Solving Linear Systems (Direct Methods)

Recommended reading:

- Lectures in [4]: 6,7,8 for QR; 20,21 for LU; 23 for $LL^\top$
- Section I.4 in [3]
- Sections 2.6, 2.7 in [2]

References

[1] R. Rannacher.
Numerik 0 - Einführung in die Numerische Mathematik.
Heidelberg University Publishing, 2017.

[2] G. Strang.
Introduction to Linear Algebra.
Wellesley-Cambridge Press, 2003.

[3] G. Strang.
Linear Algebra and Learning from Data.
Wellesley-Cambridge Press, 2019.

[4] L.N. Trefethen and D. Bau.
Numerical linear algebra.
SIAM, Soc. for Industrial and Applied Math., Philadelphia, 1997.

2 Solving Linear Systems with Direct Methods

Aim:

$$\textit{Given } A \in \mathbb{R}^{m \times n} \textit{ } (m \neq n \textit{ possible) and } b \in \mathbb{R}^m, \textit{ find } x \in \mathbb{R}^n \textit{ such that}$$
$$Ax = b.$$

2.1 The Idea of "Factor and Solve"

A general theme in numerical mathematics is to reduce the general (possibly complicated) problem to one or more simpler problems with the help of transformations for which the property of interest is an invariant.

Separate: Factorization and Solution!

- Since the *factorization* (*elimination/triangularization/orthogonalization*) step is typically much more expensive than the *solution* step, it makes sense to perform them separately, if the same matrix A is used for multiple right-hand sides b.

- The number of factors in a decomposition is the number of systems to be solved in the *solution* step.

  From the factors we may also gain information about the

  - the image and kernel of A
  - invertibility of the matrix A in case $m = n$
  - solvability of the system (i.e., the cardinality $|S|$ of the solution set S)

- We will have a look at the following decompositions
  - $A = QR$ (two systems to be solved)
  - $A = P^T LU$ (three systems to be solved)
  - $A = LL^T$ (two systems to be solved)

Never compute the inverse!

- There are only very rare occasions, where an inverse matrix A^{-1} is needed. In most cases, one needs inverse matrix times a vector, i.e., $A^{-1}b$.
- For computing the inverse of an $(n \times n)$-matrix A we have to solve n linear systems:

- Thus, never do

```
x = inv(A) @ b
```

  instead of

```
x = solve(A, b).
```

2.2 The Gram-Schmidt Algorithm and the QR decomposition

2.2.1 Projectors

For a fixed vector $x \in \mathbb{R}^n \setminus \{0\}$ we have derived the orthogonal projection onto its span by

$$\text{proj}_x(y) = \frac{x}{\|x\|_2} \cdot \frac{x^\top y}{\|x\|_2} = \frac{xx^\top}{x^\top x} \cdot y =: P_x y,$$

where $P_x := \frac{xx^\top}{x^\top x} \in \mathbb{R}^{n \times n}$.

For any projector $P \in \mathbb{R}^{n\times n}$ we can show:

Lemma 2.3 (*Projector Properties*) *Let $P \in \mathbb{R}^{n\times n}$ be such $P^2 = P$, then*

i) $\operatorname{Im} P = \ker(I - P)$,
ii) $\ker(P) \cap \ker(I - P) = \{0\}$,
iii) $(I - P)$ *is also a projector,*
iv) $\forall y \in \mathbb{R}^n \ \exists_1 v \in \operatorname{Im}(P), r \in \operatorname{Im}(I - P)\colon \ y = v + r$,
with other words, the mapping $\operatorname{Im}(P) \times \operatorname{Im}(I - P) \to \mathbb{R}^n, (v, r) \mapsto v + r$ is bijective.

Remark:

In view of Lemma 2.3 iv) we say that $\mathbb{R}^n$ is the direct sum of the subspaces $\mathrm{Im}(P)$ and $\mathrm{Im}(I-P)$. Each vector $y \in \mathbb{R}^n$ uniquely splits into the additive components v and r. Due to i) and iii) of this lemma, also the same is true for the subspaces $\mathrm{Im}(P)$ and $\ker(P)$.

Orthogonal Projection with Orthonormal Basis

Let us now consider more than just one vector. In fact, let $q_1, \ldots, q_r \in \mathbb{R}^m$ be orthonormal; thus $r \leq m$. Then let us define the matrices

$$\widehat{Q} := [q_1, \ldots, q_r] \in \mathbb{R}^{m \times r},$$
$$P := P_{q_1,\ldots,q_r} := \widehat{Q}\widehat{Q}^\top \in \mathbb{R}^{m \times m}.$$

Attention: For $r < m$, $\widehat{Q}$ is not an orthogonal matrix and thus $P = \widehat{Q}\widehat{Q}^\top \neq I$ in general. However, for the Gramian matrix which collects all possible pairs of inner products we have $\widehat{Q}^\top \widehat{Q} = I$ ($\widehat{Q}^\top$ is only a left-inverse).

Example 2.6 Let us consider:

$$q_1 = \frac{1}{\sqrt{2}}(1,1,0)^\top, q_2 = \frac{1}{\sqrt{2}}(-1,1,0), y = (0,1,1)^\top \in \mathbb{R}^3$$

Orthogonal Projection with Arbitrary Basis

Let $a_1, \ldots, a_n$ be linearly independent vectors in $\mathbb{R}^m$ $(m \geq n)$ and let us put the vectors into a matrix $A = [a_1, \ldots, a_n] \in \mathbb{R}^{m \times n}$.
How to define an orthogonal projector on the image of A?

Example 2.7 Let us consider:

$$a_1 = (1,1,0)^\top, a_2 = (0,1,0), y = (0,1,1)^\top \in \mathbb{R}^3.$$

We observe that $\mathrm{Im}(A) = \mathrm{Im}(\widehat{Q})$, so that we would expect the same projector as in the example above.

2.2.2 A=QR from the (classical) Gram–Schmidt Algorithm

Classical Gram–Schmidt Orthogonalization Algorithm

$r_{11} = \|a_1\|$
$q_1 = \frac{a_1}{r_{11}}$

for $k = 2, \ldots, n$ **do**
  for $\ell = 1, \ldots, k-1$ **do**
    $r_{\ell k} = a_k^\top q_\ell$
  end
  $\widetilde{q}_k = a_k - \sum_{\ell=1}^{k-1} r_{\ell k}\, q_\ell \;\; = (I - P_{q_1,\ldots,q_{k-1}})a_k \in \mathrm{Im}(\widehat{Q}_{k-1})^\perp$
  $r_{kk} = \|\widetilde{q}_k\|$
  if $r_{kk} = 0$ **then**
    pick arbitrary $q_k \in \mathrm{Im}(\widehat{Q}_{k-1})^\perp = \ker(\widehat{Q}_{k-1}^\top)$
    // for example by solving $\widehat{Q}_{k-1}^\top q_k = 0$ and normalizing
  end
  else
    $q_k = \frac{\widetilde{q}_k}{r_{kk}}$
  end
end
// For full QR:
Fill up columns of $\widehat{Q} := \widehat{Q}_n$ with $(m-n)$ orthonormal vectors of $\ker(\widehat{Q}_{k-1}^\top)$
Fill up rows of $\widehat{R} := (r_{ij})$ with $(m-n)$ zero rows

Observation: This algorithm successively orthogonalizes the columns of A!

These ideas lead to

Theorem 2.8 (*QR decomposition*) *Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, then there exists a matrix $\widehat{Q} \in \mathbb{R}^{m\times n}$ with orthonormal columns and an upper triangular matrix $\widehat{R} \in \mathbb{R}^{n\times n}$ such that*

$$A = \widehat{Q}\widehat{R}.$$

*We call this the **reduced QR decomposition** of A.*

One can extend the columns of $\widehat{Q}$ with orthonormal columns to obtain an orthogonal matrix $Q \in \mathbb{R}^{m\times m}$ and the rows of $\widehat{R}$ by zero rows to obtain a matrix $R = \begin{pmatrix} \widehat{R} \\ 0 \end{pmatrix} \in \mathbb{R}^{m\times n}$ and thereby obtain

$$A = QR.$$

*We call this the **QR decomposition** of A.*

Example 2.9

Let $m \geq n$.

(1) Factorization
Gram-Schmidt orthogonalization algorithm, Householder reflections or Givens Rotations can be used.

In Python

$$Q, R = \texttt{scipy.linalg.qr(A)}$$

or for the reduced version run

$$\widehat{Q}, \widehat{R} = \texttt{scipy.linalg.qr(A, mode="economic")}$$

(2) Solving

2.3 Gaussian Elimination and the LU Decomposition

(1) Factorization: Row elementary operations

Apply transformations to A (and b), which do not affect the solution x (more precisely the solution set S will not change), to bring A into a simple form and collect all transformations on the fly.

- the factorization **process** is known as *Gaussian Elimination*
- the **result** will be an invertible lower triangular matrix L, some sort of upper triangular matrix U and an orthogonal (more precisely a permutation) matrix P, such that

$$A = P^T LU.$$

(2) Solve: Forward/Backward substitution

$$Ax = b \Leftrightarrow Ux = L^{-1}Pb$$

Example 1: Frobenius Matrices

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 0 & 2 & 3 \\ 2 & 4 & 6 \end{pmatrix},\ b = \begin{pmatrix} 4 \\ 2 \\ 6 \end{pmatrix} \qquad Ax = b \quad \Leftrightarrow \quad \begin{array}{rrrl} x_1 & +3x_2 & +5x_3 & = 4 \\ & 2x_2 & +3x_3 & = 2 \\ 2x_1 & +4x_2 & +6x_3 & = 6 \end{array}$$

Frobenius Matrices

Properties

Lemma 2.10 *Let $\ell_j := (0,\dots,0,\ell_{j+1,j},\dots,\ell_{m,j})^\top \in \mathbb{R}^m$, $e_j \in \mathbb{R}^m$ be the j-th unit vector and $I \in \mathbb{R}^{m\times m}$ be the identity matrix. Then the matrix*

$$L_j := I + \ell_j e_j^\top \in \mathbb{R}^{m\times m}$$

satisfies:

i) *The matrix L_j is an invertible lower triangular matrix.*

ii) *The inverse of L_j is given by $L_j^{-1} = I - \ell_j e_j^\top \in \mathbb{R}^{m\times m}$.*

iii) *For $i \le j$ it holds that $L_i L_j = I + \ell_j e_j^\top + \ell_i e_i^\top$ and $L_i^{-1} L_j^{-1} = I - \ell_j e_j^\top - \ell_i e_i^\top$.*

Example 2: Permutation Matrices

Permutation Matrices

Definition 2.11 (*Permutation Matrix*) *A matrix $P \in \mathbb{R}^{m \times m}$ is called **permutation matrix**, if it has exactly one entry "1" in each row and column and zero otherwise.*

In each step of the Gaussian elimination (with pivoting) we only swap two rows at a time: The matrix

$$P_{ik} = \begin{pmatrix} 1 & 0 & & & & \cdots & & & & 0 \\ 0 & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & 0 & & \cdots & 0 & 1 & & \\ & & & & 1 & & & 0 & & \vdots \\ \vdots & & & \vdots & & \ddots & & \vdots & & \\ & & & 0 & & & 1 & & & \\ & & & 1 & 0 & \cdots & & 0 & & \\ & & & & & & & & \ddots & 0 \\ 0 & & & & \cdots & & & & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \leftarrow\ k-th \\ \\ \\ \end{matrix}$$

swaps rows k and i, when multiplied from the left to a matrix.

$\rightarrow$ *We illuminate further properties in the exercises.*

Algorithm: Elimination with row pivoting

In-place Gaussian Elimination with Row Pivoting (for $m = n$)

INPUT: $A \in \mathbb{R}^{n \times n}$ (and $b \in \mathbb{R}^n$)
OUTPUT: LU decomposition $PA = LU$ (and if $Ax = b$ is uniquely solvable the solution $x \in \mathbb{R}^n$)

```
# FACTORIZATION
initialize piv = [1,2,...,n]
for j = 1,...,n − 1 do
    # Find the j-th pivot
    k_j := arg max_{k≥j} |a_kj|
    if a_{k_j j} ≠ 0 then
        # Swap rows
        A[k_j,:] ↔ A[j,:]
        # by hand we also transform b on the fly
        b[k_j] ↔ b[j]
        piv[k_j] ↔ piv[j]
        # Elimination
        for k = j + 1,...,n do
            ℓ_kj := a_kj / a_jj
            a_kj = ℓ_kj
            for i = j + 1,...,n do
                a_ki = a_ki − ℓ_kj a_ji
            end
            # by hand we also transform b on the fly
            (b_k = b_k − ℓ_kj b_j)
        end
    end
end
# SOLVE
...
```

As in the algorithm consider $A \in \mathbb{R}^{n\times n}$. Then the algorithm eventually leads to

$$U := A^{(n)} = L_{(n-1)}P_{(n-1)k_{(n-1)}} \dots L_3 P_{3k_3} L_2 P_{2k_2} L_1 P_{1k_1} A.$$

By construction of the algorithm, $A^{(n)} =: U$ has row echelon form (no rigorous proof provided in this course).

Question: Can we group all L_i and P_{ik_i} in such a way that $PA = LU$?

Lemma 2.13 *Let $m \in \mathbb{N}$. Let $P_{ik_i} \in \mathbb{R}^{m\times m}$ be the permutation matrix which results from interchanging the i-th and k_i-th column of the identity matrix in $\mathbb{R}^{m\times m}$, where $k_i \geq i$. Further for $\ell_j := (0,\dots,0,\ell_{j+1,j},\dots,\ell_{m,j})^\top \in \mathbb{R}^m$ and the j-th unit vector $e_j \in \mathbb{R}^m$, let $L_j := I + \ell_j e_j^\top \in \mathbb{R}^{m\times m}$. Then show that for all $1 \leq j < i \leq k_i \leq m$ we have*

$$P_{ik_i} L_j = \widehat{L}_j P_{ik_i}$$

where $\widehat{L}_j := I + (P_{ik_i}\ell_j)e_j^\top$.

By applying this result multiple times, we obtain

Summary: **Row echelon form and LU decomposition**

Definition 2.14 (*REF*)

a) ***Row elementary operations*** *are*
   1) *add a nonzero multiple of one row to another*
   2) *swap two rows*

b) *A matrix is in* ***row echelon form (REF)*** *when it satisfies the following conditions:*
   1) *The first non-zero element in each row (called the leading entry) is in a column to the right of the leading entry in the previous row.*
   2) *Rows with all zero elements, if any, are below rows having a non-zero element.*

By applying these types of operations we find:

Theorem 2.15 (*LU Decomposition*) *Every matrix $A \in \mathbb{F}^{m\times n}$ can be transformed to REF by row elementary operations. Thus there is a matrix $U \in \mathbb{F}^{m\times n}$ in REF, a lower triangular matrix $L \in GL(m,\mathbb{F})$ with $\ell_{ij} = 0, \forall j > i$, and $\ell_{ii} = 1, \forall i$, and a permutation matrix $P \in GL(m,\mathbb{F})$ with exactly one entry "1" in each row and column and zero otherwise, such that*

$$P \cdot A = L \cdot U.$$

Since triangular matrices are invertible if and only if the diagonal elements are nonzero, we find for the square case $m = n$, that:

Corollary 2.16 (*Invertibility of A*) *A matrix $A \in \mathbb{F}^{n\times n}$ is invertible, if and only if its REF $U \in \mathbb{F}^{n\times n}$ has a nonzero diagonal, i.e, $u_{ii} \neq 0, \forall i = 1, \ldots, n$.*

(1) Factorization: `Lu, Piv = scipy.linalg.lu_factor(A)`

- determine factors L, U and permutation matrix P such that $LU = PA$
- the factors L, U are compactly stored in one matrix `Lu` $\in \mathbb{R}^{n \times n}$ of the same size as A and the permutation matrix P in CSR format as integer vector `Piv` $\in \mathbb{N}^n$.

(2) Solution: `x = scipy.linalg.lu_solve((Lu, Piv), b)`

(2.1) permute right-hand side $\bar{b} = Pb$

(2.2) solve lower triangular system $Lz = \bar{b}$ for z (forward substitution)

(2.3) solve REF system $Ux = z$ for x (backward substitution)

Both, (1) and (2), are combined in the routine

`x` $=$ `scipy.linalg.solve(A,b)`.

2.3.1 Identify the number of solutions from the REF

2.4 The Cholesky Decomposition

For symmetric and positive definite matrices $A \in \mathbb{R}^{n\times n}_{\text{spd}} \subset \mathrm{GL}_n(\mathbb{R})$ we obtain the following improvement:

Theorem 2.17 *We have the equivalence*

$$A \in \mathbb{R}^{n\times n}_{spd} \quad \Leftrightarrow \quad \exists_1\, L \in \mathbb{R}^{n\times n} \text{lower triangular, } \ell_{ii} > 0\colon \quad A = LL^\top.$$

- The Decomposition $A = LL^\top$ is called the **Cholesky decomposition of** A.
- Named after the french Mathematician André-Louis Cholesky (1875-1918) who developed this decomposition during his surveying work to solve the normal equation $A^\top Ax = A^\top b$.
- We can derive an algorithm to compute the factor L.

- **Solving using** $A = LL^\top$

$$Ax = b \quad \Leftrightarrow \quad LL^\top x = b \quad \Leftrightarrow \quad Lz = b, \quad L^\top x = z \quad \text{(forward/backward Subst.)}$$

In Python:

(1) Factorization: `L, lower = scipy.linalg.cho_factor(A)`

(2) Solution: `x = scipy.linalg.cho_solve((L, lower), b)`

Numerical Comparison: LU vs. Choleskly

- **Computational Costs:**
  The Cholesky decomposition is roughly twice as fast as Gaussian elimination (in terms of number of floating point operations). Clearly, we only need to compute one instead of two factors.

- **Stability** (="robustness against rounding errors") :
  - Gaussian Elimination: Is only stable if a pivoting strategy is applied.
  - Cholesky: Is stable even without pivoting.
    (To avoid square roots, one can compute the LDL decomposition)

All in all:

For symmetric and positive definite matrices (of moderate size),
the Cholesky factorization is the preferred algorithm!