See video on how this course is organised in Youtube

Self-study guide

Week 1

Keywords: Introduction, Permutation matrices, Block matrix notation, Gaussian elimination, Back-substitution, LU-factorisation.

Homework: Problems 9, 10, 21 and 26. In addition, solve any additional four problems from 1-27 to gain extra points.

See outline of Week 1 in Youtube

Pages: 3-34.

Synopsis: During the first week we prepare for proving the existence of the Cholesky factorisation by discussing permutation matrices, LU-factorisation, block matrix notation, and recursive definition of matrix algorithms. There is lots of revision material on Gaussian elimination that can be skipped, so do not worry about the large number of pages.

Week 2

Keywords: Cholesky factorisation, fill-in, fill-in reducing permutation, minimum degree ordering.

Homework: Problems, 29, 30, 35 and 37. In addition, solve any additional four problems from 28-37 to gain extra points.

See outline of Week 2 in Youtube

Pages: 35-49.

Synopsis: The topic of the second week is Cholesky factorisation of sparse matrices. First, we prove existence of the Cholesky factorisation for s.p.d. matrices without taking sparsity into account. Our existence proof uses block matrix notation and induction with respect to dimension of the matrix. Unfortunately, the Cholesky factor of a sparse matrix can be dense. To mitigate this, we discuss methods for predicting location of non-zero entries in the factor without actually computing it. Then we introduce the minimum degree ordering method with the aim of obtaining a sparse factor by permuting the matrix before computing it's Cholesy factorisation.

Chapter 1

Direct solution of sparse linear systems

In this Chapter, we study solution methods for linear systems: Find $\mathbf{x} \in \mathbb{R}^n$ s.t.

$$A\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where $\mathbf{b} \in \mathbb{R}^n$ and the coefficient matrix $A \in \mathbb{R}^{n\times n}$ is large, sparse, symmetric and positive definite (s.p.d.). By *sparse matrix*, we mean a matrix with mostly zero entries. If a matrix is not sparse it is called as a *dense matrix*.

Large, sparse, s.p.d. coefficient matrices are related, e.g., to solution of partial differential equations (PDEs) using finite element method (FEM) or finite difference method (FDM). For example, application of FDM to two dimensional Laplace operator leads to a coefficient matrix having at most five non-zero entries on every row. If accurate discretisation is required, the dimension of these coefficient matrices can be of the order $n \approx 10^5 - 10^6$.

We use the sparse Cholesky factorisation to solve (1.1). In sparse Cholesky factorisation, sparse, s.p.d. matrix $A \in \mathbb{R}^{n\times n}$ is decomposed as

$$P^TAP = LL^T, \tag{1.2}$$

where $P \in \mathbb{R}^{n\times n}$ is a *permutation* matrix and $L \in \mathbb{R}^{n\times n}$ is a lower triangular matrix. As a permutation matrix P is invertible, and equation (1.1) is equivalent to

$$P^TAPP^{-1}\mathbf{x} = P^T\mathbf{b} \quad \text{and} \quad LL^TP^{-1}\mathbf{x} = P^T\mathbf{b}.$$

Hence, the solution of (1.1) is obtained by solving the auxiliary problems

$$L\mathbf{z} = P^T\mathbf{b}, \quad L^T\mathbf{y} = \mathbf{z}, \quad \text{and setting} \quad \mathbf{x} = P\mathbf{y}.$$

As L is a lower triangular matrix, the first two equations above are solved using back-substitution.

If $P = I$ in (1.2), it becomes the Cholesky factorisation of A that is related to the Gaussian elimination process. Recall that writing the row-operations conducted during the Gaussian elimination process using elimination matrices yields the LU-factorisation of the coefficient matrix. In LU-factorisation, matrix A is written as $A = LU$ where L is a lower triangular and U an upper triangular matrix. The Cholesky factorisation is derived using the same elimination matrices but taking advantage of symmetry and positive definiteness of A. In sparse Cholesky factorisation, additional permutations are used to obtain a sparse factor L for a sparse matrix A.

To convince the reader that sparse matrices appear in practice, we begin this Chapter by application of finite difference method to solution of the Poisson's equation that results in a linear system with a sparse, s.p.d. coefficient matrix. Next, we discuss how sparse matrices are stored in the memory of a computer. Then we prepare to prove existence of the Cholesky factorisation by recalling the Gaussian elimination process and LU-factorisation. Our existence proof uses block matrix notation that is discussed next. Finally, we show existence of the Cholesky factorisation and introduce minimum degree ordering method for obtaining a sparse factor L for a sparse matrix A. We end the section by studying numerical stability or accuracy of solving linear systems using Cholesky factorisation computed using floating-point numbers.

1.1 Preliminaries

1.1.1 Permutation matrices

See video on permutation matrices in Youtube

In this section, we discuss permutation matrices that encode information on changing the order of rows or the columns of a matrix. Vector $\mathbf{p} \in \mathbb{R}^n$ is called as a *permutation vector*, if it's entries satisfy the conditions: $p_i \in \{1, \ldots, n\}$ and $p_i \neq p_j$ for all $i, j \in \{1, \ldots, n\}$, $i \neq j$. This is, a permutation vector is a re-ordering of $\begin{bmatrix} 1 & \cdots & n \end{bmatrix}$. Matrix $P \in \mathbb{R}^{n\times n}$ is called as a *permutation matrix*, if

$$P = \begin{bmatrix} \mathbf{e}_{p_1} & \cdots & \mathbf{e}_{p_n} \end{bmatrix} \quad \text{where} \quad \mathbf{p} \in \mathbb{R}^n \text{ is a permutation vector.}$$

As P has orthonormal columns it is unitary, i.e., $P^{-1} = P^T$.

Let $P \in \mathbb{R}^{n\times n}$ be a permutation matrix corresponding to permutation vector $p \in \mathbb{R}^n$ and split $A, B \in \mathbb{R}^{n\times n}$ into column and row vectors as

$$A = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} \mathbf{b}_1^T \\ \vdots \\ \mathbf{b}_n^T \end{bmatrix}.$$

Recall that $\mathbf{e}_i^T A$ and $A\mathbf{e}_i$ are the ith row and column of a matrix $A \in \mathbb{R}^{n\times n}$, respectively. By direct computation

$$AP\mathbf{e}_i = A\mathbf{e}_{p_i} = \mathbf{a}_{p_i} \quad \text{and} \quad \mathbf{e}_i^T P^T B = (P\mathbf{e}_i)^T B = \mathbf{e}_{p_i}^T B = \mathbf{b}_{p_i}^T.$$

Hence, these operations reorder the columns and rows according to permutation vector $\mathbf{p}$, this is,

$$AP = \begin{bmatrix} \mathbf{a}_{p_1} & \cdots & \mathbf{a}_{p_n} \end{bmatrix} \quad \text{and} \quad P^T B = \begin{bmatrix} \mathbf{b}_{p_1}^T \\ \vdots \\ \mathbf{b}_{p_n}^T \end{bmatrix}.$$

Example 1.1. *The permutation matrix changing rows* 2 *and* 3 *of a* 3×3-*matrix is related to the permutation vector is* $\mathbf{p} = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix}$ *and obtained simply as*

$$P = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_3 & \mathbf{e}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

1.1.2 Problems

P1. (0.5p) Let

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}.$$

Find the permutation matrix corresponding to operations

(a) Swap rows 2 and 3

(b) Swap column 1 and 4

(c) Order rows as $3, 2, 1$

P2. (0.5p) Prove the claim:

Let $A \in \mathbb{R}^{n\times n}$ have orthonormal column vectors. Then A is unitary.

1.2 Block matrix notation

Block matrix notation is extensively used in this lecture note. Hence, this section should be studied with care.

See video introduction to block matrices in Youtube

In this section, we introduce block matrix notation which is used to avoid index notation in proofs and derivations. We limit the discussion to 2×2 block matrices, which are sufficient for our needs. Block matrices are obtained by splitting entries of a matrix vertically and horizontally into sub-matrices called blocks. In the following, we often divide matrices to 2×2 matrix blocks. For example, split $A \in \mathbb{R}^{n \times k}$ as

$$A = \begin{bmatrix} \underset{n_1 \times p}{A_{11}} & \underset{n_1 \times q}{A_{22}} \\ \underset{n_2 \times p}{A_{12}} & \underset{n_2 \times q}{A_{22}} \end{bmatrix} \quad \text{where } n = n_1 + n_2, \text{ and } k = p + q.$$

In the above equation, the size of each sub-matrix is written under it's symbol.

Example 1.2. *Consider the block decomposition of* 3×3 *matrix*

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

to 2×2 *block matrix as*

$$A = \begin{bmatrix} a_{11} & \mathbf{a}_{12}^T \\ \mathbf{a}_{21} & A_{22} \end{bmatrix} \quad \textit{where} \quad a_{11} = 1, \mathbf{a}_{12} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \mathbf{a}_{21} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}, A_{22} = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}.$$

This is, we have sliced A *as* $\left[\begin{array}{c|cc} 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}\right].$

We proceed to derive 2×2 block-matrix-matrix-product formula. Let $A \in \mathbb{R}^{n \times k}$, $B \in \mathbb{R}^{k \times m}$, and recall the matrix-matrix product formula

$$AB \in \mathbb{R}^{n \times m} \quad \text{and} \quad (AB)_{ij} = \sum_{l=1}^{k} a_{il} b_{lj}.$$

Matrices are often written using their column and row vectors as

$$A = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_m \end{bmatrix},$$

where $\{\mathbf{a}_i\}_{i=1}^n \subset \mathbb{R}^k$ and $\{\mathbf{b}_i\}_{i=1}^m \subset \mathbb{R}^k$. Observe, that we use column vectors, hence, $\mathbf{a}_1^T$ is a row vector. Using row and column vectors, the matrix-matrix product AB can be written as

$$AB = \begin{bmatrix} A\mathbf{b}_1 & \cdots & A\mathbf{b}_m \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^T B \\ \vdots \\ \mathbf{a}_n^T B \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^T\mathbf{b}_1 & \cdots & \mathbf{a}_1^T\mathbf{b}_m \\ \vdots & \ddots & \vdots \\ \mathbf{a}_n^T\mathbf{b}_1 & \cdots & \mathbf{a}_n^T\mathbf{b}_m \end{bmatrix}. \tag{1.3}$$

Using the above formula gives a Lemma for computing 2×2 block-matrix-matrix-product:

See video on computing product of 2×2 matrices in Youtube

See video on proving the product formula of 2×2 matrices in Youtube

Lemma 1.1. *Let* $A = \begin{bmatrix} \underset{n_1\times p}{A_{11}} & \underset{n_1\times q}{A_{12}} \\ \underset{n_2\times p}{A_{21}} & \underset{n_2\times q}{A_{22}} \end{bmatrix} \in \mathbb{R}^{n\times k}$ *and* $B = \begin{bmatrix} \underset{p\times m_1}{B_{11}} & \underset{p\times m_2}{B_{12}} \\ \underset{q\times m_1}{B_{21}} & \underset{q\times m_2}{B_{22}} \end{bmatrix} \in \mathbb{R}^{k\times m}$ *Then*

$$AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}. \tag{1.4}$$

Observe, that the 2×2 block-matrix-matrix product AB is computed similar to the 2×2 matrix-matrix product. This holds in general for all block-matrix-matrix-products. The sizes of matrix blocks must match in the sense that all products appearing in (1.4) are well defined. We prove Lemma 1.1 after giving a helper result.

Lemma 1.2. *Let* $\begin{bmatrix} \underset{n\times p}{C} & \underset{n\times q}{D} \end{bmatrix} \in \mathbb{R}^{n\times k}$ *and* $\begin{bmatrix} \underset{p\times m}{F} \\ \underset{q\times m}{G} \end{bmatrix} \in \mathbb{R}^{k\times m}$ *for* $k = p + q$. *Then*

$$\begin{bmatrix} C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix} = CF + DG. \tag{1.5}$$

Observe that the sizes of matrix blocks match in the sense that products CF and DG are well defined.

Proof. Denote the row vectors of C, D and column vectors of F, G as

$$C = \begin{bmatrix} \mathbf{c}_1^T \\ \vdots \\ \mathbf{c}_n^T \end{bmatrix}, \quad D = \begin{bmatrix} \mathbf{d}_1^T \\ \vdots \\ \mathbf{d}_n^T \end{bmatrix}, \quad F = \begin{bmatrix} \mathbf{f}_1 & \cdots & \mathbf{f}_m \end{bmatrix}, \quad \text{and} \quad G = \begin{bmatrix} \mathbf{g}_1 & \cdots & \mathbf{g}_m \end{bmatrix}.$$

We proceed to give a formula for computing entries of the product matrix $\begin{bmatrix} C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix} \in \mathbb{R}^{n \times m}$. The entry ij of the product matrix is obtained as

$$\mathbf{e}_i^T \begin{bmatrix} C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix} \mathbf{e}_j$$

where $\mathbf{e}_i \in \mathbb{R}^n$ and $\mathbf{e}_j \in \mathbb{R}^m$ are the ith and jth unit vectors. A direct calculation

$$\mathbf{e}_i^T \begin{bmatrix} C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix} \mathbf{e}_j = \begin{bmatrix} \mathbf{c}_i^T & \mathbf{d}_i^T \end{bmatrix} \begin{bmatrix} \mathbf{f}_j \\ \mathbf{g}_j \end{bmatrix} = \mathbf{c}_i^T \mathbf{f}_j + \mathbf{d}_i^T \mathbf{g}_j = (CF)_{ij} + (DG)_{ij}$$

gives the formula

$$\begin{bmatrix} C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix} = CF + DG. \tag{1.6}$$

□

Proof of Lemma 1.1. To prove (1.4) observe that by (1.3)

$$AB = \begin{bmatrix} \begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} & \begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \begin{bmatrix} B_{12} \\ B_{22} \end{bmatrix} \\ \\ \begin{bmatrix} A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} & \begin{bmatrix} A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{12} \\ B_{22} \end{bmatrix} \end{bmatrix}.$$

Application of product formula (1.5) completes the derivation. □

See video on Example 1.3 in Youtube

Example 1.3. *Next, we illustrate how block matrix notation is used in proofs and show that the product of two $n \times n$ lower triangular matrices is a lower triangular matrix. We formulate an induction proof with respect to the dimension of the lower triangular matrix using suitable 2×2 block division.*

Base step $n = 1$: *Trivially true.*

Induction assumption: *Product of two $k \times k$ lower triangular matrices is lower triangular.*

Induction step: *Let $L, T \in \mathbb{R}^{(k+1)\times(k+1)}$ be lower triangular matrices. Split*

$$L = \begin{bmatrix} \underset{1\times 1}{l_{11}} & 0 \\ \underset{k\times 1}{\mathbf{l}_{21}} & \underset{k\times k}{L_{22}} \end{bmatrix} \quad \textit{and} \quad T = \begin{bmatrix} \underset{1\times 1}{t_{11}} & 0 \\ \underset{k\times 1}{\mathbf{t}_{21}} & \underset{k\times k}{T_{22}} \end{bmatrix},$$

where L_{22}, T_{22} lower triangular matrices. Using the 2×2 block matrix-matrix product formula gives

$$LT = \begin{bmatrix} l_{11}t_{11} & 0 \\ \mathbf{l}_{21}t_{11} + L_{22}\mathbf{t}_{21} & L_{22}T_{22} \end{bmatrix}.$$

By induction assumption $L_{22}T_{22}$ is lower triangular matrix, which completes the proof.

1.2.1 Problems

P3. (1p) Let

$$A = \begin{bmatrix} \underset{n\times n}{A_{11}} & 0 \\ \underset{m\times n}{A_{21}} & \underset{m\times m}{A_{22}} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} \underset{n\times n}{B_{11}} & 0 \\ \underset{m\times n}{B_{21}} & \underset{m\times m}{B_{22}} \end{bmatrix}.$$

(a) Compute the block-matrix-matrix product AB.

(b) Find the inverse matrix of A. Hint: find B_{11}, B_{12}, B_{22} such that

$$\begin{bmatrix} \underset{n\times n}{A_{11}} & 0 \\ \underset{m\times n}{A_{21}} & \underset{m\times m}{A_{22}} \end{bmatrix} \begin{bmatrix} \underset{n\times n}{B_{11}} & 0 \\ \underset{m\times n}{B_{21}} & \underset{m\times m}{B_{22}} \end{bmatrix} = \begin{bmatrix} \underset{n\times n}{I} & 0 \\ 0 & \underset{m\times m}{I} \end{bmatrix}.$$

List assumptions (if any) that you have to make on A_{11}, A_{12}, and A_{22}.

(c) Argue that $\det A = 0$ implies that either $\det A_{11} = 0$ or $\det A_{22} = 0$.

P4. (1p) Let $E = \begin{bmatrix} \underset{1\times 1}{1} & \underset{1\times n}{0} \\ \underset{n\times 1}{-\mathbf{a}_{21}} & \underset{n\times n}{I} \end{bmatrix}$.

(a) Compute the product $E \begin{bmatrix} 1 & \underset{1\times n}{\mathbf{a}_{12}^T} \\ \underset{n\times 1}{\mathbf{a}_{21}} & \underset{n\times n}{A_{22}} \end{bmatrix}$

(b) Find the inverse matrix of E using the formula derived in the previuos problem. Check that your inverse is correct by computing the product EE^{-1}.

P5. (2p)

(a) Show that

$$\det \begin{bmatrix} \underset{n\times n}{I} & 0 \\ 0 & \underset{m\times m}{A_{22}} \end{bmatrix} = \det A_{22}.$$

Hint: recall the Laplace expansion for computing determinants and use induction with respect to parameter n.

(b) Modify the proof in (a) to show that

$$\det \begin{bmatrix} \underset{n\times n}{I} & \underset{n\times m}{A_{12}} \\ 0 & \underset{m\times m}{A_{22}} \end{bmatrix} = \det A_{22}. \tag{1.7}$$

P6. (0.5p)

(a) Compute $\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}$.

(b) Use properties of determinant, Problem 3, and (a) to show that $\det \begin{bmatrix} A_{11} & 0 \\ 0 & I \end{bmatrix} = \det A_{11}$.

P7. (1p) Consider the block matrix $A = \begin{bmatrix} \underset{n\times n}{A_{11}} & \underset{n\times m}{A_{12}} \\ 0 & \underset{m\times m}{A_{22}} \end{bmatrix}$, where A_{11} and A_{22} are invertible matrices.

(a) Compute the product

$$\begin{bmatrix} A_{11} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1} A_{12} A_{22}^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & A_{22} \end{bmatrix}.$$

(b) Use, equation (1.7), Problems 3,4, and decomposition in (a) to show that $\det A = \det A_{11} \det A_{22}$.

(c) Argue by Problem 3 that $\det A = \det A_{11} \det A_{22}$ even if A_{11} or A_{22} are not invertible.

P8. (1p) Let

$$M = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.8}$$

(a) Use suitable 2×2 block decomposition to compute M^2.

(b) Use inverse matrix formula from Problem 3 to compute M^{-1}.

1.2.2 Back-substitution in block matrix notation

This section gives a recursive definition of the back-substitution algorithm. Using recursion is necessary to express the algorithm in block matrix notation. This section should be studied with care.

In this section, we use block matrix notation to define the back - substitution algorithm. Our definition is recursive with respect to dimension of the linear system. Using such definition allows simple treatment of matrices with different dimension using the block matrix notation. We use similar techniques to study the LU and the Cholesky factorisations.

See video on solution of upper triangular systems in Youtube

Consider the linear system: Find $\mathbf{x} \in \mathbb{R}^n$ satisfying

$$U\mathbf{x} = \mathbf{b},$$

where the coefficient matrix $U \in \mathbb{R}^{n \times n}$ is upper triangular and $\mathbf{b} \in \mathbb{R}^n$.

Definition 1.1. *Matrix $U \in \mathbb{R}^{n\times n}$ is upper triangular, if*

$$U_{ij} = 0 \quad \textit{for } i > j.$$

This is

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix} \quad \text{or} \quad U = \begin{bmatrix} \# & \# & \cdots & \# \\ & \# & \cdots & \# \\ & & \ddots & \vdots \\ & & & \# \end{bmatrix}.$$

Here we use notational convention where the location of non-zero entries in the matrix is indicated by # and zero entries are omitted. Such convention is used when the location of non-zero entries is important but their value is not.

Triangular linear systems are solved using back-substitution algorithm. We use a definition that is recursive with respect to the dimension of the coefficient matrix. The function $triusolve(U, \mathbf{b})$ returns solution to linear system $U\mathbf{x} = \mathbf{b}$ for invertible upper triangular matrix $U \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$.

For $n = 1$, $triusolve(U, b) = \frac{b}{U}$.

For $n > 1$, we use a recursive definition. First, split the linear system $U\mathbf{x} = \mathbf{b}$ as

$$\begin{bmatrix} U_{11} & \mathbf{u}_{12} \\ 0 & u_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ b_2 \end{bmatrix} \tag{1.9}$$

where x_2, b_2 ja u_{22} are scalars, $U_{11} \in \mathbb{R}^{(n-1)\times(n-1)}$ and $\mathbf{u}_{12}, \mathbf{x}_1, \mathbf{b}_1 \in \mathbb{R}^{n-1}$. As U is invertible, $u_{22} \neq 0$, U_{11} is invertible[1], and

$$x_2 = \frac{b_2}{u_{22}}.$$

First equation in (1.9) states $U_{11}\mathbf{x}_1 = \mathbf{b}_1 - \mathbf{u}_{12}x_2$. As coefficient matrix $U_{11} \in \mathbb{R}^{(n-1)\times(n-1)}$ is invertible and upper triangular, $\mathbf{x}_1$ is obtained recursively as $\mathbf{x}_1 = triusolve(U_{11}, \mathbf{b}_1 - \mathbf{u}_{12}x_2)$. Hence,

$$triusolve(U, b) = \begin{bmatrix} \mathbf{x}_1 \\ x_2 \end{bmatrix}.$$

See a video on implementing the back substitution algorithm in Youtube

An example implementation of the above function is given below.

```
function x = triusolve2(U,b)

n = size(U,2);
x = zeros(n,1);

% Define matrix and vector blocks.
U11 = U(1:(n-1),1:(n-1));
u12 = U(1:(n-1),n);
u22 = U(n,n);

b1 = b(1:(n-1));
b2 = b(n);
```

[1]See problem 7 on page 10

```
% solve x2.
x(n) = b2/u22;

if( n > 1 )
% solve x1 using recursive function call.
x(1:(n-1)) = triusolve2(U11,b1-u12*x(n));
end

end
```

Using recursive function calls is not very efficient. A better strategy is to update the vector **b** during the algorithm and use a for-loop to conduct the computation. An example implementation using such *update strategy* is given below.

```
function x = triusolve(U,b)

N = size(U,2);

x = zeros(N,1);

for n=N:-1:1
    % Define matrix and vector blocks.

    U11 = U(1:(n-1),1:(n-1));
    u12 = U(1:(n-1),n);
    u22 = U(n,n);

    b1 = b(1:(n-1));
    b2 = b(n);

    % solve x(i).
    x(n) = b2/u22;

    % update vector b
    b(1:(n-1)) = b1 - u12*x(n);
end
```

The above algorithm can be easily modified to solve lower triangular linear systems.

1.2.3 Problems

P9. (2p) Use block matrix notation to give a recursive definition of function $trilsolve(L, \mathbf{b})$ that returns solution of linear system $L\mathbf{x} = \mathbf{b}$ where L is a lower triangular matrix.

P10. (2p)

(a) Give a recursive implementation of *trilsolve* in Matlab

(b) Modify recursive implementation in (a) to use the update strategy.

P11. (1p)

(a) Compute, how many arithmetic operations are needed to solve a $N \times N$ - upper triangular system.

(b) Measure the time required to solve upper triangular linear systems using Matlab backslash, back substitution using recursive implementation, and back substitution using update strategy. Generate random upper triangular matrices with dimension $N = 10$, 50, 100, 200, 300, 400, and 500 using commands `rand` and `triu`. For each dimension, compute average solution time for each method from 100 solves. Plot average solution times as a function of N using a logarithmic scale. Does the result correspond to (a) ?

1.3 Finite difference method

This section gives an example application that leads to linear system with large, sparse and s.p.d coefficient matrix. It is extra material and can be skipped. Or just have a look at the video.

See video introduction to finite difference method

Let $\Omega \subset \mathbb{R}^2$ be a bounded open set with sufficiently regular boundary and recall the definition of the Laplace operator Δ in $\mathbb{R}^2$,

$$\Delta := \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}.$$

The Poisson's equation in Ω is: Find $u \in C^2(\Omega) \cap C(\overline{\Omega})$ such that

$$\begin{cases} -\Delta u = f \text{ in } \Omega \\ \quad u = 0 \text{ on } \partial\Omega \end{cases} \tag{1.10}$$

where f is a given function[2]. The Poisson's equation is a simple model problem for other PDEs that appear, e.g., in electrical or mechanical engineering.

[2]Here $C^2(\Omega)$ and $C(\overline{\Omega})$ are spaces of functions that have two derivatives in open set Ω and functions that are continuous in closure of Ω, respectively. The differentiability is required for the equation $-\Delta u = f$ to be well defined, and continuity up to boundary for the boundary condition $u = 0$ to be meaningful

Several different numerical methods have been developed to find approximate solutions to (1.10). We use the finite difference method, in which one seeks for an approximation to the point-wise values of u. The first step is to derive the central difference approximation of the Laplace operator.

Let $h \in \mathbb{R}$, $h > 0$. The Taylor expansion[3] of the solution u with respect to the variable x_1 gives

$$u(x_1+h,x_2) = u(x_1,x_2) + \frac{\partial u}{\partial x_1}(x_1,x_2)h + \frac{1}{2}\frac{\partial^2 u}{\partial x_1^2}(x_1,x_2)h^2 + \frac{1}{6}\frac{\partial^3 u}{\partial x_1^3}(x_1,x_2)h^3 + h.o.t.$$
$$u(x_1-h,x_2) = u(x_1,x_2) - \frac{\partial u}{\partial x_1}(x_1,x_2)h + \frac{1}{2}\frac{\partial^2 u}{\partial x_1^2}(x_1,x_2)h^2 - \frac{1}{6}\frac{\partial^3 u}{\partial x_1^3}(x_1,x_2)h^3 + h.o.t.,$$

where *h.o.t* is used to denote higher order terms with respect to h. Subtracting the two above equations and dividing by h^2 gives

$$\frac{\partial^2 u}{\partial x_1^2}(x_1,x_2) \approx \frac{u(x_1+h,x_2) - 2u(x_1,x_2) + u(x_1-h,x_2)}{h^2}. \tag{1.11}$$

Similar computations for the x_2 - component give

$$\frac{\partial^2 u}{\partial x_2^2}(x_1,x_2) \approx \frac{u(x_1,x_2+h) - 2u(x_1,x_2) + u(x_1,x_2-h)}{h^2}. \tag{1.12}$$

Combining (1.11) and (1.12) yields the *central difference approximation* of the Laplace operator:

$$(\Delta u)(x_1,x_2) \approx \frac{u(x_1-h,x_2) + u(x_1+h,x_2) - 4u(x_1,x_2) + u(x_1,x_2-h) + u(x_1,x_2+h)}{h^2}.$$

The accuracy of this approximation depends on h as well as on the properties of the function u.

Next, consider the domain $\Omega = (0,1)^2$ and a uniform $N \times N$-grid composed of points

$$\mathbf{x}_{ij} = \frac{1}{N-1}\begin{bmatrix} i-1 \\ j-1 \end{bmatrix} \quad \text{for} \quad i,j \in \{1,\ldots,N\}$$

see Figure 1.1. The distance between grid points is denoted by $h := \frac{1}{N-1}$ and the value of u at the grid point $\mathbf{x}_{ij}$ by $\mathbf{u}_{ij} := u(\mathbf{x}_{ij})$.

Observe that the indices of interior grid points $\mathbf{x}_{ij} \in \Omega$ and boundary grid points $\mathbf{x}_{ij} \in \partial\Omega$ are

$$I := \{\ (i,j) \mid i,j \in \{2,\ldots,N-1\}\ \}$$

[3]Observe that the expansion requires additional regularity of u, i.e $u \in C^3(\Omega)$.

and

$$B := \{\ (i,j) \mid i,j \in \{1,\dots,N\}\ \} \setminus I,$$

respectively. At interior grid points, the finite difference approximation states that:

$$\frac{\mathbf{u}_{(i-1)j} + \mathbf{u}_{(i+1)j} + \mathbf{u}_{i(j-1)} + \mathbf{u}_{i(j+1)} - 4\mathbf{u}_{ij}}{h^2} \approx f(\mathbf{x}_{ij}). \tag{1.13}$$

Due to the boundary condition $u = 0$ on $\partial\Omega$,

$$\mathbf{u}_{ij} = 0 \tag{1.14}$$

at boundary grid points.

In finite difference method, one poses (1.13) as equality and seeks for *approximate point wise values of u satisfying* the resulting linear system. For notional simplicity, we denote the FD-approximation also by $\mathbf{u}_{ij}$. The challenge in solving $\mathbf{u}_{ij}$ is constructing the coefficient matrix of the linear system (1.13)-(1.14), which requires careful index handling. First, collect the variables $\mathbf{u}_{ij}$ into the vector $\mathbf{U} \in \mathbb{R}^{N^2}$ as

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_{11} \\ \mathbf{u}_{12} \\ \mathbf{u}_{13} \\ \vdots \\ \mathbf{u}_{21} \\ \mathbf{u}_{22} \\ \mathbf{u}_{23} \\ \vdots \end{bmatrix}$$

It is helpful to explicitely define mapping $\sigma(i,j) = (i-1)N + j$ that aids in index handling. The value $\mathbf{u}_{ij}$ resides in the element $\sigma(i,j)$ of vector $\mathbf{U}$. The vector $\mathbf{U}$ satisfies

$$A\mathbf{U} = \mathbf{b}.$$

The non-zero entries of the coefficient matrix $A \in \mathbb{R}^{N^2 \times N^2}$ and vector $\mathbf{b} \in \mathbb{R}^{N^2}$ are:

$$\begin{aligned} a_{\sigma(i,j)\sigma(i-1,j)} &= 1, & a_{\sigma(i,j)\sigma(i+1,j)} &= 1, \\ a_{\sigma(i,j)\sigma(i,j-1})) &= 1, & a_{\sigma(i,j)\sigma(i,j+1)} &= 1, \\ a_{\sigma(i,j)\sigma(i,j)} &= -4, & b_{\sigma(ij)} &= f(\mathbf{x}_{ij}). \end{aligned}$$

for interior indices $i,j \in I$ and

$$a_{\sigma(i,j)\sigma(i,j)} = 1, \quad b_{\sigma(ij)} = 0$$

for boundary indices $i, j \in B$. The matrix A is assembled in the following code.

```
N = 50;
A = sparse( N^2,N^2);
h = 1/(N-1);

ijmap = @(i,j)( (i-1)*N + j);
active = []; % collect not boundary nodes here.

for i=1:N
    for j=1:N

        x(i,j) = (i-1)/(N-1); y(i,j) = (j-1)/(N-1);

        if( (i > 1) & (i < N) & ( j > 1) & ( j < N))

        % This is the row corresponding to point (i,j)
        I1 = ijmap(i,j);

        active = [active I1];

        A(I1, ijmap(i-1,j)) =  -1/h^2;
        A(I1, ijmap(i+1,j)) =  -1/h^2;
        A(I1, ijmap(i,j-1)) =  -1/h^2;
        A(I1, ijmap(i,j+1)) =  -1/h^2;
        A(I1, I1)           =   4/h^2;


        b(I1,1) = 1;
        end

    end
end

% system without active rows
A = A(active,active);
b = b(active);

% solve !
u = zeros(N^2,1);
u(active) = A\b;

% visualize u.
U = reshape(u,N,N);
figure;S = surf(x',y',U);
set(S,'facecolor','interp');
```

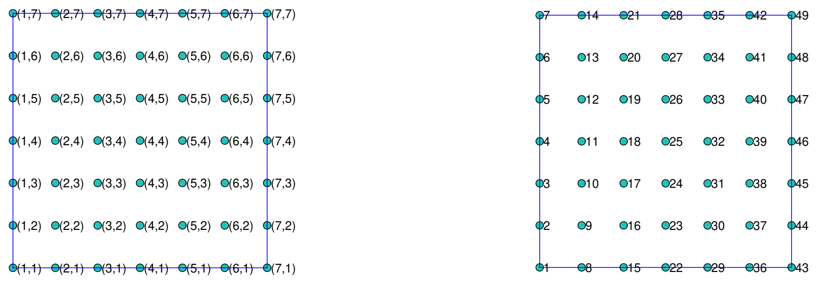

Figure 1.1: Node numbering in i, j - system vs. node numbering corresponding to vector $\mathbf{U}$

The rows of A related to boundary indices are not interesting and they are eliminated. Let $P \in \mathbb{R}^{N^2 \times N^2}$ be a permutation matrix ordering the rows of U as

$$P^T \mathbf{U} = \begin{bmatrix} \mathbf{U}_I \\ \mathbf{U}_B \end{bmatrix}$$

where $\mathbf{U}_I \in \mathbb{R}^{(N-2)^2}$ and $\mathbf{U}_B \in \mathbb{R}^{4(N-1)}$ are the values of $\mathbf{u}_{ij}$ related to interior and boundary grid points, respectively. Application of the same splitting to A and $\mathbf{b}$ gives

$$P^T AT = \begin{bmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{bmatrix} \quad \text{and} \quad P^T \mathbf{b} = \begin{bmatrix} \mathbf{b}_I \\ \mathbf{b}_B \end{bmatrix}.$$

As $\mathbf{U}_B = 0$ by (1.14), $\mathbf{U}_I$ satisfies the system $A_{II}\mathbf{U}_I = \mathbf{b}_I$ where the matrix A_{II} depends on the permutation P. The matrix $A_{II} \in \mathbb{R}^{N^2 \times N^2}$ is symmetric and has at most five non-zero entries on ever column. It's sparsity structure, i.e. location of non-zero entries, generated by the above code is visualized in Figure 1.2 using the Matlab command `spy(A)`. The accuracy of the computed approximate point-wise values depends on h. If accurate solutions are sought for, h is small and the number of grid points N can be large. For example, N can be of the order $N = 1000$, which results to linear system with dimension $(N-2)^2 \approx 10^6$.

1.3.1 Problems

P12. (1p)

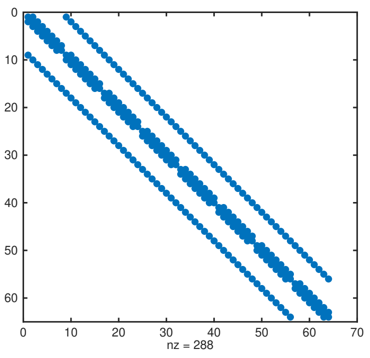

Figure 1.2: Nonzero entries of the matrix A_{II} related to the linear system given in equation (1.13).

(a) Derive the finite difference approximation of Laplace operator in 1D.

(b) Write a Matlab code to solve the 1D Poisson's equation: find $u(x) \in C^2((0,1)) \cap C([0,1])$ satisfying

$$-u''(x) = 1 \text{ in } (0,1) \quad \text{and} \quad u(0) = u(1) = 0.$$

Plot the solution u.

P13. (2p) Let $A \in \mathbb{R}^{2n\times 2n}$, $n > 3$, satisfy

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}. \tag{1.15}$$

(a) Let $\mathbf{x}$ satisfy $A\mathbf{x} = 0$. Show that $\mathbf{x}$ also satisfies

$$\begin{bmatrix} x_{i+1} \\ x_{i+2} \end{bmatrix} = C \begin{bmatrix} x_{i-1} \\ x_i \end{bmatrix} \quad \text{for } i \in \{1, \ldots, 2n-2\} \quad \text{and} \quad C = \begin{bmatrix} -1 & 2 \\ -2 & 3 \end{bmatrix}$$

(b) Use the Jordan decomposition of C to show that

$$\begin{bmatrix} x_{2n-1} \\ x_{2n} \end{bmatrix} = \begin{bmatrix} -2n+1 & 2n \\ -2n & 2n+1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

(c) Show that x_2 and x_1 satisfy

$$\begin{bmatrix} 2 & -1 \\ -2n-1 & 2n+2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0.$$

Use (b) to argue that $N(A) = \{0\}$ and A is invertible.

P14. (1p) Consider the matrix A defined in (1.15).

(a) Show by direct computation that $\mathbf{x}^T A\mathbf{x} \geq 0$, for any $\mathbf{x} \in \mathbb{R}^n$ i.e. A is positive semi-definite matrix.

(b) Argue that any symmetric and positive semi-definite matrix with trivial null-space is positive definite.

(c) Use (b) and Problem 13 to argue that A is positive definite.

1.4 Compressed column storage format

This section discusses sparse matrix storage formats used in practical implementation of sparse matrix data types. The aim is to highlight the fact that computational complexity of accessing matrix rows, columns, and elements depends on the chosen storage format. This has to be taken into account when designing high-level matrix algorithms. It also explains why sparse matrix literature gives several alternative ways to compute, e.g., the Cholesky factorisation. This Section is extra material and can be skipped.

In this section, we discuss how sparse matrices are stored in the memory of a computer. The applied storage format affects the time required to access matrix elements which should be taken into account when designing sparse matrix algorithms.

See video on CCS storage format in Youtube

A dense matrix is typically stored as a two-dimensional array of numbers, whereas only non-zero entries of a sparse matrix are stored. There are several data structures used for this purpose, the most common ones being compressed row storage (CRS) and compressed column storage (CCS) formats. For example, Matlab uses CCS format to store sparse matrices.

The compressed column storage format uses three arrays:

- **Values:** List of matrix entries ordered column wise.
- **Row indices:** The row index for each of the entries
- **Column pointers:** Index of the first entry of a every column in the values and row index lists.

The CCS format is best illustrated by examples.

Example 1.4. *Let*

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

In CCS format A is stored as

$$\begin{aligned} vals &= \begin{bmatrix} a_{11} & a_{21} & a_{12} & a_{22} \end{bmatrix} \\ row\_ind &= \begin{bmatrix} 1 & 2 & 1 & 2 \end{bmatrix} \\ col\_ptr &= \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \end{aligned}$$

Example 1.5. *Let*

$$B = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}.$$

In CCS format, B is stored as

$$\begin{aligned} vals &= \begin{bmatrix}-2 & 1 & 1 & -2 & 1 & 1 & -2\end{bmatrix} \\ row\_ind &= \begin{bmatrix}1 & 2 & 1 & 2 & 3 & 2 & 3\end{bmatrix} \\ col\_ptr &= \begin{bmatrix}1 & 3 & 6 & 8\end{bmatrix}\end{aligned}$$

In the above examples, the column pointer has an extra entry with value $length(vals)+1$ that is used to simplify implementation of matrix operations. If the extra entry is used, the column i is accessed simply as

```
A.col_ptr = [1 3 6 8];
A.rowind =  [1  2   1  2 3    2  3  ];
A.val =     [-2 1   1 -2 1    1 -2  ];

col_i = A.val( A.col_ptr(i):(A.col_ptr(i+1)-1) );
```

The CCS format has constant access time for columns of a matrix. Accessing rows requires looping over the row index array, hence the required time depends linearly on the size of the matrix. Element access is done by first accessing the column and then finding the desired entry. If the row indices are sorted, the desired entry can be sought for using, e.g., bisection search. In this case, the access time for the element ij has logarithmic dependency on the number of nonzero entries in the column j.

The access times in Matlab can be studied with the following test code. The resulting times are plotted in Figure 1.3

```
Nlist = floor(linspace(1,1e5,10));
row_timer = []; col_timer = []; ele_timer = [];

for n = Nlist

    e = ones(n,1);
    A = spdiags([e -2*e e], -1:1, n, n);

    I = randi(n,1e3,1); J = randi(n,1e3,1);

    T = tic;
    for j=1:1e3
       x=A(I(j),J(j));

    end
    ele_timer = [ele_timer toc(T)/1e3];

    T = tic;
```

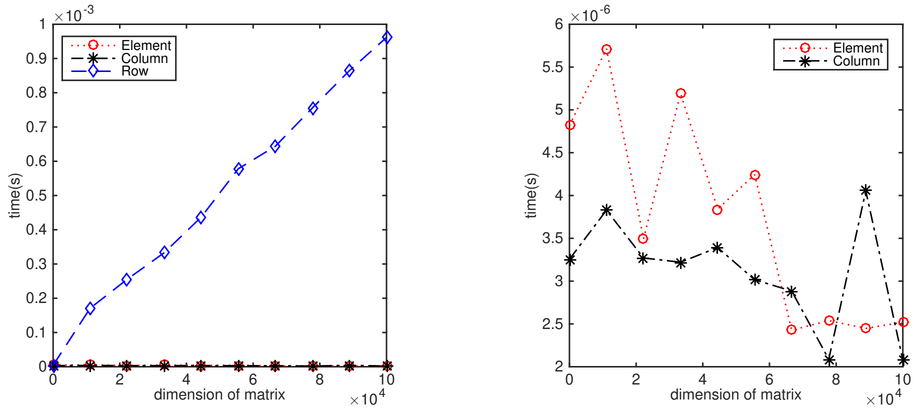

Figure 1.3: Example of access times for elements, rows, and columns of the one dimensional finite difference matrix $A \in \mathbb{R}^{n\times n}$ in (1.15) as a function of the dimension n. The test is done in Matlab.

```
    for j=1:1e3
       x=A(:,I(j));
    end
    col_timer = [col_timer toc(T)/1e3];

    T = tic;
    for j=1:1e3
       x=A(I(j),:);

    end
    row_timer = [row_timer toc(T)/1e3];

end

figure; plot(Nlist,ele_timer,'ro:',Nlist,col_timer,'k*-.',Nlist,row_timer,'bd--');
legend('Element','Column','Row');
ylabel('time(s)'); xlabel('dimension of matrix');

figure; plot(Nlist,ele_timer,'ro:',Nlist,col_timer,'k*-.');
legend('Element','Column');
ylabel('time(s)'); xlabel('dimension of matrix');
```

1.4.1 Additional material

- For more information on sparse matrices in Matlab, see

  John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in matlab: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13(1):333–356, 1992

1.4.2 Problems

P15. (0.5p) Let

$$A_1 := \begin{bmatrix} 1 & 0 & 2 & 0 \\ 3 & 0 & 4 & 0 \\ 0 & 5 & 0 & 6 \\ 7 & 8 & 9 & 10 \end{bmatrix}. \tag{1.16}$$

and

```
N = 5;
A2 = 2*eye(N) + diag(-ones(N-1,1),1)+ diag(-ones(N-1,1),-1)
```

Write A_1 and A_2 using the compressed column storage scheme.

P16. (1p) Write a Matlab-function `[val,row,col] = mat2ccs(A)` that returns the CCS representation of matrix A. Test your implementation using matrices A_1 and A_2 defined in Problem 15.

P17. (1p) Write Matlab functions `coli = ccs_col(val,row,col,i)` and `rowi = ccs_row(val,row,col,i)` that return column and row i of a matrix represented in CCS format by val, row, and col-vectors. Repeat the column and row access time test using your own functions.

1.5 Gaussian elimination

This is section is a review of the Gaussian elimination process. Read it to refresh your memory, or skip it.

See video introduction to Gaussian elimination in Youtube

Let $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, and consider the linear system: Find $\mathbf{x} \in \mathbb{R}^n$ satisfying

$$A\mathbf{x} = \mathbf{b}. \tag{1.17}$$

Gaussian elimination is an algorithm that transforms (1.17) to the equivalent system: Find $\mathbf{x} \in \mathbb{R}^n$ satisfying

$$U\mathbf{x} = \widetilde{\mathbf{b}}, \tag{1.18}$$

where the coefficient matrix $U \in \mathbb{R}^{n\times n}$ is upper triangular and $\widetilde{\mathbf{b}} \in \mathbb{R}^n$. System (1.18) can be easily solved using the back substitution algorithm, see Section 1.2.2.

We proceed by applying the Gaussin elimination to (1.17) in it's component form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3n}x_n &= b_3 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n &= b_n \end{aligned}. \tag{1.19}$$

For simplicity, assume that entry $a_{11} \neq 0$. The case $a_{11} = 0$ is discussed in Section 1.5.1. The variable x_1 is solved from the first equation in (1.19) as

$$x_1 = \frac{b_1}{a_{11}} - \sum_{j=2}^{n} \frac{a_{1j}}{a_{11}} x_j.$$

Using this expression, we eliminate variable x_1 from equations $\{2,\ldots,n\}$ in (1.19). This yields new linear system for $\mathbf{x}$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \ldots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ a_{32}^{(2)}x_2 + a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n &= b_3^{(2)} \\ &\vdots \\ a_{n2}^{(2)}x_2 + a_{n3}^{(2)}x_3 + \ldots + a_{nn}^{(2)}x_n &= b_n^{(2)}, \end{aligned} \tag{1.20}$$

with coefficients $a_{ij}^{(2)}$

$$a_{ij}^{(2)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} \quad \text{for} \quad i,j \in \{2,\ldots,n\}.$$

This is, the transformed system is obtained by multiplying the first equation in (1.19) with $-a_{i1}a_{11}^{-1}$ and adding it to the equation i in (1.19). Observe that the resulting equations $\{2,\ldots,n\}$ in (1.20) are independent of x_1.

The above process is the first step of the Gaussian elimination algorithm. Assuming that $a_{22}^{(2)} \neq 0$, the algorithm proceeds by eliminating variable x_2 from the transformed equations {3,. . . ,n} in system (1.20). Under assumption $a_{22}^{(2)} \neq 0$,

$$x_2 = \frac{b_{22}^{(2)}}{a_{22}^{(2)}} - \sum_{j=3}^{n} \frac{a_{2j}^{(2)}}{a_{22}^{(2)}} x_j.$$

Identically, variable x_2 is eliminated from the transformed equations $\{3, \ldots, n\}$ in (1.20). New coefficients are computed as :

$$a_{ij}^{(3)} = a_{ij}^{(2)} - \frac{a_{i2}^{(2)}}{a_{22}^{(2)}} a_{2j} \quad \text{for} \quad i, j \in \{3, \ldots, n\}.$$

Assuming $a_{ii}^{(i)} \neq 0$ for $i \in \{3, \ldots, n\}$, the above process can be repeated until (1.19) has been transformed to the system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \ldots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ a_{33}^{(3)}x_3 + \ldots + a_{3n}^{(3)}x_n &= b_3^{(3)} . \\ &\vdots \\ a_{nn}^{(n)}x_n &= b_n^{(n)} \end{aligned}$$

The matrix elements $a_{ii}^{(i)}$ for $i \in \{1, \ldots, n\}$ are called pivots. Here and in the following we set $a_{ij}^{(1)} := a_{ij}$.

We denote the coefficient matrix of intermediate transformed system on step $k \in \{1, \ldots, n\}$ as $A^{(k)} \in \mathbb{R}^{n \times n}$. For $k = 1$ we define $A^{(1)} := A$. The systems $A^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$ and $A^{(3)}\mathbf{x} = \mathbf{b}^{(3)}$ are given in (1.19) and (1.20). For $k \in \{2, \ldots, n\}$, matrix $A^{(k)}$ has the block structure

$$A^{(k)} = \begin{bmatrix} U^{(k)} & A_{12}^{(k)} \\ 0 & A_{22}^{(k)} \end{bmatrix}.$$

where the matrix $U^{(k)} \in \mathbb{R}^{(k-1)\times(k-1)}$ is upper triangular.

Example 1.6 demonstrates how the Gaussian elimination algorithm is used in hand calculations.

Example 1.6. *Consider the linear system*

$$\begin{cases} x_1 + x_2 + x_3 & = 0 \\ x_1 + 2x_2 + 4x_3 & = 1 \\ x_1 + 3x_2 + 2x_3 & = 7. \end{cases}$$

In matrix form, the above system is: find $\mathbf{x} \in \mathbb{R}^3$ *satisfying*

$$A\mathbf{x} = \mathbf{b}, \quad \textit{where} \quad A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 2 \end{bmatrix} \quad \textit{and} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 7 \end{bmatrix}.$$

When running Gaussian elimination algorithm by hand, matrix A *and vector* $\mathbf{b}$ *are written in the same table as*

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 1 \\ 1 & 3 & 2 & 7 \end{array}\right].$$

The row operations are marked on the left hand side of the table.

$$\begin{array}{r} \\ -Y1 \\ -Y1 \end{array} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 1 & 2 & 4 & 1 \\ 1 & 3 & 2 & 7 \end{array}\right] \rightarrow \begin{array}{r} \\ \\ -2Y2 \end{array} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 2 & 1 & 7 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & -5 & 5 \end{array}\right].$$

The resulting linear system is solved using the back-substitution algorithm.

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & -5 & 5 \end{array}\right] \overset{x_3=-1}{\rightarrow} \left[\begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 1 & 4 \end{array}\right] \overset{x_2=4}{\rightarrow} \left[\begin{array}{c|c} 1 & -3 \end{array}\right] \rightarrow x_1 = -3.$$

This process yields the solution $\mathbf{x} = \begin{bmatrix} -3 & 4 & -1 \end{bmatrix}^T$.

Problems

P18. (0.5p) Solve the linear system

$$\begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 2 & 2 \\ -2 & 1 & 0 & 1 \\ -1 & 0 & -4 & 2 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

by hand using Gaussian elimination and back-substitution. Check your solution using Matlab.

P19. (1p) Let $A \in \mathbb{R}^{n\times n}$. Assume, that all pivots during Gaussian elimination are no-zeros. Estimate the total number of arithmetic operations $\cdot, +, -, /$ in the elimination process of A.

Use the identity

$$\sum_{x=1}^{n-1} (x+\alpha)^k \le \int_0^{n-1} (x+\alpha+1)^k, \tag{1.21}$$

for $\alpha \in \mathbb{R}$ and $k \ge 0$ to give a simple upper bound for the number of operations. Identity (1.21) follows from geometric interpretation of the sum, see Figure 1.4.

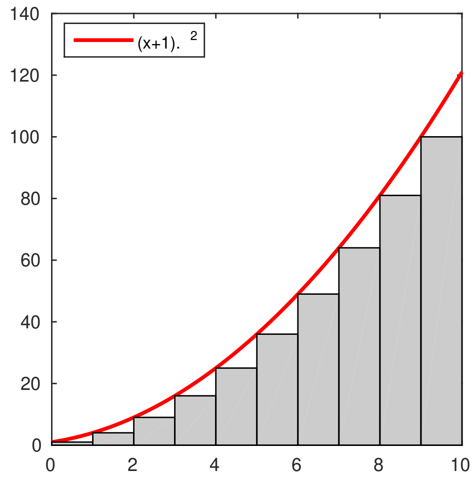

Figure 1.4: Geometry interpretation of estimate (1.21)

1.5.1 Pivoting

In this section, we modify Gaussian elimination process to cope with zero pivot elements. If pivot is zero, an additional pivoting step changing the order of equations or unknowns is conducted before the elimination step. Changing the order of rows and/or columns is expressed using permutation matrices.

Example 1.7. *Consider the linear system*

$$\begin{cases} x_1 + x_2 + x_3 & = 0 \\ x_1 + x_2 + 4x_3 & = 3 \\ x_1 + 3x_2 + 2x_3 & = 7. \end{cases}$$

To perform Gaussian elimination by hand, we write the system in a table:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 1 & 1 & 4 & 3 \\ 1 & 3 & 2 & 7 \end{array}\right]$$

First step of elimination yields:

$$\begin{array}{c} \\ -Y1 \\ -Y1 \end{array} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 1 & 1 & 4 & 3 \\ 1 & 3 & 2 & 7 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & 0 & 3 & 3 \\ 0 & 2 & 1 & 7 \end{array}\right]$$

Because the pivot $a_{22}^{(2)} = 0$ we exchange rows two and three. This corresponds to changing the order of equations in the original linear system and does not change the solution. We obtain,

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & 2 & 1 & 7 \\ 0 & 0 & 3 & 3 \end{array}\right]. \tag{1.22}$$

The coefficient matrix has now been transformed to upper triangular one, and $\mathbf{x}$ is solved using back-substitution.

The permutation vector corresponding to changing rows 2 and 3 is $\mathbf{p} = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix}$ and the related permutation matrix

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

In this example, transformed system (1.22) *is obtained by applying Gaussian elimination without pivoting to linear system*

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

We show in Section 1.6 that changing the order of equations or unknowns during the elimination process does not change the solution of the linear system. Further, identical transformed system is obtained by applying Gaussian elimination without pivoting to the permuted linear system

$$P^T\mathbf{A}Q(Q^{-1}\mathbf{x}) = P^T\mathbf{b},$$

where P and Q are permutation matrices re-ordering eqautions and entries of $\mathbf{x}$.

When running the Gaussian elimination process by hand, the pivot is chosen so that the resulting computations are as simple as possible. When Gaussian elimination is implemented using a computer, pivoting is applied on every step to improve numerical stability of the algorithm. Numerical stability is discussed later in this course.

Different pivoting strategies on step k are:

- **Column-pivoting**: Choose entry $a_{ik}^{(k)}$ for $i \in \{k, \dots, n\}$ with largest absolute value as pivot
- **Row-pivoting**: Choose entry $a_{kj}^{(k)}$ for $j \in \{k, \dots, n\}$ with largest absolute value as pivot
- **Full-pivoting**: Choose entry $a_{ij}^{(k)}$ for $i, j \in \{k, \dots, n\}$ with largest absolute value as pivot

1.5.2 Problems

P20. (0.5p) Solve the linear system

$$\begin{bmatrix} 1 & 0 & 3 & 4 \\ 2 & 0 & 9 & 9 \\ 0 & 1 & 3 & 2 \\ 0 & 3 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Using Gaussin elimination and back substitution.

1.5.3 Elimination matrices and LU-factorisation

In this section, we express row operations conducted during Gaussian elimination process using elimination matrices. This representation allows us to prove equivalence between the original and the transformed linear system. It also yields the LU factorisation of a matrix A.

For simplicity, assume that all pivot elements are nonzero. On step k of the elimination process, row k is first multiplied with a $-\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ and then added to row i for $i \in \{k+1, \dots, n\}$. The corresponding linear mapping is

$$f_k(\mathbf{x})_i = \begin{cases} \mathbf{x}_i & i \leq k \\ \mathbf{x}_i - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\mathbf{x}_k & i > k \end{cases}.$$

When pivots $a_{kk}^{(k)} \neq 0$, the mapping f_k is invertible and

$$f_k^{-1}(\mathbf{x})_i = \begin{cases} \mathbf{x}_i & i \leq k \\ \mathbf{x}_j + \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\mathbf{x}_k & i > k \end{cases}.$$

First step of the elimination process can be stated as $f_1(A\mathbf{x}) = f_1(\mathbf{b})$. Let $E_1 \in \mathbb{R}^{n\times n}$ be the matrix representation of the linear mapping f_1, this is $f_1(\mathbf{x}) = E_1\mathbf{x}$. The matrix representation is obtained as $E_1 = \begin{bmatrix} f_1(\mathbf{e}_1) & f_1(\mathbf{e}_2) & \cdots & f_1(\mathbf{e}_n)\end{bmatrix}$, where $\{\mathbf{e}_i\}_{i=1}^n$ are the Cartesian unit vectors. This yields

$$E_1 = \begin{bmatrix} 1 & & & \\ -\frac{a_{21}}{a_{11}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{a_{n1}}{a_{11}} & & & 1 \end{bmatrix}$$

Using the above matrix representation gives the relation

$$A^{(2)} = E_1 A \quad \text{and} \quad \mathbf{b}^{(2)} = E_1\mathbf{b},$$

where $A^{(2)}$ is the transformed coefficient matrix obtained from step 1. Transformation of linear system $A\mathbf{x} = \mathbf{b}$ to upper triangular form corresponds to

$$f(A\mathbf{x}) = f(\mathbf{b}). \tag{1.23}$$

where $f = f_{n-1} \circ \dots f_1$. Let E_k be the matrix representation of the linear mapping f_k. Then the final transformed system satisfies

$$A^{(n)} = E_{n-1}\dots E_2E_1A \quad \text{and} \quad \mathbf{b}^{(n)} = E_{n-1}\dots E_2E_1\mathbf{b}. \tag{1.24}$$

As $A^{(n)}$ is an upper triangular matrix, we denote $U = A^{(n)}$. Observe that the structure of elimination matrices changes for every k making them difficult to write using block matrix notation. This difficulty is addressed in Section 1.6 using recursive definition of the Gaussian elimination process.

Observe, that $f^{-1} = f_1^{-1} \circ \ldots f_{n-1}^{-1}$. Hence f has an inverse, and $f(x) = 0 \Rightarrow \mathbf{x} = 0$. Thus

$$f(A\mathbf{x} - \mathbf{b}) = 0 \Rightarrow A\mathbf{x} - \mathbf{b} = 0.$$

This is, the solution to transformed linear system produced by Gaussian elimination is also the solution to the original system.

Let $A \in \mathbb{R}^{n\times n}$ be invertible matrix and assume non-zero pivots. By (1.24) it holds that $E_{n-1} \ldots E_2 E_1 A = U$ where U is an upper triangular matrix. Inverting the product of elimination matrices yields the LU factorisation

$$A = LU \quad \text{for} \quad L = E_1^{-1} \cdots E_{n-1}^{-1}. \tag{1.25}$$

By Problem 21 on page 32, the matrix L is lower-triangular. Recall that entries of matrix L can be obtained directly from the row multipliers used in the elimination process. This fact is tricky to prove using index notation, hence, it is proven in Section 1.6 using block matrix notation.

Linear system

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

is reduced to two sub-problems using LU-factorisation of $A = LU$

$$L\mathbf{y} = \mathbf{b} \quad \text{and} \quad U\mathbf{x} = \mathbf{y}.$$

Both sub-problems have triangular coefficient matrices and can be efficiently solved using back-substitution, see Section 1.2.2.

Problems

P21. (2p) Show that the inverse of any $n \times n$ lower triangular matrix is lower triangular. Formulate an induction proof with respect to the dimension n and use Problem 3 on page 9

P22. (1p) Let $A \in \mathbb{R}^{n\times n}$ be invertible matrix. Show that on step $k \in \{2, \ldots, n\}$ of Gaussian elimination there exists a nozero pivot on column k. Hint: argue by contradiction and recall the block form of $A^{(k)}$ and use Problem 7 on page 10.

1.6 LU Factorization in block matrix notation

video on introduc- to recursive algo- n for computing the decomposition

In this section, we use block matrix notation to define a recursive process that returns the LU factorisation of a given invertible matrix. Recall that the elimination matrices related to the elimination process all have different structure, and hence, they cannot be easily treated using block matrix notation. This problem is remedied by recursive definition that allows us to formulate the elimination process using only the first elimination matrix. The given process could be easily turned into an existence proof of the LU - decomposition. It also shows that the matrix L can be constructed from multipliers related to row operations and there is no need to save or construct elimination matrices $E_1, \dots, E_{n-1}$ or their inverses during the elimination process. We do not assume non-zero pivots and use row pivoting. In this case, the LU factorisation of invertible matrix $A \in \mathbb{R}^{n\times n}$ is

See video on recursive algorithm for computing the LU decomposition

$$P^T A = LU \quad \text{where } P \text{ is a permutation matrix.}$$

Next, we give a recursive definition of $[P, L, U] = lu(A)$ that returns the LU factorisation of invertible matrix A.

For $n = 1$, $lu(A) = [1, 1, A]$.

For $n > 1$, we use recursive definition. First, we seek the permutation P such that $(P^T A)_{11} \neq 0$. Next, split $P^T A$ as

$$P^T A = \begin{bmatrix} a_{11} & \mathbf{a}_{12}^T \\ \mathbf{a}_{21} & A_{22} \end{bmatrix} \quad \text{where } a_{11} \in \mathbb{R},\ \mathbf{a}_{12}, \mathbf{a}_{21} \in \mathbb{R}^{(n-1)} \text{ and } A_{22} \in \mathbb{R}^{(n-1)\times(n-1)}.$$

The elimination matrix corresponding to first step of Gauss algorithm is

$$E = \begin{bmatrix} 1 & 0 \\ -\frac{\mathbf{a}_{21}}{a_{11}} & I \end{bmatrix} \quad \text{and} \quad EP^T A = \begin{bmatrix} a_{11} & \mathbf{a}_{12}^T \\ 0 & A_{22} - \frac{\mathbf{a}_{21}\mathbf{a}_{12}^T}{a_{11}} \end{bmatrix}.$$

Let $[P_2, L_2, U_2] = lu(A_{22} - \frac{\mathbf{a}_{21}\mathbf{a}_{12}^T}{a_{11}})$ so that $A_{22} - \frac{\mathbf{a}_{21}\mathbf{a}_{12}^T}{a_{11}} = P_2^{-T} L_2 U_2$ and

$$P^T A = \begin{bmatrix} 1 & 0 \\ \frac{\mathbf{a}_{21}}{a_{11}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & P_2^{-T} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & L_2 \end{bmatrix} \begin{bmatrix} a_{11} & \mathbf{a}_{12}^T \\ 0 & U_2 \end{bmatrix}.$$

By direct computation,

$$\begin{bmatrix} 1 & 0 \\ \frac{\mathbf{a}_{21}}{a_{11}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & P_2^{-T} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & P_2^{-T} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ P_2^T \frac{\mathbf{a}_{21}}{a_{11}} & I \end{bmatrix}.$$

Thus

$$\begin{bmatrix}1 & 0\\ 0 & P_2^T\end{bmatrix} P^T A = \begin{bmatrix}1 & 0\\ P_2^T \frac{\mathbf{a}_{21}}{a_{11}} & L_2\end{bmatrix}\begin{bmatrix}a_{11} & \mathbf{a}_{12}^T\\ 0 & U_2\end{bmatrix}.$$

And finally

$$lu(A) = \left[P\begin{bmatrix}1 & 0\\ 0 & P_2\end{bmatrix}, \begin{bmatrix}1 & 0\\ P_2^T \frac{\mathbf{a}_{21}}{a_{11}} & L_2\end{bmatrix}, \begin{bmatrix}a_{11} & \mathbf{a}_{12}^T\\ 0 & U_2\end{bmatrix}\right].$$

We deduce from the above algorithm that the Gaussian elimination with pivoting is Gaussian elimination applied matrix

$$P^T A,$$

where P collects all row permutations done during the process. Same holds for row- and full-pivoting. The matrix L is obtained by collecting the multipliers from step k as

$$L = \begin{bmatrix}1 & & & & \\ \alpha_{21} & 1 & & & \\ \alpha_{31} & \alpha_{32} & 1 & & \\ \vdots & \vdots & \dots & \ddots & \\ \alpha_{n1} & \alpha_{n2} & \dots & \alpha_{n(n-1)} & 1\end{bmatrix} \quad \text{where} \quad \alpha_{ij} = \frac{a_{ij}^{(j)}}{a_{jj}^{(j)}}.$$

Problems

P23. (0.5p) Write down the elimination matrices used in Example 1.6 and compute the corresponding LU-decomposition

P24. (0.5p) Write the LU decomposition corresponding to Example 1.7.

P25. (2p) Modify the definition of function lu to use column pivoting instead of row pivoting.

P26. (2p) Write a recursive implementation of the function $[P, L, U] = lu(A)$ in Matlab. Device a test verifying that your decomposition is correct.

P27. (2p) Modify the recursive implementation of function lu to utilise the update strategy.

1.7 Cholesky factorisation

This section gives existence proof for the Cholesky factorisation, which should be studied with care. The left-looking variant of the Cholesky decomposition is included because it yields a simpler formula for computing entries of the Cholesky factor and can be skipped.

Symmetric matrix $A \in \mathbb{R}^{n\times n}$, $A = A^T$ is also *positive definite* if there exists $\alpha > 0$ such that

$$\mathbf{x}^T A\mathbf{x} \geq \alpha \|\mathbf{x}\|_2^2 \quad \text{for any } \mathbf{x} \in \mathbb{R}^n. \tag{1.26}$$

In this section, we prove that every such matrix has a Cholesky decomposition:

Theorem 1.1. *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric and positive definite. Then there exists a lower triangular matrix $L \in \mathbb{R}^{n\times n}$ such that $A = LL^T$.*

The matrix L is called as the Cholesky factor of A. We prove Theorem 1.1 using induction with respect to the dimension of the matrix, block matrix notation, and the following technical result:

Lemma 1.3. *Let $F \in \mathbb{R}^{n\times m}$ have a trivial null-space and $A \in \mathbb{R}^{n\times n}$ be a symmetric and positive definite matrix. Then the $m \times m$ matrix $F^T AF$ is positive definite.*

See video proof of this lemma in Youtube

Note that by the rank-nullity Theorem it holds that $m \leq n$.

Proof. As A is s.p.d. there exists $\alpha > 0$ such that

$$\mathbf{x}^T F^T AF\mathbf{x} \geq \alpha \mathbf{x}^T F^T F\mathbf{x} \quad \text{for any } \mathbf{x} \in \mathbb{R}^n. \tag{1.27}$$

As $F^T F$ is symmetric, $F^T F = U\Lambda U^T$ where $U \in \mathbb{R}^{m\times m}$, $U = \begin{bmatrix}\mathbf{u}_1 & \cdots & \mathbf{u}_m\end{bmatrix}$ is unitary and $\Lambda \in \mathbb{R}^{m\times m}$, $\Lambda = diag(\lambda_1, \ldots, \lambda_m)$ is a diagonal matrix. Matrix $F^T F$ has the expansion

$$F^T F = \sum_{i=1}^{m} \lambda_i \mathbf{u}_i \mathbf{u}_i^T. \tag{1.28}$$

As eigenvectors $\{\mathbf{u}_i\}$ are orthonormal and $N(F) = \{0\}$, it follows that $\lambda_i = \mathbf{u}_i^T F^T F\mathbf{u}_i = \|F\mathbf{u}_i\|_2^2 > 0$. Using (1.28) and estimating λ_i from below by $\lambda_{min} := \min_{i\in\{1,\ldots,m\}} \lambda_i$ gives

$$\mathbf{x}^T F^T F\mathbf{x} = \sum_{i=1}^{m} \lambda_i (\mathbf{x}^T \mathbf{u}_i)^2 \geq \lambda_{min} \sum_{i=1}^{m} (\mathbf{x}^T \mathbf{u}_i)^2 = \lambda_{min} \mathbf{x}^T \mathbf{x}.$$

Noticing that $\lambda_{min} > 0$ and using (1.27) completes the proof. $\square$

proof of Theorem 1.1. The proof proceeds by induction with respect to the dimension n. **Base step n=1:** $A \in \mathbb{R}$, $A > 0$. Hence, $L = \sqrt{A}$.

Induction Assumption: The claim holds for $n = k$

Induction step: Let $A \in \mathbb{R}^{(k+1)\times(k+1)}$ and split

$$A = \begin{bmatrix} a_{11} & \mathbf{a}_{21}^T \\ \mathbf{a}_{21} & A_{22}. \end{bmatrix}$$

where $a_{11} \in \mathbb{R}$, $\mathbf{a}_{21} \in \mathbb{R}^k$ and $A_{22} \in \mathbb{R}^{k\times k}$. Let

$$E = \begin{bmatrix} 1 & 0 \\ -a_{11}^{-1}\mathbf{a}_{21} & I \end{bmatrix}.$$

By direct calculation

$$EAE^T = \begin{bmatrix} a_{11} & 0 \\ 0 & A_{22} - \mathbf{a}_{21}a_{11}^{-1}\mathbf{a}_{21}^T \end{bmatrix}.$$

See video on existence proof of Cholesky factorisation in Youtube

Before applying the induction assumption to the matrix $A_{22} - \mathbf{a}_{21}a_{11}^{-1}\mathbf{a}_{21}^T$, we have to show that it is positive definite. Observe that

$$\left(A_{22} - \mathbf{a}_{21}a_{11}^{-1}\mathbf{a}_{21}^T\right) = \begin{bmatrix} \underset{k\times 1}{0} & \underset{k\times k}{I} \end{bmatrix} EAE^T \begin{bmatrix} \underset{k\times 1}{0} & \underset{k\times k}{I} \end{bmatrix}^T \tag{1.29}$$

As both $\begin{bmatrix} 0 & I \end{bmatrix}^T$ and E have trivial null-spaces, so does $F = E^T \begin{bmatrix} 0 & I \end{bmatrix}^T$. Hence by (1.29) and Lemma 1.3, $A_{22} - \mathbf{a}_{21}a_{11}^{-1}\mathbf{a}_{21}^T$ is positive definite. Applying the induction assumption gives $A_{22} - \mathbf{a}_{21}a_{11}^{-1}\mathbf{a}_{21}^T = L_2L_2^T$, where $L_2 \in \mathbb{R}^{k\times k}$ is a lower triangular matrix. Note that $a_{11} = \mathbf{e}_1^T A\mathbf{e}_1 > 0$. Hence,

$$EAE^T = \begin{bmatrix} a_{11} & 0 \\ 0 & A_{22} - \mathbf{a}_{21}a_{11}^{-1}\mathbf{a}_{21}^T \end{bmatrix} = \begin{bmatrix} \sqrt{a_{11}} & 0 \\ 0 & L_2 \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & 0 \\ 0 & L_2^T \end{bmatrix}.$$

Inverting E gives

$$L = \begin{bmatrix} 1 & 0 \\ a_{11}^{-1}\mathbf{a}_{21} & I \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & 0 \\ 0 & L_2 \end{bmatrix} = \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{\mathbf{a}_{21}}{\sqrt{a_{11}}} & L_2 \end{bmatrix} \tag{1.30}$$

□

The above proof is constructive, this is, it also gives a method for computing L.

The function $L = rchol(A)$ returns the Cholesky factorisation of a s.p.d. matrix $A \in \mathbb{R}^{n \times n}$.

For $n = 1$, $rchol(A) = \sqrt{A}$.
For $n > 1$, we use recursive definition. Split A as

$$A = \begin{bmatrix} a_{11} & \mathbf{a}_{21}^T \\ \mathbf{a}_{21} & A_{22} \end{bmatrix} \quad \text{and let } L_2 = rchol(A_{22} - \tfrac{\mathbf{a}_{21}\mathbf{a}_{21}^T}{a_{11}}).$$

By (1.30) we have

$$rchol(A) = \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{\mathbf{a}_{21}}{\sqrt{a_{11}}} & L_2 \end{bmatrix}$$

Similar to functions *triusolve* and *lu*, function *rchol* can be implemented using recursive function calls or using the update strategy. The implementation utilising update strategy is called as the down-looking Cholesky factorisation because the lower right corner is updated on each step of the algorithm.

There exist (at least) two other strategies for computing the Cholesky factorisation. The difference between these variants is the order in which the matrix elements are accessed. One has to choose the best strategy for each sparse matrix storage format and computer architecture. For example, the down-looking variant accesses data column wise and works well with compressed column storage format.

To derive the left-looking Cholesky factorisation, we split

$$L = \begin{bmatrix} \# & & & & \\ \mathbf{l}_i^T & l_{ii} & & & \\ \# & \# & \# & & \\ \mathbf{l}_j^T & l_{ji} & \# & \# & \\ \# & \# & \# & \# & \# \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} \# & & & & sym. \\ \mathbf{a}_i^T & a_{ii} & & & \\ \# & \# & \# & & \\ \mathbf{a}_j^T & a_{ji} & \# & \# & \\ \# & \# & \# & \# & \# \end{bmatrix}.$$

where indices i and j refer to rows i and j of matrices L and A. Computing the matrix product LL^T gives

$$\begin{bmatrix} \# & & & & \\ \mathbf{l}_i^T & l_{ii} & & & \\ \# & \# & \# & & \\ \mathbf{l}_j^T & l_{ji} & \# & \# & \\ \# & \# & \# & \# & \# \end{bmatrix} \begin{bmatrix} \# & \mathbf{l}_i & \# & \mathbf{l}_j & \# \\ & l_{ii} & \# & l_{ji} & \# \\ & & \# & \# & \# \\ & & & \# & \# \\ & & & & \# \end{bmatrix} = \begin{bmatrix} \# & & & & sym. \\ \# & \mathbf{l}_i^T\mathbf{l}_i + l_{ii}^2 & & & \\ \# & \# & \# & & \\ \# & \mathbf{l}_j^T\mathbf{l}_i + l_{ji}l_{ii} & \# & \# & \\ \# & \# & \# & \# & \# \end{bmatrix}.$$

Using the relation $A = LL^T$ yields

$$\mathbf{l}_i^T\mathbf{l}_i + l_{ii}^2 = a_{ii} \quad \text{and} \quad \mathbf{l}_j^T\mathbf{l}_i + l_{ji}l_{ii} = a_{ji}. \tag{1.31}$$

Note that the entry l_{ii} is not uniquely defined by (1.31). The usual choice, $l_{ii} \in \mathbb{R}, l_{ii} > 0$, gives the formulas

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \quad \text{and} \quad l_{ji} = \frac{1}{l_{ii}}(a_{ji} - \sum_{k=1}^{i-1} l_{ik}l_{jk}) \quad \text{for } j > i \tag{1.32}$$

We use (1.32) in Section 1.8.2 to the study location of non-zero entries of L.

1.7.1 Additional material

A different inductive existence proof for the Cholesky factorisation is outlined in blog posting What Is Choklesky Factorisation.

A survey on Cholesky factorisation aimed for computer scientist is given in

Nicholas J. Higham. Cholesky factorization. *WIREs Computational Statistics*, 1(2):251–254, 2009

1.7.2 Problems

P28. (1p) Let $A \in \mathbb{R}^{n\times n}$ be s.p.d.

(a) Starting from the definition (1.26), show that $a_{ii} > 0$ and A is invertible. Hint : Show that system $Ax = 0$ has only zero solution, i.e., $N(A) = \{0\}$.

(b) Show that all eigenvalues of A are positive.

(c) Assume, that A also satisfies $A = F^TF$ for some $F \in \mathbb{R}^{n\times n}$. Show that F is invertible.

P29. (2p)

(a) Compute by hand the Cholesky decomposition of $\begin{bmatrix} 1 & 2 & 2 \\ 2 & 8 & 4 \\ 2 & 4 & 15 \end{bmatrix}$.

(b) Show that the matrix $\begin{bmatrix} 15 & 2 & 4 \\ 2 & 1 & 2 \\ 4 & 2 & 8 \end{bmatrix}$ is positive definite.

P30. (2p) Write a recursive implementation of function *rchol*.

P31. (2p) Modify your recursive implementation of *rchol* to use the update strategy.

P32. (1p) Let $F \in \mathbb{R}^{n\times n}$ and $A = F^TF$.

(a) Show that $\|A\|_2 = \|F\|_2^2$. Hint: Use the definition of operator norm to obtain the estimates $\|A\|_2 \le \|F\|_2^2$ and $\|F\|_2 \le \|A\|_2^{1/2}$.

(b) Validate (a) by numerical examples.

P33. (2p) Let $A_N \in \mathbb{R}^{N\times N}$ be the $1D$-finite difference matrix

$$A_N = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Define matrices $A_n \in \mathbb{R}^n$ for $n \in \{N-1, \dots, 1\}$ as follows. Split $A_n \in \mathbb{R}^{n\times n}$ for $n \in \{N, \dots, 2\}$ as

$$A_n = \begin{bmatrix} \underset{1\times 1}{\alpha_n} & \mathbf{a}_n^T \\ \underset{(n-1)\times 1}{\mathbf{a}_n} & \underset{(n-1)\times(n-1)}{\widehat{A}_n} \end{bmatrix}$$

and set $A_{n-1} = \widehat{A}_n - \frac{\mathbf{a}_n\mathbf{a}_n^T}{\alpha_n}$.

(a) Compute the block matrix product to verify that A_n can be factorised as

$$A_n = \begin{bmatrix} \sqrt{\alpha_n} & 0 \\ \frac{\mathbf{a}_n}{\sqrt{\alpha_n}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A_{n-1} \end{bmatrix} \begin{bmatrix} \sqrt{\alpha_n} & \frac{\mathbf{a}_n^T}{\sqrt{\alpha_n}} \\ 0 & I \end{bmatrix}$$

(b) Use induction to show that

$$A_n = \begin{bmatrix} 1 + \frac{1}{(N+1-n)} & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \quad \text{for } n \in \{N, \dots, 1\}$$

(c) Give a formula for the Cholesky factor of A_N.

1.8 Sparse Cholesky Factorisation

This section demonstrates that the Cholesky factor of a sparse matrix can be dense and that in some cases sparse factor can be obtained by a suitable symmetric permutation. Core content.

See video on sparse Cholesky factorisation in Youtube

Let L be a Choklesky factor of a sparse s.p.d. matrix A. In this Section, we are study the location of the non-zero entries of L. Observe that L is an invertible lower triangular matrix, and thus $l_{ii} \neq 0$.

Entries l_{ij} of L satisfying

$$l_{ij} \neq 0 \quad \text{and} \quad a_{ij} = 0$$

are called *fill-in*. Fill-in increases the amount of memory required to store L as well as the time required to compute it's entries. To save computational resources, fill-in is reduced by permuting rows and columns of matrix A before computing it's the Cholesky factorisation. We call the resulting factorisation

$$P^T AP = LL^T$$

where $P \in \mathbb{R}^{n\times n}$ is a fill-in minimising permutation and $L \in \mathbb{R}^{n\times n}$ a lower triangular matrix as the *sparse Cholesky factorisation.*

Example 1.8. *Consider*

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 10 & 0 & 0 & 0 \\ 1 & 0 & 10 & 0 & 0 \\ 1 & 0 & 0 & 10 & 0 \\ 1 & 0 & 0 & 0 & 10 \end{bmatrix}.$$

The Cholesky factor of A is

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 \\ 1 & -0.33333 & 2.9814 & 0 & 0 \\ 1 & -0.33333 & -0.37268 & 2.958 & 0 \\ 1 & -0.33333 & -0.37268 & -0.42258 & 2.9277 \end{bmatrix}$$

Observe, that L is a full matrix. The fill-in is reduced by permuting the entries of A. In our example, changing row 1 to row 5 and column 1 to *column 5 gives*

$$P^TAP = \begin{bmatrix} 10 & 0 & 0 & 0 & 1 \\ 0 & 10 & 0 & 0 & 1 \\ 0 & 0 & 10 & 0 & 1 \\ 0 & 0 & 0 & 10 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{1.33}$$

where P is the permutation matrix corresponding to permutation vector $\begin{bmatrix} 5 & 2 & 3 & 4 & 1 \end{bmatrix}$. The Cholesky factor of P^TAP is

$$\tilde{L} = \begin{bmatrix} 3.1623 & 0 & 0 & 0 & 0 \\ 0 & 3.1623 & 0 & 0 & 0 \\ 0 & 0 & 3.1623 & 0 & 0 \\ 0 & 0 & 0 & 3.1623 & 0 \\ 0.31623 & 0.31623 & 0.31623 & 0.31623 & 3.0984 \end{bmatrix}.$$

The factor $\tilde{L}$ does not have any fill-in.

Finding an optimal permutation that minimizes the fill-in is an NP-hard problem, hence, heuristics are used instead. In Section 1.8.2, we discuss minimal degree-ordering, which is a method for finding fill-in reducing permutations by utilising an efficient method for determining the location of non-zero entries of L.

1.8.1 Problems

P34. (2p) Let

$$A = \begin{bmatrix} a_{11} & \mathbf{a}_{21}^T \\ \mathbf{a}_{21} & I \end{bmatrix} \quad \text{for} \quad a_{11} \in \mathbb{R}, \mathbf{a}_{21} \in \mathbb{R}^{n-1}.$$

(a) Show that the matrix A is positive definite if $a_{11} > \|\mathbf{a}_{21}\|_2^2$. Hint: use the definition (1.26) with suitable splitting of $\mathbf{x}$.

(b) Consider the linear system $A\mathbf{x} = \mathbf{e}_1$. Decompose $\mathbf{x} = \begin{bmatrix} x_1 & \mathbf{x}_2^T \end{bmatrix}^T$, where $x_1 \in \mathbb{R}$ and $\mathbf{x}_2 \in \mathbb{R}^{n-1}$. Show that the solution satisfies

$$(a_{11} - \mathbf{a}_{21}^T\mathbf{a}_{21})x_1 = 1 \quad \text{and} \quad \mathbf{x}_2 = -\mathbf{a}_{21}x_1.$$

1.8.2 Non-zero structure of the Cholesky factor

This section gives tools for computing non-zero entres of L without knowing their exact values. These tools are then used to construct fill-in reducing *permutations. Core content.*

See video on graph associated to matrix in Youtube

The Cholesky factorisation of a sparse matrix is computed in two steps: First, *symbolic factorisation* step constructs a fill-in reducing permutation and finds the location of non-zero entries of the Cholesky factor. The location of nonzero entries is used to set up sparse matrix data structure for storing L. The entries of the Cholesky factor are then computed in the *numerical factorization* step.

The location of non-zero entries in the Cholesky factor of $A \in \mathbb{R}^{n\times n}$ is predicted from the undirected graph $\mathcal{G}(A) = (\mathcal{V}(A), \mathcal{E}(A))$ consisting of a set of vertices $\mathcal{V}(A) = \{1,\ldots,n\}$ and a set of edges

$$\mathcal{E}(A) = \{\ (i,j) \mid a_{ij} \neq 0 \quad i,j = 1,\ldots,n \quad \text{and} \quad i > j\ \}.$$

This is, vertices i and j of the graph $\mathcal{G}(A)$ are connected by an edge if the entry a_{ij} is nonzero.

Example 1.9. *Let*

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 10 & 0 & 0 & 0 \\ 1 & 0 & 10 & 0 & 0 \\ 1 & 0 & 0 & 10 & 0 \\ 1 & 0 & 0 & 0 & 10 \end{bmatrix} \tag{1.34}$$

and

$$A_2 = \begin{bmatrix} 20 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 20 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 20 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 20 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 20 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 20 \end{bmatrix} \tag{1.35}$$

The graphs corresponding to matrices A_1 and A_2 are visualized in Fig. 1.5

Off-diagonal entries of the Cholesky factor L are computed using Eq. (1.32) as

$$l_{ij} = \frac{1}{l_{jj}}(a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}), \quad \text{when } i > j. \tag{1.36}$$

Thus the entry l_{ij} can be non-zero (Possible numerical cancellations are neglected in the following) if

$$a_{ij} \neq 0 \tag{1.37}$$

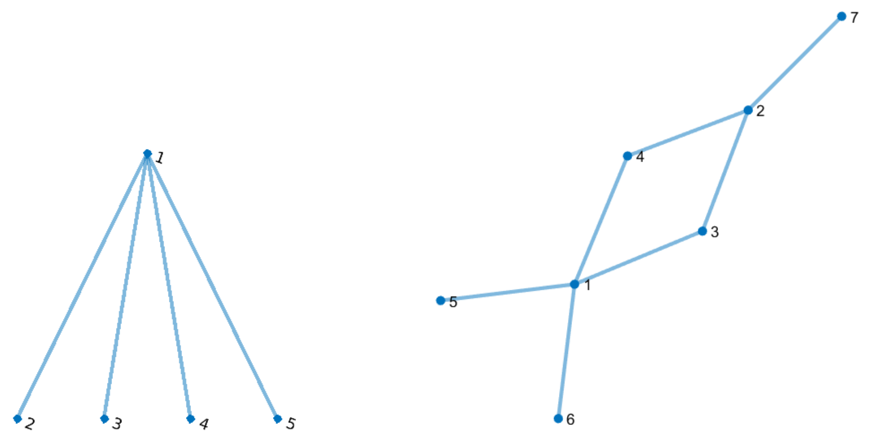

Figure 1.5: Graphs corresponding to the matrices given in (1.34) and (1.35), respectively.

or

$$l_{jk} \neq 0 \quad \text{and} \quad l_{ik} \neq 0 \quad \text{for some } k < j. \tag{1.38}$$

Based on equation (1.37), the number of nonzeros in L will always be greater or equal to the number of nonzeros in A.

See video on graph notation in Youtube

Before proceeding, we need some notation. We call the ordered set of vertices $(v_1, v_2, \ldots, v_k) \subset \mathcal{V}(A)$ as a *path*, if $(v_i, v_{i+1}) \in \mathcal{E}(A)$ for $i \in \{1, \ldots, k-1\}$. Vertex $x \in \mathcal{V}(A)$ is said to be reachable from vertex $y \in \mathcal{V}(A)$ via set $S \subset \mathcal{V}(A)$, if there exists a path $(y, v_1, \ldots, v_k, x)$ satisfying[4] $v_i \in S$ for $i \in \{1, \ldots, k\}$. The reachable set of $y \in \mathcal{V}(A)$ through $S \subset \mathcal{E}(A)$ is defined as

$$Reach(y, S) = \{x \in \mathcal{V}(A) \setminus S \mid x \text{ is reachable from } y \text{ via } S\ \}. \tag{1.39}$$

Examples of path and reachable set are depicted in Figure 1.6.

See video proof of the following Theorem in Youtube.

The edges of $\mathcal{G}(L + L^T)$ corresponding to non-zero off-diagonal entries of L are characterized by the following Theorem.

[4]to make the presentation simpler, we abuse notation and use the same notation also for paths (y, x) and (x, v_1, y).

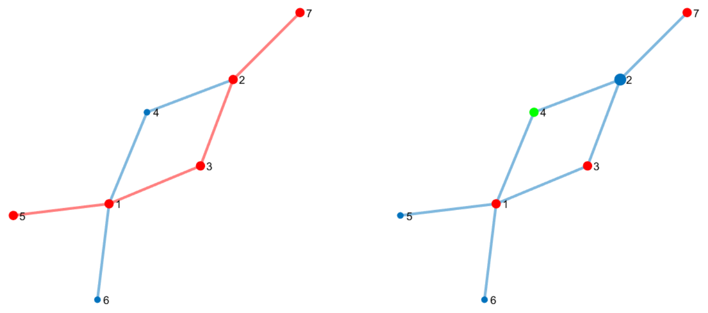

Figure 1.6: Path $(5, 1, 3, 2, 7)$ is marked in red. Reachable set of vertex 2 via $S = \{4\}$ is $\{1, 3, 7\}$.

Theorem 1.2. *Let $A \in \mathbb{R}^{n \times n}$ be a s.p.d. and L the Cholesky factor of A. Then*

$$\mathcal{E}(L + L^T) \subset \{ \ (i, j) \mid i \in Reach(j, \{1, \dots j-1\}) \ \}$$

Recall that diagonal entries of L are always nonzero as L is an invertible lower triangular matrix. These entries are not edges of $\mathcal{G}(L + L^T)$.

Proof. Let $i > j$ and $(i, j) \in \mathcal{E}(L + L^T)$. Then $l_{ij} \neq 0$. We proceed by induction with respect to j.

Base case: $j = 1$ If $j = 1$, l_{i1} is nonzero iff $a_{i1} \neq 0$.

Induction assumption: Assume that the claim hods for any $j < t$.

Induction step: Let $j = t$. Then $l_{ij} \neq 0$ if $a_{ij} \neq 0$ or there exists index $k < j$ such that $l_{ik} \neq 0$ and $l_{jk} \neq 0$. By induction assumption, there then exists paths $(k, v_1, \dots, v_l, i)$ and $(k, \hat{v}_1, \dots, \hat{v}_m, j)$ satisfying $v_q < k$ for $q \in \{1, \dots, l\}$ and $\hat{v}_{\hat{q}} < k$ for $\hat{q} \in \{1, \dots, m\}$. As paths can be "walked" in both directions, there also exists path $(i, v_l, \dots v_1, k)$. Thus, $(i, v_l, \dots v_1, k, \hat{v}_1, \dots, \hat{v}_m, j)$ is a path between vertices i and j between nodes via vertices with index smaller than t. □

A set including edges $\mathcal{E}(L+L^T)$ is computed by finding the reachable set for ever node of $\mathcal{V}(A)$. Such computation can be implemented as a depth-fist search (DFS). A naive example implementation is given below.

See Wikipedia for more information on DFS

```
%  Call as : R = my_reach(A, v, S)
%
%  A is a matrix, v is the current node, S is a vector of nodes.
%
function [R,visited] = my_reach(A, v, S, R, visited)

    if(nargin == 3)
        R = [];
        visited(1:size(A,2)) = false;
    end

    visited(v) = true;

    edges = find( abs( A(:,v)) > 0);

    if( isempty(S))
        R = setdiff(edges,v);
        return;
    end

    for w=edges(:)'
        if( not(visited(w)))
            if( not(ismember(S,w)) )
                R = [R w];
            else
                [R,visited] = my_reach(A,w,S,R,visited);
            end
        end
    end
end
```

In the worst case, the cost of computing single reachable set using DFS algorithm is $O(|N(A)| + |E(A)|)$. Due to this potentially high cost, more efficient methods have been developed for computing the location of non-zero entries of L.

Example 1.10. *Consider the matrix A_2 in (1.35). The off-diagonal non-zero entries of the Cholesky factor are obtained as*

See video on this example on Youtube

- *off-diagonal non-zeros on column 1 are reach*$(1, \emptyset) = \{3, 4, 5, 6\}$.
- *off-diagonal non-zeros on column 2 are reach*$(2, \{1\}) = \{3, 4, 7\}$
- *off-diagonal non-zeros on column 3 are reach*$(3, \{1, 2\} = \{4, 5, 6, 7\}$
- *off-diagonal non-zeros on column 4 are reach*$(4, \{1, 2, 3\}) = \{5, 6, 7\}$
- *off-diagonal non-zeros on column 5 are reach*$(5, \{1, 2, 3, 4\}) = \{6, 7\}$

- *off-diagonal non-zeros on column* 6 *are* $reach(6, \{1, 2, 3, 4, 5\}) = \{7\}$

The non-zeros of the computed factor are

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 & 0 \\ \times & 0 & \times & \times & \times & 0 & 0 \\ \times & 0 & \times & \times & \times & \times & 0 \\ 0 & \times & \times & \times & \times & \times & \times \end{bmatrix}.$$

1.8.3 Problems

P35. (2p) Consider the matrix $A \in \mathbb{R}^{5\times 5}$ such that

```
A = zeros(5);
A(1,2) = 1; A(2,3) = 1;
A(2,5) = 1; A(3,4) = 1;
A = 100*eye(5) + A + A';
```

(a) Draw the graph $\mathcal{G}(A)$

(b) For each vertex $i \in \mathcal{V}(A)$ compute the set $reach(i, \{1, \ldots, i-1\})$. Use `my_reach.m` to validate your answer.

(c) Predict the location of non-zero entries in the Cholesky factor of A.

(d) Compute the Cholesky factorization of A and validate (c)

P36. (1p) Let s.p.d. $A \in \mathbb{R}^{n\times n}$ be a banded matrix with bandwidth $b \in \mathbb{N}$. This is,

$$a_{ij} = 0 \quad \text{if} \quad i > j + b \quad \text{or} \quad i < j - b.$$

(a) Let $n = 10$ and $b = 2$. Draw the dependency graph $\mathcal{G}(A)$.

(b) Use $\mathcal{G}(A)$ to predict the location of nonzero entries of the corresponding Cholesky factor L.

1.8.4 Minimum degree ordering

This section outlines how minimum degree ordering is used to construct a fill-in reducing permutation. Core content.

Minimum degree (MD) ordering is a widely used heuristic for finding a fill-in reducing permutation for the matrix A. The MD method constructs a permutation vector $\mathbf{p} \in \mathbb{R}^n$ by choosing entry p_i from the set of free indices $\{1, \dots, n\} \setminus \{p_1, \dots, p_{i-1}\}$ so that the number of non-zero entries that appear in the ith column of L is minimised. The number of non-zero entries on column i does not depend on entries $\{p_{i+1}, \dots, p_n\}$ and can be computed using the `my_reach.m` function. A naive implementation is given below.

See video on MD on Youtube

```
% Construct a fill-in reducing permutation vector for
% A using minimum degree ordering method. (this is a naive example
% implementation)

function p = my_md(A)
n = size(A,1);
p = 1:n;

for i=1:(n-1)
   i
    % try all remaining entries as entry i
    nnzLi = zeros(1,n);
    for j=(i+1):n

        tmp = p; tmp(i) = p(j); tmp(j) = p(i);

        nnzLi(j) = length(unique(my_reach(A(tmp,tmp), i, [1:(i-1)])));
    end
    % choose permutation minimising nnz in column i.
    [~,I] = min(nnzLi((i+1):n));
    I = I(1)+i;
    pi = p(i); p(i) = p(I(1)); p(I) = pi;
end
```

Example 1.11. *Consider the matrix*

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 10 & 0 & 0 \\ 1 & 0 & 10 & 0 \\ 1 & 0 & 0 & 10 \end{bmatrix}.$$

Initially, $p = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$. *In the first step of MD-algorithm, we test permutations*

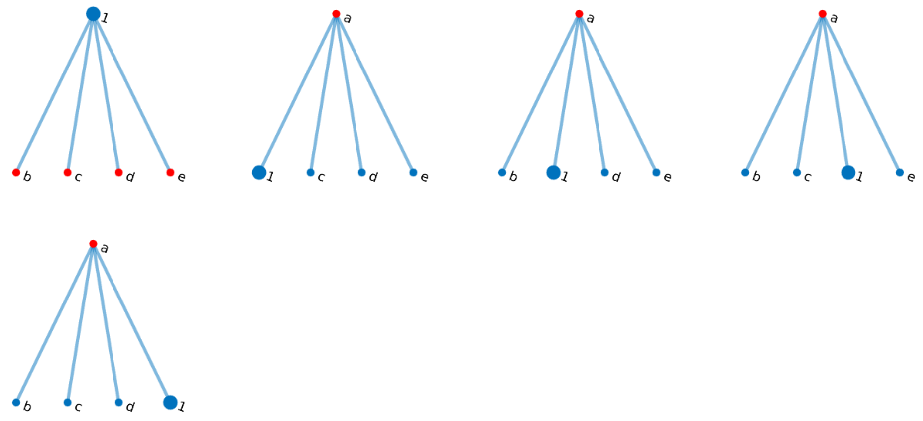

Figure 1.7: The first step of the MD - algorithm

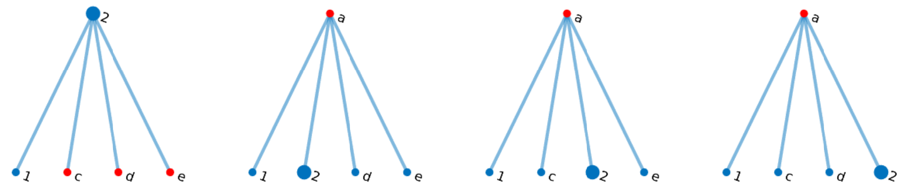

Figure 1.8: The second step of the MD - algorithm

$$\begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix}, \begin{bmatrix}2\\1\\3\\4\\5\end{bmatrix}, \begin{bmatrix}3\\2\\1\\4\\5\end{bmatrix}, \begin{bmatrix}4\\2\\3\\1\\5\end{bmatrix}, \quad \textit{and} \quad \begin{bmatrix}5\\2\\3\\4\\1\end{bmatrix}.$$

The resulting number of non-zeros in $L(2 : end, 1)$ is computed using function `my_reach` *operator, see Fig. 1.7. In Fig. 1.7 and 1.8, letters $\{a, b, c, d, e\}$ refer to entries $\{1, 2, 3, 4, 5\}$ of the original matrix that after permutation have indices larger than 1 and 2, respectively. The alternative choices give $5, 2, 2, 2, 2$ - nonzero entries in the first column. According to this, $p_1 = 2$. The process is then repeated for p_2 see Fig. 1.8. Different options give $4, 2, 2, 2$ - nonzero entries in $L(3 : end, 2)$. Accordingly, we set $p_2 = 3$.*

Computing the number of non-zero entries in the column i, i.e. evaluation of the `my_reach` is the most expensive part of the MD method. This cost is reduced in the approximate minimum degree (AMD) algorithm that approximates the number of non-zeros in the column i. As AMD is much faster and yields almost as good orderings as MD, latest versions of Matlab only implement it.

1.8.5 Problems

P37. (2p)

(a) Find a fill-in reducing permutation P for the matrix A_2 in (1.35) using function `my_md`.

(b) Compute the number of non-zeros in the Cholesky factors of A_2 and $P^T A_2 P$.

(c) Repeat (a) and (b) using permutation generated by Matlab function `amd`.

Bibliography

[1] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in matlab: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13(1):333–356, 1992.

[2] Nicholas J. Higham. Cholesky factorization. *WIREs Computational Statistics*, 1(2):251–254, 2009.