Numerical and Symbolic Algorithms Modeling (MODEL, MU4IN901)

Jérémy Berthomieu, Vincent Neiger and Mohab Safey El Din

Introduction

Unless every attending student speaks French, the course will be given in English.

This course is taught on the Friday from 8:30 to 10:30 in different rooms depending on the week. by:

- Jérémy Berthomieu (in charge), jeremy.berthomieu@lip6.fr;

The tutorial are supervised

- on the Monday from 8:30 to 10:30 in different rooms depending on the week and from 10:45 to 12:45 in different rooms depending on the week by
  - Kevin Tran, kevin.tran@lip6.fr and Robin Kouba robin.kouba@lip6.fr.
- on the Wednesday from 13:45 to 15:45 in different rooms depending on the week and from 16:00 to 18:00 in different rooms depending on the week by
  - Dimitri Lesnoff, dimitri.lesnoff@lip6.fr.

The evaluation is done through:

- short multiple-choice questions (around 4 during the semester), evaluating how you learnt the course (20% of the final grade);
- one implementation project (20% of the final grade);
- one mid-term exam evaluating your ability to manipulate the concepts taught during the first half of the semester (30% of the final grade);
- one final exam evaluating your ability to manipulate the concepts taught during the whole semester but with an emphasis on the second half (30% of the final grade).

Objectives of the course and outline.

The course provides an introduction to fundamental linear algebra techniques with *approximate* or *exact* computation. This finds application in many areas of computer science (cryptography, high-performance computing, big data, operational research, imagery, etc.).

1. Linear system solving, Gaussian elimination and PLUQ decomposition
2. Approximate linear system solving and QR decomposition

3. Matrix and vector compression, SVD and FFT algorithms
4. Complexity, non-naive polynomial multiplication algorithms
5. Non-naive matrix multiplication algorithms
6. Reduction of linear algebra operations to PLUQ decomposition
7. Solving sparse and structured linear systems

Contents

I. Exact linear system solving and Gaussian elimination

We first recall floating-point arithmetic. Then, the goal is to solve dense linear systems.

I.1. Computer arithmetics

A normalized floating-point number $x \in \mathbb{F}$ is a number

$$x = \pm x_0 . \underbrace{x_1 \dots x_{p-1}}_{\text{mantissa}} \times b^e, \; 0 \le x_i \le b-1, \; x_0 \neq 0$$

where b is the base, p the precision and e the exponent with $e_{\min} \le e \le e_{\max}$.

The machine precision is $\varepsilon = b^{1-p}$. Any $x \in \mathbb{R}$ can be approximated by $\mathrm{fl}(x) = x(1+\delta)$, with $\delta \le \mathbf{u}$. The unit round-off $\mathbf{u}$ equals $\frac{\varepsilon}{2}$ when rounding to nearest.

Arithmetic operations $(+, -, \times, /, \sqrt{})$ are performed as if they were computed with infinite precision before rounding the result. That is $\mathrm{fl}(x \circ y) = (x \circ y)(1+\delta)$ with $\delta \le \mathbf{u}$ for $\circ \in \{+, -, \times, /\}$.

Type	Size	Mantissa	Exponent	Unit round-off	Interval
Simple	32 bits	23 + 1 bits	8 bits	$\mathbf{u} = 2^{-24} \approx 5.96 \times 10^{-8}$	$\approx 10^{\pm 38}$
Double	64 bits	52 + 1 bits	11 bits	$\mathbf{u} = 2^{-53} \approx 1.11 \times 10^{-16}$	$\approx 10^{\pm 308}$

The arithmetic is closed: every operation returns a result. NaN (Not a Number) is generated by computations such as $\frac{0}{0}$, $0 \times \infty$, $\frac{\infty}{\infty}$, $\infty - \infty$ and $\sqrt{-1}$.

Infinities and zeroes satisfies sign rules: $\infty + \infty = \infty$, $(-1) \times \infty = -\infty$, $\frac{1}{\infty} = +0$.

Each time we perform an operation, we lose precision: we say we have rounding errors. The two main sources of rounding errors are cancellation and absorption.

There are four types of rounding modes

- toward 0: truncation, it is similar to the common behavior of float-to-integer conversions, which convert -3.9 to -3 and 3.9 to 3;
- toward $+\infty$: rounding up;
- toward $-\infty$: rounding down;
- to nearest: with ties rounding to nearest even digits in the required position (default).

Problem I.1. *Give the floating-point numbers representing $1/3$ and $-1/5$ in simple precision for the 4 rounding modes.*

Problem I.2. *Let us recall that for x small enough, $\sqrt{1+x} \approx 1 + \frac{x}{2}$, or even $\sqrt{1+x} \approx 1 + \frac{x}{2} - \frac{1}{8}x^2$.*

1. *Let us assume that we have a C function* `double S(double x)` *computing $\sqrt{1+x}$ for a small x using these approximations. We implement the function $f(x) = \sqrt{1+x} - 1$ using* `S`*. What does the call $f(2^{-53})$ return with rounding towards to nearest?*

2. *Write $f(x)$ as a quotient without any subtraction.*

3. *Compute $f(2^{-53})$ using this new form and the corresponding implementation of the function* `S`*. What can be noticed?*

I.2. Linear system solving

We let $\mathbb{K}$ denote $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$.

Assume that the linear system $Ax = b$, with $A \in \mathbb{K}^{n\times n}$ and $b \in \mathbb{K}^n$ we want to solve has a upper triangular shape

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ & a_{1,1} & \cdots & a_{1,n-1} \\ & & \ddots & \vdots \\ & & & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

Then, if $a_{n-1,n-1} \neq 0$, $x_{n-1} = \frac{b_{n-1}}{a_{n-1,n-1}}$.

Theorem I.1. *An upper triangular system $Ax = b$ with $A \in \mathbb{K}^{n\times n}$ and $b \in \mathbb{K}^n$ has a unique solution if, and only if, $a_{0,0} \cdots a_{n-1,n-1} \neq 0$.*

Problem I.3. *Prove this theorem using the reasoning above.*

Problem I.4. *Practice solving triangular systems of size 4 or 5 over $\mathbb{Q}$.*

From this reasoning, we get the following iterative algorithm for solving an upper triangular system.

```
Algorithm 1: UpperTriangularSystemAlgorithm
Input: An upper triangular matrix $A = (a_{i,j})_{0\le i,j\le n-1}$ with coefficients in $\mathbb{K}$, with
       nonzero diagonal coefficients, and a vector $b = (b_i)_{0\le i\le n-1}$, with coefficients in $\mathbb{K}$.
Output: The vector $x = (x_i)_{0\le i\le n-1}$, with coefficients in $\mathbb{K}$, such that $Ax = b$.
For $i$ from $n-1$ to $0$ do $x_i := b_i$
For $i$ from $n-1$ to $0$ do
    $x_i := \frac{x_i}{a_{i,i}}$
    For $j$ from $i-1$ to $0$ do $x_j := x_j - x_i a_{j,i}$
Return $x$
```

Theorem I.2. *Algorithm 1,* UpperTriangularSystemAlgorithm, *is correct and requires at most $\frac{n^2+n}{2}$ multiplications in $\mathbb{K}$.*

The main advantage of upper triangular systems is that solving them comes down to solving, one by one, linear equations in one variable. Indeed, in the last equations $x_0, \dots, x_{n-2}$ do not appear, allowing us to solve for x_{n-1}. Then, plugging this value in the equation above and using the fact that $x_0, \dots, x_{n-3}$ do not appear, we only need to solve a linear equation in x_{n-2}. And so on, and so forth.

The idea of *Gaussian elimination* is, at its name suggests, to eliminate variables from some equations to recover such a beneficial situation. That is, we want to reduce the linear system to an upper triangular one. This is done through a type of *elementary operation*: adding to a line the product of another line above with an element of $\mathbb{K}$. This leads to the following algorithm

```
Algorithm 2: GaussianEliminationAlgorithm
Input: A matrix $A = (a_{i,j})_{0 \le i,j \le n-1}$, with coefficients in $\mathbb{K}$, and a vector $b = (b_i)_{0 \le i \le n-1}$,
       with coefficients in $\mathbb{K}$.
Output: An upper triangular matrix and a vector yielding an equivalent system as the
        input or “No unique solution”.
For $i$ from $0$ to $n-1$ do
    $j := i$
    While $j < n \wedge a_{j,i} = 0$ do $j := j+1$
    If $j = n$ then Return “No unique solution”.
    $t_{i..n-1} := a_{i,i..n-1}$, $u := b_i$
    $a_{i,i..n-1} := a_{j,i..n-1}$, $b_i := b_j$
    $a_{j,i..n-1} := t_{i..n-1}$, $b_j := u$
    For $j$ from $i+1$ to $n-1$ do
        $b_j := b_j - a_{j,i} b_i / a_{i,i}$
        $a_{j,i..n-1} := a_{j,i..n-1} - a_{j,i} a_{i,i..n-1} / a_{i,i}$
Return $A$, $b$
```

Theorem I.3. *Algorithm 2,* GaussianEliminationAlgorithm, *is correct and solves the input linear system of size n using approximatively $\frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2}$ operations in $\mathbb{K}$.*

Problem I.5.

1. *Check that, for the following input matrix and vector,*

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}, \quad \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix},$$

GaussianEliminationAlgorithm *returns*

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 0 & \frac{1}{10} & \frac{2}{5} & \frac{1}{10} \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}, \quad \begin{pmatrix} 32 \\ \frac{3}{5} \\ 5 \\ \frac{1}{2} \end{pmatrix}.$$

2. *Finish solving the system.*

Problem I.6. *Practice solving linear systems.*

I.3. Determinant

Definition I.4. *Let $A = (a_{i,j})_{0 \le i,j \le n-1} \in \mathbb{K}^{n \times n}$. The* determinant *of A is*

$$\det A = \sum_{\sigma \in \mathfrak{S}_n} (-1)^{\varepsilon(\sigma)} a_{0,\sigma(0)} \cdots a_{n-1,\sigma(n-1)},$$

where $\mathfrak{S}_n$ is the set of bijection from $\{0, \dots, n-1\}$ to itself and for $\sigma \in \mathfrak{S}_n$,

$$\varepsilon(\sigma) = \# \{(i,j) | 0 \le i < j \le n-1,\ \sigma(i) > \sigma(j)\}.$$

Problem I.7.

1. *Expand this sum for matrices of sizes 1, 2, 3 and 4.*

2. *What about matrices of size 5?*

3. *How many terms does this sum have for matrices of size n?*

Proposition I.5. *Let A and B two matrices in $\mathbb{K}^{n \times n}$,*

1. *then* $\det(AB) = \det A \det B$*;*

2. *let $A_{i,j}$ be the submatrix obtained by removing from A its ith row and jth column, then* Laplace's expansion *is*

$$\forall 0 \le i \le n-1,\ \det A = \sum_{0 \le j \le n-1} (-1)^{i+j} a_j \det A_{i,j};$$

3. *then $\det A = 0$ if, and only if, A is a singular matrix, that is its rows (resp. columns) are not linearly independent;*

4. *let A^{T} be the transposed matrix of A, then $\det A^{\mathrm{T}} = \det A$.*

Problem I.8. *Prove these properties.*

Problem I.9. *How many recursive calls does Laplace's method perform?*

Theorem I.6. *Let $T = (t_{i,j})_{0 \le i,j \le n-1}$ be a triangular matrix. Then,*

$$\det T = \prod_{i=0}^{n-1} t_{i,i}.$$

Problem I.10. *Prove Theorem I.6.*

Determinants can be useful to solve linear system. Let $b, x \in \mathbb{K}^{n \times 1}$.

Theorem I.7. *The linear system $Ax = b$ has a unique solution x if, and only if,* $\det A \neq 0$. *In that case,* Cramer's formula *ensures that*

$$x_i = \frac{\det B_i}{\det A}, \quad \text{where } B_i = \begin{pmatrix} a_{0,0} & \cdots & a_{0,i-1} & b_0 & a_{0,i+1} & \cdots & a_{0,n-1} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,i-1} & b_{n-1} & a_{n-1,i+1} & \cdots & a_{n-1,n-1} \end{pmatrix}.$$

Definition I.8. *Let $A \in \mathbb{C}^{n \times n}$. We say that $\lambda \in \mathbb{C}$ is an* eigenvalue *of A if there exists a nonzero vector $v \in \mathbb{C}^n$ such that $Av = \lambda v$. In this case, v is an* eigenvector associated to λ. *The vector space containing all such v is the* eigenspace associated to λ.

Problem I.11. *Prove that eigenvalues are exactly the roots of the polynomial in ℓ,* $\det(\ell \operatorname{Id} - A)$, *which is called the* characteristic polynomial of A.

Problem I.12.

1. *Give a matrix of size 2 over $\mathbb{Q}$ with 2 distinct rational eigenvalues.*
2. *Give a matrix of size 2 over $\mathbb{Q}$ with only one eigenvalue. What are the possible dimensions for the eigenspace associated to this eigenvalue?*
3. *Give a matrix of size 2 over $\mathbb{Q}$ with no rational eigenvalue.*

Eigenvalues play a crucial role in many areas of computer science: data-mining (PageRank), decision theory, imagery, scientific computing, etc. But first, we need good algorithms for solving linear systems (with polynomial bit complexity) and this leads to good algorithm for computing determinants.

I.4. Gaussian elimination and LU factorization

When performing Gaussian elimination, the goal is to compute linear combinations of the matrix rows to make appear some zeroes and obtain an upper triangular matrix.

These operations can be summed up as a *factorization*, the so-called LU factorization, of the matrix into two: one lower triangular matrix with 1's on the diagonal and one upper triangular matrix.

Example I.9.

$$\begin{aligned}
\begin{pmatrix} 4 & 4 & 8 & 1 \\ 2 & 8 & 7 & 1 \\ 1 & 3 & 6 & 1 \\ -4 & 6 & 5 & 1 \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{4} & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 & 1 \\ 0 & 6 & 3 & \frac{1}{2} \\ 0 & 2 & 4 & \frac{3}{4} \\ 0 & 10 & 13 & 2 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{3} & 1 & 0 \\ -1 & \frac{5}{3} & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 & 1 \\ 0 & 6 & 3 & \frac{1}{2} \\ 0 & 0 & 3 & \frac{7}{12} \\ 0 & 0 & 8 & \frac{7}{6} \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{3} & 1 & 0 \\ -1 & \frac{5}{3} & \frac{8}{3} & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 & 1 \\ 0 & 6 & 3 & \frac{1}{2} \\ 0 & 0 & 3 & \frac{7}{12} \\ 0 & 0 & 0 & -\frac{7}{18} \end{pmatrix}.
\end{aligned}$$

The LU factorization is not always possible: whenever the pivot is 0, the LU factorization does not exist. In Gaussian elimination, the problem is circumvented by permuting rows. For matrix factorization, this is done by factoring by a permutation matrix yielding a PLU factorization.

Definition I.10. *A permutation matrix is a matrix whose entries are all* 0 *or* 1*. For each row and each column, only one coefficient is nonzero. A permutation matrix P satisfies $P^{-1} = P^{\mathrm{T}}$.*

Gaussian elimination and LU factorization do not behave well with floating-point arithmetic: absorption or cancellation can arise easily. Likewise, a very small pivot will ill behave.

Example I.11.

$$\begin{aligned}
\begin{pmatrix} 4 & 4 & 8 \\ 2 & 2 & 7 \\ 1 & 3 & 6 \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{4} & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 0 & 3 \\ 0 & 2 & 4 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{4} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 0 & 3 \\ 0 & 2 & 4 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 \\ \frac{1}{4} & 1 & 0 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 2 & 4 \\ 0 & 0 & 3 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 2 & 4 \\ 0 & 0 & 3 \end{pmatrix}.
\end{aligned}$$

What happens for $A = \begin{pmatrix} 2^{-60} & 1 \\ 1 & 1 \end{pmatrix}$*?*

The LU factorization of A is

$$A = \begin{pmatrix} 1 & 0 \\ 2^{60} & 1 \end{pmatrix} \begin{pmatrix} 2^{-60} & 1 \\ 0 & 1 - 2^{60} \end{pmatrix}.$$

In double precision, $1 - 2^{60}$ is represented by -2^{60}, so that the LU factorization is stored as

$$\begin{pmatrix} 1 & 0 \\ 2^{60} & 1 \end{pmatrix} \begin{pmatrix} 2^{-60} & 1 \\ 0 & -2^{60} \end{pmatrix}.$$

Yet, expanding this decomposition using double-precision arithmetic, we obtain

$$\begin{pmatrix} 2^{-60} & 1 \\ 1 & 0 \end{pmatrix} \neq A!$$

If we swap the rows, then we have the following PLU decomposition,

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2^{-60} & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 2^{-60} & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Solving $Ax = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ yields $x = \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$ with the first decomposition and $x = \left(\begin{smallmatrix} -1 \\ 1 \end{smallmatrix}\right)$ with the second one. In exact arithmetic, the solution is $\left(\begin{smallmatrix} -1/(1-2^{-60}) \\ 1/(1-2^{-60}) \end{smallmatrix}\right)$ which is rounded to $\left(\begin{smallmatrix} -1 \\ 1 \end{smallmatrix}\right)$.

Problem I.13. *Expand the last decomposition of A using double-precision arithmetic. What can be noticed?*

Problem I.14. *Solve the following system over $\mathbb{Q}$ using a $A = PLU$ decomposition,*

$$\begin{pmatrix} 2 & 3 & 1 & 5 \\ 6 & 9 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 8 & 44 & 20 & 76 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 22 \\ 76 \\ 99 \\ 256 \end{pmatrix}.$$

I.4.1. Swapping rows and columns

To avoid this kind of problem, it is necessary to swap rows to take the greatest number in the column (below the current row) as the pivot. In fact, it is even better to take the greatest number in the whole submatrix as the pivot. This makes us swapping rows and columns, yielding a LU factorization with permutations matrices on the left and on the right. This is the PLUQ factorization.

The method is straight-forward, at step i (with i from 0 to $n - 1$):

- Pick the greatest number in absolute value in the submatrix made from rows i to $n - 1$ and column from i to $n - 1$ for the pivot.
- Swap the rows and columns, so that the pivot is in position (i, i).
- Update L and U.

Example I.12.

$$
\begin{aligned}
A &= \begin{pmatrix} 1 & 2^{20} & 2^{40} \\ 2 & 2^{40} & 2^{108} \\ 2^{30} & 2^{54} & 2^{10} \end{pmatrix} \\
&= \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2^{20} & 2^{40} \\ 2 & 2^{40} & 2^{108} \\ 2^{30} & 2^{54} & 2^{10} \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}^{\mathrm{T}} \\
&= \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2^{108} & 2^{40} & 2 \\ 2^{40} & 2^{20} & 1 \\ 2^{10} & 2^{54} & 2^{30} \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 2^{-68} & 1 & 0 \\ 2^{-98} & 0 & 1 \end{pmatrix} \begin{pmatrix} 2^{108} & 2^{40} & 2 \\ 0 & 2^{20}-2^{-28} & 1 \\ 0 & 2^{54} & 2^{30} \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 2^{-68} & 1 & 0 \\ 2^{-98} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}^{\mathrm{T}} \\
&\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 2^{108} & 2^{40} & 2 \\ 0 & 2^{20}-2^{-28} & 1 \\ 0 & 2^{54} & 2^{30} \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 2^{-98} & 1 & 0 \\ 2^{-68} & 0 & 1 \end{pmatrix} \begin{pmatrix} 2^{108} & 2^{40} & 2 \\ 0 & 2^{54} & 2^{30} \\ 0 & 2^{20}-2^{-28} & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 2^{-98} & 1 & 0 \\ 2^{-68} & 2^{-34}-2^{-82} & 1 \end{pmatrix} \begin{pmatrix} 2^{108} & 2^{40} & 2 \\ 0 & 2^{54} & 2^{30} \\ 0 & 0 & 1-2^{-4} \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}
\end{aligned}
$$

.

Problem I.15.

1. *Check all the computations and explain the absorption steps of Example I.12.*
2. *What would be the LU decomposition of the same matrix without any pivoting?*
3. *Do the two decompositions yield the same determinant?*

Problem I.16. *Prove that if P and Q are the permutation matrices obtained in the PLUQ decomposition of A, then $\det P = (-1)^p$ and $\det Q = (-1)^q$, where p (resp. q) is the number of swaps for the rows (resp. columns) that have been performed.*

I.4.2. Solving linear systems

Solving a linear system $Ax = b$ comes down to solving $PLUQx = b$. This is equivalent to solving $LUQx = P^{\mathrm{T}}b$ or $LUx' = b'$ with $x' = Qx$ and $b' = P^{\mathrm{T}}b$.

Then, we first solve $Ly = b'$, which is a lower triangular system and then $Ux' = y$, which is an upper one. Finally, since $x' = Qx$, we have $x = Q^{\mathrm{T}}x'$.

Example I.13. *Solving* $Ax = \begin{pmatrix} 0 \\ 2^{34} \\ 1 \end{pmatrix}$ *in floating-point arithmetic with double precision, gives*

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 2^{-98} & 1 & 0 \\ 2^{-68} & 2^{-34} - 2^{-82} & 1 \end{pmatrix}\begin{pmatrix} 2^{108} & 2^{40} & 2 \\ 0 & 2^{54} & 2^{30} \\ 0 & 0 & 1 - 2^{-4} \end{pmatrix}\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2^{34} \\ 1 \end{pmatrix}$$

which is equivalent to

$$\begin{pmatrix} 1 & 0 & 0 \\ 2^{-98} & 1 & 0 \\ 2^{-68} & 2^{-34} - 2^{-82} & 1 \end{pmatrix}\begin{pmatrix} 2^{108} & 2^{40} & 2 \\ 0 & 2^{54} & 2^{30} \\ 0 & 0 & 1 - 2^{-4} \end{pmatrix}\begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix} = \begin{pmatrix} 2^{34} \\ 1 \\ 0 \end{pmatrix}.$$

Solving $Ly = \begin{pmatrix} 2^{34} \\ 1 \\ 0 \end{pmatrix}$ *yields* $y = \begin{pmatrix} 2^{34} \\ 1 \\ -2^{-33}+2^{-82} \end{pmatrix}$.

Then, solving $Ux' = \begin{pmatrix} 2^{34} \\ 1 \\ -2^{-33}+2^{-82} \end{pmatrix}$ *yields* $x' = \begin{pmatrix} 2^{-74}-2^{-122} \\ 2^{-54} \\ 0 \end{pmatrix} = Qx$, *hence* $x = \begin{pmatrix} 0 \\ 2^{-54} \\ 2^{-74}-2^{-122} \end{pmatrix}$.

I.4.3. Computing determinants

Using Proposition I.5, we can deduce that if $A = PLUQ$, then $\det A = \det P \det L \det U \det Q$.

Problem I.17. *Give an algorithm to compute the determinant of a matrix $A \in \mathbb{C}^m$ using its PLUQ decomposition or during the computation of its PLUQ decomposition.*

I.5. Cholesky Method

A matrix $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite if $A^{\mathrm{T}} = A$ and for all $x \neq 0$, $x^{\mathrm{T}}Ax > 0$.

The LU factorization of a symmetric positive definite matrix is always possible. Furthermore, since $A = \begin{pmatrix} \alpha^2 & \omega^{\mathrm{T}} \\ \omega & K \end{pmatrix}$, we have

$$A = \begin{pmatrix} 1 & 0 \\ \frac{\omega}{\alpha^2} & \mathrm{Id} \end{pmatrix}\begin{pmatrix} \alpha^2 & \omega \\ 0 & K - \frac{\omega\omega^{\mathrm{T}}}{\alpha^2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{\omega}{\alpha^2} & \mathrm{Id} \end{pmatrix}\begin{pmatrix} \alpha^2 & 0 \\ 0 & K - \frac{\omega\omega^{\mathrm{T}}}{\alpha^2} \end{pmatrix}\begin{pmatrix} 1 & \frac{\omega}{\alpha^2} \\ 0 & \mathrm{Id} \end{pmatrix} = LDL^{\mathrm{T}}$$
$$= \begin{pmatrix} \alpha & 0 \\ \frac{\omega}{\alpha} & \mathrm{Id} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & K - \frac{\omega\omega^{\mathrm{T}}}{\alpha^2} \end{pmatrix}\begin{pmatrix} \alpha & \frac{\omega}{\alpha} \\ 0 & \mathrm{Id} \end{pmatrix}$$

For symmetric positive definite matrix, Cholesky decomposition gives $A = LL^{\mathrm{T}}$, with L lower triangular or $A = LDL^{\mathrm{T}}$, with L lower triangular with 1's on the diagonal and D diagonal.

Example I.14. *The Cholesky decomposition of the following matrix is*

$$\begin{aligned}
A &= \begin{pmatrix} 9 & 3 & 12 \\ 3 & 5 & -6 \\ 12 & -6 & 105 \end{pmatrix} \\
&= \begin{pmatrix} 3 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & -10 \\ 0 & -10 & 89 \end{pmatrix}\begin{pmatrix} 3 & 1 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 3 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & -5 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 64 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 3 & 1 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 3 & 0 & 0 \\ 1 & 2 & 0 \\ 4 & -5 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 64 \end{pmatrix}\begin{pmatrix} 3 & 1 & 4 \\ 0 & 2 & -5 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 0 \\ 1 & 2 & 0 \\ 4 & -5 & 8 \end{pmatrix}\begin{pmatrix} 3 & 1 & 4 \\ 0 & 2 & -5 \\ 0 & 0 & 8 \end{pmatrix}.
\end{aligned}$$

Problem I.18. *Compute the Cholesky decomposition of*

$$A = \begin{pmatrix} 1 & -1 & 2 \\ -1 & -3 & 2 \\ 2 & 2 & 7 \end{pmatrix}.$$

Implementation I.1. *Implement triangular linear system solving and the Gaussian elimination over double-precision floating-point numbers.*

Implementation I.2. *Implement the LU and the PLUQ factorization and linear system solving over double-precision floating-point numbers.*

Implementation I.3. *Install the MPFR library*[1] *and implement the LU and the PLUQ factorization over multi-precision floating-point numbers (*`mpfr_t`*).*

Compare the results with the previous implementation.

Implementation I.4. *Implement the Cholesky decomposition for symmetric matrix. Compare its efficiency with the LU decomposition.*

[1] https://www.mpfr.org/

II. Approximate solving of over-determined linear systems

II.1. QR Decomposition

Whenever a matrix A has size $m \times n$ with $m \geq n$, even if it has full rank, a solution of the linear system $Ax = b$ may not exist. The least-square method, then, tries to minimize the error for the Euclidean norm. That is find x' such that $\|Ax' - b\|_2$ is minimal.

The QR decomposition factors A into QR with Q unitary and R upper triangular.

II.1.1. Euclidean norm, orthogonal and unitary matrices

We start by recalling the definition of the *Euclidean norm* and of the classical *scalar product* in dimension m. These extends the known definitions in dimensions 1, 2 or 3.

Definition II.1 (Scalar product)**.** *Let x and y be two vectors in $\mathbb{R}^m$. Then, their* scalar product *is the scalar $\langle x \mid y \rangle$ defined as*

$$\langle x \mid y \rangle = x^{\mathrm{T}} \cdot y = \sum_{i=1}^{m} x_i y_i.$$

This definition can be extended to complex vectors $x, y \in \mathbb{C}^m$ as follows using the notation $x^\star = \bar{x}^{\mathrm{T}}$:

$$\langle x \mid y \rangle = x^\star \cdot y = \sum_{i=1}^{m} \bar{x}_i y_i.$$

Definition II.2 (Euclidean norm)**.** *Let x be a vector in $\mathbb{R}^m$. Then, its* Euclidean norm*, or* norm *if there is no ambiguity, is the scalar $\|x\|_2$, or $\|x\|$ again if there is no ambiguity, defined as*

$$\|x\|_2 = \sqrt{\langle x \mid x \rangle} = \sqrt{\sum_{i=1}^{m} x_i^2}.$$

This can be extended to a complex vector $x \in \mathbb{C}^m$ as follows

$$\|x\|_2 = \sqrt{\langle x \mid x \rangle} = \sqrt{\sum_{i=1}^{m} |x_i|^2}.$$

Problem II.1. *Let $m, n \in \mathbb{N}$ with $m > n > 0$. Let $x \in \mathbb{C}^m$ and let us denote $y = x_{1,\dots,n} \in \mathbb{C}^n$ the vector formed by the first n rows of x. Likewise, let us denote $z = x_{n+1,\dots,m} \in \mathbb{C}^{m-n}$ the vector formed by the last $m - n$ rows of x.*

Show that $\|x\|_2^2 = \|y\|_2^2 + \|z\|_2^2$.

Orthogonal and unitary matrices are the one that preserve this scalar product, or this norm.

Definition II.3 (Orthogonal or unitary matrix)**.** *A matrix* $Q \in \mathbb{R}^{m\times m}$ *is* orthogonal *if one of the following equivalent conditions is fulfilled*

- $QQ^{\mathrm{T}} = Q^{\mathrm{T}}Q = \mathrm{Id}$*;*
- *for all* $x \in \mathbb{R}^m$, $\|Qx\|_2 = \|x\|_2$*;*
- *for all* $x, y \in \mathbb{R}^m$, $\langle Qx \mid Qy\rangle = \langle x \mid y\rangle$.

A matrix $Q \in \mathbb{C}^{m\times m}$ *is* unitary *if one the following equivalent conditions is fulfilled*

- $QQ^\star = Q^\star Q = \mathrm{Id}$*;*
- *for all* $x \in \mathbb{C}^m$, $\|Qx\|_2 = \|x\|_2$*;*
- *for all* $x, y \in \mathbb{C}^m$, $\langle Qx \mid Qy\rangle = \langle x \mid y\rangle$.

Problem II.2.

1. *Prove that in the orthogonal or unitary cases, the three conditions are indeed equivalent.*
2. *Show that the set of orthogonal matrices of size n is a group, that is it satisfies the three following conditions:*
   - Id *is orthogonal;*
   - *if* A *is orthogonal, then so is* A^{-1}*;*
   - *if* A *and* B *are orthogonal, then so is* AB.
3. *Show that the set of unitary matrices of size n is a group.*

Example II.4. *The following matrices A and B are respectively orthogonal and unitary.*

$$A = \begin{pmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 & 0 \\ \frac{\mathrm{i}}{2} & 0 & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & 0 & -\frac{\mathrm{i}}{2} \end{pmatrix}.$$

Problem II.3. *Is the matrix A of Example II.4 unitary? Is the matrix B of Example II.4 orthogonal?*

Problem II.4. *Let* $Q \in \mathbb{C}^{m\times m}$ *be unitary. Let* $n \in \mathbb{N}$, $m > n > 0$ *and let* $\tilde{Q}$ *be the matrix formed by the first n columns of* Q.

1. *Show that* $\tilde{Q}^\star\tilde{Q} = \mathrm{Id}$.
2. *What can be said about* $\tilde{Q}\tilde{Q}^\star$*?*

II.1.2. Solving a least-square problem

Theorem II.5. *For any full-rank matrix $A \in \mathbb{C}^{m\times n}$ with $m \geq n$, a QR decomposition of A with Q unitary and R upper triangular exists.*

Proof. We denote $u_1, \ldots, u_n$, the column vectors of A. We let $q_1 = \frac{u_1}{\|u_1\|_2}$. This is an orthonormal basis of the vecyor space spanned by u_1.

Assuming $q_1, \ldots, q_i$ form an orthonormal basis of the vector space spanned by $u_1, \ldots, u_i$, we let q_{i+1} be a vector in $\mathrm{Span}(u_1, \ldots, u_{i+1})$ such that $q_1, \ldots, q_{i+1}$ is an orthonormal family.

The family $q_1, \ldots, q_n$ can be extended into an orthonormal family $q_1, \ldots, q_m$ (for instance by first extending the free family $u_1, \ldots, u_n$ into a basis $u_1, \ldots, u_m$ and applying the same process).

Then, the matrix Q whose columns are $q_1, \ldots, q_m$ is unitary. Furthermore, for all i, there exist $r_{i,1}, \ldots, r_{i,i}$ such that $u_i = \sum_{j=1}^{i} r_{i,j} q_j$, hence $R = (r_{i,j})_{1\leq i,j\leq n}$ is upper triangular and $A = QR$. □

First, it is important to notice that minimizing $\|Ax' - b\|_2$ is equivalent to minimizing

$$\|Ax' - b\|_2^2 = \|Q^\star(Ax' - b)\|_2^2 = \|Rx' - c\|_2^2, \quad c = Q^\star b.$$

Now, if $m > n$, then R has $m - n$ rows of zeroes and

$$\|Ax' - b\|_2^2 = \|R_{1,\ldots,n}x' - c_{1,\ldots,n}\|_2^2 + \|c_{n+1,\ldots,m}\|_2^2.$$

If A has full rank, then so is $R_{1,\ldots,n}$ and x' is found by solving the triangular system $R_{1,\ldots,n}x' = c_{1,\ldots,n}$, where $R_{1,\ldots,n}$ is the matrix formed with the n first rows of R and $c_{1,\ldots,n}$ (resp. $c_{n+1,\ldots,m}$) is the vector form with the n first (resp. $m - n$ last) rows of c.

Example II.6. *A QR decomposition of*

$$A = \begin{pmatrix} 3 & -3 \\ 4 & -4 \\ 0 & 40 \end{pmatrix} = \begin{pmatrix} \frac{3}{5} & 0 & \frac{4}{5} \\ \frac{4}{5} & 0 & -\frac{3}{5} \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 5 & -5 \\ 0 & 40 \\ 0 & 0 \end{pmatrix}.$$

Thus, finding x' such that $\|Ax' - b\|_2$ is minimal with $b = \left(\begin{smallmatrix} 5 \\ 10 \\ 2 \end{smallmatrix}\right)$ comes down to solving

$$\begin{pmatrix} 5 & -5 \\ 0 & 40 \end{pmatrix} x' - \begin{pmatrix} \frac{3}{5} & \frac{4}{5} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 10 \\ 2 \end{pmatrix} = 0 \iff \begin{pmatrix} 5 & -5 \\ 0 & 40 \end{pmatrix} x' - \begin{pmatrix} 11 \\ 2 \end{pmatrix} = 0.$$

Thus $x' = \left(\begin{smallmatrix} \frac{9}{4} \\ \frac{1}{20} \end{smallmatrix}\right)$.

Problem II.5. *Let $A \in \mathbb{C}^{m\times n}$, $Q \in \mathbb{C}^{m\times m}$ and $R \in \mathbb{C}^{m\times n}$ such that $A = QR$, Q is unitary and R is upper triangular.*

Let $Q' \in \mathbb{C}^{m\times n}$ made from the first n columns of Q and $R' \in \mathbb{C}^{n\times n}$ made from the first n rows of R.

Show that $A = Q'R'$.

Problem II.6. *Find $x' \in \mathbb{C}^3$ such that $\|QRx' - b\|_2$ is minimal for*

$$Q = \begin{pmatrix} \frac{5}{13} & 0 & \frac{12}{13} & 0 \\ 0 & -\frac{3}{5} & 0 & \frac{4}{5}\mathrm{i} \\ \frac{12}{13} & 0 & -\frac{5}{13} & 0 \\ 0 & -\frac{4}{5}\mathrm{i} & 0 & \frac{3}{5}\mathrm{i} \end{pmatrix}, \quad R = \begin{pmatrix} 1 & -3 & 0 \\ & -1 & 4 \\ & & 140 \\ & & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 130 \\ 130 \\ 130 \\ 130 \end{pmatrix}.$$

II.2. Computing a QR Decomposition

II.2.1. Givens' method

As in the Gaussian elimination, the goal is to make some zeroes appear under the diagonal of the matrix. However, given a column, we cannot make all the zeroes appear in this column in one round.

The idea to put a 0 in position (i, j) in the matrix R, with $i > j$, is to multiply the matrix by a $m \times m$ rotation matrix $G_{i,j}$ on the left. Over $\mathbb{R}$, $G_{i,j}$ is a matrix with 0 coefficients everywhere except:

- $g_{k,k} = 1$ for all $1 \le k \le m$, $k \ne i, j$;
- $g_{i,i} = g_{j,j} = c$, $g_{j,i} = -g_{i,j} = s$ with $c^2 + s^2 = 1$.

$$G_{i,j} = \begin{pmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, i = 3, j = 1$$

for some c and s. The coefficients c and s are directly given by $r_{j,j}$ and $r_{i,j}$ with

$$c = \frac{r_{j,j}}{\sqrt{r_{j,j}^2 + r_{i,j}^2}}, \qquad s = \frac{r_{i,j}}{\sqrt{r_{j,j}^2 + r_{i,j}^2}}.$$

Then, the algorithm is

```
**Algorithm 3:** Givens' algorithm
**Input:** A real matrix $A$ of size $m \times n$.
**Output:** Its QR Decomposition.
$Q \leftarrow \mathrm{Id}_m$
$R \leftarrow A$
For $j$ from 1 to $n$ do
    For $i$ from $j+1$ to $m$ do
        $R \leftarrow G_{i,j} R$
        $Q \leftarrow Q G_{i,j}^{\mathrm{T}}$
Return $Q, R$
```

Problem II.7. *Using Givens' method, compute a QR decomposition of*

$$\begin{pmatrix} 3 & -3 & -2 \\ 4 & -4 & 14 \\ 12 & -12 & 24 \\ 0 & 3 & -5 \end{pmatrix}.$$

Problem II.8.

1. *Show that if $A \in \mathbb{R}^{m\times m}$ is orthogonal, then* $\det A = \pm 1$.
2. *Show that at the end of Givens' method,* $\det Q = 1$.

II.2.2. Gram–Schmidt's method

The main idea of the Gram–Schmidt method is to construct an orthonormal basis from the column vector of the input matrix A. More precisely, if $a_1, \dots, a_n$ are the columns of A, we want $q_1, \dots, q_n$ to satisfy

1. $(q_1, \dots, q_n)$ is an orthonormal family;
2. for all i, $q_1, \dots, q_i$ span the same vector space as $a_1, \dots, a_i$.

In the following algorithm, $r_{i,j}$ is the coefficient of R in position (i, j) while q_j (resp. a_j) is the jth column vector of Q (resp. A).

```
**Algorithm 4:** Gram–Schmidt's algorithm
**Input:** A matrix $A$ of size $m \times n$.
**Output:** Its QR Decomposition.
$r_{1,1} \leftarrow \|a_1\|_2$
$q_1 \leftarrow \frac{a_1}{r_{1,1}}$
**For** $j$ **from** $2$ **to** $n$ **do**
    $q_j = a_j$
    **For** $i$ **from** $1$ **to** $j-1$ **do**
        $r_{i,j} \leftarrow q_i^\star q_j$
        $q_j \leftarrow q_j - r_{i,j} q_i$
    $r_{j,j} \leftarrow \|q_j\|_2$
    $q_j \leftarrow \frac{q_j}{r_{j,j}}$
**Return** $Q, R$
```

Problem II.9. *Using Gram–Schmidt's method, compute a QR decomposition of*

$$\begin{pmatrix} -7 & 21 \\ -4 & 26 \\ -4 & -2 \\ 0 & 7 \end{pmatrix}.$$

Problem II.10. 1. *Compute a QR decomposition of*

$$A = \begin{pmatrix} 3 & 2 & 16 \\ 4 & 11 & 13 \\ 0 & 0 & 12 \\ 0 & 0 & 9 \end{pmatrix}.$$

2. *Compute the vector x' such that $\|Ax' - b\|_2$ is minimal for*

$$b = -\begin{pmatrix} 21 \\ 3 \\ 33 \\ 6 \end{pmatrix}.$$

Problem II.11 (Householder transformation). *The goal is to study another QR decomposition method.*

1. *Let z be a vector of size m. Let $v = z - \alpha e_1$ with $\alpha = \|z\|$ and e_1 the first vector of the canonical basis and let $u = v/\|v\|$. Let*

   $$Q = \mathrm{Id} - 2uu^\star.$$

   Show that Q is unitary and that $Qz = \alpha e_1$. The matrix Q is a Householder transformation.

2. *Let R be a matrix of size $m \times n$. Let z be the first column vector of R and let Q be defined as above. Show that the first column of QR has a nonzero first coefficient and only zeroes below.*

3. *Let R_0 be a matrix of size $m \times n$, we know, using the previous question, how to determine a Householder transformation Q_1 such that $R_1 = Q_1 R_0$ only has zeroes under the diagonal of its first column. Explain how to determine*

   $$Q_2 = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{Id} - 2u_2 u_2^\star \end{pmatrix}$$

   so that $R_2 = Q_2 R_1$ only has zeroes under the diagonal of its first and second columns.

4. *Give an algorithm iterating this process in order to obtain an upper triangular matrix through multiplications by Householder matrices.*

II.3. Diagonalization

Definition II.7. *A $n \times n$ matrix A is diagonalizable if there exists P invertible such that $A = PDP^{-1}$ with D diagonal. The coefficients of D are the* eigenvalues *and the column vectors of P are the* eigenvectors *of A.*

We know that A is diagonalizable if

- A is real symmetric ($A = A^{\mathrm{T}}$) or complex Hermitian ($A^\star = \bar{A}^{\mathrm{T}} = A$), the eigenvalues are then real;
- A is normal ($AA^\star = A^\star A$);
- the eigenvalues of A are all distinct.

Problem II.12. *Show whether the matrices*

$$A_\varepsilon = \begin{pmatrix} 1 & 1 \\ 0 & 2+\varepsilon \end{pmatrix}, \quad B_\varepsilon = \begin{pmatrix} 1 & 1 \\ 0 & 1+\varepsilon \end{pmatrix}$$

are diagonalizable for $\varepsilon \geq 0$.

II.3.1. Computing the eigenvalues

Numerically, it is possible to compute the eigenvalues and eigenvectors of a $n \times n$ matrix A without computing its characteristic polynomial.

1. Find V unitary such that $V^\star AV = H$ with H
   - tridiagonal if A is Hermitian;
   - upper Hessenberg (the coefficients below the first subdiagonal are 0).

   Notice that if $H = PDP^{-1}$, then $A = VPDP^{-1}V^\star = (VP)D(VP)^{-1}$.
2. Compute the diagonalization of H iterating about $5n$ times the following:
   a. Compute the QR decomposition of $H = QR$;
   b. Replace H by RQ. Since $RQ = Q^\star QRQ = Q^\star HQ$, the new H has the same eigenvalues as the former H.
3. The elements outside the diagonal vanish allowing us to read the eigenvalues of H.

Implementation II.1. *Implement the three QR decompositions algorithms over double-precision floating-point numbers.*

Compare their efficiency.

Implementation II.2. *Implement the three QR decompositions algorithms over multi-precision floating-point numbers.*

Compare their efficiency.

III. Matrix and vector compression, SVD and FFT

III.1. Singular Value Decomposition

The Singular Value Decomposition is a decomposition that exists for any $m \times n$ complex matrix. It is related to the diagonalization of matrices. Furthermore, if a grayscale picture is seen as a matrix, its SVD allows us to compress the picture.

Definition III.1. *Let $A \in \mathbb{C}^{m\times n}$ with $m \geq n$. There exists U, V, Σ such that*

- *$A = U\Sigma V^\star$;*
- *U has size $m \times m$ and $U^\star U = \mathrm{Id}$;*
- *$\Sigma \in \mathbb{R}^{m\times n}$ and its nonzero elements are on the diagonal and satisfy $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$;*
- *V has size $n \times n$ and $V^\star V = \mathrm{Id}$.*

If $A = U\Sigma V^\star$, then $A^\star A = V\Sigma^\star\Sigma V^\star$. Therefore, $\sigma_1^2, \ldots, \sigma_n^2$ are the eigenvalues of $A^\star A$ and the columns of V are their eigenvectors.

Example III.2. *The SVD of the matrix*

$$
\begin{aligned}
M &= \begin{pmatrix} 2 & 4 & 4 \\ 0 & 8 & 3 \\ 12 & 16 & 11 \end{pmatrix} \\
&= \begin{pmatrix} -2.368 & -6.928 \cdot 10^{-1} & -9.691 \\ -2.984 & -9.441 & 1.404 \\ -9.246 & 3.224 & 2.029 \end{pmatrix} \begin{pmatrix} 2.463 \cdot 10^{1} & & \\ & 4.609 & \\ & & 1.409 \end{pmatrix} \begin{pmatrix} -4.696 & -7.359 & -4.877 \\ 8.094 & -5.795 & 9.492 \\ 3.524 & 3.502 & -8.678 \end{pmatrix} \cdot 10^{-2}.
\end{aligned}
$$

By setting to 0, the last two singular values, we can compress the representation into

$$
\begin{pmatrix} -2.368 \\ -2.984 \\ -9.246 \end{pmatrix} \begin{pmatrix} 2.463 \cdot 10^{1} \end{pmatrix} \begin{pmatrix} -4.696 & -7.359 & -4.877 \end{pmatrix} \cdot 10^{-2} = \begin{pmatrix} 2.740 & 4.293 & 2.845 \\ 3.452 & 5.409 & 3.585 \\ 1.070 \cdot 10^{1} & 1.676 \cdot 10^{1} & 1.111 \cdot 10^{1} \end{pmatrix}.
$$

III.1.1. Computing the SVD

The computation is straightforward:

1. Compute $A^\star A$.

2. Compute its diagonalization $A^\star A = VDV^\star$.

3. Let Σ be the $m \times n$ matrix whose diagonal elements are the non-negative square roots of the diagonal elements of D in decreasing order.

4. Solve $U\Sigma = AV$ with U unitary.

III.1.2. Properties of the SVD

Problem III.1. *Let $A = U\Sigma V^{\mathrm{T}}$ be the SVD of a matrix A of size $m \times n$ with $m \geq n$.*

1. *Show that if A has full rank, then the solution to $\min_x \|Ax - b\|_2$ is $x = V\Sigma^{-1}U^{\mathrm{T}}b$.*

2. *Let us recall that*
$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2 = 1} \|Ax\|_2.$$
Show that $\|A\|_2 = \sigma_1$ and that if A is an invertible square matrix, then $\|A^{-1}\|_2 = \sigma_n^{-1}$ and $\|A\|_2 \cdot \|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}$.

3. *Let us write $U = [u_1, u_2, \ldots, u_n]$ and $V = [v_1, v_2, \ldots, v_n]$, with $u_1, \ldots, u_n, v_1, \ldots, v_n$ column-vectors. We have*
$$A = U\Sigma V^{\mathrm{T}} = \sum_{i=1}^{n} \sigma_i u_i v_i^{\mathrm{T}}.$$
Show that the closest matrix of rank $k < n$ (for $\|\cdot\|_2$) to A is $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^{\mathrm{T}}$ and that $\|A - A_k\|_2 = \sigma_{k+1}$.

The SVD can also be used for computing the pseudo-inverse of a matrix.

Definition III.3 (Pseudoinverse). *Let $A \in \mathbb{C}^{m\times n}$ and $A = U\Sigma V^\star$ be its SVD. Denote $\Sigma = (\sigma_{i,j})_{\substack{0\leq i<m \\ 0\leq j<n}}$. Let $T = (\tau_{i,j})_{\substack{0\leq i<n \\ 0\leq j<m}} \in \mathbb{C}^{n\times m}$ defined by $\tau_{i,j} = 0$ if $i \neq j$, $\tau_{i,i} = \sigma_{i,i}^{-1}$ if $\sigma_{i,i} \neq 0$ and $\tau_{i,i} = 0$ otherwise.*

Then, the pseudoinverse *of A is $A^\dagger = VTU^\star$.*

Problem III.2.

1. *Show that if Σ only has nonzero coefficients on its diagonal, then $\Sigma^\dagger = T$, as defined in Definition III.3.*

2. *Show that if $A \in \mathbb{C}^{m\times m}$ is invertible, then $A^\dagger = A^{-1}$.*

III.2. Fast Fourier Transform

The FFT was invented by Carl Friedrich Gauß in 1866. The modern FFT algorithm is due to James W. Cooley and John W. Tukey in 1965.

The FFT is a bijective linear map on vectors in $\mathbb{C}^n$. It can be used to multiply fast polynomials, by identifying a polynomial of degree at most $n-1$ with its vector of coefficients of size n. Technically, the FFT is the transformation and the inverse FFT is the inverse transformation. We shall see that the inverse FFT is a kind of FFT itself.

III.2.1. Definition

Definition III.4. *For n a positive integer, the nth roots of unity are all the roots of the polynomial $z^n - 1$. They form exactly the set $\left\{ \mathrm{e}^{\frac{2\mathrm{i}\ell\pi}{n}} \,\middle|\, \ell \in \{0, \dots, n-1\} \right\}$ in $\mathbb{C}$.*

An nth root of unity is primitive *if it is not a kth root of unity for $1 \leq k \leq n-1$. In $\mathbb{C}$, $\mathrm{e}^{\pm\frac{2\mathrm{i}\pi}{n}}$ are always primitive.*

Furthermore, if z is a (primitive) 2^kth root of unity, then z^2 is a (primitive) 2^{k-1}th root of unity.

Definition III.5. *Let $n \in \mathbb{N}$, $n > 0$, and ω be a primitive nth root of unity in $\mathbb{C}$. Let $v \in \mathbb{C}^n$. The FFT of v is the vector $\Omega_n v$, where*

$$\Omega_n = \left(\omega^{ij}\right)_{0 \leq i,j < n} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)^2} \end{pmatrix}.$$

This matrix of the Vandermonde matrix of $1, \omega, \dots, \omega^{n-1}$.

Problem III.3.

1. *What are the primitive nth root of unity in $\mathbb{C}$ for $n = 1, 2, 3, 4, 5, 6$?*
2. *Let $n \in \mathbb{N}^*$, what is the sum of all the nth roots of unity?*
3. *What is their product?*
4. *What is the FFT of $(1, 0, 0, 0)^{\mathrm{T}}$, assuming $\omega = \mathrm{i}$?*
5. *Which vector has $(1, 0, 0, 0)^{\mathrm{T}}$ as its FFT with the same assumption?*

Problem III.4.

1. *Compute the inverse of the matrix Ω_n for $n = 1, 2, 3, 4$.*
2. *Let Ω_2 and Ω_4 be the matrix of Definition III.5 for respectively $n = 2$ and $n = 4$.*
   a. *Let Ω_4' be the matrix obtained from Ω_4 after permuting the columns 1 and 2 (numbered from 0 to 3). Give Ω_4'.*

b. Show that Ω'_4 is a (2×2)-block matrix whose blocks are derived from Ω_2.

Lemma III.6. *Let $\omega \in \mathbb{C}$ be a primitive nth root of unity and let $p = (p_0, \dots, p_{n-1})^{\mathrm{T}} \in \mathbb{C}^n$. Let $\Omega_n = \left(\omega^{ij}\right)_{0 \le i,j < n}$ be the Vandermonde matrix of $1, \omega, \dots, \omega^{n-1}$ and $P = p_{n-1}x^{n-1} + \cdots + p_0$ be the polynomial whose vector of coefficients is p.*

Then, $\Omega_n p$ is the vector $\left(P(1), P(\omega), \dots, P(\omega^{n-1})\right)^{\mathrm{T}}$.

Problem III.5. *Prove Lemma III.6.*

To perform this matrix-vector efficiently, we will rely polynomial evaluation.

III.2.2. Evaluation by Divide and Conquer

We now assume that $n = 2^k$. Let us notice that since ω is a primitive nth root of unity, then for all i, ω^i and $-\omega^i$ are also nth roots of unity. Indeed, the powers of ω are in fact $1, \omega, \omega^2, \dots, \omega^{\frac{n}{2}-1}, \omega^{\frac{n}{2}} = -1, \omega^{\frac{n}{2}+1} = -\omega, \omega^{\frac{n}{2}+2} = -\omega^2, \dots, \omega^{n-1} = -\omega^{\frac{n}{2}-1}$.

Lemma III.7. *Let $P = p_{n-1}x^{n-1} + \cdots + p_0$ be a polynomial of degree $n-1$. Let P_o and P_e be polynomials of degree $\frac{n}{2} - 1$ defined by*

$$P = P_e(x^2) + xP_o(x^2).$$

Then,

$$\begin{aligned} P_e &= p_{n-2}x^{\frac{n-2}{2}} + p_{n-4}x^{\frac{n-4}{2}} + \cdots + p_2x + p_0 \\ P_o &= p_{n-1}x^{\left\lfloor\frac{n-1}{2}\right\rfloor} + p_{n-3}x^{\left\lfloor\frac{n-3}{2}\right\rfloor} + \cdots + p_3x + p_1. \end{aligned}$$

Furthermore, evaluating P in $1, \omega, \omega^2, \dots, \omega^{n-1}$ comes down to evaluating P_e and P_o in $1, \omega^2, \omega^4, \dots, \omega^{n-2}$.

Proof. Since $P(\omega^i) = P_e(\omega^{2i}) + \omega^i P_o(\omega^{2i})$ and $P(-\omega^i) = P_e(\omega^{2i}) - \omega^i P_o(\omega^{2i})$, we can evaluate P in ω^i and $-\omega^i$ by evaluating P_e and P_o in ω^{2i} plus one multiplication by ω^i and two additions or subtractions. □

Example III.8. *$P = 10x^5 + x^4 + 2x^3 + 6x^2 + 4x + 3 = (x^4 + 6x^2 + 3) + x(10x^4 + 2x^2 + 4)$ so that $P_e = x^2 + 6x + 3$ and $P_o = 10x^2 + 2x + 4$.*

To evaluate P in $1, \omega = \mathrm{i}, \omega^2 = -1, \omega^3 = -\mathrm{i}$,

- *we evaluate P_e and P_o in $1, \omega^2 = -1$*
  - *$P_e(1) = 10$ and $P_e(-1) = -2$;*
  - *$P_o(1) = 16$ and $P_o(-1) = 12$;*
- *we recombine these evaluations*
  - *$P(1) = P_e(1) + 1 \cdot P_o(1) = 10 + 1 \cdot 16 = 26$;*

- $P(\mathrm{i}) = P_e(-1) + \mathrm{i} \cdot P_o(-1) = -2 + \mathrm{i} \cdot 12 = -2 + 12\mathrm{i}$;
- $P(-1) = P_e(1) - 1 \cdot P_o(1) = 10 - 1 \cdot 16 = -6$;
- $P(-\mathrm{i}) = P_e(-1) - \mathrm{i} \cdot P_o(-1) = -2 - \mathrm{i} \cdot 12 = -2 - 12\mathrm{i}$.

Since $\frac{n}{2} = 2^{k-1}$, we can reapply this process on P_e and P_o to evaluate them in $1, \omega^2, \ldots, \omega^{n-2}$ by building $P_{ee}, P_{eo}, P_{oe}, P_{oo}$ and evaluating them in $1, \omega^4, \ldots, \omega^{n-4}$ until we end up on the base case: evaluating a polynomial in 1.

Example III.9 (Continuation of Example III.8). *To evaluate $P_e = x^2 + 6x + 3$ in $1, \omega^2 = -1$, we split it into $P_{ee} = x + 3$ and $P_{eo} = 6$. Likewise, we split $P_o = 10x^2 + 2x + 4$ into $P_{oe} = 10x + 4$ and $P_{oo} = 2$.*

- *We evaluate P_{ee}, P_{oe}, P_{oe} and P_{oo} in 1*
  - $P_{ee}(1) = 4$ *and* $P_{eo}(1) = 6$;
  - $P_{oe}(1) = 14$ *and* $P_{oo}(1) = 2$.
- *We recombine these evaluations*
  - $P_e(1) = P_{ee}(1) + 1 \cdot P_{eo}(1) = 4 + 1 \cdot 6 = 10$;
  - $P_e(-1) = P_{ee}(1) - 1 \cdot P_{eo}(1) = 4 - 1 \cdot 6 = -2$;
  - $P_o(1) = P_{oe}(1) + 1 \cdot P_{oo}(1) = 14 + 1 \cdot 2 = 16$;
  - $P_o(-1) = P_{oe}(1) - 1 \cdot P_{oo}(1) = 14 - 1 \cdot 2 = 12$.

This yields the following algorithm.

```
Algorithm 5: FFT
Input: A polynomial $P$ of degree strictly less than $n = 2^k$ and $\omega$ a primitive $n$th root of
       unity.
Output: The evaluation of $P$ in $1, \omega, \ldots, \omega^{n-1}$
If $\omega = 1$ then Return $(P(1))$
Split $P$ into $P_e$ and $P_o$.
Call FFT on $P_e$ and $\tau = \omega^2$ to compute $(P_e(1), P_e(\tau), \ldots, P_e(\tau^{\frac{n}{2}-1}))$.
Call FFT on $P_o$ and $\tau = \omega^2$ to compute $(P_o(1), P_o(\tau), \ldots, P_o(\tau^{\frac{n}{2}-1}))$.
For $j$ from $0$ to $\frac{n}{2} - 1$ do
    $P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j}) = P_e(\tau^j) + \omega^j P_o(\tau^j)$.
    $P(\omega^{\frac{n}{2}+j}) = P_e(\omega^{2j}) - \omega^j P_o(\omega^{2j}) = P_e(\tau^j) - \omega^j P_o(\tau^j)$.
Return $(P(1), P(\omega), \ldots, P(\omega^{n-1}))$.
```

Example III.10. *Assume, we want to evaluate $P = 32x^5 + 16x^4 + 8x^3 + 4x^2 + 2x + 1$ with the FFT algorithm. The polynomial has size 6, so a primitive 8th root of unity ω is needed. We can choose $\omega = \mathrm{e}^{\frac{\mathrm{i}\pi}{4}} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}\mathrm{i}$, we then have $\omega^2 = \mathrm{i}$, $\omega^4 = -1$ and $\omega^8 = 1$.*

1. *We call FFT*(P,ω).
   a. *We split it into* $P_e = 16x^2 + 4x + 1$ *and* $P_o = 32x^2 + 8x + 2$.
   b. *We call FFT*(P_e, i).
      i. *We split into* $P_{ee} = 16x + 1$ *and* $P_{eo} = 4$.
      ii. *We call FFT*$(P_{ee}, -1)$.
         . *We split into* $P_{eee} = 1$ *and* $P_{eeo} = 16$.
         . *We call FFT*$(P_{eee}, 1)$ *and it returns* (1).
         . *We call FFT*$(P_{eeo}, 1)$ *and it returns* (16).
         . *It returns* $(1 + 16, 1 - 16) = (17, -15)$.
      iii. *We call FFT*$(P_{eo}, -1)$.
         . *We split into* $P_{eoe} = 4$ *and* $P_{eoo} = 0$.
         . *We call FFT*$(P_{eoe}, 1)$ *and it returns* (4).
         . *We call FFT*$(P_{eoo}, 1)$ *and it returns* (0).
         . *It returns* $(4 + 0, 4 - 0) = (4, 4)$.
      iv. *It returns* $(17 + 4, -15 + 4\mathrm{i}, 17 - 4, -15 - 4\mathrm{i}) = (21, -15 + 4\mathrm{i}, 13, -15 - 4\mathrm{i})$.
   c. *We call FFT*(P_o, i).
      i. *We split into* $P_{oe} = 32x + 2$ *and* $P_{oo} = 8$.
      ii. *We call FFT*$(P_{oe}, -1)$.
         . *We split into* $P_{oee} = 2$ *and* $P_{oeo} = 32$.
         . *We call FFT*$(P_{oee}, 1)$ *and it returns* (2).
         . *We call FFT*$(P_{oeo}, 1)$ *and it returns* (32).
         . *It returns* $(2 + 32, 2 - 32) = (34, -30)$.
      iii. *We call FFT*$(P_{oo}, -1)$.
         . *We split into* $P_{ooe} = 8$ *and* $P_{ooo} = 0$.
         . *We call FFT*$(P_{ooe}, 1)$ *and it returns* (8).
         . *We call FFT*$(P_{ooo}, 1)$ *and it returns* (0).
         . *It returns* $(8 + 0, 8 - 0) = (8, 8)$.
      iv. *It returns* $(34 + 8, -30 + 8\mathrm{i}, 34 - 8, -30 - 8\mathrm{i}) = (42, -30 + 8\mathrm{i}, 26, -30 - 8\mathrm{i})$.
   d. *It returns*

$$\begin{aligned}&(21 + 42, -15 + 4\mathrm{i} + (-30 + 8\mathrm{i})\omega, 13 + 26\mathrm{i}, -15 - 4\mathrm{i} + (-30 - 8\mathrm{i})\omega^3,\\ &\quad 21 - 42, -15 + 4\mathrm{i} - (-30 + 8\mathrm{i})\omega, 13 - 26\mathrm{i}, -15 - 4\mathrm{i} - (-30 - 8\mathrm{i})\omega^3)\end{aligned}$$

$$\begin{aligned}&= (63, -15 - 19\sqrt{2} + 4\mathrm{i} - 11\mathrm{i}\sqrt{2}, 13 + 26\mathrm{i}, -15 + 19\sqrt{2} - 4\mathrm{i} - 11\mathrm{i}\sqrt{2}\\ &\qquad - 21, -15 + 19\sqrt{2} + 4\mathrm{i} + 11\mathrm{i}\sqrt{2}, 13 - 26\mathrm{i}, -15 - 19\sqrt{2} - 4\mathrm{i} + 11\mathrm{i}\sqrt{2}).\end{aligned}$$

This algorithm uses the structure of the Vandermonde matrix Ω_n made from all the nth roots of unity.

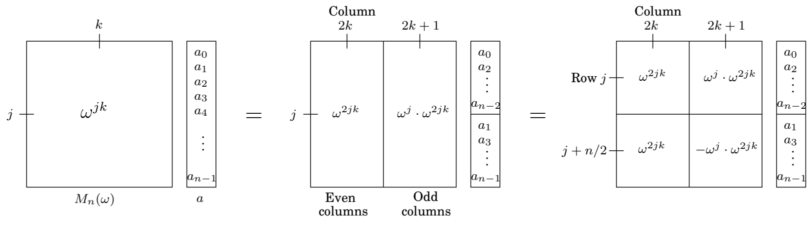
k
j
ω^{jk}
a_0 a_1 a_2 a_3 a_4 ⋮ a_{n-1}
$M_n(\omega)$
a
=
Column
2k
2k + 1
j
ω^{2jk}
$\omega^j \cdot \omega^{2jk}$
a_0 a_2 ⋮ a_{n-2} a_1 a_3 ⋮ a_{n-1}
Even columns
Odd columns
=
Column
2k
2k + 1
Row j
j + n/2
ω^{2jk}
$\omega^j \cdot \omega^{2jk}$
ω^{2jk}
$-\omega^j \cdot \omega^{2jk}$
a_0 a_2 ⋮ a_{n-2} a_1 a_3 ⋮ a_{n-1}

Figure III.1.. Recursive submatrix product

Example III.11. *The product*

$$V_{(1,\mathrm{i},-1,-\mathrm{i}),4}\begin{pmatrix}p_0\\p_1\\p_2\\p_3\end{pmatrix} = \begin{pmatrix}1 & 1 & 1 & 1\\ 1 & \mathrm{i} & -1 & -\mathrm{i}\\ 1 & -1 & 1 & -1\\ 1 & -\mathrm{i} & -1 & \mathrm{i}\end{pmatrix}\begin{pmatrix}p_0\\p_1\\p_2\\p_3\end{pmatrix}$$

can be rewritten

$$\left(\begin{array}{cc|cc}1 & 1 & 1 & 1\\ 1 & -1 & \mathrm{i} & -\mathrm{i}\\ \hline 1 & 1 & -1 & -1\\ 1 & -1 & -\mathrm{i} & \mathrm{i}\end{array}\right)\left(\begin{array}{c}p_0\\ \hline p_2\\ \hline p_1\\ p_4\end{array}\right).$$

III.2.3. Interpolating by Divide and Conquer

The inverse operation of the FFT is the inverse FFT. It consists in multiplying a vector by the inverse matrix of Ω_n. Since multiplying a vector by Ω_n consists in evaluating a polynomial in the powers of ω, the inverse operation is an *interpolation*: from a vector $(y_0, \ldots, y_{n-1})^{\mathrm{T}} \in \mathbb{C}^n$, we want to find the unique polynomial P of degree at most $n-1$ such that $P(\omega^i) = y_i$ for all $0 \leq i < n$.

Proposition III.12. *Let $\omega \in \mathbb{C}$ be a primitive nth root of unity. Let $\Omega_n = \left(\omega^{ij}\right)_{0\leq i,j<n}$. Then,*

$$\Omega_n^{-1} = \frac{1}{n}\left(\omega^{-ij}\right)_{0\leq i<n} = \frac{1}{n}\bar{\Omega}_n.$$

In other words, the inverse FFT is an FFT with $\omega^{-1} = \bar{\omega}$ as the primitive nth root of unity followed by a division by n.

Problem III.6. *Prove this statement.*

Example III.13. *If we have found that* $P(1) = 10$, $P(\mathrm{i}) = -2-2\mathrm{i}$, $P(-1) = -2$ *and* $P(-\mathrm{i}) = -2+2\mathrm{i}$, *then its inverse FFT is the FFT of the polynomial* $S = s_3x^3 + s_2x^2 + s_1x + s_0 = (-2+2\mathrm{i})x^3 - 2x^2 - (2+2\mathrm{i})x + 10$ *with* $\omega = -\mathrm{i}$ *divided by* 4. *The FFT of* S *is* $(4, 8, 12, 16)$ *so that* $P = 4x^3 + 3x^2 + 2x + 1$.

Problem III.7 (Polynomial multiplication)**.** *Let* P *and* Q *be two polynomials over* $\mathbb{C}$ *of respective degrees* $\ell - 1$ *and* $m - 1$. *We shall use the FFT to multiply them.*

1. *Show that* $R = PQ$ *has exactly* $n = \ell + m - 1$ *coefficients.*
2. *Let* p, q *and* r *be the vectors of coefficients of* P, Q *and* R *seen as polynomials of degree at most* $n - 1$. *Let* ω *be a primitive* n*th root of unity in* $\mathbb{C}$. *What is the relation between the FFT of* p *and* q *on the one hand and the FFT of* r *on the other hand.*
3. *Propose a multiplication algorithm for* P *and* Q, *i.e. that computes* R, *using the FFT.*

IV. Complexities and Arithmetics

Given $n \in \mathbb{N} \setminus \{0\}$, we recall that $\mathbb{Z}/n\mathbb{Z}$ is the set of integers modulo n. Running operations over such a set should be routine.

Cryptography, computer vision, and many other areas require routinely to perform operations with integers, polynomials and matrices of *large* size. Current software can routinely perform operations such as:

- multiply integers with more than 30 000 000 digits;
- multiply univariate polynomials of degree around 650 000 (with coefficients in $\mathbb{Z}/p\mathbb{Z}$ where p is a prime number like 67 108 879 (less than 2^{26});

Such operations require *fast* arithmetics and care about the *size* of the data which are manipulated during these computations.

IV.1. Complexity models

We mainly focus on time complexity; space complexity issues will be mostly ignored in this course.

We will use the Random Access Machine model which is the most standard one. In this model, a program reads and writes integers on different tapes, it is also allowed to use an arbitrary number of integer registers. Elementary operations (which coincide with those supported by an assembly code) are

- write and read (on a tape or a register);
- addition, substraction, multiplication and division;
- three jump instructions: unconditional jump, jump when a register is 0 or jump when a register is not 0.

We will not use subtelties of this model in what follows; the students are encouraged to read a little bit more about this model in the academic litterature.

We will measure two kinds of complexities of algorithms

Arithmetic complexity. Most of algorithms running on mathematical data are expected to run on an abstract algebraic structure equipped with binary operations.

In this model, we only count the number of such binary operations and predicate tests performed by the algorithm.

This model reflects what is observed in practice when the cost of running binary operations is mostly a constant.

For most of algorithms running over $\mathbb{Q}$, such a model is not so convenient when the size of manipulated numbers grows significantly during the computations.

Note that in this model, we do not take into account copy operations and countings performed for running loops, etc.

Bit complexity. This model is devoted to taking into account the growth of numbers when running algorithms. It is primarily defined for $\mathbb{Z}$: we will see below that every integer can be seen as a *vector* of elements of $\{0, \ldots, b-1\}$ for a given integer $b \in \mathbb{N}\setminus\{0,1\}$. In the bit complexity model, we count the number of operations performed in $\mathbb{Z}/b\mathbb{Z}$.

Again memory access is neglected in this model (for instance, transposing a matrix, even an extremely large one, is free in this model).

We will use the $O(\cdot)$ notation; in short we say that $f(n) = O(g(n))$ when there exists $K > 0$ and $M > 0$ such that for all $n \geq M$, the following inequality holds:

$$\|f(n)\| \leq K\|g(n)\|.$$

Careful attention must be paid when several parameters are involved in complexity estimates. In those cases, without further precisions, all of them are expected to tend to infinity and the constant K should not depend on any of these parameters.

We may also use the $\tilde{O}(\cdot)$ notation: we say that $f(n) = \tilde{O}(g(n))$ when there exists $a \geq 0$ such that

$$f(n) = O\left(g(n)\log_2^a(\max(2, \|g(n)\|)\right).$$

Definition IV.1. *Let N be the sum of the sizes of the input and output data. An algorithm is said to be* nearly optimal *(or* quasi-optimal*) when its runtime is bounded by $\tilde{O}(N)$.*

Problem IV.1.

1. *What is the complexity of sorting a list of integers of size n using the bubble sort algorithm? Is this algorithm quasi-optimal?*
2. *Same questions with the quicksort algorithm.*

IV.2. Integers

Here, we present the encoding of integers in computers. Key requirements are uniqueness and generality of the representation.

In the integer encodings presented below, uniqueness is actually a consequence of the uniqueness of the quotient and remainder in the Euclidean division of integers.

Proposition IV.2. *Let a and b be in $\mathbb{N}$, with $b \neq 0$. There exists a unique (q, r) in $\mathbb{N} \times \mathbb{N}$ such that $a = bq + r$ and $0 \leq r < b$.*

Proof. Since $b \neq 0$, the sequence $(a - bq)_{q \in \mathbb{N}}$ is decreasing, and takes the value $a \geq 0$ for $q = 0$ and tends to $-\infty$ when $q \longrightarrow +\infty$. Therefore there is q such that $a - bq \geq 0$ and $a - b(q + 1) < 0$. This shows the existence: taking this q and $r = a - bq$, by definition of q we have that $r \geq 0$ and $r = a - bq = a - b(q + 1) + b < b$.

For the uniqueness, we consider two pairs (q, r) and (q', r') such that $0 \leq r < b$ and $0 \leq r' < b$ and $a = bq + r = bq' + r'$. From this identity we obtain

$$b(q - q') = r' - r. \tag{IV.1}$$

On the other hand, from the inequalities on r and r' we obtain

$$-b < r' - r < b. \tag{IV.2}$$

Combining (IV.1) and (IV.2) we obtain $-b < b(q - q') < b$, which implies $-1 < q - q' < 1$ since $b \neq 0$. Since $q - q'$ is an integer, this means $q - q' = 0$. It follows that $q = q'$ and $r = r'$. □

Theorem IV.3. *Let $n \in \mathbb{N}$ and $b \in \mathbb{N} \setminus \{0, 1\}$. There exists a unique sequence $a_0, \dots, a_h$ in $\{0, \dots, b - 1\}$ such that $a_h \neq 0$ and*

$$a_0 + a_1 b^1 + \cdots + a_h b^h = n.$$

This sequence is called the decomposition of n in base b*, and h is called the* height of n in base b*.*

Proof. Our proof is by induction on n. The cases $n = 0$ and $n = 1$ are obvious. Our induction assumption is that for any $m < n$, the decomposition of m in base b exists and is unique.

Define a_0 as the remainder of the Euclidean division of n by b, and let q be the corresponding quotient. By Proposition IV.2, the pair (q, a_0) is unique.

Observe that $q < n$, since $b \geq 2$. Hence we can apply our induction assumption on q. This shows that it can be written uniquely as $q = a_1 + \cdots + a_h b^{h-1}$ with $a_h \neq 0$ and all a_i's in $\{0, \dots, b - 1\}$. This concludes the proof. □

Lemma IV.4. *Let $n \in \mathbb{N} \setminus \{0\}$ and $b \in \mathbb{N}$. Then the height of n in basis b is exactly $\lfloor \log_b n \rfloor$.*

Proof. Using notation from the previous theorem, we obviously have $b^h \leq n$. □

Problem IV.2. *Prove that $n < b^{h+1}$, and conclude the proof of lemma IV.4.*

Problem IV.3.

1. *How many bits do you need to store integers of the form a^t for $a \in \mathbb{N}$ and $t \in \mathbb{N}$?*
2. *What is the bit complexity of the basic algorithm computing the nth term of the Fibonacci sequence? Recall that it is defined by $F_{k+2} = F_{k+1} + F_k$ for all $k \in \mathbb{N}$ with $F_0 = 0$ and $F_1 = 1$.*

Bit complexity counts the number of single / bit operations and hence allows us to take into account the growth of coefficients.

Theorem IV.5. *Let a and b be integers of respective bitsize h_1 and h_2; we also let $h = \max(h_1, h_2)$. There exist algorithms for*

- *adding a and b in bit complexity $O(h)$;*
- *multiplying a and b in bit complexity $O(h_1h_2)$ (naive algorithm);*
- *multiplying a and b in bit complexity $O(h^{1.59})$ (Karatsuba algorithm);*
- *multiplying a and b in bit complexity $O(h \log h \log\log h)$ (Fast Fourier Transform).*

In the sequel, we assume that an integer multiplication is fixed (for example, one in the above list), and the notation $\mathsf{M}_{\mathbb{Z}}(h)$ stands for the bit complexity of multiplying two integers of bitsize $\leq h$ using that algorithm (for example, one of the complexities in the above list).

Problem IV.4. *Consider an integer n, and computations in $\mathbb{Z}/n\mathbb{Z}$.*

1. *What is the (binary) cost of performing addition and multiplication in $\mathbb{Z}/n\mathbb{Z}$?*
2. *Would you implement arithmetic in $\mathbb{Z}/n\mathbb{Z}$ the same way for $n \leq 2^{32}$ and for $n > 2^{32}$?*

IV.3. Polynomials and the Karatsuba algorithm

Let R be a ring (for instance $\mathbb{Z}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$). Univariate polynomials in $R[x]$ such as $c_0 + c_1x + \cdots + c_dx^d$ can be represented by vectors of coefficients $[c_0, \ldots, c_d]$. Hence there is a strong similarity with integers (the variable x in some sense plays the same role as the base b chosen to represent integers).

Remark IV.6. *This encoding is often, but not always, the most appropriate. For example, one noticeable family is that of* sparse *polynomials: these are polynomials whose number of nonzero coefficients is* negligible *compared to their degree. For such polynomials, the proposed encoding as vectors of coefficients may be quite inefficient: the polynomial $x^{425147} + 7x^{26245} - 11x^{24} + 3$ is stored as a vector of 425148 coefficients, which are all zero except four of them.*

Theorem IV.7. *Let f and g be two polynomials in $R[x]$ of degree at most n. Then the arithmetic complexity of multiplying f and g is*

- *$O(n^2)$ operations in R using the naive algorithm;*
- *$O(n^{1.59})$ operations in R using the Karatsuba algorithm;*
- *$O(n \log n \log\log n)$ operations in R (Fast Fourier Transform).*

Problem IV.5. *Can we deduce from the above theorem the bit complexity of multiplying polynomials in $\mathbb{Z}[X]$?*

The proof of the first statement is obvious.

We take now advantage of the mention of the Karatsuba algorithm to study the important "master theorem" for the complexity of divide-and-conquer algorithms.

First let us recall that the Karatsuba algorithm relies on the very basic following observation: writing

$$f = f_1x + f_0 \qquad \text{and} \qquad g = g_1x + g_0$$

their product $h = fg = h_2x^2 + h_1x + h_0$ can be obtained through an evaluation-interpolation technique at 0, 1 and ∞:

$$h_0 = f_0g_0, \quad h_2 = f_1g_1 \quad h_1 = (f_0 + f_1)(g_0 + g_1) - h_0 - h_2.$$

Hence, to obtain the coefficients of h, only 3 multiplications (and 4 additions) are needed.

Problem IV.6.

1. *What are the evaluations of f, g and h in 0 and 1?*
2. *How are they related to the computations in the Karatsuba algorithm?*
3. *How can we define an evaluation of these polynomials in ∞ that satisfy the properties that we expect from evaluations and such that the Karatsuba algorithm computes the evaluations of f and g in ∞ in order to deduce that of h?*

The idea of reducing the number of multiplications to speed up fundamental algorithms will be used repeatedly in this course and related ones.

IV.4. Divide-and-conquer algorithms

One derives a multiplication algorithm by truncating polynomials at half degree. This is a famous example of a divide-and-conquer strategy which is at the foundation of many asymptotically fast algorithms.

On input data of size n, this principle consists in reducing to m recursive calls with input data divided by p (often $p = 2$) and finally recombine the result. The cost of the splitting and recombination steps is bounded by a function T. When the input data has size smaller than s, another algorithm running in time κ is called.

We describe below how the runtime of a divide-and-conquer algorithm can be estimated. For this asymptotic analysis, one can reasonably assume $s \geq p$. The total cost can be written as

$$C(n) = \begin{cases} T(n) + mC(\lceil n/p \rceil) & \text{when } n \geq s \geq p \\ \kappa & \text{otherwise.} \end{cases} \tag{IV.3}$$

We state now the divide-and-conquer lemma, leading to the master theorem.

Lemma IV.8 (Divide-and-conquer lemma)**.** *Let C be a function as in Equation* (IV.3) *with $m > 0, \kappa > 0$ and T such that $T(pn) \geq qT(n)$ with $q > 1$. Then, if n is a positive power of p, one has*

$$C(n) \leq \begin{cases} \left(1 - \frac{m}{q}\right)^{-1} T(n) + \kappa n^{\log_p m} & \text{if } q > m \\ T(n) \log_p n + \kappa n^{\log_p q} & \text{if } q = m \\ n^{\log_p m} \left(\frac{T(n)}{n^{\log_p q}} \frac{q}{m-q} + \kappa \right) & \text{if } q < m. \end{cases}$$

Proof. The repeated use of (IV.3) yields

$$\begin{aligned} C(n) &\leq & T(n) + mC\left(\frac{n}{p}\right) \\ &\leq & T(n) + mT\left(\frac{n}{p}\right) + \cdots + m^{k-1}T\left(\frac{n}{p^{k-1}}\right) + m^k C\left(\frac{n}{p^k}\right) \\ &\leq & T(n)\left(1 + \frac{m}{q} + \cdots + \left(\frac{m}{q}\right)^{k-1}\right) + m^k \kappa \end{aligned}$$

for $k = \left\lfloor \log_p(n/s) \right\rfloor + 1$. This choice implies

$$m^k \kappa \leq n^{\log_p m} \kappa$$

which bounds the second term of the above inequality. It remains to bound the first term.

When $q > m$, the sum in brackets is bounded by the sum of a geometric series.

When $q = m$, this is a sum with all k terms equal to 1.

When $q < m$, it can be rewritten as

$$\left(\frac{m}{q}\right)^{k-1} \left(1 + \frac{q}{m} + \cdots + \left(\frac{q}{m}\right)^{k-1}\right)$$

which we also bound using a geometric series. □

From this lemma, one easily deduces the following theorem.

Theorem IV.9 (Master theorem)**.** *Let C be a function as in Equation* (IV.3) *with $m > 0, \kappa > 0$ and T be an* increasing *function such that there exist q and r with $1 < q \leq r$ sastisfying*

$$qT(n) \leq T(pn) \leq rT(n), \qquad \text{for all } n \text{ large enough.}$$

Then

$$C(n) = \begin{cases} O(T(n)) & \text{if } q > m, \\ O(T(n) \log n) & \text{if } q = m, \\ O(n^{\log_p(m/q)} T(n)) & \text{if } q < m. \end{cases}$$

The master formula for the Karatsuba algorithm is

$$K(n) \leq 3K(n/2) + 4n.$$

In the rest of the course, we will assume that polynomials are encoded with vectors of coefficients.

Problem IV.7. *In all this exercise, n is the size of the polynomials (they are thus of degree $n-1$).*

1. *Give a complete description in pseudo-code of the Karatsuba algorithm.*
2. *In a language where the memory is handled by the user, like in C, how to implement this algorithm?*

   Be careful concerning the fact that polynomials are encoded as array of coefficients.
3. *If n is a power of 2, give a* bound *on the number of operations performed by the Karatsuba algorithm.*
4. *Let n be a power 2. Give a hybrid multiplication algorithm for two polynomials of degree n in $R[X]$ (where R is a ring) which calls the Karatsuba algorithm if $n > 2^d$ and the naive one otherwise.*
5. *Show that the arithmetic complexity $C(n)$ of this algorithm satisfies $C(n) \leq \gamma(d)n^{\log_2 3} - 8n$ for all $n \geq 2^d$, where γ is a function that only depends on d.*
6. *Find the value d that minimizes γ. What can you deduce from it?*

Implementation IV.1. *Implement the Karatsuba multiplication for polynomials over $\mathbb{Z}/p\mathbb{Z}$. Find experimentally the best size of polynomials to stop the recursive calls and instead rely on the already implemented naive algorithm.*

Implementation IV.2. *Improve your the Karatsuba algorithm implementation so that two of the three recursive calls can store their partial results in the allocated memory for the global result.*

V. Linear algebra complexities

Matrices will be mostly considered with entries in a field $\mathbb{K}$ (either a finite one like $\mathbb{Z}/p\mathbb{Z}$ with p prime or a field of characteristic 0). Dense matrices are encoded with an array; sparse and structured matrices are encoded with ad-hoc representations minimizing space complexity.

Adding matrices is easy.

Lemma V.1. *Let M and N be $p \times q$-matrices with entries in $\mathbb{K}$. Computing $M + N$ is done using $O(pq)$ operations in $\mathbb{K}$.*

Problem V.1. *Prove this lemma.*

V.1. Matrix multiplication

Multiplying matrices is more difficult.

Lemma V.2. *Let M and N be $n \times n$-matrices with entries in $\mathbb{K}$. Computing the product $M \cdot N$ with the naive multiplication algorithm uses $O(n^3)$ operations in $\mathbb{K}$.*

Problem V.2.

1. *Prove this lemma.*
2. *What is the complexity of the naive multiplication algorithm if M is $m \times n$ and N is $n \times p$?*

Remark V.3. *The size of the input M and N is $O(n^2)$; the size of the output is $O(n^2)$. We deduce that the cost of multiplying M by N is* not *asymptotically optimal. Reducing the exponent 3 as much as possible towards 2 is a major research open question. Corollary: never write and/or say that matrix multiplication is quadratic or quasi-optimal (unless it is proved in the future which is quite unsure): you will appear as ridiculous if you do that.*

Hence, the quest for nearly optimal algorithms for matrix multiplication is not finished. It is actually a difficult problem. We shall see in the following problem a first algorithm that can multiply two 2×2-matrices in fewer than 8 multiplications.

Problem V.3 (Winograd and Waksman's algorithms.)**.**

1. *How many (exactly) additions and multiplications are needed by the naive algorithm for matrices of size n?*

2. *Let $n = 2k$, $v = (v_1, \dots, v_n)$ et $w = (w_1, \dots, w_n)$ two vectors. Let $\sigma(v) = v_1v_2 + \cdots + v_{2k-1}v_{2k}$ and $\sigma(w) = w_1w_2 + \cdots + w_{2k-1}w_{2k}$.*

   Show that the scalar product of v by w is $(v_1 + w_2)(v_2 + w_1) + \cdots + (v_{2k-1} + w_{2k})(v_{2k} + w_{2k-1}) - \sigma(v) - \sigma(w)$.

3. *How many additions and multiplications are done with this formula?*

4. *What can you deduce for the matrix product?*

5. *We now assume that 2 is invertible in the base field (what does it mean?) and that the division by 2 is free. Let $R = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ z & t \end{smallmatrix}\right)$. How many additions and multiplications are done with the following formula: $R = \frac{1}{2}\left(\begin{smallmatrix} \alpha_1-\alpha_2 & \beta_1-\beta_2 \\ \gamma_1-\gamma_2 & \delta_1-\delta_2 \end{smallmatrix}\right)$? where $\alpha_1 = (a+z)(b+x)$, $\alpha_2 = (a-z)(b-x)$, $\beta_1 = (a+t)(b+y)$, $\beta_2 = (a-t)(b-y)$, $\gamma_1 = (c+z)(d+x)$, $\gamma_2 = (c-z)(d-x)$, $\delta_1 = (c+t)(d+y)$ and $\delta_2 = (c-t)(d-y)$.*

6. *Using $\gamma_1 + \gamma_2 + \beta_1 + \beta_2 = \alpha_1 + \alpha_2 + \delta_1 + \delta_2 = 2(ab + cd + xz + ty)$, what can you deduce on the number of additions and multiplications needed to compute the product? What do you conclude?*

7. *Can we use this algorithm recursively, like Karatsuba's?*

The first algorithm with a time complexity better than $O(n^3)$ is due to Strassen. Like for Karatsuba algorithm, it is based on an improvement in the case $n = 2$ (for Karatsuba, we were considering linear polynomials to build the general improvement) and a decrease of the number of multiplications.

So, let

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \text{and} \qquad N = \begin{pmatrix} x & y \\ z & t \end{pmatrix}.$$

To multiply M by N, compute the following quantities

$$\begin{aligned} &q_1 = a(x+z) && q_2 = d(y+t) && \\ &q_3 = (d-a)(z-y) && q_4 = (b-d)(z+t) && \\ &q_5 = (b-a)z && q_6 = (c-a)(x+y) && q_7 = (c-d)y, \end{aligned}$$

compute the sums

$$\begin{aligned} &r_{1,1} = q_1 + q_5 && r_{1,2} = q_2 + q_3 + q_4 - q_5 \\ &r_{2,1} = q_1 + q_3 + q_6 - q_7 && r_{2,2} = q_2 + q_7. \end{aligned}$$

The entry at row i and column j of the product MN is exactly $r_{i,j}$, for $1 \le i, j \le 2$.

Hence, this remark allows us to compute matrix multiplication by splitting $2n \times 2n$-matrices into blocks of size n. It performs 7 recursive calls to the matrix multiplication algorithm and $C = 18$ additions.

The recurrence formula for estimating the cost of multiplying square matrices is

$$S(n) \leq 7S\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right)^2.$$

Using Lemma IV.8 for estimating the cost of divide and conquer strategies with $m = 7$, $p = s = 2$, $\kappa = 1$ and $q = 4$, we deduce

$$S(n) \leq \left(1 + \frac{C}{3}\right) n^{\log_2 7}.$$

This leads to the following theorem (using $\log_2 7 \leq 2.81$).

Theorem V.4. *One can multiply matrices in $\mathbb{K}^{n \times n}$ using $O(n^{2.81})$ operations in $\mathbb{K}$.*

Problem V.4. *Check Strassen's formulae.*

Strassen's algorithm becomes competitive for matrices of size around 64 with entries which are integers of small size.

Problem V.5.

1. *Practice multiplying polynomials of size 4 and 6 using Karatsuba's algorithm!*

2. *Practice multiplying matrices of size 2 using Strassen's algorithm!*

Remark V.5. *Until October 2022[1], multiplying two 4×4-matrices required 49 multiplications in the based field through a two-level Strassen algorithm. In the paper mentioned above, the authors used machine learning to discover an algorithm using only 47 multiplications.*

They also improved the 5×5-matrix multiplication over $\mathbb{Z}/2\mathbb{Z}$ from 98 to 96 multiplications, though no extension to $\mathbb{Z}/p\mathbb{Z}$ for $p > 2$ prime or to $\mathbb{Q}$ is known. A few days later[2] another, non-equivalent, 4×4-matrix multiplication over $\mathbb{Z}/2\mathbb{Z}$ requiring 47 multiplications was proposed. The authors also improved the 5×5-case over $\mathbb{Z}/2\mathbb{Z}$ requiring only 95 multiplications.

Problem V.6. *What is the multiplication matrix complexities using the algorithms given in Remark V.5?*

We now write $O(n^\omega)$ for the complexity of multiplying two matrices of size n.

Problem V.7. *Let $m, p \geq n$ and assume that we want to multiply two matrices of size $m \times n$ and $n \times p$. Show that this can be done in $O(mn^{\omega-2}p)$ operations.*

V.2. Inverting matrices

In Chapter II, we saw how to compute the PLUQ decomposition of a matrix.

[1] https://www.nature.com/articles/s41586-022-05172-4
[2] https://arxiv.org/abs/2210.04045

Theorem V.6. *Let $A \in \mathbb{K}^{n\times n}$ be a matrix. Assuming A is invertible, then one can compute its PLUQ decomposition in $O(n^3)$ operations.*

Problem V.8. *Prove this statement.*

Corollary V.7. *Let $A \in \mathbb{K}^{n\times n}$ be an invertible matrix. Then, one can compute its inverse in in $O(n^3)$ operations.*

Proof. Compute in $O(n^3)$ operations the PLUQ decomposition of A. For each $1 \leq i \leq n$, let e_i be the ith vector of the canonical basis. Furthermore, the ith column of A^{-1} is the solution of the linear system $Ax = PLUQx = e_i$ which can be solved in $O(n^2)$ operation. Thus, we need $O(n^3)$ operations to solve the n systems. □

Proposition V.8. *Let us assume that there exists an algorithm that compute the inverse of an invertible matrix of $\mathbb{K}^{n\times n}$ in $O(n^\theta)$ operations. Then, one can multiply two matrices of $\mathbb{K}^{n\times n}$ in $O(n^\theta)$ operations.*

Proof. Let $A, B \in \mathbb{K}^{n\times n}$ and let Id be the identity matrix of size n. The matrix

$$T = \begin{pmatrix} \mathrm{Id} & A & 0 \\ 0 & \mathrm{Id} & B \\ 0 & 0 & \mathrm{Id} \end{pmatrix}$$

is upper triangular with 1's on the diagonal, thus it is invertible. Using the algorithm given in the hypotheses, we can compute

$$T^{-1} = \begin{pmatrix} \mathrm{Id} & -A & AB \\ 0 & \mathrm{Id} & -B \\ 0 & 0 & \mathrm{Id} \end{pmatrix}$$

in $O(n^\theta)$. Furthermore, this inverse allows us to read the product AB. □

We shall now prove that inverting a *generic* $n \times n$-matrix is not harder than multiplying two $n \times n$-matrices. The notion of *generic matrix* is subtle and we will not go too much into detail. For a matrix over $\mathbb{R}$ or $\mathbb{C}$, this is the situation that we encounter when picking it at random with coefficients in $]-1;1[\times i]-1,1[$ with a uniform distribution. Observe that having such coefficients is not restrictive, should we scale the matrix.

Theorem V.9. *Assume that there exists an algorithm for multiplying two matrices of $\mathbb{K}^{n\times n}$ in $O(n^\omega)$ operations in $\mathbb{K}$, with $2 \leq \omega \leq 3$.*

If all the encountered submatrices in Algorithm 6 are invertible, then one can invert A in $\tilde{O}(n^\omega)$ operations in $\mathbb{K}$.

Proof. Clearly the algorithm terminates. Let us prove that it is correct. Using the identity

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ ca^{-1} & 1 \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & Z \end{pmatrix} \begin{pmatrix} 1 & a^{-1}b \\ 0 & 1 \end{pmatrix},$$

```
Algorithm 6: Strassen's inversion algorithm
Input: A generic matrix $A \in \mathbb{K}^{n\times n}$, where $n = 2^k$.
Output: Its inverse $A^{-1}$.
If $n = 1$ then Return $A^{-1}$.
Split $A = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ with $a, b, c, d \in \mathbb{K}^{\frac{n}{2}\times\frac{n}{2}}$.
Compute $e = a^{-1}$ recursively.
Compute $Z = d - ceb$.
Compute $t = Z^{-1}$ recursively.
Compute $y = -ebt$, $z = -tce$, $x = e + ebtce$.
Return $\left(\begin{smallmatrix} x & y \\ z & t \end{smallmatrix}\right)$.
```

where $Z = d - ca^{-1}b$ is Schur's complement, we conclude that

$$\begin{aligned} A^{-1} &= \begin{pmatrix} 1 & -a^{-1}b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a^{-1} & 0 \\ 0 & Z^{-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -ca^{-1} & 1 \end{pmatrix} \\ &= \begin{pmatrix} a^{-1} + a^{-1}bZ^{-1}ca^{-1} & -a^{-1}bZ^{-1} \\ -Z^{-1}ca^{-1} & Z^{-1} \end{pmatrix}. \end{aligned}$$ □

Problem V.9. *Prove the complexity statement.*

Remark V.10. *To the contrary of the Quicksort algorithm, which falls in the case $q = m$ of Theorem IV.9, and of Karatsuba and Strassen multiplication algorithms, which fall in the case $q < m$ of Theorem IV.9, Algorithm 6 falls in the case $q > m$.*

Corollary V.11. *Assume that there exists an algorithm for multiplying two matrices of $\mathbb{K}^{n\times n}$ in $O(n^\omega)$ operations in $\mathbb{K}$, with $2 \le \omega \le 3$.*

Let $A \in \mathbb{K}^{n\times n}$ be a generic matrix and $b \in \mathbb{K}^n$. One can solve the linear system $Ax = b$ in $O(n^\omega)$ operations in $\mathbb{K}$.

Likewise, one can compute $\det A$ in $O(n^\omega)$ operations in $\mathbb{K}$.

Proof. It suffices to compute A^{-1} using Algorithm 6 in this complexity and then to multiply A^{-1} with b in $O(n^2)$, which is not the bottleneck. This proves the first statement. □

Problem V.10. *Prove the second statement of Corollary V.11 by tweaking Algorithm 6 so that it computes $\det A$ in $O(n^\omega)$ operations.*

Implementation V.1. *Implement Karatsuba multiplication for polynomials over $\mathbb{Z}/p\mathbb{Z}$. Find experimentally the best size of polynomials to stop the recursive calls and instead rely on the already implemented naive algorithm.*

Implementation V.2. *Improve your Karatsuba algorithm implementation so that two of the three recursive calls can store their partial results in the allocated memory for the global result.*

VI. Structured linear algebra I: evaluation and interpolation

VI.1. Polynomial evaluation

Given a polynomial $F = f_{d-1}x^{d-1} + \cdots + f_0$, what is the cost of evaluation F at a given point x?

This can be done in $O(d)$ arithmetic operations using Horner's scheme:

$$F(x) = (\cdots((f_{d-1} \cdot x + f_{d-2}) \cdot x + f_{d-1}) \cdot \cdots \cdot x + f_1) \cdot x + f_0.$$

Problem VI.1. *Assuming the polynomial has integer coefficients, $\sigma = \operatorname{bitsize} x$ and $\tau = \max_{0 \le i \le d-1} \operatorname{bitsize} f_i$, what is the bit complexity of Horner's scheme? As preliminaries questions, you may look for the bitsize of a product $\prod_{i=1}^n f_i$ and a sum $\sum_{i=1}^n f_i$.*

The *multipojnt evaluation* problem is the problem of evaluating F in distinct $x_0, \ldots, x_{d-1}$.

Problem VI.2. *Give and prove a naive bound of the arithmetic complexity of the multipoint evaluation of a polynomial F of degree $d-1$ in d distinct points.*

From a linear algebra point of view, the multipoint evaluation problem consists in performing the product

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{d-1} \\ 1 & x_1 & \cdots & x_1^{d-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{d-1} & \cdots & x_{d-1}^{d-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{d-1} \end{pmatrix}.$$

Problem VI.3. *What is*

1. *the size of the input in the multipoint evaluation problem?*
2. *the size of the output in the multipoint evaluation problem?*
3. *the cost of modeling the multipoint evaluation problem through linear algebra?*

VI.2. Lagrange interpolation

The *interpolation* problem is the inverse of the multipoint evaluation problem. Given d distncts elements $x_0, \ldots, x_{d-1} \in \mathbb{K}$, a field, and d elements $y_0, \ldots, y_{d-1} \in \mathbb{K}$, the goal is to find a polynomial $F \in \mathbb{K}[x]$ such that

$$\begin{cases} F(x_0) & = y_0 \\ & \vdots \\ F(x_{d-1}) & = y_{d-1}. \end{cases}$$

In other word, we know the evaluation of F in d points and we want to recover F from this information.

Observe that F is not uniquely defined: if such a F exists, then for any polynomial Q, the polynomial

$$F + Q \prod_{i=0}^{d-1} (x - x_i)$$

also satisfies the above system.

Lemma VI.1. *If a polynomial F of degree at most $d-1$ satisfies the above system, then it is unique.*

Proof. Assume G satisfies the above system and $\deg G < d$. Then, the polynomial $F-G$ vanishes on $x_0, \dots, x_{d-1}$ and has degree less than d. However, a nonzero polynomial of degree $\delta \geq 0$ has at most δ roots, which contradicts the fact that $F - G$ has degree $\delta < d$ and has d roots. Hence, $F - G = 0$ and $G = F$. □

Assuming $\deg F < d$, then this can be written with a *Vandermonde matrix*, as

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{d-1} \\ 1 & x_1 & \cdots & x_1^{d-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{d-1} & \cdots & x_{d-1}^{d-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{d-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{d-1} \end{pmatrix}.$$

Theorem VI.2. *Let $x_0, \dots, x_{d-1}$ be d elements in $\mathbb{K}$. Then, the determinant of the Vandermonde matrix associated to $x_0, \dots, x_{d-1}$ is*

$$\prod_{0 \leq i < j \leq d-1} (x_i - x_j).$$

In particular, the matrix is invertible if, and only if, $x_0, \dots, x_{d-1}$ are pairwise distinct.

Problem VI.4. *Prove that the interpolation problem is indeed a linear algebra problem and compute the determinant of the matrix of this linear system.*

While this system can be solved with linear algebra techniques, these yield a complexity which is $O(d^3)$, assuming naive linear alegbra routines and $O(d^\omega)$ at best. However, we can notice that this matrix is particular: its entries only depend on d parameters. We shall see that thanks to that, we can actually solve this problem faster.

The main idea of the Lagrange interpolation is to compute d Lagrange polynomials $L_0, \dots, L_{d-1}$ which have a useful property:

$$L_i(x_i) = 1, \quad \forall j \neq i,\ L_i(x_j) = 0.$$

If these polynomials do exist, then we can notice that the polynomial

$$\sum_{i=0}^{d-1} y_i L_i$$

is solution of the interpolation problem. It remains to prove that such $L_0, \ldots, L_{d-1}$ exist, can be computed efficiently and have degree at most $d-1$ so that the polynomial above is the target solution.

Lemma VI.3. *Let $x_0, \ldots, x_{d-1}$ be d distinct elements in $\mathbb{K}$. Let*

$$L_i = \prod_{\substack{1 \le j \le d-1 \\ j \ne i}} \frac{x - x_j}{x_i - x_j}.$$

Then, $L_i(x_i) = 1$, for $j \ne i$, $L_i(x_j) = 0$ and $\deg L_i = d - 1$.

Theorem VI.4. *Let $x_0, \ldots, x_{d-1}$ be d distinct elements in $\mathbb{K}$. Let $y_0, \ldots, y_{d-1}$ in $\mathbb{K}$.*

There exists a unique polynomial F of degree at most $d-1$ such that

$$F(x_0) = y_0, \ldots, \; F(x_{d-1}) = y_{d-1}$$

and this polynomial is

$$\sum_{i=0}^{d-1} y_i \prod_{\substack{0 \le j \le d-1 \\ j \ne i}} \frac{x - x_j}{x_i - x_j}.$$

It can be computed with Algorithm 7 in $O(d^2)$ operations.

```
Algorithm 7: LagrangeAlgorithm
Input: A list $[x_0, \ldots, x_{d-1}]$ of pairwise distinct elements in a field $\mathbb{K}$ and a list
       $[y_0, \ldots, y_{d-1}]$ of elements in $\mathbb{K}$
Output: The polynomial $F$ of smallest degree such that $F(x_0) = y_0, \ldots, F(x_{d-1}) = y_{d-1}$
$P := 1$.
For $i$ from $0$ to $d-1$ do $P := P \cdot (x - x_i)$
For $i$ from $0$ to $d-1$ do
    $L_i := \frac{P}{x - x_i}$
    $L_i := \frac{L_i}{L_i(x_i)}$
$F := 0$
For $i$ from $0$ to $d-1$ do $F := F + y_i L_i$
Return $F$
```

Problem VI.5. *Prove the complexity statement in Theorem VI.4.*

Problem VI.6. *Practice interpolating polynomials.*

Problem VI.7. *Let us assume that new fast algorithms for linear algebra are designed, say for multiplying, inverting, solving, etc. These algorithms could even be quasi-optimal.*

How would solving the interpolation problem with a linear algebra modeling compared to solving it with Lagrange interpolation?

Problem VI.8. *Assume that F has been interpolated thanks to $x_0, \dots, x_{d-1}, y_0, \dots, y_{d-1}$. Assume that x_d, distinct from $x_0, \dots, x_{d-1}$, and y_d are given.*

1. *Give an algorithm to interpolate G such that $G(x_0) = y_0, \dots, G(x_{d-1}) = y_{d-1}, G(x_d) = y_d$.*
2. *What is its complexity?*
3. *Assume now that (x_i, y_i) are always given by one by one and a polynomial interpolating $(x_0, y_0), \dots, (x_i, y_i)$ is computed at each step (what we call an* online algorithm*). What is the complexity of this algorithm for $(x_0, y_0), \dots, (x_d, y_d)$. Compare with Lagrange's interpolation.*
4. *Modify the step in the for loop to make it linear.*

VII. Structured linear algebra II and sparse linear algebra: guessing linear recurrence relations

VII.1. The Berlekamp–Massey algorithm

Definition VII.1. *A sequence* $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$ *with terms in a field* $\mathbb{K}$ *is said to be* linear recurrent of order d *if there exist* $v_0, \dots, v_{d-1} \in \mathbb{K}$ *such that*

$$\forall\, i \in \mathbb{N}, \quad b_{i+d} + v_{d-1}b_{i+d-1} + \cdots + v_0 b_i = 0.$$

If there exists $d \in \mathbb{N}$ *such that* $\mathbf{b}$ *is linear recurrent of order* d*, then* $\mathbf{b}$ *is said* linear recurrent.

Example VII.2.

1. *For* $d = 1$, $\mathbf{b}$ *is a geometric sequence of common ratio* $-v_0$ *if it satisfies for all* $i \in \mathbb{N}$, $b_{i+1} + v_0 b_i = 0$.
2. *The sequence* $\mathbf{b} = (i)_{i\in\mathbb{N}}$ *satisfies* $b_{i+1} - b_i - 1 = 0$ *for all* $i \in \mathbb{N}$*. Notice that the constant term* -1 *prevents us from concluding that the sequence is linear recurrent of order* 1*. In fact, it is recurrent of order* 2*, since it also satisfies for all* $i \in \mathbb{N}$, $b_{i+2} - 2b_{i+1} + b_i = 0$.

Problem VII.1. *Show that for any linear recurrent sequence* $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$ *with coefficients in* $\mathbb{K}$*, there is a unique smallest integer* d *and unique* $v_0, \dots, v_{d-1} \in \mathbb{K}$ *such that for all* $i \in \mathbb{N}$,

$$b_{i+d} + v_{d-1}b_{i+d-1} + \cdots + v_0 b_i = 0.$$

Problem VII.2. *Show that the sum of two linear recurrent sequences is linear recurrent.*

Question VII.1. *For a sequence* $\mathbf{b}$ *whose terms are not given by a formula, testing that a linear recurrence relation is satisfied by* $\mathbf{b}$ *is not possible.*

Guessing a linear recurrence relation *satisfied by* $\mathbf{b}$ *is the problem of computing a suitable linear recurrence relation based on finitely many terms of* $\mathbf{b}$.

Since it is impossible to know all the sequence terms, we assume that we know the D first terms, $b_0, \dots, b_{D-1}$, of the input sequence. Computing a linear recurrence relation satisfied by $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$ comes down to computing the smallest integer d and unique $v_0, \dots, v_{d-1}$ such that

$$\begin{cases} v_0 b_0 + \cdots + v_{d-1}b_{d-1} + b_d & = 0 \\ v_0 b_1 + \cdots + v_{d-1}b_d + b_{d+1} & = 0 \\ & \vdots \\ v_0 b_{D-1-d} + \cdots + v_{d-1}b_{D-2} + b_{D-1} & = 0. \end{cases}$$

Thus, we look for the smallest d such that the *Hankel matrix*

$$\begin{pmatrix} b_0 & \cdots & b_{d-1} & b_d \\ b_1 & \cdots & b_d & b_{d+1} \\ \vdots & & \vdots & \vdots \\ b_{D-1-d} & \cdots & b_{D-2} & b_{D-1} \end{pmatrix}$$

has a kernel of dimension 1 spanned by a vector $\begin{pmatrix} v_0 \\ \vdots \\ v_{d-1} \\ 1 \end{pmatrix}$.

We can notice that this system is *structured*: knowing the first row and the last column of the matrix is enough to complextely know the matrix. Thanks to this, we can find efficiently a vector in the kernel, *without assuming the value of d*! To do so, we present the so-called Berlekamp-Massey algorithm.

```
Algorithm 8: Berlekamp-Massey
Input: The first $D$ terms $b_0, \dots, b_{D-1}$ of a sequence.
Output: A linear recurrence relation of smallest order satisfied by these terms.
$R := 1$, $F := 0$, polynomials.
$d := 0$, $s := 0$, $f := -1$, degrees.
For $i$ from $0$ to $D-1$ do
    $s := i - d$.
    $e := r_d b_i + \cdots + r_0 b_{i-d}$.
    If $e \neq 0$ then
        If $s \leq f$ then $R := R - eFx^{f-s}$.
        Else
            $R, F := Rx^{s-f} - eF, R/e$.
            $d := d + s - f$.
            $f := s$.
Return $R$.
```

Theorem VII.3. *Line 8 terminates, is correct and computes a minimal recurrence relation in $O(D^2)$ operations.*

Problem VII.3. *Prove Theorem VII.3.*

Example VII.4. *Let us consider the Fibonacci sequence and more precisely its first 10 terms.*

- $i = 0$, $e = b_0 = 0$.
- $i = 1$, $e = b_1 = 1$, $R = x^2$, $F = 1$.
- $i = 2$, $e = b_2 = 1$, $R = x^2 - x$.

- *$i = 3$, $e = b_3 - b_2 = 1$, $R = x^2 - x - 1$.*
- *$i = 4$, $e = b_4 - b_3 - b_2 = 0$.*
- *etc.*

Problem VII.4. *Let $\mathbf{u} = (u_i)_{i\in\mathbb{N}} = (2^i + i^2 - 1)_{i\in\mathbb{N}}$ be a sequence defined over $\mathbb{Z}/7\mathbb{Z}$.*

1. *Show that $\mathbf{u}$ satisfies the linear recurrence relation for all $i \in \mathbb{N}$, $u_{i+21} - u_i = 0$.*
2. *Guess a relation of order 4 satisfied by $\mathbf{u}$, using only its first 10 terms.*

VII.2. Application to sparse matrices

A matrix is said to be *sparse* if many of its coefficients are zero. In general, we consider that a family $M_n \in \mathbb{K}^{n\times n}$ is *sparse* if the number of nonzero entries of M_n is in $O(n^{1+\varepsilon})$ with $\varepsilon < 1$.

Definition VII.5. *A* sparse representation *of a matrix is such that only nonzero entries are stored.*

Example VII.6. *A matrix M is stored a list of triplets $(a_{i,j}, i, j)$ with $a_{i,j} \neq 0$.*

Over $\mathbb{F}_2$, we even can avoid writing the nonzero coefficient, since it is 1! The M4RI library proposes the JCF format based on that and a text input must follow this template

```
nrows ncols modulus
nonzero_entries_upper_bound
column_index
```

where `nrows` (resp. `ncols`) is the number of rows (resp. of columns) of the matrix, `modulus` is the modulus (always 2, here) and `nonzero_entries_upper_bound` is an upper bound on the number of nonzero entries of the matrix. This format does not allow the matrix to have a zero row. The number `column_index` is the column index of the coefficient; the row index is implicit, it is increased by 1 each time `column_index` is negative.

```
3 7 2
8

-2
4
7
-1
2
3
-2
3
```

represents the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

VII.2.1. Multiplication

Given a matrix with m nonzero entries, its product with a vector requires $O(m)$ operations in the base field.

Problem VII.5. *Prove this assertion.*

For matrix product, we can expect likewise that the complexity depends on m, especially if both matrices have $O(m)$ nonzero entries. Yet, if they have size n, then the Coppersmith–Winograd algorithm still has complexity $O(n^{2.38})$. However, the naive algorithm can take advantage of the sparsity of the matrices, as long as the product $a_{i,k}b_{k,j}$ is not computed whenever $a_{i,k} = 0$ or $b_{k,j} = 0$ (which is naturally the case with a sparse representation). Let $\bar{a}_k$ (resp. $\bar{b}_k$) be the number of nonzero entries of the kth column (resp. row) of A (resp. B), then the number of products to do is $\sum_{k=1}^n \bar{a}_k \bar{b}_k$. Since $\bar{b}_k \leq n$, then $\sum_{k=1}^n \bar{a}_k \bar{b}_k \leq (\sum_{k=1}^n \bar{a}_k) n \leq mn$.

In general, sparse matrices are also structured matrices. To optimize the complexity, we need an algorithm taking advantage of this structure.

VII.2.2. Sparse linear system solving

The common strategy to solve a general linear system is to compute the PLU decomposition of the matrix and then to solve two triangular systems.

PLU decomposition

We know that for dense matrices, PLU decomposition and inversion have computation complexities equivalent to that of matrix product. However, a sparse matrix can have a dense PLU decomposition

The family $(M_n)_{n\in\mathbb{N}}$ of sparse matrices

$$M_n = \begin{pmatrix} 1 & 2 & 2 & \cdots & 2 \\ 2 & 1 & 0 & \cdots & 0 \\ 2 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 2 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

has the following LU decomposition

$$M_n = L_n U_n$$

$$= \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ 2 & \frac{4}{3} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ 2 & \frac{4}{3} & \cdots & \frac{4}{4n-9} & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 2 & 2 & \cdots & 2 \\ & -3 & -4 & -4 & \cdots & -4 \\ & & \frac{7}{3} & \frac{4}{3} & \cdots & \frac{4}{3} \\ & & & \frac{11}{7} & \ddots & \vdots \\ & & & & \ddots & \frac{4}{4n-13} \\ & & & & & \frac{4n-5}{4n-9} \end{pmatrix}.$$

Yet, we can obtain a sparse PLU decomposition, swapping the first and last rows.

The Wiedemann algorithm

The goal of this algorithm is to efficiently compute a nonzero vector in the kernel of a matrix. This is a probabilistic algorithm.

In all this subsection, $M \in \mathbb{K}^{n\times n}$ is a sparse matrix. We let x_0 be a randomly picked vector of $\mathbb{K}^n$ and we let $x = Mx_0$. We also pick at random a vector $y \in \mathbb{K}^n$.

The matrix M has a minimal polynomial $P = z^r + p_{r-1}z^{r-1} + \cdots + p_0$ of degree $r \leq n$, that is, there exist $r \in \mathbb{N}$ minimal and $p_0, \ldots, p_{r-1} \in \mathbb{K}$ such that

$$M^r + p_{r-1}M^{r-1} + \cdots + p_0 \operatorname{Id} = 0.$$

Multiplying on the right this equality by x, we obtain $s_r + p_{r-1}s_{r-1} + \cdots + p_0 s_0 = 0$, where $s_i = M^i x$. We denote $\mathbf{s} = (s_i)_{i\in\mathbb{N}} = (x, Mx, M^2x, \ldots) = (M^i x)_{i\in\mathbb{N}}$. The terms of this vector sequence are computed recursively, $s_0 = x$ and for all $i \in \mathbb{N}$, $s_{i+1} = Ms_i$.

Since $x = Mx_0$, then

$$s_r + p_{r-1}s_{r-1} + \cdots + p_0 s_0 = M(M^r x_0 + p_{r-1}M^{r-1}x_0 + \cdots + p_0 x_0) = 0.$$

Hence, $M^r x_0 + p_{r-1}M^{r-1}x + \cdots + p_0 x_0$ is a vector in the kernel of M. Unfortunately, this vector is the zero one.

Let us assume, however, that there exists $d \in \mathbb{N}$ such that $d < r$ and $q_0, \ldots, q_{d-1} \in \mathbb{K}$ such that

$$\begin{aligned} s_d + q_{d-1}s_{d-1} + \cdots + q_0 s_0 = M(M^d x_0 + q_{d-1}M^{d-1}x_0 + \cdots + q_0 x_0) = 0 \\ M^d x_0 + q_{d-1}M^{d-1}x_0 + \cdots + q_0 x_0 \neq 0. \end{aligned}$$

Then, $M^d x_0 + q_{d-1}M^{d-1}x_0 + \cdots + q_0 x_0$ is a nonzero vector in the kernel of M.

Wiedemann's idea is to compute these $q_0, \ldots, q_{d-1}$ by multiplying first this equation on the left by $y^{\mathrm{T}}M^i$ for all i. We, thus, obtain

$$\forall i \in \mathbb{N}, \quad b_{i+d} + q_{d-1}b_{i+d-1} + \cdots + q_0 b_i = 0,$$

where $b_i = y^{\mathrm{T}}M^i x$ and $\mathbf{b} = (b_i)_{i\in\mathbb{N}} = (y^{\mathrm{T}}x, y^{\mathrm{T}}Mx, y^{\mathrm{T}}M^2x, \ldots) = (y^{\mathrm{T}}M^i x)_{i\in\mathbb{N}}$. These sequence terms can be computed as follows $b_i = y^{\mathrm{T}}s_i$.

In other words, the sequence $\mathbf{b}$ satisfies the *linear recurrence relation* for all i, $b_{i+d} + q_{d-1}u_{i+d-1} + \cdots + q_0 b_i = 0$.

To compute these d and $q_0, \dots, q_{d-1}$, we use the Berlekamp–Massey algorithm which determines the linear recurrence relation of smallest order satisfied by the sequence.

Theorem VII.7. *The Wiedemann algorithm called on a sparse matrix of size n with at most $m \geq n$ nonzero entries requires at most $O(nm)$ operations in the base field.*

Proof. The computation of the sequence $\mathbf{s}$ is done in $O(nm)$ operations while the one of the sequence $\mathbf{b}$ is done in $O(n^2)$ operations. The call to the Berlekamp–Massey algorithm requires at most $O(n^2)$ operations.

All in all, the total complexity is $O(n \max(m, n))$. Since $m \geq n$, building the sequence is the bottleneck of the algorithm. □

Problem VII.6. *We want to solve the linear system over $\mathbb{F}_{13}$*

$$Av = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = b.$$

1. *Show that this comes down to computing a vector, whose coordinates depend on v_1, v_2 and v_3, in the kernel of*
$$A' = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

2. *Compute the vector sequence of the Wiedemann algorithm for $x_0 = \left(\begin{smallmatrix} 11 \\ 11 \\ 10 \\ 10 \end{smallmatrix}\right)$ and the scalar sequence for $y = \left(\begin{smallmatrix} 4 \\ 12 \\ 3 \\ 12 \end{smallmatrix}\right)$.*

3. *Determine the smallest linear recurrence relation satisfied by $\mathbf{u}$.*

4. *Deduce a potential vector in the kernel of A'.*

5. *Check that this vector is indeed in the kernel of A'.*

6. *Deduce the solution of the original system.*

Problem VII.7. *Let $\mathbf{u} = (u_i)_{i \in \mathbb{N}}$ be a sequence and let $H_{m,n} = (h_{i,j})_{\substack{0 \leq i \leq m \\ 0 \leq j \leq n}} = (u_{i+j})_{\substack{0 \leq i \leq m \\ 0 \leq j \leq n}}$ a Hankel matrix with $m \geq n$. We consider the property $\mathcal{P}$: "the last column of the matrix is linearly independant from the previous ones".*

Let us assume that $H_{m,n}$ has rank $n-1$ and does not satisfy property $\mathcal{P}$. Let us assume that the matrix $H_{m+1,n}$ does satisfy $\mathcal{P}$ and thus has rank n.

Show that all matrices $H_{m+1-p,n+p}$, for $0 \leq p \leq m-n+1$, satisfy $\mathcal{P}$ but that $H_{n-1,m+2}$ does not.

We can start checking that this is indeed the case for the sequence $\mathbf{u} = (0, 1, 1, 2, 3, 5, 9, \dots)$ for $n = 2$ and $m = 3$.)

Implementation VII.1. *Implement the sparse linear system solving algorithm using the Wiedemann algorithm.*

Estimate its probability of success depending for small primes p and sizes n.