Acta Numerica (2025), pp. 891–1010
doi:10.1017/S0962492924000059

Sparse linear least-squares problems

Jennifer Scott
Department of Mathematics and Statistics,
School of Mathematical, Physical and Computational Sciences,
University of Reading, Reading RG6 6AQ, UK
and Scientific Computing Department, STFC Rutherford Appleton Laboratory,
Harwell Campus, Didcot, Oxfordshire, OX11 0QX, UK
E-mail: jennifer.scott@reading.ac.uk

Miroslav Tůma
Department of Numerical Mathematics, Faculty of Mathematics and Physics,
Charles University, Sokolovská 49/83, 186 75 Praha 8, Czech Republic
E-mail: mirektuma@karlin.mff.cuni.cz

Least-squares problems are a cornerstone of computational science and engineering. Over the years, the size of the problems that researchers and practitioners face has constantly increased, making it essential that sparsity is exploited in the solution process. The goal of this article is to present a broad review of key algorithms for solving large-scale linear least-squares problems. This includes sparse direct methods and algebraic preconditioners that are used in combination with iterative solvers. Where software is available, this is highlighted.

2020 Mathematics Subject Classification: Primary 65F20, 65F50
Secondary 65F05, 65F10, 65F08

CONTENTS

1. Introduction and basic concepts

This article seeks to present an overview of current approaches to solving large-scale linear least-squares problems. These problems typically occur as part of a more complex computational problem. Most real-world problems are nonlinear, and many approaches to understanding and solving them proceed by locally replacing the nonlinear problem with a linear one. Specifically, analysing and solving nonlinear least-squares problems frequently requires the solution of a sequence of linear least-squares problems. More generally, almost any problem with sufficient data to overdetermine a solution needs an approximation method: least-squares – the first and best-known technique for fitting models to data – remains central to scientific computing

1.1. Background

The use of models is fundamental throughout scientific computing. Models are used in science and engineering, finance and economics and, more recently, in rapidly developing areas such as machine learning, AI and speech processing. Models are only useful when calibrated against real-world systems and driven by data. The huge explosion in the availability of data and computing resources has led to many new opportunities to improve the effectiveness and reliability of models. For this, algorithms that incorporate the data into the models are needed; therein lies the continued importance of the least-squares method.

The least-squares method provides a solution to the problem of adjusting the parameters of a model function to best fit a given data set. A simple model in two dimensions is a straight line. In this case, linear least-squares computes the best-fitting line through a set of points, that is, it minimizes the sum of the squares of the distances between the points and the line. The method originated during the eighteenth century in the fields of astronomy and geodesy, as scientists and mathematicians sought to address the challenges of ocean navigation. Being able to accurately and reliably predict the behaviour of celestial bodies was key for ships in open seas where land sightings cannot be used for navigation. The least-squares method was first published by Adrien-Marie Legendre (1805), although it is also credited to Carl Friedrich Gauss (1809), who made significant contributions to the mathematical formulation and theoretical understanding of least-squares around the same time as Legendre. Gauss successfully used the method to approximate the orbit of the newly discovered asteroid Ceres from the few observations that had been made of it before it was lost in the glare of the sun.

Today, least-squares is an essential tool in mathematics and statistics that is widely used in a variety of fields. It is the simplest and most commonly applied form of linear regression, which in turn is the most straightforward machine learning algorithm. Least-squares problems can be linear or nonlinear, depending on whether or not the residuals are linear in the unknowns. The nonlinear problem is

usually solved by an iterative procedure; at each iteration the system is approximated by a linear one, and thus the core calculation is similar in both cases.

Diverse application areas of least-squares include the following.

- Medicine, e.g. to study the impact of environmental factors on human health by using variables such as air quality, water quality, pollution levels and exposure levels to assess risks, design interventions and monitor outcomes.
- Finance, e.g. to help quantify the relationship between two or more variables, such as a stock's share price and its earnings per share.
- Marketing, e.g. to model the relationship between advertising spending and sales.
- Image processing, e.g. to enhance, restore or compress images by using models that capture the features, structures or patterns of the images.
- Geophysics, e.g. to interpret subsurface structures from field data by minimizing a target objective function expressed as the sum of squared residuals between observed and modelled data, to estimate the physical properties, locations and shapes of geological formations or anomalies.
- Climate change, e.g. to study the relationship between greenhouse gas emissions and global temperature, and to predict future temperature changes based on emissions scenarios.
- Sports science, e.g. to measure the performance of athletes by using variables such as scores, statistics, rankings and ratings to predict outcomes, evaluate strengths and weaknesses, and identify trends and patterns.

The focus in this article is on solution methods for large-scale least-squares problems arising from large data sets; we do not examine specific application areas but introduce general techniques that are widely applicable.

1.2. Introduction to linear least-squares fitting

To set the scene, let us assume we have m data points

$$(t_1, b_1),\ (t_2, b_2),\ \ldots,\ (t_m, b_m),$$

and the relationship

$$b_i = \Gamma(t_i) + e_i, \quad 1 \le i \le m,$$

where the unknown function $\Gamma(t)$ describes the noise-free data. The (unknown) data errors or noise $e_1, e_2, \ldots, e_m$ represent measurement errors and/or random variations in the physical process that generates the data. The b_i are termed the observations. We want to approximate $\Gamma(t)$ in the interval $[t_1, t_m]$.[1] The approximation is given by the fitting (or prediction) model $\mathcal{H}(x, t)$, where the n

[1] Without loss of generality, we assume $t_1 \le t_2 \le \ldots \le t_m$.

parameters $x = (x_1, x_2, \ldots, x_n)^\top$ that characterize the model are to be determined from the given noisy data. The least-squares method is a standard technique for determining the unknown parameters in the fitting model. Let the residual r_i associated with the ith data point be given by

$$r_i = r_i(x) = b_i - \mathcal{H}(x, t_i), \quad i = 1, 2, \ldots, m. \tag{1.1}$$

If we assume that all the data errors are uncorrelated and of the same size, that is, the errors e_i have mean zero and identical variance, then the least-squares fit minimizes the sum of the squared residuals

$$\min_{x \in \mathbb{R}^n} \mathcal{F}(x) = \sum_{i=1}^{m} r_i(x)^2 = \sum_{i=1}^{m} (b_i - \mathcal{H}(x, t_i))^2. \tag{1.2}$$

More generally, if the standard deviation depends on i, then from the Maximum Likelihood Principle, the weighted residuals are minimized, with the non-zero weights equal to the reciprocals of the standard deviations, that is,

$$\min_x \sum_{i=1}^{m} \left(\frac{r_i(x)}{\varsigma_i} \right)^2 = \min_x \sum_{i=1}^{m} \left(\frac{b_i - \mathcal{H}(x, t_i)}{\varsigma_i} \right)^2,$$

where ς_i is the standard deviation of e_i. Observe that the residual can be written as the sum of two terms:

$$r_i = e_i + (\Gamma(t_i) - \mathcal{H}(x, t_i)), \quad i = 1, 2, \ldots, m.$$

The data error e_i comes from the measurements, and the approximation error $\Gamma(t_i) - \mathcal{H}(x, t_i)$ is the difference between the data function and the fitting model.

In linear data fitting, the model is of the form

$$\mathcal{H}(x, t) = \sum_{j=1}^{n} x_j f_j(t),$$

where the $f_j(t)$ are called the model basis functions. The choice of the $f_j(t)$ depends on the objective of the data fitting. They may be given by the underlying mathematical model that describes the data or chosen from functions that give the desired approximation and enable stable computations. Defining the matrix $A \in \mathbb{R}^{m \times n}$ to be

$$A = \begin{pmatrix} f_1(t_1) & f_2(t_1) & \ldots & f_n(t_1) \\ f_1(t_2) & f_2(t_2) & \ldots & f_n(t_2) \\ \ldots & \ldots & \ldots & \ldots \\ f_1(t_m) & f_2(t_m) & \ldots & f_n(t_m) \end{pmatrix},$$

for the unweighted (ordinary) linear data-fitting problem we have the relations

$$r = b - Ax \quad \text{and} \quad \sum_{i=1}^{m} r_i^2 = \|r\|_2^2 = \|b - Ax\|_2^2.$$

Here $\|.\|_2$ denotes the Euclidean vector norm $\|x\|_2 = (x^\top x)^{1/2}$. The basic linear least-squares problem that we seek to solve is the following.

Given $A \in \mathbb{R}^{m\times n}$ with $\text{rank}(A) = rk \le \min(m, n)$ and $b \in \mathbb{R}^m$,
find $x \in \mathbb{R}^n$ that minimizes $\|b - Ax\|_2$.

For the weighted problem, the covariance matrix W is a diagonal matrix with entries $w_{ii} = 1/\varsigma_i^2$. We then have

$$\sum_{i=1}^{m} \left(\frac{r_i(x)}{\varsigma_i} \right)^2 = \|W^{-1/2}(b - Ax)\|_2^2.$$

More generally, for a symmetric positive (semi)definite matrix W, the weighted linear least-squares problem is

$$\min_{x\in\mathbb{R}^n} \mathcal{F}(x) = \min_{x\in\mathbb{R}^n} \|b - Ax\|_{W^{-1}}, \tag{1.3}$$

where $\|u\|_{W^{-1}} = (u^\top W^{-1} u)^{1/2}$.

Our main emphasis is on overdetermined problems, i.e. $m > n$. In this case, it is straightforward to show that x is a solution of the least-squares problem if and only if it satisfies the $n \times n$ normal equations

$$Cx = A^\top b, \quad C = A^\top A, \tag{1.4}$$

or, in the weighted case, the generalized normal equations

$$A^\top W^{-1} A x = A^\top W^{-1} b.$$

The concept of a pseudoinverse generalizes that of the inverse of a square matrix; it can be used to express the general solution of the linear least-squares problem; details are given in Björck (2024) and Laub (2005).

Lemma 1.1. Let $A \in \mathbb{R}^{m\times n}$ with $\text{rank}(A) = rk$. Then there exist unitary matrices $U = (u_1, \ldots, u_m) \in \mathbb{R}^{m\times m}$, $V = (v_1, \ldots, v_n) \in \mathbb{R}^{n\times n}$ such that

$$A = U\Sigma V^\top = U \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} V^\top, \tag{1.5}$$

with $\Sigma_1 = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{rk})$. The σ_i are the singular values of A, which are assumed to be ordered so that

$$\sigma_{\max} = \sigma_1 \ge \ldots \ge \sigma_{rk} > \sigma_{rk+1} = \cdots = \sigma_{\min} = 0.$$

The factorization (1.5) is the singular value decomposition (SVD) of A, and the matrix

$$A^\dagger = V\Sigma^\dagger U^\top = V \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^\top$$

is termed the Moore–Penrose pseudoinverse (or simply the pseudoinverse) of A. The general expression for the solution of the linear least-squares problem $\min_x \|b - Ax\|_2$ is

$$x = A^\dagger b + (I - A^\dagger A)w,$$

where $w \in \mathbb{R}^n$ is an arbitrary vector.

In most practical cases, the least-squares solution can be expressed in a simpler way, as shown by the following results; see, for example, the book by Björck (2024) for details.

Lemma 1.2. Let $A \in \mathbb{R}^{m\times n}$ with $m > n$. The normal matrix $C = A^\top A$ is symmetric positive definite (SPD) if and only if $\text{rank}(A) = n$. The unique least-squares solution and corresponding residual are then given by

$$x = (A^\top A)^{-1} A^\top b \quad \text{and} \quad r = b - A(A^\top A)^{-1} A^\top b, \tag{1.6}$$

and the pseudoinverse of A is $A^\dagger = (A^\top A)^{-1} A^\top$.

If $\text{rank}(A) < n$ (i.e. A is rank-deficient) then the null space of A is of dimension $n - \text{rank}(A) > 0$ and the solution of the least-squares problem is not unique. In this case, we often seek the unique least-norm solution, that is,

$$\min_{x\in\mathcal{S}} \|x\|_2, \quad \mathcal{S} = \{x \in \mathbb{R}^n \mid \|b - Ax\|_2 = \min\}.$$

But note that because there are infinitely many possible solutions, in practice the choice as to which solution is singled out should reflect the physical model that underlies the problem (Hansen 2010).

Lemma 1.3. Let x be a solution of the problem $\min_x \|b - Ax\|_2$. Then x is a least-squares least-norm solution if and only $x = A^\top z$, $z \in \mathbb{R}^m$. If the system $Ax = b$ is consistent, then the solution of the least-norm problem

$$\min_x \|x\|_2 \quad \text{subject to} \quad Ax = b, \tag{1.7}$$

satisfies the normal equations of the second kind

$$AA^\top z = b, \quad x = A^\top z. \tag{1.8}$$

If $\text{rank}(A) = m$, then $AA^\top$ is non-singular and the solution to (1.8) is unique. The solution is $x = A^\dagger b$, where $A^\dagger = A^\top (AA^\top)^{-1}$ is the pseudoinverse.

As originally proposed in Bartels, Golub and Saunders (1970), the normal equations (1.4) are equivalent to the linear equations $r = b - Ax$ and $A^\top r = 0$. Together these can be written as the $(m + n) \times (m + n)$ augmented system

$$K \begin{pmatrix} z \\ x \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \quad \text{with} \quad K = \begin{pmatrix} I & A \\ A^\top & 0 \end{pmatrix}, \tag{1.9}$$

with $z = r$ and $c = 0$. The symmetric indefinite matrix K is non-singular if and only if $\text{rank}(A) = n$.

Lemma 1.4. If $\operatorname{rank}(A) = n$, then the augmented system (1.9) has a unique solution that solves the primal and dual least-squares and least-norm problems

$$\min_{x \in \mathbb{R}^n} \left(\frac{1}{2} \|b - Ax\|_2^2 + c^\top x \right)$$

and

$$\min_{z \in \mathbb{R}^m} \frac{1}{2} \|z - b\|_2^2 \quad \text{subject to} \quad A^\top z = c.$$

Our interest is in the case that A is large and sparse, that is, many of its entries are zero and these zeros need to be exploited in the solution process. Indeed, unless the sparsity is exploited, the size of problems that can be tackled is severely limited, even when using modern computers with large memories. The sparsity pattern $\mathcal{S}\{A\}$ of $A = \{a_{ij}\}$ is defined to be the set of non-zeros, that is,

$$\mathcal{S}\{A\} = \{(i, j) \mid a_{ij} \neq 0,\ 1 \leq i \leq m,\ 1 \leq j \leq n\}.$$

Although Lemma 1.1 shows theoretically that the pseudoinverse based on the SVD gives the least-norm least-squares solution, our discussions avoid the SVD as it is impractical for large sparse A. Many of the approaches we consider target $A \in \mathbb{R}^{m \times n}$ with $m > n$ and $\operatorname{rank}(A) = n$. But similarities between this and the underdetermined least-squares problem ($m < n$), where the least-norm solution is described in Lemma 1.3, enable techniques applied to the normal matrix $A^\top A$ to be applied to the matrix of the normal equations of the second kind (1.8). Moreover, as will be seen in Section 1.7, regularizing the underdetermined problem leads to an overdetermined problem of full rank.

1.3. An introduction to sparse linear solvers

The normal equations (1.4) and the augmented system (1.9) are both sparse square linear systems of equations; solving either yields the solution of the linear least-squares problem. Consider the generic large sparse linear system of equations

$$By = d, \tag{1.10}$$

where $B \in \mathbb{R}^{n \times n}$ and $d \in \mathbb{R}^n$. Assuming B is of full rank, there are many methods for computing the solution $y \in \mathbb{R}^n$; see, for example, the recent books by Duff, Erisman and Reid (2017) and Scott and Tůma (2023) and the review article by Davis, Rajamanickam and Sid-Lakhdar (2016), and the comprehensive bibliographies that they include. For least-squares problems, we are interested in the symmetric case ($B = B^\top$).

The majority of algorithms for solving (1.10) fall into two main categories: direct methods and iterative methods (with hybrid methods combining techniques from both classes). Direct methods transform B using a finite sequence of elementary transformations into a product of simpler sparse matrices in such a way that solving linear systems of equations with these factor matrices is comparatively easy and

inexpensive. For the SPD normal matrix $A^\top A$, we can compute a Cholesky factorization $A^\top A = LL^\top$, where the factor L is a lower triangular matrix, while for the augmented system we can compute a factorization $K = LDL^\top$, where L is a unit lower triangular matrix and D is block diagonal with the blocks on the diagonal of size 1 and 2. Alternatively, symmetry can be ignored and a factorization of the form $K = LU$ computed, where L and U are lower and upper triangular matrices, respectively. Solving linear systems with a triangular matrix is much more straightforward than for a general matrix and historically has been much less expensive than the cost of the factorization. As the use of parallel algorithms on modern computer architectures has substantially reduced the cost of matrix factorizations, triangular solves have become relatively more expensive because of their inherently serial nature, but see the progress in parallel triangular solves discussed in Jin, Pei, Wang and Qi (2021) and the references therein. Moreover, in general, L has more non-zero entries than A, and if the amount of fill-in is high, then (some of) the advantage of having a triangular matrix will be lost. An important question is whether the system can be preordered to reduce the fill-in in the factors; this is discussed in Section 2.3. For an LDLT or LU factorization of an indefinite matrix, further permutations are generally required during the factorization to ensure numerical stability.

For the linear least-squares problem, rather than factorize the normal matrix or the augmented system matrix, the QR factorization

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1 \quad Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R, \quad Q_1 \in \mathbb{R}^{m \times n}, \tag{1.11}$$

can be computed (Francis 1961). Here $Q = (Q_1 \quad Q_2) \in \mathbb{R}^{m \times m}$ is an orthogonal matrix and $R \in \mathbb{R}^{n \times n}$ is an upper triangular matrix. Because multiplication by orthogonal matrices does not change the Euclidean norm, it follows that

$$\|b - Ax\|_2^2 = \|Q^\top (b - Ax)\|_2^2 = \|Q_1^\top b - Rx\|_2^2 + \|Q_2^\top b\|_2^2.$$

The solution of the least-squares problem and the residual can be computed by solving

$$Rx = d_1, \; r = Q \begin{pmatrix} 0 \\ d_2 \end{pmatrix}, \quad \text{where} \quad Q^\top b = \begin{pmatrix} Q_1^\top b \\ Q_2^\top b \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}.$$

$Q^\top b$ can formally be obtained by applying the QR factorization to $(A \quad b)$. This enables storing Q to be avoided. Note that

$$A^\top A = (Q_1 R)^\top Q_1 R = R^\top (Q_1^\top Q_1) R = R^\top R, \tag{1.12}$$

and thus the R factor is mathematically equivalent to the transpose of the L factor of the Cholesky factorization of the normal matrix. The normal equations can be rewritten as

$$R^\top Rx = A^\top b. \tag{1.13}$$

These are called the semi-normal equations (SNE). They offer an advantage if a QR factorization has been computed but the orthogonal factor Q has not been stored and a problem with a new right-hand side b must be solved.

Direct methods built on matrix factorizations are designed to be robust so that, properly implemented, they can be confidently used as block-box solvers for computing solutions with predictable accuracy. However, they can be expensive, requiring large amounts of memory, which can increase rapidly with the size of the system matrix B, and they may compute solutions to an accuracy that is either not needed or not warranted by the supplied data. In large-scale applications, the matrix B, even though it is sparse, may be too large to be held explicitly. In such instances, solution methods are needed that use subroutines or functions to compute multiplications with B and $B^\top$ in a matrix-free fashion, often exploiting graphics processing units (GPUs) or other hardware accelerators. Iterative methods for solving the linear system (1.10) compute a sequence of approximations

$$y^{(1)}, y^{(2)}, y^{(3)}, \ldots$$

that (hopefully) converge to the solution y of the linear system in an acceptable number of iterations. The number of iterations depends on the initial guess $y^{(1)}$, the matrix B and right-hand side vector d as well as the accuracy that is wanted in y. Iterative methods use B indirectly, through matrix–vector products, and their memory requirements are limited to a (small) number of vectors of length the size of B. They can be terminated as soon as the required accuracy in the computed solution is achieved. Unfortunately, frequently convergence does not happen or the number of iterations is unacceptably large; in such cases, preconditioning is needed. The aim of preconditioning is to speed up convergence by transforming the given linear system into an equivalent system (or one from which it is easy to recover the solution of the original system) that has nicer numerical properties. For example, for the consistent underdetermined least-squares problem $\min \|x\|_2$ subject to $Ax = b$, with the non-singular preconditioner M_L, the left-preconditioned problem is

$$\min_{x \in \mathbb{R}^n} \|x\|_2 \quad \text{subject to} \quad M_L^{-1} A x = M_L^{-1} b.$$

For the overdetermined least-squares problem, using right-preconditioning, the problem becomes

$$\min_{z \in \mathbb{R}^n} \|b - A M_R^{-1} z\|_2, \quad x = M_R^{-1} z. \tag{1.14}$$

Observe that in this case, left-preconditioning changes the objective function. Right-preconditioning corresponds to symmetric (or split) preconditioning of the normal equations

$$M_R^{-\top} A^\top A M_R^{-1} z = M_R^{-\top} A^\top b, \quad M_R x = z.$$

Here $M = M_R M_R^\top$ is the normal matrix preconditioner in factored form. Choosing a suitable preconditioner is a challenging task. Multiple factors influence the

preconditioner effectiveness, including features of the sparse matrix, the computational architecture, and the data structures employed. Possible preconditioners for least-squares problems are discussed in Section 6.

1.4. The condition number of least-squares problems

When solving least-squares problems, we need to be more aware of rank deficiency than for linear systems of equations. While the latter often come from applications for which there are guarantees on the non-singularity and possibly also on conditioning of the system matrix, this may not be the case for least-squares problems. The condition number of a problem quantifies its sensitivity to perturbations to the data. If A is square and of full rank, then its normwise condition number is $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$. For an arbitrary rectangular matrix, this generalizes to $\kappa_2(A) = \|A\|_2\|A^{\dagger}\|_2$. If A has rank rk, then $\kappa_2(A)$ is equal to the ratio of the largest to the smallest non-zero singular values, that is,

$$\kappa_2(A) = \sigma_{\max}(A)/\sigma_{rk}(A). \tag{1.15}$$

A matrix with a large condition number is said to be ill-conditioned, otherwise it is well-conditioned. In particular, if $\kappa_2(A) > \epsilon^{-1}$, where ϵ is the machine precision, then A is said to be numerically rank-deficient. If the columns of A are orthonormal, then $\kappa_2(A) = 1$, indicating a perfectly conditioned matrix. More generally, if A is of full rank and R is its QR factor then

$$\kappa_2(A) = \kappa_2(R) = (\kappa_2(A^{\top}A))^{1/2}. \tag{1.16}$$

In many practical applications, ill-conditioning and possible rank deficiency is a common problem that is often not observable in A before its factorization is attempted. In regression problems, the columns of A correspond to explanatory factors. As a simple example, we may want to use height, weight and age to explain the probability of some disease. In this case, ill-conditioning occurs when these factors are correlated; for instance, in the sample population, height and age may be good predictors of weight.

Unlike for square linear systems, the sensitivity of a least-squares problem depends on the vector b as well as on the matrix A. The normwise condition number $\kappa_{LS}(A, b)$ that expresses sensitivity to perturbations in A is given by Hansen, Pereyra and Scherer (2013):

$$\kappa_{LS}(A, b) = \kappa_2(A)\left(1 + \kappa_2(A)\,\frac{\|r\|_2}{\|A\|_2\|x\|_2}\right), \quad r = b - Ax.$$

We also refer to Wedin (1973), Golub and Van Loan (1996), Demmel (1997) and Björck (2024). This definition can be interpreted as saying that the sensitivity of the least-squares problem is measured by $\kappa_2(A)$ when the residual of the least-squares solution is small or zero, and by $\kappa_2(A)^2$ otherwise, that is, the accuracy of the computed solution depends on the square of the condition number of A.

Error bounds for the least-squares solution combine expressions for the sensitivity with expressions for the perturbations within the solution method. There are two potential problems in using the normal equations. Firstly, information may be lost when the inner products to compute the entries of the normal matrix $A^\top A$ are accumulated (Björck 2024). Even if the inner products are accumulated in double precision arithmetic, a serious loss of information can occur when the computed $A^\top A$ is stored in the working precision. In general, whenever $\kappa_2(A) \geq \epsilon^{-1/2}$, we can expect the computed normal matrix to be singular (or indefinite), in which case a Cholesky factorization will break down. Secondly, although Cholesky factorization algorithms are backward stable, solution methods that explicitly form the normal equations are not backward stable because it can be shown that the best backward error bound contains a factor $\kappa_2(A)$; this is discussed in Higham (2002). Using the normal equations is attractive if the problem is well-conditioned or if only modest accuracy is required; see Higham and Stewart (1987) for a discussion of why the use of the normal equations can be justified.

Using a QR factorization is often more stable because it is always backward stable and the relative error in the solution can be bound using $\kappa_{LS}(A, b)$. However, if $\kappa_2(A)$ and the norm of the residual is large, the QR method can also return an inaccurate solution. In practice, sparse QR factorization methods (which will be discussed in Section 4) are significantly more expensive than sparse Cholesky factorizations (Section 3), making them impractical for very large problems that require real-time solutions. Furthermore, most algebraic preconditioners are for the normal equation formulation (Section 6). Thus, although, as stated in Higham (2002) and Higham and Pranesh (2021), solving the normal equations is often deprecated by numerical analysts, in practice the normal equations are widely used. For example, in many statistical applications, the entries of A can be contaminated by measurement errors that are large relative to the round-off level; the effects of rounding errors are then likely to be insignificant compared with the effects of the measurement errors, justifying the use of the normal equations. Furthermore, many iterative algorithms avoid explicitly forming and storing the normal equations.

Using the semi-normal equations (1.13) also avoids forming $A^\top A$ and, as R is determined by using a stable QR factorization, it might be expected that this approach would offer a stable alternative to solving the normal equations. However, the forward error (i.e. the norm of the difference between the exact and computed solutions) is bounded by a term involving $\kappa_2(A)^2$ (Björck 1987, Björck and Paige 1994).

For the weighted least-squares problem, if the weighting matrix W is diagonal then the errors in the data are uncorrelated, but they can vary in their accuracy; for example, the most recent observations may be more accurate than older ones. If the ratio of the largest to the smallest weight is large, then W is ill-conditioned and the least-squares problem is said to be stiff (by analogy to the terminology used in the field of differential equations). Stiff problems can arise in many application areas, including in barrier and interior-point methods for optimization and electrical

networks. For stiff problems the condition number $\kappa_2(W^{-1/2}A)$ is large. An upper bound is given by

$$\kappa_2(W^{-1/2}A) \leq \kappa_2(W^{-1/2})\kappa_2(A) = \max_i (W^{-1/2})_{ii} / \min_i (W^{-1/2})_{ii}\, \kappa_2(A).$$

Whilst in many cases it may be possible to solve a weighted problem via the ordinary least-squares problem

$$\min_x \|\hat{b} - \hat{A}x\|_2, \quad \hat{b} = W^{-1/2}b, \quad \hat{A} = W^{-1/2}A,$$

if the problem is stiff then this will not generally be a numerically stable approach.

1.5. Numerical rank

A may have full rank mathematically, but if one or more of its singular values is very small then, for computational purposes, A is rank-deficient. In some applications, such as discrete inverse problems, there can be a large gap in the sequence of the singular values; in this case, A is again said to be rank-deficient (Hansen 2010). Conversely, if $\text{rank}(A) < n$ and some entries of A are perturbed, then the rank may change. The numerical rank of A depends on a tolerance that reflects the level of errors in A. To determine the numerical rank in terms of the singular values of A, we can use the following result of Hansen *et al.* (2013) that presents sensitivity bounds.

Lemma 1.5. Let the singular values of $A \in \mathbb{R}^{m\times n}$ be $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$. Then the singular values $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \ldots \geq \tilde{\sigma}_n$ of the perturbed matrix $A + E$ satisfy

$$|\sigma_i - \tilde{\sigma}_i| \leq \|E\|_2 \quad \text{and} \quad \sum_{i=1}^{n} |\sigma_i - \tilde{\sigma}_i|^2 \leq \|E\|_F^2.$$

Given a tolerance $\epsilon > 0$, the numerical ϵ-rank of A (considered here in the Euclidean norm) is rk_ϵ if

$$rk_\epsilon = \min\{\text{rank}(E) \mid \|A - E\|_2 \leq \epsilon\}.$$

Using the above lemma, it can be shown that A has numerical ϵ-rank rk_ϵ if and only if

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{rk_\epsilon} > \epsilon \geq \sigma_{rk_{\epsilon+1}} \geq \ldots \geq \sigma_n.$$

The tolerance should be chosen to be consistent with the machine precision and the general level of relative errors in the data.

1.6. Scaling least-squares problems

Scaling can be used to improve the conditioning of the problem. Row scaling involves left multiplication of A and b by a diagonal matrix. This is not allowed as it is equivalent to the use of weights and so changes the least-squares objective

function and its solution. Column scaling involves multiplication of A on the right by a diagonal matrix S. That is,

$$\min_{z \in \mathbb{R}^n} \|b - ASz\|_2, \quad x = Sz,$$

This corresponds to a two-sided symmetric scaling $(AS)^\top AS$ of the normal matrix and right-preconditioning (recall (1.14) with $M^{-1} = S$). If A is of full rank, the unique least-squares solution is obtained correctly from the computed z. However, if A is rank-deficient, although the correct set of residual-minimizing vectors is obtained, the least-norm vector is chosen to minimize $\|S^{-1}x\|_2$, not $\|x\|_2$, and therefore the computed solution does not correspond to the correct least-norm solution.

The following result of Van der Sluis (1969) shows how to choose S to reduce $\kappa_2(AS)$.

Lemma 1.6. Let B be an SPD matrix of order n with all its diagonal elements equal. Then

$$\kappa_2(B) \leq n \min\{\kappa_2(S^\top BS) \mid S \text{ is a diagonal matrix}\}.$$

When applied to the normal matrix $A^\top A$, this result states that, in the full-rank case, if S is constructed to make each column of AS of unit Euclidean norm – so that all the diagonal entries of $A^\top A$ are 1 – then $\kappa_2(AS)$ is within a factor $\sqrt{n}$ of the optimum. Observe that because column scaling affects the singular values, a scheme to determine the numerical rank may not return the same estimates when applied to A and AS.

1.7. Regularized linear least-squares problems

When A has singular values that are close to the origin, any error that is present in b will be amplified, sometimes so much as to make the computed solution useless. The idea of regularization is to extract the linearly independent information from the matrix and the noisy right-hand side. One of the most popular methods is Tikhonov regularization. For the unweighted problem, it considers

$$\min_{x \in \mathbb{R}^n} \mathcal{F}(x) = \|b - Ax\|_2^2 + \gamma^2 \|x\|_2^2, \tag{1.17}$$

where the regularization parameter $\gamma > 0$ controls the trade-off between minimizing the residual norm and minimizing the norm of the solution. Problem (1.17) is equivalent to the regularized (or damped) linear least-squares problem

$$\min_{x \in \mathbb{R}^n} \left\| \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} A \\ \gamma I \end{pmatrix} x \right\|_2. \tag{1.18}$$

Observe that even if the original problem is underdetermined, this is an over-determined least-squares problem of size $(m + n) \times n$. Moreover, the regularized

matrix $(A^\top \ \gamma I)^\top$ involves only n additional entries compared to the original A and, as it is of full rank, (1.18) has a unique solution. If $\gamma > \sigma_{\min}(A)$, the condition number of the corresponding normal matrix satisfies

$$\kappa_2(A^\top A + \gamma^2 I) \approx (\|A\|_2/\gamma)^2 \tag{1.19}$$

(Saunders 1995). An important issue is how to choose γ: too little regularization ('small' γ) can lead to the solution method having numerical difficulties or not being able to find a useful solution because of the level of noise in the data, while for excessive regularization ('large' γ), the computed objective value may be unacceptably different from the optimum for the original problem. γ is not known *a priori* and has to be selected based on the problem data. Finding a good regularization parameter can be difficult, especially for large-scale problems (Hansen 1998).

The regularized least-norm problem is

$$\min_{x,y} \left\| \begin{pmatrix} x \\ y \end{pmatrix} \right\|_2 \quad \text{subject to} \quad (A \ \ \gamma I) \begin{pmatrix} x \\ y \end{pmatrix} = b. \tag{1.20}$$

The linear system $Ax + \gamma y = b$ is consistent for all $\gamma > 0$ with solution $x = A^\top z$, $y = \gamma z$, where z is the solution of the normal equations of the second kind

$$(A \ \ \gamma I) \begin{pmatrix} A^\top \\ \gamma I \end{pmatrix} z = (AA^\top + \gamma^2 I)z = b.$$

From (1.20), $\gamma y = b - Ax = r$. Using this to eliminate y, it follows that for A of arbitrary dimensions, both (1.17) and (1.20) are equivalent to the regularized augmented system

$$\begin{pmatrix} I & A \\ A^\top & -\gamma^2 I \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix},$$

or, equivalently,

$$K_\gamma \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \quad K_\gamma = \begin{pmatrix} \gamma I & A \\ A^\top & -\gamma I \end{pmatrix}, \quad r = \gamma s. \tag{1.21}$$

For $\gamma > \sigma_{\min}(A)$, the condition number satisfies $\kappa_2(K_\gamma) \approx \|A\|_2/\gamma$, which implies that solving (1.21) is a feasible approach provided γ is not very small. Note also that small γ could mean $\|s\|_2 > \|x\|_2$ and may not imply good accuracy in x (Saunders 1995, Hansen 1998).

In some applications, prior information regarding the smoothness of the solution may be known, and instead of using $\|x\|_2$ as the penalty term it may be preferable to use $\|Fx\|_2$, where F is a $p \times n$ matrix. If $\operatorname{rank}(F) = n$ then $\|Fx\|_2$ is a norm, otherwise F has a non-trivial null space and $\|Fx\|_2$ is a semi-norm; in the latter case, the Tikhonov solution is unique if $\mathcal{N}(A) \cap \mathcal{N}(F) = \emptyset$. Possible choices for F

include finite-difference approximations to the first or second derivatives of x. The regularized weighted least-squares problem becomes

$$\min_{x\in\mathbb{R}^n} \left\| \begin{pmatrix} W^{-1}b \\ 0 \end{pmatrix} - \begin{pmatrix} W^{-1}A \\ \gamma F \end{pmatrix} x \right\|_2. \tag{1.22}$$

The change of variables $\bar{A} = AF^{-1}$, $\bar{x} = Fx$, transforms (1.22) to the standard form

$$\min_{x\in\mathbb{R}^n} \|W^{-1}(b - \bar{A}\bar{x})\|_2^2 + \gamma^2\|\bar{x}\|_2^2. \tag{1.23}$$

In Bayesian statistics, this transformation is sometimes referred to as priorconditioning (see e.g. Calvetti and Somersalo 2005).

The regularized augmented system is

$$K_W \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \quad \text{with} \quad K_W = \begin{pmatrix} W & A \\ A^\top & -\gamma^2 F^\top F \end{pmatrix}. \tag{1.24}$$

Provided W and $F^\top F$ are SPD matrices, the system matrix K_W is a symmetric quasi-definite (SQD) matrix. It has m positive and n negative eigenvalues, is non-singular and its inverse is also SQD (Vanderbei 1995). An SQD matrix is strongly factorizable, that is, for any permutation matrix P, there exists a non-singular diagonal matrix D with positive and negative entries on the diagonal and a unit lower triangular matrix L such that

$$PK_WP^\top = LDL^\top.$$

This is known as a signed Cholesky factorization. Note that although this factorization always exists, numerical stability is not guaranteed for all P. The connections between various symmetric indefinite and SQD linear systems and the solution of linear least-squares problems are given in Orban and Arioli (2017). The signed Cholesky factorization is computationally attractive, but stability analysis shows that the relative error in the solution is bounded by an expression in which the standard condition number is replaced by the effective condition number for K_W, which is not smaller than the standard condition number. But the Cholesky-based approach may still be preferable if the factors are sufficiently sparse and the original problem is well-scaled (Gill, Saunders and Shinnerl 1996, Saunders 1996).

1.8. The nonlinear least-squares problem

We end this section by briefly discussing the nonlinear least-squares problem

$$\min_{x\in\mathbb{R}^n} \mathcal{F}(x) = \|r(x)\|_2^2.$$

Here $r(x) = (r_1(x), r_2(x), \ldots, r_m(x))^\top$ is a smooth vector of nonlinear residual functions. A popular approach is the Gauss–Newton method. This iterative method avoids the use of the Hessian matrix that is required by the standard Newton's method. It is based on the first-order approximation of the residual function in a

neighbourhood of the current iterate $x^{(j)}$, that is,

$$r(x) \approx r(x^{(j)}) + \mathcal{J}(x^{(j)})(x - x^{(j)}),$$

where $\mathcal{J}(x^{(j)})$ denotes the Jacobian of $r(x)$ at $x^{(j)}$. A necessary condition for x to be a local minimum of $r(x)$ is

$$\mathcal{J}(x^{(j)})^\top (r(x^{(j)}) + \mathcal{J}(x^{(j)})(x - x^{(j)})) = 0.$$

The next iterate is taken to be $x^{(j+1)} = x^{(j)} + s^{(j)}$, where $s^{(j)}$ is the solution of the linear least-squares problem

$$\min_{s^{(j)} \in \mathbb{R}^n} \|r(x^{(j)}) + \mathcal{J}(x^{(j)}) s^{(j)}\|_2. \tag{1.25}$$

Here $s^{(j)}$ can be determined by solving the normal equations, but any approach for solving a sequence of linear least-squares problems can be used. More generally, the regularized weighted nonlinear problem is

$$\min_{x \in \mathbb{R}^n} \mathcal{F}(x) = \|r(x)\|^2_{W^{-1}} + \gamma^2 \|x\|^2_2, \quad \gamma > 0,$$

where W is an SPD matrix. The linear subproblem to be solved is then

$$\min_{s^{(j)} \in \mathbb{R}^n} F^{(j)}(s^{(j)}) = \|r(x^{(j)}) + \mathcal{J}(x^{(j)}) s^{(j)}\|^2_{W^{-1}} + \gamma^2 \|s^{(j)} + x^{(j)}\|^2_2,$$

and the equivalent normal equations are

$$(\mathcal{J}(x^{(j)})^\top W^{-1} \mathcal{J}(x^{(j)}) + \gamma^2 I) s^{(j)} = -\mathcal{J}(x^{(j)})^\top W^{-1} r(x^{(j)}). \tag{1.26}$$

Sufficient conditions for the convergence of the Gauss–Newton method are known in the case where the normal equations for the linearized least-squares problem are solved exactly at each iteration. In large-scale applications, this may be impractical. A common approach is to solve the linearized problem approximately using an 'inner' iteration method that is truncated before full accuracy is reached. This is known as the truncated Gauss–Newton method. A discussion is given in Gratton, Lawless and Nichols (2007).

2. Sparse matrices, their graphs and ordering algorithms

For any symmetric positive definite matrix C, there exists a unique lower triangular matrix L with positive diagonal entries called the Cholesky factor such that

$$C = LL^\top. \tag{2.1}$$

The Cholesky factorization was invented by André-Louis Cholesky for least-squares problems arising in geodesy, sometime before 1914, while he was working for the French Geographic Service; it was posthumously published by Benoit (1924). The Cholesky factorization is based on the systematic annihilation of the entries in the lower triangular part of C column-by-column; the order in which this

is done is the elimination order and it determines the symmetric permutation of C. Algorithms for computing sparse Cholesky factorizations start with a symbolic factorization (or analyse phase) that uses the sparsity pattern $\mathcal{S}\{C\}$ to determine the non-zero structure of L and other important properties (such as the number of entries in each row and column of L) without computing the numerical values of the non-zeros. This is then used to set up data structures for the subsequent numerical factorization. Tools from graph theory are a key ingredient of the symbolic factorization.

2.1. *Introduction to undirected graphs*

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a finite set $\mathcal{V}$ of vertices (or nodes), and a set $\mathcal{E}$ of edges defined as pairs of distinct vertices. When there is no distinction between the pairs of vertices (u, v) and (v, u), the edges are represented by unordered pairs and the graph is undirected. In this section, we assume $\mathcal{G}$ is undirected. A labelling (or ordering) of a graph $\mathcal{G}$ with n vertices is a bijection of $\{1, 2, \ldots, n\}$ onto $\mathcal{V}$. The integer i $(1 \leq i \leq n)$ assigned to a vertex in $\mathcal{V}$ is called the label (or simply the number) of that vertex. The standard choice of vertices is $\mathcal{V} = \{1, \ldots, n\}$ so that the vertices are directly identified by their labels.

Vertices i and j in $\mathcal{V}$ are adjacent (or neighbours) if $e = (i, j) \in \mathcal{E}$. The notation $(i \longleftrightarrow j)$ is also used for an edge (or $(i \overset{\mathcal{G}}{\longleftrightarrow} v)$ to emphasize that the edge belongs to the graph $\mathcal{G}$). The degree $\deg(i)$ of $i \in \mathcal{V}$ is the number of $j \in \mathcal{V}$ that are adjacent to i, and the adjacency set $\mathrm{adj}\{i\}$ is the set of adjacent vertices (thus $|\,\mathrm{adj}\{i\}| = \deg(i)$). $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$ is a subgraph of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ if and only if $\mathcal{V}_s \subseteq \mathcal{V}$ and $\mathcal{E}_s \subseteq \mathcal{E}$, and $(u_s, v_s) \in \mathcal{E}_s$ implies $u_s, v_s \in \mathcal{V}_s$. A subgraph is a clique when every pair of vertices is adjacent.

Adjacency graphs provide a link between sparse matrices and graphs. Let $C = A^\top A$ be the normal matrix of order n; then an adjacency graph $\mathcal{G}(C) = (\mathcal{V}(C), \mathcal{E}(C))$ with vertices $\mathcal{V}(C) = \{1, \ldots, n\}$ can be associated with it. The edge set is

$$\mathcal{E}(C) = \{(i, j) \mid (A^\top A)_{ij} \neq 0,\ i \neq j\}.$$

When using graphs to capture sparsity structures of matrices, it is standard to assume that the result of adding, subtracting or multiplying two non-zero entries is non-zero. This is the non-cancellation assumption.

The normal matrix can be expressed as a sum of m matrices of rank one, that is,

$$C = A^\top A = \sum_{i=1}^{m} a_i a_i^\top,$$

where $a_i^\top$ is row i of A. By the non-cancellation assumption, $\mathcal{S}(C)$ is the binary sum of the sparsity patterns $\mathcal{S}(a_i a_i^\top)$, $i = 1, \ldots, m$. This is unchanged by discarding any row of A whose sparsity structure is a subset of that of another row and hence $\mathcal{G}(A^\top A)$ is the sum of the m graphs $\mathcal{G}(a_i a_i^\top)$, $i = 1, \ldots, m$. Observe that

each $\mathcal{G}(a_i a_i^\top)$ is a clique, and an alternative characterization of $\mathcal{G}(A^\top A)$ is that $(A^\top A)_{jk} \neq 0$ if and only if the sparsity patterns of columns j and k have an entry in common, that is, if $a_{ij} \neq 0$ and $a_{ik} \neq 0$, for some i ($1 \leq i \leq m$). Because of this, $\mathcal{G}(A^\top A)$ is also termed the column intersection graph of A.

A sequence of k edges in the graph $\mathcal{G}$ given by $i_0 \longleftrightarrow i_1 \longleftrightarrow \cdots \longleftrightarrow i_{k-1} \longleftrightarrow i_k$ is called a walk of length k. The walk is closed if $i_0 = i_k$; a closed walk is called a cycle. Graphs that do not contain cycles are acyclic. A path is a walk in which all the vertices (and therefore also all the edges) are distinct. $\mathcal{G}$ is connected if every pair of (distinct) vertices is connected by a path. If $\mathcal{G}$ is connected, then $\mathcal{U} \subset \mathcal{V}$ is a separator if $\mathcal{G}$ becomes disconnected when the vertices $\mathcal{U}$ are removed.

2.2. *Elimination graphs*

Graphs can be used to symbolically compute the Cholesky factors of a symmetric positive definite matrix C. Assume the vertices of $\mathcal{G}(C)$ are labelled v_i, $1 \leq v_i \leq n$, and the elimination order is $v_1, v_2, \ldots, v_n$. Then, starting with $\mathcal{G}^1 \equiv \mathcal{G}(C)$, a sequence of graphs is generated recursively using Parter's rule (Parter 1961):

> To obtain the elimination graph $\mathcal{G}^{k+1}$ from $\mathcal{G}^k$, delete vertex v_k and all its incident edges, and add all possible edges between vertices that are adjacent to vertex v_k in $\mathcal{G}^k$.

In terms of graph theory, Parter's rule says that when v_k is eliminated, $\mathrm{adj}(v_k)$ becomes a clique of size equal to the number of off-diagonal entries in the matrix corresponding to $\mathcal{G}^k$. Thus the symbolic interpretation of Gaussian elimination is that it systematically generates cliques. As the elimination process progresses, cliques grow or more than one clique join together to form larger cliques, a process known as clique amalgamation.

An implementation difficulty is that because edges are added with each elimination, the space required to represent the elimination graph $\mathcal{G}^{k+1}$ is potentially greater than for $\mathcal{G}^k$. For large matrices, creating and explicitly storing the edges in the sequence of elimination graphs is impractical, and a more compact and efficient representation is needed. In place of standard elimination graphs, special quotient graphs are used that do not delete the vertices implied by Parter's rule. Rather, the eliminated vertices are kept and are distinguished from the remaining vertices by assigning them a special flag. Each quotient graph in such a representation can be interpreted as a collection of cliques, including the original graph $\mathcal{G}$, which can be regarded as having $|\mathcal{E}|$ cliques, each with two vertices (or, equivalently, one edge). Let $\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_q\}$ be the set of cliques for $\mathcal{G}^k$ and $\{\mathcal{V}_{s_1}, \mathcal{V}_{s_2}, \ldots, \mathcal{V}_{s_t}\}$ be the subset of cliques to which the vertex v_k to be eliminated belongs. Two steps are then required.

(1) Remove the cliques $\{\mathcal{V}_{s_1}, \mathcal{V}_{s_2}, \ldots, \mathcal{V}_{s_t}\}$ from $\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_q\}$.

(2) Add the new clique $\mathcal{V}_v = \{\mathcal{V}_{s_1} \cup \ldots \cup \mathcal{V}_{s_t}\} \setminus \{v_k\}$ into the set of cliques.

Hence the degree of the eliminated vertex v_k in the quotient graph satisfies

$$\deg(v_k) = |\mathcal{V}_{v_k}| < \sum_{i=1}^{t} |\mathcal{V}_{s_i}|,$$

and because $\{\mathcal{V}_{s_1}, \mathcal{V}_{s_2}, \ldots, \mathcal{V}_{s_t}\}$ can now be discarded, the storage required for the representation of the sequence of these quotient graphs never exceeds that needed for $\mathcal{G}(C)$. The index of the eliminated vertex can be used as the index of the new clique. This is called an element or e-node, to distinguish it from an uneliminated vertex, which is termed an s-node.

While Parter's rule describes how fill-in in L develops locally, fill-in can also be described more globally using fill-paths. Assume there is a path between distinct vertices i and j in the undirected graph $\mathcal{G}$. If all intermediate vertices on the path are less than $\min\{i, j\}$, then the path is called a fill-path. It is well known that an off-diagonal entry l_{ij} $(i > j)$ in the Cholesky L of C is non-zero if and only if there is a fill-path in $\mathcal{G}(C)$ between i and j (Rose, Tarjan and Lueker 1976).

2.3. *Sparse matrix orderings*

The number of operations needed to perform a sparse Cholesky factorization is $O\left(\sum_{j=1}^{n} |L_{:,j}|^2\right)$, where $|L_{:,j}|$ is the number of entries in column j of L. Thus fill-in in L can render a direct method infeasible and so the symbolic factorization of a direct solver typically starts by finding a symmetric permutation (ordering) of the rows and columns of C to limit fill-in.

The problem of minimizing the fill-in in L is NP-complete. Instead, heuristics are used, with no one approach resulting in the best ordering for every problem. Note that the ordering of the rows of A has no effect on the normal matrix or its Cholesky factor and, because L is equal to the factor R in the QR factorization of A, the same column ordering can be used for the Cholesky and the QR factorizations.

P is a permutation matrix if it is a square matrix with exactly one entry equal to unity in each row and column, and all remaining entries are zeros (i.e. it is a permutation of the identity matrix). Premultiplying a matrix by a permutation P_r reorders the rows and postmultiplying by a permutation P_c reorders the columns. If the matrix is symmetric, setting $P_c^\top = P_r = P$ preserves symmetry and the graph of the symmetrically permuted matrix is unchanged; only the labelling (i.e. the ordering) of the vertices changes. Thus, for the normal matrix, we can either relabel $\mathcal{G}(A^\top A)$ or permute the columns of A.

Two main classes of methods are commonly used to compute orderings that limit factor fill-in.

- Local orderings use a greedy approach in an attempt to limit fill-in by repeated local decisions based on $\mathcal{G}(A^\top A)$ or $\mathcal{G}(A)$.
- Global orderings consider the whole of $\mathcal{S}\{A^\top A\}$ and seek to find a P using a divide-and-conquer approach, often in conjunction with a local ordering, as the latter generally works well for matrices that are not really large.

In the following, it is convenient to assume that $\mathcal{G}(A^\top A)$ has a single component, that is, $A^\top A$ is irreducible. Otherwise, the ordering algorithms are applied to each component of $\mathcal{G}(A^\top A)$ (in particular, any rows/columns with a single entry are removed and ordered first). We also assume here that A has no rows that are (almost) dense. If it does, a simple strategy is to remove such rows before applying the ordering algorithm to the remaining rows. Afterwards, the variables corresponding to the dense rows can be appended to the end of the computed ordering to give the final ordering.

2.4. Minimum fill-in and minimum degree

One way to reduce fill-in is to use a local minimum fill-in criterion that, at each step, selects as the next variable in the ordering one that will introduce the least fill-in in the factor at that step (Markowitz 1957); see also Rothberg and Eisenstat (1998) and Reißig (2007). This can produce good orderings, but the cost is often considered prohibitive because the updated sparsity pattern and the fill-in associated with the possible candidates must be determined.

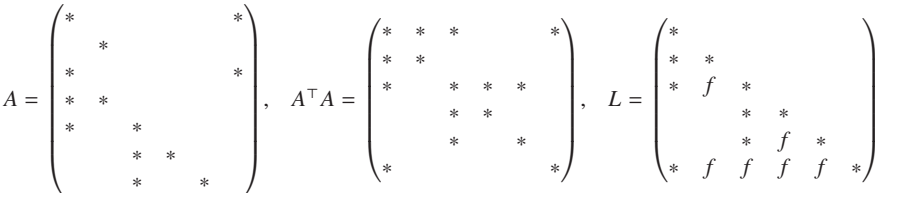
The minimum degree (MD) and approximate minimum degree (AMD) algorithms are the best-known and most widely used greedy heuristics for limiting fill-in. The minimum degree approach seeks to find a permutation such that at each step of the factorization the number of entries in the corresponding column of the factor is minimized. It does this by selecting a vertex of minimum degree in the current elimination graph. The approach is derived from a method proposed in 1957 (Markowitz 1957) for non-symmetric linear systems; a symmetric variant was published 10 years later (Tinney and Walker 1967). A graph-theoretic version in which the factorization is only simulated was subsequently presented; it was termed the minimum degree algorithm (Rose 1972). Efficient implementations employ special quotient graphs and cliques.

The number of updates to the vertex degrees can be decreased using indistinguishable vertices. Mutually adjacent vertices $\mathcal{V}_r = \{v_1, \ldots, v_r\}$ are termed indistinguishable if they have the same neighbours, that is,

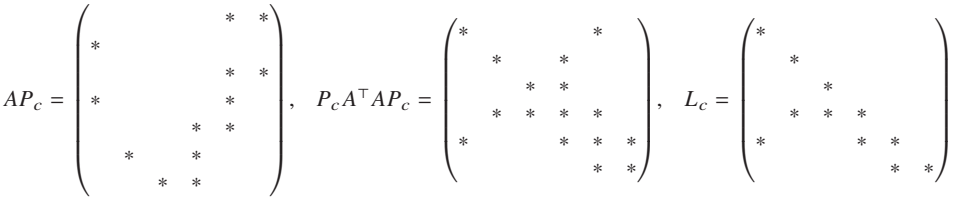
$$\mathrm{adj}(v_i) \cup v_i = \mathrm{adj}(v_j) \cup v_j, \quad 1 \le i, j \le r.$$

The set $\mathcal{V}_r$ can be represented by a single vertex, called a supervariable. If $v \in \mathcal{V}_r$ is eliminated then the degree of each remaining vertex in $\mathcal{V}_r$ will reduce by one and they are all of minimum degree. Thus the vertices in $\mathcal{V}_r$ can be eliminated together and the graph transformation and vertex update only needs to be performed once.

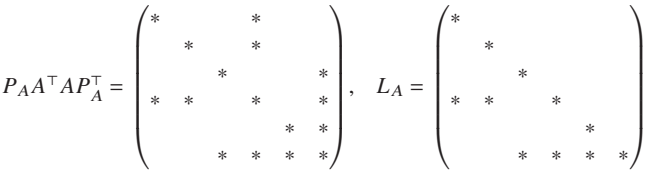
The idea behind the popular AMD variant is to inexpensively compute an upper bound on a vertex degree in place of the degree, and to use this when selecting vertices within the MD algorithm (Amestoy, Davis and Duff 1996*a*). Even though vertex degrees are not determined exactly, the quality of the resulting ordering is competitive with using the MD algorithm and, on some problems, it can be better. Importantly, the AMD run-time is typically significantly less than that of the MD algorithm.



(a)



(b)



(c)

Figure 2.1. An example to illustrate the COLAMD and AMD orderings. (a) The pattern of the original matrix A, the normal matrix and its factor (entries in L that have filled in are denoted by f). (b) The column permuted matrix AP_c, the normal matrix $P_cA^\top AP_c$ and its factor, where P_c corresponds to the COLAMD ordering. (c) The symmetrically permuted normal matrix $P_AA^\top AP_A$ and its factor, where P_A corresponds to the AMD ordering.

2.5. *Column version of minimum degree*

Applying the minimum degree approach to the normal matrix $A^\top A$ requires the sparsity pattern $\mathcal{S}(A^\top A)$. Even if A is sparse, $\mathcal{S}(A^\top A)$ may contain significantly more entries than $\mathcal{S}(A)$. Thus, to potentially save time and storage, an attractive alternative is to order the columns of A using only $\mathcal{G}(A)$. As observed in Section 2.1, the structure of each row of A corresponds to a clique in $\mathcal{G}(A^\top A)$, and $A^\top A$ can be represented as a sequence of cliques. This allows the minimum degree algorithm for $A^\top A$ to be implemented in a way that bypasses forming $\mathcal{S}(A^\top A)$, leading to savings in work and storage.

This approach is known as COLAMD (Davis, Gilbert, Larimore and Ng 2004*a*). The COLAMD and AMD orderings are illustrated in Figure 2.1. The COLAMD ordering of A is 2, 4, 5, 3, 1, 6 and the AMD ordering of the normal matrix $A^\top A$ is

Algorithm 2.1. RCM algorithm for band and profile reduction
Input: Graph $\mathcal{G}$ of a symmetrically structured matrix and a starting vertex s.
Output: Permutation vectors p_{cm} and p_{rcm} that define new labellings of the vertices of $\mathcal{G}$.

1: label$(1:n)$ = false
2: Compute adj$\{u\}$ and deg(u) for all $u \in \mathcal{V}(\mathcal{G})$
3: $k = 1,\ v_1 = s,\ p_{cm}(1) = v_1,$ label(v_1) = true
4: **for** $i = 1 : n - 1$ **do**
5: **for** $w \in$ adj$\{v_i\}$ with label(w) = false in order of increasing degree **do**
6: $k = k + 1,\ v_k = w,\ p_{cm}(k) = v_k,$ label(v_k) = true
7: **end for**
8: **end for**
9: Set $p_{rcm}(i) = p_{cm}(n - i + 1), i = 1, 2, \ldots, n.$

5, 4, 6, 3, 2, 1. In this example, there is no fill-in in the Cholesky factor following the COLAMD and the AMD reorderings (although the sparsity patterns are different). In general this is not the case, but it is not possible to determine *a priori* which approach will result in the sparsest factor.

2.6. Profile-reducing orderings

An alternative way of limiting the fill-in locally is to add another criterion to the relabelling of the vertices, such as restricting the non-zeros of the permuted matrix to specific positions. The most popular approach is to force them to lie close to the main diagonal, that is, to reduce the profile or envelope of C. The envelope is the set of index pairs that lie between the first non-zero entry in each row and the diagonal, that is,

$$\text{env}(C) = \{(i, j) \mid 0 < i - j \leq \beta_i(C)\},$$

where $\beta_i(C) = i - \min\{j \mid 1 \leq j \leq i \text{ with } c_{ij} \neq 0\}$, $1 \leq i \leq n$. The profile of C is $|\text{env}(C)|$ (the envelope size) plus n.[2] During a Cholesky factorization, all fill-in takes place between the first non-zero entry in a row and the diagonal so that $\text{env}(C) = \text{env}(L)$.

The most well-known method for reducing the profile of a symmetrically structured matrix is the reverse Cuthill–McKee (RCM) algorithm (Cuthill and McKee 1969), outlined here as Algorithm 2.1. Note that line 9 reverses the ordering. This can reduce (but not increase) the profile of the permuted matrix. The quality of the ordering is highly dependent on the choice of the starting vertex s. The diameter of $\mathcal{G}$ is the maximum distance between any pair of its vertices. The endpoints of a diameter provide good starting vertices. In practice, finding a diameter is expensive

[2] Sometimes in the literature the profile is defined to be the envelope size.

and an endpoint of a pseudo-diameter is used (a pseudo-diameter is defined by a pair of vertices whose distance apart in $\mathcal{G}$ is close to the diameter). The GPS approach for finding such vertices is based on constructing level sets (Gibbs, Poole and Stockmeyer 1976), where for a given vertex r (the root), vertices within the same level set have the same distance to r. Modifications to the original algorithm have been made to improve efficiency but the use of level sets remains key.

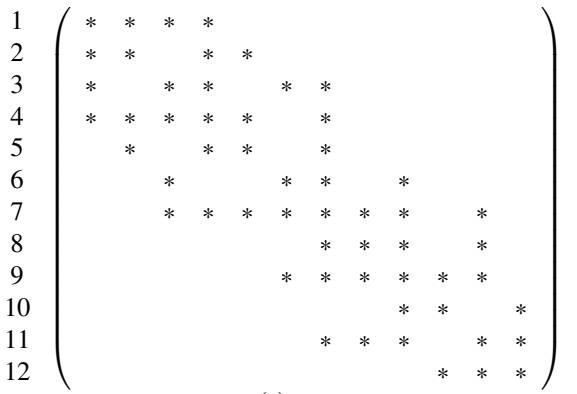
Over the years, a large number of profile reduction algorithms have been proposed, many of which have their origins in the Cuthill–McKee and GPS algorithms. A widely used two-stage variant is the Sloan algorithm (Sloan 1986); see Reid and Scott (1999) for details of an efficient implementation. Spectral methods offer an alternative approach (Barnard, Pothen and Simon 1995). A crucial difference between profile reduction ordering algorithms and minimum degree strategies is that the former is based solely on $\mathcal{G}$: the costly construction of quotient graphs is not needed. However, unless the profile after reordering is very small, there can be significantly more fill-in in the factor.

2.7. *Nested dissection*

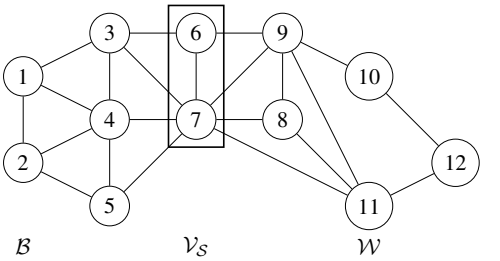
Nested dissection, which was first introduced in the early 1970s (George 1973), is a global ordering strategy for matrices with a symmetric sparsity pattern; it is particularly effective for very large sparse problems (and problems with an underlying grid structure but this is not the case for least-squares problems). Subsequent research gave theoretical guarantees for linear solvers (Lipton, Rose and Tarjan 1979) and provided an important framework for graph partitioning, hierarchical solvers and many related tasks; see, for example, Spielman and Teng (2014), Cambier *et al.* (2020) and the survey of graph partitioning by Bichot and Siarry (2011). Given a symmetric matrix C, nested dissection works with the adjacency graph $\mathcal{G}$ and proceeds by identifying a small separator $\mathcal{V}_\mathcal{S}$ that if removed separates the graph into two disjoint subgraphs described by the vertex subsets $\mathcal{B}$ and $\mathcal{W}$ (commonly called 'black' and 'white', respectively). The rows and columns belonging to $\mathcal{B}$ are labelled first, then those belonging to $\mathcal{W}$ and finally those in $\mathcal{V}_\mathcal{S}$. The reordered matrix has the form

$$\begin{pmatrix} C_{\mathcal{B},\mathcal{B}} & 0 & C_{\mathcal{B},\mathcal{V}_\mathcal{S}} \\ 0 & C_{\mathcal{W},\mathcal{W}} & C_{\mathcal{W},\mathcal{V}_\mathcal{S}} \\ C_{\mathcal{B},\mathcal{V}_\mathcal{S}}^\top & C_{\mathcal{W},\mathcal{V}_\mathcal{S}}^\top & C_{\mathcal{V}_\mathcal{S},\mathcal{V}_\mathcal{S}} \end{pmatrix}.$$

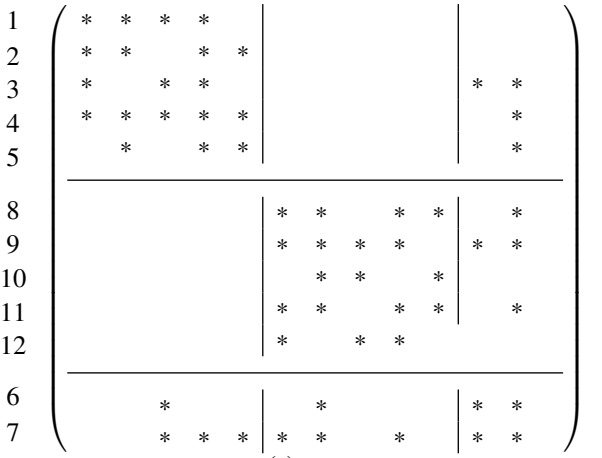
This is illustrated in Figure 2.2. Provided the variables are eliminated in the permuted order, no fill occurs within the zero off-diagonal blocks. If $|\mathcal{V}_\mathcal{S}|$ is small and $|\mathcal{B}| \approx |\mathcal{W}|$, these zero blocks account for approximately half the possible entries in the matrix. The reordering can be applied recursively to the submatrices $C_{\mathcal{B},\mathcal{B}}$ and $C_{\mathcal{W},\mathcal{W}}$ until the vertex subsets are of size less than a chosen threshold. At this stage, a local ordering technique (such as AMD) is normally more effective than nested dissection, and so a switch is made (Liu 1989); see also a more general combination that involves hypergraphs in Çatalyürek, Aykanat and Kayaaslan (2011). The



(a)



(b)



(c)

Figure 2.2. A simple example to illustrate nested dissection. (a) The pattern of the original matrix. (b) The partitioned graph. (c) The corresponding symmetrically permuted matrix.

Algorithm 2.2. Nested dissection algorithm
Input: Graph $\mathcal{G}$ of a symmetric matrix C and a partitioning algorithm PartitionAlg.
Output: A permutation vector p that defines a new labelling of the vertices of $\mathcal{G}$.

recursive function (p = nested_dissection(C, PartitionAlg))
  if dissection has terminated **then** ▷ Vertex subsets are sufficiently small
    p = AMD($\mathcal{V}, \mathcal{E}$) ▷ Compute an AMD ordering
  else
    Use PartitionAlg($\mathcal{V}, \mathcal{E}$) to obtain the vertex partitioning ($\mathcal{B}, \mathcal{W}, \mathcal{V}_S$)
    $p_{\mathcal{B}}$ = nested_dissection($C_{\mathcal{B},\mathcal{B}}$, PartitionAlg)
    $p_{\mathcal{W}}$ = nested_dissection($C_{\mathcal{W},\mathcal{W}}$, PartitionAlg)
    $p_{\mathcal{V}_S}$ is an ordering of $\mathcal{V}_S$
    Set $p = \begin{pmatrix} p_{\mathcal{B}} \\ p_{\mathcal{W}} \\ p_{\mathcal{V}_S} \end{pmatrix}$
  end if
end recursive function

approach is summarized in Algorithm 2.2. Here PartitionAlg specifies the algorithm used in determining the partitioning of the vertices. Most current approaches use multilevel techniques that create a hierarchy of graphs, each representing the original graph, but with a successively smaller dimension (Karypis and Kumar 1998, Davis, Hager, Kolodziej and Yeralan 2020). The smallest (i.e. the coarsest) graph is partitioned and this is then propagated back through the sequence of graphs, while being periodically refined.

2.8. *Software for ordering sparse matrices*

Sparse direct solvers typically offer users a number of ordering algorithms. These may call external packages or be built into the solver. COLAMD and AMD are published as Algorithms 836 and 837, respectively, in *ACM Transactions on Mathematical Software* (Amestoy, Davis and Duff 2004, Davis, Gilbert, Larimore and Ng 2004*b*). The package `HSL_MC68` from the HSL Mathematical Software Library[3] offers implementations of MD and AMD algorithms. `MC61` is an efficient implementation of the Sloan profile reduction algorithm and `HSL_MC73` has entries to compute multilevel variants of Sloan's algorithm and spectral ordering. SuiteSparse[4] offers a number of sparse matrix ordering algorithms, including COLAMD and AMD; these also appear in MATLAB as functions `colamd` and `amd`. A Julia

[3] https://www.hsl.rl.ac.uk/
[4] https://people.engr.tamu.edu/davis/suitesparse.html

version of AMD is available.[5] The most well-known and widely used nested dissection ordering package is METIS;[6] ParMETIS is an MPI-based parallel version.[7] The Scotch library[8] offers variants of AMD ordering, approximate minimum fill-in ordering and nested dissection.

3. Sparse Cholesky factorizations

It is convenient to introduce the following matrix notation. For any square matrix $B = \{b_{ij}\}$, let $B_{i:j,k:l}$ be the submatrix comprising rows i to j, columns k to l, and let $B_{i:j,k}$ denote the submatrix comprising rows i to j, column k.

We again assume that the matrix $C = \{c_{ij}\} \in \mathbb{R}^{n\times n}$ is sparse and symmetric positive definite (SPD) and irreducible. If C is not reducible, it can be permuted to a non-trivial block diagonal form and the Cholesky factorization of each block on the diagonal computed independently.

3.1. Column replication in sparse Cholesky factorizations

The Cholesky factorization of C can be written as a recursive sequence of Schur complements. The first step is

$$C = \begin{pmatrix} c_{11} & C_{2:n,1}^\top \\ C_{2:n,1} & C_{2:n,2:n} \end{pmatrix} = \begin{pmatrix} c_{11}^{1/2} & \\ C_{2:n,1}c_{11}^{-1/2} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & S^{(2)} \end{pmatrix} \begin{pmatrix} c_{11}^{1/2} & c_{11}^{-1/2}C_{2:n,1}^\top \\ & I \end{pmatrix}, \tag{3.1}$$

where the $(n-1)\times(n-1)$ submatrix

$$S^{(2)} = C_{2:n,2:n} - C_{2:n,1}c_{11}^{-1}C_{2:n,1}^\top = C_{2:n,2:n} - L_{2:n,1}L_{2:n,1}^\top$$

is the Schur complement of C with respect to c_{11}. If C is SPD then so too is $S^{(2)}$, allowing the process to be repeated to give

$$S^{(1)} = C, \quad S^{(j+1)} = S^{(j)}_{2:n-j+1,2:n-j+1} - L_{j+1:n,j}L_{j+1:n,j}^\top, \quad j = 1, 2, \ldots, n-1,$$

where $S^{(j+1)}$ is of order $n-j$. Equivalently, the Schur complement can be written using an outer product as

$$S^{(j+1)} = C_{j+1:n,j+1:n} - \sum_{k=1}^{j} \begin{pmatrix} l_{j+1,k} \\ \vdots \\ l_{nk} \end{pmatrix} \begin{pmatrix} l_{j+1,k} & \ldots & l_{nk} \end{pmatrix}. \tag{3.2}$$

In the sparse case, many entries of L are zero. Let the first non-zero subdiagonal entry in column k of L be in row $i > k$. Then the first column of the Schur

[5] https://github.com/JuliaSmoothOptimizers/AMD.jl
[6] https://github.com/KarypisLab/METIS
[7] https://github.com/KarypisLab/ParMETIS
[8] https://gitlab.inria.fr/scotch/

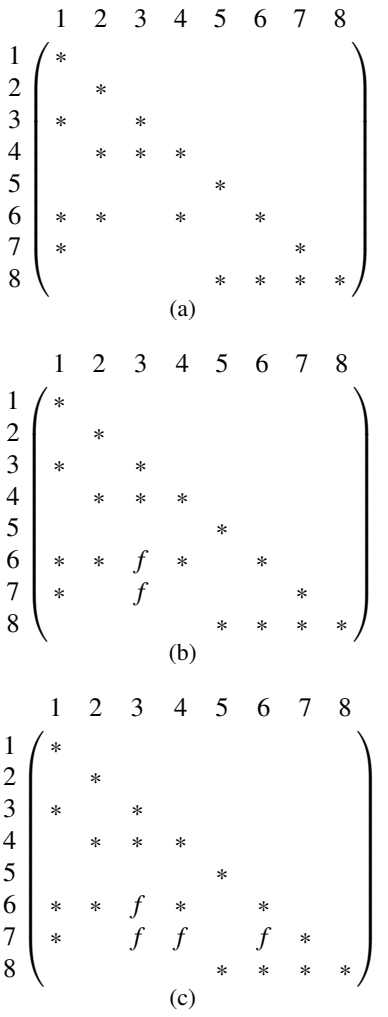


Figure 3.1. An illustration of column replication. (a) The entries in L before the start of the Cholesky factorization, i.e. the entries in the lower triangular part of C. (b) The replication of the non-zeros from column 1 in the pattern of column 3 (entries that have filled in are denoted by f). (c) The situation after the two remaining replications in the pattern of columns 4 and 6.

complement to be updated by column k is i. Determining the sparsity structure $\mathcal{S}(L)$ can therefore be described as the recursive replication of non-zeros in the columns of L with a key role played by the leading subdiagonal entries. This is illustrated in Figure 3.1. The replication of the pattern of column k of L (rows i to n) in the pattern of column $i > k$ of L is called the column replication principle,

that is, for any $i > k \geq 1$ such that $l_{ik} \neq 0$ the sparsity patterns of columns i and k of L satisfy $\mathcal{S}\{L_{i:n,k}\} \subseteq \mathcal{S}\{L_{i:n,i}\}$.

3.2. Elimination trees

Undirected graphs were discussed in Section 2.1. We now introduce trees, directed graphs and DAGs, which we use to discuss the elimination trees that are key to developing fast memory-efficient algorithms for the symbolic phase of sparse Cholesky factorizations.

An undirected graph is connected if every pair of vertices is connected by a path. A connected acyclic graph is called a tree, that is, a tree is an undirected graph in which any two vertices are connected by exactly one path. Every tree has at least two vertices of degree 1. Such vertices are leaf vertices. Leaf vertices have no children.

In a directed graph (or digraph) $\mathcal{G}$, the pairs of vertices that define the edges are ordered. The notation $(u \rightarrow v)$ indicates the direction of the edge from u to v. Any undirected tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ can be converted to a directed rooted tree $\mathcal{T}' = (\mathcal{V}, \mathcal{E}')$ by specifying a root vertex v_r. Note that v_r can be chosen arbitrarily: any choice gives a directed rooted tree. An edge $(u, v) \in \mathcal{E}$ becomes a directed edge $(u \rightarrow v) \in \mathcal{E}'$ if there is a path from u to v_r such that the first edge of this path is from u to v. Given v_r, this directed path is unique. v is called the parent of u if the directed edge $(u \rightarrow v) \in \mathcal{E}'$; u is said to be a child of v. A rooted tree is a special case of a directed acyclic graph (DAG). A topological ordering of $\mathcal{G}$ is a labelling of its vertices such that for every edge $(i \rightarrow j)$, vertex i precedes vertex j (i.e. $i < j$). It can be shown that a topological ordering is possible if and only if $\mathcal{G}$ is a DAG.

Provided the matrix C is irreducible, each column of its Cholesky factor L (except the final one) contains at least one non-zero subdiagonal entry (Liu 1986*a*). If, as before, the first such entry in column k is in row k_1 (or, equivalently, the first non-zero entry in row k of $L^\top$ is in column k_1), then the elimination tree $\mathcal{T}(C)$ (or simply $\mathcal{T}$) is defined to be the directed graph that is obtained by removing from the directed graph $\mathcal{G}(L^\top)$ all edges $(k \rightarrow i)$ for which $i > k_1$. It is straightforward to see that $\mathcal{T}(C)$ is a DAG and the ordering of its vertices is a topological ordering. Note that because of the non-cancellation rule, $(L^\top)_{ik} \neq 0$ is equivalent to stating that $(k \rightarrow i)$ is an edge of $\mathcal{G}(L^\top)$.

The elimination tree for the matrix C from Figure 3.1 is shown in Figure 3.2. The root vertex is 8. Following conventional notation, directional arrows are omitted from $\mathcal{T}(C)$ because an edge (k, i) is always directed from k to i with $i > k$.

The time complexity for constructing $\mathcal{T}(C)$ is $O(nz(C)\, g(nz(C), n))$ (Tarjan 1975, Liu 1990), where $nz(C)$ is the number of non-zeros in C and $g(nz(C), n)$ is a very slowly increasing function called the functional inverse of Ackermann's function. This means that, in practice, the elimination tree can be efficiently constructed in time that is essentially linear in $nz(C)$ (which is generally much smaller than $nz(L)$).

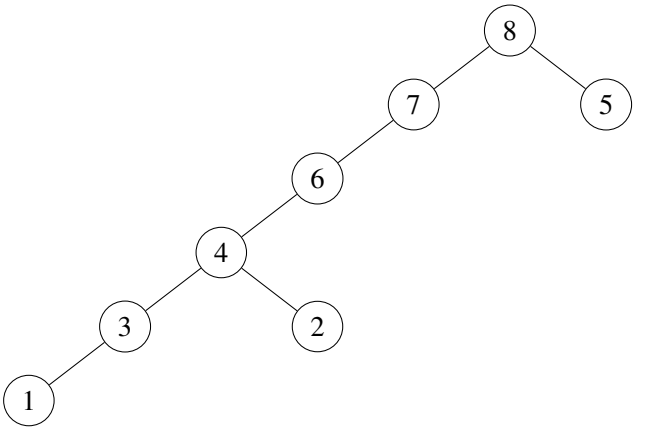


Figure 3.2. The elimination tree $\mathcal{T}(C)$ for the matrix from Figure 3.1.

The importance of $\mathcal{T}(C)$ is that it allows key characteristics of the Cholesky factor L to be computed symbolically. These include its row and column counts, the maximum intermediate memory required during the factorization, and the sparsity pattern $\mathcal{S}\{L\}$. The algorithms for doing this can depend on the ordering of $\mathcal{T}(C)$. A topological ordering of $\mathcal{T}(C)$ defines a labelling of its vertices corresponding to a symmetric permutation of C that does not affect the amount of fill-in in L (Liu 1990). An important class of topological orderings is obtained using a depth-first search starting at the root vertex. Once vertex i has been visited, all the vertices of the subtree rooted at i and denoted by $\mathcal{T}(i)$ are visited immediately after i and i is labelled as the last vertex of $\mathcal{T}(i)$. A topological ordering of $\mathcal{T}(C)$ is a postordering if and only if the set of vertex labels of any subtree $\mathcal{T}(i)$, $1 \leq i \leq n$, is a contiguous sublist of $1, \ldots, n$. Unless additional rules on how vertices are selected are imposed, a postordering is usually not unique.

Note that although the ordering algorithms discussed in Section 2.3 mainly target the reduction of fill-in (and thus $nz(L)$) and the number of operations required to compute L, the ordering also has a significant impact on the shape of the elimination tree, and this subsequently affects the potential to exploit parallelism within the factorization algorithm.

3.3. Supervariables

The performance of most algorithms used in the symbolic phase of a sparse Cholesky factorization can be enhanced by employing supervariables (recall Section 2.4). Let the vertex set $\mathcal{V}$ of the graph $\mathcal{G}(C)$ be partitioned into $nsup \geq 1$ non-empty disjoint subsets of indistinguishable vertices

$$\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_{nsup}. \tag{3.3}$$

If the vertices belonging to each subset $\mathcal{V}_1, \ldots, \mathcal{V}_{nsup}$ are numbered consecutively, with those in $\mathcal{V}_i$ preceding those in $\mathcal{V}_{i+1}$ ($1 \le i < nsup$), and if P is the permutation matrix corresponding to this ordering, then the permuted matrix $PCP^\top$ has a block structure in which the blocks are dense; the dimensions of the blocks are equal to the sizes of the indistinguishable sets. The matrix $PCP^\top$ can be condensed to a matrix of order equal to *nsup*; the corresponding graph is the supervariable graph. If the average number of variables in each supervariable is l, using the supervariable graph will reduce the amount of integer data that is handled during the symbolic phase by a factor of about l^2. Algorithms for finding supervariables are discussed in Hogg and Scott (2013*a*); see also Ashcraft (1995) and Saad (2003*a*).

To illustrate supervariables, consider the following example. For the given matrix C, we cannot immediately see sets of indistinguishable vertices. But by symmetrically permuting the matrix, we obtain the matrix on the right. The permutation matrix P corresponds to the new labelling of the rows and columns. It is now clear that there are three sets of indistinguishable vertices $\{(2,4), (1,5), (3)\}$:

$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} * & & * & & * \\ & * & * & * & \\ * & * & * & * & * \\ & * & * & * & \\ * & & * & & * \end{pmatrix} \end{array}, \quad PCP^\top = \begin{array}{c} \\ 2 \\ 4 \\ 1 \\ 5 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 4 & 1 & 5 & 3 \end{array} \\ \begin{pmatrix} * & * & & & * \\ * & * & & & * \\ & & * & * & * \\ & & * & * & * \\ * & * & * & * & * \end{pmatrix} \end{array}.$$

3.4. Basic left- and right-looking sparse Cholesky factorizations

There are several classes of algorithms that implement sparse Cholesky factorizations. Their major differences relate to how they schedule the computations. This affects the use of dense linear algebra kernels, memory requirements during the factorization and the potential for parallel implementations. Given the factor sparsity pattern $\mathcal{S}\{L\}$, define $\{j\_r\}$ to be the set of subdiagonal row indices of the non-zero entries in column j of L, that is,

$$\{j\_r\} = \{i > j \mid l_{ij} \neq 0\}.$$

Thus, $L_{\{j\_r\},j}$ denotes the non-zero subdiagonal entries in column j of L and, for any $k > j$, $L_{\{j\_r\},k}$ denotes the non-zero entries in column k of L with row indices belonging to $\{j\_r\}$. Algorithms 3.1 and 3.2 use this notation to outline simplified left- and right-looking variants of the sparse Cholesky factorization. In the left-looking version, update operations are not applied immediately to the remaining columns. Instead, at the start of major step j, all updates from columns 1 to $j-1$ are applied together to column j (lines 3 to 7 of Algorithm 3.1) and then it is factorized (lines 8 to 11). In the right-looking approach, outer product updates are applied to the part of the matrix that is still to be factorized as they are generated (lines 7 to 11 of Algorithm 3.2).

Algorithm 3.1. Simplified sparse left-looking Cholesky factorization
Input: SPD matrix A and sparsity pattern $\mathcal{S}\{L\}$.
Output: Cholesky factor L such that $A = LL^\top$.

for $j = 1 : n$ **do** ▷ Start of major step j
  $l_{jj} = a_{jj}$, $L_{\{j\_r\},j} = A_{\{j\_r\},j}$
  for $k \in \{k < j \mid l_{jk} \neq 0\}$ **do** ▷ Apply updates from previous columns
    $l_{jj} = l_{jj} - l_{jk} l_{jk}$ ▷ Update diagonal entry
    $L_{\{j\_r\},j} = L_{\{j\_r\},j} - L_{\{j\_r\},k}\, l_{jk}$ ▷ Update subdiagonal entries
  end for
  $l_{jj} = (l_{jj})^{1/2}$
  $L_{\{j\_r\},j} = L_{\{j\_r\},j} / l_{jj}$ ▷ Scale current column by the pivot
end for

Algorithm 3.2. Simplified sparse right-looking Cholesky factorization
Input: SPD matrix A and sparsity pattern $\mathcal{S}\{L\}$.
Output: Cholesky factor L such that $A = LL^\top$.

for $j = 1 : n$ **do**
  $l_{jj} = a_{jj}$, $L_{\{j\_r\},j} = A_{\{j\_r\},j}$
end for
for $j = 1 : n$ **do** ▷ Start of major step j
  $l_{jj} = (l_{jj})^{1/2}$
  $L_{\{j\_r\},j} = L_{\{j\_r\},j} / l_{jj}$ ▷ Scale current column by the pivot
  for $k \in \{k > j \mid l_{kj} \neq 0\}$ **do** ▷ Update remaining columns
    $l_{kk} = l_{kk} - l_{jk} l_{jk}$ ▷ Update diagonal entry in column k
    $L_{\{k\_r\},k} = L_{\{k\_r\},k} - L_{\{k\_r\},j}\, l_{jk}$ ▷ Update subdiagonal in column k
  end for
end for

3.5. Supernodes and the assembly tree

The simplified schemes above form the basis of more sophisticated algorithms. For efficiency, it is essential to take advantage of dense blocks within the factorization; these are able to exploit Level 3 BLAS routines. Column replication leads to the columns of L becoming denser as the factorization proceeds. Furthermore, fill-reducing orderings (such as the minimum fill and minimum degree algorithms) seek to choose vertices of $\mathcal{G}(C)$ that, at each stage, add a small number of new non-zeros to L, which contributes to the early columns of L being significantly

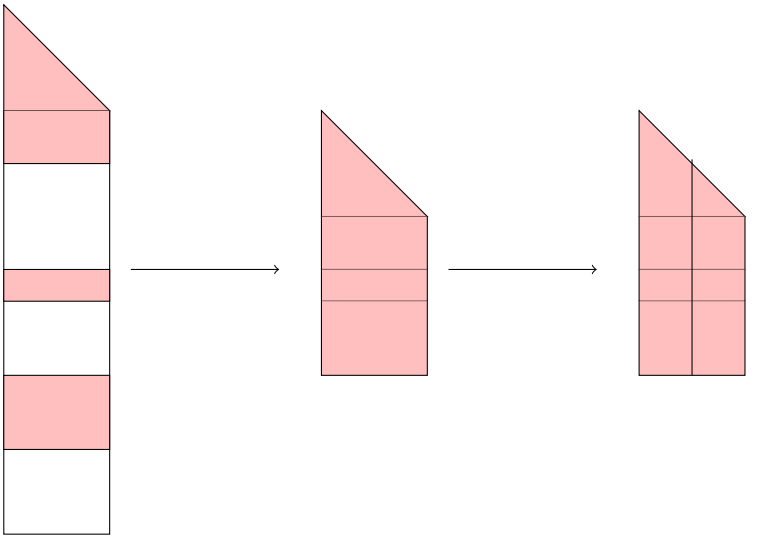


Figure 3.3. An illustration of a supernode (a), the corresponding nodal matrix (b) and the nodal matrix with two panels (c). The shaded lower triangular part of the block on the diagonal and the shaded block rows are treated as dense (the white row blocks are not stored).

sparser than those towards the end of factorization. Exploiting density within L can significantly reduce the computation time and memory of the numerical factorization. In particular, columns of L with the same sparsity structure can be grouped and the resulting block treated as a dense matrix for storage and computation purposes. Let $1 \le i, j \le n$ with $i + j - 1 \le n$. A set of contiguously numbered columns of L with indices $\mathcal{V}_L = \{i, i+1, \ldots, i+j-1\}$ is defined to be a supernode of L if

$$\mathcal{S}\{L_{:,i}\} \cup \{i\} = \mathcal{S}\{L_{:,i+j-1}\} \cup \mathcal{V}_L,$$

and $\mathcal{V}_L$ cannot be extended for $i > 1$ by adding $i - 1$ or for $i + j - 1 < n$ by adding $i + j$. In graph terminology, a supernode is a maximal clique of contiguous vertices of $\mathcal{G}(L + L^\top)$. A supernode is stored as a dense trapezoidal matrix (only the lower triangular part of the block on the diagonal is needed), but rows of zeros in the columns of the supernode are not held. This is termed a nodal matrix. This is illustrated in Figure 3.3.

Supernodes can be small (i.e. they may contain very few columns or even just a single column), in which case the costs associated with identifying them may not be offset by the increase in performance resulting from the potential for block operations. Thus, it can be advantageous to merge supernodes that have similar (but not exactly the same) non-zero patterns. This process is termed supernode

amalgamation, and the resultant nodes are referred to as relaxed supernodes (Ashcraft and Grimes 1989). Using relaxed supernodes increases the number of entries in L and the floating-point operations needed to compute it and use it, but this can be outweighed by the benefits of better use of Level 3 BLAS.

The supernodal elimination tree, which is also commonly called the assembly tree, is the reduction of the elimination tree that contains only supernodes. Each vertex of the elimination tree is associated with one elimination and a single integer (the index of its parent) is needed. Associated with each vertex of the assembly tree is an index list of the row indices of the non-zeros in the columns of the supernode. These implicitly define the sparsity pattern $\mathcal{S}\{L\}$.

3.6. Supernodal and DAG-based factorization

Assume L has $ns \leq n$ supernodes and let js and ks denote two of these supernodes. Generalizing the notation of Section 3.4, we let $\{js\_c\}$ denote the set of columns in js and let $\{js\_r\}$ be the set of row indices of the non-null subdiagonal row blocks belonging to js ($1 \leq js \leq ns$). Then $L_{\{js\_c\},\{js\_c\}}$ and $L_{\{js\_r\},\{js\_c\}}$ denote the block on the diagonal and the non-null subdiagonal row blocks in supernode js, and $L_{\{js\_c\},\{ks\_c\}}$ and $L_{\{js\_r\},\{ks\_c\}}$, respectively, denote the blocks with columns belonging to supernode ks and row indices belonging to $\{js\_c\}$ and $\{js\_r\}$. Using this notation, Algorithm 3.3 outlines a left-looking supernodal factorization. The for loop at line 3 applies updates from previous supernodes. At lines 4 and 7, only the lower triangular part of the block $L_{\{js\_c\},\{js\_c\}}$ on the diagonal is used. At line 4 it is updated and then at line 7 it is overwritten by its dense Cholesky factor. At line 8, a dense triangular solve is used to scale the columns in js. This supernodal left-looking approach is discussed in Ng and Peyton (1993*a*,*b*).

Algorithm 3.3. Supernodal left-looking Cholesky factorization
Input: Sparse SPD matrix C and sparsity pattern $\mathcal{S}\{L\}$ with ns supernodes.
Output: Cholesky factor L such that $C = LL^\top$.

1: **for** $js = 1 : ns$ **do**
2: $\quad L_{\{js\_c\},\{js\_c\}} = A_{\{js\_c\},\{js\_c\}}, \quad L_{\{js\_r\},\{js\_c\}} = A_{\{js\_r\},\{js\_c\}}$
3: $\quad$ **for** $ks \in \{ks < js \mid L_{\{js\_c\},\{ks\_c\}} \neq 0\}$ **do**
4: $\quad\quad L_{\{js\_c\},\{js\_c\}} = L_{\{js\_c\},\{js\_c\}} - L_{\{js\_c\},\{ks\_c\}} L^\top_{\{js\_c\},\{ks\_c\}}$
5: $\quad\quad L_{\{js\_r\},\{js\_c\}} = L_{\{js\_r\},\{js\_c\}} - L_{\{js\_r\},\{ks\_c\}} L^\top_{\{js\_c\},\{ks\_c\}}$
6: $\quad$ **end for**
7: $\quad L_{\{js\_c\},\{js\_c\}} = L_{\{js\_c\},\{js\_c\}} L^\top_{\{js\_c\},\{js\_c\}}$
8: $\quad L_{\{js\_r\},\{js\_c\}} = L_{\{js\_r\},\{js\_c\}} L^{-\top}_{\{js\_c\},\{js\_c\}}$
9: **end for**

Let us denote the tasks at lines 4 and 5 by update(js_r, ks) and update(js_c, ks), and those at lines 7 and 8 by factorize(js) and solve(js), respectively. The factorization is then comprised of a set of tasks connected by dependences that can be expressed in the form of a task DAG. For example, all update tasks for which $ks \in \{ks < js \mid L_{\{js\_c\},\{ks\_c\}} \neq 0\}$ must be performed before the task factorize(js) can be carried out. That is, the DAG contains edges update(js_r, ks) $\rightarrow$ factorize(js) and update(js_c, ks) $\rightarrow$ factorize(js). Compared to a left- or right-looking algorithm, a DAG-driven factorization allows greater freedom in the order in which the tasks are carried out, improving the scope for exploiting parallelism.

It can be beneficial in terms of the run-time to increase the number of tasks (i.e. replace 'large' tasks by a number of smaller tasks). This can be done by splitting the nodal matrix into a number of panels (Hogg, Reid and Scott 2010) (see Figure 3.3).

3.7. *The multifrontal method*

Another approach that has proved very successful is the multifrontal method. It was introduced in the 1980s (Duff and Reid 1983) (but see also the earlier work by Speelpenning 1978). It uses a postordering of the assembly tree and organizes the operations that take place during the factorization in such a way that the entire factorization is performed through the partial factorizations of a sequence of dense submatrices.

To assist in describing the multifrontal method, observe that the single step of a Cholesky factorization given in (3.1) can be generalized to $j \geq 1$ steps (block elimination), that is,

$$\begin{aligned} C &= \begin{pmatrix} C_{11} & C_{j+1:n,1:j}^{\top} \\ C_{j+1:n,1:j} & C_{j+1:n,j+1:n} \end{pmatrix} \\ &= \begin{pmatrix} L_1 & \\ C_{j+1:n,1:j} L_1^{-\top} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & S^{(j+1)} \end{pmatrix} \begin{pmatrix} L_1^{\top} & L_1^{-1} C_{j+1:n,1:j}^{\top} \\ & I \end{pmatrix}. \end{aligned}$$

Here $C_{11} = L_1 L_1^{\top}$ is the $j \times j$ Cholesky factorization of the $(1,1)$ block of C and the Schur complement

$$S^{(j+1)} = C_{j+1:n,j+1:n} - C_{j+1:n,1:j} C_{11}^{-1} C_{j+1:n,1:j}^{\top}$$

is the part of the matrix that is still to be factorized. The term $C_{j+1:n,1:j} C_{11}^{-1} C_{j+1:n,1:j}^{\top}$ represents the update contributions from the first j rows and columns to the $(2,2)$ block $C_{j+1:n,j+1:n}$. It can also be expressed in the outer product form (3.2) and this is the basis of the multifrontal method, which can be viewed as providing an effective management of the outer product updates when C is sparse.

For each supernode js, the multifrontal algorithm creates a matrix $F(js)$ called a frontal matrix. This is a small symmetric dense matrix with columns and rows corresponding to the groups of columns to be eliminated and all the rows in which

Algorithm 3.4. Multifrontal Cholesky factorization
Input: Sparse SPD matrix C and its postordered assembly tree.
Output: Cholesky factor L such that $C = LL^\top$.

1: **for** $js = 1 : ns$ **do** ▷ Follow the postordering of the tree
2: Assemble frontal matrix $F(js)$ using rows/columns of C and contribution blocks from children of js
3: Permute and then partially factorize $F(js)$ ▷ Results in a block column of L and contribution block $V(js)$
4: Push $V(js)$ onto the stack. ▷ $V(js)$ will be popped from the stack when assembling $F(\text{parent}(js))$
5: **end for**

these columns have non-zeros. At the leaf vertices, $F(js)$ is assembled by adding into it the entries in the corresponding rows and columns of the matrix C. At each stage, $F(js)$ is permuted to have a 2×2 block structure

$$F(js) = \begin{pmatrix} F_{11} & F_{21}^\top \\ F_{21} & F_{22} \end{pmatrix},$$

in which all variables in the (1,1) block can be eliminated (i.e. they are fully summed) but the remaining variables cannot be eliminated until later in the factorization because further contributions are still to be added (assembled). At leaf vertices, $F_{22} = 0$. The Schur complement formed by the elimination of the fully summed variables within $F(js)$ is called the contribution block $V(js)$ (it is also sometimes referred to as the generated element or update matrix). Because of symmetry, only the lower triangular parts of $F(js)$ and $V(js)$ need to be computed.

After the partial factorization of $F(js)$, the factor columns that have been computed are added into the factor L and $V(js)$ is stored. When all the children of a parent ks have been eliminated, the parent retrieves the contribution blocks of its children, and assembles them (together with the rows and columns of the matrix C corresponding to ks) into its own frontal matrix $F(ks)$. Variables in ks are fully summed (and ready for elimination) if all descendants of the corresponding vertex in the assembly tree have been eliminated. The process of performing a partial factorization and then storing the contribution block is repeated until the root vertex is reached. At the root, all the variables are fully summed and so can be eliminated to complete the factorization. The approach is summarized as Algorithm 3.4.

At each stage js (except the root), the contribution block $V(js)$ must be stored until $ks = \text{parent}(js)$ is processed. It is convenient to use a stack for this. It is

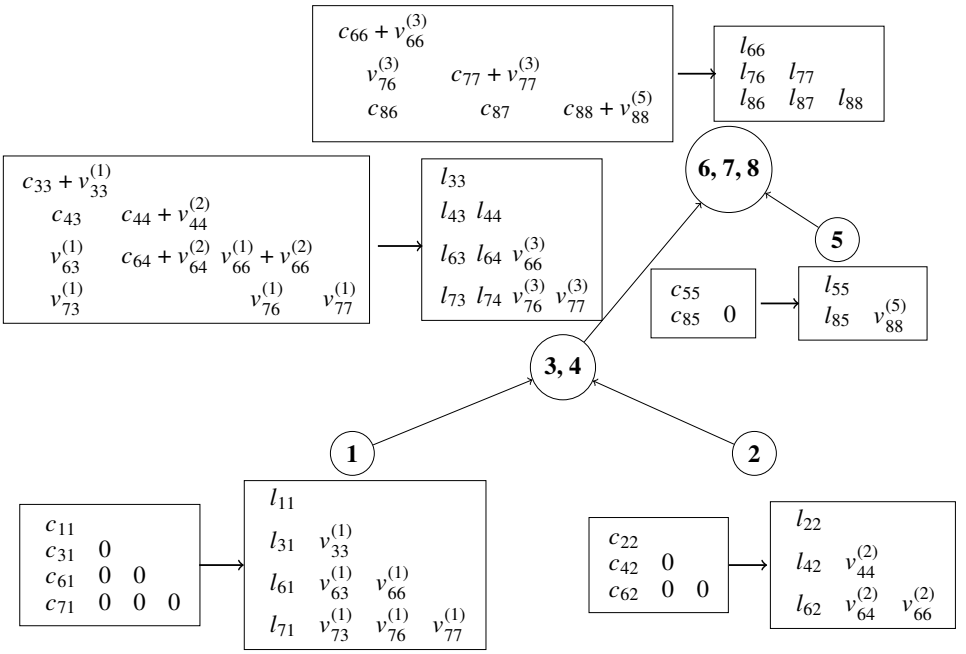


Figure 3.4. The multifrontal method applied to the matrix $C = \{c_{ij}\}$ from Figure 3.1. The assembly tree is shown. Each vertex shows the transformation from its frontal matrix to the computed entries of L and contribution block that is passed from child to parent. To illustrate the method, global indices are used for the entries of the contribution blocks; superscripts are used to indicate the supernode the contribution comes from.

easy to see that for any subtree $\mathcal{T}(js)$ of the postordered assembly tree, js is the last vertex in $\mathcal{T}(js)$ to be processed and the contribution blocks from its children are those on the top of the stack. Thus, they can be popped from the stack when assembling $F(js)$.

Figure 3.4 illustrates the multifrontal method applied to the 8×8 matrix from Figure 3.1. It has two non-trivial supernodes $(3, 4)$ and $(6, 7, 8)$ and the corresponding assembly tree has five vertices.

The contributions to the frontal matrix vary in size, so careful attention must be paid to the indices and the mapping between global and local indices. The assembly operation is called extend-add (Liu 1992). The 'Achilles heel' of sparse direct methods is the need for indirect addressing. The multifrontal method cannot avoid this, but it only occurs during the assembly operations while all the arithmetic is performed using direct addressing in the dense frontal matrices.

Implementations of the multifrontal method may include an option to reduce the main memory requirements by using disk storage, thus enabling larger problems to be solved (this option was very necessary when computers had limited main

memories). An out-of-core method writes the columns of L to disk as they are computed and may also hold the original matrix, the stack (and possibly the frontal matrix) in files on disk. During the subsequent triangular solves, the factor data must be read back in, which adds to the overall solution cost. The efficient use of disk storage is discussed in Reid and Scott (2009).

An important aspect of the postordered assembly tree is that it only partially defines an ordering of the factorization tasks. This is because it is only necessary for the elimination operations at a child vertex to be completed before those at its parent can be performed. This freedom enables tree level parallelism to be exploited (Amestoy, Duff and L'Excellent 2000). In addition, advantage can be taken of parallelism within the partial factorizations of the dense frontal matrices; this is referred to as node level parallelism. This is key for good performance, particularly at the large vertices that are at (or close too) the root vertex.

3.8. Numerical rank deficiency in the Cholesky factorization

So far in our discussion of Cholesky factorizations we have assumed that A is of full column rank. As observed in Section 1.5, when solving least-squares problems, we need to be more aware of rank deficiency than when solving general linear systems of equations. Formally, the Cholesky factorization of $A^\top A$ does not break down if

$$c\, n^{3/2}\, \epsilon\, \kappa_2(A^\top A) \leq 1,$$

where c is a small constant (Wilkinson 1968, Golub and Van Loan 1996). If A is potentially ill-conditioned or if $A^\top A$ is positive semidefinite, diagonal pivoting can be incorporated into the factorization. At each step, the largest diagonal entry in the Schur complement is selected as the next pivot (i.e. the next to be eliminated). It is permuted to be the leading entry of the Schur complement and thus the Cholesky factorization of $(AP)^\top AP$, for some permutation matrix P, is computed. The rank rk of $A^\top A$ is revealed by the factorization

$$P^\top A^\top AP = LL^\top, \quad L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & 0 \end{pmatrix}, \tag{3.4}$$

where $L_{11} \in \mathbb{R}^{rk \times rk}$ is lower triangular. Such a factorization exists (Higham 1990), although incorporating pivoting adds significantly to the complexity of sparse factorization algorithms. Once the factorization is obtained, the least-squares solution can be computed from the relations

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = P^\top A^\top b, \quad L_{11} z = b_1, \quad L_{11}^\top y_1 = z - L_{21}^\top y_2, \quad x = P \begin{pmatrix} y_1^\top & y_2^\top \end{pmatrix}^\top,$$

where $b_1, y_1, z \in \mathbb{R}^{rk}$ and $y_2 \in \mathbb{R}^{n-rk}$ is an arbitrary vector. The fact that y_2 can be chosen arbitrarily follows from the singularity of $A^\top A$. If $A^\top A$ is close to being positive definite, the numerical rank is better revealed using a QR factorization (Section 4.5).

3.9. Software for sparse Cholesky factorizations

Since the 1970s, software packages that implement algorithms for sparse Cholesky factorizations have been developed. One of the earliest and perhaps the most well-known serial multifrontal solver is `MA27` (Duff and Reid 1983). It solves sparse symmetric linear systems that are not necessarily positive definite. It was written for inclusion in the HSL library[9] (prior to 2000, it was known as the Harwell Subroutine Library (Scott 2023)) and, although later HSL codes have been designed and developed with the intention of superseding it, `MA27` remains in use.[10] CHOLMOD[11] is a supernodal Cholesky solver for sparse SPD linear systems (Chen, Davis, Hager and Rajamanickam 2008); it offers CUDA GPU acceleration. CHOLMOD is used by `backslash` within MATLAB. PaStiX[12] also has a parallel supernodal Cholesky factorization solver (Hénon, Ramet and Roman 2002). `HSL_MA87` implements a sparse DAG-based Cholesky factorization for shared memory architectures (Hogg *et al.* 2010). The package `HSL_MA85` is specifically designed to use sparse direct methods to solve large-scale diagonally weighted linear least-squares problems; it calls `HSL_MA87` if the user chooses to use the normal matrix formulation. Within the cuSolver library,[13] cuSolverSP includes a sparse Cholesky code.

In addition to `MA27`, there are a number of important packages that are designed for general sparse symmetric systems and consider positive definite systems as a special case. Pardiso[14] implements a left–right looking sparse Cholesky algorithm; the Intel oneAPI Math Kernel Library (oneMKL) includes a variant of this solver. The MUMPS package[15] and WSMP[16] both provide distributed memory general-purpose multifrontal solvers. The current HSL library includes a number of multifrontal codes. From the early 2000s, `MA57` (Duff 2004) remains very popular; it only exploits parallelism through the use of Level 3 BLAS during the factorization of the frontal matrices. A more recent package is the shared memory solver `HSL_MA97` (Hogg and Scott 2013*b*). It has the attractive feature that, although it is a parallel solver, it computes bit-compatible solutions, which can be important for some applications. `HSL_MA77` (Reid and Scott 2009) is designed to minimize memory requirements, thereby potentially enabling larger matrices to be factorized. The user can choose to allow the factors and the multifrontal stack to be efficiently held outside of main memory (an option that is also offered by MUMPS).

[9] https://www.hsl.rl.ac.uk/
[10] https://www.hsl.rl.ac.uk/archive/
[11] https://people.engr.tamu.edu/davis/suitesparse.html
[12] https://solverstack.gitlabpages.inria.fr/pastix/
[13] https://docs.nvidia.com/cuda/cusolver/
[14] https://panua.ch/pardiso/
[15] https://mumps-solver.org/
[16] http://researcher.watson.ibm.com/researcher/view_group.php?id=1426

4. Sparse QR factorizations

4.1. Introduction to QR factorizations

Let us again assume that $A \in \mathbb{R}^{m\times n}$ $(m > n)$ is of full column rank. Then the QR factorization (1.11) of A is unique if the diagonal entries of the matrix R are positive. Approaches for computing a QR factorization can be based on Givens rotations, Householder reflectors or Gram–Schmidt orthogonalization. They often differ significantly in their numerical properties as well as in their efficiency, memory demands and computational output, particularly when A is sparse.

4.1.1. Givens rotations

Givens' method is based on elementary orthogonal transformations that represent rotations in a plane spanned by two coordinate axes. Consider an anticlockwise rotation of a non-zero vector $w = (w_1\ w_2)^\top \in \mathbb{R}^2$ through an angle θ such that the second entry of the rotated vector $y = (y_1\ y_2)^\top$ is zero. The rotation can be written as the matrix transformation

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}, \quad \text{where } s = w_2/d,\ c = -w_1/d,\ d = \|w\|_2.$$

This transformation can be expressed as a plane rotation in the extended space $\mathbb{R}^{m\times m}$. If the two axes correspond to row indices i and j of A, then the $m \times m$ matrix $G(i, j)$ given by

$$G(i,j) = \begin{array}{c} \\ i \\ \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & i & & j & & \end{array} \\ \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \end{array},$$

with 1s on the diagonal except rows i and j, is an orthogonal transformation that when applied to A affects only the entries in rows i and j. This is a Givens rotation, named after the pioneering work of Wallace Givens (1953) (although it was used even earlier by Jacobi (Jacobi 1845, Golub and Meurant 1997)). Explicit calculation of the angle θ is rarely necessary or desirable. Instead, the scalars c and s, which correspond to $\cos(\theta)$ and $\sin(\theta)$, respectively, are computed.

Givens rotations can be used to systematically eliminate individual subdiagonal entries of A by applying them one by one to pairs of rows. In particular, applying them in the following order produces the QR factorization

$$G(n, m) \dots G(2, m) \dots G(2, 3) \dots G(1, m) \dots G(1, 2)A = R, \tag{4.1}$$

where the Q factor is the transpose of the product of the Givens rotations. When A

is sparse, it is sufficient to eliminate only the non-zero entries. But, as the process of elimination of non-zero entries in A continues, new non-zero fill-in entries typically arise (this is termed intermediate fill-in) and they must also be eliminated. The order in which the rotations are applied must satisfy some rules. A basic rotation-based strategy by columns in (4.1) eliminates the subdiagonals in column 1 of A, followed by those in column 2, and so on. Significant research has been devoted to developing row-ordering schemes, sometimes combined with column-ordering schemes, to minimize intermediate fill-in; see, for example, George, Liu and Ng (1986) and the references therein.

Givens rotations provided the first technique for computing the QR factorization of a sparse matrix and their sophisticated use resulted in an efficient computational framework for solving linear least-squares problems (George and Heath 1980).

4.1.2. Householder reflectors

An alternative to Givens rotations is to use Householder reflectors, the name recognizing the pioneering work of Alston Householder (1958). A Householder reflector (also known as a Householder transformation or Householder matrix) is a symmetric orthogonal matrix of the form

$$H = I - \beta w w^\top,$$

where β is a scalar and w is a non-zero vector chosen such that if $y = Hx$ then $|(y)_1| = \|x\|_2$ and all other entries are zero (Golub and Van Loan 1996). The vector w is called a Householder vector. The application of an $m \times m$ Householder reflector $H^{(1)}$ to the matrix A with a_1 as its first column can be written as

$$\begin{aligned} H^{(1)}A &= H^{(1)} \begin{pmatrix} a_1 & A_{1:m,2:n} \end{pmatrix} \\ &= \left(I - \beta_1 w^{(1)} (w^{(1)})^\top\right) \begin{pmatrix} a_1 & A_{1:m,2:n} \end{pmatrix} = \begin{pmatrix} R_{1,1} & R_{1,2:n} \\ 0 & A^{(1)} \end{pmatrix}. \end{aligned} \tag{4.2}$$

The elimination of subdiagonal non-zero entries can be continued by applying an $(m-1) \times (m-1)$ Householder reflector $H_2 = I - \beta_2 w^{(2)} (w^{(2)})^\top$ to $A^{(1)}$ such that its $(1, 1)$ entry becomes zero. H_2 can be extended to an $m \times m$ matrix by setting

$$H^{(2)} = \begin{pmatrix} 1 & \\ & H_2 \end{pmatrix}.$$

Setting $A^{(0)} = A$, the iteration formula is

$$A^{(j)} = H^{(j)} A^{(j-1)}. \tag{4.3}$$

Repeating the process yields the factorization

$$H^{(n)} \dots H^{(2)} H^{(1)} A = A^{(n)},$$

from which the QR factorization is obtained by setting $Q = H^{(1)} H^{(2)} \dots H^{(n)}$ and $R = A^{(n)}$. In the dense case, compared with using Givens rotations, Householder reflectors reduce the floating-point operation count by a third.

If the Q factor is required to be retained then it can be held explicitly as the product of the Householder reflectors $H^{(1)}H^{(2)}\dots H^{(n)}$, or implicitly as the sequence of Householder vectors $w^{(1)}, w^{(2)}\dots$. For sparse A, the latter generally requires significantly less memory because the $w^{(i)}$ are typically sparse. Storing blocks of Householder vectors allows the use of Level 3 BLAS (Schreiber and Van Loan 1989); see also Amestoy, Duff and Puglisi (1996*b*) and Davis (2011).

4.1.3. Gram–Schmidt QR factorization

The Gram–Schmidt process computes vectors $q_1, q_2, \dots, q_n$ such that each column a_k, $k = 1, \dots, n$, of A can be expressed as a linear combination

$$a_k = r_{1k}q_1 + r_{2k}q_2 + \dots r_{nk}q_n, \quad r_{kk} \neq 0,$$

with $\langle q_i,\ q_i\rangle = 1$ and $\langle q_i,\ q_j\rangle = 0$ for $i \neq j$, where $\langle \cdot\,, \cdot\rangle$ denotes the inner product. This can be written as the QR factorization (1.11) with the orthonormal matrix $Q_1 = (q_1, q_2, \dots, q_n) \in \mathbb{R}^{n\times n}$. Unlike the Householder QR factorization, in which orthogonal transformations are applied to reduce A to upper triangular form R and Q is (implicitly) defined as the product of the Householder reflectors, Q_1 is held explicitly as linear combinations of the columns of A.

The classical Gram–Schmidt (CGS) process generates q_k by orthonormalizing a_k against $Q_{k-1} = (q_1, q_2, \dots, q_{k-1})$, that is, it computes the vector

$$a = a_k - Q_{k-1}Q_{k-1}^{\top}a_k$$

and then sets $r_{kk} = \|a\|_2$ and $q_k = a/r_{kk}$. CGS is suited to parallel computation (the main work can be performed as matrix–vector multiplications), but it often produces a non-orthogonal set of vectors because of cancellations in the subtractions.

In the modified Gram–Schmidt (MGS) algorithm, q_k is obtained by first projecting column a_k onto the subspace orthogonal to q_1, then the resulting vector is projected onto the subspace orthogonal to $\text{span}\{q_1, q_2\}$, up to the projection onto the subspace orthogonal to $\text{span}\{q_1, q_2, \dots, q_{k-1}\}$. This limits the amplification of the rounding errors affecting the orthogonality of q_k with respect to the previously computed vectors. In finite precision arithmetic, MGS is not equivalent to CGS. The loss of orthogonality is proportional to the condition number $\kappa_2(A)$ for MGS and to $\kappa_2(A)^2$ for a variant of CGS (Björck 1967*b*, Giraud, Langou, Rozložník and van den Eshof 2005).

Some applications require that the computed vectors are orthogonal to machine precision. In this case, it may be necessary to reapply the orthogonalization. This can be done at each step or selectively if the computed q_k is not acceptable. At step k, q_k is accepted if $\|q_k\|_2 > \alpha\|a_k\|_2$ for some chosen parameter α. Otherwise, q_k is reorthogonalized against Q_{k-1}. Typically $\alpha \in (0.1,\ 1/\sqrt{2})$ is used. If reorthogonalization of CGS is applied once at each step then there is no significant difference between this approach and MGS. Fast Gram–Schmidt algorithms use a block approach. A recent overview of such algorithms and their stability properties is given in Carson, Lund, Rozložník and Thomas (2022).

4.2. *Symbolic QR factorization*

Contemporary sparse QR algorithms typically start with a symbolic preprocessing phase. This first orders the columns of A to limit fill-in in the R factor using, for example, a column variant of the minimum degree algorithm (Section 2.5). Further symbolic steps may provide the size of R or its sparsity pattern and, possibly, that of Q. If Householder reflectors are used, the size or sparsity pattern of the matrix of Householder vectors can be predicted in advance.

A Givens rotation $G(i, j)$ applied to the row vectors $A_{i,i:n}$ and $A_{j,i:n}$ of A can be written as

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} A_{i,i:n} \\ A_{j,i:n} \end{pmatrix} = \begin{pmatrix} A'_{i,i:n} \\ A'_{j,i:n} \end{pmatrix},$$

where c and s are chosen to eliminate $A_{j,i}$ (i.e. $A'_{j,i} = 0$). Consider the following simple example:

$$\begin{pmatrix} A_{i,i:n} \\ A_{j,i:n} \end{pmatrix} = \begin{pmatrix} * & * & * & & * & & * \\ * & & & * & & & \end{pmatrix}.$$

Applying $G(i, j)$ gives

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} A_{i,i:n} \\ A_{j,i:n} \end{pmatrix} = \begin{pmatrix} * & * & * & * & * & & * \\ & * & * & * & * & & * \end{pmatrix} = \begin{pmatrix} A'_{i,i:n} \\ A'_{j,i:n} \end{pmatrix}.$$

Observe that the $(1, 1)$ entry $A'_{i,i}$ remains non-zero (it is the Euclidean norm of the vector $(A_{ii}\ A_{ji})^\top$) and the sparsity patterns of columns 2 to n satisfy

$$\mathcal{S}\left(A'_{i,i+1:n}\right) = \mathcal{S}(A_{i,i+1:n}) \cup \mathcal{S}(A_{j,i+1:n}), \quad 1 \le i \le n-1.$$

This local merge rule provides a symbolic rule for the sparsity patterns of row vectors to which a Givens rotation is applied. It appears reasonable because, for a non-zero vector $(u\ v)^\top$ transformed by an arbitrary Givens rotation

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} cu - sv \\ su + cv \end{pmatrix} = \begin{pmatrix} u' \\ v' \end{pmatrix},$$

both entries u' and v' are generally non-zero (unless θ is a multiple of a right angle) (George and Heath 1980). However, the fill-in can be overestimated. This is illustrated using the next example, in which $a, b \neq 0$ (Gentleman 1976)

$$\begin{pmatrix} * & a & b & & \\ * & & & * & * \\ * & & & * & * \end{pmatrix} \rightarrow \begin{pmatrix} * & c'ca & c'cb & * & * \\ & sa & sb & * & * \\ & s'ca & s'cb & * & * \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} * & a & b & * & * \\ & c''sa - s''s'ca & c''sb - s''s'cb & * & * \\ & s''sa + c''s'ca & s''sb + c''s'cb & * & * \end{pmatrix}.$$

The second matrix is obtained from the first one by applying $G(2,1)$ with c, s chosen to eliminate the $(2,1)$ entry. The subsequent Givens rotation $G(3,1)$ with parameters c', s' is applied to the result to eliminate the $(3,1)$ entry. Then, applying the rotation with parameters c'', s'' to eliminate the intermediate fill-in at position $(3,2)$ to the second matrix gives the third one, in which we should have $s''sa + c''s'ca = 0$. But this entry is just a non-zero multiple of the entry $s''sb + c''s'cb$ at $(3,3)$, independently of the values of a and b. In this case, the row merge rule is not able to predict that the $(3,3)$ entry also always becomes zero.

Next, consider a Householder reflector applied as in (4.2). Because its application has the form of an outer product of sparse row and column vectors, the sparsity pattern of a row of $\binom{R_{1,2:n}}{A^{(1)}}$ can be obtained as

$$\mathcal{S}(\text{row}) = \mathcal{S}(R_{1,2:n}) \bigcup_k \mathcal{S}\big(A^{(1)}_{k,1:n-1}\big).$$

This can be regarded as an extension of the row merge rule for Householder reflectors. That is, $\mathcal{S}(\text{row})$ unifies the sparsity patterns of all the rows involved in the outer product update (George and Ng 1983, George, Liu and Ng 1988). But, as in the case of Givens rotations, the extended row merge rule may overestimate the actual fill-in.

It is potentially possible to predict fill-in in R using the sparsity pattern of $A^\top A$ and the relations (1.12). Uniqueness of the Cholesky factorization implies that an estimate of the sparsity pattern of the Cholesky factor of $A^\top A$ can be used to predict $\mathcal{S}(R)$. However, this can again lead to an overestimate. Consider the matrix given by

$$A = \begin{pmatrix} * & * & * \\ & * & \\ & & * \\ & & * \end{pmatrix}. \tag{4.4}$$

In this case, $A^\top A$ is dense and consequently $\mathcal{S}(L)$ is predicted to be dense. However, the QR factorization of A simply requires the elimination of the $(4,3)$ entry, and this can be done by an orthogonal transformation that changes only the last two rows of A. Consequently, $\mathcal{S}(R)$ is equal to the sparsity pattern of the first three rows of A.

The relationship between the different predictions is summarized by the following result (George and Heath 1980, Coleman, Edenbrandt and Gilbert 1986), which is independent of the numerical values of the non-zero entries of A.

Lemma 4.1. $\mathcal{S}(R) \subseteq \{\text{prediction of } \mathcal{S}(R) \text{ based on row merge rule }\} \subseteq \{\text{prediction of } \mathcal{S}(R) \text{ based on } A^\top A\}$.

4.2.1. The Dulmage–Mendelsohn decomposition

The strong Hall property, which is based solely on the structure of A, can be used to examine when the row merge rule and the prediction based on $A^\top A$ do not give overestimates. An $m \times n$ matrix A with $m \geq n$ is said to be a Hall matrix (or to have the Hall property) if every set of k columns has non-zeros in at least k rows ($1 \leq k \leq n$). Note that a full-rank matrix must have the Hall property. A is a strong Hall matrix (or to have the strong Hall property) if every set of k columns ($1 \leq k < n$) has non-zeros in at least $k + 1$ rows. The matrix in (4.4) does not have the strong Hall property because its first column has a single entry.

It can be shown that if A has the strong Hall property then $\mathcal{S}(R)$ is exactly predicted by the local merge rule and the Cholesky factorization of $A^\top A$ (Coleman *et al.* 1986). Furthermore, exact predictions for Q and for the matrix W whose columns are the Householder vectors are possible (Ng and Peyton 1992, Hare, Johnson, Olesky and van den Driessche 1993); see Pothen (1993) for a discussion of exact predictions of sparsity patterns (even if A does not have the strong Hall property).

The strong Hall property can be exploited by using the Dulmage–Mendelsohn decomposition of A (Pothen and Fan 1990). This decomposition, which is obtained using maximum matching algorithms, provides a precise structural characterization of rectangular matrices. For an overdetermined matrix, the Dulmage–Mendelsohn decomposition comprises row and column permutations P_1 and P_2 such that

$$P_1 A P_2 = \begin{pmatrix} A_1 & A_2 \\ 0 & A_3 \end{pmatrix}.$$

Here A_1 is an $m_1 \times m_1$ matrix and A_3 is an $m_3 \times n_3$ overdetermined matrix ($m_3 > n_3$ or $m_3 = n_3 = 0$) with the strong Hall property. If the permutations are chosen so that A_1 is additionally block upper triangular, then the square blocks on the diagonal of A_1 also have the strong Hall property; this is termed the fine Dulmage–Mendelsohn decomposition. A simple example illustrating this is

$$P_1 A P_2 = \left(\begin{array}{cccc|cc} * & & & & & * \\ * & * & & * & & \\ & & * & * & & \\ & & * & * & * & \\ \hline & & & & * & * \\ & & & & & * \\ & & & & * & \end{array}\right).$$

If the QR factorizations of the blocks on the diagonal of A_1 and the overdetermined matrix A_3 are computed, then, because they have the strong Hall property, the sparsity patterns of their respective factors can be exactly predicted. Note that,

Algorithm 4.1. Row ordering of A for QR algorithm
Input: The column indices $f_i(A)$ and $l_i(A)$ of the first and last non-zero entries in row i of A.
Output: Row permutation of A.

1: Order the rows by increasing $f_i(A)$.
2: **for** $k = 1 : \max_i f_i(A)$ **do**
3: Order all rows with $f_i(A) = k$ by increasing $l_i(A)$.
4: **end for**

despite this, the Dulmage–Mendelsohn decomposition is not always recommended. If A is ill-conditioned or close to rank-deficient, then it may not be sufficient to factorize only the blocks with the strong Hall property (Pothen 1993).

4.2.2. Row-ordering algorithms

Although the computed R factor does not depend on the order of the rows of A, the row ordering can significantly affect the intermediate fill and the work needed to compute the factorization. This is illustrated by the following matrix:

$$A = \begin{pmatrix} * & * & * \\ * & & \\ * & & \\ * & & \\ & * & \\ & & * \end{pmatrix}, \quad PA = \begin{pmatrix} * & & \\ * & & \\ * & & \\ * & * & * \\ & * & \\ & & * \end{pmatrix}.$$

Eliminating the $(2, 1)$ and $(3, 1)$ entries using Givens rotations $G(1, 2)$ and $G(1, 3)$, there is intermediate fill-in in all remaining columns, but if rows 1 and 4 of A are exchanged then this fill-in does not occur when applying the same rotations to PA. Heuristic algorithms have been proposed, including the simple approach outlined in Algorithm 4.1. Ties at line 3 can be resolved by ordering the rows in ascending order of the number of new non-zero entries that are created.

An alternative strategy is to order the rows in ascending order of the column index $l_i(A)$ of the last entry in the row. When row $a_i^\top$ is processed, because all previous rows have entries only in columns with index at most $l_i(A)$, there is no fill in $a_i^\top$ in columns $l_{i+1}(A)$ to n.

4.3. Numerical sparse QR factorization

A significant step in the development of efficient QR factorizations was the introduction of the row merge tree (Liu 1986*b*). This generalizes Givens rotations to so-called submatrix rotations that merge triangular submatrices and makes them

more efficient. At the same time, the approach shows that the elimination tree $\mathcal{T}(A^\top A)$ (recall Section 3.2) can be used to control the order in which the triangular submatrices are merged. To illustrate the basic principle, consider the sparse matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} * & & * & \\ * & & * & \\ * & & * & * \\ & * & * & \\ & * & * & \\ & * & * & * \\ & & * & * \end{pmatrix} \end{array}.$$

Let A_1 be the submatrix comprising rows $1, 2, 3$ and columns $1, 3, 4$ of A, and let A_2 be the submatrix comprising rows $4, 5, 6$ and columns $2, 3, 4$. Perform the QR factorizations $A_1 = Q_1' R_1'$ and $A_2 = Q_2' R_2'$. These can be computed independently (and different orthogonalization techniques can be used). If Q_1 and Q_2 denote the orthogonal matrices corresponding to extending Q_1' and Q_2' to 7×7 matrices, then we obtain a partial orthogonal transformation of A:

$$A = Q_1 Q_2 \begin{pmatrix} * & & * & * \\ & & * & * \\ & & & * \\ & * & * & * \\ & & * & * \\ & & & * \\ & & * & * \end{pmatrix}.$$

Here, the last row $A_{7,1:4}$ is unchanged. The next step is to permute the rows of the partial transformation corresponding to the first row of R_1' and the first row of R_2' to be rows 1 and 2 of the final factor R of A. This gives

$$\begin{array}{c} \\ 1 \\ 4 \\ 2 \\ 3 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} * & & * & * \\ & * & * & * \\ & & * & * \\ & & & * \\ & & * & * \\ & & & * \\ & & * & * \end{pmatrix} \end{array}.$$

The remaining rows of each of the upper trapezoidal matrices are called the QR contribution blocks. In the example, they correspond to rows 3 and 4 and rows 5 and 6 of the permuted partially transformed matrix. To minimize the intermediate fill-in, the rows of the submatrix coming from the two contribution blocks and row

7 of A are ordered using Algorithm 4.1 to give

$$\begin{array}{c} \\ 1 \\ 4 \\ 2 \\ 5 \\ 7 \\ 3 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} * & & * & * \\ & * & * & * \\ & & * & * \\ & & * & * \\ & & * & * \\ & & & * \\ & & & * \end{pmatrix} \end{array}.$$

A third orthogonalization transformation is applied to rows 3 to 7 of this permuted matrix to yield the final QR factorization.

This approach can be generalized to more than two blocks. The blocks that are independently orthogonalized corresponded to sets of rows of A that have the first non-zero in the same column. The key observation that implies the precedence relations among the computed triangular factors of such blocks of rows is determined by $\mathcal{S}(\mathcal{R})$. Namely, before performing a factorization of a block of rows with the first non-zero in column k, all block factorizations that result in upper triangular factors with a non-zero in column k must already have been performed. This order is determined by precedence relations given by the elimination tree $\mathcal{T}(A^\top A)$.

4.4. Multifrontal QR factorization

Significant advances in efficient implementations of QR factorizations of sparse problems have come from numerous contributions over many years, including those of Matstoms (1994), Amestoy *et al.* (1996*b*), Pierce and Lewis (1997) and Edlund (2002). The seminal approach of Davis (2011) encompasses many previous ideas and adds new ones, such as simulating the factorization symbolically to predetermine the work needed. The enormous progress in the development of computational facilities motivated GPU implementations that can factorize multiple frontal matrices at the same time (Yeralan, Davis, Sid-Lakhdar and Ranka 2017), fine-grained multithreading (Buttari 2013) and exploitation of DAG-based parallelism (Agullo, Buttari, Guermouche and Lopez 2016). A summary of other contributions can be found in the review by Davis *et al.* (2016), while contemporary progress in hierarchical QR factorizations for solving least-squares problems is given in Gnanasekaran and Darve (2022). Here we present a short introduction to the multifrontal QR algorithm, which shares many ideas and concepts with the multifrontal Cholesky factorization. In practice, the columns of A are preordered to preserve sparsity, using $\mathcal{S}(A^\top A)$; the permutation is omitted here to simplify the notation.

Supernodes (or, for efficiency, relaxed supernodes) and the postordered assembly tree for $A^\top A$ (which can be computed without explicitly forming $A^\top A$ (Gilbert,

Algorithm 4.2. Multifrontal QR factorization
Input: Matrix A of full column rank and the postordered assembly tree of $A^\top A$.
Output: Upper triangular factor R of the QR factorization, orthogonal transformations used to transform A stored implicitly or as their product Q.

1: **for** $js = 1 : ns$ **do** ▷ Follow the postordering of the tree
2: Assemble $F(js)$ using rows of A for which the index of the first non-zero entry belongs to js and QR contribution blocks from children of js
3: Compute QR factorization of $F(js)$ ▷ Results in block row of R and contribution block $R(js)$
4: Push $R(js)$ onto the stack. ▷ $R(js)$ will be popped from the stack when assembling $F(\text{parent}(js))$
5: **end for**

Li, Ng and Peyton 2001)) are exploited. For each supernode js, a small dense rectangular matrix $F(js)$, called a QR frontal matrix, is created. It comprises the rows of A for which the index of the first non-zero entry belongs to js together with the contribution blocks from the children of js in the postordered assembly tree. The contribution block $R(ks)$ from child ks is the upper triangular factor obtained by the QR factorization of $F(ks)$. It is not necessary to complete the factorization of a QR frontal matrix to transform it to an upper triangular matrix. For $F(js)$, it is sufficient to perform $|js|$ steps to compute $|js|$ rows of R (where $|js|$ is the number of columns in supernode js). However, computing its QR factorization fully can significantly reduce workspace requirements (Amestoy *et al.* 1996*b*). The multifrontal QR approach is summarized as Algorithm 4.2. Note that the orthogonal transformations computed in the algorithm must always be extended to be of order equal to the row dimension of A.

Figure 4.1 illustrates the QR multifrontal method applied to a matrix with the following sparsity pattern:

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} * & & * & & & & & \\ * & & & & & & * & \\ & * & & * & & * & & \\ & * & & & & * & & \\ & & & * & & * & & \\ * & & & & & * & & \\ & & & & * & & & * \\ & & & & & * & & * \\ & & & & & & * & * \\ & & * & * & & & & \end{pmatrix} \end{array}. \tag{4.5}$$

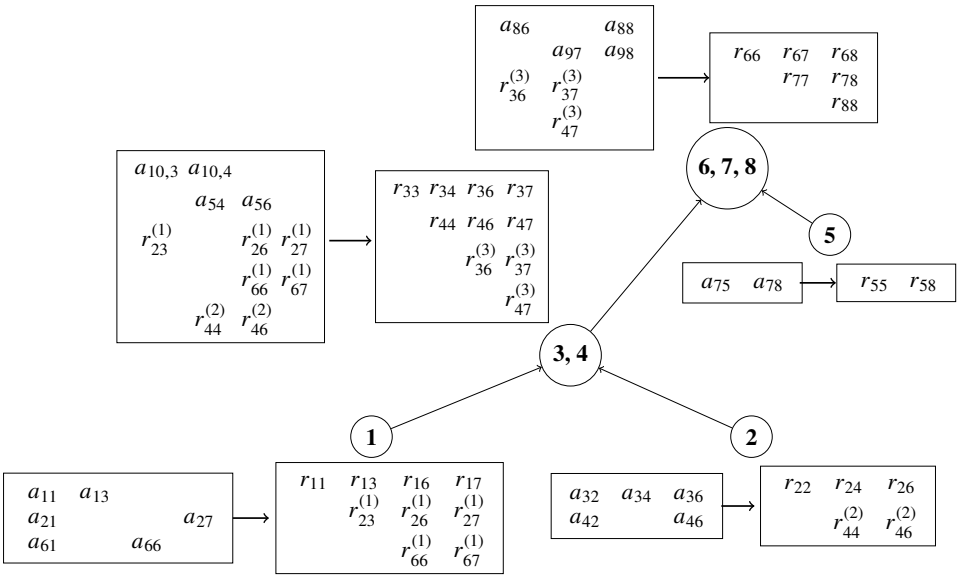


Figure 4.1. The QR multifrontal method applied to the matrix $A = \{a_{ij}\}$ from (4.5). Each vertex shows the transformation from its QR frontal matrix $F(js)$ to the entries of the upper triangular factor $R = \{r_{ij}\}$ of the QR factorization and the QR contribution block $R(js)$ that is passed from child to parent. To illustrate the method, global indices are used for the entries of the QR contribution blocks; the superscripts indicate which supernode the QR contribution comes from.

Here $\mathcal{S}(A^\top A)$ is the same as the sparsity pattern of the matrix C in Figure 3.1 that was used to demonstrate the multifrontal Cholesky factorization and so the assembly tree is as in Figure 3.2. Entries belonging to the QR contribution blocks are indicated using superscripts. We do not show the transformations that can be composed to obtain the Q factor or how to store it implicitly.

4.5. QR factorization when A is rank-deficient

The QR factorization is backward stable, but if A is (close to) rank-deficient then the computed R factor is ill-conditioned. This usually leads to the computed least-squares solution having a very large norm. If $\operatorname{rank}(A) = rk < n$ then theoretically there is a column permutation matrix P (which is not necessarily unique) and an orthogonal matrix Q such that

$$AP = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix},$$

where $R_{11} \in \mathbb{R}^{rk \times rk}$ is upper triangular with positive diagonal entries. This is

column-pivoted QR (CPQR). In place of (4.3), the iteration formula becomes

$$A^{(j)} = H^{(j)} A^{(j-1)} P^{(j)},$$

where $P^{(j)}$ is the permutation matrix that swaps column j of $A^{(j-1)}$ with the column $k \geq j$ that has the largest 2-norm. The column k is called the pivot column. The computed factorization is

$$H^{(n)} \dots H^{(2)} H^{(1)} AP = A^{(n)} = R, \quad P = P^{(1)} P^{(2)} \dots P^{(n-1)}.$$

The first rk columns of AP are linearly independent and the least-squares solution x can be obtained from

$$R_{11} y_1 = d_1 - R_{12} y_2, \quad x = P^\top \begin{pmatrix} y_1^\top & y_2^\top \end{pmatrix}^\top,$$

where

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = Q^\top \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \quad b_1, d_1 \in \mathbb{R}^{rk},$$

and $y_2 \in \mathbb{R}^{n-rk}$ is an arbitrary vector (Golub 1965, Björck 2024). In finite precision arithmetic, the R factor generally does not have zeros on its diagonal, even if A is rank-deficient. Instead, the computed factorization is of the form

$$AP = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}.$$

Although the rank may be revealed by the presence of small diagonal entries, this does not imply that the rest of the rows are negligible. A diagonal entry $r_{rk+1,rk+1}$ of large absolute value computed after rk steps of the QR factorization may hide the fact that the rank of A is rk. Handling rank deficiency is an important component of approaches based on orthogonal factorizations for solving sparse least-squares problems (Heath 1982, Ng 1991, Davis 2011).

Let the singular values $\sigma_i(A)$ of A be ordered in decreasing order and let $\sigma_{rk}(R_{11})$ be the smallest singular value of the first rk columns of AP and $\sigma_1(R_{22})$ be the largest singular value of R_{22}. Then the factorization is said to be a rank-revealing QR (RRQR) factorization if

$$\sigma_{rk}(R_{11}) \geq \sigma_{rk}(A)/c \quad \text{and} \quad \sigma_1(R_{22}) \leq c\, \sigma_{rk+1}(A),$$

where $c = c(k, n) > 0$ is bounded by a low-degree polynomial in rk and n (Björck 2024). RRQR algorithms can be classified by whether they seek to find $\sigma_{rk}(R_{11}) \approx \sigma_{rk}(A)$ or $\sigma_1(R_{22}) \approx \sigma_{rk+1}(A)$ (Chandrasekaran and Ipsen 1994). There has been significant research into theory and algorithms for RRQR factorizations. This includes seeking to detect columns that can be considered as redundant (linearly dependent on the others) (Foster 1986, Pierce and Lewis 1997), postprocessing the computed QR factorization (Chan 1987), improving the bounds on the quality of RRQR factorizations (Hong and Pan 1992) and communication-avoiding RRQR factorizations (Demmel, Grigori, Gu and Xiang 2015).

4.6. *Techniques for column-pivoted QR*

Making classical QR factorization algorithms efficient on modern hardware is challenging, and incorporating pivoting typically incurs significant overheads. The problem is that it involves a sequence of $n-1$ rank-one updates (BLAS 2 operations), making it communication-intensive. In principle, this can be resolved by blocking. Let nb denote a chosen block size. In a blocked Householder QR algorithm, nb pivot columns are chosen and nb Householder reflectors are computed, allowing updates to the remainder of the matrix to be performed using BLAS 3 operations. Many techniques for blocking Householder QR have been proposed, including, for example, those of Bischof and Hansen (1991, 1992).

Randomized sampling can be used to overcome the problem of determining subsets of pivot columns. The key observation is that a measure of the quality for a subset of pivot columns is its spanning volume in $\mathbb{R}^m$ (defined as the product of the singular values of the matrix defined by these columns); this volume should be maximal (Çivril and Magdon-Ismail 2009). This criterion is closely related to how well the subset of columns represents the column space of A, which is a problem that is well suited to randomized sampling. Consider the task of identifying a subset of nb pivot columns in the first step of the blocked QR process. Start by choosing a small oversampling parameter p (typically $p = 10$). Then draw a Gaussian random matrix Ω of size $(nb + p) \times m$, and form a sampling matrix $Y = \Omega A$. Next, perform classical CPQR on the columns of Y, which is inexpensive because Y is small and fits into fast memory. This determines $P^{(1)}$. The process is then repeated. To maximize performance, it is possible to update the sample matrix used in the first step, which obviates the need to draw a new random matrix at each stage and renders the overhead cost induced by randomization almost negligible. The resulting algorithm is Householder QR with randomization for pivoting (HQRRP) (Martinsson, Quintana-Ortí, Heavner and van de Geijn 2017). The use of random sampling to obtain rank-revealing matrix factorizations is discussed in Duersch and Gu (2020), and Martinsson and Tropp (2020) provides a survey of randomized methods in numerical linear algebra.

4.7. *QR for strongly overdetermined systems*

The least-squares problem with $A \in \mathbb{R}^{m \times n}$ is said to be strongly overdetermined if $m \gg n$. Matrices that have many more rows than columns are often referred to as tall-and-skinny (TS). They commonly arise in big data applications with billions of data points and only a few hundred descriptors. TSQR algorithms (Demmel, Grigori, Hoemmen and Langou 2012, Benson, Gleich and Demmel 2013) are numerically stable, efficient, communication-avoiding parallel approaches to computing the QR factorization of TS matrices. They can be used as panel factorizations within a square QR factorization. The first stage partitions the rows of the TS matrix into (non-overlapping) blocks A_i and then computes the QR factorization of each row block, i.e. $A_i = Q_i R_i$, $i = 1, 2, \ldots, N$. Subsequent stages merge the resulting

upper triangular matrices in a divide-and-conquer fashion until a single factor R is obtained. This requires about $\log_2 N$ stages. To illustrate this, let $N = 4$. After the first stage,

$$A = \begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix} = \begin{pmatrix} Q_1 & & & \\ & Q_2 & & \\ & & Q_3 & \\ & & & Q_4 \end{pmatrix} \begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{pmatrix}.$$

The second step stacks the upper triangular factors in pairs and computes $N/2$ factorizations

$$\begin{pmatrix} \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} \\ \begin{pmatrix} R_3 \\ R_4 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} Q_{1,2} & \\ & Q_{3,4} \end{pmatrix} \begin{pmatrix} R_{1,2} \\ R_{2,3} \end{pmatrix}.$$

Finally,

$$\begin{pmatrix} R_{1,2} \\ R_{2,3} \end{pmatrix} = Q_{1,2,3,4}\, R.$$

In general, the combination process informs a tree in which the A_i are the leaf vertices and the final R is the root. Combining in pairs as in the example corresponds to a binary tree. More generally, the tree can be chosen to minimize communication between processors or the volume of memory traffic between the main memory and the cache memory of each processor. Note that the row blocks A_i are generally not sparse and TSQR algorithms target large strongly overdetermined dense matrices. Furthermore, while the TSQR approach is more communication-efficient than the standard Householder algorithm for the QR factorization of highly overdetermined matrices, it produces a different representation of the orthogonal factor, and this requires additional software development (Ballard *et al.* 2015).

Another communication-avoiding QR algorithm for TS matrices is the Cholesky-QR algorithm. If $A^\top A = R^\top R$ is the Cholesky factorization of the normal matrix then the Cholesky-QR algorithm computes $Q_1 = AR^{-1}$ by block forward substitution giving $A = Q_1 R$. This is attractive in terms of high performance computing as it requires only one global reduction between parallel processing units, and most of the computational work can be performed using matrix–matrix operations. However, the method is rarely used in practice because the loss of orthogonality $\|I - Q_1^\top Q_1\|_F$ is $O(\kappa_2(A)^2)$ and breakdown can occur even when A has full numerical rank. To overcome this, the modified Cholesky-QR2 algorithm refines the R and Q_1 factors. The matrix $Q_1^\top Q_1$ is formed and then its Cholesky factorization $Q_1^\top Q_1 = S^\top S$ computed. The refined factorization is taken to be $A = \hat{Q}\hat{R}$, where $\hat{Q} = Q_1 S^{-1}$ and $\hat{R} = SR$. Provided the initial Cholesky factorization does not break down, this approach can be shown to have good stability properties (Yamamoto, Nakatsukasa, Yanagisawa and Fukaya 2015). For ill-conditioned matrices, a possible strategy is to use the LU factors of A to precondition before computing a

Cholesky factorization (Terao, Ozaki and Ogita 2020). Recently, a new approach that combines randomized preconditioning, column pivoting and Cholesky-QR has been proposed (Melnichenko *et al.* 2024).

4.8. Software for sparse QR factorizations

There are many challenges to address in designing and developing efficient and robust QR software for sparse matrices, but a number of library-quality packages that implement the QR multifrontal method are available. The serial QR package `MA49` was developed in the 1990s (Amestoy *et al.* 1996*b*) and is part of the HSL library. More recently, there are the parallel solvers SuiteSparseQR (Davis 2011, Yeralan *et al.* 2017),[17,18] qr_mumps (Buttari 2013)[19] and the Intel Math Kernel Library Sparse QR. These are general-purpose codes that incorporate numerical pivoting. The RRQR approach used in SuiteSparseQR sets to zero the whole row of R if the diagonal entry is less than some tolerance. While this may not determine the rank of A exactly, it is efficient because the changes in the precomputed sparsity pattern of R are straightforward. SuiteSparseQR offers an option to compute minimum-norm solutions to full-rank underdetermined least-squares problems (but not rank-deficient problems).

Within the cuSolver library,[20] cuSolverSP provides routines for sparse QR factorizations. For matrices whose sparsity pattern is not good for exploiting parallelism, there is a CPU option; for those with abundant parallelism potential, the GPU option delivers higher performance. cuSolver also offers a refactorization package cuSolverRF. This can give good performance when solving a sequence of problems where the coefficients of A change but the sparsity pattern remains the same.

Limited implementations of TSQR are available. In MATLAB, the parallel dataflow programming and execution framework `MapReduce`[21] (Constantine and Gleich 2011) can be used to compute a TSQR factorization of highly overdetermined matrices. SLEPc[22] is a software library for the solution of large-scale sparse eigenvalue problems on parallel computers; it includes TSQR.

5. Direct methods for the augmented system formulation

Recall from (1.9) that the linear least-squares problem is mathematically equivalent to the non-singular symmetric indefinite linear system

$$K \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \quad \text{with} \quad K = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}, \quad r = b - Ax. \tag{5.1}$$

[17] https://faculty.cse.tamu.edu/davis/suitesparse.html
[18] https://github.com/DrTimothyAldenDavis/SuiteSparse
[19] https://gitlab.com/qr_mumps/qr_mumps
[20] https://docs.nvidia.com/cuda/cusolver/
[21] https://github.com/arbenson/mrtsqr
[22] https://slepc.upv.es/

Introducing a scaling parameter $\beta > 0$, this is equivalent to

$$K_\beta \begin{pmatrix} \beta^{-1} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \quad \text{with} \quad K_\beta = \begin{pmatrix} \beta I & A \\ A^\top & 0 \end{pmatrix}, \tag{5.2}$$

and so solving this augmented system also gives the solution to the original least-squares problem. It was shown (Björck 1967*a*) that if the singular values of A are σ_i $(1 \le i \le n)$, then the $m + n$ eigenvalues of K_β are

$$\lambda = \begin{cases} \dfrac{\beta}{2} \pm \sqrt{\dfrac{\beta^2}{4} + \sigma_i^2}, & i = 1, \ldots, n, \\ \beta, & \text{otherwise.} \end{cases}$$

If $\sigma_n > 0$, then $\min_\beta \kappa_2(K_\beta) \approx \sqrt{2}\kappa_2(A)$ is attained when $\beta = \sigma_n/\sqrt{2}$. If $\text{rank}(A) = rk \le n$, then the eigenvalue β has multiplicity $m - rk$, and 0 is an eigenvalue of multiplicity $n - rk$. The conditioning of K_β varies significantly with β: K_β can be larger than $\kappa_2^2(A)$ or smaller than $2\kappa_2(A)$. With an appropriate choice of β, the conditioning of K_β is much better than for the normal matrix. An automatic technique for selecting β is proposed in Arioli, Duff and de Rijk (1989), where it was demonstrated that the use of a direct method applied to the augmented system formulation is an accurate and efficient approach to solving sparse least-squares problems.

In addition to the conditioning, there are a number of other reasons for opting to solve the augmented system. The most obvious is to avoid the difficulties associated with using the normal matrix. In particular, $C = A^\top A$ can be much denser than A (and hence K) and so a sparse Cholesky factorization may be impractical (sparse–dense problems are discussed further in Section 8). Even if C is not explicitly formed, a solution approach based on factorizing C can still be expensive because C is used implicitly (with entries computed as they are needed). Moreover, iterative refinement with the Cholesky factors of C may not recover full precision in the least-squares solution (Section 7 considers iterative refinement).

There exist sophisticated approaches to computing LDLT factorizations of symmetric indefinite matrices that incorporate pivoting for numerical stability. Note also that the reduction of K to the normal equations corresponds to the explicit elimination of the (1,1) block, that is, choosing the m diagonal entries of this block as the first m pivots in the factorization of K. It follows that systematic pivoting at all steps of an LDLT factorization (based only on the sparsity pattern) will generally lead to the fill-in being less than in the Cholesky factor (Tůma 2002).

5.1. LDLT factorizations using threshold pivoting

In this section, the focus is on the factorization of general sparse symmetric indefinite matrices. Factorizing such matrices is challenging because of the need to maintain numerical stability. The Cholesky factorization of a symmetric positive definite matrix takes the pivots in order from the main diagonal, and the fill in the

computed factor is as predicted from the sparsity pattern of the matrix. This is not the case for indefinite matrices. Modifications to the pivot (elimination) order are needed and this generally leads to extra fill in the factors. Moreover, incorporating pivoting strategies can limit the potential for parallelism and can be associated with significant data movement that hinders the scalability of the methods.

Let B be a general sparse symmetric indefinite matrix and, to simplify the notation, we assume that B has been prescaled for numerical stability and preordered using a fill-reducing ordering. The ordering algorithm normally assumes that all the diagonal entries b_{ii} of B are non-zero, that is, the sparsity pattern $\mathcal{S}(B + I)$ is used and the same ordering algorithms as for symmetric positive definite matrices are employed (Section 2). Let $B^{(k)}$ denote the kth partially eliminated matrix (i.e. the matrix after the first $k - 1$ elimination operations). If the diagonal entry $b_{kk}^{(k)}$ is zero then it cannot be chosen as the next pivot. Furthermore, if $|b_{kk}^{(k)}|$ is small (relative to the other entries below the diagonal in column k of $B^{(k)}$) then using it as a pivot will lead to large entries in the factor, that is, numerical instability. The growth factor ρ_{growth} is defined to be

$$\rho_{\text{growth}} = \max_{\substack{i,j \geq k \\ 2 \leq k \leq n}} \left(|b_{ij}^{(k)}| \, / \, |b_{ij}| \right).$$

If rows within $B^{(k)}$ are permuted to bring a non-zero off-diagonal entry onto the diagonal then symmetry is destroyed, which means an LU factorization must be performed. This is often not attractive because it essentially doubles the factorization cost in terms of both the storage requirements and operation counts. Extending the notion of a pivot to 2×2 blocks allows symmetry to be preserved. Consider the following matrix:

$$B = \begin{pmatrix} \delta & 1 \\ 1 & 0 \end{pmatrix}.$$

If $\delta = 0$, an LDLT factorization in which D is a diagonal matrix does not exist. Furthermore, if $\delta \ll 1$ then an LDLT factorization with D diagonal is not stable because $\rho_{\text{growth}} = 1/\delta$. However, if the LDLT factorization is generalized to allow D to be a block diagonal matrix with 1×1 and 2×2 blocks, then a factorization can be computed that preserves symmetry and is nearly as stable as an LU factorization. This is illustrated by the factorization of the following 3×3 symmetric indefinite matrix:

$$B = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = LDL^{\top}.$$

Here D has one 1×1 block and one 2×2 block.

For sparse matrices, it is necessary to balance pivoting for stability with limiting the amount of fill-in in the factors. The compromise strategy that seeks to achieve this is called threshold partial pivoting (TPP). Let $\tau > 0$ be a chosen threshold

parameter. Typical values used for factorizing a suitably scaled matrix are 0.1 or 0.01. Limiting the size of the entries of L so that

$$|l_{ij}| \leq \tau^{-1} \tag{5.3}$$

for all i, j, together with a backward stable scheme for solving 2×2 linear systems, suffices to show backward stability for the entire solution process. The stability test for a 1×1 pivot in column t of the active submatrix at stage k is the standard threshold test

$$\max_{i \neq t,\, i \geq k} |b_{it}^{(k)}| \leq \tau^{-1} |b_{tt}^{(k)}|. \tag{5.4}$$

For a 2×2 pivot in rows and columns s and t of $B^{(k)}$, the corresponding test is

$$\left| \begin{pmatrix} b_{ss}^{(k)} & b_{st}^{(k)} \\ b_{st}^{(k)} & b_{tt}^{(k)} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{i \neq s,t; i \geq k} |b_{is}^{(k)}| \\ \max_{i \neq s,t; i \geq k} |b_{it}^{(k)}| \end{pmatrix} \leq \tau^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \tag{5.5}$$

where the absolute value of the matrix is interpreted element-wise (Duff *et al.* 1991). If $b_{tt}^{(k)}$ is accepted as a 1×1 pivot, it becomes the next diagonal entry of D, and row and column t are permuted (if necessary) to the pivotal position k. The corresponding diagonal entry of L is 1, and from the inequality (5.4), the off-diagonal entries of column k of L are bounded in absolute value by τ^{-1}. If

$$\begin{pmatrix} b_{ss}^{(k)} & b_{st}^{(k)} \\ b_{st}^{(k)} & b_{tt}^{(k)} \end{pmatrix}$$

is accepted as a 2×2 pivot, it becomes the next diagonal block of D, and rows and columns s and t are permuted (if necessary) to the next two pivotal positions, k and $k+1$. The corresponding diagonal block of L is the identity matrix of order 2, and inequality (5.5) ensures that the off-diagonal entries of these columns of L are bounded in absolute value by τ^{-1}.

In addition to bounding the size of the entries in L, the ability to stably apply the inverse of D to a vector is required. This is trivially the case for 1×1 pivots, but for 2×2 pivots it is necessary to check that the determinant $|b_{ss}^{(k)} b_{tt}^{(k)} - b_{st}^{(k)} b_{st}^{(k)}|$ is sufficiently large and cancellation does not occur during the application of the inverse.

A major difficulty when TPP tests are incorporated into a sparse matrix factorization algorithm is that, at a particular stage, a pivot satisfying the stability criteria may not exist. As in the Cholesky case, the assembly tree is constructed during the analyse phase of the solver using the sparsity pattern of the matrix (with the diagonal entries assumed to be present). If no stable 1×1 or 2×2 pivot is available within a supernode, then pivot candidates that have not been selected are passed up the assembly tree and eliminated later at an ancestor in the tree. These are known as delayed pivots. Provided $\tau \leq 0.5$, it can be shown that a complete set of pivots can be chosen at the root vertex. Delaying a pivot candidate results in additional

fill-in, more work (in the factorization and solve phases of the solver) and, very importantly, hinders the exploitation of parallelism and so is undesirable.

5.2. *Avoiding delayed pivots*

A number of strategies have been proposed that seek to limit the occurrence of delayed pivots. One possibility if entry b_{ii} is small is to symmetrically permute B before the numerical factorization commences, to put a large off-diagonal entry b_{ij} onto the subdiagonal to give a 2×2 block

$$\begin{pmatrix} b_{ii} & b_{ij} \\ b_{ij} & b_{jj} \end{pmatrix}$$

that is potentially a good 2×2 candidate pivot. This can be achieved using a symmetric matching-based ordering and scaling algorithm (Duff and Pralet 2005, Schenk and Gärtner 2006). A reordering P_1, based on a symmetrized maximum weight matching of B is followed by a fill-reducing reordering P_2 that maintains the diagonal block structure. The matrix can also be scaled by a symmetric scaling S that is also obtained through the weighted matching algorithm. Thus, the factorization algorithm is applied to the matrix $\widehat{B} = P_2 P_1 S B S^{\top} P_1^{\top} P_2^{\top}$. Disadvantages are that computing the matching can be expensive, and as the numerical values of the entries of B are used, if a series of matrices with the same sparsity pattern but different numerical values need to be factorized (such as in a nonlinear least-squares problem) then the whole procedure may have to be rerun for each matrix, potentially adding significantly to the total solution time. Furthermore, the computed ordering may lead to the analyse phase of the direct solver applied to $\mathcal{S}\{\widehat{B}\}$ predicting more entries in the factors than for an ordering computed using nested dissection or minimum degree applied directly to $\mathcal{S}\{B\}$. This can result in the run-time of the factorization and subsequent triangular solves being significantly increased over a standard ordering if little or no pivoting is actually needed. However, for 'tough' indefinite problems for which standard orderings lead to large numbers of delayed pivots, combining a matching-based approach with a numerically aware nested dissection ordering can deliver a significantly lower operation count while also limiting the number of delayed pivots (Hogg, Scott and Thorne 2017).

Static (or restricted) pivoting schemes respect the data structures obtained by the analyse phase (symbolic factorization) that precedes the numerical factorization. The aim is to (closely) follow the pivot order selected in the analyse phase to limit fill-in in the factors and the factorization time at the potential cost of reduced accuracy in the factorization. During the factorization, when a forecast pivot is too small, a prescribed perturbation is added to maintain numerical stability. The computed factors are for a perturbed matrix that is 'close' to the original matrix (Schenk and Gärtner 2006, Duff and Pralet 2007). Relaxed pivoting, in which the threshold parameter τ is increased to allow pivots to be chosen, is another possible

strategy. These ideas are reviewed in Hogg and Scott (2013*c*) and Davis *et al.* (2016). Because such strategies weaken the stability tests, the factorization and hence the computed solution may be less accurate, and it may be necessary to try and improve the solution using iterative refinement (Section 7.2) or by employing the factors as a preconditioner for a Krylov subspace solver. This often works well, but for some very ill-conditioned systems this may not be enough to obtain an accurate solution, so awareness of the potential issues is important.

More recently, *a posteriori* threshold pivoting (APTP) has been proposed (Duff, Hogg and Lopez 2020). APTP involves a fail-in-place approach that keeps the failed columns in place, updates them during the factorization and then handles them at the end of the factorization. Additionally, it uses speculative execution: it speculatively runs a task assuming that no numerical issues have occurred in other tasks that might affect the current one. The overhead is the need for a backup of entries and implementing a backtracking strategy if numerical instability is detected.

5.3. *Refactorization for sequences of linear least-squares problems*

Solving a nonlinear least-squares problem involves solving a sequence of linear least-squares problems (1.25). For each problem, the least-squares matrix $A = \mathcal{J}(x^{(j)})$ has the same sparsity pattern but different values. Thus, unless an ordering that uses the numerical values is employed, it is only necessary to perform the symbolic analysis for the first augmented system. The numerical factorization with pivoting is also performed for the first system. Then, for each subsequent system, if the values of the entries of the matrix have not changed substantially, only the numerical factorization is performed, reusing both the symbolic analysis and, importantly, the pivot sequence obtained for the first system. This reuse of the pivot sequence within the numerical factorization is referred to as refactorization, and is possible because sparse solvers normally offer separate calls to the analyse and factorization phases. Iterative refinement can be employed to improve the accuracy of the computed solution (Section 7.2). Periodically, it may be necessary to recompute the numerical factorization with pivoting to obtain a new pivot sequence.

5.4. *Software for sparse augmented systems*

Since the 1980s, significant effort has gone into developing robust LDLT solvers for symmetric indefinite systems. The HSL library includes a number of packages that are designed for symmetric indefinite systems, most notably the multifrontal codes `MA57` and `HSL_MA97`, and the supernodal DAG-based code `HSL_MA86` (Hogg and Scott 2013*b*). These codes all include the threshold partial pivoting described in Section 5.1. The sparse linear least-squares package `HSL_MA85` calls `HSL_MA97` for (weighted) and optionally regularized least-squares problems. It also handles the case that A contains a small number of dense rows (Section 8). Other well-known parallel sparse direct solvers that support symmetric indefinite

systems include MUMPS, Pardiso and WSMP (Section 3.9). Pardiso combines dense Bunch–Kaufman pivoting (Bunch and Kaufman 1977) with a static pivoting strategy, and for augmented systems the use of a matching-based preordering is recommended. The APTP algorithm is implemented within the SSIDS sparse symmetric indefinite direct solver, which is part of the Sparse Parallel Robust Algorithms Library (SPRAL).[23] SSIDS is a multifrontal solver that is able to exploit GPU devices on heterogeneous CPU-GPU architectures. As with `HSL_MA97`, when used with a bit-compatible BLAS library, SSIDS guarantees bit-compatibility results.

There are standalone routines for computing maximum weighted matchings and scalings. The most well-known is the HSL package `MC64` (and the later version `HSL_MC64`). `HSL_MC80` combines a matching algorithm with a fill-reducing ordering algorithm to compute an elimination order that is suitable for use with a sparse direct solver for general symmetric indefinite systems.

6. Iterative solvers and algebraic preconditioners

6.1. Stationary iterative methods

The earliest mention of using an iterative method to solve a linear system of equations is attributed to the original work of Gauss in the mid-1820s. A brief history of developments in the nineteenth and twentieth centuries is given in Saad and van der Vorst (2000). Let us assume that the normal system of equations is to be solved. The basic idea behind stationary iterative methods is to split the system matrix so that at each iteration we only have to solve a simple linear system. Specifically, the normal matrix is split into

$$A^T A = M - N,$$

where M is non-singular and easy to invert. Starting with an initial guess $x^{(1)}$, the iterations are then given by

$$Mx^{(k+1)} = Nx^{(k)} + A^T b, \quad k = 1, 2, \ldots \tag{6.1}$$

This can be rewritten as

$$x^{(k+1)} = x^{(k)} + M^{-1}A^T(b - Ax^{(k)}) = x^{(k)} + M^{-1}A^T r^{(k)}, \quad k = 1, 2, \ldots, \tag{6.2}$$

where $r^{(k)}$ is the residual vector that corresponds to the approximation solution $x^{(k)}$. Observe that by substituting $b = r^{(k)} + Ax^{(k)}$ into $x = (A^T A)^{-1} A^T b$, we obtain

$$x = (A^T A)^{-1}A^T(r^{(k)} + Ax^{(k)}) = x^{(k)} + (A^T A)^{-1}A^T r^{(k)},$$

and if M is used to approximate $A^T A$, we again get the iteration (6.2). From (6.2),

$$r^{(k+1)} = b - A(x^{(k)} + M^{-1}A^T r^{(k)}) = (I - AM^{-1}A^T)\, r^{(k)} = \cdots = (I - AM^{-1}A^T)^k\, r^{(1)}.$$

[23] https://github.com/ralna/SPRAL

The matrix $G = I - M^{-1}A^\top A = M^{-1}N$ is called the iteration matrix.

Lemma 6.1. For any initial vector $x^{(1)}$ and vector b, the iteration (6.1) converges if and only if the spectral radius of the iteration matrix $G = M^{-1}N$ is less than unity.

In general, it is impractical to compute the spectral radius, and sufficient conditions that guarantee convergence are used. Because $\rho(B) \leq \|B\|$ for any square matrix and any consistent matrix norm, a sufficient condition is $\|G\| < 1$. A small spectral radius leads to rapid convergence and the closer the eigenvalues of $M^{-1}A^\top A$ are to unity, the faster the convergence. However, the eigenvalue distribution (not just the spectral radius) is important in evaluating the rate of convergence. Often it is desirable for the iteration matrix to have real eigenvalues. This is the case if the iterative method is symmetrizable, that is, if there exists a non-singular matrix Y such that $YM^{-1}A^\top AY^{-1} = Y(I - G)Y^{-1}$ is symmetric positive definite.

Several standard stationary methods are obtained from the splitting

$$C = A^\top A = D_C + L_C + U_C,$$

where D_C is a diagonal matrix that represents the diagonal part of C, and L_C and U_C are the strictly lower and upper triangular parts of C, respectively. If $\omega > 0$ is a scalar parameter, classical methods include:

- the relaxed Richardson method, $M = \omega^{-1}I$,
- the Jacobi and damped Jacobi methods, $M = D_C$ and $M = \omega^{-1}D_C$,
- the Gauss–Seidel and SOR methods, $M = D_C + L_C$ and $M = \omega^{-1}D_C + L_C$.

These can be implemented without forming the normal matrix. The Jacobi method is symmetrizable but the Gauss–Seidel method is not. However, there exists a symmetrizable variant of the SOR method called SSOR, and the Gauss–Seidel method, as a special case of the SOR method, can also be symmetrized. Compared to Gauss–Seidel, the Jacobi method can be more readily adapted to parallel computation. Both methods can be generalized to block matrices (Elfving 1980).

For Richardson's method, from (6.1), if $x^{(1)} \in \mathcal{R}(\mathcal{A}^\top)$ then $x^{(k)} \in \mathcal{R}(\mathcal{A}^\top)$ for all $k > 1$. It follows that if A is rank-deficient then Richardson's method converges to the pseudoinverse solution $x = A^\dagger b$. However, in practice rounding errors result in a small error component in $\mathcal{N}(\mathcal{A})$ that grows linearly with k. This is also the case for many other iterative methods for solving the normal equations.

An approach that is related to Richardson's method and has received attention in recent years is the (block) Cimmino method (Benzi 2004). It is potentially attractive because the main computational effort involves independent subproblems, defined by a partitioning of the system matrix into blocks of rows or blocks of columns. These can be solved efficiently in parallel using existing software. Variants of the approach have been proposed but the method generally lacks robustness (Duff, Guivarch, Ruiz and Zenadi 2015, Dumitraşc *et al.* 2018). Recently, extending the

augmented block Cimmino method to full-rank least-squares problems has been proposed (Dumitraşc *et al.* 2021)

The concept of splitting can be applied directly to rectangular matrices A, avoiding the use of the normal equations and its poorer conditioning. Suppose $A = M_A - N_A$. For the splitting to be valid it must be a so-called proper splitting, that is, the range and null spaces of A and M_A must be equal. For a proper splitting, it can be shown that the iteration

$$x^{(k+1)} = M_A^{\dagger}(N_A x^{(k)} + b)$$

converges to the pseudoinverse solution $x = A^{\dagger}b$ for every $x^{(1)}$ if and only if the spectral radius of the iteration matrix is less than unity, i.e. $\rho(M_A^{\dagger}N_A) < 1$ (Berman and Plemmons 1974, Climent and Perea 2003).

6.2. *Krylov subspace methods*

Let $B \in \mathbb{R}^{n\times n}$. Given $v \in \mathbb{R}^n$ ($v \neq 0$), the sequence of vectors $v, Bv, B^2v, B^3v, \ldots$ is called a Krylov sequence and the subspace spanned by the first k vectors is a Krylov subspace, denoted by

$$\mathcal{K}^{(k)}(B, v) = \text{span}\{v, Bv, \ldots, B^{k-1}v\}.$$

Krylov subspace-based methods provide powerful tools for solving linear systems of equations; they form the most widely used class of preconditioned iterative methods. Importantly, for very large problems they do not need the matrix B to be stored; instead, each iteration requires products with B (and, in the general case, with $B^{\top}$). Because Krylov subspace methods build a basis, convergence is achieved in exact arithmetic in at most n iterations. In the presence of rounding errors, this is not guaranteed. If n is large, it is impractical to perform $O(n)$ iterations; the hope is that the process returns a sufficiently accurate solution far earlier. Indeed, in some practical applications, it may only be feasible to perform a very limited number of iterations.

For SPD linear systems, the Krylov subspace method of choice is the classical conjugate gradient (CG) method. It can be shown that the approximate solution $y^{(k)}$ at iteration k computed using the CG method satisfies

$$\|y - y^{(k)}\|_B \leq 2\left(\frac{\sqrt{\kappa_2(B)} - 1}{\sqrt{\kappa_2(B)} + 1}\right)^k \|y - y^{(0)}\|_B, \tag{6.3}$$

where $\|\cdot\|_B$ is the B-norm, and $\kappa_2(B) = \lambda_{\max}/\lambda_{\min}$ is the spectral condition number ($\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of B). Clearly there is good (fast) convergence when $\kappa_2(B)$ is small, but poor (slow) convergence can occur if $\kappa_2(B) \gg 1$ ($\lambda_{\min}$ is close to zero). The error bound (6.3) is often highly pessimistic. It does not show the potential for the CG method to converge superlinearly or that the rate of convergence depends on the distribution of all the eigenvalues of B. In practice, it is not normally possible to obtain detailed spectral information for B.

Algorithm 6.1. CGLS (conjugate gradients for least-squares)
Input: Matrix $A \in \mathbb{R}^{m \times n}$, right-hand side vector b, initial approximation $x^{(1)}$.
Output: least-squares solution x.

$r^{(1)} = b - Ax^{(1)}$, $z^{(1)} = p^{(1)} = A^\top r^{(1)}$, $\gamma^{(1)} = \|z^{(1)}\|_2^2$
for $j = 1, 2, \ldots$ *until* convergence **do**
  $q^{(j)} = Ap^{(j)}$
  $\alpha^{(j)} = \gamma^{(j)} / \|q^{(j)}\|_2^2$
  $x^{(j+1)} = x^{(j)} + \alpha^{(j)} p^{(j)}$
  $r^{(j+1)} = r^{(j)} - \alpha^{(j)} q^{(j)}$
  $z^{(j+1)} = A^\top r^{(j+1)}$
  $\gamma^{(j+1)} = \|z^{(j+1)}\|_2^2$
  $\beta^{(j)} = \gamma^{(j+1)} / \gamma^{(j)}$
  $p^{(j+1)} = z^{(j+1)} + \beta^{(j)} p^{(j)}$
end for

For non-singular indefinite matrices, possible methods are the short-term recurrence methods SYMMLQ and MINRES (MINimal RESidual) (Paige and Saunders 1975). When using preconditioning, the preconditioned system matrix also needs to be symmetric and this generally requires that the preconditioner is SPD. If no good SPD preconditioner is available, or if there is a very good non-symmetric preconditioner, the potential advantage of the symmetry of B is lost and it may be necessary to use a solver for non-symmetric matrices. In this case, the most widely used Krylov subspace method is GMRES (Generalized Minimal RESidual) (Saad and Schultz 1986). GMRES has the disadvantage of not being a short-term recurrence method and it may be necessary to incorporate a restarting strategy to limit the number of vectors that must be held and the work involved in each iteration.

Krylov subspaces with $B = A^\top A$ or $B = AA^\top$ play a fundamental role in non-stationary iterative methods for solving large-scale least-squares problems. If A is ill-conditioned then the CG method applied naively to the normal equations will generally perform poorly. CGLS (Hestenes and Stiefel 1952), which is derived by a slight algebraic rearrangement of the CG method, has better numerical properties, at the expense of a small amount of additional storage and work per iteration. Specifically, CGLS avoids explicit formation of the normal matrix and the residual $r = b - Ax$ is recurred (rather than the residual $A^\top r$ of the normal equations); at each iteration $\|r\|_2$ is minimized. The method is outlined in Algorithm 6.1. Each iteration requires one matrix–vector multiplication with A and one with $A^\top$.

Recall that if $\text{rank}(A) < n$, the least-squares solution is not unique. However, it is straightforward to verify that if $x^{(1)} \in \mathcal{R}(\mathcal{A}^\top)$ (e.g. $x^{(1)} = 0$) then $x^{(j)} \in \mathcal{R}(\mathcal{A}^\top)$, $j = 0, 1, \ldots$. Hence, in exact arithmetic, CGLS converges to the pseudoinverse

Algorithm 6.2. *k*-**step GK bidiagonalization (for LSQR)**
Input: $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.
Output: Orthonormal bases $\{q^{(1)}, q^{(2)}, \ldots, q^{(k)}\}$ and $\{p^{(1)}, p^{(2)}, \ldots, p^{(k)}\}$ for $\mathcal{K}^{(k)}(AA^\top, q^{(1)})$ and $\mathcal{K}^{(k)}(A^\top A, p^{(1)})$, respectively.

$\beta^{(1)} = \|b\|_2$, $q^{(1)} = b/\beta^{(1)}$, $\gamma^{(1)} = \|A^\top q^{(1)}\|_2$, $p^{(1)} = A^\top q^{(1)}/\gamma^{(1)}$
for $j = 1, 2, \ldots, k$ **do**
  $s^{(j)} = Ap^{(j)} - \gamma^{(j)} q^{(j)}$
  $\beta^{(j+1)} = \|s^{(j)}\|_2$
  $q^{(j+1)} = s^{(j)}/\beta^{(j+1)}$
  $r^{(j)} = A^\top q^{(j+1)} - \beta^{(j+1)} p^{(j)}$
  $\gamma^{(j+1)} = \|r^{(j)}\|_2$
  $p^{(j+1)} = r^{(j)}/\gamma^{(j+1)}$
end for

solution $x = A^\dagger b \in \mathcal{R}(\mathcal{A}^\top)$ and, in theory, CGLS can be employed to solve least-squares problems of any rank, either overdetermined or underdetermined.

LSQR (Paige and Saunders 1982) is also algebraically equivalent to applying CG to the normal equations. It is popular because it should be more reliable for ill-conditioned problems, again at the cost of extra storage and work per iteration (although as with CGLS, each iteration requires one multiplication with A and one with $A^\top$ and these are frequently the dominant cost). LSQR is based on the Golub–Kahan (GK) bidiagonalization of A (sometimes called Golub–Kahan–Lanczos bidiagonalization). Again, let $x^{(1)}$ be an initial approximation to the solution with initial residual $r^{(1)} = b - Ax^{(1)}$. Generically, $k \ll \min(m, n)$ steps of GK bidiagonalization determine orthonormal bases $\{q^{(1)}, q^{(2)}, \ldots, q^{(k)}\}$ and $\{p^{(1)}, p^{(2)}, \ldots, p^{(k)}\}$ for the Krylov subspaces

$$\mathcal{K}^{(k)}(AA^\top, q^{(1)}) = \text{span}\{q^{(1)}, AA^\top q^{(1)}, \ldots, (AA^\top)^{k-1} q^{(1)}\},$$
$$\mathcal{K}^{(k)}(A^\top A, p^{(1)}) = \text{span}\{p^{(1)}, A^\top A\, p^{(1)}, \ldots, (A^\top A)^{k-1} p^{(1)}\},$$

respectively, with initial vectors $q^{(1)} = r^{(1)}/\|r^{(1)}\|_2$ and $p^{(1)} = A^\top q^{(1)}/\|A^\top q^{(1)}\|_2$. With $x^{(1)} = 0$, the k-step GK bidiagonalization procedure is given in Algorithm 6.2.

In exact arithmetic, if $B^{(k)}$ is the lower bidiagonal matrix with $\gamma^{(1)}, \gamma^{(2)}, \ldots, \gamma^{(k)}$ on the diagonal and $\beta^{(2)}, \beta^{(3)}, \ldots, \beta^{(k+1)}$ on the subdiagonal, the bidiagonalization can be written in matrix form as

$$Q^{(k+1)}\beta^{(1)} e_1 = b,$$
$$AP^{(k)} = Q^{(k+1)} B^{(k)},$$
$$A^\top Q^{(k+1)} = P^{(k)} (B^{(k)})^\top + \gamma^{(k+1)} p^{(k+1)} e_{k+1}^\top,$$

where the columns of $P^{(k)}$ and $Q^{(k+1)}$ are

$$\{p^{(1)}, p^{(2)}, \ldots, p^{(k)}\} \quad \text{and} \quad \{q^{(1)}, q^{(2)}, \ldots, q^{(k+1)}\},$$

respectively. It follows that

$$\min_{x=P^{(k)}y} \|b - Ax\|_2 = \min_{y\in\mathbb{R}^k} \|\beta^{(1)}e_1 - B^{(k)}y\|_2,$$

and the k-step LSQR solution is

$$x^{(k)} = P^{(k)}y^{(k)}, \quad y^{(k)} = \arg\min_{y\in\mathbb{R}^k} \|\beta^{(1)}e_1 - B^{(k)}y\|_2 = (B^{(k)})^{\dagger}\beta^{(1)}e_1.$$

Practical implementations use a recursive formula to obtain $x^{(k+1)}$ from $x^{(k)}$, avoiding solving the projected least-squares problems at each iteration.

LSQR was generalized by Benbow (1999) to use a non-standard inner product to solve the weighted least-squares problem

$$\min_x \|b - Ax\|_{W^{-1}},$$

where W is SPD. The only changes needed are in lines 1 and 6 of Algorithm 6.2, where $A^{\top}q^{(1)}$ and $A^{\top}q^{(j+1)}$ are replaced by $A^{\top}W^{-1}q^{(1)}$ and $A^{\top}W^{-1}q^{(j+1)}$, and in line 5, $\beta = \|s^{(j)}\|_{W^{-1}}$.

For least-squares problems with a well-conditioned matrix A, LSQR often converges quickly, yielding an approximation to the solution with the desired accuracy long before linear dependence is encountered in the Krylov subspaces. However, when A is ill-conditioned, LSQR (and CGLS) can require prohibitively many iterations. A contributing reason is that in finite precision arithmetic, storing and using only a few basis vectors at a time cannot maintain orthogonality among all previously non-stored basis vectors. This can be overcome by keeping previously computed basis vectors and reorthogonalizing. However, for large k, this can be too computationally expensive and require too much storage to be practical. Solutions include partial reorthogonalization and implicit restarting (Baglama and Richmond 2014), or it can be sufficient to orthogonalize only one set of the column vectors (Barlow 2013).

Both CGLS and LSQR compute an approximate solution by minimizing

$$\|r^{(k)}\|_2 = \|b - Ax^{(k)}\|_2 \quad \text{for } x^{(k)} \in x^{(1)} + \mathcal{K}^{(k)}(A^{\top}A, p^{(1)}).$$

The associated residual vector $r^{(k)} = b - Ax^{(k)}$ lies in $\mathcal{K}^{(k)}(AA^{\top}, q^{(1)})$ and the norm $\|r^{(k)}\|_2$ reduces monotonically. For underdetermined systems, LSQR solves the problem $\min \|x\|_2$ subject to $Ax = b$. More generally, it solves $\min \|x\|_2$ subject to $A^{\top}Ax = A^{\top}b$, where A may have any shape or rank.

For underdetermined systems, CGNE (or Craig's method (Craig 1955)) is the CG method implicitly applied to the problem $AA^{\top}z = b$ with $x = A^{\top}z$. The error $\|x - x^{(k)}\|_2$ decreases monotonically but the residual $\|r^{(k)}\|_2$ can oscillate. Because the stopping criterion for consistent systems is usually based on the size of $\|r^{(k)}\|_2$,

it may be preferable to use CGLS for underdetermined systems. However, for ill-conditioned noisy problems, CGLS can behave poorly (Hnětynková, Kubínová and Plešinger 2017).

Within CGLS and LSQR, the residuals $\|r^{(k)}\|_2$ decrease monotonically but, in general, the normal equation residuals $\|A^\top r^{(k)}\|_2$ oscillate. When A is ill-conditioned, the oscillations can be large. This behaviour is undesirable, because practical stopping criteria for least-squares problems use $\|A^\top r^{(k)}\|_2$. Specifically, the iterations may be terminated when

$$\|A^\top r^{(k)}\|_2 \leq \eta \left(\|A\|_2 \, \|r^{(k)}\|_2\right), \tag{6.4}$$

where $\eta > 0$ is a prescribed small tolerance. Quantities within the Krylov method can be used for this test. The criterion (6.4) is sufficient to obtain a backward stable solution, but it is not necessary. An alternative based on estimation of the error norm is described in Papež and Tichý (2023); see also references therein.

The LSMR algorithm (Fong and Saunders 2011) is also based on GK bidiagonalization. It is mathematically equivalent to the MINRES method applied to the normal equations, with $\|r^{(k)}\|_2$ and $\|A^\top r^{(k)}\|_2$ decreasing monotonically. This may allow LSMR to follow the convergence more easily and terminate after fewer iterations than CGLS and LSQR. But while CGLS and LSQR are based on minimizing the norm of the residual $\|r^{(k)}\|_2$, at each iteration LSMR minimizes $\|A^\top r^{(k)}\|_2$.

Iterative methods exhibit semi-convergence on least-squares problems originating from discretized ill-posed problems, with the scaled errors $\|x - x^{(k)}\|_2/\|x\|_2$ decreasing initially, but at some point they begin to increase. Terminating the iterations before divergence occurs is an important challenge; see Reichel, Sadok and Zhang (2020).

6.3. *Introduction to algebraic preconditioners*

The rate of convergence of the methods of the last section depends on the condition number of A and on the distribution of its singular values. Convergence may be slow when A has unfavourably distributed singular values and a preconditioner may be needed to try and accelerate convergence. Recall (1.14). The non-singular preconditioner M should be chosen so that:

- $\kappa_2(AM^{-1}) = \sigma_{\max}(AM^{-1})/\sigma_{\min}(AM^{-1})$ is small (clustered singular values) and less than $\kappa_2(A)$,
- matrix–vector products with M^{-1} and $M^{-\top}$ can be performed efficiently.

Consider the stopping criterion (6.4). When preconditioning is used, if the unpreconditioned residual is not available, then terminating when

$$\|(AM^{-1})^\top r^{(k)}\|_2 \leq \eta(\|AM^{-1}\|_2 \, \|r^{(k)}\|_2)$$

is an option. In this case, the stopping criterion is based on the preconditioned problem, not the original one.

The choice of preconditioner is influenced by many factors, including the order of the matrix, its eigenvalues, its conditioning, its sparsity pattern, density and positive definiteness. Finding good preconditioners for least-squares problems is often difficult because they come from a wide range of applications with different characteristics that require different preconditioners (there is no 'one size fits all' preconditioner). The choice can be delicate if we are to avoid accelerating the convergence of those components dominated by amplified errors. Often it is down to user experience, availability of the preconditioner in an accessible form, or even trial-and-error. For least-squares problems, A usually lacks the properties that make preconditioning techniques for linear systems arising from PDEs successful. Hence we use algebraic preconditioners that do not require knowledge of the provenance of the system.

Compared with other classes of linear systems, the development of preconditioners for sparse least-squares problems may be regarded as still being in its infancy. Most algebraic preconditioners are least-squares adaptations of approaches for square linear systems (Bru, Marín, Mas and Tůma 2014, Gould and Scott 2017). In general, it is desirable to avoid explicitly computing the normal matrix and, in some applications, a preconditioner that can work in a matrix-free environment is needed. In this case, the construction of the preconditioner should only involve matrix–vector products with A and $A^\top$ and, ideally, only a small number of such products should be performed. In some large-scale applications where limiting the time for computing the solution is critical (e.g. in numerical weather forecasting), products with A represent the dominant cost and practical algorithms must restrict the number of such products.

An obvious idea is to approximate A by M_R and then employ $M = M_R M_R^\top$ as a split preconditioner for the normal matrix. However, although the singular values of AM_R^{-1} may be favourably distributed and M_R may provide an excellent preconditioner for A, the eigenvalues of the symmetrically preconditioned system $M_R^{-\top} A^\top A M_R^{-1}$ can have an adverse distribution and M can be arbitrarily poor as a preconditioner for $A^\top A$. This was noted in Braess and Peisker (1986) and recently discussed and illustrated in Gratton, Gürol, Simon and Toint (2018) and Wathen (2022); see also the notes on eigenvalues clustering in Liesen and Strakoš (2013).

Recall (1.14). Applying an iterative method such as CGLS, LSQR or LSMR to the right-preconditioned problem

$$\min_{z\in\mathbb{R}^n} \|b - AM_R^{-1} z\|_2, \quad x = M_R^{-1} z, \quad M = M_R M_R^\top,$$

can be performed by replacing matrix–vector products with A and $A^\top$ by products with AM_R^{-1} and $M_R^{-\top} A^\top$ (i.e. the matrix and preconditioner factor are applied together). The overhead per iteration is one solve with M_R and one with $M_R^\top$. This approach is commonly used if the factors of M are available; other possibilities for incorporating preconditioners into Krylov subspace methods are discussed in classical monographs such as that of Saad (2003*b*). Factorization-free preconditioned LSQR and LSMR algorithms are presented in Arridge, Betcke and Harhanen

(2014) and Cerdán, Guerrero, Marín and Mas (2020), respectively. Combining preconditioning with iterative solvers for augmented systems that are quasi-definite is described in Orban and Arioli (2017), including flexible preconditioning of GMRES (the FGMRES approach) that allows the preconditioner to change within the solver. The development of new sophisticated iterative strategies for least-squares problems is on-going (Estrin, Orban and Saunders 2019).

Finally, an interesting alternative approach is given by the AB- and BA-GMRES iterative methods (Hayami, Yin and Ito 2010). AB-GMRES solves

$$\min_{y\in\mathbb{R}^m} \|b - ABy\|_2, \quad x = By,$$

with $B \in \mathbb{R}^{n\times m}$ as a right-preconditioner, while BA-GMRES solves

$$\min_{x\in\mathbb{R}^n} \|Bb - BAx\|_2,$$

with $B \in \mathbb{R}^{n\times m}$ as a left-preconditioner. If $\mathcal{R}(B) = \mathcal{R}(A^\top)$, then AB-GMRES computes the least-squares solution x for all $b \in \mathbb{R}^m$ if and only if $\mathcal{R}^\top(B) = \mathcal{R}(A^\top)$. If $B = A^\top$, then AB-GMRES is mathematically equivalent to LSQR and CGLS. It can also be shown that the problems $\min_x \|b - Ax\|_2$ and $\min_x \|Bb - BAx\|_2$ are mathematically equivalent for all $b \in \mathbb{R}^m$ if and only if $\mathcal{R}(B^\top BA) = \mathcal{R}(A)$, e.g. if $\mathcal{R}(B^\top) = \mathcal{R}(A)$. If $B = A^\top$, then BA-GMRES is mathematically equivalent to LSMR.

In X-ray computed tomography (CT), A is the forward projector and $A^\top$ represents the so-called back projector. For large-scale instances, it is common to use different discretization techniques for the forward and back projectors. This means that if $B \in \mathbb{R}^{n\times m}$ represents the back projector then B is typically not equal to the transpose $A^\top$ of the forward projector; B is termed an unmatched back projector or an unmatched transpose. As a result, instead of the normal equations, the so-called unmatched normal equations in one of the forms

$$ABy = b,\ x = By, \quad \text{or} \quad BAx = Bb, \quad \text{with} \quad B \approx A^\top,$$

are solved. AB-GMRES or BA-GMRES can still be used, but applying LSQR or LSMR is potentially problematic because BA is neither symmetric nor positive semidefinite (Hansen, Hayami and Morikuni 2022).

In the following subsections, we discuss some possible algebraic preconditioners for Krylov subspace methods for solving large sparse least-squares problems.

6.4. Diagonal preconditioning

As observed in Section 1.6, the simplest form of preconditioning is diagonal preconditioning, that is,

$$\min_{z\in\mathbb{R}^n} \|b - ASz\|_2, \quad x = Sz,$$

where S is a diagonal matrix that scales the columns of A to give each unit 2-norm. This requires only the diagonal entries of the normal matrix $A^\top A$ to be computed

or, equivalently, the squares of the 2-norms of the columns of A. Theoretical results show that diagonal scaling is important because it reduces the condition number (Van der Sluis 1969), but it is also important because it can be trivially applied in parallel. It is generally advantageous to first apply diagonal scaling and then, if convergence of the iterative solver is unacceptably slow, to try one or more of the methods outlined in the following sections, applied to the scaled problem. In many situations, block diagonal preconditioners (Elfving 1980) corresponding to block scaling are another efficient way achieving good parallelism.

6.5. *Incomplete Cholesky factorization preconditioners*

Incomplete Cholesky (IC) factorizations approximate the exact Cholesky factorization of a given SPD matrix C by disallowing some of the entries that occur in a complete factorization. Thus $C \approx \widetilde{L}\widetilde{L}^\top$, where the incomplete factor $\widetilde{L}$ is sparse and lower triangular. The split preconditioned normal equations are

$$\widetilde{L}^{-1}A^\top A\widetilde{L}^{-\top}y = \widetilde{L}^{-1}A^\top b, \quad x = \widetilde{L}^{-\top}y.$$

IC factorizations were first introduced for model PDE problems (Buleev 1959) but soon after were considered as a class of general algebraic procedures (Varga 1960). The simplest sparsity pattern allows no entries in $\widetilde{L}$ outside the sparsity pattern of C, i.e. $\mathcal{S}\{\widetilde{L}+\widetilde{L}^\top\} = \mathcal{S}\{C\}$. The resulting factorization is called an $IC(0)$ (or no-fill) factorization. We always assume that $\mathcal{S}\{\widetilde{L}\}$ contains the positions of diagonal entries. Motivation for considering $\mathcal{S}\{\widetilde{L}\}$ that is a superset of $\mathcal{S}\{C\}$ is given by the following straightforward but important result that can be found in Chan and van der Vorst (1997) and van der Vorst (2003).

Lemma 6.2. Consider the incomplete Cholesky factorization $C + E = \widetilde{L}\widetilde{L}^\top$ with sparsity pattern $\mathcal{S}\{\widetilde{L}+\widetilde{L}^\top\}$. The entries of the error matrix E are zero at positions $(i, j) \in \mathcal{S}\{\widetilde{L}+\widetilde{L}^\top\}$.

In practice, sophisticated and systematic ways of extending $\mathcal{S}\{\widetilde{L}\}$ are necessary to obtain robust high-quality preconditioners. An early choice for banded C, motivated by the simple discretization of a PDE on a rectangular grid, allows $\mathcal{S}\{\widetilde{L}+\widetilde{L}^\top\}$ to include fill-in along a few additional diagonals within the band. This can be extended to more general systems using the concept of levels (Watts III 1981). Entries of $\widetilde{L}$ that correspond to non-zero entries of C are assigned the level 0 while each potential filled entry in position (i, j) is assigned a level as follows:

$$\text{level}(i, j) = \min_{1\le k<\min\{i,j\}} (\text{level}(i, k) + \text{level}(k, j) + 1). \tag{6.5}$$

Given $\ell \ge 0$, during the factorization a filled entry is permitted at position (i, j) provided $\text{level}(i, j) \le \ell$. It can be shown that using levels can be interpreted as allowing fill-paths of limited length (Hysom and Pothen 2002). In particular, $\text{level}(i, j) = k$ for some $k \le \ell$ if and only if there is a shortest fill-path between i and j of length $k + 1$ in the adjacency graph $\mathcal{G}(C)$. A weakness of the resulting

$IC(\ell)$ preconditioner is that the number of entries in the incomplete factor can grow quickly with ℓ and only small values of ℓ are practical.

Threshold-based incomplete factorizations determine the locations of permissible fill-in in conjunction with the numerical factorization of C. Entries of $\widetilde{L}$ of absolute value smaller than a prescribed threshold $\tau > 0$ are dropped as they are computed. Obtaining a good preconditioner is highly sensitive to the choice of τ, and this is problem-dependent and influenced by the scaling of C. Memory-based methods prescribe the amount of memory available for the incomplete factorization, and only the largest entries in each row (or column) are retained. A practical implementation of this kind can be found in Jones and Plassmann (1995). Many refinements, variants and hybrids of the different approaches have been proposed; see, for example, Scott and Tůma (2011) for a brief historical overview.

A more sophisticated class of schemes employs additional memory during the construction of the incomplete factors that is then discarded. The aim is to obtain a high-quality preconditioner while maintaining sparsity and allowing the user to control how much memory is used (Scott and Tůma 2014*b*). Consider the decomposition

$$C = (\widetilde{L} + \widetilde{T})\,(\widetilde{L} + \widetilde{T})^{\top} - E, \tag{6.6}$$

where the incomplete factor $\widetilde{L}$ is a lower triangular matrix with positive diagonal entries, $\widetilde{T}$ is a strictly lower triangular matrix and the error matrix is $E = \widetilde{T}\widetilde{T}^{\top}$. At each step, the next column of $\widetilde{L}$ and of $\widetilde{T}$ is computed and then the remaining Schur complement is modified. On step j, the first column of the Schur complement is split into the sum $\widetilde{L}_{j:n,j} + \widetilde{T}_{j:n,j}$, where $\widetilde{L}_{j:n,j}$ contains the entries that are retained in column j of the final incomplete factor, $\widetilde{T}_{jj} = 0$, and $\widetilde{T}_{j+1:n,j}$ contains the entries that are not included in $\widetilde{L}$. In a complete factorization, the Schur complement would be updated by subtracting

$$(\widetilde{L}_{j+1:n,j} + \widetilde{T}_{j+1:n,j})\,(\widetilde{L}_{j+1:n,j} + \widetilde{T}_{j+1:n,j})^{\top}.$$

However, the incomplete factorization discards the term

$$E^{(j)} = \widetilde{T}_{j+1:n,j}\,\widetilde{T}_{j+1:n,j}^{\top}$$

that would be subtracted in complete factorization. Thus the matrix $E^{(j)}$ is implicitly added to C, and because $E^{(j)}$ is positive semidefinite, the approach does not break down.

An obvious choice is for the largest entries in the column to be retained in $\widetilde{L}$. Figure 6.1 depicts the first step $j = 1$. In the first row and column, $*$ and δ denote the entries of $\widetilde{L}_{1:n,1}$ and $\widetilde{T}_{1:n,1}$, respectively. Because a standard sparsification scheme does not store the smallest entries, using such a scheme gives no fill-in in the rows and columns corresponding to the discarded entries; this is shown in Figure 6.1(a). The fill-in in the factorization that uses intermediate memory is depicted in Figure 6.1(b). Clearly, more fill entries are used in constructing L than

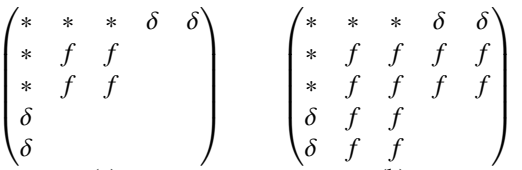


Figure 6.1. An illustration of the fill-in in a standard sparsification-based incomplete factorization (a) and in the approach that uses intermediate memory (b) after one step of the factorization. Entries with small absolute value in row and column 1 are denoted by δ. The filled entries are denoted by f.

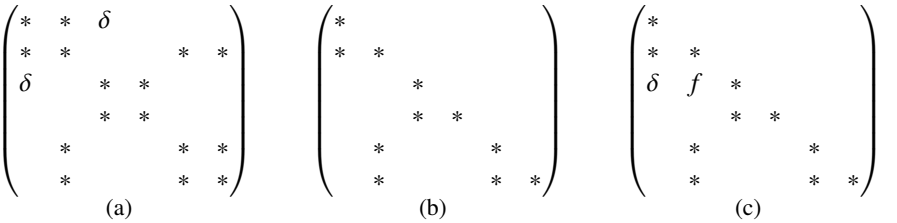


Figure 6.2. (a) An SPD matrix with an entry of small absolute value in positions $(1,3)$ and $(3,1)$. (b) $\mathcal{S}\{\widetilde{L}\}$ computed using a standard incomplete factorization that drops the small entry δ at position $(3,1)$ (there are no filled entries in this case). (c) The partially computed $\mathcal{S}\{\widetilde{L}+\widetilde{T}\}$ after the first step of the incomplete factorization using intermediate memory. The filled entry is denoted by f.

in the standard factorization and the structure of the complete factorization can be followed more closely. This is illustrated in Figure 6.2. If the small entries at positions $(1,3)$ and $(3,1)$ are not discarded then there is a fill entry in position $(3,2)$ and this allows the incomplete factorization using intermediate memory to involve the (large) off-diagonal entries in positions $(5,2)$ and $(6,2)$ in the second step of the incomplete factorization.

The columns of $\widetilde{T}$ must be held until the end of the factorization, independently of the order of operations used by the implementation. A practical solution to decrease the computational complexity and reduce the memory costs is to sacrifice the breakdown-free property and combine the factorization with the threshold-based approach, dropping entries of small absolute value from $\widetilde{L}$ and $\widetilde{T}$. In addition, a limit on the number of entries allowed in each column of $\widetilde{L}$ and $\widetilde{T}$ can be fixed in advance. Algorithm 6.3 describes a left-looking memory-limited IC factorization. In practice, C should be symmetrically scaled before the factorization commences. It can be beneficial to preorder C but no single approach works best for all problems. The advantages of reordering are typically less than for a complete factorization of a sparse matrix, because if the maximum number of entries in each column of $\widetilde{L}$ and $\widetilde{T}$ is held constant, the amount of fill in the incomplete factors is essentially

Algorithm 6.3. Left-looking memory-limited IC factorization
Input: SPD matrix $C \in \mathbb{R}^{n\times n}$ and $lsize > 0$ (maximum number of entries in a column of $\widetilde{L}$) and $tsize \geq 0$ (maximum number of entries in a column of $\widetilde{T}$).
Output: Incomplete Cholesky factorization $C \approx \widetilde{L}\widetilde{L}^\top$.

1: $w_i = 0,\ 1 \leq i \leq n$
2: **for** $j = 1 : n$ **do**
3: **for** $i \in \{i \geq j \mid (i,j) \in \mathcal{S}\{C\}\}$ **do**
4: $w_i = c_{ij}$
5: **end for**
6: **for** $k \in \{k < j \mid \tilde{l}_{jk} \neq 0\}$ **do**
7: **for** $i \in \{i \geq j \mid \tilde{l}_{ik} \neq 0\}$ **do**
8: $w_i = w_i - \tilde{l}_{ik}\,\tilde{l}_{jk}$
9: **end for**
10: **for** $i \in \{i \geq j \mid \tilde{t}_{ik} \neq 0\}$ **do**
11: $w_i = w_i - \tilde{t}_{ik}\,\tilde{l}_{jk}$
12: **end for**
13: **end for**
14: **for** $k \in \{k < j \mid \tilde{t}_{jk} \neq 0\}$ **do**
15: **for** $i \in \{i \geq j \mid \tilde{l}_{ik} \neq 0\}$ **do**
16: $w_i = w_i - \tilde{l}_{ik}\,\tilde{t}_{jk}$
17: **end for**
18: **end for**
19: Copy the *lsize* entries of w of largest absolute value into $\widetilde{L}_{j:n,j}$
20: Copy the next largest *tsize* entries of w into $\widetilde{T}_{j+1:n,j}$.
21: Scale $\tilde{l}_{jj} = (w_j)^{1/2}$, $\widetilde{L}_{j+1:n,j} = \widetilde{L}_{j+1:n,j}/\tilde{l}_{jj}$, $\widetilde{T}_{j+1:n,j} = \widetilde{T}_{j+1:n,j}/\tilde{l}_{jj}$
22: Reset w to zero.
23: **end for**

independent of the ordering of C that is used. AMD or nested dissection can be used (recall Sections 2.4 and 2.7). Alternatively, orderings based on reducing the profile of C can sometimes be more effective (Section 2.6).

When implementing an incomplete Cholesky factorization algorithm it is essential to handle the possibility of breakdown, which occurs if a very small or non-positive pivot is encountered (line 21 in Algorithm 6.3); this cannot normally be determined *a priori*. A simple remedy is to perturb a diagonal value if it is found to be too small (Kershaw 1978). Another possible approach is to modify

Algorithm 6.4. Shifted incomplete IC factorization
Input: SPD matrix C, diagonal scaling matrix S and initial shift $\alpha_S > 0$.
Output: Shift $\alpha \geq 0$ and incomplete Cholesky factorization $S^{-1}CS^{-1} + \alpha I \approx LL^\top$.

```
1: Ĉ = S^{-1} C S^{-1}                          ▷ Symmetrically scale C
2: α_0 = 0
3: for k = 0, 1, 2, ... do
4:     Ĉ + α_k I ≈ L L^⊤                        ▷ Algorithm 6.3 can be used
5:     If successful then set α = α_k and return
6:     α_{k+1} = max(2α_k, α_S)
7: end for
```

both the diagonal and off-diagonal entries (Ajiz and Jennings 1984). An alternative and generally more successful strategy is to terminate the factorization when breakdown is detected and to compute the incomplete factorization of a globally shifted matrix $C + \alpha I$, where $\alpha > 0$ is a scalar parameter. Choosing α is discussed in Lin and Moré (1999) and Scott and Tůma (2014*a*); see also Higham and Mary (2022). Algorithm 6.4 incorporates scaling of the normal matrix and uses a simple doubling strategy to increase the shift until the factorization is successful. More sophisticated strategies aim to limit the number of restarts and, if a sequence of related problems are to be solved, the initial shift may take advantage of knowledge of a suitable shift for the previous problem.

When the least-squares matrix A is rank-deficient, the normal matrix for the (unweighted) regularized problem is

$$C_\gamma = A^\top A + \gamma I,$$

where $\gamma > 0$ is the regularization parameter. If γ is chosen large enough then the IC factorization is breakdown-free. But, because the intention is to use the incomplete factors as a preconditioner for the original (unregularized) system, γ should be chosen to be small. Both requirements can make it difficult to choose appropriate γ. One possibility is to select γ to be sufficiently large and then to update the preconditioner to obtain a better approximation of the original problem (Cerdán *et al.* 2020).

6.6. Incomplete QR factorization preconditioners

Over the years, various incomplete orthogonal factorizations of the least-squares matrix A have been proposed but few have survived the test of time. They can be divided into two classes: those that give a factorization $A = \widetilde{Q}\widetilde{R}$ in which $\widetilde{Q}$ is not necessarily orthogonal, and those in which $\widetilde{Q}$ is orthogonal. The factor $\widetilde{R}$ can be used to obtain a preconditioner. Computing an incomplete QR (IQR) factorization

normally involves more work than an incomplete Cholesky factorization but the hope is that it will lead to a higher-quality preconditioner. Unfortunately, the QR factorization often suffers significant fill-in, and this makes finding effective sparse incomplete variants inherently difficult.

One possibility is to employ an incomplete modified Gram–Schmidt process (IMGS) (Jennings and Ajiz 1984). Dropping is restricted to off-diagonal entries of the $\widetilde{R}$ factor, and column j of $\widetilde{Q}$ is computed after dropping in the jth column of $\widetilde{R}$. That is, $\tilde{q}_j$ is computed using a linear combination of previously computed columns of $\tilde{Q}$ and the jth column a_j of A as

$$\tilde{q}_j = q/\|q\|_2, \quad \text{where} \quad q = a_j - \sum_{k<j, \tilde{r}_{kj}\neq 0} \tilde{r}_{kj}\tilde{q}_k.$$

Provided A has full column rank, this process does not break down because a_j cannot be expressed as a linear combination of less than j previously computed $\tilde{q}_i, i < j$. But dropping entries in $\widetilde{R}$ results in the computed columns of $\widetilde{Q}$ not being orthogonal. Incorporating dropping within the computation of $\widetilde{Q}$ can make the incomplete factorization less expensive, but there is then no guarantee that the factorization will not break down.

An interesting connection between the IMGS factorization of A and the incomplete Cholesky factorization of $A^\top A$ (both without dropping) is the following result (Wang, Gallivan and Bramley 1997).

Lemma 6.3. Assume A has full column rank and that an incomplete QR factorization $A \approx \widetilde{Q}\widetilde{R}$ is computed using the IMGS approach. Then, in exact arithmetic, $\widetilde{R}$ is the same as the incomplete Cholesky factor $\widetilde{L}^\top$ of $C = A^\top A$ computed using the decomposition (6.6) using the prescribed pattern $\mathcal{S}(\widetilde{L}) = \mathcal{S}(\widetilde{R}^\top)$.

There have been attempts to compute IQR factorizations using Givens rotations or Householder reflections. An early use of Givens rotations is in Ajiz and Jennings (1984); see also Papadopoulos, Duff and Wathen (2005) and Bai and Yin (2009). However, the modification and compensation strategies within these approaches are still not well understood and this remains an important open problem.

An alternative direction is to use QR factorizations within a hierarchical approach. An example of this is the multilevel incomplete Gram–Schmidt QR (MIQR) factorization (Li and Saad 2006). When A is sparse, many of its columns are likely to be orthogonal because of their structure. These structurally orthogonal columns form an independent set S_c. Once S_c is found, its columns are normalized and permuted to be the leading columns. The remaining columns of A are then orthogonalized against the first set. Because the matrix of remaining columns will, in general, still be sparse, it is natural to recursively repeat the process until the number of columns is small enough to orthogonalize with standard methods, or a prescribed number of reductions (levels) has been reached, or the matrix cannot be

reduced further. This gives a QR factorization of a column-permuted A and forms the basis of the MIQR factorization. In practice, because the QR factorization causes significant fill-in, sparsity is retained by relaxing the orthogonality and incorporating dropping strategies. The approach avoids computing the normal matrix $A^\top A$ explicitly as only one row of $A^\top A$ is needed at any given time. Moreover, because $A^\top A$ is symmetric, only its upper triangular part (i.e. the inner products between the ith column of A and columns i to n) is needed.

A recent hierarchical sparse approximate QR factorization (spaQR) is built on top of a nested dissection-based multifrontal QR approach (Gnanasekaran and Darve 2022). Low-rank approximations of the frontal matrices are used to sparsify vertex separators at every level in the elimination tree. A two-step sparsification scheme reduces the number of columns and maintains the ratio of rows to columns in each front without introducing additional fill-in. The resulting approximate QR factorization is stored as a sequence of sparse orthogonal and upper-triangular factors, which are straightforward to apply/solve with a vector. The approach avoids the problems associated with dropping strategies within traditional IQR approaches. Moreover, there is greater potential for exploiting parallelism than within a complete QR factorization.

6.7. *RIF preconditioner*

The Robust Incomplete Factorization (RIF) algorithm (Benzi and Tůma 2003*a*,*b*) computes a Cholesky factorization of the normal matrix C without forming any entries of C, working only with A. It is based on C-orthogonalization, that is, orthogonalization with respect to the C-inner product defined for all $x, y \in \mathbb{R}^n$ by

$$\langle x, y\rangle_C := x^\top C y = (Ax)^\top (Ay). \tag{6.7}$$

Given the n linearly independent vectors $e_1, e_2, \ldots, e_n$ (e_i is the ith unit basis vector), a C-orthogonal set of vectors $z_1, z_2, \ldots, z_n$ is built using a Gram–Schmidt process with respect to (6.7). This can be written in the form

$$Z^\top C Z = I, \quad I = L^\top Z,$$

where $Z = [z_1, z_2, \ldots, z_n]$ is upper triangular with positive diagonal entries. In exact arithmetic, $L^\top$ is the transposed Cholesky factor of C and Z is its inverse. The relationship between L and Z can be found, for example, in Hestenes and Stiefel (1952). It can be shown that L can be obtained as a by-product of the C-orthogonalization process at no extra cost.

Two different preconditioners can be obtained by carrying out the C-orthogonalization process incompletely. The first drops small entries from the computed vectors as the C-orthogonalization proceeds, resulting in a sparse matrix $\widetilde{Z} \approx L^{-\top}$; that is, an incomplete inverse factorization of the form

$$C^{-1} \approx \widetilde{Z}\widetilde{Z}^\top,$$

where $\widetilde{Z}$ has positive diagonal entries, is computed. This factored sparse approximate inverse can be used as a preconditioner and is generally known as the stabilized approximate inverse (SAINV) preconditioner. It is guaranteed to be positive definite and can be applied in parallel because its application requires only matrix–vector products.

The second approach is the RIF preconditioner, which is obtained by discarding the computed sparsified vector $\tilde{z}_i$ as soon as it has been used to form the corresponding parts of $\widetilde{L}$. This gives an algorithm for computing an incomplete Cholesky factorization $C \approx \widetilde{L}\widetilde{L}^\top$. Again, the preconditioner is positive definite, and (in exact arithmetic) breakdown during its computation is not possible. An important feature of the RIF preconditioner is that it incurs only modest intermediate storage costs, although implementing the algorithm to exploit the sparsity of A within the preconditioner construction is far from straightforward (Scott and Tůma 2016).

6.8. *LU-based factorization approaches*

Standard algorithms for solving square linear systems of equations are usually based on an LU factorization of the system matrix. For least-squares problems in which the system matrix is nearly square (the number of equations is not much more than the number of unknowns), using an LU factorization to obtain a preconditioner is a possible approach. This is also potentially useful for the (nearly square) weighted least-squares problem, even if the weighting matrix W is highly ill-conditioned. The Peters–Wilkinson method (Peters and Wilkinson 1970) for linear least-squares problems starts by computing an LU factorization of A (or $W^{-1/2}A$), using Gaussian elimination with row and column interchanges, that is,

$$P_1 A P_2 = LDU,$$

where the permutation matrices P_1 and P_2 are chosen to preserve sparsity and so that for $i > j$, $|l_{ij}| \leq \tau$ for some modest threshold parameter $\tau > 0$; this is likely to keep L well-conditioned (although less sparse than A) and any ill-conditioning in A will usually be seen in D. For the weighted problem with W diagonal, the weights are reflected in D. If A is of rank $k \leq n$, the factors can be written in the form

$$L = \begin{pmatrix} L_1 & 0 \\ L_2 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} D_1 & 0 \\ 0 & 0 \end{pmatrix}, \quad U = \begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix},$$

where L_1 is an $k \times k$ unit lower triangular matrix, L_2 is of order $(m-k) \times k$, D_1 is an $k \times k$ diagonal matrix, U_1 is an $k \times k$ unit upper triangular matrix and U_2 is of order $k \times (n-k)$. The least-squares problem becomes

$$\min_y \|P_1 b - Ly\|_2, \quad y = DUP_2^\top x,$$

and the corresponding normal equations, known as the L-normal equations, are

$$C_L y = L^\top P_1 b, \quad C_L = L^\top L.$$

If y and $P_1 b$ are conformally partitioned so that

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad P_1 b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix},$$

with $y_1 \in \mathbb{R}^k$, $y_2 \in \mathbb{R}^{n-k}$, $b_1 \in \mathbb{R}^k$ and $b_2 \in \mathbb{R}^{m-k}$, then y_1 is the solution of the $k \times k$ system

$$C_L\, y_1 = L_1^\top b_1 + L_2^\top b_2,$$

and we can set $y_2 = 0$. Thus $z = (z_1 \quad z_2)^\top$ with $z_1 \in \mathbb{R}^k$ is found by back substitution:

$$D_1 U_1 z_1 = y_1 \quad \text{and} \quad z_2 = 0.$$

Setting $x = P_2\, z$ gives the least-squares solution. If the L-normal equations are sufficiently well-conditioned, employing, for example, CGLS to solve them can require fewer iterations than when solving the standard normal equations (Howell and Baboulin 2016).

The DU factor can be used as a right-preconditioner within an iterative solver (Björck 1976, Saunders 1979). Use of the operator $A(DU)^{-1}$ involves scaling with D^{-1}, back substitutions with U and multiplications with A (L need not be stored). An important limitation is that the method is not robust when D has near-zero entries. The near-singularity of D is often undetected until the expensive LU factorization has been attempted.

An alternative strategy is to permute and partition A so that

$$PA = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix},$$

with A_1 of order $n \times n$ and non-singular, and then employ A_1 as a right-preconditioner. Its application requires a sparse LU (or QR) factorization of A_1. The original idea was introduced for dense problems in the early 1960s (Läuchli 1961) and there has been work in recent years focusing on how to determine P so as to preserve significant information from A in A_1; see, for example, the concept of maximum volume in pseudoskeleton approximations (Goreinov, Tyrtyshnikov and Zamarashkin 1997, Osinsky and Zamarashkin 2018, de Hoog and Hegland 2023).

6.9. Partial Cholesky factorizations

Partial Cholesky factorizations have been used in the area of large-scale optimization in which a sequence of weighted linear least-squares problems arising from a nonlinear problem must be solved (Bellavia, Gondzio and Morini 2013). The approach attempts to identify the largest eigenvalues of the normal matrix $C = A^\top W A$ and to ensure that the spectral radius of the preconditioned matrix satisfies $\kappa_2(M^{-1}C) < \kappa_2(C)$ by constructing a partial factorization that is terminated once the first k columns of the factor have been found. Omitting the permutation matrix that brings the largest k diagonal entries of C to the front of the matrix,

ready for use as pivots, a partial factorization of the form

$$C = LDL^\top = \begin{pmatrix} L_1 & \\ L_2 & I \end{pmatrix} \begin{pmatrix} D_1 & \\ & S \end{pmatrix} \begin{pmatrix} L_1^\top & L_2^\top \\ & I \end{pmatrix}$$

is computed. Here L_1 is an $k \times k$ unit lower triangular matrix and S is the Schur complement after k eliminations. Approximating S by its diagonal gives the preconditioner

$$M = \begin{pmatrix} L_1 & \\ L_2 & I \end{pmatrix} \begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix} \begin{pmatrix} L_1^\top & L_2^\top \\ & I \end{pmatrix}.$$

The factorization can be computed using an implicit process in which C and its Schur complements are not fully formulated: only the diagonal and selected columns of the Schur complements are calculated. In the applications of interest, A can be accessed by rows, and this makes computing M relatively straightforward and inexpensive compared to the cost of the matrix–vector products performed within the preconditioned iterative solver. The choice of k is dependent on the memory available for holding the first k columns of L.

While $\kappa_2(M^{-1}C)$ is generally significantly smaller than $\kappa_2(C)$, the smallest eigenvalues of C typically either remain unchanged or move towards the origin. If necessary, these small eigenvalues are handled by combining the partial factorization preconditioner with the deflated-CGLS algorithm (Saad, Yeung, Erhel and Guyomarch 2000). This requires using a Rayleigh–Ritz procedure to approximate the eigenvectors corresponding to some of the smallest eigenvalues of $M^{-1}C$. This comes at a non-negligible cost, and a reduction in the total solution time is not guaranteed.

6.10. Algebraic domain decomposition preconditioners

For extremely large problems, it can be infeasible (in terms of time and possibly memory requirements) to construct the factorization-based preconditioners discussed in the previous sections. The need for robust algebraic preconditioners that can be implemented efficiently in parallel recently led to the development of a two-level additive Schwarz preconditioner for the normal equations (Al Daas, Jolivet and Scott 2022). Multilevel domain decomposition preconditioners are a popular divide-and-conquer approach for solving challenging linear systems arising from discretizing PDEs. A fully algebraic approach for non-PDE problems is theoretically possible but, so far, it has been found to be prohibitively expensive for general linear systems (Al Daas and Grigori 2019). However, for least-squares problems, the similarity between the structure of the normal matrix and that of the weak formulation of PDEs can be exploited to develop new effective preconditioners.

Start by considering any $n \times n$ SPD matrix B. A graph-partitioning algorithm such as nested dissection (Section 2.7) is employed to partition the graph $\mathcal{G}(B)$ into $2 \leq N \ll n$ non-overlapping subdomains; this splits the vertices $1, 2, \ldots, n$ into N disjoint subsets Ω_{Ii} of size n_{Ii} $(1 \leq i \leq N)$. To obtain overlapping subdomains,

let $\Omega_{\Gamma i}$ be the subset of vertices that are distance one in $\mathcal{G}(B)$ from the vertices in Ω_{Ii}; let its size be $n_{\Gamma i}$. The overlapping subdomain is then $\Omega_i = \Omega_{Ii} \cup \Omega_{\Gamma i}$, of size $n_i = n_{Ii} + n_{\Gamma i}$. Associated with Ω_i is a restriction matrix Π_i that maps from the global domain to the subdomain; a prolongation matrix is given by $\Pi_i^\top$. A two-level additive Schwarz preconditioner is given by

$$M^{-1} = \sum_{i=0}^{N} \Pi_i^\top B_i^{-1} \Pi_i, \quad B_i = B(\Omega_i, \Omega_i), \; B_0 = \Pi_0 B \Pi_0^\top,$$

where Π_0 is of full rank and is constructed so that the preconditioned system is well-conditioned. For any Π_0 it is possible to cheaply obtain upper bounds on the largest eigenvalue of the preconditioned system, independently of n and N. However, bounding the smallest eigenvalue is highly dependent on Π_0. Thus, this choice is key to building efficient two-level Schwarz preconditioners. Other two-level variants exist that can yield better convergence behaviour.

For least-squares problems, the normal matrix C is not a general SPD matrix but the product $A^\top A$. Practical two-level Schwarz preconditioners can be derived by exploiting this structure (Al Daas *et al.* 2022). The main work in building the preconditioner is, for each N, computing the Cholesky factors of two local matrices of order n_i; these factorizations can be performed in parallel. In addition, Π_0 must be constructed and then the Cholesky factorization of $C_0 = (A\Pi_0^\top)^\top (A\Pi_0^\top)$ computed. It can be shown that the spectral condition number of the preconditioned normal matrix is bounded from above independently of the number of subdomains and the size of the problem. Moreover, this upper bound depends on a single parameter that can be chosen to decrease (respectively, increase) the upper bound with the costs of setting up the preconditioner being larger (respectively, smaller).

6.11. Two-level limited-memory preconditioners

Consider the SPD normal equations and assume that a preconditioner is available that clusters most of the eigenvalues of the preconditioned normal matrix at +1 with relatively few outliers. Provided the outlying eigenvalues are small, the convergence of the preconditioned iterative method is typically not adversely effected by this, but if they belong to the right-hand end of the spectrum, convergence can be significantly delayed (Liesen and Strakoš 2013). It may be possible to improve the performance of this first-level preconditioner by adding a second-level preconditioner. For example, consider the normal equations (1.26) for the linear subproblem that comes from linearizing a nonlinear least-squares problem. The structure of the normal matrix $C = W_2^{-1} + H_j^\top W_1^{-1} H_j$ suggests that a natural candidate for a preconditioner is W_2^{-1}, and the symmetrically preconditioned system is then of the form

$$\hat{C}s = W_2^{1/2} C W_2^{1/2} s = (I + X)s = \hat{b}, \quad X = W_2^{1/2} H_j^\top W_1^{-1} H_j W_2^{1/2}.$$

Here, the preconditioned normal matrix $\hat{C}$ is the sum of the identity matrix and a symmetric positive semidefinite matrix X of rank $q \ll n$. Hence, $\hat{C}$ is SPD and has a cluster of $n - q$ eigenvalues at +1 and q eigenvalues that are greater than one; its spectral condition number is equal to $\lambda_{\max}(\hat{C})$. Second-level preconditioning seeks to reduce the condition number while preserving the cluster of eigenvalues at +1. More generally, the aim is to capture the directions that have been left out by the first-level preconditioner and slow down the convergence of the Krylov solver.

A class of limited-memory preconditioners (LMPs) has been proposed with this two-level structure in mind. These preconditioners are based on the explicit knowledge of a full-rank matrix Z with $k \ll n$ columns and on the product of Z with $\hat{C}$. Specifically, let C be an SPD matrix of order n and M_1 an SPD preconditioner for C (the first-level preconditioner). Assume that Z is any $n \times k$ matrix with $\operatorname{rank}(Z) = k \ll n$. The symmetric matrix

$$M = (I - M_2 C)M_1^{-1}(I - CM_2) + M_2, \quad M_2 = ZE^{-1}Z^\top, \quad E = Z^\top CZ, \tag{6.8}$$

is called an LMP (e.g. Gratton, Sartenaer and Tshimanga 2011, Daužickaitė, Lawless, Scott and Van Leeuwen 2021). In domain decomposition, it is known as a balancing preconditioner (Mandel 1993); see also Tang, Nabben, Vuik and Erlangga (2009) and Zhao (2016) for an analysis of subspace enhanced preconditioners. If $M_1 = W_2$, $\hat{C} = W_2^{1/2} C W_2^{1/2}$ and $\hat{Z} = W_2^{-1/2} Z$, then from (6.8) we obtain the preconditioner

$$\hat{M} = W_2^{-1/2} M W_2^{-1/2} = (I - \hat{M}_2\hat{C})(I - \hat{C}\hat{M}_2) + \hat{M}_2, \quad \hat{M}_2 = \hat{Z}\hat{E}^{-1}\hat{Z}^\top, \quad \hat{E} = \hat{Z}^\top\hat{C}\hat{Z}. \tag{6.9}$$

Assume $\hat{E}$ is available in factored form, i.e. $\hat{E}\,\hat{Z} = Y^\top Y$, where Y is $k \times k$. Then (6.9) can be factorized as $\hat{M} = M_3^\top M_3$, where

$$M_3 = I - \hat{Z}Y^{-1}Y^{-\top}\hat{Z}^\top\,\hat{C} + \hat{Z}Y^{-1}.$$

A potential problem for practical applications is the need for expensive matrix–matrix products with $\hat{C}$. Simpler formulations are obtained by imposing more conditions on the columns $z_1, \ldots, z_k$ of $\hat{Z}$. Two approaches used, for example, in ocean data assimilation are the spectral-LMP and Ritz-LMP.

Let $z_1, \ldots, z_n$ be orthonormal eigenvectors of $\hat{C}$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$. Set $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ and let the columns of $\hat{Z}$ be $z_1, \ldots, z_n$, so that $\hat{C}\hat{Z} = \hat{Z}\Lambda$ and $\hat{Z}^T\hat{Z} = I$. Substituting into (6.9) and simplifying, the spectral-LMP (or deflation preconditioner Giraud and Gratton 2006) is given by

$$M^{sp} = I - \sum_{i=1}^{n}(1 - \lambda_i^{-1})z_i z_i^\top.$$

M^{sp} moves the eigenvalues λ_i, $i = 1, \ldots, k$ to +1 and leaves the rest of the spectrum

unchanged. In factored form, $M^{sp} = M_3^{sp}(M_3^{sp})^\top$ with

$$M_3^{sp} = \prod_{i=1}^{n} \left(I - \left(1 - \left(\sqrt{\lambda_i}\right)^{-1}\right) z_i z_i^\top\right).$$

In most applications, exact eigenpairs are not available, and so Ritz values and vectors are used. Setting $z_1, \ldots, z_n$ to be orthogonal Ritz vectors and $\Theta = \mathrm{diag}(\theta_1, \ldots, \theta_n)$ to be the corresponding Ritz values, the Ritz-LMP is given by

$$M^{Rt} = (I - \hat{Z}\Theta^{-1}\hat{Z}^\top \hat{C})(I - \hat{C}\hat{Z}\Theta^{-1}\hat{Z}^\top) + \hat{Z}\Theta^{-1}\hat{Z}^\top. \tag{6.10}$$

Each application of M^{Rt} requires a matrix–matrix product with $\hat{C}$. But if a sequence of problems needs to be solved and if the Ritz vectors are obtained by a Lanczos process, then (6.10) can be simplified and matrix–matrix products with $\hat{C}$ avoided (Gratton *et al.* 2011). In practice, both spectral-LMP and Ritz-LMP use Ritz vectors and values to construct the LMPs. Only approximations to the largest eigenvalues and corresponding eigenvectors are required. Randomized algorithms can be used. These are explored for problems arising in data assimilation in Daužickaitė *et al.* (2021) (see also Scotto di Perrotolo 2022). In practice, these problems can be extremely large and products involving C are very expensive, so much so that only a few iterations of an iterative solver can be performed (the solver is typically run for a fixed number of iterations, not to convergence). In such situations, having a good-quality preconditioner that is cheap to compute and apply is important and challenging.

6.12. Sketch-and-precondition

As observed above, randomized methods can be used in constructing LMPs. More generally, in recent years there has been significant growth in the development and employment of randomized techniques for tackling very large problems within numerical linear algebra; see, for example, the review by Drineas and Mahoney (2016) and the survey article by Martinsson and Tropp (2020). Sketch-and-precondition algorithms for least-squares problems use sketching to construct a preconditioner to be used in an iterative solver. Matrix sketching is a data compression technique that is characterized by the property of preserving most of the linear information that is present in the data. Algorithm 6.5 outlines the classical sketch-and-precondition approach for highly overdetermined least-squares problems (Meng, Saunders and Mahoney 2014). Here, an SVD factorization of the sketched matrix is computed, but a QR factorization could be used (Avron, Ng and Toledo 2009); see also Ipsen and Wentworth (2014) for related theoretical aspects of uniform samplings.

The approach can easily be extended to handle Tikhonov regularization and a similarly structured algorithm works for strongly underdetermined systems. Note that A is used only for matrix–vector and matrix–matrix operations and so it can be sparse or dense or a linear operator. The preconditioned system is

Algorithm 6.5. A sketch-and-precondition LS solver (overdetermined case)
Input: $A \in \mathbb{R}^{m\times n}$ of rank rk, $m \gg n$ and $b \in \mathbb{R}^m$.
Output: Least-squares solution x.

1: Choose an oversampling factor $\gamma > 1$ and set $s = [\gamma n]$ ▹ e.g. $\gamma = 2$
2: Draw a Gaussian random matrix $\Omega \in \mathbb{R}^{s\times m}$
3: Form the sketch $Y = \Omega A \in \mathbb{R}^{s\times n}$ ▹ Row sampling.
4: Compute the compact SVD factorization $Y = U\Sigma V^\top$, where $U \in \mathbb{R}^{s\times rk}$, $\Sigma \in \mathbb{R}^{rk\times rk}$, $V \in \mathbb{R}^{n\times rk}$
5: Set $M^{-1} = V\Sigma^{-1}$ and compute least-norm solution of $\min_{z\in\mathbb{R}^{rk}} \|b - AM^{-1}z\|_2$.
6: Return $x = M^{-1}z$

well-conditioned and, when LSQR is used in line 5, the algorithm can be shown to have a fully predictable run-time performance, just like direct solvers, and it scales well in parallel environments.

More recent sketch-and-precondition based-methods for regularized problems are given in Ozaslan, Pilanci and Arikan (2023) and Meier and Nakatsukasa (2022). One uses a Cholesky-based sketch-and-precondition technique while the other is very efficient for matrices with small statistical dimension $sd_\gamma(A) = tr(A(A^\top A + \gamma I_n)^{-1}A^\top$, where γ is the regularization parameter in (1.17). Unfortunately, the sketch-and-precondition approach in its most commonly used form can be numerically unstable for ill-conditioned problems. This has recently led to investigations into variants that seek to ensure improved stability properties (Epperly 2024, Epperly, Meier and Nakatsukasa 2024, Meier, Nakatsukasa, Townsend and Webb 2024).

6.13. Preconditioning the augmented system formulation

There has been a wealth of research devoted to developing preconditioners for symmetric indefinite linear systems of the form

$$\begin{pmatrix} H & G \\ G^\top & -E \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \tag{6.11}$$

where H and E are square symmetric matrices. These so-called saddle-point systems arise in a wide range of practical applications that lead to blocks having different properties. The augmented system formulation of linear least-squares problems is just one important area. There are a number of extensive survey articles on preconditioning saddle-point problems that include discussions relevant to sparse least-squares problems, most notably those of Benzi, Golub and Liesen (2005), Wathen (2015), and Pearson and Pestana (2020).

An interesting result is that every preconditioner for the normal equations yields an equivalent preconditioner for the augmented system but the converse is not true (Oliveira and Sorensen 2005). Thus working with the augmented system potentially provides greater opportunities to find effective preconditioners. Many approaches for constructing preconditioners exploit the block structure in (6.11). For the augmented system obtained from a (regularized) least-squares problem the blocks are very special. In particular, if the matrix of weights W is diagonal then H is diagonal. Thus, in contrast to the augmented systems coming from PDE problems, most of the complexity lies in the off-diagonal blocks and this influences which approaches are potentially appropriate.

Block-based preconditioners with the same 2×2 structure as the saddle-point system are used in many application areas. The basic block diagonal preconditioner for the least-squares augmented system (5.1) with a zero $(2,2)$ block is

$$M = \begin{pmatrix} I & \\ & -\widetilde{S} \end{pmatrix},$$

where $\widetilde{S}$ approximates the Schur complement $S = -A^\top A$. A straightforward choice is $\widetilde{S} = -\widetilde{R}^\top \widetilde{R}$, where $\widetilde{R}$ is the factor of an incomplete QR factorization of A:

$$(\widetilde{Q}_1 \;\; \widetilde{Q}_2) \begin{pmatrix} \widetilde{R} \\ 0 \end{pmatrix} = \widetilde{Q}_1 \widetilde{R}. \tag{6.12}$$

This yields the positive definite preconditioner

$$M = \begin{pmatrix} I & 0 \\ 0 & \widetilde{R}^\top \widetilde{R} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & \widetilde{R}^\top \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & \widetilde{R} \end{pmatrix} = M_1 M_1^\top.$$

Two-sided application of this preconditioner gives

$$M_1^{-1} K M_1^{-\top} = M_1^{-1} \begin{pmatrix} I & A \\ A^\top & 0 \end{pmatrix} M_1^{-\top} = \begin{pmatrix} I & A\widetilde{R}^{-1} \\ \widetilde{R}^{-\top} A^\top & 0 \end{pmatrix} = \begin{pmatrix} I & \widetilde{Q}_1 \\ \widetilde{Q}_1^T & 0 \end{pmatrix}.$$

Constraint preconditioners are an important alternative family of block preconditioners. These are (possibly indefinite) preconditioners in which the off-diagonal blocks (the constraints) are as in the saddle-point matrix and the $(1,1)$ and $(2,2)$ blocks are approximated (e.g. by their diagonal entries); see di Serafino and Orban (2021). For the augmented systems in least-squares problems, approximating these blocks is unnecessary (assuming the matrix of weights is the identity or diagonal). Instead, the off-diagonal blocks are approximated, resulting in inexact constraint preconditioners (Coleman and Verma 2001, Zilli and Bergamaschi 2022). In the special case of the augmented system with the exact constraint preconditioner in which the right-hand side is of the form $(b^\top \; 0^\top)^\top$, preconditioned CG can be used to solve the indefinite preconditioned system (Lukšan and Vlček 1998); see also Benzi *et al.* (2005). Given the inexact QR factorization (6.12), a natural inexact

constraint preconditioner is

$$M = \begin{pmatrix} I & \widetilde{Q}_1 \widetilde{R} \\ \widetilde{R}^\top \widetilde{Q}_1^\top & 0 \end{pmatrix}.$$

If a complete QR factorization of A is available, it is not necessary to use the augmented formulation because the factors can be used to solve the normal equations directly. However, if the complete factorization is performed using lower precision arithmetic (for example, single precision might be used with the possible objective of speeding up the computation or saving memory), then the computed factors can be used within block-based preconditioners to try and recover full (double) precision accuracy in the least-squares solution. This is analysed in Carson, Higham and Pranesh (2020).

6.14. Approximate factorizations of the augmented matrix

Another possibility for preconditioning the augmented system formulation is a signed IC factorization that exploits the block structure. Specifically, compute a signed incomplete factorization of the form

$$SKS + \begin{pmatrix} \alpha_1 I & \\ & -\alpha_2 I \end{pmatrix} \approx \widetilde{L} \widetilde{D} \widetilde{L}^\top, \tag{6.13}$$

where S is a diagonal scaling matrix, $\widetilde{L}$ is a unit lower triangular matrix, and $\widetilde{D}$ is a diagonal matrix with positive and negative entries (Scott and Tůma 2014*c*). The two non-negative shifts α_1 and α_2 are chosen to prevent breakdown of the factorization. Such a shifting strategy is closely connected to the regularization techniques used by the numerical optimization community. In exact arithmetic, the shifts can always be chosen such that a signed IC factorization exists. If a pivot is found to be too small then if it corresponds to an entry in the $(1, 1)$ block (respectively, $(2, 2)$ block) then the factorization is terminated and restarted with a larger value of α_1 (respectively, α_2). In practice, K is generally preordered. A key attraction of this approach is that it avoids the need for numerical pivoting. It is straightforward to modify the memory-limited IC approach of Algorithm 6.3 to obtain the incomplete factorization (6.13).

The computed incomplete factors can be used as preconditioners for the SQMR method (Freund 1997) or GMRES. The SYMMLQ and MINRES methods are not directly applicable as they require positive definite preconditioners. However, they can be used on the symmetrically preconditioned system

$$|\widetilde{D}|^{-1/2} \widetilde{L}^{-1} SKSL^{-\top} |\widetilde{D}|^{-1/2} \begin{pmatrix} s \\ y \end{pmatrix} = |\widetilde{D}|^{-1/2} \widetilde{L}^{-1} \begin{pmatrix} b \\ 0 \end{pmatrix}, \quad \begin{pmatrix} r \\ x \end{pmatrix} = L^{-\top} |\widetilde{D}|^{-1/2} \begin{pmatrix} s \\ y \end{pmatrix},$$

where the entries of $|\widetilde{D}|$ are the absolute values of the entries of $\widetilde{D}$. Results given in Gould and Scott (2017) demonstrate that this approach with MINRES can outperform using GMRES with (6.13).

A related strategy is to exploit the fact that the system matrix K_W for the regularized weighted least-squares problem matrix (1.24) is symmetric quasi-definite (SQD) (recall Section 1.7). Consequently, if L and D are its exact factors, the preconditioned operator

$$|D|^{-1/2}L^{-1}K_W L^{-\top}|D|^{-1/2} = |D|^{-1}D,$$

possesses only two distinct eigenvalues: $+1$ and -1. The hope is therefore that incomplete factors have the potential to yield a preconditioned system with a favourable spectral structure while keeping the computational effort reasonable. When $\widetilde{L}$ and $\widetilde{D}$ are limited-memory factors of K_W, the corresponding preconditioned operator remains symmetric and indefinite so that it is possible to use MINRES or SYMMLQ. This is explored in Orban (2015).

6.15. Incomplete factorizations of symmetric indefinite matrices

In comparison with research into incomplete factorization preconditioners for SPD systems, work on reliable incomplete factorization techniques for general symmetric indefinite problems is limited. It is significantly more challenging because numerical pivoting must often be incorporated to limit growth in the sizes of the factor entries. The Bunch–Kaufman partial pivoting strategy is widely employed for factorizing dense symmetric indefinite matrices using 1×1 and 2×2 pivots (Bunch and Kaufman 1977). It is also often used in the factorization of dense blocks within sparse factorizations and has been integrated within incomplete factorizations for sparse symmetric indefinite matrices (Li and Saad 2005, Greif, He and Liu 2017).

For sparse problems, the need for pivoting may be reduced by preprocessing the matrix. The use of a matching-based ordering and scaling algorithm was discussed in Section 5.2 for complete factorizations but it can also be applied to avoid dynamic pivoting in incomplete factorizations (Hagemann and Schenk 2006, Chen, Huang and Li 2012, Scott and Tůma 2017*a*). The matching-based ordering is used to *a priori* symmetrically permute large entries to the subdiagonal positions. Tridiagonal pivoting then restricts the search for pivots to the diagonal and subdiagonal. If a 1×1 pivot candidate is found to be too small and pairing it with the subdiagonal entry in its row does not give a suitable a 2×2 pivot, then breakdown can be prevented by replacing the pivot candidate with a small predefined quantity. Details of how to do this while guaranteeing backward stability are given in Chen *et al.* (2012). Combined with dropping the smallest entries as each column of the factor is computed, the result is an incomplete factorization that is potentially efficient to compute (because the search for pivot candidates is limited) as well as more straightforward than more sophisticated approaches to selecting stable pivots. For instance, strategies that monitor growth in the factors and incorporate pivot modifications when potential instability is observed have been proposed within a limited-memory incomplete factorization algorithm (Scott and Tůma 2017*a*).

6.16. *Software for algebraic preconditioners*

Despite the interest in preconditioning large-scale linear least-squares problems, there is limited high-quality software available that is specifically designed for such problems and is fully documented and supported. A study comparing the performance of software for sparse problems is given in Gould and Scott (2017). The sequential package `HSL_MI35` from the HSL library[24] implements Algorithm 6.3, incorporating global shifts to handle potential breakdown. This software avoids forming the normal matrix explicitly and uses it in the implicit form of A only; it is based on Scott and Tůma (2014*a*). BA-GMRES software is available[25] but this does not appear to be actively maintained.

Prototype codes for left- and right-looking implementations of RIF have been written[26] but they have not been developed into library-quality software. The multilevel QR factorization described in Section 6.6 is available within the 2005 package MIQR.[27]

A parallel implementation of the the two-level additive Schwarz preconditioner discussed in Section 6.10 is available in PCHPDDM,[28] which is a part of PETSc.[29] PETSc also offers a wide range of algebraic preconditioners aimed primarily at solving sparse linear systems (rather than specifically for least-squares problems). To employ an incomplete Cholesky factorization with the PETSc implementation of left-preconditioned LSQR, the normal matrix must be formed.

The HSL package `HSL_MC81` uses randomized algorithms to compute low-rank approximations of a given matrix. These can be used to construct the LMPs introduced in Section 6.11.

If A is dense and of full rank then a well-known sketching-based solver is Blendenpik.[30] It uses a QR factorization of the sketch (rather than an SVD factorization). For sparse (and possibly rank-deficient) A, the sequential code LSRN[31] implements Algorithm 6.5. A comparison of sketch-and-precondition techniques (including Blendenpik and LSRN) is given in Ozaslan *et al.* (2023). More recent software is the sketching for least-squares package Ski-LLS[32] (Cartis, Fiala and Shao 2021). However, this appears to be research code that is not being actively maintained. The monograph by Murray *et al.* (2023) discusses the development of standard libraries for randomized numerical linear algebra, including least-squares.

[24] https://www.hsl.rl.ac.uk/
[25] https://github.com/morikuni-keiichi
[26] https://www.karlin.mff.cuni.cz/ mirektuma/sparslab.html
[27] https://www-users.cse.umn.edu/ saad/software/MIQR.tar.gz
[28] https://petsc.org/main/manualpages/PC/PCHPDDM/
[29] https://petsc.org/release/
[30] https://github.com/haimav/Blendenpik
[31] https://web.stanford.edu/group/SOL/software/lsrn/
[32] https://github.com/numericalalgorithmsgroup/Ski-LLS

There is little robust software for computing sparse incomplete LDLT factorizations of the augmented system matrix. The package `HSL_MI30` implements a memory-limited signed IC algorithm for solving saddle-point systems that can be used for the augmented system formulation combined with either preconditioned GMRES or preconditioned MINRES (Scott and Tůma 2014*c*). sym-ildl[33] computes incomplete factorizations of general symmetric indefinite matrices. It builds upon ideas proposed in Li, Saad and Chow (2003) and Li and Saad (2005), and incorporates scaling, preordering and Bunch–Kaufman pivoting (Greif *et al.* 2017). As the memory-limiting parameter increases, the incomplete factors converge to the exact factors. This software also does not appear to be currently maintained. LLDL[34] is a modification of the limited-memory Cholesky factorization code ICFS from 1999 for symmetric positive definite matrices described in Lin and Moré (1999). It implements a similar scheme for symmetric indefinite matrices that possess a LDLT factorization with D diagonal. This includes SQD matrices.

Pardiso[35] is primarily a package of direct solvers but it also offers a preconditioning approach for general symmetric indefinite linear systems based on using maximum weighted matching orderings and algebraic multilevel incomplete LDLT factorizations. ILUPACK[36] includes routines for incomplete Cholesky, LDLT and ILU factorizations as well as corresponding iterative methods that exploit reliable (inverse-based) diagonal pivoting and multilevel framework. Comparisons between ILUPACK and other related incomplete and complete factorizations for solving mainly non-symmetric systems, but also symmetric indefinite systems, can be found in Chen, Ghai and Jiao (2021). A recent strategy that exploits numerical rank deficiency of the off-diagonal blocks of the incomplete Cholesky factor has been compared with ILUPACK (Napov 2023).

7. Iterative refinement for least-squares problems

Direct linear solvers theoretically provide exact solutions, but if the problem is ill-conditioned, then in finite floating-point precision, rounding errors may significantly degrade the accuracy of the computed solution. Unfortunately, this can pass unnoticed because the associated residual can be relatively small. The method of iterative refinement seeks to improve accuracy of the computed approximate solution. For the square linear system $By = d$, the computed solution $y^{(1)}$ is refined by computing the residual vector $r^{(1)}$ (possibly using a higher precision) and solving the linear system $B\delta y^{(1)} = r^{(1)}$, with the residual as the right-hand side vector, to obtain a correction $\delta y^{(1)}$ to the solution. The process may be repeated until

[33] https://github.com/where-is-paul/matrix-factor
[34] https://github.com/optimizers/lldl
[35] https://panua.ch/pardiso/
[36] http://ilupack.tu-bs.de/

Algorithm 7.1. Solve the LS problem using basic iterative refinement in two precisions

Input: Overdetermined full-rank matrix A, vector b and precisions u_h and u, where $u_h \leq u^2$.

Output: least-squares solution x and least-squares residual r.

1: Compute an initial approximate least-squares solution $x^{(1)}$
2: **for** $i = 1, 2, \ldots$ **do**
3: Compute the residual vector $r^{(i)} = b - Ax^{(i)}$ in precision u_h
4: If converged then return $x = x^{(i)}$, $r = r^{(i)}$ and stop
5: Solve $\min_{\delta x^{(i)}} \|r^{(i)} - A\,\delta x^{(i)}\|_2$ in precision u ▷ Correction equation
6: Update the solution $x^{(i+1)} = x^{(i)} + \delta x^{(i)}$ in precision u
7: **end for**

the solution is sufficiently accurate or stagnation occurs using the following steps: (a) compute the residual $r^{(i)} = d - By^{(i)}$; (b) solve $B\delta y^{(i)} = r^{(i)}$ for the correction $\delta y^{(i)}$; (c) update the solution $y^{(i+1)} = y^{(i)} + \delta y^{(i)}$. A comprehensive summary of rounding error analysis for iterative refinement for linear systems is given in Carson and Higham (2018).

For least-squares problems, various iterative refinement strategies have been proposed. This case is more challenging than solving square linear systems. In particular, because the least-squares residual $r = b - Ax$ may be non-zero, some of the convergence guarantees that are available for linear systems are not valid. Moreover, the least-squares matrix A can be so ill-conditioned that the computed solution has few, if any, correct digits. If the overdetermined system is nearly consistent (i.e. there exists x for which the residual norm $\|r\|_2 = \|b - Ax\|_2$ is close to zero), then the straightforward Algorithm 7.1 can be used; this is analogous to iterative refinement for linear systems (Businger and Golub 1965, Golub 1965). The approach used to compute the initial approximate solution $x^{(1)}$ can be reapplied to solve for the each correction $\delta x^{(i)}$. In particular, if a QR factorization of A is computed or a Cholesky factorization of the normal matrix, then the factors can be reused, thereby limiting the cost of each refinement step. Alternatively, if a preconditioned iterative solver is used, then the same preconditioner can be used for each solve.

A limitation of Algorithm 7.1 is that it is shown in Golub and Wilkinson (1966) that when the Householder QR factorization is used to solve the correction equation, the least-squares solution may not be found unless the system is nearly consistent. Regardless of the precision used, there will be vectors b for which it will fail to give solutions that are correct to working accuracy. Furthermore, the approach can be sensitive to the quality of the initial solution $x^{(1)}$.

Algorithm 7.2. Solve the LS problem using the semi-normal equations and iterative refinement
Input: Overdetermined full-rank matrix A and its R factor and vector b.
Output: Least-squares solution x and least-squares residual r.

1: Compute an initial approximate least-squares solution $x^{(1)}$
2: **for** $i = 1, 2, \ldots$ **do**
3: Compute the residual vector $r^{(i)} = b - Ax^{(i)}$
4: If converged then return $x = x^{(i)}$, $r = r^{(i)}$ and stop
5: Solve $R^\top R\, \delta x^{(i)} = A^\top r^{(i)}$ ▷ Semi-normal equations
6: Update the solution $x^{(i+1)} = x^{(i)} + \delta x^{(i)}$
7: **end for**

7.1. Refinement using the semi-normal equations

An alternative idea mentioned in Golub and Wilkinson (1966) is to employ the semi-normal equations (1.13). Algorithm 7.2 uses the QR factorization of A and, at each refinement step, the semi-normal equations are solved for the correction. The initial solution $x^{(1)}$ can be computed using any appropriate approach. If it is obtained using the semi-normal equations and a single correction is computed, then Algorithm 7.2 is the method of corrected semi-normal equations (CSNE). In general, unless the problem is well-conditioned, several refinement steps may be required. The numerical properties of this type of iterative refinement and different variations of the semi-normal equations are discussed in Björck (1987) and Rozložník, Smoktunowicz and Kopal (2014).

7.2. Refinement using the augmented system

A generalization of iterative refinement that can be effective whether or not the system is close to being consistent employs the augmented system (5.1). The approach outlined in Algorithm 7.3 simultaneously refines the computed solution and the corresponding residual (Björck 1967*a*).

If the QR factorization of A has been computed then the corrections can be obtained by reusing the factors. Consider the augmented system

$$\begin{pmatrix} I & A \\ A^\top & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} w \\ t \end{pmatrix}. \tag{7.1}$$

Using the QR factorization

$$\begin{pmatrix} AP & b \end{pmatrix} = Q \begin{pmatrix} R & c \\ 0 & d \end{pmatrix} \quad \text{and} \quad QQ^\top = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \in \mathbb{R}^{m \times m},$$

Algorithm 7.3. Solve the LS problem using the augmented formulation and iterative refinement
Input: Overdetermined full-rank A and vector b.
Output: Least-squares solution x and least-squares residual r.

1: Compute an initial approximate least-squares solution $x^{(1)}$ and residual $r^{(1)}$
2: **for** $i = 1, 2, \ldots$ **do**
3: Compute the residual vector for the augmented system

$$\begin{pmatrix} f^{(i)} \\ g^{(i)} \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} I & A \\ A^\top & 0 \end{pmatrix} \begin{pmatrix} r^{(i)} \\ x^{(i)} \end{pmatrix} = \begin{pmatrix} b - r^{(i)} - Ax^{(i)} \\ -A^\top r^{(i)} \end{pmatrix}$$

4: If converged then return $x = x^{(i)}$, $r = r^{(i)}$ and stop.
5: Solve $\begin{pmatrix} I & A \\ A^\top & 0 \end{pmatrix} \begin{pmatrix} \delta r^{(i)} \\ \delta x^{(i)} \end{pmatrix} = \begin{pmatrix} f^{(i)} \\ g^{(i)} \end{pmatrix}$ ▷ Correction equation
6: Update $\begin{pmatrix} r^{(i+1)} \\ x^{(i+1)} \end{pmatrix} = \begin{pmatrix} \delta r^{(i)} \\ \delta x^{(i)} \end{pmatrix} + \begin{pmatrix} r^{(i)} \\ x^{(i)} \end{pmatrix}$
7: **end for**

we have

$$\begin{pmatrix} I & AP \\ P^\top A^\top & 0 \end{pmatrix} \begin{pmatrix} u \\ P^\top v \end{pmatrix} = \begin{pmatrix} Q & \\ & I \end{pmatrix} \begin{pmatrix} I & 0 & R \\ 0 & I & 0 \\ R^\top & 0 & 0 \end{pmatrix} \begin{pmatrix} Q^\top & \\ & I \end{pmatrix} \begin{pmatrix} u \\ P^\top v \end{pmatrix} = \begin{pmatrix} w \\ P^\top t \end{pmatrix},$$

so that

$$\begin{pmatrix} I & 0 & R \\ 0 & I & 0 \\ R^\top & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ f \\ P^\top v \end{pmatrix} = \begin{pmatrix} c \\ d \\ P^\top t \end{pmatrix},$$

where

$$\begin{pmatrix} c \\ d \end{pmatrix} = Q^\top w \quad \text{and} \quad u = Q \begin{pmatrix} e \\ f \end{pmatrix} = Q \begin{pmatrix} e \\ d \end{pmatrix}.$$

The component e is found by solving

$$PR^\top e = t,$$

and finally v is the solution of

$$RP^\top v = c - e.$$

Hence each refinement step requires a solve with R and with $R^\top$ plus a matrix–vector product with Q and $Q^\top$.

Without round-off errors, the process would converge to the correct solution in a single iteration. In practice, after a few iterations stagnation occurs (i.e. a point

is reached after which little further accuracy is achieved). Thus, the refinement is terminated when either the correction is sufficiently small, or it stagnates, or a prescribed maximum number of iterations is reached. While the computed solution initially improves with each iteration, this is usually not reflected in a corresponding decrease in the norm of the residual, which typically stays about the same. Convergence analysis is given by Björck (1967*a*). Note that in the iterative refinement of Algorithm 7.3, the solver choice in line 5 for obtaining the correction is crucial and the system can be regularized; see (1.7).

7.3. Mixed precision refinement

There are a number of variants of iterative refinement that involve using different precisions for all or part of the computation. In fixed precision refinement, the same precision is used throughout. In mixed precision iterative refinement, the most expensive operations (the matrix factorization and solving the correction equation) are performed in the working precision u and the residual computation performed in a higher precision $u_h < u$ by accumulating the matrix–vector product using precision u_h. This has been used since the 1960s (Businger and Golub 1965, Björck 1967*a*) (see e.g. Algorithm 7.1). If u and u_h are single and double precision, respectively, employing mixed precision can be attractive because single precision arithmetic is often significantly faster than double precision. Moreover, holding the factors in single precision substantially reduces memory requirements and the amount of data movement. Alternatively, for highly ill-conditioned problems, it may be necessary to select u to be double precision and u_h to be quad (double-double) precision (Demmel, Hida, Riedy and Li 2009).

Most recently, there has been interest in using half precision (16 bit) for the working precision because it can be very fast, it reduces memory requirements and data movement, and can thereby potentially yield significant savings in energy consumption and enable larger problems to be solved (Carson *et al.* 2020, Scott and Tůma 2022*a*, 2024). The initial work in this area builds on the GMRES-IR method (Carson and Higham 2017, 2018), which is a mixed precision variant of iterative refinement for linear systems. At each iteration, the correction equation is solved using GMRES preconditioned using the factors of the system matrix. Observe that if the working precision is chosen to be low precision then the notion of what is an ill-conditioned problem changes accordingly. For example, if $\kappa_2(A) \approx 10^3$ and the working precision is double precision, then the problem may not be regarded as being ill-conditioned. However, if u is IEEE half precision, then A is ill-conditioned in the working precision.

For least-squares problems, if A is well-conditioned, then GMRES-IR with Cholesky preconditioning can be applied to the normal equations (Higham and Pranesh 2021). More generally, the QR factorization of A can be computed using low precision arithmetic and then the augmented system solved using GMRES preconditioned by a matrix based on the low precision QR factors to obtain the

Algorithm 7.4. Solve the LS problem using GMRES-based iterative refinement with precisions $u_f \geq u \geq u_r$
Input: Overdetermined full-rank matrix A, vector b.
Output: Least-squares solution x and least-squares residual r.

1: Compute QR factorization of A using precision u_f.
2: Store the R factor using precision u_f.
3: Compute an initial solution $x^{(1)}$ and residual $r^{(1)}$ using precision u
4: Set $y^{(1)} = \begin{pmatrix} r^{(1)} \\ x^{(1)} \end{pmatrix}$.
5: **for** $i = 1, 2, \ldots$ **do**
6: Compute $s^{(i)} = \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} I & A \\ A^\top & 0 \end{pmatrix} y^{(i)} \equiv \begin{pmatrix} b \\ 0 \end{pmatrix} - K y^{(i)}$ in precision u_r.
7: Store $s^{(i)}$ in precision u.
8: Apply preconditioned GMRES to correction equation $K\delta y^{(i)} = s^{(i)}$ using precision u, with matrix–vector products computed using precision u_r.
9: Compute $y^{(i+1)} = y^{(i)} + \delta y^{(i)}$ using precision u.
10: If converged then return $x = x^{(i+1)}$, $r = r^{(i+1)}$ and stop.
11: **end for**

least-squares solution to working precision (Carson *et al.* 2020); see Algorithm 7.4. It can be shown under reasonable assumptions that, with an appropriate choice of preconditioner, this approach yields a forward least-squares error, and a backward error for the augmented system, of the order of the working precision.

A recent study by Carson and Daužickaitė (2024) discusses the theoretical and practical aspects of three two-precision iterative refinement methods, and proposes a new approach that is based on the augmented system and involves solving three least-squares problems at each refinement iteration to obtain the corrections.

8. Updating techniques and sparse–dense problems

In some applications it may be convenient to split the rows of the system matrix A into disjoint sets. A practical example in which the rows are naturally split is the addition of rows as a result of incorporating new data into the least-squares estimation of parameters in a linear model (or, conversely, the removal of rows because variables are removed from the problem). Updating procedures seek to handle such instances by updating the solution of the original problem, rather than recomputing the factorization or updating the factorization to incorporate the effects of the extra rows. The history of updating algorithms for adjusting a least-squares solution when new equations are added dates back to Gauss; see Gauss and Stewart (1995) and the recent discussion in Magnus (2022).

Another example of splitting the rows of A that is often faced in practice is when some rows contain significantly more non-zeros than the other rows. The latter are referred to as sparse rows and the former as dense rows (although they may contain far fewer than n entries). Dense rows need to be handled carefully. If A contains just a single dense row, then the normal matrix $A^\top A$ suffers catastrophic fill-in, and if n is large, it may not be possible to store or factorize it using a direct solver. Furthermore, it may be difficult (as well as computationally expensive) to construct an IC factorization. Clearly, an IC factorization with sparsity pattern-based dropping that includes the lower triangular part of $\mathcal{S}(A^\top A)$ in the prescribed pattern is not feasible if $A^\top A$ is close to dense. Furthermore, threshold-based dropping is a poor strategy if there are many non-zero entries that have to be discarded. The error in the factorization can be so large as to prohibit its effectiveness as a preconditioner. A discussion of the interplay of structure-based and threshold-based dropping is given in Scott and Tůma (2011).

Other modifications to least-squares problems are possible. For example, A may be modified by matrices of low rank, adding new columns to A or removing columns. Björck (2024) provides a general reference for classical modification approaches (see also Marín, Mas, Guerrero and Hayami 2017).

Assume A has m_d rows that are either to be treated as dense or they correspond to new added (or removed) sparse rows, and let us suppose that these rows have been permuted to be the last rows of A. The remaining rows are all assumed to be sparse. With a conformal partitioning of the vector b (and omitting the row permutation matrix for simplicity of notation), we have

$$\begin{aligned} A &= \begin{pmatrix} A_s \\ A_d \end{pmatrix}, \quad A_s \in \mathbb{R}^{m_s \times n},\ A_d \in \mathbb{R}^{m_d \times n}, \\ b &= \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \quad b_s \in \mathbb{R}^{m_s},\ b_d \in \mathbb{R}^{m_d}, \end{aligned} \tag{8.1}$$

where $m = m_s + m_d$, $m_s \geq n$ and $m_d \ll m_s$. The linear least-squares problem is then

$$\min_x \|b - Ax\|_2 = \min_x \left\| \begin{pmatrix} b_s \\ b_d \end{pmatrix} - \begin{pmatrix} A_s \\ A_d \end{pmatrix} x \right\|_2 . \tag{8.2}$$

It can be shown that if A is of full rank then $C_s = A_s^\top A_s$ is positive definite on the null space of A_d (Scott and Tůma 2022*b*). Note that ordering the rows so that A_d forms the last rows can always be assumed because the least-squares solution (but not necessarily the solution approaches) is independent of the row reordering. For convenience, in the remainder of this section we refer to problems of the form (8.1)–(8.2) as sparse–dense least-squares problems (although A_d is not necessarily fully dense).

The simplest way to deal with A_d is to drop it before factorizing the remaining matrix A_s and then employing the computed factors as a preconditioner for an

iterative solver applied to the original problem. This was discussed in the case of QR factorizations in Avron *et al.* (2009), with the conclusion that if adding (or even dropping) a small number of rows is considered to be a perturbation of A_s, the R factor from the QR factorization of A_s can provide an effective least-squares preconditioner for A. The targeted sophisticated approaches we discuss here attempt to either avoid the need for an iterative solver and/or seek to obtain higher-quality preconditioners by taking the rows of A_d into account in their construction. They also address the common case that on dropping A_d, the remaining matrix contains null columns. This situation occurs when the rows of A_d are dense or when A_d corresponds to added rows that contain new components of the least-squares solution x.

8.1. *Updating a Cholesky factorization*

Updating a Cholesky factorization is an efficient approach if A_d corresponds to added sparse rows. Consider the case where $C_s = A_s^\top A_s$ is SPD and the sparse Cholesky factorization of C_s has been computed before additional sparse rows are appended to A_s to give problem (8.2). The normal equations for (8.2) are given by

$$Cx = (C_s + A_d^\top A_d)\, x = c, \quad c = A_s^\top b_s + A_d^\top b_d.$$

Computing the Cholesky factorization of C involves a rank-m_d update, and a sparse Cholesky factorization update algorithm can be used (Davis and Hager 2001). This exploits and modifies the elimination tree $\mathcal{T}(C_s)$ of the normal matrix of C_s. Suppose A_d comprises a single sparse row, $a^\top$ ($m_d = 1$), and let i be the index of the first non-zero in $a^\top$. A rank-one update to C_s modifies all columns along the path from i to the root of $\mathcal{T}(C_s)$. If the sparsity pattern changes, the path in the new tree is followed. The entire algorithm (finding the path, modifying both $\mathcal{S}(L_s)$ and the values of L_s, and modifying $\mathcal{T}(C_s)$) takes time proportional to the number of entries in L_s that change. This can be extended to an asymptotically optimal rank-m_d update that modifies a set of m_d paths in the tree. A state-of-the-art implementation for general sparse SPD systems is available in the software package CHOLMOD[37] (Chen *et al.* 2008). It uses the concept of dynamically defined supernodes. It also includes downdating a Cholesky factorization in which rows are removed, rather than added.

8.2. *Updating a QR factorization*

Using a QR factorization involves the factorization of the sparse row block A_s and not the factorization of the normal matrix, and so the rows of A_d can be either sparse or dense. Assume that A_s is of full rank and that its QR factorization with column

[37] https://people.engr.tamu.edu/davis/suitesparse.html

Algorithm 8.1. Updating QR for solving problem (8.2)
Input: A and b of the form (8.1), the factorization (8.3) and the solution y of (8.4).
Output: Least-squares solution x.

1: Solve $P_s R_s^\top K_d^\top = A_d^\top$ for K_d. ▹ Triangular solve
2: Solve $R_s P_s^\top y = c_s$ for y and form $r_d = b_d - A_d y$. ▹ Triangular solve and residual computation
3: Compute least-squares minimum-norm solution of

$$\begin{pmatrix} K_d & I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = r_d. \quad \triangleright \text{ Use dense linear algebra}$$

4: Solve $R_s P_s^\top z = u$ for z and set $x = y + z$. ▹ Triangular solve

permutation and formal embedding of the right-hand side vector b_s is given by

$$\begin{pmatrix} A_s P_s & b_s \end{pmatrix} = Q_s \begin{pmatrix} R_s & c_s \\ 0 & d_s \end{pmatrix}, \tag{8.3}$$

where $P_s \in \mathbb{R}^{n\times n}$ represents the column permutation of A_s, $Q_s \in \mathbb{R}^{m_s \times m_s}$ is an orthogonal matrix and $R_s \in \mathbb{R}^{m_s \times m_s}$ is an upper triangular matrix. Furthermore, let $y \in \mathbb{R}^n$ be the solution of the sparse least-squares problem

$$\min_y \|b_s - A_s y\|_2. \tag{8.4}$$

The solution x of (8.2) can be computed using Algorithm 8.1 (Heath 1982). Observe that the least-squares minimum-norm problem in line 3 is the same as the following least-squares problem of size $(m_d + n) \times m_d$:

$$\min_u \left\| \begin{pmatrix} 0 \\ r_d \end{pmatrix} - \begin{pmatrix} K_d \\ I \end{pmatrix} v \right\|_2, \quad u = K_d^\top v,$$

with normal equations $(I + K_d K_d^\top)v = r_d$. The solution can be efficiently computed using dense linear algebra (e.g. the LAPACK routine `_getsls`).

If A_s is close to being rank-deficient then Algorithm 8.1 is not stable. In this case, QR updates can be applied using the techniques of Section 4.5 that handle the QR factorization of rank-deficient problems. Once the factorization is computed, the conditioning of R should be checked, and, if necessary, further rows added and rotated into R using Givens rotations. The well-conditioned R factor of the extended matrix can be employed as a preconditioner. In exact arithmetic, the number of appended rows bounds the number of LSQR iterations (Avron *et al.* 2009).

For strongly overdetermined problems ($m \gg n$), a Gram–Schmidt-based QR factorization that generates R and only the first n columns of Q (rather than the whole of Q) may be a more suitable approach, which can also be easily updated.

8.3. *Handling null columns in A_s*

As already observed, it frequently happens that A_d contains a few columns that correspond to new solution components that are not represented in A_s. That is, A_s, after being combined with A_d to give A, has one or more null columns and these cause Algorithm 8.1 to break down. The problem can be solved as follows. Assume $\operatorname{rank}(A) = n$ and A_s has n_2 null columns with $n_2 \ll n$. If these are permuted to be the final columns and x is partitioned conformally, then we have

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix} = \begin{pmatrix} A_{s_1} & 0 \\ A_{d_1} & A_{d_2} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{8.5}$$

where $\operatorname{rank}(A_1) = n_1$, $A_{d_1} \in \mathbb{R}^{m_d \times n_1}$ and $A_{d_2} \in \mathbb{R}^{m_d \times n_2}$ are obtained from column split of A_d, and $x_1 \in \mathbb{R}^{n_1}$, $x_2 \in \mathbb{R}^{n_2}$ with $n_1 + n_2 = n$. Let $z \in \mathbb{R}^{n_1}$ and $Z \in \mathbb{R}^{n_1 \times n_2}$ be the solutions to the problems

$$\min_z \|b - A_1 z\|_2 \quad \text{and} \quad \min_W \|A_2 - A_1 Z\|_F, \tag{8.6}$$

respectively, where $\|\cdot\|_F$ denotes the Frobenius norm. It can be shown (Scott and Tůma 2017*b*) that the solution to the least-squares problem (8.1) is given by

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z - Z x_2 \\ x_2 \end{pmatrix},$$

where x_2 is the solution of the least-squares problem

$$\min_{x_2} \|(b - A_1 z) - (A_2 - A_1 Z) x_2\|_2.$$

Because n_2 is small, this can be solved using dense linear algebra. The partial solutions z and Z can be computed by employing Algorithm 8.1 with A_1 replacing A and $n_2 + 1$ right-hand sides.

8.4. *Updating by complete or approximate solution of an SQD system*

The solution of the linear least-squares problem expressed in the form (8.2) can be computed by solving the equivalent $(n + m_d) \times (n + m_d)$ symmetric indefinite system

$$\begin{pmatrix} C_s & A_d^\top \\ A_d & -I \end{pmatrix} \begin{pmatrix} x \\ A_d x \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix}, \quad C_s = A_s^\top A_s, \quad c = A_s^\top b_s + A_d^\top b_d. \tag{8.7}$$

Provided A_s has full column rank, this is a symmetric quasi-definite system. If L_s denotes the Cholesky factor of C_s, then the signed Cholesky factorization is

$$\begin{pmatrix} C_s & A_d^\top \\ A_d & -I \end{pmatrix} = \begin{pmatrix} L_s & \\ B_d & L_d \end{pmatrix} \begin{pmatrix} I & \\ & -I \end{pmatrix} \begin{pmatrix} L_s^\top & B_d^\top \\ & L_d^\top \end{pmatrix}, \tag{8.8}$$

where $L_s B_d^\top = A_d^\top$ and $S_d = I + B_d B_d^\top = L_d L_d^\top$. Here L_d is the (dense) Cholesky factor of the $m_d \times m_d$ Schur complement S_d. Algorithm 8.2 summarizes the steps

Algorithm 8.2. Block factorization approach for sparse–dense problems
Input: B_d, the Cholesky factors L_s and L_d, and $c = A_s^\top b_s + A_d^\top b_d$.
Output: Least-squares solution x.

1: Solve $L_s u_s = c$.	▹ Sparse triangular solve
2: Compute $w_d = B_d u_s$.	
3: Solve $L_d u_d = w_d$ and then $L_d^\top y_d = u_d$.	▹ Dense triangular solves
4: Form $w_s = u_s - B_d^\top y_d$.	
5: Solve $L_s^\top x = w_s$.	▹ Sparse triangular solve

to compute the least-squares solution once L_s, B_d and L_d have been computed. If A_d represents a set of rows that have been appended to the original matrix A_s then this algorithm can be viewed as an updating procedure in which the normal matrix factorization remains fixed, and lines 2–4 represent the additional work involved. As in the previous section, if A_s contains null columns then (8.5) and (8.6) can be used with Algorithm 8.2.

Another possibility is to compute an approximate solution of the SQD system. It is straightforward to verify that the normal matrix satisfies the following relationship:

$$(C_s + A_d^\top A_d)^{-1} = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} C_s & A_d^\top \\ A_d & -I \end{pmatrix}^{-1} \begin{pmatrix} I \\ 0 \end{pmatrix}.$$

This implies that the inverse of the updated normal matrix applied to a given vector $z \in \mathbb{R}^n$ to obtain the solution vector $y \in \mathbb{R}^n$ as $y = (C_s + A_d^\top A_d)^{-1} z$ can be computed from the solution of the system

$$\begin{pmatrix} C_s & A_d^\top \\ A_d & -I \end{pmatrix} \begin{pmatrix} y \\ w \end{pmatrix} = \begin{pmatrix} z \\ 0 \end{pmatrix}. \tag{8.9}$$

If $C_s \approx \widetilde{L}_s \widetilde{L}_s^\top$ is an IC factorization then an incomplete version of (8.8) can be employed to give a preconditioner for use with an iterative solver. If C_s is rank-deficient (including the case that it has some null rows and columns) then a global shift $\alpha > 0$ can be used and the factorization of $C_s + \alpha I$ computed (Section 6.5). Exploiting (8.9), each application of the preconditioner involves using a modified version of Algorithm 8.2 in which the complete factors are replaced by the incomplete ones. Alternatively, the techniques discussed in Sections 6.13–6.15 can also be used for solving the SQD system.

Because the matrix in (8.7) is symmetric indefinite, an obvious approach is to employ an existing sparse direct solver to compute an LDLT factorization (Section 5). The major advantage of this is that it is straightforward: all the work of ordering the matrix to limit fill in the factors (including exploiting any zero entries within the rows of A_d) and ensuring numerical stability is handled by the solver. However, the 2-block structure of the matrix is ignored and the factorization cannot be easily updated if different row blocks A_d are appended to A_s; but see some early

work on rank-one updates (Sorensen 1977) and a recent strategy for a sequence of saddle-point systems with the same sparsity pattern (Kuřátko 2019).

8.5. *Updating by solving an SQD system with an embedded QR factorization of A_s*

The problem of updating can be also formulated via the solution of an augmented system in which the QR factorization of A_s is embedded. This can be useful, for example, for iterative refinement that was discussed in Section 7.2 and which we extend here to problems with split A.

Consider the following $(m_s + m_d + n) \times (m_s + m_d + n)$ augmented system formulation of the least-squares problem used in the correction equation of the iterative refinement given in (7.1). Assume A is of the form (8.1) and that we have a conformal partitioning of u and w so that

$$\begin{pmatrix} I & & A_s \\ & I & A_d \\ A_s^T & A_d^T & 0 \end{pmatrix} \begin{pmatrix} u_s \\ u_d \\ v \end{pmatrix} = \begin{pmatrix} w_s \\ w_d \\ t \end{pmatrix}. \tag{8.10}$$

Using the QR factorization

$$\begin{pmatrix} A_s P_s & w_s \end{pmatrix} = Q_s \begin{pmatrix} R_s & c_s \\ 0 & d_s \end{pmatrix}$$

yields

$$\begin{pmatrix} I & & & R_s \\ & I & & 0 \\ & & I & A_d P_s \\ R_s^T & 0 & P_s^T A_d^T & 0 \end{pmatrix} \begin{pmatrix} e_s \\ f_s \\ u_d \\ P_s^T v \end{pmatrix} = \begin{pmatrix} c_s \\ d_s \\ w_d \\ P_s^T t \end{pmatrix},$$

where

$$\begin{pmatrix} c_s \\ d_s \end{pmatrix} = Q_s^T w_s \quad \text{and} \quad u_s = Q_s \begin{pmatrix} e_s \\ f_s \end{pmatrix} = Q_s \begin{pmatrix} e_s \\ d_s \end{pmatrix}. \tag{8.11}$$

Setting $z = R_s P_s^T v$ and $P_s R_s^T K_d^T = A_d^T$, we have

$$\begin{pmatrix} I & & I \\ & I & K_d \\ I & K_d^T & 0 \end{pmatrix} \begin{pmatrix} e_s \\ u_d \\ z \end{pmatrix} = \begin{pmatrix} c_s \\ w_d \\ p \end{pmatrix},$$

where $P_s R_s^T p = t$. The first block gives $z = c_s - e_s$, and eliminating e_s from the above equation gives

$$\begin{pmatrix} I & K_d \\ K_d^T & -I \end{pmatrix} \begin{pmatrix} u_d \\ z \end{pmatrix} = \begin{pmatrix} w_d \\ p - c_s \end{pmatrix}.$$

This is another SQD system, and to solve it we can use techniques from Section 8.4 with C_s replaced by I. Eliminating z, we can also use the transformed normal

equations

$$(I + K_d K_d^\top)\, u_d = w_d + K_d(p - c_s).$$

Once u_d has been computed, v is found by forming $e_s = p - K_d^\top u_d$, then solving $R_s P_s^\top v = c_s - e_s$, and finally computing u_s from (8.11).

Observe that K_d is independent of the right-hand side vector in (8.10) and so it can be reused, limiting the amount of work needed to perform iterative refinement. For small m_d, it is clear that the work per iteration for the sparse–dense case is essentially the same as for A with no dense rows.

If we simply need to solve the least-squares problems with extra rows A_d (and not the iterative refinement correction equations); this can be done by setting $t = 0$ in (8.10), which implies $p = 0$.

8.6. *Updating using the Woodbury formula*

A standard tool when the factors of C_s are known and one or more rows A_d is appended is the Woodbury formula; see the classical papers by Woodbury (1949, 1950) and the review article by Hager (1989). Assume A_s is of full column rank and is well-conditioned. Then this formula, which is sometimes referred to as the Sherman–Morrison–Woodbury formula, expresses the inverse of the extended normal matrix $C = A^\top A$ in the compact form

$$C^{-1} = (C_s + A_d^\top A_d)^{-1} = C_s^{-1} - C_s^{-1} A_d^\top (I + A_d C_s^{-1} A_d^\top)^{-1} A_d C_s^{-1}. \tag{8.12}$$

The least-squares solution may be explicitly expressed as

$$x = x_s + C_s^{-1} A_d^\top (I + A_d C_s^{-1} A_d^\top)^{-1}(b_d - A_d x_s) \quad \text{with} \quad x_s = C_s^{-1} A_s^\top b_s.$$

It is straightforward to show that, in exact arithmetic, Algorithm 8.2 is equivalent to the Woodbury formula.

The formula (8.12) leads to a direct method for updating the least-squares solution. Alternatively, if an IC factorization $C_s \approx \widetilde{L}_s \widetilde{L}_s^\top$ is computed, then $I + A_d C_s^{-1} A_d \approx I + (A_d \widetilde{L}_s^{-1})(A_d \widetilde{L}_s^{-1})^\top = \widetilde{L}_d \widetilde{L}_d^\top$. The approximate inverse of the updated normal matrix expressed via the Woodbury formula using two incomplete factorizations provides a preconditioner for the conjugate gradient method (Scott and Tůma 2017*b*). Recently, a preconditioning strategy that is derived from an alternating splitting scheme and combined with (8.12) has been proposed and successfully applied to solve sparse–dense least-squares problems (Benzi and Faccio 2024).

8.7. *Matrix stretching*

Matrix stretching targets the case when A_d is composed of rows that are significantly denser than the rest of the rows of A. It aims to split each dense row into a number of sparser rows and to formulate a (larger) modified problem from which the solution to the original least-squares problem can be derived (Adlers and Björck 2000).

Suppose A_d represents a single dense row, $a^\top$. Stretching starts by splitting $a^\top$ into two $a^\top = (a_1^\top \;\; a_2^\top)$. Assume a conformal splitting of the sparse row block A_s and the solution x. Then, introducing a linking variable s, the x component of the solution of the extended least-squares problem

$$\min_{(x^\top \; s)^\top} \left\| \begin{pmatrix} b_s \\ b_d \\ 0 \end{pmatrix} - \begin{pmatrix} A_{s1} & A_{s2} & 0 \\ a_1^\top & a_2^\top & 0 \\ a_1^\top & -a_2^\top & \sqrt{2} \end{pmatrix} \begin{pmatrix} x_a \\ x_b \\ s \end{pmatrix} \right\|_2$$

is the solution of (8.1). Next, an orthogonal transformation is applied to replace $a_2^\top$ in the second block row and $a_1^\top$ in the third block row by zeros. Orthogonal invariance of the norm leads to the equivalent stretched problem

$$\min_z \|\hat{b} - \widehat{A} z\|_2,$$

where

$$\widehat{A} = \begin{pmatrix} A_{sa} & A_{sb} & 0 \\ \sqrt{2}\, a_1^\top & 0 & 1 \\ 0 & \sqrt{2}\, a_2^\top & -1 \end{pmatrix}, \quad z = \begin{pmatrix} x_a \\ x_b \\ s \end{pmatrix}, \quad \widehat{b} = \begin{pmatrix} b_s \\ b_d/\sqrt{2} \\ b_d/\sqrt{2} \end{pmatrix}.$$

The approach can be generalized by splitting the dense row into $k > 1$ parts, resulting in a stretched problem in which $\widehat{A}$ and the normal matrix $\widehat{A}^\top \widehat{A}$ are of the form

$$\widehat{A} = \begin{pmatrix} A_s & \\ B^\top & S \end{pmatrix} \quad \text{and} \quad \widehat{C} = \widehat{A}^\top \widehat{A} = \begin{pmatrix} A_s^\top A_s + BB^\top & BS \\ S^\top B^\top & S^\top S \end{pmatrix},$$

where $B^\top \in \mathbb{R}^{k \times n}$ and the $k \times (k-1)$ linking matrix S has 1s on the diagonal and -1s on the first subdiagonal (and all other entries are 0). If $m_d > 1$, then each dense row can be stretched independently.

Stretching replaces the effect of adding $\mathcal{S}\{A_d^\top A_d\}$ to $\mathcal{S}\{A_s^\top A_s\}$ to obtain the sparsity pattern of $A^\top A$ by adding the sparsity pattern of $(B^\top \; S)^\top (B^\top \; S)$ to $\mathcal{S}\{A_s^\top A_s\}$ to get the sparsity pattern of $\widehat{C}$, while seeking to have $|\mathcal{S}\{\widehat{C}\}|$ much smaller than $|\mathcal{S}\{A^\top A\}|$. Standard stretching splits A_d into sets of (almost) equal contiguous segments without any reference to A_s. This is a simple approach but it can result in significant fill in $\widehat{C}$. A more sophisticated sparse stretching strategy considers the pattern of $A_s^\top A_s$ and, for each row in A_d, chooses the subsets of row indices in the splitting such that $\mathcal{S}(BB^\top) \subseteq \mathcal{S}(A_s^\top A_s)$, thereby limiting the number of entries in $\widehat{C}$ (Scott and Tůma 2019). Finding the subsets of row indices can be formulated as a bipartite graph matching problem.

In some cases, $\widehat{A}$ can be much larger than the original A (particularly if A_s is highly sparse) and the cost of solving the stretched problem (in terms of time and memory) may still be prohibitive. Another potential problem is that stretching increases the condition number of the normal equations (Adlers and Björck 2000, Scott and Tůma 2019). A compromise is partial stretching (Scott and Tůma 2021).

This selects a small subset of the rows of A_d that contain non-zero entries at column positions that are null in the rows of A_s. Sparse stretching is applied to each row in this subset. The stretched rows are sparse so they are added to an enlarged sparse row block $\widehat{A}_s$ that has no null columns, while the remaining dense rows are moved to a dense block $\widehat{A}_d$ that has fewer rows than A_d. The rows in $\widehat{A}_d$ can be handled by applying Algorithm 8.1 to the partially stretched problem.

9. Equality constrained least-squares problems

Least-squares problems with equality constraints (LSE problems) arise in a variety of fields, including constrained optimization, scattered data approximation, fitting curves to data, surface fitting, and in various tasks of control and communication. For instance, when fitting curves to data, equality constraints may arise from the need to interpolate some data or from a requirement for adjacent fitted curves to match with continuity. In applications such as beam-forming or spatial filtering it is necessary to solve a sequence of LSE problems in which the equality constraints change. Moreover, solving least-squares problems with more general inequality constraints can sometimes be reduced to solving sequences of LSE problems, e.g. Dehghani, Lambe and Orban (2020). Motivations for LSE problems together with solution strategies are summarized in the research monographs by Lawson and Hanson (1995) and Björck (2024); see also Scott and Tůma (2022*c*) for sparse–dense LSE problems and numerical results comparing different approaches.

We assume that $A \in \mathbb{R}^{m\times n}$ is sparse and that $B \in \mathbb{R}^{p\times n}$, with $m > n \gg p$, represents a few linear constraints (which may be sparse or dense). Given $b \in \mathbb{R}^m$ and $d \in \mathbb{R}^p$, the LSE problem is

$$\min_{x\in\mathbb{R}^n} \|b - Ax\|_2 \tag{9.1}$$

$$\text{such that} \quad Bx = d. \tag{9.2}$$

If B has full row rank then (9.2) is consistent for any d. A solution to the LSE problem exists if and only if (9.2) is consistent. It is unique if and only if $\mathcal{N}(A) \cap \mathcal{N}(B) = \{0\}$. This is equivalent to the extended matrix $(B^\top \; A^\top)^\top$ having full column rank. In the case of non-uniqueness, there is a unique minimum-norm solution. In the following, we assume that B is of full row rank.

9.1. The use of weighting

The simplest approach to solving the LSE problem is to solve the full-rank weighted least-squares problem

$$\min_{x_\omega\in\mathbb{R}^n} \left\| \begin{pmatrix} \omega d \\ b \end{pmatrix} - \begin{pmatrix} \omega B \\ A \end{pmatrix} x_\omega \right\|_2, \tag{9.3}$$

with $\omega \gg 1$. Because $\lim_{\omega\to\infty} x_\omega = x$, the weighted problem can be used to approximately solve the LSE problem. If B is sparse then an obvious strategy is to

Algorithm 9.1. The LSE problem with B sparse
Input: A, B, b and d defining the LSE problem (9.1)–(9.2) and weight $\omega > 0$.
Output: LSE solution x.

1: Solve the sparse weighted least-squares problem (9.3)
2: Set $x^{(1)} = x_\omega$
3: **for** i = 1, 2, ... **do**
4: Set $s^{(i)} = d - Bx^{(i)}$ ▷ $s^{(i)}$ is the residual of the constraints.
5: If converged then return $x = x^{(i)}$
6: Solve $\min \left\| \begin{pmatrix} s^{(i)} \\ 0 \end{pmatrix} - \begin{pmatrix} \omega B \\ A \end{pmatrix} \delta x^{(i)} \right\|_2$
▷ The factors computed in line 1 can be reused
7: Set $x^{(i+1)} = x^{(i)} + \delta x^{(i)}$
8: **end for**

either compute a QR factorization of $\binom{\omega B}{A}$ or to solve the normal equations

$$C_\omega x_\omega = \begin{pmatrix} \omega B^\top & A^\top \end{pmatrix} \begin{pmatrix} \omega B \\ A \end{pmatrix} x_\omega = (A^\top A + \omega^2 B^\top B)\, x_\omega = A^\top b + \omega^2 B^\top d.$$

The appeal is that no special methods are required. However, a very large weight may be needed for the constraints to be tightly satisfied, even for well-conditioned problems. With a large ω, the problem is stiff (recall Section 1.4), C_ω is dominated by the $\omega^2 B^\top B$ term, and information in A can potentially be lost.

A smaller ω can be successfully used by employing an iterative procedure; this is outlined in Algorithm 9.1. An appropriate weight is $\omega \approx \epsilon^{-1/2}$ (Van Loan 1985). The approach can be used even if B is not of full rank.

In practice, the constraint matrix B often contains one or more dense rows. For such problems, one possibility is to introduce a regularization parameter $\gamma > 0$ and solve the $(m + p + n) \times (m + p + n)$ augmented system

$$\begin{pmatrix} \gamma I & 0 & A \\ 0 & \gamma I & \omega B \\ A^\top & \omega B^\top & -\gamma I \end{pmatrix} \begin{pmatrix} y_s \\ y_c \\ x_\omega \end{pmatrix} = \begin{pmatrix} b \\ \omega\, d \\ 0 \end{pmatrix}.$$

This 3-block system is structurally similar to the system (8.10) and can be solved by embedding a QR factorization of A and modifying the approaches proposed in Section 8.5. Alternatively, the block structure can be ignored and a numerically stable LDLT factorization computed using a sparse symmetric indefinite direct solver. Or, eliminating y_s and choosing the parameters such that $\gamma\omega = 1$ yields

$$\begin{pmatrix} -C_\gamma & B^\top \\ B & \gamma^2 I \end{pmatrix} \begin{pmatrix} x_\omega \\ y_c \end{pmatrix} = \begin{pmatrix} -A^\top b \\ d \end{pmatrix}, \quad C_\gamma = \begin{pmatrix} A^\top & \gamma I \end{pmatrix} \begin{pmatrix} A \\ \gamma I \end{pmatrix} = A^\top A + \gamma^2 I. \quad (9.4)$$

This system is similar to that discussed in Section 8.4. A sparse QR factorization of $(A^\top \ \gamma I)^\top$ can be computed and employed in the solution of this augmented system. If more than one problem with the same A but different constraints B are to be solved, an attractive approach may be to compute a block signed (incomplete) Cholesky factorization of the augmented matrix (9.4) and employ it as a preconditioner. In particular, if $C_\gamma \approx \widetilde{L}_\gamma \widetilde{L}_\gamma^\top$ then the right-preconditioned system is

$$\begin{pmatrix} -C_\gamma & B^\top \\ B & \gamma^2 I \end{pmatrix} M^{-1} \begin{pmatrix} w_\gamma \\ w_c \end{pmatrix} = \begin{pmatrix} -A^\top b \\ d \end{pmatrix}, \quad M \begin{pmatrix} x_\omega \\ y_c \end{pmatrix} = \begin{pmatrix} w_\gamma \\ w_c \end{pmatrix},$$

with the preconditioner in factored form given by

$$M = \begin{pmatrix} \widetilde{L}_\gamma & \\ B_\gamma & I \end{pmatrix} \begin{pmatrix} -I & \\ & S_\gamma \end{pmatrix} \begin{pmatrix} \widetilde{L}_\gamma^\top & B_\gamma^\top \\ & I \end{pmatrix},$$

with

$$\widetilde{L}_\gamma B_\gamma^\top = -B^\top \quad \text{and} \quad S_\gamma = \gamma^2 I + B_\gamma B_\gamma^\top.$$

As the preconditioner is indefinite, it needs to be used with a general non-symmetric iterative method based on full recurrences such as GMRES. A positive definite preconditioner for use with MINRES can be obtained by replacing $-I$ with I.

9.2. *The null-space approach*

There are two standard ways to derive an unconstrained linear least-squares problem of lower dimension that is equivalent to the LSE problem: the null-space approach (Hanson and Lawson 1969, Lawson and Hanson 1995) and the method of direct elimination (Björck and Golub 1967). When suitably implemented, both offer good numerical stability, but retaining sparsity is challenging and may compromise stability.

The null-space approach is based on constructing a matrix $Z \in \mathbb{R}^{n\times(n-p)}$ whose columns form a basis for $\mathcal{N}(B)$. Any $x \in \mathbb{R}^n$ satisfying the constraints can be written in the form

$$x = x_1 + Z x_2,$$

where $x_1 \in \mathbb{R}^n$ is a particular solution of the underdetermined system (9.2). Substituting into (9.1) yields the reduced least-squares problem

$$\min_{x_2 \in \mathbb{R}^{n-p}} \|(b - Ax_1) - AZx_2\|_2. \tag{9.5}$$

An overview of the use of the null-space approach for solving large-scale saddle-point systems is given in Rees and Scott (2018). Using null-space methods leads to the problem of how to compute null-space bases that preserve sparsity and lead to a stable transformed system. Significant attention has been devoted to developing approaches to find a sparse null-space basis of a sparse matrix; a brief historical review is given in Scott and Tůma (2022*b*).

For the LSE problem, $B \in \mathbb{R}^{p\times n}$ with $p \ll n$, so that B is a wide matrix. One possibility is to compute the QR factorization of $B^\top$, that is,

$$B^\top = (Q_1 \;\; Q_2)\begin{pmatrix} R_B \\ 0 \end{pmatrix}.$$

Here $Q_1 \in \mathbb{R}^{n\times p}$ while $Q_2 \in \mathbb{R}^{n\times(n-p)}$ gives an orthogonal basis for $\mathcal{N}(B)$ and we can choose $Z = Q_2$. If B is of full rank, the upper triangular matrix $R_B \in \mathbb{R}^{p\times p}$ is non-singular and hence $x_1 = Q_1 R_B^{-\top} d$. Unfortunately, this Z is typically dense and AZ is much denser than A and, if n is large, solving (9.5) is challenging. In particular, forming and factorizing the potentially ill-conditioned normal matrix $Z^\top A^\top AZ$ may be impractical. If a preconditioned iterative solver is used, forming $Z^\top A^\top AZ$ can be avoided, and because Z only needs to be applied implicitly, the need for sparsity can be relaxed. However, finding a good and sufficiently general robust preconditioner for this system remains an open problem.

Another option is to construct a banded Z by exploiting the fact that each column of B that is linearly dependent on previous columns can be written as a linear combination of at most p of these columns. Consider the following wide 2×7 matrix and its null-space basis Z:

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 1 & 6 \\ 3 & 4 & 7 & 10 & 11 & 1 & 12 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 1 & & & & \\ 1 & & 1 & & & \\ -1 & 1 & 1 & & & \\ & -1 & & -1 & & \\ & & -1 & 1 & 1 & \\ & & & -1 & 1 & \\ & & & & -1 \end{pmatrix}.$$

Here column 5 of B is the sum of columns 2 and 3. This linear dependence can be expressed by a vector in $\mathbb{R}^7$ that has non-zero entries in positions 2, 3 and 5 only. More generally, expressing each linearly dependent column j of B as a linear combination of previous columns whose column indices i are as close to j as possible results in Z with a band structure. The dependences can be determined stably using QR factorizations of submatrices of B; the incorporation of pivoting can be used to balance stability of the factorization with limiting the bandwidth. This is discussed in Scott and Tůma (2022*b*); recent work on solving LSE problems based on this approach is presented in Scott and Tůma (2022*c*). While this strategy can be useful, in general the columns of the computed Z are not orthogonal and, in some cases, the norm of the constraints residual may be larger than is desirable.

9.3. The method of direct elimination

The basic idea is to express the dependence of p selected components of the solution vector x on the remaining $n - p$ components and to substitute this into the least-squares problem (9.1). The p components need to be chosen to retain sparsity

in the transformed problem. The method starts by permuting and splitting so that

$$Bx = BPy = \begin{pmatrix} B_1 & B_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = d,$$

where the permutation $P \in \mathbb{R}^{n \times n}$ is chosen to ensure that $B_1 \in \mathbb{R}^{p \times p}$ is non-singular. Let $AP = (A_1 \quad A_2)$ be a conformal partitioning of A. Substituting the expression

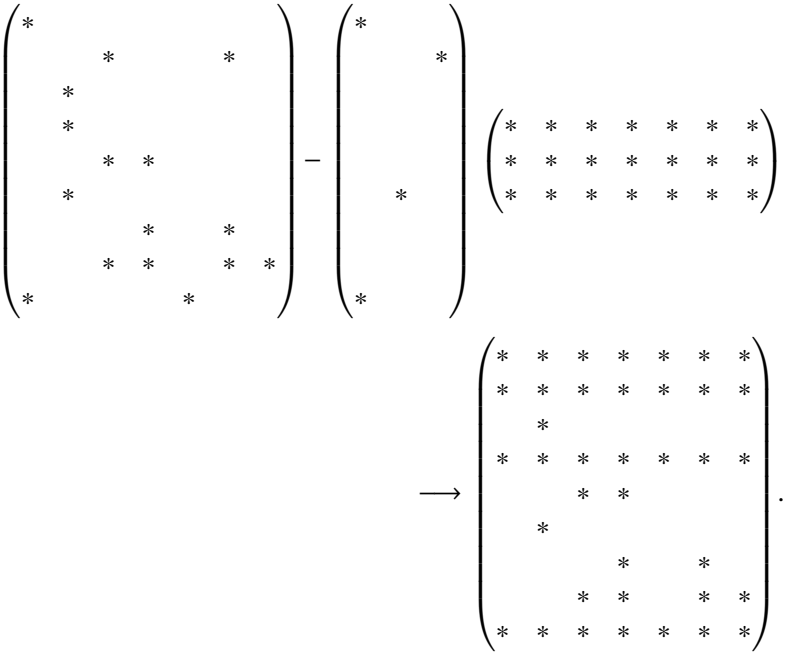
$$y_1 = B_1^{-1}(d - B_2\, y_2) \in \mathbb{R}^p$$

into (9.1) gives the transformed least-squares problem

$$\min_{y_2} \|(b - A_1 B_1^{-1} d) - A_T\, y_2\|_2, \quad A_T = A_2 - A_1 B_1^{-1} B_2 \in \mathbb{R}^{m \times (n-p)} \tag{9.6}$$

for the remaining $(n - p)$ solution components.

Note that if B_1 is irreducible, the transformation combines all the rows of B_2. If both A and B are sparse then A_T is sparse, but if B contains dense rows then A_T has more dense rows than A and (9.6) is a sparse–dense least-squares problem. This is illustrated by the following example, in which $m = 9$, $p = 3$, $n = 7$:



The matrices (from the left) represent the transformation $A_T = A_2 - A_1 B_1^{-1} B_2$; the matrix $B_1^{-1} B_2 \in \mathbb{R}^{p \times n}$ is depicted as fully dense. In this instance, A_T has four dense rows.

The permuting and splitting of B cannot be separated from consideration of $\mathcal{S}(A)$ because the splitting also determines A_1 and A_2. Thus there needs to be a balance

Algorithm 9.2. Straightforward updating approach based on Lagrange multipliers for the LSE problem with B having full row rank
Input: A, B, b and d defining the LSE problem (9.1)–(9.2).
Output: LSE solution x.

1: Solve the sparse unconstrained least-squares problem $\min_y \|b - Ay\|^2$
2: Solve $A^\top AJ = B^\top$ for $J \in \mathbb{R}^{n\times p}$
3: Set $Y = BJ$ and solve $Y\lambda = By - d$ ▷ $Y \in \mathbb{R}^{p\times p}$ is SPD.
4: Set $x = y - J\lambda$

between ensuring there is a sufficiently well-conditioned factorization of B_1 while limiting the number of dense rows in A_T (Scott and Tůma 2022*c*).

9.4. Lagrange multiplier approach

Complementary approaches for solving the LSE problem are based on substitution from the unconstrained least-squares problem into the constraints. The Lagrangian for (9.1)–(9.2) is

$$\mathcal{L}(x, \lambda) = \|b - Ax\|_2^2 + 2\lambda^\top(d - Bx),$$

where $\lambda \in \mathbb{R}^p$ is the vector of Lagrange multipliers. Setting the partial derivatives to zero gives the first-order optimality condition

$$A^\top(b - Ax) + B^\top\lambda = 0 \quad \text{and} \quad d - Bx = 0.$$

Combining with the residual equation $r = b - Ax$ yields a 3-block augmented system

$$\begin{pmatrix} 0 & A^\top & B^\top \\ A & I & 0 \\ B & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ r \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ b \\ d \end{pmatrix}. \tag{9.7}$$

The x component is the solution of the LSE problem. Again, the system (9.7) can be solved using an LDLT factorization. Alternatively, eliminating r we have a 2-block system

$$\begin{pmatrix} A^\top A & B^\top \\ B & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} A^\top b \\ d \end{pmatrix}.$$

Algorithm 9.2 is a straightforward updating scheme for computing x that can be used whether or not the rows of B are dense. Any appropriate direct or iterative method can be used for line 1, which is usually the most expensive part of the computation. The method used to solve the system with a block of p right-hand sides in line 2 may be chosen to exploit line 1; for example, a complete or incomplete Cholesky factorization of the normal matrix $A^\top A$ can be reused. Line 3 involves a small dense system.

A numerically superior but more expensive method that avoids both forming the normal matrix and computing the multipliers λ can be derived using a QR factorization of A and modifying the updating approach of Section 8.2.

10. Summary and outlook

In this article, we have sought to provide an overview of modern approaches for solving large-scale linear least-squares problems. In particular, having briefly introduced important tools used in sparse matrix technology, we have addressed the following topics.

- Sparse direct methods for (i) the symmetric positive definite but possibly highly ill-conditioned normal equations, (ii) the mathematically equivalent symmetric indefinite augmented system, and (iii) the QR factorization of the least-squares matrix.
- The development of algebraic preconditioners for use with Krylov subspace methods for least-squares problems.
- The use of iterative refinement for least-squares problems.
- Updating techniques, including methods for solving sparse–dense problems that arise in many practical applications.
- Linear least-squares problems with equality constraints.

We are conscious that it is not possible to cover all aspects of such a wide field in one paper and so we have aimed to provide a self-contained introduction to the fundamental ideas and concepts, along with useful references where the underlying theory can be found, and pointers to the available software. To cite all the relevant publications would increase the bibliography to thousands of items; an extensive bibliography of over 1100 historical and recent references is included within the new book by Björck (2024). We have chosen in the main to reference key papers, books, research monographs and survey articles that themselves include many other useful references, as well as citing recent publications related to algorithms and theory for solving least-squares problems that are not yet so well-known.

We have included basic outline algorithmic descriptions using pseudocode that is independent of any programming language, together with simple examples to provide an insight into sparse factorization techniques. The complex implementation details that are needed in the development of high-quality sophisticated (parallel) production software are outside the scope of the paper. Indeed, the design and development of general-purpose high-quality software for efficiently solving sparse least-squares problems using modern computer architectures remains a challenge.

Despite the fact that solving linear least-squares problems is a mature scientific field that is employed in many practical applications, there remain significant theoretical as well as computational challenges. One such challenge is the development

of robust preconditioners, with underlying theory and efficient implementations. In the case of general-purpose Cholesky factorization methods, novel regularization techniques seamlessly combined with shifting strategies may be needed. QR-based preconditioners are even less well-developed. We have mentioned their relation to memory-limited incomplete Cholesky factorizations; this relationship may lead to better understanding and new insights and developments. The techniques for solving large-scale sparse–dense problems may potentially be extended to more general problems where the interaction of partially sparse structures in the matrix of the normal equations may lead to efficient preconditioners (Scott and Tůma 2021). The use of mixed precision arithmetic in the development of preconditioners and their use in iterative solvers has only recently started to be considered, but may offer significant future potential (Carson *et al.* 2020, Higham and Pranesh 2021, Scott and Tůma 2022*a*, 2024, 2025).

While much research has been devoted to robust stopping criteria for preconditioned iterative solvers (Chang, Paige and Titley-Peloquin 2009, Papež and Tichý 2023), for least-squares problems originating from discretized ill-posed problems, an excessive number of iterations can lead to overfitting and thus useless approximations (Hansen 1998). In some practical applications (such as the extremely large problems that arise in weather forecasting), early stopping (possibly after a fixed number of iterations) is essential because of real-time constraints. It may be possible to resolve the crucial task of determining early stopping by exploiting *a priori* information, such as statistical properties of noise and solution smoothness (Havelková and Hnětynková 2023). However, further work is needed to better understand the interplay between stopping criteria and other parameters.

Finally, many challenges are connected to embedding linear least-squares solvers into more general problems, including nonlinear least-squares problems, least-squares problems with inequality constraints, and the efficient solution of sequences of least-squares problems.

Acknowledgements

We would like to thank Petr Tichý for comments on iterative methods used to solve least-squares problems and Iveta Hnětynková for reading a draft manuscript.

Jennifer Scott is grateful for funding over many years from the Engineering and Physical Sciences Research Council, the Science and Technology Facilities Research Council, and the University of Reading, and would like to thank both former and current collaborators at the Rutherford Appleton Laboratory.

Miroslav Tůma thanks his colleagues for the stimulating atmosphere at his current workplace, the Faculty of Mathematics and Physics of Charles University.

References

M. Adlers and Å. Björck (2000), Matrix stretching for sparse least squares problems, *Numer. Linear Algebra Appl.* **7**, 51–65.

E. Agullo, A. Buttari, A. Guermouche and F. Lopez (2016), Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems, *ACM Trans. Math. Softw.* **43**, art. 13.

M. A. Ajiz and A. Jennings (1984), A robust incomplete Choleski-conjugate gradient algorithm, *Int. J. Numer. Methods Engrg* **20**, 949–966.

H. Al Daas and L. Grigori (2019), A class of efficient locally constructed preconditioners based on coarse spaces, *SIAM J. Matrix Anal. Appl.* **40**, 66–91.

H. Al Daas, P. Jolivet and J. A. Scott (2022), A robust algebraic domain decomposition preconditioner for sparse normal equations, *SIAM J. Sci. Comput.* **44**, A1047–A1068.

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

P. R. Amestoy, T. A. Davis and I. S. Duff (2004), Algorithm 837: AMD, an approximate minimum degree ordering algorithm, *ACM Trans. Math. Softw.* **30**, 381–388.

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

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

M. Arioli, I. S. Duff and P. P. M. de Rijk (1989), On the augmented system approach to sparse least-squares problems, *Numer. Math.* **55**, 667–684.

S. R. Arridge, M. M. Betcke and L. Harhanen (2014), Iterated preconditioned LSQR method for inverse problems on unstructured grids, *Inverse Problems* **30**, art. 075009.

C. Ashcraft (1995), Compressed graphs and the minimum degree algorithm, *SIAM J. Sci. Comput.* **16**, 1404–1411.

C. Ashcraft and R. Grimes (1989), The influence of relaxed supernode partitions on the multifrontal method, *ACM Trans. Math. Softw.* **15**, 291–309.

H. Avron, E. Ng and S. Toledo (2009), Using perturbed *QR* factorizations to solve linear least-squares problems, *SIAM J. Matrix Anal. Appl.* **31**, 674–693.

J. Baglama and D. J. Richmond (2014), Implicitly restarting the LSQR algorithm, *Electron. Trans. Numer. Anal.* **42**, 85–105.

Z.-Z. Bai and J.-F. Yin (2009), Modified incomplete orthogonal factorization methods using Givens rotations, *Computing* **86**, 53–69.

G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, N. Knight and H. D. Nguyen (2015), Reconstructing Householder vectors from tall-skinny QR, *J. Parallel Distrib. Comput.* **85**, 3–31.

J. L. Barlow (2013), Reorthogonalization for the Golub–Kahan–Lanczos bidiagonal reduction, *Numer. Math.* **124**, 237–278.

S. T. Barnard, A. Pothen and H. D. Simon (1995), A spectral algorithm for envelope reduction of sparse matrices, *Numer. Linear Algebra Appl.* **2**, 317–334.

R. H. Bartels, G. H. Golub and M. A. Saunders (1970), Numerical techniques in mathematical programming, in *Nonlinear Programming (Proc. Sympos. Univ. Wisconsin–Madison)*, Academic Press, pp. 123–176.

S. Bellavia, J. Gondzio and B. Morini (2013), A matrix-free preconditioner for sparse symmetric positive definite systems and least-squares problems, *SIAM J. Sci. Comput.* **35**, A192–A211.
S. J. Benbow (1999), Solving generalized least-squares problems with LSQR, *SIAM J. Matrix Anal. Appl.* **21**, 166–177.
C. Benoit (1924), Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés à un système d'équations linéaires en nombre inférieur à celui des inconnues: Application de la méthode à la résolution d'un systeme défini d'équations linéaires, *Bulletin Géodésique* **2**, 5–77.
A. R. Benson, D. F. Gleich and J. Demmel (2013), Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures, in *2013 IEEE International Conference on Big Data*, IEEE, pp. 264–272.
M. Benzi (2004), Gianfranco Cimmino's contributions to numerical mathematics. *Atti del Seminario di Analisi Matematica dell'Università di Bologna*, pp. 87–109.
M. Benzi and C. Faccio (2024), Solving linear systems of the form $(A + \gamma UU^T)x = b$ by preconditioned iterative methods, *SIAM J. Sci. Comput.* **46**, S51–S70.
M. Benzi and M. Tůma (2003*a*), A robust incomplete factorization preconditioner for positive definite matrices, *Numer. Linear Algebra Appl.* **10**, 385–400.
M. Benzi and M. Tůma (2003*b*), A robust preconditioner with low memory requirements for large sparse least squares problems, *SIAM J. Sci. Comput.* **25**, 499–512.
M. Benzi, G. Golub and J. Liesen (2005), Numerical solution of saddle point problems, *Acta Numer.* **14**, 1–137.
A. Berman and R. J. Plemmons (1974), Cones and iterative methods for best least squares solutions of linear systems, *SIAM J. Numer. Anal.* **11**, 145–154.
C.-E. Bichot and P. Siarry, eds (2011), *Graph Partitioning*, ISTE and Wiley.
C. H. Bischof and P. C. Hansen (1991), Structure-preserving and rank-revealing *QR*-factorizations, *SIAM J. Sci. Comput.* **12**, 1332–1350.
C. H. Bischof and P. C. Hansen (1992), A block algorithm for computing rank-revealing *QR* factorizations, *Numer. Algorithms* **2**, 371–391.
Å. Björck (1967*a*), Iterative refinement of linear least squares solutions I, *BIT Numer.* **7**, 257–278.
Å. Björck (1967*b*), Solving linear least squares problems by Gram–Schmidt orthogonalization, *BIT Numer.* **7**, 1–21.
Å. Björck (1976), Methods for sparse least squares problems, in *Sparse Matrix Computations* (J. R. Bunch and D. J. Rose, eds), Academic Press, pp. 177–199.
Å. Björck (1987), Stability analysis of the method of seminormal equations for linear least squares problems, *Linear Algebra Appl.* **88**, 31–48.
Å. Björck (2024), *Numerical Methods for Least Squares Problems*, second edition, SIAM.
Å. Björck and G. Golub (1967), ALGOL programming, contribution no. 22: Iterative refinement of linear least square solutions by Householder transformation, *BIT Numer. Math.* **7**, 322–337.
Å. Björck and C. C. Paige (1994), Solution of augmented linear systems using orthogonal factorizations, *BIT Numer. Math.* **34**, 1–24.
D. Braess and P. Peisker (1986), On the numerical solution of the biharmonic equation and the role of squaring matrices for preconditioning, *IMA J. Numer. Anal.* **6**, 393–404.
R. Bru, J. Marín, J. Mas and M. Tůma (2014), Preconditioned iterative methods for solving linear least squares problems, *SIAM J. Sci. Comput.* **36**, A2002–A2022.

N. I. Buleev (1959), A numerical method for solving two-dimensional diffusion equations (in Russian), *Atomnaja Energija* **6**, 338–340.
J. R. Bunch and L. Kaufman (1977), Some stable methods for calculating inertia and solving symmetric linear systems, *Math. Comp.* **31**, 162–179.
P. Businger and G. H. Golub (1965), Linear least squares solutions by Householder transformations, *Numer. Math.* **7**, 269–276.
A. Buttari (2013), Fine-grained multithreading for the multifrontal *QR* factorization of sparse matrices, *SIAM J. Sci. Comput.* **35**, C323–C345.
D. Calvetti and E. Somersalo (2005), Priorconditioners for linear systems, *Inverse Problems* **21**, art. 1397.
L. Cambier, C. Chen, E. G. Boman, S. Rajamanickam, R. S. Tuminaro and E. Darve (2020), An algebraic sparsified nested dissection algorithm using low-rank approximations, *SIAM J. Matrix Anal. Appl.* **41**, 715–746.
E. Carson and I. Daužickaitė (2024), A comparison of mixed precision iterative refinement approaches for least-squares problems. Available at arXiv:2405.18363.
E. Carson and N. J. Higham (2017), A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems, *SIAM J. Sci. Comput.* **39**, A2834–A2856.
E. Carson and N. J. Higham (2018), Accelerating the solution of linear systems by iterative refinement in three precisions, *SIAM J. Sci. Comput.* **40**, A817–A847.
E. Carson, N. J. Higham and S. Pranesh (2020), Three-precision GMRES-based iterative refinement for least squares problems, *SIAM J. Sci. Comput.* **42**, A4063–A4083.
E. Carson, K. Lund, M. Rozložník and S. Thomas (2022), Block Gram–Schmidt algorithms and their stability properties, *Linear Algebra Appl.* **638**, 150–195.
C. Cartis, J. Fiala and Z. Shao (2021), Hashing embeddings of optimal dimension, with applications to linear least squares. Available at arXiv:2105.11815.
U. V. Çatalyürek, C. Aykanat and E. Kayaaslan (2011), Hypergraph partitioning-based fill-reducing ordering for symmetric matrices, *SIAM J. Sci. Comput.* **33**, 1996–2023.
J. Cerdán, D. Guerrero, J. Marín and J. Mas (2020), Preconditioners for rank deficient least squares problems, *J. Comput. Appl. Math.* **372**, art. 112621.
T. F. Chan (1987), Rank revealing *QR* factorizations, *Linear Algebra Appl.* **88/89**, 67–82.
T. F. Chan and H. A. van der Vorst (1997), Approximate and incomplete factorizations, in *Parallel Numerical Algorithms* (D. E. Keyes, A. Sameh and V. Venkatakrishnan, eds), Kluwer Academic, pp. 167–202.
S. Chandrasekaran and I. C. F. Ipsen (1994), On rank-revealing factorisations, *SIAM J. Matrix Anal. Appl.* **15**, 592–622.
X.-W. Chang, C. C. Paige and D. Titley-Peloquin (2009), Stopping criteria for the iterative solution of linear least squares problems, *SIAM J. Matrix Anal. Appl.* **31**, 831–852.
D. Chen, T.-Z. Huang and L. Li (2012), An algorithm for symmetric indefinite linear systems, *J. Comput. Anal. Appl.* **14**, 767–784.
Q. Chen, A. Ghai and X. Jiao (2021), HILUCSI: Simple, robust, and fast multilevel ILU for large-scale saddle-point problems from PDEs, *Numer. Linear Algebra Appl.* **28**, art. e2400.
Y. Chen, T. A. Davis, W. W. Hager and S. Rajamanickam (2008), Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate, *ACM Trans. Math. Softw.* **35**, 1–14.

A. Çivril and M. Magdon-Ismail (2009), On selecting a maximum volume sub-matrix of a matrix and related problems, *Theoret. Comput. Sci.* **410**, 4801–4811.
J.-J. Climent and C. Perea (2003), Iterative methods for least-square problems based on proper splittings, *J. Comput. Appl. Math.* **158**, 43–48.
T. F. Coleman and A. Verma (2001), A preconditioned conjugate gradient approach to linear equality constrained minimization, *Comput. Optim. Appl.* **20**, 61–72.
T. F. Coleman, A. Edenbrandt and J. R. Gilbert (1986), Predicting fill for sparse orthogonal factorization, *J. Assoc. Comput. Mach.* **33**, 517–532.
P. G. Constantine and D. F. Gleich (2011), Tall and skinny QR factorizations in mapreduce architectures, in *Proceedings of the Second International Workshop on MapReduce and its Applications (MapReduce '11)*, ACM, pp. 43–50.
E. J. Craig (1955), The N-step iteration procedures, *J. Math. Phys.* **34**, 64–73.
E. H. Cuthill and J. McKee (1969), Reducing the bandwidth of sparse symmetric matrices, in *Proceedings of the 24th National Conference of the ACM*, ACM Press, pp. 157–172.
I. Daužickaitė, A. S. Lawless, J. A. Scott and P. J. Van Leeuwen (2021), Randomised preconditioning for the forcing formulation of weak-constraint 4D-Var, *Quart. J. R. Meteorol. Soc.* **147**, 3719–3734.
T. A. Davis (2011), Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization, *ACM Trans. Math. Softw.* **38**, art. 8.
T. A. Davis and W. W. Hager (2001), Multiple-rank modifications of a sparse Cholesky factorization, *SIAM J. Matrix Anal. Appl.* **22**, 997–1013.
T. A. Davis, J. R. Gilbert, S. I. Larimore and E. G. Ng (2004*a*), Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm, *ACM Trans. Math. Softw.* **30**, 377–380.
T. A. Davis, J. R. Gilbert, S. I. Larimore and E. G. Ng (2004*b*), A column approximate minimum degree ordering algorithm, *ACM Trans. Math. Softw.* **30**, 353–376.
T. A. Davis, W. W. Hager, S. P. Kolodziej and S. N. Yeralan (2020), Algorithm 1003: Mongoose, a graph coarsening and partitioning library, *ACM Trans. Math. Softw.* **46**, art. 7.
T. A. Davis, S. Rajamanickam and W. M. Sid-Lakhdar (2016), A survey of direct methods for sparse linear systems, *Acta Numer.* **25**, 383–566.
F. de Hoog and M. Hegland (2023), A note on error bounds for pseudo skeleton approximations of matrices, *Linear Algebra Appl.* **669**, 102–117.
M. Dehghani, A. Lambe and D. Orban (2020), A regularized interior-point method for constrained linear least squares, *Inf. Syst. Oper. Res.* **58**, 202–224.
J. Demmel, L. Grigori, M. Hoemmen and J. Langou (2012), Communication-optimal parallel and sequential QR and LU factorizations, *SIAM J. Sci. Comput.* **34**, A206–A239.
J. W. Demmel (1997), *Applied Numerical Linear Algebra*, SIAM.
J. W. Demmel, L. Grigori, M. Gu and H. Xiang (2015), Communication avoiding rank revealing QR factorization with column pivoting, *SIAM J. Matrix Anal. Appl.* **36**, 55–89.
J. W. Demmel, Y. Hida, J. E. Riedy and X. S. Li (2009), Extra-precise iterative refinement for overdetermined least squares problems, *ACM Trans. Math. Softw.* **35**, 1–32.
D. di Serafino and D. Orban (2021), Constraint-preconditioned Krylov solvers for regularized saddle-point systems, *SIAM J. Sci. Comput.* **43**, A1001–A1026.
P. Drineas and M. W. Mahoney (2016), RandNLA: Randomized numerical linear algebra, *Commun. Assoc. Comput. Mach.* **59**, 80–90.

J. A. Duersch and M. Gu (2020), Randomized projection for rank-revealing matrix factorizations and low-rank approximations, *SIAM Rev.* **62**, 661–682.
I. Duff, J. Hogg and F. Lopez (2020), A new sparse LDL^T solver using *a posteriori* threshold pivoting, *SIAM J. Sci. Comput.* **42**, C23–C42.
I. S. Duff (2004), MA57: A code for the solution of sparse symmetric definite and indefinite systems, *ACM Trans. Math. Softw.* **30**, 118–154.
I. S. Duff and S. Pralet (2005), Strategies for scaling and pivoting for sparse symmetric indefinite problems, *SIAM J. Matrix Anal. Appl.* **27**, 313–340.
I. S. Duff and S. Pralet (2007), Towards stable mixed pivoting strategies for the sequential and parallel solution of sparse symmetric indefinite systems, *SIAM J. Matrix Anal. Appl.* **29**, 1007–1024.
I. S. Duff and J. K. Reid (1983), The multifrontal solution of indefinite sparse symmetric linear, *ACM Trans. Math. Softw.* **9**, 302–325.
I. S. Duff, A. M. Erisman and J. K. Reid (2017), *Direct Methods for Sparse Matrices*, second edition, Oxford University Press.
I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott and K. Turner (1991), The factorization of sparse symmetric indefinite matrices, *IMA J. Numer. Anal.* **11**, 181–204.
I. S. Duff, R. Guivarch, D. Ruiz and M. Zenadi (2015), The augmented block Cimmino distributed method, *SIAM J. Sci. Comput.* **37**, A1248–A1269.
A. Dumitraşc, P. Leleux, C. Popa, U. Ruede and D. Ruiz (2021), Extensions of the augmented block Cimmino method to the solution of full rank rectangular systems, *SIAM J. Sci. Comput.* **43**, S516–S539.
A. Dumitraşc, P. Leleux, C. Popa, D. Ruiz and S. Torun (2018), The augmented block Cimmino algorithm revisited. Available at arXiv:1805.11487.
O. Edlund (2002), A software package for sparse orthogonal factorization and updating, *ACM Trans. Math. Softw.* **28**, 448–482.
T. Elfving (1980), Block-iterative methods for consistent and inconsistent linear equations, *Numer. Math.* **35**, 1–12.
E. N. Epperly (2024), Fast and forward stable randomized algorithms for linear least-squares problems, *SIAM J. Matrix Anal. Appl.* **45**, 1782–1804.
E. N. Epperly, M. Meier and Y. Nakatsukasa (2024), Fast randomized least-squares solvers can be just as accurate and stable as classical direct solvers. Available at arXiv:2406.03468.
R. Estrin, D. Orban and M. A. Saunders (2019), LSLQ: An iterative method for linear least-squares with an error minimization property, *SIAM J. Matrix Anal. Appl.* **40**, 254–275.
D. C.-L. Fong and M. Saunders (2011), LSMR: An iterative algorithm for sparse least-squares problems, *SIAM J. Sci. Comput.* **33**, 2950–2971.
L. V. Foster (1986), Rank and null space calculations using matrix decomposition without column interchanges, *Linear Algebra Appl.* **74**, 47–71.
J. G. F. Francis (1961), The QR transformation a unitary analogue to the LR transformation, Part 1, *Comput. J.* **4**, 265–271.
R. W. Freund (1997), Preconditioning of symmetric, but highly indefinite linear systems, in *15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*, Vol. 2, pp. 551–556.
C. F. Gauss (1809), *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*, Cambridge Library Collection, Cambridge University Press. 2011 reprint of the 1809 original.

C. F. Gauss and G. W. Stewart (1995), *Theory of the Combination of Observations Least Subject to Errors Part One, Part Two, Supplement*, SIAM.
W. M. Gentleman (1976), Row elimination for solving sparse linear systems and least squares problems, in *Numerical Analysis* (G. A. Watson, ed.), Vol. 506 of Lecture Notes in Mathematics, Springer, pp. 122–133.
A. George (1973), Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal.* **10**, 345–363.
A. George and M. T. Heath (1980), Solution of sparse linear least squares problems using Givens rotations, *Linear Algebra Appl.* **34**, 69–83.
A. George and E. Ng (1983), On row and column orderings for sparse least squares problems, *SIAM J. Numer. Anal.* **20**, 326–344.
A. George, J. Liu and E. Ng (1986), Row-ordering schemes for sparse Givens transformations, III: Analyses for a model problem, *Linear Algebra Appl.* **75**, 225–240.
A. George, J. Liu and E. Ng (1988), A data structure for sparse *QR* and *LU* factorizations, *SIAM J. Sci. Comput.* **9**, 100–121.
N. E. Gibbs, W. G. J. Poole and P. K. Stockmeyer (1976), An algorithm for reducing the bandwidth and profile of a sparse matrix, *SIAM J. Numer. Anal.* **13**, 236–250.
J. R. Gilbert, X. S. Li, E. G. Ng and B. W. Peyton (2001), Computing row and column counts for sparse *QR* and *LU* factorization, *BIT Numer.* **41**, 693–710.
P. E. Gill, M. A. Saunders and J. R. Shinnerl (1996), On the stability of Cholesky factorization for symmetric quasidefinite systems, *SIAM J. Matrix Anal. Appl.* **17**, 35–46.
L. Giraud and S. Gratton (2006), On the sensitivity of some spectral preconditioners, *SIAM J. Matrix Anal. Appl.* **27**, 1089–1105.
L. Giraud, J. Langou, M. Rozložník and J. van den Eshof (2005), Rounding error analysis of the classical Gram–Schmidt orthogonalization process, *Numer. Math.* **101**, 87–100.
W. Givens (1953), A method of computing eigenvalues and eigenvectors suggested by classical results on symmetric matrices, in *Simultaneous Linear Equations and the Determination of Eigenvalues*, Vol. 29 of National Bureau of Standards Applied Mathematics Series, US Government Printing Office, pp. 117–122.
A. Gnanasekaran and E. Darve (2022), Hierarchical orthogonal factorization: Sparse least squares problems, *J. Sci. Comput.* **91**, art. 50.
G. H. Golub (1965), Numerical methods for solving linear least squares problems, *Numer. Math.* **7**, 206–216.
G. H. Golub and G. Meurant (1997), Matrices, moments and quadrature, II: How to compute the norm of the error in iterative methods, *BIT Numer.* **37**, 687–705.
G. H. Golub and C. F. Van Loan (1996), *Matrix Computations*, fourth edition, Johns Hopkins University Press.
G. H. Golub and J. H. Wilkinson (1966), Note on the iterative refinement of least squares solution, *Numer. Math.* **9**, 139–148.
S. A. Goreinov, E. E. Tyrtyshnikov and N. L. Zamarashkin (1997), A theory of pseudo-skeleton approximations, *Linear Algebra Appl.* **261**, 1–21.
N. I. M. Gould and J. A. Scott (2017), The state-of-the-art of preconditioners for sparse linear least-squares problems, *ACM Trans. Math. Softw.* **43**, art. 36.
S. Gratton, S. Gürol, E. Simon and P. L. Toint (2018), A note on preconditioning weighted linear least-squares, with consequences for weakly constrained variational data assimilation, *Quart. J. R. Meteorol. Soc.* **144**, 934–940.

S. Gratton, A. S. Lawless and N. K. Nichols (2007), Approximate Gauss–Newton methods for nonlinear least squares problems, *SIAM J. Optim.* **18**, 106–132.
S. Gratton, A. Sartenaer and J. Tshimanga (2011), On a class of limited memory preconditioners for large scale linear systems with multiple right-hand sides, *SIAM J. Opt.* **21**, 912–935.
C. Greif, S. He and P. Liu (2017), SYM-ILDL: Incomplete LDL^T factorization of symmetric indefinite and skew-symmetric matrices, *ACM Trans. Math. Softw.* **44**, art. 1.
M. Hagemann and O. Schenk (2006), Weighted matchings for preconditioning symmetric indefinite linear systems, *SIAM J. Sci. Comput.* **28**, 403–420.
W. W. Hager (1989), Updating the inverse of a matrix, *SIAM Rev.* **31**, 221–239.
P. C. Hansen (1998), *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*, SIAM.
P. C. Hansen (2010), *Discrete Inverse Problems: Insight and Algorithms*, Vol. 7 of Fundamentals of Algorithms, SIAM.
P. C. Hansen, K. Hayami and K. Morikuni (2022), GMRES methods for tomographic reconstruction with an unmatched back projector, *J. Comput. Appl. Math.* **413**, art. 114352.
P. C. Hansen, V. Pereyra and G. Scherer (2013), *Least Squares Data Fitting with Applications*, Johns Hopkins University Press.
R. J. Hanson and C. L. Lawson (1969), Extensions and applications of the Householder algorithm for solving linear least squares problems, *Math. Comp.* **23**, 787–812.
D. R. Hare, C. R. Johnson, D. D. Olesky and P. van den Driessche (1993), Sparsity analysis of the *QR* factorization, *SIAM J. Matrix Anal. Appl.* **14**, 655–669.
E. Havelková and I. Hnětynková (2023), Iterative hybrid regularization for extremely noisy full models in single particle analysis, *Linear Algebra Appl.* **656**, 131–157.
K. Hayami, J.-F. Yin and T. Ito (2010), GMRES methods for least squares problems, *SIAM J. Matrix Anal. Appl.* **31**, 2400–2430.
M. T. Heath (1982), Some extensions of an algorithm for sparse linear least squares problems, *SIAM J. Sci. Statist. Comput.* **3**, 223–237.
P. Hénon, P. Ramet and J. Roman (2002), PaStiX: A high-performance parallel direct solver for sparse symmetric positive definite systems, *Parallel Comput.* **28**, 301–321.
M. R. Hestenes and E. Stiefel (1952), Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bureau Standards* **49**, 409–436.
N. J. Higham (1990), Analysis of the Cholesky decomposition of a semi-definite matrix, in *Reliable Numerical Computation* (M. G. Cox and S. J. Hammarling, eds), Oxford University Press, pp. 161–185.
N. J. Higham (2002), *Accuracy and Stability of Numerical Algorithms*, second edition, SIAM.
N. J. Higham and T. Mary (2022), Mixed precision algorithms in numerical linear algebra, *Acta Numer.* **31**, 347–414.
N. J. Higham and S. Pranesh (2021), Exploiting lower precision arithmetic in solving symmetric positive definite linear systems and least squares problems, *SIAM J. Sci. Comput.* **43**, A258–A277.
N. J. Higham and G. W. Stewart (1987), Numerical linear algebra in statistical computing, in *The State of the Art in Numerical Analysis* (I. Iserles and M. J. D. Powell, eds), Oxford University Press, pp. 41–57.
I. Hnětynková, M. Kubínová and M. Plešinger (2017), Noise representation in residuals of LSQR, LSMR, and CRAIG regularization, *Linear Algebra Appl.* **533**, 357–379.

J. D. Hogg and J. A. Scott (2013*a*), An efficient analyse phase for element problems, *Numer. Linear Algebra Appl.* **20**, 397–412.
J. D. Hogg and J. A. Scott (2013*b*), New parallel sparse direct solvers for multicore archiectures, *Algorithms* **6**, 702–725.
J. D. Hogg and J. A. Scott (2013*c*), Pivoting strategies for tough sparse indefinite systems, *ACM Trans. Math. Softw.* **40**, art. 4.
J. D. Hogg, J. K. Reid and J. A. Scott (2010), Design of a multicore sparse Cholesky factorization using DAGs, *SIAM J. Sci. Comput.* **32**, 3627–3649.
J. D. Hogg, J. A. Scott and S. Thorne (2017), Numerically aware orderings for sparse symmetric indefinite linear systems, *ACM Trans. Math. Softw.* **44**, art. 13.
Y. P. Hong and C.-T. Pan (1992), Rank-revealing *QR* factorizations and the singular value decomposition, *Math. Comp.* **58**, 213–232.
A. S. Householder (1958), Unitary triangularization of a nonsymmetric matrix, *J. Assoc. Comput. Mach.* **5**, 339–342.
G. W. Howell and M. Baboulin (2016), LU preconditioning for overdetermined sparse least squares problems, in *Parallel Processing and Applied Mathematics, Part I*, Vol. 9573 of Lecture Notes in Computer Science, Springer, pp. 128–137.
D. Hysom and A. Pothen (2002), Level-based incomplete LU factorization: Graph model and algorithms. Technical report UCRL-JC-150789, Lawrence Livermore National Laboratory, California, USA.
I. C. F. Ipsen and T. Wentworth (2014), The effect of coherence on sampling from matrices with orthonormal columns, and preconditioned least squares problems, *SIAM J. Matrix Anal. Appl.* **35**, 1490–1520.
C. G. J. Jacobi (1845), Über eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkommenden lineären Gleichungen, *Astronom. Nachr.* **523**, 297–306.
A. Jennings and M. A. Ajiz (1984), Incomplete methods for solving $A^T Ax = b$, *SIAM J. Sci. Statist. Comput.* **5**, 978–987.
S. Jin, S. Pei, Y. Wang and Y. Qi (2021), A parallel sparse triangular solve algorithm based on dependency elimination of the solution vector, *Cluster Comput.* **24**, 1317–1330.
M. T. Jones and P. E. Plassmann (1995), An improved incomplete Cholesky factorization, *ACM Trans. Math. Softw.* **21**, 5–17.
G. Karypis and V. Kumar (1998), A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.* **20**, 359–392.
D. S. Kershaw (1978), The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations, *J. Comput. Phys.* **26**, 43–65.
J. Kuřátko (2019), Factorization of saddle-point matrices in dynamical systems optimization: Reusing pivots, *Linear Algebra Appl.* **566**, 61–85.
A. J. Laub (2005), *Matrix Analysis for Scientists & Engineers*, SIAM.
P. Läuchli (1961), Jordan-Elimination und Ausgleichung nach kleinsten Quadraten, *Numer. Math.* **3**, 226–240.
C. L. Lawson and R. J. Hanson (1995), *Solving Least Squares Problems*, Vol. 15 of Classics in Applied Mathematics, SIAM. Revised reprint of the 1974 original.
A. M. Legendre (1805), *Nouvelles Méthodes pour la Détermination des Orbites des Comètes*, Vol. 116 of Libraire pour les Mathématiques, la Marine, l'Architecture, Firmin Didot.
N. Li and Y. Saad (2005), Crout versions of ILU factorization with pivoting for sparse symmetric matrices, *Electron. Trans. Numer. Anal.* **20**, 75–85.

N. Li and Y. Saad (2006), MIQR: A multilevel incomplete QR preconditioner for large sparse least-squares problems, *SIAM J. Matrix Anal. Appl.* **28**, 524–550.
N. Li, Y. Saad and E. Chow (2003), Crout versions of ILU for general sparse matrices, *SIAM J. Sci. Comput.* **25**, 716–728.
J. Liesen and Z. Strakoš (2013), *Krylov Subspace Methods: Principles and Analysis*, Numerical Mathematics and Scientific Computation, Oxford University Press.
C.-J. Lin and J. J. Moré (1999), Incomplete Cholesky factorizations with limited memory, *SIAM J. Sci. Comput.* **21**, 24–45.
R. J. Lipton, D. J. Rose and R. E. Tarjan (1979), Generalized nested dissection, *SIAM J. Numer. Anal.* **16**, 346–358.
J. W. H. Liu (1986*a*), A compact row storage scheme for Cholesky factors using elimination trees, *ACM Trans. Math. Softw.* **12**, 127–148.
J. W. H. Liu (1986*b*), On general row merging schemes for sparse Givens transformations, *SIAM J. Sci. Comput.* **7**, 1190–1211.
J. W. H. Liu (1989), The minimum degree ordering with constraints, *SIAM J. Sci. Comput.* **10**, 1136–1145.
J. W. H. Liu (1990), The role of elimination trees in sparse factorizations, *SIAM J. Matrix Anal. Appl.* **11**, 134–172.
J. W. H. Liu (1992), The multifrontal method for sparse matrix solution: Theory and practice, *SIAM Rev.* **34**, 82–109.
L. Lukšan and J. Vlček (1998), Indefinitely preconditioned inexact Newton method for large sparse equality constrained non-linear programming problems, *Numer. Linear Algebra Appl.* **5**, 219–247.
J. R. Magnus (2022), Gauss on least-squares and maximum-likelihood estimation, *Arch. Hist. Exact Sci.* **76**, 425–430.
J. Mandel (1993), Balancing domain decomposition, *Commun. Numer. Methods Engrg* **9**, 233–241.
J. Marín, J. Mas, D. Guerrero and K. Hayami (2017), Updating preconditioners for modified least squares problems, *Numer. Algorithms* **75**, 491–508.
H. M. Markowitz (1957), The elimination form of the inverse and its application to linear programming, *Manag. Sci.* **3**, 255–269.
P.-G. Martinsson and J. A. Tropp (2020), Randomized numerical linear algebra: Foundations and algorithms, *Acta Numer.* **29**, 403–572.
P.-G. Martinsson, G. Quintana-Ortí, N. Heavner and R. van de Geijn (2017), Householder QR factorization with randomization for column pivoting (HQRRP), *SIAM J. Sci. Comput.* **39**, C96–C115.
P. Matstoms (1994), *Sparse QR Factorization with Applications to Linear Least Squares Problems*, Vol. 337 of Linköping Studies in Science and Technology, Linköping University, Department of Mathematics.
M. Meier and Y. Nakatsukasa (2022), Randomized algorithms for Tikhonov regularization in linear least squares. Available at arXiv:2203.07329.
M. Meier, Y. Nakatsukasa, A. Townsend and M. Webb (2024), Are sketch-and-precondition least squares solvers numerically stable?, *SIAM J. Matrix Anal. Appl.* **45**, 905–929.
M. Melnichenko, O. Balabanov, R. Murray, J. Demmel, M. W. Mahoney and P. Luszczek (2024), CholeskyQR with randomization and pivoting for tall matrices (CQRRPT). Available at arXiv:2311.08316.

X. Meng, M. A. Saunders and M. W. Mahoney (2014), LSRN: A parallel iterative solver for strongly over-or underdetermined systems, *SIAM J. Sci. Comput.* **36**, C95–C118.
R. Murray, J. Demmel, M. W. Mahoney, N. B. Erichson, M. Melnichenko, O. A. Malik, L. Grigori, P. Luszczek, M. Dereziński, M. E. Lopes, T. Liang, H. Luo and J. Dongarra (2023), Randomized numerical linear algebra: A perspective on the field with an eye to software. Available at arXiv:2302.11474.
A. Napov (2023), An incomplete Cholesky preconditioner based on orthogonal approximations, *SIAM J. Sci. Comput.* **45**, A729–A752.
E. Ng (1991), A scheme for handling rank-deficiency in the solution of sparse linear least squares problems, *SIAM J. Sci. Comput.* **12**, 1173–1183.
E. G. Ng and B. W. Peyton (1992), A tight and explicit representation of Q in sparse QR factorization. Report ORNL/TM–12059, Oak Ridge National Laboratory, TN.
E. G. Ng and B. W. Peyton (1993*a*), Block sparse Cholesky algorithms on advanced uniprocessor computers, *SIAM J. Sci. Comput.* **14**, 1034–1056.
E. G. Ng and B. W. Peyton (1993*b*), A supernodal Cholesky factorization algorithm for shared-memory multiprocessors, *SIAM J. Sci. Comput.* **14**, 761–769.
A. R. L. Oliveira and D. C. Sorensen (2005), A new class of preconditioners for large-scale linear systems from interior point methods for linear programming, *Linear Algebra Appl.* **394**, 1–24.
D. Orban (2015), Limited-memory LDL^T factorization of symmetric quasi-definite matrices with application to constrained optimization, *Numer. Algorithms* **70**, 9–41.
D. Orban and M. Arioli (2017), *Iterative Solution of Symmetric Quasi-Definite Linear Systems*, SIAM.
A. I. Osinsky and N. L. Zamarashkin (2018), Pseudo-skeleton approximations with better accuracy estimates, *Linear Algebra Appl.* **537**, 221–249.
I. K. Ozaslan, M. Pilanci and O. Arikan (2023), M-IHS: An accelerated randomized preconditioning method avoiding costly matrix decompositions, *Linear Algebra Appl.* **678**, 57–91.
C. C. Paige and M. A. Saunders (1975), Solution of sparse indefinite systems of linear equations, *SIAM J. Numer. Anal.* **12**, 617–629.
C. C. Paige and M. A. Saunders (1982), LSQR: An algorithm for sparse linear equations and sparse least squares, *ACM Trans. Math. Softw.* **8**, 43–71.
A. T. Papadopoulos, I. S. Duff and A. J. Wathen (2005), A class of incomplete orthogonal factorization methods, II: Implementation and results, *BIT Numer.* **45**, 159–179.
J. Papež and P. Tichý (2023), Estimating error norms in CG-like algorithms for least-squares and least-norm problems, *Numer. Algorithms*.
S. Parter (1961), The use of linear graphs in Gaussian elimination, *SIAM Rev.* **3**, 119–130, 364–369.
J. W. Pearson and J. Pestana (2020), Preconditioners for Krylov subspace methods: An overview, *GAMM-Mitteilungen* **43**, art. e202000015.
G. Peters and J. H. Wilkinson (1970), The least squares problem and pseudo-inverse, *Comput. J.* **131**, 309–316.
D. J. Pierce and J. G. Lewis (1997), Sparse multifrontal rank revealing *QR* factorization, *SIAM J. Matrix Anal. Appl.* **18**, 159–180.
A. Pothen (1993), Predicting the structure of sparse orthogonal factors, *Linear Algebra Appl.* **194**, 183–203.

A. Pothen and C. J. Fan (1990), Computing the block triangular form of a sparse matrix, *ACM Trans. Math. Softw.* **16**, 303–324.
T. Rees and J. A. Scott (2018), A comparative study of null-space factorizations for sparse symmetric saddle point systems, *Numer. Linear Algebra Appl.* **25**, art. e2103.
L. Reichel, H. Sadok and W.-H. Zhang (2020), Simple stopping criteria for the LSQR method applied to discrete ill-posed problems, *Numer. Algorithms* **84**, 1381–1395.
J. K. Reid and J. A. Scott (1999), Ordering symmetric sparse matrices for small profile and wavefront, *Int. J. Numer. Method Engrg* **45**, 1737–1755.
J. K. Reid and J. A. Scott (2009), An out-of-core sparse Cholesky solver, *ACM Trans. Math. Softw.* **36**, art. 9.
G. Reißig (2007), Local fill reduction techniques for sparse symmetric linear systems, *Electr. Engrg* **89**, 639–652.
D. J. Rose (1972), A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, in *Graph Theory and Computing* (R. Read, ed.), Academic Press, pp. 183–217.
D. J. Rose, R. E. Tarjan and G. S. Lueker (1976), Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.* **5**, 266–283.
E. Rothberg and S. Eisenstat (1998), Node selection strategies for bottom-up sparse matrix ordering, *SIAM J. Matrix Anal. Appl.* **19**, 682–695.
M. Rozložník, A. Smoktunowicz and J. Kopal (2014), A note on iterative refinement for seminormal equations, *Appl. Numer. Math.* **75**, 167–174.
Y. Saad (2003*a*), Finding exact and approximate block structures for ILU preconditioning, *SIAM J. Sci. Comput.* **24**, 1107–1123.
Y. Saad (2003*b*), *Iterative Methods for Sparse Linear Systems*, second edition, SIAM.
Y. Saad and M. H. Schultz (1986), GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Comput.* **7**, 856–869.
Y. Saad and H. A. van der Vorst (2000), Iterative solution of linear systems in the 20th century, *J. Comput. Appl. Math.* **123**, 1–33.
Y. Saad, M. Yeung, J. Erhel and F. Guyomarch (2000), A deflated version of the conjugate gradient algorithm, *SIAM J. Sci. Comput.* **21**, 1909–1926.
M. A. Saunders (1979), Sparse least squares problems by conjugate gradients: A comparison of preconditioning methods, in *Proceedings of Computer Science and Statistics: Twelfth Annual Conference on the Interface*, University of Waterloo, Canada.
M. A. Saunders (1995), Solution of sparse rectangular systems using LSQR and Craig, *BIT Numer.* **35**, 588–604.
M. A. Saunders (1996), Cholesky-based methods for sparse least squares: The benefits of regularization, in *Linear and Nonlinear Conjugate Gradient-Related Methods* (L. Adams and J. L. Nazareth, eds), SIAM, pp. 92–100.
O. Schenk and K. Gärtner (2006), On fast factorization pivoting methods for symmetric indefinite systems, *Electron. Trans. Numer. Anal.* **23**, 158–179.
R. Schreiber and C. Van Loan (1989), A storage-efficient WY representation for products of Householder transformations, *SIAM J. Sci. Comput.* **10**, 53–57.
J. A. Scott (2023), HSL@60: A brief history of the HSL mathematical software library. Technical report STFC-TR-2023-002, Science and Technology Facilities Council, Didcot.
J. A. Scott and M. Tůma (2011), The importance of structure in incomplete factorization preconditioners, *BIT Numer.* **51**, 385–404.

J. A. Scott and M. Tůma (2014*a*), HSL_MI28: An efficient and robust limited-memory incomplete Cholesky factorization code, *ACM Trans. Math. Softw.* **40**, art. 24.
J. A. Scott and M. Tůma (2014*b*), On positive semidefinite modification schemes for incomplete Cholesky factorization, *SIAM J. Sci. Comput.* **36**, A609–A633.
J. A. Scott and M. Tůma (2014*c*), On signed incomplete Cholesky factorization preconditioners for saddle-point systems, *SIAM J. Sci. Comput.* **36**, A2984–A3010.
J. A. Scott and M. Tůma (2016), Preconditioning of linear least squares by robust incomplete factorization for implicitly held normal equations, *SIAM J. Sci. Comput.* **38**, C603–C623.
J. A. Scott and M. Tůma (2017*a*), Improving the stability and robustness of incomplete symmetric indefinite factorization preconditioners, *Numer. Linear Algebra Appl.* **24**, art. e2099.
J. A. Scott and M. Tůma (2017*b*), Solving mixed sparse-dense linear least-squares problems by preconditioned iterative methods, *SIAM J. Sci. Comput.* **39**, A2422–A2437.
J. A. Scott and M. Tůma (2019), Sparse stretching for solving sparse-dense linear least-squares problems, *SIAM J. Sci. Comput.* **41**, A1604–A1625.
J. A. Scott and M. Tůma (2021), Strengths and limitations of stretching for least-squares problems with some dense rows, *ACM Trans. Math. Softw.* **47**, art. 1.
J. A. Scott and M. Tůma (2022*a*), A computational study of using black-box QR solvers for large-scale sparse-dense linear least squares problems, *ACM Trans. Math. Softw.* **48**, art. 5.
J. A. Scott and M. Tůma (2022*b*), A null-space approach for large-scale symmetric saddle point systems with a small and non zero (2, 2) block, *Numer. Algorithms* **90**, 1639–1667.
J. A. Scott and M. Tůma (2022*c*), Solving large linear least squares problems with linear equality constraints, *BIT Numer.* **62**, 1765–1787.
J. A. Scott and M. Tůma (2023), *Algorithms for Sparse Linear Systems*, Nečas Center Series, Birkhäuser/Springer.
J. Scott and M. Tůma (2024), Avoiding breakdown in incomplete factorizations in low precision arithmetic, *ACM Trans. Math. Software* **50**, art. 9.
J. Scott and M. Tůma (2025), Developing robust incomplete Cholesky factorization in half precision arithmetic, *Numer. Algorithms*. Available at https://doi.org/10.1007/s11075-025-02015-x.
A. Scotto di Perrotolo (2022), Randomized numerical linear algebra methods with application to data assimilation. PhD thesis, Institut Superieur de l'Aeronautique et de l'Espace, Toulouse.
S. W. Sloan (1986), An algorithm for profile and wavefront reduction of sparse matrices, *Int. J. Numer. Method Engrg* **23**, 239–251.
D. C. Sorensen (1977), Updating the symmetric indefinite factorization with applications in a modified Newton's method. PhD thesis, University of California, San Diego.
B. Speelpenning (1978), Generalized element method. Technical report, Department of Computer Science, Illinois University, Urbana, IL.
D. A. Spielman and S.-H. Teng (2014), Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems, *SIAM J. Matrix Anal. Appl.* **35**, 835–885.
J. M. Tang, R. Nabben, C. Vuik and Y. A. Erlangga (2009), Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods, *J. Sci. Comput.* **39**, 340–370.

R. E. Tarjan (1975), Efficiency of a good but not linear set union algorithm, *J. Assoc. Comput. Mach.* **22**, 215–225.
T. Terao, K. Ozaki and T. Ogita (2020), LU-Cholesky QR algorithms for thin QR decomposition, *Parallel Comput.* **92**, art. 102571.
W. F. Tinney and J. W. Walker (1967), Direct solutions of sparse network equations by optimally ordered triangular factorization, *Proc. IEEE* **55**, 1801–1809.
M. Tůma (2002), A note on the LDL^T decomposition of matrices from saddle-point problems, *SIAM J. Matrix Anal. Appl.* **23**, 903–925.
A. Van der Sluis (1969), Condition numbers and equilibration of matrices, *Numer. Math.* **14**, 14–23.
H. A. van der Vorst (2003), *Iterative Krylov Methods for Large Linear Systems*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press.
C. Van Loan (1985), On the method of weighting for equality-constrained least-squares problems, *SIAM J. Numer. Anal.* **22**, 851–864.
R. J. Vanderbei (1995), Symmetric quasidefinite matrices, *SIAM J. Optim.* **5**, 100–113.
R. S. Varga (1960), Factorization and normalized iterative methods, in *Boundary Problems in Differential Equations* (R. E. Langer, ed.), University of Wisconsin Press, pp. 121–142.
X. Wang, K. A. Gallivan and R. Bramley (1997), CIMGS: An incomplete orthogonal factorization preconditioner, *SIAM J. Sci. Comput.* **18**, 516–536.
A. J. Wathen (2015), Preconditioning, *Acta Numer.* **24**, 329–376.
A. J. Wathen (2022), Some comments on preconditioning for normal equations and least squares, *SIAM Rev.* **64**, 640–649.
J. W. Watts III (1981), A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation, *Soc. Petroleum Engrg J.* **21**, 345–353.
P. Wedin (1973), Perturbation theory for pseudo-inverses, *BIT Numer.* **13**, 217–232.
J. H. Wilkinson (1968), *A priori* error analysis of algebraic processes, in *Proc. Internat. Congr. Math. (Moscow, 1966)*, Izdat. 'Mir', pp. 629–640.
M. A. Woodbury (1949), The stability of out–input matrices. Chicago, IL.
M. A. Woodbury (1950), Inverting modified matrices. Memorandum Report 42, Statistical Research Group, Princeton University, NJ.
Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa and T. Fukaya (2015), Roundoff error analysis of the CholeskyQR2 algorithm, *Electron. Trans. Numer. Anal.* **44**, 306–326.
S. N. Yeralan, T. A. Davis, W. M. Sid-Lakhdar and S. Ranka (2017), Algorithm 980: Sparse QR factorization on the GPU, *ACM Trans. Math. Softw.* **44**, art. 17.
T. Zhao (2016), A spectral analysis of subspace enhanced preconditioners, *J. Sci. Comput.* **66**, 435–457.
G. Zilli and L. Bergamaschi (2022), Block preconditioners for linear systems in interior point methods for convex constrained optimization, *Ann. Univ. Ferrara Sez. VII Sci. Mat.* **68**, 337–368.