4. Direct Methods for Solving Linear Systems of Equations

They are all over the place . . .

4.1. Preliminary Remarks

Linear Systems of Equations

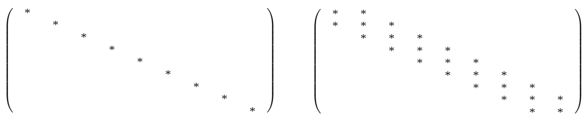
- Another important field of application for numerical methods is **numerical linear algebra** that deals with solving problems of linear algebra numerically (matrix-vector product, finding eigenvalues, solving linear systems of equations).
- Here, the **solution of linear systems of equations**, i.e.

  for $A = (a_{i,j})_{1\leq i,j\leq n} \in \mathbb{R}^{n,n}\,, \quad b = (b_i)_{1\leq i\leq n} \in \mathbb{R}^n\,,$
  find $x \in \mathbb{R}^n$ mit $A \cdot x = b\,,$

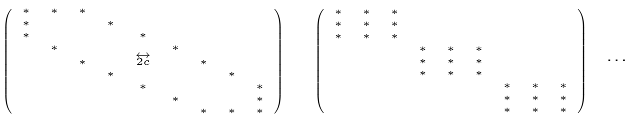
  is of major significance. Linear systems of equations are omnipresent in numerics:

  – interpolation: construction of the cubic spline interpolant (see section 2.3)
  – boundary value problems (BVP) of ordinary differential equations (ODEs) (see chapter 5)
  – partial differential equations (PDEs)
  – ...

- In terms of the problem given, one distinguishes between:
  – **full** matrices: the number of non-zero values in A is of the same order of magnitude as the number of all entries of the matrix, i.e. $O(n^2)$.
  – **sparse** matrices: here, zeros clearly dominate over the non-zeros (typically $O(n)$ or $O(n\log(n))$ non-zeros); those sparse matrices often have a certain **sparsity pattern** (diagonal matrix, tridiagonal matrix ($a_{i,j} = 0$ for $|i-j| > 1$), general band structure ($a_{i,j} = 0$ for $|i-j| > c$) etc.), which simplifies solving the system.



diagonal tridiagonal



band (bandwidth $2c$) block-diagonal

Linear Systems of Equations (2)

- One distinguishes between different solution approaches:
  – **direct** solvers: provide the exact solution x (modulo rounding errors) (covered in this chapter);
  – **indirect** solvers: start with a first approximation $x^{(0)}$ and compute **iteratively** a sequence of (hopefully increasingly better) approximations $x^{(i)}$, without ever reaching x (covered in chapter 6).
- Reasonably, we will assume in the following an *invertible* or *non-singular* matrix A, i.e. det$(A) \neq 0$ or rank$(A) = n$ or $Ax = 0 \Leftrightarrow x = 0$, respectively.
- Two approaches that seem obvious at first sight are considered as **numerical mortal sins** for reasons of complexity:
  – $x := A^{-1}b$, i.e. the explicit computation of the inverse of A;
  – The use of *Cramer's rule* (via the determinant of A and of the n matrices which result from A by substituting a column with the right-hand side b).

- Of course, the following general rule also applies to numerical linear algebra: Have a close look at the way a problem is posed before starting, because even the simple term

$$y \;:=\; A\cdot B\cdot C\cdot D\cdot x\,, \qquad A,B,C,D \in \mathbb{R}^{n,n}\,, \qquad x \in \mathbb{R}^n\,,$$

  can be calculated stupidly via

$$y \;:=\; (((A\cdot B)\cdot C)\cdot D)\cdot x$$

  with $O(n^3)$ operations (matrix-matrix products!) or efficiently via

$$y \;:=\; A\cdot(B\cdot(C\cdot(D\cdot x)))$$

  with $O(n^2)$ operations (**only matrix-vector products!**)!
- Keep in mind for later: Being able to *apply* a linear mapping in form of a matrix (i.e. to be in control of its effect on an arbitrary vector) is generally a lot cheaper than via the explicit construction of the matrix!

Vector Norms

- Concept of norms for vectors and matrices: necessary to analyze the condition of the problem of solving linear systems of equations as well as to analyze the behavior of convergence of iterative methods in chapter 6.
- A **vector norm** is a function $\|.\| : \mathbb{R}^n \to \mathbb{R}$ with the three properties
  – *positivity*: $\|x\| > 0 \;\; \forall x \neq 0$;
  – *homogeneity*: $\|\alpha x\| \;=\; |\alpha|\cdot\|x\|$ for arbitrary $\alpha \in \mathbb{R}$;
  – *triangle inequality*: $\|x+y\| \leq \|x\| + \|y\|$.
- The set $\{x \in \mathbb{R}^n : \;\|x\| = 1\}$ is called **norm sphere** regarding the norm $\|.\|$.
- Examples for vector norms relevant in our context (verify the above norm attributes for every one):
  – **Manhattan norm**: $\|x\|_1 \;:=\; \sum_{i=1}^n |x_i|$
  – **Euclidean norm**: $\|x\|_2 \;:=\; \sqrt{\sum_{i=1}^n |x_i|^2}$ (the common vector length)
  – **maximum norm**: $\|x\|_\infty \;:=\; \max_{1\leq i\leq n} |x_i|$

Matrix Norms

- By extending the concept of vector norm, a **matrix norm** can be defined or rather **induced** according to

$$\|A\| \;:=\; \max_{\|x\|=1} \|Ax\|\,.$$

  In addition to the three properties of a vector norm (rephrased correspondingly), which also apply here, a matrix norm is
  – **sub-multiplicative**: $\|AB\| \leq \|A\|\cdot\|B\|$;
  – **consistent**: $\|Ax\| \leq \|A\|\cdot\|x\|$.
- The **condition number** $\kappa(A)$ is defined as

$$\kappa(A) \;:=\; \frac{\max_{\|x\|=1}\|Ax\|}{\min_{\|x\|=1}\|Ax\|}\,.$$

  – $\kappa(A)$ indicates how strongly the norm sphere is deformed by the matrix A or by the respective linear mapping.
  – In case of the identity matrix I (ones in the diagonal, zeros everywhere else) and for certain classes of matrices there are no deformations at all – in these cases we have $\kappa(A) = 1$.
  – For non-singular A, we have

$$\kappa(A) \;=\; \|A\|\cdot\|A^{-1}\|\,.$$

The Condition of Solving Linear Systems of Equations

- Now, we can begin to determine the condition of the problem of solving a linear system of equations:
  – In

$$(A+\delta A)(x+\delta x) \;=\; b+\delta b,$$

  we now have to deduce the error δx of the result x from the perturbations $\delta A, \delta b$ of the input A, b . Of course, δA has to be so small that the perturbed matrix remains invertible (this holds for example for changes with $\|\delta A\| < \|A^{-1}\|^{-1}$).
  – Solve the relation above for δx and estimate with the help of the sub-multiplicativity and the consistency of an induced matrix norm (this is what we needed it for!):

$$\|\delta x\| \;\leq\; \frac{\|A^{-1}\|}{1-\|A^{-1}\|\cdot\|\delta A\|}\cdot(\|\delta b\| + \|\delta A\|\cdot\|x\|)\,.$$

  – Now, divide both sides by $\|x\|$ and bear in mind that the relative input perturbations $\|\delta A\|/\|A\|$ as well as $\|\delta b\|/\|b\|$ should be bounded by ε (because we assume small input perturbations when analyzing the condition).
  – With this, it follows

$$\frac{\|\delta x\|}{\|x\|} \;\leq\; \frac{\varepsilon\kappa(A)}{1-\varepsilon\kappa(A)}\cdot\left(\frac{\|b\|}{\|A\|\cdot\|x\|}+1\right) \leq \frac{2\varepsilon\kappa(A)}{1-\varepsilon\kappa(A)}$$

  because $\|b\| = \|Ax\| \leq \|A\|\cdot\|x\|$.

The Condition of Solving Linear Systems of Equations (2)

- Because it's so nice and so important, once more the result we achieved for the condition:
$$\frac{\|\delta x\|}{\|x\|} \leq \frac{2\varepsilon\kappa(A)}{1-\varepsilon\kappa(A)}\,.$$
  - The bigger the condition number $\kappa(A)$ is, the bigger our upper bound on the right for the effects on the result becomes, the worse the condition of the problem "solve $Ax = b$" gets.
  - The term "condition number" therefore is chosen reasonably – it represents a measure for condition.
  - Only if $\varepsilon\kappa(A) \ll 1$, which is restricting the order of magnitude of the acceptable input perturbations, a numerical solution of the problem makes sense. In this case, however, we are in control of the condition.
  - At the risk of seeming obtrusive: This only has to do with the problem (i.e. the matrix) and nothing to do with the rounding errors or approximate calculations!
- **Note**: The condition of a problem can often be improved by adequate rearranging. If the system matrix A in $Ax = b$ is ill-conditioned, the condition of the new system matrix MA in $MAx = Mb$ might be improved by choosing a suitable prefactor M.

The Residual

- An important quantity is the **residual** r. For an approximation $\tilde{x}$ of x, r is defined as
$$r := b - A\tilde{x} = A(x-\tilde{x}) =: -Ae$$
with the **error** $e := \tilde{x} - x$.
- **Caution**: Error and residual can be of very different order of magnitude. In particular, from $r = O(\bar{\varepsilon})$ does not at all follow $e = O(\bar{\varepsilon})$ – the correlation even contains the condition number $\kappa(A)$, too.
- Nevertheless the residual is helpful: Namely,
$$r = b - A\tilde{x} \quad \Leftrightarrow \quad A\tilde{x} = b - r$$
shows that $\tilde{x}$ can be interpreted as *exact* result of slightly perturbed input data (A original, instead of b now $b-r$) for small residual; therefore, $\tilde{x}$ is an **acceptable result** in terms of chapter 1!
- We will mainly use the residual for the construction of iterative solvers in chapter 6: In contrast to the unknown error, the residual can easily be determined in every iteration step.

4.2. Gaussian Elimination

The Principle of Gaussian Elimination

- The classical solution method for linear systems of equations, familiar from linear algebra, is **Gaussian elimination**, the natural generalization of solving two equations with two unknowns:
  - Solve one of the n equations (e.g. the first one) for one of the unknowns (e.g. x_1).
  - Replace x_1 by the resulting term (depending on $x_2, \ldots, x_n$) in the other $n-1$ equations – therefore, x_1 is **eliminated** from those.
  - Solve the resulting system of $n-1$ equations with $n-1$ unknowns analogously and continue until an equation only contains x_n, which can therefore be explicitly calculated.
  - Now, x_n is inserted into the elimination equation of x_{n-1}, so x_{n-1} can be given explicitly.
  - Continue until at last the elimination equation of x_1 provides the value for x_1 by inserting the values for $x_2, \ldots, x_n$ (known by now).
- Simply spoken, the elimination means that A and b are modified such that there are only zeros below $a_{1,1}$ in the first column. Note that the new system (consisting of the first equation and the remaining x_1-free equations), of course, is solved by the same vector x as the old one!

The Algorithm

- Those thoughts result in the following algorithm:

<u>Gaussian elimination:</u>

```
for j from 1 to n do
   for k from j to n do u[j,k]:=a[j,k] od;
   y[j]:=b[j];
   for i from j+1 to n do
      l[i,j]:=a[i,j]/u[j,j];
      for k from j+1 to n do a[i,k]:=a[i,k]-l[i,j]*u[j,k] od;
      b[i]:=b[i]-l[i,j]*y[j]
   od
od;
for i from n downto 1 do
   x[i]:=y[i];
   for j from i+1 to n do x[i]:=x[i]-u[i,j]*x[j] od;
   x[i]:=x[i]/u[i,i]
od;
```


Initial system

$$\left(\begin{array}{ccccc|c} 1 & 2 & -2 & -1 & 1 & 13 \\ 2 & 3 & -3 & 2 & 3 & 41 \\ 1 & 2 & 5 & 3 & -2 & 6 \\ 3 & -3 & 2 & 1 & -2 & -11 \\ 1 & 2 & 3 & -1 & 4 & 4 \end{array}\right)$$

Eliminate first column

$$\left(\begin{array}{ccccc|c} 1 & 2 & -2 & -1 & 1 & 13 \\ 0 & -1 & 1 & 4 & 1 & 15 \\ 0 & 0 & 7 & 4 & -3 & -7 \\ 0 & -9 & 8 & 4 & -5 & -50 \\ 0 & 0 & 5 & 0 & 3 & -9 \end{array}\right)$$

Eliminate second column

$$\left(\begin{array}{ccccc|c} 1 & 2 & -2 & -1 & 1 & 13 \\ 0 & -1 & 1 & 4 & 1 & 15 \\ 0 & 0 & 7 & 4 & -3 & -7 \\ 0 & 0 & -1 & -32 & -14 & -185 \\ 0 & 0 & 5 & 0 & 3 & -9 \end{array}\right)$$

Eliminate third column

$$\left(\begin{array}{ccccc|c} 1 & 2 & -2 & -1 & 1 & 13 \\ 0 & -1 & 1 & 4 & 1 & 15 \\ 0 & 0 & 7 & 4 & -3 & -7 \\ 0 & 0 & 0 & -\frac{220}{7} & -\frac{101}{7} & -186 \\ 0 & 0 & 0 & -\frac{20}{7} & \frac{36}{7} & -4 \end{array}\right)$$

Eliminate fourth column

$$\left(\begin{array}{ccccc|c} 1 & 2 & -2 & -1 & 1 & 13 \\ 0 & -1 & 1 & 4 & 1 & 15 \\ 0 & 0 & 7 & 4 & -3 & -7 \\ 0 & 0 & 0 & -\frac{220}{7} & -\frac{101}{7} & -186 \\ 0 & 0 & 0 & 0 & \frac{497}{77} & \frac{142}{11} \end{array}\right)$$

Now do a backward substitution to get the final result:
$$x = U^{-1}y = \begin{pmatrix} 2 & 4 & -3 & 5 & 2 \end{pmatrix}^T$$

Discussion of Gaussian Elimination

- **outer j-loop**: eliminate variables (i.e. cancel subdiagonal columns) one after the other
- **first inner k-loop**: store the part of the row j located in the upper triangular part in the auxiliary matrix U (we only eliminate *below* the diagonal);
store the (modified) right-hand side b_j in y_j;
the values $u_{j,k}$ stored this way as well as the values y_j won't be changed anymore in the following process!
- **inner i-loop**:
  - determine the required factors in column j for the cancellation of the entries below the diagonal and store them in the auxiliary matrix L (here, we silently assume $u_{j,j} \neq 0$ for the first instance)
  - subtract the corresponding multiple of the row j from row i
  - also modify the right side accordingly (such that the solution x won't be changed)
- finally: substitute the calculated components of x backwards (i.e. starting with x_n) in the equations stored in U and y ($u_{i,i} \neq 0$ is silently assumed again)
- Counting the arithmetic operations accurately gives a total effort of
$$\frac{2}{3}n^3 + O(n^2)$$
basic arithmetic operations.

Discussion of Gaussian Elimination (2)

- Together with the matrix A, the matrices L and U appear in the algorithm of Gaussian elimination. We have (cf. algorithm):
  - In U, only the upper triangular part (inclusive the diagonal) is populated.
  - In L, only the strict lower trianguler part is populated (without the diagonal).
  - If filling the diagonal in L with ones, we get the fundamental relation
$$A = L \cdot U\,.$$
  Such a **decomposition** or **factorization** of a given matrix A in factors with certain properties (here: triangular form) is a very basic technique in numerical linear algebra.
- The insight above means for us: Instead of using the classical Gauss elimination, we can solve $Ax = b$ with the triangular decomposition $A = LU$, namely with the algorithm resulting from
$$Ax = LUx = L(Ux) = Ly = b.$$

Algorithm of the LU decomposition

- 1. step: **triangular decomposition**: decompose A into factors L (lower triangular matrix with ones in the diagonal) and U (upper triangular matrix); the decomposition is – with this specification of the respective diagonal values – unique!
- 2. step: **forward substitution**: solve $Ly = b$ (by inserting, from y_1 to y_n)
- 3. step: **backward substitution**: solve $Ux = y$ (by inserting, from x_n to x_1)

In English, this is usually denoted as *LU factorization* with a lower triangular matrix L and an upper triangular matrix U. In German though, this method is denoted as *LR factorization* with a left triangular matrix L and a right triangular matrix R.

LU Factorization

- The previous considerations lead to the following algorithm:

<u>LU factorization:</u>

```
for i from 1 to n do
   for k from 1 to i-1 do
      l[i,k]:=a[i,k];
      for j from 1 to k-1 do l[i,k]:=l[i,k]-l[i,j]*u[j,k] od;
      l[i,k]:=l[i,k]/u[k,k]
   od;
   for k from i to n do
      u[i,k]:=a[i,k];
      for j from 1 to i-1 do u[i,k]:=u[i,k]-l[i,j]*u[j,k] od
   od
od;
for i from 1 to n do
   y[i]:=b[i];
   for j from 1 to i-1 do y[i]:=y[i]-l[i,j]*y[j] od
od;
for i from n downto 1 do
   x[i]:=y[i];
   for j from i+1 to n do x[i]:=x[i]-u[i,j]*x[j] od;
   x[i]:=x[i]/u[i,i]
od;
```


Discussion of LU factorization

- To comprehend the first part (the decomposition into the factors L and U), start with the general formula for the matrix product:

$$a_{i,k} \;=\; \sum_{j=1}^{n} l_{i,j} \cdot u_{j,k}\,.$$

  However, here quite unusually A is known, and L and U have to be determined (the fact that this works is due to the triangular shape of the factors)!
- In case of $i > k$, one only has to sum up to $j = k$, and $l_{i,k}$ can be determined (solve the equation for $l_{i,k}$: everything on the right-hand side is known already):

$$a_{i,k} \;=\; \sum_{j=1}^{k-1} l_{i,j} \cdot u_{j,k} + l_{i,k} \cdot u_{k,k} \quad \text{and, therefore,} \quad l_{i,k} \;:=\; \left(a_{i,k} - \sum_{j=1}^{k-1} l_{i,j} \cdot u_{j,k} \right) / u_{k,k}.$$

- If $i \leq k$, one only has to sum up to $j = i$, and $u_{i,k}$ can be determined (solve the equation for $u_{i,k}$: again, all quantities on the right are already given; note $l_{i,i} = 1$):

$$a_{i,k} \;=\; \sum_{j=1}^{i-1} l_{i,j} \cdot u_{j,k} + l_{i,i} \cdot u_{i,k} \quad \text{and, therefore,} \quad u_{i,k} \;:=\; a_{i,k} - \sum_{j=1}^{i-1} l_{i,j} \cdot u_{j,k}.$$

Discussion of LU Factorization (2)

- It is clear: If you fight your way through all variables with increasing row and column indices (the way it's happening in the algorithm, starting at $i = k = 1$), every $l_{i,k}$ and $u_{i,k}$ can be calculated one after the other – on the right-hand side, there is always something that has already been calculated!
- The forward substitution (second i-loop) follows and – as before at Gaussian elimination – the backward substitution (third and last i loop).
- It can be shown that the two methods "Gauss elimination" and "LU factorization" are identical in terms of carrying out the same operations (i.e. they particularly have the same cost); only the order of the operations is different!

Initial matrix

$$\begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & * & 1 & & \\ * & * & * & 1 & \\ * & * & * & * & 1 \end{pmatrix} \cdot \begin{pmatrix} * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{pmatrix}$$

$i = 1$: Fill first row of U.

$$\begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & * & 1 & & \\ * & * & * & 1 & \\ * & * & * & * & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{pmatrix}$$

$i = 2$: Fill second row of L and U.

$$\begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ * & * & 1 & & \\ * & * & * & 1 & \\ * & * & * & * & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ & -1 & 1 & 4 & 1 \\ & & * & * & * \\ & & & * & * \\ & & & & * \end{pmatrix}$$

$i = 3$: Fill third row of L and U

$$\begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ 1 & 0 & 1 & & \\ * & * & * & 1 & \\ * & * & * & * & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ & -1 & 1 & 4 & 1 \\ & & 7 & 4 & -3 \\ & & & * & * \\ & & & & * \end{pmatrix}$$

$i = 4$: Fill fourth row of L and U.

$$\begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ 1 & 0 & 1 & & \\ 3 & 9 & -\frac{1}{7} & 1 & \\ * & * & * & * & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ & -1 & 1 & 4 & 1 \\ & & 7 & 4 & -3 \\ & & & -\frac{220}{7} & -\frac{101}{7} \\ & & & & * \end{pmatrix}$$

$i = 5$: Fill last row of L and U.

$$\begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ 1 & 0 & 1 & & \\ 3 & 9 & -\frac{1}{7} & 1 & \\ 1 & 0 & \frac{5}{7} & \frac{1}{11} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ & -1 & 1 & 4 & 1 \\ & & 7 & 4 & -3 \\ & & & -\frac{220}{7} & -\frac{101}{7} \\ & & & & \frac{497}{77} \end{pmatrix}$$

Solve linear system with right hand side $b = \begin{pmatrix} 13 & 41 & 6 & -11 & 4 \end{pmatrix}^T$:

- $Ax = b \Leftrightarrow LUx = b \Leftrightarrow x = U^{-1}L^{-1}b$
- First do a forward substitution with L:
  $y = L^{-1}b = \begin{pmatrix} 13 & 15 & -7 & -186 & \frac{142}{11} \end{pmatrix}^T$
- Then do a backward substitution with U:
  $x = U^{-1}y = \begin{pmatrix} 2 & 4 & -3 & 5 & 2 \end{pmatrix}^T$

Discussion of LU Factorization (2)

- In the special case of positive definite matrices A ($A = A^T$ and $x^T A x > 0$ for all $x \neq 0$), this can be accomplished even cheaper than with the algorithm just shown:
  - decompose the factor U of $A = LU$ into a diagonal matrix D and an upper triangular matrix $\tilde{U}$ with ones at the diagonal (this is always possible):

$$A \;=\; L \cdot U \;=\; L \cdot D \cdot \tilde{U} \qquad \text{with } D \;=\; \operatorname{diag}(u_{1,1}, \ldots, u_{n,n})\,;$$

- then, it follows from the symmetry of A

$$A^T \;=\; (L \cdot D \cdot \tilde{U})^T \;=\; \tilde{U}^T \cdot D \cdot L^T \;=\; L \cdot D \cdot \tilde{U} \;=\; A\,,$$

  and the uniqueness of the decomposition forces $L \;=\; \tilde{U}^T$ and thus

$$A \;=\; L \cdot D \cdot L^T \;=:\; \tilde{L} \cdot \tilde{L}^T\,,$$

  if splitting the diagonal factor D in equal shares into the triangular factors ($\sqrt{u_{i,i}}$ in both diagonals; the values $u_{i,i}$ are all positive because A is positive definite).

The Cholesky Factorization

- The method described above, with which the calculation of the $u_{i,k}$, $i \neq k$, in the LU factorization can be avoided and with that about half of the total computing time and required memory, is called **Cholesky factorization** or **Cholesky decomposition**. We write $A = LL^T$.

  Cholesky factorization:

```
for k from 1 to n do
   l[k,k]:=a[k,k];
   for j from 1 to k-1 do l[k,k]:=l[k,k]-l[k,j]^2 od;
   l[k,k]:=(l[k,k])^0.5;
   for i from k+1 to n do
      l[i,k]:=a[i,k];
      for j from 1 to k-1 do l[i,k]:=l[i,k]-l[i,j]*l[k,j] od;
      l[i,k]:=l[i,k]/l[k,k]
   od
od;
```

- Of course, the algorithm above only delivers the triangular decomposition. As before, forward and backward substitution still have to be carried out to solve the linear system of equations $Ax = b$.

Discussion of the Cholesky Factorization

- At the construction of the Cholesky algorithm, we once more start with the formula for the matrix product:

$$a_{i,k} \;=\; \sum_{j=1}^{n} l_{i,j} \cdot l_{j,k}^T \;=\; \sum_{j=1}^{k} l_{i,j} \cdot l_{j,k}^T \;=\; \sum_{j=1}^{k} l_{i,j} \cdot l_{k,j}\,, \qquad i \geq k\,.$$

- From this, calculate L (i.e. the lower triangular part) column by column, starting in every column with the diagonal element

$$a_{k,k} \;=\; \sum_{j=1}^{k} l_{k,j}^2\,, \qquad l_{k,k} \;:=\; \sqrt{a_{k,k} - \sum_{j=1}^{k-1} l_{k,j}^2}\,,$$

  and then process the remaining rows $i > k$:

$$a_{i,k} \;=\; \sum_{j=1}^{k} l_{i,j} \cdot l_{k,j}\,, \qquad l_{i,k} \;:=\; \left(a_{i,k} - \sum_{j=1}^{k-1} l_{i,j} \cdot l_{k,j} \right) / l_{k,k}\,.$$

- As mentioned earlier, the cost decreases to about half, i.e.

$$\frac{1}{3} n^3 + O(n^2)\,.$$

- To close this chapter, we now turn to the division by the diagonal element, so far always silently assumed to be possible.

4.3. Choice of Pivot

Pivots

- In the algorithm just introduced, we assumed that divisions of the kind $a_{i,j}/u_{j,j}$ or $x_i/u_{i,i}$ do not cause any problems, i.e. particularly that there occur no zeros in the diagonal of U.
- As everything centers on those values in the diagonal, they are called **Pivots** (French word).
- In the positive definite case, all eigenvalues are positive, which is why zeros are impossible in the diagonal – i.e. in the Cholesky method everything is alright.
- However, in the general case, the requirement $u_{j,j} \neq 0$ for all j is not granted. If a zero emerges, the algorithm has to be modified, and a feasible situation, i.e. a non-zero in the diagonal, has to be forced by permutations of rows or columns (which is possible, of course, when A is not singular!). Consider for example the matrix

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 1 & 1 \end{pmatrix}.$$

- A possible partner for exchange of a zero $u_{i,i}$ in the diagonal can be found either in the column i below the diagonal (**column pivot search**) or in the entire remaining matrix (everything from the row and the column $i+1$ onward, **total pivot search**).

Pivot Search

- **Partial pivoting** or **column pivot search**:
  If $u_{i,i} = 0$, then one has to search in the column i below the row i for an entry $a_{k,i} \neq 0$, $k = i+1, \ldots, n$:
  - If there is no such $a_{k,i} \neq 0$: The remaining matrix has a first column that completely vanishes, which means $\det(A) = 0$ (case of error).
  - If there are non-zeros: Then usually the element with the biggest absolute value (let it be $a_{k,i}$) is chosen as pivot and the rows k and i in the matrix and on the right-hand side are switched.
  - Big pivots are convenient because they lead to small elimination factors $l_{k,i}$ and they do not increase the entries in L and U too much.
  - Of course, such a switching of rows does not change the system of equations and its solution at all!
- **Total pivoting** or **total pivot search**:
  Here, one searches not only in the column i of the remaining matrix but in the total remaining matrix, instead of an exchange of rows also exchanges of columns (rearranging of the unknowns x_k) can be realized here; total pivoting therefore is more expensive.
- Even if no zeros occur – for numerical reasons, pivoting is always advisable!

4.4. Applications for Direct Solving Methods

Linear Systems of Equations

- From the multitude of problems that lead to solving a linear system of equations, we pick one: the so called **radiosity method** in computer graphics with which photo-realistic images can be produced. Radiosity is particularly suited for ambient light (fume, fog, dust in sunlight . . .).
  - The image generation is carried out in four steps: Division of the entire surface of the scene into several *patches*, computation of the *form factors* between the patches (they describe the light flow), setting up and solving of the radiosity system of equations (aha!) with the radiosity values (energy densities) as variables, rendering of the patches with the calculated radiosity values.
  - For the **radiosity** B_i of the patch i, the relation

$$B_i \;=\; E_i + \varrho_i \cdot \sum_{j=1}^{n} B_j F_{ij}\,,$$

    holds, where E_i denotes the emissivity of the patches (how much light is newly produced and emitted – especially important for light sources), ϱ_i the absorption and F_{ij} the form factor. In short: On patch i there is the light which is produced (emitted) there or has arrived from other patches.

  - For F_{ij} the relations

$$F_{ij} \;=\; \frac{\cos\theta_i \cos\theta_j}{\pi r^2} A_j V_{ij}\,, \qquad F_{ii} \;=\; 0\,, \qquad \sum_{j=1}^{n} F_{ij} \;=\; 1$$

    hold. Here, θ_i denotes the angle between the normal of patch i and the vector of length r which connects the centers of the patches i and j. The term A_j denotes the area of patch j, and V_{ij} indicates the visibility (1 for free sight from patch i to patch j, 0 else).

- The relation above obviously forms a linear system of equations – n equations with n variables B_i.