

Combining direct and iterative methods for the solution of large systems in different application areas¹

Iain S. Duff²

CERFACS Technical Report TR/PA/04/128

November 29, 2004

ABSTRACT

We first consider the size of problems that can currently be solved by sparse direct methods. We then discuss the limitations of such methods, where current research is going in moving these limitations, and how far we might expect to go with direct solvers in the near future. This leads us to the conclusion that very large systems, by which we mean three dimensional problems in more than a million degrees of freedom, require the assistance of iterative methods in their solution. However, even the strongest advocates and developers of iterative methods recognize their limitations when solving difficult problems, that is problems that are poorly conditioned and/or very unstructured. It is now universally accepted that sophisticated preconditioners must be used in such instances.

A very standard and sometimes successful class of preconditioners are based on incomplete factorizations or sparse approximate inverses, but we very much want to exploit the powerful software that we have developed for sparse direct methods over a period of more than thirty years. We thus discuss various ways in which a symbiotic relationship can be developed between direct and iterative methods in order to solve problems that would be intractable for one class of methods alone. In these approaches, we will use a direct factorization on a “nearby” problem or on a subproblem.

We then look at examples using this paradigm in four quite different application areas; the first solves a subproblem and the others a nearby problem using a direct method.

Keywords: sparse linear systems, distributed memory codes, multifrontal, supernodal, direct methods, comparison of codes.

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

¹Current reports available at http://www.cerfacs.fr/algos/reports/algos_reports_2004.html. Also appeared as Report RAL-TR-2004-033 from Rutherford Appleton Laboratory. The work was supported by the EPSRC Grant GR/S42170.

²duff@cerfacs.fr. CERFACS, 42 Ave G. Coriolis, 31057 Toulouse Cedex 01, France, and Atlas Centre, RAL, Oxon OX11 0QX, England.

Contents

1	Introduction	1
2	Extent and limitations of sparse direct codes	1
3	Iterative methods	2
4	The use of sparse direct codes with iterative techniques	3
5	Domain decomposition	4
6	Constrained optimization	7
7	Nonlinear water waves	8
8	Electromagnetics	9
9	Conclusions	10

1 Introduction

We examine approaches that combine direct and iterative methods to solve very large sparse problems. We believe that this is really the only way to solve systems when the number of degrees of freedom exceeds several million. We illustrate this approach by reference to earlier work by this author and others. The main novelty of this paper is that we collect and identify applications where an off-the-shelf sparse direct solver is used so that advantage can be taken of the increasing number of readily available sparse direct codes.

We first emphasize that sparse direct codes can be used to solve really quite large problems with test cases in the order of a million degrees of freedom becoming quite common. We summarize the current leading edge of the purely direct approach in Section 2. In this section we also indicate the limitations and indicate why other techniques are necessary.

In Section 4 we describe two ways in which direct solvers can be used and in the remaining sections give examples of each of these. We discuss domain decomposition in Section 5, an optimization application in Section 6, an application to nonlinear water waves in Section 7, and the preconditioning of dense systems from electromagnetics applications in Section 8.

2 Extent and limitations of sparse direct codes

In direct methods for solving

$$Ax = b, \tag{2.1}$$

we factorize the matrix into (normally) the product of a (sparse) lower triangular matrix, L say, and a (sparse) upper triangular matrix, U so that

$$PAQ = LU, \tag{2.2}$$

where P and Q are permutation matrices. The solution to (2.1) can then be easily accomplished through solving the two triangular systems

$$Ly = Pb$$

and

$$UQ^T x = y.$$

We refer the reader to Duff, Erisman and Reid (1986) for a more detailed presentation on sparse direct methods. If modern codes are used this can be remarkably efficient, the main limitation being the storage of L and U . Although the algorithms for matrix factorization attempt to maintain sparsity in these factors they are often far denser than A . For example, if a nested dissection ordering is used on the matrix from the discretization of a simple elliptic operator on a square (cubic) grid of side k , the storage for the factors

will be $\mathcal{O}(k^2 \log k)$ ($\mathcal{O}(k^4)$). Furthermore, it can be proved that this is an asymptotic lower bound for a direct method using Gaussian elimination.

The power of direct methods for solving sparse linear systems is not always appreciated, particularly by people working in the computational solution of partial differential equations. It is important to emphasize that systems of order more than one million are now solved almost routinely and matrices of this order are now included in benchmark tests. Although the conventional wisdom is that direct methods are more suited for two-dimensional discretizations, many of the larger standard test cases (for example from the PARASOL test set, <http://www.parallab.uib.no/parasol/>) are from three-dimensional models.

There has been much recent work on extending the range of problems that can be solved by direct methods. Most of this has been directed at exploiting parallelism and there are several codes available for doing this. It is important to note that nearly all modern sparse direct codes use dense matrix kernels at their inner loop and can thus exploit modern machine architectures very efficiently. A typical rule of thumb is that sparse codes will run at about half the rate of dense matrix-matrix multiply for large scale realistic factorizations. Other recent work on sparse direct solvers attempts to address the most severe limitation of direct methods by holding the factors out-of-core and even performing some of the factorization operations out-of-core.

However, for the rest of this presentation, we will look at another way of extending the scope of sparse direct methods so that we can solve systems one or two orders of magnitude larger. In addition, we will do this in a way that capitalizes on the increasing body of software for sparse direct methods by using a sparse direct code with very little or no modification. We will call this approach a hybrid direct-iterative approach. Another example of this can be found in many implementations of the multigrid approach where often the coarse grid problem (that could be considered a “nearby” problem) is solved by a direct method within an overall iterative approach.

Before continuing we will first very briefly discuss iterative methods or a subset of them.

3 Iterative methods

In an iterative method for solving (2.1), we start with a guess for the solution (often just the zero vector) and then successively refine this guess hopefully getting closer to the solution at each stage. The power of most iterative methods lies in the cheapness with which each iteration is performed. The problem is that very many iterations may be needed. This can be reduced by preconditioning the matrix but then the cost of each iteration is increased. Most of the current iterative methods use Krylov sequences and it is these that we will consider exclusively here.

In a Krylov-sequence based method, we compute an approximate solution to our problem from subspaces of increasing dimension usually with some optimality condition over all vectors in the subspace so that the solution will be obtained when the subspace is

of sufficient dimension. The Krylov sequence is defined as

$$\mathcal{K}^k(A; r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \dots, A^{k-1}r_0\}, \quad (3.1)$$

where $r_0 = b - Ax^{(0)}$ with $x^{(0)}$ the original “guess” at the solution. From the above, the approximate solution at stage k will be such that

$$x^{(k)} \in x^{(0)} + \mathcal{K}^k(A; r_0)$$

and we will require some optimality condition, for example that

$$\|b - Ax^{(k)}\|_2 \quad (3.2)$$

is minimized. One such method, applicable to general unsymmetric systems is **GMRES** (Saad and Schultz 1986). Krylov methods are discussed in detail in a recent book by van der Vorst (2003).

The idea of preconditioning the matrix/system can be thought of as multiplying the matrix by another so that the resulting product matrix requires far fewer iterations to approximate the solution with sufficient accuracy. If this is expressed as solving

$$MAx = Mb, \quad (3.3)$$

then it is important that the cost of multiplying by M and computing it is kept low. At one extreme, M could be the LU factorization (strictly its inverse), resulting in an iterative method with convergence in one step but with an expensive preconditioning.

However, although some of the techniques that we will now describe can be thought of as preconditioning, we will not view them in this way.

4 The use of sparse direct codes with iterative techniques

The essence of the approaches in this paper is that we will use a direct code with little or no modification in combination with an iterative method to effect the solution. Effectively, we will use our direct code either

1. on a subproblem
- or
2. on a “nearby” problem.

A very simple example of this approach might be when we have a coefficient matrix that is block diagonal except for very few outlying entries. If we then ran the direct method on the block diagonal part, it would both be trivially parallel and very efficient if the blocks on the diagonal were not too large. The iterative solution of the overall problem will thus

depend on the outliers and we might assume that the number of iterations would be closely related to the number of outliers. This would correspond to nothing other than a block Jacobi preconditioning of the original system.

In the remainder of this paper, we will consider the solution of large sparse equations in four quite different application areas by four different approaches. The first can be considered as solving a subproblem and the remaining three as solving a nearby system. These are:

1. Domain decomposition in semiconductor modelling,
2. Solution of augmented systems in constrained optimization,
3. Application in nonlinear water waves, and
4. Solution of boundary element problems in electromagnetics

and will be considered in the following four sections respectively.

5 Domain decomposition

My first example comes from the solution of highly nonlinear partial differential equations in semiconductor device modelling. This was the subject of recent work between my colleagues, Luc Giraud and Jean-Christophe Rioual, at CERFACS and Americo Marrocco at INRIA on some large scale industrial problems. The partial differential equations are discretized using mixed finite-element methods and the resulting nonlinear equations are solved by Newton's method. The earlier methods for solving the system that were used by INRIA included a direct method based on a skyline solver that was very memory hungry and required about 24 hours execution time on an large HP machine in serial mode on a problem with 30 thousand elements. They also tried Krylov methods preconditioned by ILU but found they were not robust enough for their application.

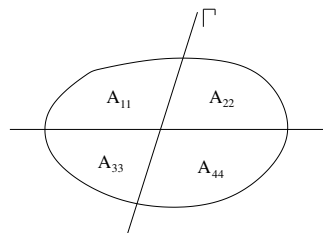


Figure 5.1: Illustration of domain decomposition

The approach used by Giraud, Marrocco and Rioual (2002) to solve this problem is based on domain decomposition where the region over which the problem is solved is split into subregions with artificial boundaries. For example, Figure 5.1 shows the subdivision of a region into four subregions with the artificial boundaries Γ . In this domain decomposition

approach, the individual subproblems (four in the case of Figure 5.1) can be discretized separately, leaving a remaining problem on the interface corresponding to the artificial boundary Γ . If the linearization of each subproblem is defined by the matrices A_{ii} , $i = 1, 4$, then the matrix representation of the subdivided problem in Figure 5.1 is given by

$$\begin{pmatrix} A_{11} & & & & A_{1\Gamma} \\ & A_{22} & & & A_{2\Gamma} \\ & & A_{33} & & A_{3\Gamma} \\ & & & A_{44} & A_{4\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma 3} & A_{\Gamma 4} & A_{\Gamma\Gamma} \end{pmatrix} \quad (5.1)$$

The solution to the system partitioned as in (5.1) can be obtained by first solving for the interface variables corresponding to the border blocks from the equations whose coefficient matrix is the Schur complement matrix

$$A_{\Gamma\Gamma} - \sum_{i=1,4}^4 A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma}.$$

A solution for the remaining variables is then found by backsubstitution and the solution of the decoupled systems with coefficient matrices A_{ii} , $i = 1, 4$. One way of implementing this domain decomposition approach is to use a direct method on the A_{ii} subproblems, which is one of the paradigms promoted in this paper, namely using a direct method on a subproblem.

If we expand the local problem and its interface, we obtain the matrix

$$\begin{pmatrix} A_{ii} & A_{i\Gamma} \\ A_{\Gamma i} & A_{\Gamma\Gamma}^{(i)} \end{pmatrix} \quad (5.2)$$

where

- A_{ii} : is the local subproblem,
- $A_{i\Gamma}$: is the boundary of the local problem, and
- $A_{\Gamma\Gamma}^{(i)}$: is the contribution to the stiffness matrix entries from variables on the artificial interface (Γ_i) around the i th subregion.

resulting in a contribution to the Schur complement of

$$S^{(i)} = A_{\Gamma\Gamma}^{(i)} - A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma},$$

called a local Schur (complement).

There are two quite distinct ways of tackling these subproblems that are termed “implicit” and “explicit”. In the implicit method, only A_{ii} is factorized (yielding factors L_i and U_i). In this case only the A_{ii} are passed to the direct solver and the local Schur complement $S^{(i)}$ is not computed but can be used to multiply a vector by multiplying the

vector by $A_{i\Gamma}$ and then solving for A_{ii} before multiplying the resulting vector by $A_{\Gamma i}$. In the explicit method, the whole of the matrix (5.2) is passed to the direct solver although pivots are only chosen from the A_{ii} block and the $S^{(i)}$ matrix is generated explicitly as a Schur complement of the factorization.

We would thus expect the factorizations for the implicit method to be cheaper but with more work required when using the Schur complements. Giraud et al. (2002) performed some experiments on a regular grid of 400×400 elements divided into 16 subdomains and obtained the results shown in Table 5.1. This means that, for the full solution of the linear

	Factorization	Matrix-vector
Implicit	10.2	1.60
Explicit	18.4	0.07

Table 5.1: Times in seconds on Origin 2000. Matrix-vector is the cost of computing the product of a vector with the Schur complement.

system, the explicit method will be faster if more than 5 iterations are required. Since this number of iterations is likely to be exceeded and since preconditioning and scaling for the interface problem is much simpler if the Schur complement is assembled, Giraud et al. (2002) propose using an explicit method and future results assume this.

For their further experiments, Giraud et al. (2002) use the code MUMPS (Amestoy, Duff, Koster and L'Excellent 2001) as the sparse direct solver. They try several approaches to solve the overall system. These include: running MUMPS on the whole system, using an explicit factorization on the local problems and MUMPS on the interface problem (Schur complement), and using an explicit factorization on the local problems and GMRES, with an additive Schwarz preconditioner (Carvalho, Giraud and Meurant 2001), on the interface problem. On a realistic problem of order 1,214,758 partitioned into 32 subdomains, they obtain the results shown in Table 5.2. We can see that the partitioning induced by

	Newton steps	Simulation time (seconds)
MUMPS on whole system, using AMD	166	3995
MUMPS on whole system, using ND	166	3250
MUMPS on local and on interface	166	2527
MUMPS on local, GMRES on interface	175	1654

Table 5.2: Runs on industrial problem on Origin 2000.

the domain decomposition technique has a benefit even if the whole system is solved by a direct method. Indeed looking at the first three lines of the table we see that the “ordering” induced by the domain decomposition is not only better than an approximate minimum degree ordering but is also better than the nested dissection ordering from MeTiS (METIS_NODEND) that is used by MUMPS. More significantly we see that using a hybrid

technique only marginally increases the number of nonlinear iterations but significantly reduces the overall time for the simulation.

The end result, reported by Giraud et al. (2002), is that the the code is far more robust and that the time for the main INRIA production runs was reduced from about 24 hours to 2 minutes, a very substantial reduction by anyone's reckoning.

6 Constrained optimization

The second application that we shall study is the solution of linear systems arising in constrained optimization. Specifically we will look at augmented matrices of the form

$$\begin{pmatrix} H + D & A^T \\ A & 0 \end{pmatrix} \quad (6.1)$$

where H is an approximation of the Hessian, D comes from penalty terms on inequality constraints (note that $d_{ii} = 0$ for unconstrained variables), and A is a matrix of constraints.

Here we will use a direct method on the nearby matrix

$$\begin{pmatrix} B & A^T \\ A & 0 \end{pmatrix} \quad (6.2)$$

and use the factorization of this to precondition the original system.

Duff and Orban have been experimenting with this approach and have used the **MA57** package (Duff 2004) as the direct solver and a conjugate gradient method as the iterative method. Even though the matrix (6.1) is indefinite, the conjugate gradient method is appropriate since the iterations can be constrained to lie in the subspace corresponding to the null space of A (Polyak 1969, Coleman 1994, Gould, Hribar and Nocedal 2001).

They have experimented with a range of values for B and we show some of these in Table 6.1. The problem is **CVXQP3_L** from CUTer (Gould, Orban and Toint 2003a) and the dimensions of the constraint matrix are 7500×10000 , with 39984 entries in the (1,1) block and 22497 in the matrix A . We see that the direct method can be used to produce

B	Matrix factorization		Iterations	
	Storage	Time	# its	Time
$H + D$	1,092,370	146.72	1	7.64
I	262,051	0.35	109	25.86
$\text{diag}(H + D)$	262,051	0.35	75	18.81

Table 6.1: Runs for problem **CVXQP3_L** on DEC Alpha workstation.

a very effective preconditioner and that, by adjusting the (1,1) block, the performance of the preconditioner can be improved. The overall time and storage for the solution are significantly less than for a direct method on the original problem. The options for the preconditioner discussed in Table 6.1 are included in the **GALAHAD** optimization library (Gould, Orban and Toint 2003b).

7 Nonlinear water waves

Our next example is taken from a model that is used for the study of the propagation of nonlinear wind-generated waves in harbours and coastlines conducted by Fuhrman and Bingham (2004). The equations governing this application form a highly coupled set of three 5th-order partial differential equations of the form

$$\begin{bmatrix} \mathcal{A}_{11} - \eta_x \mathcal{B}_{11} & \mathcal{A}_2 - \eta_x \mathcal{B}_{12} & \mathcal{B}_{11} + \eta_x \mathcal{A}_1 \\ \mathcal{A}_2 - \eta_y \mathcal{B}_{11} & \mathcal{A}_{22} - \eta_y \mathcal{B}_{12} & \mathcal{B}_{12} + \eta_y \mathcal{A}_1 \\ \mathcal{A}_{01} + h_x \mathcal{C}_{11} + h_y \mathcal{C}_{21} & \mathcal{A}_{02} + h_x \mathcal{C}_{12} + h_y \mathcal{C}_{22} & \mathcal{B}_0 - h_x \mathcal{C}_{13} - h_y \mathcal{C}_{23} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} U \\ V \\ 0 \end{bmatrix} \quad (7.1)$$

that are discretized in time using a 4-th order, 4-stage Runge-Kutta method and in space using finite differences.

Although the discretized equations can be solved by a direct method (the matrix order can be up to say 100,000 with about 100 entries in each row), it is very expensive since the matrix changes at each time step. However, if we drop the time dependent terms in the local wave elevation η , the matrix of the discretized system is of the form

$$\begin{pmatrix} A_{11} & A_2 & B_{11} \\ A_2 & A_{22} & B_{12} \\ A_{01} + h_x C_{21} & A_{02} + h_x C_{12} + h_y C_{22} & B_0 - h_x C_{13} - h_y C_{23} \end{pmatrix} \quad (7.2)$$

and Fuhrman and Bingham (2004) factorize the matrix (7.2) using the HSL routine **MA41** and use this “linearized” matrix as a preconditioner for GMRES.

For the solution of the Boussinesq-type equations in shallow water on a 33×33 grid (with 3 variables per grid point), they get the results shown in Table 7.1. From this it

Preconditioning	# its per time step	Time
MA41	3-12	80.5
ILUT	3-14	61.2
NONE	12-23	78.5

Table 7.1: Shallow water runs. Times in seconds on a Dell P4.

would appear that the preconditioner using **MA41** is not very successful but, if we look at the performance for deep(er) water and more nonlinearity, we see from the results in Table 7.2 that convergence even with an ILUT preconditioner is very slow. Their experiments for even deeper water show that only the **MA41** preconditioning is effective. The times for solving the full problem (7.1) using **MA41** are also shown in this table. From this we see that, although faster than using an ILUT preconditioning, the hybrid approach that we are championing in this paper is very much the best. Such differences become even more pronounced when the depth is increased further or the grid is refined.

Preconditioning	# its per time step	Time
MA41	2-9	64.9
ILUT	14-40	309
Direct		269

Table 7.2: Deeper water runs. Times in seconds on a Dell P4.

8 Electromagnetics

This application differs significantly from the previous ones inasmuch as the coefficient matrices are dense since they are obtained from the boundary element discretization of problems in electromagnetic scattering. We will solve the resulting set of linear equations using GMRES where we use the fast multipole method to effect the matrix-vector multiplication and obtain a preconditioning matrix through a sparsification of the dense coefficient matrix. There are a wide range of preconditioners described by Carpentieri, Duff and Giraud (2000), one of which (termed SLU) is obtained by factorizing the sparsified matrix using a direct code, thereby following the paradigm of using the direct method on a nearby problem. The direct code used by Carpentieri et al. (2000) is MUMPS (Amestoy et al. 2001).

If we compare the number of iterations required by GMRES(80) on a model problem of a satellite of order 1701, then we have the results shown in Table 8.1 where different sparsifications of the matrix are used so that the resulting preconditioners have the same work to apply per iteration. Most of the names for the preconditioners are self-explanatory; the SPAI code uses a modification of the algorithm of Grote and Huckle (1997) to compute the sparse approximate inverse whereas FROB just computes a sparse approximate inverse of prefixed sparsity.

Preconditioning	# iterations
JACOBI	491
SSOR	301
ILU(0)	474
SPAI	157
FROB	59
SLU	25

Table 8.1: Performance of different preconditioners for the satellite problem.

Alléon, Carpentieri, Duff, Giraud, Martin and Sylvand (2003) have tried to use this approach on more realistic industrial problems and have met with limited success, largely because the number of entries in the factors becomes prohibitive if the SLU preconditioner is good enough to be effective. That is, the nearby problem is too nearby for a sparse factorization to be sensible. However, we show some results on larger problems in Table 8.2

Test case	Order	Entries in millions			GMRES its	
		FROB	SLU	sp(A)	SLU	FROB
AIRCRAFT	94,704	25	49	12.5	366	745
ALMOND	104,793	21	87	10.6	201	233
CETAF	134,775	20	79	9.9	516	617

Table 8.2: Performance of SLU and FROB preconditioners on large test cases.

where we see that the SLU approach requires less iterations than FROB, although the fact that FROB is implemented out-of-core means that it is currently our preferred approach for very large problems.

9 Conclusions

We have shown several examples in very different application areas where we can use a combination of direct and iterative methods to solve really large problems. In all cases an off-the-shelf sparse direct solver has been used enabling it to solve problems of greater complexity than it could have done if run on the original problem.

Acknowledgments

We are very grateful to Luc Giraud for his comments and discussion on Sections 5 and 8, and to David Fuhrman for his comments and discussion on Section 7. We would also like to thank Andy Wathen and Nick Gould for their comments on a draft of this paper.

References

- Alléon, G., Carpentieri, B., Duff, I. S., Giraud, L., Martin, E. and Sylvand, G. (2003), Efficient parallel iterative solvers for the solution of large dense linear systems arising from the boundary element method in electromagnetism, Technical Report TR/PA/03/65, CERFACS, Toulouse, France.
- Amestoy, P. R., Duff, I. S., Koster, J. and L'Excellent, J.-Y. (2001), ‘A fully asynchronous multifrontal solver using distributed dynamic scheduling’, *SIAM J. Matrix Analysis and Applications* **23**(1), 15–41.
- Carpentieri, B., Duff, I. S. and Giraud, L. (2000), ‘Sparse pattern selection strategies for robust Frobenius-norm minimization preconditioners in electromagnetism’, *Numerical Linear Algebra with Applications* **7**(7-8), 667–685.

- Carvalho, L. M., Giraud, L. and Meurant, G. (2001), ‘Local preconditioners for two-level non-overlapping domain decomposition methods’, *Numerical Linear Algebra with Applications* **8**(4), 207–227.
- Coleman, T. F. (1994), Linearly constrained optimization and projected preconditioned conjugate gradients, in J. Lewis, ed., ‘Proceedings of the Fifth SIAM Conference on Applied Linear Algebra’, SIAM Press, Philadelphia, PA, pp. 118–122.
- Duff, I. S. (2004), ‘MA57 – A code for the solution of sparse symmetric indefinite systems’, *ACM Trans. Math. Softw.* **30**(2), 118–144.
- Duff, I. S., Erisman, A. M. and Reid, J. K. (1986), *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, England.
- Fuhrman, D. R. and Bingham, H. B. (2004), ‘Numerical solutions of fully nonlinear and extremely dispersive Boussinesq equations in two horizontal dimensions’, *Int. J. Numer. Meth. Fluids* **44**(3), 231–256.
- Giraud, L., Marrocco, A. and Rioual, J.-C. (2002), Iterative versus direct parallel substructuring methods in semiconductor device modeling, Technical Report TR/PA/02/114, CERFACS, Toulouse, France. To appear in *Numerical Linear Algebra with Applications*.
- Gould, N. I. M., Hribar, M. E. and Nocedal, J. (2001), ‘On the solution of equality constrained quadratic programming problems arising in optimization’, *SIAM J. Scientific Computing* **23**(4), 1376–1395.
- Gould, N. I. M., Orban, D. and Toint, P. L. (2003a), ‘CUTEr and SifDec: A constrained and unconstrained testing environment, revisited’, *ACM Trans. Math. Softw.* **29**(4), 373–394.
- Gould, N. I. M., Orban, D. and Toint, P. L. (2003b), ‘DGALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization’, *ACM Trans. Math. Softw.* **29**(4), 353–372.
- Grote, M. J. and Huckle, T. (1997), ‘Parallel preconditioning with sparse approximate inverses’, *SIAM J. Scientific Computing* **18**(3), 838–853.
- Polyak, B. T. (1969), ‘The conjugate gradient method in extremal problems’, *U.S.S.R. Computational Mathematics and Mathematical Physics* **9**, 94–112.
- Saad, Y. and Schultz, M. H. (1986), ‘GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems.’, *SIAM J. Scientific and Statistical Computing* **7**, 856–869.
- van der Vorst, H. A. (2003), *Iterative Krylov Methods for Large Linear systems*, Cambridge University Press.