Computational Optimization and Applications manuscript No.
(will be inserted by the editor)

Variable Projection for Nonlinear Least Squares Problems

Dianne P. O'Leary · Bert W. Rust

Received: date / Accepted: date

Abstract The variable projection algorithm of Golub and Pereyra (1973) has proven to be quite valuable in the solution of nonlinear least squares problems in which a substantial number of the parameters are linear. Its advantages are efficiency and, more importantly, a better likelihood of finding a global minimizer rather than a local one. The purpose of our work is to provide a more robust implementation of this algorithm, include constraints on the parameters, more clearly identify key ingredients so that improvements can be made, compute the Jacobian matrix more accurately, and make future implementations in other languages easy.

Keywords Data fitting · model fitting · variable projection method · nonlinear least squares problems · Jacobian approximation · least squares approximation · statistical software · mathematical software design and analysis

---

Certain commercial software products are identified in this paper in order to adequately specify the computational procedures. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology nor does it imply that the software products identified are necessarily the best available for the purpose. Part of this work was supported by the National Science Foundation under Grant NSF DMS 1016266.

Dianne P. O'Leary
National Institute of Standards and Technology and University of Maryland Computer Science Department and Institute for Advanced Computer Studies, College Park, MD 20742. E-mail: oleary@cs.umd.edu.

Bert W. Rust
National Institute of Standards and Technology, Applied and Computational Mathematics Division, NIST Stop 8910, Gaithersburg, MD 20899-8910. E-mail: bert.rust@nist.gov

1 Introduction

Fitting a model to measured data provides the most basic tool in science and engineering, essential to discovering and analyzing patterns and causality. Software that accomplishes this task should be both efficient and robust, providing reliable estimates of the model's parameters in a reasonable amount of time.

Yet data fitting problems are often quite challenging numerically. In fitting exponential models, for example, small changes in the data can make large changes in the optimal parameters. Equally serious is the fact that data fitting problems are most often nonconvex, so a set of parameters can be optimal among nearby sets of parameters without being globally optimal, and software can be fooled into accepting a suboptimal solution.

We focus in this work on fitting nonlinear models in a (weighted) least-squares sense. Most nonlinear models have some parameters – perhaps quite a few – that appear linearly. For example, in fitting a sum of two exponentials, the model for the data observations $y(t_1), \ldots, y(t_m)$ might be

$$y(t) \approx c_1 e^{\alpha_1 t} + c_2 e^{\alpha_2 t} \equiv \eta(\boldsymbol{\alpha}, \boldsymbol{c}, t). \tag{1}$$

The parameters $\boldsymbol{c} = [c_1, c_2]^T$ appear linearly, so for every choice of nonlinear parameters $\boldsymbol{\alpha} = [\alpha_1, \alpha_2]^T$, optimal values for $\boldsymbol{c}$ can be found by solving a *linear least-squares problem*, a rather simple computational task. How can this be used to our advantage? Consider the nonlinear least-squares problem

$$\min_{\boldsymbol{\alpha}, \boldsymbol{c}} \|\boldsymbol{y} - \boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c})\|_2^2, \tag{2}$$

where the ith component of the vector $\boldsymbol{y}$ is the observed value $y(t_i)$, and the ith component of the vector $\boldsymbol{\eta}$ is the model prediction at t_i $(i = 1, \ldots, m)$. Then the solution to (2) is the same as the solution to

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{y} - \boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c}(\boldsymbol{\alpha}))\|_2^2, \tag{3}$$

where $\boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c}(\boldsymbol{\alpha}))$ denotes the model predictions when, given $\boldsymbol{\alpha}$, we determine the parameter values $\boldsymbol{c}$ optimally.

This simple observation was made and exploited in 1971 by Lawton and Sylvestre [16], who credit Norman E. Dahl for the idea, in an unpublished 1965 manuscript. In 1973 Gene Golub and Victor Pereyra [12] made these same observations, but whereas Lawton and Sylvestre were satisfied with approximating derivatives for (3) using finite differences, Golub and Pereyra showed how these could be computed exactly from the derivatives of (2). This was an important step for efficiency and reliability of the method. They called (2) a *separable least squares problem* and proposed solving it using the *variable projection algorithm* (3). In 1974, Fred Krogh [15] discussed some implementation issues and noted that this approach solved problems for which the standard nonlinear least squares algorithm failed. The original FORTRAN implementation of the variable projection algorithm [11] was modified by John Bolstad [7], including changes to improve memory management, use ideas of Linda Kaufman [14] to speed up the computation of derivatives, allow weights on the

observations, and compute the covariance matrix. Bolstad's 1977 `varpro.f` has been very widely used, but, inevitably, it is showing its age. In particular, the thousand-line program relies on 1970s technology for overcoming the limitations of static storage allocation, and it uses an outdated implementation of a minimization algorithm.

A beautiful implementation of the variable projection algorithm was written for the Port Library in 1977 (revised in 1980, 1981, 1983, 1984) by Linda Kaufman and David Gay, built on Gay's high-quality nonlinear least squares solver. The code `NSGB` is well designed and runs efficiently by sacrificing easily readability. The algorithm, which allows bounds on the variables but does not allow for weighting, is spread over more than 50 files. The documentation is quite complete, but it is necessary to read many documents. As in our implementation, the underlying nonlinear least squares solver is modular and could be easily replaced if a better version became available. As in `varpro.f`, all integer auxiliary storage is packed into a single one-dimensional array, and similarly for floating point values. This was a necessary design decision for early versions of Fortran but makes it difficult to extract information about the solution. In contrast, our implementation sacrifices efficiency by using an interpreted language, but its brevity (about 160 executable lines) makes it an easily understood model for translation to other languages, and it provides statistical diagnostics (e.g., the covariance matrix) that are unavailable in the Netlib version of `NSGB`.

The history and wide range of applications of variable projection are summarized in [13] and more recently in [20]. The beauty of variable projection is that it reduces the number of parameters in the minimization problem, thus improving efficiency and possibly reducing the number of local minimizers. Convergence to the globally optimal solution is therefore more likely. Implementation of the idea is rather complicated, though, since the Jacobian matrix for (3) must be derived from that for (2). Because of this, many recent applications of the variable projection idea rely on numerical computation of derivatives [9, Chap. 3], [18]. This can cause inefficiency and unreliability.

There are two implementations of variable projection in the statistical computing language R: one by the developers of the language as part of their `nls.R` code [21], and one by Mullen and van Stokkum in their TIMP package [17] to allow more efficient computations with multiple data sets. Both implementations rely on finite difference approximations for the Jacobian matrix. Recently, Borges [8] developed and experimented with a full-Newton variant of variable projection, constructing the complete Hessian matrix rather than just using the Jacobian of the residual vector, but his implementation has not been published. The purpose of our work is to provide a 21st century sample implementation of the variable projection method (in Matlab) that allows bounds on the parameters, provides statistical diagnostics, more clearly identifies key ingredients so that improvements can be made, computes the Jacobian matrix accurately in code that is easy to understand, and makes future implementations in other languages easy. Our Matlab implementation varpro.m contains over 400 lines of comments for fewer than 160 lines of code.

2 The Structure of the Algorithm

Suppose we have measured data $y(t)$ at $t = t_1, \ldots, t_m$, and we want to determine a model so that the predicted values closely match the measured values.

A general nonlinear model can be written as

$$\eta(\boldsymbol{\alpha}, \boldsymbol{c}, t) = \sum_{j=1}^{n} c_j \Phi_j(\alpha, t), \tag{4}$$

where the functions Φ_j are the basis functions for the model. In the special case considered in (1),

$$\Phi_1(\alpha, t) = e^{\alpha_1 t}, \;\; \Phi_2(\alpha, t) = e^{\alpha_2 t}.$$

It is sometimes desirable to have one additional term in (4), $\Phi_{n+1}(\boldsymbol{\alpha}, t)$, without a corresponding c parameter. Our algorithm allows for this, but for simplicity in discussion we will ignore this term.

For physical reasons, we might have constraints on various parameters in the model; for example, we might require some of the coefficients to be nonnegative. We therefore require that

$$\begin{aligned} \boldsymbol{\alpha} &= [\alpha_1, \ldots, \alpha_q]^T \in \mathcal{S}_\alpha \subseteq \mathcal{R}^q \\ \boldsymbol{c} &= [c_1, \ldots, c_n]^T \in \mathcal{S}_c \subseteq \mathcal{R}^n, \end{aligned}$$

where $\mathcal{S}_\alpha$ and $\mathcal{S}_c$ are given domains.

Our full problem is then

$$\min_{\boldsymbol{\alpha} \in \mathcal{S}_\alpha, \boldsymbol{c} \in \mathcal{S}_c} \| \boldsymbol{W}(\boldsymbol{y} - \boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c})) \|_2^2, \tag{5}$$

where, for $i = 1, \ldots, m$, the ith component of the vector $\boldsymbol{y}$ is the observed value $y(t_i)$, and the ith component of the vector $\boldsymbol{\eta}$ is the model prediction at t_i. The diagonal matrix $\boldsymbol{W}$ contains weights, chosen to make the standard deviation of the errors in the weighted observations $w_j y(t_j)$ approximately equal.[1]

Since *linear* least squares problems are rather easy to solve, it is possible, for any choice of parameters α, to determine the corresponding optimal parameters $c(\alpha)$. We observe that the solution to (5) is the same as the solution to

$$\min_{\boldsymbol{\alpha} \in \mathcal{S}_\alpha} \| \boldsymbol{W}(\boldsymbol{y} - \boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c}(\boldsymbol{\alpha}))) \|_2^2, \tag{6}$$

where, given a fixed value of $\boldsymbol{\alpha}$, the variables $\boldsymbol{c}(\boldsymbol{\alpha})$ solve the (constrained) linear least squares problem

$$\min_{\boldsymbol{c} \in \mathcal{S}_c} \| \boldsymbol{W}(\boldsymbol{y} - \boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c})) \|_2^2.$$

[1] More general matrices $\boldsymbol{W}$ can be used to simplify the correlation structure in the errors, but in our work we assume that the matrix is diagonal.

This linear problem takes the form

$$\min_{\boldsymbol{c} \in \mathcal{S}_c} \| \boldsymbol{W}(\boldsymbol{y} - \boldsymbol{\Phi c}))\|_2^2, \tag{7}$$

where the (i, k) element of the $m \times n$ matrix $\boldsymbol{\Phi}$ is $\Phi_k(\boldsymbol{\alpha}, t_i)$. This is the basic idea of Golub and Pereyra: solve (5) by solving the problem (6) obtained by variable projection.

Our general approach is this:

Step 1: Use a high-quality (constrained) nonlinear least-squares algorithm to solve (6).

1a: Whenever a function evaluation (and possibly a Jacobian matrix) is required for (6), solve (7), using a high-quality linear least-squares algorithm.

1b: Since the most reliable nonlinear least-squares algorithms require the Jacobian matrix of partial derivatives with respect to the $\boldsymbol{\alpha}$ variables, derive these from the Jacobian matrix for the nonlinear parameters in the full problem (5), if the user can supply this information.

1c: If Jacobian information is supplied, require only the non-zero partial derivatives from the Jacobian for the full problem.

Step 2: Compute statistical diagnostics from the solution in order to help the user validate the computed parameters.

We discuss each of these ingredients in turn.

2.1 Solving Constrained Nonlinear Least-Squares Problems

Inside the original `varpro.f` program there was an implementation of both a Gauss-Newton algorithm and a Levenberg-Marquardt algorithm for solving (6). Neither algorithm included a line search or constraints on the range of values of $\boldsymbol{\alpha}$, and this contributed to stalling on difficult problems.

We advocate using the best available solver. Therefore, we believe that the solver should be external to the variable projection program so that new software can easily be swapped in. In our Matlab implementation, we chose to use Mathworks' `lsqnonlin.m`, which uses either a trust-region-reflective or a Levenberg-Marquardt algorithm. Our program functions as a wrapper to `lsqnonlin.m`, so user-specified options to this routine are passed through, and upper and lower bounds on $\boldsymbol{\alpha}$ can be specified.

If Matlab's Optimization Toolbox is not available, then another solver could be used in place of `lsqnonlin.m`; for example, `LMFnlsq.m` [2] or `LMFsolve.m` [3].

2.2 Solving Constrained Linear Least-Squares Problems

The formulation of (5) is well-defined even if there are constraints on the linear parameters $\boldsymbol{c}$, and this fact was exploited by Sima and Van Huffel [24]. For

least-squares problems with linear constraints or upper and lower bounds, a variety of algorithms are currently available [5, Sec. 1.5, 2.2, 2.3, 4.4, etc.], most commonly feasible-direction/active-set methods and interior-point methods; see, for example, MATLAB's `lsqlin`. However, having an active constraint on one of the linear parameters in (7) at the optimal solution introduces complications into the Jacobian matrix for the nonlinear parameters in (6). To avoid such complications, our sample implementation does not allow constraints in the linear problem. Any variables that require constraints should be included in $\boldsymbol{\alpha}$, not in $\boldsymbol{c}$, so that the linear least-squares problem (7) is unconstrained (i.e., $\mathcal{S}_c = \mathcal{R}^n$)

For unconstrained linear least-squares problems, it is well established that there are two good algorithmic options for dense problems: the QR factorization is fast and reliable for well-conditioned problems, while the SVD (singular value decomposition) is advised for ill-conditioned problems.[2] Intermediate in work between these algorithms is the pivoted-QR factorization, which allows a modest degree of ill-conditioning. See [6, Chap. 2] for a discussion of these options. For large, sparse problems, a Krylov-subspace method such as `lsqr` [19] is appropriate.

In our MATLAB implementation, we used a truncated SVD algorithm, which, for ill-conditioned problems, computes a minimum-norm solution to the linear least-squares problem, after setting to zero the singular values less than $m\,\epsilon_{mach}$ times the largest singular value.[3] In theory, this is never necessary for a "good" model, since the basis functions should be sufficiently linearly independent that the model matrix $\boldsymbol{\Phi}$ is not this ill-conditioned, ensuring that the linear parameters are well determined. In practice, some models do have basis functions that are nearly colinear (i.e., almost redundant), which means that the linear parameters are not well determined even though the model predictions might be. Sums of exponentials are particularly prone to this difficulty.

2.3 Computing the Jacobian Matrix

When the nonlinear least-squares solver requires an evaluation of the function $\|\boldsymbol{y}-\boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c}(\boldsymbol{\alpha}))\|_2^2$ in (6), we solve the linear least-squares problem. At the same time, the nonlinear solver might require some derivative information, obtained through evaluation of the Jacobian matrix $\boldsymbol{J}$ for the function $\boldsymbol{\eta}(\boldsymbol{\alpha}, \boldsymbol{c}(\boldsymbol{\alpha}))$. This matrix has entries

$$\boldsymbol{J}_{ik} = \frac{\partial \eta(\boldsymbol{\alpha}, \boldsymbol{c}(\boldsymbol{\alpha}), t_i)}{\partial \alpha_k}, \quad i = 1, \ldots, m, \;\; k = 1, \ldots, q.$$

Assuming that the user provides a way to evaluate the derivatives of $\boldsymbol{\Phi}$ with respect to the nonlinear parameters, we can compute this Jacobian $\boldsymbol{J}$ as described in [12] using matrices and tensors. Golub and Pereyra [13, p. R5] break

[2] Use of the normal equations gives a less reliable solution, and we do not consider it.

[3] ϵ_{mach} is the gap between 1 and the next larger floating-point number.

these formulas down by columns, and this specifies how they are implemented. To compute $\boldsymbol{J}$, we need several matrices.

First, we need user-supplied derivative information. Since, in most cases, each basis function Φ_k depends on a small subset[4] of the parameters $\boldsymbol{\alpha}$, it is customary to require the user to supply only the non-zero partial derivatives of each of the $\boldsymbol{\Phi}_j$. For simplicity in this discussion, though, we will let $\boldsymbol{D}_k$ $(k = 1, \ldots, q)$ denote $\boldsymbol{W}$ times the $m \times n$ matrix of partial derivatives of $\boldsymbol{\Phi} = [\boldsymbol{\Phi}_1(t), \ldots, \boldsymbol{\Phi}_n(t)]$ with respect to the single parameter α_k. This means that the (i, j) element of $\boldsymbol{D}_k$ is $w_i \, \partial\Phi_j / \partial\alpha_k$, evaluated at t_i. We postpone discussion of efficient representation of $\boldsymbol{D}_k$ to the next section.

Second, we need the matrix from the current linear least-squares problem:

$$\boldsymbol{\Phi}_w = \boldsymbol{W}\boldsymbol{\Phi}.$$

From solving the linear least-squares problem we have a decomposition of $\boldsymbol{\Phi}_w$ as

$$\boldsymbol{\Phi}_w = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T,$$

where the columns of the $(m \times \mathrm{rank}(\boldsymbol{\Phi}_w))$ matrix $\boldsymbol{U}$ are orthonormal, as are those of the $(\mathrm{rank}(\boldsymbol{\Phi}_w) \times \mathrm{rank}(\boldsymbol{\Phi}_w))$ matrix $\boldsymbol{V}$, and $\boldsymbol{\Sigma}$ is diagonal.[5] All of the quantities relevant to the solution can be expressed in terms of this factorization:

- The *pseudo-inverse of* $\boldsymbol{\Phi}_w$ is $\boldsymbol{\Phi}_w^{\dagger} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T$.
- The *solution to the linear least-squares problem*[6] is $\boldsymbol{c} = \boldsymbol{\Phi}_w^{\dagger}\,\boldsymbol{W}\boldsymbol{y}$.
- The *projection onto the orthogonal complement of the range of* $\boldsymbol{\Phi}_w$ is $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^T$, with $\boldsymbol{P}^T = \boldsymbol{P}$.
- The *weighted residual* $\boldsymbol{r}_w = \boldsymbol{W}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{c}) = \boldsymbol{P}\boldsymbol{y}$ is the orthogonal projection of $\boldsymbol{y}$ onto the orthogonal complement of the range of $\boldsymbol{\Phi}_w$.

Given these matrices, we compute the Jacobian in two pieces, as described in [13, p. R5], derived using the product rule for differentiation:

$$\boldsymbol{J} = -(\boldsymbol{A} + \boldsymbol{B}). \tag{8}$$

The kth column of $\boldsymbol{A}$ is

$$\begin{aligned} \boldsymbol{a}_k &= \boldsymbol{P}\boldsymbol{D}_k\boldsymbol{\Phi}_w^{\dagger}\boldsymbol{y} \\ &= \boldsymbol{P}\boldsymbol{D}_k\boldsymbol{c} \\ &= \boldsymbol{D}_k\boldsymbol{c} - \boldsymbol{U}(\boldsymbol{U}^T(\boldsymbol{D}_k\boldsymbol{c})), \qquad k = 1, \ldots, q. \end{aligned}$$

[4] For example, in (4), $\Phi_1(t) = e^{\alpha_1 t}$ depends only on α_1, and $\Phi_2(t) = e^{\alpha_2 t}$ depends only on α_2.

[5] The decomposition diagnoses any redundancy in the basis functions, so the user need not be concerned about it.

[6] This solution is unique if $\mathrm{rank}(\boldsymbol{\Phi}_w) = n$; otherwise we specify the minimum-norm solution. Again, for a "good" model, $\boldsymbol{\Phi}_w$ should have rank n, but we allow for rank deficiencies.

The kth column of $\boldsymbol{B}$ is

$$\begin{aligned}\boldsymbol{b}_k &= (\boldsymbol{P}\boldsymbol{D}_k\boldsymbol{\Phi}_w^\dagger)^T\boldsymbol{y},\\ &= (\boldsymbol{\Phi}_w^\dagger)^T\boldsymbol{D}_k^T\boldsymbol{P}^T\boldsymbol{y}\\ &= (\boldsymbol{\Phi}_w^\dagger)^T\boldsymbol{D}_k^T\boldsymbol{r}_w\\ &= \boldsymbol{U}(\boldsymbol{\Sigma}^{-1}(\boldsymbol{V}^T(\boldsymbol{D}_k^T\boldsymbol{r}_w))), \quad k=1,\ldots,q.\end{aligned}$$

Note that we have grouped the operations so that a column can be computed using only matrix-vector products, not matrix-matrix products, and the only inverse involves a diagonal matrix. In order to speed the computation of $\boldsymbol{B}$, it might be advantageous to compute $\boldsymbol{D}_k^T\boldsymbol{r}_w$ for several values of k and then apply the pseudo-inverse to the entire batch at once. Similar savings can be made in the computation of $\boldsymbol{A}$.

Linda Kaufman [14] observed that the matrix $\boldsymbol{B}$ is negligible whenever the residual $\boldsymbol{r}_w$ is small, and we certainly expect this to be true for a good model. Therefore, because $\boldsymbol{B}$ is somewhat more expensive to compute than $\boldsymbol{A}$, Bolstad's FORTRAN implementation [7] omits this term from the Jacobian computation, and good computational experience with this algorithm variant has been reported (e.g., [18]). This omission, however, is relatively more serious when we are far from the optimal value of the parameters, and this can sometimes cause the nonlinear least squares algorithm to stall unnecessarily. Because of this, we advise including $\boldsymbol{B}$ in all computations of the Jacobian.

2.4 Representation of the Jacobian Information for the Full Problem

As mentioned in the previous section, we need q matrices $\boldsymbol{D}_k$ containing the partial derivative information.

To reduce this burden on the user, and to make computation more efficient, it is better to ask for only the nonzero columns. For example, if there are p nonzero columns, we might use an $m \times p$ array `dPhi` to store them and a $2 \times p$ array `Ind` to index them, using the convention that column ℓ of `dPhi` contains the partial derivative of Φ_j with respect to α_i, evaluated at the current value of $\boldsymbol{\alpha}$, where $j = \texttt{Ind(1,}\ell\texttt{)}$ and $i = \texttt{Ind(2,}\ell\texttt{)}$. For example, if Φ_1 is a function of α_2 and α_3, and Φ_2 is a function of α_1 and α_2, then `Ind` could be defined as

$$\begin{bmatrix} 1 & 1 & 2 & 2\\ 2 & 3 & 1 & 2\end{bmatrix}.$$

In this case, the $p = 4$ columns of `dPhi` contain, in order, $\partial\Phi_1/\partial\alpha_2$, $\partial\Phi_1/\partial\alpha_3$, $\partial\Phi_2/\partial\alpha_1$, and $\partial\Phi_2/\partial\alpha_2$.

2.5 Statistical Diagnostics

Bolstad's FORTRAN implementation of the variable projection algorithm computes a covariance matrix for the parameters and an estimate of the variance

of the observations. The wrapper program `invar.f` of Rall and Funderlic [22] and the variant `invar2.f` by Wolfe, Rust, Dunn, and Brown [25] compute additional statistical information, and we advocate providing this more complete set of options, derived from their programs:

- The *weighted residual mean square*, often called the *sigma of the regression* or the *regression standard error*, is the weighted residual norm (the square root of the weighted sum squared residual (SSR), which measures variance about the model fit) divided by the square root of the number of degrees of freedom:
$$\sigma = \frac{\|\boldsymbol{r}_w\|}{\sqrt{m-n-q}}.$$
- The estimate of the *standard deviation* σ is the square-root of the weighted residual mean square.
- The $(n+q)\times(n+q)$ matrix $\boldsymbol{C}_v$ is the estimated *variance/covariance matrix* for the parameters. The linear parameters $\boldsymbol{c}$ are ordered first, followed by the nonlinear parameters $\boldsymbol{\alpha}$, and the formula is
$$\boldsymbol{C}_v = \sigma^2(\boldsymbol{H}^T\boldsymbol{H})^{-1},$$
where $\boldsymbol{H}$ contains the weighted partial derivatives with respect to the linear and nonlinear parameters:
$$\boldsymbol{H} = \boldsymbol{W}\left[\Phi,\, \boldsymbol{J}\right].$$
We use the $\boldsymbol{R}$ factor from a pivoted QR factorization of $\boldsymbol{H}$ to compute $\boldsymbol{C}_v$ efficiently and accurately.
- The estimated *correlation matrix* for the parameters is formed by dividing each element c_{ij} in the variance/covariance matrix by $\sqrt{c_{ii}c_{jj}}$.
- The estimate of the *standard deviation for parameter* i is $\sqrt{c_{ii}}$.
- The *t-ratio for each parameter* is equal to the parameter estimate divided by its estimated standard deviation.
- The *coefficient of determination* for the fit, also called the *square of the multiple correlation coefficient* or simply R^2, is computed using the SSR and the $CTSS$:
$$R^2 = 1-\frac{SSR}{CTSS} = 1-\frac{\|\boldsymbol{r}_w\|^2}{\|\boldsymbol{W}(\boldsymbol{y}-\bar{\boldsymbol{y}})\|^2}.$$
The *corrected total sum of squares* $CTSS$ measures variance about the average value for the measurements, which is
$$\bar{\boldsymbol{y}} = \frac{\boldsymbol{e}^T\boldsymbol{W}\boldsymbol{y}}{\boldsymbol{e}^T\boldsymbol{W}\boldsymbol{e}}\,\boldsymbol{e},$$
where $\boldsymbol{e}$ is the vector of all ones. A value of 0.95 for R^2, for example, indicates that 95% of the $CTTS$ is accounted for by the fit.

– The kth component of the *standardized weighted residual* is the kth component of the weighted residual divided by the estimate of its standard deviation [4, pp. 19–20].[7] The standard deviation is estimated by $\sigma\sqrt{1-h_k}$, where h_k is the kth main diagonal element of the covariance matrix for the weighted residual. This is the same as the corresponding element in $\boldsymbol{H}(\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T$. If $\boldsymbol{Q}$ is a pivoted QR factor of $\boldsymbol{H}$, then this is easily computed as the kth main diagonal element of $\boldsymbol{Q}\boldsymbol{Q}^T$

3 Experiments

We validated the code `varpro.m` on many testproblems, comparing with results obtained from the FORTRAN version. For example, on the solar cell power output analysis problem presented by Krogh [15], our results were similar to his; using his first starting point, `varpro.m` converged within 4 function evaluations to the correct weighted residual norm, while `lsqnonlin.m` failed to converge in 100 function evaluations.[8]

Next we report on the effects of the Jacobian approximation and on the reliability of `varpro.m`.

3.1 Effects of Approximating the Jacobian by $\boldsymbol{A}$

We tested the effects of neglecting $\boldsymbol{B}$ in (8), the equation for the Jacobian matrix, on the simple problem

$$\eta(t) = c_1\cos(-\alpha_1 t) + c_2\cos(-\alpha_2 t) + c_3\cos(-\alpha_3 t) + c_4\cos(-\alpha_4 t),$$

with true parameters $\boldsymbol{c} = [1,2,3,4]^T$ and $\boldsymbol{\alpha} = [3,-4,6,1]^T$. We supplied data at $t = 0.00, 0.05, \cdots, 1.00$. The results are shown in Table 1. In the table, "feval" denotes the cumulative number of function evaluations, with $f(x)$ denoting the value of the weighted residual norm-squared from (3). The optimality measure, labeled "opt", is the infinity norm of the gradient of the minimization function. We see that although $\|\boldsymbol{B}\|_2/\|\boldsymbol{J}\|_2$ is never bigger than 2%, neglecting $\boldsymbol{B}$ increases the number of iterations from 3 to 7 and the number of function evaluations from 8 to 12. Note that `varpro.m` stalls three times when $\boldsymbol{B}$ is neglected and only once when it is included[9] but ultimately converges in both cases. Therefore, neglecting the $\boldsymbol{B}$ term in the Jacobian for this problem causes additional expense in the solution process. Gay and Kaufman [10] found similar evidence in 1991 but reached a different conclusion; they reasoned that the additional expense of computing $\boldsymbol{B}$ overwhelmed the cost of the additional iterations. We believe that, since then, two factors have tipped the balance

[7] This definition differs from that used in [25].

[8] We believe that the true values of the linear parameters should be [7.4334e-08, 2.0094e-11, 8.0493e-07] rather than the [2.0094e-11, 7.4334e-08, 3.1559e-05] reported by Krogh.

[9] We show 2 digits in the table, but the output indicates that to 6-digits, no progress is made during the stalls.

Table 1 Results of neglecting $\boldsymbol{B}$ in the Jacobian matrix when fitting a sum of cosine functions

		Neglecting $\boldsymbol{B}$			Including $\boldsymbol{B}$		
Iter	feval	$f(x)$	opt	$\Vert\boldsymbol{B}\Vert/\Vert\boldsymbol{J}\Vert$	$f(x)$	opt	$\Vert\boldsymbol{B}\Vert/\Vert\boldsymbol{J}\Vert$
0	5	5.0e-5	4.2e-4	1.8e-2	5.0e-5	4.2e-4	1.4e-3
1	6	5.0e-5	4.2e-4	3.8e-3	1.0e-6	6.6e-5	1.3e-2
2	7	1.1e-5	2.8e-4	8.7e-4	1.0e-6	6.6e-5	5.9e-5
3	8	2.1e-7	1.5e-5	4.2e-2	6.9e-8	8.2e-7	5.9e-5
4	9	2.1e-7	1.5e-5	6.5e-4			
5	10	8.6e-8	5.1e-6	2.2e-3			
6	11	8.6e-8	5.1e-6	1.6e-4			
7	12	4.1e-8	1.1e-6	1.6e-4			

in the other direction. First, the efficiency of matrix operations on modern computer architectures often greatly exceeds that of function evaluations, and second, evaluating functions of practical interest often requires a simulation whose cost far outweighs the cost of forming $\boldsymbol{B}$ regardless of architecture; see, for example, our next example.

3.2 Fitting a Supernova Light Curve

Rust, O'Leary, and Mullen [23] used the FORTRAN version `varpro.f` to fit the parameters in a differential equation to light curve data measured after the explosions of several supernovae. In this example, we use just the blue passband data for one supernova, SN1991T. In our model, the light is produced from radioactive decay of nickel (with relative abundance $N_1(t)$) to cobalt ($N_2(t)$) to iron ($N_3(t)$). We model the explosion by a Weibull pulse

$$W(t;\alpha_1,\alpha_2,\alpha_3) = \frac{\alpha_2}{\alpha_3}\left(\frac{t-\alpha_1}{\alpha_3}\right)^{(\alpha_2-1)} \exp\left[-\left(\frac{t-\alpha_1}{\alpha_3}\right)^{\alpha_2}\right] ,$$

where α_1 is the time of the explosion, α_2 is a shape parameter, and α_3 is a scale parameter. The decay reactions are

$$\begin{aligned}
\frac{dN_1}{dt} &= W(t;\alpha_1,\alpha_2,\alpha_3) - \frac{1}{8.764\,\alpha_4}N_1 , & N_1(\alpha_1) = 0 ,\\
\frac{dN_2}{dt} &= \frac{1}{8.764\,\alpha_4}N_1 - \frac{1}{111.42\,\alpha_4}N_2 , & N_2(\alpha_1) = 0 ,\\
\frac{dN_3}{dt} &= \frac{1}{111.42\,\alpha_4}N_2 , & N_3(\alpha_1) = 0 .
\end{aligned}$$

The observed light is modeled as

$$\eta(\boldsymbol{\alpha},\boldsymbol{c},t) = c_1\Phi_1(t;\boldsymbol{\alpha}) + c_2\Phi_2(t;\boldsymbol{\alpha}) .$$

In this model, c_1 and c_2 are parameters and

$$\Phi_1(t;\boldsymbol{\alpha}) = \frac{1}{8.764\,\alpha_4}N_1(t) , \qquad \Phi_2(t;\boldsymbol{\alpha}) = \frac{1}{111.42\,\alpha_4}N_2(t) , \tag{9}$$

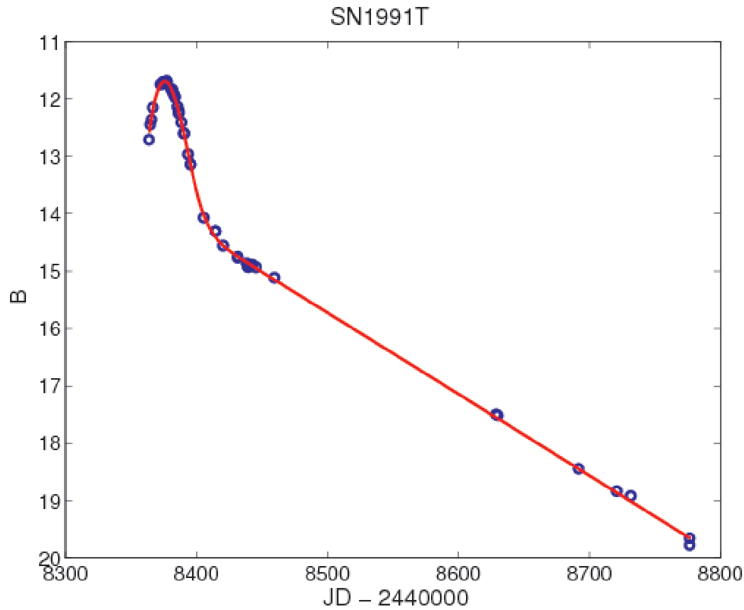

Fig. 1 The data fit for the supernova light curve problem. The time axis is measured in Julian days, shifted by 2,440,000, and the vertical axis is the magnitude of the light emission.

where Φ_1 and Φ_2 are the relative contributions to the total luminosity by the decays of nickel and cobalt, respectively. We determine the four (nonlinear) α parameters and the two linear parameters c_1 and c_2 to fit the measured data. The first data observation is at time 16.99 days, and the optimal solution is (to 1 decimal digit) $\boldsymbol{\alpha} = [5.3, 2.4, 19.5, 0.7]$, $\boldsymbol{c} = [16.8, 5.6]$. The weights in $\boldsymbol{W}$ were set to the inverses of the observed values.[10]

The system of three differential equations above is augmented by eight more equations, tracking over time the partial derivatives of N_j with respect to α_k, for $j = 1, 2$ and $k = 1, 2, 3, 4$. This gives us the information necessary to compute the Jacobian matrix. Integration is done using MATLAB's `ode23s.m`. We used default tolerances[11] for `lsqnonlin.m`, `varpro.m`, and `ode23s.m`, and a maximum of 100 Newton iterations. When convergent, this produced 4 digits of accuracy in $\boldsymbol{\alpha}$ and $\boldsymbol{c}$. The resulting fit to the data is shown in Figure 1, and statistical diagnostics for the residual are shown in Figure 2.

First we used 1000 different starting points, choosing α_1 by uniform sampling from the interval $[-2.9, 7.1]$ (which is ± 5 around the point where the

[10] This weighting is widely used (but seldom justified) when the data arises from counts. It seems appropriate when the dominant source of error is conversion of observations to numbers with a fixed number of significant digits, since the standard deviation of the error is then proportional to the observation. Other applications of this weighting are given in [1].

[11] `varpro.m` and `lsqnonlin.m` stop if the change in the parameters is less than 10^{-6} or if the change in the weighted sum of squared residuals is less than 10^{-6} times (1 plus the old value).

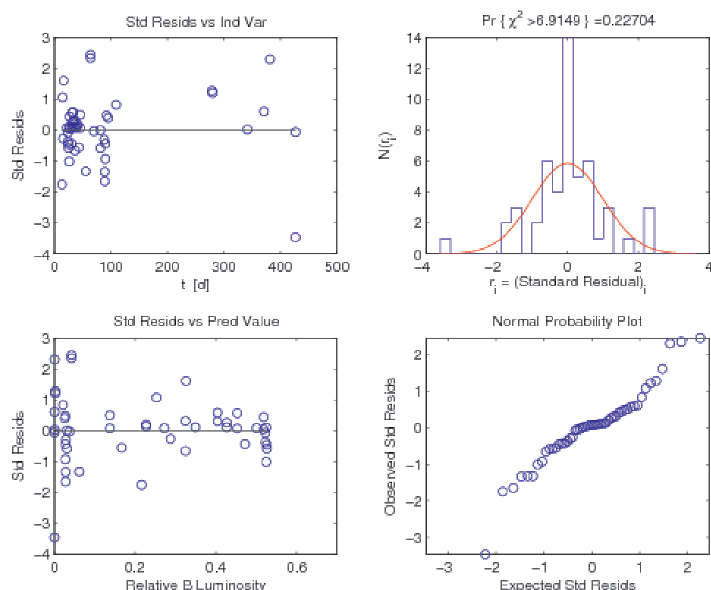

Fig. 2 Statistical diagnostics for the supernova light curve problem. The plots on the left show the standardized residuals, with the vertical scale corresponding to standard deviations from the mean. The plots on the right validate the use of least squares as a minimization function for this problem. The upper right plot shows the difference between the distribution of residuals and a normal distribution, and the deviation of the bottom plot from a straight line through 0 with slope 1 measures the deviation of the residuals from normality.

Table 2 Results of `lsqnonlin.m` and `varpro.m` on the supernova problem, difficult starting points.

	`lsqnonlin.m`	`varpro.m`
Convergent trials:		
# of convergent trials	263	345
Average runtime (seconds)	20.8	3.1
Median runtime (seconds)	1.9	0.9
Average # of fevals	45.8	13.1
Median # of fevals	22.0	12.0
Failed trials:		
# nonconvergent trials	632	211
# trials reporting no soln.	105	444

curve extrapolates to zero) and choosing the starting values for the other parameters uniformly in the interval $[0, 100]$. These starting points can be very far from the solution, and both solvers, `varpro.m` and `lsqnonlin.m`, have trouble with them. The results are summarized in Table 2, indicating how often they achieved a weighted sum of squared residuals less than 0.0729.

For these difficult starting points, the success rate for `varpro.m` in converging to the globally optimal solution was 34.5%, vs. 26.3% for `lsqnonlin.m`. The median and average run times on successful trials for `varpro.m` were much

Table 3 Results of `lsqnonlin.m` and `varpro.m` on the supernova problem, easy starting points.

	`lsqnonlin.m`	`varpro.m`
Convergent trials:		
# of convergent trials	1000	1000
Average runtime (seconds)	0.6	0.8
Median runtime (seconds)	0.5	0.6
Average # of fevals	10.6	10.5
Median # of fevals	9.0	9.0

less than for `lsqnonlin.m`, and it used many fewer function evaluations. In addition `varpro.m` was much less likely to stop with a solution other than the global minimizer (nonconvergent trials); instead, it was more likely to report failure to converge, which might signal a user to try again with a different starting point.

We then generated an easier set of starting points by uniform random sampling in the intervals

$$\alpha_1 \in [-2.9, 7.1], \alpha_2 \in [1.64, 3.68], \alpha_3 \in [13.3, 30.7], \alpha_4 \in [0.5, 1.0],$$

$$c_1 \in [10, 20], c_2 \in [3, 7].$$

These were chosen as guesses an expert might use after fitting a previous set of supernova data. Both algorithms were successful on all trials, averaging about 10.5 function evaluations per trial, with `varpro.m` having somewhat more overhead time in its formation of the Jacobian matrix.

4 Conclusions

We have presented a model implementation of the variable projection method for solving nonlinear least squares problems in which some parameters appear linearly. Our use of the SVD makes this implementation appropriate for small- and medium-scale problems; an iterative linear least squares solver could be used to extend the algorithm to large-scale problems. We have demonstrated that including the full Jacobian matrix reduces the number of iterations. We have compared the algorithm to a standard nonlinear least squares package, demonstrating better reliability on the rather difficult supernova problem. We demonstrated that the algorithm has many advantages over a full nonlinear least squares approach for scientific datafitting problems.

Acknowledgements We are grateful to Ronald F. Boisvert, Julianne Chung, David E. Gilsinn, Katharine M. Mullen, and the referee for helpful comments on the manuscript.

References

1. Amemiya, T.: Regression analysis when the variance of the dependent variable is proportional to the square of its expectation. J. of the American Statistical Assn **68**, 928–934 (1973)
2. Balda, M.: `LMFnlsq`, efficient and stable Levenberg-Marquard-Fletcher method for solving of nonlinear equations (15 Nov 2007 (Updated 27 Jan 2009))
3. Balda, M.: `LMFsolve.m`: Levenberg-Marquardt-Fletcher algorithm for nonlinear least squares problems (23 Aug 2007 (Updated 11 Feb 2009))
4. Belsley, D.A., Kuh, E., Welsch, R.E.: Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. John Wiley and Sons, New York (1980)
5. Bertsekas, D.P.: Nonlinear Programming: 2nd Edition. Athena Scientific, Nashua, NH (2004). ISBN: 1-886529-00-0
6. Björck, Å.: Numerical Methods for Least Squares Problems. Society for Industrial Mathematics, Philadelphia, PA (1996)
7. Bolstad, J.: VARPRO computer program (January 1977). Computer Science Department, Serra House, Stanford University
8. Borges, C.F.: A full-Newton approach to separable nonlinear least squares problems and its application to discrete least squares rational approximation. Electronic Trans. on Numerical Analysis **35**, 57–68 (2009)
9. Chung, J.: Numerical approaches for large-scale ill-posed inverse problems. Ph.D. thesis, Mathematics and Computer Science Department, Emory University, Atlanta, Georgia (2009)
10. Gay, D.M., Kaufman, L.: Tradeoffs in algorithms for separable nonlinear least squares. In: Proceedings of the 13th World Congress on Computational and Applied Mathematics. Criterion Press, Dublin (1991)
11. Golub, G.H., Pereyra, V.: The differentiation of pseudoinverses and nonlinear least squares problems whose variables separate. Tech. Rep. STAN-CS-72-261, Computer Science Department, Stanford University, Stanford, CA (1972)
12. Golub, G.H., Pereyra, V.: The differentiation of pseudoinverses and nonlinear least squares problems whose variables separate. SIAM Journal on Numerical Analysis **10**, 413–432 (1973)
13. Golub, G.H., Pereyra, V.: Separable nonlinear least squares: The variable projection method and its applications. Inverse Problems **19**(2), R1–R26 (2003)
14. Kaufman, L.: A variable projection method for solving separable nonlinear least squares problems. BIT Numerical Mathematics **15**(1), 49–57 (1975)
15. Krogh, F.T.: Efficient implementation of a variable projection algorithm for nonlinear least squares problems. Communications of the ACM **17**(3), 167–169 (March 1974)
16. Lawton, W.H., Sylvestre, E.A.: Estimation of linear parameters in nonlinear regression. Technometrics **13**(3), 461–467 (1971)
17. Mullen, K.M., van Stokkum, I.H.M.: TIMP: An R package for modeling multi-way spectroscopic measurements. Journal of Statistical Software **18**(3), 1–46 (2007)
18. Mullen, K.M., Vengris, M., van Stokkum, I.H.M.: Algorithms for separable nonlinear least squares with application to modelling time-resolved spectra. J. Global Optimization **38**, 201–213 (2007). DOI 10.1007/s10898-006-9071-7
19. Paige, C.C., Saunders, M.A.: LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**(1), 43–71 (1982). DOI http://doi.acm.org/10.1145/355984.355989
20. Pereyra, V., Scherer, G. (eds.): Exponential Data Fitting and Its Applications. Bentham Science Publishers Ltd. (2010)
21. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2010). URL http://www.R-project.org. ISBN 3-900051-07-0
22. Rall, J.E., Funderlic, R.E.: Interactive VARPRO (INVAR), a nonlinear least squares program. Tech. rep., Oak Ridge National Lab., TN, Report ORNL/CSD-55 (1980)
23. Rust, B.W., O'Leary, D.P., Mullen, K.M.: Modelling type 1a supernova light curves. In: V. Pereyra, G. Scherer (eds.) Exponential Data Fitting and Its Applications, pp. 169–186. Bentham Science Publishers Ltd. (2010)

24. Sima, D.M., Huffel, S.V.: Separable nonlinear least squares fitting with linear bound constraints and its application in magnetic resonance spectroscopy data quantification. J. of Computational and Applied Mathematics **203**(1), 264–278 (2007)
25. Wolfe, C.M., Rust, B.W., Dunn, J.H., Brown, I.E.: An interactive nonlinear least squares program. Tech. rep., National Bureau of Standards (now NIST), Gaithersburg, MD, NBS Technical Note 1238 (1987)

```
function [alpha, c, wresid, wresid_norm, y_est, Regression] = ...
          varpro(y, w, alpha, n, ada, lb, ub, options)
%VARPRO Solve a separable nonlinear least squares problem.
% [alpha, c, wresid, wresid_norm, y_est, Regression] =
%              VARPRO(y, w, alpha, n, ada, lb, ub, options)
%
% Given a set of m observations y(1),...,y(m)
% this program computes a weighted least squares fit using the model
%
%    eta(alpha,c,t) =
%            c_1 * phi_1 (alpha,t) + ...  + c_n * phi_n (alpha,t)
% (possibly with an extra term  + phi_{n+1} (alpha,t) ).
%
% This program determines optimal values of the q nonlinear parameters
% alpha and the n linear parameters c, given observations y at m
% different values of the "time" t and given evaluation of phi and
% (optionally) derivatives of phi.
%
% On Input:
%
%   y    m x 1   vector containing the m observations.
%   w    m x 1   vector of weights used in the least squares
%                fit.  We minimize the norm of the weighted residual
%                vector r, where, for i=1:m,
%
%                r(i) = w(i) * (y(i) - eta(alpha, c, t(i,:))).
%
%                Therefore, w(i) should be set to 1 divided by
%                the standard deviation in the measurement y(i).
%                If this number is unknown, set w(i) = 1.
%   alpha q x 1  initial estimates of the parameters alpha.
%                If alpha = [], Varpro assumes that the problem
%                is linear and returns estimates of the c parameters.
%   n            number of linear parameters c
%   ada          a function handle, described below.
%   lb    q x 1  lower bounds on the parameters alpha.
%   (Optional)   (Omit this argument or use [] if there are
%                no lower bounds.)
%   ub    q x 1  upper bounds on the parameters alpha.
%   (Optional)   (Omit this argument or use [] if there are
%                no upper bounds.)
```

```
%   options      The Matlab optimization parameter structure,
%   (Optional)   set by "optimset", to control convergence
%                tolerances, maximum number of function evaluations,
%                information displayed in the command window, etc.
%                To use default options, omit this parameter.
%                To determine the default options, type
%                    options = optimset('lsqnonlin')
%                After doing this, the defaults can be modified;
%                to modify the display option, for example, type
%                    options = optimset('lsqnonlin');
%                    optimset(options,'Display','iter');
%
% On Output:
%
%  alpha  q x 1  estimates of the nonlinear parameters.
%  c      n x 1  estimates of the linear parameters.
%  wresid m x 1  weighted residual vector, with i-th component
%                w(i) * (y(i) - eta(alpha, c, t(i,:))).
%  wresid_norm   norm of wresid.
%  y_est  m x 1  the model estimates = eta(alpha, c, t(i,:)))
%  Regression    a structure containing diagnostics about the model fit.
%                **************************************************
%                *                C a u t i o n:                  *
%                *   The theory that makes statistical            *
%                *   diagnostics useful is derived for            *
%                *   linear regression, with no upper- or         *
%                *   lower-bounds on variables.                   *
%                *   The relevance of these quantities to our     *
%                *   nonlinear model is determined by how well    *
%                *   the linearized model (Taylor series model)   *
%                *         eta(alpha_true, c_true)                *
%                *            +  Phi * (c  - c_true)              *
%                *            + dPhi * (alpha - alpha_true)       *
%                *   fits the data in the neighborhood of the     *
%                *   true values for alpha and c, where Phi       *
%                *   and dPhi contain the partial derivatives     *
%                *   of the model with respect to the c and       *
%                *   alpha parameters, respectively, and are      *
%                *   defined in ada.                              *
%                **************************************************
%  Regression.report:
%                This structure includes information on the solution
%                process, including the number of iterations,
%                termination criterion, and exitflag from lsqnonlin.
%                (Type 'help lsqnonlin' to see the exit conditions.)
%                Regression.report.rank is the computed rank of the
```

```
%                matrix for the linear subproblem.  If this equals
%                n, then the linear coefficients are well-determined.
%                If it is less than n, then although the model might
%                fit the data well, other linear coefficients might
%                give just as good a fit.
%  Regression.sigma:
%                The estimate of the standard deviation is the
%                weighted residual norm divided by the square root
%                of the number of degrees of freedom.
%                This is also called the "regression standard error"
%                or the square-root of the weighted SSR (sum squared
%                residual) divided by the square root of the
%                number of degrees of freedom.
%  Regression.RMS:
%                The "residual mean square" is equal to sigma^2:
%                RMS = wresid_norm^2 / (m-n+q)
%  Regression.coef_determ:
%                The "coefficient of determination" for the fit,
%                also called the square of the multiple correlation
%                coefficient, is sometimes called R^2.
%                It is computed as 1 - wresid_norm^2/CTSS,
%                where the "corrected total sum of squares"
%                CTSS is the norm-squared of W*(y-y_bar),
%                and the entries of y_bar are all equal to
%                (the sum of W_i^2 y_i) divided by (the sum of W_i^2).
%                A value of .95, for example, indicates that 95 per
%                cent of the CTTS is accounted for by the fit.
%
%  Regression.CovMx: (n+q) x (n+q)
%                This is the estimated variance/covariance matrix for
%                the parameters.  The linear parameters c are ordered
%                first, followed by the nonlinear parameters alpha.
%                This is empty if dPhi is not computed by ada.
%  Regression.CorMx: (n+q) x (n+q)
%                This is the estimated correlation matrix for the
%                parameters.  The linear parameters c are ordered
%                first, followed by the nonlinear parameters alpha.
%                This is empty if dPhi is not computed by ada.
%  Regression.std_param: (n+q) x 1
%                This vector contains the estimate of the standard
%                deviation for each parameter.
%                The k-th element is the square root of the k-th main
%                diagonal element of Regression.CovMx
%                This is empty if dPhi is not computed by ada.
%  Regression.t_ratio:   (n+q) x 1
%                The t-ratio for each parameter is equal to the
```

```
%                parameter estimate divided by its standard deviation.
%                (linear parameters c first, followed by alpha)
%                This is empty if dPhi is not computed by ada.
%  Regression.standardized_wresid:
%                The k-th component of the "standardized weighted
%                residual" is the k-th component of the weighted
%                residual divided by its standard deviation.
%                This is empty if dPhi is not computed by ada.
%
%---------------------------------------------------------------
% Specification of the function ada, which computes information
% related to Phi:
%
%   function [Phi,dPhi,Ind] = ada(alpha)
%
%     This function computes Phi and, if possible, dPhi.
%
%     On Input:
%
%        alpha q x 1    contains the current value of the alpha parameters.
%
%        Note:  If more input arguments are needed, use the standard
%               Matlab syntax to accomplish this.  For example, if
%               the input arguments to ada are t, z, and alpha, then
%               before calling varpro, initialize t and z, and in calling
%               varpro, replace "@ada" by "@(alpha)ada(t,z,alpha)".
%
%     On Output:
%
%        Phi   m x n1   where Phi(i,j) = phi_j(alpha,t_i).
%                       (n1 = n if there is no extra term;
%                        n1 = n+1 if an extra term is used)
%        dPhi  m x p    where the columns contain partial derivative
%                       information for Phi and p is the number of
%                       columns in Ind
%                       (or dPhi = [] if derivatives are not available).
%        Ind   2 x p    Column k of dPhi contains the partial
%                       derivative of Phi_j with respect to alpha_i,
%                       evaluated at the current value of alpha,
%                       where j = Ind(1,k) and i = Ind(2,k).
%                       Columns of dPhi that are always zero, independent
%                       of alpha, need not be stored.
%        Example:  if  phi_1 is a function of alpha_2 and alpha_3,
%                  and phi_2 is a function of alpha_1 and alpha_2, then
%                  we can set
%                          Ind = [ 1 1 2 2
```

```
%                                  2 3 1 2 ]
%                  In this case, the p=4 columns of dPhi contain
%                          d phi_1 / d alpha_2,
%                          d phi_1 / d alpha_3,
%                          d phi_2 / d alpha_1,
%                          d phi_2 / d alpha_2,
%                  evaluated at each t_i.
%                  There are no restrictions on how the columns of
%                  dPhi are ordered, as long as Ind correctly specifies
%                  the ordering.
%
%        If derivatives dPhi are not available, then set dPhi = Ind = [].
%
%---------------------------------------------------------------
%
%  Varpro calls lsqnonlin, which solves a constrained least squares
%  problem with upper and lower bounds on the constraints.  What
%  distinguishes varpro from lsqnonlin is that, for efficiency and
%  reliability, varpro causes lsqnonlin to only iterate on the
%  nonlinear parameters.  Given the information in Phi and dPhi, this
%  requires an intricate but inexpensive computation of partial
%  derivatives, and this is handled by the varpro function formJacobian.
%
%  lsqnonlin is in Matlab's Optimization Toolbox.  Another solver
%  can be substituted if the toolbox is not available.
%
%  Any parameters that require upper or lower bounds should be put in
%  alpha, not c, even if they appear linearly in the model.
%
%  The original Fortran implementation of the variable projection
%  algorithm (ref. 2) was modified in 1977 by John Bolstad
%  Computer Science Department, Serra House, Stanford University,
%  using ideas of Linda Kaufman (ref. 5) to speed up the
%  computation of derivatives.  He also allowed weights on
%  the observations, and computed the covariance matrix.
%  Our Matlab version takes advantage of 30 years of improvements
%  in programming languages and minimization algorithms.
%  In this version, we also allow upper and lower bounds on the
%  nonlinear parameters.
%
%  It is hoped that this implementation will be of use to Matlab
%  users, but also that its simplicity will make it easier for the
%  algorithm to be implemented in other languages.
%
%  This program is documented in
%  Dianne P. O'Leary and Bert W. Rust,
```

```
%  Variable Projection for Nonlinear Least Squares Problems,
%  US National Inst. of Standards and Technology, 2010.
%
%  Main reference:
%
%    0.  Gene H. Golub and V. Pereyra, 'Separable nonlinear least
%        squares: the variable projection method and its applications,'
%        Inverse Problems 19, R1-R26 (2003).
%
%  See also these papers, cited in John Bolstad's Fortran code:
%
%    1.  Gene H. Golub and V. Pereyra, 'The differentiation of
%        pseudo-inverses and nonlinear least squares problems whose
%        variables separate,' SIAM J. Numer. Anal. 10, 413-432
%         (1973).
%    2.  ------, same title, Stanford C.S. Report 72-261, Feb. 1972.
%    3.  Michael R. Osborne, 'Some aspects of non-linear least
%        squares calculations,' in Lootsma, Ed., 'Numerical Methods
%        for Non-Linear Optimization,' Academic Press, London, 1972.
%    4.  Fred Krogh, 'Efficient implementation of a variable projection
%        algorithm for nonlinear least squares problems,'
%        Comm. ACM 17:3, 167-169 (March, 1974).
%    5.  Linda Kaufman, 'A variable projection method for solving
%        separable nonlinear least squares problems', B.I.T. 15,
%        49-57 (1975).
%    6.  C. Lawson and R. Hanson, Solving Least Squares Problems,
%        Prentice-Hall, Englewood Cliffs, N. J., 1974.
%
%  These books discuss the statistical background:
%
%    7.  David A. Belsley, Edwin Kuh, and Roy E. Welsch, Regression
%        Diagnostics, John Wiley & Sons, New York, 1980, Chap. 2.
%    8.  G.A.F. Seber and C.J. Wild, Nonlinear Regression,
%        John Wiley & Sons, New York, 1989, Sec. 2.1, 5.1, and 5.2.
%
%  Dianne P. O'Leary, NIST and University of Maryland, February 2011.
%  Bert W. Rust,      NIST                            February 2011.
%  Comments updated 07-2011.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global mydebug myneglect   % test neglect of extra term in Jacobian

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Initialization: Check input, set default parameters and options.
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[m,ell] = size(y);      % m = number of observations
[m1,ell1] = size(w);

if (m1 ~= m) | (ell > 1) | (ell1 > 1)
   error('y and w must be column vectors of the same length')
end

[q,ell] = size(alpha);   % q = number of nonlinear parameters

if (ell > 1)
   error('alpha must be a column vector containing initial guesses for nonlinear parameters')
end

if (nargin < 6)
    lb = [];
else
    [q1,ell] = size(lb);
    if (q1 > 0) & (ell > 0)
       if (q1 ~= q) | (ell > 1)
          error('lb must be empty or a column vector of the same length as alpha')
       end
    end
end

if (nargin < 7)
    ub = [];
else
    [q1,ell] = size(ub);
    if (q1 > 0) & (ell > 0)
       if (q1 ~= q) | (ell > 1)
          error('ub must be empty or a column vector of the same length as alpha')
       end
    end
end

if (nargin < 8)
    options = optimset('lsqnonlin');
end

if (~strcmp(options.Display,'off'))
    disp(sprintf('\n-------------------'))
    disp(sprintf('VARPRO is beginning.'))
end
```

```
W = spdiags(w,0,m,m);  % Create an m x m diagonal matrix from the vector w

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Make the first call to ada and do some error checking.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[Phi, dPhi, Ind] = feval(ada, alpha);

[m1,n1] = size(Phi);     % n1 = number of basis functions Phi.

[m2,n2] = size(dPhi);
[ell,n3] = size(Ind);

if (m1 ~= m2) & (m2 > 0)
   error('In user function ada: Phi and dPhi must have the same number of rows.')
end

if (n1 < n) | (n1 > n+1)
   error('In user function ada: The number of columns in Phi must be n or n+1.')
end

if (n2 > 0) & (ell ~= 2)
   error('In user function ada: Ind must have two rows.')
end

if (n2 > 0) & (n2 ~= n3)
   error('In user function ada: dPhi and Ind must have the same number of columns.')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Solve the least squares problem using lsqnonlin or, if there
% are no nonlinear parameters, using the SVD procedure in formJacobian.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (q > 0) % The problem is nonlinear.

   if ~isempty(dPhi)
      options = optimset(options,'Jacobian','on');
   end

   [alpha, wresid_norm2, wresid, exitflag,output] = ...
```

```
                  lsqnonlin(@f_lsq, alpha, lb, ub, options);
   [r, Jacobian, Phi, dPhi, y_est, rank] = f_lsq(alpha);
   wresid_norm = sqrt(wresid_norm2);
   Regression.report = output;
   Regression.report.rank = rank;
   Regression.report.exitflag = exitflag;

else       % The problem is linear.

   if (~strcmp(options.Display,'off'))
      disp(sprintf('VARPRO problem is linear, since length(alpha)=0.'))
   end

   [Jacobian, c, wresid, y_est, Regression.report.rank] =  ...
                                       formJacobian(alpha, Phi, dPhi);
   wresid_norm = norm(wresid);
   wresid_norm2 = wresid_norm * wresid_norm;

end % if q > 0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Compute some statistical diagnostics for the solution.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Calculate sample variance,  the norm-squared of the residual
%    divided by the number of degrees of freedom.

sigma2 = wresid_norm2 / (m-n-q);

% Compute  Regression.sigma:
%                square-root of weighted residual norm squared divided
%                by number of degrees of freedom.

Regression.sigma = sqrt(sigma2);

% Compute  Regression.coef_determ:
%                The coeficient of determination for the fit,
%                also called the square of the multiple correlation
%                coefficient, or R^2.
%                It is computed as 1 - wresid_norm^2/CTSS,
%                where the "corrected total sum of squares"
%                CTSS is the norm-squared of W*(y-y_bar),
%                and the entries of y_bar are all equal to
%                (the sum of W_i y_i) divided by (the sum of W_i).
```

```
y_bar = sum(w.*y) / sum(w);
CTTS = norm(W * (y - y_bar)) ^2;
Regression.coef_determ = 1 - wresid_norm^2 / CTTS;

% Compute  Regression.RMS = sigma^2:
%                 the weighted residual norm divided by the number of
%                 degrees of freedom.
%                 RMS = wresid_norm / sqrt(m-n+q)

Regression.RMS = sigma2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Compute some additional statistical diagnostics for the
%  solution, if the user requested it.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (nargout==6)

   if (isempty(dPhi))

      Regression.CovMx = [];
      Regression.CorMx = [];
      Regression.std_param = [];
      Regression.t_ratio = [];
      Regression.standardized_wresid = [];

   else

% Calculate the covariance matrix CovMx, which is sigma^2 times the
% inverse of H'*H, where
%               H = W*[Phi,J]
% contains the partial derivatives of  wresid  with
% respect to the parameters in alpha and c.

      [xx,pp] = size(dPhi);
      J = zeros(m,q);
      for kk = 1:pp,
         j = Ind(1,kk);
         i = Ind(2,kk);
         if (j > n)
             J(:,i) = J(:,i) + dPhi(:,kk);
         else
             J(:,i) = J(:,i) + c(j) * dPhi(:,kk);
```

```
        end
      end
      [Qj,Rj,Pj] = qr(W*[Phi(:,1:n),J], 0);      % Uses compact pivoted QR.
      T2 = Rj \ (eye(size(Rj,1)));
      CovMx = sigma2 * T2 * T2';
      Regression.CovMx(Pj,Pj) = CovMx;  % Undo the pivoting permutation.

% Compute  Regression.CorMx:
%                estimated correlation matrix (n+q) x (n+q) for the
%                parameters.  The linear parameters are ordered
%                first, followed by the nonlinear parameters.

      d = 1 ./ sqrt(diag(Regression.CovMx));
      D = spdiags(d,0,n+q,n+q);
      Regression.CorMx = D * Regression.CovMx * D;

% Compute  Regression.std_param:
%                The k-th element is the square root of the k-th main
%                diagonal element of CovMx.

      Regression.std_param = sqrt(diag(Regression.CovMx));

% Compute  Regression.t_ratio:
%                parameter estimates divided by their standard deviations.

      alpha = reshape(alpha, q, 1);
      Regression.t_ratio = [c; alpha] .* d;

% Compute  Regression.standardized_wresid:
%                The k-th component is the k-th component of the
%                weighted residual, divided by its standard deviation.
%                Let X = W*[Phi,J],
%                    h(k) = k-th main diagonal element of covariance
%                           matrix for wresid
%                         = k-th main diagonal element of X*inv(X'*X)*X'
%                         = k-th main diagonal element of Qj*Qj'.
%                Then the standard deviation is estimated by
%                sigma*sqrt(1-h(k)).

      for k=1:m
          temp(k,1) = Qj(k,:) * Qj(k,:)';
      end
      Regression.standardized_wresid = wresid ./(Regression.sigma*sqrt(1-temp));

   end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% End of statistical diagnostics computations.
% Print some final information if desired.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if (~strcmp(options.Display,'off'))
  disp(sprintf(' '))
  disp(sprintf('VARPRO Results:'))
  disp(sprintf(' Linear Parameters:'))
  disp(sprintf(' %15.7e ',c))

  disp(sprintf(' Nonlinear Parameters:'))
  disp(sprintf(' %15.7e ',alpha))
  disp(sprintf(' '))

  disp(sprintf(' Norm-squared of weighted residual  = %15.7e',wresid_norm2))
  disp(sprintf(' Norm-squared of data vector        = %15.7e',norm(w.*y)^2))
  disp(sprintf(' Norm         of weighted residual  = %15.7e',wresid_norm))
  disp(sprintf(' Norm         of data vector        = %15.7e',norm(w.*y)))
  disp(sprintf(' Expected error of observations     = %15.7e',Regression.sigma))
  disp(sprintf(' Coefficient of determination       = %15.7e',Regression.coef_determ))
  disp(sprintf('VARPRO is finished.'))
  disp(sprintf('-------------------\n'))
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The computation is now completed.
%
% varpro uses the following two functions, f_lsq and formJacobian.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%--------------------------- Beginning of f_lsq --------------------------

function [wr_trial, Jacobian, Phi_trial, dPhi_trial, y_est,myrank] = ...
                                                         f_lsq(alpha_trial)

% function [wr_trial,Jacobian,Phi_trial,dPhi_trial,y_est,myrank] = ...
%                                                        f_lsq(alpha_trial)
%
```

```
% This function is used by lsqnonlin to compute
%          wr_trial   the current weighted residual
%          Jacobian   the Jacobian matrix for the nonlinear parameters
% It also computes
%          Phi_trial  the current Phi matrix
%          dPhi_trial the partial derivatives of Phi_trial (if available).
%          y_est      the model estimates of y
%          myrank     the rank of the matrix W*Phi in the linear problem.
%
% It uses the user-supplied function ada and the Varpro function formJacobian.

[Phi_trial, dPhi_trial, Ind] = feval(ada, alpha_trial);

[Jacobian, c, wr_trial, y_est,myrank] = ...
         formJacobian(alpha_trial, Phi_trial, dPhi_trial);

end %-------------------------- End of f_lsq ---------------------------

%---------------------- Beginning of formJacobian ----------------------

function [Jacobian, c, wresid, y_est,myrank] = formJacobian(alpha, Phi, dPhi)

% function [Jacobian, c, wresid, y_est,myrank] =  formJacobian(alpha, Phi, dPhi)
%
% This function computes the Jacobian, the linear parameters c,
% and the residual for the model with nonlinear parameters alpha.
% It is used by the functions Varpro and f_lsq.
%
% Notation: there are m data observations
%                     n1 basis functions (columns of Phi)
%                     n linear parameters c
%                       (n = n1, or n = n1 - 1)
%                     q nonlinear parameters alpha
%                     p nonzero columns of partial derivatives of Phi
%
% Input:
%
%      alpha  q x 1   the nonlinear parameters,
%      Phi    m x n1  the basis functions Phi(alpha),
%      dPhi   m x p   the partial derivatives of Phi
%
% The variables W, y, q, m, n1, and n are also used.
%
% Output:
%
%      Jacobian  m x p the Jacobian matrix, with J(i,k) = partial
```

```
%                      derivative of W resid(i) with respect to alpha(k).
%      c      n x 1 the optimal linear parameters for this choice of alpha.
%      wresid    m x 1 the weighted residual = W(y - Phi * c)
%      y_est     m x 1 the model estimates = Phi * c)
%      myrank    1 x 1 the rank of the matrix W*Phi.


% First we determine the optimal linear parameters c for
% the given values of alpha, and the resulting residual.
%
% We use the singular value decomposition to solve the linear least
% squares problem
%
%    min_{c} || W resid ||.
%       resid =  y - Phi * c.
%
% If W*Phi has any singular value less than m * its largest singular value,
% these singular values are set to zero.

[U,S,V] = svd(W*Phi(:,1:n));

% Three cases: Usually n > 1, but n = 1 and n = 0 require
% special handling (one or no linear parameters).

if (n > 1)
  s = diag(S);
elseif (n==1)
  s = S;
else    %  n = 0
 if isempty(Ind)
     Jacobian = [];
 else
     Jacobian = zeros(length(y),length(alpha));
     Jacobian(:,Ind(2,:)) = -W*dPhi;
 end
 c = [];
 y_est = Phi;
 wresid = W * (y - y_est);
 myrank = 1;
 return
end

tol = m * eps;
myrank = sum(s > tol*s(1) ); % number of singular values > tol*norm(W*Phi)
s = s(1:myrank);
```

```
if (myrank < n) & (~strcmp(options.Display,'off'))
   disp(sprintf('Warning from VARPRO:'))
   disp(sprintf('   The linear parameters are currently not well-determined.'))
   disp(sprintf('   The rank of the matrix in the subproblem is %d',myrank))
   disp(sprintf('   which is less than the n=%d linear parameters.',n))
end

yuse = y;
if (n < n1)
   yuse  = y - Phi(:,n1); % extra function Phi(:,n+1)
end
temp  = U(:,1:myrank)' * (W*yuse);
c = V(:,1:myrank) * (temp./s);
y_est = Phi(:,1:n) * c;
wresid = W * (yuse - y_est);
if (n < n1)
   y_est = y_est + Phi(:,n1);
end

if (q == 0) | (isempty(dPhi))
   Jacobian = [];
   return
end

% Second, we compute the Jacobian.
% There are two pieces, which we call Jac1 and Jac2,
% with Jacobian = - (Jac1 + Jac2).
%
% The formula for Jac1 is (P D(W*Phi) pinv(W*Phi) y,
%             and Jac2 is ((W*Phi)^+})^T (P D(W*Phi))^T y.
% where  P           is the projection onto the orthogonal complement
%                       of the range of W*Phi,
%        D(W*Phi)    is the m x n x q tensor of derivatives of W*Phi,
%        pinv(W*Phi) is the pseudo-inverse of W*Phi.
%  (See Golub&Pereyra (1973) equation (5.4).  We use their notational
%   conventions for multiplications by tensors.)
%
% Golub&Pereyra (2003), p. R5 break these formulas down by columns:
%     The j-th column of Jac1 is P D_j pinv(W*Phi) y
%                              =  P D_j c
% and the j-th column of Jac2 is (P D_j pinv(W*Phi))^T y,
%                              =  (pinv(W*Phi))^T D_j^T P^T y
%                              =  (pinv(W*Phi))^T D_j^T wresid.
% where D_j is the m x n matrix of partial derivatives of W*Phi
%     with respect to alpha(j).
```

```
% We begin the computation by precomputing
%       WdPhi, which contains the derivatives of W*Phi, and
%       WdPhi_r, which contains WdPhi' * wresid.

WdPhi = W * dPhi;
WdPhi_r = WdPhi' * wresid;
T2 = zeros(n1, q);
ctemp = c;
if (n1 > n)
   ctemp = [ctemp; 1];
end

%   Now we work column-by-column, for j=1:q.
%
%   We form Jac1 = D(W*Phi) ctemp.
%   After the loop, this matrix is multiplied by
%        P = U(:,myrank+1:m)*(U(:,myrank+1:m)'
%   to complete the computation.
%
%   We also form  T2 = (D_j(W*Phi))^T wresid  by unpacking
%   the information in WdPhi_r, using Ind.
%   After the loop, T2 is multiplied by the pseudoinverse
%       (pinv(W*Phi))^T = U(:,1:myrank) * diag(1./s) * (V(:,1:myrank)'
%   to complete the computation of Jac2.
%   Note: if n1 > n, last row of T2 is not needed.

for j=1:q,                            % for each nonlinear parameter alpha(j)
    range = find(Ind(2,:)==j);        % columns of WdPhi relevant to alpha(j)
    indrows = Ind(1,range);           % relevant rows of ctemp
    Jac1(:,j) = WdPhi(:,range) * ctemp(indrows);
    T2(indrows,j) = WdPhi_r(range);
end


Jac1 = U(:,myrank+1:m) * (U(:,myrank+1:m)' * Jac1);

T2 = diag(1 ./ s(1:myrank)) * (V(:,1:myrank)' * T2(1:n,:));
Jac2 = U(:,1:myrank) * T2;

Jacobian = -(Jac1 + Jac2);

if (mydebug)
   disp(sprintf('VARPRO norm(neglected Jacobian)/norm(Jacobian) = %e',...
         norm(Jac2,'fro')/norm(Jacobian,'fro') ))
   if (myneglect)
       disp('neglecting')
```

```
       Jacobian = -Jac1;
   end
end

end %-------------------------- End of formJacobian ----------------------

end %------------------------------ End of varpro ------------------------
```