Electronic Transactions on Numerical Analysis.
Volume 35, pp. 57-68, 2009.

ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

A FULL-NEWTON APPROACH TO SEPARABLE NONLINEAR LEAST SQUARES PROBLEMS AND ITS APPLICATION TO DISCRETE LEAST SQUARES RATIONAL APPROXIMATION*

CARLOS F. BORGES†

Abstract. We consider a class of non-linear least squares problems that are widely used in fitting experimental data. A defining characteristic of the models we will consider is that the solution parameters may be separated into two classes, those that enter the problem linearly and those that enter non-linearly. Problems of this type are known as separable non-linear least squares (SNLLS) problems and are often solved using a Gauss-Newton algorithm that was developed in Golub and Pereyra [SIAM J. Numer. Anal., 10 (1973), pp. 413–432] and has been very widely applied. We develop a full-Newton algorithm for solving this problem. Exploiting the structure of the general problem leads to a surprisingly compact algorithm which exhibits all of the excellent characteristics of the full-Newton approach (e.g. rapid convergence on problems with large residuals). Moreover, for certain problems of quite general interest, the per iteration cost for the full-Newton algorithm compares quite favorably with that of the Gauss-Newton algorithm. We explore one such problem, that of discrete least-squares fitting of rational functions.

Key words. separable nonlinear least squares, rational approximation

AMS subject classifications. 65F20, 65D10, 41A20

1. Introduction. We wish to consider fitting a set of experimental data (t_i, y_i) for $i = 1, 2, ..., m$ with a model of the form

$$y = \sum_{j=1}^{n} c_j \phi_j(\alpha, t),$$

where $\alpha = \left[\begin{array}{cccc} \alpha_1 & \alpha_2 & \ldots & \alpha_d \end{array} \right]^T$ is a set of parameters that enter non-linearly into the model, and the coefficients $\mathbf{c} = \left[\begin{array}{cccc} c_1 & c_2 & \ldots & c_n \end{array} \right]^T$ obviously enter linearly. This problem can be more usefully viewed by constructing a *model matrix* A whose i, j element is given by $A(i, j) = \phi_j(\alpha, t_i)$. It is clear that A is a function of both α and the t_i although we suppress that in the notation for the sake of clarity. As the t_i are fixed, our goal is to find values for α and $\mathbf{c}$ that minimize the length of the residual $\mathbf{r} = \mathbf{y} - A\mathbf{c}$ where $\mathbf{y} = \left[\begin{array}{cccc} y_1 & y_2 & \ldots & y_m \end{array} \right]^T$.

It is clear that for a fixed value of α the linear parameters $\mathbf{c}$ can be found by solving a linear least squares problem. Linear least squares problems are well understood and a variety of excellent methods are known for their solution [1, 6, 7].

Henceforth, we shall assume that A is a full rank $\mathbb{R}^{m \times n}$ matrix with $m > n$ and let P be the projection matrix that projects vectors onto the range of A. In particular, we let $P = AA^\dagger$ where $A^\dagger = \left(A^T A\right)^{-1} A^T$ is the Moore-Penrose generalized inverse[1] of A which can be used to solve the linear least squares problem yielding $\mathbf{c} = A^\dagger \mathbf{y}$.

We note that P is both symmetric ($P^T = P$) and idempotent ($P^2 = P$). The orthogonal projector $P^\perp$ that projects onto the null space of A^T is given by $P^\perp = I - P$.

It is clear at this point that the residual, which is in fact a function of α, is given by $\mathbf{r} = P^\perp \mathbf{y}$ and we have effectively removed the linear parameters and now need to solve a non-linear least squares problem to find α. One particularly useful technique for solving this problem is developed in [4] and involves applying the Gauss-Newton method to the variable

*Received August 30, 2008. Accepted for publication December 15, 2008. Published online on April 14, 2009. Recommended by M. Benzi.
†Department of Applied Mathematics, Naval Postgraduate School, Monterey, CA 93943 (borges@nps.edu).
[1] This is a formal derivation and it is worth noting that one should not form generalized inverses as a method for *computing* solutions to linear least-squares problems.

projection functional, in essence the norm of $P^{\perp}\mathbf{y}$. The convergence of this approach and of a variant first proposed in [8] are elegantly analyzed in [11]. This variable projection approach has found wide application in a variety of fields; see the excellent overview of the history and applications in [5]. In the sequel we shall develop a full-Newton approach to solving this problem which follows by extending the ideas in [4].

Although the algorithm we derive is quite general, the specific practical problem that motivated this development is rapid non-linear curve fitting to smaller data sets. This is a problem of serious interest in systems that use contour encoding for data compression where one must fit curves to many thousands of small data sets. One such system is developed in [10] for compression of hand-written documents and it uses the Gauss-Newton algorithm developed in [2] for fitting B-spline curves to pen strokes to achieve the needed efficiencies for a practical compression algorithm. Another related application is compressing track information for moving objects, etc. Although each individual problem may be small, the huge number of problems that need to be solved makes it worthwhile to apply more sophisticated techniques to their solution.

In light of this motivation we demonstrate the usefulness of the full-Newton approach first by briefly applying it to the fitting Bézier curves that was described in [2]. We then carefully develop a specific algorithm for discrete rational approximation in the least-squares sense which we demonstrate on a few simple examples.

2. The Newton step. The key to deriving a full Newton algorithm for solving a non-linear least squares problem is to build a quadratic model for the squared norm of the residual at the current point α_c and minimize that at each step; see, for instance, [3]. The Newton step is given by

$$\alpha_N = -H^{-1}J^T\mathbf{r}, \tag{2.1}$$

where $J = \nabla\mathbf{r}$ is the Jacobian and

$$H = J^TJ + S, \tag{2.2}$$

with

$$S = \sum_{i=1}^{m} r_i(\alpha)\nabla^2 r_i(\alpha). \tag{2.3}$$

In order to construct J and S we will need the first and second partial derivatives of $P^{\perp}$ with respect to the non-linear parameters α_i. Henceforth, we shall use the subscript i on a matrix to denote differentiation with respect to the variable α_i.

To compute the first partial we follow the derivation presented in [4]. We begin by noting that the projection matrix satisfies the equation $PA = A$. Differentiating both sides of this equation with respect to α_i yields $P_iA + PA_i = A_i$. Subtracting PA_i from both sides and regrouping yields $P_iA = (I - P)\,A_i$. Which implies that $P_iA = P^{\perp}A_i$. Multiplying both sides on the right by $A^{\dagger}$ and using $AA^{\dagger} = P$ yields $P_iP = P^{\perp}A_iA^{\dagger}$. Now transposing and exploiting symmetry on the left gives $PP_i = \left(P^{\perp}A_iA^{\dagger}\right)^T$. And finally, since P is idempotent ($P^2 = P$) differentiation yields the widely known result from [4],

$$\begin{aligned} P_i &= P_iP + PP_i \\ &= P^{\perp}A_iA^{\dagger} + \left(P^{\perp}A_iA^{\dagger}\right)^T. \end{aligned} \tag{2.4}$$

Now we need to extend this derivation in order to compute the second partial derivative. We begin by considering (2.4) and using the product rule to take its partial derivative with respect to α_j. Therefore,

$$
\begin{aligned}
P_{ij} &= \left(P^{\perp}\right)_j A_i A^{\dagger} + P^{\perp} A_{ij} A^{\dagger} + P^{\perp} A_i \left(A^{\dagger}\right)_j \\
&\qquad + \left(\left(P^{\perp}\right)_j A_i A^{\dagger} + P^{\perp} A_{ij} A^{\dagger} + P^{\perp} A_i \left(A^{\dagger}\right)_j\right)^T .
\end{aligned} \tag{2.5}
$$

Now we note that

$$\left(P^{\perp}\right)_j = (I - P)_j = -P_j,$$

which we may use to simplify (2.5) giving

$$
\begin{aligned}
P_{ij} &= -P_j A_i A^{\dagger} + P^{\perp} A_{ij} A^{\dagger} + P^{\perp} A_i \left(A^{\dagger}\right)_j \\
&\qquad + \left(-P_j A_i A^{\dagger} + P^{\perp} A_{ij} A^{\dagger} + P^{\perp} A_i \left(A^{\dagger}\right)_j\right)^T .
\end{aligned} \tag{2.6}
$$

Next, we need to compute the partial derivative of $A^{\dagger}$ with respect to α_j. We begin by differentiating both sides of $P = AA^{\dagger}$ with respect to α_j, which yields

$$P_j = A_j A^{\dagger} + A \left(A^{\dagger}\right)_j .$$

Rearranging gives

$$A \left(A^{\dagger}\right)_j = P_j - A_j A^{\dagger}.$$

Multiplying on the left by $A^{\dagger}$ and invoking the fact that $A^{\dagger} A = I$ yields[2]

$$\left(A^{\dagger}\right)_j = A^{\dagger} P_j - A^{\dagger} A_j A^{\dagger}.$$

Substituting for P_j from (2.4) yields

$$\left(A^{\dagger}\right)_j = A^{\dagger} \left(P^{\perp} A_j A^{\dagger} + \left(P^{\perp} A_j A^{\dagger}\right)^T\right) - A^{\dagger} A_j A^{\dagger}.$$

And finally, invoking $A^{\dagger} P^{\perp} = 0$ yields

$$\left(A^{\dagger}\right)_j = A^{\dagger} \left(P^{\perp} A_j A^{\dagger}\right)^T - A^{\dagger} A_j A^{\dagger}.$$

We can now substitute this expression into (2.6) giving

$$
\begin{aligned}
P_{i,j} &= -P_j A_i A^{\dagger} + P^{\perp} A_{ij} A^{\dagger} + P^{\perp} A_i A^{\dagger} \left((A_j A^{\dagger})^T P^{\perp} - A_j A^{\dagger}\right) + \\
&\quad \left(-P_j A_i A^{\dagger} + P^{\perp} A_{ij} A^{\dagger} + P^{\perp} A_i A^{\dagger} \left((A_j A^{\dagger})^T P^{\perp} - A_j A^{\dagger}\right)\right)^T .
\end{aligned} \tag{2.7}
$$

In the interest of brevity, we define

$$D_{i,j} := -P_j A_i A^{\dagger} + P^{\perp} A_{ij} A^{\dagger} + P^{\perp} A_i A^{\dagger} \left((A_j A^{\dagger})^T P^{\perp} - A_j A^{\dagger}\right).$$

Substituting this definition into (2.7) yields a compact expression for the second partial derivative of the projector. In particular,

$$P_{i,j} = D_{i,j} + D_{i,j}^T. \tag{2.8}$$

[2]The assumption that A be full-rank is important here.

Now that we have expressions for the first and second partial derivatives we may proceed to construct the Newton step. To solve (2.1) we need to construct the J and S matrices. We begin by defining two useful quantities. First, the current residual is given by

$$\mathbf{r} = P^{\perp}\mathbf{y}, \tag{2.9}$$

and second, the current solution is given by

$$\mathbf{c} = A^{\dagger}\mathbf{y}. \tag{2.10}$$

Now, since $J = \nabla\mathbf{r}$ it follows that the jth column of J is given by $-P_j\mathbf{y}$ which, invoking (2.4), is

$$-P^{\perp}A_j\mathbf{c} - (A^{\dagger})^T A_j^T\mathbf{r}. \tag{2.11}$$

To construct S we note that following (2.3) the i,j element of S is given by

$$S(i,j) = -\mathbf{r}^T P_{ij}\mathbf{y}.$$

Making use of (2.8), we find that

$$S(i,j) = -\mathbf{r}^T\left(D_{i,j} + D_{i,j}^T\right)\mathbf{y}. \tag{2.12}$$

We will evaluate this in two parts. Consider first the quantity

$$-\mathbf{r}^T D_{ij}\mathbf{y} = -\mathbf{r}^T\left(-P_jA_iA^{\dagger} + P^{\perp}A_{ij}A^{\dagger} + P^{\perp}A_iA^{\dagger}\left((A_jA^{\dagger})^TP^{\perp} - A_jA^{\dagger}\right)\right)\mathbf{y}. \tag{2.13}$$

Invoking $P^{\perp}\mathbf{r} = \mathbf{r}$ and (2.9) and (2.10), we reduce (2.13) to

$$-\mathbf{r}^T D_{ij}\mathbf{y} = \mathbf{r}^T P_jA_i\mathbf{c} - \mathbf{r}^T A_{ij}\mathbf{c} - \mathbf{r}^T A_iA^{\dagger}(A_jA^{\dagger})^T\mathbf{r} + \mathbf{r}^T A_iA^{\dagger}A_j\mathbf{c}. \tag{2.14}$$

Substituting in for P_j from (2.4) yields

$$-\mathbf{r}^T D_{ij}\mathbf{y} = \mathbf{r}^T(P^{\perp}A_jA^{\dagger} + {A^{\dagger}}^T A_j^TP^{\perp})A_i\mathbf{c} - \mathbf{r}^T A_{ij}\mathbf{c} - \mathbf{r}^T A_iA^{\dagger}(A_jA^{\dagger})^T\mathbf{r} + \mathbf{r}^T A_iA^{\dagger}A_j\mathbf{c}. \tag{2.15}$$

Now we note that $A^{\dagger}\mathbf{r} = 0$ and this reduces to

$$-\mathbf{r}^T D_{ij}\mathbf{y} = \mathbf{r}^T A_jA^{\dagger}A_i\mathbf{c} - \mathbf{r}^T A_{ij}\mathbf{c} - \mathbf{r}^T A_iA^{\dagger}(A_jA^{\dagger})^T\mathbf{r} + \mathbf{r}^T A_iA^{\dagger}A_j\mathbf{c}. \tag{2.16}$$

Next, we consider the second quantity. Omitting the details, it can be reduced to

$$\begin{aligned} -\mathbf{y}^T D_{ij}\mathbf{r} &= -\mathbf{y}^T\left(-P_jA_iA^{\dagger} + P^{\perp}A_{ij}A^{\dagger} + P^{\perp}A_iA^{\dagger}\left((A_jA^{\dagger})^TP^{\perp} - A_jA^{\dagger}\right)\right)\mathbf{r} \\ &= -\mathbf{r}^T A_iA^{\dagger}(A_jA^{\dagger})^T\mathbf{r}. \end{aligned} \tag{2.17}$$

Substituting (2.16) and (2.17) into (2.12) and combining terms yields

$$S(i,j) = \mathbf{r}^T A_jA^{\dagger}A_i\mathbf{c} + \mathbf{r}^T A_iA^{\dagger}A_j\mathbf{c} - 2\mathbf{r}^T A_i(A^TA)^{-1}A_j^T\mathbf{r} - \mathbf{r}^T A_{ij}\mathbf{c}. \tag{2.18}$$

It is important to exploit symmetries and use care when constructing the S matrix in order to minimize the operation count and enhance accuracy.

3. Implementation details. Implementation details are important. First and foremost is the careful construction of both the J and S matrices. A simple observation can greatly simplify this process; in particular, terms of the form $A_j\mathbf{c}$ and $A_j^T\mathbf{r}$ are ubiquitous in (2.11) and (2.18). We begin by constructing two $m \times d$ auxiliary matrices. We shall call the first matrix U. Its jth column is given by

$$A_j\mathbf{c}. \tag{3.1}$$

We shall call the second matrix V. Its jth column is given by

$$A_j^T\mathbf{r}. \tag{3.2}$$

Using these definitions in (2.11), we see that

$$J = -\left(P^{\perp}U + (A^{\dagger})^T V\right).$$

At this point we will introduce two intermediate quantities to simplify the notation going forward. In particular, we define

$$K = P^{\perp}U \tag{3.3}$$

$$L = (A^{\dagger})^T V. \tag{3.4}$$

With these definitions, we find that

$$J = -\left(K + L\right), \tag{3.5}$$

or, equivalently,

$$J = -\left(U - P(U - L)\right). \tag{3.6}$$

It is also true that

$$J^T J = K^T K + L^T L, \tag{3.7}$$

since the columns of L are clearly elements of the range of A, and those of K are in the null space of A^T.

Now we examine (2.18). We begin by considering the first term and noting that it can be written in terms of the jth column of V, which we shall denote by $\mathbf{v}_j$ and the ith column of U, which we shall denote by $\mathbf{u}_i$,

$$\mathbf{r}^T A_j A^{\dagger} A_i \mathbf{c} = \mathbf{v}_j^T A^{\dagger} \mathbf{u}_i,$$

which is the i, j element of the matrix $U^T L$. By the same argument, the quantity in the second term of (2.18) is the i, j element of the matrix $L^T U$. Similarly, the quantity in the third term of (2.18) is twice the i, j element of the matrix $L^T L$.

Hence, if we let $\hat{S}$ be a matrix whose i, j element is given by

$$\hat{S}(i, j) = \mathbf{r}^T A_{ij}\mathbf{c}, \tag{3.8}$$

then it follows that (2.18) leads to

$$S = U^T L + L^T U - 2L^T L - \hat{S}. \tag{3.9}$$

ETNA
Kent State University
http://etna.math.kent.edu

Using (3.7) and (3.9) in (2.2) gives

$$H = U^T L + L^T U + K^T K - L^T L - \hat{S}, \tag{3.10}$$

or, noting that $PL = L$, more compactly,

$$H = U^T U - (U - L)^T P(U - L) - \hat{S}. \tag{3.11}$$

Note that the most significant difference, in terms of computational load, between the Full Newton and the Gauss-Newton approaches is the construction of the $\hat{S}$ matrix. This requires working with $O(d^2)$ partial derivatives (both first and second order partials) as opposed to just $O(d)$ first partials for the Gauss-Newton approach. This can be significant if all of the second partial derivatives are full rank. However, in a number of applications many of the mixed partial derivatives vanish, are sparse, or have low rank, and there can be a significant savings by exploiting such structure. In Section 4, we present an example from [2] involving the fitting of a Bézier curve to ordered two-dimensional data where all of the *mixed* partial derivatives are identically zero. In this specific example we see only a 50% increase in the work per step by using a Full-Newton approach and reap all the benefits of greatly accelerated convergence. In order to maximize efficiencies for any specific model one must take full advantage of any special structure of the partial derivatives.

The other critical step is to deal with $P^\perp$ and $A^\dagger$ in a computationally responsible manner. We begin by computing the full QR factorization of the model matrix A. In particular,

$$A = [Q_1 \quad Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix} \Pi,$$

where $[Q_1 \quad Q_2]$ is an $m \times m$ orthogonal matrix, R is $n \times n$, upper-triangular, and invertible, and Π is an $n \times n$ permutation. It is easily verified that

$$\begin{aligned} K &= Q_2 Q_2^T U, \\ L &= Q_1 R^{-T} \Pi V, \end{aligned}$$

and these forms should be used in any implementation. The Newton step can then be calculated using either (3.5) or (3.6) to construct J, and either (3.10) or (3.11) to construct H. It should be noted that one clear advantage to using (3.6) in conjunction with (3.11) is that one need not compute a full QR factorization in that case, a *skinny* QR factorization will suffice. Of course, the speed advantage of, for example, a modified Gram-Schmidt approach needs to be weighed against the numerical risk if the model matrix A is poorly conditioned. We note, anecdotally, that this has not been an issue in any of the examples we have considered.

4. A brief example. As a brief example we consider the problem of fitting a Bézier curve to ordered two-dimensional data. This problem is carefully developed in [2] and we refer the reader there for the details. The basic problem can be stated as follows. Given an ordered set of data points $\{(u_i, v_i)\}_{i=1}^m$ and a non-negative integer $n < m$, find nodes $\{\alpha_i\}_{i=1}^m$ and control points $\{(x_j, y_j)\}_{j=1}^n$ that minimize

$$\sum_{i=1}^{m} \left\{ \left(u_i - \sum_{j=0}^{n} B_j^n(\alpha_i) x_j \right)^2 + \left(v_i - \sum_{j=0}^{n} B_j^n(\alpha_i) y_j \right)^2 \right\}, \tag{4.1}$$

where $B_j^n(\alpha)$ is the j'th Bernstein polynomial of degree n, that is,

$$B_j^n(\alpha) = \begin{pmatrix} n \\ j \end{pmatrix} \alpha^j (1 - \alpha)^{n-j}.$$

ETNA
Kent State University
http://etna.math.kent.edu

It is clear that the control points enter the problem linearly and the nodes are the non-linear parameters.

The critical observation is that all of the mixed partial derivatives taken with respect to the nodes are identically zero. By exploiting this fact and the other structure inherent to the problem we find that we can compute the full-Newton step using only 50% more time than computing the Gauss-Newton step. This in turn indicates that a 33% reduction in iterations will be the break even point for switching to a full-Newton code.

We tested this by using an existing Gauss-Newton code developed in [2] and modifying it to use the full-Newton step; we refer the reader to [2] for the details of the underlying algorithm. We then applied it to a data set containing 23 points taken from an experiment on a reacting chemical system which may have multiple steady states; see [9]. The experiment samples the steady state oxidation rate R achieved by a catalytic system for an input concentration of carbon monoxide C_{CO}. The data is in log-log format and we show the results of fitting with a sixth-degree Bézier curve. Note that in (4.1) this corresponds to $m = 23$ and $n = 6$. Even though the squared residual at the solution is very small (0.00217) the full-Newton algorithm converges in just 32 iterations as compared to 174 iterations for the Gauss-Newton code. This greatly enhanced convergence more than makes up for the increase in per iteration cost.

5. A full-Newton algorithm for discrete least squares rational approximation. As a specific detailed example we now consider the important problem of fitting a rational function of the form

$$y = \frac{c_1 + c_2 t + \ldots + c_n t^{n-1}}{1 + \alpha_1 t + \alpha_2 t^2 + \ldots + \alpha_k t^k}$$

to data in the least-squares sense. This is clearly a separable non-linear least squares problem with the coefficients of the numerator entering the problem linearly and those of the denominator non-linearly. The model matrix A for this problem can be written in the suggestive form $A = D^{-1}N$ where the numerator matrix N is an $m \times n$ Vandermonde matrix,

$$N = \begin{bmatrix} 1 & t_1 & \ldots & t_1^{n-1} \\ 1 & t_2 & \ldots & t_2^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & t_m & \ldots & t_m^{n-1} \end{bmatrix}, \tag{5.1}$$

and the denominator matrix D^{-1} is an $m \times m$ diagonal matrix,

$$D^{-1} = \begin{bmatrix} \frac{1}{q(t_1)} & 0 & \ldots & 0 \\ 0 & \frac{1}{q(t_2)} & \ldots & 0 \\ \vdots & \ddots & & \vdots \\ 0 & 0 & \ldots & \frac{1}{q(t_m)} \end{bmatrix},$$

where $q(t) = 1 + \alpha_1 t + \alpha_2 t^2 + \ldots + \alpha_k t^k$.

We define T to be the $m \times m$ diagonal matrix,

$$T = \begin{bmatrix} t_1 & 0 & \ldots & 0 \\ 0 & t_2 & \ldots & 0 \\ \vdots & \ddots & & \vdots \\ 0 & 0 & \ldots & t_m \end{bmatrix}.$$

ETNA
Kent State University
http://etna.math.kent.edu

It is easily verified that

$$\begin{aligned} A_j &= -D^{-2}T^jN \\ &= -T^jD^{-1}A \end{aligned} \tag{5.2}$$

and

$$\begin{aligned} A_{i,j} &= 2D^{-3}T^{i+j}N \\ &= 2T^{i+j}D^{-2}A. \end{aligned} \tag{5.3}$$

Combining (5.2) with (3.1), we see that the jth column of U is given by

$$-\,T^jD^{-1}A\mathbf{c}, \tag{5.4}$$

and also, combining (5.2) with (3.2), we see that the jth column of V is given by

$$-A^TD^{-1}T^j\mathbf{r}.$$

It will be convenient to define an intermediate quantity $\hat{V}$ whose jth column is given by

$$-\,D^{-1}T^j\mathbf{r}. \tag{5.5}$$

We note then that

$$V = A^T\hat{V}. \tag{5.6}$$

Using (5.3) in (3.8) yields

$$\begin{aligned} \hat{S}(i,j) &= 2\mathbf{r}^TT^{i+j}D^{-2}A\mathbf{c} \\ &= 2\mathbf{r}^TT^iD^{-1}T^jD^{-1}A\mathbf{c}. \end{aligned}$$

It is not difficult to see then that

$$\hat{S} = 2\hat{V}^TU. \tag{5.7}$$

Close inspection of (5.4) and (5.5) reveals that the columns of U and $\hat{V}$ are just diagonally scaled columns of a Vandermonde matrix. We introduce the matrix M defined by

$$M = \begin{bmatrix} t_1 & t_1^2 & \dots & t_1^k \\ t_2 & t_2^2 & \dots & t_2^k \\ \vdots & \vdots & & \vdots \\ t_m & t_m^2 & \dots & t_m^k \end{bmatrix}. \tag{5.8}$$

It follows that

$$U = -\Phi D^{-1}M, \tag{5.9}$$

where $\Phi = \operatorname{diag}(A\mathbf{c})$, and also that

$$\hat{V} = -\Psi D^{-1}M, \tag{5.10}$$

where $\Psi = \operatorname{diag}(\mathbf{r})$. Moreover, we note that combining (5.6) with (3.4) yields

$$\begin{aligned} L &= (A^\dagger)^TA^T\hat{V} \\ &= P\hat{V}, \end{aligned} \tag{5.11}$$

where P is the current projection, i.e., $P = Q_1 Q_1^T$. Using (5.11) in (3.6) and invoking the idempotence of P gives

$$J = -\left(U - P(U - \hat{V})\right).$$

Next, we invoke (5.9) and (5.10) to factor $D^{-1}M$ from the right to get a formula for the Jacobian,

$$J = (\Phi - P(\Phi - \Psi))\, D^{-1} M. \tag{5.12}$$

Next, using (5.7) and (5.11) in conjunction with the idempotence of P in (3.11) gives

$$H = U^T U - (U - \hat{V})^T P (U - \hat{V}) - 2\hat{V}^T U.$$

We then invoke (5.9) and (5.10) to factor $D^{-1}M$ from the right and its transpose from the left to get

$$H = M^T D^{-1} \left(\Phi^2 - (\Phi - \Psi) P (\Phi - \Psi) - 2\Phi\Psi\right) D^{-1} M, \tag{5.13}$$

or, equivalently,

$$H = M^T D^{-1} \left((\Phi - \Psi) P^{\perp} (\Phi - \Psi) - \Psi^2\right) D^{-1} M.$$

5.1. The algorithm. With the formulas necessary for computing the Newton step for rational approximation we developed a basic MATLAB code implementing the full-Newton approach. The first algorithm uses (5.12) for J and (5.13) for H. The user specifies the degree of the numerator $n-1$ and the denominator k and may supply a starting guess for the coefficients of the denominator, α_i for $i = 1, \ldots, q$. If no starting guess is given the code generates one by solving the widely known linear least squares extension of the Newton-Padé approximant. In particular, we find the least squares solution to

$$\sum_{i=1}^{m} \left(c_1 + c_2 t_i + \ldots + c_n t_i^{n-1} - y_i(\alpha_1 t + \alpha_2 t_i^2 + \ldots + \alpha_k t_i^k) - y_i\right)^2.$$

This is implemented by solving

$$\min_{\mathbf{c},\alpha} \|N\mathbf{c} - YM\alpha - \mathbf{y}\|, \tag{5.14}$$

where Y is diagonal matrix with $y_{i,i} = y_i$, and N and M are defined in (5.1) and (5.8), respectively.

Once we have a current guess for α each successive iteration begins by computing J and H using (5.12) and (5.13). We then evaluate the gradient using J and the current residual, and then solve for the Newton step using the Cholesky factorization of H. One weakness of full-Newton methods is that H may not be positive definite in a region of mixed curvature and hence the Newton direction may not be a descent direction. Although there are a number of methods for dealing with this we have implemented a very simple one that we have found to be very reliable. In particular, if the Cholesky factorization fails we regularize H by arbitrarily shifting its spectrum to the right by a bit more than the absolute value of its leftmost eigenvalue (we used $|1.2 \times \lambda_{\min}|$). Once that is done we compute the Cholesky factorization and solve for the Newton step.

Once a descent direction has been established it is essential to use some form of step size control in order to get reasonable convergence. We have chosen to use a backtracking line

ETNA
Kent State University
http://etna.math.kent.edu

search and have found this to work surprisingly well. We omit the implementation details which can be found in [3].

Finally, as a stopping criterion we terminated the algorithm when the relative change in the squared norm of the residual was less than 10^{-12}.

For purposes of comparison we also created a Gauss-Newton code, identical to the full-Newton code except that all calculations not needed for computing the Gauss-Newton direction are removed. In particular, after constructing the Jacobian with (5.12) we set $H = J^T J$ and proceed to solve for the Gauss-Newton step using the Cholesky factorization.[3] No regularization is needed in this case so that code is also removed.

5.2. Experimental results. We begin by considering two examples using measured data, both of these are taken from the NIST Statistical Reference Datasets for Nonlinear Regression.[4] The first involves measured data from an experiment on electron mobility. This data is due to R. Thurber and is considered to be of higher difficulty. There are 37 data points to be fit to a model of the form

$$y = \frac{c_0 + c_1 t + c_2 t^2 + c_3 t^3}{1 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3}.$$

We consider two scenarios in order to compare the approaches. In the first case we use the starting guess suggested in the NIST testbed of $\alpha_0 = \begin{bmatrix} 1 & .4 & .05 \end{bmatrix}^T$. In this case we converge to the known solution of $\alpha = \begin{bmatrix} .9663 & .3980 & .0497 \end{bmatrix}^T$ in 6 iterations with the full Newton code as compared to 20 iterations with the Gauss-Newton code. In this case the full Newton code resorts to regularization for the first two steps. In the second case we provide no starting guess and hence the codes generate one using (5.14). In this case the Full-Newton code converges in 7 iterations as compared to 30 for Gauss-Newton. In this case the full Newton code resorts to regularization only for the first step. It is worth noting that the squared norm of the residual at the solution is roughly 5642.7. Since it is well known that the full-Newton approach generally outperforms Gauss-Newton when the residuals are large this outcome is not surprising, although its magnitude is noteworthy.

To get a baseline notion of relative speed both codes were tested using MATLAB 7.0.4 on a Pentium(R) 4 with 504MB of RAM. Both scenarios were run and the average time per run over $10,000$ runs was calculated. In the first scenario both codes run with the starting guess above, in the second scenario both codes were run with no starting guess. The results are summarized in Table 5.1, where we show the number of steps to converge, the average time (measured in milliseconds), and the time ratio.

TABLE 5.1
Experimental Results for the Thurber data set

	Full-Newton		Gauss-Newton		Ratio
	Steps	Time	Steps	Time	
Case I	6	1.5454	20	2.9902	52%
Case II	7	1.5007	30	4.5454	33%

[3]We note that solving the normal equations is not a computationally responsible approach. However, for the purpose of the experiments that follow we found that carefully solving for the Gauss-Newton step did not change the convergence or final results but did add additional overhead which biased the time comparisons rather unfairly to the benefit of the Full-Newton algorithm. Therefore we produced all of the timings using Cholesky in order that the algorithms would be competing on a level playing field.

[4]This data is available online at www.itl.nist.gov.

ETNA
Kent State University
http://etna.math.kent.edu

The second example involves measured data from a NIST study involving scanning electron microscope line width standards. This data set is due to R. Kirby and is considered to be of average difficulty. There are 151 data points to be fit to a model of the form

$$y = \frac{c_0 + c_1 t + c_2 t^2}{1 + \alpha_1 t + \alpha_2 t^2}$$

We consider the same two scenarios in order to compare the approaches. In the first case we use the starting guess suggested in the NIST testbed of $\alpha_0 = \begin{bmatrix} -.0015 & .00002 \end{bmatrix}^T$. The results of this experiment are summarized in Table 5.2. It is worth noting that the squared norm of the residual at the solution is roughly 3.905 and we see the full-Newton approach comparing less favorably to Gauss-Newton when the residual is small as we might generally expect. It is also noteworthy that for this data set the full-Newton code did not resort to regularization in either case.

TABLE 5.2
Experimental Results for the Kirby data set

	Full-Newton		Gauss-Newton		Ratio
	Steps	Time	Steps	Time	
Case I	5	2.4767	7	3.8659	64%
Case II	4	2.2382	7	4.4928	50%

As an additional test we consider fitting a rational of the form

$$y = \frac{c_0 + c_1 t + c_2 t^2}{1 + \alpha_1 t + \alpha_2 t^2}$$

to the function $y = \sqrt{1 - x^2}$ on the interval $[-1, 1]$ and also to the function $y = \cos x$ on the interval $[-\pi, \pi]$. To give some notion of scaling we do this for three different cases: using 11, 101, and 501 evenly spaced points over the interval. Regularization was not required in any of these examples. The results are summarized in Tables 5.3 and 5.4.

TABLE 5.3
Experimental Results for the $y = \sqrt{1 - x^2}$ data set

# Points	Full-Newton		Gauss-Newton		Ratio	Residual
	Steps	Time	Steps	Time		
11	4	0.6268	5	0.8305	75%	8.91×10^{-4}
101	4	1.3843	8	2.6605	52%	3.68×10^{-2}
501	4	98.211	7	168.78	58%	8.50×10^{-2}

TABLE 5.4
Experimental Results for the $y = \cos x$ data set

# Points	Full-Newton		Gauss-Newton		Ratio	Residual
	Steps	Time	Steps	Time		
11	4	0.6356	7	0.8549	74%	2.42×10^{-2}
101	4	1.3955	7	2.3812	59%	1.30×10^{-1}
501	4	86.802	7	167.88	52%	5.94×10^{-1}

Finally, we consider a function with more complex behavior. In particular, we will try to fit $y = e^{-x \cos 4x}$ on the interval $[0, \pi]$. We begin by trying to fit the function with a rational with both numerator and denominator being degree 4 and using 20 equally spaced points. In this case the full-Newton algorithm converges in just 12 iterations with a final squared residual of 6.6916×10^{-1} and it resorts to regularization for each of the first four steps. The Gauss-Newton algorithm converges in 13 iterations but to a very unsatisfying approximation with a squared residual of 6.9470 (there are two poles inside the interval). This is not unusual in our experience; anecdotally, we have observed that the full-Newton algorithm is rather less likely to stall out far from the optimal solution.

We ran this test a second time using 100 evenly spaced points and using a higher order rational (numerator and denominator both degree 6). In this case both algorithms converge to a nice solution with a squared residual of 2.3965×10^{-1} but the full-Newton code requires only 20 iterations in contrast to the 25 required by the Gauss-Newton code, moreover it runs in just 71% of the time required by the latter. It is very interesting to note that in this experiment the full-Newton code resorts to regularization for each of the first 10 steps. However, even though it is regularizing fully half of the time it still noticeably outperforms the Gauss-Newton code.

6. Conclusion. We have derived a full-Newton approach for separable non-linear least squares problems. The derivation results in a surprisingly compact formula for computing the Newton step. Experiments show that the method can substantially improve the convergence rate at the expense of additional per iteration costs. It is seen that for problems where the second partial derivatives of the model matrix have special structure the additional costs of using a full-Newton approach may be minimal and hence the improved convergence rate can lead to substantially faster solutions. This was briefly demonstrated using an example from parametric curve fitting where all of the mixed partials are identically zero (and structured).

We then applied our derivation of the Newton step to the problem of discrete least squares rational approximation. This very important problem has a structure that leads to a surprisingly compact form for the Newton step. We showed with several examples that the full-Newton approach can significantly outperform the Gauss-Newton approach.

REFERENCES

[1] Å. BJÖRCK, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, PA, 1996.
[2] C. F. BORGES AND T. A. PASTVA, *Total least squares fitting of Bézier and B-spline curves to ordered data*, Comput. Aided Geom. Design, 19 (2002), pp. 275–289.
[3] J. E. DENNIS AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
[4] G. H. GOLUB AND V. PEREYRA, *The differentiation of pseudo-inverses and nonlinear least-squares problems whose variables separate*, SIAM J. Numer. Anal., 10 (1973), pp. 413–432.
[5] ———, *Separable nonlinear least squares: the variable projection method and its applications*, Inverse Problems, 19 (2003), pp. R1–R26.
[6] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Third ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[7] R. J. HANSON AND C. L. LAWSON, *Extensions and applications of the Householder algorithm for solving linear least squares problems*, Math. Comp., 23 (1969), pp. 787–812.
[8] L. KAUFMAN, *A variable projection method for solving separable nonlinear least squares problems*, BIT, 15 (1975), pp. 49–57.
[9] S. P. MARIN AND P. W. SMITH, *Parametric approximation of data using ODR splines*, Comput. Aided Geom. Design, 11 (1994), pp. 247–267.
[10] C. NELSON, *Contour encoded compression and transmission*, Master's thesis, Department of Computer Science, Brigham Young University, Provo, UT, 2006.
[11] A. RUHE AND P.-Å. WEDIN, *Algorithms for separable nonlinear least squares problems*, SIAM Rev., 22 (1980), pp. 318–337.