RAL-TR-96-015

NUMERICAL ANALYSIS GROUP PROGRESS REPORT

January 1994 – December 1995

Edited by Iain S. Duff

Computing and Information Systems Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

February 1996.

CONTENTS

Personnel in Numerical Analysis Group.

Staff

Iain Duff.
Group Leader. Sparse matrices and vector and parallel computers and computing.

Nick Gould.
Optimization and nonlinear equations particularly for large systems.

John Reid.
Sparse matrices and development of the Fortran programming language.

Jennifer Scott.
Sparse matrix software. Volterra integral equations. Seminar organization.

Visitors and Attached Staff

Andy Conn (EPSRC supported visitor from IBM Yorktown Heights)
Optimization.

David Cowey (CASE Student from RMCS Shrivenham)
Automatic differentiation.

Jean-Yves L'Excellent (Visitor from ENSEEIHT-IRIT supported by Alliance Programme)
Large-scale optimization. Element-by-element preconditioning.

Mike Hopper (Consultant)
Support for Harwell Subroutine Library. TSSD.

Richard Lehoucq (Visitor from Argonne National Lab)
Solution of sparse eigenvalue problems.

David Miron (Visitor from ANU Canberra)
Solution of sparse equations.

Philippe Toint (Visitor from Namur, Belgium)
Optimization

1 Introduction (I.S. Duff)

This report covers the period from January 1994 to December 1995 and describes work performed by the Numerical Analysis Group within the Computing and Information Systems Department at the Rutherford Appleton Laboratory.

The details of our activities are documented in the following pages. These words of introduction are intended merely to provide an introduction and additional information on activities that are not appropriate for the detailed reports.

The support and development of the Harwell Subroutine Library (HSL) continues to be one of our major activities. This reporting period has seen new releases of both HSL (Release 12) and the NAG-marketed Harwell Sparse Matrix Library (Mark 2). The turbulent times at AEA Technology have led to some confusion in their support for the marketing and sales of HSL with three changes of management by them over the last year. For the moment, at least some stability reigns and we are developing a good working relationship with Scott Roberts and Richard Lee who took over from John Harding on the eve of Release 12. As in the previous Release, we benefited greatly from the consultancy of Mike Hopper who, in modern parlance, performed the quality control in the latter stages of code development. Our contacts with Ian Jones and his CFDS group at Harwell continue on a very informal basis, and we have continued our collaboration with Andrew Cliffe on the solution of finite-element equations by frontal methods.

Most of our visitors have been fairly short-term although the interaction with them has been quite intense. We continue our interaction with Oxford in support of the Joint Computational Mathematics and Applications Seminar series and have hosted several talks at RAL through that programme. We were involved in the successful CCLRC bid to host an HPCI Centre at Daresbury, and we hosted a course on Numerical Calculations with Matrices at RAL in December 1995 with the support of this Centre. It was a very successful course in which all Group members participated and may well be the forerunner to subsequent similar courses in numerical computation.

John has continued to combine his interests in Fortran and sparse matrices giving several talks on these topics during the last two years. He has been very involved through ISO WG5 in influencing the development of Fortran 95 and has been their main architect of a proposal for the inclusion of exception handling in Fortran 2000. He attended several X3J3 meetings, primarily to promote and develop this proposal, and discusses this and other related Fortran activities in Section 6.2. John has given talks on automatic differentiation at Oxford; Fortran 90 and Fortran 95 at RAL, NAG Users' Association, RMCS Shrivenham, and JPL; and has given courses on Fortran 90 at RAL and Shrivenham, where he is a visiting Professor. He had a six-week visit to Australia, primarily hosted by ANU Canberra, during which time he worked and gave talks on Fortran 90, sparse matrices, constrained least squares, and automatic differentiation, the latter topic in which he supervises a CASE student, David

Cowey. John gave some MSc lectures on iterative methods at Reading University and has given invited talks at meetings in Hamburg, RAL, St Girons, and Toulouse, and a contributed talk in Linköping.

Nick's collaboration with Conn and Toint continues to expand the theory and practice of large-scale optimization. Much of their work is embodied in the LANCELOT package for which they were awarded the Beale-Orchard-Hays prize for excellence in computational mathematical programming in August 1994. He still has joint research activities with contacts made during his visit to CERFACS in 1993 and has had an Alliance grant from the British Council to support this activity. He was a co-supervisor of Jean-Yves L'Excellent, who completed his thesis at ENSEEIHT-IRIT in Toulouse in November 1995. Another student of Nick's, Marli Hernandez at the Unversity of Hertfordshire, completed her thesis successfully in 1995. Nick is a Visiting Fellow at RMCS Shrivenham. He was an invited speaker at conferences at CORE in Belgium, Manchester, RAL, St Girons, and Stockholm, has given seminars at RAL and Durham and contributed talks at Dundee and Minneapolis.

Jennifer has developed several international collaborative projects over the past year, primarily in the computation of sparse eigenvalues. Although she has continued her part-time working, now on a five-day per week basis, she remains so productive that it is easy to forget this fact. In addition to her work on eigensystems (in which she hosted a visit from Richard Lehoucq), she has continued with her work on frontal solvers and has developed a suite of iterative solvers and an approximate inverse preconditioning technique in collaboration with Nick. Jennifer continues to coordinate our joint seminar series with Oxford University. She gave an invited talk at Toulouse and posters and contributed talks at Utah, Dundee, and Manchester.

Iain still leads a project at the European Centre for Research and Advanced Training in Scientific Computation (CERFACS) at Toulouse in France and has welcomed all Group members to Toulouse during the last year (see Section 6.1). Iain is a chief editor of the IMA Journal of Numerical Analysis, editor of the IMANA Newsletter, chairman of the IMA Programme Committee, an adjudicator for the Fox Prize, IMA representative on the CCIAM International Committee that overseas the triennial international conferences on applied mathematics, and a Visiting Professor at Strathclyde. In high performance computing, he has given tutorials at Supercomputing '94 (Washington DC), Supercomputing'95 (San Diego), and HPCN Europe (Milan), is on the Scientific Council of the CRIHAN Centre in Rouen, and was on an international panel that reviewed proposals for Supercomputing in Sweden. Iain has been on the Programme Committee for several international meetings and was on the organizing committee for the Linear Algebra meeting in Manchester in July 1994. He has given seminars in Berkeley, Boulder, Copenhagen, NASA Ames, Oak Ridge, Tennessee, and Umeå, and has given invited presentations at Ascona (Switzerland), Bonn, Copenhagen, Grenoble, Hamburg, Linköping, Oxford, San Francisco, St Girons, St Malo, Skalsky Dvur (Czech Republic), Toulouse, and Yokohama.

We have tried to subdivide our activities to facilitate the reading of this report. This is to some extent an arbitrary subdivision since much of our work spans these subdivisions. Our main research areas and interests lie in sparse matrix research, nonlinear algebra and optimization, applied mathematics, and numerical linear algebra. Work pertaining to these areas is discussed in Sections 2 to 5, respectively. We group some miscellaneous topics in Section 6. Much of our research and development results in high quality advanced mathematical software for the Harwell Subroutine Library. The organization, maintenance, documentation, and distribution of this Library is in itself a major task and we report on work in these areas in Section 7. Lists of seminars (in the joint series with Oxford), technical reports, and publications are given in Sections 8, 9, and 10, respectively. Current information on the activities of the Group and on Group members can be found through page http://www.cis.rl.ac.uk/struct/ARCD/NUM.html of the World Wide Web.

2 Sparse Matrices

2.1 The direct solution of sparse unsymmetric linear sets of equations (I.S. Duff and J.K. Reid)

We have completed a new code, `MA48`, for the direct solution of a sparse unsymmetric set of linear equations

$$\mathbf{Ax} = \mathbf{b}, \tag{1}$$

where **A** is usually square and nonsingular. The main features are:

(i) a fast and robust solution to sparse unsymmetric sets of linear equations,

(ii) a user-friendly input format (entries in any order in a real array and corresponding row and column indices in two parallel integer arrays, with duplicates allowed and summed),

(iii) the code switches to full-matrix processing when the reduced matrix is sufficiently dense, using Basic Linear Algebra Subprograms (BLAS) at Levels 1, 2, and 3,

(iv) the pivot sequence is normally chosen automatically from anywhere in the matrix, but the choice may be limited to the diagonal or the pivot sequence may be specified,

(v) in the event of insufficient storage allocation by the user, the package continues with the computation to obtain a good estimate of the amount required,

(vi) the code computes and uses the block triangular form,

(vii) entries smaller than a threshold are dropped from the factorization,

(viii) singular or rectangular matrices are permitted,

(ix) another matrix of the same pattern may be factorized with or without additional row interchanges for stability,

(x) there is an option of specifying that some columns have not changed when factorizing another matrix,

(xi) another problem with the same matrix or its transpose may be solved,

(xii) iterative refinement of the solution is available to improve its accuracy or provide an error estimate.

A most exciting development is the work of Gilbert and Peierls [3] for economically generating the patterns of the columns of the factors when factorizing with a given column sequence but allowing for row interchanges. The overall complexity is $O(n)+O(f)$ where f is the number of floating-point operations. There are overheads, associated with the recomputation of the sparsity patterns of the columns and the row interchanges may cause extra fill-ins, so in `MA48` we provide two ‘factorizes’ which we call ***first*** and ***fast.*** The first factorize must be provided with a column sequence. Thus, the analyse phase need only provide a recommended pivot sequence; there is no need for this phase to provide the sparsity pattern of the factorized matrix. We have therefore designed the analyse phase to provide the permutations without the actual factors. This saves storage since working storage is then needed only for the active submatrix of the block on the diagonal of the block triangular form that is currently being processed. It may save time since the vectors that hold the active columns are shorter and data compressions are much less likely to be needed.

Since the Harwell Subroutine Library code MA28 is a benchmark standard in the solution of sparse unsymmetric equations, we have compared MA48 with MA28 on three computing environments, taking ratios of storage and execution times for the various phases in the solution of (1). The results are summarized in Table 2.1.1 which shows the median and quartiles for these ratios. MA28 always produces a factorization when it performs an analysis and its only form of factorization is without pivoting. The MA28 analyse time is therefore strictly comparable with the sum of the analyse and factorize times of MA48, and this comparison is shown in column “Analyse + Fact.”. However, analyse alone or factorize with pivoting may also be needed by the user, so we also use the MA28 analyse time to compare separately with the analyse (column “Analyse”) and first factorize (column “First Fact.”) times of MA48.

In view of these satisfactory results, we regard MA28 as having been rendered obsolescent by MA48 and have thus flagged it for removal in a later Release of HSL. The design of MA48 is described in [1] and in more detail in [2].

		Array size reqd	Analyse	First Fact.	Analyse + Fact.	Fast Fact.	Solve
CRAY	lower quartile	0.50	2.87	7.84	2.03	1.92	1.79
	median	0.69	4.47	12.10	3.26	2.61	2.20
	upper quartile	0.71	7.21	21.41	5.11	3.41	2.35
SUN	lower quartile	0.50	2.20	3.88	1.38	1.27	0.70
	median	0.68	3.17	5.09	1.89	1.79	0.83
	upper quartile	0.74	6.39	9.71	3.52	2.21	1.03
RS/6000	lower quartile	0.50	2.67	7.99	2.04	2.03	1.03
	median	0.68	3.58	11.69	2.64	2.61	1.16
	upper quartile	0.74	7.07	24.33	5.06	4.17	1.36

Table 2.1.1. MA28 results divided by those with MA48.

References

[1] Duff, I.S. and Reid, J.K. (1995). The design of MA48, a code for direct solution of sparse unsymmetric linear systems of equations. Report RAL-TR-95-039, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear in *ACM Trans. Math. Softw.*

[2] Duff, I.S. and Reid, J.K. (1993). MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations. Report RAL-93-072, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[3] Gilbert, J. R. and Peierls, T. (1988). Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Stat. Comput.* **9**, 862-874.

2.2 The design and use of algorithms for permuting large entries to the diagonal (I. S. Duff and J. Koster)

We have developed software for permuting a sparse matrix so that the diagonal of the permuted matrix has entries of large absolute value. We have modified and extended Harwell Subroutine Library routine `MC21`, which finds a maximum transversal of any unsymmetric sparse matrix. A maximum transversal is a set of entries of maximum cardinality where no two members of the set are in the same row or column. For a structurally nonsingular matrix, this cardinality will be equal to the matrix order. Our extension consists of taking into account the values of the matrix entries and finding, among all maximum transversals, one that maximizes the minimum entry on the diagonal. We call such a transversal a bottleneck transversal. We have explored several ways of doing this and have designed one that is in practice similar in execution time to only a few calls to `MC21` itself.

There are many ways in which the permutation of a matrix to put the bottleneck transversal on the diagonal can be useful and we are investigating several of these. For example, many current algorithms for the solution of sparse sets of linear equations (for example `MA37` and `MA41`) perform an analysis phase that assumes that diagonal entries will be suitable as pivots during a subsequent numerical factorization phase. Now, while there is no guarantee that having a bottleneck transversal on the diagonal will ensure the pivots selected by the analysis are numerically satisfactory, we intuitively feel that this should be better than a more arbitrary set of entries on the diagonal. We are currently experimenting with this preordering using the `MA41` code. Initial findings are encouraging.

We are also examining the use of bottleneck transversals in the context of a block iterative scheme like that developed in [1]. In such a scheme, a direct solution scheme is used on blocks of the matrix, and the overall solution is obtained by iterating over the blocks. By placing large entries on to the diagonal, we can improve the robustness of our direct solver, in a similar fashion to the discussion in the previous paragraph, while at the same time helping the iterative part of the solver by reducing the influence of the overlapping part of the blocks.

Finally, we are also examining the use of bottleneck transversals in techniques for obtaining preconditioning matrices for iterative solution. Clearly, it would appear that a diagonal preconditioner consisting of the bottleneck transversal might perform better than an arbitrary one and this effect might also influence more sophisticated preconditioners.

A fuller discussion of this work will shortly be available [2].

References

[1] Arioli, M., Duff, I. S., Noailles, J., and Ruiz, D. (1992). A block projection method for sparse equations. *SIAM J. Sci. Stat. Comput.* **13**, 47-70.

[2] Duff, I. S. and Koster, J. (1996). The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. To appear.

2.3 An approximate minimum degree ordering (P.R. Amestoy, I.S. Duff, and T.A. Davis)

The minimum degree reordering scheme for symmetric matrices is one of the oldest known reordering schemes for sparse matrices [3]. At each stage of the elimination, the next pivot is chosen as a diagonal entry in the reduced matrix with least number of other entries in its row. This very simple algorithm is a highly successful ordering scheme for reducing work and fill-in when performing Gaussian elimination. However, on larger problems, as commonly occur today, it can be quite slow. Most of the time is spent on the degree update, that is updating the row counts after a pivoting step. This cost is even more noticeable because of the recent increase in the speed of the numerical factorization algorithms.

The purpose of our work, which is described in more detail in [1], is to design an ordering scheme that closely approximates the minimum degree ordering, and so retains its good properties, while at the same time executes much more quickly than the best available implementation of minimum degree.

We use the same quotient graph model as most current implementations of minimum degree to represent the symbolic factorization but replace the costly degree update by an approximate update, which in many cases will be exact. The resulting algorithm is typically much faster than previous minimum degree ordering algorithms and produces results that are comparable in quality with the best orderings from other minimum degree algorithms.

We show a comparison in Table 2.3.1 of our new algorithm with `MMD`, the multiple minimum degree algorithm of Liu [2], and the minimum degree ordering from the Harwell Subroutine Library code `MA27`. This clearly illustrates the power of our new algorithm. It has been incorporated in the Harwell Subroutine Library at Release 12 as subroutine `MC47`.

Matrix	Order	Entries	Number of entries in **L** (in thousands)			Time (seconds on SUN SPARC 10)		
			`MC47`	`MMD`	`MA27`	`MC47`	`MMD`	`MA27`
RAEFSKY3	21200	733784	4709	4779	5041	1.05	2.79	1.23
BCSSTK31	35588	572914	5115	5231	6056	4.55	11.60	7.92
FINAN512	74752	261120	4778	8180	8159	15.03	895.23	40.31
BBMAT	38744	1274141	19673	19876	21139	27.80	200.86	134.58
ORANI678	2529	85426	147	147	147	5.49	124.99	124.66
PSMIGR1	3140	410781	3020	2974	2966	10.61	186.07	229.51

Table 2.3.1. Comparison of `MC47` with `MMD` and `MA27` orderings.

References

[1] Amestoy, P. R., Davis, T. .A., and Duff, I. S. (1995). An approximate minimum degree

ordering algorithm. Technical Report TR/PA/95/09, CERFACS, Toulouse. To appear in *SIAM J. Matrix Anal. and Applics.*

[2] Liu, J. W. H. (1985). Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Softw.* **11**, 141-153.

[3] Tinney, W. F. and Walker, J. W. (1967). Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE* **55**, 1801-1809.

2.4 Developments in frontal solvers (I. S. Duff and J. A. Scott)

We are concerned with the solution of $n \times n$ linear systems of equations

$$\mathbf{Ax} = \mathbf{b}, \tag{1}$$

where **A** is a large sparse matrix arising from finite-element analysis. The matrix **A** is a sum of elemental matrices

$$\mathbf{A} = \sum_{k=1}^{m} \mathbf{A}^{(k)}. \tag{2}$$

Each matrix $\mathbf{A}^{(k)}$ has entries only in the principal submatrix corresponding to the variables in element k and represents contributions from this element. This principal submatrix is assumed to be dense. The matrix **A** may be unsymmetric but the form (2) implies that the sparsity pattern is symmetric with nonzero diagonal entries. One possible direct solution method for (1), and the one which is still frequently the method of choice in many structural engineering applications, is the frontal method.

There are two major deficiencies with the frontal method.

• Far more arithmetic may be done than is required by the numerical factorization

• There is little scope for parallelism, other than that which can be obtained through the use of high level BLAS.

We have been concerned with looking at ways in which we can improve the performance of the frontal method and, in particular, the performance of our frontal code `MA42` (Duff and Scott [1], [2]).

One possible approach is to extend the basic frontal algorithm to use multiple fronts (Duff and Scott [3]). In a multiple front algorithm, the finite-element domain is partitioned into a number of subdomains and a frontal decomposition is performed on each subdomain separately. Since the factorizations of the subproblems are independent, this can be done in parallel. Once the assembly and eliminations on the subdomains are complete, there remains an interface problem to be solved. In our experiments, we solved the interface problem using a frontal method. To implement the multiple front algorithm, we developed the package `MA52`.

This is a collection of subroutines, that can be used in conjunction with the MA42 package.

We examined the performance of MA52 and MA42 in two parallel environments: on an eight processor shared memory CRAY Y-MP8I and on a network of five DEC Alpha workstations using PVM. The results we obtained for a model finite-element problem are encouraging and indicate that, for sufficiently large problems, high performance and good speedups can be achieved. Full details are given in [3].

For high performance computers where data must be cache resident before arithmetic operations can be performed on it, we are looking at enhancing the performance of MA42 by increasing the proportion of arithmetic operations performed using Level 3 BLAS. This is done by only performing eliminations once a block of pivots of a predetermined size is available. This can increase the amount of arithmetic but enables better reuse of data in the cache. Preliminary experiments on the Silicon Graphics Power Challenge machine using different pivot block sizes suggest that significant improvements in performance can be achieved using this approach.

References

[1] Duff, I. S. and Scott, J. A. (1993). MA42 – a new frontal code for solving sparse unsymmetric systems. Report RAL-93-064, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[2] Duff, I. S. and Scott, J. A. (1996). The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Math. Softw.*, to appear.

[3] Duff, I. S. and Scott, J. A. (1994). The use of multiple fronts in Gaussian elimination. Report RAL-94-040, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.5 Fortran 90 version of Harwell frontal code MA42 (J.K. Reid and J.A. Scott)

The code MA42 [1] performs frontal elimination for a finite-element problem, normally using direct-access files for holding the factors. There may be significant efficiency gains from splitting a large problem into subdomains, using MA42 on each subdomain to create a Schur complement matrix for the variables on the boundary of the subdomain, and finally using MA42 for the interface problem. This multiple front approach is hard to organize because of the long argument lists and the need for distinct workarrays and direct-access files. We have therefore constructed a Fortran 90 version that collects everything pertaining to a problem in a single object.

Using the Fortran 90 version, a typical run looks like this:

```
USE HSL_MA42
TYPE(MA42_DATA) DATA  ! DATA holds everything about the problem.
```

```
CALL MA42_INITIALIZE (DATA)
DO I = 1, NUMELT ! Read the integer data for the elements, one by one.
   READ . . .
   CALL MA42_ANALYSE (LIST(1:LEN),DATA)
END DO
DO I = 1, NUMELT ! Read the real data for the elements, one by one.
   READ . . .
   CALL MA42_FACTORIZE (LIST(1:LEN), &
                REALS(1:LEN,1:LEN),DATA)
END DO
READ . . . ! Read the data for the right-hand side.
CALL MA42_SOLVE (B,X,DATA)
```

The module looks after everything in DATA, except a few control or informative variables. Options may be specified in extra calls such as

```
CALL MA42_FILES(LENBUF,LENFILE,DATA)
```

which provides information on the files, by use of optional arguments, such as

```
CALL MA42_SOLVE(B,X,DATA,TRANS)
```

or setting components of the data structure, such as

```
DATA%ALPHA = 0.001
```

and extra information may be retrieved by accessing components, such as

```
FLOP_COUNT = DATA%FLOPS
```

The Fortran 90 code does run a little slower that the Fortran 77 code, hardly surprising since we call the old code from within a Fortran 90 jacket. Depending on the compiler and computer, we have found that the overhead varies from about 5% to about 50%.

References

[1] Duff, I. S. and Scott, J. A. (1993). MA42 – a new frontal code for solving sparse unsymmetric systems. Report RAL-93-064, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.6 Element resequencing for use with a multiple front solver (J. A. Scott)

The efficiency of a frontal scheme, in terms of both storage and computation time, is dependent upon the ordering of the elements. This is because, in the frontal method, the system matrix **A** is never assembled explicitly but the assembly and Gaussian elimination processes are interleaved, with each variable being eliminated as soon as its row and column are fully summed. This allows all intermediate working to be performed in a full matrix, termed the *frontal matrix*, whose rows and columns correspond to variables that have not yet been eliminated but occur in at least one of the elements that have been assembled. Since the order of the frontal matrix increases when a variable appears for the first time and decreases whenever a variable is eliminated, the order in which the elements is input is critical. In recent years, many algorithms for automatically ordering finite elements have been proposed in the literature.

In a multiple front algorithm (Duff and Scott [1]), the finite-element domain is partitioned into a number of subdomains and a frontal decomposition is performed on each subdomain separately. For a given partitioning of the domain, the efficiency of the multiple front algorithm depends on the ordering of the elements within each subdomain. The aim of our study was to look at the limitations of existing element reordering algorithms when applied to a subdomain and to consider how these limitations may be overcome.

The problem of ordering elements for use with a multiple front algorithm is more complicated than that of sequencing elements for a frontal solver on a single domain since it is necessary to distinguish between variables which can be eliminated once they are fully summed and interface variables that cannot be eliminated within the subdomain. We have considered two approaches which involve two different ways of locating a suitable starting element s for the reordering procedure. Once a starting element has been selected, both methods use a modification of the method of Sloan [2] to reorder the remaining elements. Our first method for choosing s is based on finding pseudo-peripheral nodes of the element communication graph. The second method introduces an artificial element, the guard element, and uses this extra element to find an element lying as far from the interface boundary as possible and uses this to start the reordering. We have tested both approaches on a range of problems and compared their performance with that of the Harwell Subroutine Library code `MC43`, which is designed for single domain problems. Both approaches give significant improvements over `MC43`, and the second method was almost always the method of choice. On the basis of our findings, a code `MC53` implementing this second method has been developed and is included in Release 12 of the Harwell Subroutine Library.

References

[1] Duff, I. S. and Scott, J. A. (1994). The use of multiple fronts in Gaussian elimination. Report RAL-94-040, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[2] S. W. Sloan (1986). An algorithm for profile and frontsize reduction of sparse matrices. *Int. J. Numer. Meth. Engng.* **23**, 239-251.

[3] J. A. Scott (1995). Element resequencing for use with a multiple front algorithm. Report RAL-95-029, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.7 Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems (I.S. Duff and J.K. Reid)

We consider the direct solution of sparse sets of *n* linear equations

$$\mathbf{Ax} = \mathbf{b}, \tag{1}$$

when the matrix **A** is symmetric and has a significant number of zero diagonal entries. An example of applications in which such linear systems arise is the equality-constrained least-squares problem

$$\underset{\mathbf{x}}{\text{minimize}} \ \|\mathbf{Bx} - \mathbf{c}\|_2 \tag{2}$$

subject to

$$\mathbf{Cx} = \mathbf{d}. \tag{3}$$

This is equivalent to solving the sparse symmetric linear system

$$\begin{pmatrix} \mathbf{I} & & \mathbf{B} \\ & \mathbf{0} & \mathbf{C} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{C}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \lambda \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{c} \\ \mathbf{d} \\ \mathbf{0} \end{pmatrix}. \tag{4}$$

Our earlier Harwell Subroutine Library code MA27 uses a multifrontal solution technique and is unusual in being able to handle indefinite matrices. It has a preliminary analysis phase that chooses a tentative pivot sequence from the sparsity pattern alone, assuming that the matrix is definite so that all the diagonal entries are nonzero and suitable as 1×1 pivots. For the indefinite case, this tentative pivot sequence is modified in the factorization phase to maintain stability by delaying the use of a pivot if it is too small or by replacing two pivots by a 2×2 block pivot.

The assumption that all the diagonal entries are nonzero is clearly violated in the above example. For such problems, the fill-in during the factorization phase of MA27 can be significantly greater than predicted by the analysis phase. Duff, Gould, Reid, Scott, and Turner [1] found that the use of 2×2 pivots with zeros on the diagonal alleviated this problem and also assisted the preservation of sparsity during the analysis phase. Our new code, MA47, is based upon this work and, like MA27, uses a multifrontal method. It will work for the definite case, but there are many opportunities for simplifications and efficiency improvements, so we plan to provide a separate code for this special case.

The design of the new code, MA47, is described in [2] and in more detail in [3]. It was hoped that MA47 would replace MA27 for the indefinite case, but our experience is that it is not always superior. A comparison summary is shown in Table 2.7.1 of runs on a CRAY Y-MP and a SUN SPARCstation 10 on a collection of test problems. The analyse phase of MA47 is more complicated and is inevitably more expensive, but we expect actual applications to amortize this over several factorizations and many solves. For factorization and solution, in general, MA47 with its default outperforms MA27 with its default, but it is clear that MA27 will continue to be needed for some problems. We are keeping both codes in the Harwell Subroutine Library.

		Total storage	Storage for factors		Time		
		Predicted	Predicted	Actual	Analyse	Factorize	Solve
CRAY	lower quartile	0.96	0.59	0.31	1.69	0.53	1.44
	median	1.45	1.15	0.57	2.64	1.38	1.82
	upper quartile	1.86	1.46	1.21	5.41	2.04	2.18
SUN	lower quartile	0.96	0.59	0.31	1.88	0.25	0.45
	median	1.45	1.15	0.57	3.32	0.96	0.75
	upper quartile	1.86	1.46	1.12	5.58	1.88	1.40

Table 2.7.1. `MA47` to `MA27` ratios using default parameter settings.

References

[1] Duff, I.S., Gould, N.I.M., Reid, J.K., Scott, J.A. and Turner, K. (1990). Factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal.* **11**, 181-204.

[2] Duff, I.S. and Reid, J.K. (1995). Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. Report RAL-TR-95-040, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear in *ACM Trans. Math. Softw.*

[3] Duff, I.S. and Reid, J.K. (1995). MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Report RAL-95-001, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.8 A combined unifrontal/multifrontal method (T.A. Davis and I.S. Duff)

Frontal methods (see Section 2.4) are particularly powerful inasmuch as they make considerable use of the Level 3 BLAS kernels while keeping the amount of integer overhead and data movement low. However, they usually require a far greater number of floating-point operations than methods using an ordering based on Markowitz or minimum degree. Multifrontal methods attempt to capture the benefits of both the full arithmetic of frontal schemes and the orderings designed to reduce fill-in and arithmetic. They are quite successful at this but still involve more data movement and indirect addressing than a frontal method. The object of this work is to combine elements of a frontal solver into a multifrontal solver in order to obtain a factorization that is efficient in arithmetic, number of operations, and overhead operations.

In the multifrontal method, several frontal matrices are used. Each is used for one or more pivot steps, and the resulting Schur complement is summed with other Schur complements to generate another frontal matrix. Although this means that arbitrary sparsity patterns can be handled efficiently, extra work is required to add the Schur complements together and can be costly because indirect addressing is required. The (uni–)frontal method avoids this extra work by factorizing the matrix with a single frontal matrix. Rows and columns are added to the frontal matrix, and pivot rows and columns are removed. Data movement is simpler, but higher fill-in can result if the matrix cannot be permuted into a variable-band form with small profile. We consider a combined unifrontal/multifrontal algorithm to enable general fill-in reducing orderings to be applied without the data movement of previous multifrontal approaches.

Suppose we attempt to factorize a matrix with the frontal method, but we give it a smaller working array than it requires. The factorization proceeds until the point at which the front size would grow larger than the working array. At this point the frontal method halts, unable to proceed. However, the remaining terms in the current frontal matrix form a Schur complement (which we refer to as a contribution block). We store the contribution block and deallocate the working array. A new pivot is then selected based on a fill-in reducing heuristic on the whole reduced matrix, and a new frontal matrix started with this starting pivot, in a newly allocated working array. The contribution block from the first frontal matrix is eventually assembled into a subsequent frontal matrix. This is the combined unifrontal/multifrontal method, described in more detail in [1].

In Table 2.8.1, we compare an earlier version of our unsymmetric multifrontal code (old) with one incorporating this new device (new) and include a comparison of these codes with the frontal code, `MA42`. The test matrices, from the Harwell-Boeing Collection and some large examples collected by Tim Davis, show quite clearly the advantage of using our combined unifrontal/multifrontal technique.

Matrix	Order	Entries	Total storage required (10^6)			Time (seconds on CRAY C-98)		
			`MA42`	Multifrontal		`MA42`	Multifrontal	
				(old)	(new)		(old)	(new)
GRE1107	1107	5664	0.1	0.3	0.2	0.21	0.30	0.30
GEMAT11	4929	33185	2.2	0.4	0.3	15.95	0.18	0.20
ORANI678	2529	90158	4.1	1.1	1.0	9.62	2.07	1.36
LHR04	4101	82682	0.7	1.5	0.8	4.21	2.51	1.79
HYDR1	5308	23752	1.1	0.6	0.4	8.01	1.05	1.10

Table 2.8.1. Comparison of unsymmetric multifrontal codes with `MA42`.

References

[1] Davis, T. A. and Duff, I. S. (1995). A combined unifrontal/multifrontal method for unsymmetric sparse matrices. Technical Report TR-95-020, Computer and Information Science Department, University of Florida.

2.9 An unsymmetric multifrontal code (T.A. Davis and I.S. Duff)

We have used the work discussed in Section 2.8 to design and develop a new multifrontal code for the solution of sets of sparse unsymmetric equations and have included this code in Release 12 of HSL as code `MA38`. Unlike `MA37` and `MA41`, `MA38` does not assume any symmetry in the matrix and its initial pivot selection considers also the values of the matrix entries and only selects a pivot if it satisfies the normal threshold test. In the current version of our code, we only offer a factorize which uses the same pivot sequence as in an earlier call to the analyse-factorize entry.

The underlying representation for this unsymmetric multifrontal method is a directed acyclic graph, rather than the elimination or assembly trees used by our other multifrontal codes. The frontal matrices are rectangular rather than square and the internal data organization during factorization is more complicated since frontal matrices are not usually absorbed at the parent node in the graph. In this respect, the approach is similar to that of the HSL code `MA47` (see 2.7).

In common with other multifrontal codes, high computational performance is obtained through the use of higher Level BLAS on the factorizations within the dense frontal matrices. This code will thus be most efficient when the matrix structure gives rise to large dense subblocks during the factorization. The algorithm also incorporates the use of unifrontal processing as discussed in Section 2.8.

We show, in Table 2.9.1 below, a comparison of the performance of `MA38` with `MA48` on a range of our standard test problems from the Harwell-Boeing Collection and some matrices collected by Tim Davis. We see clearly that, on some matrices, this new code is very competitive with `MA48` and can significantly outperform it on some problems. We note, however, that the functionality of the codes is somewhat different which more than justifies keeping both in the Harwell Subroutine Library.

Matrix	Order	Entries	Factorization time (secs)		Total storage (10^6 words)		Number fl-pt ops (10^6)	
			`MA48`	`MA38`	`MA48`	`MA38`	`MA48`	`MA38`
GRE1107	1107	5664	0.38	0.30	0.3	0.2	8.1	6.7
ORANI678	2529	90158	1.01	1.36	0.8	1.0	14.2	7.2
PSMIGR1	3140	543162	28.86	22.67	20.9	25.2	10465.3	9489.8
LNS 3937	3937	25407	3.37	1.78	2.2	1.2	280.4	84.1
HYDR1	5308	23752	0.81	1.10	0.4	0.4	0.9	2.7

Table 2.9.1. Comparison of `MA38` with `MA48`. Times in seconds on a CRAY C-98.

References

[1] Davis, T. A. and Duff, I. S. (1993). An unsymmetric-pattern multifrontal method for sparse

LU factorization. Report RAL 93-036, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear in *SIAM J. Matrix Anal. and Applics.*

[2] Davis, T. A. and Duff, I. S. (1995). A combined unifrontal/multifrontal method for unsymmetric sparse matrices. Technical Report TR-95-020, Computer and Information Science Department, University of Florida.

2.10 Unsymmetric multifrontal methods for finite-element problems (A.C. Damhaug and J.K. Reid)

We have constructed a new multifrontal code for unsymmetric finite-element sets of linear equations $\mathbf{AX} = \mathbf{B}$, which has been included in the Harwell Subroutine Library as `MA46`. The matrix **A** must be input by elements and be of the form

$$\mathbf{A} = \sum_{k=1}^{m} \mathbf{A}^{(k)}$$

where $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns that correspond to variables of the nodes of the *k*–th element. Optionally, the user may pass an additional matrix $\mathbf{A}_d$ of coefficients for the diagonal. **A** is then of the form

$$\mathbf{A} = \sum_{k=1}^{m} \mathbf{A}^{(k)} + \mathbf{A}_d .$$

The right-hand side **B** is expressed through the summation

$$\mathbf{B} = \sum_{k=1}^{m} \mathbf{B}^{(k)} .$$

The analysis phase accepts the matrix pattern by element-node connectivity lists and chooses diagonal pivots for Gaussian elimination to preserve sparsity while disregarding numerical values. The ordering is done with the minimum-degree heuristic. The final assembly tree is reordered to reduce the size of the working stack.

To support any kind of data base that might hold the element stiffness matrices, we use ‘reverse communication’ for the matrix factorization. The routine must be called by the user `NB` times, where `NB` is the number of assembly steps (internal nodes of the assembly tree), determined by the analysis code. In each call, the user must pass a specified sequence of finite-element coefficient matrices. Pivoting with the usual relative pivot tolerance is included.

For good performance on machines with cache storage, the user may provide the cache size. If necessary, the actual elimination is performed in blocks of columns of a size that allows the active part of the matrix to reside in the cache. On the biggest problem that we ran in performance tests, this reduced the factorization time with pure Fortran code from 1607 to 1149 secs on a SUN Sparcserver 670 mp and from 342 to 229 secs on a DEC 3000-400.

We use Level 3 BLAS for the actual computation. Where optimized vendor versions are available, very good performance can be obtained. Indeed, on the DEC our blocking for good cache usage did not help, presumably because the vendor-supplied BLAS itself uses blocking. For the big problem mentioned in the previous paragraph, the time was reduced to 197 secs by using vendor-supplied BLAS.

The new code comfortably outperformed the code MA37 in our tests. These are summarized in Table 2.10.1.

	DEC 3000-400			SUN Sparcserver 670 mp			CRAY Y-MP		
	Analyse	Factorize	Solve	Analyse	Factorize	Solve	Analyse	Factorize	Solve
lower quartile	2.3	2.5	1.1	2.1	1.8	0.9	1.5	3.2	1.0
median	3.3	2.7	1.4	2.9	2.0	1.1	1.7	3.8	1.2
upper quartile	13.7	3.7	1.6	11.5	2.2	1.2	7.0	4.0	1.4

Table 2.10.1. MA37 times divided by MA46 times.

More detail is available in the report [1].

References

[1] Damhaug, A.C. and Reid, J.K. (1996). MA46, a FORTRAN code for direct solution of sparse unsymmetric linear systems of equations from finite-element applications. Report RAL-TR-96-010, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.11 MUPS: a MUltifrontal Parallel Solver for sparse unsymmetric sets of linear equations (P.R. Amestoy and I.S. Duff)

We have developed our prototype MUPS package to produce a new HSL code, MA41, for the solution of sets of sparse unsymmetric equations using a multifrontal method. MA41 is similar to MA37 in the sense that the initial analysis is performed on a symmetrized pattern of the original matrix and assumes that pivots can be chosen from the diagonal in any order. It is thus best suited to systems which are structurally symmetric or nearly so and which are diagonally dominant. It differs from MA37 in several respects.

We have reorganized the structure of the code along the lines described in [3] and [4] so that it is no longer assumed that the code is being executed on a single processor. This makes the internal data structures a little more complicated but the overhead for uniprocessor running is almost negligible. We also require a more complex data management policy so that garbage collections do not interfere with parallel efficiency, see [2]. Another main change from MA37 is that we make extensive use of the higher Level BLAS which increases computational efficiency on all machines, particularly those with caches or vector chips [1]. Other enhancements to MA37 include options for scaling, permutation so that the diagonal is zero-free, error analysis, and iterative refinement.

The uniprocessor version is included in the standard release of the Harwell Subroutine

Library (Release 12), but versions for shared memory parallel computers are also available on request. The code is designed so that it is relatively simple to develop a version for any shared memory machine since the only primitives used by the code involve task starting and locks. At present, we have fully tested versions for the CRAY YMP/C90 range of machines and for an Alliant FX/80 and are currently writing a version for multiprocessor Sun workstations. Versions of the code have also been designed and run on some virtual shared memory machines, for example on a BBN TC 2000 and a KSR-1.

We show, in Table 2.11.1 below, a comparison of the performance of the uniprocessor version of `MA41` with `MA37` on a range of our standard test problems from the Harwell-Boeing Collection and some matrices collected by Tim Davis. These results indicate the improvements we have made and illustrate the power of using vendor-supplied BLAS on the HP 715/64.

Matrix	Order	Entries	Analyse time		Factorization time		Solve time	
			`MA41`	`MA37`	`MA41`	`MA37`	`MA41`	`MA37`
ORANI678	2529	90158	4.15	28.41	46.02	251.18	0.29	0.36
BCSSTK15	3948	117816	0.61	0.96	11.79	104.80	0.18	0.34
LNS 3937	3937	25407	0.32	0.34	1.79	13.22	0.07	0.12
SHERMAN3	5005	20033	0.61	0.31	1.07	3.23	0.05	0.06
GOODWIN	7320	324784	1.09	1.81	8.18	510.65	0.23	0.77
WANG3	26024	177168	8.63	5.09	299.50	1591.21	2.17	2.70

Table 2.11.1. Comparison of `MA41` with `MA37`. Times in seconds on an HP 715/64 workstation.

References

[1] Amestoy, P. R. and Duff, I. S. (1989). Vectorization of a multiprocessor multifrontal code. *Int. J. Supercomputer Applics.* **3** (3), 41-59.

[2] Amestoy P. R. and Duff I. S. (1993). Memory management issues in sparse multifrontal methods on multiprocessors. *Int. J. Supercomputer Applics.* **7**, 64-82.

[3] Duff, I. S. (1989). Multiprocessing a sparse matrix code on the Alliant FX/8. *J. Comput. Appl. Math.* **27**, 229-239.

[4] Duff, I. S. (1986). Parallel implementation of multifrontal schemes. *Parallel Computing* **3**, 193-204.

2.12 Multifrontal QR factorization in a multiprocessor environment (P.R. Amestoy, I.S. Duff, and C. Puglisi)

We have designed and implemented an algorithm for the parallel **QR** decomposition of a sparse matrix **A**. Our current version is designed for shared memory or virtual shared memory parallel architectures and only uses the primitives *task start* and *lockon*/*lockoff* or their equivalents. The algorithm is based on the multifrontal approach and makes use of Householder transformations. The tasks are distributed among processors according to an assembly tree which is built from the symbolic factorization of the matrix $\mathbf{A}^T\mathbf{A}$. Our approach is unusual inasmuch as we keep the orthogonal factors **Q** which may be required in some applications (for example if an orthogonal basis for the column space of **A** is wanted) and may be crucial in the solution of ill-conditioned or weighted least-squares problems. Another novel aspect of this work is that we use relaxation of the sparsity structure of both the original matrix and the frontal matrices to improve the performance by enabling more efficient use of the Level 3 BLAS.

We discussed this work extensively in the previous Progress Report (articles 2.11, 2.12, and 2.13 in [2]). Since then we have completed our main report on this work [1] and have developed and tuned codes for using the factorization to solve a least-squares problem.

If the **QR** factorization is being used to solve the sparse least-squares problem min $\|\mathbf{b}-\mathbf{Ax}\|_2$, and we have the factors **Q** and **R** then the solution is obtained by first multiplying **b** by $\mathbf{Q}^T$ and then solving the triangular system $\mathbf{Rx}=\mathbf{b}$. Since **Q** and **R** were produced by the multiprocessor multifrontal factorization, we must also use the underlying tree structure to effect an efficient parallel algorithm for these two phases. For the first calculation ($\mathbf{Q}^T\mathbf{b}$) we process the assembly tree from the leaf nodes to the root as in the factorization itself. The data management is, however, much simpler than in the factorization and we can update components of the product vector without synchronization worries. In the back substitution phase ($\mathbf{Rx}=\mathbf{b}$), we process the assembly tree from the root node to the leaf nodes. If the entries of **Q** are discarded and we want to solve the least-squares problem using the semi-normal equations then we also require the forward substitution step $\mathbf{R}^T\mathbf{y}=\mathbf{c}$ so we have also implemented this in parallel. We show, in Table 2.12.1 below, that these three phases in the solution of least-squares problems all parallelize well. The runs for this table were performed on an Alliant FX/80.

Matrix	$(m \times n)$	Number proc.	$\mathbf{Q}^T\mathbf{b}$		$\mathbf{Rx}=\mathbf{b}$		$\mathbf{R}^T\mathbf{y}=\mathbf{c}$	
			Time	Speedup	Time	Speedup	Time	Speedup
LARGE	(28254×6400)	1	2.59		2.69		2.44	
		8	0.62	4.18	0.42	6.40	0.43	5.66
MEDIUM2	(18794×12238)	1	2.84		1.79		1.66	
		8	0.77	3.69	0.32	5.59	0.37	4.52
LARGE2	(56508×34528)	1	10.98		5.49		3.10	
		8	3.53	3.11	1.11	4.95	1.19	2.59
NFAC80	(24964×6400)	1	1.80		1.02		0.97	
		8	0.34	5.29	0.18	5.67	0.18	5.25
NFAC100	(39204×10000)	1	2.98		1.65		1.54	
		8	0.70	4.26	0.27	6.11	0.31	4.98

Table 2.12.1. Time in seconds and speedups for multifrontal **QR** solution phases on an Alliant FX/80.

References

[1] Amestoy, P. R., Duff, I. S., and Puglisi, C. (1994). Multifrontal **QR** factorization in a multiprocessor environment. Technical Report TR/PA/94/09, CERFACS, Toulouse. To appear in *Numerical Linear Algebra and Applics.*

[2] Duff, I. S. (Editor). (1994). Numerical Analysis Group. Progress Report. January 1991 – December 1993. Report RAL 94-062, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.13 The solution of linear least-squares problems (J. Cardenal, I.S. Duff, and J. Jimenez)

We have designed and developed a general method for the linear least-squares solution of overdetermined and underdetermined systems. The method is particularly efficient when the coefficient matrix is quasi-square, that is when the number of rows and number of columns is almost the same. The numerical methods proposed in the literature for linear least-squares problems and minimum-norm solutions do not generally take account of this special characteristic. The proposed method is based on an **LU** factorization of the original quasi-square matrix **A**, assuming that **A** has full rank.

We have developed a framework based on the augmented system $\begin{pmatrix} \mathbf{I} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{c} \end{pmatrix}$ for presenting our algorithm and give a unified approach to solving both least-squares problems and minimum-norm problems.

In the overdetermined case, the **LU** factors are used to compute a basis for the null space of

$\mathbf{A}^T$. The right-hand side vector **b** is then projected onto this subspace and the least-squares solution is obtained from the solution of this reduced problem. In the case of underdetermined systems, the desired solution is again obtained through the solution of a reduced system. The use of this method may lead to important savings in computational time for both dense and sparse matrices.

We have studied the performance of our algorithm on practical test problems arising in the solution of problems from the computer simulation of the kinematic behaviour of multibody systems. Some results from this study are shown in Table 2.13.1.

	Code	Test 1	Test 2	Test 3	Test 4	Test 5
Rows		117	170	181	385	574
Columns		105	161	180	361	526
Entries		615	1069	1200	1661	3365
Normal equations	LAPACK	57.4	203.3	238.6	1914.1	5990.1
	MA27	12.1	32.3	37.0	100.3	288.6
Augmented system	MA27	7.3	15.6	13.1	43.4	120.7
Proposed Method	LAPACK	29.8	47.7	56.4	391.7	2124.2
	MA48	4.5	7.3	4.1	21.1	67.7

Table 2.13.1. Solution of sparse least-squares problems. Times in seconds on Silicon Graphics Onyx workstation (150 MHz).

The test cases all come from multibody simulation problems and, although small by current sparse matrix standards, they are typical of realistic problems arising in this application. Indeed the three smaller cases are more usual. It is thus quite surprising that the dense codes from LAPACK do so badly relatively to the sparse codes. From these results, we see that our proposed method does very well, especially if a good sparse code is used for the initial **LU** factorization of **A**. A full report on this work is presented in [1].

References

[1] Jimenez, J. M., Cardenal, J., and Duff, I. S. (1996). A projection method for the solution of linear least-squares problems. Report RAL-TR-96-013, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.14 Approximate-inverse preconditioners (N. I. M. Gould and J. A. Scott)

We are interested in using iterative techniques to solve large, sparse systems of linear equations

$$\mathbf{Ax} = \mathbf{b}. \tag{1}$$

Such methods invariably require one or more matrix-vector products per iteration; convergence is normally accelerated using a preconditioner $\mathbf{P}$, in which case the required matrix-vector products involve $\mathbf{AP}$ or $\mathbf{PA}$, not $\mathbf{A}$.

The solution of such problems remains a considerable challenge, especially if we are interested in robust methods for general problems. The difficulties are two-fold. Firstly, the theory of iterative methods for (1) is, at best, incomplete. In particular, it is difficult to be confident, when faced with a new problem, that a given algorithm will converge in a reasonable time, if at all. Secondly, while it is recognized that preconditioning the system often improves the convergence of a particular method, this is not always so. In particular, a successful preconditioner for one class of problems may prove ineffective on another class. Thus, it has long been recognized that the construction of successful general purpose preconditioners is unlikely to be possible.

We are interested in constructing approximations, $\mathbf{M}$, to the inverse of $\mathbf{A}$ for which $\| \mathbf{AM} - \mathbf{I} \|$ is small ($\mathbf{I}$ is the $n \times n$ identity matrix). So long as matrix-vector products involving $\mathbf{M}$ are inexpensive, the matrix $\mathbf{M}$ may then be a suitable preconditioner. There has been a lot of recent interest in such preconditioners because they have considerable scope for parallelization, and there is some evidence that they are perform well in practice. However, a proper assessment of their effectiveness in comparison with other preconditioners and of the correctness of the assumptions on which they are based has been lacking.

In this study, we investigate the use of sparse approximate-inverse preconditioners. We propose a number of enhancements which are shown to significantly improve their performance on some problems. We compare the use of sparse approximate-inverse preconditioners with incomplete LU factorization (ILU) preconditioners. Using a range of problems, we find that the sparse approximate-inverse methods are significantly more expensive to use on a single processor machine. However, the sparse approximate-inverse methods can be successful when ILU preconditioners fail, and preliminary studies suggest the imbalance in the computation times may be redressed when the methods are used in parallel.

References

[1] N. I. M. Gould and J. A. Scott (1995). On approximate-inverse preconditioners. Report RAL-95-026, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.15 Linear algebra kernels for parallel domain decomposition methods (L. Carvalho, I. S. Duff, and L. Giraud)

The modelling of many applications in physics and engineering gives rise to systems of nonlinear partial differential equations (PDEs). Approximate solutions to these PDEs are obtained by nonlinear iterative procedures that linearize and discretize the equations. Each step of the nonlinear procedure involves the solution of large sparse linear systems. These systems, for example those arising in 3D numerical simulations, can be very large and can seldom be solved using only direct methods as they may often require too much memory. Equally, the use of standard iterative techniques, such as those based on Krylov sequences, is not practical due to the slow convergence. We thus combine iterative techniques with direct methods for solving subproblems to accelerate the convergence. Examples of this hybrid approach are block preconditioned Krylov methods and domain decomposition methods. These are additionally suitable for parallel computing on distributed memory machines.

In this work, we study the parallel implementation of a domain decomposition method based on a Schur complement approach. In this approach, we divide the problem into subdomains and, within each subdomain, we partition the variables into internal and interface variables. We can then solve for the internal variables of each subdomain using a direct method. The resulting problem in the interface variables has the Schur complement matrix as coefficient matrix. We solve this using an iterative method. The main concern with this class of techniques is how to precondition the Schur complement problem.

The preconditioning technique that we use is a block Jacobi preconditioning, where the blocks correspond to the contribution to the diagonal blocks of the Schur complement matrix from each subdomain. We approximate the inverse of each of these blocks by tridiagonal matrices using the probing technique of Chan [2]. This approximation is then factorized and used as a preconditioner to accelerate the conjugate gradient iteration on the interface variables. We have found that the method is strongly influenced by the direct method used to solve the subproblems and show, in Table 2.15.1 below, the effect of using different direct solvers. The band solver used was DPBTRF/S from LAPACK. The PCG code used in solving the subproblems was an ICCG code developed by Notay et al. [3].

Solver	Memory requirement (in Mbytes)	Factorization times (in sec)	Solve times (in sec)
Band	2.12	0.875	0.103
Skyline	1.31	1.034	0.061
MA27	0.70	0.409	0.027
PCG	0.63	0.008	0.875

Table 2.15.1. Comparison between the different linear solvers on a single 64×64 domain. Times on a single processor of a Meiko CS2-HA.

We thus choose to solve the subproblems with `MA27` and have experimented with this solution scheme on various parallel environments including a Meiko CS2-HA and a network of Sun SPARC-10 workstations. We have studied the effect of different partitioning strategies and have examined speedup and scaled speedup. More details of these results can be found in [1].

References

[1] Carvalho, L., Duff, I. S., and Giraud, L. (1995). Linear algebra kernels for parallel domain decomposition methods. Report TR/PA/95/26, CERFACS, Toulouse.

[2] Chan, T. F. and Mathew, T. P. (1992). The interface probing technique in domain decomposition. *SIAM J. Matrix Anal. and Applics.* **13**, 212-238.

[3] Notay, Y., Gheur, V., Ould Amar, Z., Petrova, S., and Saint-Georges, P. (1995). ITSOL: an evolving routine for the iterative solution of symmetric positive definite systems. Report GANMN 9502, Université Libre de Bruxelles.

2.16 Solution of large sparse unsymmetric linear systems with a block iterative method in a multiprocessor environment (M. Arioli, L. A. Drummond, I. S. Duff, and D. Ruiz)

Block iterative methods for the solution of large sparse sets of linear equations are well suited for implementation on parallel computers since the solutions of the partial problems on each block can be performed independently. We were particularly interested in the Block Cimmino algorithm since there is no data communication between the subblocks and these subproblems can be solved simultaneously. We have already designed and experimented with the Block Cimmino algorithm, accelerated by conjugate gradients and block conjugate gradients, on shared memory parallel computers [3], [4]. This work was discussed in Section 2.15 of the previous Progress Report [6].

We have extended this work to enable the implementation on distributed memory computers and networks of workstations. This has involved the consideration of many issues not present in the shared memory version. The issues of scheduling and load balancing have resulted in the design of a scheduler for heterogeneous computing environments that is discussed in detail in the thesis of Tony Drummond [5].

We made a detailed study of the block conjugate gradient algorithm using various strategies for distributing the computation. Each strategy has its advantages and disadvantages but we found that, on most platforms and on most problems, it was best to use a master-slave distributed implementation in which the master performs an initial scheduling and distributes the tasks to the slaves. Thereafter the master is only used for global operations like computing inner products and testing for termination. The alternatives that we compared this strategy with were only to distribute matrix-vector multiplies and one where there is no master process but more redundant computation. We used the master-slave distributed implementation of the block conjugate gradient algorithm as the preconditioner for a distributed block Cimmino algorithm.

We have found it very beneficial to allow the scheduler to group closely related tasks on the same processor and have also experimented with different strategies for the initial subdivision of the equations for the block Cimmino algorithm. Choosing the best strategies in each case, we obtain impressive speedups, as is illustrated in Table 2.16.1 below.

Further information on this work can be found in [1], [2] and [5].

Number of processors	1	2	4	8	10
Time	8.3	4.3	2.5	1.4	1.0
Speedup	–	1.9	3.3	5.9	8.3
Efficiency	–	0.95	0.83	0.74	0.83

Table 2.16.1. Time in seconds on an IBM SP2 for solution of problem SHERMAN4 from the Harwell-Boeing Collection.

References

[1] Arioli, M., Drummond, A., Duff, I. S., and Ruiz, D. (1995). A parallel scheduler for block iterative solvers in heterogeneous computing environments. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing.* Edited by David H Bailey et al. SIAM, Philadelphia., 460-465.

[2] Arioli, M., Drummond, A., Duff, I. S., and Ruiz, D. (1995). Parallel block iterative solvers for heterogeneous computing environments. In *Algorithms and Parallel VLSI Architectures III* Edited by M. Moonen and F. Catthoor. Elsevier, Amsterdam, 97-108.

[3] Arioli, M., Duff, I. S., Noailles, J., and Ruiz, D. (1992). A block projection method for sparse equations. *SIAM J. Sci. Stat. Comput.* **13**, 47-70.

[4] Arioli, M., Duff, I. S., Ruiz, D., and Sadkane, M. (1992). Block Lanczos techniques for accelerating the Block Cimmino method. *SIAM J. Sci. Comput.* **16**, 1478-1511.

[5] Drummond, L. A. (1995). Solution of general linear systems of equations using block Krylov based iterative methods on distributed computing environments. PhD Thesis. Report TH/PA/95/40, CERFACS, Toulouse.

[6] Duff, I. S. (Editor). (1994). Numerical Analysis Group. Progress Report. January 1991 – December 1993. Report RAL 94-062, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.17 An evaluation of software for sparse nonsymmetric matrices (R. B.Lehoucq and J. A. Scott)

The last few years has seen a significant increase in research into numerical methods for computing selected eigenvalues (and eigenvectors) of large sparse nonsymmetric matrices. This interest has led to a large number of papers and reports on possible numerical methods for solving this problem. It has also begun to lead to the development of high quality mathematical software. However, the published numerical results are extremely limited and, in general, the authors of the software have provided few results comparing the performance of their software with that of rival software. The aim of the our study is to evaluate this state-of-the-art software in terms of the following criteria:

• The user interface

• Storage requirements

• Performance

• Accuracy and stability

• Reliability and robustness.

There are three methods which have received significant attention by the numerical analysis community. These are subspace, or simultaneous, iteration, Arnoldi's method and the (nonsymmetric) Lanczos' method. It is our intention to provide a comprehensive comparative study of these three methods together with the recent Jacobi-Davidson method. So far, we have considered subspace iteration and Arnoldi methods, for which several high quality codes have appeared. The scope of our study is restricted to software which is available either in the public domain or under licence. For the Lanczos method, there is currently only a very limited amount of such software. As far as the authors are aware, the only code which falls within the criteria for inclusion in this study is the code `EIGLAL` of Freund and Nachtigal. At present, there is no software implementing the Jacobi-Davidson method which meets our criteria.

The results of our subspace iteration and Arnoldi experiments on a wide-range of practical problems show that the algorithms are very sensitive to implementation details. None of the codes is able to solve all the problems included in our test set. Difficulties were encountered in the case of multiple eigenvalues and highly non-normal matrices. We also conclude that an implicitly restarted Arnoldi iteration algorithm appears to provide a promising way forward, but there are many aspects of the practical algorithm that require further investigation before a robust code becomes available.

References

[1] Lehoucq, R. B. and Scott, J. A. (1996). An evaluation of subspace iteration software for sparse nonsymmetric matrices. Technical Report, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear.

[2] Lehoucq, R. B. and Scott, J. A. (1996). An evaluation of Arnoldi software for computing eigenvalues of sparse nonsymmetric matrices. Technical Report, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear.

2.18 Sparse BLAS (I.S. Duff, M. Marrone, G. Radicati, and C. Vittoli)

The Basic Linear Algebra Subprograms (BLAS) have had a profound influence on the development of algorithms and software for the solution of systems of linear equations with full coefficient matrices. Although a set of sparse BLAS was developed [1], their use and manufacturer-supported availability are not very widespread. This is in part due to the fact that they are only Level 1 BLAS and suffer from the same shortcomings of inefficiency on modern computer architectures as in the full case. Another reason why they are not widely accepted is that, at least for direct sparse solution, it is more efficient to code the kernels using full linear algebra so that the full matrix BLAS can be used (see, for example, [2]).

However, in the solution of sparse equations using iterative methods, there is an urgent need for a standard interface for a sparse matrix by matrix multiplication and a sparse triangular solution routine. We have therefore designed routines for these cases as Level 3 BLAS so that the Level 2 equivalents are available as a particular case. In the sparse case, there is the problem of how the matrix is stored, and indeed storage schemes differ very widely and are usually applications dependent. We have therefore also designed routines for data conversion with a standard interface to include all the data structures we are familiar with. Finally, we also include two permutation routines to facilitate the efficient use of the kernels in iterative solvers.

In response to comments on our earlier report [3], we have made many changes to that proposal and have developed a revised proposal to address these points including designing a harness in Fortran 90 [4]. A prototype code for the sparse BLAS is available from the anonymous ftp server at RAL in file spblas.f in directory pub on seamus.cc.rl.ac.uk.

References

[1] Dodson, D. S., Grimes, R. G., Lewis, J. G. (1991). Sparse extensions to the Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **17**, 253-263.

[2] Duff, I. S. (1981). Full matrix techniques in sparse Gaussian elimination. In *Numerical Analysis Proceedings, Dundee 1981.* Lecture Notes in Mathematics 912. G.A. Watson (editor). Springer-Verlag, Berlin, 71-84.

[3] Duff, I. S., Marrone, M., and Radicati, G. (1992). A proposal for user level sparse BLAS. SPARKER Working Note #1. Report RAL 92-087, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[4] Duff, I. S., Marrone, M., Radicati, G., and Vittoli, C. (1995). A set of Level 3 Basic Linear Algebra Subprograms for sparse matrices. Report RAL-TR-95-049, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

2.19 Sparse matrix test problems (I.S. Duff)

Release I of the Harwell-Boeing Collection of sparse matrices consists of sparse matrices in a standard format from a wide range of application areas. The format used is basically a column-pointer row-index format and is described in [1]. There are both symmetric and unsymmetric systems, some rectangular matrices, and a few in unassembled element format. Sometimes only the pattern of the sparse matrix is given, and sometimes a right-hand side vector is supplied in addition to the reals.

We have nearly completed the design of Release II of the Collection and will shortly circulate it for comment [2]. This redesign has proved to be a significant task since we wish to enhance the value and scope of the Collection while effectively maintaining upward compatibility with the present release. In particular, we have extended the format to include more information on matrices and on systems being solved. These include starting guesses, exact solutions, eigenvalues and vectors, singular values and vectors, permutations, partitions, and geometric data. We have also made it a requirement that the format of the matrices enables them to be read easily from C programs. Although we will include matrix generation programs in the Release II repository, we will not include generated matrices in the Collection unless they are explicitly submitted and accepted for the Collection. We propose this because we feel that one of the main strengths of the collection is its reproducibility and do not want to jeopardize this aspect.

We have written complete Fortran subroutines both to read matrices in the Collection and to write a matrix in Harwell-Boeing format. These routines are included in the Harwell Subroutine Library as `MC36` and `MC37` and are discussed in more detail in Section 7.4 of this report. They are also distributed with the test Collection.

Release I is available by anonymous ftp from both RAL (seamus.cc.rl.ac.uk) and CERFACS (ftp.cerfacs.fr), in both cases in directory pub/harwell_boeing. More recently, some researchers at NIST in Washington DC and their collaborators have been developing an efficient World Wide Web interface to the Collection.

References

[1] Duff, I. S., Grimes, R. G., and Lewis, J. G. (1992). Users' Guide for the Harwell-Boeing Sparse Matrix Collection. Report RAL 92-086, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[2] Duff, I. S., Grimes, R. G., and Lewis, J. G. (1996). The organization of Release II of the Harwell-Boeing Sparse Matrix Collection. To appear.

3 Optimization

3.1 LANCELOT (A.R. Conn, N.I.M. Gould, and Ph.L. Toint)

Since its release in 1992, the large-scale nonlinear optimization package LANCELOT (Release A) [1] has been installed at over 150 sites throughout the world. Over the past two years, research has commenced on its successor. In order to design an improved algorithm, it has been necessary to test the existing algorithm on as large a class of representative problems as possible [2], and to compare the algorithm against existing state-of-the-art codes [3].

We have identified a number of areas in which the algorithm needs to be improved.

(i) The algorithm does not handle linear constraints very effectively. Linear constraints are treated as if they were general nonlinear constraints.

(ii) The algorithm is sensitive to degeneracy at the solution to the problem. Although the algorithm will converge to a degenerate solution, the rate of convergence may significantly degrade in the presence of degeneracy.

(iii) Inequality constraints are handled inefficiently by means of slack variables. The resulting increase in problem dimension is sometimes severe.

Current research is concentrating on interior-point methods for handling the model subproblem as it is felt that such an approach is capable of coping with all of the above-mentioned disadvantages of LANCELOT A. However, complications arise as the model problem may be non-convex, thus requiring a significant reappraisal of interior-point methods.

In the interim, LANCELOT A has been translated into Fortran 90, using many of the advanced features of the new language to simplify the calling sequences and to remove restrictions on fixed-size workspace. The new code has one new feature, the opportunity to generate problem derivatives automatically. This is possible using the new Harwell Subroutine Library Fortran 90 package `AD01` (see Section 4.1).

References

[1] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). LANCELOT *: a Fortran package for large-scale nonlinear optimization (Release A).* Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York.

[2] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. Technical Report 92-075, Rutherford Appleton Laboratory, Chilton, England, to appear *Mathematical Programming.*

[3] Bongartz, I., Conn, A. R., Gould, N. I. M., Saunders, M. A., and Toint, Ph. L. (1996). A numerical comparison between the LANCELOT and MINOS packages for large-scale

nonlinear optimization Technical Report in preparation, Rutherford Appleton Laboratory, Chilton, England.

3.2 Iterated-subspace minimization methods (A. R. Conn, N. I. M. Gould, A.Sartenaer, and Ph. L. Toint)

The problem of minimizing a function of a large number of variables is considered. We are interested in the case where the value of the objective function and possibly its derivatives are cheap to compute. We consider a class of Iterated-Subspace Minimization (ISM) methods for solving these problems.

At each major iteration of such a method, a low-dimensional manifold, the iterated subspace, is constructed and an approximate minimizer of the objective function in this manifold then determined. The iterated subspace is chosen to contain vectors which ensure global convergence of the overall scheme and may also contain vectors which encourage fast asymptotic convergence.

The iterated subspace is determined by applying the conjugate gradient method to the Newton equations at the current major iteration. The conjugate gradient method generates a sequence of search directions as it attempts to minimize the current quadratic model. The most promising of these directions make up the iterated subspace. In particular, the subspace comprises

(a) the steepest descent direction, which guarantees convergence;

(b) the truncated-Newton direction, which guarantees rapid convergence; and

(c) a collection of the remaining directions, chosen to give large decreases in the model, or because they give approximations to dominant eigenvalues.

This approach is compared with truncated-Newton and limited-memory methods, as well as with LANCELOT. The ISM approach is seen to be competitive with, and in some cases superior to, these approaches. Extensions to bound-constrained and linearly constrained minimization methods are suggested.

References

[1] Conn, A. R., Gould, N. I. M., Sartenaer, A., and Toint, Ph. L. (1994). On iterated-subspace minimization methods for nonlinear optimization. Technical Report 94-069, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. Revised December 1995.

3.3 Element-by-element preconditioners in optimization (M. J. Daydé, J.–Y. L'Excellent, and N. I. M. Gould)

Most functions f, of a large number of variables $\mathbf{x}$, can be expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}),$$

where each of the *element* functions $f_i(\mathbf{x})$ has a large invariant subspace. Such a function is said to be *partially separable* [5]. Normally, an element function will have a large invariant subspace because it only depends on a small number of variables, but this is not always the case [3]. However, any sufficiently differentiable function with a sparse Hessian matrix is partially separable so this decomposition is extremely general [5]. Note that the decomposition is not unique.

At the heart of any iterative algorithm for minimizing a partially separable function, an approximate solution to the Newton equations

$$\left(\sum_{i=1}^{m} \mathbf{H}_i\right) \mathbf{s} = -\left(\sum_{i=1}^{m} \mathbf{g}_i\right),$$

where $\mathbf{g}_i = \nabla_{\mathbf{x}} f_i(\mathbf{x})$ and $\mathbf{H}_i = \nabla_{\mathbf{xx}} f_i(\mathbf{x})$, is sought. The method of choice is now considered to be the preconditioned conjugate gradient method (see, for example, [4]).

In [1], we consider preconditioners which aim to mimic the structure implicit in the Newton equations. These element-by-element methods ([6], [7]) have proved most effective when applied to problems arising from the solution of partial differential equations. We consider a number of existing and new element-by-element preconditioners, and report that these preconditioners appear to be effective in the optimization context. However, their performance can be significantly improved if groups of elements are amalgamated – treated as a dense block – before the preconditioner is constructed. A variety of amalgamation techniques are examined, and the best of these is seen to generate very effective preconditioners. All of the proposed methods are highly parallelizable and are seen to be effective on parallel machines [2].

References

[1] Daydé, M. J., L'Excellent, J.–Y., and Gould, N. I. M. (1995). On the use of element-by-element preconditioners to solve large-scale partially separable optimization problems. Report RAL-TR-95-010, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[2] Daydé, M. J., L'Excellent, J.–Y., and Gould, N. I. M. (1995). Solution of structured systems of linear equations using element-by-element preconditioners. ENSEEIHT-IRIT technical report RT/APO/95/1, Toulouse, France.

[3] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1990). An introduction to the structure of

large scale nonlinear optimization problems and the LANCELOT project. In R. Glowinski and A. Lichnewsky, editors, *Computing Methods in Applied Sciences and Engineering*, pages 42-54, SIAM, Philadelphia, USA.

[4] Golub, G. H., and Van Loan, C. F. (1989). *Matrix Computations*. Second Edition. Johns Hopkins University Press, Baltimore.

[5] Griewank, A., and Toint, Ph. L. (1982). On the unconstrained optimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301-312, Academic Press, London and New York.

[6] Hughes, T. J. R., Levit, I., and Windget, J. (1983). An element-by-element solution algorithm for problems of structural and solid mechanics. *Computational Methods in Applied Mechanics and Engineering*, **36**, 241-254.

[7] Wathen, A. J. (1989). An analysis of some element-by-element techniques. *Computational Methods in Applied Mechanics and Engineering*, **74**, 271-287.

3.4 Linearly constrained minimization (N. I. M. Gould)

We consider solving the problem

$$\text{(1)} \qquad \underset{\mathbf{x}\in R^n}{\text{minimize}}\, f(\mathbf{x})$$

subject to a set of *m* independent linear equations

$$\text{(2)} \qquad \mathbf{Ax} = \mathbf{b},$$

where *f* is twice continuously differentiable. We assume that $\nabla_{\mathbf{xx}} f(\mathbf{x})$ is available and that we wish to exploit this curvature information; contrary to popular belief, this assumption holds for a wide variety of applications.

Our interest here lies in algorithms which require the solution of problems of the form (1)–(2) as subproblems. It is likely that the next version of LANCELOT [1] will be based, in part, upon such subproblems. We are particularly concerned with the case where *n* and possibly *m* are large, and the matrices **A** and $\nabla_{\mathbf{xx}} f(\mathbf{x})$ are sparse.

In [2], we discuss general issues of convergence for schemes for solving (1)–(2) and lay the foundations for the linear algebraic processes we later employ. In particular, if $\mathbf{Ax_e} = \mathbf{b}$, we exploit the structure of the Newton equations

$$\text{(3)} \qquad \begin{pmatrix} \nabla_{\mathbf{xx}} f(\mathbf{x}) & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ \lambda \end{pmatrix} = - \begin{pmatrix} \nabla_{\mathbf{x}} f(\mathbf{x}) \\ \mathbf{0} \end{pmatrix},$$

for a correction **p** to $\mathbf{x_e}$. When $f(\mathbf{x})$ is not convex, these Newton equations may not be appropriate and we discuss suitable replacements for $\nabla_{\mathbf{xx}} f(\mathbf{x})$ in (3). Our aim is to ensure that the replacement is positive definite in the null-space of the constraints, while not adversely affecting the convergence of Newton's method nor incurring a significant computational overhead.

We consider methods which form a sparse factorization of the coefficient matrix in (3). The matrix is reordered to ensure a stable factorization, and the factorization is continued so long as it determined that $\nabla_{\mathbf{xx}} f(\mathbf{x})$ need not be altered. Once it is realized that $\nabla_{\mathbf{xx}} f(\mathbf{x})$ is insufficient, diagonal perturbations are made. Two different approaches are considered, an implicit method in which any potentially bad diagonal is modified, and subsequently unmodified if the modification turns out to be unnecessary, and an explicit method in which perturbations are only made if they can be shown to be necessary.

Numerical experiments [2] show that the technique can be effective, in that modified factorizations are possible at little additional cost over an unmodified (but, in this application, unusable) factorization. However, it is unclear which of the two proposed modifications is the more successful, each having good and bad instances. A software package incorporating many of these ideas is currently under development.

References

[1] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). LANCELOT *: a Fortran package for large-scale nonlinear optimization (Release A).* Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York.

[2] Gould, N. I. M. (1995). Constructing appropriate models for large-scale, linearly-constrained, nonconvex, nonlinear, optimization algorithms. Report RAL-TR-95-037, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

4 Applied Mathematics

4.1 Automatic differentiation in Fortran 90 (D. Cowey, J.D. Pryce, and J.K. Reid)

We have developed a Fortran 90 package for automatic differentiation and placed it in the Harwell Subroutine Library. It contains modules for both the forward and the backward method.

In the forward method, for each independent variable and each dependent variable, the module holds a representation of the values of the variable and all its derivatives of up to a requested order. For each elementary operation, the desired derivatives of the result are calculated from those of the primaries by the chain rule. For example, if

$$a = b*c,$$

we have

$$\frac{\partial a}{\partial t} = \frac{\partial b}{\partial t}\,c + b\,\frac{\partial c}{\partial t}.$$

All the derivatives are calculated at the same time as the values.

In the backward method, a graph is constructed to represent the whole computation, with a node i for each independent variable, $i = 1, 2, ..., m$, and a node i for the result of each elementary operation, $i = m{+}1, 2, ..., n$, with links to nodes for the primaries of the operation. The nodes are in execution-order sequence, so the links are always to nodes with lesser indices. Only the values are constructed initially in the forward pass that constructs the tree. Let us use the notation x_i for the value at node i and suppose that derivatives of $f = x_n$ are required. As well as x_i, $\frac{\partial f}{\partial x_i}$ is held at node i, $i = 1, 2, ..., n$. The derivatives are calculated by `AD01` subroutines. Initially, all the variables are regarded as independent so that all the derivatives are zero except at node n where the derivative value is 1. One by one, from n backwards to m+1, the variables are changed to be dependent and the derivatives updated by the chain rule. For example, if

$$a = g(b,c),$$

when a is changed to dependent, we have

$$\frac{\partial f}{\partial b}^{new} = \frac{\partial f}{\partial b}^{old} + \frac{\partial f}{\partial a}^{old}\frac{\partial g}{\partial b}.$$

The forward method is likely to be best if the number of independent variables is small, since then the extra work and storage to compute and hold all the derivatives as the computation proceeds is modest, though we store only the non-zero derivatives if that is advantageous. If some variables are dependent on a large number of independent variables, the forward method becomes impractical, but the work of the backward method is bounded by a small fixed multiple of the work needed for the values themselves. The disadvantage of the backward method is that the whole computational tree has to be stored, which is not practical for very long computations.

The user must make changes in every program unit that contains a variable whose derivatives are required or is invoked in the course of calculating the value of such a variable. The changes are

1. Add a `USE` statement for the module.
2. Change the type of all independent variables and variables whose values vary with the independent variables to `TYPE(AD01_REAL)` and give them the special initial value `AD01_UNDEFINED`.
3. Initialize the module with a call to `AD01_INITIALIZE`.
4. Execute the modified code
5. Obtain the required derivatives with calls to `AD01` subroutines.

First and second derivatives may be obtained with the backward method and derivatives of any order with the forward method.

There are facilities for storing and restoring the module data, which permits the calculation to be suspended while a subsidiary calculation takes place.

5 Numerical Linear Algebra

5.1 Computational kernels on vector and parallel machines and on RISC architectures (M.J. Daydé and I.S. Duff)

The Basic Linear Algebra Subprograms (BLAS), particularly the Level 3 BLAS, are one of the main platforms for the efficient implementation of a wide range of algorithms on modern high performance computers. The rapid development of new machines means that it is important to maintain a watching brief over new architectures and to develop and tune the BLAS in cases when the manufacturer fails to do so.

It is an impossible task to so redesign all of the BLAS (even just the Level 3 BLAS) for each machine so we have developed BLAS based on the GEMM, or matrix-matrix multiply, kernel so that only this one routine need be tuned for the machine. The GEMM kernel is usually the first to be optimized by the vendor and it is relatively easy to tune code for this operation. This work is described in detail in [2] and [3].

Our recent work, which will shortly appear in the report [2], shows that, even using a standard version for GEMM, our Level 3 BLAS can outperform some of the vendor-supplied BLAS on most machines including the Silicon Graphics Power Challenge and the DEC 8400 5/300 Turbo laser. Furthermore, if our kernels use the vendor-supplied version of GEMM, they can be made to run even faster.

We make extensive use of these new building blocks in linear algebra software and demonstrate that the high speed of these kernel operations can be reflected in the speed of the linear equation solvers [1].

References

[1] Amestoy P. R., Daydé M. J., Duff I. S., and Morère P. (1995). Linear algebra calculations on a virtual shared memory computer. *Int. J. High Speed Computing* **7**, 21-43.

[2] Daydé M. J. and Duff I. S. (1996). A blocked implementation of Level 3 BLAS for RISC processors. Report RAL-TR-96-014, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[3] Daydé M. J., Duff I. S., and Petitet A. (1994). A parallel block implementation of Level 3 BLAS kernels for MIMD vector processors. *ACM Trans. Math. Softw.* **20**, 178-193.

5.2 Infinite weights in the Powell-Reid method for linear least squares problems (J.K. Reid)

Powell and Reid [1] considered the application of Golub's method to the linear least squares problem

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{1}$$

in the case that the norms of the rows of **A** vary widely. They showed that Householder transformations could be applied to produce a QR factorization

$$\mathbf{A}\,\mathbf{P} = \mathbf{Q}\,\mathbf{R} \tag{2}$$

of a column permutation of **A** in a backwards stable manner provided both row and column interchanges were included. They had in mind the case of weighted least squares problems with widely varying weights where the square roots of the weights were applied explicitly. That is, to find

$$\min\ (\mathbf{c}-\mathbf{Bx})^T\mathbf{W}^2(\mathbf{c}-\mathbf{Bx}) \tag{3}$$

they solved the unweighted problem (1) with

$$\mathbf{A} = \mathbf{WB}, \quad \mathbf{b} = \mathbf{Wc}. \tag{4}$$

We have shown [2] that the algorithm and its error analysis extends very simply to the case of infinite weights, that is, to the constrained and weighted least squares problem.

References

1. Powell, M.J.D. and Reid, J.K. (1969). On applying Householder transformations to linear least squares problems. In Information Processing 68, Ed. Morrell, A.J.H., North-Holland, 122-126.

2. Reid, J.K. (1996). Infinite weights in the Powell-Reid method for linear least squares problems. To appear.

6 Miscellaneous Activities

6.1 CERFACS (I.S. Duff)

Iain has continued to lead a project at CERFACS on Parallel Algorithms and several of the contributions to this report reflect interactions with that team. One of the main activities of the “Algo” team over the past year has been the hosting of an International Linear Algebra Year. This Year was preceded by a Workshop in the Pyrenean town of St Girons in July 1994. Attendance was by invitation and around forty international researchers in sparse matrices participated, including all members of the Group at RAL. The Linear Algebra Year is being held from September 1995 until September 1996 and includes four workshops and a visitor’s programme. Iain is co-chairing the international scientific committee with Gene Golub from Stanford and is participating in all the workshops. John spoke at the first Workshop on “Direct Methods” and Jennifer at the second on “Eigenvalues and beyond”. Nick is helping to organize the optimization workshop that will take place in Albi in April 1996.

The main areas of research in the Parallel Algorithms Group are the development and tuning of kernels for numerical linear algebra, the solution of sparse systems using direct methods or iterative methods or a combination of the two, heterogeneous computing including the use of PVM and MPI and the design of schedulers, large eigensystem calculations, optimization, and the reliability of computations. Other activities of the Group include advanced training by both courses and research and the porting of industrial codes [4].

During the reporting period, two students completed their PhDs at CERFACS. Thierry Braconnier completed his thesis in May 1994 [1], and Tony Drummond who finished in December 1995 was co-supervised by Iain [5]. Jean-Yves L’Excellent, whom Nick co-supervised, was a regular visitor to CERFACS and both Nick and Iain were on his jury in November 1995 [6].

Nick continued to visit both CERFACS and ENSEEIHT-IRIT with the support of a British Council grant to enable him to develop and extend some of the work commenced when he spent a year at CERFACS in 1994.

The main projects at CERFACS still include Parallel Algorithms and Computational Fluid Dynamics although recent emphasis on environmental modelling has led to a significant increase in the size of the Climate Modelling Group. There are smaller groups in electromagnetics, signal processing, shape optimization, and computational chemistry, the latter currently a subgroup of the Parallel Algorithms Group. At the beginning of 1995, Jean-Claude André from the French meteorological service succeeded Roland Glowinski as the Director of CERFACS. A brief summary of activities at CERFACS, especially in the Parallel Algorithms Project, can be found in [3] and a fuller account of recent activities of all groups can be found in [2]. Current information on the Parallel Algorithms Group can be found on page http://www.cerfacs.fr/algor/ of the World Wide Web.

References

[1] Braconnier, T. (1995). Sur le calcul de valeurs propres en précision finie. PhD Thesis. Report TH/PA/94/24, CERFACS, Toulouse.

[2] CERFACS Scientific Report 1994. CERFACS, Toulouse. June 1995.

[3] Daydé, M. and Duff, I. S. (1994). The CERFACS Experience. In *Parallel Scientific Computing. Proceedings First International Workshop, PARA '94, Lyngby, Denmark, June 20-23, 1994.* Lecture Notes in Computer Science **879**, Springer-Verlag, Berlin, 169-176.

[4] Daydé M. J. and Duff I. S. (1995). Porting industrial codes and developing sparse linear solvers on parallel computers. *Computing Systems in Engineering,* **6**, 295-305.

[5] Drummond, L. A. (1995). Solution of general linear systems of equations using block Krylov based iterative methods on distributed computing environments. PhD Thesis. Report TH/PA/95/40, CERFACS, Toulouse.

[6] L'Excellent, J-Y. (1995). Utilisation de Préconditionneurs Elément-par-Elément pour la Résolution de Problèmes d'Optimisation de Grande Taille. Ph D Thesis, ENSEEIHT-IRIT.

6.2 Fortran 90 (J.K. Reid)

Although no longer a member of X3J3, the ANSI Fortran Standardization Committee, John Reid attended several meetings during the reporting period in the hope of getting exception handling into the language.

There were two attempts to provide such a feature during the development of Fortran 90. The first included tasking and was abandoned as too ambitious. The second involved a new construct, `ENABLE`, and was simpler, but was relegated to a 'Journal of Development' with the pressure to keep Fortran 90 small and with the need to devote resources to the development of the rest of the language.

John Reid played a leading role in developing a simplified `ENABLE` proposal during 1994 at the February and May meetings of X3J3 and the August meetings of WG5 and X3J3 in Edinburgh. He represented IFIP WG 2.5 (Numerical Software), which is strongly in favour of the provision of such facilities, at all these meetings. The final version was the subject of an X3J3 letter ballot. Unfortunately, following this ballot, X3J3 decided that to continue to work on it would put the whole schedule for Fortran 95 in jeopardy and therefore stopped.

This decision was endorsed by WG5, the ISO Fortran Standardization Committee, but at its April 1995 meeting in Tokyo it decided that handling floating-point exceptions was too important to leave until Fortran 2000. It therefore decided to establish a development body to create a 'Type 2 Technical Report'. The intention is to finalize this soon (to be ready for formal balloting by April 1996). It will permit vendors to implement the feature as an extension of Fortran 95, confident that it will be part of Fortran 2000, unless experience in their implementation and use demand changes. It is a kind of beta-test facility for a new language feature. John Reid agreed to act as project editor.

The development body was requested to consider both the `ENABLE` approach and the use of a set of intrinsic procedures to provide more basic support for exceptions. The `ENABLE` approach is summarized in [1] but the other approach is simpler to understand and provides some support for other features of the IEEE floating-point standard. In November 1995, X3J3 decided to take the latter route, again with John Reid playing a leading role.

John Reid and Mike Metcalf kept their book *Fortran 90 explained* up to date with interpretations by revisions in two reprintings over the period.

Good optimizing compilers are now available on all platforms, and we have access to compilers from Nag (site licence), EPC (SUN and IBM workstations), Cray Y-MP, IBM (RS/6000), Digital (Alpha), and Fujitsu (SUN).

The technical content of the next revision of Fortran, informally known as Fortran 95 was finalized in November 1995. It is a minor revision that incorporates items in the corrigenda and editorial improvements, and these new features:

- FORALL
- Nested WHERE
- Pure procedures
- Elemental procedures
- Pointer initialization
- Automatic component initialization
- Allocation status always defined
- User procedures in specifications
- CPU_TIME
- The following new obsolescent features: computed GO TO, statement functions, DATA among executables, assumed-length character functions, fixed source form, and character * declarations.

References

1. Reid, J.K. (1995). Exception handling in Fortran. ACM Fortran Forum, **14**, 9-14.

7 Computing and Harwell Subroutine Library

7.1 The computing environment within the Group (N.I.M. Gould)

Our policy of upgrading the group's workstations has continued over the past two years. The main change has been that Nick's old IBM RISC Systems/6000 320H has been put out to grass at home and replaced by a newer model RISC Systems/6000 3BT, a machine which is theoretically capable of up to 300 Megaflops. The rest of the group continues to favour SUN equipment, and an order was placed at the end of 1994 for a SUN Ultra Sparc 1 to replace Iain's aging Sparc 10/30. We continue to maintain three Sparc 1s for use by our short and medium-term visitors, one of which has now been upgraded to run the Solaris operating system. A further order was recently placed for an IBM Thinkpad 701 portable computer which is intended for use by group members on their frequent travels.

Our system has been generally stable over the past two years. We continue to rely on the CISD Unix system support team for major system administration, although group members have found it more convenient to get their hands dirty for simple tasks. The group has developed a series of WWW pages describing its activities. This has resulted in much wider publicity for the group, and made it easier for external users to access our technical reports and publicly-available software. In addition, we maintain pages of links to other relevant numerical analysis information.

We continue to benefit from other public CISD machines, in particular the DEC Alpha 3000 and HP-9000 farms, the CRAY Y-MP8I and the new DEC 8400 six processor system. Regular system backups are taken via the CISD IBM virtual tape reader. We now have access to six Fortran 90 compilers, some on our own machines and some on other CISD machines. This has enabled our gradual transformation from Fortran 77 to 90 to proceed, and in some cases we have found these compilers uncovered previously hidden errors in our existing 77 codes.

7.2 Harwell Subroutine Library

The Group continues to act as consultants for the Harwell Subroutine Library. Our collaboration with Harwell was marred by three changes of principal contact person over the period, but nevertheless Release 2 of the Harwell Sparse Matrix Library was completed in June 1995 and Release 12 of the main Library was made in December 1995.

The second release of the Harwell Sparse Matrix Library now contains 45 packages, including many that were new in Release 11 of the main Library. It is over twice the size of the first release (1988).

We employed Mike Hopper as a consultant to help with the preparation of Release 2 of the Harwell Sparse Matrix Library and we greatly appreciated his assistance in designing a suitable Unix file system for this library, applying extra checks to the codes and sorting out the problems that he found. We also employed him to apply his Unix tools to the new routines of the Release 12 of the main Library, and he provided us with a new release of the computer typesetting system TSSD.

7.3 Release 12 of the Harwell Subroutine Library

Release 12 of the Library was made in December 1995. It contains a few Fortran 90 modules, with a very similar style for the documentation as presently used for Fortran 77 packages. The names take the form `HSL_lldd` where `l` stands for a letter and `d` stands for a digit. The library contains the following significant new routines:

`HSL_AD01` This Fortran 90 package provides automatic differentiation facilities for variables specified by Fortran code.

`DC05` Solves a system of ordinary differential equations or algebraic differential equations of index no higher than one.

`DC06` Solves the initial value problem for a system of $\frac{1}{2}$ordinary differential equations or differential algebraic $\frac{1}{2}$equations of index no higher than one.

`DC07` Front end to `DC06`.

`EB13` Given a real unsymmetric matrix $\mathbf{A}=\{a_{ij}\}$, this routine uses Arnoldi based methods to calculate the r eigenvalues λ_i, $i=1,2,...,r$, that are of largest absolute value, or are right-most, or are of largest imaginary parts.

`HSL_FA04` Fortran 90 version of the random-number generator `FA04`.

`MA38` Solves a sparse unsymmetric system of linear equations using an unsymmetric multifrontal variant of Gaussian elimination.

`MA41` To solve a sparse unsymmetric system of linear equations on a shared-memory multiprocessor, using a parallel direct method based on a sparse multifrontal variant of Gaussian elimination.

`HSL_MA42` Solves one or more sets of sparse linear equations by the frontal method, optionally using direct access files for the matrix factors. It is a Fortran 90 module based on `MA42` and offers a much more friendly interface.

`MA46` Solves one or more set of sparse unsymmetric linear equations $\mathbf{AX}=\mathbf{B}$ from finite-element applications, using a multifrontal elimination scheme. The matrix $\mathbf{A}$ is input by elements.

`MA51` For use in conjunction with the `MA48` and `MA50` packages for solving sparse unsymmetric sets of linear equations. It identifies which equations are ignored when solving $\mathbf{Ax}=\mathbf{b}$ and which solution components are always set to zero.

`MA52` This collection of subroutines, when used in conjunction with the `MA42` package, solves finite-element equations using a multiple front algorithm.

`MC36` To read a sparse matrix, coded in a Harwell-Boeing format with possible right-hand sides.

The subroutine reads assembled as well as unassembled matrices, and returns them in a column oriented compressed sparse format.

`MC37` Given a sparse symmetric matrix **A**, this subroutine computes a set of element matrices that, if assembled, would yield the same matrix. Note that this set of elements is not unique.

`MC38` Given a sparse matrix held in a compressed column oriented format, this subroutine generates the transpose of the matrix, holding it in compressed column format.

`MC44` Given the structure of an unassembled finite-element matrix, this subroutine groups the variables into supervariables and optionally generates either the element connectivity graph or the supervariable connectivity graph.

`MC47` Given a representation of the nonzero pattern of a symmetric matrix, this subroutine performs an approximate minimum degree ordering.

`MC52` To write a sparse matrix in Harwell-Boeing format with possible right-hand sides.

`MC53` This subroutine generates an ordering for finite-element matrices within a subdomain that is efficient when subsequently used with a multiple front algorithm.

`MF36` To read a complex sparse matrix, coded in a Harwell-Boeing format with possible right-hand sides. The subroutine reads assembled as well as unassembled matrices, and returns them in a column oriented compressed sparse format.

`MI01` Uses the Conjugate Gradient method to solve a symmetric positive-definite linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning.

`MI03` Uses the CGS (Conjugate Gradient Squared) method to solve an unsymmetric linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning.

`MI04` This routine uses the Generalized Minimal Residual method with restarts every m iterations, GMRES(m), to solve an unsymmetric linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning.

`MI05` This routine uses the BiCG (BiConjugate Gradient) method to solve an unsymmetric linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning.

`MI06` This routine uses the BiCGStab (BiConjugate Gradient Stabilized) method to solve an unsymmetric linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning.

`MI11` This routine forms an incomplete **LU** factorization of a sparse unsymmetric matrix **A**.

`MI12` This routine finds an approximate inverse **M** of a sparse unsymmetric matrix **A** by attempting to minimize the difference between **AM** and the identity matrix in the Frobenius norm.

`HSL_VH01` This package uses the genetic algorithm to find the smallest value of an objective function of n binary (zero-one) variables.

`HSL_ZA03` This package provides kind values for 1- and 2-byte Fortran 90 `LOGICAL` variables. If a particular kind is not supported, a kind offering at least as much storage is substituted.

7.4 New Library subroutines

The following routines were added to the Library during this period. Although we now issue well-defined Releases to the Library, we continue to develop it internally on an incremental basis and may offer single routines commercially before they are included in a specific Release. The following routines are all included in Release 12 of HSL.

`HSL_AD01` (J. K. Reid and D. Cowey)

This Fortran 90 package provides automatic differentiation facilities for variables specified by Fortran code. Each independent variable and each variable whose value depends on the value of any independent variable must be declared to be of type `AD01_REAL` instead of default `REAL`. Note that Fortran variables of type default `REAL` and default `INTEGER` may enter the computation provided their values do not vary with the values of the independent variables. Both the backward and the forward method are available.

First and second derivatives are available with both the forward and backward methods. Derivatives of any order are available with the forward method. They are stored in a hyper-triangular format so that only one copy of identical derivatives is held.

It is possible to store the current state of the module data, perform a subsidiary calculation, and then return to the main calculation. For example, this mode may be used to compute the local derivatives of a unary function of the main calculation.

A record is kept of the number of occurrences of errors. By default, execution continues after an error in a motoring mode where each operation is executed as an immediate return. Alternatively, an immediate stop may be requested.

`DC05` (A. H. Harker)

This is a package for solving a system of ordinary differential equations or differential algebraic equations

$$y_i' = f_i(y_1, y_2, ..., y_n, x) \qquad i=1,2,...,n.$$

The solution starts from given initial values y_i^0 at $x = x^0$. It is numerically similar to `DC03`; but it presents a different user interface, possibly less convenient for ordinary use, intended for special-purpose packages. Function evaluation and linear algebra is done in the calling program by using ‘reverse communication’. Also, it can solve a wider range of problems, including implicit differential equations. The method is especially efficient on stiff problems.

`DC06` (A. H. Harker)

This solves a system of ordinary differential equations or differential algebraic equations of index less than or equal to one. There is a system of n variables, y_i where $i=1,...,n$, which satisfy the equations

$$y_i' = f_i(y_1, y_2, ..., y_n, x), \qquad i=1,2,...,n,$$

where $y_i' = dy_i/dx$ is the x derivative of variable y_i. DC06 solves the initial value problem for these equations; that is given the initial values of the variables, $y_i(x_0)$, at some point, x_0, it advances the solution forward in x. The user must supply a subroutine to evaluate the derivative function f_j and may supply subroutines to evaluate the Jacobian and K-Jacobian required during the solution.

DC07 (A. H. Harker)

This solves the initial value problem for a full system of explicit ordinary differential equations or a system of differential algebraic equations of index less than or equal to one. That is, equations of the form

$$y_i' = f_i(y_1, y_2, ..., y_n, x) \qquad i=1,2,...,n.$$

where $y_i' = dy_i/dx$ is the x derivative of variable y_i. The Jacobian of the right hand side is assumed to be full.

DC07 solves the initial value problem; that is given the initial values of the variables, $y_i(x_0)$, at some point, x_0, it advances the solution to equations forward in x.

HSL_FA04 (N. I. M. Gould and J. K. Reid)

This package generates uniformly distributed pseudo-random numbers. Random reals are generated in the range $0 < \xi < 1$ or the range $-1 < \eta < 1$ and random integers in the range $1 \leq k \leq N$ where N is specified by the user.

A multiplicative congruent method is used where a 31 bit generator word g is maintained. On each call to a procedure of the package, g_{n+1} is updated to $7^5 g_n \text{mod}(2^{31}-1)$; the initial value of g is $2^{16}-1$. Depending upon the type of random number required the following are computed $\xi = g_{n+1}/(2^{31}-1)$; $\eta = 2\xi - 1$ or $k = \text{int.part}\{\xi N\}+1$.

The package also provides the facility for saving the current value of the generator word and for restarting with any specified value.

MA38 (T. A. Davis and I. S. Duff)

This package solves a sparse unsymmetric system of n linear equations in n unknowns using an unsymmetric multifrontal variant of Gaussian elimination. There are facilities for choosing a good pivot order, factorizing another matrix with a nonzero pattern identical to that of a previously factorized matrix, and solving a system of equations using the factorized matrix. An option exists for solving triangular systems using the factors from the Gaussian elimination.

`MA41` (P. A. Amestoy and I. S. Duff)

To solve a sparse unsymmetric system of linear equations on a shared-memory multiprocessor. Given an unsymmetric square sparse matrix **A** of order *n* and an *n*-vector **b**, this subroutine solves the system $\mathbf{Ax}=\mathbf{b}$ or $\mathbf{A}^T\mathbf{x}=\mathbf{b}$.

The method used is a parallel direct method based on a sparse multifrontal variant of Gaussian elimination. An initial ordering for the pivotal sequence is chosen using the pattern of the matrix $\mathbf{A} + \mathbf{A}^T$ and is later modified for reasons of numerical stability. Thus this code performs best on matrices whose pattern is symmetric, or nearly so. For symmetric sparse matrices or for very unsymmetric and very sparse matrices, other software might be more appropriate (for example, `MA47` or `MA48`).

There is a version of the code for uniprocessors which is in Fortran 77. The parallel versions are machine dependent but only require simple features like starting parallel tasks and locks. In principle, a code can be supplied for any shared memory parallel machine but the only two platforms on which this has been tested extensively are the CRAY Y-MP and the ALLIANT FX/80.

`HSL_MA42` (J. A. Scott)

The module `HSL_MA42` is a Fortran 90 interface to the `MA42` package that shields the user from most of the complexities in the calling sequences to that package.

`MA46` (A.C. Damhaug and J.K. Reid)

This solves one or more set of sparse unsymmetric linear equations $\mathbf{AX}=\mathbf{B}$ from finite-element applications, using a multifrontal elimination scheme. The matrix **A** must be input by elements and be of the form

$$\mathbf{A}=\sum_{k=1}^{m}\mathbf{A}^{(k)}$$

where $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns that correspond to variables of the nodes of the *k*-th element. Optionally, the user may pass an additional matrix $\mathbf{A}_d$ of coefficients for the diagonal. **A** is then of the form

$$\mathbf{A}=\sum_{k=1}^{m}\mathbf{A}^{(k)}+\mathbf{A}_d$$

The right-hand side **B** should be assembled through the summation

$$\mathbf{B} = \sum_{k=1}^{m} \mathbf{B}^{(k)},$$

before calling the solution routine.

`MA51` (J.K. Reid)

This is for use in conjunction with the `MA48` and `MA50` packages for solving sparse unsymmetric sets of linear equations. It identifies which equations are ignored when solving $\mathbf{Ax} = \mathbf{b}$ and which solution components are always set to zero. The roles are reversed for $\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$. There are such equations and/or components in the singular or rectangular case. Note that, if $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$ is not consistent, there may be large residuals for the equations that are ignored.

`MA52` (J. A. Scott)

This collection of subroutines, when used in conjunction with the `MA42` package, solves finite-element equations using a multiple front algorithm. It is assumed that the underlying finite-element mesh has been partitioned into (non-overlapping) subdomains. In the multiple front algorithm, a frontal method is applied to each subdomain separately. This can be done in parallel. Using multiple fronts can also reduce the amount of work required.

At the end of the assembly and elimination processes for the subdomains, for each subdomain i there remains a frontal matrix $\mathbf{F}_i$ and a corresponding right-hand side vector $\mathbf{c}_i$ satisfying

$$\mathbf{F}_i \mathbf{y}_i = \mathbf{c}_i. \tag{1}$$

These equations may be assembled to give a system of the form

$$\mathbf{Fy} = \mathbf{c}. \tag{2}$$

By treating each of the subdomain frontal matrices $\mathbf{F}_i$ as an elemental matrix, (2) may be solved by a frontal method. Once (2) has been solved, back-substitution on the subdomains completes the solution.

`MA52` provides routines for generating lists of variables lying on the subdomain interfaces, for preserving the partial factorization of a matrix when the sequence of calls to `MA42B/BD` is incomplete, and for performing forward or back-substitution on a subdomain.

`MA52` uses reverse communication.

The use of Harwell Subroutine Library routine `MC53` to obtain an efficient element ordering in each subdomain is recommended before `MA52` is used.

For further details of multiple fronts, see Duff, I. S. and Scott, J. A. (1994), *The use of multiple fronts in Gaussian elimination.* Rutherford Appleton Laboratory Report RAL-94-040.

`MC36` (I. S. Duff)

To read a sparse matrix, coded in a Harwell-Boeing format with possible right-hand sides. The subroutine reads assembled as well as unassembled matrices, and returns them in a column oriented compressed sparse format. `MF36` must be used if the matrix is complex.

`MC37` (I. S. Duff)

Given a sparse symmetric matrix **A**, this subroutine computes a set of element matrices that, if assembled, would yield the same matrix. Note that this set of elements is not unique. The matrix can be input by the user either in compressed column format (column pointer/row index scheme) or by row and column index pairs in any order.

`MC38` (I. S. Duff)

Given a sparse matrix held in a compressed column oriented format, this subroutine generates the transpose of the matrix, holding it in compressed column format. It can also be viewed as a conversion between a column oriented scheme and a row oriented one. This subroutine differs from `MC46` inasmuch as it preserves the input data and should be faster, particularly on vector machines. However, it does require storage for both the matrix and its transpose.

`MC44` (J. K. Reid and J. A. Scott)

Given the structure of an unassembled finite-element matrix, this subroutine groups the variables into supervariables and optionally generates either the element connectivity graph or the supervariable connectivity graph.

A supervariable is a collection of one or more variables, such that each variable belongs to the same set of finite elements. In the supervariable connectivity graph, the nodes are the supervariables and the edges are constructed by making the supervariables of each finite element pairwise adjacent. The supervariable connectivity graph, together with the number of variables in each supervariable, provide a compact representation of the variable connectivity graph. In the element connectivity graph, the nodes are the elements and the edges are constructed by defining two elements to be adjacent whenever they have one or more variables in common.

`MC47` (P. R. Amestoy, T. A. Davis, and I. S. Duff)

Given a representation of the nonzero pattern of a symmetric matrix, **A**, this subroutine performs an approximate minimum degree ordering to compute a pivot order so that the number of nonzeros in the Cholesky factors of **A** is kept low. At each step, the pivot selected is the one that minimizes an easily-computed upper-bound on the (external) degree. A permutation corresponding to this ordering is returned, together with information to assist the subsequent numerical factorization of the matrix.

The code is typically faster than other minimum degree algorithms and produces comparable results to other minimum external degree algorithms in terms of fill-in and the number of floating-point operations needed to compute the factors.

The version of the code described here is based on work done by Amestoy, Davis and Duff (An approximate minimum degree ordering algorithm, *Report TR-94-039, Computer and Information Sciences Department, University of Florida*), and is a symmetric analogue of the ordering used in the code of Davis and Duff (An unsymmetric-pattern multifrontal method for sparse LU factorization. *Report RAL 93-036, Rutherford Appleton Laboratory*.)

`MC52` (I. S. Duff)

To write a sparse matrix in Harwell-Boeing format with possible right-hand sides. The matrix can be input as an assembled matrix in either column-oriented or coordinate form, or as an unassembled finite-element matrix. The right-hand sides must be in full format.

`MC53` (J. A. Scott)

This subroutine generates an ordering for finite-element matrices within a subdomain that is efficient when subsequently used with a multiple front algorithm. In a multiple front algorithm, the finite-element domain is partitioned into a number of subdomains and a frontal decomposition is performed on each subdomain separately. The storage required by a multiple front algorithm and the time taken to run it are dependent upon the order in which the elements in each subdomain are input; the variation in the performance of different element orderings can be significant. The ordering obtained by `MC53` is designed to reduce the maximum and root mean-squared frontsizes and to reduce the floating-point operation count for the frontal solver on the subdomain. If *nelt* is the number of elements in the subdomain and $fsize_i$ is the number of variables in the front after the assembly of the *i*th element, the maximum frontsize *fmax* in the subdomain is defined to be

$$fmax = \max_{1 \le i \le nelt} \{fsize_i\}$$

and the root mean-squared frontsize *frms* in the subdomain is defined to be

$$frms = \sqrt{\frac{1}{nelt}\sum_{i=1}^{nelt} fsize_i^2}\,.$$

The user is required to use reverse communication to supply a list of the variables belonging to each element in the subdomain one at a time followed by a list of the variables lying on the subdomain interface.

MF36 (I. S. Duff)

To read in a complex sparse matrix, coded in a Harwell-Boeing format with possible right-hand sides. The subroutine reads assembled as well as unassembled matrices, and returns them in a column oriented compressed sparse format. If the matrix being read is real MC36A/AD should be used.

MI01 (N. I. M. Gould and J. A. Scott)

This routine uses the Conjugate Gradient method to solve the $n \times n$ symmetric positive definite linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning. If $\mathbf{PP}^{\mathrm{T}}$ is the preconditioning matrix, the routine actually solves the preconditioned system

$$\overline{\mathbf{Ax}}=\overline{\mathbf{b}},$$

with $\overline{\mathbf{A}}=\mathbf{PAP}^{T}$ and $\overline{\mathbf{b}}=\mathbf{Pb}$ and recovers the solution $\mathbf{x}=\mathbf{P}^{\mathrm{T}}\overline{\mathbf{x}}$. Reverse communication is used for preconditioning operations and matrix-vector products of the form $\mathbf{Az}$.

MI03 (N. I. M. Gould and J. A. Scott)

This routine uses the CGS (Conjugate Gradient Squared) method to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning. If $\mathbf{P}_{\mathrm{L}}$, $\mathbf{P}_{\mathrm{R}}$ are the preconditioning matrices, the routine actually solves the preconditioned system

$$\overline{\mathbf{Ax}}=\overline{\mathbf{b}},$$

with $\overline{\mathbf{A}}=\mathbf{P}_{\mathrm{L}}\mathbf{AP}_{\mathrm{R}}$ and $\overline{\mathbf{b}}=\mathbf{P}_{\mathrm{L}}\mathbf{b}$ and recovers the solution $\mathbf{x}=\mathbf{P}_{\mathrm{R}}\overline{\mathbf{x}}$. If $\mathbf{P}_{\mathrm{L}}=\mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_{\mathrm{R}}=\mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations and matrix-vector products of the form $\mathbf{Az}$.

MI04 (N. I. M. Gould and J. A. Scott)

This routine uses the Generalized Minimal Residual method with restarts every m iterations, GMRES(m), to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax}=\mathbf{b}$, optionally using preconditioning. If $\mathbf{P}_{\mathrm{L}}$, $\mathbf{P}_{\mathrm{R}}$ are left and right preconditioning matrices, the routine actually solves the preconditioned system

$$\overline{\mathbf{Ax}}=\overline{\mathbf{b}},$$

with $\overline{\mathbf{A}}=\mathbf{P}_{\mathrm{L}}\mathbf{AP}_{\mathrm{R}}$ and $\overline{\mathbf{b}}=\mathbf{P}_{\mathrm{L}}\mathbf{b}$. The solution may be recovered as $\mathbf{x}=\mathbf{P}_{\mathrm{R}}\overline{\mathbf{x}}$. If $\mathbf{P}_{\mathrm{L}}=\mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_{\mathrm{R}}=\mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations and matrix-vector products of the form $\mathbf{Az}$.

`MI05` (N. I. M. Gould and J. A. Scott)

This routine uses the BiCG (BiConjugate Gradient) method to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If $\mathbf{P}_\mathrm{L}$, $\mathbf{P}_\mathrm{R}$ are the preconditioning matrices, the routine actually solves the preconditioned system

$$\overline{\mathbf{Ax}} = \overline{\mathbf{b}},$$

with $\overline{\mathbf{A}} = \mathbf{P}_\mathrm{L}\mathbf{AP}_\mathrm{R}$ and $\overline{\mathbf{b}} = \mathbf{P}_\mathrm{L}\mathbf{b}$ and recovers the solution $\mathbf{x} = \mathbf{P}_\mathrm{R}\overline{\mathbf{x}}$. If $\mathbf{P}_\mathrm{L} = \mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_\mathrm{R} = \mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations $\mathbf{Pz}$ and $\mathbf{P}^\mathrm{T}\mathbf{z}$, where $\mathbf{P}$=$\mathbf{P}_\mathrm{L}\mathbf{P}_\mathrm{R}$, and for matrix-vector products of the form $\mathbf{Az}$ and $\mathbf{A}^\mathrm{T}\mathbf{z}$.

`MI06` (N. I. M. Gould and J. A. Scott)

This routine uses the BiCGStab (BiConjugate Gradient Stabilized) method to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If $\mathbf{P}_\mathrm{L}$, $\mathbf{P}_\mathrm{R}$ are the preconditioning matrices, the routine actually solves the preconditioned system

$$\overline{\mathbf{Ax}} = \overline{\mathbf{b}},$$

with $\overline{\mathbf{A}} = \mathbf{P}_\mathrm{L}\mathbf{AP}_\mathrm{R}$ and $\overline{\mathbf{b}} = \mathbf{P}_\mathrm{L}\mathbf{b}$ and recovers the solution $\mathbf{x} = \mathbf{P}_\mathrm{R}\overline{\mathbf{x}}$. If $\mathbf{P}_\mathrm{L} = \mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_R = \mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations and matrix-vector products of the form $\mathbf{Az}$.

`MI11` (N. I. M. Gould and J. A. Scott)

This routine forms an incomplete $\mathbf{LU}$ factorization of an $n \times n$ sparse unsymmetric matrix $\mathbf{A}$. No fill-in is allowed. The entries of $\mathbf{A}$ are stored by rows. If $\mathbf{A}$ has zeros on the diagonal, the routine first finds a row permutation $\mathbf{Q}$ which makes the matrix have nonzeros on the diagonal. The incomplete $\mathbf{LU}$ factorization of the permuted matrix $\mathbf{QA}$ is then formed. $\mathbf{L}$ is lower triangular and $\mathbf{U}$ is unit upper triangular. The incomplete factorization may be used as a preconditioner when solving the linear system $\mathbf{Ax} = \mathbf{b}$. A second entry performs the preconditioning operations

$$\mathbf{y} = \mathbf{Pz} \quad \text{and} \quad \mathbf{y} = \mathbf{P}^\mathrm{T}\mathbf{z},$$

where $\mathbf{P}^{-1}\mathbf{Q}$ is the preconditioner.

`MI12` (N. I. M. Gould and J. A. Scott)

This routine finds an approximate inverse $\mathbf{M}$ of an $n \times n$ sparse unsymmetric matrix $\mathbf{A}$ by attempting to minimize the difference between $\mathbf{AM}$ and the identity matrix in the Frobenius norm. The process may be improved by first performing a block triangularization of $\mathbf{A}$ and then finding approximate inverses to the resulting diagonal blocks.

A second entry allows the user to form the matrix-vector products

$$\mathbf{y} = \mathbf{M}\mathbf{z} \quad \text{and} \quad \mathbf{y} = \mathbf{M}^{\mathrm{T}}\mathbf{z}.$$

The principal use of such an approximate inverse is likely to be in preconditioning iterative methods for solving the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

`HSL_VH01` (N. I. M. Gould)

This package uses the genetic algorithm to search for a small value of an objective function of n binary (zero-one) variables. Each *string* of binary variables is stored in a logical array. A *population* of p such strings is maintained along with their associated objective function values. The population evolves in a sequence of iterations. The best features of the population at iteration k are passed to the population at iteration k+1 by means of *mutation* and *crossover*.

The package obtains function values by reverse communication. The user is periodically required to check for termination.

`HSL_ZA03` (N. I. M. Gould)

This package provides kind values for 1- and 2-byte Fortran 90 `LOGICAL` variables. If a particular kind is not supported, a kind offering at least as much storage is substituted.

8 Seminars

27 January 1994	N. I. M. Gould (Rutherford) Optimization research in the parallel algorithms team at CERFACS
3 February 1994	P. Graves-Morris (Bradford) Is SOR being revived?
11 May 1994	R. da Cunha (Kent) Designing a portable package for parallel architectures
9 June 1994	C-H. Lai (Greenwich) Large-scale computing and domain decomposition methods
26 August 1994	J. Bunch (California) Bounding the subspaces obtained by rank revealing two-sided orthogonal decompositions
3 November 1994	Z. Jia (Bielefeld) A refined strategy in orthogonal projection methods for the unsymmetric eigenproblem
24 November 1994	A. Ramage (Strathclyde) Preconditioned conjugate gradients for irregular finite element grids
2 February 1995	E. Hinton (Swansea) Structural optimisation – numerical issues
9 February 1995	A. Erisman (Boeing Computer Services) Technology change and what we do with it
2 March 1995	A. Wathen (Bristol) The iterative solution of discrete saddle-point problems
4 May 1995	J. Du Croz (NAG Ltd) The design of the NAG Fortran 90 Library
12 May 1995	J. Mason (Huddersfield) Parallel and structured data approximations
23 November 1995	D. Higham (Dundee) Time-stepping analysis using concepts from numerical linear algebra
7 December 1995	P. Sweby (Reading) Dynamics of discretisation

9 Reports issued in 1994-95

We give a full listing of Rutherford Reports during the period of this Progress Report. The other report listings, from organizations with which we collaborate, only include reports not already included as RAL Reports.

Rutherford Reports

RAL 94-018	Porting industrial codes and developing sparse linear solvers on parallel computers. M.J. Daydé and I.S. Duff.
RAL 94-040	The use of multiple fronts in Gaussian elimination. I.S. Duff and J.A. Scott.
RAL 94-062	Numerical Analysis Group – Progress Report. January 1991 – December 1993. I.S. Duff (Editor).
RAL 94-069	On iterated-subspace minimization methods for nonlinear optimization. A.R. Conn, N.I.M. Gould and Ph.L. Toint.
RAL 95-001	MA47, A Fortran code for direct solution of indefinite sparse symmetric linear systems. I.S. Duff and J.K. Reid.
RAL 95-009	Convergence properties of an augmented Lagrangian algorithm for optimization with a combination of general equality and linear constraints. A.R. Conn, N.I.M. Gould, A. Sartenaer, and Ph.L. Toint.
RAL 95-010	On the use of element-by-element preconditioners to solve large-scale partially separable optimization problems. M.J. Daydé, J.Y. l'Excellent, and N.I.M. Gould.
RAL-TR-95-026	On approximate-inverse preconditioners. N.I.M. Gould and J.A. Scott.
RAL-TR-95-029	Element resequencing for use with a multiple front algorithm. J.A. Scott.
RAL-TR-95-037	Constructing appropriate models for large-scale, linearly-constrained, nonconvex, nonlinear, optimization algorithms. N.I.M. Gould.
RAL-TR-95-039	The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. I.S. Duff and J.K. Reid.
RAL-TR-95-040	Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. I.S. Duff and J.K. Reid.
RAL-TR-95-049	A set of Level 3 Basic Linear Algebra Subprograms for sparse matrices. I.S. Duff, M. Marrone, G. Radicati, and C. Vittoli.

AEA Technology Reports

HARWELL SUBROUTINE LIBRARY. A Catalogue of Subroutines (Release 12). October 1995.

HARWELL SUBROUTINE LIBRARY. Release 12. Specifications. Volume 1 and Volume 2. December 1995.

CERFACS Reports

TR/PA/94/10 The CERFACS Experience. M.J. Daydé and I.S. Duff.

TR/PA/94/15 A parallel scheduler for block iterative solvers in heterogeneous computing environments. M. Arioli, A. Drummond, I.S. Duff, and D. Ruiz.

TR/PA/95/09 An approximate minimum degree ordering algorithm. P. Amestoy, T.A. Davis, and I.S. Duff.

TR/PA/95/26 Linear algebra kernels for parallel domain decomposition methods. L. Carvalho, I. Duff and L. Giraud.

ENSEEIHT-IRIT Reports

RT/APO/95/1 Solution of structured systems of linear equations using element-by-element preconditioners. M.J. Daydé, J.Y. l’Excellent, and N.I.M. Gould.

University of Florida Reports

TR-95-020 A combined unifrontal/multifrontal method for unsymmetric sparse matrices. T.A. Davis and I.S. Duff.

10 External Publications in 1994-1995

Amestoy P. R., Daydé M. J., Duff I. S., and Morère P. (1995). Linear algebra calculations on a virtual shared memory computer. *Int. J. High Speed Computing* **7**, 21-43.

Arioli, M., Duff, I. S., Ruiz, D., and Sadkane, M. (1995). Block Lanczos techniques for accelerating the Block Cimmino method. *SIAM J. Sci. Comput.* **16**, 1478-1511.

Arioli, M., Drummond, A., Duff, I. S., and Ruiz, D. (1995). A parallel scheduler for block iterative solvers in heterogeneous computing environments. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing.* Edited by David H Bailey et al. SIAM, Philadelphia., 460-465.

Arioli, M., Drummond, A., Duff, I. S., and Ruiz, D. (1995). Parallel block iterative solvers for heterogeneous computing environments. In *Algorithms and Parallel VLSI Architectures III* Edited by M. Moonen and F. Catthoor. Elsevier, Amsterdam, 97-108.

Bongartz, I., Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1995). CUTE: Constrained and Unconstrained Testing Environment, *ACM Trans. Math. Softw.* **21**, 123-160.

Conn, A. R., Gould, Lescrenier, M., N. I. M., and Toint, Ph. L. (1994). Performance of a multifrontal scheme for partially separable optimization, in *Advances in numerical partial differential equations and optimization, Proceedings of the sixth Mexico-United States Workshop* (S. Gomez and J.P. Hennart and R.A. Tapia, eds) Kluwer Academic Publishers.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1994). Improving the decomposition of partially separable functions in the context of large-scale optimization: a first approach, in *Large Scale Optimization: State of the Art* (W. W. Hager, D. W. Hearn and P.M. Pardalos, eds.) Kluwer Academic Publishers B.V., 82-94.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1994). Large-scale nonlinear constrained optimization: a current survey, in *Algorithms for continuous optimization: the state of the art* (E. Spedicato, editor). Kluwer Academic Publishers, Dordrecht, The Netherlands, 287-332.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1994). A note on using alternative second-order models for the subproblems arising in barrier function methods for minimization, *Numerische Math.* **68**, 17-33.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1994). A note on exploiting structure when using slack variables, *Math. Programming* **67**, 89-98.

Daydé, M. and Duff, I. S. (1994). The CERFACS Experience. In *Parallel Scientific Computing. Proceedings First International Workshop, PARA '94, Lyngby, Denmark, June 20-23, 1994.* Lecture Notes in Computer Science **879**, Springer-Verlag, Berlin, 169-176.

Daydé M. J. and Duff I. S. (1995). Porting industrial codes and developing sparse linear solvers on parallel computers. *Computing Systems in Engineering.* **6**, 295-305.

Daydé M. J., Duff I. S., and Petitet A. (1994). A parallel block implementation of Level 3 BLAS kernels for MIMD vector processors. *ACM Trans. Math. Softw.* **20**, 178-193.

Duff, I. S. (1994). The solution of augmented systems. In *Numerical Analysis 1993.* D.F. Griffiths and G.A. Watson (Editors). Pitman research Notes in Mathematical Series **303**. Longman, 40-55.

Duff, I. S. (1994). The solution of sparse equations on high performance computers. In *Matrix Analysis and Parallel Computing.* M. Natori and T. Nodera (Editors). Advances in Numerical Methods for Large Sparse Sets of Linear Equations **10**. Keio University, Japan, 1-13.

Duff, I. S. (1994). A review of frontal methods for solving linear systems. In *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra.* J.G. Lewis (editor). SIAM Press, Philadelphia, 135-139.

Duff, I. S. and Scott, J. A. (1994). The use of multiple fronts in Gaussian elimination. In *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra.* J.G. Lewis (editor). SIAM Press, Philadelphia, 567-571.

Metcalf, M. and Reid, J.K. (1995). Fortran 90 explained (reprints with corrections). Oxford University Press.

Reid, J.K. (1995). Fortran 90 shapes up to the future. *Scientific Computing* **5**, 16-22.

Reid, J.K. (1995). Exception handling in Fortran. ACM Fortran Forum, **14**, 9-15.

Scott, J. A. (1995). An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices. *ACM Trans. Math. Softw.*, **21**, 423-475.