RAL-TR-2002-010

Numerical Analysis Group Progress Report January 2000 - December 2001

Iain S. Duff (Editor)

ABSTRACT

We discuss the research activities of the Numerical Analysis Group in the Computational Science and Engineering Department at the Rutherford Appleton Laboratory of CLRC for the period January 2000 to December 2001. This work was supported by EPSRC grants M78502, until October 2001, and R46441 thereafter.

Keywords: sparse matrices, frontal and multifrontal methods, numerical linear algebra, large-scale eigenvalue computations, large-scale optimization, automatic differentiation, Fortran, Harwell Subroutine Library, HSL

AMS(MOS) subject classifications: 65F05, 65F50.

Current reports available by anonymous ftp to ftp.numerical.rl.ac.uk in directory pub/reports. This report is in file duffRAL2002010.ps.gz. Report also available through URL `http://www.numerical.rl.ac.uk/reports/reports.html`.

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

March 22, 2002

Contents

Personnel in Numerical Analysis Group

Staff

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

Mario Arioli (from October 1st 2000).
Numerical linear algebra, numerical solution of PDEs, error analysis.

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

Karl Meerbergen (until March 31st 2000).
Large-scale eigenvalue problems.

Jennifer Scott.
Sparse linear systems and sparse eigenvalue problems.

Kath Vann.
Administrative and secretarial support.

Consultants

Mike Hopper Support for Harwell Subroutine Library and for TSSD.

John Reid HSL, sparse matrices, automatic differentiation, and Fortran.

Visitors

Richard Byrd (UC Boulder) Optimization.

Bob Gate (Dundee) Optimization.

Yifan Hu (CLRC Daresbury) Sparse linear systems.

David Kay (Sussex) Eigensystem calculations.

Erricos Kontoghiorghes (Neuchatel) Optimization.

Gianmarco Manzini (CNR Pavia) Iterative methods.

Jorge Nocedal (Northwestern University) Optimization.

Daniel Ruiz (ENSEEIHT) Linear algebra.

Annick Sartenaer (Namur) Optimization.

Philippe Toint (University of Namur) Optimization.

1 Introduction (I. S. Duff)

This report covers the period from January 2000 to December 2001 and describes work performed by the Numerical Analysis Group within the Computational Science and Engineering Department at the CLRC Rutherford Appleton Laboratory. This work was supported by EPSRC grants M78502, until October 2001, and R46441 thereafter.

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

The last two years have been amongst the most stable in the life of the Group. It would be good to think that this state of affairs could continue. Our only change in personnel was that Karl Meerbergen left to return to Belgium and we were delighted to be able to recruit Mario Arioli from CNR Pavia, who had previously been a long term visitor of the Group at Harwell and was a senior scientist in the Parallel Algorithms Team at CERFACS.

The hoops designed by EPSRC were successfully negotiated in March 2000 when we heard that our grant had been extended from two years to four. However, as I write this, the next set of hoops are being prepared for our rebid for continuation of funding from October 2003. In April 2001, Professor John Wood from the University of Nottingham became CEO of CCLRC. We have since been able to meet with him and find him sympathetic both to basic research and to the work of our Group. We believe and hope that this augurs well for the future.

The support and development of HSL, formerly the Harwell Subroutine Library, continues to be one of our major activities. There have been two releases of HSL during the period of this report, one in October 2000 and the other at the end of 2001. The HSL marketing effort from AEA Technology PLC has again seen changes of personnel, this time apparently for the better. Without any prior consultation or warning, Nick Brealey of the Electromagnetics Department at Culham Laboratory ceased handling the Library at the end of 2000. Happily we were taken over within AEA by Lawrence Daniels and his team from Hyprotech, who have a good knowledge of HSL and some marketing skills in mathematical software. Most importantly, they also fully support the availability of HSL to the UK academic community as originally agreed by Nick Brealey. We are still able to employ John Reid as a consultant using HSL funds. We have benefited greatly from the consultancy of Mike Hopper, who has helped us both in typesetting and the ongoing commitment to higher software standards, including a recent exercise in making the whole Library threadsafe.

We maintain our close links with the academic community in Britain and abroad. Iain and Nick continue as Visiting Professors at Strathclyde University and Edinburgh University, respectively. All members of the Group gave presentations at the Dundee

Numerical Analysis meeting in 2001, with Nick giving an invited talk. We had several visitors during the period, including Gianmarco Manzini who was funded by an EPSRC visiting fellowship. Iain has helped Andy Wathen of Oxford in the supervision of his student, Andreas Papadopoulos. Our CASE student with Oxford, Carsten Keller, successfully defended his D Phil and Nick now has another CASE student, Bob Gate, with the University of Dundee whose university supervisor is Roger Fletcher. Iain was on the jury for the PhD thesis of Elisabeth Traviesas in Toulouse, for the cotutelle thesis of Dominique Orban in Namur and Toulouse, for Pierre Ramet in Bordeaux, Wim Bomhof in Utrecht, and for the habilitation theses of Valérie Frayssé and of Luc Giraud in Toulouse, and Nick was the external examiner for the PhD thesis of Eric Chin from Dundee. Nick is also an external assessor for the optimization MSc at the Open University and is on the advisory board for the MSc in Bath. Iain and Nick are both on the Mathematics College of the EPSRC.

We continue our close association with Oxford University through the Joint Computational Mathematics and Applications Seminar series and have hosted several talks at RAL through that programme (see Section 10). Nick Trefethen, the professor of Numerical Analysis at Oxford University, has made an office available to the Group that has been used for visits by all Group members, significantly by Nick who visits on a regular basis. Nick taught an MSc course on numerical optimization at Oxford in Trinity term 2001 (see `http://www.numerical.rl.ac.uk/nimg/oumsc/`), and Iain and Jennifer gave a series of four lectures on direct methods for sparse matrices in Michaelmas term of 2001.

Nick's collaboration with Toint and others continues to expand the theory and practice of large-scale optimization. During the period of this report, his 959 page book, co-authored with Conn and Toint, on trust-region methods was published by SIAM. In addition to the book and closely related research, he has developed algorithms and codes for the solution of large-scale quadratic programming problems, both using barrier-function methods and active-set techniques. Both approaches will be used in the new nonlinearly constrained optimization package GALAHAD, that will eventually supersede the well known and highly successful augmented Lagrangian package LANCELOT. His work with Toint and others has occasioned visits to CERFACS in Toulouse, Namur in Belgium, and Northwestern in Illinois. Nick was an invited speaker at conferences at Atlanta, Delhi, Hans-sur-Leys, Oxford, and Trier, and gave seminars in Daresbury, Edinburgh, Oxford, and Reading. He was on the interview panel for an appointment at Oxford University.

Jennifer has continued with her national and international collaborations. Although she has continued her short-hours working, she remains so productive that it is easy to forget this fact. Much of her work in this period has been on enhancing the performance of frontal solvers and using them in practical industrial problems. In particular, she has

used powerful partitioning techniques and MPI to enable frontal schemes to exploit low level parallelism. She has developed very effective ordering techniques for use with frontal methods both when the matrix is assembled and when it is held as a set of element matrices. She has made a speciality of chemical engineering applications and has published several papers in one of the main journals of that discipline. Jennifer presented an invited talk at CERFACS and continues to coordinate our joint seminar series with Oxford University.

Mario was no stranger when he joined us in October 2000 and so it is no surprise that he quickly became a fully integrated member of the Group. He gives us a far greater knowledge of partial differential equations and has used this to good effect in designing sophisticated techniques and preconditioners for problems in groundwater flow. He is also a great asset in being not only very adept at but also enjoying the fine detail of error analysis. He has recently established and become coordinator of a new ERCIM Working Group on Mathematics. He has given seminars at Cambridge, CERFACS, and ENSEEIHT-IRIT in Toulouse.

Iain still leads a project at the European Centre for Research and Advanced Training in Scientific Computation (CERFACS) at Toulouse in France (see Section 7.1). He was the Principal Investigator for a grant from the France-Berkeley Fund for exchange visits with NERSC in Berkeley. His research interests continue to be in all aspects of sparse matrices, including more recently iterative methods as well as direct methods, and in the exploitation of parallel computers. He is an Editor of the IMA Journal of Numerical Analysis, an Honorary Secretary of the IMA, editor of the IMANA Newsletter, chairman of the IMA Programme Committee, and IMA representative on the International Committee that oversees the ICIAM international conferences on applied mathematics. In high performance computing, he has given tutorials at VECPAR 2000 (Porto), EuroPar 2000 (Munich), SC 2000 (Dallas) and SC 2001 (Denver). He gave lectures at summer schools in Porto and Lyngby in Denmark and was workshop coordinator for a meeting in Copper Mountain, Colorado. He has been on the Programme and Organizing Committee for several international meetings including a preconditioning meeting in Lake Tahoe, California and the two EuroPar meetings. He also helped to edit the proceedings for the Copper Mountain and Tahoe meetings. He has given invited talks at meetings in Maryland, Merida (Mexico), and Rabat (Morocco), and has presented seminars in Cambridge, Lawrence Livermore National Laboratory, NERSC, Oxford, Stanford, and Strathclyde. He was on an international panel to evaluate Dutch supercomputing and to advise NWO about future mechanisms for support of this activity.

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 numerical linear algebra, and nonlinear systems and optimization. We are particularly concerned with large-scale

systems when the matrix or system is sparse or structured. We discuss the solution of linear systems by frontal or multifrontal methods in Section 2 and other numerical linear algebra activities in Section 3. Work on optimization is considered in Section 5. We group some miscellaneous topics in Section 7. Much of our research and development results in high quality advanced mathematical software, and we report on our computer infrastructure and software developments in Section 8. Lists of seminars (in the joint series with Oxford), technical reports, and publications are given in Sections 10, 11, and 12, respectively. Current information on the activities of the Group and on Group members can be found through page `http://www.cse.clrc.ac.uk/Group/CSENAG` of the World Wide Web.

2 Frontal and multifrontal methods

2.1 MUMPS - a distributed memory multifrontal solver (P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent and J. Koster)

MUMPS, a MUltifrontal Massively Parallel Solver was developed from the earlier MUPS code (Amestoy and Duff 1993) with the support of the EU LTR project PARASOL. Although the PARASOL project finished on June 30th 1999, work has continued on the MUMPS solver and many of the original team are still involved in this project. Some aspects of this work were supported until the middle of 2000 by a grant from the France-Berkeley fund. Patrick Amestoy is still at ENSEEIHT-IRIT in Toulouse and continues to supervise or co-supervise students and stagiaires working on aspects of MUMPS. Jean-Yves L'Excellent was unable to get much time for research while at NAG but now is back in the thick of things in his new post with INRIA in Lyon. Jacko Koster has now a confirmed staff position at Parallab in Bergen and continues to develop some aspects of MUMPS, in particular its use within the Parallab domain decomposition code DDM.

MUMPS is designed to solve symmetric positive-definite, general symmetric, and unsymmetric linear systems whose coefficient matrices are possibly rank deficient. The MUMPS package uses a multifrontal approach to factorise the matrix (Duff and Reid, 1983, Duff and Reid, 1984). Similar to serial HSL solvers, the parallel MUMPS package solves in three main steps: an analysis step, a factorization step and a solution step. MUMPS is described in the two papers Amestoy, Duff and L'Excellent (2000) and Amestoy, Duff, Koster and L'Excellent (2001).

MUMPS achieves high performance by exploiting two kinds of parallelism: tree parallelism that comes from the sparsity of the problem and node parallelism from dense matrix kernels. MUMPS uses dynamic data structures and dynamic scheduling of computational tasks to accommodate extra fill-in in the factors due to numerical considerations (not taken into account during the analysis step). This dynamic approach also allows the parallel code to cope with load variations on the processors and we have investigated and developed this over the last two years (see Section 2.1.2). MUMPS overlaps computation with communication by using asynchronous communication and care has to be taken in the MPI implementation (see Section 2.1.1).

The origins of MUMPS within a large EU project with many partners and many demands means that it has a functionality quite unrivaled by any other sparse package. In the last two years, the functionality has only been slightly extended with more emphasis being put on improving the efficiency, particularly when using many processors.

The MUMPS software has been extensively tested on problems from the industrial partners in the PARASOL project. Typical PARASOL test cases are from application

areas such as computational fluid dynamics, structural mechanics, modelling compound devices, modelling ships and mobile offshore platforms, industrial processing of complex non-Newtonian liquids, and modelling car bodies and engine components. The largest problem we have solved to date is a model of an AUDI crankshaft. The corresponding linear system is symmetric positive-definite and of order 943,695 with more than 39 million entries in its lower triangular part. With the best ordering of the unknowns that we tried, MUMPS created 1.4 billion entries in the factors and required 5.9×10^{12} floating-point operations for the factorization.

The MUMPS software is written in Fortran 90. It requires MPI for message passing and makes use of BLAS, LAPACK, BLACS, and ScaLAPACK subroutines. It has been ported to a wide range of computers including the top-line supercomputers from Compaq, Cray, IBM, and SGI.

We discuss the performance of MUMPS further in Section 2.2.

References

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

P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods in Appl. Mech. Engng.*, **184**, 501–520, 2000.

P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Analysis and Applications*, **23**(1), 15–41, 2001

I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.*, **9**, 302–325, 1983.

I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM J. Scientific and Statistical Computing*, **5**, 633–641, 1984.

2.1.1 An analysis of `MPI` send/receive in the context of `MUMPS` (P. R. Amestoy, I. S. Duff, J. Y. L'Excellent and X. S. Li)

This work was developed from the research performed as part of the France-Berkeley project (Amestoy, Duff, L'Excellent and Li, 2000) and is intimately associated with the tuning of the MUMPS and SuperLU sparse direct solvers on distributed memory computers using MPI for message passing.

We examined the send and receive mechanisms of MPI in detail and considered how to implement message passing robustly so that performance is not significantly affected by changes to the MPI system. We discussed this within the context of two different parallel algorithms for sparse Gaussian elimination: a multifrontal solver (MUMPS), and a supernodal one (SuperLU). The performance of our initial strategies based on simple MPI point-to-point communication primitives is very sensitive to the MPI system, particularly the way MPI buffers are used. Using more sophisticated non-blocking communication primitives improves the performance robustness and scalability, but at the cost of increased code complexity.

We have submitted our report (Amestoy, Duff, L'Excellent and Li, 2001) to the journal Parallel Computing.

References

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis, tuning and comparison of two general sparse solvers for distributed memory computers. Technical Report LBNL-45992, NERSC, Lawrence Berkeley National Laboratory, June 2000.

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Impact of the implementation of MPI point-to-point communications on the performance of two general sparse solvers. Technical Report RT/APO/01/4, ENSEEIHT-IRIT, Toulouse, October 2001.

2.1.2 Candidate-based dynamic scheduling for a distributed direct linear solver (P. R. Amestoy, I. S. Duff and C. Vömel)

The asynchronous distributed memory multifrontal solver MUMPS exploits two kinds of parallelism when a sparse matrix is factorized. A first natural source of parallelism is established by independent branches of the assembly tree. Furthermore, tree nodes with a large enough contribution block can be updated in parallel by splitting the update between several slaves of the master that is factorizing the block of fully summed variables, and the root node can be treated in parallel if it is big enough.

While the *master* processor of each node in the tree (that is, the one that is responsible for the factorization of the block of fully summed variables) is chosen during the analysis phase, the *slaves* for the parallel update of large contribution blocks are only chosen during the factorization phase. This dynamic task scheduling takes place in order to balance the work load of the processors at run-time. Problems arise from offering too much freedom to the dynamic scheduling. If every processor is a candidate for a slave then, on each processor, enough workspace has to be reserved during the analysis phase for the corresponding computational tasks. However, during the factorization, typically not all processors are actually needed as slaves (and, on a large number of processors, only a very few are needed),

so the prediction of the required workspace will be overestimated. Thus the size of the problems that can be solved is reduced unnecessarily because of this difference between the prediction and allocation of memory in the analysis phase and the memory actually used during the factorization.

With the concept of *candidate processors* it is possible to address this issue. The concept originates in an algorithm presented by Pothen and Sun (1993) and extends efficiently to MUMPS. For each node that requires slaves to be chosen dynamically during the factorization because of the size of its contribution block, we introduce a limited set of processors from which the slaves can be selected. While the master previously chose slaves from among all less loaded processors, the freedom of the dynamic scheduling can be reduced so that the slaves are only chosen from the candidates. This effectively allows us to exclude all non-candidates from the estimation of workspace during the analysis phase and leads to a more realistic prediction of workspace needed. Furthermore, the candidate concept allows us to structure the computation better since we can explicitly restrict the choice of the slaves to a certain group of processors and enforce a 'subtree-to-subcube' mapping principle.

Preliminary tests with a prototype version have shown the benefits of the concept that is currently being integrated into a compact scheduling module for MUMPS. It unifies static and dynamic mapping while at the same time taking account of tree modifications by node amalgamation and splitting.

References

A. Pothen and C. Sun. A mapping algorithm for parallel sparse Cholesky factorization. *SIAM J. Scientific Computing*, **14**(5), 1253–1257, 1993. Timely Communication.

2.2 Analysis and comparison of distributed memory sparse solvers (P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent and X. Li)

We conducted an in depth analysis comparing the merits of a supernodal solver, SuperLU (Demmel, Gilbert and Li, 1999), with MUMPS (see Section 2.1). This showed broadly that MUMPS generally outperformed SuperLU although the latter showed somewhat better scalability and was competitive on a large number of processors. Many ideas for improvements to both codes were generated during this investigation and both are being enhanced as a result.

We show some of our comparisons in Table 2.1 to indicate the relative performance of the codes. In Table 2.2, we show results from a sequence of 3D grids where the problem

sized is increased with the number of processors so that the operations per processor remain close to constant. From these results we see the good scalability of both codes.

Matrix	Ordering	Solver	Number of processors						
			1	4	8	16	32	64	128
BBMAT	AMD	MUMPS	—	44.8	23.6	15.7	12.6	10.1	9.5
		SuperLU	—	64.7	36.6	21.3	12.8	9.2	7.2
	ND	MUMPS	—	32.1	10.8	12.3	10.4	9.1	7.8
		SuperLU	—	132.9	72.5	39.8	23.5	15.6	11.1
ECL32	AMD	MUMPS	—	53.1	31.3	20.7	14.7	13.5	12.9
		SuperLU	—	106.8	56.7	31.2	18.3	12.3	8.2
	ND	MUMPS	—	23.9	13.4	9.7	6.6	5.6	5.4
		SuperLU	—	48.5	26.6	15.7	9.6	7.6	5.6

Table 2.1: Factorization time (in seconds) of large test matrices on the CRAY T3E. "—" indicates not enough memory.

Processors	Grid size			$\mathbf{LDL}^T$ factorization		$\mathbf{LU}$ factorization			
				MUMPS-SYM		MUMPS-UNS		SuperLU	
	NX	NY	NZ	flops $\times 10^9$	time	flops $\times 10^9$	time	flops $\times 10^9$	time
Cubic grids (nested dissection)									
1	29			3.6	18.9	7.2	24.0	7.2	56.3
2	33			8.0	21.2	16.0	29.0	15.9	61.8
4	36			13.4	20.3	26.8	27.6	26.8	52.0
8	41			30.1	18.3	60.1	32.8	60.0	60.2
16	46			59.1	19.5	118.1	32.6	117.9	59.8
32	51			112.7	21.3	225.3	41.2	224.9	64.7
64	57			222.7	28.4	445.1	57.5	444.7	67.3
128	64			444.2	48.3	887.8	95.7	886.4	71.1
Rectangular grids (nested dissection)									
1	96	24	12	2.2	13.2	4.5	16.6	4.5	33.3
2	110	28	13	4.8	12.9	9.5	17.2	9.6	37.6
4	120	30	15	9.0	12.1	17.9	16.7	17.9	36.3
8	136	34	17	18.4	13.7	36.8	20.1	36.6	36.3
16	152	38	19	36.5	12.5	72.8	21.0	72.7	42.2
32	168	42	21	67.8	14.3	135.5	25.4	135.3	43.8
64	184	46	23	118.2	16.3	236.2	32.5	236.0	46.6
128	208	52	26	243.1	24.7	485.8	44.4	485.6	56.1

Table 2.2: Factorization time (in seconds) on Cray T3E. $\mathbf{LU}$ factorization is performed for MUMPS-UNS and SuperLU, $\mathbf{LDL}^T$ for MUMPS-SYM.

The France-Berkeley Fund, who supported Iain's travel to NERSC at Lawrence Berkeley National Laboratory, accepted the main report (Amestoy et al., 2000) and a shorter version has been published in the ACM Transactions on Mathematical Software (Amestoy, Duff, L'Excellent and Li, 2001*b*). The work was presented at the SIAM Parallel Processing Conference in March 2001 (Amestoy, Duff, L'Excellent and Li, 2001*a*).

References

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis, tuning and comparison of two general sparse solvers for distributed memory computers. Technical Report LBNL-45992, NERSC, Lawrence Berkeley National Laboratory, June 2000.

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Performance and tuning of two distributed memory sparse solvers. *in* 'Proceedings of Tenth SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, Virginia, March 12th-14th, 2001', 2001*a*.

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, **27**(4), 388–421, December 2001*b*.

J. W. Demmel, J. R. Gilbert, and X. S. Li. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination. *SIAM J. Matrix Analysis and Applications*, **20**(4), 915–952, 1999.

2.3 Two-stage ordering for unsymmetric parallel row-by-row frontal solvers (J. A. Scott)

The row-by-row frontal method may be used to solve general large sparse linear systems of equations. By partitioning the matrix into (nearly) independent blocks and applying the frontal method to each block, a coarse-grained parallel frontal algorithm is obtained (see Section 2.4). The success of this multiple front approach depends on preordering the matrix. This can be done in two stages: (1) order the matrix to singly bordered block diagonal form, and (2) order the rows within each block to minimise the size of each frontal matrix. A number of recent papers have considered stage (1). In this study, we looked at developing an algorithm for stage (2).

Consider the singly bordered block diagonal matrix

$$\begin{pmatrix} \mathbf{A}_{11} & & & \mathbf{C}_1 \\ & \mathbf{A}_{22} & & \mathbf{C}_2 \\ & & \ldots & . \\ & & \mathbf{A}_{NN} & \mathbf{C}_N \end{pmatrix}, \tag{2.1}$$

where the rectangular diagonal blocks $\mathbf{A}_{ll}$ are $m_l \times n_l$ matrices with $m_l \geq n_l$, and the border blocks $\mathbf{C}_l$ are $m_l \times k$ with $k \ll n_l$. We want to order the rows within each submatrix

$$\begin{pmatrix} \mathbf{A}_{ll} & \mathbf{C}_l \end{pmatrix} \tag{2.2}$$

so that, when the frontal method is applied to the submatrix, the frontsize is kept as small as possible. The difficulty is that, as the rows of (2.2) are assembled, the k_l nonzero columns of $\mathbf{C}_l$ do not become fully summed because they have entries in at least one other border block $\mathbf{C}_j$ $(j \neq l)$. Thus, once a variable corresponding to a column of $\mathbf{C}_l$ enters the front, it remains there. These variables are termed *interface* variables. A row ordering algorithm is therefore needed that distinguishes between interface and non-interface variables and which aims to delay introducing the former into the front.

Our algorithm generalises the MSRO row ordering algorithm that we developed for ordering **all** the rows of an unsymmetric matrix $\mathbf{A}$ for use with a frontal solver (see Scott (1999)). The MSRO algorithm selects a global ordering which it uses to guides the local reordering. The local ordering is based on a *priority function*. The basic idea is to select the next row in the ordering by choosing, from a set of eligible rows, a row with minimum priority. The priority P_i for row i is given by

$$P_i = -W_1 \; rcgain_i - W_2 \; g_i, \tag{2.3}$$

where W_1 and W_2 are positive weights, g_i is the global priority for row i, and $rcgain_i$ is the sum of the increases to the row and column frontsizes resulting from assembling (ordering) row i next. As each row is reordered, the priorities of the remaining (unordered) rows are updated. In this way, a balance is maintained between a small frontsize and ordering early on rows with a low global priority.

The MSRO algorithm assumes that when any variable appears for the last time it is fully summed and so can be eliminated (removed from the front). To generalise the approach to allow only the rows of a submatrix to be ordered, we flag the interface variables so that once in the front they are not eliminated. In addition, we modify the priority function by adding a third term

$$P_i = -W_1 \; rcgain_i - W_2 \; g_i + W_3 \; nold_i, \tag{2.4}$$

where W_3 is another (positive) weight and $nold_i$ is the number of non-interface variables in row i that have already been introduced into the front. As rows are assembled, $nold_i$ increases, so that rows with a large number of non-interface variables already lying in the front are given preference. The aim is to ensure non-interface variables become fully summed and eliminated as soon as possible after entering the front.

The modified MSRO algorithm has been tested on problems from chemical process engineering; results are presented by Scott (2001). To illustrate the performance, in Figure 2.1 we show the sparsity pattern for problem `bayer04`: using the initially supplied ordering, after it has been reordered to bordered block diagonal form using the HSL code `MC66`, and after the modified MSRO algorithm has been used to reorder the rows within the blocks.

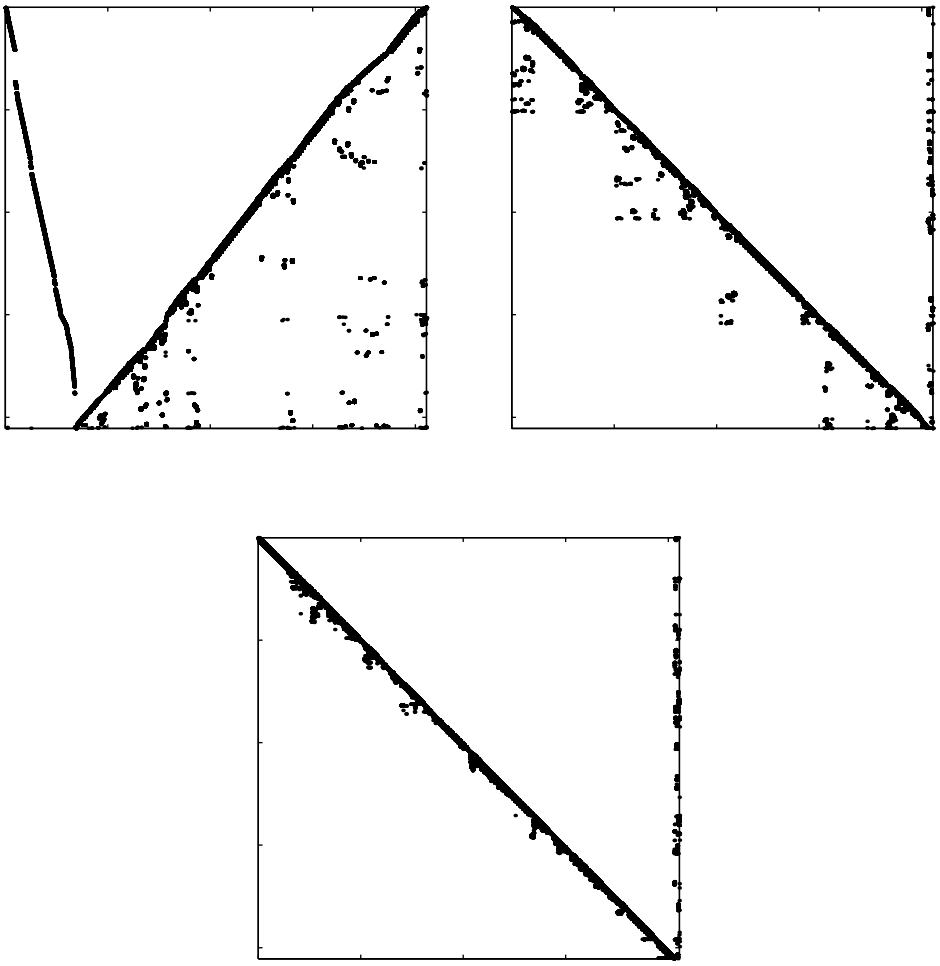


Figure 2.1: Problem `bayer04` with original ordering, in bordered block diagonal form, and after reordering with modified MSRO.

The code MC62, which is available through HSL, has been modified so that it can be used to order either all the rows of **A** or, for the multiple front method, the rows of a submatrix.

References

J.A. Scott. A new row ordering strategy for frontal solvers. *Numerical Linear Algebra and Applications*, **6**, 1–23, 1999.

J.A. Scott. Two-stage ordering for unsymmetric parallel row-by-row frontal solvers. *Computers and Chemical Engineering*, **25**, 323–332, 2001.

2.4 The design of a portable parallel frontal solver for highly unsymmetric linear systems (J. A. Scott)

The row-by-row frontal method is often used for solving the large sparse systems of linear equations that arise in large-scale chemical process simulation and optimization problems. However, for modern computers, a major deficiency of the frontal method lies in its lack of scope for parallelism other than that which can be obtained within the high-level BLAS that are used in the innermost loop of the factorization. The multiple front algorithm aims to overcome this shortcoming by partitioning the problem into a number of nearly independent subproblems and then applying the frontal method to each subproblem in parallel.

We have designed and developed a package MP43 that implements the multiple front algorithm for unsymmetric linear systems that have been preordered to the singly bordered block diagonal form (2.1). A partial **LU** decomposition is performed on each of the submatrices

$$\begin{pmatrix} \mathbf{A}_{ll} & \mathbf{C}_l \end{pmatrix} \tag{2.5}$$

using the frontal method. MP43 performs these decompositions in parallel. As the rows of (2.5) are assembled, n_l variables become fully summed and may be eliminated. These variables correspond to the columns of $\mathbf{A}_{ll}$; the k_l nonzero columns of $\mathbf{C}_l$ do not become fully summed because they have entries in at least one other border block $\mathbf{C}_j$ $(j \neq l)$. Because the $\mathbf{A}_{ll}$ are, in general, rectangular, at the end of the assembly and elimination operations, for each block there will remain a frontal matrix $\mathbf{F}_l$ of order $(m_l - n_l) \times k_l$. The variables that remain in the front are the *interface variables* and the sum $\mathbf{F}$ of these remaining frontal matrices is the *interface matrix*. MP43 also factorizes the $k \times k$ interface matrix using the frontal method. Once $\mathbf{F}$ has been factorized, block forward eliminations and back-substitutions are performed (in parallel) to complete the solution.

MP43 is similar in design to the HSL parallel frontal solvers MP42 and MP62, which implement the multiple front approach for finite-element problems (Scott, 2001*b*). MP43 is written in Fortran 90 and uses MPI for message passing. It may be used on shared or distributed memory machines and may be run on a single processor or on up to N processors, where N is the number of submatrices in the block diagonal form. The interface is designed to be straightforward, with the user required to specify a minimum number of parameters. Essentially, he or she needs only to provide the partitioning of the matrix into submatrices and to input the matrix data in a suitable format. All allocation of workspace, division of work between processors, and ordering of the rows of the submatrices for the frontal solver is done automatically. However, for flexibility, control parameters allow a number of options to be specified. For example, the matrix factors may optionally be held in files, enabling large problems to be solved using relatively small amounts of main memory. Additionally, the user may choose how to divide the submatrices between processors and how to order the rows. The wide range of options are intended to make the code suitable for those with minimal knowledge of the multiple front method and for experts with specific requirements.

Experiments have been performed on a set of test problems and comparisons have been made with the serial frontal solver MA42 and the well known general sparse direct solver MA48. The MC66 implementation of the MONET algorithm (Hu, Maguire and Blake, 2000) was used to preorder the matrices to the form (2.1); in each case the number of submatrices was $N = 8$. In Table 2.3, wallclock timings (in seconds) for MP43 run on $p = 1$, 2, 4 and

Identifier	Order	MA42	MA48	MP43 ($N = 8$)					
				$p = 1$	2		4		8	
10cols	29496	3.64	16.54	1.92	1.05	*1.83*	0.62	*3.10*	0.43	*4.47*
bayer01	57735	8.70	6.52	4.78	2.67	*1.79*	1.74	*2.75*	1.05	*4.42*
icomp	75724	7.61	0.88	4.51	2.45	*1.84*	1.66	*2.72*	0.96	*4.70*
lhr34c	35152	13.56	24.20	15.82	8.52	*1.86*	5.00	*3.16*	3.69	*4.29*
lhr71c	70304	32.86	51.26	35.01	18.92	*1.85*	10.26	*3.41*	7.10	*4.93*
pesa	11738	2.47	1.90	1.22	0.69	*1.77*	0.44	*2.77*	0.32	*3.81*
poli_large	15575	1.87	0.06	0.90	0.53	*1.70*	0.36	*2.50*	0.21	*4.28*
Zhao2	33861	67.54	656.5	45.35	28.89	*1.57*	20.56	*2.21*	18.12	*2.50*

Table 2.3: Timings MP43. The numbers in italics are the speedups for MP43 compared with using a single process.

8 processors of a 12 processor SGI Origin2000 are presented, together with timings for MA42 and MA48 run on a single processor. The timings are for factorizing the matrix and then solving for a single right-hand side. We see that, even on one processor, MP43 can be competitive with the serial codes and, as the number of processors in increased, good

speedups are achieved. More detailed performance results are included in Scott (2001*a*).

References

Y.F. Hu, K.C.F. Maguire, and R.J. Blake. A multilevel unsymmetric matrix ordering for parallel process simulation. *Computers in Chemical Engineering*, **23**, 1631–1647, 2000.

J.A. Scott. The design of a portable parallel frontal solver for chemical process engineering problems. *Computers in Chemical Engineering*, **25**, 1699–1709, 2001*a*.

J.A. Scott. A parallel solver for finite element applications. *Inter. Journal on Numerical Methods in Engineering*, **50**, 1131–1141, 2001*b*.

2.5 A multilevel wavefront and profile reduction algorithm (J. A. Scott and Y. Hu)

If Gaussian elimination is applied to a symmetric positive-definite matrix $\mathbf{A}$ of order n, all zeros between the first entry of a row and the diagonal usually fill in (this happens if rows 2, 3, ... , n all have at least one entry to the left of the diagonal). Therefore, the total number of entries in each triangular factor is the sum of the lengths of the rows of the original matrix, where each length is counted from the first entry to the diagonal. This sum is also known as the **profile**. The sum of the squares of the lengths gives the **wavefront**.

A variety of methods have been proposed for choosing a permutation of the matrix to reduce the profile and wavefront, including the well known Reverse Cuthill-McKee and Sloan algorithms. More recently, spectral orderings based on the Fiedler vector of the Laplacian matrix associated with a matrix have been developed. Kumfert and Pothen (1997) proposed combining an enhanced version of Sloan's algorithm with the spectral ordering. The resulting *hybrid* algorithm has been shown to give significantly better orderings for large problems than either the spectral method or the Sloan method alone. Its main disadvantage is that it requires the Fiedler vector to be computed, adding a considerable computational overhead. Our aim was to develop a profile reduction algorithm that produces profiles that are comparable with those obtained using the hybrid algorithm but at substantially less cost.

Motivated by the success of the multilevel approach in graph partitioning, we decided to look at whether a similar approach could be employed for profile reduction. The idea is to generate, given the adjacency graph of the matrix, a series of graphs, each coarser than the preceding one. The coarsest graph is then ordered. This ordering is recursively prolonged to the next finer graph, local refinement is performed at each level, and the final

ordering on the finest graph gives an ordering for $\mathbf{A}$. A full description of our multilevel algorithm is given by Hu and Scott (2001).

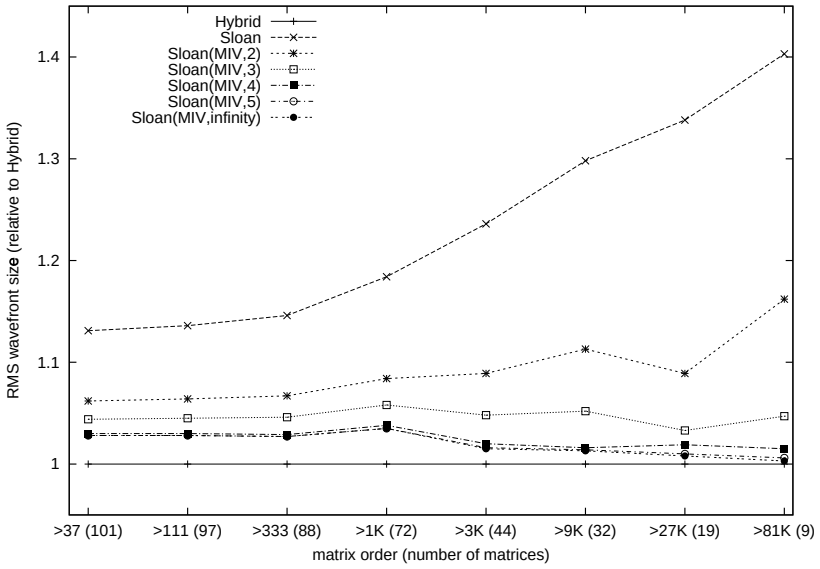


Figure 2.2: A comparison of the RMS wavefronts for the Sloan, the Hybrid and the $Sloan(MIV, K)$ algorithms.

Numerical experiments have been performed on a suite of 101 test problems, ranging in order from $n = 66$ to 224,617. The experiments were carried out on a Compaq computer with a 300 MHz Alpha EV5 processor, using the DIGITAL Fortran 90 V5.2 compiler. Full details of the test problems and numerical results are provided by Hu and Scott (2000) and Hu and Scott (2001). Figure 2.2 compares the RMS (root-mean-squared) wavefront for the Sloan and the hybrid algorithms with $Sloan(MIV, K)$. Here $Sloan(MIV, K)$ denotes our multilevel algorithm that combines Sloan's algorithm on the coarsest graph together with a maximal independent vertex (MIV) set coarsening strategy on up to K levels. Comparisons are given with respect to the hybrid algorithm so that the RMS wavefront for each algorithm is divided by the corresponding RMS wavefront for the hybrid algorithm, and geometrically averaged over the test cases to give a relative score for the algorithm. To show the effect of matrix order, the scores for each algorithm for matrices of order greater than $37 \times 3^k (1 \leq k \leq 8)$ are plotted separately, with the number of matrices over the threshold printed in brackets. A log scale is used for the x axis (matrix order n).

We observe that, relative to the hybrid algorithm, the RMS wavefront given by the Sloan algorithm deteriorates as n increases. However, as K increases, the multilevel orderings improve. In particular, the multilevel algorithm without a preset maximum number of levels, $Sloan(MIV, \infty)$, produces orderings of comparable quality (within 3.5%) to the hybrid algorithm and, in terms of CPU time (Figure 2.3), is substantially faster. Since

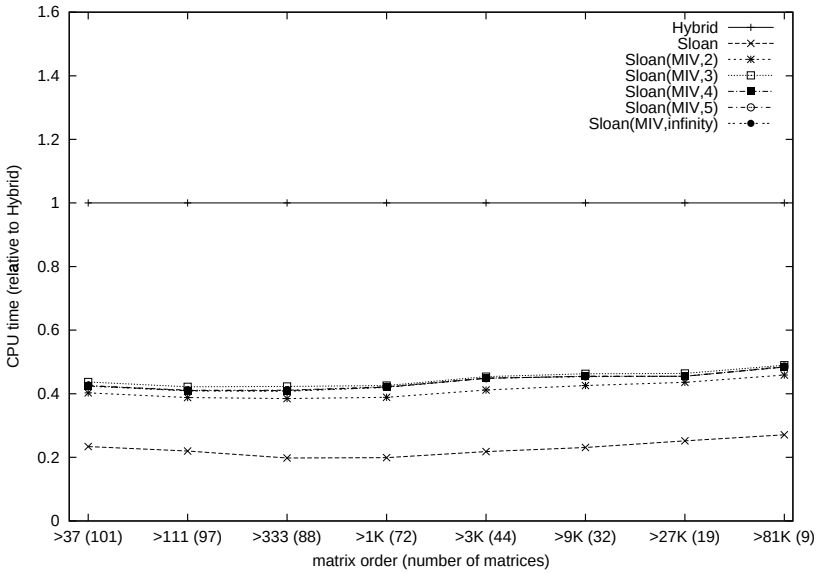


Figure 2.3: A comparison of the CPU times for the Sloan, the Hybrid and the $Sloan(MIV, K)$ algorithms.

$Sloan(MIV, \infty)$ is generally no more expensive in terms of CPU time than $Sloan(MIV, K)$ with $K > 2$ and it produces the smallest RMS wavefronts, we recommend not imposing a maximum number of levels on the multilevel algorithm.

Thus, through our multilevel Sloan algorithm, we have achieved our goal of developing a combinatorial algorithm for wavefront and profile reduction that performs as well as the hybrid algorithm in significantly less time.

References

G. Kumfert and A. Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT*, **18**, 559–590, 1997.

Y.F. Hu and J.A. Scott, A multilevel algorithm for wavefront reduction. Technical Report RAL-TR-2000-031, Rutherford Appleton Laboratory, Oxfordshire, 2000.

Y.F. Hu and J.A. Scott, A multilevel algorithm for wavefront reduction. *SIAM J. Scientific Computing*, **23**, 1352–1375, 2001.

2.6 Implementing Hager's exchange methods for matrix profile reduction (J. K. Reid and J. A. Scott)

As we discussed in Section 2.5, a variety of methods have been developed for computing a permutation of a symmetric matrix to reduce its profile. Recently, Hager (2000) introduced

two methods for improving any given permutation for profile reduction. His down exchange algorithm involves a cyclic permutation, that is, the successive exchange of rows $(k, k+1)$, $(k+1, k+2)$, ... , $(l-1, l)$ of the permuted matrix and interchanging the corresponding columns. For a given k, Hager finds the value of l that most reduces the profile. He performs a pass over the matrix with k taking the values $n-1$, $n-2$, ... , 1; he calculates l for each k and, if this gives a profile reduction, applies the corresponding permutation.

Hager's up exchange is similar, with the direction reversed. For a given k, he exchanges rows and columns $(k, k-1), (k-1, k-2), \ldots, (l+1, l)$, finding the value of l that most reduces the profile. He performs a pass over the matrix with k taking the values 2, 3, ... , n.

Hager proposes using the down exchange and up exchange schemes in an iterative fashion: the down exchange algorithm is first applied, followed by the up exchange algorithm, followed by the down exchange algorithm, and so on.

In many practical applications, it is important that reordering the matrix to reduce the profile is done as quickly and efficiently as possible. If a large number of matrices having the same sparsity pattern are to be factorized or if storage restrictions require the smallest possible profile, it may be worthwhile to spend a relatively large amount of time computing a permutation that minimizes the profile. However, if the matrix needs to be factorized only once, the cost of reducing the profile must be compared with that required for the matrix factorization; in such circumstances, a slightly larger profile may be acceptable if it can be computed cheaply. Hager presents timings for his exchange algorithms that show they are expensive to run compared with algorithms such as the Sloan algorithm that are used to produce the initial reordering. Hager also reports that, in general, he found the down exchange algorithm to be significantly faster than the up exchange algorithm (typically by a factor of between 4 and 10). We have considered carefully how the exchange algorithms should be implemented. In particular, we have been able to implement the up exchange algorithm so that it runs much faster than Hager reported. Full details of our implementation of the exchange algorithms are given by Reid and Scott (2001).

In Table 2.4, we present the normalised profiles and CPU times for applying the exchange algorithms to the orderings obtained using the HSL profile reduction code `MC60` (`MC60` implements an enhanced version of the Sloan algorithm). The test examples are taken from the set used by Kumfert and Pothen (1997). The CPU timings are for a Compaq DS20, using the Compaq Fortran 90 compiler V5.4A-1472 with the -O option. Results are given for a single application of Hager down/up, for repeating the down/up exchanges 5 times, and for repeating the down/up exchanges without limit until there is no further reduction in the profile. The greatest reductions result from the first application of the exchange algorithm, although for a number of problems, including `ford2` and `tandem_dual`, useful further reductions are achieved by repeatedly applying the exchange algorithm.

Identifier	MC60		+Hager down/up (1)		+Hager down/up (5)		+Hager down/up (inf.)	
	Profile	Time	Profile	Time	Profile	Time	Profile	Time
barth5	91.8	0.10	85.4	0.08	84.9	0.22	82.7	0.94
commanche_dual	42.3.2	0.03	39.6	0.02	39.3	0.09	39.1	0.18
copter2	685.2	0.60	653.7	0.70	650.9	1.88	650.0	7.72
ford1	126.3	0.09	105.5	0.14	99.9	0.36	97.5	1.41
ford2	407.9	0.71	349.3	2.71	334.2	5.14	328.9	15.31
nasasrb	346.0	1.29	344.3	0.55	344.3	2.80	344.3	2.80
onera_dual	1025.2	0.65	955.0	3.39	933.7	6.62	910.0	28.13
pds10	559.0	0.13	536.6	0.10	535.5	0.35	535.0	0.48
skirt	808.0	1.28	807.8	0.56	807.9	1.69	807.9	1.69
tandem_dual	701.3	0.66	650.1	2.11	626.2	4.90	611.0	17.34

Table 2.4: Normalised profiles and CPU times for MC60 and for applying the Hager exchange algorithm to the MC60 orderings. The numbers in parentheses are the number of times the down/up exchange algorithms are applied; inf. indicates no limit.

For a number of problems, the cost of a single application of the exchange algorithms is significantly greater than the initial MC60 ordering cost but, because of our efficient implementation of the Hager up exchanges, we feel that most users would be unlikely to find this cost prohibitive.

Based on our findings, we plan to include our codes in HSL as routine MC67. MC67 will allow the user to apply the Hager down/up exchanges to any given ordering; in particular, the user interface will be designed so that it will be straightforward for the user to run the exchange algorithms to refine the ordering produced by the HSL code MC60. The limit on the number of iterations will be a parameter under the user's control.

References

W.W. Hager. *Minimizing the profile of a matrix.* Department of Mathematics, University of Florida (www.math.ufl.edu/~hager/), 2000. To appear in *SIAM J. Scientific Computing.*

G. Kumfert and A. Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT*, **18**, 559–590, 1997.

J.K. Reid and J.A. Scott. Implementing Hager's exchange methods for matrix profile reduction. Technical Report RAL-TR-2001-039, Rutherford Appleton Laboratory, Oxfordshire, 2001.

2.7 A new symmetric indefinite sparse multifrontal solver (I. S. Duff)

There is a long history of sparse symmetric codes in HSL. The code MA17 was written by Curtis and Reid (1971) and used an entry-tracking code implemented using linked lists. Only 1×1 pivots were used so that the factorization could fail on indefinite systems. It would continue if the pivots changed sign or were very small and would only terminate with failure if a zero pivot was encountered.

Duff and Reid (1982) used pivots of order 1 and 2 in MA27 which was the first code in a software library to use the multifrontal technique. MA27 was widely used by the numerical optimization community and, for constrained problems, was often employed on an augmented system of the form

$$\begin{pmatrix} \mathbf{H} & \mathbf{A} \\ \mathbf{A^T} & \mathbf{0} \end{pmatrix} \tag{2.6}$$

so, in 1993, Duff and Reid developed another multifrontal code MA47 (Duff and Reid, 1995) that used structured pivots of the form $\begin{pmatrix} \times & \times \\ \times & 0 \end{pmatrix}$ or $\begin{pmatrix} 0 & \times \\ \times & 0 \end{pmatrix}$ and used data structures to preserve the block structure during the factorization. At the same time, several improvements were made to the implementation including the use of Level 3 BLAS and the avoidance of COMMON blocks. However, although there were some notable successes with this code (see Duff and Reid, 1996), the complexity often meant that it performed worse than MA27, sometimes even on matrices of the form (2.6) for which it was designed. Thus it never did replace or supersede MA27 in HSL.

One of the main reasons for the decision to develop MA57 was to keep to the relative simplicity of the MA27 algorithm but provide a code with a better user interface, that was threadsafe, and that uses higher level BLAS in both factorization and solve phases. In addition, there were several new features that had been often requested by users of MA27 that we have added to the new code.

Some of these new features of MA57 include:

In the analysis phase:

1. Use of approximate minimum degree (Amestoy, Davis and Duff, 1996) instead of minimum degree with a version that is efficient even if the input matrix has some dense rows.

In the factorization phase:

1. Use of drop tolerances. Entries smaller than a predefined value are dropped from the factorization.

2. A range of pivoting options.

   - Numerical pivoting using the Bunch, Kaufman, Parlett decomposition (Bunch, Kaufman and Parlett, 1976) with threshold pivoting
   - 1×1 pivoting only, and error return if matrix is discovered to be non-definite
   - 1×1 pivoting, and only exit if zero pivot is found
   - 1×1 pivoting only but, if matrix is not definite, then modify pivots dynamically using a variant of the Schnabel-Eskow scheme (Eskow and Schnabel, 1991) to obtain a factorization of a bounded diagonal perturbation of the matrix.

3. The ability to restart the computation from where it stops if it runs out of storage. It is also possible to discard the factors to provide more space so that the factorization can continue and provide accurate information on the space required for a subsequent factorization of the same matrix.

4. The option to return the pivots to the user and to alter them if desired.

In the solve phase:

1. A range of entries for error analysis and iterative refinement. This can either be automatic, using the strategy of Arioli, Demmel and Duff (1989) with either

   - No estimate of solution provided

     or

   - Estimate of solution provided

   or can be left more to the control of the user when only one step of iterative refinement is performed on each call. There are five possible options.

   - Solve and return residual
   - Solve, return residual, and perform one iterative correction
   - Estimate of solution provided. Compute residual and perform one iterative correction
   - Estimate of solution and residual provided. Perform one iterative correction
   - Estimate of solution and residual provided. Perform one iterative correction and return correction and new residual

2. The solution of multiple right-hand sides using Level 3 BLAS.

3. If the factorization is written as

$$\mathbf{A = PLDL^T P^T}$$

then a partial solution facility offering the solution of equations with coefficient matrix $\mathbf{A}$, $\mathbf{PLP^T}$, $\mathbf{PDP^T}$, or $\mathbf{PL^T P^T}$.

We show some performance statistics of our new code in Section 5.3 of this report.

References

P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Analysis and Applications*, **17**(4), 886–905, 1996.

M. Arioli, J. W. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Analysis and Applications*, **10**, 165–190, 1989.

J. R. Bunch, L. Kaufman, and B. N. Parlett. Decomposition of a symmetric matrix. *Numerische Mathematik*, **27**, 95–110, 1976.

A. R. Curtis and J. K. Reid. Fortran subroutines for the solution of sparse sets of linear equations. Technical Report AERE R 6844, Her Majesty's Stationery Office, London, 1971.

I. S. Duff and J. K. Reid. MA27 – A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical Report AERE R10533, Her Majesty's Stationery Office, London, 1982.

I. S. Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL 95-001, Rutherford Appleton Laboratory, Oxfordshire, England, 1995.

I. S. Duff and J. K. Reid. Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.*, **22**(2), 227–257, 1996.

E. Eskow and R. B. Schnabel. Algorithm 695: Software for a new modified Cholesky factorization. *ACM Trans. Math. Softw.*, **17**, 306–312, 1991.

3 Other numerical linear algebra

3.1 The Sparse BLAS (I. S. Duff, M. Heroux, R. Pozo, and C. Vömel)

The effort by the BLAS Technical Forum to produce an updated version of the Basic Linear Algebra Subprograms continued for many years and it was with some relief that everything came to a conclusion in 2001. The standard itself (BLAS Technical Forum, 2001) should be appearing in two parts in the International Journal of High Performance Computing Applications and an article on the standard (Blackford, Demmel, Dongarra, Duff, Hammarling, Henry, Heroux, Kaufman, Lumsdaine, Petitet, Pozo, Remington and Whaley, 2002) has been submitted to ACM Transactions on Mathematical Software. Major extensions to the earlier BLAS include added functionality, mixed precision BLAS, and sparse BLAS.

We have played a major part in the design of Basic Linear Algebra Subprograms for Level 2 and Level 3 kernels for sparse matrices. It is envisaged that these kernels will be widely used in the solution of sparse equations by iterative methods. Some time ago, we wrote a paper on User Level codes (Duff, Marrone, Radicati and Vittoli, 1997) and ideas from this paper influenced the design of the kernels within the sparse BLAS developed by the BLAS Technical forum (`http://www.netlib.org/cgi-bin/checkout/blast/blast.pl`). Recent work included a radical redesign of the sparse BLAS to avoid explicit specification of the sparse data structures. A key part of this design is the idea of matrix handles so that the user need not be concerned with the details of the storage schemes for the sparse matrix. This design makes it easy to add further functionality to the sparse BLAS in the future.

We illustrate, in our report (Duff, Heroux and Pozo, 2001), the use of the Sparse BLAS with examples in the three supported programming languages, Fortran 95, Fortran 77, and C. In collaboration with a PhD student, Christof Vömel, and some stagiaires at CERFACS, we have developed the Fortran 95 instantiation of the sparse BLAS for the BLAS Technical forum project. The Fortran 95 implementation is described in the report by Duff and Vömel (2001).

References

S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). *Submitted to TOMS*, 2002.

I. S. Duff and C. Vömel. The implementation of the Sparse BLAS in Fortran 95. Technical Report TR/PA/01/27, CERFACS, Toulouse, France, 2000.

I. S. Duff, M. A. Heroux, and R. Pozo. The Sparse BLAS. Technical Report RAL-TR-2001-032, Rutherford Appleton Laboratory, Oxfordshire, 2001.

I. S. Duff, M. Marrone, G. Radicati, and C. Vittoli. Level 3 Basic Linear Algebra Subprograms for sparse matrices: a user level interface. *ACM Trans. Math. Softw.*, **23**(3), 379–401, 1997.

BLAS Technical Forum. Basic Linear Algebra Technical (BLAST) Forum Standard. *Int. J. High Performance Computing Applications*, **15** (6), 2001.

3.2 A parallel version of MA48 for unsymmetric linear systems (I. S. Duff and J. A. Scott)

In Section 2.4, we discussed the design of a parallel frontal solver MP43 for unsymmetric linear systems. MP43 required the system matrix **A** to be preordered to the singly bordered block diagonal form (2.1); a frontal algorithm was then used to form the partial **LU** decomposition of each block submatrix in parallel. Clearly, it is possible to use other direct solvers in place of the frontal solver. In particular, our interest is in using the HSL general purpose sparse direct solver MA48.

MA48 uses a sparse variant of Gaussian elimination to compute a decomposition of **A** into its **LU** factors. It employs pivoting to preserve sparsity in the factors and controls numerical stability using a threshold criterion. Numerical experimentation has shown MA48 to be ideally suited to solving problems that are highly unsymmetric and very sparse; it is frequently used for solving the large sparse systems that arise in chemical process engineering problems. It is also very efficient when there is a need to solve repeatedly for different right-hand sides.

The heart of MA48 lies in a separate package called MA50. MA48 (optionally) permutes **A** to block triangular form; MA50 then factorizes each block separately. MA50 may also be called directly by the user, but offers a less convenient interface and performs no error checking. MA50 (and MA48) may be used on rectangular systems.

Our aim is to develop a parallel solver in which a variant of MA50 is used to partially factorize the submatrices $(\mathbf{A}_{ll}\ \mathbf{C}_l)$ of the partitioned matrix (2.1); MA48 will be used to factorize the interface matrix that remains once the submatrix factorizations are complete. Our new solver is called MP48. MP48 adopts many of the design principles used by the HSL parallel frontal solvers MP42, MP43, and MP62. For portability, MP48 is written in Fortran 90 and uses MPI for message passing. Like the parallel frontal solvers, it may be used on

shared or distributed memory machines and may be run on a single process or on up to N processes, where N is the number of submatrices in the block diagonal form.

A weakness of the serial code MA48 is that the matrix $\mathbf{A}$ and its factors are held in main memory; this limits the size of problem that can be solved. To circumvent this, MP48 allows the matrix data to be "shared" between the processes, with each process only requiring access to the submatrices assigned to it. Furthermore, at the end of the submatrix factorization, the user may optionally choose to write the matrix factors to unformatted sequential files. This potentially increases the size of problem that can be solved, although using files for the factors can, in some computing environments, add a significant overhead and so should be avoided if there is sufficient main memory for the factors.

Preliminary results obtained for MP48 on a 12-processor SGI Origin2000 are encouraging. In Table 3.1, wallclock timings (in seconds) for MP48 run on $p = 1$, 2, and 4 processors are presented, together with timings for MA48 run on a single processor. The timings are for factorizing the matrix and then solving for a single right-hand side. We emphasise that the code is not yet finished but, nevertheless, good speedups are achieved and, for some problems, MP48 is faster than MA48 on a single processor. Further tests are currently underway and we plan to make MP48 available in the next release of HSL.

Identifier	Order	MA48	MP48 ($N = 8$)		
			$p = 1$	2	4
4cols	11770	2.34	0.70	0.45	0.31
10cols	29496	16.54	2.70	1.50	0.93
bayer01	57735	6.52	4.09	2.89	1.47
icomp	75724	0.88	1.61	1.04	0.76
lhr34c	35152	24.20	33.40	18.94	11.18
lhr71c	70304	51.26	72.05	39.53	26.19

Table 3.1: Timings for MP48 for chemical process engineering test problems.

3.3 Solving symmetric sparse systems of linear equations with zeros on the diagonal (J. K. Reid)

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

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

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

$$\min_{\mathbf{x}} ||\mathbf{Bx} - \mathbf{c}||_2$$

subject to

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

This is equivalent to solving the sparse symmetric linear system

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

To take good advantage of the sparsity, we use block pivots of the forms

$$\begin{pmatrix} \mathbf{0} & \mathbf{A}_1 \\ \mathbf{A}_1^\mathbf{T} & \mathbf{A}_2 \end{pmatrix} \text{ and } \begin{pmatrix} \mathbf{0} & \mathbf{A}_1 \\ \mathbf{A}_1^\mathbf{T} & \mathbf{0} \end{pmatrix},$$

which we call 'tile' and 'oxo' pivots, as well as ordinary block pivots that are treated as full even if they have some zeros. These lead to Schur complements of the form

$$\begin{pmatrix} \mathbf{0} & \mathbf{B}_2 & \mathbf{B}_3 \\ \mathbf{B}_2^\mathbf{T} & \mathbf{B}_1 & \mathbf{B}_4 \\ \mathbf{B}_3^\mathbf{T} & \mathbf{B}_4^\mathbf{T} & \mathbf{0} \end{pmatrix}$$

that have to be added to the reduced matrix. The zero blocks may be large, which is why these kinds of pivots may be advantageous.

We have written a new HSL code `MA67` that holds the matrix by blocks. It starts by identifying columns of $\mathbf{A}$ with the same structure, and groups the corresponding variables into supervariables. It then stores a copy of the matrix by blocks corresponding to supervariables. Zero blocks are not stored and only one copy of each off-diagonal block is held.

Each block pivot is chosen using the Markowitz criterion (see Duff and Reid, 1996), based on the block sparsity pattern and the sizes of the blocks and a temporary copy of the block row or rows is made. We refer to this copy as the 'front'. For a full block pivot, it is a full rectangular matrix. For an oxo pivot, it has the form

$$\begin{pmatrix} 0 & \mathbf{A}_1 & \mathbf{A}_5 & \mathbf{A}_6 & 0 \\ \mathbf{A}_1^\mathbf{T} & 0 & 0 & \mathbf{A}_7 & \mathbf{A}_8 \end{pmatrix}$$

and for a tile pivot, it has the form

$$\begin{pmatrix} 0 & \mathbf{A}_1 & \mathbf{A}_6 & 0 \\ \mathbf{A}_1^\mathbf{T} & \mathbf{A}_2 & \mathbf{A}_7 & \mathbf{A}_8 \end{pmatrix}.$$

For a full block pivot, as many simple 1×1 pivots and full 2×2 pivots are chosen as the stability criterion in Section 2.3 of Duff and Reid (1996) allows and the front is revised. Similarly, for a block oxo or tile pivot, as many simple oxo or tile pivots as the stability

criterion permits are chosen and the front is revised. In all cases, the Schur complement is formed and added back into the main data structure, which may need to be enlarged to accommodate fill-in blocks. The chosen pivots rows are stored for the solution phase. Any rows of the pivot block that are not chosen as pivots are returned to the main data structure and are not considered again as potential pivots until modified by a later Schur complement. If any set of the supervariables involved in the front now have identical column structures, they are merged into a new bigger supervariable. With the aim of efficiency, Level 3 BLAS are used for the operations of forming the Schur complement.

The HSL code `MA47` uses the same kinds of block pivots, but has a separate analysis phase that works with the sparsity pattern alone and constructs an assembly tree to represent the factorization. Numerical factorization is performed separately using this tree. If the stability criterion disallows the use of a pivot or part of a pivot, the uneliminated part is passed to the parent node. The hope was that the volume of such passes would be small, but unfortunately we found that the uneliminated parts may be passed up through many levels of the tree and become so voluminous that much more work is performed than was anticipated during the analysis phase. The intention of `MA67` is to avoid these problems by merging the phases and having the actual numerical values to hand when choosing the pivots. After the rejection of pivots on stability grounds, the later choices of pivot take into account the actual sparsity pattern that is a consequence of the rejections.

We have run preliminary tests on about 15 genuine test cases, which have shown that further tuning of the code is needed. The current version often shows performance broadly comparable with `MA47`, but in several cases it was about three times faster and in one case it was 2.5 times slower. When compared with the new HSL code `MA57` (see Section 2.7), which does not use tile or oxo pivots, it was usually significantly slower, though in one case it was about twice as fast.

References

I. S. Duff and J. K. Reid. Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.*, **22**(2), 227–257, 1996.

3.4 Solving unsymmetric banded systems of linear equations (J. K. Reid)

Consider the solution of an unsymmetric banded system of n linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\mathbf{A}$ is an unsymmetric band matrix with lower semibandwidth kl and upper semibandwidth ku, that is, $a_{ij} = 0$ for $i > j + kl$ or $j > i + ku$. If the matrix

is factorized using Gaussian elimination with row interchanges, fill-in is limited to kl additional diagonals of the upper triangle and the computation may be performed within an array of size $n(2kl + ku + 1)$

Conventionally, codes do not take advantage of any zeros within the original structure. This is the case, for example, for the LAPACK code _GBTRF, though this does take account of zeros at the end of the pivot row beyond the original limit if the interchanges are such that they could not have caused fill-in.

Irregularities in the underlying geometry are likely to lead to many zeros within the structure at the ends of the rows and the ends of the columns. We have written an HSL code MA65 that takes advantage of this. At each pivotal step, operations are avoided on any row with a zero in the pivot column and on any column beyond the last with an entry in the pivot row.

We were motivated by the experience of a potential customer of our most commercially successful sparse matrix solver MA48 who wanted to solve a large number of sparse systems of quite modest order and reported that MA48 'is significantly slower than our own dedicated solver for our test cases (typically by a factor 3 or 4)'. We thought perhaps that a band solver might be applied after the matrix had been reordered for a small bandwidth, but found that this was no better than MA48 because it was doing many operations on zeros.

We require the user of MA65 to supply arrays ROWEND and COLEND of size n holding the indices of the last entries of the rows and columns. To simplify the logic, we always regard the diagonal as nonzero even if it contains some zeros. To simplify the row exchanges, we start by increasing COLEND(j) to max(COLEND($1 : j$), $j = 1, 2, \ldots, n$), that is, making COLEND monotonic. Corresponding explicit zeros are set in the array A that holds the matrix. This ensures that all entries of each row from the first to the last are held explicitly.

Each pivot is found by searching the pivot column for its largest entry and exchanging its row with the pivot row. For pivot j, the search can be confined to rows j to COLEND(j). When rows are exchanged, the corresponding components of ROWEND must be exchanged. This may cause the non-pivot row to end ahead of its diagonal. In this case, we insert zeros explicitly as far as the diagonal and adjust ROWEND.

An example of the performance of MA65 is provided by one of the cases supplied by our potential customer. This had order 208. After ordering by MC61 the semi-bandwidths were both 19 and after ordering by MC62 they were 8 and 42. The actual time taken by MA65 an a DEC Alpha machine with default optimization was 0.00023 and 0.00016 seconds, respectively, whereas the factorize call of MA48 took 0.00059 seconds. Thus we achieve the goal of providing a general purpose code with comparable performance to the potential customer's special purpose dedicated code.

3.5 Use of orderings for large entries on the diagonal (I. S. Duff and J. Koster)

We have continued the development of several algorithms based on bipartite weighted matching algorithms for permuting a matrix so that the entries on the diagonal of the permuted matrix are large relative to the off-diagonal entries. We have also implemented a scaling with one of the options which gives a unit diagonal with no off-diagonal entries larger than one. This work is based on earlier work reported in Duff and Koster (1999) and is described in detail in Duff and Koster (2001). A highly efficient Fortran code `MC64` has been written to implement this algorithm and is included in HSL. The algorithm and code have been used extensively over this reporting period by many people in the solution of large sparse systems of equations and in preconditioning techniques for sparse matrices.

The routine can be very helpful when factorizing very unsymmetric systems, as has been our own experience with `MA41` and `MUMPS` (Amestoy et al., 2001). It has also proved to be almost a necessary preprocessor for algorithms that use static pivoting strategies as in `SuperLU` (Li and Demmel, 1999). More recently, Gupta (2001) has reaffirmed the importance of such a permutation in his work on sparse direct solvers and Benzi and his colleagues (Benzi, Haws and Tůma, 2000) have found that it is absolutely necessary to use `MC64` if preconditioning techniques are to be successful on highly indefinite and nonsymmetric matrices.

References

P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Analysis and Applications*, **23**(1), 15–41, 2001.

M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Scientific Computing*, **22**(4), 1333–1353, 2000.

I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Analysis and Applications*, **20**(4), 889–901, 1999.

I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Analysis and Applications*, **22**(4), 973–996, 2001.

A. Gupta. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. Technical Report RC 22137 (99131), IBM T.J. Watson Research Center, Yorktown Heights, NY, August 2001.

X. S. Li and J. W. Demmel. A scalable sparse direct solver using static pivoting. *in* 'Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing', San Antonio, Texas, March 22–24 1999.

3.6 A stopping criterion for the conjugate gradient algorithm in a finite element method framework (M. Arioli)

Arioli (2000) combined linear algebra techniques with finite-element techniques to obtain a reliable stopping criterion for the conjugate gradient algorithm. Although the conjugate gradient method is very effective for solving finite-element equations, our experiments give very good evidence that the usual stopping criterion based on the Euclidean norm of the residual $\mathbf{b} - \mathbf{Ax}$ can be totally unsatisfactory and frequently misleading.

When using an iterative method for solving the linear system

$$\mathbf{Au} = \mathbf{b} \tag{3.1}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is symmetric and positive-definite and $\mathbf{b} \in \mathbb{R}^N$, we normally incorporate a stopping criterion based on the *a posteriori* componentwise or normwise backward errors (Arioli, Duff and Ruiz, 1992, Higham, 1996). If we use the conjugate gradient method, it is quite natural to have a stopping criterion that takes advantage of the minimization property of this method. At each step k, the conjugate gradient method minimizes the energy norm of the error $\boldsymbol{\delta}\mathbf{u} = \mathbf{u} - \mathbf{u}^{(k)}$ on a Krylov space $\mathcal{K}_k$ (Greenbaum, 1997):

$$\min_{\mathbf{u}^{(k)}\in\mathcal{K}_k} \boldsymbol{\delta}\mathbf{u}^{\mathbf{T}}\mathbf{A}\boldsymbol{\delta}\mathbf{u}.$$

The space $\mathbb{R}^N$ with the norm

$$\|\mathbf{y}\|_{\mathbf{A}} = (\mathbf{y}^{\mathbf{T}}\mathbf{A}\mathbf{y})^{1/2}$$

induces on its dual space the dual norm

$$\|\mathbf{f}\|_{\mathbf{A}^{-1}} = (\mathbf{f}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{f})^{1/2}.$$

Therefore, a $\mathbf{u}^{(k)}$ that satisfies a stopping criterion of the form

$$\text{IF} \quad \|\mathbf{Au}^{(k)} - \mathbf{b}\|_{\mathbf{A}^{-1}} \le \eta\|\mathbf{b}\|_{\mathbf{A}^{-1}} \quad \text{THEN STOP}\ , \tag{3.2}$$

where $\eta < 1$ is a threshold parameter set by the user, is the solution of the perturbed linear system

$$\begin{aligned} \mathbf{Au}^{(k)} &= \mathbf{b} + \mathbf{r}^{(k)}, \\ \|\mathbf{r}^{(k)}\|_{\mathbf{A}^{-1}} &\le \eta\|\mathbf{b}\|_{\mathbf{A}^{-1}}. \end{aligned}$$

(Arioli, Noulard and Russo, 2001).

The choice of η will depend on the properties of the problem that we want to solve, and, in practice, η can be frequently much larger than ε , the roundoff unit of the computer finite-precision arithmetic. When (3.1) is a linear system obtained from the finite-element approximation of a partial differential equation, a reasonable choice for η would be $\eta = h$ or $\eta = h^2$, where $h = \max_{T \in \mathcal{T}_h} \text{diameter}(T)$ with $\mathcal{T}_h$ a set of disjoint triangles $\{T\}$ which covers the domain Ω where the partial differential equation is defined (see Arioli (2000) for a detailed analysis).

First of all we need to add, within the conjugate gradient algorithm, some tool for estimating the value $e_{\mathbf{A}}^{(k)} = \mathbf{r}^{(k)\mathbf{T}} \mathbf{A}^{-1} \mathbf{r}^{(k)}$ at each step k. This can be achieved using a Gauss quadrature rule as proposed by Golub and Meurant (1997).

In particular, this variant of the conjugate gradient produces a lower bound ξ_k for $e_{\mathbf{A}}^{(k)}$. As suggested by Golub and Meurant (1997) and Meurant (1999), the Gauss quadrature based lower bound can be made reasonably close to the value of $e_{\mathbf{A}}^{(k)}$ at the price of d additional steps of the conjugate gradient algorithm. Therefore, ξ_k will be the estimate of $e_{\mathbf{A}}^{(k-d)}$. In Golub and Meurant (1997), $d = 10$ is indicated as a successful compromise, and numerical experiments support this conclusion.

Finally, we must estimate $\mathbf{b}^{\mathbf{T}} \mathbf{A}^{-1} \mathbf{b}$. Taking into account that

$$| \, \|\mathbf{b}\|_{\mathbf{A}^{-1}} - \|\mathbf{u}^{(k)}\|_{\mathbf{A}} \, | \leq \sqrt{e_{\mathbf{A}}^{(k)}},$$

we could replace $\|\mathbf{b}\|_{\mathbf{A}^{-1}}$ with $\|\mathbf{u}^{(k)}\|_{\mathbf{A}}$ at step k if the current estimate ξ_k is less than or equal to $\eta^2 \mathbf{b}^{\mathbf{T}} \mathbf{b}$. Therefore, we can only use (3.2) after an additional check:

$$\begin{array}{l} \text{IF} \quad \sqrt{\xi_k} \leq \eta \, \|\mathbf{b}\|_2 \quad \text{THEN} \\ \quad \text{IF} \quad \sqrt{\xi_k} \leq \eta \, \|\mathbf{u}^{(k)}\|_{\mathbf{A}} \quad \text{THEN STOP} \\ \text{ENDIF} \end{array} \tag{3.3}$$

Moreover, using (3.3) we can avoid too many additional matrix-vector products.

Introducing a preconditioner to speed up the convergence rate of the conjugate gradient method, we do not need to update the previous technique for the evaluation of $e_{\mathbf{A}}^{(k)}$.

We defined an elliptic problem with discontinuous coefficients on an L-shaped domain. We generated a mesh in which the largest triangle has an area of 5×10^{-5}; the resulting linear system (3.1) has 34385 degrees of freedom. The symmetric and positive-definite matrix $\mathbf{A}$ has an estimated condition number $\kappa(\mathbf{A}) = ||\mathbf{A}||_2 ||\mathbf{A}^{-1}||_2 \approx 10^{10}$. In the preconditioned conjugate gradient algorithm, we chose $\eta^2 = 5 \times 10^{-5}$, the area of the largest triangle. We used two preconditioners: the classical Jacobi diagonal matrix, $\mathbf{M} = \text{diag}(\mathbf{A})$, and the incomplete Cholesky decomposition of $\mathbf{A}$ with zero fill-in (Greenbaum 1997). Finally, we chose $\mathbf{u}^{(0)} = \mathbf{0}$, and we assumed that the solution $\mathbf{u}$ computed by a direct solver applied to (3.1) is exact.

The stopping criteria normally used are based on the values of $\|\mathbf{A}\mathbf{u}^{(k)} - \mathbf{b}\|_2/\|\mathbf{b}\|_2$ and $\|\mathbf{A}\mathbf{u}^{(k)} - \mathbf{b}\|_2/(\|\mathbf{A}\|_2\|\mathbf{u}^{(k)}\|_2 + \|\mathbf{b}\|_2)$, (Arioli et al., 1992). In practice, the conjugate gradient algorithm is stopped when $\|\mathbf{A}\mathbf{u}^{(k)} - \mathbf{b}\|_2/\|\mathbf{b}\|_2 \leq \sqrt{\varepsilon}$.

In Figures 3.4 and 3.5, we compare the behaviour of $\|\mathbf{u} - \mathbf{u}^{(k)}\|_{\mathbf{A}}/\|\mathbf{u}\|_{\mathbf{A}} = \|\mathbf{A}\mathbf{u}^{(k)} - \mathbf{b}\|_{\mathbf{A}^{-1}}/\|\mathbf{b}\|_{\mathbf{A}^{-1}}$, with the corresponding estimate ξ_k, and the values of $\|\mathbf{A}\mathbf{u}^{(k)} - \mathbf{b}\|_2/\|\mathbf{b}\|_2$ and $\|\mathbf{A}\mathbf{u}^{(k)} - \mathbf{b}\|_2/(\|\mathbf{A}\|_2\|\mathbf{u}^{(k)}\|_2 + \|\mathbf{b}\|_2)$, for the Jacobi and the incomplete Cholesky decomposition preconditioners, respectively.

For our test problem, both criteria based on the Euclidean norm of the residual give misleading information about the iterative process. Moreover, the final true error $||\mathbf{u} - \mathbf{u}^{(k^*)}||_2/||\mathbf{u}||_2 \approx 10^{-5}$ for both the preconditioners.

Finally, the estimate of the energy norm stops before the final iteration because we chose $d = 10$ in the algorithm.

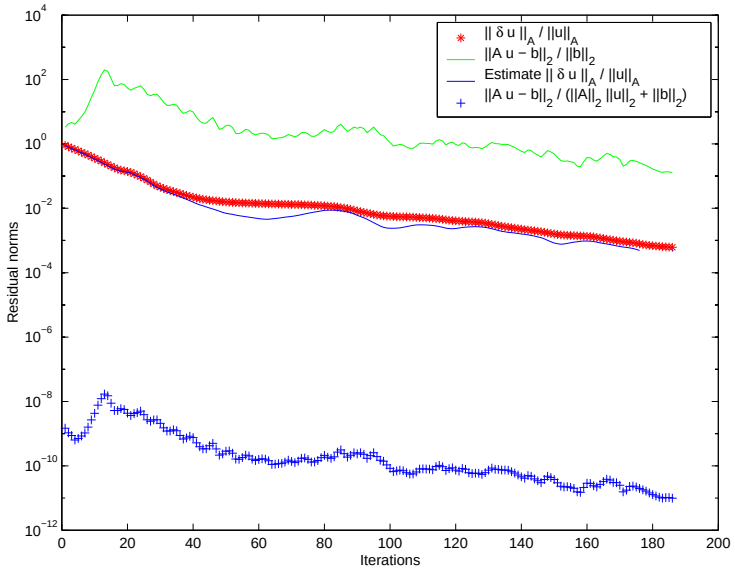


Figure 3.4: Behaviour of the norms of the residual for the Jacobi preconditioner.

References

M. Arioli. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. Technical Report Tech. Report IAN-1179, IAN, 2000.

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

M. Arioli, E. Noulard, and A. Russo. Stopping criteria for iterative methods: Applications to PDE's. *CALCOLO*, **38**, 97–112, 2001.

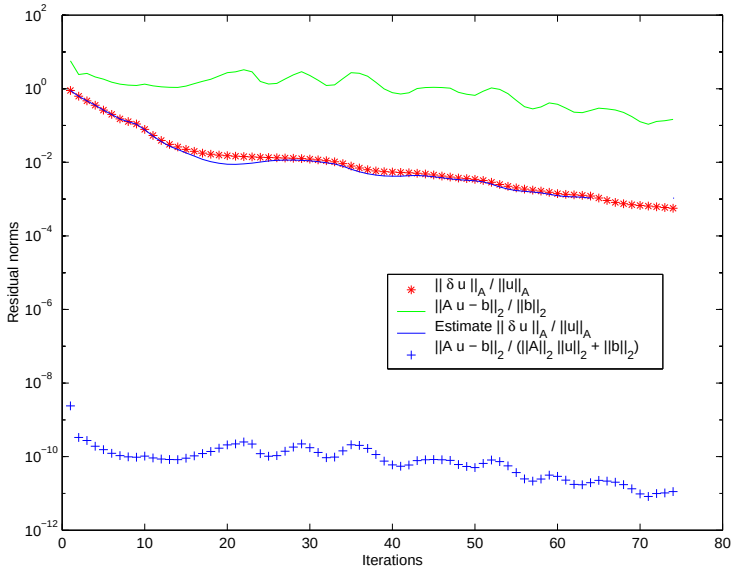


Figure 3.5: Behaviour of the norms of the residual for the incomplete Cholesky preconditioner.

G. H. Golub and G. Meurant. Matrices, moments and quadrature ii; how to compute the norm of the error in iterative methods. *BIT*, **37**, 687–705, 1997.

A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.

G. Meurant. Numerical experiments in computing bounds for the norm of the error in the preconditioned conjugate gradient algorithm. *Numerical Algorithms*, **22**, 353–365, 1999.

3.7 Robust preconditioning of dense problems from electromagnetics (B. Carpentieri, I. S. Duff and L. Giraud)

In recent years, there has been a significant amount of work on the simulation of electromagnetic wave propagation phenomena, addressing various topics ranging from radar cross section to electromagnetic compatibility, to absorbing materials, and antenna design. To address these problems the Maxwell equations are often solved in the frequency domain leading to singular integral equations of the first kind. The discretization by the boundary element method (BEM) results in linear systems with dense complex matrices

which are very challenging to solve. In this project we propose preconditioning strategies for the iterative solution of these systems. In our first study (Carpentieri, Duff and Giraud, 2000*b*), we compare different preconditioners of both implicit and explicit form in connection with Krylov methods. We emphasize in particular sparse approximate inverse techniques based on Frobenius norm minimization that use a static nonzero pattern selection. The novelty of our approach comes from using a different nonzero pattern selection for the original matrix from that for the preconditioner and from exploiting geometric or topological information from the underlying meshes instead of using methods based on the magnitude of the entries. An extract from this work was published in a conference proceedings (Carpentieri, Duff and L.Giraud, 2000*a*). The results of our numerical experiments suggest that the new strategies are viable approaches for the solution of large-scale electromagnetic problems using preconditioned Krylov methods. In particular, our strategies are applicable when fast multipole techniques are used for the matrix-vector product on parallel distributed memory computers (see Section 3.7.1). A paper (Carpentieri, Duff and Giraud, 2000*c*) related to this research has been accepted for publication. We are currently testing the numerical scalability of our preconditioner on large problems in collaboration with AEDS, implemented in its FMM code. In a second project (Carpentieri, Duff, Giraud and monga Made, 2001), we consider implicit preconditioners based on incomplete factorization algorithms. Imaginary diagonal perturbations are incorporated, which significantly improves the performance (see Section 3.7.3).

References

B. Carpentieri, I.S. Duff, and L.Giraud. Robust preconditioning of dense problems from electromagnetics. *in* L. Vulkov, J. Waśniewski and P. Yalamov, eds, 'Numerical Analysis and Its Applications. Lecture Notes in Computer Science 1988', pp. 170–178. Springer-Verlag, 2000*a*.

B. Carpentieri, I. S. Duff, and L. Giraud. Experiments with sparse preconditioning of dense problems from electromagnetic applications. Technical Report TR/PA/00/05, Rutherford Appleton Laboratory, Oxfordshire, England, 2000*b*.

B. Carpentieri, I. S. Duff, and L. Giraud. Sparse pattern selection strategies for robust Frobenius-norm minimization preconditioners in electromagnetism. *Numerical Linear Algebra with Applications*, **7**(7-8), 667–685, 2000*c*.

B. Carpentieri, I.S. Duff, L. Giraud, and M. Magolu monga Made. Sparse symmetric preconditioners for dense linear systems in electromagnetism. Technical Report TR/PA/01/35, CERFACS, Toulouse, France, 2001.

3.7.1 Combining fast multipole techniques and approximate inverse preconditioners for large calculations in electromagnetism (B. Carpentieri, I. S. Duff, L. Giraud and G. Sylvand)

For large electromagnetic calculations that can involve a few million unknowns, the use of fast multipole techniques is mandatory to evaluate the matrix-vector product. For such simulations, we first investigate the numerical scalability of the approximate inverse preconditioner (Carpentieri, Duff and Giraud, 2000) implemented in a parallel distributed code (Sylvand, 2002). Because the preconditioner naturally becomes more local when the size of the problem is increased, even though the Green functions decay rapidly, we observe that the convergence rate deteriorates when the size of the linear system increases. To overcome this drawback and improve the numerical robustness of the linear solver we propose an embedded scheme; it consists of a FGMRES Krylov solver for the outer iterations and a preconditioned GMRES inner scheme. For this outer solver, we use an accurate fast multipole calculation for the matrix-vector evaluation, preconditioned by a few inner preconditioned GMRES iterations. For the inner GMRES scheme, we use a less accurate fast multipole calculation for the matrix-vector computation and our Frobenius norm minimization approach as preconditioner. The efficiency of this numerical scheme is demonstrated on large test problems. The benefit of the new scheme is highlighted in Table 3.2, where we display the number of outer and inner fast multipole matrix-vector products as well as the elapsed time to solve a problem with around one million unknowns arising from a simulation on an Airbus aircraft. We consider GMRES(30) and FGMRES(5)/GMRES(20) because both use the same amount of memory. The target computer is a Compaq SC Alpha server.

Sphere with one million degrees of freedom				
GMRES(30)		FGMRES(5) + GMRES(20)		
# accurate FMM	Elapsed time	# accurate FMM	# less accurate FMM	Elapsed time
1196	11 hours	17	260	1 hour 30 m
Airbus aircraft with 1.1 million degrees of freedom				
GMRES(30)		FGMRES(5) + GMRES(20)		
# accurate FMM	Elapsed time	# accurate FMM	# less accurate FMM	Elapsed time
no convergence		19	300	4 hour 20 m

Table 3.2: Numerical behaviour observed on a 16 processor Alpha Compaq Server.

References

B. Carpentieri, I. S. Duff, and L. Giraud. Sparse pattern selection strategies for robust Frobenius-norm minimization preconditioners in electromagnetism. *Numerical Linear Algebra with Applications*, **7**(7-8), 667–685, 2000.

G. Sylvand. *Résolution Itérative de Formulation Intégrale pour Helmholtz 3D : Applications de la Méthode Multipôle à des Problèmes de Grande Taille.* PhD thesis, Ecole Nationale des Ponts et Chaussées, 2002.

3.7.2 Spectral two-level preconditioners (B. Carpentieri, I. S. Duff, L. Giraud and J.-C. Rioual)

When solving the left preconditioned linear system $\mathbf{M_1 A x} = \mathbf{M_1 b}$ with a Krylov method, the smallest eigenvalues of $\mathbf{M_1 A}$ often slow down the convergence. In the symmetric positive-definite case this situation is well understood and arguments exist for unsymmetric systems to explain the bad effect of the smallest eigenvalues on the rate of convergence of the unsymmetric Krylov solver. We propose a class of spectral two-level preconditioners based on a low-rank update that aims at shifting these smallest eigenvalues of $\mathbf{M_1 A}$ to be close to one. Consequently the resulting two-level preconditioner no longer suffers from the effect of those small eigenvalues. Our technique requires the explicit computation of a few eigenvalues that makes it independent of the Krylov solver being used. Symmetric and symmetric positive-definite variants can be derived for symmetric and symmetric positive-definite linear systems. In that latter situation, the resulting preconditioner is similar to those proposed by Carvalho, Giraud and Le Tallec (2001) in the framework of domain decomposition for elliptic equations where the shape of the smallest eigenvectors might be approximated *a priori*. The effectiveness of the new preconditioners is demonstrated on symmetric non-Hermitian problems arising from electromagnetism (Carpentieri, 2002) (similar problems to those considered in Section 3.7.3) and on symmetric positive-definite and unsymmetric linear systems arising in domain decomposition for the simulation of semiconductor devices (Rioual, 2002). Although we use this technique for left preconditioners, this spectral two-level technique is equally applicable for right preconditioners.

References

B. Carpentieri. *Sparse preconditioners for dense linear systems in electromagnetic applications.* PhD thesis, CERFACS, Toulouse, France, 2002.

L. M. Carvalho, L. Giraud, and P. Le Tallec. Algebraic two-level preconditioners for the Schur complement method. *SIAM J. Scientific Computing*, **22**(6), 1987–2005, 2001.

J.-C. Rioual. *Solving linear systems in semiconductor device modeling on parallel distributed computers.* PhD thesis, CERFACS, Toulouse, France, 2002.

3.7.3 Sparse symmetric preconditioners for dense linear systems in electromagnetism (B. Carpentieri, I. S. Duff, L. Giraud and M. Magolu monga Made)

In order to further develop the study of Carpentieri et al. (2000), we consider symmetric preconditioning strategies for the iterative solution of dense complex symmetric non-Hermitian systems arising in computational electromagnetics. In particular, we report on the numerical behaviour of the classical Incomplete Cholesky factorization as well as some of its recent variants and consider also well known factorized approximate inverses. We illustrate the difficulties that these techniques encounter on the linear systems under consideration and give some clues to explain their disappointing behaviour. We propose two symmetric preconditioners based on Frobenius-norm minimization that use a prescribed sparsity pattern. The numerical and computational efficiency of the proposed preconditioners are illustrated on a set of model problems arising both from academic and from industrial applications. More details on this work are available in the report of Carpentieri et al. (2001).

References

B. Carpentieri, I. S. Duff, and L. Giraud. Sparse pattern selection strategies for robust Frobenius-norm minimization preconditioners in electromagnetism. *Numerical Linear Algebra with Applications*, **7**(7-8), 667–685, 2000.

B. Carpentieri, I.S. Duff, L. Giraud, and M. Magolu monga Made. Sparse symmetric preconditioners for dense linear systems in electromagnetism. Technical Report TR/PA/01/35, CERFACS, Toulouse, France, 2001.

3.8 Rank-revealing factorizations and incremental norm estimation (I. S. Duff and C. Vömel)

We have developed an incremental approach to 2-norm estimation for triangular matrices which is important for the detection of ill-conditioning, one of the basic problems arising in the numerical solution of linear systems. Applications of our scheme include the calculation of forward error bounds based on the condition number, robust pivot selection criteria and rank-revealing factorizations, in particular, when *inverse* factors arise in the factorization. Duff and Vömel (2001) introduced such a scheme applicable for both dense and sparse matrices which can arise for example from a **QR**, a Cholesky or a **LU** factorization. If the explicit inverse of a triangular factor is available, as in the case of an implicit version of the LU factorization, we can relate our results to incremental condition estimation (ICE) presented by Bischof (1990). Incremental norm estimation (INE) extends directly from

the dense to the sparse case without needing the modifications that are necessary for the sparse version of ICE. INE can be applied to complement ICE, since the product of the two estimates gives an estimate for the matrix condition number. Furthermore, when applied to matrix inverses, INE can be used as the basis of a rank-revealing factorization. The quality of our results on standard test cases is consistently high and demonstrates the general reliability of our scheme. A revised version of the paper of Duff and Vömel (2001), which also contains a theoretical analysis of our scheme, will appear in BIT.

References

C. H. Bischof. Incremental condition estimation. *SIAM J. Matrix Analysis and Applications*, **11**, 312–322, 1990.

I. S. Duff and C. Vömel. Incremental norm estimation for dense and sparse matrices. Technical Report RAL-TR-2001-005, Rutherford Appleton Laboratory, Oxfordshire, 2001. Also Technical Report TR/PA/00/83 from CERFACS, Toulouse. To appear in *BIT*.

3.9 Incomplete QR factorizations (Z.-Z. Bai, I. S. Duff, A. Papadopoulos, and A. J. Wathen)

A paper on the theoretic work with Zhong-Zhi Bai was published by BIT in 2001 (Bai, Duff and Wathen, 2001) and for the last two years, Wathen and Duff have been working with Andreas Papadopoulos to develop practical algorithms and codes and test the efficiency of using the IQR factorization as a preconditioner. These incomplete orthogonal factorization methods are based upon Givens rotations. The methods implemented include two that perform the Givens reductions in a columnwise fashion: column Incomplete Givens Orthogonalization (cIGO-method) that drops entries by position only and column Threshold Incomplete Givens Orthogonalization (cTIGO-method) that drops entries dynamically by both their magnitude and positions, and a third method, row Threshold Incomplete Givens Orthogonalization (rTIGO-method), that again drops entries dynamically but only by magnitude with the reduction now performed in a rowwise fashion. Our theoretical analyses showed that these methods can produce a nonsingular sparse incomplete upper triangular factor and either a complete orthogonal factor or a sparse nonsingular incomplete orthogonal factor for a general nonsingular matrix. Therefore, these methods can potentially generate efficient preconditioners for Krylov subspace iterations for solving large sparse systems of linear equations. Moreover, the upper triangular factor is an incomplete Cholesky factorization preconditioner for the normal equations from least-squares problems.

Our early experience with these implementations is that they sometimes outperform ILU preconditioners in terms of reducing the number of iterations although they are roughly twice as expensive to generate. Initial experiments with using our IQR preconditioners in the solution of least-squares problems using CGNE do, however, look very promising. In this case, the IQR preconditioner can be used because the incomplete Q factor is still orthogonal while the L factor from ILU clearly is not.

References

Z.-Z. Bai, I. S. Duff, and A. J. Wathen. A class of incomplete orthogonal factorization methods. i: Methods and theories. *BIT*, **41**(1), 53–70, 2001.

3.10 EA16: a new block Lanczos code (K. Meerbergen and J.A. Scott)

EA16 is a new Fortran code that has been designed and developed for HSL 2002 for the computation of selected eigenvalues and the corresponding eigenvectors of large sparse real symmetric matrices. EA16 may be used for either the standard eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \ , \tag{3.4}$$

where $\mathbf{A}$ is an $n \times n$ symmetric matrix, or for the generalized eigenvalue problem

$$\mathbf{K}\mathbf{x} = \lambda\mathbf{M}\mathbf{x} \ , \tag{3.5}$$

where $\mathbf{K}$ and $\mathbf{M}$ are $n \times n$ symmetric matrices, and either $\mathbf{M}$ or $\mathbf{K}$ is positive-semidefinite. If $\mathbf{K}$ is positive-semidefinite the generalized eigenvalue problem (3.5) is known as the *buckling problem*. Applications of the form (3.4) arise in quantum physics and chemistry while (3.5) arises in structural analysis, acoustics, and the stability analysis of Stokes problems.

Since the 1950s, the development of new eigensolvers, or the improvement of existing ones, has been the subject of continuing research. We believe that there are a number of key features that are desirable for a large-scale symmetric eigensolver. These include the use of implicit restarting, automatic pole selection, and a block option. However, although some well known software packages have been written over the years, each offers only a subset of these features. Our aim in designing EA16 was thus to develop a state of the art block Lanczos code for solving both the standard and generalized eigenvalue problems, incorporating cheap orthogonalization, implicit restarting, and automatic pole selection. In addition, to avoid throwing away the old subspace when the pole is changed, a key design feature of EA16 is its use of rational Krylov.

Through the number of different options that it offers, EA16 is a flexible code. In particular, it can be used for the computation of eigenvalues lying in different parts of the

spectrum. It can be used to compute the eigenvalues of largest or smallest modulus or those lying furthest from a specified point. It will also compute the leftmost or rightmost eigenvalues, or those lying to the left or right of a chosen point. In addition, the user may specify the desired eigenvalues to be those lying inside an interval.

A key feature of `EA16` is its use of reverse communication for the action of the matrices $\mathbf{A}$, $\mathbf{K}$ and $\mathbf{M}$, as well as the spectral transformations $(\mathbf{A} - \sigma\mathbf{I})^{-1}$, $(\mathbf{K} - \sigma\mathbf{M})^{-1}\mathbf{M}$, and $(\mathbf{K} - \sigma\mathbf{M})^{-1}\mathbf{K}$, on sets of vectors. This makes `EA16` particularly effective when the user is able to provide an efficient code for the product of one or more of these matrices with a number of vectors equal to the blocksize of the Lanczos method.

We illustrate the performance of `EA16` for the buckling problem `BCSST27` from the Harwell Boeing Collection. This problem is of dimension 1224. The computed eigenvalues and selected poles are shown in Figure 3.6. The order in which the poles are chosen is given by the numbers 1 to 6. Note that the second and third poles are chosen too far away from the wanted eigenvalues. The 50 eigenvalues lying in the trust interval $[7.3\ 10^{-5}, 11.5]$ were computed. Five factorizations and a total of 270 linear solves were needed.

Figure 3.6: Computed Ritz values (dots) and selected poles (ticks) for problem `BCSST27`.

Full details of `EA16` as well as a number of numerical examples illustrating its performance on a range of problems are given by Meerbergen and Scott (2000).

References

K. Meerbergen and J. Scott. Design and development of a block rational Lanczos method with partial reorthogonalization and implicit restarting. Technical Report RAL-TR-2000-011, Rutherford Appleton Laboratory, Oxfordshire, 2000.

4 Partial Differential Equations

4.1 Null space algorithms for mixed finite-element approximation of Darcy's equation (M. Arioli and G. Manzini)

Let Ω be a simply connected, bounded, polygonal domain in $\mathbb{R}^2$, defined by a closed curve Γ. Γ is usually the union of two parts Γ_D and Γ_N, where different Dirichlet and Neumann type boundary conditions are imposed, that is $\Gamma = \Gamma_D \cup \Gamma_N$.

Darcy's laws can be formulated as follows:

$$\begin{cases} \mathbf{u}(\mathbf{x}) &= -K(\mathbf{x})\mathrm{grad}\, p(\mathbf{x}), & \mathbf{x} \in \Omega \\ \mathrm{div}\, \mathbf{u}(\mathbf{x}) &= f(\mathbf{x}), & \mathbf{x} \in \Omega \end{cases} \tag{4.1}$$

with a set of boundary conditions for both $\mathbf{u}$ and p:

$$\begin{cases} p(\mathbf{x}) &= g_D(\mathbf{x}), & \mathbf{x} \in \Gamma_D \\ \mathbf{u} \cdot \mathbf{n} &= g_N(\mathbf{x}), & \mathbf{x} \in \Gamma_N \end{cases} \tag{4.2}$$

using two regular functions g_D and g_N for Dirichlet and Neumann conditions, and where $\mathbf{n}$ denotes the external normal to Γ. In the following, we will assume that $g_N = 0$.

Darcy's law describes the relationship between the pressure $p(\mathbf{x})$ (the total head) and the velocity field $\mathbf{u}(\mathbf{x})$ (the visible effect) in ground-water flow. In system (4.1), $K(\mathbf{x})$ is the hydraulic conductivity tensor and $f(\mathbf{x})$ is a source-sink term.

The former equation in (4.1) relates the vector field $\mathbf{u}$ to the scalar field p via the permeability tensor K, which accounts for the soil characteristics. The latter equation in (4.1) relates the divergence of $\mathbf{u}$ to the source-sink term $f(\mathbf{x})$.

Let $\mathcal{T}_h$ be a family of triangulations of Ω, i.e. each $\mathcal{T}_h$ is a set of disjoint triangles τ which cover Ω in such a way that no vertex of any triangle lies in the interior of an edge of another triangle. Let $h = \max\{\mathrm{diam}\,(\tau) : \tau \in \mathcal{T}_h\}$. The mixed finite element approximation of system (4.1) with the usual Raviart-Thomas space leads to the solution of the following system of linear equations:

$$\begin{bmatrix} \mathbf{M} & \mathbf{A} \\ \mathbf{A^T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{q} \\ \mathbf{b} \end{bmatrix}, \tag{4.3}$$

where, denoting by n the number of edges and by m the number of triangles, we have that $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a symmetric and positive-definite matrix, and that $\mathbf{A} \in \mathbb{R}^{n \times m}$ is a submatrix of a totally unimodular matrix with $m + 1$ columns. Therefore, $\mathbf{A}$ is full rank and its entries are equal to either 1, -1, or 0. Moreover, the augmented system (4.3) is nonsingular because $\mathrm{Ker}\left(\mathbf{A^T}\right) \cap \mathrm{Ker}(\mathbf{M}) = \mathbf{0}$.

The classical null space algorithm (see Gill, Murray and Wright, 1981), for the minimisation of linearly constrained quadratic forms can be described as follows.

Let $\mathbf{Y} \in \mathbb{R}^{n \times m}$ and $\mathbf{Z} \in \mathbb{R}^{n \times (n-m)}$ be two matrices such that

$$\mathbf{Y}^{\mathbf{T}}\mathbf{A} = \mathbf{I}_m \qquad \text{and} \qquad \mathbf{Z}^{\mathbf{T}}\mathbf{A} = \mathbf{0}_{n-m,m}.$$

Null Space Algorithm:

1. $\mathbf{u}_0 = \mathbf{Y}\mathbf{b}$,
2. $\mathbf{Z}^{\mathbf{T}}\mathbf{M}\mathbf{Z}\,\mathbf{w} = \mathbf{Z}^{\mathbf{T}}\mathbf{q} - \mathbf{Z}^{\mathbf{T}}\mathbf{M}\mathbf{u}_0 = \mathbf{s}$,
3. $\mathbf{u} = \mathbf{u}_0 + \mathbf{Z}\mathbf{w}$,
4. $\mathbf{p} = \mathbf{Y}^{\mathbf{T}}\mathbf{q} - \mathbf{Y}^{\mathbf{T}}\mathbf{M}\mathbf{u}$.

The matrices $\mathbf{Y}$ and $\mathbf{Z}$ can be computed using either an orthogonal factorisation or a Gaussian factorisation of the matrix $\mathbf{A}$. In Step 2, we also need to solve a system involving $\mathbf{Z}^{\mathbf{T}}\mathbf{M}\mathbf{Z}$, the projected Hessian matrix. We have two alternative ways to proceed. If $n - m$ is small (the number of constraints is very close to the number of unknowns), or the projected Hessian matrix is still sparse, we can explicitly compute $\mathbf{Z}^{\mathbf{T}}\mathbf{M}\mathbf{Z}$, and then solve the system $\mathbf{Z}^{\mathbf{T}}\mathbf{M}\mathbf{Z}\,\mathbf{w} = \mathbf{s}$ using the Cholesky factorisation. Otherwise, when the product $\mathbf{Z}^{\mathbf{T}}\mathbf{M}\mathbf{Z}$ cannot be performed directly because both the complexity would be too high – $\mathcal{O}(n^3)$ – or the resulting matrix would be fairly dense, despite the sparsity of $\mathbf{M}$, we can solve the linear system $\mathbf{Z}^{\mathbf{T}}\mathbf{M}\mathbf{Z}\,\mathbf{w} = \mathbf{s}$ using a conjugate gradient algorithm, and implicitly compute the matrix by vector products. Within the conjugate gradient algorithm, we use the stopping criterion described by Arioli (2000) (see Section 3.6). At iteration j, this stopping criterion estimates the energy norm of the error at step $j - d$, where d is an *a priori* selected integer value. If the error is less than η times the energy norm of $\mathbf{w}^{(j)}$, we stop. The choice of η will depend on the properties of the problem that we want to solve, and, in practical cases $\eta >> \epsilon$, where ϵ is the rounding unit. Following the results described by Arioli (2000), we chose η equal to h.

References

M. Arioli. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. Technical Report Tech. Report IAN-1179, IAN, 2000.

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

4.1.1 The Householder approach

Arioli and Manzini (2002) have analysed the Householder approach. Amestoy, Duff and Puglisi (1996) have shown that, by a sparse version of the Householder algorithm, the

matrix $\mathbf{A}$ can be factorized as

$$\mathbf{A} = \mathbf{H} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}$$

where $\mathbf{R}$ is an $m \times m$ sparse, non singular, upper triangular matrix and $\mathbf{H}$ is an $n \times n$ orthonormal matrix. The matrix $\mathbf{H}$ can be stored implicitly as the set of sparse vectors that generate the elementary Householder transformations, see (Amestoy et al., 1996) for details on the sparsity of $\mathbf{R}$ and on the sparse storage of $\mathbf{H}$. With respect to the previous choice, we have

$$\mathbf{Y} = \mathbf{H}\mathbf{E}_1\mathbf{R}^{-T} \qquad \text{and} \qquad \mathbf{Z} = \mathbf{H}\mathbf{E}_2.$$

The matrix $\mathbf{Z}$ is an orthonormal basis of the kernel of $\mathbf{A}^{\mathbf{T}}$. It is very important to observe that we never need to explicitly compute either $\mathbf{Y}$ or $\mathbf{Z}$. Indeed the Householder factorisation gives the possibility of using its sparse result for implicitly computing all the matrix-vector products required by the algorithm. We can perform the product of the projected Hessian matrix $\mathbf{Z^T M Z}$ by a vector and the product of $\mathbf{Y}$ or $\mathbf{Y^T}$ by a vector in the following ways:

$$\begin{aligned} \mathbf{Z^T M Z}\,\mathbf{w} &= \mathbf{E^T}\,(\mathbf{H^T}\,(\mathbf{M}\,(\mathbf{H}\,(\mathbf{E}_2\mathbf{w})\,)\,)\,), \\ \mathbf{Y^T y} &= \mathbf{R}^{-1}\,(\mathbf{E}_1^{\mathbf{T}}\,(\mathbf{H^T y})\,), \\ \mathbf{Y x} &= \mathbf{H}\,(\mathbf{E}_1\,(\mathbf{R}^{-\mathbf{T}}\mathbf{x})\,). \end{aligned}$$

This approach has the advantage of performing backward and forward substitutions for sparse triangular matrices, and of using the sparse Householder elementary matrices to perform matrix-vector products.

In Arioli and Manzini (2002), the experiments were performed on a SUN workstation using the sparse HSL code `MA49`, described by Amestoy et al. (1996). Arioli and Manzini (2002) used a L-shape domain Ω with a discontinuous $K(\mathbf{x})$ which takes two values $K_0 = 1$ and K_1. In the test problems, the values of K_1 range from 10^{-2} to 10^{-8}. We compared the exact solution $\mathbf{u}$ and $\mathbf{p}$ of (4.3) with the values $\mathbf{u}^*$ and $\mathbf{p}^*$ computed by the null space algorithm where, in step 2, we used the conjugate gradient method, and we chose $\mathbf{w}^{(0)} = \mathbf{0}$. We assume that the values computed by the HSL routine `MA47` which implements a sparse Gaussian factorisation applied to (4.3) are exact.

In Table 4.1, we show the values n, m (number of degrees of freedom of the problem), and the number of nonzero entries in $\mathbf{M}$ and $\mathbf{A}$ for three meshes. Moreover, in Table 4.1, we show the storage (measured in words) needed for $\mathbf{H}$ and $\mathbf{R}$, and on the values of η.

In Table 4.2, we summarise the numerical performance of the algorithm for each test problem and each mesh when, before using `MA49`, we perform a symmetric scaling on the problem. After the scaling, the diagonal entries of the scaled $\mathbf{M}$ are equal to 1.

	n	m	$nnz(A)$	$nnz(M)$	$nnz(R)$	$stg(H)$	η
Mesh 1	237	147	416	1119	940	3023	0.16
Mesh 2	2309	1497	4396	11291	16058	47120	$5.8\,10^{-2}$
Mesh 3	22919	15136	45084	113735	250944	649793	$1.92\,10^{-2}$

Table 4.1: Parameters of the runs.

Mesh		$K_1 = 10^{-2}$	$K_1 = 10^{-4}$	$K_1 = 10^{-6}$	$K_1 = 10^{-8}$
1	N_{iter}	6	6	6	6
	$\frac{\|\|u-u^*\|\|_2}{\|\|u\|\|_2}$	$1.1\,10^{-5}$	$4.0\,10^{-6}$	$2.0\,10^{-6}$	$1.9\,10^{-6}$
	$\frac{\|\|p-p^*\|\|_2}{\|\|p\|\|_2}$	$4.4\,10^{-7}$	$1.5\,10^{-6}$	$1.6\,10^{-6}$	$1.6\,10^{-6}$
2	N_{iter}	7	7	7	7
	$\frac{\|\|u-u^*\|\|_2}{\|\|u\|\|_2}$	$1.4\,10^{-6}$	$5.4\,10^{-6}$	$4.5\,10^{-6}$	$4.5\,10^{-6}$
	$\frac{\|\|p-p^*\|\|_2}{\|\|p\|\|_2}$	$2.8\,10^{-6}$	$1.0\,10^{-6}$	$1.1\,10^{-6}$	$1.1\,10^{-6}$
3	N_{iter}	8	8	8	8
	$\frac{\|\|u-u^*\|\|_2}{\|\|u\|\|_2}$	$4.5\,10^{-6}$	$4.4\,10^{-6}$	$4.4\,10^{-6}$	$4.4\,10^{-6}$
	$\frac{\|\|p-p^*\|\|_2}{\|\|p\|\|_2}$	$1.8\,10^{-8}$	$1.9\,10^{-8}$	$1.9\,10^{-8}$	$1.9\,10^{-8}$

Table 4.2: Stopping iteration and final residual using symmetric scaling ($d = 5$)

We can see from the numerical experiments that the null space algorithm based on the Householder factorisation gives a projected Hessian matrix asymptotically independent of n and m and, thus, of the parameter h. We point out that, for 3D problems, the computer memory requirement for storing $\mathbf{H}$ can become prohibitive.

References

P.R. Amestoy, I.S. Duff, and C. Puglisi. Multifrontal QR factorization in a multiprocessor environment. *Numerical Linear Algebra with Appl.*, **3**, 275–300, 1996.

M. Arioli and G. Manzini. A null space algorithm for mixed finite-element approximations of Darcy's equation. *Commun. Numer. Meth. Engng*, **18**, 2002.

4.1.2 The network programming approach

Arioli and Manzini (2001) have carefully analysed the structure of the matrix $\mathbf{A}$ in equation (4.3). $\mathbf{A}$ is the nonsingular submatrix of a totally unimodular matrix, and its

$\mathbf{LU}$ factorization can be computed by identifying a spanning tree of a graph built using the mesh structure. Moreover, the $\mathbf{LU}$ factorization can be computed without fill-in by permuting the rows and the columns of the matrix. As for the orthogonal factorization approach, we do not need to compute explicitly the matrices $\mathbf{Z} = \mathbf{L}^{-\mathbf{T}}\mathbf{E}_2$ and $\mathbf{Y} = \mathbf{L}^{-\mathbf{T}}\mathbf{E}_1$, and we can perform the product $\mathbf{Z^T M Z}$ by a vector and the product of $\mathbf{Y}$ or $\mathbf{Y^T}$ by a vector by means of solving lower and upper triangular systems.

The test problem has a unit square domain Ω and $K(\mathbf{x})$ has a random distribution. The values of $K(\mathbf{x})$ range from 1 to 10^{-12}. In Table 4.3, we report on the values n, m (number of degrees of freedom of the problem), and the number of nonzero entries in $\mathbf{M}$ and $\mathbf{A}$ for four meshes. In Table 4.4, we show some of the results from Arioli and Manzini (2001). In the test runs, we compare the performance of our approach with the performance of the HSL code `MA47`. The package `MA47` implements a version of the $\mathbf{LDL^T}$ decomposition for symmetric indefinite matrices that takes advantage of the structure of the augmented system. The package is divided into three parts corresponding to the symbolic analysis where the reordering of the matrix is computed, the factorization phase, and the final solution using the triangular matrices.

Similarly, the null space algorithm can be subdivided into three phases: a first symbolic phase where the shortest path tree and the quotient tree are computed, a second phase where the projected Hessian system is solved by the conjugate gradient algorithm, and a final third phase where we compute the pressure. This enables us to compare the direct solver `MA47` with the null space approach in each single phase.

Generally, in the test runs that we will present, we fix the parameter d in the stopping criterion to the value of 10.

	Mesh 1	Mesh 2	Mesh 3	Mesh 4
m	153	1567	15304	155746
n	246	2406	23130	234128
$nnz(M)$	1164	11808	114954	1168604
$nnz(A)$	429	4599	45601	466319
h	0.2090	0.0649	0.0225	0.0069

Table 4.3: Data relative to the meshes for a boxed domain with random permeability.

Even if our implementation can be 10 times slower than the direct solver, the absence of fill-in makes our code competitive for large problems. In particular, we want to highlight that the absence of fill-in is promising when we need to solve Darcy's equations in 3D domains. One might expect that the fill-in and complexity of a direct solver, applied to the augmented systems related to the approximation of Darcy's equations in 3D domains, will grow as $\mathcal{O}(m^{2/3})$ and $\mathcal{O}(m^2)$ respectively. Our algorithm will instead have no fill-

Mesh	MA47				null space algorithm		
	Symb.	Fact.	Sol.	Stor.	Symb.	CG(#Iterations)	Sol.
1	0.008	0.013	0.001	0.048	0.002	0.013 (12)	0.002
2	0.088	0.173	0.006	0.634	0.017	0.145 (19)	0.018
3	1.058	3.028	0.114	9.15	0.911	4.681 (41)	0.290
4	14.63	264.6	1.543	132.32	7.8	210.6 (176)	2.941

Table 4.4: MA47 vs null space algorithm: CPU times (in seconds) and storage (in MBytes).

in and its complexity will only depend on the condition number of the scaled projected Hessian matrix $\mathbf{Z^T M Z}$. We can reasonably assume that this condition number will not change with the dimension of the domain Ω, analogous to the behaviour of the classical finite-element method. Therefore, the stopping criterion will stop the conjugate gradient algorithm after a number of steps which will not depend on the dimension of the domain Ω.

References

M. Arioli and G. Manzini. A network programming approach in solving Darcy's equations by mixed finite-element methods. Technical Report RAL-TR-2001-037, Rutherford Appleton Laboratory, Oxfordshire, 2001.

4.1.3 Mixed-Hybrid approximation

Finally, in Arioli, Maryška, Rozložník and Tůma (2001), we compare the orthogonal and Gaussian approaches, when we approximate Darcy's equation by mixed-hybrid finite-element method. In Arioli et al. (2001), we use a set of 3D test problems coming from underground water flow modelling in the Stráž pod Ralskem uranium mine.

References

M. Arioli, J. Maryška, M. Rozložník, and M. Tůma. Dual variable methods for mixed-hybrid finite element approximation of the potential fluid flow problem in porous media. Technical Report RAL-TR-2001-023, Rutherford Appleton Laboratory, Oxfordshire, 2001.

5 Optimization

5.1 Quadratic Programming (N. I. M. Gould and Ph. L. Toint)

We have now finished developing two new quadratic programming (QP) methods, both of which lie at the heart of our forthcoming nonlinear programming library GALAHAD (see Section 5.6), and which are also available as part of HSL. Both algorithms are designed to handle large, sparse, nonconvex problems, the ultimate aim being to solve problems in hundreds of thousands of unknowns.

The *quadratic programming* problem is to

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}) \equiv \tfrac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{g}^T\mathbf{x} \tag{5.1}$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T\mathbf{x} \leq c_i^u, \quad i = 1, \ldots, m, \tag{5.2}$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1, \ldots, n. \tag{5.3}$$

for given vectors $\mathbf{g}$, $\mathbf{a}_i$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, $\mathbf{x}^u$ and a given symmetric (but not necessarily definite) matrix $\mathbf{H}$. The required solution $\mathbf{x}$ to (5.1)–(5.3) necessarily satisfies the primal optimality conditions

$$\mathbf{A}\mathbf{x} = \mathbf{c} \tag{5.4}$$

and

$$\mathbf{c}^l \leq \mathbf{c} \leq \mathbf{c}^u, \;\; \mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u, \tag{5.5}$$

the dual optimality conditions

$$\mathbf{H}\mathbf{x} + \mathbf{g} = \mathbf{A}^T\mathbf{y} + \mathbf{z}, \;\; \mathbf{y} = \mathbf{y}^l + \mathbf{y}^u \;\text{ and }\; \mathbf{z} = \mathbf{z}^l + \mathbf{z}^u, \tag{5.6}$$

and

$$\mathbf{y}^l \geq 0, \;\; \mathbf{y}^u \leq 0, \;\; \mathbf{z}^l \geq 0 \;\text{ and }\; \mathbf{z}^u \leq 0, \tag{5.7}$$

and the complementary slackness conditions

$$(\mathbf{A}\mathbf{x} - \mathbf{c}^l)^T\mathbf{y}^l = 0, \;\; (\mathbf{A}\mathbf{x} - \mathbf{c}^u)^T\mathbf{y}^u = 0, \;\; (\mathbf{x} - \mathbf{x}^l)^T\mathbf{z}^l = 0 \;\text{ and }\; (\mathbf{x} - \mathbf{x}^u)^T\mathbf{z}^u = 0, \tag{5.8}$$

where the vectors $\mathbf{y}$ and $\mathbf{z}$ are known as the Lagrange multipliers for the general linear constraints, and the dual variables for the bounds, respectively, and where the vector inequalities hold componentwise.

5.1.1 An Interior-Point Approach

Our first method is based on a feasible interior-point trust-region approach. Primal-dual interior point methods iterate towards a point that satisfies the optimality conditions (5.4)–(5.8) by ultimately aiming to satisfy (5.4), (5.6) and (5.8), while ensuring that (5.5) and (5.7) are satisfied as strict inequalities at each stage. Appropriate norms of the amounts by which (5.4), (5.6) and (5.8) fail to be satisfied are known as the primal and dual infeasibility, and the violation of complementary slackness, respectively. The fact that (5.5) and (5.7) are satisfied as strict inequalities gives such methods their name, interior-point methods.

The problem is solved in two phases. The goal of the first "initial feasible point" phase is to find a strictly interior point which is primal feasible, that is that (5.4) is satisfied. The HSL package `VE13` (or `LSQP` in GALAHAD) is used for this purpose, and offers the options of either accepting the first strictly feasible point found, or preferably of aiming for the so-called "analytic centre" of the feasible region. Having found such a suitable initial feasible point, the second "optimality" phase ensures that (5.4) remains satisfied while iterating to satisfy dual feasibility (5.6) and complementary slackness (5.8). The optimality phase proceeds by approximately minimizing a sequence of barrier functions

$$q(\mathbf{x}) - \mu \left[\sum_{i=1}^{m} \log(c_i - c_i^l) + \sum_{i=1}^{m} \log(c_i^u - c_i) + \sum_{j=1}^{n} \log(x_j - x_j^l) + \sum_{j=1}^{n} \log(x_j^u - x_j) \right],$$

for an appropriate sequence of positive barrier parameters μ converging to zero while ensuring that (5.4) remain satisfied and that $\mathbf{x}$ and $\mathbf{c}$ are strictly interior points for (5.5). Note that terms in the above summations corresponding to infinite bounds are ignored, and that equality constraints are treated specially.

Each of the barrier subproblems is solved using a trust-region method. Such a method generates a trial correction step $\Delta(\mathbf{x}, \mathbf{c})$ to the current iterate $(\mathbf{x}, \mathbf{c})$ by replacing the nonlinear barrier function locally by a suitable quadratic model, and approximately minimizing this model in the intersection of (5.4) and a trust region $\|\Delta(\mathbf{x}, \mathbf{c})\| \leq \Delta$ for some appropriate strictly positive trust-region radius Δ and norm $\|\cdot\|$. The step is accepted/rejected and the radius adjusted on the basis of how accurately the model reproduces the value of barrier function at the trial step. If the step proves to be unacceptable, a linesearch is performed along the step to obtain an acceptable new iterate. In practice, the natural primal "Newton" model of the barrier function is frequently less successful than an alternative primal-dual model, and consequently the primal-dual model is usually to be preferred.

The trust-region subproblem is approximately solved using the combined conjugate-gradient/Lanczos method (see Gould, Hribar and Nocedal, 2001 and Gould, Lucidi, Roma and Toint, 1999*a*) implemented in the HSL code `VF05` (`GLTR` in GALAHAD). Such a method requires a suitable preconditioner, and in our case, the only flexibility we have is in

approximating the model of the Hessian. Although using a fixed form of preconditioning is sometimes effective, we have provided the option of an automatic choice, that aims to balance the cost of applying the preconditioner against the needs for an accurate solution of the trust-region subproblem. The preconditioner is applied using the HSL factorization packages `MA27` or `MA57`, but options at this stage are to factorize the preconditioner as a whole (the so-called "augmented system" approach), or to perform a block elimination first (the "Schur-complement" approach). The latter is usually to be preferred when a (nonsingular) diagonal preconditioner is used, but may be inefficient if any of the columns of $\mathbf{A}$ is too dense.

The theoretical justification of the overall scheme, for problems with general objectives and inequality constraints, is given by Conn, Gould, Orban and Toint (2000*b*), in which we also present numerical results that suggest that it is indeed able to solve some problems of the size we had been aiming for. More recently, we investigated the ultimate rate of convergence of such schemes, and have shown that, under fairly general conditions, a componentwise superlinear rate is achievable both for quadratic and general nonlinear programs (see Gould, Orban, Sartenaer and Toint, 1999*b*).

The method has been implemented as a Fortran 90 module `HSL_VE12` in HSL, while an updated version will shortly be available as `QPB` in the GALAHAD library. Full advantage is taken of any zero coefficients in the matrix $\mathbf{H}$ or the vectors $\mathbf{a}_i$, while any of the constraint bounds c_i^l, c_i^u, x_j^l and x_j^u may be infinite.

References

A. R. Conn, N. I. M. Gould, D. Orban, and Ph. L. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, **87**(2), 215–249, 2000.

N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic problems arising in optimization. *SIAM Journal on Scientific Computing*, **23**(4), 1375–1394, 2001.

N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, **9**(2), 504–525, 1999*a*.

N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Superlinear convergence of primal-dual interior-point algorithms for nonlinear programming. *SIAM Journal on Optimization*, **11**(4), 974–1002, 1999*b*.

5.1.2 An Active-Set Approach

Our second method (Gould and Toint, 2001) is of the active-set variety, and, although general in scope, is intended within GALAHAD to deal with the case where a good estimate of the optimal active set has been determined (and thus that relatively few iterations will be required). The method is actually more general in scope, and is geared towards solving ℓ_1 *quadratic programming* problems of the form

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}) + \rho_g v_g(\mathbf{x}) + \rho_b v_b(\mathbf{x}) \tag{5.9}$$

involving the quadratic objective $q(\mathbf{x})$ and the infeasibilities

$$v_g(\mathbf{x}) = \sum_{i=1}^{m} \max(c_i^l - \mathbf{a}_i^T\mathbf{x}, 0) + \sum_{i=1}^{m} \max(\mathbf{a}_i^T\mathbf{x} - c_i^u, 0)$$

and

$$v_b(\mathbf{x}) = \sum_{j=1}^{n} \max(x_j^l - x_j, 0) + \sum_{j=1}^{n} \max(x_j - x_j^u, 0).$$

At the k-th iteration of the method, an improvement to the value of the merit function $m(\mathbf{x}, \rho_g, \rho_b) = q(\mathbf{x}) + \rho_g v_g(\mathbf{x}) + \rho_b v_b(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}^{(k)}$ is sought. This is achieved by first computing a search direction $\mathbf{s}^{(k)}$, and then setting $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$, where the stepsize $\alpha^{(k)}$ is chosen as the first local minimizer of $\phi(\alpha) = m(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)}, \rho_g, \rho_b)$ as α increases from zero. The stepsize calculation is straightforward, and exploits the fact that $\phi(\alpha)$ is a piecewise quadratic function of α.

The search direction is defined by a subset of the "active" terms in $v(\mathbf{x})$, i.e., those for which $\mathbf{a}_i^T\mathbf{x} = c_i^l$ or c_i^u (for $i = 1, \ldots, m$) or $x_j = x_j^l$ or x_j^u (for j=1, ... ,n). The "working" set $W^{(k)}$ is chosen from the active terms, and is such that its members have linearly independent gradients. The search direction $\mathbf{s}^{(k)}$ is chosen as an approximate solution of the equality-constrained quadratic program

$$\underset{\mathbf{s}\in\mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}^{(k)} + \mathbf{s}) + \rho_g l_g^{(k)}(\mathbf{s}) + \rho_b l_b^{(k)}(\mathbf{s}), \tag{5.10}$$

subject to

$$\mathbf{a}_i^T\mathbf{s} = 0, \;\; i \in \{1, \ldots, m\} \cap W^{(k)}, \;\; \text{and} \;\; x_j = 0, \;\; i \in \{1, \ldots, n\} \cap W^{(k)}, \tag{5.11}$$

where

$$l_g^{(k)}(\mathbf{s}) = -\sum_{\substack{i=1 \\ \mathbf{a}_i^T\mathbf{x} < c_i^l}}^{m} \mathbf{a}_i^T\mathbf{s} \; + \sum_{\substack{i=1 \\ \mathbf{a}_i^T\mathbf{x} > c_i^u}}^{m} \mathbf{a}_i^T\mathbf{s}$$

and

$$l_b^{(k)}(\mathbf{s}) = -\sum_{\substack{j=1 \\ x_j < x_j^l}}^{n} s_j \; + \sum_{\substack{j=1 \\ x_j > x_j^u}}^{n} s_j.$$

The equality-constrained quadratic program (5.10)–(5.11) is solved by a projected preconditioned conjugate gradient method (see Gould, Hribar and Nocedal, 2001). The method terminates either after a prespecified number of iterations, or if the solution is found, or if a direction of infinite descent, along which $q(\mathbf{x}^{(k)} + \mathbf{s}) + \rho_g l_g^{(k)}(\mathbf{s}) + \rho_b l_b^{(k)}(\mathbf{s})$ decreases without bound within the feasible region (5.11), is located. Successively more accurate approximations are required as suspected solutions of (5.9) are approached.

Preconditioning of the conjugate gradient iteration requires the solution of one or more linear systems of the form

$$\begin{pmatrix} \mathbf{M}^{(k)} & \mathbf{A}^{(k)T} \\ \mathbf{A}^{(k)} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ \mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{g} \\ 0 \end{pmatrix}, \tag{5.12}$$

where $\mathbf{M}^{(k)}$ is a "suitable" approximation to $\mathbf{H}$ and the rows of $\mathbf{A}^{(k)}$ comprise the gradients of the terms in the current working set. Rather than recomputing a factorization of the preconditioner at every iteration, a Schur complement method is used, recognising the fact that gradual changes occur to successive working sets. The main iteration is divided into a sequence of "major" iterations. At the start of each major iteration (say, the overall iteration l), a factorization of the current "reference" matrix, that is the matrix

$$\begin{pmatrix} \mathbf{M}^{(l)} & \mathbf{A}^{(l)T} \\ \mathbf{A}^{(l)} & 0 \end{pmatrix} \tag{5.13}$$

is obtained using the HSL matrix factorization package `MA57` (in the GALAHAD version, the older HSL code `MA27` is used instead). This reference matrix may be factorized as a whole, using the augmented system approach, or by performing a block elimination first, using the "Schur-complement" approach. The latter is usually to be preferred when a (nonsingular) diagonal preconditioner is used, but may be inefficient if any of the columns of $\mathbf{A}^{(l)}$ is too dense. Subsequent iterations within the current major iteration obtain solutions to (5.12) via the factors of (5.13) and an appropriate (dense) Schur complement, obtained from the HSL package `MA69` (`SCU` in GALAHAD). The major iteration terminates once the space required to hold the factors of the (growing) Schur complement exceeds a given threshold.

The working set changes by (a) adding an active term encountered during the determination of the stepsize, or (b) the removal of a term if $\mathbf{s} = 0$ solves (5.10)–(5.11). The decision on which to remove in the latter case is based upon the expected decrease upon the removal of an individual term, and this information is available from the magnitude and sign of the components of the auxiliary vector $\mathbf{u}$ computed in (5.12). At optimality, the

components of $\mathbf{u}$ for $\mathbf{a}_i$ terms will all lie between 0 and ρ_g—and those for the other terms between 0 and ρ_b—and any violation of this rule indicates further progress is possible.

To solve quadratic programs of the form (5.1)–(5.3), a sequence of problems of the form (5.9) are solved, each with a larger value of ρ_g and/or ρ_b than its predecessor. The required solution has been found once the infeasibilities $v_g(\mathbf{x})$ and $v_b(\mathbf{x})$ have been reduced to zero at the solution of (5.9) for the given ρ_g and ρ_b.

The method has been implemented as a Fortran 90 module `HSL_VE19` in HSL, while, again, an updated version will shortly be available, as `QPA`, in the GALAHAD library.

References

N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic problems arising in optimization. *SIAM Journal on Scientific Computing*, **23**(4), 1375–1394, 2001.

N. I. M. Gould and Ph. L. Toint. An iterative working-set method for large-scale non-convex quadratic programming. Technical Report RAL-TR-2001-026, Rutherford Appleton Laboratory, Oxfordshire, 2001.

5.1.3 Comparing the two approaches

Having proposed and implemented two very different quadratic programming methods, an obvious question is: how do the methods compare? We examined this question in Gould and Toint (2001) by comparing `QPA` and `QPB` on the CUTE QP test set.

While for modest sized problems, started from "random" points, the two methods are roughly comparable, the advantages of the interior-point approach become quite clear when problem dimensions increase. For problems involving tens of thousands of unknowns and/or constraints, our active set approach simply takes too many iterations, while the number of iterations required by the interior point approach seems relatively insensitive to dimension size. For general problems involving hundreds of thousands or even millions of unknowns/constraints, the active set approach is impractical, while we illustrate in Table 5.1 that `QPB` is able to solve problems of this size.

While such figures might seem to indicate that `QPB` should always be preferred to `QPA`, this is not the case. In particular, if a good estimate of the solution—more particularly, the optimal active set— is known, active-set methods may exploit this, while interior-point methods are (currently) less able to do so. In particular, Gould and Toint (2001) illustrate that `QPA` often outperforms `QPB` on warm-started problems unless the problem is (close to) degenerate or very ill conditioned. Thus, since nonlinear optimization (SQP) algorithms

Name	n	m	type	its	time
QPBAND	100000	50000	C	13	157
QPBAND	200000	100000	C	17	1138
QPBAND	400000	200000	C	17	2304
QPBAND	500000	250000	C	17	2909
QPNBAND	100000	50000	NC	12	32
QPNBAND	200000	100000	NC	13	71
QPNBAND	400000	200000	NC	14	156
QPNBAND	500000	250000	NC	13	181
PORTSQP	10	1	C	11	0.02
PORTSQP	100	1	C	15	0.03
PORTSQP	1000	1	C	26	0.09
PORTSQP	10000	1	C	37	1.26
PORTSQP	100000	1	C	20	9.48
PORTSQP	1000000	1	C	11	72.31
PORTSNQP	10	2	NC	21	0.03
PORTSNQP	100	2	NC	30	0.04
PORTSNQP	1000	2	NC	39	0.17
PORTSNQP	10000	2	NC	32	1.70
PORTSNQP	100000	2	NC	107	58.69
PORTSNQP	1000000	2	NC	22	209.53

Table 5.1: GALAHAD QPB on large QP examples. Runs performed on a Compaq AlphaServer DS20 (3.5 Gbytes RAM), time in CPU seconds. n is the number of unknowns, and m is the number of general constraints. C indicates a convex problem, while NC is a non-convex one.

often solve a sequence of related problems for which the optimal active sets are almost or actually identical, there is good reason to maintain both QPA and QPB in GALAHAD.

References

N. I. M. Gould and Ph. L. Toint. Numerical methods for large-scale non-convex quadratic programming. Technical Report RAL-TR-2001-017, Rutherford Appleton Laboratory, Oxfordshire, 2001.

5.1.4 Preprocessing

The purpose of preprocessing (or presolving) quadratic programming problems is to exploit the optimality equations (5.4)–(5.8) so as to both simplify the problem and reduce the problem to a standard form (that makes subsequent manipulation easier), defined as follows:

- The variables are ordered so that their bounds appear in the order

free			x_j		
non-negativity	0	$\leq$	x_j		
lower	x_j^l	$\leq$	x_j		
range	x_j^l	$\leq$	x_j	$\leq$	x_j^u
upper			x_j	$\leq$	x_j^u
non-positivity			x_j	$\leq$	0

  Fixed variables are removed. Within each category, the variables are further ordered so that those with non-zero diagonal Hessian entries occur before the remainder.

- The constraints are ordered so that their bounds appear in the order

non-negativity	0	$\leq$	$(\mathbf{A}\mathbf{x})_i$		
equality	c_i^l	$=$	$(\mathbf{A}\mathbf{x})_i$		
lower	c_i^l	$\leq$	$(\mathbf{A}\mathbf{x})_i$		
range	c_i^l	$\leq$	$(\mathbf{A}\mathbf{x})_i$	$\leq$	c_i^u
upper			$(\mathbf{A}\mathbf{x})_i$	$\leq$	c_i^u
non-positivity			$(\mathbf{A}\mathbf{x})_i$	$\leq$	0

  Free constraints are removed.

- In addition, constraints may be removed or bounds tightened, to reduce the size of the feasible region or simplify the problem if this is possible, and bounds may be tightened on the dual variables and the multipliers associated with the problem.

The presolving algorithm (see Gould and Toint, 2002) proceeds by applying a (potentially long) series of simple transformations to the problem, each transformation introducing a further simplification of the problem. These involve the removal of empty and singleton rows, the removal of redundant and forcing primal constraints, the tightening of primal and dual bounds, the exploitation of linear singleton, linear doubleton and linearly unconstrained columns, merging of dependent variables, row sparsification and splitting equalities. Transformations are applied in successive passes, each pass involving the following actions:

1. remove empty and singletons rows,
2. try to eliminate variables that are linearly unconstrained,
3. attempt to exploit the presence of linear singleton columns,
4. attempt to exploit the presence of linear doubleton columns,
5. complete the analysis of the dual constraints,
6. remove empty and singletons rows,
7. possibly remove dependent variables,
8. analyse the primal constraints,
9. try to make $\mathbf{A}$ sparser by combining its rows,
10. check the current status of the variables, dual variables and multipliers.

All these transformations are applied on the structure of the original problem, which is only permuted to standard form after all transformations are completed. The reduced problem may then solved by a quadratic or linear programming solver. Finally, the solution of the simplified problem is re-translated to the variables/constraints/format of the original problem in a "restoration" phase.

At the overall level, the presolving process follows one of the following two sequences:

$$\boxed{\text{initialize}} \rightarrow \left[\boxed{\text{apply transformations}} \rightarrow (\text{solve problem}) \rightarrow \boxed{\text{restore}} \right] \rightarrow \boxed{\text{terminate}}$$

or

$$\boxed{\text{initialize}} \rightarrow \left[\boxed{\begin{array}{l}\text{read}\\ \text{specfile}\end{array}} \rightarrow \boxed{\begin{array}{l}\text{apply}\\ \text{transformations}\end{array}} \rightarrow \begin{pmatrix}\text{solve}\\ \text{problem}\end{pmatrix} \rightarrow \boxed{\text{restore}} \right] \rightarrow \boxed{\text{terminate}}$$

where the procedure's control parameter may be modified by reading an external "specfile", and where (solve problem) indicates that the reduced problem is solved. Each of the "boxed" steps in these sequences corresponds to calling a specific routine of the package. In the above diagrams, bracketed subsequences of steps means that they can be repeated with problems having the same structure.

An implementation will shortly be available as `PRESOLVE` in the GALAHAD library. Gould and Toint (2002) indicate that, when considering all 178 linear and quadratic programming problems in the CUTE test set, an average reduction of roughly 20% in both the number of unknowns and the number of constraints results from applying `PRESOLVE`. When applying our interior point QP solver `QPA`, an overall average reduction of roughly 10% in CPU time results. In some cases, the gain is dramatic. For example, for the problems `GMNCASE4`, `STNPQ1`, `STNQP2` and `SOSQP1`, `PRESOLVE` removes all the variables and

constraints, and thus reveals the complete solution to the problem without resorting to a QP solver.

References

N. I. M. Gould and Ph. L. Toint. Preprocessing for quadratic programming. Technical Report RAL-TR-2002-001, Rutherford Appleton Laboratory, Oxfordshire, 2002.

5.2 A backward error analysis of a null space algorithm in sparse quadratic programming (M. Arioli and L. Baldini)

Let $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ be a symmetric and positive-semidefinite matrix, and let $\mathbf{A} \in \mathbb{R}^{n\times m}$, $m \leq n$, be a real full rank matrix, $\mathbf{q} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$. The quadratic programming problem with equality constraints

$$\min_{\mathbf{A^T x = b}} \frac{1}{2}\mathbf{x^T M x} + \mathbf{q^T}\boldsymbol{x} \tag{5.14}$$

has a unique solution $\hat{\mathbf{x}}$ if and only if $\mathrm{Ker}(\mathbf{A^T}) \cap \mathrm{Ker}(\mathbf{M}) = \{\mathbf{0}\}$. Introducing the vector $\mathbf{u} \in \mathbb{R}^m$ of the Lagrangian parameters, the problem (5.14) is equivalent to the augmented system

$$\begin{bmatrix} \mathbf{M} & \mathbf{A} \\ \mathbf{A^T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} -\mathbf{q} \\ \mathbf{b} \end{bmatrix}.$$

The augmented matrix is invertible, and the solution $[\hat{\mathbf{x}}^T, \hat{\mathbf{u}}^T]^T$ is composed of the solution of problem (5.14) and the Lagrangian parameters of the gradient of the objective function at $\hat{\mathbf{x}}$.

Arioli and Baldini (2001) present a roundoff error analysis of a null space method which uses a mixture of direct and iterative solvers. In the following, we will denote the augmented matrix by $\mathfrak{A}$, and by $\mathbf{E}_1$ and $\mathbf{E}_2$ the matrices

$$\mathbf{E}_1 = \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0}_{n-m,m} \end{bmatrix} \qquad \text{and} \qquad \mathbf{E}_2 = \begin{bmatrix} \mathbf{0}_{m,n-m} \\ \mathbf{I}_{n-m} \end{bmatrix}.$$

Given an $n \times m$ matrix $\mathbf{B}$ of entries b_{ij} and an n-vector $\mathbf{v}$ of entries v_i, we will denote by $|\mathbf{B}|$ and $|\mathbf{v}|$ the matrix and the vector whose entries are the absolute values of the entries of $\mathbf{B}$ and $\mathbf{v}$.

Arioli and Baldini (2001) chose a formulation of the null space algorithm based on the factorization of the augmented matrix. The matrix $\mathbf{A}$ can be factorized as

$$\mathbf{PAQ} = \mathbf{L}\begin{bmatrix} \mathbf{U} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_1 & \mathbf{0} \\ \mathbf{L}_2 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{0} \end{bmatrix},$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ is an upper triangular matrix, $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a nonsingular lower triangular matrix generated by the Gaussian algorithm applied to $\mathbf{PAQ}$, and $\mathbf{P}$, $\mathbf{Q}$ are permutation matrices that cope with numerical pivoting and sparsity (Duff, Erisman and Reid 1986). For the sake of simplicity, we will omit $\mathbf{P}$, $\mathbf{Q}$ in the following, assuming that $\mathbf{M}$, $\mathbf{A}$, $\mathbf{q}$, and $\mathbf{b}$ have been consistently permuted. Let

$$\tilde{\mathbf{M}} = \mathbf{L}^{-1}\mathbf{M}\mathbf{L}^{-\mathbf{T}}.$$

The augmented matrix $\mathfrak{A}$ can then be factorized as

$$\mathfrak{A} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}^{\mathbf{T}} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{M}}_{11} & \tilde{\mathbf{M}}_{12} & \mathbf{I}_m \\ \tilde{\mathbf{M}}_{12}^{\mathbf{T}} & \tilde{\mathbf{M}}_{22} & \mathbf{0} \\ \mathbf{I}_m & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{L}^{\mathbf{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U} \end{bmatrix}.$$

Moreover, the matrix $\mathbf{Z} = \mathbf{L}^{-\mathbf{T}}\mathbf{E}_2$ is a nonorthonormal basis of the kernel of $\mathbf{A}^{\mathbf{T}}$.

In the null space algorithm, we also need to solve $\tilde{\mathbf{M}}_{22}\mathbf{x}_2 = \mathbf{E}_2^{\mathbf{T}}\tilde{\mathbf{M}}\mathbf{E}_2\mathbf{x}_2 = \mathbf{p}$. We have two alternative ways of proceeding. If $n - m$ is small (the number of constraints is very close to the number of unknowns) or $\tilde{\mathbf{M}}_{22}$ is still sparse, we can explicitly compute $\tilde{\mathbf{M}}_{22}$ and then solve the system using the Cholesky factorization. Otherwise, we can solve the linear system $\tilde{\mathbf{M}}_{22}\mathbf{x}_2 = \mathbf{p}$ using the conjugate gradient algorithm without explicitly computing $\tilde{\mathbf{M}}_{22}$ and perform the matrix by vector products using the formula

$$\tilde{\mathbf{M}}_{ij}\mathbf{y} = \mathbf{E}_i^{\mathbf{T}}\mathbf{L}^{-1}(\mathbf{M}(\mathbf{L}^{-\mathbf{T}}\mathbf{E}_j\mathbf{y})) \qquad \text{with } (i,j) \;=\; (1,2).$$

This approach has the advantage of performing backward and forward substitution for triangular matrices.

Arioli and Baldini (2001) analysed several variants of the null space algorithm. In particular, they used, within the conjugate gradient method, a stopping criterion similar to the one described in Section 3.6, viz.

$$\text{IF} \quad \|\widehat{\mathbf{M}}_{22}\overline{\mathbf{x}}_2^{(j)} - \tilde{\mathbf{h}}\|_{\widehat{\mathbf{M}}_{22}^{-1}} \leq \eta\|\tilde{\mathbf{h}}\|_{\widehat{\mathbf{M}}_{22}^{-1}} \quad \text{THEN STOP}, \tag{5.15}$$

where $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ are the computed factor $\mathbf{L}$ and $\mathbf{U}$, and $\widehat{\mathbf{M}}_{22} = \mathbf{E}_2^{\mathbf{T}}\widehat{\mathbf{M}}\mathbf{E}_2$ with $\widehat{\mathbf{M}} = \bar{\mathbf{L}}^{-1}\mathbf{M}\bar{\mathbf{L}}^{-\mathbf{T}}$. In practical applications, the threshold η is chosen such that $\varepsilon \ll \eta$, with ε the unit roundoff.

Arioli and Baldini (2001) proved the following theorem, where c_i, $i = 1, 2$ are constants.

Theorem 1 *Let* $\bar{\mathbf{x}}$ *and* $\bar{\mathbf{u}}$ *be the values of* $\mathbf{x}$ *and* $\mathbf{u}$, *solutions of* (2), *computed with the null space algorithm. If* $\varepsilon\|\bar{\mathbf{L}}^{-1}\|_\infty\|\bar{\mathbf{L}}\|_\infty \ll 1$, *then there exist matrices* $\boldsymbol{\delta}\mathbf{M} \in I\!R^{n\times n}$ *and* $\boldsymbol{\delta}\mathbf{A}_1$, $\boldsymbol{\delta}\mathbf{A}_2 \in I\!R^{n\times m}$ *and a vector* $\boldsymbol{\delta}\mathbf{q} \in I\!R^n$ *such that*

$$\begin{bmatrix} \mathbf{M} + \boldsymbol{\delta}\mathbf{M} & \mathbf{A} + \boldsymbol{\delta}\mathbf{A}_1 \\ (\mathbf{A} + \boldsymbol{\delta}\mathbf{A}_2)^{\mathbf{T}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} -(\mathbf{q} + \boldsymbol{\delta}\mathbf{q}) \\ \mathbf{b} \end{bmatrix}.$$

Furthermore,

$$\begin{aligned} |\boldsymbol{\delta}\mathbf{A}_1| &\leq c_1 m\varepsilon(|\mathbf{A}| + |\bar{\mathbf{L}}|\mathbf{E}_1|\bar{\mathbf{U}}|) + \mathcal{O}(\varepsilon^2), \\ |\boldsymbol{\delta}\mathbf{A}_2| &\leq c_1 m\varepsilon(|\mathbf{A}| + |\bar{\mathbf{L}}|\mathbf{E}_1|\bar{\mathbf{U}}|) + \mathcal{O}(\varepsilon^2). \end{aligned}$$

If we use an iterative solver with (5.15) *and a threshold* η*,*

$$|\boldsymbol{\delta}\mathbf{M}| \leq c_2\varepsilon\mathbf{H}(|\mathbf{M}| + |\bar{\mathbf{L}}||\widehat{\mathbf{M}}||\bar{\mathbf{L}}|^{\mathbf{T}})\mathbf{H}^{\mathbf{T}} + \mathcal{O}(\varepsilon^2),$$

$$\|\boldsymbol{\delta}\mathbf{q}\|_{\mathbf{E}_2\widehat{\mathbf{M}}_{22}^{-1}\mathbf{E}_2^{\mathbf{T}}} \leq \eta\|\overline{\mathbf{x}}_2\|_{\widehat{\mathbf{M}}_{22}} + \mathcal{O}(\eta^2),$$

where $\mathbf{H} = \mathbf{I} + \mathbf{E}_2\mathbf{E}_2^{\mathbf{T}}|\bar{\mathbf{L}}^{-1}|\mathbf{E}_1\mathbf{E}_1^{\mathbf{T}}$.

In the numerical experiments, Arioli and Baldini (2001) used augmented systems obtained from the modelling of electrical networks and the numerical results agree with the theory.

References

M. Arioli and L. Baldini. A backward error analysis of a null space algorithm in sparse quadratic programming. *SIAM J. Matrix Anal. and Applics.*, **23**, 425–442, 2001.

I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, 1986.

5.3 Use of MA57 in optimization packages (I. S. Duff)

In interior point methods for the solution of nonlinear programming problems, a vital subproblem is

$$\min \frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{H}\mathbf{x} + \mathbf{g}^{\mathbf{T}}\mathbf{x}, \text{ subject to } \mathbf{A}\mathbf{x} = \mathbf{0} \tag{5.16}$$

for which the coefficient matrix is of the form

$$\begin{pmatrix} \mathbf{D}+\mathbf{H} & \mathbf{A}^{\mathbf{T}} \\ \mathbf{A} & \mathbf{0} \end{pmatrix}, \tag{5.17}$$

where $\mathbf{D}$ is a positive-semidefinite matrix used to penalize inequality constraints.

A common solution scheme is to perform conjugate gradient iterations respecting the constraint $\mathbf{A}\mathbf{x} = \mathbf{0}$ and to use

$$\begin{pmatrix} \mathbf{B} & \mathbf{A}^{\mathbf{T}} \\ \mathbf{A} & \mathbf{0} \end{pmatrix}. \tag{5.18}$$

as a preconditioner that is factorized by a sparse direct code.

Nick Gould uses this approach and our software in both his interior point code VE12, and in VE19, an active set code with a Schur complement update (Gould and Toint, 2001). Both Jorge Nocedal (Northwestern) and Steve Wright (Wisconsin) have used MA57 as a linear solver in their optimization packages, KNITRO (Byrd, Hribar and Nocedal, 1999) and OOQP (Gertz and Wright, 2001), respectively.

This application of a sparse direct solver is a particularly rich one for testing the linear code. Obviously any matrix of the form (5.18), with $\mathbf{B}$ symmetric, is symmetric and indefinite. In addition, the matrix $\mathbf{B}$ is often not positive-definite and may even be singular, so that really quite evil linear systems can result. A solution to the linear system is sometimes only possible because the right-hand side lies in a subspace for which the projection of the augmented matrix is nonsingular. Additionally, the matrices $\mathbf{A}$ and $\mathbf{B}$ can have a very wide range of sparsity patterns depending on the original source of the optimization problem. Such matrices, and the sophistication of the optimization codes which call the linear solver, result in a varied and severe test for all three phases of the direct solution. Although we will concentrate on the normally most costly factorization phase in the remainder of this section, we comment first on the other phases.

For the analysis phase, we are using the approximate minimum degree algorithm of Amestoy et al. (1996). In common with most minimum degree approaches, a straightforward implementation of this algorithm can be rather slow when the matrix has one or more very dense rows. Although these dense rows will not be chosen for pivoting until near the end of the symbolic factorization, they are connected to most other rows and will be repeatedly accessed during the symbolic factorization. In some of the test cases, there were very dense rows. For example, in a matrix of order 30007 supplied by Gould, the analysis phase for a version of the algorithm that did not take special account of dense rows, required 214 seconds on a DEC Alpha DS20 workstation, contrasting with 2.47 and 0.26 seconds for factorization and solve phases, respectively. When we developed a modified version of our approximate minimum degree code to deal more explicitly with dense rows, the analysis time dropped to 1.4 seconds and the quality of the ordering was essentially unaffected.

Often, in the solution of sparse linear systems, less attention is placed on the solve phase because it is usually more than an order of magnitude faster than the factorization. However, in the context of these optimization codes, there are many more calls to the solve routines than to the factorization (for example, in a typical run of KNITRO, there were 34,174 calls to the solve routines but only 125 matrix factorizations) primarily because the solution is used as a preconditioner for an iterative method. Thus the overall time can be significantly affected by the solve times. Indeed, on early runs using the new MA57 code, we were dismayed to find that, although the factorization times were less than half that of the

earlier MA27 code, the overall times for many of the optimization runs were much longer when MA57 was used in place of MA27. This early problem was traced to a slower solve time for MA57 which, on further investigation, was found to be caused by the overhead of the loop on the number of right-hand sides in the case when there was only one right-hand side. We have now written separate code for the case of one right-hand side that is in general faster than the MA27 solve phase and, in these cases, the optimization code with MA57 included then outperformed the version with MA27.

	Analyse	Factorize	Solve	
			1 RHS	10 RHS
MA27	0.99	12.78	0.148	1.48
MA57	0.43	5.72	0.070	0.40

Table 5.2: Times (in seconds on a Sun ULTRA 5) for all phases of MA27 and MA57 on example STCQP1 from CUTE. Matrix of order 5036 with 38045 entries.

In Table 5.2, we show the results of a typical run of our two sparse symmetric multifrontal codes on a system of linear equations obtained from test problem STCQP1 from the CUTE collection (Bongartz, Conn, Gould and Toint, 1995). We see that MA57 is significantly superior to MA27 in all phases. This is true for all the linear systems that we have tested, and we have observed larger cases on which the MA57 factorize is ten times faster than that of MA27, largely because of the use of the Level 3 BLAS. Note also that the use of Level 3 BLAS when solving for a block of right-hand sides gives a superlinear speedup in terms of number of right-hand sides.

Identifier	# vars	# inequalities	MA27	MA57
AUG3DCQP	27543	8000	95.8	54.9
AUG3DQP	27543	8000	95.8	56.9
BLOWEYA	20002	10002	18.3	35.9
DEGENQP	50	125025	49.1	19.2
CVXQP1	15000	7500	906.5	2.9
STCQP1	8193	4095	107.8	161.3
STCQP2	8193	4095	108.1	48.3

Table 5.3: Total times to solution of CUTE problems in seconds on a DEC alpha workstation.

When we look, in Table 5.3, at times for complete runs of a prototype primal-dual interior point method of Gould (HSL_VE12), using both the codes, we would naturally expect the performance to mirror that in Table 5.2. In nearly all cases, runs of the optimization package with the new code are faster, sometimes significantly so. There

are, however, a few cases that cause us concern (for example BLOWEYA and STCQP1 in Table 5.3). Of course, the times for the solution of the nonlinear problem will be affected strongly by convergence rates and the convergence path taken by the algorithm and even a small change in the solution provided by the linear solver may influence this. One important issue is that the systems (5.17) become very ill conditioned, particularly near the solution of the optimization problem which is exactly when an accurate solution to the linear problem is required. Additionally, if we look at the results shown in Table 5.4, we see it is important from an efficiency point of view to keep the threshold very low but then there is significant risk of obtaining a poor solution, as happens when the threshold is reduced to 10^{-12} in this example. Because of this risk and because the solution must satisfy the $\mathbf{Ax} = \mathbf{0}$ constraints to high accuracy, the optimization codes have built in a facility for iterative refinement. Because of differences in pivoting strategies of `MA27` and `MA57`, on some examples it would appear that `MA57` required far more iterative refinements causing the overall time for the optimization to increase.

We investigated the apparent paradox between the relative times for problem STCQP1 in Tables 5.2 and 5.3 further. The two runs of the optimization code required about the same number of function evaluations and conjugate gradient iterations but the run with `MA57` required far more iterative refinements. On further examination, we found that, for the same linear equations with the same threshold value of 10^{-10}, `MA57` returned a scaled residual of 10^{-6} while the scaled residual using `MA27` was 10^{-10}. The `MA57` value triggered the iterative refinement option. We then studied both solvers more closely to see why the residuals were so different and found that the `MA57` code had inherited a pivoting strategy from `MA47` whereby, if the candidate 2×2 pivot fails the threshold test

$$\left| \begin{pmatrix} a_{kk} & a_{k\ k+1} \\ a_{k+1\ k} & a_{k+1\ k+1} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max |a_{kj}| \\ \max |a_{k+1\ j}| \end{pmatrix} \leq \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}, \tag{5.19}$$

the (2,2) entry is immediately tested as a possible 1×1 pivot. If it satisfies the test, then the modified (1,1) entry is then tested and if it passes the test, the 2×2 pivot is accepted. This was a reasonable strategy for `MA47` since there is a high premium on being able to choose pivots of the form $\begin{pmatrix} 0 & \times \\ \times & \times \end{pmatrix}$. However, the resulting 2×2 pivot is potentially less stable than the normal test would allow. In fact, although the pivot has been accepted, the bound on the growth for the 2×2 pivot is now $1/u^2$ rather than $1/u$. This was certainly not intended for `MA57` and when this was removed, the resulting scaled residual was the same as for `MA27` and the optimization code ran faster with `MA57` than with `MA27`. Now, although this was all caused by effectively a bug in an early version of `MA57`, we have discussed this in some detail to illustrate the sensitivity of the optimization code to seemingly small changes to the linear equation pivoting strategy.

Threshold	Factorization		
u	Reals	Integers	Time
Analysis	250005	66252	
10^{-8}	6117748	274287	444
10^{-10}	512262	38225	3.84
10^{-12}	494956	37504	3.72

Table 5.4: Effect of changing threshold. Times in seconds on a DEC alpha workstation.

Of course, some of the sensitivity to the threshold parameter that we see most dramatically in Table 5.4, might be significantly affected by scaling the matrix prior to numerical pivoting. When we used the HSL routine `MC30` to scale the matrix of this table, we were able to factorize the matrix with a threshold of 10^{-8} in under 3 seconds with the number of reals and integers in the factors 464934 and 36259, respectively. Although scaling works extremely well in this case, Gould (private communication) suggests there are other problems on which the reverse is true and we await test matrices from him to study this phenomenon further.

Some of these preliminary results were discussed at a meeting in Morocco (Duff, 2002).

References

P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Analysis and Applications*, **17**(4), 886–905, 1996.

I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Trans. Math. Softw.*, **21**(1), 123–160, 1995.

R.H. Byrd, M.E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM J. Optimization*, **9**(4), 877–900, 1999.

I. S. Duff. Direct methods for the solution of large sparse systems. *in* 'Proceedings of the Conference Algèbre Linéaire et Arithmétique Calcul Numérique et Parallèle, held in Rabat, Morocco, 28-31 May, 2001', 2002.

E. M. Gertz and S. J. Wright. *OOQP User Guide: Object-Oriented Software for Quadratic Programming.* Argonne National Laboratory, October 2001.

N. I. M. Gould and Ph. L. Toint. Numerical methods for large-scale non-convex quadratic programming. Technical Report RAL-TR-2001-017, Rutherford Appleton Laboratory, Oxfordshire, 2001.

5.4 Filter Methods (N. I. M. Gould and Ph. L. Toint)

Although we have recently concentrated on QP methods, we have not neglected our ultimate aim, that of embedding such methods within the sequential quadratic programming (SQP) framework for solving large-scale nonlinear programming problems. Our research in this area has been restricted to investigating the convergence properties of the newly emerging class of SQP Filter methods.

Most current methods for constrained optimization cope with the conflicting requirements of feasibility and optimality by combining the objective function and a measure of the constraint infeasibility within a single "'merit" function. Filter methods were first proposed by Fletcher and Leyffer (2002) as a means of assessing the suitability of steps computed by SQP methods. Their primary aim is to avoid the use of merit functions, since it is far from obvious how best to combine the objective and constraints. Filter methods instead treat the objective and constraints as independent objects, and essentially assess the suitability of an SQP step by rejecting it only if neither the objective nor constraint violation improves following the step. Although a general purpose SQP Filter method is necessarily far more complicated than this simple idea, there is strong evidence that the approach is worthwhile, and offers more flexibility than other merit-function based approaches.

Unfortunately, convergence of the basic SQP Filter method depends upon being able to solve the step-finding QP subproblem. Since in general this is a non-convex optimization problem, it is unreasonable in practice to hope to be able to do so in every case. Thus our research was based on alternatives that do not require the exact solution of the QP subproblem.

One way to do this is to relax the requirements on the step, but to insist that the step is constructed as the sum of two components, one of which aims towards (linearized) feasibility, and the other towards objective-function decrease. Both components have to be chosen to ensure "Cauchy-like" decrease conditions so familiar in trust-region methods, but fortunately there are good methods to guarantee this. The global convergence of just such a scheme was established in Fletcher, Gould, Leyffer, Toint and Waechter (2002).

An alternative in which the SQP step is attempted first, but in which the Fletcher et al. (2002) method is used as a fall-back is also possible, and has been analysed by Gould and Toint (2001).

References

R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, **91**(2), 239–269, 2002.

R. Fletcher, N. I. M. Gould, S. Leyffer, Ph. L. Toint, and A. Waechter. Global convergence of a trust-region SQP-filter algorithm for nonlinear programming. *SIAM Journal on Optimization*, (to appear), 2002.

N. I. M. Gould and Ph. L. Toint. Global convergence of a hybrid trust-region SQP-filter algorithm for general nonlinear programming. Technical Report RAL-TR-2001-033, Rutherford Appleton Laboratory, Oxfordshire, 2001.

5.5 CUTEr, an Optimization Testing Environment (N. I. M. Gould, D. Orban and Ph. L. Toint)

The CUTE testing environment for optimization software and the associated test problem collection originated from the need to perform extensive and documented testing on the LANCELOT package (Conn, Gould and Toint, 1992). Because the large set of test problems and testing facilities produced in this context were useful in their own right, they were extended to provide easy interfaces with other commonly used optimization packages, gathered in a coherent multi-platform framework and made available, via anonymous ftp, to the research community. Bongartz, Conn, Gould and Toint (1995) provide an overview of the environment, and a full documentation of the tools and interfaces available at the time.

Since 1993, the CUTE environment and test problems have been widely used by the community of optimization software developers. However, this continued success inevitably led to a clearer awareness of the deficiencies of the original design, and also created a demand for new tools and new interfaces. The environment has evolved over time by the addition of new test problems and minor updates to a number of tools. Consequently, we believe that it is now time for its next major evolution: CUTEr, in which the ideas behind CUTE are revisited and revised. This new release is characterized by

- a set of new tools, including a unified facility to report the performance of the various optimization packages being tested,
- a set of new interfaces to additional optimization packages,
- extra, larger test examples, and
- some Fortran 90/95 support.

The SIF optimization test-problem decoder, which used to be a constituent part of the CUTE environment, has been isolated into a separate package named SifDec. Any software which could require the decoding of a SIF file may now rely on it, as a package in its own right. It is characterized by

- the definition and support of an extension to SIF (the Standard Input Format) allowing for easier input of quadratic programs and for casting the problem against a selection of parameters, such as the problem size, and

- the ability to generate input files suited to automatic differentiation tools, such as the HSL `AD01` and `AD02` packages (Pryce and Reid, 1998).

Both CUTEr and SifDec have the following features:

- Completely new organization of the various files that make up the environment, now allowing concurrent installations on a single machine and shared installations on a network, and

- a new simplified and automated installation procedure, but

- the restriction of the environment to UNIX systems.

Although CUTEr has not yet been released, its release is imminent, and much time over the previous two years has been spent in developing and testing the new environment. On release, CUTEr will be freely downloadable from its Web page,

`http://cuter.rl.ac.uk/cuter-www` .

The full scope of the package(s) is described in Gould, Orban and Toint (2002).

References

I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, **21**(1), 123–160, 1995.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for Large-scale Nonlinear Optimization (Release A)*. Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. Technical Report in preparation, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2002.

J. D. Pryce and J. K. Reid. `AD01`, a Fortran 90 code for automatic differentiation. Technical Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Oxfordshire, 1998.

5.6 GALAHAD (N. I. M. Gould, D. Orban and Ph. L. Toint)

As we reported in the previous progress report, our well known, large-scale optimization package LANCELOT (Conn, Gould and Toint, 1992) is showing its age. Our intention is to produce a successor (or more likely two), within the next couple of years, and we are currently preparing the ground for this.

Rather than providing a single nonlinear programming package, we have decided instead to produce a library, GALAHAD, of optimization-related packages. We are currently preparing for the first release, and the key components will be

- `QPA`, see Section 5.1.2,
- `QPB`, see Section 5.1.1,
- `LSQP`, an interior-point method for minimizing a linear or separable convex quadratic function over a polyhedral region,
- `PRESOLVE`, see Section 5.1.4,
- `LANCELOT B`, an updated version of the old dinosaur,
- `GLTR`, a method for minimizing a quadratic function within or on a (scaled) ball (the ℓ_2 trust-region subproblem) based on Gould, Lucidi, Roma and Toint (1999)), and
- a variety of sparse-matrix manipulation tools.

The enlivened `LANCELOT B` offers a number of improvements over its predecessor, but is still far from the state-of-the-art. New features include

- the automatic allocation of workspace,
- a non-monotone descent strategy to be used by default,
- the optional use of Moré and Toraldo (1991)-type projections during the subproblem solution phase,
- an interface to Lin and Moré's (1999) public domain incomplete Cholesky factorization package ICFS for use as a preconditioner, and
- the optional use of structured trust regions to better model structured problems (see Conn, Gould, Sartenaer and Toint, 1996).

The main reason for extending `LANCELOT`'s life is as a prototype for what may be achieved using Fortran 90 in preparation for future GALAHAD SQP solvers, since the problem data structure is unlikely to change.

Much as for CUTEr, the library will be fully documented, and capable of supporting multi-platform, simultaneous use (within a Unix-like environment). The library is coded entirely in Fortran 90, and is threadsafe.

On release, GALAHAD will be freely downloadable from its Web page,

`http://galahad.rl.ac.uk/galahad-www` .

The full scope of the package(s) is described in Gould, Orban and Toint (2002).

References

A. R. Conn, N. I. M. Gould, A. Sartenaer, and Ph. L. Toint. Convergence properties of minimization algorithms for convex constraints using a structured trust region. *SIAM Journal on Optimization*, **6**(4), 1059–1086, 1996.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for Large-scale Nonlinear Optimization (Release A).* Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

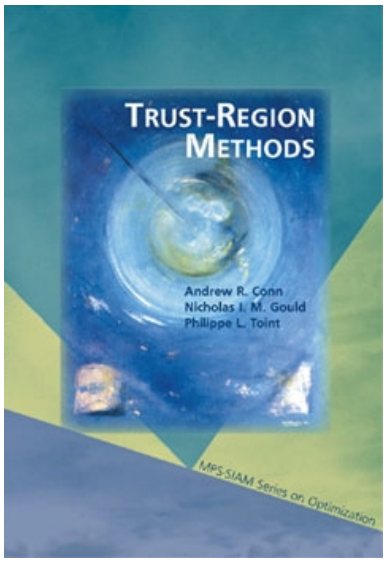
N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, **9**(2), 504–525, 1999.

N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. Technical Report in preparation, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2002.

C. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, **9**(4), 1100–1127, 1999.

J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, **1**(1), 93–113, 1991.

5.7 Trust-region methods (A. R. Conn, N. I. M. Gould and Ph. L. Toint)



Trust-region methods are one of the most popular techniques for solving nonlinear optimization problems. Our book on the subject was published by SIAM in time for the Mathematical Programming Society's triennial symposium in Atlanta, Georgia in August, 2000.

References

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-region methods*. 966+xvii pages, SIAM, Philadelphia, USA, 2000.

6 Automatic Differentiation

6.1 Automatic differentiation for core calculations (S. A. Forth, J. D. Pryce, J. K. Reid, and M. Tadjouddine)

John Reid collaborated with Shaun Forth, John Pryce and Mohamed Tadjouddine of RMCS Shrivenham over applying source translation techniques to calculate Jacobians for key calculations that are executed many thousands of times in a typical computer run.

Automatic differentiation is a means of computing the derivatives of a function specified by a computer code for a given set of values of the independent variables. If the independent variables are $x_1, x_2, \ldots, x_n$, we may write the calculation as the sequence of operations

$$x_i = f_i(x_1, x_2, \ldots, x_{l(i)}),\ l(i) = min(i-1, m-k),\ i = n+1, n+2, \ldots, m, \tag{6.1}$$

where the output variables are $f_i = x_{m-k+i}$, $i = 1, 2, \ldots, k$.

By differentiating equation (6.1), we find that the derivatives are related thus

$$\frac{\partial x_i}{\partial x_j} = \sum_{k=1}^{l(i)} \frac{\partial f_i}{\partial x_k}\frac{\partial x_k}{\partial x_j},\ \ j = 1, 2, \ldots, n. \tag{6.2}$$

If we suppose that the intermediate variables are $y_i = x_{n+i}$, $i = 1, 2, \ldots, m-k-n$, we can rewrite equation (6.2) as

$$\begin{bmatrix} -\mathbf{I} & & \\ \mathbf{F} & \mathbf{L} & \\ \mathbf{G} & \mathbf{H} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \nabla\mathbf{x} \\ \nabla\mathbf{y} \\ \nabla\mathbf{f} \end{bmatrix} = \begin{bmatrix} -\mathbf{I} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{6.3}$$

where $\mathbf{L}$ is lower triangular with diagonal entries -1 and the elements of $\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$, and the lower triangle of $\mathbf{L}$ are the 'local derivatives' $\frac{\partial f_i}{\partial x_k}$ of equation (6.2).

If Gaussian elimination is applied to equation (6.3), pivoting only on the diagonal entries of $\mathbf{L}$, we find the equation

$$\begin{bmatrix} -\mathbf{I} & & \\ \mathbf{F}' & \mathbf{L}' & \\ \mathbf{J} & \mathbf{0} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \nabla\mathbf{x} \\ \nabla\mathbf{y} \\ \nabla\mathbf{f} \end{bmatrix} = \begin{bmatrix} -\mathbf{I} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{6.4}$$

and we see that the Jacobian $\mathbf{J}$ has been calculated. Note that each pivotal operation preserves the form, that is, the matrix remains lower triangular with all diagonal entries of value -1. All the pivots are -1, regardless of the pivot order.

The forward method corresponds to the forward order $n+i$, $i = 1, 2, \ldots, m-k-n$ and the backward method corresponds to the backward order $n+i$, $i = m-k-n, m-$

$k-n-1,\ldots,1$. Other pivotal orders, sometimes called 'cross-country' ordering, may be better than either.

A typical application is the Roe flux routine. This computes the numerical fluxes of mass, energy and momentum across a cell face in a finite-volume compressible flow calculation. Roe's flux takes as input 10 variables describing the flow either side of a cell face and returns as output the 5 variables for the numerical flux (200 lines of code). On the platforms shown in Table 6.1, we have compared our code (VE), with several elimination strategies, with finite differences (FD) and two of the established packages, Adifor and TAMC. We obtained the results shown in Table 6.2. We see that on the SGI and ALP platforms, our approach is about twice as fast as both conventional AD tools and finite differences. On the SUN, NAG and FUJ platforms the times are less straightforward, but the fastest is always one of our variants, so we are encouraged. Tests on other problems is in progress.

Platform	Processor				Compiler	
	Processor	CPU	L1-Cache	L2-Cache	Compiler	Options
SGI	R12000	300MHz	64KB	8MB	f90 MIPSPro 7.3	-Ofast
ALP	EV6	667MHz	128KB	8MB	Compaq f95 5.4	-O5
SUN	Ultra10	440MHz	32KB	2MB	Workshop f90 6.0	-fast
NAG	Ultra10	440MHz	32KB	2MB	Nagware f95 4.0a	-O3 -native
FUJ	Ultra1	143MHz	32KB	0.5	Fujitsu f90 5.0	-Mfrt, Am

Table 6.1: Platforms (Processors and Compilers)

Method	SGI	ALP	SUN	NAG	FUJ
FD (1-sided)	12.1	12.6	12.6	13.3	11.9
VE Forward	7.5	5.5	13.2	12.2	10.2
VE Reverse	6.5	4.6	8.8	8.8	6.9
VE Mark	6.4	5.1	9.8	10.4	6.7
VE Mark	6.6	5.0	14.1	10.2	6.6
VE VLR	6.0	4.5	8.9	9.1	6.1
VE VLR RB	6.2	4.6	9.0	9.6	6.0
Adifor	15.9	9.8	31.4	50.5	14.4
TAMC-ftl	14.4	10.2	11.9	56.7	14.9
TAMC-ad	12.2	8.5	13.2	42.0	10.0

Table 6.2: Ratios of Jacobian to function CPU timings on various platforms.

6.2 Threadsafe automatic differentiation in Fortran 95 (J. K. Reid)

The HSL automatic differentiation package AD01 is seriously thread-unsafe because it relies extensively on arrays in modules and there can only be one copy of each such array. The decision that HSL 2002 should be threadsafe, see Section 9.3, forced us to make significant changes. We decided to take the opportunity to make further improvements by using the additional language features of Fortran 95. We know of no Fortran 90 compiler that is being actively maintained and marketed that does not now conform to Fortran 95. The new package is called AD02.

The big advantage of Fortran 95 for this application is that derived types can be specified to have initial values for components. Wherever an object of such a type is created, these components are given their initial values. This means that all active variables (variables whose value depends on the independent variables), declared by the user of AD02 to be of type AD02_REAL, have an initial value that can be tested. This is very important since an assignment to an active variable should free the memory associated with the previous value to avoid serious memory leakage. It is therefore essential to know whether there is a previous value. In AD01, we relied on the user explicitly setting a special undefined value for each active variable; this is tedious for the user and it is very easy for some variables to be overlooked.

AD01 holds information about each calculation as module data and there are facilities for saving and restoring this data to allow for subsidiary calculations. In AD02, we have extended this idea to have information about every calculation held in a structure of type AD02_DATA, which means that any number of calculations may be active at any one time and the need for facilities for saving and restoring disappears. When a calculation is commenced, we need to specify the structure within which the calculation is to be held. We allow such a structure to be reused because often an old calculation will no longer be needed once the new one has started. We therefore introduced separate subroutines for initializing and finalizing such a structure.

Every object of type AD02_REAL has a pointer component that is associated with an AD02_DATA structure when it is active. Unfortunately, there is no easy way to make all the pointers associated with a calculation become disassociated when the AD02_DATA structure is reused or finalized. We have therefore decided to hold an integer component CASE to label the calculation in both the AD02_REAL variable and the AD02_DATA structure. Also, the finalize subroutine does not actually deallocate the structure; instead, it deallocates its array components and sets a special value in the component CASE. Any active variable that is associated with this structure is now regarded as undefined and we can test for this status.

Automatic initialization of components and the introduction of `CASE` components allows us to test any active variable that is involved in an operation for being undefined. It also allow us to check that two active variables involved in a binary operation belong to the same calculation.

There is a difficulty with respect to the assignment of a constant or an inactive variable to an active variable. The only possible place for a pointer to the calculation is within the left-hand side variable, and we use this provided it is associated with an `AD02_DATA` structure that has been initialized and has not been finalized. Whether the left-hand side is defined is not relevant and it is quite likely to be undefined solely as a result of belonging to an earlier calculation for a structure that has been reused. We therefore introduce the concept of a 'data-undefined' value and provide a subroutine to set this value in an `AD02_REAL` variable.

Our final difficulty is associated with where to place error messages when things go wrong. We have decided to hold an integer array for error and warning flags in each `AD02_DATA` structure for calculations associated with it and directly in the module for computations not associated with a calculation.

As a result of all these changes, `AD02` is far safer, more convenient, and more powerful than `AD01`. A disadvantage, however, is that our tests for the validity of each operation are necessarily more complicated and will slow the execution. We feel that these tests are essential for the development phase of a program since it is so easy to make a mistake. We plan, however, to consider providing a version for production use that omits these tests.

7 Miscellaneous Activities

7.1 CERFACS (I. S. Duff)

Iain has continued to lead a project at CERFACS on Parallel Algorithms and several of the contributions to this report reflect interactions with that team.

The main areas of research in the Parallel Algorithms Project are the development and tuning of kernels for numerical linear algebra, the solution of sparse systems using direct methods or iterative methods or a combination of the two, heterogeneous computing including the use of PVM and MPI, large eigensystem calculations, optimization, and the reliability of computations. Other activities of the Project include advanced training by both courses and research. A major initiative of the Team is in the study of inner-outer iterations where they have shown a radically different performance for Newton-type iterations from cases where a Krylov method is user in the outer iteration. In the latter case, as is well known, increasing accuracy is normally required in the inner iterations as the outer converges. However, as is not so well known, the reverse has been shown to be true in the latter case. Four short international meetings were hosted by the Parallel Algorithms Project during the period of this report, and were attended by members of the Group.

During the reporting period, five students completed their PhDs at CERFACS. Iain was a jury member for the thesis defence of two of them, Dominique Orban and Elisabeth Traviesas. He was also on the jury for two habilitation theses by Valérie Frayssé and Luc Giraud both seniors at CERFACS.

Nick visited CERFACS to collaborate with Annick and a PhD student, Dominique Orban, and to work on his book with Philippe Toint, visiting CERFACS from Belgium.

The home page for CERFACS is `http://www.cerfacs.fr` and current information on the Parallel Algorithms Project can be found on page `http://www.cerfacs.fr/algor/`. Full details on the activities of the Parallel Algorithms Team for the last two years can be found in the reports (Project 2001*a*, Project 2001*b*).

References

The Parallel Algorithms Project. Scientific Activity Report for 2000. Technical Report TR/PA/01/23, CERFACS, Toulouse, France, 2001*a*.

The Parallel Algorithms Project. Scientific Activity Report for 2001. Technical Report TR/PA/01/105, CERFACS, Toulouse, France, 2001*b*.

7.2 ERCIM (M. Arioli and I. S. Duff)

As stated in each ERCIM News, The European Research Consortium for Informatics and Mathematics (ERCIM) is an organisation dedicated to the advancement of European research and development, in information technology and applied mathematics. Its national member institutions aim to foster collaborative work within the European research community and to increase co-operation with European industry.

ERCIM started in 1989 with the three Laboratories CWI (Amsterdam), GMD (Germany), and INRIA (France) and were joined by RAL in the following year. There are now members from 13 countries in the EU and Eastern Europe.

In the early days there were quite active groups and ERCIM meetings in mathematics, but we have been concerned in recent years that the M of ERCIM was becoming neglected. Thus there were two initiatives launched in 2001 to rectify this.

One was an ERCIM Working Group on Numerical Linear Algebra and Statistics coordinated by Erricos Kontoghiorghes (Université de Neuchatel, Switzerland) and Bernard Philippe (IRISA, France) to which both Mario and Iain belong and which has had two meetings in 2001, one of which was attended by Mario.

The other ERCIM Group is coordinated by Mario and has a wider mathematical remit involving several ERCIM institutions interested in Applications of Numerical Mathematics in Science. Although we anticipate that many application areas will benefit from the results and activities of the working group, it will focus on the following four areas:

- Numerical Linear Algebra.
- Numerical Solution of Differential Equations.
- Continuous Optimization and Optimal Control.
- Large Scale Scientific Computing.

It is planned to hold the inaugural meeting of this Group in the Czech Republic in August. The Working Group looks forward to broadening the scope of it main research topics into additional numerical areas. The Group strongly believes that the best way to build stronger links between the ERCIM laboratories is to encourage young scientists to act as intermediaries, and we plan to promote the ERCIM fellowship programme among young scientists. The potential recruitment of young scientists justifies the involvement of several Universities in our initiative.

In Table 7.2, we summarize our current information on the interest of each organization in each topic.

Finally, the Working Group will promote through its members all possible initiatives within the European Programmes for Research. We will encourage grant applications

and involvement in the research, technological development and demonstration (RTD) framework programmes of the EU.

The web site for the Working Group is
`http://www.numerical.rl.ac.uk/ercim/WGanms.html`.

Organization	Numerical Linear Algebra	Numerical Solution of Differential Equations	Continuous Optimization	Large Scale Scientific Computing
CLRC	X	X	X	X
CNR	X	X	X	X
CWI	X	X		X
DTU	X			X
FO.R.T.H./IACM	X	X		X
FhG		X		X
ICS CAS	X		X	X
INRIA-IRISA	X	X	X	X
SARIT	X		X	X
SINTEF	X	X		X
Trinity College		X		X
Univ. Patras	X		X	
Univ. Utrecht	X			X
Univ. of Wales-Cardiff	X			X

Table 7.1: Institutions involved in the ERCIM Working Group

8 Computing and mathematical software

8.1 The computing environment within the Group

Our policy of upgrading the Group's workstations has continued over the past two years. Although the Group's "high-performance" machine remains a Compaq Alpha DS20 dual EV6-processor server, we have taken the opportunity to upgrade one of our SUN workstations, and to evaluate a couple of Linux-based Dell PCs. We are delighted with the Dell machines, and our policy is likely to move from Sun machines to PCs over the next year or so. In addition, we now have four Dell Latitude laptop computers of varying configurations, and have currently replaced the eldest of these with a Dell Precision M40.

The responsibility for SUN software support continues to be delegated to other parts of the Laboratory, although group members have still found it more convenient to get their hands dirty for simple tasks—in particular, we maintain our own Linux systems on the laptops and PCs. The Group continues to support a series of Web pages describing its activities, and we have recently replaced our IBM RS/6000 by one of the PCs as a Web server. In addition, we now allow restricted (and recorded) access to HSL codes via the same PC.

We still benefit from other public RAL machines, in particular the Compaq multiprocessor systems. The Group's files continue to reside on a central UNIX data store, which is backed up daily by the Information Technology Department. We continue to have access to a number of Fortran 95 compilers, some on our own machines, and have recently introduced our first Fortran 95 codes into HSL. We have also made use of MPI, and associated parallel language support systems, on both our own machines and on those provided by ITD.

In combination with our Grant application, we obtained computing time on national facilities: namely on Columbus at RAL, and the CSAR machines (SGI Origin 2000 and CRAY T3E) at Manchester.

9 HSL (Harwell Subroutine Library)

9.1 Collaboration with AEA Technology

The year 2000 was a dramatic one in respect of our collaboration with AEA Technology. It began with the sale of the part of AEA Technology for which Nick Brealey worked and the passing of responsibility for HSL to a management trainee Katie North with no experience of mathematical software. After only a few months, an internal sale to Hyprotech took place, and we began to collaborate with Lawrence Daniels, Iain Strachan, and Lesley Vernon.

The collaboration with Hyprotech continued thereafter and has been happy and successful. They bought the AEAT rights in HSL because they value its software and want to use it in their products. They have agreed to pay a fixed annual fee for rights to incorporate the software in their packages and this is enough to support John Reid's consultancy. Lesley Vernon embraced the marketing tasks associated with HSL with enthusiasm and brought her experience with other Hyprotech software to bear. We were very sorry that family reasons led her to resign, but the reigns have been ably taken up by Pascale Hicklin.

9.2 HSL 2000 and HSL Archive

We intended that a new release of HSL would take place early in 2000, but the uncertainties of our collaboration with AEA Technology (see previous subsection) caused it to be delayed until October.

Many of the older HSL packages have gradually been superseded by newer versions, with increases in functionality, improved interfaces, or speed of execution. We decided that the time had come to separate HSL into two parts, the main library, HSL 2000; and an archive, HSL Archive. The HSL Archive comprises older packages that were part of previous releases of HSL, many of which have been superseded by more modern codes. The HSL Archive packages are not completely frozen since we aim to correct any errors that come to light, but it is not our intention to develop any of its packages further.

The split allows us to focus our attention in the packages of HSL 2000. While this is a commercial product, it is also available without charge to UK academics for teaching and academic research purposes. This innovation is a direct result of much of our core funding being provided by a grant from the Engineering and Physical Science Research Council (GR/R46441).

The HSL Archive may be licenced without charge to anyone (in the UK and elsewhere), so long as the packages are not then supplied to a third party as part of another software package. Access to the HSL Archive is by way of the HSL web page
`http://www.cse.clrc.ac.uk/Activity/HSL`

Reorganizing HSL into two parts and arranging separate catalogues for the two parts was a significant undertaking and we would like to express our thanks to Mike Hopper for his assistance as a consultant for these tasks. He was also invaluable in altering the electronic representations of the changed AEAT logos to a compact form, suitable for inclusion in each specification document.

9.3 HSL 2002

We prepared a new release of HSL during 2001 and completed this at the very end of the year. The significant new packages in this release were AD02, EA16, HSL_MA48, MA65, HSL_MC65, HSL_MC66, ME57, HSL_VE19, HSL_KB12, and HSL_MA69. More details are given in the next section.

A further significant change is that we decided that all packages in HSL 2002 should be threadsafe. This is clearly desirable anyway as parallel processing becomes more pervasive, but our motivation was also to prepare for a DLL (Dynamic Linked Library) version in the future. It was also seen as very desirable by our Hyprotech colleagues for their software. All reliance on SAVE, COMMON, and module variables has been removed. Unfortunately, in many cases this could not be done without altering the argument list. In such a case, the new routine is not compatible with the old one, so we felt obliged to change its name. Once again, we were indebted to Mike Hopper, for assisting us in this task as a consultant. This was a major program of work and we would like to thank Hyprotech for funding this part of Mike's work.

HSL 2002 is also available without charge to members of the UK academic community.

9.4 New HSL packages

The following new packages were added to the Library during the reporting period.

9.4.1 AD02 Automatic differentiation (J. K. Reid)

This Fortran 95 package provides automatic differentiation facilities for variables specified by Fortran code. Each independent variable and each variable whose value depends on the value of any independent variable must be declared to be of type AD02_REAL instead of default REAL (double precision REAL in the DOUBLE version). Note that Fortran variables of type default REAL (double precision REAL in the DOUBLE version) and default INTEGER may enter the computation provided their values do not vary with the values of the independent variables. Both the backward and the forward method are available.

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

Unlike HSL_AD01, this code allows several independent calculations to be performed at the same time. Each calculation is stored in a structure to which each of its variables has access through a pointer.

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

return. Alternatively, an immediate stop may be requested.

9.4.2 EA16 Eigenvalues and eigenvectors of real symmetric matrices (K. Meerbergen and J. A. Scott)

This package uses an implicitly restarted block Lanczos method or rational Lanczos method to compute selected eigenpairs of $\mathbf{Ax} = \lambda\mathbf{x}$ where $\mathbf{A}$ is a large real symmetric matrix or $\mathbf{Ax} = \lambda\mathbf{Mx}$, with $\mathbf{A}$ and $\mathbf{M}$ large real symmetric matrices and either $\mathbf{A}$ or $\mathbf{M}$ positive-semidefinite.

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are Ritz vectors. If we denote by OP the operator that is applied to the vectors in the Lanczos process, `EA16` may be used to compute Ritz pairs for one of the following problems:

1. Standard eigenvalue problem : $\mathbf{Ax} = \lambda\mathbf{x}$, $\mathbf{A}$ symmetric. This is solved using one of the following modes:

   1a. Regular mode. Here $OP = \mathbf{A}$.

   1b. Shift-invert mode. Here $OP = (\mathbf{A} - \sigma\mathbf{I})^{-1}$ with σ not an eigenvalue of $\mathbf{A}$.

   The computed Ritz vectors are orthogonal with respect to the standard inner product $< \mathbf{x}, \mathbf{y} >= \mathbf{x^T y}$.

2. Generalised eigenvalue problem : $\mathbf{Ax} = \lambda\mathbf{Mx}$, $\mathbf{A}$ symmetric, $\mathbf{M}$ symmetric positive-semidefinite. This is solved using one of the following modes:

   2a. Regular inverse mode. Here $OP = \mathbf{M}^{-1}\mathbf{A}$ with $\mathbf{M}$ nonsingular.

   2b. Shift-invert mode. Here $OP = (\mathbf{A} - \sigma\mathbf{M})^{-1}\mathbf{M}$ with σ not an eigenvalue of $\mathbf{Ax} = \lambda\mathbf{Mx}$.

   The computed Ritz vectors are orthogonal with respect to the $\mathbf{M}$ inner product $< \mathbf{x}, \mathbf{y} >= \mathbf{x^T M y}$.

3. Buckling problem : $\mathbf{Ax} = \lambda\mathbf{Mx}$, $\mathbf{A}$ symmetric positive-semidefinite, $\mathbf{M}$ symmetric. This is solved using the following mode:

   3a. Buckling mode. Here $OP = (\mathbf{A} - \sigma\mathbf{M})^{-1}\mathbf{A}$ with σ not equal to zero or to an eigenvalue of $\mathbf{Ax} = \lambda\mathbf{Mx}$.

   The computed Ritz vectors are orthogonal with respect to the $\mathbf{A}$ inner product $< \mathbf{x}, \mathbf{y} >= \mathbf{x^T A y}$.

In the shift-invert and buckling modes, σ is called the pole.

The method is described in detail by Meerbergen and Scott (2000), *The design of a block rational Lanczos code with partial reorthogonalization and implicit restarting,* Rutherford Technical Report RAL-TR-2000-011.

Note that `EA16` is designed for computing a limited number of eigenvalues and eigenvectors. `EA16` **cannot** be used for computing all the eigenpairs of $\mathbf{Ax} = \lambda\mathbf{x}$ or $\mathbf{Ax} = \lambda\mathbf{Mx}$. The matrices $\mathbf{A}$ and $\mathbf{M}$ need not be available in an explicit form.

9.4.3 HSL_KB12 Sorting reals using the Heapsort method (N. I. M. Gould)

`HSL_KB12` is a suite of Fortran 90 procedures for successively arranging a set of real numbers, $\{a_1, a_2, ..., a_n\}$, in order of increasing size using the Heapsort method of J. W. J. Williams. At the k-th stage of the method the k-th smallest member of the set is found. The method is particularly appropriate if it is not known in advance how many smallest members of the set will be required as the Heapsort method is able to calculate the $k+1$st smallest member of the set efficiently once it has determined the first k smallest members. The method is guaranteed to sort all n numbers in $O(n \log n)$ operations. If a complete sort is required, the Quicksort algorithm, `KB05`, may be preferred.

9.4.4 HSL_MA48 Solve a sparse unsymmetric system: driver for conventional direct method (I. S. Duff)

This solves a sparse unsymmetric system of linear equations. Given a sparse matrix $\mathbf{A} = \{a_{ij}\}_{m\times n}$ and an n-vector $\mathbf{b}$, this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ or the system $\mathbf{A^T x} = \mathbf{b}$. The matrix $\mathbf{A}$ can be rectangular. There is an option for iterative refinement and return of error estimates.

`HSL_MA48` is a Fortran 95 encapsulation of `MA48` and offers some additional facilities to the Fortran 77 version.

9.4.5 MA57 Solve a sparse symmetric system: multifrontal method (I. S. Duff)

This solves a sparse symmetric system of linear equations. Given a sparse symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n\times n}$ and an n-vector $\mathbf{b}$, this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ ($\mathbf{AX} = \mathbf{b}$). The matrix $\mathbf{A}$ need not be definite.

The multifrontal method is used. It is a direct method based on a sparse variant of Gaussian elimination and supersedes `MA27`.

9.4.6 HSL_MA57 Solve a sparse symmetric system: multifrontal method (I. S. Duff)

This solves a sparse symmetric system of linear equations. Given a sparse symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n\times n}$ and an n-vector $\mathbf{b}$, this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ ($\mathbf{AX} = \mathbf{b}$). The matrix $\mathbf{A}$ need not be definite.

The multifrontal method is used. It is a direct method based on a sparse variant of Gaussian elimination.

`HSL_MA57` is a Fortran 90 version of `MA57`.

9.4.7 MA65 Solve an unsymmetric banded system of linear equations (J. K. Reid)

This solves an unsymmetric banded system of linear equations. Given a unsymmetric band matrix $\mathbf{A} = \{a_{ij}\}_{n\times n}$ and an n-vector $\mathbf{b}$, this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$.

The matrix is factorized using Gaussian elimination with row interchanges. If the lower semibandwidth is kl and the upper semibandwidth is ku, that is, if $a_{ij} = 0$ for $i > j + kl$ or $j > i + ku$, fill-in is limited to kl additional diagonals of the upper triangle and the computation is performed within an array of size $n(2kl + ku + 1)$. At each pivotal step, operations are avoided on any row with a zero in the pivot column and on any column beyond the last with an entry in the pivot row.

9.4.8 MA67 Solve a sparse symmetric system with some zeros on the diagonal (J. K. Reid)

This solves a sparse symmetric indefinite system of linear equations. Given a sparse symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n\times n}$ and an n-vector $\mathbf{b}$, this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$.

The method used is a direct method using an $\mathbf{LDL^T}$ factorization, where $\mathbf{L}$ is unit lower triangular and $\mathbf{D}$ is block diagonal with blocks of order 1 and 2. Advantage is taken of the extra sparsity available with 2×2 pivots (blocks of $\mathbf{D}$) with one or both diagonal entries of value zero. The numerical values of the entries are taken in account during the first choice of pivots.

9.4.9 HSL_MA69 Unsymmetric system whose leading subsystem is easy to solve (N. I. M. Gould)

HSL_MA69 is a suite of Fortran 90 procedures for computing the the solution to an extended system of $n + m$ sparse real linear equations in $n + m$ unknowns,

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}.$$

in the case where the n by n matrix $\mathbf{A}$ is nonsingular and solutions to the systems

$$\mathbf{Ax} = \mathbf{b} \text{ and } \mathbf{A^T y} = \mathbf{c}$$

may be obtained from an external source, such as an existing factorization. The subroutine uses reverse communication to obtain the solution to such smaller systems. The method makes use of the Schur complement matrix

$$\mathbf{S} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}.$$

The Schur complement is stored and factorized as a dense matrix and the subroutine is thus appropriate only if there is sufficient storage for this matrix. Special advantage is taken of symmetry and definiteness in the coefficient matrices. Provision is made for introducing additional rows and columns to, and removing existing rows and columns from, the extended matrix.

9.4.10 MC54 Write a sparse matrix in Rutherford-Boeing format (I. S. Duff)

This writes a sparse matrix in Rutherford-Boeing format. The matrix can be input as an assembled matrix in either column-oriented or coordinate form, or as an unassembled finite-element matrix.

9.4.11 MC55 Write a supplementary file in Rutherford-Boeing format (I. S. Duff)

This writes a supplementary file in Rutherford-Boeing format. There are many types of supplementary file for which the user should read the documentation on the Rutherford-Boeing Sparse Matrix Collection (RAL Report RAL-TR-97-031). These include right-hand sides, solution vectors, orderings, eigenvalues etc.

9.4.12 MC56 Read a file or a supplementary file held in Rutherford-Boeing format (I. S. Duff)

This reads a file held in Rutherford-Boeing format. It may contain either a sparse matrix or supplementary data. There are many types of supplementary data for which the user

should read the documentation on the Rutherford-Boeing Sparse Matrix Collection (RAL Report RAL-TR-97-031). These include right-hand sides, solution vectors, orderings, eigenvalues etc.

9.4.13 MC59 Sort a sparse matrix to an ordering by columns (I.S. Duff and J.A. Scott)

This subroutine performs an in-place sort on a sparse matrix to an ordering by columns. There is an option for ordering the entries within each column by increasing row indices and an option for checking for indices that are out of range or duplicated.

This subroutine supersedes `MC20`, `MC39`, `MC49`, `ME20`, and `MF49`.

9.4.14 HSL_MC65 Construct and manipulate matrices in compressed sparse row format (Y. F. Hu)

For a general sparse matrix, the compressed sparse row format consists of three arrays, `PTR`, `COL` and `VAL`. `PTR` holds the starting positions of the rows in the `COL` and `VAL` arrays. The indices of the entries of row `I` are held in `COL(PTR(I):PTR(I+1)-1)` and the corresponding values are held in `VAL(PTR(I):PTR(I+1)-1)`. However, the user should not need to deal with these arrays individually; `HSL_MC65` encapsulates them in a sparse matrix object of the derived type `HSL_ZD01_TYPE`. `HSL_MC65` provides procedures that perform basic operations, such as sparse matrix summation and multiplication, on these sparse matrix objects. There is an option for omitting `VAL`, that is, for a pattern-only matrix.

9.4.15 HSL_MC66 Order an unsymmetric matrix into singly bordered blocked diagonal form (Y. F. Hu)

This orders an unsymmetric matrix $\mathbf{A}$ into singly bordered blocked diagonal (SBBD) form. Given the sparsity pattern of a matrix, this routine generates a row and column ordering that can be used to reorder the matrix into the following SBBD form

$$\begin{pmatrix} \mathbf{A}_{11} & & & & \mathbf{S}_1 \\ & \mathbf{A}_{22} & & & \mathbf{S}_2 \\ & & \ddots & & \vdots \\ & & & \mathbf{A}_{KK} & \mathbf{S}_K \end{pmatrix}.$$

Here K is the user-defined number of blocks. The aim is to minimize the size of the border in the above matrix, also known as the *net-cut*, and to achieve *load balance* by ensuring that the rectangular matrices $\mathbf{A}_{ii}$ are of similar sizes.

The `HSL_MC66` algorithm uses a multilevel approach combined with a Kernighan-Lin type refinement algorithm. Full details are discussed in Hu, Maguire and Blake, *Computers*

and Chemical Engng. **21** (2000), pp.1631-1647.

HSL_MC66 may be used to preorder an unsymmetric matrix for use with the sparse matrix solver HSL_MP43.

9.4.16 ME38 Sparse unsymmetric system: multifrontal method (T. A. Davis and I. S. Duff)

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

9.4.17 ME57 Sparse complex symmetric system: multifrontal method (I. S. Duff)

This solves a sparse complex symmetric system of linear equations. Given a sparse complex symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n\times n}$ and an n-vector $\mathbf{b}$ (or an $n \times s$ matrix $\mathbf{B}$), this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ ($\mathbf{AX} = \mathbf{B}$). Later versions of this package will allow $\mathbf{A}$ to be Hermitian or complex skew-symmetric.

9.4.18 HSL_MP43 Solve sparse unsymmetric system: multiple-front method, equation entry (J. A. Scott)

The module HSL_MP43 uses the multiple front method to solve sets of linear equations $\mathbf{Ax} = \mathbf{b}$ (or $\mathbf{AX} = \mathbf{B}$) where $\mathbf{A}$ has been preordered to singly-bordered block-diagonal form

$$\begin{pmatrix} \mathbf{A}_{11} & & & \mathbf{C}_1 \\ & \mathbf{A}_{22} & & \mathbf{C}_2 \\ & & \cdots & \cdots \\ & & \mathbf{A}_{NN} & \mathbf{C}_N \end{pmatrix}$$

The HSL routines MA42 and MA52 are used with MPI for message passing.

In the multiple front method, a partial frontal decomposition is performed on each of the submatrices $(\mathbf{A}_{ll}\ \mathbf{C}_l)$ separately. Thus, on each submatrix, $\mathbf{L}$ and $\mathbf{U}$ factors are computed. Once all possible eliminations have performed, for each submatrix there remains a frontal matrix $\mathbf{F}_l$. The variables that remain in the front are called interface variables and the interface matrix $\mathbf{F}$ is formed by summing the matrices $\mathbf{F}_l$. The interface matrix $\mathbf{F}$ is also factorized using the frontal method. Block back-substitution completes the solution.

The matrix data and/or the matrix factors are optionally held in direct-access files.

9.4.19 HSL_VE19 Quadratic programming problem: working-set method (N. I. M. Gould)

This package uses a working-set method to solve the ℓ_1 **quadratic programming problem**

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}) + \rho_g v_g(\mathbf{x}) + \rho_b v_b(\mathbf{x}) \tag{9.1}$$

involving the quadratic objective

$$q(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{g}^T\mathbf{x} + f$$

and the infeasibilities

$$v_g(\mathbf{x}) = \sum_{i=1}^{m} \max(c_i^l - \mathbf{a}_i^T\mathbf{x}, 0) + \sum_{i=1}^{m} \max(\mathbf{a}_i^T\mathbf{x} - c_i^u, 0)$$

and

$$v_b(\mathbf{x}) = \sum_{j=1}^{n} \max(x_j^l - x_j, 0) + \sum_{j=1}^{n} \max(x_j - x_j^u, 0),$$

where the n by n symmetric matrix $\mathbf{H}$, the vectors $\mathbf{g}$, $\mathbf{a}_i$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, $\mathbf{x}^u$ and the scalars f, ρ_g and ρ_b are given. Full advantage is taken of any zero coefficients in the matrix $\mathbf{H}$ or the vectors $\mathbf{a}_i$. Any of the constraint bounds c_i^l, c_i^u, x_j^l and x_j^u may be infinite.

The package may also be used to solve the **quadratic programming problem**

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}), \tag{9.2}$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T\mathbf{x} \leq c_i^u, \quad i = 1, \ldots, m, \tag{9.3}$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1, \ldots, n, \tag{9.4}$$

by automatically adjusting the parameters ρ_g and ρ_b in (9.1). Similarly, the package is capable of solving the **bound-constrained ℓ_1 quadratic programming problem**

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}) + \rho_g v_g(\mathbf{x}), \tag{9.5}$$

subject to the simple bound constraints (9.4), by automatically adjusting ρ_b in (9.1).

If the matrix $\mathbf{H}$ is positive-semidefinite, a global solution is found. However, if $\mathbf{H}$ is indefinite, the procedure may find a (weak second-order) critical point that is not the global solution to the given problem.

9.4.20 YM11 Generate a random sparse matrix (I. S. Duff)

These subroutines generate an m by n random sparse matrix with user-specified options such as structural nonsingularity and bandedness. The matrix is held in a packed form in a standard sparse matrix format, and there is an option to write it to a file in Rutherford Boeing format (Report RAL-TR-97-031).

`YM11` is a threadsafe version of `YM01` and includes entries for generating complex valued and integer valued matrices.

10 Seminars

20 January 2000	Dr Bruce Christianson (Hertfordshire) Cheap Newton steps for discrete time optimal control problems: automatic differentiation and Pantoja's algorithm.
18 February 2000	Dr Kurt Lust (Warwick) Continuation and bifurcation analysis of periodic solutions of partial differential equations.
4 May 2000	Professor Nick Higham (Manchester) Analysis of the Cholesky method with iterative refinement for solving the symmetric definite generalized eigenproblem.
15 June 2000	Dr Steven Benbow (Quintessa Ltd) Augmented linear systems - methods and observations.
23 November 2000	Dr Mario Arioli (Rutherford Appleton Laboratory) A stopping criterion for the conjugate gradient algorithm in a finite-element method framework.
8 February 2001	Dr Colin Campbell (Bristol) Support vector machines and related kernel methods.
1 March 2001	Professor Mark Stadtherr (Notre Dame, Indiana) Reliable process modelling and optimization using interval analysis.
17 May 2001	Dr Lawrence Daniels and Dr Iain Strachan (Hyprotech) On the robust solution of process simulation problems.
7 June 2001	Professor M.J.D. Powell (Cambridge) Some properties of thin plate spline interpolation.
22 November 2001	Dr Milan Mihajlovic (Manchester) A new preconditioning technique for the solution of the biharmonic problem.

11 Reports issued in 2000-2001

We give a full listing of Rutherford Technical Reports issued 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. All of our current technical reports are publicly accessible via the internet from

`http://www.numerical.rl.ac.uk/reports/reports.html`.

Rutherford Reports

RAL-TR-2000-001 Numerical Analysis Group Progress Report. January 1998 - December 1999. I. S. Duff (Editor).

RAL-TR-2000-009 Some sparse pattern selection strategies for robust Frobenius norm minimization preconditioners in electromagnetism. B. Carpentieri, I. S. Duff, and L. Giraud.

RAL-TR-2000-010 The Lanczos method with semi-inner product. K. Meerbergen.

RAL-TR-2000-011 The design of a block rational Lanczos code with partial reorthogonalization and implicit restarting. K. Meerbergen and J. Scott.

RAL-TR-2000-014 Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint.

RAL-TR-2000-030 Two-stage ordering for unsymmetric parallel row-by-row frontal solvers. J. A. Scott.

RAL-TR-2000-031 Multilevel algorithms for wavefront reduction. Y. F. Hu and J. A. Scott.

RAL-TR-2000-040 Componentwise fast convergence in the solution of full-rank systems of nonlinear equations. N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint.

RAL-TR-2001-003 Analysis and comparison of two general sparse solvers for distributed memory computers. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent and X. S. Li.

RAL-TR-2001-004 Performance and tuning of two distributed memory sparse solvers. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent and X. S. Li.

RAL-TR-2001-005 Incremental norm estimation for dense and sparse matrices. I. S. Duff and C. Vömel.

RAL-TR-2001-006 A null space algorithm for mixed finite element approximation of Darcy's equation. M. Arioli and G. Manzini.

RAL-TR-2001-011 The design of a portable parallel frontal solver for chemical process engineering problems. J. A. Scott.

RAL-TR-2001-017 Numerical methods for large-scale non-convex quadratic programming. N. I. M. Gould and Ph. L. Toint.

RAL-TR-2001-023 Dual variable methods for mixed-hybrid finite element approximation of the potential fluid flow problem in porous media. M. Arioli, J. Maryška, M. Rozložník, and M. Tůma.

RAL-TR-2001-026 An iterative working set method for large-scale non-convex quadratic programming. N. I. M. Gould and Ph. L. Toint.

RAL-TR-2001-032 The Sparse BLAS. I. S. Duff, M. A. Heroux, and R. Pozo.

RAL-TR-2001-033 Global convergence of a hybrid trust-region SQP-filter algorithm for general nonlinear programming. N. I. M. Gould and Ph. L. Toint.

RAL-TR-2001-034 A scaling algorithm to equilibrate both row and column norms in matrices. D. Ruiz.

RAL-TR-2001-037 A network programming approach in solving Darcy's equations by mixed finite-element methods. M. Arioli and G. Manzini.

RAL-TR-2001-039 Implementing Hager's exchange methods for matrix profile reduction. J.K. Reid and J.A. Scott.

CERFACS Reports

TR/PA/00/04 Experiments with sparse preconditioning of dense problems from electromagnetic applications. B. Carpentieri, I. S. Duff, and L. Giraud.

TR/PA/00/18 Level 2 and Level 3 Basic Linear Algebra Subprograms for Sparse Matrices: A Fortran 95 instantiation. I. S. Duff and C. Vömel.

TR/PA/00/72 Analysis, tuning and comparison of two general sparse solvers for distributed memory computers. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent and X. S. Li.

TR/PA/00/82 Implementing the Sparse BLAS in Fortran 95. I. S. Duff, C. Vömel, and M. Youan.

TR/PA/01/27 The implementation of the Sparse BLAS in Fortran 95. I. S. Duff and C. Vömel.

TR/PA/01/35 Sparse symmetric preconditioners for dense linear systems in electromagnetism. B. Carpentieri, I.S. Duff, L. Giraud, and M. Magolu monga Made.

ENSEEIHT-IRIT Reports

RT/APO/01/4 Impact of the implementation of MPI point-to-point communications on the performance of two general sparse solvers. P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, and X. Li.

IAN-CNR Reports

IAN-1179 A stopping criterion for the conjugate gradient algorithm in a finite element method framework. M. Arioli.

12 External Publications in 2000-2001

P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods in Appl. Mech. Engng.*, **184**, 501–520, 2000.

P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Analysis and Applications*, **23**(1), 15–41, 2001.

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. `MUMPS`: a general purpose distributed memory sparse solver. *in* A. H. Gebremedhin, F. Manne, R. Moe and T. Sørevik, eds, 'Proceedings of PARA2000, the Fifth International Workshop on Applied Parallel Computing, Bergen, June 18-21. Lecture Notes in Computer Science **1947**', pp. 122–131. Springer-Verlag, 2000.

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, **27**(4), 388–421, December 2001.

P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Performance and tuning of two distributed memory sparse solvers. *in* 'Proceedings of Tenth SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, Virginia, March 12th-14th, 2001', 2001.

M. Arioli. The use of QR factorization in sparse quadratic programming and backward error issues. *SIAM J. Matrix Anal. and Applics.*, **21**, 825–839, 2000.

M. Arioli and L. Baldini. A backward error analysis of a null space algorithm in sparse quadratic programming. *SIAM J. Matrix Analysis and Applications*, **23**, 425–442, 2001.

M. Arioli, E. Noulard, and A. Russo. Stopping criteria for iterative methods: Applications to PDE's. *CALCOLO*, **38**, 97–112, 2001.

Z.-Z. Bai, I. S. Duff, and A. J. Wathen. A class of incomplete orthogonal factorization methods. I: Methods and theories. *BIT*, **41**(1), 53–70, 2001.

B. Carpentieri, I. S. Duff, and L. Giraud. Robust preconditioning of dense problems from electromagnetics. *in* L. Vulkov, J. Waśniewski and P. Yalamov, eds, 'Numerical Analysis and Its Applications. Lecture Notes in Computer Science **1988**', pp. 170–178. Springer-Verlag, 2000.

B. Carpentieri, I. S. Duff, and L. Giraud. Sparse pattern selection strategies for robust Frobenius-norm minimization preconditioners in electromagnetism. *Numerical Linear Algebra with Applications*, **7**(7-8), 667–685, 2000.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-region methods*. SIAM, Philadelphia, 2000.

A. R. Conn, N. I. M. Gould, D. Orban, and Ph. L. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, **87**(2), 215–249, 2000.

I. S. Duff. The impact of high performance computing in the solution of linear systems: trends and problems. *J. Computational and Applied Mathematics*, **123**, 515–530, 2000.

I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Analysis and Applications*, **22**(4), 973–996, 2001.

N. I. M. Gould and Ph. L. Toint. SQP methods for large-scale nonlinear programming. *in* M. J. D. Powell and S. Scholtes, eds, 'System Modelling and Optimization, Methods, Theory and Applications', pp. 149–178, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic problems arising in optimization. *SIAM Journal on Scientific Computing*, **23**(4), 1375–1394, 2001.

N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Exploiting negative curvature directions in linesearch methods for unconstrained optimization. *Optimization Methods and Software*, **14**(1-2), 75–98, 2000.

N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. *SIAM Journal on Optimization*, **11**(4), 974–1002, 2001.

Y.F. Hu and J.A. Scott. A multilevel algorithm for wavefront reduction. *SIAM Journal on Scientific Computing*, **23**, 1352–1375, 2001.

C. Keller, N. I. M. Gould, and A. J. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM Journal on Matrix Analysis and Applications*, **21**(4), 1300–1317, 2000.

J. K. Reid. Implicit scaling of linear least squares problems. *BIT*, **40**(1), 146–157, 2000.

J. K. Reid, J. M. Rasmussen, and P. C. Hansen. The LINPACK Benchmark in Co-Array Fortran. *in* 'Proceedings of Sixth European SGI/Cray MPP Workshop, Manchester, 7-8 September 2000', 2000. `http://mrccs.man.ac.uk/mpp-workshop6/proc/index.htm`.

J.K. Reid and J.A. Scott. Reversing the row order for the row-by-row frontal method. *Numerical Linear Algebra with Applications*, **8**, 1–6, 2001.

Y. Saad, O. Axelsson, I. Duff, W.-P. Tang, H. van der Vorst, and A. Wathen, editors. *Special Issue: Preconditioning Techniques for Large Sparse Matrix Problems in Industrial Applications, SPARSE'99*, Vol. 7. Numerical Linear Algebra with Applications, 2000.

J.A. Scott. Row ordering for frontal solvers in chemical process engineering. *Computers in Chemical Engineering*, **24**, 1865–1880, 2000.

J.A. Scott. The design of a portable parallel frontal solver for chemical process engineering problems. *Computers in Chemical Engineering*, **25**, 1699–1709, 2001.

J.A. Scott. A parallel solver for finite element applications. *Int. Journal on Numerical Methods in Engineering*, **50**, 1131–1141, 2001.

J.A. Scott. Two-stage ordering for unsymmetric parallel row-by-row frontal solvers. *Computers in Chemical Engineering*, **25**, 323–332, 2001.