4 Solving Systems of Linear Equations

We are interested in numerically solving equations of the form

$$Ax = b, \tag{1}$$

where A is a (square) $n \times n$ matrix, b is an n-(column) vector and x is a (column) vector containing the unknowns $x_1, \ldots, x_n$.

In general, there are two classes of methods:

- **Direct methods**, such as Gaussian elimination, LU factorization, Cholesky factorization, Doolittle factorization or Crout factorization. These methods *theoretically* lead to an "exact" solution of the problem (1) in finitely many steps.
- **Iterative methods**, such as Jacobi iteration, Gauss-Seidel iteration, SOR, SSOR, CG. These methods provide an approximate solution to (1).

Another distinction is usually made between *full* (or dense) matrices and *sparse* matrices, i.e., such matrices whose majority of entries are zero. Iterative solvers are especially well-suited for sparse matrix problems.

4.1 Matrix Algebra

We summarize some of the basic ingredients used in the discussion later on. More details can be found in the textbook.

Elementary Row Operations:

These operations do not change the solution of a linear system.

1. $E_i \leftrightarrow E_j$: swap equation i with equation j.
2. $\lambda E_i \to E_i$, $\lambda \neq 0$: replace equation i by a nonzero multiple of itself.
3. $E_i + \lambda E_j \to E_i$: add a multiple of equation j to equation i and replace equation i by the result.

We can represent these basic operations by left-multiplication by so-called *elementary matrices.*

1. Permutation matrix

   Ex.:
$$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \text{swaps rows 2 and 3}$$

2. Ex.:
$$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{scales row 2 by } \lambda$$

3. Ex.:

$$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \lambda & 1 \end{bmatrix} \qquad \text{adds } \lambda \text{ times row 2 to 3}$$

One possible application of elementary matrices is the reduction of an invertible matrix A to the identity matrix, i.e.,

$$\underbrace{E_m E_{m-1} \cdots E_2 E_1}_{=A^{-1}} A = I.$$

This, in fact, provides an "algorithm" for computing the inverse A^{-1} of A:

> Use elementary row operations to reduce A to I, and simultaneously apply the same operations to I, resulting in A^{-1}.

Ex.: Consider

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 5 \\ 4 & 6 & 8 \end{bmatrix}, \qquad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Then

$$E_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}: \quad E_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 3 \\ 4 & 6 & 8 \end{bmatrix}, \quad E_1 I = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix}: \quad E_2 E_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 3 \\ 0 & 2 & 4 \end{bmatrix}, \quad E_2 E_1 I = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix},$$

$$E_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}: \quad E_3 E_2 E_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 4 \\ 0 & 0 & 3 \end{bmatrix}, \quad E_3 E_2 E_1 I = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 0 & 1 \\ -2 & 1 & 0 \end{bmatrix},$$

$$E_4 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/3 \end{bmatrix}: \quad E_4 E_3 E_2 E_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix}, \quad E_4 E_3 E_2 E_1 I = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 0 & 1 \\ -2/3 & 1/3 & 0 \end{bmatrix},$$

$$E_5 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}: \quad E_5 \cdots E_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad E_5 \cdots E_1 I = \begin{bmatrix} 1 & 0 & 0 \\ -4/3 & -4/3 & 1 \\ -2/3 & 1/3 & 0 \end{bmatrix},$$

$$E_6 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}: \quad E_6 \cdots E_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad E_6 \cdots E_1 I = \begin{bmatrix} 1 & 0 & 0 \\ -2/3 & -2/3 & 1/2 \\ -2/3 & 1/3 & 0 \end{bmatrix},$$

$$E_7 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}: \quad E_7 \cdots E_1 A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad E_7 \cdots E_1 I = \begin{bmatrix} 5/3 & -1/3 & 0 \\ -2/3 & -2/3 & 1/2 \\ -2/3 & 1/3 & 0 \end{bmatrix},$$

$$E_8 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} : \qquad E_8 \cdots E_1 A = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{=I}, \qquad E_8 \cdots E_1 I = \underbrace{\begin{bmatrix} 7/3 & 1/3 & -1/2 \\ -2/3 & -2/3 & 1/2 \\ -2/3 & 1/3 & 0 \end{bmatrix}}_{=A^{-1}}.$$

Remark: This corresponds to a matrix representation of basic Gauss-Jordan elimination.

Theorem 4.1 *For an $n \times n$ matrix A the following are equivalent:*

1. *A^{-1} exists, i.e., A is nonsingular.*
2. $\det A = |A| \neq 0$.
3. $\text{rowspace}(A) = \mathbb{R}^n$, *i.e., the rows of A form a basis for $\mathbb{R}^n$.*
4. $\text{colspace}(A) = \mathbb{R}^n$, *i.e., the columns of A form a basis for $\mathbb{R}^n$.*
5. *$A : \mathbb{R}^n \to \mathbb{R}^n$ is injective (one-to-one), i.e., for each $b \in \mathbb{R}^n$ there is a unique $x \in \mathbb{R}^n$ such that $b = Ax$.*
6. *$A : \mathbb{R}^n \to \mathbb{R}^n$ is surjective (onto), i.e., every $x \in \mathbb{R}^n$ has an image $b = Ax \in \mathbb{R}^n$.*
7. *$Ax = 0$ has only the trivial solution $x = 0$.*
8. *For any $b \in \mathbb{R}^n$, $Ax = b$ has a unique solution.*
9. *$A = E_m E_{m-1} \cdots E_1$, i.e., A can be factored into elementary matrices.*
10. *0 is not an eigenvalue of A.*

Proof: Can be found in any standard linear algebra textbook. ♠

Definition 4.2 *An $n \times n$ matrix A is called* positive definite *if the quadratic form $x^T A x > 0$ for all nonzero $x \in \mathbb{R}^n$.*

Ex.: Consider
$$A = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

Its quadratic form is

$$\begin{aligned} [x_1 \ x_2] \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= [x_1 \ x_2] \begin{bmatrix} 2x_1 + x_2 \\ x_1 + x_2 \end{bmatrix} \\ &= 2x_1^2 + x_1x_2 + x_1x_2 + x_2^2 \\ &= (x_1 + x_2)^2 + x_1^2 > 0, \end{aligned}$$

provided $(x_1, x_2) \neq (0, 0)$.

Remarks:

1. In some books the definition of positive definiteness includes a symmetry requirement on A. However, there are matrices that are positive definite in our sense (and not symmetric).

   Ex.:
   $$A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}.$$

   Its quadratic form is
   $$\begin{aligned} [x_1\ x_2] \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= [x_1\ x_2] \begin{bmatrix} x_1 + 2x_2 \\ -2x_1 + x_2 \end{bmatrix} \\ &= x_1^2 + 2x_1x_2 - 2x_1x_2 + x_2^2 \\ &= x_1^2 + x_2^2 > 0, \end{aligned}$$

   if $(x_1, x_2) \neq (0, 0)$.

2. If A is symmetric and positive definite then all its eigenvalues are positive and real. This is not so for non-symmetric matrices.

   Ex.:
   $$A = \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} \quad \text{has eigenvalues } \lambda_1 = \lambda_2 = 1 > 0,$$

   and its quadratic form is
   $$x^T A x = x_1^2 + 4x_1x_2 + x_2^2 = (x_1 + x_2)^2 + 2x_1x_2,$$

   which is *indefinite*.

3. We will require symmetric and positive definite matrices for several methods below to apply.

4. Symmetric positive matrices arise frequently in practice (e.g., in methods based on energy principles, or in finite difference or finite element methods).

5. Symmetric positive definite matrices have "weighty" diagonals. To see this, consider
   $$A = \begin{bmatrix} a & b \\ b & c \end{bmatrix},$$

   with A positive definite, i.e., $x^T A x > 0$ for all $x \neq 0$. Let

   $$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}: \qquad x^T A x = a > 0,$$

   $$x = \begin{bmatrix} 0 \\ 1 \end{bmatrix}: \qquad x^T A x = d > 0,$$

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}: \qquad x^T A x = a + 2b + d > 0 \Longrightarrow -b < \frac{a+d}{2},$$

$$x = \begin{bmatrix} 1 \\ -1 \end{bmatrix}: \qquad x^T A x = a - 2b + d > 0 \Longrightarrow b < \frac{a+d}{2}.$$

Thus, $|b| < \frac{a+d}{2}$, and A has large positive entries on its diagonal. This is also true for $n \times n$ matrices.

6. Any non-symmetric positive definite matrix A can be decomposed into the sum of a skew-symmetric and a symmetric (positive definite) matrix:

$$A = \underbrace{\frac{1}{2}\left(A + A^T\right)}_{\text{symm. pos. def.}} + \underbrace{\frac{1}{2}\left(A - A^T\right)}_{\text{skew-symm.}} = S + T.$$

In general, $x^T A x = x^T S x$.

Partitioned Matrices

If $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and $B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \end{bmatrix}$ with matrix blocks A_{ij} and B_{ij} with compatible block sizes, then

$$AB = C = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \end{bmatrix},$$

where $C_{ij} = \sum_{k=1}^{2} A_{ik} B_{kj}$.

Remarks:

1. This result holds for larger block matrices also.
2. We will make use of partitioned matrices when dealing with some of the algorithms later.

4.2 LU and Cholesky Factorizations

We are interested in solving an $n \times n$ system $Ax = b$. Before we consider the full problem, we look at some special systems that are particularly easy to solve.

1. Diagonal systems:

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & a_{nn} \end{bmatrix}.$$

In this case $x_i = b_i / a_{ii}$, $i = 1, \dots, n$.

2. Lower triangular systems:

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ a_{n1} & \dots & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

   This kind of system can be solved by *forward substitution*, i.e.,

$$x_1 = b_1/a_{11}, \quad x_2 = (b_2 - a_{21}x_1)/a_{22}, \quad \text{etc.}$$

   An algorithm for forward substitution is therefore

   Algorithm (Forward Substitution)

```
for $i$ from 1 to $n$ do
    $x_i = \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j \right) / a_{ii}$
end
```

   Note that the summation for $i = 1$ is void, and thus not carried out.

3. Upper triangular systems:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \dots & 0 & a_{nn} \end{bmatrix}.$$

   This kind of system can be solved by *backward substitution*, i.e.,

$$x_n = b_n/a_{nn}, \quad x_{n-1} = (b_{n-1} - a_{nn}x_n)/a_{n-1,n-1}, \quad \text{etc.}$$

   An algorithm for backward substitution is therefore

   Algorithm (Backward Substitution)

```
for $i$ from $n$ by -1 to 1 do
    $x_i = \left( b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) / a_{ii}$
end
```

   Note that here the summation for $i = n$ is void.

4. Permuted triangular systems are also easily solved by first applying a permutation that converts them to a triangular system.

LU Factorization

Idea: Use elementary row operations to convert A to upper triangular form U, i.e.,

$$\underbrace{L_m L_{m-1} \cdots L_1}_{=\widetilde{L}} A = U.$$

Here the elementary matrices L_i, $i = 1, \ldots, m$, will all be lower triangular matrices. In a homework problem you will show that $\widetilde{L}$ – as a product of lower triangular matrices – is also lower triangular. Also, you will show that the inverse of a lower triangular matrix is again lower triangular. Therefore, this results in a decomposition

$$A = \widetilde{L}^{-1} U = \underbrace{L}_{\text{lower } \triangle} \underbrace{U}_{\text{upper } \triangle}$$

of A.

Why bother with LU factorization? Consider the system $Ax = b$, and assume we have computed the decomposition $A = LU$, i.e., we need to solve

$$LUx = b.$$

We now let $z = Ux$, and the task of finding x becomes a *two-stage process*:

1. Solve $Lz = b$ for z by forward substitution. According to our earlier discussions, this is easily done.
2. Solve $Ux = z$ for x by backward substitution, which is just as easily done.

The real advantage of LU factorization comes when we later have to solve another system with the *same matrix A*, but *different right-hand side b*. Then the factorization can be re-used, and we only need to repeat the simple tasks of forward and backward substitution. In general, consider solving multiple systems with the same system matrix simultaneously, i.e., $AX = B$, where both X and B are $n \times m$ matrices. Then we compute $A = LU$ *once*, let $Z = UX$, and perform the two-step process simultaneously m times:

1. Solve $LZ = B$ for Z by m simultaneous forward substitutions.
2. Solve $UX = Z$ for X by m simultaneous backward substitutions.

Remark: We will see below, that the majority of work is done during the factorization step, and that substitution is relatively inexpensive.

Theorem 4.3 *The $n \times n$ matrix A has an LU factorization $A = LU$ with unique unit lower triangular factor L and unique nonsingular upper triangular factor U if and only if all leading principal minors of A are nonsingular.*

Remark: This theorem is stronger than Theorem 1 in the textbook.

Proof: The submatrix

$$A_{11}^{(k)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kk} \end{bmatrix}$$

is called the k-th leading principal minor of A.

First we show "$\Longrightarrow$", i.e., we show all leading principal minors are nonsingular. Consider A in the following block partitioned form:

$$A = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix}. \tag{2}$$

Since A has an LU decomposition we can write

$$A = \begin{bmatrix} L_{11}^{(k)} & 0 \\ L_{21}^{(k)} & L_{22}^{(k)} \end{bmatrix} \begin{bmatrix} U_{11}^{(k)} & U_{12}^{(k)} \\ 0 & U_{22}^{(k)} \end{bmatrix} = \begin{bmatrix} L_{11}^{(k)}U_{11}^{(k)} & L_{11}^{(k)}U_{12}^{(k)} \\ L_{21}^{(k)}U_{11}^{(k)} & L_{21}^{(k)}U_{12}^{(k)} + L_{22}^{(k)}U_{22}^{(k)} \end{bmatrix}. \tag{3}$$

Now, comparing (2) and (3) we see

$$\begin{aligned} \det\left(A_{11}^{(k)}\right) &= \det\left(L_{11}^{(k)}U_{11}^{(k)}\right) \\ &= \underbrace{\det L_{11}^{(k)}}_{=1}\ \underbrace{\det U_{11}^{(k)}}_{\prod_{j=1}^{k}(U_{11})_{jj}} \neq 0, \end{aligned}$$

since L is unit lower triangular, and U is nonsingular triangular, and the determinant of a triangular matrix is given by the product of its diagonal elements.

For the proof of "$\Longleftarrow$" we use induction. The case $n = 1$ is clear since

$$A = [a_{11}] \neq 0 \quad \Longrightarrow \quad L = [1],\ U = [a_{11}].$$

Now, assume that the claim holds for n, and show that it also holds for $n+1$, i.e., we need to find a unique $n \times n$ unit lower triangular matrix $L^{(n)}$ and a unique nonsingular $n \times n$ upper triangular matrix $U^{(n)}$ as well as unique $n \times 1$ vectors ℓ and u and a unique nonzero scalar η such that

$$A^{(n+1)} = \begin{bmatrix} L^{(n)} & 0 \\ \ell^T & 1 \end{bmatrix} \begin{bmatrix} U^{(n)} & u \\ 0 & \eta \end{bmatrix}$$

or

$$A^{(n+1)} = \begin{bmatrix} A^{(n)} & b \\ c^T & \delta \end{bmatrix} = \begin{bmatrix} L^{(n)}U^{(n)} & L^{(n)}u \\ \ell^T U^{(n)} & \ell^T u + \eta \end{bmatrix}.$$

By the induction hypothesis $A^{(n)} = L^{(n)}U^{(n)}$ satisfying the required properties. Let

$$\begin{aligned} u &= (L^{(n)})^{-1}b, \\ \ell^T &= c^T (U^{(n)})^{-1} \\ \eta &= \delta - \ell^T u. \end{aligned}$$

All three quantities are uniquely determined. Furthermore, η is nonzero since by assumption

$$\begin{aligned} 0 &\neq \det A^{(n+1)} \\ &= \underbrace{\det \begin{bmatrix} L^{(n)} & 0 \\ \ell^T & 1 \end{bmatrix}}_{=1} \det \begin{bmatrix} U^{(n)} & u \\ 0 & \eta \end{bmatrix} \\ &= \det U^{(n)} \eta. \end{aligned}$$

By the induction hypothesis $\det U^{(n)} \neq 0$, and so η must also be nonzero. ♠

Derivation of the Factorization Algorithm

We want to have

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & \dots & 0 \\ \ell_{21} & \ell_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ \ell_{n1} & \ell_{n2} & \dots & \ell_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & & \vdots \\ \vdots & & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{bmatrix},$$

so

$$a_{ij} = \sum_{s=1}^{n} \ell_{is} u_{sj} = \sum_{s=1}^{\min(i,j)} \ell_{is} u_{sj} \tag{4}$$

since $\ell_{is} = 0$ for $s > i$ and $u_{sj} = 0$ for $s > j$. We will construct U row-by-row, and L column-by-column. We begin with

$$a_{11} = \ell_{11} u_{11}, \tag{5}$$

i.e., if we fix one of ℓ_{11} or u_{11}, then the other can be computed from (5). We continue with

$$a_{1j} \overset{(4)}{=} \sum_{s=1}^{1} \ell_{1s} u_{sj} = \ell_{11} u_{1j}, \qquad j = 2, 3, \dots, n,$$

which yields the entries in the first row of U as

$$u_{1j} = \frac{a_{1j}}{\ell_{11}}, \qquad j = 2, 3, \dots, n.$$

Analogously,

$$a_{i1} \overset{(4)}{=} \sum_{s=1}^{1} \ell_{is} u_{s1} = \ell_{i1} u_{11}, \qquad i = 2, 3, \dots, n,$$

which yields the entries in the first column of L as

$$\ell_{i1} = \frac{a_{i1}}{u_{11}}, \qquad i = 2, 3, \dots, n.$$

The remainder is obtained recursively, assuming that rows $1, \dots, k-1$ of U and columns $1, \dots, k-1$ of L have been determined. Then

$$a_{kk} \overset{(4)}{=} \sum_{s=1}^{k} \ell_{ks} u_{sk}$$

$$= \underbrace{\sum_{s=1}^{k-1} \ell_{ks} u_{sk}}_{\text{known}} + \ell_{kk} u_{kk}. \tag{6}$$

Thus, the diagonal elements ℓ_{kk} and u_{kk} can be determined by fixing one and computing the other from (6).

To get the remainder of the k-th row of U we consider

$$a_{kj} \overset{(4)}{=} \sum_{s=1}^{k} \ell_{ks} u_{sj} = \sum_{s=1}^{k-1} \ell_{ks} u_{sj} + \ell_{kk} u_{kj}, \qquad j = k+1, \ldots, n,$$

which yields

$$\boxed{u_{kj} = \frac{a_{kj} - \sum_{s=1}^{k-1} \ell_{ks} u_{sj}}{\ell_{kk}}, \qquad j = k+1, \ldots, n.}$$

The k-th column of L is obtained similarly, i.e.,

$$a_{ik} \overset{(4)}{=} \sum_{s=1}^{k} \ell_{is} u_{sk} = \sum_{s=1}^{k-1} \ell_{is} u_{sk} + \ell_{ik} u_{kk}, \qquad i = k+1, \ldots, n,$$

yields

$$\boxed{\ell_{ik} = \frac{a_{ik} - \sum_{s=1}^{k-1} \ell_{is} u_{sk}}{u_{kk}}, \qquad i = k+1, \ldots, n.}$$

Remark: These formulas require $\ell_{kk}, u_{kk} \neq 0$, $k = 1, \ldots, n$. The choice of one of ℓ_{kk} or u_{kk} acts as a free parameter. A common choice – in agreement with Theorem 4.3 – is $\ell_{kk} = 1$, $k = 1, \ldots, n$, i.e., L is taken as a *unit* lower triangular matrix.

Summarizing the above derivation, an algorithm for basic LU factorization is given by

Algorithm (Doolittle's Factorization)

```
Input $A$, $n$
for $k$ from 1 to $n$ do
    $\ell_{kk} = 1$
    for $j$ from $k$ to $n$ do    % rows of $U$
        $u_{kj} = a_{kj} - \sum_{s=1}^{k-1} \ell_{ks} u_{sj}$
    end
    for $i$ from $k+1$ to $n$ do    % columns of $L$
        $\ell_{ik} = \left( a_{ik} - \sum_{s=1}^{k-1} \ell_{is} u_{sk} \right) / u_{kk}$
    end
end
Output $L$, $U$
```

Remarks:

1. In the textbook LU factorization is identified with Doolittle's factorization (compare also Theorem 4.3 above).
2. If we choose $u_{kk} = 1$, $k = 1, \ldots, n$ in the general algorithm, then we obtain *Crout's factorization*.
3. Fixing n other values of L or U may lead to nonlinear conditions (cf. homework problem 4.2#50).

Cholesky Factorization

Theorem 4.4 *If A is a real symmetric positive definite matrix then it has a unique factorization $A = LL^T$, where L is lower triangular with positive (usually not unit) diagonal.*

Proof: Since A is positive definite A is nonsingular (otherwise there would exist an $x \neq 0$ such that $Ax = 0$).

Also, all k-th order leading principal minors of A are positive definite and therefore nonsingular. This can be seen by picking $x = [x_1, \ldots, x_k, 0, \ldots, 0]^T$.

Therefore, by Theorem 4.3 an LU factorization $A = LU$ of A exists.

Now, A is symmetric, i.e.,

$$LU = A = A^T = (LU)^T = U^T L^T.$$

This implies (see homework problem 4.2.1)

$$\underbrace{U\,\underbrace{(L^T)^{-1}}_{\text{upper}\Delta}}_{\text{upper}\Delta} = \underbrace{L^{-1}\quad \underbrace{U^T}_{\text{lower}\Delta}}_{\text{lower}\Delta},$$

or

$$U(L^T)^{-1} = D,$$

with D a diagonal matrix. This is equivalent to $U = DL^T$, and therefore

$$A = LDL^T. \tag{7}$$

If A is positive definite and L nonsingular, then D is positive definite (see homework problem 4.2.26). Since a positive definite diagonal matrix must have positive diagonal entries we have

$$d_{ii} > 0, \qquad i = 1, \ldots, n.$$

If we define a new diagonal matrix using the symbolic notation $D^{1/2} = \left(\sqrt{d_{ii}}\right)_{i=1}^{n}$, then

$$A = \underbrace{LD^{1/2}}_{\widetilde{L}}\underbrace{D^{1/2}L^T}_{\widetilde{L}^T}.$$

Since L and U were unique by Theorem 4.3 D is unique, and so is $D^{1/2}$. Thus the Cholesky factorization is unique. ♠

Remark: Often the Cholesky factorization is listed in the form given in (7).

Algorithm (Cholesky)

```
Input $A$, $n$

for $k = 1$ to $n$ do
    $\ell_{kk} = \left(a_{kk} - \sum_{s=1}^{k-1} \ell_{ks}^2\right)^{1/2}$
    for $i = k+1$ to $n$ do
        $\ell_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} \ell_{is}\ell_{ks}\right) / \ell_{kk}$
    end
end

Output $L$
```

Note: From the algorithm we can see

$$a_{kk} = \sum_{s=1}^{k} \ell_{ks}^2 \geq \ell_{kj}^2, \qquad j \leq k.$$

Therefore

$$|\ell_{kj}| \leq \sqrt{a_{kk}}$$

which shows that the entries of L are bounded by the (square root of) the diagonal entries of A. This has a positive effect on the stability of the Cholesky factorization (more later).

Remark: The Cholesky factorization algorithm is important in practice. The basic Doolittle (LU) factorization algorithm will be superseded by Gaussian elimination with pivoting in the next section.

4.3 Gaussian Elimination and Pivoting

We begin with an implementation of basic Gaussian elimination (LU factorization):

Algorithm (Gaussian Elimination)

```
Input $A$, $n$

for $k = 1$ to $n-1$ do        % loop over columns

    for $i = k+1$ to $n$ do        % down column $k$
        $a_{ik} = a_{ik}/a_{kk}$        % store multipliers (entries of $L$) in place of zeros
        for $j = k+1$ to $n$ do     % across row $i$
            $a_{ij} = a_{ij} - a_{ik}a_{kj}$        % overwrite $A$ with $U$
        end
    end

end

Output $A$        % contains $L$ (except for unit diagonal) and $U$
```

Ex.: Consider solving

$$Ax = b \quad \Leftrightarrow \quad \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

The Gaussian elimination algorithm would proceed as follows:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 3 & 4 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & -2 \\ 3 & -2 & -8 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & -2 \\ 3 & \frac{2}{3} & -\frac{20}{3} \end{bmatrix}.$$

Thus, the factors L and U of A are given as

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & \frac{2}{3} & 1 \end{bmatrix} \qquad \text{and} \qquad U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -2 \\ 0 & 0 & -\frac{20}{3} \end{bmatrix}.$$

Note: Even though the matrix in the preceeding example is symmetric the Cholesky algorithm cannot be applied since A is not positive definite.

Theorem 4.5 *If all pivots a_{kk} are nonzero then the algorithm listed above produces Doolittle's factorization.*

Proof: See textbook. ♠

Remark: In order to solve the linear system $Ax = b$ we need another algorithm that "updates" the right-hand side b and applies backward substitution (see below).

Pivoting

Problems will arise during Gaussian elimination if the diagonal element of A to be used for the next elimination step is very small compared to the other matrix entries or even zero.

Ex.: Consider $Ax = b$ with

$$A = \begin{bmatrix} 6 & 2 & 2 \\ 2 & \frac{2}{3} & \frac{1}{3} \\ 1 & 2 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}.$$

Without pivoting we obtain the answer $x = [1.3333335, 0, -5.0000003]^T$, whereas with pivoting we obtain $x = [2.6, -3.8, -5]^T$ which is also the exact solution. For more details please refer to the Maple worksheet `577_LUnew.mws`.

Scaled Row Pivoting

One of the most popular methods of dealing with the above mentioned problem is to introduce *scales*

$$s_i = \max_{1 \le j \le n} |a_{ij}|,$$

i.e., s_i is the largest element in the i-th row of the original matrix A. These scales are determined only once.

Before the k-th elimination step we select that row as *pivot row* whose ratio

$$\frac{|a_{p_i k}|}{s_i}, \qquad i = k, \ldots, n,$$

is greatest. Here $p = [p_1, \ldots, p_n]^T$ is a *permutation vector* (initialized with $[1, \ldots, n]^T$). If at the k-th step row j is selected as pivot row, then the entries p_k and p_j in p are swapped. Use of this permutation vector avoids swapping rows in physical memory.

Ex.: Consider again the example

$$A = \begin{bmatrix} 6 & 2 & 2 \\ 2 & \frac{2}{3} & \frac{1}{3} \\ 1 & 2 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}.$$

We initialize p as $p = [1, 2, 3]^T$, and compute the scales

$$\begin{aligned} s_1 &= \max_{1 \le j \le 3} |a_{1j}| = \max\{6, 2, 2\} = 6, \\ s_2 &= \max_{1 \le j \le 3} |a_{2j}| = \max\{2, \frac{2}{3}, \frac{1}{3}\} = 2, \end{aligned}$$

$$s_3 = \max_{1\le j\le 3} |a_{3j}| = \max\{1,2,1\} = 2.$$

Now, for k from 1 to $n-1=2$ we go through pivoting procedure:

$\underline{k=1:}$ We begin by determining the pivot row for the first step, i.e., we compute the ratios $\dfrac{|a_{p_i1}|}{s_i}$ for $i=1,2,3$:

$$\begin{aligned}\frac{|a_{11}|}{s_1} &= \frac{6}{6} = 1,\\ \frac{|a_{21}|}{s_2} &= \frac{2}{2} = 1,\\ \frac{|a_{31}|}{s_3} &= \frac{1}{2}.\end{aligned}$$

Thus, the first row is chosen as the pivot row, and no change to p is required, i.e., $p=[1,2,3]^T$.

Now we perform elimination as explained in the basic Gaussian elimination algorithm above:

$$\begin{bmatrix} 6 & 2 & 2\\ 2 & \frac{2}{3} & \frac{1}{3}\\ 1 & 2 & -1\end{bmatrix} \longrightarrow \begin{bmatrix} 6 & 2 & 2\\ \frac{1}{3} & 0 & -\frac{1}{3}\\ \frac{1}{6} & \frac{5}{3} & -\frac{4}{3}\end{bmatrix},$$

where the entries in the $(2,1)$ and $(3,1)$ position contain the multipliers $m_{p_i1} = \dfrac{a_{p_i1}}{a_{p_11}}$, $i=2,3$.

$\underline{k=2:}$ We compute the ratios $\dfrac{|a_{p_i2}|}{s_i}$ for $i=2,3$:

$$\begin{aligned}\frac{|a_{22}|}{s_2} &= \frac{0}{2} = 0,\\ \frac{|a_{32}|}{s_3} &= \frac{\frac{5}{3}}{2}.\end{aligned}$$

This time the third row is chosen as the pivot row, and we need to permute entries 2 and 3 of p, i.e., $p=[1,3,2]^T$.

Next we perform elimination. Since the multiplier $m_{p_32} = \dfrac{a_{p_32}}{a_{p_22}} = \dfrac{a_{22}}{a_{32}} = 0$ there is no more work to be done, and we can read off the factorization from the matrix

$$\begin{bmatrix} 6 & 2 & 2\\ \frac{1}{3} & 0 & -\frac{1}{3}\\ \frac{1}{6} & \frac{5}{3} & -\frac{4}{3}\end{bmatrix}.$$

The permutation vector can be converted to a *permutation matrix*

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

the multipliers (permuted back) are collected in

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{6} & 1 & 0 \\ \frac{1}{3} & 0 & 1 \end{bmatrix},$$

and (also permuted back)

$$U = \begin{bmatrix} 6 & 2 & 2 \\ 0 & \frac{5}{3} & -\frac{4}{3} \\ 0 & 0 & -\frac{1}{3} \end{bmatrix}.$$

Together, we now have a factorization of the form

$$PA = LU.$$

Solution Phase

Thus far we have concentrated on the factorization phase. Now we discuss the solution phase of the algorithm. Let's consider $Ax = b$, and – since we have computed the LU factorization of A with scaled row pivoting – see how to solve the permuted system $PAx = Pb$. Making use of the LU factorization we get

$$LUx = Pb.$$

As discussed earlier, we approach this problem with a two-step procedure:

1. Solve $Lz = Pb$, or *update* b via $z = L^{-1}Pb$.
2. Solve $Ux = z$, i.e., do backward substitution.

Note: The previous procedure is summarized in Theorem 3 on page 174 of the textbook.

Ex.: We continue the previous example where

$$b = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}, \qquad p = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix},$$

and (after the factorization phase)

$$A = \begin{bmatrix} 6 & 2 & 2 \\ \frac{1}{3} & 0 & -\frac{1}{3} \\ \frac{1}{6} & \frac{5}{3} & -\frac{4}{3} \end{bmatrix}.$$

This corresponds to

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \text{and} \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{6} & 1 & 0 \\ \frac{1}{3} & 0 & 1 \end{bmatrix}.$$

In algorithmic form the update of b (step 1) can be accomplished as

```
for $k = 1$ to $n-1$ do
    for $i = k+1$ to $n$ do
        $b_{p_i} = b_{p_i} - a_{p_i k} b_{p_k}$
    end
end
```

Thus, we have

$$\begin{array}{rrcl} k=1,\ i=2: & b_3 & = & b_3 - a_{31}b_1 \\ & & = & 0 - \frac{1}{6}(-2) = \frac{1}{3}, \\ \\ i=3: & b_2 & = & b_2 - a_{21}b_1 \\ & & = & 1 - \frac{1}{3}(-2) = \frac{5}{3}, \\ \\ k=2,\ i=3: & b_2 & = & b_2 - a_{22}b_3 \\ & & = & \frac{5}{3} - 0\frac{1}{3} = \frac{5}{3}, \end{array}$$

and therefore the updated b is $b = [-2, \frac{5}{3}, \frac{1}{3}]^T$.

The algorithm for the backward substitution (step 2) is given as

```
for $i = n$ by $-1$ to $1$ do
    $x_i = \left( b_{p_i} - \sum_{j=i+1}^{n} a_{p_i j} x_j \right) / a_{p_i i}$
end
```

This leads to

$$\begin{aligned} x_3 &= b_2/a_{23} = \frac{5}{3} / \left(-\frac{1}{3}\right) = -5, \\ x_2 &= \left(b_3 - \sum_{j=3}^{3} a_{3j}x_j \right) / a_{32} \\ &= (b_3 - a_{33}x_3)/a_{32} = \left(\frac{1}{3} - \left(-\frac{4}{3}\right)(-5) \right) / \frac{5}{3} = -\frac{19}{5} = -3.8, \\ x_1 &= \left(b_1 - \sum_{j=2}^{3} a_{1j}x_j \right) / a_{11} \\ &= (b_1 - (a_{12}x_2 + a_{13}x_3))/a_{11} \\ &= (-2 - (2(-3.8) + 2(-5)))/6 = \frac{15.6}{6} = 2.6. \end{aligned}$$

Remark: The complete algorithm can be found in the textbook on pages 171 and 172.

Operations Count

To get an idea of the efficiency of an algorithm , traditionally, the number of floating point operations are counted. On older machines multiplication and division were more expensive than addition and subtraction, so often only so-called "long operations" were counted. Today, all operations are roughly equivalent, so we count all floating point operations (flops).

Remark: Many other factors affect the efficiency of an algorithm. E.g., data movement, fine-tuning to the machines architecture, choice of programming language, matrix size for linear systems, etc. An example of making optimal use of the available resources for linear algebra problems is to use the program library LAPACK (available from http://www.netlib.org/). It uses block algorithms and is built on BLAS routines that are fine-tuned to the computer's architecture.

We now do an operations count (all flops) for the algorithms on pages 171 and 172 of the textbook. The main idea is to replace for-loops by summation.

Factorization: Since there are two floating point operations in the innermost loop, and one additional operation in each of the surrounding loops, the operations count for the factorization algorithm translate into

$$\begin{aligned}
& \sum_{k=1}^{n-1}\left[1+\sum_{i=k+1}^{n}\left(1+\sum_{j=k+1}^{n}2\right)\right] \\
= & \sum_{k=1}^{n-1}\left[1+\sum_{i=k+1}^{n}(1+2(n-k))\right] \\
= & \sum_{k=1}^{n-1}\left[1+(n-k)+2(n-k)^2\right] \\
= & n-1+\underbrace{\sum_{k=1}^{n-1}(n-k)}_{\frac{(n-1)n}{2}}+2\underbrace{\sum_{k=1}^{n-1}(n-k)^2}_{\frac{(n-1)n(2n-1)}{6}} \\
= & \frac{2}{3}n^3-\frac{n^2}{2}+\frac{5}{6}n-1=\mathcal{O}(n^3).
\end{aligned}$$

Updating b: There are only two flops inside the inner loop, thus

$$\sum_{k=1}^{n-1}\sum_{i=k+1}^{n}2=\sum_{k=1}^{n-1}2(n-k)=n^2-n=\mathcal{O}(n^2).$$

Backward substitution: Since the summation inside the for-loop involves both addition and multiplication, we get

$$\sum_{i=n}^{1}(2(n-i)+1)=\sum_{i=1}^{n}(2n-2i+1)=n^2=\mathcal{O}(n^2).$$

So, if we solve m linear systems with the same system matrix A, but m different right-hand sides b, then we get

Theorem 4.6 *The solution of $Ax = b$ with fixed matrix A and m different right-hand side vectors b by Gaussian elimination with scaled row pivoting involves approximately $\frac{2}{3}n^3 + \left(2m - \frac{1}{2}\right) n^2$ floating point operations.*

Proof: We simply add the terms of orders n^3 and n^2 from the previous three counts:

$$\begin{array}{cccc}
 & \frac{2}{3}n^3 & & -\frac{n^2}{2} \\
+ & & & mn^2 \\
+ & & & mn^2 \\
\hline
+ & \frac{2}{3}n^3 & + & \left(2m - \frac{1}{2}\right) n^2.
\end{array}$$

♠

__Remark:__ This shows that applying LU factorization ($\mathcal{O}(n^3)$) once with many backward substitutions ($\mathcal{O}(n^2)$) is more efficient than doing a complete Gaussian elimination ($\mathcal{O}(n^3)$) over and over again.

Corollary 4.7 *The inverse A^{-1} of A can be computed in roughly $\frac{8}{3}n^3$ operations.*

__Remark:__ Don't compute A^{-1} to solve $Ax = b$!

Special Matrices

Definition 4.8 *An $n \times n$ matrix A is called (strictly)* diagonally dominant *if*

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \qquad i = 1, \ldots, n.$$

Theorem 4.9 *For diagonally dominant matrices Gaussian elimination (LU factorization) requires no pivoting.*

Proof: See the textbook. ♠

__Remark:__ For symmetric positive definite matrices Cholesky factorization requires no pivoting.

Tridiagonal Matrices

A tridiagonal matrix has nonzero entries only on its main diagonal as well as the first sub- and super-diagonals, i.e., it is of the form

$$\begin{bmatrix}
d_1 & c_1 & 0 & \cdots & 0 \\
a_1 & d_2 & c_2 & & \vdots \\
0 & a_2 & \ddots & \ddots & 0 \\
\vdots & & \ddots & \ddots & c_{n-1} \\
0 & \cdots & 0 & a_{n-1} & d_n
\end{bmatrix}.$$

An efficient algorithm for solving tridiagonal linear systems can be found in the textbook on page 180.

4.4 Norms and Error Analysis

We are still interested in solving the linear system $Ax = b$, but now want to focus our attention on the accuracy and stability of a solution obtained by a numerical method.

We will denote the solution obtained by the computer with $\widetilde{x}$, and the exact solution by x. We therefore have an *error* $e = x - \widetilde{x}$. Since, in general, we don't know the exact solution x, we cannot measure e. We can, however, check the *residual*

$$r = b - A\widetilde{x} = A(x - \widetilde{x}).$$

Intuition suggests that we can *hope* that a small residual indicates also a small error.

Example: Consider a linear system with

$$A = \begin{bmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} 2 \\ 2 \end{bmatrix}.$$

Then the exact solution is $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

(a) Assume the computer returns the approximate solution $\widetilde{x} = \begin{bmatrix} 1.01 \\ 1.01 \end{bmatrix}$. Then we determine the error and the residual, respectively, to be

$$e = \begin{bmatrix} -0.01 \\ -0.01 \end{bmatrix} \quad \text{and} \quad r = \begin{bmatrix} -0.02 \\ -0.02 \end{bmatrix}.$$

Thus, everything goes according to our intuition: the residual is small, and so is the error.

(b) Now assume the approximate solution is $\widetilde{x} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$. Then

$$e = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \text{and} \quad r = \begin{bmatrix} -0.02 \\ 0.02 \end{bmatrix},$$

and the small residual no longer is an indicator of a small error.

(c) This time we change the right-hand side b to $b = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$. Then the exact solution is given by $x = \begin{bmatrix} 100 \\ -100 \end{bmatrix}$. If now the approximate solution is $\widetilde{x} = \begin{bmatrix} 101 \\ -99 \end{bmatrix}$, then

$$e = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad \text{and} \quad r = \begin{bmatrix} -2 \\ 2 \end{bmatrix},$$

and we have a large residual along with a small (relative) error.

We observe that (at least for the matrix in this example) there is no apparent relation between the size of the residual and the size of the error. What's wrong with our intuition?

In order to investigate this question further we need to introduce some tool that lets us measure the "size" of A as well as its inverse. The appropriate tool is the concept of a *norm* of a vector or a matrix.

Definition 4.10 *Let V be a vector space. Then a* norm $\|\cdot\| : V \to \mathbb{R}_{\geq 0}$ *must satisfy*

1. $\|x\| > 0$ *for all* $x (\neq 0) \in V$.
2. $\|\lambda x\| = |\lambda| \|x\|$ *for any* $\lambda \in \mathbb{R}$, $x \in V$.
3. $\|x + y\| \leq \|x\| + \|y\|$ *for all* $x, y \in V$ *(the triangle inequality).*

Examples of vector norms: Consider a vector $x = [x_1, \dots, x_n]^T \in \mathbb{R}^n$. The most popular vector norms are:

(a) $\|x\|_1 = \sum_{i=1}^{n} |x_i|$, the so-called ℓ_1-norm,

(b) $\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$, the so-called ℓ_2-norm (or Euclidean norm),

(c) $\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$ the so-called ℓ_∞-norm (maximum norm, or Chebyshev norm).

It is interesting to consider the corresponding unit "spheres" for these three norms, i.e., the location of points in $\mathbb{R}^n$ whose distance to the origin (in the respective norm) is equal to 1. Figure 1 illustrates this for the case $n = 2$.

Matrix norms:

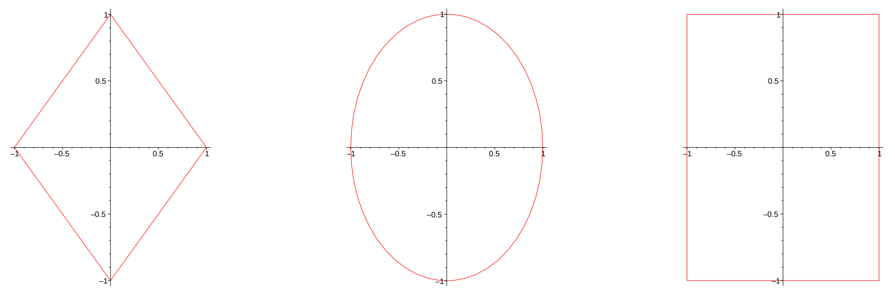

Figure 1: Unit "circles" in $\mathbb{R}^2$ for the ℓ_1, ℓ_2 and ℓ_∞ norms.

Definition 4.11 *Let $\|\cdot\|$ be some vector norm on $\mathbb{R}^n$, and let A be an $n \times n$ matrix. The* associated *or* subordinate matrix norm *is defined by*

$$\|A\| = \sup_{\|u\|=1} \|Au\|, \qquad u \in \mathbb{R}^n\,.$$

<u>**Remarks:**</u>

1. The notation sup in Definition 4.11 denotes the *supremum* or least upper bound.
2. An alternate definition of the associated matrix norm is
$$\|A\| = \max_{\substack{x\in\mathbb{R}^n\\ x\neq 0}} \frac{\|Ax\|}{\|x\|}.$$
3. $\|A\|$ can be interpreted as the maximum factor by which A can "stretch" x.

Theorem 4.12 *If V is the vector space of $n \times n$ matrices, then $\|A = \sup_{\|u\|=1} \|Au\|$ satisfies properties (1)–(3) of Definition 4.10.*

Proof:

(1) Assume $A \neq 0$. Then A has at least one nonzero column $A^{(j)}$.

Let $x = [0,\ldots,0,\underbrace{1}_{j\text{-th position}},0,\ldots,0]^T \in \mathbb{R}^n$, and note that x is a unit vector.

Then, by Definition 4.11, the definition of the supremum, the choice of x, and the properties of a vector norm,

$$\|A\| = \sup_{\|u\|=1} \|Au\| \geq \|Ax\| = \|A^{(j)}\| > 0.$$

(2) Let $\lambda \in \mathbb{R}$. Then, by Definition 4.11, and the norm property (2) on $\mathbb{R}^n$,

$$\|\lambda A\| = \sup_{\|u\|=1} \|\lambda \underbrace{Au}_{\in\mathbb{R}^n}\| = |\lambda| \sup_{\|u\|=1} \|Au\| = |\lambda|\|A\|.$$

(3) If both A and B are $n \times n$ matrices, then

$$\begin{aligned}\|A+B\| &= \sup_{\|u\|=1} \|(A+B)u\| = \sup_{\|u\|=1} \|Au+Bu\| \\ &\overset{(3)\text{ on }\mathbb{R}^n}{\leq} \sup_{\|u\|=1} (\|Au\|+\|Bu\|) \\ &\overset{\text{HW\#4}}{\leq} \sup_{\|u\|=1} \|Au\| + \sup_{\|u\|=1} \|Bu\| = \|A\|+\|B\|.\end{aligned}$$

♠

Corollary 4.13 *If A is an $n \times n$ matrix and $x \in \mathbb{R}^n$, then*

$$\|Ax\| \leq \|A\|\|x\|.$$

Proof: By the definition of a supremum

$$\|A\| = \sup_{\|u\|=1} \|Au\| \geq \|Av\| = \frac{\|Ax\|}{\|x\|},$$

where $x \neq 0$ is an arbitrary vector in $\mathbb{R}^n$ and $v = \dfrac{x}{\|x\|}$ is an associated unit vector. Note that the statement is also true if $x = 0$. ♠

<u>**Examples of matrix norms:**</u> Consider an $n \times n$ matrix A. The most popular matrix norms are:

(a) $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}|$, the so-called *column sum norm* (see HW#11),

(b) $\|A\|_2 = \max_{1 \leq i \leq n} |\sigma_i|$, where σ_i is the i-th *singular value* of A (more later, fairly difficult to compute),

(c) $\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$, the so-called *row sum norm.*

We now verify that the matrix norm induced by the ℓ_∞-norm is indeed given by the formula stated in (c):

$$\|A\|_\infty = \sup_{\|u\|_\infty=1} \|Au\|_\infty = \sup_{\|u\|_\infty=1} \max_{1 \leq i \leq n} |(Au)_i|.$$

By interchanging the supremum and maximum we obtain

$$\max_{1 \leq i \leq n} \sup_{\|u\|_\infty=1} |(Au)_i|.$$

Next, we rewrite the matrix-vector product to get

$$\max_{1 \leq i \leq n} \sup_{\|u\|_\infty=1} |\sum_{j=1}^{n} a_{ij} u_j|.$$

Finally, using formula (8) below we obtain the desired result, i.e.,

$$\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|.$$

We now derive formula (8). Consider

$$|\sum_{j=1}^{n} a_{ij} u_j| = |a_{i1}u_1 + a_{i2}u_2 + \ldots + a_{in}u_n|.$$

The supremum over all unit vectors (in the maximum norm) is attained if all terms in the above sum are positive. Therefore, we pick $u_j = \text{sign}(a_{ij})$. But then we have

$$\sup_{\|u\|_\infty=1} |\sum_{j=1}^{n} a_{ij}u_j| = |a_{i1}| + |a_{i2}| + \ldots + |a_{in}| = \sum_{j=1}^{n} |a_{ij}|. \tag{8}$$

We now return to our analysis of the relationship between the size of the residual and the size of the error in solving $Ax = b$. We consider the error

$$e = x - \widetilde{x},$$

and the residual

$$r = b - A\widetilde{x} = b - \widetilde{b},$$

where $\widetilde{x}$ is the computed solution of $Ax = b$ (and the true solution of some other problem $A\widetilde{x} = \widetilde{b}$).

Now, applying Corollary 4.13 to an arbitrary vector norm,

$$\begin{aligned} \|x - \widetilde{x}\| &= \|A^{-1}b - A^{-1}\widetilde{b}\| = \|A^{-1}(b - \widetilde{b})\| \\ &\leq \|A^{-1}\|\|b - \widetilde{b}\|, \end{aligned}$$

i.e.,

$$\|e\| \leq \|A^{-1}\|\|r\|, \tag{9}$$

and the absolute error can be bounded by the absolute residual times $\|A^{-1}\|$. Thus, if the norm of the inverse of A is small, then a small residual will indeed be a good indicator for a small error.

Next we turn our attention to relative errors. Starting from (9), and using Corollary 4.13 again, we obtain

$$\begin{aligned} \|e\| &\leq \|A^{-1}\| \underbrace{\frac{\|Ax\|}{\|b\|}}_{=1} \|r\| \leq \|A^{-1}\|\|A\|\|x\|\frac{\|r\|}{\|b\|} \\ &\iff \frac{\|e\|}{\|x\|} \leq \kappa(A)\frac{\|r\|}{\|b\|}, \end{aligned} \tag{10}$$

where

$$\kappa(A) = \|A^{-1}\|\|A\|$$

is called the *condition number of* A. Thus, if the condition number of A is small, then a small relative residual will be a good indicator for a small relative error. In other words, if the condition number of A is large, then small changes in the right-hand side b may lead to large changes in the solution. This is an *unstable* situation.

Remark: $\kappa(A) \geq 1$ for any nonsingular matrix A and any matrix norm (see HW#13).

To see the complete relationship between the relative residual and relative error, we also derive a lower bound for the relative error. Consider

$$\begin{aligned} \|r\|\|x\| &= \|b - \widetilde{b}\|\|x\| \\ &= \|Ax - A\widetilde{x}\|\|x\| = \|A(x - \widetilde{x})\|\|x\| \end{aligned}$$

$$\begin{aligned} &= \|Ae\|\|x\| \\ &= \|Ae\|\|A^{-1}b\| \leq \|A\|\|e\|\|A^{-1}\|\|b\|. \end{aligned}$$

Therefore
$$\frac{1}{\kappa(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|}. \tag{11}$$

Remark: Combining (10) and (11) we see that if A is well-conditioned, i.e., $\kappa(A) \approx 1$, then we have tight upper and lower bounds on the size of the relative error. This means, for well-conditioned matrices, the relative residual is a good error indicator. For large $\kappa(A)$ the gap between the lower and upper bound can be rather large, and the size of the (relative) residual can no longer be trusted as an error indicator.

Back to our Example:

We can determine the inverse of the matrix $A = \begin{bmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{bmatrix}$ to be

$$A^{-1} = \begin{bmatrix} 25.25 & -24.75 \\ -24.75 & 25.25 \end{bmatrix}.$$

Now we compute the ℓ_∞-condition number of A, i.e.,

$$\begin{aligned} \|A\|_\infty &= \max\{|1.01| + |0.99|, |0.99| + |1.01|\} = 2, \\ \text{and} \quad \|A^{-1}\|_\infty &= \max\{|25.25| + |-24.75|, |-24.75| + |25.25|\} = 50, \end{aligned}$$

which leads to
$$\kappa_\infty(A) = \|A\|_\infty\|A^{-1}\|_\infty = 100.$$

By (10) and (11) we get
$$\frac{1}{100}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|} \leq 100\frac{\|r\|}{\|b\|},$$

and large variations (up to a factor of 10^4) in the relative error are possible. The matrix A is *ill-conditioned*.

Remarks:

1. The condition number of A depends on the choice of matrix norm. If someone mentions *the* condition number, usually use of the ℓ_2-norm is implied. It can be computed as
$$\kappa_2(A) = \frac{\sigma_{max}}{\sigma_{min}},$$
where σ_{max} and σ_{min} are the maximum and minimum singular values of A, respectively.

2. For practical purposes there are condition number estimators which do not require knowledge of A^{-1} or the singular values of A. An example of such an estimator is the routine `sgesvx` in LAPACK.

3. The reciprocal of the condition number can be interpreted as the "distance" of A from the nearest singular matrix.

The relative error and changes in A:

We end our discussion of the condition number by taking a look at how changes in the system matrix A affect the solution of $Ax = b$. This point of view is not included in our textbook.

We are thus interested in solving the linear system $Ax = b$, but may not have the exact entries of A available. Instead, we may only have those of $\widetilde{A}$, a perturbed version of A, i.e.,

$$\widetilde{A} = A + C,$$

where C is some matrix containing the contamination of the original matrix A. We let $\widetilde{x}$ be the (exact) solution of $\widetilde{A}\widetilde{x} = b$.

Now

$$\begin{aligned} x &= A^{-1}b = A^{-1}\widetilde{A}\widetilde{x} \\ &= A^{-1}\left(\widetilde{A} - A + A\right)\widetilde{x} \\ &= A^{-1}(C + A)\widetilde{x} = A^{-1}C\widetilde{x} + \widetilde{x}. \end{aligned}$$

Therefore

$$\begin{aligned} \|x - \widetilde{x}\| &= \|A^{-1}C\widetilde{x}\| \\ &\le \|A^{-1}\|\frac{\|A\|}{\|A\|}\|C\|\|\widetilde{x}\| \end{aligned}$$

or

$$\frac{\|x - \widetilde{x}\|}{\|\widetilde{x}\|} \le \kappa(A)\frac{\|C\|}{\|A\|}. \tag{12}$$

Remark: Equation (12) implies that – for ill-conditioned matrices – a small perturbation of A may lead to large errors in the solution of $Ax = b$.

Example: Let

$$A = \begin{bmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{bmatrix} \qquad \text{and} \qquad b = \begin{bmatrix} 2 \\ -2 \end{bmatrix},$$

so that the exact solution of $Ax = b$ is $x = \begin{bmatrix} 100 \\ -100 \end{bmatrix}$. Now, with the perturbation

$$C = \begin{bmatrix} -0.01 & 0.01 \\ 0 & 0 \end{bmatrix}$$

we get

$$\widetilde{A} = \begin{bmatrix} 1 & 1 \\ 0.99 & 1.01 \end{bmatrix},$$

and we determine $\widetilde{x}$, the solution of $\widetilde{A}\widetilde{x} = b$, to be $\widetilde{x} = \begin{bmatrix} 201 \\ -199 \end{bmatrix}$. Thus, we see that a (relatively) small change in A has affected a large change in the solution.

If

$$C = \begin{bmatrix} -0.01 & 0.01 \\ 0.01 & -0.01 \end{bmatrix},$$

then

$$\widetilde{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

and the situation is so bad that $\widetilde{A}\widetilde{x} = b$ has no solution at all.

We give one more bound (without deriving it) for problems whose system matrix and right-hand side are both perturbed:

$$\underbrace{\frac{\|e\|}{\|x\|}}_{\substack{\text{relative error}\\ \text{in solution}}} \le \frac{\kappa(A)}{1-\kappa(A)\frac{\|C\|}{\|A\|}} \underbrace{\left(\frac{\|C\|}{\|A\|} + \frac{\|r\|}{\|b\|}\right)}_{\substack{\text{relative errors}\\ \text{in input}}},$$

where C is a perturbation of A (so that $\widetilde{A} = A + C$), $r = b - \widetilde{b}$ (with $\widetilde{b}$ a perturbed right-hand side), and $e = x - \widetilde{x}$ (with $\widetilde{x}$ the computed solution of the perturbed problem $\widetilde{A}\widetilde{x} = \widetilde{b}$). This bound holds only if

$$\kappa(A)\frac{\|C\|}{\|A\|} < 1.$$

4.5 Iterative Refinement

Consider applying Newton's method to the function $f(x) = b - Ax$. If we treat x as a one-dimensional variable, then $f'(x) = -A$, so that

$$\frac{1}{f'(x)} = -A^{-1}.$$

If we now assume $x \in \mathbb{R}^n$ and formally apply the same ideas, then Newton's iteration becomes

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n + A^{-1}(b - Ax_n). \qquad (13)$$

Remark: Since f is a linear function, Newton's method (theoretically) converges – for any starting value x_0 – in a single iteration. However, roundoff errors in the computation of A^{-1} usually prevent this from happening in practice.

In order to come up with an algorithm to improve a solution of a linear system $Ax = b$ with the help of (13) we introduce the following notation:

$$\begin{aligned} r_n &= b - Ax_n, \\ e_n &= x - x_n, \end{aligned}$$

so that

$$Ae_n = A(x - x_n) = \underbrace{Ax}_{=b} - Ax_n = r_n.$$

This implies

$$e_n = A^{-1} r_n.$$

With this notation (13) becomes

$$x_{n+1} = x_n + A^{-1} r_n = x_n + e_n,$$

and we can suggest the following algorithm:

Algorithm (Iterative refinement)

```
Solve $Ax = b$ using the LU factorization of $A$ to obtain a starting value $x_0$.
for $n = 0, 1, 2, \dots$ do
    Compute the residual
        $r_n = b - Ax_n$.
    Compute the Newton update
        $e_n = A^{-1} r_n$
    using the LU factorization of $A$, so that only forward and backward substitution are required.
    Update the approximate solution
        $x_{n+1} = x_n + e_n$.
end
```

Remark: This procedure is particularly effective for ill-conditioned systems when the calculation of r_n is done in double precision.

Example: See the Maple worksheet `577_IterativeRefinement.mws`.