NPL Report CMSC 25/03

Report to the National Measurement System Directorate, Department of Trade and Industry

From the Software Support for Metrology Programme

Techniques for the efficient solution of large scale calibration problems

By
M G Cox, A B Forbes, P M Fossati,
P M Harris and I M Smith

May 2003

Techniques for the efficient solution of large scale calibration problems

M G Cox, A B Forbes, P M Fossati,
P M Harris and I M Smith
Centre for Mathematics and Scientific Computing

May 2003

ABSTRACT

In this report, we describe algorithms and software for solving large scale calibration problems using nonlinear least squares approaches. We consider two approaches, one in which the sparsity structure can be exploited directly using matrix factorisation methods, the other in which iterative methods are used to solve the matrix equations involved. The former applies to systems that give rise to block-angular or banded matrices, for example, while the latter have more general application. Both approaches require modest algorithmic resources and can be encoded compactly in software and we provide specifications for some of the main software components. We compare the behaviour of these algorithms on a class of generalised regression problems. We also show how the use of solvers that employ harness components can be very effective in providing an interface between the user's application and the optimisation software. We have implemented a simple Gauss-Newton solver using the harness approach and used it to solve a range of calibration problems employing both direct and iteration approaches.

ISSN 1471–0005

Authorised by Dr Dave Rayner,
Head of the Centre for Mathematics and Scientific Computing

National Physical Laboratory,
Queens Road, Teddington, Middlesex, United Kingdom TW11 0LW

Contents

1 Introduction

An important activity in metrology is the calibration of instruments and artefacts. Calibration defines a rule which converts the values output by the instrument's sensor(s) to values that can be related to the appropriate standard (SI or derived) units. Importantly, to these calibrated values it is required to assign uncertainties that reliably take into account the uncertainties of all quantities that have an influence. As a consequence, the size and complexity of the computational tasks associated with the data analysis can be significant, even for instruments that appear to be of simple design and operation. It is thus beneficial to design and implement algorithms that are efficient with respect to computation and memory. Fortunately, many of the calibration problems give rise to systems of equations with a well defined sparsity structure. In this report we describe ways in which standard algorithms can be adapted to solve these calibration problems taking into account their sparsity structure. We also provide specifications for Fortran 90 implementations of the algorithm components.

The remainder of this report is organised as follows. In section 2, we describe how the problem of determining estimates of calibration parameters can be posed a least squares problem and solved using the Gauss-Newton algorithm. We then discuss in section 3 how the Gauss-Newton algorthm can be implemented in software using the concept of harnesses to allow the sparsity structure of the problem to be exploited. In sections 4 and 5 we show how this approach can be applied to two major classes of problem in which direct solution methods can be employed to determine updates to the parameter estimates. Methods to cater for more general sparsity structure are discussed in section 6 and in section 7 we describe briefly some of the techniques in large scale optimisation. We illustrate the behaviour of some the algorithms applied to generalised regression in section 8. Our concluding remarks are given in section 9.

2 Determining estimates of the calibration parameters

In many calibration problems, the observation equation involving measurements y_i can be expressed as $y_i = \phi_i(\mathbf{a}) + \epsilon_i$, where ϕ_i is a function depending on parameters $\mathbf{a} = (a_1, \ldots, a_n)^{\mathrm{T}}$ that specify the behaviour of the instrument and ϵ_i represents the mismatch between the model specified by $\mathbf{a}$ and the measurement y_i. Given a set of measurement data $\{y_i\}_1^m$, estimates $\mathbf{a}^*$ of the calibration parameters $\mathbf{a}$ can be determined by solving

$$\min_{\mathbf{a}} \; F(\mathbf{a}) = \sum_{i=1}^{m} f_i^2(\mathbf{a}) = \mathbf{f}^{\mathrm{T}}\mathbf{f}, \tag{1}$$

where $f_i(\mathbf{a}) = y_i - \phi_i(\mathbf{a})$. Least squares techniques are appropriate for determining parameter estimates for a broad range of calibration problems [17].

2.1 Gauss-Newton algorithm

A common approach to solving least squares problems (1) is derived from the Gauss-Newton algorithm. If $\mathbf{a}$ is an estimate of the solution and J is the $m \times n$ *Jacobian matrix* defined at $\mathbf{a}$ by $J_{ij} = \partial f_i / \partial a_j$, then an updated estimate of the solution is $\mathbf{a} + \alpha\mathbf{p}$, where $\mathbf{p}$ solves the Jacobian system

$$J\mathbf{p} = -\mathbf{f}, \tag{2}$$

in the least squares sense, i.e., $\mathbf{p}$ solves

$$\min_{\mathbf{p}} \|J\mathbf{p} + \mathbf{f}\|^2, \tag{3}$$

and α is a step length chosen to ensure that adequate progress to a solution is made. Starting with an appropriate initial estimate of $\mathbf{a}$, these steps are repeated until convergence criteria are met. For well-conditioned problems and accurate data, a Gauss-Newton algorithm will usually converge quickly to the solution taking unit steps at each iteration (i.e., $\alpha = 1$). From the optimality conditions associated with (3) we can derive the *normal equations*

$$J^{\mathrm{T}}J\mathbf{p} = -J^{\mathrm{T}}\mathbf{f}, \tag{4}$$

which determines $\mathbf{p}$ as the solution of a square system of equations. However, the QR factorisation approach, described below, is generally more numerically stable.

The Gauss-Newton algorithm is derived from Newton's algorithm to minimise a function $F(\mathbf{a})$ in which the search direction $\mathbf{p}_N$ is computed as the solution of

$$H\mathbf{p}_N = -\mathbf{g}, \tag{5}$$

where $\mathbf{g} = \nabla_{\mathbf{a}} F$ is the gradient, $g_j = \partial F/\partial a_j$, and $H = \nabla^2 F$ is the *Hessian* matrix of second partial derivatives

$$H_{jl} = \frac{\partial^2 F}{\partial a_j \partial a_l}.$$

If $F(\mathbf{a})$ is a sum of squares then the Newton step is the solution of

$$\left[J^{\mathrm{T}} J + \sum_{i=1}^{m} f_i \nabla^2 f_i \right] \mathbf{p}_N = -J^{\mathrm{T}} \mathbf{f}.$$

The Gauss-Newton algorithm assumes that the contribution of $\sum_i f_i \nabla^2 f_i$ to the Hessian can be ignored and the update step is determined from the normal equations (4).

2.1.1 Solution of the Jacobian system

As noted earlier, although the Gauss-Newton step can be found by solving (4), a more numerically stable method is to find a factorisation

$$J = Q \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{6}$$

where Q is an $m \times m$ orthogonal matrix with $Q^{\mathrm{T}} Q = I_m$, the identity matrix of order m, and R is an upper-triangular matrix of order n (see, e.g., [24, Chapter 5]). If $Q = [\ Q_1\ Q_2\ ]$, where Q_1 is formed from the first n columns of Q and Q_2 from the last $m - n$ columns, the solution $\mathbf{p}$ is determined by solving the upper-triangular system

$$R\mathbf{p} = -Q_1^{\mathrm{T}} \mathbf{f},$$

using back substitution. The matrix Q can be constructed using either *Householder reflections* which process the Jacobian matrix a column at a time, or *Givens plane rotations* which process the matrix row-wise [24, section 5.1]. For either approach the orthogonal factorisation requires $O(mn^2)$ operations. The matrix-vector multiplication $Q_1^{\mathrm{T}} \mathbf{f}$ can be implemented efficiently using the fact that Q is a product of Householder reflections or Givens rotations.

We note that if J has QR factorisation as in (6) and $\mathbf{q} = Q^{\mathrm{T}}\mathbf{f}$, then the gradient vector $\mathbf{g} = \nabla_{\mathbf{a}} F$ is given by

$$\mathbf{g} = 2J^{\mathrm{T}}\mathbf{f} = 2\begin{bmatrix} R^{\mathrm{T}} & 0 \end{bmatrix} Q^{\mathrm{T}}\mathbf{f} = 2R^{\mathrm{T}}Q_1^{\mathrm{T}}\mathbf{f} = 2R^{\mathrm{T}}\mathbf{q}_1, \tag{7}$$

where $\mathbf{q}_1$ is the first n elements of $\mathbf{q}$.

2.1.2 Line search

The step length parameter α is chosen to ensure there is a sufficient decrease in the value of the objective function $F(\mathbf{a})$ at each iteration. If $\mathbf{g}$ is the gradient of F at $\mathbf{a}$ (7) and then

$$\rho(\alpha) = \frac{F(\mathbf{a} + \alpha\mathbf{p}) - F(\mathbf{a})}{\alpha\mathbf{g}^{\mathrm{T}}\mathbf{p}} \tag{8}$$

is the ratio of the actual decrease to the predicted decrease. If F is quadratic and α^* minimises F along $\mathbf{a} + \alpha\mathbf{p}$, then $\rho(\alpha^*) = 1/2$. A common line search strategy is to find a value α^* such that

$$(1 - \eta)\alpha^*\mathbf{g}^{\mathrm{T}}\mathbf{p} > F(\mathbf{a} + \alpha^*\mathbf{p}) - F(\mathbf{a}) > \eta\alpha^*\mathbf{g}^{\mathrm{T}}\mathbf{p}, \tag{9}$$

for some pre-assigned $0 < \eta < 1/2$. This two-sided requirement is designed to ensure a reduction firstly in the objective function value and secondly in the size of the directional derivative [23, section 4.4].

We note that if $J = Q_1 R$, then $\mathbf{g}^{\mathrm{T}}\mathbf{p} = 2\mathbf{q}_1^{\mathrm{T}}\mathbf{q}_1$ where $\mathbf{q}_1$ is defined as in (7).

2.2 Uncertainty matrix associated with the fitted parameters

An estimate of the uncertainty matrix $V_{\mathbf{a}}$ associated with the fitted parameters $\mathbf{a}$ is given by

$$V_{\mathbf{a}} = \hat{\sigma}^2 \left(J^{\mathrm{T}}J\right)^{-1} = \hat{\sigma}^2 \left(R^{\mathrm{T}}R\right)^{-1}, \tag{10}$$

where $\hat{\sigma}$ is an estimate of the standard deviation of the residuals.

In general, determining individual covariance information associated with the fitted parameters requires the solution of systems involving the lower triangular matrix R^{T}. For example, if b is a linear combination $b = \mathbf{h}^{\mathrm{T}}\mathbf{a}$ of the fitted parameters then the standard uncertainty of $u(b)$ of b is estimated by

$$u(b) = \hat{\sigma}\|\tilde{\mathbf{h}}\|, \quad R^{\mathrm{T}}\tilde{\mathbf{h}} = \mathbf{h}.$$

In general, determining covariance information associated with the fitted parameters requires the solution of triangular systems involving the lower triangular matrix R^{T}.

2.3 Trust regions

The line search in a Gauss-Newton algorithm is designed to improve the convergence behaviour of the algorithm. Another popular approach is to use trust regions as in the Levenberg-Marquardt algorithm [9, 27].

3 A Gauss-Newton solver using harnesses

3.1 Standard least squares solver design

Standard library software for solving nonlinear least squares problems requires the user to supply a component FG_USER to calculate the function values f_i and the Jacobian matrix J of partial derivatives. A typical specification of the user-supplied component might look like:

3.1.1 FG_USER

```
  subroutine fg_user(imode,m,n,a,f,j)
! ------------------------------------------------
! Function and Jacobian evaluation for least squares
! ------------------------------------------------
  integer, intent(inout) :: imode
  integer, intent(in)    :: m,n
  real(8), intent(in)    :: a(n)
  real(8), intent(out)   :: f(m),j(m,n)
! ------------------------------------------------
! Parameter description
! ---------------------
! IO imode     INTEGER
!              If imode = 1, fg_user is required to calculate
!                            function values f.
!                       = 2, fg_user is required to calculate
!                            Jacobian matrix J.
!
! IN m         INTEGER
!              Number of functions f_i
!
! IN n         INTEGER
!              Number of optimisation parameters.
!
! IN a         DOUBLE PRECISION 1-dimensional array
!              n x 1
!              Estimate of parameters.
!
! OU f         DOUBLE PRECISION 1-dimensional array
!              m x 1
!              Functional values.
!
! OU j         DOUBLE PRECISION 2-dimensional array
!              m x n
!              Jacobian matrix
!              If imode = 2 on entry, then on exit
!              j(i,j) should contain the partial
!              derivative of the ith function
!              with respect to the jth parameter.
```

The solver controls the iterative scheme using the user-supplied component to provide function and gradient information from which the Gauss-Newton step is calculated. However, the solver is not designed to exploit any structure in the problem and will use full matrix methods which, for structured problems, will not generally represent the most efficient approach.

The minimal information a Gauss-Newton scheme requires are the function values $\mathbf{f}$ and the Gauss-Newton step $\mathbf{p}$. Additional information is beneficial:

$\mathbf{g}^{\mathrm{T}}\mathbf{p}$ can be used to implement a more effective line search using quadratic interpolation and appropriate optimality criteria as in (9),

$\mathbf{g}$ can be used for line searches based on cubic interpolation, more effective termination criteria or as an additional search direction [2, 8].

(We noted in section 2.1.2 that $\mathbf{g}^{\mathrm{T}}\mathbf{p}$ can be calculated without having to calculate $\mathbf{g}$ explicitly.) In order to allow for the possibility of a more efficient approach, we can instead design a solver that requires a component to calculate the Gauss-Newton step, leaving the user the freedom to implement an efficient method to make this calculation. However, this is asking the user to become more involved in the nuts and bolts of the computation and is likely to be unattractive to the non-expert.

3.2 Use of harness components to provide flexibility

In a typical application, the model specific information usually involves a model function $\phi(\mathbf{a}, X)$ depending on parameters $\mathbf{a}$ and data X. Generally the user will be happy to supply a component to evaluate ϕ and the partial derivatives $\nabla_{\mathbf{a}}\phi$. (In another report, [5], we consider the use of automatic differentiation techniques to remove the requirement to provide partial derivatives.) From this information, $\mathbf{f}$ and Gauss-Newton step $\mathbf{p}$ can then be determined by a component specific with respect to the type of application but generic with respect to the model ϕ. In order to implement this, we need a solver that has two externally supplied components: the *harness* component FPG_HARNESS and the model-specific component FG_MODEL to be called by the harness component. The model-specific component is used to supply $\phi(\mathbf{a}, X)$ and $\nabla_{\mathbf{a}}\phi$, while the harness component assembles $\mathbf{f}$ and Gauss-Newton step $\mathbf{p}$ from this information.

A general purpose Gauss-Newton solver using harnesses has a specification similar to

3.2.1 GN_SOLVER_F

```
  subroutine gn_solver_f(m,n,a,f,nrmf,fpg_harness,fg_model,&
                         ...)
! ------------------------------------------------------
! Simple Gauss-Newton solver for structured problems
! ------------------------------------------------------
  integer, intent(in)    :: m,n
  real(8), intent(inout) :: a(n)
  real(8), intent(out)   :: f(m),nrmf
  external fpg_harness,fg_model
! ------------------------------------------------------
! Parameters
! ----------
! IN m          INTEGER
!               Number of functions f_i.
!               Constraint: m >= n.
!
! IN n          INTEGER
!               Number of optimisation parameters.
!
! IO a          DOUBLE PRECISION 1-dimensional array
!               n x 1
!               Optimisation parameters. On entry a contains
!               initial estimates. On exit a contains solution estimates.
!
! OU f          DOUBLE PRECISION 1-dimensional array
!               m x 1
!               Functional values.
!
! OU nrmf       DOUBLE PRECISION
!               Norm of f = sqrt(sum_i f_i^2).
!
! EX fpg_harness
!               EXTERNAL subroutine
!               See below.
!
! EX fg_model   EXTERNAL subroutine
!               See below.
```

The specification of the harness component takes the form:

3.2.2 FPG_HARNESS

```
  subroutine fpg_harness(imode,m,n,a,f,p,g,fg_model)
! ------------------------------------------------
! Function and update step evaluation for a nonlinear
! least squares
! ------------------------------------------------
  integer, intent(inout) :: imode
  integer, intent(in)    :: m,n
```

```
  real(8), intent(in)    :: a(n)
  real(8), intent(out)   :: f(m),p(n),g(n)
  external fg_model
! ------------------------------------------------
! Parameter description
! ---------------------
! IO imode     INTEGER
!              If imode = 1, fg_harness is required to calculate
!                            function values f.
!                       = 2, fg_harness is required to calculate
!                            step p and gradient g.
!
! IN m         INTEGER
!              Number of functions f_i.
!
! IN n         INTEGER
!              Number of optimisation parameters.
!
! IN a         DOUBLE PRECISION 1-dimensional array
!              n x 1
!              Estimate of parameters.
!
! OU f         DOUBLE PRECISION 1-dimensional array
!              m x 1
!              Functional values.
!
! OU p         DOUBLE PRECISION 1-dimensional array
!              n x 1
!              If imode = 2 on entry, then on exit
!              p should contain the update step for
!              a.
!
! OU g         DOUBLE PRECISION 1-dimensional array
!              n x 1
!              If imode = 2 on entry, then on exit
!              g should contain the gradient
!              g = 2*trans(J)*f, where J is the
!              m x n Jacobian matrix of partial derivatives.
!
! EX fg_model   EXTERNAL
!              External function or subroutine to be called by
!              FPG_HARNESS.
```

Since the component FG_MODEL is only to be called by the harness model, its specification can reflect the exact needs of the class of problem being solved. In practice, the data X will need to be passed to the model-specific component. In Fortran 90/95, the use of the MODULE facility is the easiest way to achieve this. In Matlab, optional parameters can be used instead.

Using the generic solver GN_SOLVER_F it is possible to construct application-specific solvers with specification of the form:

3.2.3 GN_SOLVER_A

```
  subroutine gn_solver_a(m,n,a,f,nrmf,fg_model,x...,&
                         ...)
! ------------------------------------------------------
! Gauss-Newton solver for application A
! ------------------------------------------------------
  integer, intent(in)    :: m,n
  real(8), intent(inout) :: a(n)
  real(8), intent(out)   :: f(m),nrmf
  real(8), intent(in)    :: x...
  external fg_model
! ------------------------------------------------------
! Parameters
! ----------
! IN m          INTEGER
!               Number of functions f_i.
!               Constraint: m >= n.
!
! IN n          INTEGER
!               Number of optimisation parameters.
!
! IO a          DOUBLE PRECISION 1-dimensional array
!               n x 1
!               Optimisation parameters. On entry a contains
!               initial estimates. On exit a contains solution estimates.
!
! OU f          DOUBLE PRECISION 1-dimensional array
!               m x 1
!               Functional values.
!
! OU nrmf       DOUBLE PRECISION
!               Norm of f = sqrt(sum_i f_i^2).
!
! EX fg_model   EXTERNAL subroutine
!
! IN x...       Application-specific data...
```

This *application solver* will generally store data X in a MODULE_A and then call the generic solver GN_SOLVER_F with a harness module FPG_HARNESS_A. This harness module will be specifically designed to USE the data stored in MODULE_A along with the model-specific component to determine $\mathbf{f}$, $\mathbf{p}$, etc. The user has to supply only the model-specific information to the application solver without having any need to know how the Gauss-Newton step is determined from the information supplied.

In the following sections, we give examples of how such a solver can be used to solve a range of problems, each of which have a structure that enables the Gauss-Newton step to be calculated efficiently.

4 Regular block angular structure

Block angular structured calibration problems arise when the problem parameters $\mathbf{a} = \{\mathbf{a}_j\}_0^n$ can be partitioned in such a way the the observation equations depend on $\mathbf{a}_0 = (a_{0,1}, \ldots, a_{0,n_0})^{\mathrm{T}}$ and at most one $\mathbf{a}_j$, $j > 0$. In this case, estimates of the problem parameters can be determined by solving a least squares problem of the form

$$\min_{\{\mathbf{a}_j\}_0^n} \left\{ \mathbf{f}_0^{\mathrm{T}}(\mathbf{a}_0)\mathbf{f}_0(\mathbf{a}_0) \; + \; \sum_{j=1}^{n} \mathbf{f}_j^{\mathrm{T}}(\mathbf{a}_j, \mathbf{a}_0)\mathbf{f}(\mathbf{a}_j, \mathbf{a}_0) \right\}.$$

If each $\mathbf{a}_j = (a_{j,1}, \ldots, a_{j,k})^{\mathrm{T}}$, $j > 0$, has the same number k of parameters, then we say the system has a *regular block angular* (RBA) structure. For problems with this type of structure the Jacobian matrix can be column- and row-ordered to have the form

$$J = \begin{bmatrix} J_1 & & & & J_{0,1} \\ & J_2 & & & J_{0,2} \\ & & \ddots & & \vdots \\ & & & J_n & J_{0,n} \\ & & & & J_0 \end{bmatrix},$$

where J_j represent derivative of observations with respect to $\mathbf{a}_j$ and $J_{0,j}$ their partial derivatives with respect to $\mathbf{a}_0$. This structure is reflected in the upper-triangular factor

$$R = \begin{bmatrix} R_1 & & & & B_1 \\ & R_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & R_n & B_n \\ & & & & R_0 \end{bmatrix}$$

where each R_j, $j > 0$, is $k \times k$ upper-triangular, B_j is $k \times n_0$ and R_0 is $n_0 \times n_0$ upper-triangular.

4.1 Determination of the Gauss-Newton step

4.1.1 Orthogonal factorisation scheme

The RBA structure can be exploited efficiently during the orthogonal factorisation by updating the triangular factor R by a row or block of rows at a time. If $\mathbf{f}_i(\mathbf{a}_j, \mathbf{a}_0)$ is a set of m_i observations involving parameters $\mathbf{a}_0$ and $\mathbf{a}_j$ then the partial derivatives with respect to $\mathbf{a}_j$ ($\mathbf{a}_0$) can be stored in an

$m_i \times k$ ($m_i \times n_0$) matrix J_i ($J_{0,i}$). To update R by these m_i rows we find an $(k + n_0 + m_i) \times (k + n_0 + m_i)$ orthogonal matrix Q_i such that

$$\begin{bmatrix} R_j & B_j \\ & R_0 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} := Q_i^{\mathrm{T}} \begin{bmatrix} R_j & B_j \\ & R_0 \\ J_i & J_{0,i} \end{bmatrix}.$$

At the same time we apply Q_i to the righthand side vector:

$$\begin{bmatrix} \mathbf{q}_j \\ \mathbf{q}_0 \\ \mathbf{0} \end{bmatrix} := Q_i^{\mathrm{T}} \begin{bmatrix} \mathbf{q}_j \\ \mathbf{q}_0 \\ \mathbf{f}_i \end{bmatrix}.$$

4.1.2 Updating an upper-triangular matrix by a block of rows

In order to perform efficiently a complete orthogonal factorisation of the Jacobian, we require only to be able to update a triangular matrix by a block of rows. This can be implemented easily using Householder transformations (processing column by column) or by Givens rotations (processing row by row) [24]. For example, we recall that an $m \times m$ Householder transformation is an orthogonal matrix of the form

$$P = I - \beta \mathbf{v}\mathbf{v}^{\mathrm{T}}, \quad \beta = 2/\mathbf{v}^{\mathrm{T}}\mathbf{v}.$$

Given any $m-$vector $\mathbf{x}$, there is a simple scheme to determine a Householder vector $\mathbf{v}$ such that

$$P\mathbf{x} = \mathbf{x} - \beta(\mathbf{v}^{\mathrm{T}}\mathbf{x})\mathbf{v} = \begin{bmatrix} \|\mathbf{x}\| \\ \mathbf{0} \end{bmatrix},$$

i.e., $P\mathbf{x}$ is zero in all but the first component. Geometrically, P reflects $\mathbf{x}$ onto the first coordinate axis.

Suppose we wish to update an $n \times n$ upper triangular matrix with a block of m rows. This can be achieved using n Householder transformations of dimension $m + 1$ as indicated in Figure 1. At the qth stage, an $m + 1$ Householder transformation is applied to an $(m + 1) \times (n - q + 1)$ matrix.

4.1.3 Solution of the upper-triangular system

The Gauss-Newton step $\mathbf{p} = \{\mathbf{p}_j\}_0^n$ is found by solving the triangular system

$$\begin{bmatrix} R_1 & & & & B_1 \\ & R_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & R_n & B_n \\ & & & & R_0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_n \\ \mathbf{p}_0 \end{bmatrix} = - \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_n \\ \mathbf{q}_0 \end{bmatrix}.$$

```
bbbb      ssss      ssss      ssss      ssss
bbbb      0ccc       sss       sss       sss
bbbb      0ccc       0dd        ss        ss
bbbb      0ccc       0dd        0e         s
bbbb  ->  0ccc  ->   0dd  ->    0e  ->     0
rrrr      0ccc       0dd        0e         0
 rrr       rrr       0dd        0e         0
  rr        rr        rr        0e         0
   r         r         r         r         0
```

Figure 1: Schematic representation of the update of an upper-triangular matrix by a block of rows using Householder transformations.

To achieve this efficiently we first solve

$$R_0 \mathbf{p}_0 = -\mathbf{q}_0,$$

using backwards substitution and then, for each j, solve

$$R_j \mathbf{p}_j = -\mathbf{q}_j - B_j \mathbf{p}_0,$$

again using backwards substitution. To calculate the gradient vector $\mathbf{g} = 2J^{\mathrm{T}}\mathbf{f}$, we have

$$\mathbf{g} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_n \\ \mathbf{g}_0 \end{bmatrix} = 2 \begin{bmatrix} R_1 & & & & B_1 \\ & R_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & R_n & B_n \\ & & & & R_0 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_n \\ \mathbf{q}_0 \end{bmatrix},$$

which we calculate as

$$\mathbf{g}_0 = R_0^{\mathrm{T}} \mathbf{q}_0,$$

then, for $j = 1, \ldots, n$,

$$\mathbf{g}_j = R_j^{\mathrm{T}} \mathbf{q}_j, \quad \mathbf{g}_0 = \mathbf{g}_0 + B_j^{\mathrm{T}} \mathbf{q}_j,$$

and finally $\mathbf{g} = 2\mathbf{g}$.

4.2 Calculation of elements of the uncertainty matrix associated with the fitted parameters

As indicated in section 2.2, elements of the uncertainty matrix can be determined by solving systems of equations involving R^{T}. This calculations can be made efficiently using a scheme analogous to that described

in section 4.1.3. We have implemented Fortran and Matlab components to perform the operations

$$\mathbf{x} \mapsto R\mathbf{x}, \quad \mathbf{x} \mapsto R^{\mathrm{T}}\mathbf{x}, \quad \mathbf{x} \mapsto R^{-1}\mathbf{x}, \quad \mathbf{x} \mapsto R^{-T}\mathbf{x},$$

where R is an upper-triangular, block-angular matrix stored compactly.

4.3 A solver for regular block angular systems

We can construct a solver for RBA systems using the general solver (section 3) by supplying a harness component FPG_RBA to calculate the Gauss-Newton step from information provided by the model-specific function and gradient evaluation module. We suppose the problem involves n sets of k parameters $\mathbf{a}_j$, $j = 1, \ldots, n$, along with n_0 *border parameters* $\mathbf{a}_0$ and that there are m observations that are partitioned in n_B blocks $\mathbf{f}_i$ with the ith block having m_i rows and such that each block involves at most one set of parameters $\mathbf{a}_j$, $j > 0$. To each block, we associate indices $j(i)$ specifying the set of parameters $\mathbf{a}_j$ and $m(i)$ indicating the number of rows. If $\mathbf{f}_i$ depends only on $\mathbf{a}_0$ then $j(i) = 0$. In order to implement the Gauss-Newton scheme, the user is required to calculate the function values $\mathbf{f}_i$ along with corresponding Jacobian blocks $\{J_i\}$ and $\{J_{0,i}\}$. This information can be supplied by a model-specific component with specification:

4.3.1 FG_RBA_MODEL

```
  subroutine fg_rba_model(imode,m,n,k,n0,nb,a,jm,f,ja,ja0)
! ------------------------------------------------------
! Function and gradient blocks for a regular block
! angular problem
! ------------------------------------------------------
  integer, intent(inout) :: imode
  integer, intent(in)    :: m,n,k,n0,nb
  real(8), intent(in)    :: a(n*k+n0)
  integer, intent(out)   :: jm(nb,2)
  real(8), intent(out)   :: f(m),ja(m,k),ja0(m,n0)
! ------------------------------------------------------
! IO imode     INTEGER
!              If imode = 1, fg_rba_model is required to calculate
!                            function values f.
!                       = 2, fg_rba_model is required to calculate
!                            indices jm and Jacobian blocks
!                            ja and ja0.
!
! IN m         INTEGER
!              Total number of observations.
!
! IN n         INTEGER
```

```
!              Number of diagonal parameters aj.
!
! IN k         INTEGER
!              Number of parameters in each diagonal
!              block.
!
! IN n0        INTEGER
!              Number of border parameters a0.
!
! IN nb        INTEGER
!              Number of blocks of rows in the Jacobian matrix.
!
! IN a         DOUBLE PRECISION 1-dimensional array
!              (n*k+n0) x 1
!              Estimates of the optimisation parameters
!              a = {a1,a2,...,an,a0}.
!
! OU jm        INTEGER 2-dimensional array
!              nb x 2
!              If jm(i,1:2) = (j,m_i), then the ith block of
!              rows has m_i rows and is associated with a_j.
!              Constraints: 0 <= j   <= n
!                           0 <= m_i <= m
!                           m = sum_i jm(i,2).
!
! OU f         DOUBLE PRECISION 1-dimensional array
!              m x 1
!              Functional values.
!
! OU ja        DOUBLE PRECISION 2-dimensional array
!              m x k
!              Non-zero partial derivatives of f with respect to
!              {aj}. If fi depends on aj, then
!              ja(i,1:k) should contain the partial derivatives of
!              fi with respect to aj. If fi is independent of
!              {aj} then ja(i,1:k) should be set to 0.
!
! OU ja0       DOUBLE PRECISION 2-dimensional array
!              m x n0
!              ja0(i,1:n0) should contain the partial derivatives of
!              fi with respect to border parameters a0.
```

This module requires all the Jacobian blocks $\{J_i\}$ and $\{J_{0,i}\}$ to be calculated at once. For very large problems, this could violate memory constraints. However, we can design a similar component in which these blocks are required one at a time. If necessary, this approach can be used to supply the elements row by row. The specification of this more memory-efficient component has the form:

4.3.2 FG_RBA_MODEL_I

```
  subroutine fg_rba_model_i(imode,n,k,n0,a,i,mi,ji,fi,jai,ja0i)
! ------------------------------------------------------------------
! Ith set of function and gradient blocks for a regular block
! angular problem,
! ------------------------------------------------------------------
  integer, intent(inout) :: imode
  integer, intent(in)    :: n,k,n0
  real(8), intent(in)    :: a(n*k+n0)
  integer, intent(in)    :: i,mi
  integer, intent(out)   :: ji
  real(8), intent(out)   :: fi(mi),jai(mi,k),ja0i(mi,n0)
! ------------------------------------------------------------------
! IO imode     INTEGER
!              If imode = 1, fg_rba_model_i is required to calculate
!                            the ith set of function values f.
!                       = 2, fg_rba_model_i is required to calculate
!                            the ith set of Jacobian blocks ja and ja0.
!
! IN n         INTEGER
!              Number of diagonal parameters aj.
!
! IN k         INTEGER
!              Number of parameters in each diagonal
!              block.
!
! IN n0        INTEGER
!              Number of border parameters a0.
!
! IN a         DOUBLE PRECISION 1-dimensional array
!              (n*k+n0) x 1
!              Estimates of the optimisation parameters
!              a = {a1,a2,...,an,a0}.
!
! IN i         INTEGER
!              Index for the blocks to be calculated.
!
! IN mi        INTEGER
!              Number of rows in the ith set of blocks.
!
! OU ji        INTEGER
!              The index of the diagonal block for i.
!              Constraint: 0 <= ji <= n.
!
! OU fi        DOUBLE PRECISION 1-dimensional array
!              mi x 1
!              Functional values for the ith block.
!
! OU jai       DOUBLE PRECISION 2-dimensional array
!              mi x k
!              jai(1:mi,1:k) should contain the partial derivatives of
!              fi with respect to border parameters aj.
```

```
!
! OU ja0       DOUBLE PRECISION 2-dimensional array
!              m x n0
!              ja0i(1:mi,1:n0) should contain the partial derivatives of
!              fi with respect to border parameters a0.
```

4.4 Example application: multi-station co-ordinate measuring systems

Theodolites, photogrammetry, multilateration and laser tracking systems are all examples of multi-station co-ordinate measuring systems. A set of targets at locations $\{\mathbf{x}_j\}_1^n$ are measured from station positions specified by parameters $\mathbf{t}_l$. For example, theodolite systems record the azimuth and elevation angles, multi-lateration systems [16] record the distance to the target (usually with a fixed but unknown offset), while laser tracker systems record both angles and displacement. If meaurements $\mathbf{u}_i$ relate to the $j(i)$th target and the $l(i)$th station, then the ith set of observations are of the form

$$\mathbf{f}_i(\mathbf{u}_i, \mathbf{x}_{j(i)}, \mathbf{t}_{l(i)}) = \mathbf{0},$$

and the station and target parameters can be estimated by minimising an objective function of the form

$$F(\{\mathbf{x}_j\}, \{\mathbf{t}_l\}) = \sum_i \mathbf{f}_i^2(\mathbf{u}_i, \mathbf{x}_{j(i)}, \mathbf{t}_{l(i)}). \tag{11}$$

The associated Jacobian matrix is block angular with diagonal blocks relating to the derivatives with respect to $\mathbf{x}_j$ and the border block relating to those associated with the station parameters $\{\mathbf{t}_l\}$. A schematic representation of the structure of the Jacobian matrix for a small application is given in Figure 2.

We have implemented, in Fortan 90 and Matlab, a comprehensive suite of software components for calibrating theodolite, multi-lateration and laser tracker systems using RBA techniques. The RBA approach was also used in an earlier analysis package for photogrammetric systems [13] and multi-stage artefact calibation [7, 21].

4.5 Example application: simple generalised distance regression with curves

In many experimental situations, more than one of the variables are subject to significant measurement error [17, 19]. In the case of curves, the model equations have the form

$$y_i^* = \phi(x_i^*, \mathbf{a}), \quad y_i = y_i^* + \epsilon_i, \quad x_i = x_i^* + \delta_i,$$

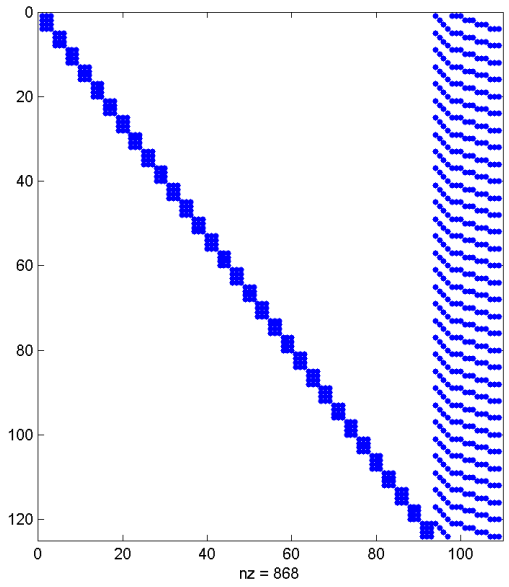

Figure 2: Block-angular structure associated with the calibration of a laser-tracker system.

where ϵ_i and δ_i represent random effects associated with measurements. In this case, estimates of the model parameters $\mathbf{a}$ along with $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m)^{\mathrm{T}}$ are determined by minimising an objective function of the form

$$\min_{\boldsymbol{\delta},\mathbf{a}} \sum_{i=1}^{m} \alpha_i^2 \delta_i^2 + \sum_{i=1}^{m} \beta_i^2 (y_i - \phi(x_i - \delta_i, \mathbf{a}))^2. \tag{12}$$

Setting

$$\mathbf{f}_i(\delta_i, \mathbf{a}) = \begin{bmatrix} \alpha_i \delta_i \\ \beta_i (y_i - \phi(x_i - \delta_i, \mathbf{a})) \end{bmatrix},$$

we see that this problem can be formulated as

$$\min_{\boldsymbol{\delta},\mathbf{a}} \sum_{i=1}^{m} \mathbf{f}_i^{\mathrm{T}}(\delta_i, \mathbf{a}) \mathbf{f}_i(\delta_i, \mathbf{a}),$$

and can be solved using the techniques described in section 4.

The model-specific information required to solve the GDR problem is confined to the evaluation of $\phi(x, \mathbf{a})$, $\partial\phi/\partial x$ and $\partial\phi/\partial a_j$. This information can be supplied by a module specified as

4.5.1 FG_SGDR_MODEL

```
! subroutine fg_sgdr_user(imode,m,n,x,a,phi,jx,ja)
! ------------------------------------------------------
! Function and gradient blocks for simple
! generalised distance regression with curves.
! ------------------------------------------------------
! integer, intent(inout) :: imode
! integer, intent(in)    :: m,n
! real(8), intent(in)    :: x(m),a(n)
! real(8), intent(out)   :: phi(m),jx(m),ja(m,n)
! ------------------------------------------------------
! IO imode     INTEGER
!              If imode = 1, fg_sgdr_model is required to calculate
!                            function values phi.
!                       = 2, fg_sgdr_model is required to calculate
!                            Jacobian blocks jx and ja.
!
! IN m         INTEGER
!              Number of data points.
!
! IN n         INTEGER
!              Number of parameters specifying the curve.
!
! IN x         DOUBLE PRECISION 1-dimensional array
!              m x 1
!              Absiscae values.
!
! IN a         DOUBLE PRECISION 1-dimensional array
!              n x 1
!              Estimates of curve parameters.
!
! OU phi       DOUBLE PRECISION 1-dimensional array
!              m x 1
!              Functional values phi(i) = phi(x(i),a).
!
! OU jx        DOUBLE PRECISION 1-dimensional array
!              m x 1
!              Partial derivative of phi(i) with respect to
!              x(i).
!
! OU ja        DOUBLE PRECISION 2-dimensional array
!              m x n
!              ja(i,1:n) should contain the partial derivatives of
!              phi(i) with respect to parameters a.
```

The user-level solver for simple GDR with curves has a specification of the form:

4.5.2 GN_SGDR_SOLVER

```
  subroutine gn_sgdr_solver(m,n,x,y,alpha,beta,a,phi,f,nrmf,fg_sgdr_model,...)
! ------------------------------------------------------------------------------
! Simple Gauss-Newton solver for simple GDR with curves.
! ------------------------------------------------------------------------------
  integer, intent(in)    :: m,n
  real(8), intent(in)    :: x(m),y(m),alpha(m),beta(m)
  real(8), intent(inout) :: a(m+n)
  real(8), intent(out)   :: phi(m),f(2*m),nrmf
  ...
```

Numerical examples of the use of this solver applied to GDR with polynomial curves [19] are given in section 8.

4.5.3 Extension to surfaces and higher dimensions

The RBA approach for curves extends to the more general problem in which the model equations have the form

$$y_i = \phi_i(\mathbf{x}_i + \boldsymbol{\delta}_i, \mathbf{a}) + \epsilon_i,$$

where ϵ_i and $\boldsymbol{\delta}_i = (\delta_{i,1}, \ldots, \delta_{i,p-1})^{\mathrm{T}}$ represent random measurement error with uncertainty matrix V_i associate with the measurements y_i and $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,p-1})^{\mathrm{T}}$. In this case, estimates of the model parameters $\mathbf{a}$ are determined by minimising an objective function of the form

$$F(\mathbf{a}, \{\boldsymbol{\delta}_i\}) = \sum_{i=1}^{m} \begin{bmatrix} \boldsymbol{\delta}_i \\ y_i - \phi_i^* \end{bmatrix}^{\mathrm{T}} V_i^{-1} \begin{bmatrix} \boldsymbol{\delta}_i \\ y_i - \phi_i^* \end{bmatrix} \tag{13}$$

where $\phi_i^* = \phi_i(\mathbf{x}_i + \boldsymbol{\delta}_i, \mathbf{a})$. If the uncertainty matrices V_i have Cholesky factorisations $V_i = L_i L_i^{\mathrm{T}}$ [24, Chapter 4], then the associated Jacobian matrix has a RBA structure with blocks $\{J_i\}$ and $\{J_{i,0}\}$ given by

$$L_i J_i = \begin{bmatrix} I \\ -\phi_{i,\mathbf{x}} \end{bmatrix}, \quad L_i J_{i,0} = \begin{bmatrix} \mathbf{0} \\ -\phi_{i,\mathbf{a}} \end{bmatrix},$$

where $\phi_{i,\mathbf{x}}$ is the vector of partial derivatives of ϕ with respect to $\mathbf{x}$ and $\phi_{i,\mathbf{a}}$ that with respect to $\mathbf{a}$ with both evaluated at $(\mathbf{x}_i + \boldsymbol{\delta}_i, \mathbf{a})$. The model specific information required can be supplied by a component to calculate $\phi(\mathbf{x}, \mathbf{a})$, $\nabla_{\mathbf{x}}\phi$ and $\nabla_{\mathbf{a}}\phi$. If L_i is diagonal, we can the exploit the fact that I is already upper-triangular to make the upper-triangularisation more efficient. An example application in co-ordinate metrology in described in [20]. The software package ODRPACK [3, 4] uses a trust region approach [27] to solve (13) for the case of diagonal L_i.

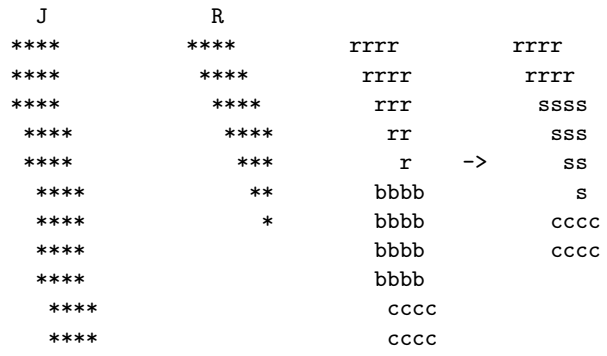

Figure 3: A schematic representation of a banded matrix J, its upper-triangular factor R and an intermediate step of an upper-triangularisation algorithm.

5 Banded structure

A banded matrix of bandwidth k is one in which the nonzero elements in any row are limited to k contiguous columns. Banded systems arise, for example, in data approximation with polynomial splines [11, 12, 17]. A schematic representation of a banded matrix is given in Figure 3.

We suppose we wish to solve a calibration problem of the form

$$\min_{\mathbf{a}} \sum_{i}^{n_B} \mathbf{f}_i^2(\mathbf{a}) \tag{14}$$

where $\mathbf{f}_i$ depends only on parameters $a_j, \ldots, a_{j+k-1}$, $j = j(i)$.

5.1 Solution of the Jacobian system

The QR factorisation of a banded matrix can be performed extremely efficiently, particularly if the rows ordered so that at the jth stage all rows with non-zero elements in columns $q < j$ have already been processed. Let $\mathbf{f}_j$ represent all the blocks of observations $\mathbf{f}_i$ for which $j(i) = j$. If $\mathbf{f}_j$ has m_j elements then the update at the jth stage involves updating a $k \times k$ upper-triangular matrix by an $m_j \times k$ matrix of partial derivatives. This can be achieved using Householder transformations as in section 4.1.3 or using Givens rotations [11, 14].

The solution of a banded triangular system is also very straightforward to implement. By exploiting the banded structure, the Gauss-Newton step can

be determine in $O(mk^2)$ steps where m is the number of observations and k is the bandwidth. Note that this is independent of the total number n of parameters.

5.2 Solver for banded systems

As for RBA systems, we can employ the solver GN_SOLVER_F to solve (14) with a model-specific component to calculate $\mathbf{f}_i$ and the corresponding blocks of partial derivatives J_i with specification

5.2.1 FG_BAND_USER

```
  subroutine fg_band_model(imode,m,n,k,nb,a,jm,f,ja)
! ------------------------------------------------------
! Function and gradient blocks for a banded problems
! ------------------------------------------------------
  integer, intent(inout) :: imode
  integer, intent(in)    :: m,n,k,nb
  real(8), intent(in)    :: a(n)
  integer, intent(out)   :: jm(nb,2)
  real(8), intent(out)   :: f(m),ja(m,k)
! ------------------------------------------------------
! IO imode     INTEGER
!              If imode = 1, fg_band_model is required to calculate
!                            function values f.
!                       = 2, fg_band_model is required to calculate
!                            indices jm and Jacobian blocks ja.
!
! IN m         INTEGER
!              Total number of observations.
!
! IN n         INTEGER
!              Number of parameters.
!
! IN k         INTEGER
!              Bandwidth.
!
! IN nb        INTEGER
!              Number of blocks of rows in the Jacobian matrix.
!
! IN a         DOUBLE PRECISION 1-dimensional array
!              n x 1
!              Estimates of the optimisation parameters.
!
! OU jm        INTEGER 2-dimensional array
!              nb x 2
!              If jm(i,1:2) = (j,m_i), then the ith block of
!              rows has m_i rows and is associated with
!              parameters a(j:j+k-1).
```

```
!              Constraints: 0 <= j   <= n-k+1
!                           0 <= m_i <= m
!                           m = sum_i jm(i,2).
!
! OU f         DOUBLE PRECISION 1-dimensional array
!              m x 1
!              Functional values.
!
! OU ja        DOUBLE PRECISION 2-dimensional array
!              m x k
!              Non-zero partial derivatives of f with respect to
!              a. If f(i) depends on a(j:j+k-1), then
!              ja(i,1:k) should contain the partial derivatives of
!              f(i) with respect to a(j:j+k-1).
```

This approach can be modified so that the user only has to supply a block of rows at a time as in the component FG_RBA_MODEL_I (section 4.3.2).

5.3 Other structures that lead to efficient QR factorisation schemes

So far we have considered block-angular matrices (section 4) and banded matrices (section 5). The same type of triangularisation strategy can also be applied, for example, to banded, bordered matrices that arise in regression problems using periodic polynomial splines [11, 15], and block-angular matrices with a banded border that arise in generalised distance regression with splines [6]. If the upper-triangular factor of a matrix has a well-defined sparsity structure then it should be possible to derive an upper-triangularisation strategy that can exploit that structure.

In section 6 below, we consider another approach for matrices with a more general sparsity structure for which the upper-triangular factor is likely to be full.

6 Block sparsity

The QR factorisation approaches to solve the Jacobian system (2) are termed direct approaches in that the solution can be determined by following a prescribed number of steps. An alternative to the direct approaches is to use iterative procedures based on conjugate gradients.

6.1 The conjugate gradient algorithm for linear equations

Suppose C is an $n \times n$ symmetric, positive definite matrix, $\mathbf{b}$ an $n \times 1$ vector and that we wish to solve

$$C\mathbf{a} = \mathbf{b}.$$

Starting with an initial guess $\mathbf{a}_0$ and a convergence tolerance τ, the conjugate gradient algorithm [24, Chapter 10] determines a solution as follows:

I Set $k = 0$, $\mathbf{r}_0 = \mathbf{b} - C\mathbf{a}_0$.

II While $\|\mathbf{r}_k\| > \tau$,

i Set $k = k + 1$.

ii Determine a search direction. If $k = 1$ set $\mathbf{p}_1 = \mathbf{r}_0$, otherwise set

$$\beta_k = \|\mathbf{r}_{k-1}\|^2 / \|\mathbf{r}_{k-2}\|^2, \quad \mathbf{p}_k = \mathbf{r}_{k-1} + \beta_k \mathbf{p}_{k-1}.$$

iii Matrix-vector product. Set $\mathbf{s}_k = C\mathbf{p}_k$.

iv Determine step length

$$\alpha_k = \|\mathbf{r}_{k-1}\|^2 / \mathbf{p}_k^{\mathrm{T}} \mathbf{s}_k.$$

v Update

$$\mathbf{a}_k = \mathbf{a}_{k-1} + \alpha_k \mathbf{p}_k, \quad \mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{s}_k.$$

III Set $\mathbf{a} = \mathbf{a}_k$ and finish.

In exact arithmetic, a) the search directions are conjugate with respect to C, i.e.,

$$\mathbf{p}_j^{\mathrm{T}} C \mathbf{p}_k = 0, \quad j \neq k,$$

b) the set of search directions $\{\mathbf{p}_1, \ldots, \mathbf{p}_k\}$ are linearly independent and have the same span as $\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\}$, and c) $\mathbf{p}_j^{\mathrm{T}} \mathbf{r}_k = 0$, $j = 1, \ldots, k$, so that $\mathbf{r}_n = \mathbf{0}$. Rounding error can cause the algorithm to take more steps, particularly if the matrix C is poorly conditioned. However, in well-conditioned problems, the method can find an acceptable solution in many fewer than n

steps, especially if C has repeated singular values. For example, if C is a rank k modification of the identity matrix, then the algorithm should converge in k iterations (in exact arithmetic). The principal advantage of conjugate algorithms is that they involve only matrix-vector multiplications and for sparse, structured matrices these multiplications can be easily implemented and made efficient.

6.2 Conjugate gradients applied to least squares

The LSQR algorithm of Paige and Saunders [30] implements an iterative approach to solving linear least squares problems that can be embedded in a Gauss-Newton algorithm. The algorithm applies a conjugate gradient algorithm to solve the normal equations (4) but avoiding any explicit formation of the product $J^{\mathrm{T}}J$. In exact arithmetic, the method will find the solution to an $m \times n$ least squares system in at most n iterations. The software implementation requires the user to supply a component to calculate matrix-vector products with J and J^{T} and the user is free to implement these calculations in a way appropriate for the type of matrix.

6.3 Block sparse matrices

For a number of applications, the Jacobian matrix is sparse but with the non-zero elements occurring in small submatrices, often with the same submatrix appearing in a number of locations within the matrix. We use the term *block sparse* matrix for this type of structure.

Let $m \times n$ matrix J be composed of n_B submatrices J_k of dimension $m_k \times n_k$. We assume that J_k is stored (columnwise or rowwise) as a column vector $\mathbf{d}_k$. The information in J can be encoded in a column vector $\mathbf{d}_J$ and an indexing set I_J such that $I_J(1:5,k) = (i_k, j_k, m_k, n_k, l_k)$ where (i_k, j_k) specifies the location of $J_k(1,1)$ in J and l_k indicates that $\mathbf{d}_k = \mathbf{d}_J(l_k : l_k + m_k n_k - 1)$. The number of elements in $\mathbf{d}_J$ is denoted by n_J. Blocks of such matrices can be easily represented by concatenating the $\mathbf{d}$-vectors and the index sets and performing some trivial index modifications. The block sparse representation $\langle \mathbf{d}_J, I_J \rangle$ can be used to store any matrix, full or sparse.

The following algorithm shows how to calculate $\mathbf{y} = J\mathbf{x}$, given $J = \langle \mathbf{d}_J, I_J \rangle$.

Starting with $\mathbf{y} = \mathbf{0}$, for $k = 1, \ldots, n_B$,

I Extract $(i_k, j_k, m_k, n_k, l_k)$ from I_J,

II Extract J_k from $\mathbf{d}_J(l_k : l_k + m_k * n_k - 1)$.

III Set

$$\mathbf{y}(i_k : i_k + m_k - 1) := \mathbf{y}(i_k : i_k + m_k - 1) + J_k \mathbf{x}(j_k : j_k + n_k - 1).$$

Similarly, the following algorithm shows how to calculate $\mathbf{x} = J^{\mathrm{T}}\mathbf{y}$. Starting with $\mathbf{x} = \mathbf{0}$, for $k = 1, \ldots, n_B$,

I Extract $(i_k, j_k, m_k, n_k, l_k)$ from I_J,

II Extract J_k from $\mathbf{d}_J(l_k : l_k + m_k * n_k - 1)$.

III Set

$$\mathbf{x}(j_k : j_k + n_k - 1) := \mathbf{x}(j_k : i_k + n_k - 1) + J_k^{\mathrm{T}} \mathbf{y}(i_k : i_k + m_k - 1).$$

We note that the calculation of the gradient $\mathbf{g} = 2J^{\mathrm{T}}\mathbf{f}$ can also be performed efficiently for block-sparse matrices.

6.4 Solver for block-sparse systems

The harness approach is ideally suited for solving block-sparse systems. The model specific information comprising the index matrix I_J and derivative information $\mathbf{d}_J$ can be supplied by a user-supplied component with the following signature:

6.4.1 FG_BS_MODEL

```
  subroutine fg_bs_model(imode,m,n,nj,nb,a,f,dj,ij)
! ------------------------------------------------
! Function and gradient for block-sparse systems.
! ------------------------------------------------
  integer, intent(inout) :: imode
  integer, intent(in)    :: m,n,nj,nb
  real(8), intent(in)    :: a(n)
  real(8), intent(out)   :: f(m),dj(nj)
  integer, intent(out)   :: ij(nb,5)
! ------------------------------------------------------------------------------
! Parameters
! ----------
!
! IO imode     INTEGER
!              If imode = 1, fg_bs_model is required to calculate
!                            function values f.
!                       = 2, fg_bs_model is required to calculate
```

```
!                           indices ij and Jacobian blocks dj.
!
! IN m          INTEGER
!               Number of functions (observation equations) fi.
!
! IN n          INTEGER
!               Number of parameters.
!
! IN nj         INTEGER
!               Number of elements required to store the Jacobian matrix.
!
! IN nb         INTEGER
!               Number of blocks in the Jacobian matrix.
!
! IN a          DOUBLE PRECISION 1-dimensional array
!               n x 1
!               Estimates of the optimisation parameters.
!
! OU f          DOUBLE PRECISION 1-dimensional array
!               m x 1
!               Function values.
!
! OU dj         DOUBLE PRECISION 1-dimensional array
!               nj x 1
!               If imode = 2 on entry, then on exit
!               dj should contain the nonzero elements
!               of the Jacobian matrix.
!
! OU ij         DOUBLE PRECISION 2-dimensional array
!               nb x 5
!               Index set to construct J from dj.
!               If imode = 2 on entry, then on exit
!               ij(k,1:5) = (ik,jk,mk,nk,lk).
```

With the information $\langle \mathbf{d}_J, I_J \rangle$ the harness component determines the Gauss-Newton step using the LSQR algorithm with a component that calculates matrix-vector products for block sparse matrices.

6.5 Calculation of statistical uncertainties for block-sparse systems

We have seen in section 2.2 that elements of the uncertainty matrix and derived uncertainties can be expressed in terms of $(J^{\mathrm{T}}J)^{-1}$. Even if J is block-sparse, $H = J^{\mathrm{T}}J$ is likely to be full, so the formation of H and the solution of equations involving H will be expensive. However H is symmetric and matrix-vector products $\mathbf{z} = H\mathbf{x}$ involving H can be performed using the two-step process

$$\mathbf{z} = J^{\mathrm{T}}\mathbf{y}, \quad \mathbf{y} = J\mathbf{x},$$

in which the block sparsity of J is exploited. This means that conjugate gradient methods for symmetric systems can be used effectively to determine uncertainty estimates (section 6.1). For example if $r(\mathbf{a})$ and $s(\mathbf{a})$ are functions of the fitted parameters with $\mathbf{r} = \nabla_{\mathbf{a}} r$, $\mathbf{s} = \nabla_{\mathbf{a}} s$ then the covariance of r with s is estimated by

$$\mathrm{cov}(r, s) = \sigma^2 \mathbf{r}^{\mathrm{T}} \tilde{\mathbf{s}}, \qquad H\tilde{\mathbf{s}} = \mathbf{s}.$$

We have implemented this approach in Fortran 90 using components from the Sparse Linear Algebra Package [25, 26]. Sparse matrix techniques are also available in Matlab.

6.6 Example application: CMM error mapping

As an example, we consider the accurate calibration of two-dimensional artefacts by a two dimensional co-ordinate measuring machine (CMM). The artefacts define the location of targets nominally aligned in a grid pattern. Let $\mathbf{y}_j$, $j = 1, \ldots, n_Y$, be the locations of the targets in a fixed frame of reference, and let

$$\mathbf{y}_{j,k} = T(\mathbf{y}_j, \mathbf{t}_k)$$

be the location of the jth target in the kth measuring position. Here, the roto-translation T is specified by three parameters $\mathbf{t}$ defining the translation vector and angle of rotation.

We suppose the systematic error of the two dimensional CMM can be expressed as

$$\mathbf{x}^* = \mathbf{x}^*(\mathbf{x}, \mathbf{b}) = \mathbf{x} + \mathbf{e}(\mathbf{x}, \mathbf{b}),$$

where $\mathbf{x}^*$ are the true point co-ordinates, $\mathbf{x}$ are the indicated point co-ordinates output by the machine and $\mathbf{e}(\mathbf{x}, \mathbf{b})$ is the error correction term depending on $\mathbf{x}$ and error parameters $\mathbf{b}$. For instance, suppose the model describes scale and orthogonality errors so that

$$x^* = x(1 + b_1) + y(1 + b_2) \sin b_3, \quad y^* = y(1 + b_2) \cos b_3.$$

If $\mathbf{x}_i$ is the measurement of the jth target with the artefact in the kth position then the associated observation equation is

$$\mathbf{x}_i + \mathbf{e}(\mathbf{x}_i, \mathbf{b}) = \mathbf{y}_{j,k} + \boldsymbol{\epsilon}_i, \tag{15}$$

where ϵ_i accounts for a random measurement effect. Given a set of such measurements $\{\mathbf{x}_i\}_1^{m_X}$ and associated index functions $(j(i), k(i))$ specifying the target and artefact position, estimates of the model parameters can be determined by solving a nonlinear least squares problem

$$\min_{\{\mathbf{y}_j\}, \{\mathbf{t}_k\}, \mathbf{b}} \quad \sum_{i=1}^{m_X} \mathbf{f}_i^{\mathrm{T}} \mathbf{f}_i,$$

where $\mathbf{f}_i(\mathbf{y}_{j(i)}, \mathbf{t}_{k(i)}, \mathbf{b}) = \mathbf{x}_i + \mathbf{e}(\mathbf{x}_i, \mathbf{b}) - \mathbf{y}_{j,k}$.

The model involves three sets of parameters: the target locations $\{\mathbf{y}_j\}$, transformation parameters $\{\mathbf{t}_k\}$ and the error parameters $\mathbf{b}$. Each observation equation depends on only one target and one transformation, so that the Jacobian matrix J of partial derivatives can be ordered to have a block-angular structure (section 4).

$$J = \begin{bmatrix} J_1 & & & J_{0,1} \\ & J_2 & & J_{0,2} \\ & & \ddots & \vdots \\ & & J_{m_X} & J_{0,m_X} \end{bmatrix},$$

where J_j corresponds to the parameters $\mathbf{y}_j$ and the border blocks $\{J_{0,j}\}$ correspond to the border parameters $\mathbf{a} = \{\{\mathbf{t}_k\}, \mathbf{b}\}$.

While scale and orthogonality errors are often major contributors to the systematic error behaviour of a CMM, there is no guarantee nor does experience show that they explain the full extent of the behaviour. For this reason, more comprehensive models have been developed [18, 31]. However, they all depend on the approximation of actual behaviour by empirical functions such as polynomials and the adequacy of the approximation is often difficult and expensive to evaluate. However, if we always rotate and translate the artefact according to the symmetries of the reference artefact so that the targets are always located (nominally) at a subset of a fixed grid of points in the CMM's working volume, then measurements are made at a finite number of machine locations. To the lth location we associate a machine error $\mathbf{e}_l$. If the ith measurement is made at the lth location then the observation equation corresponding to (15) is

$$\mathbf{x}_i + \mathbf{e}_l = \mathbf{y}_{j,k} + \boldsymbol{\epsilon}_i.$$

The advantage of this error model is that it entails no significant approximation: the systematic errors are modelled exactly. An apparent disadvantage is that there are likely to be as many error parameters as target parameters giving rise to a sparsity structure in the Jacobian matrix for which direct, structure-exploiting methods provide relatively minor efficiency gains.

Figure 4 shows on the left the sparsity structure of the Jacobian matrix J associated with the measurement of a 5×5 hole plate in eight positions, the first four corresponding to rotations by 0, 90, 180 and 270 degrees, the second four incorporating a translation as well as a rotation. In each position the location of the targets $\mathbf{y}_j$ are measured in order. The non-zero elements of the matrix are represented by a dot. The first (second) 50 columns correspond to the derivatives with respect to the machine error parameters $\mathbf{e}_l$ (target parameters $\mathbf{y}_j$) and the last 24 correspond to the eight

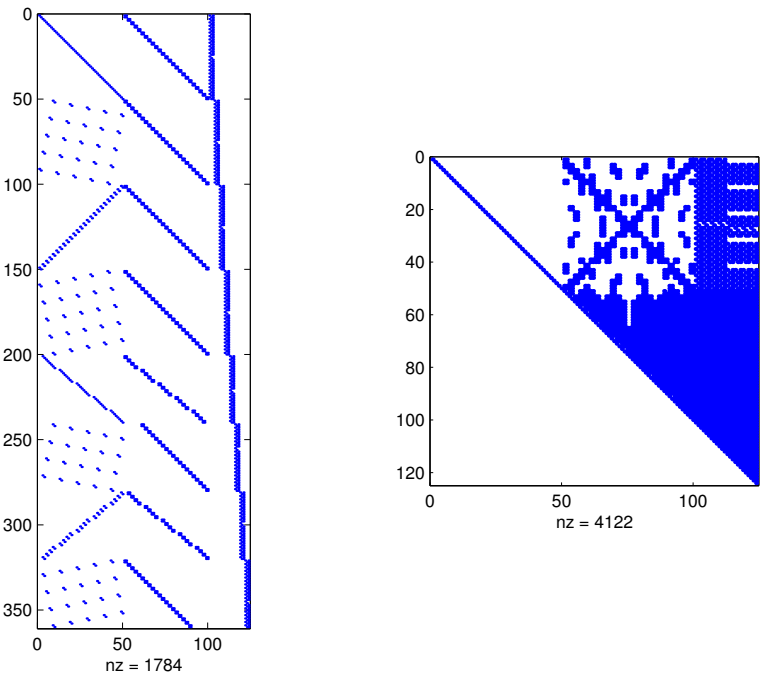

Figure 4: Sparsity structure of the Jacobian matrix and its upper triangular factor associated with the measurement of a 5×5 hole plate in eight positions.

---

sets of transformation parameters $\mathbf{t}_k$. The non-zero derivatives with respect to the machine error and target parameters occur as 2×2 identity matrix and rotation matrices, one for each position of the artefact. This information can be stored very compactly using the block-sparse representation. On the right of Figure 4 the sparsity structure of the triangular factor of J is illustrated and shows the substantial fill-in that occurs. The matrix J can be regarded as block angular, but the border contains more than half the total number of columns in the matrix so that an RBA approach will be at most four times quicker than regarding the matrix as full.

We have used the block-sparse solver in the calibration of a 13×13 grid of targets on a glass plate by a CMM with an optical probing system. The problem involved approximately 15,000 observation equations in over 800 optimisation parameters and was solved in a few seconds using a standard laboratory PC. The advantage of the error separation model is illustrated in Figure 5 which shows the residual errors associated with the first 1000 observations for models a) with no error separation (dots), and b) with error separation. The fit for the error separation model is much superior. The practical metrological consequence of adopting the enhanced model is that uncertainties associated with the target locations can be reduced by a

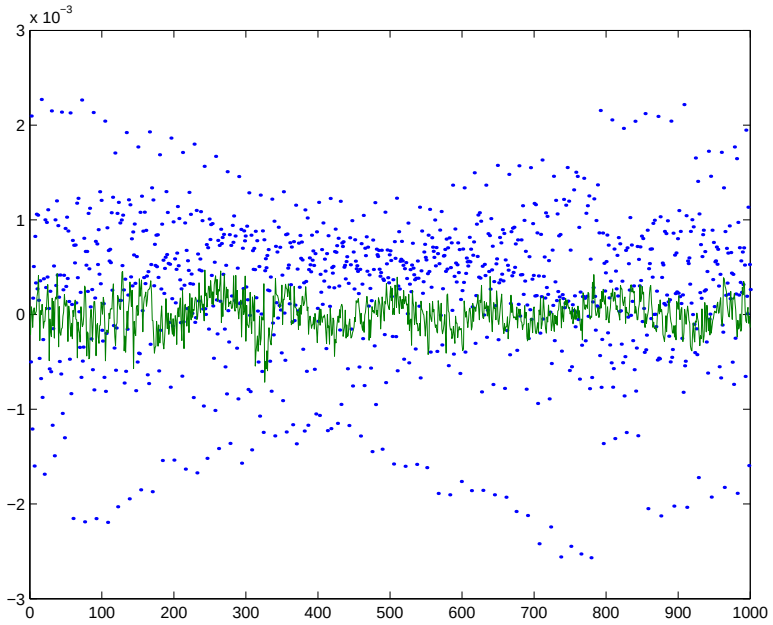

Figure 5: Residual errors associated with the first 1000 observations for models a) with no error separation (dots), and b) with error separation.

---

factor of five. Importantly, because the model is a realistic approximation of the measuring system, we can have confidence in the uncertainty estimates derived from the model.

6.7 Example application: simple generalised distance regression with curves

In section 4.5, we showed how an efficient algorithm for simple generalised distance regression could be implemented by exploiting the block angular structure of the problem. In this section, we show how the problem can also be solved effectively using the LSQR algorithm. The Jacobian matrix J associated with (12) has form

$$J = \begin{bmatrix} A_x & \mathbf{0} \\ B_x & B_{\mathbf{a}} \end{bmatrix}, \tag{16}$$

where A_x and B_x are diagonal matrices and $B_{\mathbf{a}}$ is an $m \times n$ matrix which generally will be full. (However, if we are performing generalised regression with splines, for example, $B_{\mathbf{a}}$ will be banded.) Efficient matrix-vector computations involving J are easy to implement and we can use the LSQR

algorithm to determine the Gauss-Newton step associated with the nonlinear regression problem. Numerical examples of this approach are given in section 8.

7 Nonlinear conjugate gradients and large scale optimisation

The conjugate gradient approach [23] is one of the main tools in general purpose large scale optimisation, particularly because it requires only a few vectors to be stored. Suppose we wish to find the minimum of $F(\mathbf{a})$, given an initial estimate $\mathbf{a}_0$. For nonlinear problems, the algorithm takes the form

I Set $k = 0$, $\mathbf{g}_0 = \nabla_{\mathbf{a}} F(\mathbf{a}_0)$.

II While $\|\mathbf{g}_k\| > \tau$,

i Set $k = k + 1$.

ii Determine a search direction. If $k = 1$ set $\mathbf{p}_1 = -\mathbf{g}_0$. If k is a multiple of n, set $\mathbf{p}_k = -\mathbf{g}_{k-1}$. Otherwise, set

$$\beta_k = \|\mathbf{g}_{k-1}\|^2 / \|\mathbf{g}_{k-2}\|^2, \quad \mathbf{p}_k = -\mathbf{g}_{k-1} + \beta_k \mathbf{p}_{k-1}.$$

iii Determine the step length. Find α_k to minimise $F(\mathbf{a}_{k-1} + \alpha_k \mathbf{p}_k)$.

iv Update

$$\mathbf{a}_k = \mathbf{a}_{k-1} + \alpha_k \mathbf{p}_k, \quad \mathbf{g}_k = \nabla_{\mathbf{a}} F(\mathbf{a}_k).$$

III Set $\mathbf{a} = \mathbf{a}_k$ and finish.

If we apply the algorithm above to the problem of minimising

$$\phi(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}} C \mathbf{a} - \mathbf{a}^{\mathrm{T}} \mathbf{b},$$

where C is symmetric, positive definite, we follow the same steps as the linear conjugate gradient algorithm to solve $C\mathbf{a} = \mathbf{b}$ described in section 6.1. In the linear case, the step length α_k can be determined explicitly and the residual vectors $\mathbf{r}_k$ are simply the steepest descent directions $-\mathbf{g}_k = -\nabla_{\mathbf{a}} \phi(\mathbf{a}_k)$. For the nonlinear case, the algorithm proceeds in cycles of n iterations, each cycle representing a new attempt to determine a set of n vectors conjugate with the Hessian matrix at the solution. For nonlinear least squares problems, the algorithm requires the calculation of $\nabla F = 2J^{\mathrm{T}}\mathbf{f}$. If J has structure then this calculation can be made efficiently.

We can compare the nonlinear conjugate gradients algorithm with the Gauss-Newton algorithm using LSQR. In the latter approach, an iterative scheme is used provide a reasonably accurate solution to the Jacobian system. During these inner iterations, the Jacobian matrix is held fixed. In the nonlinear conjugate gradients algorithm, the Jacobian matrix is updated at each inner

iteration. Using a truncating scheme with the LSQR algorithm in which we require a less accurate solution of the Jacobian system in the earlier iterations, we can define an algorithm that is someway in between.

There has been much research in developing efficient, large-scale optimisation algorithms, see e.g., [10, 28, 32]. One of the main approaches is to use a limited memory quasi-Newton algorithm [23, section 4.8]. In a quasi-Newton algorithm, the update step (5) is determined from an approximation to the Hessian matrix H of second partial derivatives of the objective function $F(\mathbf{a})$ or its inverse. Starting from the identity matrix, this approximation is built up from successive estimates $\mathbf{g}_k$ of the function gradients. If F is a quadratic function of n parameters, then after n steps the approximation to the Hessian is exact (in exact arithmetic). For large n, memory and computation constraints may prohibit any attempt to approximate H. Instead, the Hessian matrix is approximated by limited number of quasi-Newton updates and can be stored by a correspondingly limited number of n-vectors. The software package L-BFGS-B [32] employs a quasi-Newton approach and also uses reverse communication so that instead of supplying a function and gradient evaluation component, the user repeatedly calls the solver, supplying as input at the qth iteration the function and gradient information requested on exit from the $(q-1)$th call. This mode of operation has the advantage that the user can organise the computation and data flow appropriate for the application in hand. The optimisation toolbox in Matlab has a wide range of algorithms for large scale optimisation, including algorithms for least squares using a conjugate gradient approach similar to that described in section 6.

In our adaption of the Gauss-Newton algorithm for large scale problems, the Hessian matrix is approximated by $J^{\mathrm{T}}J$ and, for reasonably accurate data and well-conditioned problems, this approximation is an accurate one and leads to fast convergence to a solution.

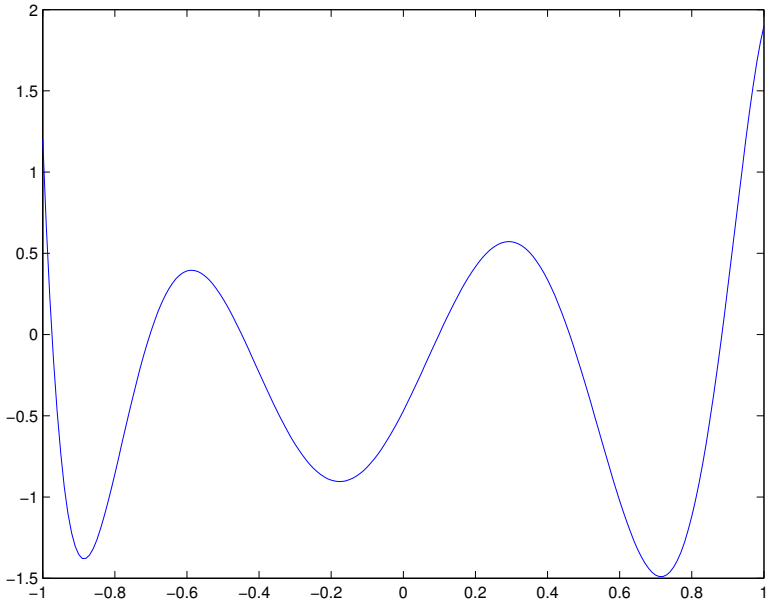

Figure 6: Polynomial of degree 9 used to generate test data.

8 Numerical examples: simple generalised distance regression

In this section we illustrate the behaviour of three algorithms for simple generalised distance regression with polynomials. The results were computed on a Dell Optiplex GX240 PC with a Intel Pentium 4 processor operating at 1.70 GHz. The first algorithm, Algorithm RBA, uses the regular block angular approach discussed in section 4.5. The second, Algorithm LSQR, uses the LSQR method to solve for the Gauss-Newton step, as discussed in section 6.7. The third, Algorithm CG, uses a nonlinear conjugate gradient approach and uses the NAG library optimisation component E04DGF [29].

We have applied these three algorithms to fitting polynomials of degree 9 to data sets involving 101, 1,001 and 10,001 data points. For the first set of results, data points were generated on the curve illustrated in Figure 6. The weights associated with the regression problem were set to 1, i.e., $\alpha_i = \beta_i = 1$.

A summary of the convergence behaviour of the algorithms on the first data set is presented in tables 1–3. For algorithms RBA and LSQR the tables list a) the Gauss-Newton iteration count N, b) the norm $\|\mathbf{f}\|$ of the residual vector, c) the change in the sum of squares $\mathbf{f}^{\mathrm{T}}\mathbf{f}$, d) the norm $\|\mathbf{p}\|$ of the Gauss-Newton step, and e) the norm $\|\mathbf{g}\|$ of the gradient. For

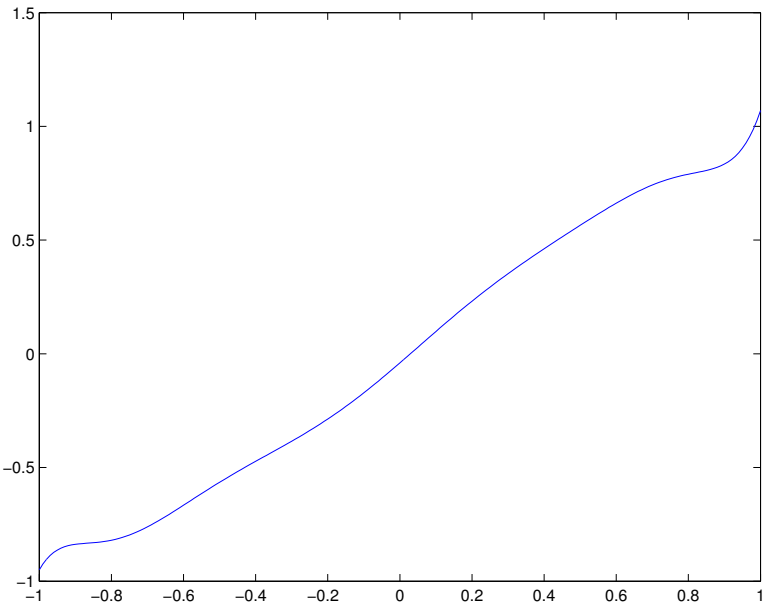

Figure 7: Polynomial of degree 9 used to generate test data approximately on a straightline.

---

algorithm LSQR the tables also give f) the number n of minor iterations required to solve for the Gauss-Newton step. For algorithm CG, the tables provide the total number of iterations N, $\|\mathbf{f}\|$ and $\|\mathbf{g}\|$. They also give the approximate difference between the CG estimate of the solution parameters and those provided by the other two algorithms (which agree with each other to the convergence tolerance of $1.0e-9$). The time in milliseconds to reach convergence is also recorded for all three algorithms. The results show:

The convergence behaviour for algorithms RBA and LSQR is very similar. This is to be expected as they are both implementations of the Gauss-Newton algorithm.

The time taken increases approximately linearly with the number of data points. A full matrix approach would increase cubically.

The CG algorithm is fastest followed by RBA and then LSQR.

The solution provided by algorithm CG is not as accurate as the other two. Typically, the CG algorithm provides estimates accurate to $1.0e-5$ while algorithms RBA and LSQR provide estimates accurate to $1.0e-9$, the limit specified by the convergence tolerance.

The number n of inner iterations required by the LSQR algorithm is

```
Algorithm RBA:  125 ms
N        |f|       del SS        |p|         |g|
1     5.9883E-03  3.0102E-05  3.5357E-03  8.5856E-02
2     2.3996E-03  3.5619E-10  3.1083E-05  2.5084E-04
3     2.3995E-03  9.5959E-14  3.9681E-07  7.0354E-07
4     2.3995E-03  6.8311E-17  1.1067E-08  1.7461E-08

Algorithm LSQR: 344 ms
N        |f|       del SS        |p|         |g|       n
1     5.9883E-03  3.0102E-05  3.5357E-03  8.5856E-02  157
2     2.3996E-03  3.5650E-10  3.1084E-05  2.5083E-04  163
3     2.3995E-03  9.6215E-14  3.9743E-07  7.0457E-07  152
4     2.3995E-03  6.7926E-17  1.0932E-08  1.7481E-08   94


Algorithm CG:    47 ms
N        |f|                                |g|
263     2.3999E-03                         1.1015E-05

Difference in CG solution coefficients:
2.0E-04
```

Table 1: Convergence of three algorithms performing generalised distance regression with a degree 9 polynomial on 101 data points (Figure 6).

only mildly dependent on the number of data points. For the third data set with 10,001 data points, there are over 10,000 optimisation parameters but the the Gauss-Newton step is determined in under 300 iterations.

We repeat the numerical experiments, this time of data generated from the curve illustrated in figure 7. The convergence behaviour of the algorithms on this data set is presented in tables 4–6. The results show:

All three algorithms require fewer iterations to converge compared to the first data set. The main reason for this is that since curvature of the polynomial is smaller, the determination of the parameters δ_i is nearer to being a linear problem, and so the whole optimisation problem is more linear.

The algorithm LSQR requires fewer inner iterations to reach converences, 40 or less. The main reason for this is again linked to the curvature. For this case, the diagonal elements submatrix B_x of the Jacobian matrix in (16) are approximately constant (as they depend only on the slope of the curve). The Jacobian matrix is approximated by a rank 10 modification of the identity matrix and the conjugate gradient method can be expected to converge in a relatively small

```
Algorithm RBA:   500 ms
N        |f|       del SS        |p|         |g|
1     1.8459E-02  2.9239E-04  5.0569E-03  2.5159E-01
2     6.9525E-03  7.7047E-09  7.5290E-05  1.2804E-03
3     6.9520E-03  4.1727E-11  5.8510E-06  1.4528E-05
4     6.9520E-03  4.8166E-13  6.2841E-07  1.4912E-06

Algorithm LSQR: 1219 ms
N        |f|       del SS        |p|         |g|       n
1     1.8459E-02  2.9239E-04  5.0569E-03  2.5159E-01  211
2     6.9525E-03  7.7056E-09  7.5294E-05  1.2804E-03  223
3     6.9520E-03  4.1737E-11  5.8517E-06  1.4530E-05  211
4     6.9520E-03  4.8178E-13  6.2848E-07  1.4913E-06  191

Algorithm CG:    172 ms
N        |f|                                 |g|
56    6.9520E-03                           2.3882E-05

Difference in CG solution coefficients:
4.0E-05
```

Table 2: Convergence of three algorithms performing generalised distance regression with a degree 9 polynomial on 1,001 data points (Figure 6).

```
Algorithm RBA:  11406 ms
N        |f|       del SS        |p|         |g|
1     5.7937E-02  2.8451E-03  1.5801E-02  8.2134E-01
2     2.2618E-02  6.2298E-08  2.1980E-04  1.1550E-02
3     2.2616E-02  3.2013E-10  1.6767E-05  4.2810E-05
4     2.2616E-02  5.3667E-12  2.2460E-06  4.8764E-06

Algorithm LSQR: 15813 ms
N        |f|       del SS        |p|         |g|       n
1     5.7937E-02  2.8451E-03  1.5801E-02  8.2134E-01  252
2     2.2618E-02  6.2302E-08  2.1982E-04  1.1550E-02  269
3     2.2616E-02  3.2015E-10  1.6768E-05  4.2812E-05  256
4     2.2616E-02  5.3669E-12  2.2461E-06  4.8767E-06  182

Algorithm CG:   4016 ms
N        |f|                                 |g|
101   2.2616E-02                           5.6607E-05

Difference in CG solution coefficients:
3.0E-05
```

Table 3: Convergence of three algorithms performing generalised distance regression with a degree 9 polynomial on 10,001 data points (Figure 6).

```
Algorithm RBA:   125  ms
N        |f|       del SS        |p|         |g|
1     5.9883E-03  1.8503E-05  3.0077E-03  2.9042E-02
2     4.1661E-03  8.1048E-11  8.2790E-06  4.2765E-05
3     4.1661E-03  3.0638E-15  6.2310E-08  2.1132E-07

Algorithm LSQR:   110 ms
N        |f|       del SS        |p|         |g|      n
1     5.9883E-03  1.8503E-05  3.0077E-03  2.9042E-02  28
2     4.1661E-03  8.1103E-11  8.2821E-06  4.2764E-05  28
3     4.1661E-03  3.0654E-15  6.2328E-08  2.1148E-07  21

Algorithm CG:      26 ms
N        |f|                               |g|
39    4.1661E-03                          7.5561E-06

Difference in CG solution coefficients:
1.0E-05
```

Table 4: Convergence of three algorithms performing generalised distance regression with a degree 9 polynomial on 101 data points (Figure 7).

number of iterations.

The parameter estimates provided by algorithm CG are more accurate than those for the first set of data, but are not as accurate as those provided by algorithms RBA and LSQR.

As a final example, we apply all three algorithms to 1,001 generated from the same curve as the first set of results (Figure 6) but with weights $\alpha_i \doteq 2\beta_i$. The problem is more nonlinear that the case $\alpha_i \doteq \beta_i$. The convergence behaviour summarised in table 7 shows that:

All the algorithms require more (major) iterations to converge than in the corresponding case summarised in table 2.

The LSQR algorithms requires more inner iterations to determine the Gauss-Newton step, although still requiring only about 500 iterations for a problem with over 1000 parameters.

```
Algorithm RBA:   438 ms
N        |f|       del SS        |p|         |g|
1     1.8459E-02  1.4455E-04  8.4792E-03  7.3300E-02
2     1.4006E-02  6.2969E-10  1.8867E-05  1.7545E-04
3     1.4006E-02  1.3030E-13  3.2268E-07  9.3547E-07

Algorithm LSQR:  219 ms
N        |f|       del SS        |p|         |g|      n
1     1.8459E-02  1.4455E-04  8.4792E-03  7.3300E-02  33
2     1.4006E-02  6.3001E-10  1.8878E-05  1.7545E-04  37
3     1.4006E-02  1.3033E-13  3.2271E-07  9.3432E-07  25

Algorithm CG:    78 ms
N        |f|                               |g|
44    1.4006E-02                          3.0625E-05

Difference in CG solution coefficients:
4.0E-06
```

Table 5: Convergence of three algorithms performing generalised distance regression with a degree 9 polynomial on 1,001 data points (Figure 7).

```
Algorithm RBA:   9203 ms
N        |f|       del SS        |p|         |g|
1     5.7937E-02  1.4291E-03  2.6766E-02  3.3678E-01
2     4.3904E-02  4.2673E-09  4.6462E-05  1.4878E-03
3     4.3904E-02  5.1230E-13  6.2052E-07  2.5200E-06

Algorithm LSQR:  1875 ms
N        |f|       del SS        |p|         |g|      n
1    2  5.7937E-02  1.4291E-03  2.6766E-02  3.3678E-01  40
2    3  4.3904E-02  4.2662E-09  4.6453E-05  1.4878E-03  40
3    4  4.3904E-02  5.1218E-13  6.2044E-07  2.4704E-06  25

Algorithm CGL    1359 ms
N        |f|                               |g|
46     4.3904E-02                         8.8603E-05

Difference in CG solution coefficients:
5.0E-07
```

Table 6: Convergence of three algorithms performing generalised distance regression with a degree 9 polynomial on 10,001 data points (Figure 7).

```
Algorithm RBA:    719 ms
N        |f|       del SS        |p|         |g|
1     2.7817E-02  7.1232E-04  7.4886E-03  5.3859E-01
2     7.8398E-03  5.8681E-08  2.8412E-04  5.6203E-03
3     7.8360E-03  7.8358E-10  3.8808E-05  4.1470E-05
4     7.8360E-03  2.4415E-11  6.9423E-06  6.2055E-06
5     7.8360E-03  8.7812E-13  1.3340E-06  1.1758E-06
6     7.8360E-03  3.2804E-14  2.5628E-07  2.2575E-07

Algorithm LSQR:  2515 ms
N        |f|       del SS        |p|         |g|       n
1     2.7817E-02  7.1232E-04  7.4885E-03  5.3859E-01  463
2     7.8398E-03  5.8640E-08  2.8400E-04  5.6205E-03  495
3     7.8360E-03  7.8368E-10  3.8814E-05  4.1460E-05  510
4     7.8360E-03  2.4432E-11  6.9449E-06  6.2078E-06  459
5     7.8360E-03  8.7896E-13  1.3347E-06  1.1765E-06  421
6     7.8360E-03  3.2838E-14  2.5642E-07  2.2589E-07  382

Algorithm CG:      531 ms
N        |f|                                 |g|
191   7.8370E-03                           8.4792E-05

Difference in CG solution coefficients:
6.0E-04
```

Table 7: Convergence of three algorithms performing generalised distance regression with a degree 9 polynomial on 1,001 data points (Figure 7) for the case $\alpha_i \doteq 2\beta_i$.

9 Concluding remarks

In this report, we have described algorithms and software for solving large scale calibration problems using nonlinear least squares approaches. We have considered two approaches, one in which the sparsity structure can be exploited directly in determining updates to the parameter estimates, the other in which iterative methods are used to provide these estimates. Both approaches require modest algorithmic resources and can be encoded compactly in software. We can compare both approaches on generalised distance regression using polynomials. The results show that:

Both approaches provide accurate estimates of parameters.

Both are able to solve regression problems involving tens of thousands of parameters in the order of ten seconds.

Nonlinear conjugate gradient methods also provide estimates efficiently but with less accuracy.

We have also shown how the use of solvers that employ harness components can be very effective in providing an interface between the user's application and the optimisation software. It is our belief that the use of harnesses greatly improves the developer's ability to provide application-specific solvers that can deal effectively with the class of calibration problem at hand. We have implemented a simple Gauss-Newton solver using the harness approach and used it to solve a range of calibration problems employing both direct and iteration approaches. The same solver can be used to solve nonlinear Gauss-Markov problems [19, 22] or linearly constrained nonlinear least squares problems, for example, as well as the problems considered here. It is anticipated that implementations of the solver and application solvers derived from it will be made available through MetroS [1].

Acknowledgements. The work described here was supported by the National Measurement System Directorate of the UK Department of Trade and Industry as part of its NMS Software Support for Metrology programme.

References

[1] R. M. Barker and G. Morgan. Design of the Metrology Software environment (METROS). Technical Report CISE 43/00, National Physical Laboratory, February 2000.

[2] M. C. Bartholomew-Biggs and A. B. Forbes. A robust search strategy for a non-linear least squares solver. *Journal of Optimisation Theory and its Applications*, 104(1):215–234, January 2000.

[3] P. T. Boggs, R. H. Byrd, and R. B. Schnabel. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM Journal of Scientific and Statistical Computing*, 8(6):1052–1078, 1987.

[4] P. T. Boggs, J. R. Donaldson, R. H. Byrd, and R. B. Schnabel. ODRPACK: software for weighted orthogonal distance regression. *ACM Trans. Math. Soft.*, 15(4):348–364, 1989.

[5] R. Boudjemaa, M. G. Cox, A. B. Forbes, and P. M. Harris. Automatic differentiation and its applications to metrology. Technical report, National Physical Laboratory, 2003. *To appear.*

[6] B. P. Butler, M. G. Cox, and A. B. Forbes. The reconstruction of workpiece surfaces from probe coordinate data. In R. B. Fisher, editor, *Design and Application of Curves and Surfaces*, pages 99–116. Oxford University Press, 1994. IMA Conference Series.

[7] B. P. Butler, A. B. Forbes, and P. D. Kenward. Position calibration software. Technical Report CISE 14/98, National Physical Laboratory, Teddington, February 1998.

[8] R. H. Byrd, R. B Schnabel, and G. A. Shultz. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Math. Prog.*, 40:247–263, 1988.

[9] T.F. Coleman and Y. Li. Trust region approach for nonlinear minimization subject to bounds. *SIAM J. Opt.*, 6:418–445, 1996.

[10] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization, release A.* Springer-Verlag, Berlin, 1992.

[11] M. G. Cox. The least squares solution of overdetermined linear equations having band or augmented band structure. *IMA J. Numer. Anal.*, 1:3 – 22, 1981.

[12] M. G. Cox. Practical spline approximation. In P. R. Turner, editor, *Lecture Notes in Mathematics 965: Topics in Numerical Analysis*, pages 79–112, Berlin, 1982. Springer-Verlag.

[13] M. G. Cox. Least-squares solution of linear equations with block-angular observation matrix. In M. G. Cox and S. J. Hammarling, editors, *Reliable numerical computing*, pages 227–240, Oxford, 1990. Clarendon Press.

[14] M. G. Cox. Linear algebra support modules for approximation and other software. In J. C. Mason and M. G. Cox, editors, *Scientific Software Systems*, pages 21–29, London, 1990. Chapman & Hall.

[15] M. G. Cox. Algorithms for spline curves and surfaces. In L. Piegl, editor, *Fundamental developments of computer-aided geometric modelling*, pages 51–76. Academic Press, London, 1993.

[16] M. G. Cox, M. P. Dainton, A. B. Forbes, P. M. Fossati, P. M. Harris, and I. M. Smith. Modelling multi-lateration co-ordinate measuring systems. Technical Report CBTLM S12, National Physical Laboratory, February 2000.

[17] M. G. Cox, A. B. Forbes, and P. M. Harris. Software Support for Metrology Best Practice Guide 4: Modelling Discrete Data. Technical report, National Physical Laboratory, Teddington, 2000.

[18] M. G. Cox, A. B. Forbes, P. M. Harris, and G. N. Peggs. Experimental design in determining the parametric errors of CMMs. In V. Chiles and D. Jenkinson, editors, *Laser Metrology and Machine Performance IV*, pages 13–22, Southampton, 1999. WIT Press.

[19] M. G. Cox, A. B. Forbes, P. M. Harris, and I. M. Smith. Classification and solution of regression problems for calibration. Technical Report CMSC 24/03, National Physical Laboratory, May 2003.

[20] A. B. Forbes. Least squares best fit geometric elements. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation II*, pages 311–319, London, 1990. Chapman & Hall.

[21] A. B. Forbes. Position calibration software for the form assessment of complex workpieces. In A. K. Kochhar, editor, *Proceedings of the 30th International MATADOR Conference*, pages 663 –670, Manchester, 1993. UMIST.

[22] A. B. Forbes, P. M. Harris, and I. M. Smith. Generalised Gauss-Markov Regression. In J. Levesley, I. Anderson, and J. C. Mason, editors, *Algorithms for Approximation IV*, pages 270–277. University of Huddersfield, 2002.

[23] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, 1981.

[24] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, third edition, 1996.

[25] A. Greenbaum. *Iterative methods for solving linear systems*. SIAM, Philadelphia, 1997.

[26] A. Greenbaum and M. K. Seager. *Sparse Linear Algebra Package*. www.netlib.org.

[27] J. J. Moré. The Levenberg-Marquardt algorithm: implementation and theory. In G. A. Watson, editor, *Lecture Notes in Mathematics 630*, pages 105–116, Berlin, 1977. Springer-Verlag.

[28] J. J. Moré and S. J. Wright. *Optimization Software Guide*. SIAM, Philadelphia, 1993.

[29] The Numerical Algorithms Group Limited, Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR. *The NAG Fortran Library, Mark 19, Introductory Guide*, 1999. *http://www.nag.co.uk/*.

[30] C. C. Paige and M. A. Saunders. LSQR: and algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1), 1982.

[31] G. Zhang, R. Ouyang, B. Lu, R. Hocken, R. Veale, and A. Donmez. A displacement method for machine geometry calibration. *Annals of the CIRP*, 37:515–518, 1998.

[32] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *Trans. Math. Soft*, 23(4), 1997.