RAL 94-062

NUMERICAL ANALYSIS GROUP

PROGRESS REPORT

January 1991 – December 1993

Edited by Iain S. Duff

Central Computing Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

June 1994.

CONTENTS

1 Introduction (I.S. Duff) …… 2

2 Sparse Matrix Research …… 6

2.1 The solution of sparse structured symmetric indefinite linear sets of equations (I.S. Duff and J.K. Reid)…… 6

2.2 The direct solution of sparse unsymmetric linear sets of equations (I.S. Duff and J.K. Reid) …… 7

2.3 Development of a new frontal solver (I. S. Duff and J. A. Scott) …… 10

2.4 A variable-band linear equation frontal solver (J. A. Scott) …… 12

2.5 The use of multiple fronts (I. S.Duff and J. A. Scott)…… 13

2.6 MUPS: a MUltifrontal Parallel Solver for sparse unsymmetric sets of linear equations (P.R. Amestoy and I.S. Duff)…… 14

2.7 Sparse matrix factorization on the BBN TC2000 (P.R. Amestoy and I.S. Duff)… 15

2.8 Unsymmetric multifrontal methods (T.A. Davis and I.S. Duff) …… 17

2.9 Unsymmetric multifrontal methods for finite-element problems (A.C. Damhaug, I.S. Duff, J.A. Scott, and J.K. Reid) …… 18

2.10 Automatic scaling of sparse symmetric matrices (J.K. Reid) …… 19

2.11 Block triangular form and **QR** decomposition (I.S. Duff and C. Puglisi)…… 20

2.12 Multifrontal **QR** factorization in a multiprocessor environment (P.R. Amestoy, I.S. Duff, and C. Puglisi) …… 22

2.13 Impact of full **QR** factorization on sparse multifrontal **QR** algorithm (I.S. Duff and C. Puglisi)…… 23

2.14 The solution of linear least squares problems using the augmented system formulation (I.S. Duff, N.I.M. Gould, and J.M. Patrício) …… 25

2.15 Solution of large sparse unsymmetric linear systems with a block iterative method in a multiprocessor environment. (M. Arioli, I.S. Duff, J. Noailles, D. Ruiz, and M. Sadkane) …… 26

2.16 Stopping criteria for iterative methods (M. Arioli, I.S. Duff, and D. Ruiz) …… 28

2.17 Computing eigenvalues of large unsymmetric matrices (J. A. Scott) …… 28

2.18 Sparse matrix test problems (I.S. Duff) …… 30

2.19 Sparse BLAS (I.S. Duff, M. Marrone, G. Radicati, and C. Vittoli) …… 30

3 Research in Optimization 32
3.1 LANCELOT (A.R. Conn, N.I.M. Gould, and Ph.L. Toint) 32
3.2 Exploiting the structure of partially separable functions (A.R. Conn, M. Daydé, J.–Y. L'Excellent, N.I.M. Gould, and Ph.L. Toint) 33
3.3 Computing search directions for large-scale linearly-constrained nonlinear optimization calculations. (M. Arioli, T.F. Chan, I.S. Duff, N.I.M. Gould, and J.K. Reid) 36
3.4 Augmented Lagrangian and barrier function methods for large-scale nonlinear optimization (A.R. Conn, N.I.M. Gould, A. Sartenaer, and Ph.L. Toint) 37
3.5 The testing of optimization software (I. Bongartz, A.R. Conn, N.I.M. Gould, and Ph.L. Toint)..... 40

4 Applied Mathematics 43
4.1 Automatic differentiation in Fortran 90 (J.K. Reid) 43

5 Research in Numerical Linear Algebra 45
5.1 Computational kernels on vector and parallel machines and on RISC architectures (J.L. Charles, M.J. Daydé, I.S. Duff, P. Morère, and A. Petitet) 45
5.2 Efficient implementation of linear algebra codes on distributed memory machines (M.J. Daydé and I.S. Duff)..... 47
5.3 Pipelining the factorization of full linear systems on multiprocessors (P.R. Amestoy, M.J. Daydé, I.S. Duff, and T. Omnès) 48

6 Miscellaneous Activities 50
6.1 CERFACS (I.S. Duff and N.I.M. Gould) 50
6.2 Performance analysis of parallel computers on a benchmark of application software from Aérospatiale (M.J. Daydé, I.S. Duff, L. Giraud, and J-Y L'Excellent) 51
6.3 Fortran 90 (J.K. Reid)..... 52
6.4 IFIP Working Group 2.5 (J.K. Reid) 53

7 Computing and Harwell Subroutine Library 55
7.1 The computing environment within the Group (N.I.M. Gould) 55
7.2 Release 11 of the Harwell Subroutine Library (I.S. Duff, N.I.M. Gould, M.J. Hopper, J.A. Scott, and J.K. Reid) 56
7.3 Release 12 of the Harwell Subroutine Library (J.K. Reid) 57
7.4 New Library subroutines 57

8 Seminars 62

9 Reports issued in 1991-93 64

10 External Publications in 1991-1993..... 67

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

Christian Damhaug (Visitor)
Solution of finite-element equations using multifrontal methods.

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

Rachel Johnson (Vacation Student)
Mathematical algorithms and software.

1 Introduction (I.S. Duff)

This report covers the period from January 1991 to December 1993 and describes work performed by the Numerical Analysis Group within the Central Computing Department at the Atlas Centre in 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 overview and background for our research and to provide additional information on activities that are not appropriate for the detailed reports.

A major focus of the work of the Group continues to be the technical support and development of the Harwell Subroutine Library (HSL). We have had a very good relationship with Tony Harker and Libby Thick of the Theoretical Studies Department at Harwell who have handled all matters concerning marketing and sales, in addition to coordinating and printing Library documentation and sometimes commenting on technical issues also. The most tangible result of this work was the issue of Release 11 of HSL in July 1993, discussed in Section 7.2. In this respect we were much aided by the diligence of Mike Hopper, whom we were delighted to welcome as a consultant employed to test existing library routines, to check if they meet the standards of modern mathematical software, and to make changes as necessary. Although we maintain contact with the Harwell CFD group of Ian Jones (formerly the Applied Mathematics Group of CSSD at Harwell), our other main contractual connection with Harwell has been with Andrew Cliffe of Decommissioning and Waste Management, who has supported some of our work on the solution of finite-element equations by frontal schemes, most recently the work discussed in Section 2.5. This work has involved collaboration with scientists at EPCC in Edinburgh. We have also been involved in projects with Oxford Parallel for whom John, Iain and Nick are consultants, and we have participated in courses and workshops of ERCIM (European Research Consortium for Informatics and Mathematics).

Although we have had few long-term visitors, we have continued the Joint Computational Mathematics and Applications Seminar series with Oxford University Computing Laboratory and have welcomed many people to RAL through that programme. A list of the speakers and their topics can be found in Section 8. During Nick's sabbatical in France, we were able to host a visit from Christian Damhaug from Norway who worked on a code for the solution of unsymmetric finite-element problems (see Section 2.9). We also welcomed our first RAL vacation student in the summer of 1992. It was a pleasure having Rachel Johnson working with us for two months and we hope that the experience proved useful to her future work and study.

Three millstones were removed during this reporting period. Release A of LANCELOT in 1992 removed one from the neck of Nick, and John and Iain were similarly relieved to include the sparse matrix packages MA47 and MA48 in Release 11 of HSL, although a report on MA47 is still pending. These represented also major milestones of our software development effort

with which we should also include the further development of our sparse eigenvalue codes by Jennifer (EB13) and extensions of the frontal solution work by Jennifer and Iain (MA42 and MA43).

John was on the Fortran Standardization Committee X3J3 until after Fortran 90 was adopted in 1991 and since then has been very active in the ISO Fortran Committee and in promoting the Fortran 90 language through many courses and talks on the language at RAL, UK, and abroad. His book with Michael Metcalf is one of the definitive texts on Fortran 90 and has been reprinted three times in the reporting period, always with corrections to keep it fully up to date with interpretations of the Standard. Some of his activities in this area are discussed further in Section 6.3. John also continues as a member of IFIP Working Group 2.5 on numerical software and has played a leading role within the group on standards activities, particularly on Language Independent Arithmetic and Fortran. For the Fortran work of WG 2.5, his principal contribution was a proposal on exception handling. More information on this can be found in Section 6.4. John is a visiting Professor at RMCS Shrivenham and has given several lectures to graduate students there and at Reading University. He has presented invited talks at major meetings in Oxford, Rome, Washington, Cannes, Munich, Berkeley, Brunel, and Paris.

Nick's main research activity was the development of the mathematics, algorithms, and software connected with the LANCELOT project for large-scale nonlinearly constrained optimization, in collaboration with Philippe Toint from Namur and Andy Conn from Yorktown Heights, and involving several other researchers as indicated in Section 3 of this report. Another major piece of work concerned the creation of a standardized test bed of nonlinear problems, the CUTE test set discussed in Section 3.5. Nick spent the 1993 calendar year as a visiting senior scientist in the Parallel Algorithm Project at CERFACS in Toulouse where he interacted with scientists both at CERFACS and at ENSEEIHT, one of the universities in Toulouse. Although his French was greatly improved by the experience, he surprisingly remained a vegetarian. Nick was an invited speaker at the Biennial Conference on Numerical Analysis in Dundee, and the Workshop on Large-Scale Optimization in Coimbra, Portugal, in 1991, at the Fourth SIAM Conference on Numerical Optimization in Chicago in 1992, and at ORBEL 7 in Brussels, the Leslie Fox memorial conference in Oxford, and the Fourth Stockholm Optimization days in 1993. In addition, he contributed talks at a number of other international meetings. Nick has also been involved as a joint supervisor of doctoral students from the University of Hertfordshire and ENSEEIHT. He has been awarded NATO and British Council travel grants to enable him to continue his international collaborations.

Jennifer has had a very productive three years. Indeed, the birth of her second child (Mark) in September 1991 and her continued part-time working, make her productivity at work even more impressive. We are delighted to record that this was recognized by her promotion to SSO in May 1991. She has developed a very powerful set of subroutines for computing selected eigenvalues and the corresponding eigenvectors of large sparse real unsymmetric

matrices. She has also continued to work on the development and extension of frontal solvers, becoming in the process one of the Group's experts on the use of PVM on networks of workstations and on macrotasking on the CRAY YMP. Jennifer was very involved with the preparation of codes and documentation for Release 11 of the Harwell Subroutine Library. She continues to coordinate our joint seminar series with Oxford. During the period of this report, she attended the Dundee Biennial meetings in 1991 and 1993, an ERCIM Workshop on Numerical Linear Algebra in Pisa in 1992, and a CERFACS Workshop on the numerical solution of eigenvalue problems in 1993.

Iain has continued his involvement with the Parallel Algorithm Group at the European Centre for Research and Advanced Training in Scientific Computation (CERFACS) at Toulouse and has jointly supervised three students who completed their Ph D degrees during the period of this report (see Section 6.1). He visits Strathclyde University about once a term in his role of visiting Professor of Mathematics and, with the changes to the Research Council structure, intends to be more proactive in future grant proposals. He was an invited speaker at several major international meetings during the period of this report in Caracas, Oxford, Naples, Fukuoka, and Utrecht in 1991, Oaxaca, Minneapolis, Bielefeld, Manchester, and Toulouse in 1992, and Caracas, Antibes, Taiwan, Dundee, Porto, and Stockholm in 1993. He also gave many contributed talks and seminars, was on the organizing or programme committee for several international conferences, assisted in several theses examinations, and helped with courses, including tutorials at the Supercomputing Meetings in the USA and at ERCIM Courses in Europe. He had two visits to Malaysia, the first sponsored by the British Council to report on the status of Supercomputing in that country, the second as an assessor for the degree courses in Applied Mathematics at the University of Science in Malaysia. In advanced computing, he was on the scientific working group of the Rubbia Committee on High Performance Computing, was a member of the Technical Option Group of the ABRC Supercomputing Management Committee, and was chairman of the benchmark working group for the ABRC massively parallel supercomputer procurement in 1993. He continues as one of the principal editors of the IMA Journal of Numerical Analysis and is still editing the IMA Numerical Analysis Newsletter, now in its eighteenth year of publication. Since 1992, he has been one of the representatives of the IMA on the international CICIAM committee and was chairman of the steering committee for planning and presenting the successful bid for Britain to host the quadrennial International Meeting on Industrial and Applied Mathematics in 1999.

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.

2 Sparse Matrix Research

2.1 The solution of sparse structured symmetric indefinite linear sets of equations (I.S. Duff and J.K. Reid)

We have been developing software for the direct solution of sparse indefinite systems using a combination of 1×1 and 2×2 pivots to maintain numerical stability. The Harwell Subroutine Library multifrontal code MA27 owes its efficiency to performing the elimination operations within a full symmetric matrix (called the frontal matrix) with a row for each nonzero of the pivot column (or columns for 2×2 pivots). The pivot row (or rows) are stored elsewhere ready for backsubstitution and the rest of the frontal matrix is held until one of its rows is pivotal, at which point it is added into the frontal matrix for that row. We refer to such matrices as ***generated*** matrices since they are generated by the operations of a pivotal step. The analysis phase of MA27 chooses its pivot sequence by assuming that the matrix is definite (so that any diagonal entry is suitable as pivot) and representing each frontal matrix by the list of row indices involved.

The pivotal strategy used by MA27 can be far from optimal. Suppose the frontal matrix has the form

$$\begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_2^T & \mathbf{0} \end{pmatrix} \tag{1}$$

and a 2×2 pivot with one entry in each block is available. Since using a 2×2 pivot is mathematically equivalent to using each of its off-diagonal entries in turn as 1×1 pivots, it is easily seen that such a 2×2 pivot preserves the zero block. Now the zero block can be large, so this property can lead to significant computational savings. Indeed, $\mathbf{A}_1$ may have order one, in which case the generated matrix consists only of the zero submatrix and need not be kept at all.

The pattern of matrix entries altered by the pivotal operations always lies within the template

$$\begin{pmatrix} \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_3 \\ \mathbf{A}_2^T & \mathbf{A}_4 & \mathbf{A}_5 \\ \mathbf{A}_3^T & \mathbf{A}_5^T & \mathbf{0} \end{pmatrix}. \tag{2}$$

Our new code, MA47, uses this form to store all the generated matrices and is based on the algorithm [1].

The new code, MA47, has been completed and a report [2] is in preparation. It was hoped that this would replace MA27, but our experience is that it is not always superior. We have therefore decided that both deserve their place in the Harwell Subroutine Library. The format for passing the matrix to the two codes is the same. The weakness of MA47 is that it can be very slow when numerical considerations during factorization prevent use of the pivots

chosen when the pattern was analysed. We apply the usual relative pivot factor *u* but with a very small default value, 0.001. We are not really comfortable with this value, which may lead to an unstable factorization, but the computing costs can be very great when the value is larger. For some problems, on the other hand, the more sophisticated pivotal strategy of MA47 leads to substantially less work and storage.

We have used the Level 3 BLAS routines [3] for performing the inner-loop calculations since good implementations are widely available. The inner loop involves calculating the symmetric matrix $\mathbf{U}_k^T \mathbf{D}_k \mathbf{U}_k$ for a symmetric block pivot $\mathbf{D}_k$. Unfortunately, there is no Level 3 BLAS routine for this task, although _SYRK is available for the case $\mathbf{D}_k = \mathbf{I}$. We have decided to break the calculation into subblocks, and use _GEMM for each block of the block upper triangular part of the result.

The symmetric positive definite case is much easier to handle since the generated matrices are all full and only one such matrix need be stored for later use at each node of the assembly tree. We will therefore provide a separate code for this case. It will be able to use the BLAS routine _SYRK directly with no need for subblocks.

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. (1994). MA47, a Fortran code for direct solution of indefinite symmetric linear systems of equations. To appear.

[3] Dongarra, J.J., Du Croz, J.J., Duff, I.S., and Hammarling, S. (1990). A set of level 3 basic linear algebra subroutines. *ACM Trans. Math. Softw.* **16**, 1-17.

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

We have completed a new code, MA48, to replace our old workhorse MA28 for solving sparse unsymmetric sets of linear equations. Since computer memories are now much larger than they were when MA28 was written, we have adopted a design philosophy of requiring more storage when this leads to worthwhile performance improvements. An example of this is to construct a map array when first permuting a matrix of a given pattern. This means that subsequent matrices can be permuted by a single vectorizable loop of length the number of entries.

As well as using the same initial data structure, MA48 follows MA28 in seeking to permute a square matrix to block triangular form. For efficiency, we merge adjacent blocks of order one and note that the resulting diagonal block is triangular and so does not need factorization. We also merge adjacent blocks of order greater than one until they have a specified minimum size. This latter merging does affect the sparsity of the subsequent factorization and is performed solely to avoid procedure-call overheads for small blocks.

If the matrix is rectangular or square but structurally singular (there is no set of entries that can be permuted onto the diagonal), we treat the matrix as a single block. Block triangularization can be extended to these cases [9], but time did not permit us to incorporate such an extension.

Iterative refinement was not included in the original design of MA28, but was added later in the form of an additional subroutine. We have taken the opportunity in MA48 to build iterative refinement into the solve subroutine as an option. We also provide options for calculating estimates of the relative backward error and of the error in the solution [2].

A separate HSL package, MA30, was provided with MA28 for users willing to order their matrix entries by rows. It was called by MA28 to perform the fundamental tasks of matrix factorization and actual solution. We have followed the same model for the new code, with MA50 providing the fundamental facilities. One significant change is that MA50 is passed just one block of the block triangular form at a time which leads to worthwhile simplifications. It also has the advantage that it will be straightforward to multitask since the factorization of each diagonal block is an independent operation. For efficiency of execution of MA50, duplicate entries are not permitted, but they are permitted by MA48 (they are summed).

The default pivotal strategy is no longer that of Markowitz [8], though this is still available as an option. The Markowitz strategy chooses each pivot to minimize the product of the number of other nonzeros in its row and the number of other nonzeros in its column, subject to a stability test. Despite our best efforts with carefully designed data structures, we found that on some very large problems the code repeatedly searched a large number of rows and columns of minimum length before deciding that none contained a suitable pivot. We therefore follow Zlatev [10] and limit the search to a small number of columns (default 3). In practice, there is little to choose between the quality of the resulting factorizations.

The ANALYSE phase of MA28 actually factorizes the matrix that it is given. The FACTORIZE phase applies exactly the same pivot sequence to another matrix, and a fresh ANALYSE for the new matrix must be performed if the stability is unsatisfactory. Gilbert and Peierls [6] have shown that it is possible to incorporate partial pivoting into FACTORIZE with a computing effort that is proportional to the number of floating-point operations actually performed. This has led us to a FACTORIZE that accepts a recommended pivot sequence, but includes further row interchanges when they are needed. This demands the use of a dynamic data structure, but it is a simple one since only row interchanges are performed and the matrix can be processed column by column. This in turn has led us to provide an ANALYSE that discards the factorization that it produces and returns only the pivot sequence. The advantage in ANALYSE is that many less nonzeros are stored with consequent reduction in total storage requirements. A prediction is provided by ANALYSE for the storage needed in FACTORIZE.

Although the Gilbert and Peierls technique involves a computing effort that is proportional to the number of floating-point operations actually performed, this effort is nevertheless

greater than is needed when no interchanges are performed and the data structure is static. We therefore also provide a second entry to FACTORIZE that is comparable to that of MA28.

An enhancement that is particularly important on a supercomputer is to switch over to full code when the reduced matrix is sufficiently dense for sparsity techniques to be no longer appropriate. We had hoped to use the LAPACK [1] routines _GETRF and _GETRS for this purpose, but their treatment of the rank-deficient case is unsatisfactory since no column interchanges are included. Another reason for rejecting _GETRF is that it tests only for exact zeros. We test for exact zeros by default, but wish to offer the option of a test against a threshold. The final factorization will be as if we had started with a matrix whose entries differ from those of **A** by at most the threshold.

In our early tests, we found that factorization routines using Basic Linear Algebra Subroutines (BLAS) at Level 1 [7] and Level 2 [4] sometimes performed better than those at Level 3 [3], and have therefore included them all as options. In later tests, we found that the Level 3 versions performed best on all three of our test platforms, so the default parameter value chooses them.

It is the switch to full code that we believe is largely responsible for the improved timings. The improvement is greatest on a vector or parallel machine. On the Cray YMP, using one processor, our median improvement factors over MA28 were 4.5 for analysis, 12 for factorization with pivoting, 3.3 for analysis and factorization, 2.6 for factorization without pivoting, and 2.2 for solution.

This work is described in more detail by Duff and Reid [5].

References

[1] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. (1992). LAPACK users' guide. SIAM, Philadelphia.

[2] Arioli, M. Demmel, J.W., and Duff, I.S. (1989). Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl.* **10**, 165-190.

[3] Dongarra, J.J., Du Croz, J., Duff, I.S., and Hammarling, S. (1990). A set of Level 3 basic linear algebra subroutines. *ACM Trans. Math. Softw.* **16**, 1-17.

[4] Dongarra, J.J., Du Croz, J., Hammarling, S., and Hanson, R.J. (1988). An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **14**, 1-17.

[5] 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.

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

[7] Lawson, C.L., Hanson, R.J., Kincaid, D.R., and Krogh, F.T. (1979). Basic linear algebra subprograms for Fortran use. *ACM Trans. Math. Softw.* **5**, 308-325.

[8] Markowitz, H.M. (1957). The elimination form of the inverse and its application to linear programming. *Management Sci.* **3**, 255-269.

[9] Pothen, A. and Fan, C-J. (1990). Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Softw.* **16**, 303-324.

[10] Zlatev, Z. (1980). On some pivotal strategies in Gaussian elimination by sparse technique. *SIAM J. Numer. Anal.* **17**, 18-30.

2.3 Development of a new frontal solver (I. S. Duff and J. A. Scott)

The Harwell Subroutine Library code MA32 ([3], [4]) is a frontal code which has been extensively used over the last decade by people working with finite elements. A major revision of the code was required to allow it to run efficiently on a wide range of modern computers and to enable further developments in frontal matrix solution to be considered. Our new frontal code is called MA42 and is included in Release 11 of the Harwell Subroutine Library. A complex version, ME42, is also included in Release 11.

The frontal method (see, for example, [5], [7], [8]) is a variant of Gaussian elimination and involves the factorization of a permutation of **A** which can be written as

$$\mathbf{A} = \mathbf{PLUQ},$$

where **P** and **Q** are permutation matrices, and **L** and **U** are lower and upper triangular matrices, respectively. The code MA32 allowed the matrix **A** to be input either by equations or by finite elements. The interface to the user was through reverse communication with control returned to the calling program each time an element or equation needed to be input.

A principal feature of MA32 was that it could solve large problems in a predetermined and relatively small amount of main memory. In general, files for the factors **PL** and **UQ** were not held in the main memory. Instead, direct access files were used (one for **PL** and one for **UQ**).

MA32 allowed the user to solve for a specified number of right-hand sides by operating on the right-hand sides at the same time as the factorization of the matrix. In addition, MA32 provided a separate entry for the solution of subsequent systems with the same coefficient matrix, and provided two further entries which together could be used to generate the factors of $\mathbf{A}^{\mathrm{T}}$ and solve the transpose equations

$$\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}.$$

Although the new code MA42 retains these features and employs many of the internal data

structures used in MA32, the structure of the package has been radically altered. The main reasons for the changes were to increase the modularity and portability of the code, to make the code more readable and easier to maintain, and to facilitate further developments in frontal matrix solution, including exploitation of parallelism and the design of codes for complex and symmetric cases. Another key reason for the restructuring was to allow greater use of Level 2 and Level 3 Basic Linear Algebra Subprograms (BLAS) [1], [2], both in the inner loops of the factorization and when performing forward and back-substitutions. Expressing the frontal algorithm in terms of the BLAS provides a means of achieving high performance portable Fortran code.

The structure of the direct access files used by MA42 has been significantly altered. The reals and integers for the factors are held in separate direct access files. This makes reading to and writing from the direct access files more straightforward and increases the portability of the code. The portability is further enhanced by only writing a buffer to its associated direct access file when either it is completely full or the factorization is complete. This is more efficient since it prevents the fragmentation of data, reducing the number of records used. The new structure also allows transpose systems to be solved as soon as the matrix **A** has been factorized. In the earlier MA32 code, the storage scheme for the factors did not permit this. Instead, the factors of $\mathbf{A}^{\mathrm{T}}$ had first to be explicitly generated from those of **A** and then stored separately.

Another important change in MA42 is that the basic logical unit is a block of eliminations, rather than the single or double pivot operation used in MA32. Using blocks of eliminations enables Levels 2 and 3 BLAS to be exploited more fully (see [6]).

In addition to retaining all the functionality of MA32, MA42 offers the user some new facilities. These include an optional symbolic factorization of the matrix to obtain lower bounds on the maximum front sizes and estimates of the file sizes required by the factors. This will greatly facilitate the user's job of choosing appropriate sizes for the direct access files and for the associated buffers. In addition, whereas MA32 terminated the computation with an error message when the matrix was detected to be singular, MA42 offers the user the option of continuing. In this case, an estimate of the deficiency of the matrix is returned to the user at the end of the factorization and in the solution vector (or solution matrix for multiple right-hand sides) components corresponding to zero pivots are set to zero.

A full discussion of MA42, as well as numerical results, is given in [6].

References

[1] Dongarra, J.J., Du Croz, J., Duff, I.S., and Hammarling, S. (1990). A set of Level 3 basic linear algebra subroutines. *ACM Trans. Math. Softw.* **16**, 1-17.

[2] Dongarra, J.J., Du Croz, J., Hammarling, S., and Hanson, R.J. (1988). An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **14**, 1-17.

[3] Duff, I. S. (1981). MA32 – A package for solving sparse unsymmetric systems using the frontal method. AERE R10079, HMSO, London.

[4] Duff, I. S. (1983). Enhancements to the MA32 package for solving sparse unsymmetric equations. AERE R11009, HMSO, London.

[5] Duff, I. S. (1984). Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM J. Sci. Stat. Comput.* **5**, 270-280.

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

[7] Hood, P. (1976). Frontal solution program for unsymmetric matrices. *Int. J. Numer. Meth. Engng.* **10**, 379-400.

[8] Irons, B. M. (1970). A frontal solution program for finite-element analysis. *Int. J. Numer. Meth. Engng.* **2**, 5-32.

2.4 A variable-band linear equation frontal solver (J. A. Scott)

The frontal code MA42 [1] may be used to solve systems of linear equations $\mathbf{Ax}=\mathbf{b}$ (or $\mathbf{A}^{\mathrm{T}}\mathbf{x}=\mathbf{b}$) where $\mathbf{A}$ is either the sum of finite-element matrices or is an assembled matrix. Reverse communication is used by MA42 and, if $\mathbf{A}$ is assembled, the user must enter the rows of $\mathbf{A}$ one at a time. This use of reverse communication may not be considered very convenient or natural if the matrix does not arise from a finite-element application. The user interface to MA42 is further complicated by offering the user the option of holding the matrix factors in direct access data sets to keep in-core memory requirements low. To simplify the use of MA42 for the solution of assembled systems of equations, we decided to design a code which would provide a straightforward interface to MA42 when entry is by equations and auxiliary storage is not required. The code we have developed is called MA43 and the complex version is ME43. Both MA43 and ME43 are included in Release 11 of the Harwell Subroutine Library.

MA43 uses a standard sparse matrix format: the user must provide the row and column indices and values of the entries of $\mathbf{A}$. A single call must be made to the analysis and factorization routines MA43A and MA43B, respectively. The routine MA43C may optionally be called to rapidly solve further systems of the form $\mathbf{Ax}=\mathbf{b}$ or $\mathbf{A}^{\mathrm{T}}\mathbf{x}=\mathbf{b}$.

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.

2.5 The use of multiple fronts (I. S.Duff and J. A. Scott)

There are two main deficiences with the frontal solution scheme for solving systems of linear equations. The first is that far more arithmetic is done than is required by the numerical factorization, and the second is that there is little scope for parallelism other than that which can be obtained within the higher level BLAS. In order to overcome these problems in the finite-element case, we propose allowing several independent fronts. With multiple fronts, we partition the underlying finite-element domain into subdomains, perform a frontal decomposition on each subdomain separately, and then factorize the remaining interface variables, perhaps by also using a frontal scheme. This strategy corresponds to a bordered block diagonal ordering of the matrix and can be nested. With judicious ordering within each subproblem, the amount of work required can be reduced and, since the factorizations of the subproblems are independent, there is much scope for parallelism.

The MA42 code is quite complicated and we were keen to avoid a major rewriting of this code to implement a multiple front strategy. We have thus designed and written three subroutines (MA52A, MA52B, and MA52C) that can be used in conjunction with MA42 to effect this implementation. The only change made to MA42 was the addition of an option to force diagonal pivoting. The elimination is performed using the following steps.

(i) MA42A and MA42B are run on each subdomain. An artificial guard element consisting of all the variables on internal boundaries is supplied to MA42A but not to MA42B. MA52A may (optionally) be used to generate the guard element.

(ii) After all calls to MA42B on the subdomain are complete, MA52B is called to output any remaining factors to the appropriate direct access files and to create an element corresponding to the Schur complement matrix.

(iii) MA42A and MA42B are then run on the problem whose elements are those created at step (ii). The number of elements in this problem is equal to the number of subdomains.

(iv) Backsubstitution is then performed on each subdomain using MA52C.

Clearly steps (i) and (iv) can be performed in parallel, and it is possible to nest step (iii), that is to treat the finite-element problem at step (iii) as we did the original problem.

We have performed some initial experiments in two parallel environments: on an eight processor shared-memory CRAY Y-MP8I and on a network of five DEC Alpha workstations using PVM [2]. The results on the CRAY are encouraging and show good speedup for large problem sizes (for details, see [1]). Those on the Alpha farm are quite comparable and indicate that, for larger problems, the overheads of PVM and communication costs do not dominate and good speedups are possible. All the present runs used the same number of processors as subdomains. However, we plan further experiments in which we will use more subdomains than processors.

References

[1] Duff, I. S. and J. A. Scott (1994). The use of multiple fronts in Gaussian elimination. To appear in Proceedings of the Fifth SIAM Conference on Applied Linear Algebra, Snowbird, Utah.

[2] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1993). PVM 3 User's Guide and Reference Manual. Report ORNL/TM-12187, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Tennessee.

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

MUPS is a package being developed to solve sparse unsymmetric systems of linear equations on shared memory parallel computers.

The method used is a direct method based on a sparse multifrontal variant of Gaussian elimination [10]. The background to this parallel implementation and further details of the algorithms used are discussed in [1], [2], [8], and [9]. A preliminary version of the code is now available and has been tested on the IBM 3090/6VF, the Alliant FX/80, the CRAY-2 and the CRAY Y-MP.

Although the overall structure of the code is similar to [10] inasmuch as the solution is divided into an analysis phase to construct an ordering and various data structures, a numerical factorization phase, and a solution phase, there are many implementation and algorithmic differences from this earlier code. For example, because of the parallel implementation, it is not appropriate to use a stack for the working area during numerical factorization and management and allocation of working storage become much more complicated. The memory allocation strategy has been designed to minimize the size of the working area while keeping the efficiency of the parallel solver. Memory allocation issues are further discussed in [4].

We have significantly modified the numerical factorization with new pivoting strategies [1] to improve the performance of the numerical factorization without adversely affecting the stability of the factorization. We also use the KJI-SAXPY block algorithm [6], which uses Level 3 BLAS [7], during the frontal matrix factorization.

In addition to the normal three phase solution process (analysis, factorization, and solution), we also provide a driver package that offers most of the standard pre-processing and post-processing facilities used in sparse matrix computations. Two classical ordering strategies are available to the user (minimum degree and nested dissection). A variety of scaling strategies, which were studied in [1], can be automatically performed before numerical factorization. Furthermore, the driver can optionally perform iterative refinement after the solution step and can also return information related to classical error analysis, using the analysis in [5].

Work is now continuing to bring this experimental version of MUPS to a sufficient software standard for distribution with the Harwell Subroutine Library. A report on this work in currently in preparation [3].

References

[1] Amestoy, P. R. (1991). Factorization of large unsymmetric sparse matrices based on a multifrontal approach in a multiprocessor environment. PhD Thesis, Report TH/PA/91/2, CERFACS, Toulouse.

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

[3] Amestoy, P. R. and Duff, I. S. (1994). MUPS: a parallel package for solving sparse unsymmetric sets of linear equations. Technical Report, CERFACS, Toulouse, France. To appear.

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

[5] Arioli, M., Demmel, J. W., and Duff, I. S. (1989). Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. and Applics.* **10**, 165-190.

[6] Daydé, M. and Duff, I. S. (1989). Use of Level 3 BLAS in **LU** factorization on the CRAY-2, the ETA 10-P, and the IBM 3090/VF. *Int. J. Supercomputer Applics.* **3** (2), 40-70

[7] Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. (1990). A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **16**, 1-17.

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

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

[10] Duff, I. S. and Reid, J. K. (1984). The multifrontal solution of unsymmetric sets of linear systems. *SIAM J. Sci. Stat. Comput.* **5**, 633-641.

2.7 Sparse matrix factorization on the BBN TC2000 (P.R. Amestoy and I.S. Duff)

We have developed a version of the MUPS code, discussed in Section 2.6, for the direct solution of large sparse sets of linear equations on the BBN TC2000 computer.

The BBN TC2000 is a distributed memory multiprocessor with up to 64 processors (Motorola MC88100 RISC chips). The peak performance of each processor is 10 Mflop/s using 64-bit arithmetic. The originality of the BBN TC2000 comes from its interconnecting

network (Butterfly switch) and from its globally addressable memory. The BBN TC2000 enables data to be considered as logically and physically private to the processor, as logically shared and physically local to the processor, or as logically shared and physically distributed among the processors. Clearly the access time will depend on the physical locality of the data. Although the BBN TC2000 computer can be used as a shared memory computer, we must modify the shared memory algorithm to make better use of the memory hierarchy of the BBN TC2000 (see also articles [2] and [3]).

In our first two implementations of the code, we used the BBN TC2000 as a shared memory multiprocessor computer. In *version* 1 of the code, all logically shared data is physically on a single processor. In this case, only a small part of the bandwidth of the interconnecting network is used. Another possibility is to uniformly distribute the shared data on the physical memory of the computer. This can be done using the interleaving feature of the computer and is independent of the number of processors actually used. This will be referred to as *version* 2 of the code. In both versions, we use uniprocessor tuned versions of the shared BLAS [2].

Note that these first two versions of the code are fairly straightforward adaptations of the shared memory multifrontal code described in [1] and [4].

In *version* 3, we modify the original algorithm to make better use of the architecture of the BBN TC2000. To take better account of the memory hierarchy, we perform the frontal matrix elimination steps on private data. Subblocks of the shared frontal matrix are copied to the private area and stored back after computation. Interleaving and cacheability are also used for all shared data. Note that, to prevent cache inconsistency problems, cache flush instructions must be inserted in the code.

In *version* 4 of the code, data is not automatically copied to private memory, and the decision to copy shared blocks of data involved in the BLAS kernels is based on a simplified model using average memory access times and average Megaflop rates. Furthermore, to create more parallelism during a node process and to use data locality better, we have modified the algorithm used in *version* 3. We overlap the parallel updating involved at step k of the blocked factorization scheme with the elimination of the next block at step k+1.

No of procs	1	2	3	4	6	8	16	20	26
version 1	328	227	170	151	117	105	103	100	95
version 2	490	300	213	179	129	117	102	98	83
version 3	216	174	124	95	75	63	48	42	38
version 4	216	155	107	86	68	55	44	36	31

Table 2.7.1. Performance summary in seconds on matrix BCSSTK15.

We show, in Table 2.7.1, timings obtained for the numerical factorization of a medium-size (3948×3948) sparse matrix, BCSSTK15, from the Harwell-Boeing set [4]. The minimum degree ordering is used. The number of floating-point operations in the factorization is 438.4 Million. We can see clearly the benefits of changing our algorithm to match the architecture of our virtual shared memory computer. With *version* 4 of our code, we achieve 14.3 Mflop/s (in double precision) which corresponds to almost three times the best performance obtained with the best straightforward shared memory implementation of the multifrontal code.

References

[1] Amestoy, P. R. (1991). Factorization of large unsymmetric sparse matrices based on a multifrontal approach in a multiprocessor environment. PhD Thesis, Report TH/PA/91/2, CERFACS, Toulouse.

[2] Amestoy P. R., Daydé M. J., Duff I. S., and Morère P. (1992). Linear algebra calculations on the BBN TC2000. In *Parallel Processing: CONPAR 92-VAPPV* Editors L. Bougé, M. Cosnard, Y. Robert, and D. Trystram. Lecture Notes in Computer Science **634**, Springer-Verlag, Berlin, Heidelberg, New York, and Tokyo, 319-330.

[3] Amestoy, P. R. and Duff, I. S. (1994). MUPS: a parallel package for solving sparse unsymmetric sets of linear equations. Technical Report, CERFACS, Toulouse, France. To appear.

[4] 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.

2.8 Unsymmetric multifrontal methods (T.A. Davis and I.S. Duff)

While the multifrontal schemes of Sections 2.6 and 2.7 are designed for structurally symmetric matrices, they can handle unsymmetric matrices by explicitly holding zero entries. It is more complicated to develop an efficient multifrontal scheme for matrices that are asymmetric in structure. The main difference is that the elimination cannot be represented by an (undirected) tree but now requires a directed acyclic graph (DAG). The frontal matrices are, of course, no longer square and, as in the case discussed in Section 2.1, the frontal matrices are not necessarily absorbed at the parent node and can persist in the DAG. Finally the complication of *a posteriori* numerical pivoting is even more of a problem with this scheme so that the approach adopted is normally to take account of the real values when computing the DAG and the pivot order. We have written a code, called UMFPACK, which implements this algorithm and are presently polishing it for distribution with the Harwell Subroutine Library. We show some results from [1] in Table 2.8.1, where the code MUPS assumes structural symmetry, while UMFPACK allows structural asymmetry. We see that, although UMFPACK is less efficient on the symmetric or near-symmetric cases (first three matrices), it is significantly better than MUPS on the more asymmetric matrices.

Matrix	Order	Number entries	Asymmetry	Entries in factors $*10^3$		Number flops $*10^6$	
				MUPS	UMFPACK	MUPS	UMFPACK
BCSSTK18	11948	149090	0.00	1540	4733	347	2335
LNS 3937	3937	25407	0.15	333	543	27	101
SHERMAN5	3312	20793	0.26	187	285	17	44
MAHINDAS	1258	7682	0.98	50	13	2.8	0.2
GRE 1107	1107	5664	1.00	188	94	25	5.7

Table 2.8.1. Comparison of MUPS (symmetric) and UMFPACK (unsymmetric) on a few matrices from the Harwell-Boeing collection. The asymmetry measures the proportion of unmatched entries.

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.

2.9 Unsymmetric multifrontal methods for finite-element problems (A.C. Damhaug, I.S. Duff, J.A. Scott, and J.K. Reid)

Good progress has been made in the construction of a multifrontal code for unsymmetric finite-element sets of linear equations $\mathbf{AX}=\mathbf{B}$. The new code will be added to 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. It uses the algorithms developed by Damhaug [1] for the symmetric positive-definite case to construct information for the factorization stage. The ordering is done

with the minimum-degree heuristic. It is possible to relax this by altering the default value of a parameter *c* from zero. Setting it greater than zero has the effect of allowing nodes with degree *c* greater than the minimum to be eliminated together with the nodes of minimum degree. Sometimes, this helps to reduce the size of the decomposition. 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.

A report describing this work is in preparation [2].

References

1. Damhaug, A.C. (1992). Sparse solution of linear finite element equations. Dr Ing. thesis, NTH Trondheim, Norway.

2. Damhaug, A.C. and Reid, J.K. (1994). MA46, a FORTRAN code for direct solution of sparse unsymmetric linear systems of equations from finite-element applications. To appear.

2.10 Automatic scaling of sparse symmetric matrices (J.K. Reid)

Curtis and Reid [1] proposed the automatic scaling of an unsymmetric sparse matrix **A** by using the method of conjugate gradients to choose ρ_i and c_j to minimize approximately the quantity

$$\sum_{a_{ij}\neq 0} (\log_\beta |a_{ij}| - \rho_i - c_j)^2,$$

where β is the machine radix. They then used as row and column scaling factors the quantities β^{-R_i} and β^{-C_j}, where R_i and C_j are the nearest integers to ρ_i and c_j, respectively. They used the machine radix β so that the logarithm could be found quickly from the exponent part of the value and so that no roundoff was caused by the scaling. However, there were portability problems and our experience was that the graininess was unsatisfactory, particularly for β = 16. We have therefore replaced our codes by codes that take the natural logarithm and use the scaling factors $e^{-\rho_i}$ and e^{-c_j}. This is the algorithm of `MC29` (real, sparse) and `MC42` (real, full) and their complex equivalents `MF29` and `MF42`.

These codes are applicable to a symmetric matrix, but it is clearly more satisfactory to have a specially tailored code. Minimization of

$$\sum_{a_{ij}\neq 0} (\log |a_{ij}| - \rho_i - c_j)^2,$$

when **A** is symmetric, corresponds to solution of the system

$$(\mathbf{M}+\mathbf{E})\rho = \sigma,$$

where **M** is the diagonal matrix whose *i*–th diagonal entry is the number of entries in row *i* of **A**, **E** is the matrix obtained from **A** by replacing each entry by unity, and σ is the vector whose *i*–th element is the sum of the logarithms of the nonzeros of row *i* of **A**. The method of conjugate gradients can be applied in a straightforward way to this and convergence is rapid. The quality of the result is as for `MC29` and the speed is typically twice as fast.

References

[1] Curtis, A.R. and Reid, J.K. (1972). On the automatic scaling of matrices for Gaussian elimination. *J. Inst. Maths. Applics.* **10**, 118-124.

2.11 Block triangular form and QR decomposition (I.S. Duff and C. Puglisi)

We use a multifrontal method to implement a sparse **QR** decomposition, by using the observation that the Cholesky factor $\mathbf{L}^T$ of the normal equation matrix $\mathbf{A}^T\mathbf{A}$ is identical to the **R** matrix of the **QR** factorization, except for possible sign differences on the diagonal. This observation was used by [2] to develop an algorithm for solving sparse least-squares problems and it is essentially the same approach that we use here. We form the normal equations matrix, $\mathbf{M}=\mathbf{A}^T\mathbf{A}$ symbolically and then order **M** using a minimum degree ordering, which gives us the pattern of the factors of $\mathbf{L}^T$ and hence **R**, and a column ordering for **A**. In addition to giving us the structure of **R**, the assembly tree from the analysis on the pattern of **M** also gives us information about the row dependencies of **R** and we can use this tree to drive our numerical factorization of **A**. When we use this tree to drive the numerical **QR** factorization of **A**, we associate the nodes of the tree with columns of the matrix **A**.

One problem with this approach is that, although the matrices **R** and $\mathbf{L}^T$ are mathematically equivalent, the structure of $\mathbf{L}^T$ can severely overestimate that of **R**, which means that our analysis will not necessarily give a good indication of the space needed by the factorization. We can see this by considering the matrix

$$\mathbf{A}=\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & & & \\ & & \times & & \\ & & & \times & \\ & & & & \times \end{bmatrix}.$$

The pattern of $\mathbf{A}^T\mathbf{A}$ is full and therefore the predicted **R** is full as well; but the actual **R** factor is equal to **A**. Note that the predicted **R** factor is full whatever the numerical values of the matrix **A**.

Coleman *et al*. [1] have found a class of matrices for which the analysis predicts the exact pattern of the matrix **R**. Matrices which have this so called ***strong Hall property*** do not suffer from the previously mentioned “bogus” fill. Furthermore, a matrix which has not the strong

Hall property can always be permuted to a block upper triangular form (BTF) such that each diagonal block has this property. Algorithms for computing the BTF of a sparse matrix are based on a canonical decomposition of the bipartite graph representing the matrix (see [3]).

We have done extensive testing to detect this extra fill-in and have found that, even when matrices do not have the strong Hall property, in practice the overestimate by the analysis is usually very slight, seldom as much as 10%.

We show in Table 2.11.1 the effect of the permutation to BTF on the memory requirement during the factorization step. We see that, by permuting the matrix to a block triangular form, we slightly reduce the fill-in in the **R** factor. Our test matrices are not very "reducible" so that the reduction in the fill-in is not very impressive. However, we do fewer Householder transformations so that we need less space to store the Householder vectors (see space needed for **Q** in Table 2.11.1). We also saw, although not recorded in Table 2.11.1, that the factorization time was sometimes reduced and the accuracy improved by using a pre-ordering to BTF. This was, however, at the cost of extra temporary storage to handle the information concerning the blocks.

A fuller discussion of these issues can be found in the thesis of Puglisi [4].

Matrix	BTF	Nonzeros in **R**	Space for **Q**
WEST2021	yes	15432	14328
	no	18911	18629
SCFXM3	yes	14574	36064
	no	14628	36296
WELL1850	yes	7476	30425
	no	7487	30602
NESM	yes	21303	119101
	no	21735	120374
LNS3937	yes	414315	262617
	no	416049	268450

Table 2.11.1. Influence of the permutation to block triangular form on the multifrontal **QR** factorization.

References

[1] Coleman, T. F., Edenbrandt, A., and Gilbert, J. R. (1986). Predicting fill for sparse orthogonal factorization. *J. ACM* **33**, 517-532.

[2] George, A. and Heath, M. T. (1980). Solution of sparse linear least squares problems using Givens rotations. *Linear Alg. and its Applics.* **34**, 69-83.

[3] Pothen, A. and Fan, C.–J. (1990). Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Softw.* **16**, 303-324.

[4] Puglisi, C. (1993). **QR** factorization of large sparse overdetermined and square matrices using a multifrontal method in a multiprocessor environment. Ph D Thesis. report TH/PA/93/33, CERFACS, Toulouse.

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

We consider the implementation of the multifrontal **QR** factorization discussed in the previous section in a multiprocessor environment. Since we can represent the factorization by an assembly tree where nodes correspond to columns of the matrix, we can use similar implementation ideas to those used in the multifrontal solution of equations using **LU** factorization as discussed in Sections 2.6 and 2.7. In particular we can exploit parallelism both from the tree and from the use of higher level BLAS within calculations at a node.

Matrix	$(m \times n)$	1 Proc.	8 Proc.		Speedup	
			(1)	(2)	(1)	(2)
WEST2021	(2021×2021)	8.84	1.45	1.44	6.10	6.14
medium2	(18794×12238)	254.34	70.85	46.38	3.59	5.48
large2	(56508×34528)	2139.57	815.92	479.96	2.62	4.46
Nfac90	(31684×8100)	123.76	22.26	19.23	5.56	6.44
Nfac100	(39204×10000)	177.84	32.63	28.35	5.45	6.27

Table 2.12.1. Time in seconds for multifrontal **QR** factorization on the Alliant FX/80.
In columns (1) only tree parallelism is exploited
In columns (2) node parallelism is also exploited

From Table 2.12.1, we see that although sometimes large speedups can be obtained using only the parallelism of the tree structure (Nfac matrices and WEST2021), the added parallelism obtained by using automatic parallelization based on loop level analysis within the node calculations gives a much improved performance on most examples. This shows that this additional parallelism can balance well the lack of tree parallelism near the root.

We have also studied the impact of node amalgamation [2] on the performance of our **QR** factorization. We have found that, on the Alliant FX/80, the use of node amalgamation was very beneficial on both 1 and 8 processors with gains of around 30% in execution time although the speedups were generally slightly less than in column 7 of Table 2.12.1.

We have also used this multifrontal **QR** algorithm in the solution of sparse least-squares problems and have compared it with a range of other methods including normal equations, semi-normal equations, and the augmented system method. We have found the **QR** method very competitive in terms of solution time, although usually somewhat more expensive in storage. It is, however, clearly the best method for accuracy especially if row pivoting and iterative refinement are used.

These experiments are described in detail in [3] and a report is currently in preparation [1].

References

[1] Amestoy, P. R., Puglisi, C., and Duff, I. S. (1994). Multifrontal **QR** factorization in a multiprocessor environment. Technical Report, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear.

[2] Duff, I. S. and Reid, J. K. (1983). The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.* **9**, 302-325.

[3] Puglisi, C. (1993). **QR** factorization of large sparse overdetermined and square matrices using a multifrontal method in a multiprocessor environment. Ph D Thesis. report TH/PA/93/33, CERFACS, Toulouse.

2.13 Impact of full QR factorization on sparse multifrontal QR algorithm (I.S. Duff and C. Puglisi)

As we have seen in the previous two sections, the entire numerical computation of the multifrontal method is carried out on full matrices. It is thus crucial that the factorization of full matrices is performed efficiently, and, to this end, we have studied a block version of the full Householder factorization [2]. We have conducted numerous experiments with sets of large overdetermined matrices described in [3] on an Alliant FX/80 and a Convex C220. Performance analysis has shown that the full matrices have to be of reasonable order and several eliminations must be performed on the same frontal matrix before block algorithms can be used efficiently. The exact size and number of eliminations is of course machine dependent. Moreover, when we examined the size of frontal matrices obtained during the factorization over a wide range of matrices, we found that the sizes were usually insufficient to make block factorizations sensible. The reason is that the frontal matrices have a block structure and therefore the full **QR** factorization step is only applied to submatrices (subblocks of the frontal matrix). These subblocks are too small for efficient block factorization. To increase the use of the block factorization algorithm, we have relaxed the sparsity structure of the frontal matrices, called the ***relaxed block algorithm***. We do this by including small blocks of zeros to build the subblocks to sufficient size. We modify the assembly of the frontal matrix to provide full column blocks of at least the block size used in the block algorithm.

Matrix	Unblocked algorithm Time	Block algorithm		Relaxed block algorithm	
		Flops ($\times 10^6$)	Time	Flops ($\times 10^6$)	Time
Alliant FX/80					
large	33.3	112	33.3	139	33.0
medium2	34.6	293	34.6	350	28.8
large2	406.1	3464	406.1	3818	219.9
EXP	119.0	1561	60.9	1574	36.0
Convex C220					
large2	280.4	3464	280.4	3701	271.0
EXP	66.4	1561	53.2	1574	46.7

Table 2.13.1. Use of relaxed block algorithm within sparse multifrontal **QR** factorization to enable more Level 3 BLAS operations. Times for factorization (in seconds) are obtained on eight processors of an Alliant FX/80 and on one processor of a Convex C220.

We show in Table 2.13.1, the influence of the relaxation of the sparsity structure of the frontal matrices on the performance of the **QR** algorithm. We see that, with relaxation of the nonzero structure, we sometimes obtain a significant decrease in the time to perform the factorization step although slightly more floating-point operations are necessary because of our relaxation of the sparsity structure. This performance improvement only comes from the relative increase in the Megaflop rate during **QR** factorization due to the increase in the use of higher Level BLAS kernels. We also see that, although the unblocked version of the code is 60% faster on the Convex than on the Alliant (see column 2, matrices large2 and EXP), higher performance is obtained (see column 6) on the Alliant than on the Convex with the relaxed block algorithm.

Much of this work is described in detail in [3] and a report on this work is currently in preparation [1].

References

[1] Amestoy, P. R., Puglisi, C., and Duff, I. S. (1994). Multifrontal **QR** factorization in a multiprocessor environment. Technical Report, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear.

[2] Puglisi, C. (1992). Modification of the Householder method based on the compact *WY* representation. *SIAM J. Sci. Stat. Comput.* **13**, 723-726.

[3] Puglisi, C. (1993). **QR** factorization of large sparse overdetermined and square matrices using a multifrontal method in a multiprocessor environment. Ph D Thesis. report TH/PA/93/33, CERFACS, Toulouse.

2.14 The solution of linear least squares problems using the augmented system formulation (I.S. Duff, N.I.M. Gould, and J.M. Patrício)

We consider the solution of full-rank linear least-squares problems by means of the augmented system formulation that leads to the solution of a linear symmetric indefinite linear system with a coefficient matrix of the form

$$\begin{pmatrix} \mathbf{I} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0} \end{pmatrix}.$$

In [3] we discuss this augmented system is some detail, describing several other areas giving rise to such systems and examining various methods of solution. In this present work, we consider using the code for sparse symmetric systems of equations considered in Section 2.1, which is especially designed for such structured indefinite systems.

The augmented matrix is not only indefinite, but also is often quite ill-conditioned. Therefore, it is desirable to consider pre-processing the problem and post-processing the computed solution to improve the accuracy of the final result, using, for instance, scaling and iterative refinement.

The iterative refinement implementation that we use is the subroutine `MA40` from the Harwell Subroutine Library. It is very efficient, but its ability to obtain an accurate solution depends on the accuracy of the first estimate. Therefore, if an initially bad solution is given (which may happen if the problem is ill-conditioned), iterative refinement may fail to produce a significant improvement.

We consider two scaling strategies: a local strategy initially presented by A. Björck, usually known as "α–scaling", and a global approach based on the pre- and post-multiplication of the matrix by a diagonal matrix whose elements are computed in order to bring the nonzero components of the resulting scaled matrix close to unity (see also Section 2.10). This latter strategy is implemented as subroutine `MC30` in the Harwell Subroutine Library [2].

We have performed some computational experiments using the abovementioned strategies, in order to understand their behaviour. The main disadvantage of the "α–scaling" is its dependence on the parameter α. In a recent paper, Björck [1] suggests an optimal value obtained from the minimization of a function of the error. The main drawback of this value is that, in order to compute it, it is necessary to have a good estimate of the smallest singular value of **A**. However, Björck's approach leads to the best 2-norm error in the solution. On the other hand, the global strategy is quite simple to implement and does not depend on any parameter, but the final solution is usually not as accurate as that obtained using Björck's scaling. The use of iterative refinement coupled with any of the two scaling strategies allows us to recover some extra precision.

As we can see from the previous paragraph, the two scaling processes are quite complementary in terms of advantages and disadvantages. Therefore, we are presently exploring a "mixed-scaling" strategy, where we first apply a global scaling to the initial

augmented matrix, and then determine the appropriate alpha parameter associated with this new augmented system. This work is still at an early stage, but the initial results seem to be promising. A report on this work is in preparation [4].

References

[1] Björck, Å. (1991). Pivoting and stability in the augmented system method. Report LiTH-MAT-R-1991-30, Dept. Maths, Linköping Univ., Sweden.

[2] Curtis, A. R. and Reid, J. K. (1972). On the automatic scaling of matrices for Gaussian elimination. *J. Inst. Maths. Applics.* **10**, 118-124.

[3] 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. Report RAL-93-084, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[4] Duff, I. S., Gould, N. I. M., and Patricio, J. (1994). Experiments in scaling augmented systems. Technical Report, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. To appear.

2.15 Solution of large sparse unsymmetric linear systems with a block iterative method in a multiprocessor environment. (M. Arioli, I.S. Duff, J. Noailles, D. Ruiz, and M. Sadkane)

We consider a block version of Cimmino's algorithm for solving general sets of consistent sparse linear equations [1]. We concentrate on the case of matrices in block tridiagonal form because they are common in many applications, including CFD, and the general case can be reduced to this form by permutations. We show how the basic method can be accelerated by using the conjugate gradient algorithm. This acceleration is very dependent on a partitioning of the original system and we discuss several possible partitionings. Underdetermined systems corresponding to the subproblems of the partitioned system are solved using the Harwell sparse symmetric indefinite solver `MA27` on an augmented system. These systems are independent and can be solved in parallel. An analysis of the iteration matrix for the conjugate gradient acceleration leads us to consider rather unusual and novel scalings of the matrix that alter the spectrum of the iteration matrix to reduce the number of conjugate gradient iterations.

We have tested the various aspects of our algorithm by runs on an eight processor Alliant FX/80. The effect of partitioning and scaling on the number of iterations and overall elapsed time for solution is studied. For iteration matrices which are not too ill-conditioned, the conjugate gradient algorithm is a good method for accelerating the convergence of the iterative scheme.

For ill-conditioned problems, the classical conjugate gradient algorithm does poorly

because of clusters of eigenvalues at the ends of the spectrum of the iteration matrix. We therefore try variants of the Block Lanczos method [2], [3], including the Block Conjugate Gradient method. In contrast to the classical conjugate gradient algorithm, these block techniques allow the computation of multiple eigenvalues and are thus capable of detecting clusters of eigenvalues in finite-precision arithmetic. However, the computations in the block Lanczos method involve the solution of symmetric positive definite systems rather than the division by scalars as in the classical conjugate gradient algorithm. Whereas the scalar division is perfectly conditioned, the matrices in the corresponding block operation may be very ill-conditioned, especially when convergence is almost reached. This can amplify round-off errors in the computations and disrupt the convergence.

From among this class of methods, we identify those that are numerically stable and compare their efficiency. Although these block acceleration techniques increase the amount of arithmetic, they can still be more efficient in a parallel environment since they allow the use of Level 3 BLAS kernels. Furthermore, in some test examples, these techniques converge whereas the conjugate gradient method does not.

At the current stage of our research, we can only decide *a posteriori* between using block or non-block conjugate gradients, after the observation of the good or bad behaviour of the classical conjugate gradient algorithm. By using roundoff analysis and approximation theory, we plan to understand better the differences between the behaviour of the classical conjugate gradient algorithm and that of the block conjugate gradient algorithm. Finally, we will investigate if some heuristics can be defined for deciding *a priori* between these block and non-block acceleration techniques.

We are now investigating more general preconditioning techniques, as well as some techniques for preprocessing general linear systems. The preconditioners actually used are linked to the partitioning strategy and preserve its block structure. At the same time, we would like to study more general partitioning strategies, and in particular, partitioning strategies with overlapping blocks. Partitionings with overlapping blocks also exhibit some similarities with domain decomposition approaches. We plan to analyse this relationship in more detail.

This work is discussed in detail in the PhD Thesis of Daniel Ruiz [4] and in the reports and papers [1], [2], and [3].

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] Arioli, M., Duff, I. S., Ruiz, D., and Sadkane, M. (1992). Techniques for accelerating the Block Cimmino method. In *Proceedings of Fifth SIAM Conference on Parallel Processing for Scientific Computing.* J. Dongarra, K. Kennedy, P. Messina, D.C. Sorensen, R.G. Voigt (editors). SIAM Press, 98-104.

[3] Arioli, M., Duff, I. S., Ruiz, D., and Sadkane, M. (1992). Block Lanczos techniques for accelerating the Block Cimmino method. Report TR/PA/92/70, CERFACS, Toulouse.

[4] Ruiz, D. (1992). Solution of large sparse unsymmetric linear systems with a block iterative method in a multiprocessor environment. Ph D Thesis TH/PA/92/6, CERFACS, Toulouse.

2.16 Stopping criteria for iterative methods (M. Arioli, I.S. Duff, and D. Ruiz)

An important aspect of any iterative method is the stopping criterion which decides when to terminate the iteration. Common criteria are to use the residual or, to achieve some amount of scaling independence, the decrease in residual over the initial residual.

We propose using for the convergence criterion the scaled residual, defined by

$$\frac{\|\mathbf{A}\mathbf{x}^{(k)}-\mathbf{b}\|_\infty}{\|\mathbf{A}\|_\infty\|\mathbf{x}^{(k)}\|_1+\|\mathbf{b}\|_\infty}$$

at step *k*, a small value for which means that the algorithm is normwise backward stable [2].

We have conducted extensive experiments comparing this measure with others as a stopping criterion for our Block Cimmino method (see Section 2.15) and have found that, when the above measure is used as a termination criterion, we usually exit before stagnation and after convergence has been obtained [1].

References

[1] Arioli, M., Duff, I. S., and Ruiz, D. (1992). Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. and Applics.* **13**, 138-144.

[2] Oettli, W. and Prager, W. (1964). Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numerische Math.* **6**, 405-409.

2.17 Computing eigenvalues of large unsymmetric matrices (J. A. Scott)

We are interested in computing a few eigenvalues and the corresponding eigenvectors of a large sparse real unsymmetric matrix. This problem is of considerable practical importance since it has a significant number of applications in scientific and engineering computing, including mathematical models in economics, Markov chain modelling of queuing networks, and bifurcation problems. Release 11 of the Harwell Subroutine Library includes a code `EB12` by Duff and Scott [1] which uses a subspace iteration algorithm, optionally combined with Chebychev acceleration, to find either the eigenvalues of largest modulus or the right-most (or left-most) eigenvalues of a sparse unsymmetric matrix. Since numerical experiments described in [2], [3], and [4] indicate that Arnoldi methods can be more effective

than subspace iteration methods in computing the outermost part of the spectrum, we designed a code which offers Arnoldi based methods. This code is called `EB13.` It will be included in Release 12 of the Harwell Subroutine Library.

A key feature of `EB13` is that it offers the user a choice of Arnoldi based methods:

(1) The basic (iterative) Arnoldi method.

(2) Arnoldi's method with Chebychev acceleration of the starting vectors.

(3) Arnoldi's method applied to the preconditioned matrix $p_l(\mathbf{A})$, where p_l is a Chebychev polynomial.

Each method is available in blocked and unblocked form. This flexibility is needed because it is not always possible to advise a user which method will give the best performance for a particular problem on a given machine. The methods are described in detail by Scott [5]. This report also presents experimental results for a range of practical problems. The performances of `EB12` and `EB13` are compared and it is seen that, for straightforward problems, subspace iteration compares well with the Arnoldi methods but, for problems where the wanted eigenvalues are unfavourably clustered, a block Arnoldi method is needed.

One of the main design features of both `EB12` and `EB13` is the use of reverse communication, so that the user is not required to supply the matrix **A** explicitly but whenever a matrix-vector product is required, control is returned to the user. This allows full advantage to be taken of the sparsity and structure of **A**, and of vectorization and parallelism. It also gives the user greater freedom in choosing how to store the matrix **A** and allows for cases where the matrix is not explicitly available but only the action of **A** on vectors is known. In addition, it allows flexibility in using the code to compute other parts of the spectrum using a shift-and-invert technique or to work with $\phi(\mathbf{A})$ in place of **A**.

References

[1] Duff, I. S. and Scott, J. A. (1993). Computing selected eigenvalues of large sparse unsymmetric matrices using subspace iteration. *ACM Trans. Math. Softw.* **19**, 137-159.

[2] Garratt, T. J. (1991). The numerical detection of Hopf bifurcation in large systems arising in fluid mechanics. Ph.D. thesis, University of Bath.

[3] Saad, Y. (1980). Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. *Linear Alg. and its Applics.* **34**, 269-295.

[4] Saad, Y. (1984). Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Math. Comp.* **42**, 567-588.

[5] Scott, J. A. (1993). An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices. Report RAL-93-097, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

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

Release 1 of the Harwell-Boeing test collection was finally launched in 1992 with the publication of the User's Guide [2] and the announcement of the availability of the collection by anonymous ftp (on directory pub/harwell_boeing on machine orion.cerfacs.fr (138.63.200.33)). The collection and its history was discussed in [1].

Release 1 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] and [2]. 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.

The matrices are widely used in testing, verifying and comparing algorithms and codes in sparse matrix research and MATLAB M-files have recently been made available by Cleve Moler so they can be easily accessed by MATLAB users. Although they have been an invaluable tool in sparse matrix research, there are several new application areas that have become important since this first set was selected. In addition, some traditional areas are generating much larger matrices than the current largest in the collection (order 44609 with 1029655 entries). We already have collected many additional matrices from a wide variety of sources and we plan, in the near future, to construct and distribute Release 2 of the collection.

References

[1] Duff, I. S., Grimes, R. G., and Lewis, J. G. (1989). Sparse matrix test problems. *ACM Trans. Math. Softw.* **15**, 1-14

[2] 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.

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

The Basic Linear Algebra Subprograms (BLAS) [2],[3],[7] 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, [4]).

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 an associated exercise, Mike Heroux of Cray Research has developed support tools for the User Level BLAS, described in the report [6].

The report [5] has been very widely distributed and many people have used these routines in the construction of sparse iterative solvers. Code for the User Level BLAS is available from Carlo Vittoli of IBM Sardinia (vittoli@VNET.IBM.COM).

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] Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. (1990). A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **16**, 1-17 and 18-28.

[3] Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. J. (1988). An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **14**, 1-17 and 18-32.

[4] 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.

[5] 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.

[6] Heroux, M. A. (1992). A proposal for a sparse BLAS toolkit. SPARKER Working Note #2. Report TR/PA/92/90, CERFACS, Toulouse.

[7] Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.* **5**, 308-323 and 324-325.

3 Research in Optimization

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

The most important, and most time-consuming, activity in optimization research during the period of this progress report has concerned the release of the large-scale nonlinear optimization package, LANCELOT. The problem of interest is that of finding the smallest value of a nonlinear objective function $f(\mathbf{x})$ of a large number of unknowns $\mathbf{x}$, where these unknowns are restricted by a set of linear and/or nonlinear constraints. The design goals have been to build an efficient, standard Fortran 77 implementation of a provably convergent algorithm ([1], [2]) for this problem, which is capable of solving problems involving, say, ten thousand unknowns in a reasonable time on a current workstation. The package was released in May 1992, and the accompanying user's manual [4] was published by Springer Verlag the following month. To date, around 150 sites have requested the package. There are currently versions for DEC VMS, ULTRIX and OSF/1, SUN SUNOS, CRAY UNICOS, HP-UNIX, and IBM VM/CMS, AIX and DOS systems.

The problem is presented to the package by the user in a Standard Input Format [3], which is subsequently translated into data and Fortran subroutines. These are then compiled and linked with the minimization procedures prior to the minimization.

The basic algorithm combines, in an augmented Lagrangian function, the objective function and the set of all constraints other than simple bounds on the variables. A sequence of problems is solved, in which the current augmented Lagrangian function is approximately minimized within the region defined by the simple bounds (see [2], [5]).

The algorithm [1] for solving the bounded problem combines a trust region approach, adapted to handle the bound constraints, projected gradient techniques and special data structures to exploit the group partially separable structure of the underlying problem [6]. The software additionally provides

– direct and iterative linear solvers, for the appropriate approximations to Newton's equations which arise at every iteration of the computation,

– a variety of preconditioning and scaling algorithms for more difficult problems,

– quasi-Newton and Newton-type methods, and

– provision for analytical and finite-difference derivatives.

References

[1] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1988). Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, **25**, 433-460. See also same journal **26**, 764-767, 1989.

[2] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1991). A globally convergent augmented

Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis,* **28**, 545-572.

[3] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1991). An introduction the the Standard Data Input Format (SDIF) for nonlinear mathematical programming problems. Technical Report 91/8, FUNDP, Namur, Belgium.

[4] 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.

[5] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds. In D. F Griffiths and G. A. Watson, editors, *Proceedings of the 14th Biennial Numerical Analysis Conference Dundee 1991,* Longmans.

[6] 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,* 42-54, SIAM, Philadelphia, USA.

3.2 Exploiting the structure of partially separable functions (A.R. Conn, M. Daydé, J.–Y. L'Excellent, N.I.M. Gould, and Ph.L. Toint)

Most functions *f*, of a large number of variables **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*** [10]. 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 [10] so this decomposition is extremely general. Note that the decomposition is not unique.

The algorithm which lies behind the LANCELOT package [6] is especially designed for partially separable objective and constraint functions, although the degree to which the partial separability is actually exploited is somewhat limited. As we contemplate what additional features to include in Release B of the package, we are considering how better to exploit such structure.

There are two possibilities. Firstly, new algorithms may be designed specifically to handle partially separable functions [4]. Secondly, the algebraic processes within existing algorithms may be redesigned to cope with such structure. It is this second issue which interests us here.

As it tends to be linear-algebraic costs which dominate the performance of LANCELOT A [5], improvements here are important.

At the heart of an iteration of the LANCELOT algorithm SBMIN ([1], [2]), – which aims to minimize $f(\mathbf{x})$ – we seek 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 $\mathbf{H}_i = \nabla_{xx} f_i(\mathbf{x})$. We would normally consider solving these equations using a preconditioned conjugate gradient method (see, for example, [9]). This method requires a sequence of matrix-vector products of the form

$$\mathbf{w} = \left(\sum_{i=1}^{m} \mathbf{H}_i\right)\mathbf{v} = \sum_{i=1}^{m} (\mathbf{H}_i \mathbf{v})$$

for given input $\mathbf{v}$. In [7], we consider grouping and regrouping elements in order to speed up the execution of these products. We see, on realistic examples, that significant gains may sometimes be made by a simple reordering algorithm. Clearly, there is also much scope for vectorization and parallelization of these products [12].

The other important ingredient of a successful conjugate gradient method for large-scale computation is the construction of an effective preconditioner. A preconditioner is an easily invertible approximation, $\mathbf{P}$, to $\mathbf{H} = \sum_{i=1}^{m} \mathbf{H}_i$. As $\mathbf{H}$ has special structure, we would hope to be able to construct appropriate preconditioners which mimic the structure of $\mathbf{H}$ or which use the structure in some other way. In [8], we consider a number of existing and new element-by-element preconditioners, originally designed for the finite-element solution of partial differential equations ([11], [13]), and report that these preconditioners appear to be extremely effective in the optimization context. All of the proposed methods are highly parallelizable and thus hold promise for many modern architectures.

References

[1] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1988). Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, **25**, 433-460. See also same journal **26**, 764-767, 1989.

[2] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1988). Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, **50**, 399-430.

[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*, 42-54, SIAM, Philadelphia, USA.

[4] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). Convergence properties of minimization algorithms for convex constraints using a structured trust region. Report 92-069, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[5] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. Report 92-067, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[6] 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.

[7] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1993). Improving the decomposition of partially separable functions in the context of large-scale optimization: a first approach. Technical Report TR/PA/93/16, CERFACS, Toulouse, France.

[8] Daydé, M., L'Excellent, J.–Y., and Gould, N. I. M. (1994). On the use of element-by-element preconditioners to solve large scale partially separable optimization problems. Technical Report (in preparation), Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[9] Golub, G. H., and Van Loan, C. F. (1989). *Matrix Computations.* Johns Hopkins University Press, Baltimore, second edition.

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

[11] 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.

[12] Saludjian, L. (1993). Etude d'une version parallele de LANCELOT dans l'environnement de programmation par transferts de messages *PVM*. Rapport de Stage de Troisieme Annee, INPT-ENSEEIHT, Toulouse, France.

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

3.3 Computing search directions for large-scale linearly-constrained nonlinear optimization calculations. (M. Arioli, T.F. Chan, I.S. Duff, N.I.M. Gould, and J.K. Reid)

We consider solving the problem

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

subject to a set of m independent linear equations

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

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

Problems of the form (1)–(2) sometimes arise in their own right (see, for instance, the collection by [2]), but arise more commonly as subproblems within more general nonlinear programming calculations (see, for example, [3] and [5]). Although these latter subproblems may at first appear to have a different form from (1)–(2), it is often possible to convert them to such a form so that algorithms here are appropriate.

In [1], 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 $\mathbf{A}\mathbf{x}_e = \mathbf{b}$, we exploit the structure of the Newton equations

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

for a correction $\mathbf{p}$ to $\mathbf{x}_e$. When $f(\mathbf{x})$ is not convex, these Newton equations may not be appropriate and we discuss suitable replacements for $\nabla_{xx} f(\mathbf{x})$ in (3).

In the first part of paper [1], we consider only unconstrained problems. We analyse methods for modifying the Hessian matrix which use negative curvature information computed during a Lanczos or conjugate gradient iteration for solving the Newton equations. We show how to ensure adequate curvature with a bounded perturbation and that no perturbation is made when the Hessian is sufficiently positive definite. Furthermore, we demonstrate that the technique based on modifying the conjugate gradient iteration chooses good search directions in practice.

In the second part of the paper, we examine several approaches to the solution of structured linear systems of the form (3) which arise as subproblems in optimization calculations. One approach directly eliminates the constraints. The resulting unconstrained problem may then be solved using the iterative techniques proposed in the first part of the paper. We also explore direct factorization techniques and describe stable factorization methods which, if necessary, modify $\nabla_{xx} f(\mathbf{x})$ so as to satisfy the second-order sufficiency condition while preserving sparsity.

We conclude that for a robust, general purpose solver, it is likely that a combination of these methods will be required. A software package incorporating many of these ideas is currently under development [4].

References

[1] Arioli, M., Chan, T. F., Duff, I. S., Gould, N. I. M., and Reid, J. K. (1993). Computing a search direction for large-scale linearly-constrained nonlinear optimization calculations. Report RAL 93-066, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[2] Bongartz, I., Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1993). CUTE : Constrained and Unconstrained Testing Environment. Technical Report TR/PA/93/10, CERFACS, Toulouse, France.

[3] Conn, A. R., Gould, N. I. M., Sartenaer, A., and Toint, Ph. L. (1993). Global convergence properties of two augmented Lagrangian algorithms for optimization with a combination of general equality and linear constraints. Technical Report TR/PA/93/26, CERFACS, Toulouse, France.

[4] Gould, N. I. M. (1994). KKTSOL / MA49 : A set of Fortran subroutines for solving sparse symmetric sets of linear equations which arise in constrained optimization calculations. Technical Report (in preparation), Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[5] Murtagh, B. A., and Saunders, M. A. (1978). Large-scale linearly constrained optimization. *Mathematical Programming*, **14**, 41-72.

3.4 Augmented Lagrangian and barrier function methods for large-scale nonlinear optimization (A.R. Conn, N.I.M. Gould, A. Sartenaer, and Ph.L. Toint)

The first release of the LANCELOT large-scale nonlinear programming package, LANCELOT A [9], occurred in May 1992. The package was a compromise between what could be achieved in the time available and what we actually wanted to achieve. Thus, at that time, we were already aware of some of the shortcomings of the implemented algorithm. Much of our work since then has been directed towards significant improvements, with the ultimate objective of a second release of the package.

Our research has been proceeding in a number of directions. The first has been a direct consequence of our acknowledgement that the method used in LANCELOT A to handle inequality constraints, $c_i(\mathbf{x}) \geq 0$, is inefficient. The traditional method for handling such constraints is to introduce extra *slack* variables, s_i to replace the inequality constraints by the equation $c_i(\mathbf{x}) = s_i$ and the simple bound $s_i \geq 0$. The principal argument in favour of this approach is that it is easier to deal with inequalities when they are simple bound constraints

than when they apply to more complicated functions. The counter argument is that the introduction of extra unknowns increases the dimension of the space that must be searched for a solution. There are thus two possible remedies.

Firstly, we might accept that an increase in dimension is possible but exploit the fact that the slack variables only appear linearly in the reformulation to keep the dominant linear algebraic costs to a minimum. This approach is considered in [7].

Secondly, we may decide that it is unacceptable to treat inequalities via slack variables and prefer to handle them directly. If we choose the latter course, we abandon the use of augmented Lagrangian terms to treat the inequality constraints and instead prefer barrier terms. Traditional barrier functions (see, for example, [11]) have long been used to solve inequality constrained optimization problems, but suffer from disadvantages of requiring feasible starting points and severe numerical conditioning difficulties. An alternative is to attack the problem

$$\underset{\mathbf{x}\in\mathbf{R}^n}{\text{minimize}}\, f(\mathbf{x}) \quad \text{subject to} \quad \mathbf{c}(\mathbf{x}) \geq \mathbf{0}$$

by a sequential minimization of the *Lagrangian barrier function*

$$\Psi(\mathbf{x},\mathbf{w},\mathbf{s}) \equiv f(\mathbf{x}) - \sum_{i=1}^{m} w_i \log(c_i(\mathbf{x}) + s_i),$$

where the components w_i and s_i of the vectors **w** and **s** are positive and, respectively, known as weights and shifts. A scheme for adjusting the weights and shifts to ensure that such a sequential minimization is successful is given in [6], its asymptotic convergence properties are analysed in [8] and its effectiveness is demonstrated in [10].

The next feature we considered undesirable in LANCELOT A is the lack of distinction between linear and nonlinear constraints. Many real applications contain a mixture of linear and nonlinear restrictions on the allowable parameter values. In LANCELOT A, we effectively treat all (non simple-bound) constraints as if they were nonlinear. We would much prefer to keep the linear constraints apart. For instance, we might consider solving the nonlinear program

$$\underset{\mathbf{x}\in\mathbf{R}^n}{\text{minimize}}\, f(\mathbf{x}) \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{and} \quad \mathbf{c}(\mathbf{x}) = \mathbf{0}$$

by a sequential minimization of the *augmented Lagrangian* ([12], [13]) subproblem

$$\underset{\mathbf{x}\in\mathbf{R}^n}{\text{minimize}}\, \Phi(\mathbf{x},\mathbf{w},\mathbf{s}) \equiv f(\mathbf{x}) - \sum_{i=1}^{m} w_i (c_i(\mathbf{x}) + s_i)^2 \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b},$$

where again the **w** and **s** are weights and shifts. Notice, we treat the linear constraints explicitly and thus need good algorithms for solving linearly constrained optimization problems (see [1]). Moreover, because different constraints may have widely different scalings, it is important that the shifts are capable of dealing effectively with potential

numerical difficulties which might result. An analysis of the global and local convergence of such a method has been carried out by [2] and [3], in which a variety of stopping rules for the solution of each linearly constrained optimization calculation are addressed. This work is now being extended to mixtures of linear and nonlinear inequality constraints ([4] and [5]).

A side issue which arose in this investigation is that, when solving a sequence of barrier function subproblems, one would imagine that the solution to one subproblem would be a good estimate of that to the next so long as the weights and shifts are well behaved. Indeed this is so. Thus one would naively expect that a traditional Newton iteration started from the solution of the previous subproblem would be an fast and effective method for solving the current subproblem. This turns out not to be the case, as full Newton steps are prevented by the singularity on the shifted-constraint boundary. However, a more careful analysis reveals that the traditional Newton method is one of an infinite class of Newton methods for an extended problem and, moreover, other members of the class avoid the defects present in the traditional method. Analysis and supporting numerical evidence are given in [10] and the ideas here form the basis of the new HSL quadratic programming routine VE14.

Other issues which are of concern to the design of LANCELOT B are described elsewhere in this progress report.

References

[1] Arioli, M., Chan, T. F., Duff, I. S., Gould, N. I. M., and Reid, J. K. (1993). Computing a search direction for large-scale linearly constrained nonlinear optimization calculations. Report RAL 93-066, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[2] Conn, A. R., Gould, N. I. M., Sartenaer, A., and Toint, Ph. L. (1993). Global convergence properties of two augmented Lagrangian algorithms for optimization with a combination of general equality and linear constraints. Technical Report TR/PA/93/26, CERFACS, Toulouse, France.

[3] Conn, A. R., Gould, N. I. M., Sartenaer, A., and Toint, Ph. L. (1993). Local convergence properties of two augmented Lagrangian algorithms for optimization with a combination of general equality and linear constraints. Technical Report TR/PA/93/27, CERFACS, Toulouse, France.

[4] Conn, A. R., Gould, N. I. M., Sartenaer, A., and Toint, Ph. L. (1994). A globally convergent Lagrangian barrier algorithm for optimization with a combination of general inequality and linear constraints. Technical Report (in preparation), Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[5] Conn, A. R., Gould, N. I. M., Sartenaer, A., and Toint, Ph. L. (1994). Local convergence properties of a Lagrangian barrier algorithm for optimization with a combination of general inequality and linear constraints. Technical Report (in preparation), Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[6] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. Report 92-067, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. England.

[7] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). A note on exploiting structure when using slack variables. Report 92-074, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[8] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear inequality constraints and simple bounds. Report 92-068, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX.

[9] 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.

[10] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1993). A note on using alternative second-order models for the subproblems arising in barrier function methods for minimization. Technical Report TR/PA/93/17, CERFACS, Toulouse, France. to appear *Numerische Mathematik.*

[11] Fiacco, A. V., and McCormick, G. P. (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques* . J. Wiley and Sons, New-York.

[12] Hestenes, M. R. (1969). Multiplier and gradient methods. *Journal of Optimization Theory and Applications,* **4**, 303-320.

[13] Powell, M. J. D. (1969). A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization,* 283-298, Academic Press, London and New York.

3.5 The testing of optimization software (I. Bongartz, A.R. Conn, N.I.M. Gould, and Ph.L. Toint)

The job of a researcher in numerical analysis does not end when a new method for solving a problem of interest has been proposed and analysed. Indeed, the job has hardly begun. For until the proposed algorithm has been implemented and tested, it is not clear that any of its perceived advantages really manifest themselves in the rough-and-tumble world of floating-point computation. Furthermore, it is essential that new proposals are demonstrably superior to existing methods on at least a well-defined class of problems.

Numerical optimization is no exception. Every year, hundreds of apparently new algorithms are proposed for solving specific classes of linear and nonlinear programming problems. With a few notable exceptions, such methods are rarely tested in a competitive environment. Perhaps the author will implement a prototype code and run a few tests on

examples which, although challenging twenty years ago, can be solved in less than a second on today's desk-top computers. The author will then move on to the next project with the mistaken belief that the method has somehow been validated by the extremely limited tests. Of course, such researchers should not be entirely blamed as they are merely indulging in the same habits as 95 percent of their fellow researchers.

When developing the LANCELOT software package [5], the authors quickly became aware of the inadequacies of proper testing facilities for nonlinear optimization algorithms. They thus started collecting as many example problems as possible; this task was initially helped by the availability of existing, small-scale, test suites of [7], [9] and others, and the collection being developed by [1]. It quickly became apparent that the simple task of correctly specifying test problems was time consuming and prone to error. With this is mind, [4] and [5] proposed a Standard Input Format (SIF) for nonlinear programming problems which made the specification and checking of large-scale problems easier. This format has subsequently become the most convenient way of presenting a specific problem to LANCELOT. The test set has grown and now includes over 600 different examples, many of which depend on a variety of parameters. Furthermore, as the primary condition-of-use for the LANCELOT package is that the user provides a specimen example from their field of application, the test set will continue to grow and tend to include more "real-life" examples as time goes on.

Although the SIF was originally developed for LANCELOT, it was soon clear that a generic interface to other optimization packages was possible. Thus Fortran subroutines were written which, for example, return the values of functions, gradients, Jacobians and Hessians, in both dense and sparse formats, for given input values. The utility of these programs is apparent as there are now interfaces to, for instance, MINOS [10], the IBM Optimization Software Library, UNCMIN [8], TENMIN [3], NPSOL [6], and a variety of Harwell Subroutine Library codes.

The whole collection of SIF test problems, Fortran subroutines, installation scripts for a variety of machines (SUNs, HPs, CRAYs, IBM PCs, RS/6000s and mainframes, DEC VAXs, workstations and ALPHAs) and other interface tools are available by anonymous FTP from Rutherford Appleton Laboratory (in the directory pub/cute on camelot.cc.rl.ac.uk), and FUNDP, Namur, Belgium. Since its release in March 1993, over 200 sites have taken the CUTE (Constrained and Unconstrained Testing Environment) package and as a consequence we have now received a number of interfaces to new optimization codes. A complete description of the scope of the package is given by [2].

References

[1] Averick, B. M., and Moré, J. J. (1991). The Minpack-2 test problem collection. Technical Report ANL/MCS-TM-157, Argonne National Laboratory, Argonne, USA.

[2] Bongartz, I., Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1993). CUTE : Constrained and Unconstrained Testing Environment. Technical Report TR/PA/93/10, CERFACS, Toulouse, France. To appear in *ACM Transactions on Mathematical Software.*

[3] Chow, T., Eskow, E., and Schnabel, R. B. (1990). A software package for unconstrained optimization using tensor methods. Technical Report CU-CS-491-90, Department of Computer Science, University of Colorado, Boulder, USA. To appear in *ACM Transactions on Mathematical Software.*

[4] Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1991). An introduction the the Standard Data Input Format (SDIF) for nonlinear mathematical programming problems. Technical Report 91/8, FUNDP, Namur, Belgium.

[5] 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.

[6] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. (1986). User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming. Technical Report SOL86-2, Department of Operations Research, Stanford University, Stanford, California 94305, USA.

[7] Hock, W., and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes.* Lectures Notes in Economics and Mathematical Systems 187, Springer Verlag, Berlin.

[8] Koontz, J. E., Schnabel, R. B., and Weiss, B. E. (1985). A modular system of algorithms for unconstrained minimization. *ACM Transactions on Mathematical Software,* **11**, 419-440.

[9] Moré, J. J., Garbow, B. S., and Hillstrom, K. E. (1981). Testing unconstrained optimization software. *ACM Transactions on Mathematical Software,* **7**, 17-41.

[10] Murtagh, B. A., and Saunders, M. A. (1987). (1991) MINOS 5. 1 USER'S GUIDE. Technical Report SOL83-20R, Department of Operations Research, Stanford University, Stanford, USA.

4 Applied Mathematics

4.1 Automatic differentiation in Fortran 90 (J.K. Reid)

The simplest form of automatic differentiation involves replacing each elementary operation in a computation by one that carries derivative values along with variable values. For example, if there are two independent variables x_1 and x_2, the operation

$$a = b * c$$

would involve the calculations

$$a = b * c$$
$$\frac{\partial a}{\partial x_1} = \frac{\partial b}{\partial x_1} * c + b * \frac{\partial c}{\partial x_1}$$
$$\frac{\partial a}{\partial x_2} = \frac{\partial b}{\partial x_2} * c + b * \frac{\partial c}{\partial x_2}.$$

Fortran 90 provides a very satisfactory environment for such work, since it allows the definition of a derived data type and for overloading of operators and function calls. To gain some experience, we have taken the `DAPRE` code of Stephens [2], written in Fortran 77, and written Fortran 90 'jacket' procedures and interfaces. We needed to make some changes to Stephens' code, but have kept these to a minimum.

The resulting module allows small changes to be made to a program to convert it to one that calculates the derivatives of variables with respect to other variables. The independent variables and all those whose values may change as the independent variables change must be given the type `TYPE(DERIV)` and preferably initialized with the value `UNDEFINED` on the type declaration statement or in a data statement. The order of derivatives to be calculated must be specified and the independent variables identified.

Variables of `TYPE(DERIV)` may be used freely in scalar arithmetic expressions, assignments, and most of the Fortran 77 intrinsic procedure calls in combination with variables of the same type or of type integer or double precision.

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 the statement

   ```
   USE DAPRE
   ```

2. Change the type of all independent variables and variables whose values vary with the independent variables to `TYPE(DERIV)`.

3. Remove intrinsic declarations for any intrinsic functions called with `TYPE(DERIV)` arguments.

The greatest problem that we have encountered is associated with the lack in Fortran 90 of a means to specify that all objects of a derived type have a given initial value. As a consequence, it is very difficult to be sure that a variable is no longer needed and can safely be discarded.

So far, we have implemented only the ‘forward’ method, which was sketched above. Where there are a large number of independent variables, the ‘backward’ method [1] may be far more economical. We plan to treat Stevens’ backward code in the same way and hope, further, to investigate other procedures, exploiting our experience with sparse matrix techniques.

References

[1] Speelpenning, B. (1980). Compiling fast partial derivatives of function given by algorithms. Ph. D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

[2] Stephens, B.R. (1991). Automatic differentiation as a general purpose numerical tool. Ph. D. Thesis, School of Mathematics, University of Bristol.

5 Research in Numerical Linear Algebra

5.1 Computational kernels on vector and parallel machines and on RISC architectures (J.L. Charles, M.J. Daydé, I.S. Duff, P. Morère, and A. Petitet)

We use the BLAS kernels as one of the main mechanisms for achieving portability and efficiency in the design of numerical software. Therefore, the tuning and the parallelization of the BLAS (mainly Level 3 BLAS) and their use in the design of linear solvers (both full and sparse) is an important aspect of our research and is also an important element in the benchmarking of new computers.

Since we wish to keep abreast of current trends in computer architecture, our current research is oriented towards massively parallel computers and distributed memory architectures. We are also concerned with developing efficient BLAS for RISC processors because of the use of this type of processing element in both workstations and high-performance multiprocessors.

We have developed [4] a version of the Level 3 BLAS using a blocking strategy that is suitable as a platform for implementing both serial and parallel versions of the Level 3 BLAS [5]. Our approach is similar to that adopted by [3]. All the Level 3 BLAS kernels are expressed in terms of general matrix-matrix multiplications (GEMM) and operations involving small triangular matrices. The efficiency of this software only relies on the availability of a tuned GEMM. The parallelization is easy and only requires loop-level parallelism (such as CRAY microtasking). This code has been implemented on most of our shared memory target computers (ALLIANT, CRAY-2, CONVEX, IBM 3090-600J). We have used this Level 3 BLAS package for improving the performance of the kernels provided by the ALLIANT FX/80 Scientific Library. We have also been able to improve the uniprocessor performance of the Level 3 BLAS compared with the VECLIB manufacturer-supplied library on the CONVEX C220, while the parallel version of those kernels obtained using microtasking has been tested on the CRAY-2. Finally, these codes have been used as a platform for the implementation of the uniprocessor version of Level 3 BLAS on the BBN TC2000 and are currently used on the IBM RS/6000-950. We also have developed, in collaboration with researchers from ENSEEIHT-IRIT in Toulouse, a complete set of parallel Level 3 BLAS kernels for the BBN TC2000 [1]. This work is a logical continuation of previous studies on the implementation of Level 3 BLAS on Transputer networks [2]. A first version of this BLAS package was distributed in September 1991.

We show in Table 5.1.1 the performance of the double precision GEMM on different computers with various numbers of processors. Note that our parallel Level 3 BLAS uses the serial manufacturer-supplied versions of GEMM on all the computers except on the BBN TC2000 where we use our tuned Fortran version. The parallel version of GEMM on the ALLIANT FX/80 is the one from the ALLIANT Scientific Library Libalgebra v5.0.

Computer	1 proc	4 procs	8 procs	16 procs	24 procs
ALLIANT FX/80	12	–	82		
BBN TC2000	2.6	9.6	18.8	37.4	47.0
CRAY-2	449	1680			
IBM 3090-600J	105	398			

Table 5.1.1. Performance in MFlop/s of GEMM on shared memory multiprocessors using 512-by-512 matrices.

Finally, the block Level 3 BLAS approach is currently being used to develop codes that can be used as a platform for the uniprocessor implementation of the Level 3 BLAS on RISC processors such as those available on the SUN SPARC 10/41 and the HP 730 workstations. We show, in Table 5.1.2, the performance of preliminary tuned versions of the Level 3 BLAS kernels DGEMM and DTRSM, written by Daydé, compared with the performance of their standard Fortran implementation on square matrices of equal size. A library of these high performance BLAS is available on our Suns at RAL.

Workstation	Computational kernel	Order of matrices			
		32	64	96	128
HP 730	DGEMM standard	13.5	11.8	11.5	11.9
HP 730	DGEMM tuned	13.5	21.3	22.3	23.5
HP 730	DTRSM standard	6.5	10.5	10.4	10.7
HP 730	DTRSM tuned	6.5	17.5	25.3	24.7
SUN SPARC 10/41	DGEMM standard	13.5	9.7	8.9	9.2
SUN SPARC 10/41	DGEMM tuned	13.5	17.7	18.8	19.2
SUN SPARC 10/41	DTRSM standard	6.5	5.8	5.7	5.9
SUN SPARC 10/41	DTRSM tuned	6.5	17.5	17.7	16.8

Table 5.1.2. Performance in MFlop/s of the block implementation of DGEMM and DTRSM on RISC workstations.

References

[1] Amestoy P. R., Daydé M. J., Duff I. S., and Morère P. (1992). Linear algebra calculations on the BBN TC2000. In *Parallel Processing: CONPAR 92-VAPPV* Editors L. Bougé, M. Cosnard, Y. Robert, and D. Trystram. Lecture Notes in Computer Science **634**, Springer-Verlag, Berlin, Heidelberg, New York, and Tokyo, 319-330.

[2] Berger, P., Daydé, M. J., and Morère, P. (1991). Implementation use of Level 3 BLAS

kernels on a Transputer T800 ring network. Report TR/PA/91/54, CERFACS, Toulouse.

[3] Dackland, K., Elmroth, E., Kågström, B., and Van Loan, C. (1992). Parallel block matrix factorizations on the shared-memory multiprocessor IBM 3090 VF/600J. *Int. J. Supercomputer Applics.* **6**, 69-97.

[4] Dayde M. J., Duff I. S., and Petitet A. (1993). A parallel block implementation of Level 3 BLAS kernels for MIMD vector processors. Report RAL 93-037, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX. *ACM Trans. Math. Softw.* To appear.

[5] Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. (1990). A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **16**, 1-17.

5.2 Efficient implementation of linear algebra codes on distributed memory machines (M.J. Daydé and I.S. Duff)

The LAPACK project [2] is an international project to develop new routines in numerical linear algebra. One particular emphasis was to develop block matrix factorization algorithms that can take advantage of the Level 3 BLAS.

The public release of the LAPACK library is available on most of our computing platforms. We have studied efficient implementations of this library on the BBN TC2000 using the parallel BLAS package discussed in Section 5.1 (see [1]).

		Number of processors					
Code	Precision	1	2	4	8	16	24
GETRF	single	5.4	10.1	18.1	31.1	49.1	60.6
	double	2.0	3.7	6.7	11.3	18.8	24.1
POTRF	single	6.1	11.6	21.4	38.6	67.6	91.3
	double	2.3	4.4	8.3	14.8	25.9	35.9

Table 5.2.1. Performance in MFlop/s of the LU and the Cholesky factorizations on the BBN TC2000 using 2000-by-2000 matrices.

Table 5.2.1 shows the performance of the LAPACK codes corresponding to the blocked LU factorization (GETRF, right-looking variant), and the blocked Cholesky factorization (POTRF, top-looking variant) on the BBN TC2000. All the matrices are declared as shared, interleaved, copyback.

References

[1] Amestoy P. R., Daydé M. J., Duff I. S., and Morère P. (1992). Linear algebra calculations on the BBN TC2000. In *Parallel Processing: CONPAR 92-VAPPV* Editors L. Bougé, M.

Cosnard, Y. Robert, and D. Trystram. Lecture Notes in Computer Science **634**, Springer-Verlag, Berlin, Heidelberg, New York, and Tokyo, 319-330.

[2] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. (1992). *LAPACK Users' Guide.* SIAM, Philadelphia.

5.3 Pipelining the factorization of full linear systems on multiprocessors (P.R. Amestoy, M.J. Daydé, I.S. Duff, and T. Omnès)

We consider the parallelization of the right-looking variants of the **LU** and the Cholesky factorizations for full linear systems on multiprocessors with globally addressable shared memory. Our aim is to develop a portable pipelined code that can easily be adapted to any shared or virtual shared memory computer. First, a timing prediction model for the pipelined **LU** factorization on an ideal shared memory computer was developed and studied.

On both shared and distributed memory multiprocessor computers with a memory hierarchy, block algorithms are used for efficiency. Daydé and Duff (see [3]) have shown that one simple way to exploit parallelism while maintaining code portability is to use parallel Level 3 BLAS. A blocked factorization involves Level 3 BLAS operations that correspond to the update of a submatrix, and factorizations of a block column or a diagonal block respectively in the **LU** and Cholesky factorizations, performed using Level 1 and Level 2 BLAS. Using parallel Level 3 BLAS, the factorization of a block column in the **LU** factorization is then the serial part of the factorization while the updates of submatrices are parallelized. One way to improve this approach in the **LU** factorization is to overlap the Level 1 and Level 2 BLAS unblocked factorization from the (k+1)–th step with the Level 3 BLAS updates arising from the k–th step.

This is called pipelining. Pipelining algorithms consists in this overlapping calculations (factorization of a block and subsequent updates) over one or several steps of the blocked factorization. We define the depth of our pipeline as the maximum number of steps of the factorization simultaneously overlapped during the calculation. With this convention, no pipelining corresponds to a depth equal to 1.

To some extent, pipelining has already been studied by Daydé and Duff who overlapped one step of the blocked factorization (see [3]) and by Dackland, Elmroth, Kågström and Van Loan who have considered an unlimited depth pipelined strategy on the IBM 3090 VF/660J (see [2]). It is also the most commonly used strategy for implementing parallel versions of the **LU** and the Cholesky factorizations on distributed memory computers.

Amestoy, Daydé, Duff, and Omnès have developed a pipelined version of the LU factorization for shared memory multiprocessors with a strategy similar to the one described in [1] and [3]. This approach allows the efficient implementation of classical factorization techniques (LU, Cholesky, and QR factorizations) on parallel computers with a large number

of processors. The main advantage of this technique is that the load balancing problem is handled naturally.

In our first experiments we used the shared memory 8-processor ALLIANT FX/80. The implementation of a pipelined version of the **LU** factorization on shared memory computers is relatively easy, while on virtual shared memory computers, the data locality is an additional parameter that plays an important role (see [1]). We present these results in Table 5.3.1 where we compare the pipelined LU factorization with the block LU factorization using the parallel BLAS available in the manufacturer-supplied library.

	Number of processors					
Matrix order	1	4	6	8	Speedup	Parallel BLAS
200	5.50	11.10	10.75	10.31	1.87	32.55
400	7.41	23.22	27.33	28.13	3.80	50.56
800	10.03	35.30	46.63	52.68	5.25	59.70
1600	11.28	41.62	58.52	71.71	6.36	65.63

Table 5.3.1. Performance in MFlops of pipelined LU factorization compared to LU factorization using parallel BLAS on the ALLIANT FX/80.

Using pipelining, we plan to develop a complete set of LAPACK-type routines for the solution of full and banded linear systems on distributed memory multiprocessors and on networks of computers using the PVM software. The PCI CS2 will be one of the main target machines for this implementation.

References

[1] Amestoy P. R., Daydé M. J., Duff I. S., and Morère P. (1992). Linear algebra calculations on virtual shared memory computers. Report TR/PA/92/89, CERFACS, Toulouse.

[2] Dackland, K., Elmroth, E., Kågström, B., and Van Loan, C. (1992). Parallel block matrix factorizations on the shared-memory multiprocessor IBM 3090 VF/600J. *Int. J. Supercomputer Applics.* **6**, 69-97.

[3] Daydé, M. and Duff, I. S. (1991). Use of Level 3 BLAS in **LU** factorization in a multitasking environment on three vector multiprocessors, the CRAY-2, the IBM 3090 VF, and the Alliant FX/80. *Int. J. Supercomputer Applics.* **5** (3), 92-110.

6 Miscellaneous Activities

6.1 CERFACS (I.S. Duff and N.I.M. Gould)

The Group has developed even closer collaborations with CERFACS in Toulouse over the period of this report. Iain has continued to lead a project there on Parallel Algorithms and many of the contributions to this report reflect interactions with that team. In particular, three students who were jointly supervised by Iain obtained their Ph Ds during this period: Patrick Amestoy on multifrontal methods on parallel computers [1], Daniel Ruiz on block iterative methods [4], and Chiara Puglisi on a multiprocessor multifrontal implementation of sparse QR factorization [3] and several articles result from this work.

Iain was also involved with CERFACS in a commercial contract to study the parallelization of industrial codes used by Aérospatiale, an industrial partner of CERFACS (see Section 6.2).

Our ties with CERFACS were made even stronger by the visit of Nick to the Parallel Algorithm Project for all of 1994. While he was there, Nick continued to work on the LANCELOT package, developed an understanding and appreciation for parallel processing (including the use of heterogeneous networks of workstations), and started research projects in optimization with people at CERFACS and ENSEEIHT, the local university of science and engineering. We have recently obtained a bilateral grant to continue joint work in this latter area. Jennifer also visited CERFACS during the period to participate in a workshop on eigenvalue problems in the summer of 1993. Plans are now afoot for a major sparse matrix meeting in the Toulouse region in 1994, which it is hoped all of the Group will attend.

The main projects at CERFACS are still Parallel Algorithms and Computational Fluid Dynamics, whose leader is now Thierry Poinsot. The Climate Modelling and Post Processing and Visualization Projects (leaders: Olivier Thual and Jacques David, respectively) continue to flourish and an Electromagnetics Project, led by Patrick Joly, has been recently established. There are two fledgling potential future projects being developed in Computational Chemistry and Shape Optimization. The Director of CERFACS is currently Roland Glowinski. A fuller account of recent activities at CERFACS can be found in [2].

References

[1] Amestoy, P. R. (1991). Factorization of large unsymmetric sparse matrices based on a multifrontal approach in a multiprocessor environment. PhD Thesis, Report TH/PA/91/2, CERFACS, Toulouse.

[2] CERFACS Scientific Report 1992 – 1993. CERFACS, Toulouse. November 1993.

[3] Puglisi, C. (1993). **QR** factorization of large sparse overdetermined and square matrices using a multifrontal method in a multiprocessor environment. Ph D Thesis. report TH/PA/93/33, CERFACS, Toulouse.

[4] Ruiz, D. (1992). Solution of large sparse unsymmetric linear systems with a block iterative

method in a multiprocessor environment. Ph D Thesis TH/PA/92/6, CERFACS, Toulouse.

6.2 Performance analysis of parallel computers on a benchmark of application software from Aérospatiale (M.J. Daydé, I.S. Duff, L. Giraud, and J-Y L'Excellent)

The aim of this contract between Aérospatiale, CERFACS, ENSEEIHT, and RAL was to evaluate the performance of a range of parallel computers: ALLIANT FX/80, ALLIANT FX/2800, BBN TC2000, CRAY-2, CONVEX C220, CONVEX C3800, and networks of workstations on three codes from Aérospatiale. These codes concern aerodynamics and structural engineering.

		Initial Version		Tuned Version	
Code	Computer	Vect/scal	Vect/par	Vect/scal	Vect/par
NS2D	ALLIANT FX/2800(8)	3250	996	–	–
	CONVEX C220(2)	1631	1151	1113	832
	CONVEX C3880(8)	–	–	358	227
	CRAY-2(4)	236	–	236	172
	CRAY-XMP(1)	231	–	–	–
	IBM RS/6000-950	–	–	1020	–
FP3D	ALLIANT FX/80(8)	8391	–	6766	2202
	CONVEX C220(2)	1691	–	808	580
	CONVEX C3880(8)	–	–	278	165
	CRAY-XMP(1)	361	–	–	–
	CRAY-2(4)	140	–	148	77
VFF0042	ALLIANT FX/80(8)	5100	–	1717	326
	BBN TC2000 (19)	–	–	2994	202
	CONVEX C220(2)	1919	–	216	119
	CONVEX C3880(8)	1858	–	86	17
	CRAY-2(4)	624	–	45	15
	CRAY-XMP(1)	590	–	–	–
	IBM RS/6000-320(8)	–	–	622	93
	Cluster RS/6000(3)	–	–	235	89

Table 6.2.1. Execution time in seconds of the initial and the tuned versions.

The first part of this study was the evaluation of the cost of porting codes to the target

computers. The second part concerned the performance we obtained with tuning and parallelizing these codes on the target machines. The execution times reported on the CRAY-XMP were supplied by Aérospatiale.

We give in Table 6.2.1 a brief summary of the results we have obtained [1]. The number of processors used for each computer is given in parenthesis. We give the performance of the scalar or the vectorized version of each code depending on whether the processors are vector or superscalar processors and the performance of the parallel version of the code. The parallel versions of the code use loop-level parallelism on ALLIANT, CRAY, and CONVEX computers, BBN Fortran extensions on the BBN TC2000, and the PVM message-passing package on networks of IBM RS/6000 RISC workstations (using Ethernet with the IBM RS/6000-320 and SOCC with the cluster of RS/6000).

Full details of codes and techniques are given in [1].

References

[1] Dayde M. J., Duff I. S., L'Excellent, J. Y., et Giraud, L. (1993). "Evaluation d'ordinateurs vectoriels et parallèles sur un jeu de programmes représentatifs des calculs intensifs à la division avions de l'Aérospatiale": Rapport final Rapport de CERFACS FR/PA/93/19.

6.3 Fortran 90 (J.K. Reid)

John Reid terminated his membership of X3J3, the ANSI Fortran Standardization Committee at the end of 1991, but has played an active role by email in the interpretation of the Fortran 90 standard and attended all meetings in the reporting period of the ISO Fortran Committee, JTC1/SC22/WG5 (WG5, for short). He has actively promoted the language with the paper [3] and by speaking at many venues, mainly in the UK but also abroad. He has given 3-day courses at the Atlas Centre for users of the Atlas facilities and these have been very well received by the participants. He was also invited to two meetings of the Cray Fortran Advisory Board.

John Reid and Mike Metcalf have kept their book *Fortran 90 explained* up to date with interpretations by revisions in three reprintings over the period. John has commented [2] on proposals for parallel extensions.

Several vendors provided optimizing compilers towards the end of 1993 and we now have access to compilers from Nag (site licence), EPC (SUN workstation), Cray YMP, and Digital (beta test on Alpha platform). We are looking forward to being able to use the language for our numerical software. Our codes will be much more convenient for the user than is possible in Fortran 77 and there will be substantial safety enhancements. Of particular benefit will be dynamic storage, assumed-shape arrays, optional arguments,

structures, and pointers. The argument list will be substantially shorter and will be checked at compile time.

Both WG5 and X3J3 are beginning to think about the next revision of Fortran 90. The plan is for a small revision in 1995, consisting only of editorial enhancements to the document and a few urgently needed features, and a larger revision in 2000. John Reid is the principal author of a proposal [1] by IFIP Working Group 2.5 for exception handling. The Group believes that such a feature is essential for the construction of robust software and would like this to be part of the 1995 revision. John has made very good progress towards a simple, powerful feature that does not disturb the rest of the language and is acceptable to X3J3.

If an operator invokes a process (in hardware or in a procedure for a defined operator) and hits a problem with which it cannot deal, such as overflow, it needs to quit and ask the caller to do something else. A simple example of the proposal is

```
ENABLE (OVERFLOW)
   :
   ... = X*Y ...
HANDLE
   :
END ENABLE
```

If the multiply is intrinsic and an overflow occurs, a transfer of control is made to the block of code following the `HANDLE` statement. Similarly, if the multiply is a defined operator, it can be arranged that `OVERFLOW` signals in comparable circumstances. The handle block may contain very carefully written code that is slow to execute but circumvents the problem, or may arrange for graceful termination.

References

[1] IFIP Working Group 2.5 (1993). The enable construct for exception handling. SIGNUM Newsletter **29**, 7-16.

[2] Reid, J.K. (1992). On PCF parallel Fortran extensions. ACM Fortran Forum, **11**, 17-23.

[3] Reid, J.K. (1992). The advantages of Fortran 90. *Computing* **48**, 219-238.

6.4 IFIP Working Group 2.5 (J.K. Reid)

John Reid remains a member of IFIP Working Group 2.5 (numerical software), and he attended all the meetings held during the reporting period.

The 1991 meeting was held in Karlsruhe, 23-29 September, and mainly consisted of a working conference on *Programming environments for high-level scientific problem solving.* Proceedings [1] have been published.

The 1992 meeting was held in Toronto, 17-19 September. The most noteworthy event of this meeting was a session on the draft international standard for language independent arithmetic at which we were addressed by one of the principal authors, Craig Shaffert. Craig was unable to convince the group of the utility of this standard, but the members undertook to discuss its pros and cons and work towards either the correction or the elimination of the proposal.

The 1993 meeting was held in Copenhagen, 21-23 October, and highlighted a technical symposium on *Numerical Software* organized in conjunction with UNI•C, the Danish Computing Centre for Research and Education.

John's main interaction with the Group during the period has concerned Fortran. He has kept the group up to date with developments concerning the Fortran standard and related activities such as PCF (Parallel Computing in Fortran) and HPF (High Performance Fortran) and has led the Group's activity on exception handling in Fortran (see Section 6.3).

Reference

[1] Gaffney, P.W. and Houstis, E.N., eds. (1992). Programming environments for high-level scientific problem solving. Proceeding of IFIP TC2/WG 2.5 working conference. North-Holland, Amsterdam, New York, and London.

7 Computing and Harwell Subroutine Library

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

Over the past three years, we have gradually been replacing or augmenting our SUN Sparc 1 workstations with more up-to-date machines. Nick was the first to benefit from new equipment when he received a colour-screen IBM Risc Systems/6000 320H in mid 1991. This machine was about twenty times faster than its predecessor and was instrumental for testing LANCELOT in the final days before its release. However, it has to be admitted that the RS/6000 has not been as reliable as we hoped, and the rest of the group opted instead to buy current colour-screen SUN equipment. Iain's Sparc 10/30 arrived in late 1992, while Jennifer and John's 10/30s were delivered at the end of 1993. We are delighted with the performance of these new SUNs. Iain's old Sparc 1 has been resurrected at home, while John and Jennifer's machines were traded for their successors. The remaining three Sparc 1s have benefited from increased memory and extra SCSI disks and continue to be used by our short and medium-term visitors.

The system administrative responsibilities have to some extent been delegated to the CCD Unix system support team. We decided that we were unhappy with having to rely on a central fileserver and the ethernet for key operating system activities, and have now opted to have a core unix kernel, and additional heavily used applications, on each machine. We have all taken this opportunity to move from SUNs proprietary windowing system to an X-based system. Other enhancements include access to object libraries of BLAS, LAPACK and Harwell Subroutine Library codes and to a growing range of Fortran 90 compilers. We also now allow anonymous FTP access to a variety of group products, including all of our recent technical reports and the CUTE database of optimization test examples, described in Section 3.5.

We have also benefited from other CCD machines, in particular the IBM RS/6000, the DEC Alpha and HP farms, and the CRAY Y-MP. Jennifer and Nick have both explored using the network as a distributed parallel machine, using the PVM message-passing system [1]. We are now able to mail to and from our workstations and thus can avoid data-conversion problems with the IBM 3090 mainframe. Regular system backups are taken via the CCD IBM virtual tape reader, and our storage capacity has recently been enlarged thanks to the departmental UNIX data store.

References

[1] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1993). PVM 3 User's Guide and Reference Manual. Report ORNL/TM-12187, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Tennessee.

7.2 Release 11 of the Harwell Subroutine Library (I.S. Duff, N.I.M. Gould, M.J. Hopper, J.A. Scott, and J.K. Reid)

The Group continues to act as consultants for the Harwell Subroutine Library and we have had a very harmonious collaboration with Tony Harker and Libby Thick of Harwell. Release 11 of the Library was made in July 1993. This contained the following significant new routines:

`EB12` Subspace iteration for the unsymmetric eigenvalue problem.

`MA42` Frontal method for unsymmetric sparse linear equations.

`MA43` Simple interface to `MA42` for equation entry with no auxiliary storage.

`MA47` Multifrontal solution of symmetric equations with zeros on diagonal.

`MA48` Solution of unsymmetric sparse sets of equations, with easy-to-use interface and computation of block triangular form.

`MA50` Solution of unsymmetric sparse sets of equations that have been ordered and without computation of block triangular form.

`MC30` Scaling factors for a sparse symmetric matrix.

`ME42` Complex version of `MA42`.

`ME43` Complex version of `MA43`.

`ME47` Complex version of `MA47`.

`ME48` Complex version of `MA48`.

`ME50` Complex version of `MA50`.

`MF30` Complex version of `MC30`.

`MF41` Complex version of `MC41`.

`VE14` Large sparse quadratic programming problem with simple bounds.

We employed Mike Hopper as a consultant to address comments that we have received from users on some of the older routines. He set about the task with his customary thoroughness and efficiency and developed some Unix tools to automate the process, including one that purposely unsets non-saved variables on entry to a procedure in order to check whether a `SAVE` statement is needed. He made revisions to about 80 routines, constructing new versions when the changes made this necessary, usually because an additional argument was needed. He also provided advice and assistance with computer typesetting.

The documentation is now fully electronic and we have replaced the single bound A5 volume of specifications by a pair of A4 volumes. It is a pity to be using so much more paper, but the new format is far easier to read and pleasant to handle.

With this release, we have introduced three categories of outdated routines:

Replaced. A change to the argument list has been forced for strict conformance to the standard, but no significant changes made. Such a routine is still present in the distributed library but not in the documentation except for a simple catalogue entry that says it has been replaced and by what. We may decide to exclude it from the next release.

Superseded. A redesigned routine does the same job better. The distributed library and documentation will include it, but not the index. We may decide to exclude it from the next release.

Deprecated. An alternative exists in LAPACK [1]. The tape and documentation will include it, and it will be in the index. We may decide to exclude it from a future release.

Reference

[1] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. (1992). LAPACK users' guide. SIAM, Philadelphia.

7.3 Release 12 of the Harwell Subroutine Library (J.K. Reid)

The next release is currently planned for October 1995. We employed Mike Hopper as a consultant to apply his Unix tools to the remaining routines of the library and he has completed this task for all but a very small number of routines for which the work would have been inappropriate.

Our plan is that this release will contain a few Fortran 90 modules, using a very similar style for the documentation as presently used for Fortran 77 packages. The names will be of the form `HSL_lldd` where `l` stands for a letter and `d` stands for a digit.

7.4 New Library subroutines

The following routines were added to the Library during this period.

`EB12` (I. S. Duff and J. A. Scott)

Given a real unsymmetric $n \times n$ matrix $\mathbf{A} = \{a_{ij}\}$, this routine uses subspace iteration to calculate the r eigenvalues λ_i, $i = 1, 2, ..., r$, that are right-most, left-most, or are of largest modulus. The right-most (respectively, left-most) eigenvalues are the eigenvalues with the most positive (respectively, negative) real part. A second entry will return the associated eigenvectors $\mathbf{y}_i$, $i = 1, 2, ..., r$, where $\mathbf{A}\mathbf{y}_i = \lambda_i \mathbf{y}_i$. The routine may also be used to calculate a group of eigensolutions elsewhere in the spectrum.

EB13 (J. A. Scott)

This routine uses Arnoldi based methods to compute the r right-most eigenvalues λ_i, $i = 1, 2, ..., r$, of a real unsymmetric $n \times n$ matrix $\mathbf{A} = \{a_{ij}\}$. The right-most eigenvalues are those with the most positive real part. There is an option to compute the associated eigenvectors $\mathbf{y}_i$, $i = 1, 2, ..., r$, where $\mathbf{A}\mathbf{y}_i = \lambda_i \mathbf{y}_i$. The routine may be used to compute the left-most eigenvalues of **A** by using –**A** in place of **A**.

The Arnoldi methods offered by EB13 are:

(1) The basic (iterative) Arnoldi method.

(2) Arnoldi's method with Chebychev acceleration of the starting vectors.

(3) Arnoldi's method applied to the preconditioned matrix $p_l(\mathbf{A})$, where p_l is a Chebychev polynomial.

Each method is available in blocked and unblocked form.

MA42 (I. S. Duff and J. A. Scott)

This set of subroutines solves one or more sets of sparse linear equations, $\mathbf{A}\mathbf{x} = \mathbf{b}$ or $\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$, by the frontal method, optionally using direct access data sets for the matrix factors. Use is made of high level BLAS kernels. The code has low in-core memory requirements. The matrix **A** may be input by the user in either of the following ways:

(i) by elements in a finite-element calculation,

(ii) by equations (matrix rows).

MA43 (J. A. Scott)

This subroutine solves one or more sets of variable-band linear equations, $\mathbf{A}\mathbf{x} = \mathbf{b}$ or $\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$ by the frontal method. MA43 provides the user with a straightforward interface to the Harwell Subroutine Library routine MA42 when entry is by equations and auxiliary storage is not required. If the user requires more sophisticated facilities, MA42 should be employed.

MA47 (I. S. Duff and J. K. Reid)

Given a sparse symmetric indefinite matrix **A** and a vector **b**, this subroutine solves the system $\mathbf{A}\mathbf{x} = \mathbf{b}$. It aims to take advantage of any zeros on the diagonal.

MA48 (I. S. Duff and J. K. Reid)

This solves a sparse unsymmetric system of m linear equations in n unknowns using block triangularization and Gaussian elimination. There are facilities for choosing a good pivot order, factorizing a matrix with a given pivot order, and solving a system of equations using the factorized matrix. Error estimates are available.

MA50 (I. S. Duff and J. K. Reid)

This solves a general sparse $m \times n$ system of linear equations (the most usual case being square, $m = n$), assuming that the matrix entries have been ordered by columns. Neither block triangularization nor error estimation is included.

MC30 (J. K. Reid)

This calculates scaling factors for a symmetric sparse matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$. They may be used, for instance, to scale the matrix prior to solving a corresponding set of linear equations, and are chosen so that the scaled matrix has its entries near to unity in the sense that the sum of the squares of the logarithms of the entries is minimized. The natural logarithms of the scaling factors s_i, $i = 1, 2, ..., n$ for the rows and columns are returned so that the scaled matrix has entries

$$b_{ij} = a_{ij} \exp(s_i + s_j).$$

ME42 (I. S. Duff and J. A. Scott)

This set of subroutines solves one or more sets of sparse linear complex equations, $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$, or $\mathbf{A}^{\mathrm{H}}\mathbf{x} = \mathbf{b}$, by the frontal method, optionally using direct access data sets for the matrix factors. ($\mathbf{A}^{\mathrm{H}}$ denotes the conjugate transpose of $\mathbf{A}$). Use is made of high level BLAS kernels. The code has low in-core memory requirements. The complex matrix $\mathbf{A}$ may be input by the user in either of the following ways:

(i) by elements in a finite-element calculation,

(ii) by equations (matrix rows).

ME42 is the complex version of MA42.

ME43 (J. A. Scott)

This set of subroutines solves one or more sets of sparse linear complex equations, $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$, or $\mathbf{A}^{\mathrm{H}}\mathbf{x} = \mathbf{b}$, by the frontal method ($\mathbf{A}^{\mathrm{H}}$ is the conjugate transpose of $\mathbf{A}$). ME43

provides the user with a straightforward interface to the Harwell Subroutine Library routine ME42 when entry is by equations and auxiliary storage is not required. If the user requires more sophisticated facilities, ME42 should be employed. ME43 is the complex version of MA43.

ME47 (I. S. Duff and J. K. Reid)

This is a complex version of MA47.

ME48 (I. S. Duff and J. K. Reid)

This is a complex version of MA48.

ME50 (I. S. Duff and J. K. Reid)

This is a complex version of MA50.

MF30 (J. K. Reid)

This is a complex version of MC30.

MF41 (J. A. Scott)

This subroutine estimates the 1-norm ($\max_j \sum_{i=1}^{n} |a_{ij}|$) of an $n \times n$ complex matrix **A** given the ability to multiply a vector by both the matrix and its conjugate transpose. Because the explicit form of **A** is not required, the subroutine can be used for estimating the norm of matrix functions such as the inverse. Additionally this subroutine is potentially useful for estimating condition numbers of a matrix when the matrix is sparse or not available explicitly. MF41 is the complex version of MC41.

VE14 (N. I. M. Gould)

This subroutine solves the general, large, quadratic programming problem

$$\text{minimize} \quad f(\mathbf{x}) = \tfrac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} h_{jk}\, x_j\, x_k + \sum_{j=1}^{n} c_j\, x_j + q\ ,$$

within a feasible region defined by simple bound constraints,

$$b_{1,i} \le x_i \le b_{2,i}, \quad 1 \le i \le n,$$

using a barrier function method. Either of the bounds $b_{1,i}$ and $b_{2,i}$ may be infinite. The matrix $\mathbf{H}$ (h_{jk}) is assumed to be symmetric but may be indefinite. Full advantage is taken of any zero coefficients h_{jk}. If the matrix **H** is positive semi-definite, a global solution is found. If, however, **H** is indefinite, the procedure may find a local solution which is not the global solution to the problem.

8 Seminars

28 February 1991	M. J. D. Powell (Cambridge) Interpolation to functions of one variable by multiquadrics
7 March 1991	D. J. Evans (Loughborough) Parallel algorithms for solving large sparse linear systems
9 May 1991	A. Spence (Bath) Stable matrices and Hopf bifurcations in large systems, with application to problems in fluid mechanics
6 June 1991	J. Pryce (RMCS, Shrivenham) Reliable computation of Sturm-Liouville resonances
17 September 1991	R. B. Schnabel (Colorado, U.S.A.) A new modified Cholesky factorization and new methods for solving large sparse systems of nonlinear equations
4 October 1991	T. Taniguchi (Okayama, Japan) Delaunay triangulation for an arbitrary 2D domain
31 October 1991	C. Budd (Bristol) Convergent and spurious solutions of nonlinear elliptic PDEs
28 November 1991	C. Greenough (Rutherford) The simulation of silicon semiconductor devices
6 February 1992	B. Christanson (Hatfield) Accurate Hessians by reverse accumulation
5 March 1992	S. W. Ellacott (Brighton) The numerical analysis of artificial neural computing systems
18 March 1992	I. D. Coope (Canterbury, New Zealand) Curve interpolation with nonlinear splines
14 May 1992	C. P. Thomson (AEA Technology, Harwell) A parallel adaptive multigrid algorithm for the Navier Stokes equations
9 June 1992	J. K. Reid (Rutherford) Fortran 90 for scientific programming
11 June 1992	J. Gregory (Brunel) Recursive subdivision generation of curves and surfaces
29 October 1992	M. Bartholomew-Biggs (Hertfordshire)

	Spacecraft ascent trajectory optimisation using Ada
26 November 1992	T. Chambers (National Physical Laboratory) Benchmarking of parallel processors
25 January 1993	M. C. Ferris (Wisconsin, U.S.A.) Robust solution of mixed complementarity problems
4 February 1993	S. Hammarling (NAG) The numerical solution of the generalized symmetric eigenvalue problem
4 March 1993	N. J. Higham (Manchester) Parallel solution of triangular systems
10 June 1993	A. Cliffe (AEA Technology, Harwell) Stability of flow in an expanding pipe
28 October 1993	A. C. Damhaug (Det Norske Veritas, Oslo) Finite element interpretation of sparse matrix preprocessing
25 November 1993	V. Swift (Portsmouth) The application of the elementary structure of Class A in the solution of sparse linear equations

9 Reports issued in 1991-93

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 91-041 Numerical Analysis Group – Progress Report. January 1989 – December 1990. I.S. Duff (Editor).

RAL 91-056 Computing selected eigenvalues of sparse unsymmetric matrices using subspace iteration. I.S. Duff and J.A. Scott.

RAL 91-057 Stopping criteria for iterative solvers. M. Arioli, I.S. Duff, and D. Ruiz.

RAL 92-017 The advantages of Fortran 90. J K Reid.

RAL 92-066 Large-scale nonlinearly constrained optimization. N I M Gould. A. R. Conn, N. I. M. Gould and Ph. L. Toint.

RAL 92-067 A globally convergent shifted barrier algorithm for optimization with general inequality constraints and simple bounds. A. R. Conn, N. I. M. Gould and Ph. L. Toint.

RAL 92-068 On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear inequality constraints and simple bounds. A. R. Conn, N. I. M. Gould and Ph. L. Toint.

RAL 92-069 Convergence properties of minimization algorithms for convex constraints using a structured trust region. A. R. Conn, N. I. M. Gould and Ph. L. Toint.

RAL 92-074 A note on exploiting structure when using slack variables. A. R. Conn, N. I. M. Gould and Ph. L. Toint.

RAL 92-075 Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. A. R. Conn, N. I. M. Gould and Ph. L. Toint.

RAL 92-086 Users' Guide for the Harwell-Boeing Sparse Matrix Collection. I. S. Duff, R. G. Grimes, and J. G. Lewis.

RAL 92-087 A proposal for user level sparse BLAS. SPARKER Working Note #1. I.S. Duff, M. Marrone, and G. Radicati.

RAL 93-036 An unsymmetric-pattern multifrontal method for sparse LU factorization. T.A. Davis and I.S. Duff.

RAL 93-037 A parallel block implementation of Level 3 BLAS for MIMD vector processors. M.J. Dayde, I.S. Duff, and A. Petitet.

RAL 93-064 MA42 – A new frontal code for solving sparse unsymmetric systems. I.S. Duff and J.A. Scott.

RAL 93-066 Computing a search direction for large-scale linearly-constrained nonlinear optimization calculations. M. Arioli, T.F. Chan, I.S. Duff, N.I.M. Gould, and J.K. Reid.

RAL 93-072 MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations. I.S. Duff and J.K. Reid.

RAL 93-073 A mathematical model of a biosensor. S. Jones, B. Jumarhon, S. McKee and J.A. Scott.

RAL 93-084 The solution of augmented systems. I.S. Duff.

RAL 93-097 An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices. J.A. Scott.

Harwell Reports

HARWELL SUBROUTINE LIBRARY. A Catalogue of Subroutines (Release 10). April 1992.

HARWELL SUBROUTINE LIBRARY. A Catalogue of Subroutines (Release 11). June 1993.

HARWELL SUBROUTINE LIBRARY. Release 11. Specifications. Volume 1 and Volume 2. July 1993. June 1993.

CERFACS Reports

TR/PA/91/63 Techniques for accelerating the Block Cimmino method. M. Arioli, I.S. Duff, D. Ruiz, and M. Sadkane.

TR/PA/92/69 Linear algebra calculations on the BBN TC2000. P.R. Amestoy, M.J. Daydé, I.S. Duff, and P Morère.

TR/PA/92/70 Block Lanczos techniques for accelerating the Block Cimmino method. M. Arioli, I.S. Duff, D. Ruiz, and M. Sadkane.

TR/PA/92/83 Memory allocation issues in sparse multifrontal methods on multiprocessors. P.R. Amestoy and I.S. Duff.

PR/PA/92/10 Evaluation d'ordinateurs vectoriels et parallèles sur un jeu de programmes représentatifs des calculs intensifs à la division avions de l'Aérospatiale: bilan de l'étape de portage. M.J. Daydé, I.S. Duff, J.Y. L'Excellent, et L. Giraud.

TR/PA/93/10 CUTE: Constrained and Unconstrained Testing Environment. I. Bongartz, A.R. Conn, N.I.M. Gould, and Ph.L. Toint.

TR/PA/93/16 Improving the decomposition of partially separable functions in the context of large-scale optimization: a first approach. A.R. Conn, N.I.M. Gould, and Ph.L. Toint.

TR/PA/93/17 A note on using alternative second-order models for the subproblems arising in barrier function methods for minimization. A.R. Conn, N.I.M. Gould, and Ph.L. Toint.

PR/PA/93/19 Evaluation d'ordinateurs vectoriels et parallèles sur un jeu de programmes représentatifs des calculs intensifs à la division avions de l'Aérospatiale: Rapport final M.J. Daydé, I.S. Duff, J.Y. L'Excellent, et L. Giraud.

TR/PA/93/26 Global convergence of two augmented Lagrangian algorithms for optimization with a combination of general equality and linear constraints. A.R. Conn, N.I.M. Gould, A. Sartenaer, and Ph.L. Toint.

TR/PA/93/27 Local convergence properties of two augmented Lagrangian algorithms for optimization with a combination of general equality and linear constraints. A.R. Conn, N.I.M. Gould, A. Sartenaer, and Ph.L. Toint.

FUNDP, Namur, Belgium Reports

91/8 An introduction the the Standard Data Input Format (SDIF) for nonlinear mathematical programming problems. A.R. Conn, N.I.M. Gould, and Ph.L. Toint.

91/10 A comprehensive description of LANCELOT. A.R. Conn, N.I.M. Gould, and Ph.L. Toint.

92/4 Announcement of the Release A of LANCELOT, a package for nonlinear optimization. A.R. Conn, N.I.M. Gould, and Ph.L. Toint.

92/15 Intensive numerical tests with LANCELOT (Release A): the complete results. A.R. Conn, N.I.M. Gould, and Ph.L. Toint.

10 External Publications in 1991-1993

Amestoy, P. R., Daydé, M., and Duff, I. S. (1991). Designing portable software for linear algebra. *Theoretica Chimica Acta.* **79**, 169-174.

Amestoy, P. R., Daydé, M., and Duff, I. S. (1993). Parallel solution of sparse linear systems using the multifrontal method. Journees du site experimental en hyperparallelisme, 27-28 janvier, 1993, ETCA, Paris. *CREA,* 226-257.

Amestoy P. R., Daydé M. J., Duff I. S., and Morère P. (1992). Linear algebra calculations on the BBN TC2000. In *Parallel Processing: CONPAR 92-VAPPV* Editors L. Bougé, M. Cosnard, Y. Robert, and D. Trystram. Lecture Notes in Computer Science **634**, Springer-Verlag, Berlin, Heidelberg, New York, and Tokyo, 319-330.

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

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.

Arioli, M., Duff, I. S., and Ruiz, D. (1992). Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. and Applics.* **13**, 138-144.

Arioli, M., Duff, I. S., Ruiz, D., and Sadkane, M. (1992). Techniques for accelerating the Block Cimmino method. In *Proceedings of Fifth SIAM Conference on Parallel Processing for Scientific Computing.* J. Dongarra, K. Kennedy, P. Messina, D.C. Sorensen, R.G. Voigt (editors). SIAM Press, 98-104. Also reprinted in Proceedings of One-DAY Workshop on Parallel Numerical Analysis, 21 June 1991. Editors D B Duncan, K I M McKinnon, and F Plab Report EPCC-TR92-05, Edinburgh Parallel Computing Centre, 1-7.

Conn, A. R., Gould, N. I. M., Sartenaer, A., and Toint, Ph. L. (1993). Global convergence of a class of trust region algorithms for optimization using inexact projections on convex constraints. *SIAM Journal on Optimization,* **3**, 164-221.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1991). Convergence of quasi-Newton matrices generated by the symmetric rank one update. *Mathematical Programming,* **50**, 177-196.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1991). A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis,* **28**, 545-572.

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. *XX* + 330 pages. Springer Verlag, Heidelberg, Berlin, New York.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). Large-scale nonlinear constrained optimization. In Jr. R. E. O'Malley, editor, *Proceedings of the Second International Conference on Industrial and Applied Mathematics,* 51-70, SIAM, Philadelphia, USA,

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1992). On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds. In D. F Griffiths and G. A. Watson, editors, *Proceedings of the 14th Biennial Numerical Analysis Conference Dundee 1991,* Longmans.

Conn, A. R., Gould, N. I. M., Lescrenier, M., and Toint, Ph. L. (1993). 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,* Kluwer Academic Publishers.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1993). 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.

Daydé, M. and Duff, I. S. (1991). Use of Level 3 BLAS in **LU** factorization in a multitasking environment on three vector multiprocessors, the CRAY-2, the IBM 3090 VF, and the Alliant FX/80. *Int. J. Supercomputer Applics.* **5** (3), 92-110.

Dongarra, J. J. and Duff, I. S. (1992). Advanced architecture computers. In *Supercomputing in Engineering Analysis.* Edited by H. Adeli. Marcel Dekker Inc., New York, 19-62.

Dongarra, J. J., Duff, I. S., Sorensen, D. C., and Van der Vorst, H. A. (1991). *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM Press.

Duff, I. S. (1991). Parallel algorithms for general sparse systems. In *Computer Algorithms for Solving Linear Algebraic Systems: The State of the Art.* Edited by E Spedicato. NATO ASI Series. Vol **F77**. Springer-Verlag, 277-297.

Duff, I. S. (1992). The solution of large sparse unstructured unsymmetric systems on modern computers. Proceedings European Symposium on Computer Aided Engineering (ESCAPE 2). 5-7 October, 1992, Toulouse, France. Editors D. Depeyre, X. Joulia, B. Koehret, J.M. Le Lann. Supplementary Volume, 83.

Duff, I. S. (1993). Exploitation of parallelism in direct and semi-direct solution of large sparse systems. In *Parallel Computation* edited by A.E. Fincham and B. Ford. Oxford University Press, 159-174.

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., Gould, N. I. M., Reid, J. K., Scott, J. A., and Turner, K. (1991). The factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis,* **11**, 181-204.

Duff, I. S. and Kahaner, D. K. (1991). Two Japanese approaches to circuit simulation. *Scientific Information Bulletin, Office of Naval Research Asian Office, NAVSO, P-3580.* **16** (1), 21-26.

Duff, I. S. and Puglisi, C. (1990). Efficient computation of a matrix-matrix kernel. In *Supercomputing Tools for Science and Engineering.* Edited by Domenico Laforenza and Raffaele Perego. Franco Angeli, N Milan, 541-547.

Duff, I. S. and Scott, J. A. (1992). Computing selected eigenvalues of sparse unsymmetric matrices. *Numerical Linear Algebra. ERCIM Workshop Reports. Pisa, May 21-22, 1992, 53-59.*

Duff, I. S. and Scott, J. A. (1993). Computing selected eigenvalues of sparse unsymmetric matrices using subspace iteration. *ACM Trans. Math. Softw.* **19**, 137-159.

Gould, N. I. M. (1991). An algorithm for large-scale quadratic programming. *IMA Journal of Numerical Analysis,* **11**, 299-324.

Gould, N. I. M. (1991). On growth in Gaussian elimination with complete pivoting. *SIAM Journal on Matrix Analysis,* **12** 354-361.

Reid, J.K. (1991). Fortran 90, the new Fortran standard. .EXE., **6**, 14-18.

Reid, J.K. and other members of IFIP WG 2.5 (1992). Comments on version 3.1 of draft ISO/IEC 10967:1991, Language Compatible Arithmetic. SIGNUM Newsletter **27**, 2-3.

Reid, J.K. (1992). The advantages of Fortran 90. *Computing* **48**, 219-238.

Reid, J.K. (1992). On PCF parallel Fortran extensions. ACM Fortran Forum, **11**, 17-23.

Reid, J.K. (1992). The Fortran 90 standard. In *Programming environments for high-level scientific problem solving,* ed. P. W. Gaffney and E. N. Houstis, IFIP Transactions A-2, North-Holland, 343-348.

Reid, J.K. (1992). The use of Fortran 90 for scientific programming. In *Managing communications in a global marketplace,* proceedings of SHARE Europe Spring meeting, 585-596.

Reid, J.K. (1993). Language Independent Arithmetic – a draft international standard. SIGNUM Newsletter **28**, 2-7.