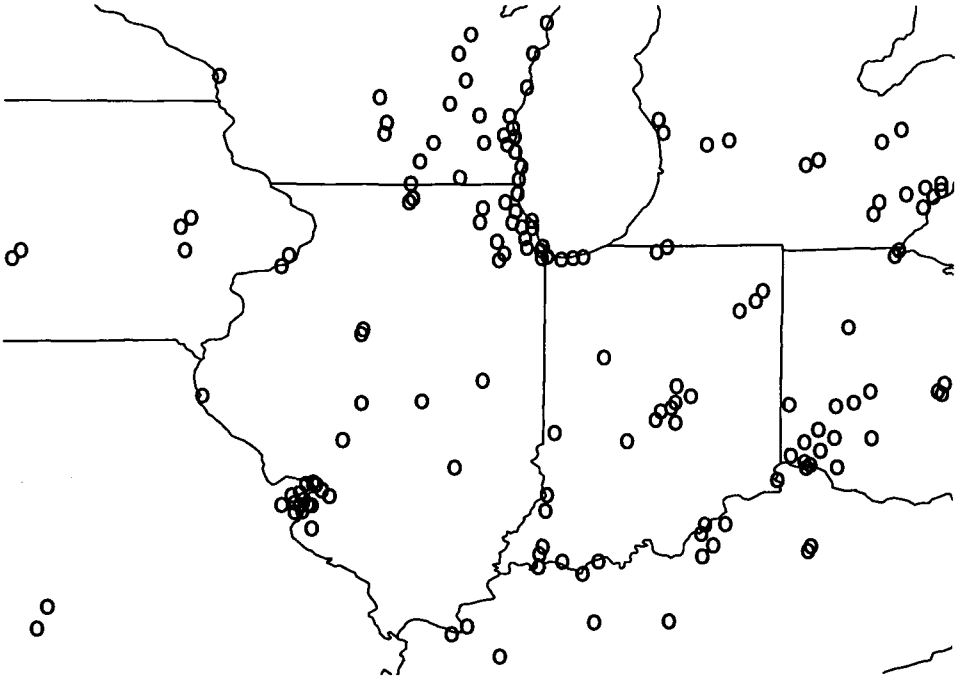


Figure 4: Locations of ozone monitoring stations

The variogram in this raw form is not very interpretable and one could use different smoothing methods such as loess or spline smoothing to smooth these points.

As an alternative, one could look at the correlogram to get an idea of the general distance scale for the correlations. Under the assumption that ozone over this period has a consistent spatial covariance from day to day one can consider the different days as replicated points and estimate the correlations directly. The following S code finds the pairwise correlations and plots them against distance.

```
> ozone.cor<- COR( ozone2$y)
> upper<- col(ozone.cor$cor)>row(ozone.cor$cor)
> cgram.oz<- list( d=rdist.earth( ozone2$lon.lat)[upper],
+                        cor= ozone.cor$cor[upper])
> plot( cgram.oz$d, cgram.oz$cor, pch=".")
```

Here we have use the FUNFITS function `COR` because it can handle missing values (in S-PLUS NA's). This is important because the missing value patterns are irregular and reducing down to a complete data set would exclude many of the days. One disadvantage is that `COR` is slow because it loops through all pairs of stations. Next one can fit an exponential correlation function to these sample correlations using nonlinear least squares:

```
> cgram.fit<- nls( cor ~ alpha* exp( -d/theta),
+       cgram.oz, start= list( alpha=.95, theta=200))

> summary( cgram.fit)

Formula: cor ~ alpha * exp( - d/theta)

Parameters:
          Value Std. Error t value
alpha   0.931247 0.00366247 254.268
theta 343.118000 2.37364000 144.554

Residual standard error: 0.120286 on 11626 degrees of freedom

Correlation of Parameter Estimates:
       alpha
theta -0.838
```

Based on this value for the range one can now remove the missing values for day 16 and use the **krig** function to fit a surface.

```
> good<-!is.na(y16)
> ozone.krig<- krig( ozone2$lon.lat[good,], y16[good],
+            exp.earth.cov, theta=343.1183)
```

In this case the nugget variance and the linear parameters of the covariance are found by GCV. Here is a summary of the results:

```
> summary(ozone.krig)
Call:
```

```
krig(x = ozone2$lon.lat[good,  ], Y = y16[good], cov.function = exp.earth.cov,
        theta = 343.1183)

 Number of Observations:                          147
 Degree of polynomial null space ( base model): 1
 Number of parameters in the null space           3
 Effective degrees of freedom:                    108.1
 Residual degrees of freedom:                     38.9
 MLE sigma                                        6.306
 GCV est. sigma                                   6.647
 MLE rho                                          2648
 Scale used for covariance (rho)                  2648
 Scale used for nugget (sigma^2)                  39.77
 lambda (sigma2/rho)                              0.01502
 Cost in GCV                                      1
 GCV Minimum                                      166.8
 Residuals:
   min  1st Q median 3rd Q   max
 -16.4 -1.697 0.0953 1.685 12.15
```

For this particular day the ozone surface has a high degree of variability, more so than a typical day in this period, and this is reflected by the large value of the rho parameter in the summary. Alternatively one could specify **sigma2** and **rho** explicitly in the call, and thus fix them at specific values. More is said about this below.

The next set of S calls evaluate the fitted surface and prediction standard errors on a grid of points. These evaluations are then converted to the format for surface plotting and some contour plots are drawn (see Figure 5).

```
set.panel(2,1)

predict.surface( ozone.krig)-> out.p
US( xlim= range( ozone2$lon.lat[,1]), ylim= range( ozone2$lon.lat[,2]))
contour( out.p, add=T)

predict.surface.se( ozone.krig)-> out2.p
US( xlim= range( ozone2$lon.lat[,1]), ylim= range( ozone2$lon.lat[,2]))
contour( out2.p,add=T)
points( ozone2$lon.lat)
```

The last part of this section illustrates how to extend this analysis to a covariance function that is not stationary. One simple departure from nonstationarity is to consider a covariance that has different marginal variances but isotropic correlations. Specifically consider

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma(\boldsymbol{x})\sigma(\boldsymbol{x}')e^{||\boldsymbol{x}-\boldsymbol{x}'||/\theta}$$

As a first step we can estimate the marginal variances, $\sigma(\boldsymbol{x})$ using a thin plate spline.

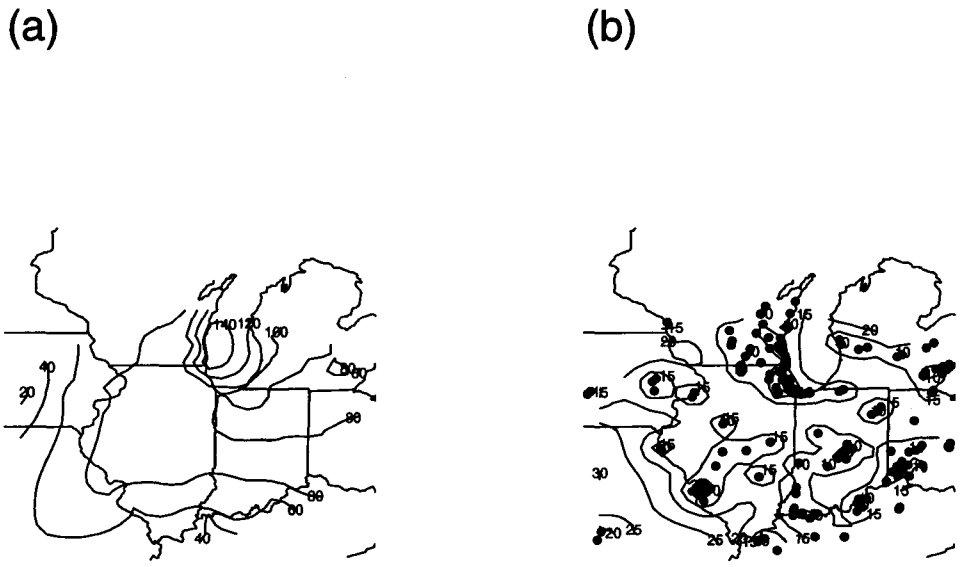


Figure 5: Fitted surface and standard errors of Midwest ozone network using an isotropic covariance function.

```
sd.tps<- tps( ozone2$lon.lat, ozone.cor$sd, return.matrices=F)
```

The **return.matrices** switch has been set to false so that the large matrices needed to find standard errors are not included in the output object. We do not need to calculate standard errors and using this smaller object in other functions is more efficient.

The next step is to call krig with a covariance function that uses this tps object to evaluate the marginal variances. The function **exp.isocor** in FUNFITS has been been written to do this, and the reader is referred to the help file for the details of its arguments. The main differences between this function and the other examples that have been given is that it needs the tps object [3] passed to it in order to evaluate $\sigma(x)$ and it also has an option to return the marginal variance at a set of locations instead of the covariance. This addition is important for **predict.se** to work correctly. Another difference is that this covariance builds in the nugget effect.

Here is an example of using **exp.isocor** based on the parameters found from the **nls** fitting of the correlogram. In the spirit of doing something useful with the results, the modeled covariances are compared to the observed ones (Figure 6). To limit the number of points on this plot only pairs of stations within 250 miles of each other are included.

```
> hold1<- diag(ozone.cor$sd)%*%(ozone.cor$cor)%*% diag(ozone.cor$sd)
> upper<- col(hold1)> row( hold1)
> hold1<- hold1[upper] # get upper triangle

> hold2<- exp.isocor( ozone2$lon.lat, obj=sd.tps, alpha = .93, theta=343, lon.lat=T)
> hold2<-hold2[upper]
> ind<- cgram.oz$D <= 250

> plot( hold1[ind], hold2[ind], pch='.', xlab='observed COV' , ylab='fitted
COV')
> abline(0,1)
```

To use **krig** to fit the surface one has two options. The first is to keep all of the covariance parameters fixed using the values estimated from the correlogram. Here is how to specify this model.

```
> weights<- 1/exp.isocor(ozone2$lon.lat, obj=sd.tps, alpha=.93, theta=343,
+     marginal =T) # reciprocal marginal variances at the observed locations

> fit.fixed<- krig( ozone2$lon.lat[good,], y16[good],
+     rho=.93, sigma2= (1-.93), weights=weights[good],
+     cov.function=exp.isocor, obj='sd.tps',  theta=343)
```

The values for **rho**, **sigma** and **weights** are set so that the covariance agrees with the nonstationary model given above. In **krig** the *name* of the tps object is passed to the covariance function as a character string to save storage.[4]

The second approach is to estimate the parameters ρ and σ from the data. The S code below is for finding both parameters using GCV.

[3] or a character string with the **tps** object's name

[4] The disadvantage is that subsequent computing with **fit.fixed** must also have the object **sd.tps** in the search path. In particular, it can not be deleted or changed!

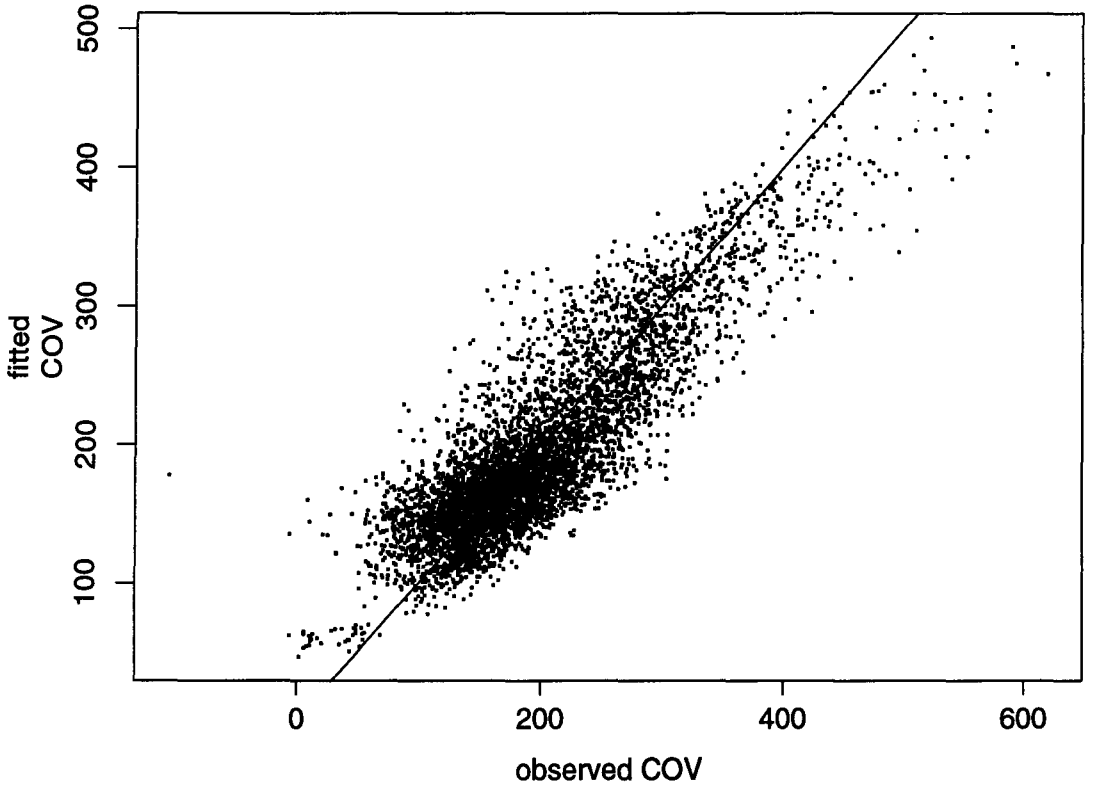


Figure 6: Fit to observed ozone covariances using isotropic correlation model. The plotted points are those pairs of stations separated by less than 250 miles.

```
> fit.GCV<-  krig( ozone2$lon.lat[good,], y16[good],
+       weights=weights[good],
+      cov.function=exp.isocor, obj='sd.tps',  theta=343)
```

This gives a very different fit to the data, and possibly reflects the fact that for this given day the ozone surface has more variability than on average. To compare the effect on the estimated prediction error using these two models, one can compare the standard errors.

```
> set.panel(2,2)
> out1.p <- predict.surface.se(fit.fixed)
> out2.p <- predict.surface.se(fit.GCV)
> contour(out1.p)
> contour(out2.p)

> out3.p<- out2.p
> out3.p<- out2.p$z/out1.p$z  #relative difference between the SE's
> contour(out3.p)
```

A compromise between these two models is to specify the ratio of "noise to signal", $\sigma^2/\rho = \lambda$ and let σ be determined by estimates based on the data at day 16. The net result is a covariance model where the average covariances are scaled by these two parameters.

```
fit.lambda<-  krig( ozone2$lon.lat[good,], y16[good], lambda=(1-.93)/.93,
     weights=weights[good],
     cov.function=exp.isocor, obj='sd.tps',  theta=343)
out4.p<- predict.surface.se( fit.lambda)
contour( out4.p)
```

6 Generalized ridge regression

Both the thin plate spline and the spatial process estimators have a common form as ridge regressions. This general form is at the heart of the numerical procedures for both methods so it is appropriate to outline it in this overview. Assume the model

$$Y = X\omega + \boldsymbol{e}$$

and suppose that H is a nonnegative definite.

The general form of the penalized (and weighted) least squares problem is minimize

$$(Y - X\omega)^T W(Y - X\omega) + \lambda\omega^T H\omega \tag{4}$$

over $\omega \in \Re^N$

The solution is

$$\omega = (X^T W X + \lambda H)^{-1} X^T W Y$$

The details of the computation are given at the end of this section. Given this general framework, the specific applications to thin plate splines and spatial process estimates involve identifying the relevant parameterization to get the X matrix and the penalty matrix, H.

Consider a function estimate of the form:

$$f(\boldsymbol{x}) = \sum_{j=1}^{t} \phi_j(\boldsymbol{x})\beta_j + \sum_{i=1}^{N} \psi_i(\boldsymbol{x})\delta_i$$

where $\{\psi_i\}$ are a set of N basis functions and ϕ are t low order polynomial functions[5] For both thin plate splines and the spatial process estimates, the parameter vectors δ and β are found by minimizing

$$(Y - T\beta - M\delta)^T W(Y - T\beta - M\delta) + \lambda\delta^T\Omega\delta$$

Here $\lambda > 0$, Ω is a nonnegative definite matrix, $T_{k,j} = \phi_j(\boldsymbol{x}_k)$, $M_{k,i} = \psi_i(\boldsymbol{x}_k)$, and W is a diagonal matrix proportional to the reciprocal variances of the errors. By stacking the parameter vectors and augmenting the penalty matrix with zero blocks this problem can be written in the same form as a general ridge regression. One important detail of this correspondence is that when a basis function is defined for each unique $\boldsymbol{x}$ (e.g $n = N$), δ must satisfy the linear constraints $T^T\delta = 0$. The discussion at the end of this section explains how this constraint is enforced by reparametrizing to a reduced model.

Note that the penalty or ridge term does not apply to the parameters of the polynomial part of the model and this division corresponds to the fixed part of the spatial model or the null space for a thin plate spline. Usually Ω is constructed so that the quadratic form $\delta^T\Omega\delta$ measures the roughness of the function. However, in connection with kriging estimators, Ω is the covariance matrix of the random surface.

6.1 Details of tps and krig

Given the ridge regression structure of the estimates, it is sufficient to describe the particular basis functions associated with each estimate and how the independent variables are scaled. Let $\{\boldsymbol{x}_k^*\}$ for $1 \leq k \leq N$ denote a set of *knots*. In most cases the knot sequence will be equal to the unique set of $\boldsymbol{x}$ values but the only real restriction is that $N \leq n$. Here we are using the knot terminology loosely since these are really join points of the function only in the case of thin-plate splines in one dimension. For higher dimensions the knots specify the location of the basis functions.

For **tps** let $\psi_i(\boldsymbol{x}) = E_{md}(\boldsymbol{x} - \boldsymbol{x}_i^*)$ where E_{md} are the radial basis functions

$$\begin{array}{rll} E_{md}(\boldsymbol{r}) & = a_{md} \parallel \boldsymbol{r} \parallel^{(2m-d)} log(\parallel \boldsymbol{r} \parallel) & d \text{ even} \\ & = a_{md} \parallel \boldsymbol{r} \parallel^{(2m-d)} & d \text{ odd} \end{array}$$

and a_{md} is a constant.

The inclusion of the log term may seem mysterious but is necessary for even dimensions. Without this nonlinear term, the basis functions would be the norm raised to an even power and thus would add together to give a single low order polynomial. Before performing the thin-plate spline calculations, the default is to scale each dependent variable to have unit standard deviation. This operation assumes that on a standard scale the function has approximately the same smoothness in each dimension. Other scaling options are also supported. Although a more intensive approach is to estimate a separate smoothing scale for each variable by GCV (Gu 1991), this extension has not been implemented in FUNFITS because of the additional computational burden.

[5] Actually, the computations do not require that these be polynomial functions.

The roughness penalty matrix for thin plate splines reduces to the simple form:

$$H_{k,j} = \psi_k(\boldsymbol{x}_j^*) = E_{md}(\boldsymbol{x}_j^* - \boldsymbol{x}_k^*)$$

and a derivation may be found in Wahba (1990). In approaching this theory the reader should keep in mind that although a Hilbert Space development simplifies the notation and is elegant, the basic tool to justify this formula is a multidimensional integration by parts formula. One point of confusion in regard to this form of the roughness penalty is that it is only correct if some specific linear constraints are placed on δ. However these constraints arise naturally in restricting the number of parameters when one takes a knot at all unique values of $\boldsymbol{x}_k$. In one dimension these constraints are the natural spline boundary conditions whereby the estimate is forced to be linear beyond the boundary knots.

For the `krig` function let $\psi_k(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}_k^*)$. Because the covariance function should include the natural scaling of the variables as part of the range parameters, the default for `krig` is *not* to do any scaling of the variables. The roughness penalty matrix for the spatial process model is

$$H_{j,k} = \psi_k(\boldsymbol{x}_j^*) = k(\boldsymbol{x}_j^*, \boldsymbol{x}_k^*)$$

The estimates for λ , σ^2 and ρ can be computed using the general form of the ridge regression problem and by default these are found using GCV and sums of squares of the residuals and predicted values. An alternative approach is to estimate these parameters by restricted maximum likelihood under the assumption of a Gaussian field (O'Connell and Wolfinger, 1996).

6.2 Computation of the ridge regression estimator

This section discusses the details of computing the ridge estimates and concludes with a discussion of the merits of this computational approach over previous work.

Recall the generic quadratic minimization problem for both the spline and spatial process estimates in equation 4. Taking partial derivatives with respect to δ and β and simplifying gives the following system of equations that characterize the minimizer.

$$\begin{aligned} -M^T W(Y - T\beta - M\delta) + \lambda\Omega\delta &= 0 \\ -T^T W(Y - T\beta - M\delta) &= 0 \end{aligned} \tag{5}$$

In the case that $M = \Omega$ and has full rank, with some substitutions these may be simplified to

$$\begin{aligned} WT\beta + (\lambda I + W\Omega)\delta &= WY \\ T^T\delta &= 0 \end{aligned} \tag{6}$$

This latter case is the usual form for the thin plate spline and kriging estimators and the standard way of solving this special case is discussed in Bates *et al.* (1987). The Bates *et al.* (1987) solution is implemented in the FUNFITS function `tpsreg` but the simplification is not used for the general case when M is not equal to Ω. To enforce the constraint on δ implied by the second equation, let $T = F_1 R$ denote the QR decomposition of T where $F = [F_1 \mid F_2]$ is an orthogonal matrix, F_1 has columns that span the column space of T and R is upper triangular. Reparametrizing by $\delta = F_2\omega_2$ for $\omega_2 \in \Re^{n-t}$ now enforces the necessary number of linear constraints on δ and makes the parameters identifiable.

We now write the general minimization problem in ridge regression form. Let, $\beta \equiv \omega_1$ and $\omega^T = (\omega_1, \omega_2)^T$. The following linear system of n equations now describes the minimization:

$$X^T W X \omega + \lambda H \omega = X^T W Y$$

where

$$H = \begin{pmatrix} 0 & 0 \\ 0 & F_2^T \Omega F_2 \end{pmatrix}$$

and $X = [T \mid MF_2]$ Clearly, once ω is found this can be translated back into the original (β,δ) parameterization.

The computational approach for solving this system is based on the need to find ω over many values of λ. Also there is an interest keeping the numerical strategy simple so the source code is accessible to others. Let B denote the inverse symmetric square root of $X^T W X$. Currently B is constructed from the singular value decomposition of $X^T W X$ but a Cholesky square root is equally reasonable. Now let UDU^T be the singular value decomposition of BHB and set $G = BU$. With these decompositions and notation the system of ridge regression equations is

$$\begin{aligned} B^{-1}(I + \lambda BHB)B^{-1}\omega &= X^T W Y \\ B^{-1}(I + \lambda UDU^T)B^{-1}\omega &= X^T W Y \\ B^{-1}U(I + \lambda D)U^T B^{-1}\omega &= X^T W Y \end{aligned}$$

giving the solution

$$\omega = G(I + \lambda D)^{-1} G^T X^T W Y \tag{7}$$

Note that the matrix inverse is conveniently evaluated because it is diagonal.

This numerical approach is closely related to constructing a Demmler-Reinsch basis for the smoothing problem. This basis is denoted by $\{g_\nu\}$ and is defined by the three following properties.

1. $\{g_\nu\}$ for $1 \leq \nu \leq N$ spans the same subspace as $\{\phi_j\}, \{\psi_i\}$
2. $\sum_k g_\mu(\boldsymbol{x}_k) W_k g_\nu(\boldsymbol{x}_k) = \delta_{\mu,\nu}$ where $\delta_{\mu,\nu}$ is Kronecker's delta function.
3. Let
$$J_m(f_1, f_2) = \int_{\Re^d} \sum \frac{m!}{\alpha_1! \ldots \alpha_d!} \left(\frac{\partial^m f_1}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}} \right) \left(\frac{\partial^m f_2}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}} \right) d\boldsymbol{x}$$
then $J_m(g_\mu, g_\nu) = D_\nu \delta_{\mu,\nu}$ for $0 \leq D_1 \leq \ldots \leq D_N$.

Technically this basis is orthogonal with respect to the inner products derived from weighted residual sums of squares and the roughness penalty. Let

$$f(\boldsymbol{x}) = \sum_{\nu=1}^{N} \alpha_\nu g_\nu(\boldsymbol{x})$$

and assume that $N = n$ (there are as many knots as unique $\boldsymbol{x}$ values). Then

$$\sum_{k=1}^{n} (Y_k - f(\boldsymbol{x}_k))^2 W_k = \sum_{\nu=1}^{n} (u_\nu - \alpha_\nu)^2$$

and

$$J_m(f) = \sum_{\nu=1}^{n} D_\nu \alpha_\nu^2$$

where $u_\nu = \sum_{k=1}^{n} g_\nu(\boldsymbol{x}_k) W_k Y_k$. Based on these relationships it is easy to show that the spline solution has the simple representation:

$$\hat{f}(\boldsymbol{x}) = \sum_{\nu=1}^{N} \frac{u_\nu}{1+\lambda D_\nu} g_\nu(\boldsymbol{x})$$

Moreover, assuming the basic observational model, the random vector $\boldsymbol{u}$ has an identity covariance matrix. Although the Demmler-Reinsch basis is not explicitly used in the **tps** function is it easy to recover and the FUNFITS function **make.drb** evaluates this basis using the output object from **tps**. In terms of the matrix expressions, $g_\nu(\boldsymbol{x}_k) = (XG)_{k,\nu}$.

Based on the decompositions outlined above, there are also simple forms for $A(\lambda)$, the trace of this matrix and the GCV function

$$A(\lambda) = XG(I+\lambda D)^{-1} G^T X^T W$$

Straight forward algebra verifies that $G^T X^T W X G = U^T B X^T W X B U = U^T U = I$ and thus

$$tr(A(\lambda)) = tr(XG(I+\lambda D)^{-1} G^T X^T W) = tr((I+\lambda D)^{-1}) = \sum_{k=1}^{n} \frac{1}{1+\lambda D_k}$$

There is also a simple formula for the weighted residual sums of squares. Add and subtract $A(0) = XGG^T X^T W$ from $A(\lambda)$ and one can derive

$$\begin{aligned} & Y^T(I-A(\lambda))W(I-A(\lambda))Y \\ &= Y^T(I-A(0))^T W(I-A(0))Y + Y^T(A(0)-A(\lambda))^T W(A(0)-A(\lambda))Y) \\ &= SS_{pure} + ||\lambda(I+\lambda D)^{-1}\boldsymbol{u}||^2 \\ &= SS_{pure} + \sum_{k=1}^{N} \left(\frac{\lambda \boldsymbol{u}_k}{1+\lambda D_k}\right)^2 \end{aligned}$$

where $SS_{pure} = Y^T(I-A(0))W(I-A(0))Y$ and $\boldsymbol{u} = G^T X^T W Y$.

Thus we see that the residual sums of squares can be evaluated in order N operations for different values of λ provided that the decompositions $\boldsymbol{u}$ and SS_{pure} have been found.

The residual sums of squares attributable to pure error, SS_{pure}, is the result of the weighted regression on the full basis without any smoothing. If $N \geq n$ and there are no replicated points then this is zero. However, in the presence of replicated observations this will be the usual sums squares based on (weighted) differences between group means and the replicated points.

The simple form of the solution in (7) is appealing but comes with some numerical disadvantages. The most serious is that $X^T W X$ may not have full numerical rank and so constructing the inverse square root is problematic. Our experience is that a large condition number usually occurs for one-dimensional problems (try **auto.paint**) or when the data points have similar dependent values ($||\boldsymbol{x}_i - \boldsymbol{x}_j||$ is small). A natural solution to this problem is to reduce the number of basis functions to a set that has full rank. This is the strategy

used in FUNFITS although the problem is that once this is done one can longer interpolate the data or fit models that have a very fine resolution. For most applications involving some measurement error this is not an issue.

Another draw back is that two intensive matrix decompositions are done instead of one. The first is computing the inverse square root of $X^T W X$ and the second is a singular value decomposition of a matrix related to to the roughness penalty. In the case when $M = \Omega$ one can avoid the first decomposition using the thin-plate spline algorithms of Bates et al. (1987) as implemented in the FUNFITS function **tpsreg**. In fact, even the second decomposition can avoided by working with tridiagonal systems (Gu 1991). In either case, one must work with a complete set of basis functions and so even with this efficiency the computation of a thin plate spline is practically limited to several hundred observations (not counting replications). By contrast, the FUNFITS **tps** algorithm provides approximate estimates for much larger sample sizes and is particularly efficient when one expects a moderate random component in the model. This is the case when the number of knots is much smaller than the number of observations.

7 Fitting single hidden layer neural networks

The form of the feedfoward single layer neural network is

$$f(\boldsymbol{x}) = \beta_0 + \sum_{k=1}^{M} \beta_k \phi(\mu_k + \gamma_k^T \boldsymbol{x})$$

where $\phi(u) = (e^u)/(1+e^u)$ is the logistic distribution function. The parameter vectors are β, μ_k and γ_k , $1 \leq k \leq M$ giving a total number of parameters equal to $1+M+M+dM = 1+M(d+2)$. These parameters are estimated by nonlinear least squares and the complexity of the model, i.e. the number of hidden units M, is chosen based on generalized cross-validation: $GCV(p) = RSS(p)/(1-Cp/n)^2$. The usual procedure is to fit a range of hidden units, for example one to five, and choose the model with the minimum value of the GCV function. This is the default model as saved in the **nnreg** object. We prefer a cost value, $C = 2$, to guard against the over fitting that is inherent with neural network representations.

7.1 Details on fitting the model

The parameter space has a very large number of local minima when nonlinear least squares is used to estimate the neural net parameters. Thus, a conservative computational strategy is to generate many parameter sets at random and use these as starting values for the minimization procedure. To make this search procedure clear, we refer to the default numbers and argument names from the **nnreg** function. Minimization of the residual sums of squares is done *independently* for each value of M. A rectangular region of *feasible* parameter values (with respect to the properties of the logistic function) is divided into a series of 250 (**ngrind**) nested boxes about the origin. For each box, 100 (**ntry**) parameter sets are generated at random from a uniform distribution. The parameter set with the lowest RMSE is used as the initial point in the iterative minimization of the RMSE and we refer to the results of these 250 (**ngrind**) trial fits as *grinds*. The best 20 (**npolish**) of these grinds, based on RMSE, are then used for a more refined minimization with a much smaller convergence tolerance. The resulting estimates are referred to as the *polished* values. Finally, the

polished parameter set with the smallest RMSE is taken to be the least squares solution. The **nnreg** function also returns the other polished models for examination.

The care in searching the parameter space reflects our experience with the pathologies of the neural network sum of squares surface and is necessary for reproducible results. Because this fitting procedure may be computationally intensive, the **nnreg** function does most of its computation outside of S-PLUS. The **nnreg** function sets up some input files, executes a stand alone FORTRAN program in the UNIX shell and then reads in an output file when the program is done. This structure has the advantage that the fitting procedure can be run in the background independently of an S-PLUS session and the **nnreg** function has switches to set up the input file or read in the output file independently of executing the program. This may seem like a throwback to FORTRAN programming days, but for this application it seems appropriate. Along these lines the print function for the nnreg object lists the summary output file from the FORTRAN program.

Another point of possible confusion is the form of the logistic function (the squasher function) used by **nnreg**. The exponential form has been approximated by a rational polynomial. This approximate form (see **nnreg.squasher**) does not change the approximation properties of the neural network and is much faster to evaluate.

7.2 Confidence sets for the model parameters

The construction of confidence intervals for the neural net regression estimates is based on the identification of a joint confidence region for the model parameters using the likelihood ratio statistic. To simplify notation, it is helpful to stack the parameters in a single vector, θ. In this form, the approximate confidence set is all the values of θ such that

$$S(\theta) \leq S(\hat{\theta}) \left[1 + \frac{p}{n-p} \mathrm{F}(p, n-p, \alpha)\right] \tag{8}$$

where $S(\theta)$ is the residual sum of squares, $\hat{\theta}$ is the least-squares estimate of θ. For the F distribution, p is the total number of parameters in the model, n is the sample size and α is the probability level.

By default, the **nnregCI** function computes a set of 500 parameter vectors which satisfy the inequality of (8) with $\alpha = .05$ (95% level of confidence). Some attempt is made to choose these representative vectors to fill out the confidence region for θ and the confidence set for any function of the parameters can be approximated using these representatives. Let $\varphi(\theta)$ be some functional of the neural network model that is of interest. For example, $\varphi(\theta)$ could be the predictions of the mean function at specific x values. A more complicated functional is to consider the Lyapunov exponent when the neural network has been used to fit an autoregressive time series model. The confidence set (or interval) is approximated by evaluating φ at all the representative parameter vectors found by **nnregCI**. When φ is univariate, a reasonable simplification of the confidence set is to report the interval given by the minimum and maximum of these values. An example of the **nnregCI** function will be included in the next section as an application of confidence intervals for Lyanpuov exponents.

8 Nonlinear Autoregressive Models

The function fitting methods in FUNFITS can be used to estimate models of the form

$$Y_t = f(Y_{t-1}, \ldots, Y_{t-d-1}, S_t) + e_t$$

where f is a smooth function, $\{S_t\}$ a sequence of covariates and $\{e_t\}$ are mean zero independent random variables. The obvious approach is to consider this time series model as a regression problem. Once a suitable "X" matrix has been created involving the lagged Y series and the covariates, this can be passed to the regression functions. The function `nlar` (and object class) is a convenient function that accomplishes these manipulations. To make these operations explicit, an example for fitting a model to a times series from the Rossler system (`rossler`) follows. The user should keep in mind that this example is given for exposition of the methods. The `nlar` function while more of a "black box" gives the same analysis with fewer steps. For the Rossler time series, the present value is modeled as a function of the past three lags and f is estimated using a neural network model with 4 hidden units. The plots from this example are given in Figure 7.

```
> rossler.xy <- make.lags(rossler,c(1,2,3))
> rossler.nnreg <- nnreg(rossler.xy$x, rossler.xy$y, 1, 4) # this takes awhile
> plot(rossler.nnreg)
> acf(rossler.nnreg$residuals) # plot autocorrelation function of the
                               # residuals
```

8.1 Lyapunov exponents

A typical analysis of a nonlinear system includes examination of the local and global Lyapunov exponents (Bailey et al. 1996). Based on a first order Taylor series, these quantities measure the effect of a small perturbation on the state of the system a specific number of time steps into the future. The FUNFITS function `lle` calculates these statistics from the partial derivatives of the estimated function,f. For the Rossler example:

```
> rossler.jac <- predict(rossler.nnreg, dervative=1)
> rossler.lle <- lle(rossler.jac)
> summary(rossler.lle)



estimated global exponent 0.03516533
summary of QR estimate
           N     mean Std.Dev.       Q1   median       Q3
 5 steps 393 0.164280 0.073891 0.138140 0.157240 0.176960
10 steps 388 0.121440 0.045294 0.091214 0.127710 0.144920
20 steps 378 0.075239 0.025980 0.056031 0.074700 0.093949
40 steps 358 0.055550 0.020770 0.040733 0.053489 0.069473
80 steps 318 0.046032 0.013269 0.036481 0.045988 0.057106

> plot(rossler.lle)
```

The summary plot of the Lyapunov exponents is given in Figure 8. A time series is defined to be chaotic if the global Lyapunov exponent is positive. For the Rossler series the estimated global exponent is positive (0.035) indicating a choatic system. However, a

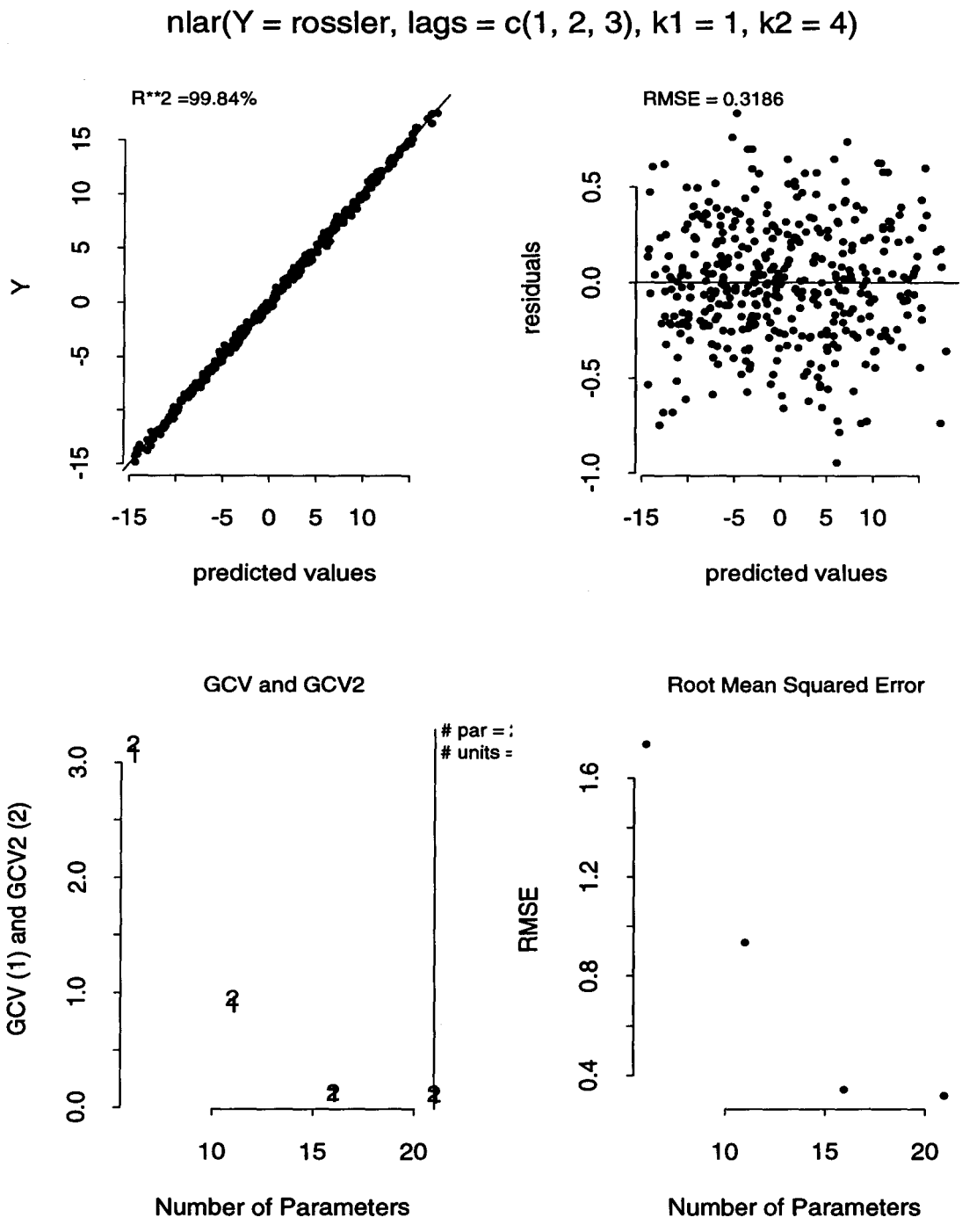


Figure 7: Diagnostic plots for fitting a neural network regression model to a lagged times from the Rossler system.

range of significantly positive and negative local exponents are found suggesting that in some regions of the state space the system is more sensitive to small perturbations and in other regions the system is more stable.

Finally this analysis is repeated using the **nlar** function:

```
> rossler.nlar <- nlar(rossler, lags=c(1,2,3), method="nnreg", k1=1, k2=4)
                      # this takes a while
> plot(rossler.nlar)
> rossler2.lle <-  lle(rossler.nlar)
```

Note that the main job of **nlar** is to keep track of the time series structure. Also the **lle** function has been overloaded to use the information directly from the returned **nlar** object.

8.2 Confidence regions

The FUNFITS function **nnregCI** will be used to calculate a 95% confidence interval for the global Lyapunov exponent of the Rossler example. To construct a confidence interval for the global Lyapunov exponent, it is necessary to evaluate the global exponent for all 500 neural net modelsretruend by **nnregCI**. The range of these values is taken to be an approximate confidence interval for the global exponent estimate.

```
> rossler.CIfit <- nnregCI(rossler.nnreg) # 500 representative  fits
                                    # based on the best 4 hidden unit fit
                                    # this takes awhile!
> hold<- rep(NA,500)
> for (i in 1:500){
        # loop through all the representative parameter vectors

        jac <- predict(rossler.CIfit$model[[i]],rossler.CIfit$x,derivative=1)
        hold[i] <- lle(jac,nprod=NA)$glb # calculate global LE
                                         # but not the locals
        }
>  rossler.le.CI <- range(hold)
```

The estimated global exponent is 0.035 with a 95% CI (0.017,0.036).

8.3 State vector form

Often a system is described in terms of a state vector and a map that updates this vector from the present time step into the next.

$$W_t = F(W_{t-1}) + E_t$$

Here $W_t \in \Re^d$ uniquely describes the state at time t, $F : \Re^d \rightarrow \Re^d$ and the stochastic terms, E_t are assumed to be uncorrelated random vectors. This section ends by briefly explaining how FUNFITS can be adapted to fit nonlinear state space models.

As an example, the usual state vector form for the Rossler system is in terms of a three dimension vector: $W_t = (X_t, Y_t, Z_t)^T$. The data set **rossler.state** gives a realization of this system in this form. Although FUNFITS has not been extended to fit this model

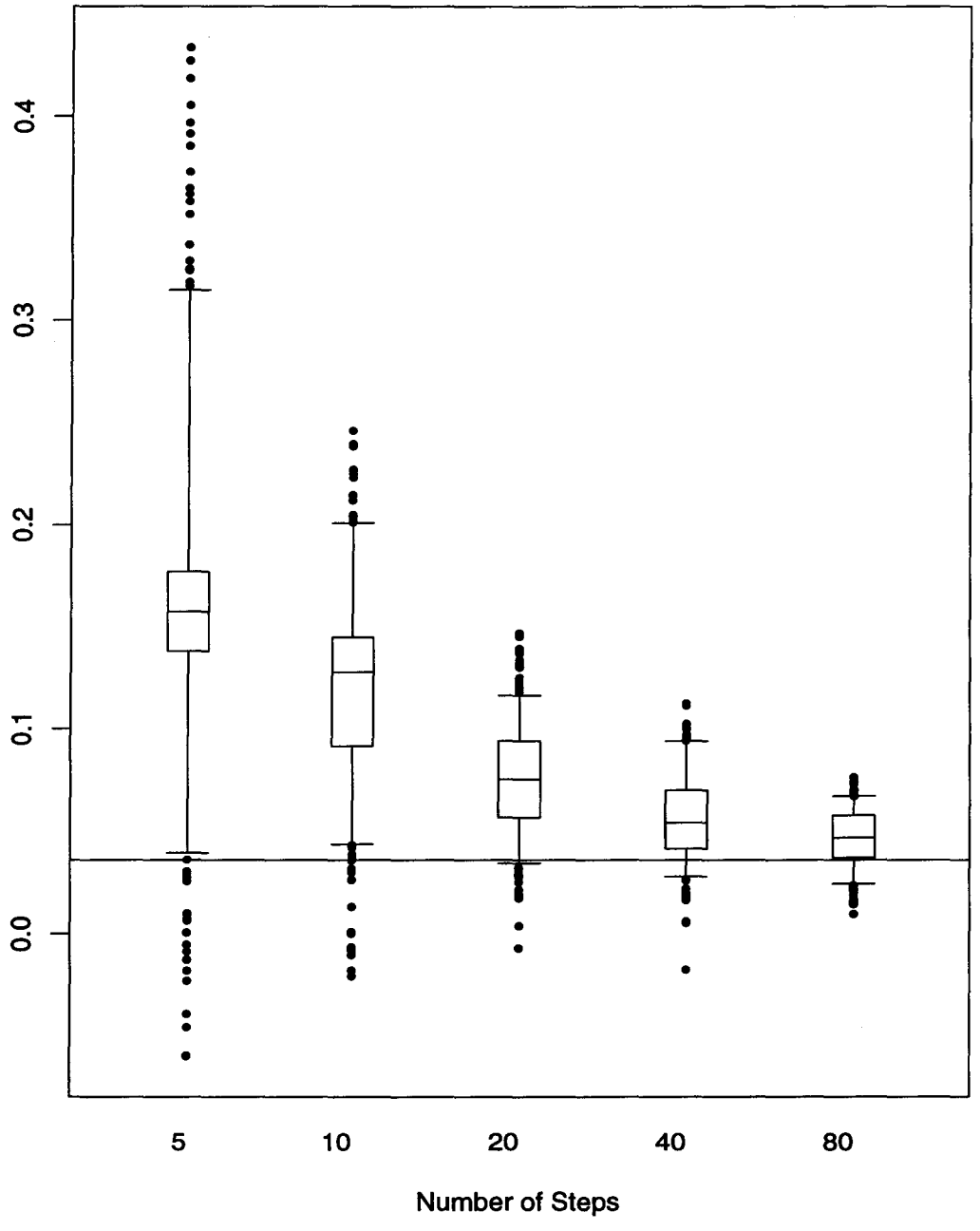


Figure 8: Summary statistics for estimated Local Lyapunov exponents for the Rossler data set.

directly it is not difficult to fit models for each component of F separately and combine these to estimate local Lyapunov exponents. First one needs to create the lagged set of values and fit separate (one dimensional models) for each component of W.

```
> out<- make.lags(rossler.state, nlags=c(1))
> fit.ross.X<- nnreg( out$x, out$y[,1], k1=4,k2=4)
> fit.ross.Y<- nnreg( out$x, out$y[,2], k1=4, k2=4)
> fit.ross.Z<-nnreg( out$x, out$y[,3], k1=4, k2=4)
```

Here the regression methods are neural networks fixed at 4 hidden units. However, any fitting method can be used here and the methods may be different among the different components. To find the local exponents one needs to create the Jacobian matrices for F evaluated at each state vector and pass them in the correct format to `lle`. In this step the important point is to string each Jacobian matrix row by row into a single long row at each time point. The resulting matrix is then the object passed to `lle`.

```
> tempx<- predict( fit.ross.X, derivative=1)
> tempy<- predict( fit.ross.Y, derivative=1)
> tempz<- predict( fit.ross.Z, derivative=1)
> ross.jacobian<- cbind( tempx, tempy, tempz)
> ross.lle.state<- lle( ross.jacobian, statevector=T)
```

In this example `ross.jacobian` will have 9 columns and the Jacobian matrix at say the $t = 25$ could be constructed by

```
matrix(ross.jacobian[25,], ncol=3,byrow=T)
```

Those familiar with state space models will also realize that the autoregressive model based on one time series also has a (trivial) state vector form. For the Rossler example presented in the beginning of this section, let $W_t = (Y_t, Y_{t-1}, Y_{t-2})^T$ and

$$F(W_{t+1}) = \begin{pmatrix} f(Y_t, Y_{t-1}, Y_{t-2}) \\ Y_t \\ Y_{t-1} \end{pmatrix}$$

Given this form, one could construct the Jacobian elements explicitly and call `lle` with the `statevector` format.

```
> jac<-predict( rossler.nnreg, derivative=1)
> test<- matrix( 0, ncol=9,nrow=nrow(jac)) # fill a matrix with zeroes
> test[,1:3]<-ross.jac
> test[,4]<- 1 > test[,8]<-1 > # the rest of the elements are zero!
> ross.lle<- lle( test)
```

However, in terms of computation, the autoregressive form is slightly more efficient for computation.

9 Model components and graphics options

The **nnreg**, **krig**, and **tps** functions produce fitted objects having a component **best.model**. For the **nnreg** function, this is the number of the model with the minimum GCV function value for cost=2. The best.model component for **tps** is the value of lambda used in the fit, and, for **krig** the best.model component is a vector containing the value of lambda, the estimated variance of the measurement error and the scale factor for the covariance used in the fit. The plot functions for **nnreg**, **krig**, and **tps** by default will use the best.model component for the summary plot.

The FUNFITS plotting functions have some intelligence in controlling the layout of the graphs. If the plotting window has already been divided up into a panel then the FUNFITS functions will use this layout when it produces multiple plots. To signal that the function should not alter the plotting panel parameters one uses the logical argument **graphics.reset**. For example the following examples will produce summary plots from two tps fits on a single page.

```
> ozone.tps1 <- tps(ozone$x,ozone$y) # thin plate spline fit to data
> ozone.tps2 <- tps(ozone$x,ozone$y,lambda=.1) # tps fit with lambda=.1
> set.panel(3,2) # sets plotting array
> plot(ozone.tps1,graphics.reset=F,main='tps fits')
# summary plot of first fit
> plot(ozone.tps2,graphics.reset=F,main=' ')
# summary plot of second fit
```

Besides changing the number of plots per panel, the graphics functions may also make other modifications to the default graphical parameters. For example the type of plotting region or the size of labels might be changed to produce a special plot. After the function is finished plotting there is the option of restoring the graphical parameters to their original values or leaving them in their new states. This decision is also controlled by **graphics.reset** argument and has the default value of true, resetting the parameters to their old values. Although this is usually what one wants, it does not make it easy to add additional features to a graph produced by a FUNFITS function. Here is an example using **graphics.reset=F** to add the ozone station locations onto a contour plot of the fitted surface.

```
> surface(ozone.tps1,graphics.reset=F)
> points(ozone.tps1$x)
```

10 Partial listing of FUNFITS functions and datasets

In this listing generic functions have an asterisk and are followed in italics by some of the more important classes in FUNFITS that are supported.

10.1 Surface Fitting and Design

addreg	Additive model fit using cubic smoothing splines
krig	Spatial process estimation
nnreg, nnregCI	Neural network regression
rtps	Robust thin plate spline regression
tps	Thin plate spline regression

10.2 Response Surface Methods and Spatial Design

cover.design	Computes a space-filling coverage design
leaps.design	Subset designs based on leaps algorithm
optim * (*tps, krig, nnreg*)	Finds the optimum of a fitted surface
path * (*tps, krig, nnreg*)	Finds the path of steepest ascent of a fitted surface
spread.design	Computes spread designs
surface * (*surface, tps, krig, nnreg*)	Graphs fitted surface
ushell.design	Computes uniform shell designs

10.3 Other Regression Methods and Statistics

nkreg	Normal kernel regression
nkden	Normal kernel density estimate
nkden.cv	Least squares CV function for normal kernel density estimate
tpsreg	Thin-plate spline calculated using GCVPACK subroutines
qsreg	Quantile spline regression
splint	Spline interpolation
sreg	Cubic spline smoothing
vgram	Variogram

10.4 Nonlinear Time Series

lle (*nlar, nnreg*)	Computes local Lyapunov exponents
make.lags	Creates a matrix of lagged values
make.lle	Calculates Lyapunov exponents from Jacobians
nlar	Fits a nonlinear autoregressive model

10.5 Plotting Functions and Graphics

.motif.options	Options for plot.window function
as.surface	Creates a surface object from a grid
bplot	Creates boxplots allowing for variable positioning
make.surface.grid	Creates X-Y values to evaluate a grid
persp.init, persp.X	Family of primitives to add to perspective plots
plot *	Gives diagnostic plots or summary of fit
plot.surface	Perspective and/or contour plots of a surface object
plot.window	Portrait plot window with aspect ratio of $8\frac{1}{2}$ by 11 paper
set.panel	Divides plot window into a panel of plots
xline	Draws vertical line
yline	Draws horizontal line

10.6 Statistics, Summaries and Printing

COR	Correlations adjusting for missing values
describe	Computes several summary statistics
fast.1way	Fast One-way ANOVA decomposition
gcv * (*tps, krig*)	GCV function for smoothing parameter
predict.se * (*tps, krig*)	Standard error of predicted values
predict.surface * (*tps, krig, nnreg*)	Evaluates predicted values and creates a surface object
predict.surface.se * (*tps, krig*)	Evaluates standard errors of fit on a grid
print *	Informative listing of an object
print.text	Prints text like the UNIX cat function
stats	Summary statistics of an object
summary.AOV	ANOVA table with type III sums of squares
summary *	Summary of fit

10.7 Supporting Functions for Methods

cover.criterion	Cover design criterion
EOF.cov	Covariance function based on orthogonal series
exp.isocor	Covariance with isotropic correlations
exp.cov	Simple version of the exponential covariance
EXP.cov	A more efficient version of **exp.cov**
GaSP.cov	GaSP covariance function
make.Amatrix * (*tps, krig*)	Calculates matrix relating observations to predicted values
make.rb	Calculates radial basis functions
make.drb	Evaluates Demmler-Reinsch basis
make.tmatrix	Creates matrix of all monomials to given degree

10.8 Numerical functions

bisection.search	Bisection search for zero
find.upcross	finds upcrossing of monotonic function
golden.section.search	Minimization using Golden section search
rdist	Euclidean distances between points
rdist.earth	Great circle distance between points

10.9 Data and Object Manipulation

cat.matrix	Finds replicates rows of matrix
cat.to.list	Converts a vector with categories to a list
dup	Identifies duplicate values
dup.matrix	Identifies duplicate rows in a matrix
rowmatch	Finds the row matches rows for two matrices
scaleA	Scale estimate used for plug in bandwidths
sort.matrix	Sorts the columns or rows of a matrix efficiently
transformx	Linear transformations for columns of a matrix
unique.matrix	Tests for unique rows of a matrix

10.10 Help, FUNFITS and System

dump.data.list	Names of FUNFITS datasets
dump.fun.list	Names of FUNFITS functions
dynload.HP	Version of dynload for HP workstations
emacs	Calls emacs editor to edit S object
extra	Lists objects in directory that are part of FUNFITS
format.help	Writes formatted help entry to a file
help.to.latex	Coverts help entry to input for LaTeX
make.dump.files, make.ftp	Dumps FUNFITS objects to .q files
make.qfiles	Creates source files of all FUNFITS functions
setup.HP	Sets up dynload function for a HP workstation
remove.file, touch.file	System file functions

10.11 Data Sets

BD	DNA amplification experiment
BD2	Results from a sequence of RSM designs in another DNA amplification experiment
actuator	Positioning jets designed experiment
actuator.calibrate	Actuator calibration data set
auto.paint	Auto paint thicknesses
climate	Climate data for 50 US Cities
flame	Ionization of a lithium solution
minitri	Triathlon Results Cary, NC
ozone	Average Chicago ozone for summer 1987 at 20 monitoring stations
ozone2	Daily ozone values for sites in the Midwest summer 1987
rossler	Rossler time series
rossler.state	Three state variables for Rossler system

11 Listing of the README installation file

```
FUNFITS is a collection of programs based in S for curve and
function fitting.

The major methods implemented as S functions include:

nnreg   Neural net regression.
tps     Thin Plate spline regression
krig    Spatial process estimate (Kriging)
nlar    Nonlinear  autoregressive time series models


splint  1-d cubic splint interpolation
sreg    1-d cubic spline smoothing
qsreg   quantile spline regression
nkden   normal kernel density estimate
nkreg   normal kernel regression estimate
nlar    fit a nonlinear autoregressive time series
lle     compute the local Lyapunov exponent of an autoregressive times series

There are also generic functions that support these methods such as
plot, summary, print and  a graphical function surface. In addition a
manual in postscript is included to describe some of the key methods.


To get  the UNIX version:

      mkdir Funfits  # or wherever you want to keep the FUNFITS package
      cd Funfits
      ftp ftp.eos.ncsu.edu
          #    ( login as anonymous when prompted for a user name)
      cd pub/stat/stattools/pub/funfits
      get funfits.tar.Z
      quit

Now in UNIX in the FUNFITS directory

     uncompress funfits.tar.Z
     tar -xvf funfits.tar
     make all

The FUNFITS manual is in postscript form in the file manual.ps
There are also help files for many of the functions along with example
```

data sets.

To use the FUNFITS programs from another directory use the attach
function in your S session. That is in S
attach("FUNFITS pathname/.Data")
Here FUNFITS pathname is the full UNIX name for the FUNFITS directory.
For example, on the eos system the name is /ncsu/stattools/Funfits2.0

Attention HP users! You must type :

```
setup.HP()
```

at the beginning of each session to get the correct version of the
dynamic loading function.

SOME DETAILS

Removing files:

Once FUNFITS has been installed you can use

```
make clean
```

to delete all nonessential files excpet for the original tar file.

Setting the directory path:

In order to dynamically load the FORTRAN subroutines
the make file will try to set the right path for FUNFITS but you
should look at the S data set FUNFITS in the .Data directory to
make sure it is right.

If it is wrong ... to correct the directory path:

In S edit the character data set FUNFITS to be the full path name of the
Funfits directory. (The result of being in the Funfits directory and
in UNIX typing pwd .)

You will also need to change the character data set FUNFITS.BIN to the
be the full path name of the bin subdirectory under the FUNFITS home
directory. If this is not changed you will not be able to use the
FORTRAN supporting programs and subroutines.

Makefile:

The file Makefile indicates how the different pieces of FUNFITS
are installed. Most of the trouble is in compiling the FORTRAN.
Also note that the init part can only be run once. So if the make all
step produces an error after init you should start the make after
this point. The function tpsreg is not an essential part of FUNFITS
and some compile time and storage is gained by NOT making tpsreg
or gcvpack.

Repacking Funfits:

To reform FUNFITS as a single tar file in the FUNFITS directory
in UNIX
make.ftp
That should do it!

As part of this process the function make.dump.files() will only bundle
together the functions and data sets listed in the data sets dump.fun.list
and dump.data.list so if you want to include any other objects add the
names to one of these data sets.

DISCLAIMER:

This is software for statistical research and the authors do not
guarantee the correctness of any function or program in this package.

Doug Nychka
Department of Statistics
North Carolina State University
Raleigh, NC 27695-8203

nychka@stat.ncsu.edu

12 References

Bates, D.M., Lindstrom, M.J., Wahba, G. and Yandell, B.S. (1987). GCVPACK - Routines for generalized cross validation. *Comm. Stat. Sim. Comp.* 16, 263-297.

Bailey, B. , Ellner S. and Nychka D. (1996) "Asymptotics and Applications of Local Lyapunov Exponents Exponents" Proceedings for the Fields/CRM Workshop: Nonlinear Dynamics and Time Series: Building a Bridge Between the Natural and Statistical Sciences, American Mathematics Society (to appear).

Cressie, N.A.C. (1991). *Statistics for Spatial Data.* Wiley, New York, NY.

Green, P.J. and Silverman, B.W. (1994) *Nonparametric Regression and Generalized Linear Models.* Chapman and Hall, London, UK.

Gu, C. and Wahba, G. (1991) Minimizing GCV/GML scores with multiple smoothing parameters via the Newton method. *SIAM Journal of Scientific and Statistical Computing,* 12, 383-398.

Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models.* Chapman and Hall, NY.

O'Connell, M. and Wolfinger, R. (1996) Spatial Regression, Response Surfaces and process optimization. J. Comp. Graph. Stat. (in revision).

Sacks, J., Welch, W.J., Mitchell, T.J. and Wynn, H.P. (1989). Design and analysis of computer experiments. *Statistical Science* 4, 409-435.

Nychka, D. (1988). "Bayesian 'Confidence' Intervals for Smoothing Splines." *Journal of the American Statistical Association,* **83**, 1134-1143.

Wahba, G. (1990). *Spline Models for Observational Data.* Society for Industrial Applied Mathematics. Philadelphia PA.

Welch, W.J., Buck, R.J., Sacks, J., Wynn, H.P., Mitchell, T.J. and Morris, M.D. (1992). Screening, Predicting and Computer Experiments. *Technometrics* 34, 15-25.