DEPARTMENT OF STATISTICS
University of Wisconsin
1210 W. Dayton St.
Madison, Wisconsin 53706

TECHNICAL REPORT NO. 775 (rev.)

October, 1986

GCVPACK – ROUTINES FOR GENERALIZED CROSS VALIDATION

by

Douglas M. Bates
Mary J. Lindstrom
Grace Wahba
Brian S. Yandell

This research has been supported in part by National Science Foundation grants DMS-8404970 and ATM-8410373, United States Department of Agriculture CSRS grant 511-100, NASA grant NAG5-316 and ONR contract N00014-77-C-0675.

GCVPACK – ROUTINES FOR GENERALIZED CROSS VALIDATION

Douglas M. Bates
Mary J. Lindstrom
Grace Wahba
Brian S. Yandell

Department of Statistics
University of Wisconsin–Madison

Key Words and Phrases: ill-posed problems; partial thin plate smoothing splines; penalized likelihood; semi-parametric models; ridge regression; thin plate smoothing splines; truncated singular value decomposition.

Abstract

A set of Fortran-77 subroutines to provide building blocks for Generalized Cross Validation calculations is presented. We outline applications in ridge regression, thin-plate smoothing splines to approximate smooth multivariate functions observed with noise and a new technique of using partial spline models. Timing tests and the structure of the main driver routines are also presented.

Purpose and Description

Purpose

These Fortran-77 subroutines provide building blocks for Generalized Cross-Validation (GCV) (Craven and Wahba, 1979) calculations in data analysis and data smoothing including ridge regression (Golub, Heath, and Wahba, 1979), thin plate smoothing splines (Wahba and Wendelberger, 1980), deconvolution (Wahba, 1982d), smoothing of generalized linear models (O'Sullivan, Yandell and Raynor (1986), Green (1984) and Green and Yandell (1985)), and ill-posed problems (Nychka et al., 1984, O'Sullivan and Wahba, 1985). We present some of the types of problems for which GCV is a useful method of choosing a smoothing or regularization parameter and we describe the structure of the subroutines.

Ridge Regression: A familiar example of a smoothing parameter is the ridge parameter λ in the ridge regression problem which we write as

$$\min_{\gamma} \quad \frac{1}{n} \; \| \mathbf{y} - \mathbf{X}\gamma \|^2 + \lambda \gamma^T \gamma$$

where γ is a p-dimensional parameter vector, $\mathbf{y}$ is an n-dimensional response vector and $\mathbf{X}$ is an $n \times p$ design matrix.

For any positive λ, an optimal γ_λ can be easily calculated. Unfortunately, this leaves the question of which value of λ to use. Golub, Heath, and Wahba (1979) demonstrated that minimization of the GCV function $V(\lambda)$ is a powerful criterion for the choice of an optimal λ, where

$$V(\lambda) = \frac{(1/n) \; \| (\mathbf{I} - \mathbf{A}(\lambda))\mathbf{y} \|^2}{[(1/n)\, \mathrm{tr}(\mathbf{I} - \mathbf{A}(\lambda))]^2}$$

and $\mathbf{A}(\lambda)$ is the $n \times n$ "hat" matrix of the ridge regression

$$\mathbf{A}(\lambda) = \mathbf{X}(\mathbf{X}^T\mathbf{X} + n\lambda\mathbf{I})^{-1}\mathbf{X}^T \; .$$

At first glance, optimization of $V(\lambda)$ seems a formidable computational problem since each value of λ has its corresponding $\mathbf{A}(\lambda)$. However, Golub, Heath, and Wahba (1979) gave a method of expressing $V(\lambda)$ as an easily-calculated rational function based on the singular value decomposition (SVD) (Dongarra et al., 1979, chapter 10)

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

where $\mathbf{U}$ is $n \times p$ with orthonormal columns, $\mathbf{V}$ is $p \times p$ and orthogonal, and $\mathbf{D}$ is $p \times p$ and diagonal with diagonal elements

$$d_1 \geq d_2 \geq \cdots \geq d_p \geq 0$$

which are the nonnegative square roots of the eigenvalues of $\mathbf{X}^T\mathbf{X}$. The "hat" matrix can then be written as

$$\mathbf{A}(\lambda) = \mathbf{U}\mathbf{D}^2(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\mathbf{U}^{\mathrm{T}} ,$$

and using

$$\mathbf{z} = \mathbf{U}^{\mathrm{T}}\mathbf{y}$$

we can write

$$V(\lambda) = \frac{n\left[\|\mathbf{y}\|^2 - \|\mathbf{z}\|^2 + \sum_{j=1}^{p}\left[\frac{n\lambda}{d_j^2+n\lambda}\right]^2 z_j^2 \right]}{\left[n - p + \sum_{j=1}^{p}\frac{n\lambda}{d_j^2+n\lambda}\right]^2} . \qquad (1.1)$$

Once the SVD of $\mathbf{X}$ is computed, it is trivial to evaluate $V(\lambda)$ for a wide range of values of λ and determine the optimum value of λ. Equation (1.1) indicates that, for most problems, $d_p^2 \le n\hat{\lambda} \le d_1^2$. After an optimal λ is chosen, the corresponding γ_λ is calculated as

$$\gamma_\lambda = \mathbf{V}(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{z} . \qquad (1.2)$$

Multivariate data smoothing with thin-plate splines: A more important application of GCV is determining smooth representations of an underlying multivariate function from which noisy data is observed. The ridge regression problem serves as an introduction to the idea of GCV and the computational steps for efficient evaluation of the GCV function but data smoothing using thin-plate smoothing splines (TPSS) is a much more common application of GCV. These methods extend the computational methods derived in Wahba and Wendelberger (1980), Wendelberger (1981), and Wahba (1984a).

For convenience we first describe the calculations for a two-dimensional ''independent'' variable $\mathbf{x}$ but the software is designed for the general case. The data model for TPSS is

$$y_i = f(\mathbf{x}_i) + \varepsilon_i , \quad i = 1, \ldots, n ,$$

where the $(\mathbf{x}_i, y_i), i = 1,2,\ldots,n$, are observed data, f is an unknown function which is assumed to be reasonably smooth, and the $\varepsilon_i, i = 1,2,\ldots,n$, are independent, zero-mean random variables.

In general we will measure smoothness of f by the integral over the entire plane of the square of the partial derivatives of f of total order 2. That is,

$$J_2(f) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left\{\left[\frac{\partial^2 f}{\partial x_1^2}\right]^2 + 2\left[\frac{\partial^2 f}{\partial x_1 \partial x_2}\right]^2 + \left[\frac{\partial^2 f}{\partial x_2^2}\right]^2\right\} dx_1 dx_2 .$$

To allow generalizations, the software uses a smoothness penalty defined by the partial derivatives of total order m as

$$J_m(f) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\sum_{i=0}^{m}\binom{m}{i}\left[\frac{\partial^m f(x_1,x_2)}{\partial x_1^i \partial x_2^{m-i}}\right]^2 dx_1 dx_2 .$$

In d dimensions,

$$J_m(f) = \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\sum\frac{m!}{\alpha_1!\cdots\alpha_d!}\left[\frac{\partial^m f}{\partial x_1^{\alpha_1}\cdots\partial x_2^{\alpha_d}}\right]^2 dx_1\cdots dx_d$$

with the sum within the integral over $\sum\alpha_i = m$. In general, one must have $2m - d > 0$ with d the dimension of $\mathbf{x}$. Using this smoothness penalty, the TPSS estimate f_λ of f is the minimizer of

$$S_\lambda(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 + \lambda J_m(f) . \qquad (2.1)$$

From Duchon (1976), the minimizer f_λ of (2.1) can be represented as

$$f_\lambda(\mathbf{x}) = \sum_{i=1}^{t}\beta_i\phi_i(\mathbf{x}) + \sum_{i=1}^{n}\delta_i E_m(\mathbf{x}-\mathbf{x}_i) \qquad (2.2)$$

where

$$E_m(\mathbf{t}) = (-1)^m 2^{1-2m}\pi^{-1}((m-1)!)^{-2}\|\mathbf{t}\|^{(2m-2)}\ln(\|\mathbf{t}\|)$$

and t is the dimension of the space of polynomials on two variables of total order at most $m-1$,

$$t = \binom{m+1}{2} .$$

A basis for this space is

$$\phi_1(\mathbf{x}) = 1$$

$$\phi_2(\mathbf{x}) = x_1$$

$$\phi_3(\mathbf{x}) = x_2$$

$$\phi_4(\mathbf{x}) = x_1^2$$

$$\phi_5(\mathbf{x}) = x_1 x_2$$

$$\cdots$$

$$\phi_t(\mathbf{x}) = x_2^{m-1} .$$

The general definition of E_m, which depends on the dimension, d, of the independent variables $\mathbf{x}$ is

$$E_m(\mathbf{t}) = \begin{cases} a_{md}\,\|\mathbf{t}\|^{(2m-d)}\ln(\|\mathbf{t}\|) & , d \text{ even} \\ a_{md}\,\|\mathbf{t}\|^{(2m-d)} & , d \text{ odd} \end{cases} , \qquad (2.3)$$

with

$$a_{m\,d} = \begin{cases} (-1)^{1+m+d/2}\, 2^{1-2m}\, \pi^{-d/2} ((m-1)!\,(m-d/2)!)^{-1} & , d \text{ even} \\ \Gamma(\frac{d}{2}-m)\, 2^{-2m}\, \pi^{-d/2} ((m-1)!)^{-1} & , d \text{ odd} \end{cases} .$$

The dimension, t, of the polynomial space is given in general by

$$t = \begin{bmatrix} m+d-1 \\ d \end{bmatrix} .$$

A property of the TPSS representation is that both the function f_λ evaluated at the data points and the smoothing penalty J_m can be expressed using the $n \times n$ matrix $\mathbf{K}$ with entries

$$\{\mathbf{K}\}_{ij} = E_m(\mathbf{x}_i - \mathbf{x}_j) \ . \tag{2.4}$$

The function f_λ also requires the $n \times t$ matrix $\mathbf{T}$ with entries

$$\{\mathbf{T}\}_{ij} = \phi_j(\mathbf{x}_i) \ .$$

The matrix $\mathbf{T}$ having full column rank guarantees a unique minimizer of (2.1). Duchon (1976) showed that $\boldsymbol{\delta}$ in (2.2) must satisfy

$$\mathbf{T}^{\mathrm{T}}\boldsymbol{\delta} = \mathbf{0} \ . \tag{2.5}$$

Then $\boldsymbol{\beta}_\lambda$ and $\boldsymbol{\delta}_\lambda$ are the minimizers of (2.1), which can be written as

$$S_\lambda(\boldsymbol{\beta},\boldsymbol{\delta}) = \frac{1}{n} \| \mathbf{y} - \mathbf{T}\boldsymbol{\beta} - \mathbf{K}\boldsymbol{\delta} \|^2 + \lambda\, \boldsymbol{\delta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\delta} \ .$$

Note that the restriction in (2.5) is important, as $\mathbf{K}$ will generally have negative eigenvalues, but for any vectors $\boldsymbol{\delta}$ satisfying (2.5) it can be shown that

$$\boldsymbol{\delta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\delta} \geq 0 \ .$$

Our objective is to reduce the calculation of the parameters $\boldsymbol{\beta}_\lambda$ and $\boldsymbol{\delta}_\lambda$, the "hat" matrix $\mathbf{A}(\lambda)$, and the GCV function $V(\lambda)$ to a simplified form as was done for the ridge regression case.

No replicates case: When there are no replicates in the $\mathbf{x}$'s, we proceed by taking a QR decomposition (Dongarra et al., 1979, chapter 9) of $\mathbf{T}$ as

$$\mathbf{T} = \mathbf{F}\mathbf{G} = [\mathbf{F}_1 \vdots \mathbf{F}_2] \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{F}_1\mathbf{G}_1$$

where $\mathbf{F}$ is $n \times n$ and orthogonal while $\mathbf{G}$ is $n \times t$ and zero below the main diagonal. $\mathbf{F}_1$ is the first t columns of $\mathbf{F}$ and $\mathbf{F}_2$ is the trailing $n-t$ columns while $\mathbf{G}_1$ is the first t rows of $\mathbf{G}$. The columns of $\mathbf{F}_2$ provide a basis for the $\boldsymbol{\delta}$ which satisfy

$$\mathbf{T}^{\mathrm{T}}\boldsymbol{\delta} = \mathbf{0}$$

so we can set

$$\boldsymbol{\delta} = \mathbf{F}_2\boldsymbol{\zeta}$$

where $\boldsymbol{\zeta}$ has dimension $n-t$. Using

$$\mathbf{w}_1 = \mathbf{F}_1^{\mathrm{T}}\mathbf{y}$$

$$\mathbf{w}_2 = \mathbf{F}_2^{\mathrm{T}}\mathbf{y}$$

the objective function of the optimization becomes

$$\begin{aligned} S_\lambda(\boldsymbol{\beta},\boldsymbol{\delta}) &= \frac{1}{n} \| \mathbf{y} - \mathbf{T}\boldsymbol{\beta} - \mathbf{K}\boldsymbol{\delta} \|^2 + \lambda\, \boldsymbol{\delta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\delta} \\ &= \frac{1}{n} \| \mathbf{F}^{\mathrm{T}}(\mathbf{y} - \mathbf{T}\boldsymbol{\beta} - \mathbf{K}\boldsymbol{\delta}) \|^2 + \lambda\, \boldsymbol{\delta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\delta} \\ &= \frac{1}{n} \| \mathbf{w}_1 - \mathbf{G}_1\boldsymbol{\beta} - \mathbf{F}_1^{\mathrm{T}}\mathbf{K}\mathbf{F}_2\boldsymbol{\zeta} \|^2 \\ &\quad + \frac{1}{n} \| \mathbf{w}_2 - \mathbf{F}_2^{\mathrm{T}}\mathbf{K}\mathbf{F}_2\boldsymbol{\zeta} \|^2 + \lambda\, \boldsymbol{\zeta}^{\mathrm{T}}\mathbf{F}_2^{\mathrm{T}}\mathbf{K}\mathbf{F}_2\boldsymbol{\zeta} \ . \end{aligned} \tag{2.6}$$

Assuming $\mathbf{G}_1$ is non-singular (that is, the points $\mathbf{x}_i$, $i = 1, \ldots, n$, are adequately dispersed so that the columns of $\mathbf{T}$ are linearly independent) the first term in (2.6) can be made zero by solving

$$\begin{aligned} \mathbf{G}_1\boldsymbol{\beta}_\lambda &= \mathbf{w}_1 - \mathbf{F}_1^{\mathrm{T}}\mathbf{K}\mathbf{F}_2\boldsymbol{\zeta}_\lambda \\ &= \mathbf{w}_1 - \mathbf{F}_1^{\mathrm{T}}\mathbf{K}\boldsymbol{\delta}_\lambda \end{aligned}$$

for $\boldsymbol{\beta}_\lambda$. In practice we check the condition of $\mathbf{G}_1$ and return an error condition if it is computationally singular, indicating that the columns of $\mathbf{T}$ are strongly correlated. This condition is equivalent to the computational singularity of the problem of least squares regression of the data onto the span $\{\phi_j\}$. Singularity will rarely occur since the column dimension of $\mathbf{T}$ is small.

We can now reduce the problem to a form like ridge regression by using the fact that $\mathbf{F}_2^{\mathrm{T}}\mathbf{K}\mathbf{F}_2$ is positive definite to form the Cholesky decomposition (Dongarra et al., 1979, chapter 8)

$$\mathbf{F}_2^{\mathrm{T}}\mathbf{K}\mathbf{F}_2 = \mathbf{L}^{\mathrm{T}}\mathbf{L}$$

where $\mathbf{L}$ is $(n-t) \times (n-t)$ and upper triangular. In practice we use a pivoted Cholesky decomposition so we can check the conditioning of $\mathbf{F}_2^{\mathrm{T}}\mathbf{K}\mathbf{F}_2$. If this matrix is computationally singular, which can occur if $\| \mathbf{x}_i - \mathbf{x}_j \|$ is very small but non-zero for some $i \neq j$, we return an error condition. A near-singular $\mathbf{F}_2^{\mathrm{T}}\mathbf{K}\mathbf{F}_2$ is usually avoided since, in checking for replicates, we declare $\mathbf{x}_i$ and $\mathbf{x}_j$ to be replicates if the distance between them is very small. See Appendix 1 for more information on the detection of replicates and the computational singularity of $\mathbf{L}$.

After ensuring that $\mathbf{L}$ is non-singular, we define

$$\boldsymbol{\gamma} = \mathbf{L}\boldsymbol{\zeta}$$

and the last two terms of $S_\lambda(\boldsymbol{\beta},\boldsymbol{\delta})$ in (2.6) can be written as

$$\frac{1}{n} \; \| \mathbf{w}_2 - \mathbf{L}^{\mathrm{T}}\gamma \|^2 + \lambda \gamma^{\mathrm{T}}\gamma \; .$$

This has the same form as the ridge regression problem with solution

$$\gamma_\lambda = (\mathbf{L}\mathbf{L}^{\mathrm{T}} + n\lambda\mathbf{I})^{-1}\mathbf{L}\mathbf{w}_2 \; .$$

We take a SVD of $\mathbf{L}^{\mathrm{T}}$ as

$$\mathbf{L}^{\mathrm{T}} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$$

and write the estimate as

$$\gamma_\lambda = \mathbf{V}(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^{\mathrm{T}}\mathbf{w}_2$$

and the ''hat'' matrix as

$$\begin{aligned} \mathbf{A}(\lambda) &= \mathbf{F}_1\mathbf{F}_1^{\mathrm{T}} + \mathbf{F}_2\mathbf{U}\mathbf{D}^2(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{F}_2^{\mathrm{T}} \\ &= \mathbf{F}\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^2(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}^{\mathrm{T}} \end{bmatrix} \mathbf{F}^{\mathrm{T}} \; . \end{aligned} \tag{2.7}$$

As in the ridge regression case, we use

$$\mathbf{z} = \mathbf{U}^{\mathrm{T}}\mathbf{w}_2$$

to write

$$V(\lambda) = \frac{n\sum_{j=1}^{n-t}\left[\dfrac{n\lambda}{d_j^2 + n\lambda}\right]^2 z_j^2}{\left[\sum_{j=1}^{n-t}\dfrac{n\lambda}{d_j^2 + n\lambda}\right]^2} \; . \tag{2.8}$$

The actual calculation of the parameter δ_λ corresponding to the E_m's is performed as

$$\begin{aligned} \delta_\lambda &= \mathbf{F}_2\mathbf{U}(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\mathbf{z} \\ &= \mathbf{F}_2\mathbf{U}(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{F}_2^{\mathrm{T}}\mathbf{y} \; . \end{aligned} \tag{2.9}$$

Replicated x values: Replicates of $\mathbf{x}$ values introduce some minor complications since we must define only one δ_i corresponding to each unique $\mathbf{x}$ position. The best way to handle this is to pre-process the data by sorting the $\mathbf{x}$ values to determine the unique $\mathbf{x}$ values and the number of replicates of each value. Let k be the number of unique $\mathbf{x}$ values. We can express the objective function optimized by δ_λ and β_λ as

$$S_\lambda(\beta,\delta) = \frac{1}{n} \; \| \mathbf{y} - \mathbf{T}\beta - \mathbf{K}\delta \|^2 + \lambda\delta^{\mathrm{T}}\mathbf{K}_U\delta \tag{3.1}$$

subject to the condition

$$\mathbf{T}_U^{\mathrm{T}}\delta = \mathbf{0}$$

where $\mathbf{T}$ and $\mathbf{K}$ are of size $n \times t$ and $n \times k$ respectively, while $\mathbf{T}_U$ and $\mathbf{K}_U$ are of size $k \times t$ and $k \times k$ respectively. These matrices are related by

$$\mathbf{T} = \mathbf{M}\mathbf{T}_U$$

$$\mathbf{K} = \mathbf{M}\mathbf{K}_U$$

where $\mathbf{M}$ is an $n \times k$ indicator matrix (all its entries are ones or zeros and there is only a single one in each row) which, for each row, indicates the unique $\mathbf{x}$ that corresponds to that observation.

If we take a QR decomposition of $\mathbf{M}$ as

$$\mathbf{M} = \mathbf{B}\mathbf{C} = [\mathbf{B}_1 \colon \mathbf{B}_2]\begin{bmatrix} \mathbf{C}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{B}_1\mathbf{C}_1$$

and pre-multiply all the vectors in the first term of (3.1) by $\mathbf{B}$, (3.1) divides into

$$\begin{aligned} S_\lambda(\beta,\delta) = \frac{1}{n} \; \| \mathbf{B}_2^{\mathrm{T}}\mathbf{y} \|^2 + \frac{1}{n} \; \| \mathbf{B}_1^{\mathrm{T}}\mathbf{y} - \mathbf{C}_1\mathbf{T}_U\beta - \mathbf{C}_1\mathbf{K}_U\delta \|^2 \\ + \lambda\delta^{\mathrm{T}}\mathbf{K}_U\delta \; . \end{aligned} \tag{3.2}$$

In practice, it is not necessary to explicitly form $\mathbf{M}$ and take its QR decomposition since $\mathbf{C}_1$ is diagonal with c_{ii} being the square root of the number of replicates of the i'th unique $\mathbf{x}$. The elements of the vector $\mathbf{C}_1^{-1}\mathbf{B}_1^{\mathrm{T}}\mathbf{y}$ are the means of the y's at the corresponding unique $\mathbf{x}$'s. Further, $\| \mathbf{B}_2^{\mathrm{T}}\mathbf{y} \|^2$ is the sum of squares due to replication.

With this information available we can write

$$\omega = \mathbf{C}_1^{-\mathrm{T}}\delta$$

to produce

$$\begin{aligned} S_\lambda(\beta,\omega) = \frac{1}{n} \; \| \mathbf{B}_2^{\mathrm{T}}\mathbf{y} \|^2 + \frac{1}{n} \; \| \mathbf{B}_1^{\mathrm{T}}\mathbf{y} - \mathbf{C}_1\mathbf{T}_U\beta - \mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^{\mathrm{T}}\omega \|^2 \\ + \lambda\omega^{\mathrm{T}}\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^{\mathrm{T}}\omega \end{aligned}$$

and proceed as in the case with no replications using $\mathbf{C}_1\mathbf{T}_U$ in place of $\mathbf{T}$, and $\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^{\mathrm{T}}$ in place of $\mathbf{K}$. That is, take a QR decomposition

$$\mathbf{C}_1\mathbf{T}_U = \mathbf{F}\mathbf{G} = [\mathbf{F}_1 \colon \mathbf{F}_2]\begin{bmatrix} \mathbf{G}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{F}_1\mathbf{G}_1$$

and form $\mathbf{F}_2^{\mathrm{T}}\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^{\mathrm{T}}\mathbf{F}_2$ which then determines the Cholesky decomposition

$$\mathbf{F}_2^{\mathrm{T}}\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^{\mathrm{T}}\mathbf{F}_2 = \mathbf{L}^{\mathrm{T}}\mathbf{L} \; .$$

A SVD of $\mathbf{L}^{\mathrm{T}}$ as

$$\mathbf{L}^{\mathrm{T}} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$$

and the product

$$\mathbf{w}_2 = \mathbf{F}_2^{\mathrm{T}}\mathbf{B}_1^{\mathrm{T}}\mathbf{y}$$

allows us to write

$$V(\lambda) = \frac{n\left[\| \mathbf{B}_2^T \mathbf{y} \|^2 + \sum_{j=1}^{k-t} \left[\frac{n\lambda}{d_j^2 + n\lambda} \right]^2 z_j^2 \right]}{\left[n - k + \sum_{j=1}^{k-t} \frac{n\lambda}{d_j^2 + n\lambda} \right]^2} .$$

Given the value of $\hat{\lambda}$, the calculation of $\boldsymbol{\delta}_\lambda$ and $\boldsymbol{\beta}_\lambda$ follow as in the no-replicates case. That is,

$$\boldsymbol{\delta}_\lambda = \mathbf{C}_1^T \mathbf{F}_2 \mathbf{U} (\mathbf{D}^2 + n\lambda \mathbf{I})^{-1} \mathbf{U}^T \mathbf{F}_2^T \mathbf{B}_1^T \mathbf{y}$$

and $\boldsymbol{\beta}_\lambda$ is the solution of

$$\mathbf{G}_1 \boldsymbol{\beta}_\lambda = \mathbf{F}_1^T (\mathbf{B}_1^T \mathbf{y} - \mathbf{C}_1 \mathbf{K}_U \boldsymbol{\delta}_\lambda) .$$

Partial spline models: These are an extension to the thin-plate smoothing spline model in which some of the coordinates of $\mathbf{x}$, the "covariates", do not enter into the thin-plate spline. See Wahba (1984b, 1985) and Shiau, Wahba, and Johnson (1985). The model is

$$y_i = f(\mathbf{x}_i) + \sum_{j=1}^{c} \alpha_j \psi_j (\mathbf{x}_i, \mathbf{s}_i) + \varepsilon_i \qquad (4.1)$$

in which $\mathbf{s}_i$ are the "covariates" and $\{\psi_j\}$ are c given functions. For convenience, we will consider these variables as forming another matrix $\mathbf{S}$ of size $n \times c$. The partial spline estimates of f and $\boldsymbol{\alpha}$ are the minimizers of

$$S_\lambda(\alpha,\beta,\delta) = \frac{1}{n} \sum_{i=1}^{n} [y_i - f(\mathbf{x}_i) - \sum_{j=1}^{c} \alpha_j \psi_j (\mathbf{x}_i, \mathbf{s}_i)]^2 + \lambda J_m(f) ,$$

and it is known that the minimizing f_λ has a representation of the form (2.2). Let $\mathbf{S}$ be the $n \times c$ matrix with ij'th entry $\psi_j(\mathbf{x}_i, \mathbf{s}_i)$. The matrix $[\mathbf{T}{:}\mathbf{S}]$ must be of full column rank. The objective function for a fixed λ becomes

$$S_\lambda(\alpha,\beta,\delta) = \frac{1}{n} \| \mathbf{y} - \mathbf{S}\alpha - \mathbf{T}\beta - \mathbf{K}\delta \|^2 + \lambda \delta^T \mathbf{K} \delta .$$

When determining replicates, we only consider the d variables which determine the spline. When there are no replicates, we proceed as in the basic TPSS case except that we take the initial QR decomposition as

$$[\mathbf{T} : \mathbf{S}] = \mathbf{F}\mathbf{G} = [\mathbf{F}_1 {:} \mathbf{F}_2] \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{F}_1 \mathbf{G}_1$$

so $V(\lambda)$ is calculated as in (2.8) with all summations running to $n - t - c$. That is, after the Cholesky decomposition of $\mathbf{F}_2^T \mathbf{K} \mathbf{F}_2$ and the SVD of the transpose of the Cholesky factor, we have

$$V(\lambda) = \frac{n \sum_{j=1}^{n-t-c} \left[\frac{n\lambda}{d_j^2 + n\lambda} \right]^2 z_j^2}{\left[\sum_{j=1}^{n-t-c} \frac{n\lambda}{d_j^2 + n\lambda} \right]^2} .$$

The calculation of $\boldsymbol{\zeta}_\lambda$ and $\boldsymbol{\delta}_\lambda$ proceeds as in the basic TPSS case. With these available, we solve for $\boldsymbol{\alpha}_\lambda$ and $\boldsymbol{\beta}_\lambda$ simultaneously. In other words, we have simply replaced $\mathbf{T}\boldsymbol{\beta}$ in (2.6) by

$$[\mathbf{T} : \mathbf{S}] \begin{bmatrix} \beta \\ \alpha \end{bmatrix} .$$

It can be shown that the implied constraint $\mathbf{S}^T \delta = \mathbf{0}$ does not change the solution.

When we have covariates as well as some replications in the d coordinates of the $\mathbf{x}$'s, we have to distinguish between those columns of $\mathbf{S}$ which follow the replication pattern of the $\mathbf{x}$'s and those which do not. If all the columns of $\mathbf{S}$ follow the replication pattern, we have an indicator matrix $\mathbf{M}$ for which

$$\mathbf{T} = \mathbf{M}\mathbf{T}_U$$

$$\mathbf{K} = \mathbf{M}\mathbf{K}_U$$

$$\mathbf{S} = \mathbf{M}\mathbf{S}_U .$$

Taking the QR decomposition of $\mathbf{M}$ as $\mathbf{M} = \mathbf{B}\mathbf{C}$, we then take a QR decomposition of $\mathbf{C}_1 [\mathbf{T}_U {:} \mathbf{S}_U]$ and proceed as above.

If there are columns of $\mathbf{S}$ which do not follow the replication pattern of the design, we need a more general approach. The covariate matrix is divided into $\mathbf{S} = [\mathbf{S}_1 {:} \mathbf{S}_2]$ in which the columns of $\mathbf{S}_1$ have the same replication structure as the design points $\mathbf{x}_i$, $i = 1, \cdots, n$. We have an indicator matrix $\mathbf{M}$ for which

$$[\mathbf{T} : \mathbf{S}_1 : \mathbf{K}] = \mathbf{M} [\mathbf{T}_U : \mathbf{S}_{1U} : \mathbf{K}_U] ,$$

and a QR decomposition of $\mathbf{M} = \mathbf{B}\mathbf{C}$ as above. However, we cannot easily reduce the objective function to a form such as (3.2) by premultiplying by $\mathbf{B}$, as $\mathbf{B}_2^T \mathbf{S}_2$ is not annihilated. Instead we choose to take a QR decomposition of

$$\mathbf{C}_1 [\mathbf{T}_U : \mathbf{S}_{1U}] = \mathbf{F}\mathbf{G} = [\mathbf{F}_1 {:} \mathbf{F}_2] \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{F}_1 \mathbf{G}_1$$

which is used to reduce the parameter vector and penalty matrix. We proceed as in the case of a general design matrix with a semi-norm penalty as described in the next section by creating the parameter vector

$$\theta = \begin{bmatrix} \beta \\ \alpha \\ \zeta \end{bmatrix} , \text{ with } \delta = \mathbf{C}_1^T \mathbf{F}_2 \zeta ,$$

and the design matrix

$$\mathbf{X} = [\mathbf{T} : \mathbf{S}_1 : \mathbf{S}_2 : \mathbf{K}\mathbf{C}_1^{\mathrm{T}}\mathbf{F}_2] \ .$$

The penalty becomes $\theta^{\mathrm{T}}\Sigma\,\theta$, with

$$\Sigma = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_2^{\mathrm{T}}\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^{\mathrm{T}}\mathbf{F}_2 \end{bmatrix} \ .$$

Partial spline models with nodes at selected points, which may not actually correspond to data points, are discussed in Appendix 2.

General design matrix with a semi-norm penalty: The ridge regression case and the TPSS cases which we have considered both have some special structure. In the ridge regression case, the design matrix, $\mathbf{X}$, is general but the penalty term, $\gamma^{\mathrm{T}}\gamma$, has a special form so we can streamline the calculations. In the TPSS cases, the penalty term, $\delta^{\mathrm{T}}\mathbf{K}\delta$ subject to $\mathbf{T}^{\mathrm{T}}\delta = \mathbf{0}$, is more general but the design matrix, $[\mathbf{T}{:}\mathbf{K}]$ is related to the penalty so, again, we can exploit this special structure to provide faster algorithms. Even in the case with both a general design and a general penalty, though, we can still form efficient computational methods for GCV.

The most general GCV calculation we consider is the penalized least squares problem with an objective function

$$S_\lambda(\theta) = \frac{1}{n} \ \| \mathbf{y} - \mathbf{X}\theta \|^2 + \lambda\,\theta^{\mathrm{T}}\Sigma\,\theta \qquad (5.1)$$

where θ is a p-dimensional parameter vector, $\mathbf{y}$ is an n-dimensional response vector, $\mathbf{X}$ is an $n \times p$ design matrix, and Σ is a $p \times p$ positive semi-definite symmetric matrix defining the smoothness penalty. Note that partial splines can be written in this form as a special case.

A partial spline model with discontinuities in the $\{\psi_j\}$ of (4.1) which fits in the context of (5.1) is described in Shiau, Wahba, and Johnson (1985). Other special cases included splines and vector splines on the sphere (Wahba (1981), Wahba (1982a, 1982b, 1982c)) and remote sensing problems (Wahba (1980a)). Appendix 2 presents some examples and the algebra needed for a partial spline model with basis functions.

The minimization of (5.1) can also be used as a step in the iterative solution of penalized GLIM models (O'Sullivan (1983), O'Sullivan, Yandell and Raynor (1986)), nonlinear regularization (O'Sullivan (1983) and O'Sullivan and Wahba (1985)) and iteratively reweighted least squares (Green (1984), Green (1985) and Green and Yandell (1985)).

We can find the GCV estimate of λ in the general case by using a series of matrix decompositions to reduce (5.1) to the form of the ridge regression calculation as was done in the TPSS case. First we must isolate the null-space of the semi-norm defined by Σ. That is, we must describe the set of θ's for which

$$\theta^{\mathrm{T}}\Sigma\theta = \mathbf{0} \ .$$

We assume the dimension, h, of this space is known and take a pivoted Cholesky decomposition (Dongarra et al., 1979, chapter 8)

$$\mathbf{E}^{\mathrm{T}}\Sigma\mathbf{E} = \mathbf{L}^{\mathrm{T}}\mathbf{L}$$

where $\mathbf{E}$ is a $p \times p$ permutation matrix and $\mathbf{L}$ is $(p-h)\times p$ with zeros below the main diagonal. The conditioning of $\mathbf{L}$ is evaluated to ensure that $\mathbf{L}$ actually has computational rank $p-h$. If $\mathbf{L}$ is rank deficient, we increase h until the resulting $(p-h)\times p$ matrix $\mathbf{L}$ is of full row rank and return a non-fatal error code. If the user's value of h was too large, we return a fatal error code as this indicates that the null space of Σ is smaller than expected. As described in Appendix 1, the technique of increasing h until $\mathbf{L}$ is of full row rank is incompatible with the partial spline code as written here.

A QR decomposition of $\mathbf{L}^{\mathrm{T}}$ as

$$\mathbf{L}^{\mathrm{T}} = \mathbf{QR} = [\mathbf{Q}_1{:}\mathbf{Q}_2]\begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1$$

provides the $h \times p$ matrix $\mathbf{Q}_2$ which is an orthogonal basis for the null space of the semi-norm defined by Σ. We can now transform to parameters γ and β of dimension $p-h$ and h, respectively, as

$$\begin{pmatrix} \gamma \\ \beta \end{pmatrix} = \begin{bmatrix} \mathbf{R}_1^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{Q}^{\mathrm{T}}\mathbf{E}^{\mathrm{T}}\theta$$

where β lies in the null space and $S_\lambda(\theta)$ from (5.1) can be written

$$S_\lambda(\beta,\gamma) = \frac{1}{n} \ \| \mathbf{y} - \mathbf{XEQ}\begin{bmatrix} \mathbf{R}_1^{-\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{pmatrix} \gamma \\ \beta \end{pmatrix} \|^2 + \lambda\gamma^{\mathrm{T}}\gamma$$

$$= \frac{1}{n} \ \| \mathbf{y} - \mathbf{Z}\begin{pmatrix} \gamma \\ \beta \end{pmatrix} \|^2 + \lambda\gamma^{\mathrm{T}}\gamma$$

with

$$\mathbf{Z} = [\mathbf{Z}_1{:}\mathbf{Z}_2] = \mathbf{XEQ}\begin{bmatrix} \mathbf{R}_1^{-\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \ .$$

This provides the desired form of the penalty term. We must now divide the least squares term into a part that can be made zero by an appropriate choice of β and a part that depends only on γ. Another QR decomposition, this time as

$$\mathbf{Z}_2 = \mathbf{FG} = [\mathbf{F}_1{:}\mathbf{F}_2]\begin{bmatrix} \mathbf{G}_1 \\ \mathbf{0} \end{bmatrix}$$

is used to form

$$S_\lambda(\beta,\gamma) = \frac{1}{n} \|\mathbf{w}_1 - \mathbf{G}_1\beta - \mathbf{J}_1\gamma\|^2 + \frac{1}{n}\|\mathbf{w}_2 - \mathbf{J}_2\gamma\|^2 + \lambda\gamma^T\gamma \tag{5.2}$$

where

$$\mathbf{w} = \begin{bmatrix}\mathbf{w}_1\\ \mathbf{w}_2\end{bmatrix} = \begin{bmatrix}\mathbf{F}_1^T\\ \mathbf{F}_2^T\end{bmatrix}\mathbf{y} = \mathbf{F}^T\mathbf{y}$$

and

$$\mathbf{J} = \begin{bmatrix}\mathbf{J}_1\\ \mathbf{J}_2\end{bmatrix} = \begin{bmatrix}\mathbf{F}_1^T\\ \mathbf{F}_2^T\end{bmatrix}\mathbf{Z}_1 = \mathbf{F}^T\mathbf{Z}_1 \ .$$

After checking that $\mathbf{G}_1$ is non-singular, the first term in (5.2) can be made zero for any value of γ by solving

$$\mathbf{G}_1\beta = \mathbf{w}_1 - \mathbf{J}_1\gamma \tag{5.3}$$

for β. This reduces the general penalized least squares to the same form as the ridge regression. A singular value decomposition

$$\mathbf{J}_2 = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{5.4}$$

produces the representation of the "hat" matrix as

$$\mathbf{A}(\lambda) = \mathbf{F}\begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{U}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{D}^2(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{U}^T\end{bmatrix}\mathbf{F}^T \ .$$

The matrix $\mathbf{D}$ is $a \times a$, with $a = min(n,p) - h$, and the matrices $\mathbf{U}$ and $\mathbf{V}$ are rectangular of sizes $(n-h)\times a$ and $(p-h)\times a$, respectively. Again, using

$$\mathbf{z} = \mathbf{U}^T\mathbf{w}_2 \tag{5.5}$$

the GCV function can be expressed as

$$V(\lambda) = \frac{n\left[\|\mathbf{w}_2\|^2 - \|\mathbf{z}\|^2 + \sum_{j=1}^{a}\left[\frac{n\lambda}{d_j^2 + n\lambda}\right]^2 z_j^2\right]}{\left[n - p + \sum_{j=1}^{a}\frac{n\lambda}{d_j^2 + n\lambda}\right]^2} \ . \tag{5.6}$$

and the parameters vectors γ_λ, β_λ, and θ_λ are determined in the usual way given $\hat{\lambda}$, with (5.3) and

$$\gamma_\lambda = \mathbf{V}(\mathbf{D}^2 + n\lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{w}_2 \ ,$$

yielding

$$\theta_\lambda = \mathbf{E}\mathbf{Q}\begin{bmatrix}\mathbf{R}_1^{-T} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\gamma_\lambda\\ \beta_\lambda\end{bmatrix} \ .$$

The biggest computational bottleneck is the SVD of $\mathbf{J}_2$ when n and p are large, particularly since $\mathbf{J}_2$ is often ill–conditioned. We can accelerate the SVD calculation by using a truncated version of the singular value decomposition (Bates and Wahba, 1982). Notice that, in (5.6) and the solution of γ_λ, values of d_j such that

$$d_j^2 \ll n\lambda$$

can be set to zero without significantly changing the results. Starting with a tolerance $\tau\rho$, usually a small multiple (τ) of the relative machine precision (ρ), the truncated SVD algorithm finds a matrix $\tilde{\mathbf{J}}_2$ which has $a^* \le a$ positive singular values and satisfies

$$\frac{\|\tilde{\mathbf{J}}_2 - \mathbf{J}_2\|_F^2}{\|\mathbf{J}_2\|_F^2} < \tau\rho \ ,$$

in which $\|\bullet\|_F$ is the Frobenius norm. For details of the truncated SVD algorithm, see Appendix 3. We replace $\mathbf{J}_2$ by $\tilde{\mathbf{J}}_2$ in (5.4), thereby reducing the effective number of parameters to a^*. With the truncation we only calculate an $(n-h)\times a^*$ matrix $\mathbf{U}$ and a $(p-h)\times a^*$ matrix $\mathbf{V}$ so the vector $\mathbf{z}$ defined in (5.5) will be a^*-dimensional, with a replaced by a^*. When $\mathbf{J}_2$ is ill–conditioned, we get a^* considerably less than a and, since the calculation of the SVD is of order $O(na^2)$, this can create substantial savings in computing time. However, V is sensitive to τ for small λ. To check on the effect of the truncation on the value of $V(\lambda)$ and hence the calculation of $\hat{\lambda}$ we return the diagnostic quantity

$$n\hat{\lambda} / (n\hat{\lambda} + \|\tilde{\mathbf{J}}_2 - \mathbf{J}_2\|_F^2) \ . \tag{5.7}$$

This is a lower bound on each of the quantities $n\hat{\lambda}/(d_j^2 + n\hat{\lambda})$ in (5.6) which are replaced by 1 when d_j is set to zero. Preliminary tests indicate that if the diagnostic quantity is above 0.999 then the truncation has negligible effect on V.

Another important method of accelerating the GCV calculations by avoiding the final reduction to diagonal form in the SVD was given by Elden (1984). This involves stopping the evaluation of the singular value decomposition at the intermediate step of the reduction of $\mathbf{J}_2$ to a bidiagonal form, then forming an expression for $V(\lambda)$.

Description

The package has three main subroutine drivers. The first driver, *dtpss* for thin plate smoothing splines, is the most efficient and the most restrictive, allowing covariates only in the case where the replication pattern is the same as that found in the design. The second driver, *dptpss* for partial thin plate smoothing splines, handles general covariates and in turn calls the third driver, *dsnsm* which handles penalized least squares problems with a semi-norm penalty. After a call to *dtpss* or *dptpss* the subroutine *dpred* can be called to evaluate predicted values for additional points not in the design.

Replicates are handled in *dtpss* and *dptpss* using the following routines. The subroutine *dreps* sorts the $\mathbf{x}$ vectors and returns $\mathbf{C}_1$ and the information necessary for the routines *duni* and *dsuy* (used only in *dtpss*). Subroutine *duni* reduces a matrix ($\mathbf{T}$ or $\mathbf{K}$) to the corresponding matrix with unique entries ($\mathbf{T}_U$ or $\mathbf{K}_U$). The routine *dsuy* sorts $\mathbf{y}$ and computes $\mathbf{B}_1^T\mathbf{y}$ and the sum of squares due to replication.

The subroutine *dtpss*, the thin plate spline driver, calls the routine *dsetup* to create the matrices $\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^T$ and $\mathbf{C}_1[\mathbf{T}_U:\mathbf{S}_{1U}]$ from the design points $\mathbf{x}_i, i=1,2,\ldots,n$ using the routines *dmakek* and *dmaket*. The LINPACK routine *dqrdc* is called to decompose $\mathbf{C}_1[\mathbf{T}_U:\mathbf{S}_{1U}]$ into its QR decomposition $\mathbf{FG}$, followed by the routine *dftkf* to calculate $\mathbf{F}^T\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^T\mathbf{F}$. *Dsgdc1* does the Cholesky decomposition of $\mathbf{F}_2^T\mathbf{C}_1\mathbf{K}_U\mathbf{C}_1^T\mathbf{F}_2$ and the singular value decomposition of the Cholesky factor. *Dgcv1* uses these results to compute the generalized cross validation estimate of λ and the corresponding estimates of the other parameters. The work in *dgcv1* is divided into application of the rotations by $\mathbf{F}^T$ in *drsap*, optimization of the $V(\lambda)$ function in *dvlop*, computation of predictive mean square error (if requested) in *dpmse*, creation of the coefficient vector in *dcfcr1*, creation of the predicted values in *dpdcr*, and creation of the diagonal of $\mathbf{A}(\hat{\lambda})$ in *ddiag*. Subroutine *dvlop* calls *dvmin* to minimize $V(\lambda)$ by repeated calls to *dvl*. The minimization is done by an initial grid search in the $\ln(n\lambda)$ scale followed by a golden ratio search in the neighborhood of the minimizing grid point. The input variable *ntbl* controls the resolution of the initial grid search. A value for *ntbl* of 100 or greater is recommended to ensure that the global optimum is located. If a plot of $V(\lambda)$ versus $\ln(n\lambda)$ indicates that a local optimum has been obtained the user may either increase the value of *ntbl* or use the option to specify a reduced range for the search. The grid of $\ln(n\lambda)$ values is returned along with the corresponding $V(\lambda)$ values in the variable *tbl*. The variable *auxtbl* is returned containing $\hat{\lambda}$, $V(\hat{\lambda})$, $V(0)$ and $V(\infty)$.

The driver *dptpss* for partial thin plate splines calls routines *dreps*, *dmaket*, *duni*, and *dmakek* to set up $[\mathbf{T}:\mathbf{S}]$, $[\mathbf{T}_U:\mathbf{S}_{1U}]$ and $\mathbf{K}_U$. These are fed to *dctsx* to create the matrices Σ and $\mathbf{X}$ which are used by the driver *dsnsm*.

The subroutine *dsnsm* is a general driver for penalized least squares problems with a semi-norm penalty. It calls *ddcom* which decomposes Σ and $\mathbf{X}$ and returns information used by *dgcv* to find $\hat{\lambda}$, θ_λ, and other results. The work in *ddcom* is split into the decomposition of Σ in a call to *dsgdc* and the transformation and decomposition of the design in *dcrtz* and *dzdc* which in turn calls *dtsvdc* or *dsvdc* to perform the singular value decomposition. The work in *dgcv* is divided into the same subroutines as *dgcv1* with the exception that *dcfcr1* is replaced by *dcfcr*.

In the general case, the driver *dsnsm* allows an option to use a truncation singular value decomposition through the routine *dtsvdc* which preprocesses the design matrix $\mathbf{J}_2$ to reduce the dimensionality before invoking *dsvdc* (see Appendix 3). The truncation tolerance, $\tau\times\rho$ is passed to *dtsvdc* as the parameter *minrat*. The drivers *dtpss* and *dptpss* would not benefit from truncation in the SVD calculation so they use the LINPACK routine *dsvdc*.

Simulation Applications: When GCVPACK is used for simulation studies the option to compute the predictive mean square error should be used. The known "true" response is input in the variable *adiag* and the predictive mean square error, $R(\lambda)$, is returned, along with $V(\lambda)$, in the variable *tbl*. It is recommended that plots of $V(\lambda)$ and $R(\lambda)$ versus $\ln(n\lambda)$ be used to evaluate the success of the GCV function in finding the optimal λ (the λ which minimizes predictive mean square error). The variable *auxtbl* contains $R(\hat{\lambda})$, $R(0)$ and $R(\infty)$.

The decomposition of the $\mathbf{X}$ matrix requires the most intensive computation. The subroutines *dtpss* and *dptpss* are both set up to take advantage of the savings in computation that exist for multiple response vectors with the same design. To modify *dtpss* to handle a problem with more than one response vector all code up to and including the call to *dsgdc1* is executed once. A loop can be added to execute the remaining code for each $\mathbf{y}$ vector. In practice this modification would involve adding only a few lines of code.

To modify *dptpss*, or any other driver which calls *dsnsm*, a loop must be added in *dsnsm*. In *dsnsm* there are two subroutines, *ddcom* which needs to be executed once, and *dgcv* which must be executed once for each response vector. In *dptpss*, after the call to *dsnsm*, a transformation is applied to the coefficient vector. This must be done to the coefficient vector corresponding to each $\mathbf{y}$ vector.

Related Algorithms

The numerical linear algebra in our routines is performed using the LINPACK (Dongarra et al., 1979) routines. The introductory comments of each GCVPACK routine list which LINPACK and BLAS (Basic Linear Algebra Subroutines) routines are called directly or indirectly. There is one machine–dependent constant, the relative machine precision, which is used in these routines to determine error conditions caused by ill-conditioning, but that constant is computed each time it is needed.

The present work generalizes algorithms for ridge regression of Golub, Heath, and Wahba (1979) and Bates and Wahba

(1982) which use the singular value decomposition. Elden (1977) gives an algorithm which terminates the singular value decomposition at an intermediate step, reducing **X** to a bidiagonal form, thereby saving time (see the **Test Results** section). This could be incorporated into GCVPACK but we have not done so yet.

Wendelberger (1981) implemented an algorithm for thin plate splines based on eigenvalue-eigenvector decompositions for one-dimensional and multi-dimensional thin plate smoothing splines. Hutchinson (1984) developed an algorithm for thin plate splines with large data sets using the thin plate basis functions of Wahba (1980b); see Appendix 2.

Reinsch (1967) initially proposed a fast algorithm for fixed λ using a Cholesky decomposition (see De Boor (1978)). In the one-dimensional case, the penalty can be written as a product of matrices with only $2m-1$ non-zero diagonals. Hutchinson and de Hoog (1985) give an $O(n)$ algorithm for computing $V(\lambda)$ using a Cholesky decomposition of these matrices. See also O'Sullivan (1985). GCVPACK is not designed to take advantage of the unique structure of one dimensional polynomial smoothing splines, and runs much slower than the code of Hutchinson and de Hoog (1985) in this case.

O'Sullivan, Yandell and Raynor (1986) developed algorithms for smooth generalized linear models based on a Cholesky decomposition of $\mathbf{X}^T\mathbf{X} + n\lambda\mathbf{I}$. Green (1985) and Green and Yandell (1985) presented algorithms for penalized likelihood schemes which include generalized linear models and other iteratively reweighted least squares methods. They present a one-dimensional algorithm based on Reinsch (1967) and a general algorithm based on the Cholesky decomposition. They have also incorporated an iterative algorithm using the SVD to automate the choice of $\hat{\lambda}$, but it needs extensive testing to determine if it is stable. Shiau (1985) developed algorithms for a particular class of partial splines consisting of discontinuities of f or higher order derivatives at known or unknown points. This includes a one-dimensional algorithm based on Hutchinson and de Hoog (1985) and a multidimensional algorithm based on the Cholesky decomposition.

Test Results

The package and drivers have been tested for internal consistency and for accuracy against other known algorithms. Here we present some timing results to show that the methods are feasible for relatively large data sets and to offer insight into which portions of the code should be avoided, if possible. For example, the code allows the computation of the diagonal of $\mathbf{A}(\hat{\lambda})$ for forming diagnostics (Eubank, 1984) but this calculation alone can take 15% or more of the total execution time.

All timing runs were performed on a Vax-11/750 computer with a floating point accelerator and running the 4.2 BSD *UNIX*™ operating system. We quote two sets of times for the example: one using the driver *dtpss* and the other using *dptpss*. Each of the drivers was timed twice: first using the Fortran version of the Basic Linear Algebra Routines (BLAS) then using Assembler Language BLAS. As explained in Dongarra et al. (1979), the BLAS are a set of low-level routines that perform such elementary tasks as accumulation of dot products and, by replacing them with Assembler language versions, the Linpack routines can be made to run faster.

The design for the example is a 9 by 9 factorial in $\mathbf{x}_1$ and $\mathbf{x}_2$ with one covariate, $\mathbf{x}_2^2$. Two replicate observations were simulated at each of the 81 design points. Thus $n = 162$, $k = 81$, $m = 2$, $d = 2$ and $c = 1$. Our timing results are shown in Tables 1 and 2. The total times are slightly greater than the sum of the times spent in the lower level subroutines since the driver routines have to do some definition of pointers, etc.

The first thing to notice from these tables is that *dtpss* is strongly preferred over *dptpss* for this example since it executes approximately 3 times faster. In general, if *dtpss* can solve the problem, it will do so more quickly. Also, the Assembler BLAS speed things up considerably with most of the gain being in the call to the Linpack SVD routine *dsvdc*.

	Fortran BLAS		Assembler BLAS	
Routine	Seconds	%	Seconds	%
dreps	3.07	3	3.03	4
dsetup	11.85	11	9.42	12
dsgdc1				
Cholesky	3.70	3	2.13	3
bidiag.	31.55	29	15.50	19
diag.	36.10	33	34.65	43
dsuy	0.05	0	0.05	0
dgcv1				
drsap	0.20	0	0.12	0
dvlop	1.70	2	1.68	2
dpmse	1.47	1	1.37	2
dcfcr1	0.28	0	0.15	0
dpdcr	0.25	0	0.12	0
ddiag	18.07	17	11.15	14
Total dtpss	108.92		79.88	

Table 1: Example 1 using *dtpss*

UNIX is a trademark of AT&T Bell Laboratories

	Fortran BLAS		Assembler BLAS	
Routine	Seconds	%	Seconds	%
dreps	3.02	1	3.03	1
make **K** and **T**	14.45	4	14.62	6
dctsx	8.38	2	3.98	2
ddcom				
dsgdc				
Cholesky	4.05	1	2.22	1
QR	10.85	3	5.40	2
dcrtz	56.30	16	27.45	12
dzdc				
bidiag.	86.43	24	41.77	18
diag.	101.55	28	82.55	36
dgcv				
drsap	0.42	0	0.23	0
dvlop	1.67	0	1.67	1
dpmse	1.65	0	1.45	1
dcfcr	0.57	0	0.28	0
dpdcr	0.47	0	0.25	0
ddiag	66.02	18	39.65	18
Total dptpss	359.43		226.28	

Table 2: Example 1 using *dptpss*

We have divided the time for *dsvdc* into two subsections, *bidiag* and *diag*. Elden (1984) gave a method of expressing the GCV function $V(\lambda)$ avoiding the *diag* step. This would result in considerable savings in the *dsgdc1* or *ddcom* routines. This savings is offset by the calculations in *dgcv1* or *dgcv* becoming more complicated and, possibly, taking longer. However, since those routines take up much less time than *diag*, we would expect that the overall savings would be worthwhile.

Notice that the calculation of the diagonal of $\mathbf{A}(\hat{\lambda})$ in *ddiag* is comparatively expensive – usually around 15% of the total execution time. If this optional information is not going to be used, it should not be calculated.

In circumstances where there are multiple **y** vectors being analysed for the same design and penalty matrices, such as in Monte-Carlo runs, the decomposition portion, *dsgdc1* or *ddcom*, should be called only once while the analysis portion, *dgcv1* or *dgcv*, called for each **y**. The analysis portion represents less than 5% of the total time if the calculation of the diagonal of $\mathbf{A}(\hat{\lambda})$ is not undertaken.

The sorting method used in *dreps* is a comparatively primitive sort (a modification of the bubble sort) but, even so, the time taken by *dreps* is a small percentage of the total time. It would be possible to speed up this step by using a more sophisticated sort, but it doesn't appear worthwhile. Also, the evaluation of $V(\lambda)$ after the matrices are decomposed is very quick. In these runs the variable *ntbl* was set to 200 so both $V(\lambda)$ and the mean squared error of prediction (since the data were simulated) were evaluated at 200 different values of λ. Even with 200 evaluations *dvlop* and *dpmse* each represented, at most, 2% of the execution time.

Appendix 1. – replicates and rank-deficient penalty matrices

Because the functions E_m defined in (2.3) are increasing functions of the length of their argument, the matrix **K** defined in (2.4) will be close to singular if $\| \mathbf{x}_i - \mathbf{x}_j \|$ is very small for some $i \neq j$. To avoid an indeterminacy in the parameters of the thin-plate spline, we determine replicates by comparing $\| \mathbf{x}_i - \mathbf{x}_j \|$ to a tolerance level rather than checking for $\mathbf{x}_i = \mathbf{x}_j$. The tolerance level is calculated as 100 times the relative machine precision times the length of the diagonal of the smallest rectangle which encloses the $\mathbf{x}_i, i = 1, \ldots, n$. In all our test cases, this check has been adequate to ensure that the matrix $\mathbf{F}_2^T \mathbf{K} \mathbf{F}_2$ is computationally positive definite.

It is important to note that the determination of replicates involves sorting the $\mathbf{x}_i, i = 1, \ldots, n$, in increasing lexicographic order. That is,

$$\text{the rows of } \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 1 & 2 \\ 2 & 2 \\ 3 & 2 \\ 4 & 2 \end{bmatrix} \text{ would be re–ordered as } \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 2 & 1 \\ 2 & 2 \\ 3 & 1 \\ 3 & 2 \\ 4 & 1 \\ 4 & 2 \end{bmatrix} .$$

As mentioned in the **Test Results** section, the sorting algorithm is comparatively primitive (a modification of a bubble sort) and, even though it does not take a substantial percentage of the total execution time, it is to the user's advantage to pass the argument *des* to *dtpss* or *dptpss* with the rows in increasing lexicographic order, if possible, as the sorting time will be minimized.

Replicates are determined in such a way as to avoid a singular penalty matrix because a singular penalty matrix has a different effect for the thin-plate smoothing spline (or partial spline) than it does for the case of a general design matrix with a semi-norm penalty. In the general case, we determine the null space of the penalty so unexpected singularities simply increase the dimension of the null space and that part of the parameter vector is incorporated into the β. Ordinary regression is used to determine β and we assume (and check) that the part of the design matrix corresponding to β is non-singular. Unless the singularity in the penalty corresponds to a singularity in the design, everything works well.

In the case of a thin-plate smoothing spline the least squares part of the objective function (2.6) uses the same matrix ($\mathbf{F}_2^T\mathbf{K}\mathbf{F}_2$) as the penalty part. Thus, when the penalty is rank-deficient, the "design" matrix (in the regression sense) is also rank deficient and the parameters which lie in the extended null space of the penalty are indeterminant. This can be seen from the form of (2.6). If there are singular values of zero, the corresponding parameters have no effect on the predictions and thus do not enter into the objective function $S_\lambda(\beta,\delta)$. There is a parameter vector which can be calculated using (2.9) even with some zero singular values but the part corresponding to the zero singular values can be changed to an arbitrary value without affecting the predictions so, in particular, it could be set to zero. More specifically, consider the last two terms in the last line of (2.6), after the Cholesky decomposition:

$$\frac{1}{n} \| \mathbf{w}_2 - \mathbf{L}^T\mathbf{L}\boldsymbol{\delta} \| + \lambda \boldsymbol{\delta}^T\mathbf{L}^T\mathbf{L}\boldsymbol{\delta} \ . \tag{A1.1}$$

If $\mathbf{L}$ is not of full row rank, any $\boldsymbol{\delta}$ satisfying

$$\mathbf{L}^T\mathbf{L}\mathbf{w}_2 = [(\mathbf{L}^T\mathbf{L})^2 + n\lambda\,\mathbf{L}^T\mathbf{L}]\,\boldsymbol{\delta}$$

minimizes (A1.1), and in particular we could take

$$\boldsymbol{\delta} = (\mathbf{L}^T\mathbf{L} + n\lambda\mathbf{I})^{-1}\mathbf{w}_2 \ .$$

However, we have chosen not to write the special code that would be required to handle this case. We have eliminated one source of a computationally singular penalty matrix for the thin plate spline by merging nearly replicated data points. If the computational singularity of $\mathbf{F}_2^T\mathbf{K}\mathbf{F}_2$ is due to other than nearly replicated data points, i.e., due to very large sets of highly irregularly spaced data, the user should consider using thin plate basis functions as described in Appendix 2.

Appendix 2. – partial splines with basis functions

One can use the algorithm for a general design matrix with semi-norm penalty to find partial thin-plate smoothing splines determined by basis functions centered at specified nodes. See Shiau, Wahba, and Johnson (1985). For example, consider the model

$$y_i = \int \cdots \int K(\mathbf{x}_i,\mathbf{x}) f(\mathbf{x}) d\mathbf{x} + \varepsilon_i$$
$$= L_i f + \varepsilon_i \ .$$

The estimate f_λ of f is the minimizer, in an appropriate space, of

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - L_i f)^2 + \lambda J(f) \tag{A2.1}$$

where $J(f)$ is an appropriate (quadratic) roughness penalty. If we can approximate f_λ by

$$f_\lambda = \sum_{i=1}^{b} \theta_i B_i$$

where $\{B_i\}$ are suitably chosen basis functions, then we can define the ij'th entry of $\mathbf{X}$ as $L_i B_j$ and the matrix Σ by $J(\sum \theta_j B_j) = \boldsymbol{\theta}^T \Sigma \boldsymbol{\theta}$.

The thin plate basis functions were proposed for this purpose by Wahba (1980a). Starting with a set of suitably distributed distinct nodes $\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_b$, the approximation is

$$f_\lambda(\mathbf{x}) = \sum_{i=1}^{t} \beta_i \phi_i(\mathbf{x}) + \sum_{i=1}^{b} \delta_i E_m(\mathbf{x} - \mathbf{t}_i) \tag{A2.2}$$

where $\boldsymbol{\delta} = (\delta_1, \cdots, \delta_b)^T$ must satisfy

$$\sum_{i=1}^{b} \delta_i \phi_j(\mathbf{t}_i) = 0 \ , \ j=1,\cdots t \ .$$

If f_λ is required to be of the form (A2.2), then (A2.1) becomes

$$S_\lambda(\beta,\delta) = \frac{1}{n} \| \mathbf{y} - \mathbf{T}\boldsymbol{\beta} - \mathbf{K}\boldsymbol{\delta} \|^2 + \lambda \boldsymbol{\delta}^T \mathbf{K}_B \boldsymbol{\delta}$$

subject to $\mathbf{T}_B^T\boldsymbol{\delta} = \mathbf{0}$, with

$$\{\mathbf{T}_B\}_{ij} = \phi_j(\mathbf{t}_i) \ .$$

Here, $\mathbf{T}$ is $n \times t$ and $\mathbf{K}$ is $n \times b$, with entries

$$\{\mathbf{T}\}_{ij} = L_i \phi_j \ ,$$
$$\{\mathbf{K}\}_{ij} = L_i E_m(\bullet - \mathbf{t}_j) \ ,$$

and $\mathbf{K}_B$ is $b \times b$ with entries

$$\{\mathbf{K}_B\}_{ij} = E_m(\mathbf{t}_i - \mathbf{t}_j) \ .$$

If we are interested simply in evaluation functionals, then $L_i f = f(\mathbf{x}_i)$. The matrices $\mathbf{T}_B$ and $\mathbf{K}_B$ remain the same, but the matrices $\mathbf{T}$ and $\mathbf{K}$ have entries

$$\{\mathbf{T}\}_{ij} = \phi_j(\mathbf{x}_i) \ ,$$
$$\{\mathbf{K}\}_{ij} = E_m(\mathbf{x}_i - \mathbf{t}_j) \ .$$

We take a QR decomposition

$$\mathbf{T}_B = \mathbf{F}\mathbf{G} = [\mathbf{F}_1 \vdots \mathbf{F}_2] \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{F}_1\mathbf{G}_1$$

and use this to construct the parameter vector

$$\boldsymbol{\theta} = \begin{pmatrix} \beta \\ \zeta \end{pmatrix} \ , \text{ with } \boldsymbol{\delta} = \mathbf{F}_2\zeta \ ,$$

and to create the design matrix

$$\mathbf{X} = [\mathbf{T} : \mathbf{KF}_2]$$

and penalty matrix

$$\Sigma = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_2^{\mathrm{T}}\mathbf{K}_B\mathbf{F}_2 \end{bmatrix} .$$

We then proceed as in the case of a general design matrix with a semi-norm penalty as described earlier.

Hutchinson's (1984) code implements thin plate basis functions for the case $L_i f = f(\mathbf{x}_i)$, where b is chosen to be much less than n when n is large. Hutchinson's code, or the partial thin plate smoothing spline code described here, should be considered in the case that n is very large or $\mathbf{F}_2^{\mathrm{T}}\mathbf{KF}_2$ of (2.6) is computationally singular.

Covariates and replicates are handled as before and enter in the same way as for partial spline models. Considering here only the case of no replicates, the model with covariates is

$$f_\lambda(\mathbf{x},\mathbf{s}) = \sum_{i=1}^{t}\beta_i\phi_i(\mathbf{x}) + \sum_{i=1}^{b}\delta_i E_m(\mathbf{x}-\mathbf{t}_i) + \sum_{j=1}^{c}\alpha_j\psi_j(\mathbf{x},\mathbf{s}) .$$

The objective function for a fixed λ becomes

$$S_\lambda(\alpha,\beta,\delta) = \frac{1}{n}\, \| \mathbf{y} - \mathbf{S}\alpha - \mathbf{T}\beta - \mathbf{K}\delta \|^2 + \lambda\,\delta^{\mathrm{T}}\mathbf{K}_B\,\delta$$

subject to $\mathbf{T}_B^{\mathrm{T}}\delta = \mathbf{0}$, in which $\mathbf{S}$ is $n \times c$ with entries

$$\{\mathbf{S}\}_{ij} = L_i\psi_j(\bullet,\mathbf{s}_i) \ ,$$

or, for evaluation functionals,

$$\{\mathbf{S}\}_{ij} = \psi_j(\mathbf{x}_i,\mathbf{s}_i) \ .$$

The design matrix becomes

$$\mathbf{X} = [\mathbf{T} : \mathbf{S} : \mathbf{KF}_2]$$

with parameter vector

$$\theta = \begin{bmatrix} \beta \\ \alpha \\ \zeta \end{bmatrix} .$$

The penalty Σ has the same form, with the addition of rows and columns of zeroes corresponding to $\boldsymbol{\alpha}$. One would then proceed with the general design matrix with semi-norm penalty.

Appendix 3. – the truncated singular value decomposition

The following theorem of Mirsky (1960) provides a bound for the error in the singular values when using an approximation to a matrix.

Theorem 1: Let $\mathbf{X}$ and $\mathbf{Y}$ be $n \times p$ $(n \geq p)$ matrices with singular value decompositions $\mathbf{UDV}^{\mathrm{T}}$ and $\mathbf{RSW}^{\mathrm{T}}$ respectively. Denote the ordered singular values of $\mathbf{X}$ as $\{d_i\}, i=1,\cdots,p$ with $d_1 \geq d_2 \geq \cdots \geq d_p$ and the ordered singular values of $\mathbf{Y}$ as $\{s_i\}, i=1,\cdots,p$. Then

$$\sum_{i=1}^{p}(d_i - s_i)^2 \leq \| \mathbf{X} - \mathbf{Y} \|_F^2 = tr\,[(\mathbf{X}-\mathbf{Y})^{\mathrm{T}}(\mathbf{X}-\mathbf{Y})]$$

We will take advantage of this theorem to calculate the SVD of a matrix $\mathbf{X}_{a*}$ which is close to $\mathbf{X}$ in the sense that $\| \mathbf{X} - \mathbf{X}_{a*} \|$ is small but is better conditioned than is $\mathbf{X}$ so the iterative portion of the SVD tends to converge faster and the computational burden is reduced. First, we take a pivoted QR decomposition of $\mathbf{X}$ using the pivoting scheme from LINPACK (Dongarra et al., 1979). That is, we determine $\mathbf{Q}$, $n \times n$ orthogonal, $\mathbf{R}$, $n \times p$ and zero below the main diagonal, and $\mathbf{E}$, a $p \times p$ permutation matrix, such that

$$\mathbf{XE} = \mathbf{QR} \qquad \text{(A3.1)}$$

and $\mathbf{R}$ has the property that

$$r_{a^*,a^*}^2 \geq \sum_{i=a^*}^{j} r_{i,j}^2 \quad (j = a^*, a^*+1, \cdots, p). \qquad \text{(A3.2)}$$

If we take the SVD of $\mathbf{R}_p$, the triangular matrix composed of the first p rows of $\mathbf{R}$, as

$$\mathbf{R}_p = \mathbf{KDL}^{\mathrm{T}} \qquad \text{(A3.3)}$$

we can produce the SVD of $\mathbf{X}$ as

$$\mathbf{X} = \mathbf{Q}_p\mathbf{KDL}^{\mathrm{T}}\mathbf{E}^{\mathrm{T}} = \mathbf{UDV}^{\mathrm{T}} \qquad \text{(A3.4)}$$

where $\mathbf{Q}_p$ is the $n \times p$ matrix composed of the first p columns of $\mathbf{Q}$ and $\mathbf{U} = \mathbf{Q}_p\mathbf{K}$ is $n \times p$ while $\mathbf{V} = \mathbf{EL}$ is $p \times p$ and orthogonal. This method would not, however, produce better conditioning for the SVD algorithm since the singular values of $\mathbf{R}_p$ are the same as the singular values of $\mathbf{X}$.

To provide better conditioning, we truncate the matrix $\mathbf{R}_p$ after the a^*'th row and take the SVD of the resulting $n \times a^*$ matrix $\mathbf{R}_{a*}$ $(a^* \leq p)$ as

$$\mathbf{R}_{a*} = \mathbf{K}_{a*}\mathbf{D}_{a*}\mathbf{L}_{a*} \qquad \text{(A3.5)}$$

where $\mathbf{K}_{a*}$ is $a^* \times a^*$ and $\mathbf{L}_{a*}$ is $a^* \times p$. The diagonal elements of $\mathbf{D}_{a*}$ are no longer the singular values of $\mathbf{X}$ but now represent the singular values of a matrix

$$\mathbf{X}_{a*} = \mathbf{Q}_p \begin{bmatrix} \mathbf{R}_{a*} \\ 0 \end{bmatrix} \mathbf{E}^{\mathrm{T}} \qquad \text{(A3.6)}$$

which is different from $\mathbf{X}$. However,

$$\| \mathbf{X} - \mathbf{X}_{a*} \|_F = \left[\sum_{i=a^*+1}^{p} \sum_{j=i}^{p} r_{i,j}^2 \right]^{1/2} \qquad \text{(A3.7)}$$

so we can choose a^* to be as small as possible subject to the constraint that

$$\frac{\| \mathbf{X}-\mathbf{X}_{a*} \|_F}{\| \mathbf{X} \|_F} \leq \tau\rho \qquad \text{(A3.8)}$$

where ρ is the relative machine precision (the smallest number such that $1+\rho > 1$ in floating point arithmetic) and τ is a small multiplier.

We initially choose τ as unity but increase it if the LINPACK singular value decomposition routine (*dsvdc*) fails to converge. When such a convergence failure occurs, the user can either increase the number of iterations per singular value allowed in *dsvdc* (we increase this from 30 to 90) or increase τ or both. To increase the maximum allowable number of iterations, change the value of MAXIT in *dsvdc*.

Allowing τ to get too large can result in inaccuracies in the calculation of V. The effect of the truncation is measured by the diagnostic ratio defined in (5.7). In general, values of τ above 100 are not recommended.

The double sum on the right of (A3.7) is easily evaluated a row at a time starting at the p'th row until the constraint (A3.8) is violated and the smallest $a*$ is determined.

By theorem 1, if $\{d_i\}$, i=1,...,p are the ordered singular values of $\mathbf{X}$ and $\{d_{i,a*}\}$, i=1,...,p are the ordered singular values of $\mathbf{X}_{a*}$, then

$$(\sum_{i=1}^{p}(d_i-d_{i,a*})^2)^{1/2} \leq \tau\rho \| \mathbf{X} \| = \tau\rho\, (\sum_{i=1}^{p} d_i^2)^{1/2} \qquad \text{(A3.9)}$$

If $n \leq p$, the same procedure is applied to $\mathbf{X}^T$.

Acknowledgements

This research has been supported in part by National Science Foundation grants DMS-8404970 and ATM-8410373, United States Department of Agriculture CSRS grant 511-100, NASA grant NAG5-316 and ONR contract N00014-77-C-0675. Computing was performed on the UW–Madison Statistics Research Computer. We wish to thank Mr. Shee Ham, Dept. of Economics, UW-Madison, for helping us debug early drafts of the code and document.

Bibliography

Bates, D. M. and Wahba, G. (1982), "Computational Methods for Generalized Cross Validation with Large Data Sets," in *Treatment of Integral Equations by Numerical Methods*, eds. C. T. H. Baker and G. F. Miller, New York: Academic Press.

Craven, P. and Wahba, G. (1979), "Smoothing Noisy Data with Spline Functions: Estimating the Correct Degree of Smoothing by the Method of Generalized Cross-Validation," *Numerische Mathematik*, 31, 377-403.

de Boor, C. (1978), *A Practical Guide to Splines*, New York: Springer.

Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W. (1979), *Linpack Users' Guide*, Philadelphia: SIAM.

Duchon, J. (1976), "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces," in *Constructive Theory of Functions of Several Variables*, eds. W. Schempp and K. Zeller, 85-100.

Elden, L. (1977), "Algorithms for the Regularization of Ill-Conditioned Least Squares Problems," *BIT*, 17, 134-145.

Elden, L. (1984), "A Note on the Computation of the Generalized Cross-Validation Function for Ill-Conditioned Least Squares Problems," *BIT*, 24, 467-472.

Eubank, R. L. (1984), "The Hat Matrix for Smoothing Splines," *Statistics & Probability Letters*, 2, 9-14.

Golub, G. H., Heath, M., and Wahba, G. (1979), "Generalised Cross Validation as a Method for Choosing a Good Ridge Parameter," *Technometrics*, 21, 215-224.

Green, P. J. (1984), "Iteratively Reweighted Least Squares for Maximum Likelihood Estimation, and Some Robust and Resistant Alternatives," *Journal of the Royal Statistical Society, Ser. B*, 46, 149-170. (Discussion 171-192)

Green, P. J. (1985) "Penalized Likelihood for General Semi-Parametric Regression Models." Technical Report #2819, Mathematics Research Center, Univ. of Wisconsin–Madison.

Green, P. J. and Yandell, B. S. (1985), "Semi-Parametric Generalized Linear Models," in *GLIM85: Proceedings of the International Conference on Generalized Linear Models, September 1985*, ed. R. Gilchrist Lecture Notes in Statistics, Springer-Verlag. (Technical Report#2847, Math. Res. Cen., U. of Wisconsin)

Hutchinson, M. F. (1984) "A Summary of Some Surface Fitting and Contouring Programs for Noisy Data.'' Technical Report #ACT84/6, Div. Math. and Stat., CSIRO.

Hutchinson, M.F. and de Hoog, F.R. (1985) ''Smoothing Noisy Data with Spline Functions.'' *Numerische Mathematik*, . (to appear)

Mirsky, L. (1960), ''Symmetric Gauge Functions and Unitarily Invariant Norms,'' *Quart. J. Math. Oxford*, 11, 50-59.

Nychka, D., Wahba, G., Goldfarb, S., and Pugh, T. (1984), ''Cross-Validated Spline Methods for the Estimation of Three Dimensional Tumor Size Distributions from Observations on Two Dimensional Cross Sections,'' *J. Am. Stat. Assoc.*, 79, 832-846.

O'Sullivan, F. (1983) ''The Analysis of Some Penalized Likelihood Estimation Schemes.'' Technical Report #726, Department of Statistics, University of Wisconsin–Madison.

O'Sullivan, F. (1985), ''Discussion of Dr. Silverman's Paper,'' *Journal of the Royal Statistical Society, Ser. B*, 47, 39-40.

O'Sullivan, F. and Wahba, G. (1985), ''A Cross Validated Bayesian Retrieval Algorithm for Non-Linear Remote Sensing Experiments,'' *Journal of Computational Physics*, 59, 441-455.

O'Sullivan, F., Yandell, B.S., and Raynor, Jr., W.J. (1986), ''Automatic Smoothing of Regression Functions in Generalized Linear Models,'' *Journal of the American Statistical Association*, 81, 96-103.

Reinsch, C.H. (1967), ''Smoothing by Spline Functions,'' *Numerische Mathematik*, 10, 177-183.

Shiau, J. (1985) ''Smoothing Spline Estimation of Functions with Discontinuities.'' Technical Report #768, Department of Statistics, University of Wisconsin–Madison.

Shiau, J., Wahba, G., and Johnson, D.R. (1985) ''Partial Spline Models for the Inclusion of the Tropopause and Frontal Boundary Information in Otherwise Smooth Two- and Three-Dimensional Objective Analysis.'' Technical Report #777, Department of Statistics, University of Wisconsin – Madison.

Wahba, G. (1980a) ''Ill-Posed Problems: Numerical and Statistical Methods for Mildly, Moderately and Severly Ill-Posed Problems with Noisy Data.'' Technical Report #595, Department of Statistics, University of Wisconsin–Madison. (to appear in *Proceedings of the International Conference on Ill-Posed Problems*, M. Z. Nashed, ed.)

Wahba, G. (1980b), ''Spline Bases, Regularization, and Generalized Cross Validation for Solving Approximation Problems with Large Quantities of Noisy Data,'' in *Approximation Theory III*, ed. W. Cheney Academic Press, 905-912.

Wahba, G. (1981), ''Spline Interpolation and Smoothing on the Sphere.,'' *SIAM Journal of Scientific and Statistical Computing*, 2, 5-16.

Wahba, G. (1982a), ''Vector Splines on the Sphere, with Application to the Estimation of Vorticity and Divergence from Discrete, Noisy Data,'' in *Multivariate Approximation Theory, Vol. 2*, eds. W. Schempp and K. Zeller Birkhauser Verlag, 407-429.

Wahba, G. (1982b), ''Erratum: Spline Interpolation and Smoothing on the Sphere,'' *SIAM Journal of Scientific and Statistical Computing*, 3, 385-386.

Wahba, G. (1982c), ''Variational Methods in Simultaneous Optimum Interpolation and Initialization,'' in *The Interaction Between Objective Analysis and Investigation*, ed. D. Williamson, Boulder, CO: NCAR, 178-185.

Wahba, G. (1982d), ''Constrained Regularization for Ill Posed Linear Operator Equations, with Applications in Meteorology and Medicine,'' in *Statistical Decision Theory and Related Topics III, Vol. 2*, eds. S. S. Gupta and J. O. Berger Academic Press, 383-418.

Wahba, G. (1984a), ''Surface Fitting with Scattered, Noisy Data on Euclidean D-Spaces and on the Sphere,'' *Rocky Mountain J. Math.*, 14, 281-299.

Wahba, G. (1984b), ''Cross Validated Spline Methods for the Estimation of Multivariate Functions from Data on Functionals,'' in *Statistics: An Appraisal, Proceedings 50th Anniversary Conference Iowa State Statistical Laboratory*, eds. H. A. David and H. T. David, Ames: Iowa State University Press.

Wahba, G. (1985), ''Comments on "Projection Pursuit" by Peter J. Huber,'' *Annals of Statistics*, 13, 518-521.

Wahba, G. and Wendelberger, J. (1980) Some New Mathematical Methods for Variational Objective Analysis Using Splines and Cross-Validation. *Monthly Weather Review*, **108**, 36-57.

Wendelberger, J.G. (1981) The Computation of Laplacian Smoothing Splines with Examples. Technical Report#648, Dept. of Statistics, U. of Wisconsin.

Table 3. GCVPACK notation correspondence

		integer constants
n	nobs	number of observations
d	dim	dimension of polynomial space
m	m	order of derivatives of penalty
c	ncov	number of covariates
c_1	ncov1	number of covariates in **S** replicating structure of **T**
$c-c_1$	ncov2	ncov – ncov1
a	nuobs	iout[4] = number of unique obs. (*dtpss* & *dptpss*)
a^*	npsing	iout[1] = number of positive singular values
t	mkpoly(m,dim)	dimension of polynomial space
$h=t+c$	nnull	iout[3] = size of null space of Σ
$p=a+t+c$	npar	iout[2] = number of parameters
$t+c_1$	ncts1	number of columns in $[\mathbf{T}:\mathbf{S}_1]$
$p-h$	pmh	npar – nnull
$n-h$	nmh	nobs – nnull
		data and parameter vectors
$\mathbf{y}$	y	response vector
$\boldsymbol{\beta}$	beta	coefficients for covariates
$\boldsymbol{\alpha}$	alpha	coefficients for polynomial
$\boldsymbol{\delta}$	delta	coefficients for smooth
$\boldsymbol{\gamma}$	coef	coefficients for well-defined smooth basis
$\boldsymbol{\theta}$	coef	coefficients (in several forms)
		matrices
$\mathbf{X}$	des	design matrix for splined variables
$[\mathbf{T}:\mathbf{S}_1]$	ts1	polynomials and replicated covariates
$[\mathbf{T}_U:\mathbf{S}_{1U}]$	tbsb1	unique polynomials and replicated covariates
$\mathbf{S}_2$	s2	unreplicated covariates
$\{\mathbf{A}(\hat{\lambda})\}_{ii}$	adiag[i]	diagonal of hat matrix
$\{\mathbf{D}\}_{ii}$	svals[i]	singular values
Σ	sigma	penalty matrix
$\mathbf{F},\mathbf{G}$	fg & fgaux	QR decomposition of $[\mathbf{T}:\mathbf{S}_1]$
$\mathbf{E}$	sgpvt	permutation for pivoted Cholesky of Σ
$\mathbf{Q},\mathbf{R}$	qr & qraux	QR decomposition of Cholesky factor of Σ
$\{\mathbf{C}_U\}_{ii}$	c1[i]	square root of number of replicates of i'th unique $\mathbf{x}$
		double precision summaries
$\hat{\lambda}$	lamhat	dout[1] = GCV estimate of lambda
$J(f)$	penlty	dout[2] = smoothing penalty
$\Vert\mathbf{I}-\mathbf{A}(\hat{\lambda})\Vert^2$	rss	dout[3] = residual sum of squares
$tr(\mathbf{I}-\mathbf{A}(\hat{\lambda}))$	–	dout[4] = trace of $\mathbf{I}-\mathbf{A}$
$\Vert\mathbf{B}_2^T\mathbf{y}\Vert^2$	ssqrep	dout[5] = sum of squares for replication (*dtpss*)
ρ	machep	relative machine precision
τ	tau	small multiple
$\tau\rho$	minrat	machine tolerance

Documentation for Driver Routines

```
::::::::::::::
dptpss.com
::::::::::::::
      subroutine dptpss(des,lddes,nobs,dim,m,s,lds,ncov1,ncov2,y,ntbl,
     * adiag,lamlim,dout,iout,coef,svals,tbl,ldtbl,auxtbl,work,
     * lwa,iwork,liwa,job,info)
      integer lddes,nobs,dim,m,lds,ncov1,ncov2,ntbl,iout(4),ldtbl,lwa,
     * liwa,iwork(liwa),job,info
      double precision des(lddes,dim),s(lds,*),y(nobs),adiag(nobs),
     * lamlim(2),dout(4),coef(*),svals(*),tbl(ldtbl,3),
     * auxtbl(3,3),work(lwa)
c
c Purpose: determine the generalized cross validation estimate of the
c       smoothing parameter and fit model parameters for a partial thin
c       plate spline model.
c
c On Entry:
c   des(lddes,dim)       design for the variables to be splined
c   lddes                leading dimension of des as declared in the
c                        calling program
c   nobs                 number of observations
c   dim                  number of columns in des
c   m                    order of the derivatives in the penalty
c   s(lds,ncov1+ncov2)   design for the covariates
c                        first ncov1 columns contain covariates which
c                          duplicate the replication structure of des
c                        next ncov2 columns contain covariates which
c                          do not duplicate the replication structure of
c                          des
c   lds                  leading dimension of s as declared in the
c                        calling program
c   ncov1                number of covariates which duplicate the
c                        replication structure of des
c   ncov2                number of covariates which do not duplicate the
c                        replication structure of des
c   y(nobs)              response vector
c   ntbl                 number of evenly spaced values for
c                        log10(nobs*lambda) to be used in the initial
c                        grid search for lambda hat
c                        if ntbl = 0 only a golden ratio search will be
c                        done and tbl is not referenced, if ntbl > 0
c                        there will be ntbl rows returned in tbl
c   adiag(nobs)          "true" y values on entry if predictive mse is
c                        requested
c   lamlim(2)            limits on lambda hat search (in log10(nobs*
c                        lambda) scale) if user input limits are
c                        requested. if lamlim(1) = lamlim(2) then lamhat
c                        is set to (10**lamlim(1))/nobs
c   ldtbl                leading dimension of tbl as declared in the
c                        calling program
c   job                  integer with decimal expansion abc
c                        if a is nonzero then predictive mse is computed
c                           using adiag as true y
c                        if b is nonzero then user input limits on search
c                           for lambda hat are used
c                        if c is nonzero then adiag will be calculated
c On Exit:
c   des(lddes,dim)       unique rows of des
c   y(nobs)              predicted values
c   adiag(nobs)          diagonal elements of the hat matrix if requested
c   lamlim(2)            limits on lambda hat search
c                        (in log10(nobs*lambda) scale)
c   dout(4)              contains:
c                        1  lamhat   generalized cross validation
c                                    estimate of the smoothing parameter
c                        2  penlty   smoothing penalty
c                        3  rss      residual sum of squares
c                        4  tr(I-A)  trace of I - A
c   iout(4)              contains:
c                        1  npsing   number of positive singular values
c                                    if info indicates nonzero info
c                                    from dsvdc then npsing contains
c                                    info as it was returned from dsvdc
c                        2  npar     number of parameters
c                                    (npar = nuobs + nnull)
c                        3  nnull    size of the null space of sigma
c                                    (m+dim-1 choose dim)+ncov1+ncov2
c                        4  nuobs    number of unique rows in des
c   coef(npar)           coefficient estimates [beta':alpha':delta']'
c                        coef must have a dimension of at least
c                        nuobs+nnull
c   svals(npsing)        singular values, svals must have a dimension,
c                        of at least nuobs-nnull.
c                        if info indicates nonzero info in dsvdc then
c                        svals is as returned from dsvdc.
c   tbl(ldtbl,3)         column  contains
c                          1     grid of log10(nobs*lambda)
c                          2     V(lambda)
c                          3     R(lambda) if requested
c   auxtbl(3,3)          auxiliary table
c                        1st row contains:
c                            log10(nobs*lamhat), V(lamhat) and
c                            R(lamhat) if requested
c                            where lamhat is the gcv estimate of lambda
c                        2nd row contains:
c                            0, V(0) and  R(0) if requested
c                        3rd row contains:
c                            0, V(infinity) and R(infinity) if requested
c   info                 error indicator
c                          0 : successful completion
c                         -1 : log10(nobs*lamhat) <= lamlim(1)
c                              (not fatal)
c                         -2 : log10(nobs*lamhat) >= lamlim(2)
c                              (not fatal)
c                          1 : dimension error
c                          2 : error in dreps, the first ncov1 columns
c                              of s do not duplicate the replication
c                              structure of des
c                          3 : lwa (length of work) is too small
c                          4 : liwa (length of iwork) is too small
c                          5 : error in dmaket
c                          6 : sigma is rank deficient
c                       1000< info : 1000 + nonzero info returned from
c                                    dsnsm
c
c Working Storage:
c   work(lwa)            double precision work vector
c   lwa                  length of work as declared in the calling
c                        program
c                        must be at least lwa1 + lwa2 where
c                          lwa1 = (nnull-ncov2)*(nobs+nuobs+1)
c                                 +npar*(nobs+npar)
c                          lwa2 = (npar-nnull)*(npar-2*nnull+2+nobs)
c                                 +npar+nobs
c
c   iwork(liwa)          integer work vector
c   liwa                 length of the iwork as declared in the calling
```

Documentation for Driver Routines

```
c                       program
c                       must be at least 3*nobs - (nnull - ncov2)
c
c Subprograms Called Directly:
c       Gcvpack - dreps dmaket duni dmakek dctsx dsnsm
c       Linpack - dqrdc dqrsl
c       Blas    - dcopy
c       Other   - dprmut dset prmut mkpoly
c
c Subprograms Called Indirectly:
c       Gcvpack - dcrtz ddcom dgcv dsgdc dtsvdc drsap ddiag
c                 dvlop dvlop dpmse dcfcr dpdcr dvmin dvl dzdc
c       Linpack - dchdc dqrdc dqrsl dtrsl dsvdc dtrco
c       Blas    - dcopy ddot dgemv dswap
c       Other   - dcpmut dprmut dset dftkf fact mkpoly
c
::::::::::::::
dsnsm.com
::::::::::::::
      subroutine dsnsm (x,ldx,y,sigma,ldsigm,nobs,npar,nnull,adiag,
     * tau,lamlim,ntbl,dout,iout,coef,svals,tbl,ldtbl,auxtbl,
     * iwork,liwa,work,lwa,job,info)
      integer ldx,ldsigm,nobs,npar,nnull,ntbl,iout(3),ldtbl,liwa,
     * iwork(liwa),lwa,job,info
      double precision x(ldx,npar),y(nobs),sigma(ldsigm,npar),
     * adiag(nobs),tau,lamlim(2),dout(5),coef(npar),svals(*),
     * tbl(ldtbl,3),auxtbl(3,3),work(lwa)
c
c Purpose: determine the generalized cross validation estimate of the
c       smoothing parameter and fit model parameters for a penalized
c       least squares problem with a semi-norm smoothing matrix.
c
c On Entry:
c   x(ldx,npar)         design matrix
c   ldx                 leading dimension of x as declared in the
c                       calling program, must be at least max(nobs,npar)
c   y(nobs)             response vector
c   sigma(ldsigm,npar)  symmetric matrix that defines the semi-norm
c   ldsigm              leading dimension of sigma as declared
c                       in the calling program
c   nobs                number of observations
c   npar                number of parameters
c   nnull               dimension of the null space of sigma
c   adiag(nobs)         "true" y values on entry if computation of
c                       predictive mse is requested
c   lamlim(2)           limits on lambda hat search (in log10(nobs*
c                       lambda) scale) if user input limits are
c                       requested if lamlim(1) = lamlim(2) then lamhat
c                       is set to (10**lamlim(1))/nobs
c   tau                 multiplier controlling the amount of truncation
c                       if truncation is requested (try tau = 1
c                       to start then try 10 and 100)
c   ntbl                number of evenly spaced values for
c                       log10(nobs*lambda) to be used in the initial
c                       grid search for lambda hat
c                       if ntbl = 0 only a golden ratio search will be
c                       done and tbl is not referenced, if ntbl > 0
c                       there will be ntbl rows returned in tbl
c   ldtbl               leading dimension of tbl as declared in the
c                       calling program
c   job                 integer with decimal expansion abcd
c                       if a is nonzero then truncation is used
c                       if b is nonzero then predictive mse is computed
c                          using adiag as true y
c                       if c is nonzero then user input limits on search
c                          for lambda hat are used
c                       if d is nonzero then the diagonal of the hat
c                          matrix is calculated
c
c On Exit:
c   x(ldx,npar)         overwritten with many intermediate results
c   y(nobs)             predicted values
c   sigma(ldsigm,npar)  overwritten with the QR decomposition of the
c                       Cholesky factor of sigma
c   adiag(nobs)         diagonal elements of the hat matrix if requested
c   lamlim(2)           limits on lambda hat search
c                       (in log10(nobs*lambda) scale)
c   dout(5)             contains:
c                       1 lamhat   generalized cross validation
c                                  estimate of the smoothing parameter
c                       2 penlty   smoothing penalty
c                       3 rss      residual sum of squares
c                       4 tr(I-A)  trace of I - A
c                       5 truncation ratio = 1/(1+(normk/(nobs*lamhat)))
c                                  where normk = norm(R - R sub k)**2
c   iout(3)             contains:
c                       1  npsing   number of positive singular
c                                   values
c                                   if info indicates nonzero info in
c                                   dsvdc then iout(1) contains info as
c                                   it was returned from dsvdc
c                       2  npar     number of parameters
c                       3  nnull    size of the null space of sigma
c   coef(npar)          coefficient estimates
c   svals(npar-nnull)   first npsing entries contain singular values
c                       of the matrix j2
c                       if info indicates nonzero info in dsvdc then
c                       svals is as it was returned from dsvdc
c   tbl(ldtbl,3)        column  contains
c                         1     grid of log10(nobs*lambda)
c                         2     V(lambda)
c                         3     R(lambda) if requested
c   auxtbl(3,3)         auxiliary table
c                       1st row contains:
c                          log10(nobs*lamhat), V(lamhat) and
c                          R(lamhat) if requested
c                          where lamhat is the gcv estimate of lambda
c                       2nd row contains:
c                          0, V(0) and  R(0) if requested
c                       3rd row contains:
c                          0, V(infinity) and R(infinity) if requested
c   info                error indicator
c                         0 : successful completion
c                        -3 : nnull is too small (not fatal)
c                        -2 : log10(nobs*lamhat) >= lamlim(2)
c                             (not fatal)
c                        -1 : log10(nobs*lamhat) <= lamlim(1)
c                             (not fatal)
c                         1 : dimension error
c                         2 : lwa (length of work) is too small
c                         3 : liwa (length of iwork) is too small
c                         4 : error in ntbl or tau
c                        100< info <200 : 100 + nonzero info returned
c                                         from ddcom
c                        200< info <300 : 200 + nonzero info returned
c                                         from dgcv
c
c Work Arrays:
```

```
c   work(lwa)            double precision work vector
c   lwa                  length of work as declared in the calling
c                        program
c                        must be at least
c                        (npar-nnull)*(npar-2*nnull+2+nobs)+npar+nobs
c   iwork(liwa)          integer work vector
c   liwa                 length of iwork as declared in the calling
c                        program
c                        must be at least 2*npar - nnull
c
c Subprograms Called Directly:
c       Gcvpack - ddcom dgcv
c
c Subprograms Called Indirectly:
c       Gcvpack - dcrtz dsgdc dcfcr drsap dvlop dtsvdc
c                 dpmse dvmin dvl dzdc dpdcr ddiag
c       Linpack - dchdc dqrdc dqrsl dtrsl dsvdc dtrco
c       Blas    - dcopy ddot dgemv dswap
c       Other   - dcpmut dprmut dset
c
::::::::::::::
dtpss.com
::::::::::::::
      subroutine dtpss(des,lddes,nobs,dim,m,s,lds,ncov,y,ntbl,adiag,
     * lamlim,dout,iout,coef,svals,tbl,ldtbl,auxtbl,work,lwa,
     * iwork,liwa,job,info)
      integer lddes,nobs,dim,m,lds,ncov,ntbl,iout(4),ldtbl,lwa,
     * liwa,iwork(liwa),job,info
      double precision des(lddes,dim),s(lds,*),y(nobs),
     * adiag(nobs),lamlim(2),dout(5),coef(*),svals(*),
     * tbl(ldtbl,3),auxtbl(3,3),work(lwa)
c
c Purpose: determine the generalized cross validation estimate of the
c       smoothing parameter and fit model parameters for a thin plate
c       smoothing spline.
c
c On Entry:
c   des(lddes,dim)       design for the variables to be splined
c   lddes                leading dimension of des as declared in calling
c                        program
c   nobs                 number of observations
c   dim                  number of columns in des
c   m                    order of the derivatives in the penalty
c   s(lds,ncov)          design for the covariates. The covariates
c                        must duplicate the replication structure of des.
c                        See dptpss to handle covariates which do not.
c   lds                  leading dimension of s as declared in calling
c                        program
c   ncov                 number of covariates
c   y(nobs)              response vector
c   ntbl                 number of evenly spaced values for
c                        log10(nobs*lambda) to be used in the initial
c                        grid search for lambda hat
c                        if ntbl = 0 only a golden ratio search will be
c                        done and tbl is not referenced, if ntbl > 0
c                        there will be ntbl rows returned in tbl
c   adiag(nobs)          "true" y values on entry if predictive mse is
c                        requested
c   lamlim(2)            limits on lambda hat search (in log10(nobs*
c                        lambda) scale) if user input limits are
c                        requested if lamlim(1) = lamlim(2) then lamhat
c                        is set to (10**lamlim(1))/nobs
c   ldtbl                leading dimension of tbl as declared in the
c                        calling program
```

```
c   job                  integer with decimal expansion abdc
c                        if a is nonzero then predictive mse is computed
c                           using adiag as true y
c                        if b is nonzero then user input limits on search
c                           for lambda hat are used
c                        if c is nonzero then adiag will be calculated
c                        if d is nonzero then there are replicates in the
c                           design
c
c On Exit:
c   des(lddes,dim)       sorted unique rows of des if job indicates that
c                        there are replicates otherwise not changed
c   s(lds,ncov)          unique rows of s sorted to correspond to des
c   y(nobs)              predicted values
c   adiag(nobs)          diagonal elements of the hat matrix if requested
c   lamlim(2)            limits on lambda hat search
c                        (in log10(nobs*lambda) scale)
c   dout(5)              contains:
c                        1  lamhat    generalized cross validation
c                                     estimate of the smoothing parameter
c                        2  penlty    smoothing penalty
c                        3  rss       residual sum of squares
c                        4  tr(I-A)   trace of I - A
c                        5  ssqrep    sum of squares for replication
c   iout(4)              contains:
c                        1  npsing    number of positive singular
c                                     values (npsing = nuobs - ncts).
c                                     if info indicates nonzero info in
c                                     dsvdc then npsing contains info as
c                                     it was returned from dsvdc.
c                        2  npar      number of parameters
c                                     (npar = nuobs + ncts)
c                        3  ncts      dimension of the polynomial space
c                                     plus ncov
c                                     ((m+dim-1 choose dim) + ncov)
c                        4  nuobs     number of unique rows in des
c   coef(npar)           coefficient estimates [beta':alpha':delta']'
c                        coef must have a dimension of at least nuobs+
c                        ncts
c   svals(npar-nnull)    singular values of the matrix j2 if info = 0
c                        if info indicates nonzero info from dsvdc then
c                        svals is as it was returned from dsvdc.
c   tbl(ldtbl,3)         column  contains
c                          1      grid of log10(nobs*lambda)
c                          2      V(lambda)
c                          3      R(lambda) if requested
c   auxtbl(3,3)          auxiliary table
c                        1st row contains:
c                           log10(nobs*lamhat), V(lamhat) and
c                           R(lamhat) if requested
c                           where lamhat is the gcv estimate of lambda
c                        2nd row contains:
c                           0, V(0) and  R(0) if requested
c                        3rd row contains:
c                           0, V(infinity) and R(infinity) if requested
c   info                 error indicator
c                          0 : successful completion
c                         -1 : log10(nobs*lamhat) <= lamlim(1)
c                              (not fatal)
c                         -2 : log10(nobs*lamhat) >= lamlim(2)
c                              (not fatal)
c                          1 : dimension error
c                          2 : error in dreps, covariates do not
c                              duplicate the replication structure of des
```

```
c                         3 : lwa (length of work) is too small
c                         4 : liwa (length of iwork) is too small
c                         10 < info < 20 : 10 + nonzero info returned
c                                          from dsetup
c                         100< info <200 : 100 + nonzero info returned
c                                          from dsgdcl
c                         200< info <300 : 200 + nonzero info returned
c                                          from dgcvl
c
c Work Arrays:
c   work(lwa)           double precision work vector
c   lwa                 length of work as declared in the calling
c                       program
c                       Must be at least nuobs(2+ncts+nuobs)+nobs
c   iwork(liwa)         integer work vector
c   liwa                length of iwork as declared in the calling
c                       program
c                       Must be at least 2*nobs + nuobs - ncts
c
c Subprograms Called Directly:
c       Gcvpack - dreps duni dsuy dsetup dsgdcl dgcvl
c       Other   - dprmut mkpoly
c
c Subprograms Called Indirectly:
c       Gcvpack - dcfcrl drsap dvlop dsvtc dpdcr dpmse
c                 dvmin dvl dmaket dmakek ddiag
c       Linpack - dchdc dqrdc dqrsl dtrsl dsvdc
c       Blas    - ddot dcopy dgemv
c       Other   - dprmut dset dftkf fact mkpoly
c
```

Major Changes in Second Release

This is a listing of the major changes to Gcvpack for the second release. Users who call only the drivers *dptpss.f* and *dtpss.f* will not have to make any modifications to their calling routines. Users who call the driver *dsnsm.f* must add the argument *factor* to the calling sequence (start with a value of 1.0) and must declare *dout* to be of length 5 rather than 4. All users must note that the search for lambda hat is now done in the log base 10 scale rather than the natural log scale. This will change the output from all drivers. Users who call any other routines must be very careful to **note the changes in calling sequences** listed below. We do not expect to release another version of Gcvpack after this one. **You must replace all of the release 1 routines with release 2 routines to insure correct computation.** The set of necessary linpack routines has not changed.

affected routines	**description**
ddsdc.f †, *ddcom.f* , *dcrtz.f* **, *dzdc.f* **	*ddsdc.f* is replaced with *dcrtz.f* , *dzdc.f* .
ddcom.f , *dzdc.f*	Nonzero *info* and *svals* from *dsvdc* are now returned in *npsing* and *svals* .
dsgdc1.f	Nonzero *info* and *svals* from *dsvdc* are now returned in *p* and *svals* .
dptpss.f , *dsnsm.f* , *dtpss.f*	Nonzero *info* and *svals* from *dsvdc* are now returned in *iout* (1) and *svals* .
dpdcr.f *, *ddiag.f* **, *dgcv.f* , *dgcv1.f*	Calculation of the diagonal of the hat matrix has been removed from *dpdcr.f* and is now done in the new routine *ddiag.f* .
dtpss.f , *dsnsm.f* , *dptpss.f* , *dgcv1.f*, *dgcv.f*	*Iout* and *dout* are now assigned as early as possible to return more information when *info* is positive.
dgcv.f , *dgcv1.f*, *dpmse.f* , *dptpss.f* , *dsnsm.f* , *dtpss.f* , *dvl.f* , *dvlop.f*	Lambda values are expressed as log base 10 rather than natural log.
dsuy.f *, *dreps.f* *, *dtpss.f* , *dptpss.f*	The argument *dfrep* is removed from the calling sequences of *dsuy.f* and *dreps.f* .
dvlop.f *, *dgcv1.f*, *dgcv.f*	The argument *npar* is removed from the calling sequence of *dvlop.f* .
dcfcr.f *, *dgcv.f*	The argument *fgaux* is removed from the calling sequence of *dcfcr.f* .
ddcom.f , *dcrtz.f* *	The argument *lwa* is removed from the calling sequence of *dcrtz.f* .
ddcom.f *, *dsnsm.f* *, *dzdc.f* *	The argument *factor* is added to allow control over the amount of truncation.
dsnsm.f	The argument *dout* must be declared with length 5 rather than length 4. The truncation ratio is now returned in *dout* (5).
ddcom.f *, *dzdc.f* *, *dtsvdc.f* *	The Frobenius norm of $R - R_k$ is now returned as an argument.
dcfcr.f , *dcfcr1.f*, *dgemv.f* **, *ddiag.f* , *dpdcr.f* , *dpmse.f* , *dpred.f* , *drsap.f*	The routine *dgemv.f* is called to perform matrix multiplication. (*Dgemv.f* is from the extended BLAS)
testtpss.out , *testptpss.out*	Changes due to the switch from natural log to log base 10 and reformatting of output.

fort. 8†, *testtpss.in* **, *testptpss.in* **, *testtpss.f* , *testptpss.f*	The input files *testtpss.in* and *testptpss.in* replace *fort.* 8 and are now read as standard input.
inteqn.f **, *inteqn.in* **, *inteqn.out* **, *mktpar.f* **, *mkxys.f* **	Routines and files used for new integral equation test driver *inteqn.f* .

* Calling sequence changed, ** New routine or file, † Deleted routine or file

Non-zero Error Codes from Driver Routines

Below are all the error codes which may arise during use of the drivers *dptpss*, *dsnsm* and *dtpss*. Refer to Description section and code for further details.

Non-zero Error Codes from *dptpss*

Code	Occured In	Meaning
-2	*dvmin*	$\log(nobs*lamhat) \geq lamlim(2)$ (not fatal)
-1	*dvmin*	$\log(nobs*lamhat) \leq lamlim(1)$ (not fatal), will occur if limits on lambda hat are input by user and are equal.
1	*dptpss*	dimension error ($nobs \leq 0$ or $m \leq 0$ or $dim \leq 0$ or $ntbl < 0$ or $ntbl > ldtbl$ or $2*m - dim \leq 0$)
2	*dptpss*	error in *dreps*, the first *ncov* 1 columns of *s* do not duplicate the replication structure of *des*
3	*dptpss*	*lwa* (length of *work*) $< ncts1*(nobs + nuobs + 1) + npar*(nobs + npar + 1) + (npar - nnull)*(npar - 2*nnull + 2 + nobs) + npar + nobs$
4	*dptpss*	*liwa* (length of *iwork*) $< 3*nobs - ncts1$
5	*dptpss*	error in *dmaket* (error in creation of **T**)
6	*dptpss*	sigma is rank deficient
1001*	*dsnsm*	dimension error ($nobs \leq 0$ or $npar \leq 0$ or $nnull \leq 0$ or $(npar - nnull) \leq 0$)
1002*	*dsnsm*	*lwa* (length of *work*) $< (npar - nnull)*(npar - 2*nnull + 2 + nobs) + npar + nobs$
1003*	*dsnsm*	*liwa* (length of *iwork*) $< 2*npar - nnull$
1004*	*dsnsm*	error in *ntbl* ($ntbl < 0$ or $ntbl > ldtbl$)
1101*	*ddcom*	dimension error ($nobs \leq 0$ or $npar \leq 0$ or $nnull \leq 0$ or $(npar - nnull) \leq 0$ or $ldx < nobs$ or $ldx < npar$)
1102*	*ddcom*	*ldcaux* (length of *dcaux*) $< (npar - nnull)^2 + 2*npar - nnull$
1103*	*ddcom*	*lwa* (length of *work*) $< (npar - nnull)*(nobs - nnull + 1) + nobs$
1104*	*ddcom*	*liwa* (length of *iwork*) $< npar - nnull$
1111*	*dsgdc*	calculated *nnull* is smaller than input *nnull*
1121	*dcrtz*	error in *dtrsl*, **R** is singular
1131*	*dzdc*	*lwa* (length of *work*) $< (npar - nnull)*(nobs - nnull) + npar - nnull + nobs$
1132	*dzdc*	transpose of $\mathbf{J}_2$ is necessary ($npar > nobs$) but $npar > ldx$
1133*	*dzdc*	failure to converge in *dsvdc* called from *dtsvdc* (using $\mathbf{J}_2^T$)
1134	*dzdc*	failure to converge in *dsvdc* (using $\mathbf{J}_2^T$), this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine *dsvdc*
1135*	*dzdc*	failure to converge in *dsvdc* called from *dtsvdc*

1136	*dzdc*	failure to converge in *dsvdc*, this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine *dsvdc*
1201*	*dgcv*	dimension error ($nobs \le 0$ or $npar \le 0$ or $nnull \le 0$ or $(npar - nnull) \le 0$)
1202*	*dgcv*	error in *ntbl* ($ntbl < 0$ or $ntbl > ldtbl$)
1203*	*dgcv*	*ldcaux* (length of *dcaux*) $< (npar - nnull)^2 + 2*npar - nnull$
1204*	*dgcv*	*lwa* (length of *work*) $< (npar - nnull) + nobs$
1205	*dgcv*	$lamlim(1) > lamlim(2)$
1211	*dvlop*	$svals(1) = 0.0d0$
1212*	*dvlop*	*npsing* is incorrect
1213*	*dvlop*	$lamlim(1) > lamlim(2)$
1221	*dcfcr*	error in *dtrco*, **G** is singular
1222	*dcfcr*	error in *dtrsl*, **R** is singular

*Should not occur in normal circumstances

Non-zero Error Codes from *dsnsm*

Code	Occured In	Meaning
-3	*dsgdc*	calculated *nnull* is larger than input *nnull* (not fatal)
-2	*dvmin*	$\log(nobs*lamhat) \ge lamlim(2)$ (not fatal)
-1	*dvmin*	$\log(nobs*lamhat) \le lamlim(1)$ (not fatal), will occur if limits on lambda hat are input by user and are equal.
1	*dsnsm*	dimension error ($nobs \le 0$ or $npar \le 0$ or $nnull \le 0$ or $(npar - nnull) \le 0$)
2	*dsnsm*	*lwa* (length of *work*) $< (npar - nnull)* (npar - 2*nnull + 2 + nobs) + npar + nobs$
3	*dsnsm*	*liwa* (length of *iwork*) $< 2*npar - nnull$
4	*dsnsm*	error in *ntbl* ($ntbl < 0$ or $ntbl > ldtbl$) or $\tau<0$
101	*ddcom*	$\tau<0$ or dimension error ($nobs \le 0$ or $npar \le 0$ or $nnull \le 0$ or $(npar - nnull) \le 0$ or $ldx < nobs$ or $ldx < npar$)
102*	*ddcom*	*ldcaux* (length of *dcaux*) $< (npar - nnull)^2 + 2*npar - nnull$
103*	*ddcom*	*lwa* (length of *work*) $< (npar - nnull)* (nobs - nnull + 1) + nobs$
104*	*ddcom*	*liwa* (length of *iwork*) $< npar - nnull$
111	*dsgdc*	calculated *nnull* is smaller than input *nnull*
121	*dcrtz*	error in *dtrsl*, **R** is singular
131*	*dzdc*	*lwa* (length of *work*) $< (npar - nnull)* (nobs - nnull) + npar - nnull + nobs$
132	*dzdc*	$\tau<0$
133	*dzdc*	transpose of $\mathbf{J}_2$ is necessary ($npar > nobs$) but $npar > ldx$
134	*dzdc*	failure to converge in *dsvdc* called from *dtsvdc* (using $\mathbf{J}_2^T$), this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine *dsvdc* or by increasing the value of the argument τ to *dsnsm* (see discussion in TR775 (rev.))
135	*dzdc*	failure to converge in *dsvdc* (using $\mathbf{J}_2^T$), this error can usually be cured by using the truncation option or by increasing the parameter MAXIT in the LINPACK routine *dsvdc*
136	*dzdc*	failure to converge in *dsvdc* called from *dtsvdc*, this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine *dsvdc* or by increasing the value of the argument τ to *dsnsm* (see discussion in TR775 (rev.))
137	*dzdc*	failure to converge in *dsvdc* (using $\mathbf{J}_2^T$), this error can usually be cured by using the truncation option or by increasing the parameter MAXIT in the LINPACK routine *dsvdc*
201*	*dgcv*	dimension error ($nobs \le 0$ or $npar \le 0$ or $nnull \le 0$ or $(npar - nnull) \le 0$)
202*	*dgcv*	error in *ntbl* ($ntbl < 0$ or $ntbl > ldtbl$)
203*	*dgcv*	*ldcaux* (length of *dcaux*) $< (npar - nnull)^2 + 2*npar - nnull$

204*	*dgcv*	*lwa* (length of *work*) < (*npar* - *nnull*) + *nobs*
205	*dgcv*	*lamlim* (1) > *lamlim* (2)
211	*dvlop*	*svals* (1) = 0.0d0
212	*dvlop*	*npsing* is incorrect
213*	*dvlop*	*lamlim* (1) > *lamlim* (2)
221	*dcfcr*	error in *dtrco*, **G** is singular
222	*dcfcr*	error in *dtrsl*, **R** is singular

*Should not occur in normal circumstances

Non-zero Error Codes from *dtpss*

Code	Occured In	Meaning
-2	*dvmin*	log(*nobs* **lamhat*) $\geq$ *lamlim* (2) (not fatal)
-1	*dvmin*	log(*nobs* **lamhat*) $\leq$ *lamlim* (1) (not fatal), will occur if limits on lambda hat are input by user and are equal.
1	*dtpss*	dimension error (*nobs* ≤ 0 or *m* ≤ 0 or *dim* ≤ 0 or *ntbl* < 0 or *ntbl* > *ldtbl* or 2**m* - *dim* ≤ 0)
2	*dtpss*	error in *dreps*, covariates do not duplicate the replication structure of *des*
3	*dtpss*	*lwa* (length of *work*) < *nuobs* *(2 + ncts + *nuobs*) + *nobs*
4	*dtpss*	*liwa* (length of *iwork*) < 2**nobs* + *nuobs* - ncts
11	*dsetup*	error in *dmaket* (error in creation of **T**)
101*	*dsgdc1*	*lwa* < 2**npar*
102	*dsgdc1*	$\mathbf{F}_2^T\mathbf{K}\mathbf{F}_2$ is not of full rank
103	*dsgdc1*	failure to converge in *dsvdc*, this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine *dsvdc*
201*	*dgcv1*	dimension error (*nuobs* ≤ 0 or *ncts* 1 ≤ 0 or *nuobs* - *ncts* 1 ≤ 0)
202*	*dgcv1*	error in *ntbl* (*ntbl* < 0 or *ntbl* > *ldtbl*)
203*	*dgcv1*	*lwa* (length of *work*) < *nuobs* - *ncts* 1 + *nobs*
204	*dgcv1*	*lamlim* (1) > *lamlim* (2)
211	*dvlop*	*svals* (1) = 0.0d0
212*	*dvlop*	*npsing* is incorrect
213	*dvlop*	*lamlim* (1) > *lamlim* (2)
221	*dcfcr1*	error in *dtrco*, **G** is singular

*Should not occur in normal circumstances

Non-zero Error Codes from *dpred*

Code	Occured In	Meaning
1	*dpred*	dimension error (*nobs* ≤ 0 or nct ≤ 0 or *m* ≤ 0 or *dim* ≤ 0)
2	*dpred*	*npar* $\neq$ ndesb + nct + *ncov* 1 + ncov2
3	*dpred*	*lwa* (length of *work*) < npred * (nct + ndesb)
4	*dpred*	error in *dmaket* (error in creation of **T**)