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) 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 \( \lambda \) in the ridge regression problem which we write as

\[
\min_{\gamma} \frac{1}{n} \| y - X\gamma \|_2^2 + \lambda \gamma^T \gamma
\]

where \( \gamma \) is a \( p \)-dimensional parameter vector, \( y \) is an \( n \)-dimensional response vector and \( X \) is an \( n \times p \) design matrix.

For any positive \( \lambda \), an optimal \( \gamma \), can be easily calculated. Unfortunately, this leaves the question of which value of \( \lambda \) 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 \( \lambda \), where

\[
V(\lambda) = \frac{1}{n} \| (I - A(\lambda)) y \|_2^2
\]

and \( A(\lambda) \) is the \( n \times n \) “hat” matrix of the ridge regression

\[
A(\lambda) = X(X^TX + n\lambda I)^{-1}X^T.
\]

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

\[
X = UDV^T
\]

where \( U \) is \( n \times p \) with orthonormal columns, \( V \) is \( p \times p \) and orthogonal, and \( D \) 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 \( X^TX \). The “hat” matrix can then be written as
\[ A(\lambda) = UD^2(D^2 + n\lambda I)^{-1}U^T, \]

and using
\[ z = U^T y \]
we can write
\[ V(\lambda) = \frac{n}{\sqrt{n - p + \sum_{j=1}^{p} \frac{n \lambda}{d_j^2 + n \lambda}}} \left( \frac{\|y\|^2 - \|z\|^2}{\sum_{j=1}^{p} \frac{n \lambda}{d_j^2 + n \lambda}} \right)^2. \]

(1.1)

Once the SVD of \( X \) is computed, it is trivial to evaluate \( V(\lambda) \) for a wide range of values of \( \lambda \) and determine the optimum value of \( \lambda \). Equation (1.1) indicates that, for most problems, \( d_j^2 \leq n\lambda \leq d_j^{-2} \). After an optimal \( \lambda \) is chosen, the corresponding \( \gamma \) is calculated as
\[ \gamma = V(D^2 + n\lambda I)^{-1}Dz. \]

(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 \( x \) but the software is designed for the general case. The data model for TPSS is
\[ y_i = f(x_i) + e_i, \quad i = 1, \ldots, n, \]
where the \((x_i, y_i), i = 1, 2, \ldots, n, \) are observed data, \( f \) is an unknown function which is assumed to be reasonably smooth, and the \( e_i, i = 1, 2, \ldots, n, \) 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,
\[ J_2(f) = \int \int \left( \frac{\partial^2 f}{\partial x_1^2} + 2 \frac{\partial^2 f}{\partial x_1 \partial x_2} + \frac{\partial^2 f}{\partial x_2^2} \right)^2 \, dx_1 dx_2. \]

To allow generalizations, the software uses a smoothness penalty defined by the partial derivatives of total order \( m \) as
\[ J_m(f) = \int \int \sum_{i=0}^{m} \left( \frac{\partial^m f}{\partial x_1^i \partial x_2^{m-i}} \right)^2 \, dx_1 dx_2. \]

In \( d \) dimensions,
\[ J_m(f) = \int \int \sum_{i=0}^{m} \frac{m!}{\alpha_1! \cdots \alpha_d!} \left( \frac{\partial^m f}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}} \right)^2 \, dx_1 \cdots dx_d. \]

with the sum within the integral over \( \alpha_i = m \). In general, one must have \( 2m - d > 0 \) with \( d \) the dimension of \( x \). Using this smoothness penalty, the TPSS estimate \( f_\lambda \) of \( f \) is the minimizer of
\[ S_\lambda(f) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda J_m(f). \]

(2.1)

From Duchon (1976), the minimizer \( f_\lambda \) of (2.1) can be represented as
\[ f_\lambda(x) = \sum_{i=1}^{t} \beta_i \phi_i(x) + \sum_{i=1}^{n} \delta_i E_m(x - x_i) \]

(2.2)

where
\[ E_m(t) = (-1)^m 2^{1-2m} \pi^{-1} ((m-1)!)^{-1} \|t\|^{2m-2} \ln(\|t\|) \]
and \( t \) is the dimension of the space of polynomials on two variables of total order at most \( m-1 \),
\[ t = \left\lfloor \frac{m+1}{2} \right\rfloor. \]

A basis for this space is
\[ \phi_1(x) = 1 \]
\[ \phi_2(x) = x_1 \]
\[ \phi_3(x) = x_2 \]
\[ \phi_4(x) = x_1^2 \]
\[ \phi_5(x) = x_1 x_2 \]
\[ \cdots \]
\[ \phi_t(x) = x_2^{m-1}. \]

The general definition of \( E_m \), which depends on the dimension, \( d \), of the independent variables \( x \) is
\[ E_m(t) = \begin{cases} a_{m,d} \|t\|^{(2m-d)} \ln(\|t\|), & d \text{ even} \\ a_{m,d} \|t\|^{(2m-d)}, & d \text{ odd} \end{cases}, \]

with
\[
\delta = \mathbf{F}_2 \zeta
\]

where \( \zeta \) has dimension \( n-t \). Using
\[
\mathbf{w}_1 = \mathbf{F}_1 \mathbf{y}
\]
\[
\mathbf{w}_2 = \mathbf{F}_2 \mathbf{y}
\]
the objective function of the optimization becomes
\[
\begin{align*}
S_\lambda(\beta, \delta) &= \frac{1}{n} \| \mathbf{y} - \mathbf{T} \beta - \mathbf{K} \delta \|^2 + \lambda \delta^T \mathbf{K} \delta \\
&\quad - \frac{1}{n} \| \mathbf{F}_1^T(\mathbf{T} \beta - \mathbf{K} \delta) \|^2 + \lambda \delta^T \mathbf{K} \delta \\
&\quad - \frac{1}{n} \| \mathbf{w}_1 - \mathbf{G}_1 \beta - \mathbf{F}_1^T \mathbf{K} \mathbf{F}_2 \zeta \|^2 \\
&\quad + \frac{1}{n} \| \mathbf{w}_2 - \mathbf{F}_2 \mathbf{K} \mathbf{F}_2 \zeta \|^2 + \lambda \zeta^T \mathbf{F}_2^T \mathbf{K} \mathbf{F}_2 \zeta.
\end{align*}
\]

(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
\[
\mathbf{G}_1 \beta = \mathbf{w}_1 - \mathbf{F}_1^T \mathbf{K} \mathbf{F}_2 \zeta
\]

for \( \beta \). 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 \( \mathbf{F}_1^T \mathbf{K} \mathbf{F}_2 \) is positive definite to form the Cholesky decomposition (Dongarra et al., 1979, chapter 8)
\[
\mathbf{F}_1^T \mathbf{K} \mathbf{F}_2 = \mathbf{L}^T \mathbf{L}
\]
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 \( \mathbf{F}_1^T \mathbf{K} \mathbf{F}_2 \). 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 \( \mathbf{F}_1^T \mathbf{K} \mathbf{F}_2 \) 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 I 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
\[
\gamma = \mathbf{L} \zeta
\]
and the last two terms of \( S_\lambda(\beta, \delta) \) in (2.6) can be written as

\[
\begin{align*}
\mathbf{w}_1 &= \mathbf{F}_1 \mathbf{y} \\
\mathbf{w}_2 &= \mathbf{F}_2 \mathbf{y}
\end{align*}
\]

The objective function of the optimization becomes
\[
\begin{align*}
S_\lambda(\beta, \delta) &= \frac{1}{n} \| \mathbf{y} - \mathbf{T} \beta - \mathbf{K} \delta \|^2 + \lambda \delta^T \mathbf{K} \delta \\
&\quad - \frac{1}{n} \| \mathbf{F}_1^T(\mathbf{T} \beta - \mathbf{K} \delta) \|^2 + \lambda \delta^T \mathbf{K} \delta \\
&\quad - \frac{1}{n} \| \mathbf{w}_1 - \mathbf{G}_1 \beta - \mathbf{F}_1^T \mathbf{K} \mathbf{F}_2 \zeta \|^2 \\
&\quad + \frac{1}{n} \| \mathbf{w}_2 - \mathbf{F}_2 \mathbf{K} \mathbf{F}_2 \zeta \|^2 + \lambda \zeta^T \mathbf{F}_2^T \mathbf{K} \mathbf{F}_2 \zeta.
\end{align*}
\]

(2.6)
\[
\frac{1}{n} \| w_2 - LL^T y \|^2 + \lambda y^T y.
\]

This has the same form as the ridge regression problem with solution
\[
\gamma = (LL^T + n\lambda I)^{-1}Lw_2.
\]

We take a SVD of \( L^T \)

\[
L^T = UDV^T
\]

and write the estimate as
\[
\gamma = V(D^2 + n\lambda I)^{-1}DU^T w_2
\]

and the "hat" matrix as
\[
A(\lambda) = F_1 F_1^T + F_2 U D^2 (D^2 + n\lambda I)^{-1} U F_2^T
\]

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

As in the ridge regression case, we use \( z = U^T w_2 \)

to write
\[
V(\lambda) = \frac{n}{\sum_{j=1}^{n} \left( \frac{n\lambda}{d_j^2 + n\lambda} \right) z_j^2}.
\]

The actual calculation of the parameter \( \delta_0 \) corresponding to the \( E_m \) 's is performed as
\[
\delta_0 = F_2 U (D^2 + n\lambda I)^{-1} z = F_2 U (D^2 + n\lambda I)^{-1} U F_2^T y.
\]

**Replanted x values:** Replicates of \( x \) values introduce some minor complications since we must define only one \( \delta_i \) corresponding to each unique \( x \) position. The best way to handle this is to pre-process the data by sorting the \( x \) values to determine the unique \( x \) values and the number of replicates of each value. Let \( k \) be the number of unique \( x \) values. We can express the objective function optimized by \( \delta_0 \) and \( \beta_0 \) as
\[
S_\lambda(\beta, \delta) = \frac{1}{n} \| y - T\beta - K\delta \|^2 + \lambda \delta^T K_U \delta
\]
subject to the condition
\[
T^T \delta = 0
\]

where \( T \) and \( K \) are of size \( n \times 1 \) and \( n \times k \) respectively, while \( T_U \) and \( K_U \) are of size \( k \times 1 \) and \( k \times k \) respectively. These matrices are related by
\[
T = MT_U
\]
\[
K = MK_U
\]

where \( M \) 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 \( x \) that corresponds to that observation.

If we take a QR decomposition of \( M \) as
\[
M = BC = [B_1; B_2] \begin{bmatrix}
C_1 \\
0
\end{bmatrix} = B_1 C_1
\]

and pre-multiply all the vectors in the first term of (3.1) by \( B \), (3.1) divides into
\[
S_\lambda(\beta, \delta) = \frac{1}{n} \| B_1^T y \|^2 + \frac{1}{n} \| B_1^T y - C_1 T_U \beta - C_1 K_U \delta \|^2
\]

\[
+ \lambda \delta^T K_U \delta
\]

In practice, it is not necessary to explicitly form \( M \) and take its QR decomposition since \( C_1 \) is diagonal with \( c_i \) being the square root of the number of replicates of the \( i \) 'th unique \( x \). The elements of the vector \( C_1^{-1} B_1^T y \) are the means of the \( y \)'s at the corresponding unique \( x \)'s. Further, \( \| B_1^T y \|^2 \) is the sum of squares due to replication.

With this information available we can write
\[
\omega = C_1^T \delta
\]

to produce
\[
S_\lambda(\beta, \omega) = \frac{1}{n} \| B_1^T y \|^2 + \frac{1}{n} \| B_1^T y - C_1 T_U \beta - C_1 K_U C_1^T \omega \|^2
\]

\[
+ \lambda \omega^T C_1 K_U C_1^T \omega
\]

and proceed as in the case with no replications using \( C_1 T_U \) in place of \( T \), and \( C_1 K_U C_1^T \) in place of \( K \). That is, take a QR decomposition
\[
C_1 T_U = FG = [F_1; F_2] \begin{bmatrix}
G_1 \\
0
\end{bmatrix} = F_1 G_1
\]

and form \( F_2^T C_1 K_U C_1^T F_2 \) which then determines the Cholesky decomposition
\[
F_2^T C_1 K_U C_1^T F_2 = L^T L.
\]

A SVD of \( L^T \) as
\[
L^T = UDV^T
\]

and the product
\[
w_2 = F_2^T B_1^T y
\]

allows us to write
Given the value of \( \lambda \), the calculation of \( \delta_\lambda \) and \( \beta_\lambda \) follow as in the no-replicates case. That is,

\[
\delta_\lambda = C^T F_2 U (D^2 + n \lambda I)^{-1} U^T F_2^T B^T y
\]

and \( \beta_\lambda \) is the solution of

\[
G_1 \beta_\lambda = F_1^T (B^T y - C_1 K_U \delta_\lambda).
\]

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

\[
y_i = f(x_i) + \sum_{j=1}^{c} \alpha_j \psi_j(x_i, s_i) + \epsilon_i \tag{4.1}
\]

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

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

and it is known that the minimizing \( f_\lambda \) has a representation of the form (2.2). Let \( S \) be the \( n \times c \) matrix with \( ij \)th entry \( \psi_j(x_i, s_i) \). The matrix \( [T:S] \) must be of full column rank. The objective function for a fixed \( \lambda \) becomes

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

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

\[
[T:S] = FG = [F_1:F_2] \begin{bmatrix} G_1 \cr 0 \end{bmatrix} = F_1 G_1
\]

so \( V(\lambda) \) is calculated as in (2.8) with all summations running to \( n - t - c \). That is, after the Cholesky decomposition of \( F_2^T K F_2 \) and the SVD of the transpose of the Cholesky factor, we have

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

The calculation of \( \lambda_\lambda \) and \( \lambda_\delta \) proceeds as in the basic TPSS case. With these available, we solve for \( \alpha_\lambda \) and \( \beta_\lambda \) simultaneously. In other words, we have simply replaced \( T \beta \) in (2.6) by

\[
[T:S] \begin{bmatrix} \beta \\ \alpha \end{bmatrix} = 0.
\]

It can be shown that the implied constraint \( S^T \delta = 0 \) does not change the solution.

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

\[
T = MT_U,
\]

\[
K = MK_U,
\]

\[
S = MS_U.
\]

Taking the QR decomposition of \( M \) as \( M = BC \), we then take a QR decomposition of \( C_1[T_U:S_U] \) and proceed as above.

If there are columns of \( S \) which do not follow the replication pattern of the design, we need a more general approach. The covariate matrix is divided into \( S = [S_1:S_2] \) in which the columns of \( S_1 \) have the same replication structure as the design points \( x \), \( i = 1, \cdots, n \). We have an indicator matrix \( M \) for which

\[
[T:S_1 : K] = M [T_U : S_U : K_U],
\]

and a QR decomposition of \( M = BC \) as above. However, we cannot easily reduce the objective function to a form such as (3.2) by premultiplying by \( B \), as \( B_2 S_2 \) is not annihilated. Instead we choose to take a QR decomposition of

\[
C_1[T_U:S_U] = FG = [F_1:F_2] \begin{bmatrix} G_1 \\ 0 \end{bmatrix} = F_1 G_1
\]

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

\[
\theta = \begin{bmatrix} \beta \\ \alpha \\ \delta \end{bmatrix},
\]

and the design matrix
\[ X = [T:S_1:S_2:KC_1^T F_2] \]

The penalty becomes \( \theta^T \Sigma \theta \), with

\[
\Sigma = \begin{bmatrix}
0 & 0 \\
0 & F_2^T C_1 K_2 C_2^T F_2
\end{bmatrix}
\]

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, \( X \), is general but the penalty term, \( \gamma \), has a special form so we can streamline the calculations. In the TPSS cases, the penalty term, \( \delta^T K \delta \), subject to \( T^T \delta = 0 \), is more general but the design matrix, \( [T:K] \), 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

\[
S_\lambda(\theta) = \frac{1}{n} \| y - X \theta \|^2 + \lambda \theta^T \Sigma \theta
\]

(5.1)

where \( \theta \) is a \( p \)-dimensional parameter vector, \( y \) is an \( n \)-dimensional response vector, \( X \) is an \( n \times p \) design matrix, and \( \Sigma \) 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 \( \{ \psi_i \} \) of (4.1) which fits in the context of (5.1) is described in Shiu, Wahba, and Johnson (1985). Other special cases included splines and vector splines on the sphere (Wahba (1981), Wahba (1982a, 1982b)) 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 GLM 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 \( \lambda \) 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 \( \Sigma \). That is, we must describe the set of \( \theta \)'s for which

\[ \theta^T \Sigma \theta = 0 \]

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

\[ E^T \Sigma E = L^T L \]

where \( E \) is a \( p \times p \) permutation matrix and \( L \) is \( (p-h) \times p \) with zeros below the main diagonal. The conditioning of \( L \) is evaluated to ensure that \( L \) actually has computational rank \( p-h \). If \( L \) is rank deficient, we increase \( h \) until the resulting \( (p-h) \times p \) matrix \( L \) 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 \( \Sigma \) is smaller than expected. As described in Appendix 1, the technique of increasing \( h \) until \( L \) is of full row rank is incompatible with the partial spline code as written here.

A QR decomposition of \( L^T \) as

\[ L^T = QR = [Q_1:Q_2] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1 \]

provides the \( h \times p \) matrix \( Q_2 \) which is an orthogonal basis for the null space of the semi-norm defined by \( \Sigma \). We can now transform to parameters \( \gamma \) and \( \beta \) of dimension \( p-h \) and \( h \), respectively, as

\[
\begin{bmatrix} \beta \\ \theta \end{bmatrix} = \begin{bmatrix} R_1^T & 0 \\ 0 & 1 \end{bmatrix} Q^T E \theta
\]

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

\[
S_\lambda(\beta, \gamma) = \frac{1}{n} \| y - XEQ \begin{bmatrix} R_1^T & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \beta \\ \theta \end{bmatrix} \|^2 + \lambda \gamma^T \gamma
\]

\[ = \frac{1}{n} \| y - Z \begin{bmatrix} \beta \\ \theta \end{bmatrix} \|^2 + \lambda \gamma^T \gamma \]

with

\[ Z = [Z_1:Z_2] = XEQ \begin{bmatrix} R_1^T & 0 \\ 0 & 1 \end{bmatrix} \]

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 \( \beta \) and a part that depends only on \( \gamma \). Another QR decomposition, this time as

\[ Z_2 = FG = [F_1:F_2] \begin{bmatrix} G_1 \\ 0 \end{bmatrix} \]

is used to form
\[ S_{\lambda}(\beta, \gamma) = \frac{1}{n} \| w_1 - G_1 \beta - J_2 \gamma \|^2 + \frac{1}{n} \| w_2 - J_2 \gamma \|^2 + \lambda \gamma^T \gamma \]  
(5.2)

where

\[ w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} F_1^T \\ F_2^T \end{bmatrix} y = F_2^T y \]

and

\[ J = \begin{bmatrix} J_1 \\ J_2 \end{bmatrix} = \begin{bmatrix} F_1^T \\ F_2^T \end{bmatrix} Z_1 = F_2^T Z_1. \]

After checking that \( G_1 \) is non-singular, the first term in (5.2) can be made zero for any value of \( \gamma \) by solving

\[ G_1 \beta = w_1 - J_2 \gamma \]
(5.3)

for \( \beta \). This reduces the general penalized least squares to the same form as the ridge regression. A singular value decomposition

\[ J_2 = U D V^T \]
(5.4)

produces the representation of the "hat" matrix as

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

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

\[ z = U^T w_2 \]
(5.5)

the GCV function can be expressed as

\[ V(\lambda) = \frac{n \| w_2 \|^2 - \| z \|^2 + \sum_{j=1}^{a} \frac{n \lambda}{d_j^2 + n \lambda}}{n - p + \sum_{j=1}^{a} \frac{n \lambda}{d_j^2 + n \lambda}} \]
(5.6)

and the parameters vectors \( \gamma_\lambda, \beta_\lambda \), and \( \theta_\lambda \) are determined in the usual way given \( \lambda \), with (5.3) and

\[ \gamma_\lambda = V(D^2 + n \lambda I)^{-1} D U^T w_2, \]

yielding

\[ \theta_\lambda = EQ \begin{bmatrix} R_1^{-T} \\ 0 \end{bmatrix} \begin{bmatrix} \gamma_\lambda \\ \beta_\lambda \end{bmatrix}. \]

The biggest computational bottleneck is the SVD of \( J_2 \) when \( n \) and \( p \) are large, particularly since \( J_2 \) 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 \( \gamma_\lambda \), 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 \( \epsilon_0 \), usually a small multiple \( \epsilon \) of the relative machine precision \( \epsilon \), the truncated SVD algorithm finds a matrix \( \tilde{J}_2 \) which has \( a^* \leq a \) positive singular values and satisfies

\[ \frac{\| \tilde{J}_2 - J_2 \|^2_F}{\| J_2 \|^2_F} < \epsilon \epsilon_0, \]

in which \( \| \cdot \|_F \) is the Frobenius norm. For details of the truncated SVD algorithm, see Appendix 3. We replace \( J_2 \) by \( \tilde{J}_2 \) in (5.4), thereby reducing the effective number of parameters to \( a^* \). With the truncation we only calculate an \((n - h) \times a^* \) matrix \( U \) and a \((p - h) \times a^* \) matrix \( V \) so the vector \( z \) defined in (5.5) will be \( a^* \)-dimensional, with \( a \) replaced by \( a^* \). When \( J_2 \) 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 \( \epsilon \) for small \( \lambda \). To check on the effect of the truncation on the value of \( V(\lambda) \) and hence the calculation of \( \lambda \), we return the diagnostic quantity

\[ n \lambda / (n \lambda + \| \tilde{J}_2 - J_2 \|^2_F). \]
(5.7)

This is a lower bound on each of the quantities \( n \lambda / (d_j^2 + n \lambda) \) 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 \( J_2 \) to a bidiagonal form, then forming an expression for \( V(\lambda) \).

**Description**

The package has three main subroutine drivers. The first driver, \textit{dpss} 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, \textit{dptps} for partial thin plate smoothing splines, handles general covariates and in turn calls the third driver, \textit{drnsm} which handles penalized least squares problems with a semi-norm penalty. After a call to \textit{dpss} or \textit{dptps} the subroutine \textit{dpred} can be called to evaluate predicted values for additional points not in the design.
Replicates are handled in dtpss and dtpass using the following routines. The subroutine drepss sorts the x vectors and returns C1 and the information necessary for the routines duni and dswy (used only in dtpss). Subroutine duni reduces a matrix (T or K) to the corresponding matrix with unique entries (T1 or K1). The routine dswy sorts y and computes B1T 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 C1K1C1T and C1[TU:S1] from the design points x1, i=1,2,...,n using the routines dmakek and dmakek. The LINPACK routine dqrde is called to decompose C1[TU:S1] into its QR decomposition FG, followed by the routine dskf to calculate FTT C1K1C1TF. Dsgdcl does the Cholesky decomposition of FTT C1K1C1TF 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 dcv2 is divided into application of the rotations by F1 in dvasp, optimization of the V(λ) function in dvlop, computation of predictive mean square error (if requested), dmns, creation of the coefficient vector in dcecr1, creation of the predicted values in dpdcr, and creation of the diagonal of A(λ) in diat. Subroutine dvlop calls dmin to minimize V(λ) by repeated calls to dvl. The minimization is done by an initial grid search in the ln(λ) scale followed by a golden ratio search in the neighborhood of the minimizing grid point. The input variable ntrt controls the resolution of the initial grid search. A value for ntrt of 100 or greater is recommended to ensure that the global optimum is located. If a plot of V(λ) versus ln(λ) indicates that a local optimum has been obtained the user may either increase the value of ntrt or use the option to specify a reduced range for the search. The grid of ln(λ) values is returned along with the corresponding V(λ) values in the variable tbl. The variable auxi is returned containing λ, V(λ), V(0) and V(∞).

The driver dtpss for partial thin plate splines calls routines drepss, dmakek, duni, and dmakek to set up [T:S], [TU:S1] and K1U. These are fed to dcv2 to create the matrices Σ and X which are used by the driver dsns.

The subroutine dsns is a general driver for penalized least squares problems with a semi-norm penalty. It calls ddcam which decomposes Σ and X and returns information used by dggc to find λ, Θ, and other results. The work in ddcam is split into the decomposition of Σ in a call to dsgdcl and the transformation and decomposition of the design in dcrts and dodd which in turn calls dsvd or dsvd to perform the singular value decomposition. The work in dggc is divided into the same subroutines as dggcl with the exception that dcecr1 is replaced by dcecr.
(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 \( \lambda \) 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 \( X^T X + n \lambda I \). 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 \( \lambda \), 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 \( A(\lambda) \) 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 "dips" and the other using "dips". 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 \( x_1 \) and \( x_2 \) with one covariate, \( x_2 \). 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 "dips" is strongly preferred over "dips" for this example since it executes approximately 3 times faster. In general, if "dips" 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 "svdcs".

<table>
<thead>
<tr>
<th>Routine</th>
<th>Fortran BLAS</th>
<th>Assembler BLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seconds</td>
<td>%</td>
</tr>
<tr>
<td>dersps</td>
<td>3.07</td>
<td>3</td>
</tr>
<tr>
<td>dsetup</td>
<td>11.85</td>
<td>11</td>
</tr>
<tr>
<td>dsgdc1</td>
<td>3.70</td>
<td>3</td>
</tr>
<tr>
<td>Cholesky</td>
<td>31.55</td>
<td>29</td>
</tr>
<tr>
<td>bdiag.</td>
<td>36.10</td>
<td>33</td>
</tr>
<tr>
<td>diag.</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td>dgcvl</td>
<td>0.20</td>
<td>0</td>
</tr>
<tr>
<td>dsgap</td>
<td>1.70</td>
<td>2</td>
</tr>
<tr>
<td>dplsv</td>
<td>1.47</td>
<td>1</td>
</tr>
<tr>
<td>dcler1</td>
<td>0.28</td>
<td>0</td>
</tr>
<tr>
<td>dpcdr</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>ddiag</td>
<td>18.07</td>
<td>17</td>
</tr>
</tbody>
</table>

Table I: Example 1 using "dips"

UNIX is a trademark of AT&T Bell Laboratories
Table 2: Example 1 using dpipss

<table>
<thead>
<tr>
<th>Routine</th>
<th>Fortran BLAS</th>
<th>Assembler BLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seconds %</td>
<td>Seconds %</td>
</tr>
<tr>
<td>dreps</td>
<td>3.02 1</td>
<td>3.03 1</td>
</tr>
<tr>
<td>make K and T</td>
<td>14.45 4</td>
<td>14.62 6</td>
</tr>
<tr>
<td>dctsx</td>
<td>8.38 2</td>
<td>3.98 2</td>
</tr>
<tr>
<td>ddcn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dsgec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cholesky</td>
<td>4.05 1</td>
<td>2.22 1</td>
</tr>
<tr>
<td>QR</td>
<td>10.85 3</td>
<td>5.40 2</td>
</tr>
<tr>
<td>dcrzt</td>
<td>56.30 16</td>
<td>27.45 12</td>
</tr>
<tr>
<td>dzc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bidiag</td>
<td>86.43 24</td>
<td>41.77 18</td>
</tr>
<tr>
<td>diag</td>
<td>101.55 28</td>
<td>82.55 36</td>
</tr>
<tr>
<td>dgcc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drsap</td>
<td>0.42 0</td>
<td>0.23 0</td>
</tr>
<tr>
<td>dvlop</td>
<td>1.67 0</td>
<td>1.67 1</td>
</tr>
<tr>
<td>dpmse</td>
<td>1.65 0</td>
<td>1.45 1</td>
</tr>
<tr>
<td>dcfc</td>
<td>0.57 0</td>
<td>0.28 0</td>
</tr>
<tr>
<td>dpdc</td>
<td>0.47 0</td>
<td>0.25 0</td>
</tr>
<tr>
<td>ddiag</td>
<td>66.02 18</td>
<td>39.65 18</td>
</tr>
</tbody>
</table>

Total dpipss 359.43 226.28

We have divided the time for dsvdc into two subsections, bidiag and diag. Elden (1984) gave a method of expressing the GCV function V(λ) avoiding the diag step. This would result in considerable savings in the dsgecl or ddcn routines. This savings is offset by the calculations in dgcvc or dgevc 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 A(λ) 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, dsgecl or ddcn, should be called only once while analysis portion, dgcvc or dgevc, called for each Y. The analysis portion represents less than 5% of the total time if the calculation of the diagonal of A(λ) 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(λ) after the matrices are decomposed is very quick. In these runs the variable nblt was set to 200 so both V(λ) 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 \|x_i - 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 \|x_i - x_j\| to a tolerance level rather than checking for x_i = 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 x_i, i = 1, . . . , n. In all our test cases, this check has been adequate to ensure that the matrix F_1K F_2 is computationally positive definite.

It is important to note that the determination of replicates involves sorting the x_i, i = 1, . . . , n, in increasing lexicographic order. That is,

\[
\begin{bmatrix}
1 & 1 \\
2 & 1 \\
3 & 1 \\
4 & 1 \\
\end{bmatrix}
\]

the rows of would be re-ordered as

\[
\begin{bmatrix}
1 & 2 \\
3 & 1 \\
4 & 2 \\
\end{bmatrix}
\]

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 dpipss or dpipss 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 \((FF_t^T)\) 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:

\[
\frac{1}{n} \| w_2 - L^T L \delta \| + \lambda \delta^T L^T L \delta.
\]  
(A1.1)

If \(L\) is not of full row rank, any \(\delta\) satisfying

\[
L^T L w_2 = [(L^T L)^2 + n \lambda L^T L] \delta
\]

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

\[
\delta = (L^T L + n \lambda I)^{-1} w_2.
\]

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 \((F_t^T)\) 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

\[
y_i = \int \cdots \int K(x_i, x) f(x) d x + \varepsilon_i = L_i f + \varepsilon_i.
\]

The estimate \(f_\lambda\) of \(f\) is the minimizer, in an appropriate space, of

\[
\frac{1}{n} \sum_{i=1}^{n} (y_i - L_i f)^2 + \lambda J(f)
\]

where \(J(f)\) is an appropriate (quadratic) roughness penalty. If we can approximate \(f_\lambda\) 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 \(X\) as \(L_i B_j\) and the matrix \(\Sigma\) by \(J(\Sigma \theta_i B_j) = 0^T \Sigma 0\).

The thin plate basis functions were proposed for this purpose by Wahba (1980a). Starting with a set of suitably distributed distinct nodes \(t_1, t_2, \ldots, t_b\), the approximation is

\[
f_\lambda(x) = \sum_{i=1}^{t} \beta_i \phi_i(x) + \sum_{i=1}^{b} \delta_i E_m(x - t_i)
\]

(A2.2)

where \(\delta = (\delta_1, \ldots, \delta_b)^T\) must satisfy

\[
\sum_{i=1}^{b} \delta_i \phi_j(t_i) = 0, j = 1, \ldots, t.
\]

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

\[
S_\lambda(\beta, \delta) = \frac{1}{n} \| y - T\beta - K \delta \|^2 + \lambda \delta^T K_\beta \delta
\]

subject to \(T^T \delta = 0\), with

\[
T_{ij} = \phi_j(t_i).
\]

Here, \(T\) is \(n \times t\) and \(K\) is \(n \times b\), with entries

\[
(T)_{ij} = L_i \phi_j,
\]

\[
(K)_{ij} = L_i E_m(x - t_j),
\]

and \(K_\delta\) is \(b \times b\) with entries

\[
(K_\delta)_{ij} = E_m(t_i - t_j).
\]

If we are interested simply in evaluation functionals, then \(L_i f = f(x_i)\). The matrices \(T_\beta\) and \(K_\delta\) remain the same, but the matrices \(T\) and \(K\) have entries

\[
(T)_{ij} = \phi_j(x_i),
\]

\[
(K)_{ij} = E_m(x_i - t_j).
\]

We take a QR decomposition

\[
T_\beta = FG = [F_1; F_2][G_1] = F_1 G_1
\]

and use this to construct the parameter vector

\[
\theta = \begin{bmatrix} \beta \end{bmatrix}, \text{ with } \delta = F_2 \xi,
\]

and to create the design matrix
\[ X = (T : KF_{2}) \]

and penalty matrix
\[ \Sigma = \begin{bmatrix}
0 & 0 \\
0 & F_{2}^T K \alpha F_{2}
\end{bmatrix}. \]

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 \( L_{i} f = f(x_{i}) \), 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 \( F_{2}^T K \alpha F_{2} \) 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
\[ f(x, s) = \sum_{i=1}^{b} \beta_{i} \phi^{i}(x) + \sum_{i=1}^{c} \gamma_{i} \psi_{i}(x, s). \]

The objective function for a fixed \( \lambda \) becomes
\[ S(\alpha, \beta, \Sigma) = \frac{1}{n} \| y - \Sigma - T \beta - K \Sigma \|^2 + \lambda \Sigma^T K \alpha \Sigma \]
subject to \( T \Sigma = 0 \), in which \( S \) is \( n \times c \) with entries
\[ (\Sigma)_{ij} = L_{i} \psi_{j}(s, s). \]

The design matrix becomes
\[ X = (T : S : KF_{2}) \]
with parameter vector
\[ \theta = \begin{bmatrix} \beta \alpha \end{bmatrix}. \]

The penalty \( \Sigma \) has the same form, with the addition of rows and columns of zeroes corresponding to \( \alpha \). 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 \( X \) and \( Y \) be \( n \times p \) \((n \geq p)\) matrices with singular value decompositions \( UDV^T \) and \( RSW^T \) respectively.

Denote the ordered singular values of \( X \) as \( \{ d_i \} \), \( i = 1, \cdots, p \) with \( d_1 \geq d_2 \geq \cdots \geq d_p \) and the ordered singular values of \( Y \) as \( \{ s_i \} \), \( i = 1, \cdots, p \). Then
\[ \sum_{i=1}^{p} \left( d_i - s_i \right)^2 \| X - Y \|_F^2 = \text{tr} \left( (X - Y)(X - Y)^T \right) \]

We will take advantage of this theorem to calculate the SVD of a matrix \( X_{a} \), which is close to \( X \) in the sense that \( \| X - X_{a} \| \) is small but is better conditioned than is \( X \) 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 \( X \) using the pivoting scheme from LINPACK (Dongarra et al., 1979). That is, we determine \( Q, n \times n \) orthogonal, \( R, n \times p \) and zero below the main diagonal, and \( E, p \times p \) permutation matrix, such that
\[ X = QR \]
and \( R \) has the property that
\[ r_{ij}^2 \geq \sum_{j=1}^{p} r_{jj}^2 (j = a^*, a^* + 1, \cdots, p). \]

If we take the SVD of \( R_{a} \), the triangular matrix composed of the first \( p \) rows of \( R \), as
\[ R_{a} = KDL^T \]
we can produce the SVD of \( X \) as
\[ X = Q_{a} KDL^T E^T = UDV^T \]
where \( Q_{a} \) is the \( n \times p \) matrix composed of the first \( p \) columns of \( Q \) and \( U = Q_{a} K \) is \( n \times p \) while \( V \) and \( E \) are orthogonal. This method would not, however, produce better conditioning for the SVD algorithm since the singular values of \( R_{a} \) are the same as the singular values of \( X \).

To provide better conditioning, we truncate the matrix \( R_{a} \) after the \( a^* \)'th row and take the SVD of the resulting \( n \times a^* \) matrix \( R_{a} \) as
\[ R_{a} = K_{a} D_{a} L_{a} \]
where \( K_{a} \) is \( a^* \times a^* \) and \( L_{a} \) is \( a^* \times p \). The diagonal elements of \( D_{a} \) are no longer the singular values of \( X \) but now represent the singular values of a matrix
\[ X_{a} = Q_{a} \begin{bmatrix} R_{a} & 0 \\ 0 & 0 \end{bmatrix} E^T \]
which is different from \( X \). However,
\[ \| X - X_{a} \|_F = \left( \sum_{i=a^*+1}^{p} \sum_{j=i}^{p} r_{ij}^2 \right)^{1/2} \]
so we can choose \( a^* \) to be as small as possible subject to the
constraint that
\[
\frac{\| X - X_{*p} \|_F}{\| X \|_F} \leq \tau \rho
\]  \hspace{1cm} (A3.8)

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

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

Allowing \( \tau \) 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 \( \tau \) 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 \( \alpha \) is determined.

By theorem 1, if \( \{ d_i \} \), \( i = 1, \ldots, p \) are the ordered singular values of \( X \) and \( \{ d_{i, \ast p} \} \), \( i = 1, \ldots, p \) are the ordered singular values of \( X_{\ast p} \), then
\[
\left( \sum_{i=1}^{p} (d_i - d_{i, \ast p})^2 \right)^{1/2} \leq \tau \rho \| X \|_F = \tau \rho \left( \sum_{i=1}^{p} d_i^2 \right)^{1/2}
\]  \hspace{1cm} (A3.9)

If \( n \leq p \), the same procedure is applied to \( X^T \).

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


Hutchinson, M. F. (1984) "A Summary of Some Surface Fitting..."


Table 3. GCVPACK notation correspondence

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>integer constants</td>
</tr>
<tr>
<td>d</td>
<td>number of observations</td>
</tr>
<tr>
<td>m</td>
<td>dimension of polynomial space</td>
</tr>
<tr>
<td>c</td>
<td>order of derivatives of penalty</td>
</tr>
<tr>
<td>c_1</td>
<td>number of covariates</td>
</tr>
<tr>
<td>c - c_1</td>
<td>number of covariates in S replicating structure of T</td>
</tr>
<tr>
<td>a</td>
<td>ncoV - ncoV1</td>
</tr>
<tr>
<td>a^*</td>
<td>iout[4] = number of unique obs. (dtpss &amp; dtpss)</td>
</tr>
<tr>
<td>t</td>
<td>iout[1] = number of positive singular values</td>
</tr>
<tr>
<td>h = t+c</td>
<td>iout[3] = size of null space of S</td>
</tr>
<tr>
<td>p = a + t + c</td>
<td>iout[2] = number of parameters</td>
</tr>
<tr>
<td>t+c_i</td>
<td>number of columns in [T:S_i]</td>
</tr>
<tr>
<td>p - h</td>
<td>npar - null</td>
</tr>
<tr>
<td>n - h</td>
<td>nobs - null</td>
</tr>
</tbody>
</table>

| y      | response vector |
| β      | coefficients for covariates |
| α      | coefficients for polynomial |
| δ      | coefficients for smooth |
| γ      | coefficients for well-defined smooth basis |
| θ      | coefficients (in several forms) |

| X      | design matrix for splined variables |
| [T:S_1] | polynomials and replicated covariates |
| [T_U:S_U] | unique polynomials and replicated covariates |
| S_2    | unreplicated covariates |
| A(λ)_{u} | diagonal of hat matrix |
| D_{u}  | singular values |
| Σ      | penalty matrix |
| F, G   | QR decomposition of [T:S_1] |
| E      | permutation for pivoted Cholesky of Σ |
| Q, R   | QR decomposition of Cholesky factor of Σ |
| C_{u}  | square root of number of replicates of i’th unique x |

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahat</td>
<td>lamhat</td>
</tr>
<tr>
<td>J(f)</td>
<td>penalty</td>
</tr>
<tr>
<td>∥I - A(λ)∥^2</td>
<td>rrs</td>
</tr>
<tr>
<td>r(I - A(λ))</td>
<td>dout[3] = residual sum of squares</td>
</tr>
<tr>
<td>∥B_{x}^2y∥^2</td>
<td>sqrtrep</td>
</tr>
<tr>
<td>ρ</td>
<td>macheap</td>
</tr>
<tr>
<td>τ</td>
<td>tau</td>
</tr>
<tr>
<td>τρ</td>
<td>minrat</td>
</tr>
<tr>
<td></td>
<td>machine tolerance</td>
</tr>
</tbody>
</table>


DOCUMENTATION FOR DRIVER Routines

subroutine dptps (des, ldes, nobs, dim, m, s, lds, ncov1, ncov2, y, ntbl, 
   * adiaq, lamlim, dout, iout, coef, svals, tbl, ldtbl, auxtbl, work, 
   * lwa, work(lwa), job, info)

integer ldes, nobs, dim, m, lds, ncov1, ncov2, iout(4), ldtbl, lwa, 
   * iwork(lwa), job, info
double precision des(ldes, dim), s(lds, *), y(nobs), adiaq(ncovs), 
   * lamlim(2), dout(4), coef(*), svals(*), tbl(ldtbl, 3), 
   * auxtbl(3, 3), work(lwa)

Purpose: determine the generalized cross validation estimate of the 
smoothing parameter and fit model parameters for a partial thin 
plate spline model.

On Entry:
    des(ldes, dim) design for the variables to be splined 
    ldes leading dimension of des as declared in the 
calling program 
    nobs number of observations 
    dim number of columns in des 
    m order of the derivatives in the penalty 
    s(lds, ncov1+ncov2) design for the covariates 
    first ncov1 columns contain covariates which 
duplicate the replication structure of des 
    next ncov2 columns contain covariates which 
do not duplicate the replication structure of des 
    lds leading dimension of s as declared in the 
calling program 
    ncov1 number of covariates which duplicate the 
    replication structure of des 
    ncov2 number of covariates which do not duplicate the 
    replication structure of des 
    y(nobs) response vector 
    ntbl number of evenly spaced values for 
    log10(nobs*lambda) to be used in the initial 
    grid search for lambda hat 
    adiaq(ncovs) *true* y values on entry if predictive mse is 
    requested 
    lamlim(2) limits on lambda hat search (in log10(nobs* 
    lambda) scale) if user input limits are 
    requested, if lamlim(1) = lamlim(2) then lamhat 
    is set to (10*lamlim(1))/nobs 
    ldtbl leading dimension of tbl as declared in the 
calling program 
    job integer with decimal expansion abd 
    a if k is nonzero then predictive mse is computed 
    using adiaq as true y 
    b if b is nonzero then user input limits on search 
    for lambda hat are used 
    c if c is nonzero then adiaq will be calculated 

On Exit:
    des(ldes, dim) unique rows of des 
    y(nobs) predicted values 
    adiaq(ncovs) diagonal elements of the hat matrix if requested 
    lamlim(2) limits on lambda hat search 
    (in log10(nobs*lambda) scale) 
    dout(4) contains: 
    1 lamhat generalized cross validation estimate of the smoothing parameter 
    2 penalty smoothing penalty 
    3 rse residual sum of squares 
    4 tr(I-A) trace of I - A 
    iout(4) contains: 
    1 npsing number of positive singular values 
    if info indicates nonzero info from 
dsvdc then npsing contains 
    info as it was returned from dsvdc 
    2 npar number of parameters 
    (npar = nobs + nnull) 
    3 nnull size of the null space of sigma 
    (m+dim-1 choose dim)*ncov1+ncov2 
    4 nuobs number of unique rows in des 
    coeficient estimates [beta*alpha*delta]' 
    coef must have a dimension of at least 
    nuobs+nnull 
    singular values, svals must have a dimension, 
of at least nuobs-nnull. 
    if info indicates nonzero info in dsvdc then 
svals is as returned from dsvdc. 
    tbl(ldtbl, 3) column contains 
    1 grid of log10(nobs*lambda) 
    2 V(lambda) 
    3 R(lambda) if requested 
    auxtbl(3, 3) auxiliary table 
    1st row contains: 
    log10(nobs*lambda), V(lambda) and 
    R(lambda) if requested 
    where lamhat is the cvv estimate of lambda 
    2nd row contains: 
    0, V(0) and R(0) if requested 
    3rd row contains: 
    0, V(infinity) and R(infinity) if requested 
    info error indicator 
    0 : successful completion 
    -1 : log10(nobs*lambda) <= lamlim(1) 
    (not fatal) 
    -2 : log10(nobs*lambda) >= lamlim(2) 
    (not fatal) 
    1 : dimension error 
    2 : error in drea, the first ncov1 columns 
of s do not duplicate the replication 
structure of des 
    3 : lwa (length of work) is too small 
    4 : lwa (length of work) is too small 
    5 : error in dsaket 
    100K info : 1000 + nonzero info returned from 
dsnam 

Working Storage:
    work(lwa) 
    lwa 
    lwork(lwa)

double precision work vector 
length of work as declared in the calling 
program 
must be at least lwa + lwa2 where 
lwa1 = (nnull+ncov2)*(nobs+nuobs+1) 
+ npar*(nobs+npar) 
lwa2 = (npar+nnull)*(npar-2*nnull+2*nobs) 
+ npar*nuobs 

integer work vector 
length of the lwork as declared in the calling 
program 
must be at least lwa + lwa2 where 
lwa1 = (nnull+ncov2)*(nobs+nuobs+1) 
+ npar*(nobs+npar) 
lwa2 = (npar+nnull)*(npar-2*nnull+2*nobs) 
+ npar*nuobs
Documentation for Driver Routines

Program must be at least 3*nobs - (nnull - ncov2)

Subprograms Called Directly:
- Gvpack - drep smake dsmake dtsdk dsdsm
- Llnpack - dqrdc dqrs
- Blas - dcopy
- Other - dpumat dprmut dtprf fact mkpoly

Subprograms Called Indirectly:
- Gvpack - dcrtz dddcom dgcov dsdvt dvsm dvdc dsvdc ddsag
dvop dvop pmse defer dpdcr dvms dvl dvsc
- Llnpack - dchdc dqrdc dqrs dlrsv dsvdc dvtrco
dcopy ddtq dgsym dswap
- Other - dpumat dprmut dtprf fact mkpoly

subroutine dsdsm (x, ldx, y, nobs, ldsig, ldnpar, nlpar, nnull, adiag, tau, lamlim(2), dout(5), coeff(npar), svals(npar-nnull), work(lwaw), lwork(lwaw), info)
double precision x(ldx,npar), y(nobs), ldsig, ldnpar, nlpar, nnull, adiag(nobs), tau, lamlim(2), dout(5), coeff(npar), svals(*), work(lwaw)
integer nobs, nlpar, nnull, ldsig, ldnpar, nnull, nlpar, adiag(nobs), tau, lamlim(2), dout(5), coeff(npar), svals(*)
integer info

Purpose: determine the generalized cross validation estimate of the
smoothing parameter and fit model parameters for a penalized
least squares problem with a semi-norm smoothing matrix.

On Entry:
- design matrix y(ld,npar)
- leading dimension of x as declared in the
- calling program, must be at least max(nobs, npar)
- response vector
- symmetric matrix that defines the semi-norm
- leading dimension of sigma as declared
- in the calling program
- number of observations
- number of parameters
- dimension of the null space of sigma
- *true* y values on entry if computation of
predictive mse is requested
- limits on lambda hat search (in log10(nobs*lambda) scale)
- requested if lamlim(1) = lamlim(2) then lambda
is set to (10**lamlim(1))/nobs
- multiplier controlling the amount of truncation
- if truncation is requested (try tau = 1
to start then try 10 and 100)
- number of evenly spaced values for
log10(nobs*lambda) to be used in the initial
grid search for lambda hat
- if nbl = 0 only a golden ratio search will be
done and tbl is not referenced, if ntbl > 0
there will be ntbl rows returned in tbl
- leading dimension of tbl as declared in the
calling program
- integer with decimal expansion abcd
- if a is nonzero then truncation is used
- b is nonzero then predictive mse is computed
  using adiag as true y

On Exit:
- x(ldx,npar)
- y(nobs)
- sigma(ldsig, npar)
- adiag(nobs)
- lamlim(2)
- dout(5)
- coeff(npar)
- svals(npar-nnull)

Work Arrays:

if c is nonzero then user input limits on search
for lambda hat are used
if d is nonzero then the diagonal of the hat
matrix is calculated

overwritten with many intermediate results
predicted values
overwritten with the QR decomposition of the
Cholesky factor of sigma
diagonal elements of the hat matrix if requested
limits on lambda hat search
(in log10(nobs*lambda) scale)

中含有:
- lamhat generalized cross validation
- estimate of the smoothing parameter
- penalty smoothing penalty
- residual sum of squares
- trace(I - A)
- 5 truncation ratio = 1/(1+normk/(nobs*lamhat)))
where normk = norm(R - R sub k)**2

contains:
- number of singular values
- if info indicates nonzero info in
dsdvc then info(1) contains info as
it was returned from dsdvc
- number of parameters
- size of the null space of sigma
- coefficient estimates
- first npsing entries contain singular values
of the matrix J2
- if info indicates nonzero info in dsdvc then
svals is as it was returned from dsdvc

Column contains
- grid of log10(nobs*lambda)
- V(lambda)
- R(lambda) if requested

Auxiliary Table
1st row contains:
- 1og(l0(nobs*lamhat), V(lamhat) and
  R(lamhat) if requested
where lamhat is the dcv estimate of lambda
2nd row contains:
- 0, V(0) and R(0) if requested
3rd row contains:
- 0, V(infinity) and R(infinity) if requested

Error Indicator
0 : successful completion
-3 : nnull is too small (not fatal)
-2 : log10(nobs*lamhat) <= lamlim(2)
  (not fatal)
-1 : log10(nobs*lamhat) <= lamlim(1)
  (not fatal)
 1 : dimension error
 2 : lwa (length of work) is too small
 3 : lwa (length of lwork) is too small
 4 : error in ntbl or tau

100* info < 200 : 100 + nonzero info returned
from ddcmm
200* info < 300 : 200 + nonzero info returned
from dgcv
Documentation for Driver Routines

```
c double precision work, lwork
lwork = length of work as declared in the calling program
must be at least (npar-nnull)*(npar-2*nnull+2*nobs)*npar+nobs

integer work, llw
llw = length of llw as declared in the calling program
must be at least 2*npar - nnull

Subroutines Called Directly:
- Gcvpack - dcv
- Gcvpack - dcv

Subroutines Called Indirectly:
- Gcvpack - dcv
dc = double precision work vector
dc = double precision work vector
dc = double precision work vector
dc = double precision work vector

Purpose: determine the generalized cross validation estimate of the smoothing parameter and fit model parameters for a thin plate smoothing spline.

On Entry:
- design for the variables to be splined
- leading dimension of design as declared in calling program
- number of observations
- number of columns in design
- order of the derivatives in the penalty
- design for the covariates. The covariates must duplicate the replication structure of design.
- leading dimension of covariates as declared in calling program
- number of covariates
- response vector
- number of evenly spaced values for grid search for
- grid search for lambda hat
- if ntr = 0 only a golden ratio search will be done and tbl is not referenced, if ntr > 0 there will be ntbl rows returned in tbl, "true" y values on entry if predictive mse is requested
- limits on lambda hat search in (log10(nobs*lambda) scale) if requested if lambda(1) = lambda(2) then lambda is set to (10**lambda(1))/nobs
- leading dimension of tbl as declared in the calling program

On Exit:
- design for the variables to be splined
- leading dimension of design as declared in calling program
- number of observations
- order of the derivatives in the penalty
- design for the covariates. The covariates must duplicate the replication structure of design.
- leading dimension of covariates as declared in calling program
- number of covariates
- response vector
- number of evenly spaced values for grid search for lambda hat
- grid search for lambda hat
- if ntr = 0 only a golden ratio search will be done and tbl is not referenced, if ntr > 0 there will be ntbl rows returned in tbl, "true" y values on entry if predictive mse is requested
- limits on lambda hat search in (log10(nobs*lambda) scale) if requested if lambda(1) = lambda(2) then lambda is set to (10**lambda(1))/nobs
- leading dimension of tbl as declared in the calling program
```
Documentation for Driver Routines

3 : lwa (length of work) is too small
4 : lliwa (length of iwork) is too small
10 < info < 20 : 10 + nonzero info returned
        from dsetup
100< info <200 : 100 + nonzero info returned
        from dseigcl
200< info <300 : 200 + nonzero info returned
        from dseigv1

Work Arrays:
  work(lwa)        double precision work vector
  lwa
  iwork(lliwa)     integer work vector
  liwa

length of work as declared in the calling program
Must be at least nuobs(2+nobs+nuobs)+nobs
length of iwork as declared in the calling program
Must be at least 2*nobs + nuobs - nobs

Subprograms Called Directly:
  Gcvpack - drepd duc1 dseig dseigcl dseigv1
  Other  - dprmut mkpoly

Subprograms Called Indirectly:
  Gcvpack - dcfcri dssap dsvlp dsvto dpmid dpmsc
dvmin dvl dmakek ddiag
  Linpack - dchdc dqrdc dqrs1 dtrsl dsvd0
  Blas     - ddot dcopy dgemv
  Other    - dprmut dset dtkf fact mkpoly
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 dtpss.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.

<table>
<thead>
<tr>
<th>affected routines</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddsc.f, ddcom.f, dcrtz.f**, dzdsc.f**</td>
<td>ddsdc.f is replaced with dcrtz.f, dzdsc.f.</td>
</tr>
<tr>
<td>ddcom.f, dzdsc.f</td>
<td>Nonzero info and svals from dsvec are now returned in npsing and svals.</td>
</tr>
<tr>
<td>dsgdcl.f</td>
<td>Nonzero info and svals from dsvec are now returned in p and svals.</td>
</tr>
<tr>
<td>dtpss.f, dsnsm.f, dtpss.f</td>
<td>Nonzero info and svals from dsvec are now returned in iout(1) and svals.</td>
</tr>
<tr>
<td>dpdcr.f*, ddiag.f**, dgcv.f, dgcvf.f</td>
<td>Calculation of the diagonal of the hat matrix has been removed from dpdcr.f and is now done in the new routine ddiag.f.</td>
</tr>
<tr>
<td>dgps.f, dsnsm.f, dpdps.f, dgcvl.f, dgcv.f</td>
<td>Iout and dout are now assigned as early as possible to return more information when info is positive.</td>
</tr>
<tr>
<td>dgcv.f, dgcvl.f, dpmse.f, dpdps.f, dsnsm.f, dpdps.f, dvl.f, dvpw.f</td>
<td>Lambda values are expressed as log base 10 rather than natural log.</td>
</tr>
<tr>
<td>dsuy.p*, drep.s*, drtp.f, dpdps.f</td>
<td>The argument dfrep is removed from the calling sequences of dsuy.p and drep.s.</td>
</tr>
<tr>
<td>dvlpw.p*, dgcvl.f, dgcv.f</td>
<td>The argument npar is removed from the calling sequence of dvlpw.p.</td>
</tr>
<tr>
<td>defcr.f*, dgcv.f</td>
<td>The argument fgaux is removed from the calling sequence of defcr.f.</td>
</tr>
<tr>
<td>ddcom.f, dcrtz.f*</td>
<td>The argument lwa is removed from the calling sequence of dcrtz.f.</td>
</tr>
<tr>
<td>ddcom.f*, dsnsm.f*, dzdsc.f*</td>
<td>The argument factor is added to allow control over the amount of truncation.</td>
</tr>
<tr>
<td>dsnsm.f</td>
<td>The argument dout must be declared with length 5 rather than length 4. The truncation ratio is now returned in dout(5).</td>
</tr>
<tr>
<td>ddcom.f*, dzdsc.f*, dsvec.f*</td>
<td>The Frobenius norm of $R - R_k$ is now returned as an argument.</td>
</tr>
<tr>
<td>defcr.f, defcr.f, dgemv.f**, ddiag.f, dpdcr.f, dpmse.f, dpred.f, dvsap.f</td>
<td>The routine dgemv.f is called to perform matrix multiplication. (Dgemv.f is from the extended BLAS)</td>
</tr>
<tr>
<td>testpss.out, testpss.out</td>
<td>Changes due to the switch from natural log to log base 10 and reformattting of output.</td>
</tr>
</tbody>
</table>
The input files testpss.in and testpss.in replace fort.8 and are now read
as standard input.

Routines and files used for new integral equation test driver integn.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 dpdpss, dsnsm and dpdpss. Refer to
Description section and code for further details.

<table>
<thead>
<tr>
<th>Code</th>
<th>Occurred In</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>dvmin</td>
<td>log(nobs*lamhat) ≥ lamlim(2) (not fatal)</td>
</tr>
<tr>
<td>-1</td>
<td>dvmin</td>
<td>log(nobs*lamhat) ≤ lamlim(1) (not fatal), will occur if limits on lambda hat are input by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user and are equal.</td>
</tr>
<tr>
<td>1</td>
<td>dpdpss</td>
<td>dimension error (nobs ≤ 0 or m ≤ 0 or dim ≤ 0 or nbl &lt; 0 or nbl &gt; ldbl or 2*m -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dim ≤ 0 )</td>
</tr>
<tr>
<td>2</td>
<td>dpdpss</td>
<td>error in drep(s, the first ncon 1 columns of s do not duplicate the replication structure of des</td>
</tr>
<tr>
<td>3</td>
<td>dpdpss</td>
<td>lwa (length of work) &lt; ncts 1*(nobs + nsubs + 1) + npar*(nobs + npar + 1) + (npar -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nnull)* (npars -2<em>nnull +2</em>nobs) +npars - nobs</td>
</tr>
<tr>
<td>4</td>
<td>dpdpss</td>
<td>liwa (length of iwork) &lt; 3*nobs - ncts 1</td>
</tr>
<tr>
<td>5</td>
<td>dpdpss</td>
<td>error in dmaket (error in creation of T)</td>
</tr>
<tr>
<td>6</td>
<td>dpdpss</td>
<td>sigma is rank deficient</td>
</tr>
<tr>
<td>1001*</td>
<td>dsnsm</td>
<td>dimension error (nobs ≤ 0 or npar ≤ 0 or nnull ≤ 0 or (npar - nnull) ≤ 0 )</td>
</tr>
<tr>
<td>1002*</td>
<td>dsnsm</td>
<td>lwa (length of work) &lt; (npars - nnull)* (npars -2*nnull +2 + nobs) + npars + nobs</td>
</tr>
<tr>
<td>1003*</td>
<td>dsnsm</td>
<td>liwa (length of iwork) &lt; 2*npars - nnull</td>
</tr>
<tr>
<td>1004*</td>
<td>dsnsm</td>
<td>error in nbl (nbl &lt; 0 or nbl &gt; ldbl)</td>
</tr>
<tr>
<td>1101*</td>
<td>ddcom</td>
<td>dimension error (nobs ≤ 0 or npar ≤ 0 or nnull ≤ 0 or (npar - nnull) ≤ 0 ) or ldx &lt; nobs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or ldx &lt; npar )</td>
</tr>
<tr>
<td>1102*</td>
<td>ddcom</td>
<td>ldeaux (length of dcaux) &lt; (npars - nnull)* (npars -2*nnull +2 *nobs - nnull)</td>
</tr>
<tr>
<td>1103*</td>
<td>ddcom</td>
<td>lwa (length of work) &lt; (npars - nnull)* (nobs - nnull + 1) + nobs</td>
</tr>
<tr>
<td>1104*</td>
<td>ddcom</td>
<td>liwa (length of iwork) &lt; npars - nnull</td>
</tr>
<tr>
<td>1111*</td>
<td>dsgec</td>
<td>calculated nnull is smaller than input nnull</td>
</tr>
<tr>
<td>1121</td>
<td>derror</td>
<td>error in dresl, R is singular</td>
</tr>
<tr>
<td>1131*</td>
<td>dsdc</td>
<td>lwa (length of work) &lt; (npars - nnull)* (nobs - nnull) + npars - nnull + nobs</td>
</tr>
<tr>
<td>1132</td>
<td>dsdc</td>
<td>transpose of F2 is necessary (npars &gt; nobs) but npars &gt; ldx</td>
</tr>
<tr>
<td>1133*</td>
<td>dsdc</td>
<td>failure to converge in dsdc called from dsvec (using JF)</td>
</tr>
<tr>
<td>1134</td>
<td>dsdc</td>
<td>failure to converge in dsdc (using JF2), this error can usually be cured by increasing the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter MAXIT in the LINPACK routine dsvec</td>
</tr>
<tr>
<td>1135*</td>
<td>dsdc</td>
<td>failure to converge in dsvec called from dsvec</td>
</tr>
</tbody>
</table>
failure to converge in dsdvec, this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine dsdvec

**Non-zero Error Codes from dsnsm**

<table>
<thead>
<tr>
<th>Code</th>
<th>Occurred In</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>dsdsc</td>
<td>calculated nnull is larger than input nnull (not fatal)</td>
</tr>
<tr>
<td>-2</td>
<td>dvmn</td>
<td>log(nobs * lamhat) &gt; lamlim (2) (not fatal)</td>
</tr>
<tr>
<td>-1</td>
<td>dvmn</td>
<td>log(nobs * lamhat) ≤ lamlim (1) (not fatal), will occur if limits on lambda hat are input by user and are equal.</td>
</tr>
<tr>
<td>1</td>
<td>dsnsm</td>
<td>dimension error (nobs ≤ 0 or npar ≤ 0 or nnull ≤ 0 or (npar - nnull) ≤ 0)</td>
</tr>
<tr>
<td>2</td>
<td>dsnsm</td>
<td>lwa (length of work) &lt; (npar - nnull) + nobs</td>
</tr>
<tr>
<td>3</td>
<td>dsnsm</td>
<td>liwa (length of iwork) &lt; (npar - nnull) + nobs</td>
</tr>
<tr>
<td>4</td>
<td>dsnsm</td>
<td>error in nbl (nbl &lt; 0 or nbl &gt; ldbil) or t &lt; 0</td>
</tr>
<tr>
<td>101</td>
<td>dcom</td>
<td>t &lt; 0 or dimension error (nobs ≤ 0 or npar ≤ 0 or nnull ≤ 0 or (npar - nnull) ≤ 0 or ldx &lt; nobs or ldx &lt; npar)</td>
</tr>
<tr>
<td>102*</td>
<td>dcom</td>
<td>ldaux (length of dcaux) &lt; (npar - nnull) + 2*npar - nnull</td>
</tr>
<tr>
<td>103*</td>
<td>dcom</td>
<td>lwa (length of work) &lt; (npar - nnull) + nobs</td>
</tr>
<tr>
<td>104*</td>
<td>dcom</td>
<td>liwa (length of iwork) &lt; npar - nnull</td>
</tr>
<tr>
<td>111</td>
<td>dsdsc</td>
<td>calculated nnull is smaller than input nnull</td>
</tr>
<tr>
<td>112</td>
<td>dcrz</td>
<td>error in dcrz, R is singular</td>
</tr>
<tr>
<td>132</td>
<td>ddcz</td>
<td>lwa (length of work) &lt; (npar - nnull) + npar - nnull + nobs</td>
</tr>
<tr>
<td>132</td>
<td>ddcz</td>
<td>t &lt; 0</td>
</tr>
<tr>
<td>133</td>
<td>ddcz</td>
<td>transpose of J2 is necessary (npar &gt; nobs) but npar &gt; ldx</td>
</tr>
<tr>
<td>134</td>
<td>ddcz</td>
<td>failure to converge in dsvdc called from dsvdc (using J2), this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine dsvdc or by increasing the value of the argument t to dsnsm (see discussion in TR775 (rev.))</td>
</tr>
<tr>
<td>135</td>
<td>ddcz</td>
<td>failure to converge in dsvdc (using J2), this error can usually be cured by using the truncation option or by increasing the parameter MAXIT in the LINPACK routine dsvdc</td>
</tr>
<tr>
<td>136</td>
<td>ddcz</td>
<td>failure to converge in dsvdc called from dsvdc, this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine dsvdc or by increasing the value of the argument t to dsnsm (see discussion in TR775 (rev.))</td>
</tr>
<tr>
<td>137</td>
<td>ddcz</td>
<td>failure to converge in dsvdc (using J2), this error can usually be cured by using the truncation option or by increasing the parameter MAXIT in the LINPACK routine dsvdc</td>
</tr>
<tr>
<td>201*</td>
<td>dgcw</td>
<td>dimension error (nobs ≤ 0 or npar ≤ 0 or nnull ≤ 0 or (npar - nnull) ≤ 0)</td>
</tr>
<tr>
<td>202*</td>
<td>dgcw</td>
<td>error in nbl (nbl &lt; 0 or nbl &gt; ldbil)</td>
</tr>
<tr>
<td>203*</td>
<td>dgcw</td>
<td>ldaux (length of dcaux) &lt; (npar - nnull) + 2*npar - nnull</td>
</tr>
<tr>
<td>Code</td>
<td>Occurred In</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>204*</td>
<td>dgcv</td>
<td>lwa (length of work) &lt; (npar - nnull) + nobsl</td>
</tr>
<tr>
<td>205</td>
<td>dgcv</td>
<td>lamlim (1) &gt; lamlim (2)</td>
</tr>
<tr>
<td>211</td>
<td>dvlop</td>
<td>svals (1) = 0.0d0</td>
</tr>
<tr>
<td>212</td>
<td>dvlop</td>
<td>npsing is incorrect</td>
</tr>
<tr>
<td>213*</td>
<td>dvlop</td>
<td>lamlim (1) &gt; lamlim (2)</td>
</tr>
<tr>
<td>221</td>
<td>dcfcr</td>
<td>error in dirco, G is singular</td>
</tr>
<tr>
<td>222</td>
<td>dcfcr</td>
<td>error in dirsl, R is singular</td>
</tr>
</tbody>
</table>

*Should not occur in normal circumstances

---

**Non-zero Error Codes from dipss**

<table>
<thead>
<tr>
<th>Code</th>
<th>Occurred In</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>dvmin</td>
<td>log(nobs *lamhat) ≥ lamlim (2) (not fatal)</td>
</tr>
<tr>
<td>-1</td>
<td>dvmin</td>
<td>log(nobs *lamhat) ≤ lamlim (1) (not fatal), will occur if limits on lambda hat are input by user and are equal.</td>
</tr>
<tr>
<td>1</td>
<td>dipss</td>
<td>dimension error (nobs ≤ 0 or m ≤ 0 or dim ≤ 0 or ntbl &lt; 0 or ntbl &gt; ldnl or 2*m - dim ≤ 0)</td>
</tr>
<tr>
<td>2</td>
<td>dipss</td>
<td>error in drep, covariates do not duplicate the replication structure of des</td>
</tr>
<tr>
<td>3</td>
<td>dipss</td>
<td>lwa (length of work) &lt; nuobs*(2 + ncts + nuobs) + nobs</td>
</tr>
<tr>
<td>4</td>
<td>dipss</td>
<td>liwa (length of iwork) &lt; 2*nobs + nuobs - ncts</td>
</tr>
<tr>
<td>11</td>
<td>dsetup</td>
<td>error in dmaket (error in creation of T)</td>
</tr>
<tr>
<td>101*</td>
<td>dsgdc1</td>
<td>lwa &lt; 2*npar</td>
</tr>
<tr>
<td>102</td>
<td>dsgdc1</td>
<td>F^TF is not of full rank</td>
</tr>
<tr>
<td>103</td>
<td>dsgdc1</td>
<td>failure to converge in dsdc, this error can usually be cured by increasing the parameter MAXIT in the LINPACK routine dsdc</td>
</tr>
<tr>
<td>201*</td>
<td>dgcvl</td>
<td>dimension error (nuobs ≤ 0 or ncts ≤ 0 or nuobs - ncts ≤ 0)</td>
</tr>
<tr>
<td>202*</td>
<td>dgcvl</td>
<td>error in ntbl (ntbl &lt; 0 or ntbl &gt; ldnl)</td>
</tr>
<tr>
<td>203*</td>
<td>dgcvl</td>
<td>lwa (length of work) &lt; nuobs - ncts 1 + nobs</td>
</tr>
<tr>
<td>204</td>
<td>dgcvl</td>
<td>lamlim (1) &gt; lamlim (2)</td>
</tr>
<tr>
<td>211</td>
<td>dvlop</td>
<td>svals (1) = 0.0d0</td>
</tr>
<tr>
<td>212*</td>
<td>dvlop</td>
<td>npsing is incorrect</td>
</tr>
<tr>
<td>213</td>
<td>dvlop</td>
<td>lamlim (1) &gt; lamlim (2)</td>
</tr>
<tr>
<td>221</td>
<td>dcfcr</td>
<td>error in dirco, G is singular</td>
</tr>
</tbody>
</table>

*Should not occur in normal circumstances

---

**Non-zero Error Codes from dprep**

<table>
<thead>
<tr>
<th>Code</th>
<th>Occurred In</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dprep</td>
<td>dimension error (nobs ≤ 0 or nct ≤ 0 or m ≤ 0 or dim ≤ 0)</td>
</tr>
<tr>
<td>2</td>
<td>dprep</td>
<td>npar ≠ ndesb + nct + nconv1 + nconv2</td>
</tr>
<tr>
<td>3</td>
<td>dprep</td>
<td>lwa (length of work) &lt; nprep * (nct + ndesb)</td>
</tr>
<tr>
<td>4</td>
<td>dprep</td>
<td>error in dmaket (error in creation of T)</td>
</tr>
</tbody>
</table>