Computational Mathematics For Learning and Data Analysis

Numerical Methods

Based on prof. Federico Giovanni Poloni's lectures

Gemma Martini

December 14, 2021

A sincere thank you to Alessandro Cudazzo,
Donato Meoli, Giulia Volpi, Ivan Grujic and all those
who helped me improving these notes in style and contents.

Contents

Chapter 1

Mathematical background for Numerical Methods

In the following chapters we will cover the part of Numerical Analysis of the course of Computational Mathematics for Data Learning and Analytics. This part of the course is held by Professor Federico Giovanni Poloni. The content of these chapters is based primarily on the material provided by the professor on the e-learning portal. In addition to that material, the introduction is based on the content of the course Linear Algebra held by Prof. Gilbert Strang from MIT. There is a nice portal called MIT OpenCourseWare where you can find a lot of recorded lectures from MIT. Linear algebra course is just super duper fantastic.

1.1 A brief journey in Linear Algebra

What is linear algebra? Linear algebra solves systems of linear equations. Period. But there is a whole world there in. So better start right now.

What is a system of equations? Well, simply put, it is a bunch of linear equations, say m, each of which is characterized by at most n unknowns, i.e. variables. Since there are two dimensions involved here, that is m and n, there are actually two points of view to look at this system. We can look at it from the row perspective and from the column perspective. Let us go through an example to show this.

Example 1.1.1. *Suppose we are given the following system of two linear equations with two unknowns:*

$$\begin{cases} -4x + 2y = 2 \\ 3x + 5y = -3 \end{cases}$$

We can write this system of linear equations in the following matrix form:

$$\begin{bmatrix} -4 & 2 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$$

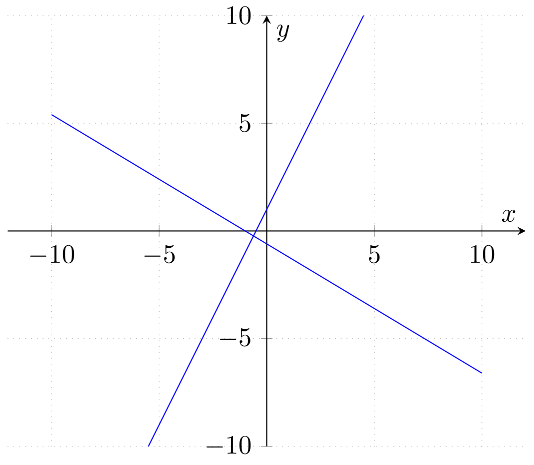


Figure 1.1: Linear system

Usually the coefficient matrix is indicated with A, the vector of unknowns is indicated with $\mathbf{x}$, and the vector of coefficients on the right hand side is usually indicated with $\mathbf{b}$.

The row perspective is the one that we are mostly used to deal with. Each equation in this example represents a line in the Cartesian plane and the problem requires to find the point (if it exists) where the two lines meet (see figure 1.1).

The column perspective is a bit different and may be completely new to you. Look at this mind blowing thing:

$$x\begin{bmatrix}-4\\3\end{bmatrix}+y\begin{bmatrix}2\\5\end{bmatrix}=\begin{bmatrix}2\\-3\end{bmatrix}$$

Why is column perspective interesting? Because now we can look at these columns of A as vectors in a column space. We are eager to find the right amount (unknowns) of each of these column vectors that produce the vector $\mathbf{b}$ (see figure 1.2).

Linear combination is the fundamental operation of the whole course. Besides the question of what is the linear combination of the columns of A that produce $\mathbf{b}$, we will also ask ourselves what are all the possible vectors $\mathbf{b}$ that we can obtain with some linear combination of the columns of A? Or what are all the possible linear combinations of the columns of A that give a certain $\mathbf{b}$?

In the previous example we have encountered a bunch of new terms. Let us now define them rigorously. We saw that the generic system of linear equations can be written as:

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

Note that we are actually abusing terminology here. Even though it is not that

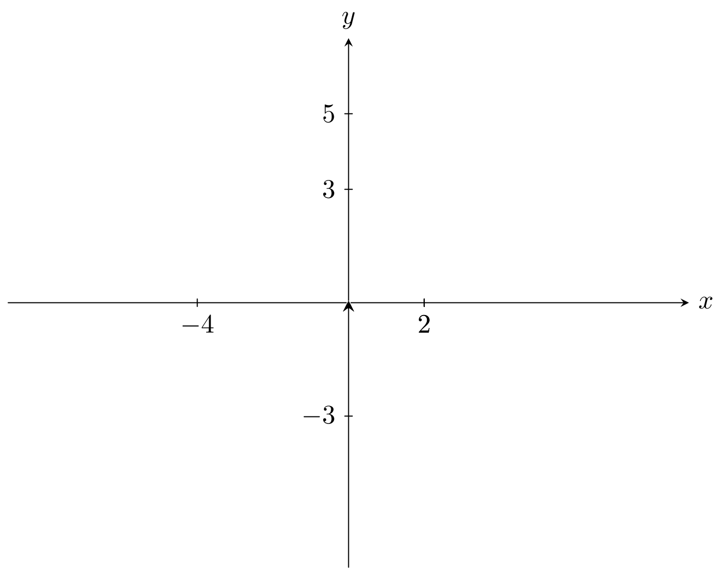


FIGURE 1.2: Linear system 2

important, bare in mind that these are actually **affine equations**. The linear equations are the equations that go through the origin, while affine need not to.

We have seen that there is a row perspective and a column perspective. These perspectives are actually two different Euclidean spaces. The first is called *row space* and the second one is the *column space*. Note how these correspond directly to the rows and columns of A respectively. When the matrix is a square matrix the dimension of the two spaces is the same (if certain properties hold). But when we have $m \times n$ matrices, then the two spaces have different dimensions (if the same properties hold). Remember all these spaces, because it is important! Especially the column space.

Clearly, two equations in two unknowns is a very basic case. Let us see another basic case before moving to definitions.

Example 1.1.2. *Let us now take three equations in three unknowns.*

TODO: I need a tikz magician here...

Now that we have said that the linear system of equations can also be seen as a bunch of vectors in the n dimensional space, we can think of solving a linear system of equations as the problem of finding a linear combination of columns of the matrix A to obtain the vector $\mathbf{b}$. Thus, given a vector $\mathbf{b}$, how can we obtain it (if it is possible at all) as a linear combination of the columns of A? If you think a bit, the vector $\mathbf{b}$ is yet another vector in the n dimensional space. Think of the columns of A as directions, i.e. n directions. You are in the origin of the space, and you want to get to the point pointed by $\mathbf{b}$. You may move in each of n directions only once and in every direction you can choose how much you want to move. You may move backwards (corresponding to negative coefficient in the linear combination) or forward (corresponding to positive coefficient in the linear combination). A zero coefficient means that you do not move in the corresponding direction. The problem is: given these n direction, can you get from the origin to the point $\mathbf{b}$ whatever $\mathbf{b}$ you are given? This basically means that these n direction must somehow "fill" the whole space, that is for every point b there must be a linear combination of directions that bring you from the origin to $\mathbf{b}$. So, is there a way to know whether a given set of columns is "sufficiently powerful" to be able to generate all the "treasury maps"? Here comes the concept of linear independence.

If the columns of A are **linearly independent**, then every vector $\mathbf{b}$ on the right hand side of the equation $A\mathbf{x} = \mathbf{b}$ can be generated with some combination of columns of A. What does it mean to be linearly independent? It means that when you take one of the n columns, you cannot find a linear combination of the remaining $n-1$ columns of A that can generate the one that you have picked. Geometrically speaking, $n-1$ columns in an n dimensional space generate a so called subspace. In an n dimensional space there are n types of subspaces: a zero subspace, the subspaces of 1 dimension, the subspaces of 2 dimensions, ..., the subspaces of $n-1$ dimensions, and finally a subspace of n dimensions. Note singularity and plurality in the previous sentence. There is only one subspace of zero dimension, namely the origin. There is only one subspace of n dimensions in an n dimensional space, the space it self. The remaining subspaces are all of dimensions 1 to $n-1$. A subspace of dimension 1 is a line. A subspace of dimension 2 is a plane. A subspace of dimension 3 is the three dimensional space. A subspace of dimension 4... well you picture it, I'm not able. So $n-1$ vectors in the n dimensional space generate a subspace (think of a plane subspace in the three dimensional space). If the vector that we have picked from A is located on the very same plane generated by the $n-1$ vectors, then there is no way that these $n-1$ vectors can generate a vector that is outside of this plane, because there is no direction that points outside of the plane itself. The vector that we have picked and that is on the plane is then linearly dependent from the $n-1$ remaining ones. Clearly, we may have more than one vectors that are linearly dependent from the rest in A. The obvious question now is what is the subspace that the columns of A generate? In other words what is the maximum number of linearly independent columns of A? In linear algebra this is the so called rank of the matrix. When the rank is n then the matrix is said to be full rank. Given this definition, the rank of the matrix then satisfy the following box constraint:

$$0 \leq \texttt{rank}(A) \leq n$$

for some $A \in M(n, \mathbb{R})$. Since we have two dimensions, namely rows and columns, each of which generates its own space, namely row space and column space, you may legitimately wonder if these are the same. Well, in case of square matrices $n \times n$ the two are the same. So the rank of row space is the same of the column space. You want a proof? Search rank of a matrix on Wikipedia for three different proofs.

Let us now turn back to the rows. The question for you is: what happens if we do instead:

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

where $\mathbf{x}^T$ is a row vector, so we have:

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$$

The row vector b is the linear combination of the rows of A where the weights are given by the vector x. Namely:

$$x_1 \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \end{bmatrix} + \dots + x_n \begin{bmatrix} a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$$

Everything we said about columns, linear independence, rank, etc, holds for rows and row space of the matrix A. Basically, when we do operations from the left of the matrix A then we are operating on rows of the matrix, when instead we are operating from the right then we are performing operations over columns of A.

Now that we know hat linearly independent columns of A can generate all the vectors in the corresponding n dimensional column space, the question is how do we generate it? In other words, given a vector $\mathbf{b}$ how do we find the coefficients of the linear combination that generate it? Note the linear combination and not a linear combination. There is only one linear combination that generates it. Why? Picture the row space. Each equation in the linear system generates a subspace of dimension $n-1$. So in three dimensional space the equations represent planes. In case of a system of n linear equations in n unknowns, we have n subspaces of dimension $n-1$ in the n dimensional row space. So if these n subspaces are linearly independent, then it must be the case that there is only one point that touches them all when they intersect. If there is more than one point that is in the intersection of all of them, then the matrix A is not full rank. Suppose then that the columns (or rows) are linearly independent, how do we find the coefficients? Note this basic math:

$$ax = b \iff \frac{1}{a}x = \frac{b}{a} \iff x = a^{-1}b$$

where every term is a simple scalar. So to get x we must multiply b with the inverse of a. The same happens with the matrices and vectors. We need something called **inverse matrix** of A, indicated with A^{-1}:

$$\mathbf{x} = A^{-1}\mathbf{b}$$

The matrix A is invertible only if it has full column rank. This is why the linear independence is so central to the whole linear algebra. Note that only square matrices may have the inverse matrix. Obviously, the inverse matrix is unique. And moreover the inverse of the inverse is the initial matrix, namely:

$$(A^{-1})^{-1} = A$$

Cool man. But how the heck are we going to find the inverse? There are many (not *that* many) algorithms that can be used for matrix inversion. But one of the standard one is the Gauss-Jordan Elimination algorithm. Are we going to cover it here? Why not? It is one of the most used algorithms in linear algebra so let us give it some love. But before doing this algorithm, we still need one ingredient: matrix multiplications.

1.1.1 Matrix multiplications: four flavors plus one of looking at it

Matrix multiplication is another of the most important operations that you can think of when you think of linear algebra. In here, we are going to cover briefly four different ways of looking at this important operation.

We start with the basic definition that you probably have seen if you have ever done some linear algebra course. The classical definition is:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \dots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix}$$

where:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

But as we all know, there is more than one of everything. Another way to look at matrix-matrix multiplication is the following. We have seen that matrix vector multiplication is basically a linear operation that send a certain vector $\mathbf{x}$ into another vector $\mathbf{y}$. Thus, $A\mathbf{x} = \mathbf{y}$ means that $\mathbf{x}$ is mapped to $\mathbf{y}$ through A. When it comes to the matrix matrix multiplication, i.e. $AB = C$, the same story happens. It is the columns of B that are mapped to columns of C. Each column i of B (denoted as B^i) is mapped to the corresponding column i of C. Similarly, we could think of all of this from the row space perspective. The rows of A are mapped to the corresponding rows of C through B. So these are the additional two ways to look at matrix matrix multiplication. What about the fourth one?

The fourth way to look at matrix-matrix multiplication is a bit more mind blowing. In the first way, we have computed the dot products between rows of A and columns of B. But nobody prevents us from doing the contrary. What if we used the columns of A and the rows of B:

$$\begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{ni} \end{bmatrix} \begin{bmatrix} b_{i1} & b_{i2} & \dots & b_{in} \end{bmatrix}$$

What do we get for each $i = 1, \dots, n$? We get an $n \times n$ matrix. So the resulting matrix C is the summation of n different matrices of size $n \times n$.

1.2 Formal definitions

At this point, we are ready to introduce the formality of linear algebra, that we will use throughout the whole course:

- **Vector-Scalar product**:
  Let $\mathbf{x} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$ we call **multiple** of vector $\mathbf{x}$ the following vector:
  $$\lambda\mathbf{x} = \mathbf{x}\lambda = \begin{pmatrix} \lambda x_1 \\ \vdots \\ \lambda x_n \end{pmatrix}$$

- **Vector-Vector product**:
  Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, the dot product between these two vectors is defined as $\mathbf{x}^T\mathbf{y} = \sum\limits_{i=1}^{n} x_i y_i$. Note in particular that $\mathbf{x}^T\mathbf{y} \in \mathbb{R}$. In other words, the dot product produces a scalar.

- **Scalar-Matrix product**:
  Let $A \in M(n, m, R)$ and $\lambda \in \mathbb{R}$ we call the **scalar-matrix product** the following:
  $$\lambda A = A\lambda = \begin{pmatrix} \lambda A_{11} & \lambda A_{12} & \cdots & \lambda A_{1m} \\ \lambda A_{21} & \lambda A_{22} & \cdots & \lambda A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda A_{n1} & \lambda A_{n2} & \cdots & \lambda A_{nm} \end{pmatrix}$$

- **Matrix-Vector product**:
  Given a matrix $A \in M(n, m, \mathbb{R})$ and a vector $\mathbf{v} \in \mathbb{R}^m$ the **matrix-vector product** $A\mathbf{v} = \mathbf{w} \in \mathbb{R}^n$ is computed as follows:
  $$\mathbf{w} = A\mathbf{v} = \begin{pmatrix} A_1\mathbf{v} \\ A_2\mathbf{v} \\ \vdots \\ A_m\mathbf{v} \end{pmatrix},\ w_i = \sum_{j=1}^{m} A_{ij} v_j$$
  This is the simple way, just a row-by-column vector product, the computational complexity of this operation is $O(n^2)$.
  The smart way to compute it: **linear combinations** of columns of A, e.g.:
  $$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$
  with **linear combinations** we have:
  $$\begin{pmatrix} A_{11} \\ A_{21} \\ A_{31} \\ A_{41} \end{pmatrix} v_1 + \begin{pmatrix} A_{12} \\ A_{22} \\ A_{32} \\ A_{42} \end{pmatrix} v_2 + \begin{pmatrix} A_{13} \\ A_{23} \\ A_{33} \\ A_{43} \end{pmatrix} v_3 = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}$$

- **Matrix-Matrix Product**:
  Given two matrices $A \in M(n,m,R)$ and $B \in M(m,k,R)$ we call **matrix-matrix product** the following: $C = AB$ such that $C_{ij} = A_i B^j$, where ${A_i}^T \in \mathbb{R}^m$ is the i-th row of A, B^i is the i-th column of B ($B^i \in \mathbb{R}^m$) and $C \in M(n,k,\mathbb{R})$. Notice that this product is **not commutative**: $AB \neq BA$ might not even make sense dimension-wise.

  Terminology

  In this notes we refer to the columns to a generic matrix A as A^1 for the first column, A^2 for the second column and so on and so forth.
  Conversely, we represent the rows of a matrix A as A_i.

  As long as the complexity is concerned, multiplying two matrices $m \times n$ and $n \times k$ requires $O(mnk)$ floating point operations (flops). Forget about fancier algorithms (e.g. Strassen)

  Order of operations

  Usual algebra properties hold, e.g.: $A(B+C) = AB+AC, A(BC) = (AB)C, \ldots$.
  Parenthesization matters a lot: if $A, B \in M(n,\mathbb{R})$, $\mathbf{v} \in \mathbb{R}^n$, then $(AB)\mathbf{v}$ costs $O(n^3)$, but $A(B\mathbf{v})$ costs $O(n^2)$. Programming languages usually do not rearrange parentheses to help.

- **Image** of a matrix A ($\text{Im}(A)$): the set of vectors that can be obtained multiplying A by any vector in the domain of A.

- **Kernel** of a matrix A ($\ker(A)$): the set of vectors $\mathbf{w}$ in its domain such that $A\mathbf{w} = \mathbf{0}$.

- Given a matrix $A \in M(n,\mathbb{R})$ we call **inverse** of A the matrix A^{-1} such that:

$$A^{-1}A = AA^{-1} = I_n = \underbrace{\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}}_{n}$$

  The **inverse of a product** (shoe-sock identity) is $(AB)^{-1} = B^{-1}A^{-1}$. Notice that this identity holds only for square matrices.

- The **transpose** of a matrix $A \in M(n,m,\mathbb{R})$ is A^T such that $A^T_{ij} = A_{ji}$. The **transpose of a product** (shoe-sock identity) is $(AB)^T = B^T A^T$. (This identity holds for square and rectangular matrices)

Definition 1.2.1. ***General linear group (GL)****: the general linear group of degree n is the set of $n \times n$ invertible matrices, together with the operation of ordinary matrix multiplication*

Fact 1.2.1. *Let $A \in GL(n, \mathbb{R})$ (aka A is a real square matrix of size n and invertible), $B, C \in M(n, m, \mathbb{R})$ and we have the equality $AB = AC$. If there is a matrix M such that $MA = I$, the following holds*

$$(MA)B = (MA)C \iff B = C,\ M = A^{-1}$$

In general, $AB = AC$ does not imply $B = C$; it holds only when A is invertible.

Terminology

$$\mathbf{v} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}, \mathbf{v}^T = \begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$$

$\mathbf{v}$ is a column vector in $\mathbb{R}^3$ (or a matrix in $M(3, 1, \mathbb{R})$) and $\mathbf{v}^T$ is a row vector (or a matrix in $M(1, 3, \mathbb{R})$).

Definition 1.2.2 (Basis)**.** *We call* ***basis*** *a set $\mathcal{B}$ of elements (vectors) in a vector space V if every element of V may be written in a unique way as a (finite) linear combination of elements of $\mathcal{B}$. The coefficients of this linear combination are referred to as components or coordinates on $\mathcal{B}$ of the vector. The elements of a basis are called* ***basis vectors****.*

Definition 1.2.3 (Canonical basis)**.** *We term* ***canonical basis*** *of a vector space $\mathbb{R}^n$ the basis made of all the column of the $n \times n$ identity matrix $I_n \in M(n, \mathbb{R})$.*

Example 1.2.1. *In $\mathbb{R}^4$ the* ***canonical basis*** *is $\mathcal{B} = \{\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}, \mathbf{e_4}\}$ such that*

$$\mathbf{e_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix},\ \mathbf{e_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix},\ \mathbf{e_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix},\ \mathbf{e_4} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

and each vector $\mathbf{w} \in R^4$ can be written as $\mathbf{w} = w_1\mathbf{e_1} + w_2\mathbf{e_2} + w_3\mathbf{e_3} + w_4\mathbf{e_4}$.

Mantra

The powerful idea behind linear algebra: many relations are true regardless of the basis we use. E.g. $\mathbf{w}$, $\mathbf{v}$ and $\mathbf{w} + \mathbf{v}$ in two different bases.

Example 1.2.2. *Let us take two vectors $\mathbf{v}, \mathbf{w}$ and let us write those with respect to two different bases $\mathcal{B}_1$ and $\mathcal{B}_2$:*

$$\mathbf{w}_{\mathcal{B}_1} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} \mathbf{v}_{\mathcal{B}_1} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

and

$$\mathbf{w}_{\mathcal{B}_2} = \begin{pmatrix} -1 \\ 3 \end{pmatrix} \mathbf{v}_{\mathcal{B}_2} = \begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix}$$

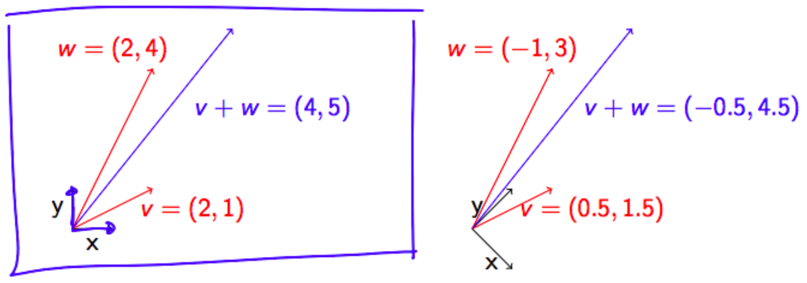


FIGURE 1.3: How a change of basis reflects on the space.

Definition 1.2.4 (Triangular matrix)**.** *Let $A \in M(n, \mathbb{R})$. We term A **upper triangular** if $(A)_{ij} = 0$ for each $i < j$. Conversely, we term A **lower triangular** if $(A)_{ij} = 0$ for each $j < i$. The set of all triangular $n \times n$ real matrices is a group and it is denoted as $T(n, \mathbb{R})$.*

Definition 1.2.5 (Diagonal matrix)**.** *Let $A \in M(n, \mathbb{R})$. We term A **diagonal** if $(A)_{ij} = 0$ for each $i \neq j$. The set of all diagonal $n \times n$ real matrices is a group and it is denoted as $D(n, \mathbb{R})$.*

Definition 1.2.6 (Symmetric matrix)**.** *Let $A \in M(n, \mathbb{R})$. We term A **symmetric** if $(A)_{ij} = (A)_{ji}$ for each $i, j = 1, \dots, n$. The set of all symmetric $n \times n$ real matrices is a group and it is denoted as $S(n, \mathbb{R})$.*

Fact 1.2.2. $A \in S(n, \mathbb{R}) \iff A^T = A$

1.3 Solving Linear Systems

The objective of this course, for the part concerning numerical methods, is solving linear systems efficiently.

Definition 1.3.1 (Linear system)**.** *Let $A \in M(n, m, \mathbb{R})$, $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{x} \in \mathbb{R}^m$. We term **linear system** the following:*

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

Sometimes it is not possible or not feasible to solve such a system, then we try to *approximate* the vector $\mathbf{x} \in \mathbb{R}^m$, trying to increase the proximity of the approximated value, trying to minimize $\|A\mathbf{x} - \mathbf{b}\|$* (see Figure 1.4).

*Where $\|\cdot\|$ denotes the norm of a vector and it is formalized in Definition 1.4.1

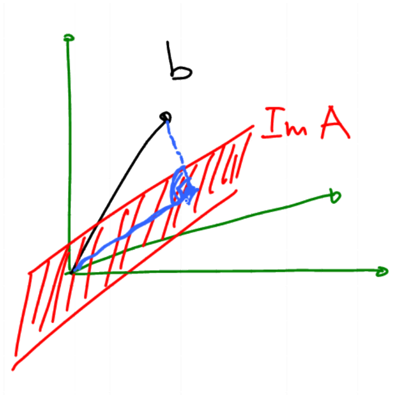


FIGURE 1.4: In this case the image of the matrix A (in red) does not contain $\mathbf{b}$ and the best one can do is to obtain a projection of $\mathbf{b}$ in the plane $Im(A)$ (drawn in blue).

Definition 1.3.2 (Least Squares Problem)**.** *Let $A \in M(n, m, \mathbb{R})$, $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{x} \in \mathbb{R}^m$. We term **least squares problem** the following:*

$$\min_{\mathbf{x} \in \mathbb{R}^m} \|A\mathbf{x} - \mathbf{b}\|$$

If we have a square and invertible matrix $A \in GL(n, \mathbb{R})$ solving a linear system means: find those coordinates $\mathbf{x_1}, \ldots, \mathbf{x_n}$ needed to write $\mathbf{b}$ as a linear combination of the columns of (square) A and in this case, the solution is given by: $\mathbf{x} = A^{-1}\mathbf{b}$.

Definition 1.3.3 (Linear combination)**.** *In a very informal way, we can define the goal of **linear combination** as the pursuit of obtaining a certain target vector $\mathbf{b} \in \mathbb{R}^n$ using m (in principle $m \neq n$) vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m} \in \mathbb{R}^n$ such that*

$$a_1\mathbf{x_1} + a_2\mathbf{x_2} + \cdots + a_m\mathbf{x_m} = \mathbf{b}$$

where $a_i \in \mathbb{R}$ are properly chosen.

Theorem 1.3.1. *Let $A \in M(n, m, \mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^n$. It holds that any linear system $A\mathbf{x} = \mathbf{b}$ is solvable iff A is invertible.*

Something on Matlab ...

Matlab provides syntactic sugar to solve linear systems.
Before introducing such syntax let we just notice the following `5\ 2` $(= 2/5) \neq$ `5/2`.
The syntax to solve $A\mathbf{x} = \mathbf{b}$ is `A\ b`, where the algorithm used in Matlab is not inverting the matrix A and then performing the multiplication, but it is a more sophisticated and efficient one.
Moreover, in Matlab the syntax `.op` means that function `op` should be performed entry by entry of the non-scalar variable. For example `b ./ c` performs the division element-wise of the two vectors **b** and **c**.

Definition 1.3.4 (Full column rank matrix)**.** *Let $A \in M(n, m, \mathbb{R})$ we say that A has* ***full column rank*** *if* $\ker A = \{\mathbf{0}\}$.

Equivalently, $rk(A) = n$ or alternatively $\nexists \mathbf{z} \in \mathbb{R}^n \backslash \{\mathbf{0}\}$ such that $A\mathbf{z} = \mathbf{0}$.

Fact 1.3.2. *Let $A \in M(n, m, \mathbb{R})$, the least square problem $\min\limits_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$ has a unique solution iff A has full column rank.*

Theorem 1.3.3. *Let $A \in M(n, m, \mathbb{R})$. A has full column rank iff $A^T A$ is positive definite.*

Proof. A has full column rank $\iff \|A\mathbf{z}\| \neq 0, \forall \mathbf{z} \in \mathbb{R}^m \backslash \{\mathbf{0}\} \iff \|A\mathbf{z}\|^2 \neq 0, \forall \mathbf{z} \in \mathbb{R}^m \backslash \{\mathbf{0}\} \iff 0 = (A\mathbf{z})^T A\mathbf{z} = \mathbf{z}^T A^T A\mathbf{z}$ □

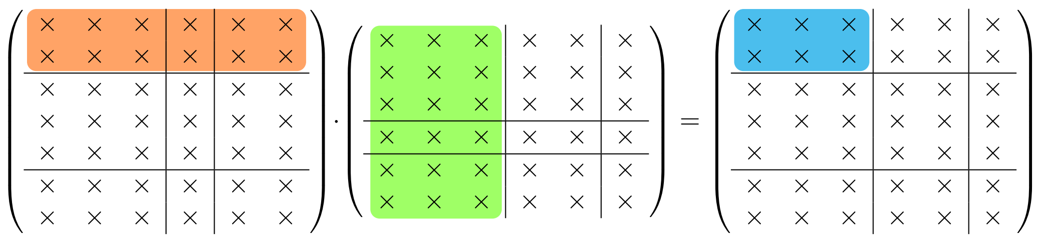
Warning: this is not the best way to solve a linear system on a computer!

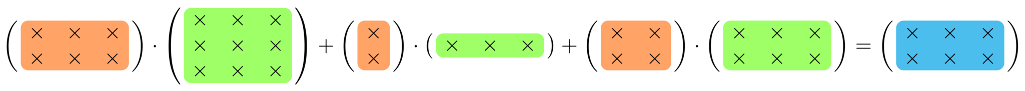
Something on Matlab ...

Notice that the machine precision is 10^{-16}, so we should pay attention when making computations, since we may incur in some error (proportional to the size of the operands).
In Matlab a matrix is written as `A=[1, 2, 3; 4, 5, 6];`, where `[1, 2, 3]` is the first row of the matrix A.
The transpose of a matrix or a vector is denoted by `A'`.
The inverse of a square matrix is denoted by `inv(A)`.
If we are interested in only a part of our matrix `A` we may write `A(1:2, 1:3)` and obtain only the rows of `A` that go from 1 to 2 and those columns from 1 to 3.
Notice that in Matlab both vector and matrices are 1-based.

Definition 1.3.5 (Block multiplications)**.** *Let $A \in M(n, m, \mathbb{R})$ and let $B \in M(m, k, \mathbb{R})$. We can compute the result of a block of the matrix AB as the product of the two blocks in A and B in the corresponding position.*

Observation 1.3.1. *When computing a matrix product, we get the same result if we use the row-by-column rule* block-wise.





Notice that block operations usually give better performance: one matrix-matrix product performs faster than n matrix-vector products (even if they have the same number of flops). This is one of the reasons why library calls usually perform better than hand-coded loops (Blas/Lapack).

Fact 1.3.4 (Block triangular matrices)**.** *Let $M \in M(n, m, \mathbb{R})$ and $B \in M(m, k, \mathbb{R})$ such that they are* ***block triangular****. Their product is a block triangular matrix as well. In other words, block triangular matrices are closed under products:*

$$MB = \begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \begin{pmatrix} D & E \\ 0 & F \end{pmatrix} = \begin{pmatrix} AD & AE + BF \\ 0 & CF \end{pmatrix}$$

Fact 1.3.5 (Properties of block triangular matrices)**.**
Let M be a block triangular matrix, where all the blocks on the diagonal are square

$$M = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ 0 & A_{22} & \cdots & A_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & A_{nn} \end{pmatrix}$$

1. *A block triangular matrix is invertible iff all diagonal blocks A_{ii} are invertible;*

2. *The eigenvalues*[†] *of a block triangular matrix are the union of the eigenvalues of each diagonal block A_{ii};*

3. *Let $M \in GL(n, \mathbb{R})$ such that $M = \begin{pmatrix} A & B \\ 0 & C \end{pmatrix}$ the inverse of M is*

$$M^{-1} = \begin{pmatrix} A^{-1} & -A^{-1}BC^{-1} \\ 0 & C^{-1} \end{pmatrix}.$$

[†]The concept of eigenvalue is exposed in Definition 1.5.1.

4. *The product of two block (upper/lower) triangular matrices (with compatible block sizes) is still block triangular*

Why are we interested in block triangular matrices? They depict a situation as shown in Figure 1.5.

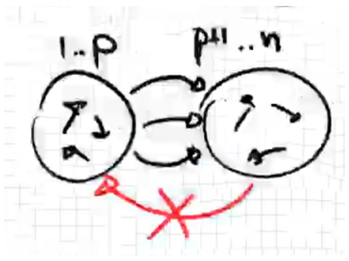


FIGURE 1.5: The adjacency matrix of a biparted graph has 0s in its bottom left part (Matlab syntax `A[p+1:n; 1:p]=0`), which means that the edges from a connected component and the other are in one direction only.

Mantra

Matrix structures matter. Block triangular linear systems have a cheaper solution than general systems as shown in Example 1.3.1.

Example 1.3.1. *Let us take a 2×2 block triangular linear system*

$$\begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{e} \\ \mathbf{f} \end{pmatrix}$$

(Again, diagonal blocks are square and all dimensions are compatible.)

$$\begin{pmatrix} A\mathbf{x} + B\mathbf{y} \\ C\mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{e} \\ \mathbf{f} \end{pmatrix} \implies \mathbf{y} = C^{-1}\mathbf{f}, \mathbf{x} = A^{-1}(\mathbf{e} - BC^{-1}\mathbf{f})$$

$$\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & -A^{-1}BC^{-1} \\ 0 & C^{-1} \end{pmatrix}$$

Informal idea: we can start solving from the variables multiplied by C.

1.4 Orthogonality

Definition 1.4.1 (Norms)**.** *Let $\mathbf{x} \in \mathbb{R}^n$. We "measure" its magnitude using so-called "norms".*

EUCLIDEAN: $\|\mathbf{x}\|_2 = \mathbf{x}^T\mathbf{x} = \sqrt{\sum_{i=1}^{n} x_i{}^2}$;

NORM 1: $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$;

p-NORM: $|\mathbf{x}|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$;

0-NORM: $\|\mathbf{x}\|_0 = |\{i \ : \ |x_i| > 0\}|$, *which accounts for $n - \#of\, 0$ entries;*

∞-NORM: $\|\mathbf{x}\|_\infty = \max_{i=1,\ldots,n} |x_i|$.

From now on in this part of the course, if not explicitly specified, we will refer to norm-2 only.

Definition 1.4.2 (Scalar product)**.** *Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ we term **standard scalar product** between $\mathbf{v}$ and $\mathbf{w}$ the real number $\mathbf{v}^T\mathbf{w} = \sum_{i=1}^{n} v_i w_i$.*

Definition 1.4.3 (Orthogonal vectors)**.** *Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$. We say that $\mathbf{v}$ is **orthogonal** to $\mathbf{w}$ (in symbols $\mathbf{v} \perp \mathbf{w}$) if their scalar product is zero.*

Formally, $\mathbf{v}^T\mathbf{w} = 0$.

Definition 1.4.4 (Orthogonal matrix)**.** *Let $U \in M(n, \mathbb{R})$ a square matrix. We term U **orthogonal** if $U^T U = U U^T = I_n$ where I_n is the identity matrix of size n (1 on the diagonal, 0 elsewhere) or equivalently if $U^{-1} = U^T$.*

The set of all orthogonal matrices in $M(n, \mathbb{R})$ is a group and it called orthogonal group and denotes as $O(n, \mathbb{R})$.

Fact 1.4.1. *Let $U \in O(n, \mathbb{R})$, $\forall \mathbf{x} \in \mathbb{R}^n$ we have that $\|U\mathbf{x}\| = \|\mathbf{x}\|$.*

Proof. Let us provide two different alternatives for proving the equality:

1. $$\|U\mathbf{x}\| = \sqrt{(U\mathbf{x})^T U\mathbf{x}} \overset{\mathbf{1}}{=} \sqrt{\mathbf{x}^T U^T U \mathbf{x}} = \sqrt{\mathbf{x}^T I_n \mathbf{x}} = \sqrt{\mathbf{x}^T \mathbf{x}} = \|\mathbf{x}\|$$

   where $\overset{(1)}{=}$ follows from the definition of transpose of a product.

2. Instead of proving that $\|U\mathbf{x}\| = \|\mathbf{x}\|$ we will prove $\|U\mathbf{x}\|^2 = \|\mathbf{x}\|^2$:

   $$\|U\mathbf{x}\|^2 = (U\mathbf{x})^T \cdot (U\mathbf{x}) \overset{(1)}{=} \mathbf{x}^T U^T U \mathbf{x} = \mathbf{x}^T I_n x = \mathbf{x}^T \mathbf{x} = \|\mathbf{x}\|$$

   where $\overset{(1)}{=}$ follows from the definition of transpose of a product.

□

Geometrically speaking, orthogonal transformations (aka matrices) preserve the norm, so an orthogonal matrix U represents a symmetry or a rotation and these operations do not alter the size of vectors.

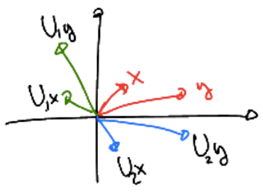


FIGURE 1.6: Matrix U_1 represents a rotation, while U_2 is a symmetry operation.

Definition 1.4.5 (Orthonormality)**.** *Let* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ *we say that* $\mathbf{x}$ *and* $\mathbf{y}$ *are* ***orthonormal*** *if they are orthogonal and have norm* 1*. Formally,* $<\mathbf{x}, \mathbf{y}>= 0$ *and* $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$*.*

Fact 1.4.2. *Let us take* $U \in O(n, \mathbb{R})$*. Then its columns* $U^1, U^2, \ldots, U^n$ *are* ***orthonormal*** *and the same holds for its rows.*

$$U^{i^T} U^j = \begin{cases} 1 \textit{ if } i = j \\ 0 \textit{ otherwise} \end{cases}$$

and

$$U_i {U_j}^T = \begin{cases} 1 \textit{ if } i = j \\ 0 \textit{ otherwise} \end{cases}$$

Fact 1.4.3. *The set of orthogonal matrices is closed under product operations. Let* $U, V \in O(n, \mathbb{R})$*, then* $U \cdot V$ *is orthogonal.*

Proof. $(UV)^T \cdot (UV) = V^T U^T U V = V^T I_n V = V^T V = I_n$ □

Since we will often deal with tall-thin rectangular matrices with orthonormal columns as U_1

$$U_1 = \begin{pmatrix} U^1 & U^2 & \cdots & U^n \end{pmatrix} \in M(m, n, \mathbb{R}) \text{where } m \geq n$$

the following fact may come in handy

Fact 1.4.4. $\forall U_1 \in M(m, n, \mathbb{R})$ *where* $M \geq n$ *and the columns of* U_1 *are orthogonal* $\exists U_2 \in M(m, m-n, \mathbb{R})$ *s.t.* $\begin{pmatrix} U_1 & U_2 \end{pmatrix} \in O(m, \mathbb{R})$*.*

1.5 Eigenvalues / Eigenvectors

Definition 1.5.1 (Eigenvectors and eigenvalues)**.** *Let* $A \in M(n, \mathbb{R})$ *and let* $\mathbf{x} \neq \mathbf{0} \in \mathbb{R}^n$ *and* $\lambda \in \mathbb{R}$*.*

If $A\mathbf{x} = \lambda \mathbf{x}$ *we say that* $\mathbf{x}$ *is an* ***eigenvector*** *of* ***eigenvalue*** λ*.*

Theorem 1.5.1. *Let* $A \in T(n, \mathbb{R})$ *(real triangular matrix). The eigenvalues of* A *are the scalars on the diagonal.*

Definition 1.5.2 (Diagonalizable matrix)**.** Some *matrices* $A \in M(n, \mathbb{R})$ *under some conditions can be decomposed as:*

$$A = V\Lambda V^{-1}$$

$$A = V\Lambda V^{-1} = \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 & & 0 & \\ & \lambda_2 & & \\ & 0 & \ddots & \\ & & & \lambda_n \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w_1} \\ \mathbf{w_2} \\ \vdots \\ \mathbf{w_n} \end{pmatrix}$$

where $V \in GL(n, \mathbb{R})$ *is the matrix that has as columns the eigenvectors of* A (V^i, λ_i) $\forall i = 1, \dots, n$ *and* $\mathbf{w_i} = V_i^{-1}$ *are the rows of the inverse of matrix* V*.*

Fact 1.5.2. $\forall V^i \in \mathbb{R}^n$ *eigenvectors of* A *they are still eigenvector of the diagonalized form of* $A = V\Lambda V^{-1}$

Proof.

$$AV^i = V\Lambda V^{-1}V^i = V\Lambda e_i = \lambda_i V^i$$

□

Another way to see the diagonalized form of A is the following:

$$A = V\Lambda V^{-1} = \sum_{i=1}^{n} v_i \lambda_i w_i^T =$$

$$\boxed{\mathbf{v_1}} \cdot \boxed{\lambda_1} \cdot \boxed{\mathbf{w_1}^T} + \boxed{\mathbf{v_2}} \cdot \boxed{\lambda_2} \cdot \boxed{\mathbf{w_2}^T} + \cdots + \boxed{\mathbf{v_n}} \cdot \boxed{\lambda_n} \cdot \boxed{\mathbf{w_n}^T}$$

Something on Matlab ...

Notice that in Matlab the eigenvalues and eigenvectors of a matrix are computed using the command `[V, Lambda] = eig(U)` and this operation has a computational complexity of $O(n^3)$.
We can check that the matrix `A` is equal to the decomposition in this way: `A - V*Lambda*inv(V)` or `norm(A - V*Lambda*inv(V))` (both should be close to zero).

Notice that not all matrices $A \in M(n, \mathbb{R})$ allow a diagonal decomposition. It may happen that such a matrix A is diagonalizable in $\mathbb{C}$ and its eigenvalues are complex.

This decomposition tell us the behavior under repeated application of a matrix A to a vector $\mathbf{x}$. This process allows to scale a general vector $\mathbf{x}$.

Something on Matlab ...

Let `A = [1 1; 1 1]` and `x = [ 1 1 ]` such that $\lambda_{\mathbf{x}} = 2$ then `A*x` is equal to `[2 2]'` and `A*A*x` is equal to `[4 4]'`

Fact 1.5.3. *If $A \in M(n, \mathbb{R})$ is diagonalizable (aka may be written as $A = V\Lambda V^{-1}$) then $A^k\mathbf{x} = \sum\limits_{i=1}^{n} \lambda_i^k \alpha_i V^i$, for some $\alpha_i \in \mathbb{R}$.*

Proof. Let us write $\mathbf{x}$ in the base of $\mathbb{R}^n$ made of the linearly independent columns of V:

$$\mathbf{x} = V^1\alpha_1 + V^2\alpha_2 + \cdots + V^n\alpha_n$$

for some $\alpha_i \in \mathbb{R}$.

- Algebraic view point:

$$\begin{aligned} A\mathbf{x} &= A \cdot (V^1\alpha_1 + V^2\alpha_2 + \cdots + V^n\alpha_n) \\ &= AV^1\alpha_1 + AV^2\alpha_2 + \cdots + AV^n\alpha_n \\ &= \lambda_1 V^1\alpha_1 + \lambda_2 V^2\alpha_2 + \cdots + \lambda_n V^n\alpha_n \\ &= V^1(\lambda_1\alpha_1) + V^2(\lambda_2\alpha_2) + \cdots + V^n(\lambda_n\alpha_n) \end{aligned} \tag{1.5.1}$$

  Then

$$\begin{aligned} A^2x &= A \cdot \Big(V^1(\lambda_1\alpha_1) + V^2(\lambda_2\alpha_2) + \cdots + V^n(\lambda_n\alpha_n)\Big) \\ &= AV^1\lambda_1\alpha_1 + AV^2\lambda_2\alpha_2 + \cdots + AV^n\lambda_n\alpha_n \\ &= {\lambda_1}^2V^1\alpha_1 + {\lambda_2}^2V^2\alpha_2 + \cdots + A{\lambda_n}^2V^n\alpha_n \end{aligned} \tag{1.5.2}$$

  The thesis follows inductively.

- Linear algebra view point:

$$\begin{aligned}
A^k\mathbf{x} &= A \cdot A \cdot \ldots \cdot A \cdot \mathbf{x} \\
&= V\Lambda\cancel{V^{-1}}\cancel{V}\Lambda\cancel{V^{-1}}\ldots\cancel{V}\Lambda V^{-1}\mathbf{x} \\
&= V\Lambda^k V^{-1}\mathbf{x} \\
&= V \begin{pmatrix} {\lambda_1}^k & & & \\ & {\lambda_2}^k & & \\ & & \ddots & \\ & & & {\lambda_n}^k \end{pmatrix} V^{-1}\mathbf{x} \qquad (1.5.3) \\
&= V \begin{pmatrix} {\lambda_1}^k & & & \\ & {\lambda_2}^k & & \\ & & \ddots & \\ & & & {\lambda_n}^k \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \ddots \\ \alpha_n \end{pmatrix}
\end{aligned}$$

□

Notice that if A is not square, $A\mathbf{v}$, $\lambda\mathbf{v}$ have different sizes and it doesn't make sense to talk about eigenvalues.

1.5.1 Eigenvector: what could possibly go wrong?

1. The eigenvalue decomposition is highly non-unique, we can:
   - Reorder eigenvalues/vectors
   - Replace an eigenvector v_i with $2v_i$, $3.5v_i$ ...
   - For matrices with repeated eigenvalues we have even more possibilities: e.g $I = VIV^{-1}$ for every invertible V
2. some matrices have only complex eigenvalues: e.g. $\begin{pmatrix} 2 & 4 \\ -3 & 3 \end{pmatrix}$
3. some matrices have fewer eigenvectors than we want and we can't use eigenvalue decomposition: e.g. $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$

Now, thanks to the eigenvalue decomposition we can prove the following:

Theorem 1.5.4. *Let $A \in M(n, \mathbb{R})$. If $|\lambda_i| < 1$ for all eigenvalues λ_i of A then* $\lim_{k\to\infty} A^k\mathbf{x} = 0$, $\forall \mathbf{x} \in \mathbb{R}^n$.

Theorem 1.5.5. *Let $A \in M(n, \mathbb{R})$. If $\forall\lambda_i$ eigenvalues of A $|\lambda_i| < |\lambda_1|$ then* $A^k\mathbf{x} \approx V^1{\lambda_1}^k\alpha_1$, $\forall \mathbf{x} \in \mathbb{R}^n$.

Fact 1.5.6. *Let $A \in M(n, \mathbb{R})$ be a diagonalizable matrix and let*

$$A = V\Lambda V^{-1} = \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

Let us now consider a reordering of V's columns and apply the same permutation to the "diagonal vector" of Λ such that

$$\hat{V} = \begin{pmatrix} V^2 & V^1 & V^3 \cdots & V^n \end{pmatrix} \text{ and } \hat{\Lambda} = \begin{pmatrix} \lambda_2 & & & & \\ & \lambda_1 & & & \\ & & \lambda_3 & & \\ & & & \ddots & \\ & & & & \lambda_n \end{pmatrix}$$

A can be diagonalized through such $\hat{V}$ and $\hat{\Lambda}$ as $A = V\Lambda V^{-1} = \hat{V}\hat{\Lambda}\hat{V}^{-1}$.

Moreover, in the case of repeated eigenvalues

Fact 1.5.7. *Let $A \in M(n, \mathbb{R})$ a diagonalizable matrix such that $A = V\Lambda V^{-1}$, where $\lambda_1 = \lambda_2$ (without loss of generality). Then V can be replaced by $\widetilde{V} =$*

$$\begin{pmatrix} V^1 + V^2 & V^1 - V^2 & V^3 & \cdots & V^n \end{pmatrix}.$$

Theorem 1.5.8 (Spectral theorem)**.** *Any real, symmetric matrix is diagonalizable. Formally, let $A \in S(n, \mathbb{R})$. Then A is diagonalizable $A = U\Lambda U^{-1}$, where eigenvalues are all real numbers and we can take U orthogonal matrix.*

Notice that for symmetric matrices none of the "unlucky" situations enumerated above may happen (it is justified by the spectral theorem).

Something on Matlab ...

If we have any symmetric matrix `B` and we compute `[V, D] = eig(B)`, matlab will always return an orthogonal matrix V.

Definition 1.5.3 (Quadratic form)**.** *Let $Q \in S(n, \mathbb{R})$ we define **quadratic form***

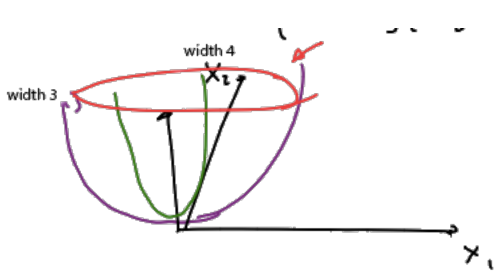
$$f(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$$

for each $\mathbf{x} \in \mathbb{R}^n$.

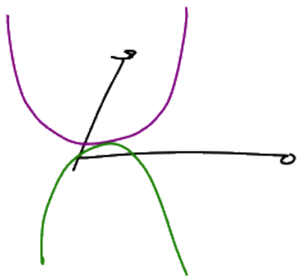
Geometrically, a quadratic form defines a paraboloid.

Example 1.5.1. *Let* $f_1(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3{x_1}^2 + 4x_1x_2 + 4{x_2}^2$ *and let* $f_2(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 2 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3{x_1}^2 + 4x_1x_2 - 4{x_2}^2$

For a graphic hint see Figure 1.7.



(a) $f_1(x)$



(b) $f_2(x)$

FIGURE 1.7: The plot of functions.

Theorem 1.5.9 (Variational characterization)**.** *Let* $Q \in S(n, \mathbb{R})$ *and let* $\mathbf{x} \in \mathbb{R}^n$. *Then*

$$\lambda_{min}\|\mathbf{x}\|^2 \le \mathbf{x}^T Q \mathbf{x} \le \lambda_{max}\|\mathbf{x}\|^2$$

where λ_{max} *and* λ_{min} *are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.*

Proof. EASY CASE WITH Q DIAGONAL:

$$\mathbf{x}^T Q \mathbf{x} = \mathbf{x}^T \cdot \begin{pmatrix} \lambda_2 & & & & \\ & \lambda_1 & & & \\ & & \lambda_3 & & \\ & & & \ddots & \\ & & & & \lambda_n \end{pmatrix} \cdot \mathbf{x} = \lambda_1 {x_1}^2 + \lambda_2 {x_2}^2 + \cdots + \lambda_n {x_n}^2$$

It is obvious that this sum is bounded by:

$$\lambda_{\min} \cdot ({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) \le \lambda_1 {x_1}^2 + \lambda_2 {x_2}^2 + \cdots + \lambda_n {x_n}^2 \le \lambda_{\max} \cdot ({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2)$$

The following holds: $\lambda_{\min} \cdot ({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) = \lambda_{\min} \cdot \mathbf{x}^T \mathbf{x} = \lambda_{\min} \cdot \|\mathbf{x}\|^2$ and, on the other hand, $\lambda_{\max} \cdot ({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2) = \lambda_{\max} \cdot \mathbf{x}^T \mathbf{x} = \lambda_{\max} \cdot \|\mathbf{x}\|^2$ and this proves the fact in the special case of diagonal matrix Q.

GENERAL CASE: Let us represent Q through its eigendecomposition: $A = U\Lambda U^{-1} = U\Lambda U^T$, where U is an orthogonal matrix.

$$\mathbf{x}^T Q\mathbf{x} = \mathbf{x}^T U\Lambda U^T\mathbf{x} \overset{(1)}{=} \mathbf{y}^T\Lambda\mathbf{y}$$

where $\overset{(1)}{=}$ is due to the change of variable $\mathbf{y} = U^T\mathbf{x}$ (that implies $\mathbf{y}^T = \mathbf{x}^T U$).

By the same argument used in the diagonal case,

$$\lambda_{\min} \cdot \|\mathbf{y}\|^2 \leq \mathbf{y}^T\Lambda\mathbf{y} \leq \lambda_{\max} \cdot \|\mathbf{y}\|^2$$

Now the point is that if we can replace $\left\|U^T\mathbf{x}\right\|^2 = \|\mathbf{y}\|^2$ with $\|\mathbf{x}\|^2$ we have proved the theorem. In fact this is true, due to the orthogonality of matrix U and Proposition 1.4.1.

□

Corollary 1.5.10. *Let $Q \in S(n,\mathbb{R})$ and let $\mathbf{x} \in \mathbb{R}^n$. If $\mathbf{x} \neq \mathbf{0}$*

$$\lambda_{min} \leq \frac{\mathbf{x}^T Q\mathbf{x}}{\|\mathbf{x}\|^2} \leq \lambda_{max}$$

where λ_{max} and λ_{min} are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.

Definition 1.5.4 (Positive semidefinite)**.** *Let $Q \in S(n,\mathbb{R})$. We say that Q is* ***positive semidefinite*** *(and we indicate $\succeq 0$) if*

$$\mathbf{x}^T Q\mathbf{x} \geq 0, \text{ for any } \|\mathbf{x}\|^2 \geq 0$$

Definition 1.5.5 (Positive definite)**.** *Let $Q \in S(n,\mathbb{R})$. We say that Q is* ***positive definite*** *(and we indicate $\succ 0$) if*

$$\mathbf{x}^T Q\mathbf{x} > 0, \text{ for any } \|\mathbf{x}\|^2 > 0$$

Fact 1.5.11. *Let $Q \in S(n,\mathbb{R})$. The following holds*

$$Q \text{ is positive semidefinite} \iff \forall\lambda \text{ eigenvalue of } Q\ \lambda \geq 0$$

And this holds with the > 0 in the positive definite case.

Proof.

($\Leftarrow$) $\mathbf{x}^T Q\mathbf{x} \geq \lambda_{\min}\|\mathbf{x}\|^2 \geq 0$ since we are in the hypothesis that all the eigenvalues are ≥ 0

($\Rightarrow$) $\forall\mathbf{v_i}$ eigenvector of Q $0 \leq \mathbf{v_i}^T Q\mathbf{v_i} = \mathbf{v_i}^T \cdot (\lambda_i\mathbf{v_i}) = \lambda_i\mathbf{v_i}^T\mathbf{v_i} = \lambda_i\|\mathbf{v_i}\|^2 \Rightarrow \lambda_i \geq 0$. □

Corollary 1.5.12. *Let $Q \in S(n,\mathbb{R})$ such that $Q \succeq 0$. The following holds:*

$$Q \text{ is invertible} \iff Q \text{ is strictly positive definite}$$

Fact 1.5.13. *Let $B \in M(m, n, \mathbb{R})$ (possibly rectangular), $B^T B \in S(n, \mathbb{R})$ is a valid product and it is a square, symmetric, positive semidefinite matrix.*

Proof. Symmetry: $(B^T B)^T = B^T \cdot (B^T)^T = B^T B$.

Positive definite: $\mathbf{x}^T B^T B\mathbf{x} = (B\mathbf{x})^T (B\mathbf{x}) = \|B\mathbf{x}\|^2 \geq 0$

□

Corollary 1.5.14. *The same holds for $BB^T \in S(m, \mathbb{R})$ and it is easily proved taking $C = B^T \in M(n, m, \mathbb{R})$.*

Something on Matlab . . .

In order to check if a matrix A is positive definite in Matlab we can look at its eigenvalues (cfr. `eig(A)`).

Notice that in the **complex** case most of these properties work as well, but it is needed to replace A^T with $\overline{A^T}$[‡] (transpose and entrywise conjugate).
The norm of a complex vector is computed as

$$\|\mathbf{x}\|_2^2 = \mathbf{x}^* \mathbf{x} = \overline{x_1} x_1 + \overline{x_2} x_2 + \cdots + \overline{x_n} x_n = |x_1|^2 + \cdots + |x_n|^2 \in \mathbb{R}^+ \cup \{0\}$$

Moreover, in the complex case, a matrix $U \in M(n, \mathbb{C})$ s.t. $UU^* = I$ is called **unitary matrix** (orthogonal + complex)

1.6 Singular value decomposition (SVD)

We are left with the task of reaching a (sort of) "eigenvalue decomposition" when the target matrix is not symmetric.
There are two ways to generalize the eigenvalue decomposition to a non-symmetric matrix A (with something that always exists):

Definition 1.6.1 (Schur decomposition)**.** *Let $A \in M(n, \mathbb{R})$, $\exists\, U \in O(n, \mathbb{R})$ orthogonal matrix and $T \in T(n, \mathbb{R})$ triangular matrix such that $A = UTU^T$ and this is called **Schur decomposition**.*

We can say more:

Definition 1.6.2 (Singular value decomposition)**.** *Let $A \in M(n, \mathbb{R})$, $\exists\, U, V \in M(n, \mathbb{R})$ orthogonal matrices (V not necessary equal to U) and $\Sigma \in Diag(n, \mathbb{R})$ such that $A = U\Sigma V^T$ and this is called **Singular Value Decomposition**.*

$$A = \begin{pmatrix} U^1 & U^2 & \cdots & U^n \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix} \cdot \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \end{pmatrix} = \sum_{i=1}^{n} u_{ii} \sigma_i v_{ii} =$$

[‡]Often denoted with A^* or A^H and called **Hermitian matrix**.

*Where σ_i are called **singular values** and they are sorted such that:*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$$

General fact on singular values:

- *Singular values $\neq$ eigenvalues*
- *They are always positive and usually more spread apart than the eigenvalues.*

$$\sigma_1 \geq |\lambda_1| \;\; and \;\; |\lambda_n| \geq \sigma_n$$

σ_1 is larger than the largest eigenvalue of a matrix A and σ_n is smaller than the smallest eigenvalue of a matrix A.

The SVD can be defined also for a rectangular matrix A:

Definition 1.6.3 (Rectangular matrices and SVD)**.** *Let $A \in M(m, n, \mathbb{R})$, there exist $U \in O(m, \mathbb{R})$ orthogonal, $V \in O(n, \mathbb{R})$ orthogonal and $\Sigma \in D(m, n, \mathbb{R})$ diagonal in the sense that $\sum_{ij} = 0$ with $i \neq j$ (padded with zeros). Matrix A has a **SVD factorization** ($A = U\Sigma V^T$), where Σ has the following shape:*

- *case $m < n$ (e.g $m = 3, n = 5$)*

$$\begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \end{pmatrix}$$

- *case $m > n$ (e.g $m = 5, n = 3$)*

$$\begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Definition 1.6.4 (Thin SVD)**.** *Let $A \in M(m, n, \mathbb{R})$, has a **thin SVD factorization**: we may restrict to compute only the first min(m, n) vectors that appear in this sum: thin SVD.*

$$A = \sum_{i=1}^{\tilde{n}\,\min(m,n)} u_{ii}\sigma_i v_{ii} = u_{11}\sigma_1 v_{11} + u_{22}\sigma_2 v_{22} + \cdots + u_{\tilde{n},\tilde{n}}\sigma_{\tilde{n}} v_{\tilde{n},\tilde{n}}$$

Something on Matlab ...

In Matlab the SVD decomposition is obtained through the command `svd(A)`, which return value is made of the three matrices U, Σ, V.
As an example, `[U, S, V] = svd(A)`. Notice that, if `svd(A)` is assigned to one variable, then such variable is an array of singular values.
The thin SVD can be computed as: `[U, S, V] = svd(A, 0)`

Computational costs
We are not going into details of algorithms for computing SVD, but we would like to add a consideration about the computational complexity of such an algorithm.

- `[U, S, V] = svd(A, 0)` (thin) costs $O(mn^2)$ ops for $A \in M(m, n, \mathbb{R})$ or $A \in M(n, m, \mathbb{R})$ with $m \geq n$
- `[U, S, V] = svd(A)` (non-thin) is more expensive, because it has to store the large $m \times m$ factor. (But there are some tricks to store orthogonal matrices compactly, more about it later).

1.6.1 Properties of SVD

The SVD reveals rank, image, and kernel of a matrix.

Definition 1.6.5 (Rank)**.** *Let $A \in M(n, \mathbb{R})$ we call the **rank** of A the number of non-zero singular values.*

*Equivalently, the **rank** is the size of the column space.*

Property 1.6.1. *A matrix $A \in M(n, \mathbb{R})$ has rank r iff all its singular values starting from the $r+1$-th are 0, formally iff $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$.*

Thanks to Property 1.6.1, we can somehow talk about an "even thinner" SVD, where all the 0s in the bottom right part of the matrix Σ, cancel out the latter columns of U and the latter rows of V (aka columns of V^T). A pictorial representation of the shape of Σ can be found below.

$$
\Sigma \;=\; \left(\begin{array}{cccc|ccc}
\sigma_1 & & & & & & \\
& \sigma_2 & & & & & \\
& & \ddots & & & & \\
& & & \sigma_r & & & \\
\hline
& & & & 0 & & \\
& & & & & \ddots & \\
& & & & & & 0 \\
\hline
\multicolumn{7}{c}{\mathbf{0}}
\end{array}\right)
$$

This factorization represents A as $\sum_{i=1}^{r} u_{ii}\sigma_i v_{ii}$.

Fact 1.6.2. *Let $A \in M(n, \mathbb{R})$ such that has the following SVD-factorization $A = U\Sigma V^T$ with $U \in O(n, \mathbb{R})$ orthogonal, $V \in O(n, \mathbb{R})$ orthogonal and $\Sigma \in D(n, \mathbb{R})$ is a diagonal matrix. The image of A is the span of $U_1, U_2, \ldots, U_r$, hence $rk(A) = r$. Moreover, $\ker(A) = span(V_{r+1}, V_{r+2}, \ldots, V_n)$, since V is orthogonal.*

Proof.

TODO: plugging in $x = V_j$, where $j > r$

□

Definition 1.6.6 (Matrix norm)**.** *Let $A \in M(m, n, \mathbb{R})$. We define the **matrix norm** of A as*

$$\|A\| := \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\mathbf{z}\ s.t. \|\mathbf{z}\|=1} \|A\mathbf{z}\|$$

Where the norm may be any of the ones defined in Definition 1.4.1 second equality is introduced in order to work in a compact set, the one of normalized vectors $\mathbf{z}$.

Property 1.6.3. *Let A and $B \in M(n, m, \mathbb{R})$ and let $\mathbf{x} \in \mathbb{R}^n$, the following holds, for any norm defined in Definition 1.4.1:*

- $\|A\| \geq 0$ *(and the equality holds iff $A = 0$);*
- $\|\alpha A\| = |\alpha| \|A\|, \forall \alpha \in \mathbb{R}$*;*
- $\|A + B\| \leq \|A\| + \|B\|$*;*
- $\|AB\| \leq \|A\| \|B\|$*;*
- $\|A\mathbf{x}\| \leq \|A\| \|\mathbf{x}\|$, $\forall \mathbf{x} \in \mathbb{R}^n$.

Fact 1.6.4. *Let $A \in (n, \mathbb{R})$ and let $U \in O(m, n, \mathbb{R})$ orthogonal, in the case of 2-norm $\|A\|_2 \overset{(1)}{=} \|AU\|_2 \overset{(2)}{=} \|UA\|_2$.*

Proof.

$\overset{(1)}{=}$

$$\begin{aligned}
\|UA\|_2 &= \max_{\mathbf{x}\in R^n,\ \mathbf{x}\neq\mathbf{0}} \frac{\|UA\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \\
&= \max_{\mathbf{x}\in R^n,\ \mathbf{x}\neq\mathbf{0}} \frac{\sqrt{(UA\mathbf{x})^T(UA\mathbf{x})}}{\|\mathbf{x}\|_2} \\
&= \max_{\mathbf{x}\in R^n,\ \mathbf{x}\neq\mathbf{0}} \frac{\sqrt{\mathbf{x}^T A^T \cancel{U^T} \cancel{U} A\mathbf{x}}}{\|\mathbf{x}\|_2} \\
&= \max_{\mathbf{x}\in R^n,\ \mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \|A\|_2
\end{aligned}$$

$\overset{(2)}{=}$

$$\begin{aligned}
\|AU\|_2 &= \max_{\mathbf{x}\in R^n,\ \mathbf{x}\neq\mathbf{0}} \frac{\|AU\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \\
&\overset{(2)}{=} \max_{\mathbf{y}\in R^n,\ \mathbf{y}\neq\mathbf{0}} \frac{\|A\mathbf{y}\|_2}{\|\mathbf{y}\|_2} = \|A\|_2
\end{aligned}$$

where $\overset{(2)}{=}$ follows from the substitution $\mathbf{y} = U\mathbf{x}$. □

Definition 1.6.7 (Frobenius norm)**.** *Let $A \in M(n, m, \mathbb{R})$, we term* ***Frobenius norm*** *of A $\|A\|_F = \sqrt{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{m} {(A)_{ij}}^2}$.*

Notice that all the properties enumerated in Property 1.6.3 hold for the Frobenius norm as well.

Fact 1.6.5. *Let $A \in M(n, m, \mathbb{R})$ and let $A = U\Sigma V^T$ be its singular value decomposition. The following holds:*

1. $\|A\|_2 = \|\Sigma\|_2 = \sigma_1$
2. $\|A\|_F = \|\Sigma\|_F = \sum\limits_{i=1}^{\tilde{n}} {\sigma_i}^2$*, where* $\tilde{n} = \min(n, m)$

Proof. 1. The first equality follows from Proposition 1.6.4, while the second is proved as follows:

$$\|\Sigma\|_2 = \max_{\mathbf{x}\in\mathbb{R}^n,\ \mathbf{x}\neq\mathbf{0}} \frac{\|\Sigma\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{\mathbf{x}\in\mathbb{R}^n,\ \mathbf{x}\neq\mathbf{0}} \frac{\left\| \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ & & 0 & \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \right\|_2}{\left\| \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \right\|_2}$$

$$= \max_{x\in\mathbb{R}^n,\ x\neq 0} \frac{\left\| \begin{pmatrix} \sigma_1 x_1 \\ \sigma_2 x_2 \\ \vdots \\ \sigma_n x_n \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right\|_2}{\left\| \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \right\|_2} = \frac{\sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2 + \cdots + (\sigma_n x_n)^2}}{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}}$$

$$\leq \frac{\sqrt{(\sigma_1 x_1)^2 + (\sigma_1 x_2)^2 + \cdots + (\sigma_1 x_n)^2}}{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}} = \sqrt{{\sigma_1}^2} \cdot \frac{\cancel{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}}}{\cancel{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}}} = \sigma_1 \tag{1.6.1}$$

The equality is achieved if we pick $\mathbf{x} = \mathbf{e_1} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$.

2. The proof of this assertion is similar to the other and it is left to the reader.

$\square$

Theorem 1.6.6 (Eckart-Young). *Let $A \in M(n, m, \mathbb{R})$ and let $A = U\Sigma V^T$ be its singular value decomposition.*

The solution of $\min\limits_{rk(X) \leq k} \|A - X\|_F$ is given by the truncated SVD*:*

$$X = \begin{pmatrix} U^1 & U^2 & \cdots & U^k \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_k \end{pmatrix} \cdot \begin{pmatrix} V^1 \\ V^2 \\ \vdots \\ V^k \end{pmatrix}$$

Where the norm is both $\|\cdot\|_2$ and $\|\cdot\|_F$.

Fact 1.6.7. *Let $A \in M(n, \mathbb{R})$ and let A be invertible. The following holds: $\|A^{-1}\| = \frac{1}{\sigma_n}$*

Proof. Since A is invertible, none of the σ_i is 0, hence the smaller (namely σ_n) is not 0.

$$A^{-1} = (U\Sigma V^T)^{-1} \overset{(1)}{=} V^{T^{-1}} \Sigma^{-1} U^{-1} = V \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} U^T$$

Where $\overset{(1)}{=}$ follows from the orthogonlaity of V and U.

Notice that this is *almost* an SVD, because the values on the diagonal are not sorted in a decreasing order.

Plugging in the norm, we have:

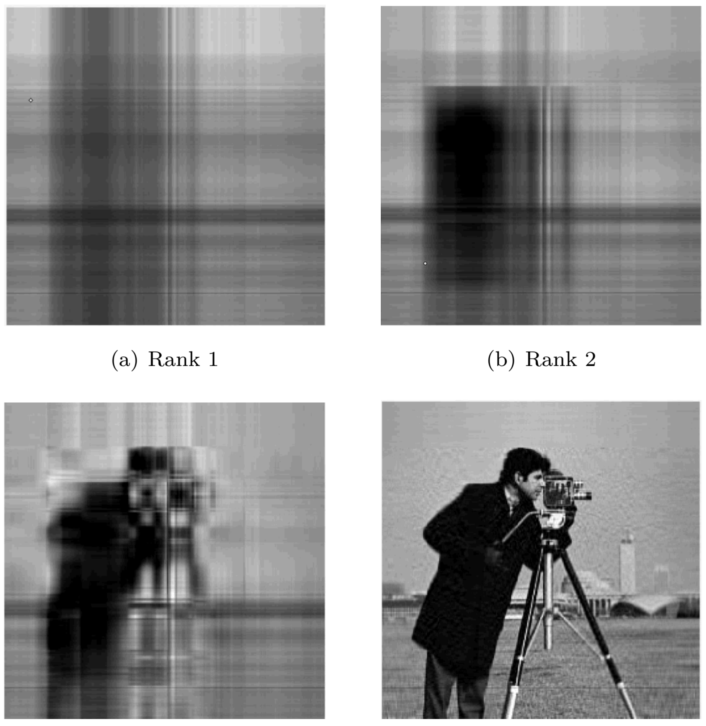
$$\|A^{-1}\| = \left\| V \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} U^T \right\| = \left\| \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} \right\| = \frac{1}{\sigma_n}$$

$\square$

Example 1.6.1. *For example, given a certain image, that can be represented as a matrix of values in the range $[0, 255]$, the rank-1 SVD of such image, results in a very abstract picture, see Figure 1.8. The more we increase the rank, the better is the similarity of the approximated image with respect to the original one.*

Something on Matlab ...

Given a certain matrix A, we can compute the SVD decomposition using the command `[U, S, V] = svd(A)`



(a) Rank 1 (b) Rank 2

(c) Rank 5 (d) Full rank

Figure 1.8: How the approximation of a matrix changes with respect to the different ranks.

Definition 1.6.8 (Principal component analysis)**.** *Given a matrix A, we term **principal component analysis** the analysis of features of such matrix via the rows and columns of U and V respectively, where U and V are the matrices of the SVD decomposition.*

Chapter 2

Least Squares Problem

2.1 LSP: the task

Do you recall?

In Definition 1.3.2 we said that for $A \in M(n, m, \mathbb{R})$, $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{x} \in \mathbb{R}^m$, the **least squares problem** is the following:

$$\min_{\mathbf{x} \in \mathbb{R}^m} \|A\mathbf{x} - \mathbf{b}\|$$

In this chapter we are going to present three different methods for solving the LS problem, that are:

- Normal Equations (direct method);
- QR Factorization (iterative method);
- SVD Factorization (iterative method).

2.2 NE direct method

2.2.1 Background

Fact 2.2.1. *Given $A \in \mathcal{M}(m, n, \mathbb{R})$ the following conditions are equivalent:*

- *$A^T A$ is positive definite;*
- *A has full column rank;*
- *the columns of A are linearly independent;*
- *$Ker(A) = \{0\}$.*

Fact 2.2.2. *Let $A \in \mathcal{M}(m,n,\mathbb{R})$, s.t. $Q = A^TA \in M(n,\mathbb{R})$ is* ***positive semidefinite****. The quadratic function $f(\mathbf{x}) = \mathbf{x}^TQ\mathbf{x} - \mathbf{q}^T\mathbf{x} + \mathbf{b}^T\mathbf{b}$ is* ***strongly*** *(or* ***strictly****)* ***convex****. In other words, $f(\mathbf{x})$ has a* ***unique*** *minimum.*

Fact 2.2.3. *Let $A \in \mathcal{M}(m,n,\mathbb{R})$, s.t. $Q = A^TA \in M(n,\mathbb{R})$ is* ***positive semidefinite****. The quadratic function $f(\mathbf{x}) = \mathbf{x}^TQ\mathbf{x} - \mathbf{q}^T\mathbf{x} + \mathbf{b}^T\mathbf{b}$ has gradient*

$$\nabla f(\mathbf{x}) = 2A^TA\mathbf{x} - 2A^T\mathbf{b}$$

Proof. We know that $f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + (\nabla f(\mathbf{x}))^T \cdot \mathbf{h} + o(\|\mathbf{h}\|)$

$$\begin{aligned} f(\mathbf{x}+\mathbf{h}) &= (\mathbf{x}+\mathbf{h})^TA^TA(\mathbf{x}+\mathbf{h}) - 2\mathbf{b}^TA(\mathbf{x}+\mathbf{h}) + \mathbf{b}^T\mathbf{b} \\ &= \mathbf{x}^TA^TA\mathbf{x} + \mathbf{x}^TA^TA\mathbf{h} + \mathbf{h}^TA^TA\mathbf{x} + \mathbf{h}^TA^TA\mathbf{h} - \mathbf{2b^TAx} - 2\mathbf{b}^TA\mathbf{h} + \mathbf{b^Tb} \\ &= f(\mathbf{x}) + (2x^TA^TA\mathbf{h} - 2\mathbf{b}^TA\mathbf{h}) + o(\|\mathbf{h}\|) \\ &= f(\mathbf{x}) + (2A^TA\mathbf{x} - 2A^T\mathbf{b})^T\mathbf{h} + o(\|\mathbf{h}\|) \end{aligned} \tag{2.2.1}$$

So, $\nabla f(\mathbf{x}) = 2A^TA\mathbf{x} - 2A^T\mathbf{b}$ □

We would like to know when the gradient is $\mathbf{0}$.

Fact 2.2.4. *Let $A \in \mathcal{M}(m,n,\mathbb{R})$, s.t. $Q = A^TA \in M(n,\mathbb{R})$ is* ***positive semidefinite****. The quadratic function $f(\mathbf{x}) = \mathbf{x}^TQ\mathbf{x} - \mathbf{q}^T\mathbf{x} + \mathbf{b}^T\mathbf{b}$ has a unique minimum in*

$$\mathbf{x} = (A^TA)^{-1}(A^T\mathbf{b})$$

Proof. $\nabla f(\mathbf{x}) \overset{?}{=} 0 \iff A^TA\mathbf{x} = A^T\mathbf{b}$ Since A^TA is a **square** matrix and also **non singular** (which means invertible) we may find $\mathbf{x}$ by solving a linear system $x^{-1} = (A^TA)^{-1}(A^T\mathbf{b})$. □

2.2.2 The closed formula

The theory exposed in the previous section comes in handy after some algebra on the least squares problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\| \equiv \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\|^2 &= \min_{\mathbf{x}\in\mathbb{R}^n} (A\mathbf{x}-\mathbf{b})^T \cdot (A\mathbf{x}-\mathbf{b}) \\ &= \min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{x}^TA^TA\mathbf{x} - \mathbf{x}^TA^T\mathbf{b} - \mathbf{b}^TA\mathbf{x} + \mathbf{b}^T\mathbf{b} \\ &= \min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{x}^TA^TA\mathbf{x} - 2\mathbf{b}^TA^T\mathbf{x} + \mathbf{b}^T\mathbf{b} \\ &\overset{*}{=} \min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{x}^TQ\mathbf{x} - \mathbf{q}^T\mathbf{x} + \mathbf{b}^T\mathbf{b} \end{aligned}$$

where $\overset{*}{=}$ follows from the substitutions $Q = A^TA$ and $\mathbf{q}^T = 2\mathbf{b}^TA$.

Therefore

LSP: closed formula

A solution to the least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\|$$

is given by

$$\mathbf{x} = (A^TA)^{-1}A^T\mathbf{b}$$

Notice that, independently of the shape of A, the matrix Q is a square matrix. The algorithm for computing $\mathbf{x}$ follows the steps:

- Compute A^TA, where the complexity is $2mn^2$, where $m > n$;
- Compute $A^T\mathbf{b}$, that has a complexity of $2mn$;
- Solve $A^TA\mathbf{x} = A^T\mathbf{b}$, which complexity depends on the algorithm.

Fact 2.2.5. *Let $A \in M(n, m, \mathbb{R}), \mathbf{b} \in \mathbb{R}^n$ and let the least squares problem be*

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\|$$

The residual $A\mathbf{x}-\mathbf{b}$ is orthogonal to any vector $\mathbf{v} \in span(A)$. Formally,

$$(A\mathbf{v})^T(A\mathbf{x}-\mathbf{b}) = \mathbf{0}$$

Proof. $\mathbf{v}^T(A^TA\mathbf{x} - A^T\mathbf{b}) = 0$. □

Definition 2.2.1 (Moore-Penrose pseudoinverse)**.** *Let A be a matrix in $\mathcal{M}(n, m, \mathbb{R})$.*

- *if A has* full column rank*, the **Moore-Penrose pseudoinverse** of A is* $\mathbf{A}^\dagger := (A^TA)^{-1}A^T$
- *if A has* full row rank*, the **Moore-Penrose pseudoinverse** of A is* $\mathbf{A}^\dagger := A^T(AA^T)^{-1}$

Thanks to this definition, we can rewrite the solution of a LS problem as $\mathbf{x} = A^\dagger\mathbf{b}$.

Notice that the solution of $\min \|A\mathbf{x} - (\mathbf{b}_1 + \mathbf{b}_2)\|$ can be written as the sum of two solutions of $\min \|A\mathbf{x_1} - \mathbf{b_1}\|$ and $\min \|A\mathbf{x_2} - \mathbf{b_2}\|$.

Fact 2.2.6. *Let A be a matrix in $\mathcal{M}(n, m, \mathbb{R})$.*

- *if A has* full column rank*, the Moore-Penrose pseudoinverse is a* left-inverse *of A: $A^\dagger A = I$*
- *if A has* full row rank*, the Moore-Penrose pseudoinverse is a* right-inverse *of A: $AA^\dagger = I$*

Let us study a different kind of methods for solving the least squares problem, namely *factorization* methods, that reveal properties of matrices and can be used as intermediate steps in algorithms.

2.3 QR Factorization

2.3.1 Background

Theorem 2.3.1 (QR factorization)**.** *$\forall A \in \mathcal{M}(m, \mathbb{R})$, $\exists Q \in \mathcal{O}(m, \mathbb{R})$, $\exists R \in \mathcal{T}(m, \mathbb{R})$ upper triangular such that $A = QR$.*

Fact 2.3.2. *Let $A \in M(n, m, \mathbb{R}), \mathbf{b} \in \mathbb{R}^n$. A solution to the least squares problem*

$$(P) \qquad \min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$$

can be computed as follows:

1. *first compute the QR factorization ($A = QR$) and then obtain $\mathbf{x} = A^{-1}\mathbf{b} = R^{-1}Q^{-1}\mathbf{b}$*
2. *compute then $\mathbf{c} = Q^T\mathbf{b}$*
3. *and then $\mathbf{x} = R^{-1}\mathbf{c}$*

And the computational cost is expressed as:

1. *$QR \ \rightarrow \ O(m^3)$*
2. *compute $\mathbf{c} \ \rightarrow \ O(m^2)$*
3. *compute $\mathbf{x} \ \rightarrow \ O(m^2)$*

Definition 2.3.1 (Housholder reflector)**.** *Let $\mathbf{v} \in \mathbb{R}^m$. An **Householder reflector** is a matrix $H \in M(m, \mathbb{R})$ such that*

$$H = I - \frac{2}{\mathbf{v}^T\mathbf{v}} \cdot \mathbf{v}\mathbf{v}^T$$

Equivalently, since $\mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2 \in \mathbb{R}$

$$H = I - \frac{2}{\|\mathbf{v}\|^2}\mathbf{v}\mathbf{v}^T = I - 2\mathbf{v}_u{\mathbf{v}_u}^T$$

where $vectv_u = \frac{1}{\|\mathbf{v}\|}\mathbf{v}$.

Lemma 2.3.3. *Householder reflectors are orthogonal.*

Proof.

$$\begin{aligned}
HH^T &= (I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T) \cdot (I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T) \\
&= I \cdot I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T I - I \cdot \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T + \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T \cdot \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T \\
&= I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v}\mathbf{v}^T + \frac{4}{\|\mathbf{v}\|^4} \cdot \mathbf{v}\mathbf{v}^T\mathbf{v}\mathbf{v}^T \\
&= I - \frac{4}{\|\mathbf{v}\|^2}\mathbf{v}\mathbf{v}^T + \frac{4}{\|\mathbf{v}\|^4}\mathbf{v}\|\mathbf{v}\|^2\mathbf{v}^T \\
&= I
\end{aligned} \tag{2.3.1}$$

□

Corollary 2.3.4. *The Householder reflector is Hermitian (symmetric, in the real case), unitary and involutory. Formally,*

$$H = H^{-1} = H^T$$

Notice that geometrically, an Householder reflector identifies an hyperplane and reflects any vector with respect to such hyperplane. See Figure 2.1

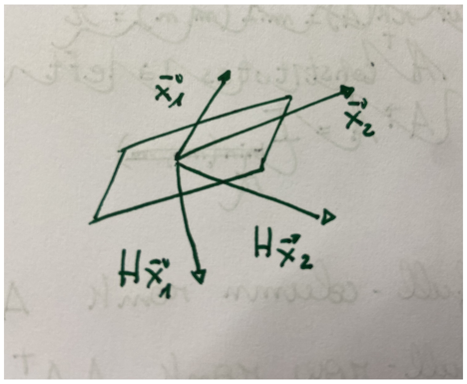


FIGURE 2.1: Reflection with respect to the hyperplane.

Example 2.3.1. *Let us be given the problem of computing the product $H\mathbf{x}$ for $H \in M(m, \mathbb{R})$ and $\mathbf{x} \in \mathbb{R}^m$.*

$$H\mathbf{x} = (I - \frac{2}{\|\mathbf{v}\|^2}\mathbf{v}\mathbf{v}^T)\mathbf{x} = \mathbf{x} - \frac{2}{\|\mathbf{v}\|^2}\mathbf{v}(\mathbf{v}^T\mathbf{x})$$

Let us first compute $a = \mathbf{v}^T \cdot \mathbf{x}$, then $b = \mathbf{v}^T \cdot \mathbf{v}$. We are now ready to compute the result $\mathbf{y} = \mathbf{x}\frac{2*a}{b} \cdot \mathbf{v}$

All these operations are linear operations, so the complexity is $O(m)$, cheaper than generic matrix-vector product ($O(m^2)$).

At this point, we are interested in finding an Householder reflector H that maps $\mathbf{x}$ to $\mathbf{y}$, formally $H\mathbf{x} = \mathbf{y}$ (Figure 2.2).

Lemma 2.3.5. $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^m$ *s.t* $\|\mathbf{x}\| = \|\mathbf{y}\| \exists H \in M(m, \mathbb{R})$ *s.t.* $H\mathbf{x} = \mathbf{y}$*, where the Householder matrix H is obtained from $\mathbf{v} = \mathbf{x} - \mathbf{y} \in \mathbb{R}^m$.*

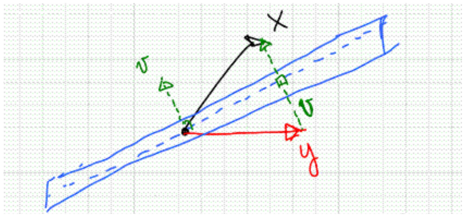


FIGURE 2.2: H is the hyperplane that "bisects" the angle between $\mathbf{x}$ and $\mathbf{y}$.

Example 2.3.2. *Let us take* $\mathbf{y} = \begin{pmatrix} \|\mathbf{x}\| \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^m$.

$$\mathbf{v} = \mathbf{x} - \mathbf{y} = \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix} - \begin{pmatrix} s \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix} = \begin{pmatrix} x_1 - s \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix}$$

where $s = \|\mathbf{x}\|$

Something on Matlab ...

In Matlab, a function is defined using the keyword `function` and specifying the return parameters, as follows: `function[v,s] = householder_vector(x)`, where x is the argument.

ALGORITHM 2.3.1 Householder vector Matlab implementation.

```
function[v,s] = householdervector(x)
s = norm(x);
v = x;
v(1) = v(1) - s;
v = v / norm(v);
```

Notice that the problem of **??** is the subtraction may create a problem with machine numbers, if s and $\|\mathbf{x}\|$ are very close. If we take $\|\mathbf{x}\| = -s$ the subtraction becomes and addition, and everything works well.

In the end, we would like to obtain this behaviour for every possible value for **x** and s, so line 2 may be modified as `s = - sign(x(1)) * norm(x)`.

2.3.2 Iterative algorithm via Householder's reflectors

Do you recall?

A **QR decomposition**, also known as a QR factorization or QU factorization, is a decomposition of a matrix $A \in M(n, \mathbb{R})$ into a product $A = QR$ of an orthogonal matrix Q and an upper triangular matrix R.

Terminology

From now on, for every matrix $M \in M(n, \mathbb{R})$ we will denote as $M(i : \text{end}, j)$ the vector taken as the rows from i to n of the j-th column of matrix M.

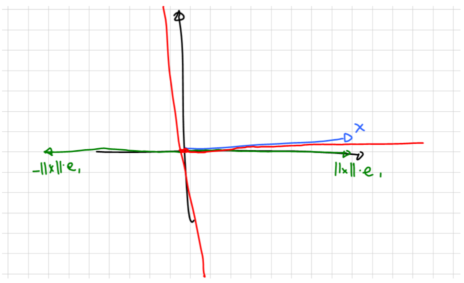


FIGURE 2.3: If $\mathbf{x}$ is oriented as in the plot it is better if we choose $-\|\mathbf{x}\|\mathbf{e_1}$ verse, because it is opposite to $\mathbf{x}$.

Example 2.3.3. *Let us start describing the algorithm through an example: given*

$$A = \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix} \in \mathcal{M}(m, \mathbb{R}),$$

where $m = 5$, we would like to calculate the QR factorization of A.

STEP 1 : *construct a Householder matrix $M \in M(m, \mathbb{R})$that sends $A(:, 1)$ (first column of A) to a multiple of $\mathbf{e_1}$ and keeps the rest of the matrix the same.*

Then we have $H_1 A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}$ *Let us write $Q_1 \in M(m, \mathbb{R})$*

as $Q_1 = H_1$.

Step 2 : *take* $H_2 \in \mathcal{M}(m-1,\mathbb{R})$ *such that* $H_2A(2:end,2) = \begin{pmatrix} \times \\ 0 \\ 0 \\ 0 \end{pmatrix}$ *and compute:*

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & & & & \\ 0 & & H_2 & & \\ 0 & & & & \\ 0 & & & & \end{pmatrix} \cdot Q_1 A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & & & & \\ 0 & & H_2 & & \\ 0 & & & & \\ 0 & & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \tag{2.3.2}$$

let us write $Q_2 \in M(m,\mathbb{R})$ *as a block matrix* $Q_2 = \begin{pmatrix} I_{1\times 1} & 0 \\ 0 & H_2 \end{pmatrix}$, $Q_1 = H_1$.

Therefore, we get $Q_2 \cdot Q_1 \cdot A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}$

Step 3 : *take* $H_3 \in \mathcal{M}(m-2,\mathbb{R})$ *such that* $H_3A(3:end,3) = \begin{pmatrix} \times \\ 0 \\ 0 \end{pmatrix}$ *and we compute:*

$$Q_3 \cdot (Q_2Q_1A) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & & & \\ 0 & 0 & & H_3 & \\ 0 & 0 & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \tag{2.3.3}$$

Where, $Q_3 = \begin{pmatrix} I_{2\times 2} & 0 \\ 0 & H_3 \end{pmatrix} \in M(m,\mathbb{R})$;

Step 4 *: take $H_4 \in \mathcal{M}(m-3, m-3, \mathbb{R})$ such that $H_4 A(4 : end, 4) = \begin{pmatrix} \times \\ 0 \end{pmatrix}$ and we compute:*

$$\begin{aligned} Q_4 \cdot (Q_3 Q_2 Q_1 A) &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & & \\ 0 & 0 & & H_4 & \\ 0 & 0 & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \\ &= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{pmatrix} = R \end{aligned} \tag{2.3.4}$$

Where, $Q_4 = \begin{pmatrix} I_{3\times 3} & 0 \\ 0 & H_4 \end{pmatrix}$.

In the end, since Q_is are an orthogonal matrices and the product of orthogonal matrices is orthogonal, $Q_1 \cdot Q_2 \cdot Q_3 \cdot Q_4 A = T$, which is an upper triangular matrix.

Theorem 2.3.6 (Product of block matrices)**.** *Let $I \in \mathcal{M}(k, \mathbb{R})$, let $H_i \in \mathcal{M}(m-k, \mathbb{R})$ and let $B_i \in \mathcal{M}(k, \mathbb{R})$, $C_i \in \mathcal{M}(k, m-k, \mathbb{R})$ and $A_i \in \mathcal{M}(m-k, m-k, \mathbb{R})$, then the product between the two following block matrices is exactly the one showed below.*

$$\begin{pmatrix} I & 0 \\ 0 & H_i \end{pmatrix} \cdot \begin{pmatrix} B_i & C_i \\ 0 & A_i \end{pmatrix} = \begin{pmatrix} B_i I & C_i \\ 0 & H_i \cdot A_i \end{pmatrix}$$

Proof. It's trivial computation, using the definition of matrix product. □

2.3.3 Matlab implementation

Algorithm 2.3.2 First implementation of QR factorization.

```
function [Q, R] = myqr(A)
[m, n] = size(A);
Q = eye(m);
for j = 1:n
v = householder_vector(A(j:end, j));
H = eye(length(v)) - 2*v*v';
A(j:end,j:end) = H * A(j:end,j:end);
Q(:, j:end) = Q(:, j:end) * H;
end
R = A;
```

Fact 2.3.7. *The cost of this implementation when A is a square matrix is $O(n^3 + (n-1)^3 + \cdots + 1^3)$. If A is a rectangular matrix, then the computational complexity is $O(m \cdot n^2 + (m-1) \cdot (n-1)^2 + \cdots + (m-n+1)^3)$.*

Proof. At line 7 we perform a matrix product between matrices of size n, $n-1$, $\dots$, 1, so the resulting cost is $O(m \cdot n^2 + (m-1) \cdot (n-1)^2 + \cdots + (m-n+1)^3)$. $\square$

We may design a faster algorithm, since $HA_j = A_j - 2v(v^T A_j)$.

Algorithm 2.3.3 More efficient implementation of QR factorization.

```
1  function [Q, A] = myqr(A)
2  [m, n] = size(A);
3  Q = eye(m);
4  for j = 1:n-1
5  [v, s] = householder_vector(A(j:end, j));
6  A(j,j) = s; A(j+1:end,j) = 0;
7  A(j:end,j+1:end) = A(j:end,j+1:end) - ...
8  2*v*(v'*A(j:end,j+1:end));
9  Q(:, j:end) = Q(:, j:end) - Q(:,j:end)*v*2*v';
10 end
```

2.3.4 QR factorization for tall-thin A

What if $A \in M(m, n, \mathbb{R})$ such that $m >> n$? Since $Q \in M(m, \mathbb{R})$ we would need a lot of space to store it. For this purpose the tall-thin QR factorization may come in handy.

Fact 2.3.8 (Thin QR factorization)**.** *We may replace $Q \in \mathcal{M}(m, \mathbb{R})$ and $R \in \mathcal{M}(m, \mathbb{R})$ with $Q_1 \in \mathcal{M}(m, n, \mathbb{R})$ and $R_1 \in \mathcal{M}(n, \mathbb{R})$ and the same factorization holds: $A = QR = Q_1 R_1$. This is called **thin QR factorization**.*

Proof. $A_1 \in \mathcal{M}(m, n, \mathbb{R})$, $A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \cdot \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q \cdot R = \begin{pmatrix} Q_1 \end{pmatrix} \cdot (R_1) + \begin{pmatrix} Q_2 \end{pmatrix} (0) = Q_1 \cdot R_1$ $\square$

The Matlab code can be modified computing Q as

$$\begin{pmatrix} 1 & \times & \cdots & \times \\ 0 & & & \\ \vdots & & \text{I-2}\mathbf{v}_1\mathbf{v}_1^T & \\ 0 & & & \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} I_{n-1} & 0 \\ 0 & \text{I-2}\mathbf{v}_2\mathbf{v}_2^T \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} I_n \\ \mathbf{0} \end{pmatrix}$$

In the end, this approach has a computational cost of $2mn^2 - \frac{2}{3}n^3 +)(mn)$ and we have two different scenarios:

1. $m \approx n$, then the cost is $\frac{4}{3}n^3$
2. $m >> n$, then the cost is linear in m which is the largest dimension

We have two different storing approaches:

1. Return $\mathbf{v_1}, \dots, \mathbf{v_n}$
2. Store Q

Fun fact

There are some libraries that store the $\mathbf{v_i}$ vectors in the lower part of matrix R which is upper triangular and has only zeros below the main diagonal.

2.3.5 QR factorization for LS problem

Do you recall?

Let us be given a matrix $A \in M(m, n, \mathbb{R})$ and a vector $\mathbf{b} \in \mathbb{R}^m$. We term **least squares problem** the problem of minimizing

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$$

We may write first the QR factorization of A, so $\forall A \in \mathcal{M}(m, n, \mathbb{R})$, $\exists Q \in O(m, \mathbb{R})$, $\exists R \in \mathcal{M}(m, n, \mathbb{R})$ such that $A = QR$, where

$$Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \text{ and } R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

Fact 2.3.9. *Let us be given a matrix $A \in M(m, n, \mathbb{R})$ and a vector $\mathbf{b} \in \mathbb{R}^m$. A solution of the minimum problem*

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$$

is $\mathbf{x} = {R_1}^{-1}{Q_1}^T\mathbf{b}$.

Proof.

$$\begin{aligned}
\|A\mathbf{x} - \mathbf{b}\| &\overset{(1)}{=} \|Q^T(A\mathbf{x} - \mathbf{b})\| = \|Q^T QR\mathbf{x} - Q^T\mathbf{b}\| \\
&= \|R\mathbf{x} - Q^T\mathbf{b}\| = \left\| \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \mathbf{x} - \begin{pmatrix} {Q_1}^T \\ {Q_2}^T \end{pmatrix} \mathbf{b} \right\| \\
&= \left\| \begin{pmatrix} R_1\mathbf{x} - {Q_1}^T\mathbf{b} \\ -{Q_2}^T\mathbf{b} \end{pmatrix} \right\|
\end{aligned}$$

where $\overset{(1)}{=}$ is due to the orthogonality of Q.

Provided that $-{Q_2}^T\mathbf{b}$ is constant with respect of $\mathbf{x}$, we can minimize the norm by picking $\mathbf{x}$ such that $R_1\mathbf{x} - {Q_1}^T\mathbf{b} = 0$. Hence, $\mathbf{x} = {R_1}^{-1}{Q_1}^T\mathbf{b}$. $\square$

Notice that we used the fact that R_1 is invertible, but is that always true?

Lemma 2.3.10. *R_1 is invertible $\Leftrightarrow$ A has full column rank.*

Proof. A has full column rank $\Leftrightarrow$ A^TA is positive definite $\Leftrightarrow$ A^TA is positive semidefinite and invertible, but A^TA is positive semidefinite, so we only need to prove its invertibility.

$$(QR)^TQR = R^TQ^TQR = R^TR = \begin{pmatrix} {R_1}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = {R_1}^T \cdot R_1$$

we need the quantity ${R_1}^T \cdot R_1$ to be invertible and this holds if and only if R_1 and ${R_1}^T$ are invertible, or equivalently iff R_1 is invertible. $\square$

Fact 2.3.11. *The cost of solving a LS problem via QR factorization is the summation of:*

1. *QR factorization $O(mn^2)$*

2. $\mathbf{y} = {Q_1}^T\mathbf{b}$

3. $\mathbf{x} = {R_1}^{-1}y \equiv R_1\mathbf{x} = \mathbf{y}$. *Solving this linear system only takes $O(n^2)$ operations, because we can perform* backsubstitution*: starting from the last row $y_n = r_{nn}x_n$, we can substitute upwards in the system and compute in each row one single unknown.*

Notice that, once the QR factorization has been computed, we can solve multiple LS problems with the same A by making only little effort.

Note

${R_1}^TR_1$ is the Cholesky factorization of A^TA.

2.4 SVD Factorization

Do you recall?

Tall thin SVD: A matrix $A \in M(m,n,\mathbb{R})$ can be written as $A = USV^T$, where $U \in O(n,\mathbb{R})$ is orthogonal, $S \in M(m,n,\mathbb{R})$ is a diagonal matrix and $V \in O(n,\mathbb{R})$ is orthogonal as well. In the case of a tall, thin A the decomposition has the following shape:

$$\begin{pmatrix} \\ \\ U^1 & U^2 & \cdots & U^m \\ \\ \\ \end{pmatrix} \cdot S \cdot \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix}^T$$

where

$$S = \begin{pmatrix} \sigma_1 & & & & & \\ & \sigma_2 & & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & \sigma_n \\ & & & 0 & & \end{pmatrix}$$

And if we denote U_1 the matrix obtained as the first n columns of U we have the tall, thin SVD: $A = U_1 \cdot S \cdot V^T$.

2.4.1 SVD for solving LS problems

Fact 2.4.1. *Let us be given a matrix $A \in M(m,n,\mathbb{R})$ and a vector $\mathbf{b} \in \mathbb{R}^m$. A solution of the minimum problem*

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$$

is $\mathbf{x} = VS^{-1}(U_1)^T\mathbf{b}$.

Proof.

$$\begin{aligned}
\|A\mathbf{x}-\mathbf{b}\| &= \left\|USV^T\mathbf{x}-\mathbf{b}\right\| && \leftarrow \text{ Def. of SVD} \\
&= \left\|U^T(USV^T\mathbf{x}-\mathbf{b})\right\| && \leftarrow U^T\text{is orthogonal} \\
&= \left\|SV^T\mathbf{x}-U^T\mathbf{b}\right\| && \leftarrow \text{Distributivity + orthogonality} \\
&= \left\|S\mathbf{y}-U^T\mathbf{b}\right\| && \leftarrow \mathbf{y}=V^T\mathbf{x} \\
&= \left\| \begin{pmatrix} \sigma_1 & & & & & \\ & \sigma_2 & & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & \sigma_n \\ & & & 0 & & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{pmatrix} - \begin{pmatrix} (U^1)^T\mathbf{b} \\ (U^2)^T\mathbf{b} \\ \vdots \\ (U^n)^T\mathbf{b} \\ (U^{n+1})^T\mathbf{b} \\ \vdots \\ (U^m)^T\mathbf{b} \end{pmatrix} \right\| && (2.4.1) \\
&= \left\| \begin{pmatrix} \sigma_1 y_1 - (U^1)^T\mathbf{b} \\ \sigma_2 y_2 - (U^2)^T\mathbf{b} \\ \vdots \\ \sigma_n y_n - (U^n)^T\mathbf{b} \\ -(U^{n+1})^T\mathbf{b} \\ \vdots \\ -(U^m)^T\mathbf{b} \end{pmatrix} \right\|
\end{aligned}$$

Where the first n rows may be assigned to 0 iff $y_i = -\frac{{U^i}^T\mathbf{b}}{\sigma_1}$ (if $\sigma_i \neq 0 \forall i$), while the latter $m-n$ do not depend on $\mathbf{y}$.

We compute $\mathbf{x}$ as

$$\begin{aligned}
\mathbf{x} &= V\mathbf{y} \qquad\qquad \leftarrow \text{Orthogonality of } V \\
&= V^1 y_1 + V^2 y_2 + \cdots + V^n y_n \\
&= V^1 \frac{1}{\sigma_1}(U^1)^T\mathbf{b} + V^2 \frac{1}{\sigma_2}(U^2)^T\mathbf{b} + \cdots + V^n \frac{1}{\sigma_n}(U^n)^T\mathbf{b} \\
&= V \cdot \begin{pmatrix} \frac{1}{\sigma_1} & & & & & \\ & \frac{1}{\sigma_2} & & & & \\ & & \cdot & & & 0 \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & \frac{1}{\sigma_n} \end{pmatrix} \cdot U^T \cdot \mathbf{b} \\
&= V \cdot \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & \frac{1}{\sigma_n} \end{pmatrix} \cdot {U_1}^T \cdot \mathbf{b}
\end{aligned} \tag{2.4.2}$$

where U_1 comes from the so-called thin SVD: $A = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \cdot \begin{pmatrix} S_1 \\ 0 \end{pmatrix} V^T = U_1 S_1 V^T$.
Notice that $V{S_{-1}}^{-1}{V_1}^T$ is the pseudoinverse of A: $A^\dagger = (A^T A)^{-1} A^T$ □

Fact 2.4.2. *The σ_i are different from 0 iff A has full column rank.*

Proof. A has full column rank

$$\Updownarrow$$

$A^T A$ is invertible

$$\Updownarrow$$

$$(USV^T)^T(USV^T) = VS^TU^TUSV^T = V \cdot \begin{pmatrix} \sigma_1^2 & & & & & \\ & \sigma_2^2 & & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & \sigma_n^2 \end{pmatrix} \cdot V^T \text{ is}$$

invertible

$$\Updownarrow$$

$\forall i\ \sigma_i \neq 0$, since V is orthogonal

□

Observation 2.4.1. *This lemma also proves that the factorization is also a QR factorization.*

Something on Matlab ...

`svd(A, 0)` and `qr(A, 0)` express that we are only interested in the parts of the factorization without zeros, in case of a tall, thin matrix A.

We may observe that the computational complexity is $O(15 \cdot n^3)$ for square matrices, while it's $O(m \cdot n^2)$ in the tall, thin case.

2.4.2 Behaviour in case of zeros as singular values

What happens when there are some zeros as singular values?

Do you recall?

We may recall that the singular values are sorted on the diagonal in decreasing order (the largest in top left position). From this assumption, we may say that if there are some $\sigma_i = 0$ then they are in the bottom right part of the matrix.

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$$

Fact 2.4.3. *Let us be given a matrix $A \in M(m, n, \mathbb{R})$ and a vector $\mathbf{b} \in \mathbb{R}^m$. A solution of the minimum problem*

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$$

where there are some 0 singular values. The solution is

$$\mathbf{x} = (V^1)^T \frac{1}{\sigma_1} (U^1)^T \mathbf{b} + \cdots + (V^r)^T \frac{1}{\sigma_r} (U^r)^T \mathbf{b}$$

Proof.

$$\|A\mathbf{x} - \mathbf{b}\| = \left\| \begin{pmatrix} \sigma_1 y_1 - (U^1)^T \mathbf{b} \\ \sigma_2 y_2 - (U^2)^T \mathbf{b} \\ \vdots \\ \sigma_r y_r - (U^r)^T \mathbf{b} \\ \sigma_{r+1} y_{r+1} - (U^{r+1})^T \mathbf{b} \\ \vdots \\ \sigma_n y_n - (U^n)^T \mathbf{b} \\ \sigma_{n+1} \cdot y_{n+1} - (U^{n+1})^T \mathbf{b} \\ \vdots \\ \sigma_m \cdot y_m - (U^m)^T \mathbf{b} \end{pmatrix} \right\| = \left\| \begin{pmatrix} \sigma_1 y_1 - (U^1)^T \mathbf{b} \\ \sigma_2 y_2 - (U^2)^T \mathbf{b} \\ \vdots \\ \sigma_r y_r - (U^r)^T \mathbf{b} \\ -(U^{r+1})^T \mathbf{b} \\ \vdots \\ -(U^n)^T \mathbf{b} \\ -(U^{n+1})^T \mathbf{b} \\ \vdots \\ -(U^m)^T \mathbf{b} \end{pmatrix} \right\|$$

No matter what value we choose for $y_{r+1} \cdots y_n$, the norm does not change since it's multiplied by 0. Therefore we get infinite solutions of the form $\mathbf{y} = \begin{pmatrix} \frac{({}^T U^1)\mathbf{b}}{\sigma_1} \\ \frac{(U^2)^T\mathbf{b}}{\sigma_2} \\ \vdots \\ \frac{(U^r)^T\mathbf{b}}{\sigma_r} \\ * \end{pmatrix}$

In order to make the solution unique, we can modify the problem, by taking the value that minimize the norm:

$$\min_{x \in \arg\min(\|Ax-\mathbf{b}\|)} \|\mathbf{x}\| . \tag{P2}$$

Notice that $\|\mathbf{x}\| = \|\mathbf{y}\|$, because $\mathbf{x} = V\mathbf{y}$. It follows from the expression of $\mathbf{y}$ that the choice that minimizes its norm is $y_{r+1} = \cdots = y_n = 0$.]

Hence, the solution of $P2$ is given by

$$\mathbf{x} = V\mathbf{y} = \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix} \cdot \begin{pmatrix} \frac{(U^1)^T\mathbf{b}}{\sigma_1} \\ \frac{(U^2)^T\mathbf{b}}{\sigma_2} \\ \vdots \\ \frac{(U^r)^T\mathbf{b}}{\sigma_r} \\ 0 \end{pmatrix} = (V^1)^T \frac{1}{\sigma_1} (U^1)^T \mathbf{b} + \cdots + (V^r)^T \frac{1}{\sigma_r} (U^r)^T \mathbf{b}$$

□

Note

For practical purposes, it is crucial to stress that when dealing with machine precision it might be the case that the singular values from $r+1$ to n are non zero, but very small. If the check of $\sigma_i = 0$ fails, $\frac{1}{\sigma_i}$, with $i > r$ becomes very big. A way to circumvent this problem is to find the linear dependencies between the columns, as shown in the following section.

2.4.3 Truncated SVD

In many real world setups first singular components correspond to the most prominent features of the dataset, while the smallest ones are fine details, noise or unreliable data due to the issue of machine precision stated above. Note, though, that in the sum $\sum_{i=1}^{n} V^i \frac{{U^i}^T\mathbf{b}}{\sigma_i}$ the small singular values may have a large impact, because σ_i is in the denominator.

We can modify the solution to cope with real world data problems:
$\mathbf{x} = \sum_{i=1}^{n} V^i \frac{(U^i)^T\mathbf{b}}{\sigma_i}$ becomes $\mathbf{x}_{\text{reg}} = \sum_{i=1}^{k} V^i \frac{(U^i)^T\mathbf{b}}{\sigma_i}$
for a certain k, ignoring small singular values.

Another way to modify the problem is the following.

2.4.4 Tikhonov's regularization / ridge regression

The Tikhonov regularization is a smoother version of truncated SVD.

$$\mathbf{x}_{\text{Tik}} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\|^2 + \alpha^2\|\mathbf{x}\|^2$$

Fact 2.4.4. *The Tikhonov regularization is equivalent to*

$$\mathbf{x}_{Tik} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \left\| \begin{pmatrix} A \\ \alpha I \end{pmatrix} \mathbf{x} - \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} \right\|^2 \tag{2.4.3}$$

Proof.

$$\begin{aligned} \|A\mathbf{x}-\mathbf{b}\|^2 + \alpha\cdot\|\mathbf{x}\|^2 &= \|A\mathbf{x}-\mathbf{b}\|^2 + \|\alpha\mathbf{x}\|^2 \\ &= \left\| \begin{pmatrix} A\mathbf{x}-\mathbf{b} \\ \alpha\mathbf{x} \end{pmatrix} \right\|^2 \\ &= \left\| \begin{pmatrix} A\mathbf{x} \\ \alpha\mathbf{x} \end{pmatrix} - \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} \right\|^2 \\ &= \left\| \begin{pmatrix} A \\ \alpha I \end{pmatrix} \mathbf{x} - \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} \right\|^2 \end{aligned}$$

□

Fact 2.4.5. *The solution of the Tikhonov regularization is given by the formula* $\mathbf{x}_{Tik} = (A^TA + \alpha^2 I)^{-1} A^T\mathbf{b}$.

Proof. Let us write the explicit solution of Equation (2.4.3) using the pseudoinverse

$$\begin{aligned} \mathbf{x}_{\text{Tikz}} = \begin{pmatrix} A \\ \alpha I \end{pmatrix}^{\dagger} \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} &= \left(\begin{pmatrix} A \\ \alpha I \end{pmatrix}^T \begin{pmatrix} A \\ \alpha I \end{pmatrix} \right)^{-1} \begin{pmatrix} A \\ \alpha I \end{pmatrix}^T \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} \\ &= \left(\begin{pmatrix} A^T & \alpha I \end{pmatrix} \begin{pmatrix} A \\ \alpha I \end{pmatrix} \right)^{-1} \begin{pmatrix} A^T & \alpha I \end{pmatrix} \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix} \\ &= (A^TA + \alpha^2 I)^{-1} A^T\mathbf{b}. \end{aligned}$$

□

Fact 2.4.6. *The matrix* $(A^TA + \alpha^2 I)$ *is positive definite.*

Proof. Let us take $\mathbf{z} \in (\mathbb{R}\setminus\{0\})^m$

$$\mathbf{z}^T \cdot (A^T A + \alpha^2 I) \cdot \mathbf{z} = \underbrace{\mathbf{z}^T A^T A \mathbf{z}} \geq 0 + \underbrace{\alpha^2 \mathbf{z}^T \mathbf{z}}_{=\alpha^2 \|\mathbf{x}\|^2 > 0}$$

where $\mathbf{z}^T A^T A\mathbf{z} \geq 0$ holds since $A^T A \succeq 0$. □

Fact 2.4.7. *Let us be given a matrix $A \in M(m, n, \mathbb{R})$ and a vector $\mathbf{b} \in \mathbb{R}^m$. Tikhonov's solution of the minimum problem*

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$$

can be written as

$$\mathbf{x}_{Tik} = \sum_{i=1}^{n} V^i \frac{\sigma_i}{\sigma_i^2 + \alpha^2} (U^i)^T \mathbf{b}.$$

where σ_i, $\forall i = 1, \ldots, m$ are the singular values of A.

Fare proof

Notice that, if we denote $f : \mathbb{R} \to \mathbb{R}$ *s.t.* $f(\sigma) = \frac{\sigma}{\sigma^2+\alpha^2}$:

- if $\sigma \gg \alpha$, then $f(\sigma) \approx \frac{1}{\sigma}$
- if $\sigma \ll \alpha$, then $f(\sigma) \approx \frac{\sigma}{\alpha^2}$

This behaviour simulates the truncated SVD, as shown in Figure 2.4.

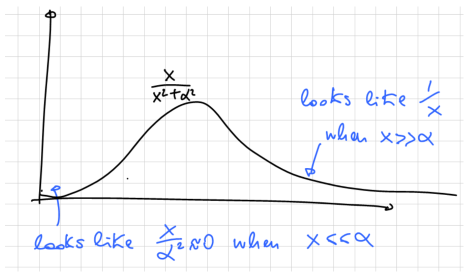


FIGURE 2.4: Here is the shape of the formula for the singular values.

In practice, the value α (or k) is chosen as the "amount of noise" expected in our data, meaning that all the σ values that are dropped are those which have an absolute value smaller than the noise.

2.5 LSP: wrap up

So far, we presented three methods for solving a least squares problem in addition to the classical gradient method descent. Let us now compare the performances:

	Normal Equations	**QR**	**SVD**	**GD**
$m \approx n$	$\frac{4}{3}n^3$	$\frac{4}{3}n^3$	$\approx 13n^3$	??
$m \gg n$	mn^2	$2mn^2$	$2mn^2$	??

Table 2.1: Comparison between algorithms for solving LS problem

According to this table, one could ask himself why to choose SVD, since for the almost square case it is much slower than the other methods. The answer is that it works much better, providing much more information in the singular case.

Moreover, since the direct Normal Equations method is always the fastest why did we need to build the other models? In the case of an "almost singular" matrix A, the result is very far from the optimum. An example of such a matrix $A \in M(n, 3, \mathbb{R})$ is

$$A = \begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_1} + \mathbf{v_2} + \varepsilon \mathbf{e_n} \end{pmatrix}$$

Chapter 3

Conditioning And Stability

3.1 Conditioning

Definition 3.1.1 (Sensitivity)**.** *We call* ***sensitivity*** *the measure of how much the output of a problem changes when we perturb its input.*

Example 3.1.1. *As an example, let:*

1. $f : \mathbb{R} \to \mathbb{R}$ *such that* $f(x, y) = x + 2y$
2. $g : \mathbb{R} \to \mathbb{R}$ *such that* $g(x) = x^2$

Let us make some perturbations of the parameters of f *and* g*:*

1. (f, x)*:*
$$\mathbf{v} = f(\widetilde{x}, y) - f(x, y) = x + \delta + 2y - (x + 2y) = \delta$$
(f, y)*:*
$$\mathbf{v} = f(x, \widetilde{y}) - f(x, y) = x + 2(y + \delta) - (x + 2y) = 2\delta.$$
We can conclude that f *is more sensitive to* y *than to* x*.*
2. (g, x)*:*
$$\mathbf{v} = f(\widetilde{x}) - f(x) = (x + \delta)^2 - x^2 = x^2 + 2\delta x + \delta^2 - x^2 = 2\delta x + \delta^2 = 2\delta x + o(\delta)$$

A real life example of very sensitive function is the temperature of water coming from the shower: in particular, when we rotate little the knob the water becomes too cold or too hot very fast, see Figure 3.5.

Definition 3.1.2 (Absolute condition measure)**.** *The* ***absolute condition number*** *of a univariate function* $f : \mathbb{R} \to \mathbb{R}$ *is the* ***maximum*** *possible output change / input change ratio in the limit for a* ***small*** *change of the input.*

$$\kappa_{abs}(f, x) = \lim_{\varepsilon \to 0} \sup_{|\tilde{x} - x| \le \varepsilon} \frac{|f(\tilde{x}) - f(x)|}{|\tilde{x} - x|}.$$

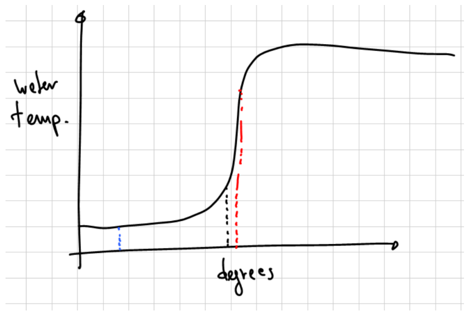


FIGURE 3.1: Geometric idea of temperature of the water in the shower

Equivalently, we term ***condition number*** *of the problem $f : \mathbb{R} \to \mathbb{R}$ s.t. $y = f(x)$ with respect to an input x as the best bound of the form*

$$|f(x+\delta) - f(x)| \leq \kappa|\delta| + \underbrace{o(\delta)}_{\textit{higher order terms}}$$

Or, equivalently, if f is differentiable,

$$\kappa_{abs}(f,x) = \frac{\partial f}{\partial x}$$

Example 3.1.2. *Let us proceed further with the previous example:*

1. $\kappa_{abs}(f,x)$:

$$\kappa_{abs}(f,x) = \lim_{\delta\to 0} \frac{|\delta|}{|\delta|} = 1$$

$\kappa_{abs}(f,y)$:

$$\kappa_{abs}(f,y) = \lim_{\delta\to 0} \frac{|2\delta|}{|\delta|} = 2$$

We can conclude that f is more sensitive to y than to x.

2. $\kappa_{abs}(g,x)$:

$$\kappa_{abs}(g,x) = \lim_{\delta\to 0} \frac{\left|2x\delta + \delta^2\right|}{|\delta|} = 2x$$

Definition 3.1.3 (Absolute condition measure - multivariate)**.** *The* ***absolute condition number*** *of a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$ is the* ***maximum*** *possible output change / input change ratio in the limit for a* ***small*** *change of the input.*

$$\kappa_{abs}(f,\mathbf{x}) = \lim_{\delta\to 0} \sup_{\|d\|\leq\delta} \frac{\|f(\tilde{\mathbf{x}}) - f(\mathbf{x})\|}{\|\tilde{\mathbf{x}} - \mathbf{x}\|}.$$

Equivalently, if f is differentiable,

$$\kappa_{abs}(f,\mathbf{x}) = \|\nabla f(\mathbf{x})\|$$

Example 3.1.3. *Let $f(x) = x^T Qx$, for instance for $x \in \mathbb{R}^2$ so that we can plot its graph in $\mathbb{R}^3$.*

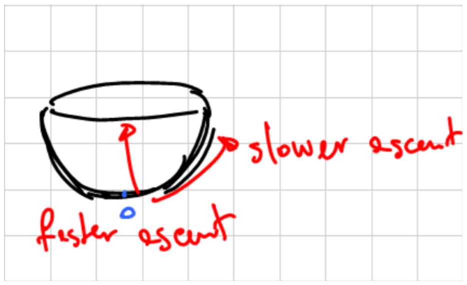


FIGURE 3.2: Paraboloid

We shall take a general example where the cross-section are ellipses, so that there is a direction of faster and slower ascent; this is not just a circular "cup" seen in perspective. Note that these directions of faster ascent and lower ascent correspond to the eigenvectors of the matrix Q.

In this case the absolute condition number is $\lim\limits_{\varepsilon \to 0} \max\limits_{\widetilde{x} \in B(x,\varepsilon)} \frac{\left\| f(\widetilde{x}) - f(x) \right\|}{\left\| \widetilde{x} - x \right\|}$, and one can see that the output/input ratio varies with the direction in which $\tilde{x}$ is, so we have to take a maximum in the whole ball $B(x, \varepsilon)$.

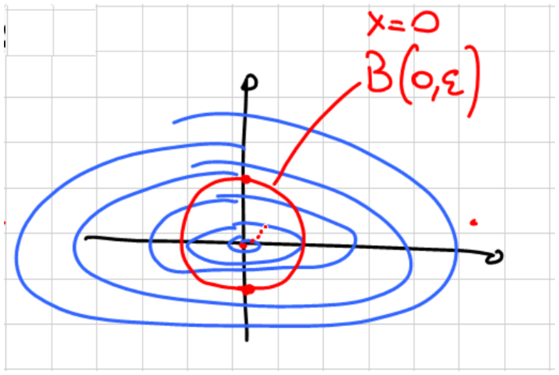


FIGURE 3.3: Level curves of a quadratic function ("seen from above").

Example 3.1.4. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ such that $f(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x_1 + x_2$.*

- $\mathbf{d} = \delta \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$$f(\mathbf{x} + \mathbf{d}) - f(\mathbf{x}) = (x_1 + \delta) + (x_2 + \delta) - (x_1 + x_2) = 2\delta$$

$$\lim_{\delta \to 0} \frac{\|f(\mathbf{x}+\mathbf{d}) - f(\mathbf{x})\|}{\|\mathbf{d}\|} = \lim_{\delta \to 0} \frac{2\delta}{\sqrt{2}\delta} = \sqrt{2}$$

- $\mathbf{d} = \delta \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

$$f(\mathbf{x}+\mathbf{d}) - f(\mathbf{x}) = (x_1+\delta) + (x_2+\delta) - (x_1+x_2) = 2\delta$$

$$\lim_{\delta \to 0} \frac{\|f(\mathbf{x}+\mathbf{d}) - f(\mathbf{x})\|}{\|\mathbf{d}\|} = \lim_{\delta \to 0} \frac{0}{\sqrt{2}\delta} = 0$$

- *general direction* $\mathbf{d} \in \mathbb{R}^2$

$$\begin{aligned} \|f(\mathbf{x}+\mathbf{d}) - f(\mathbf{x})\| &= \left\| \begin{pmatrix} 1 & 1 \end{pmatrix} (\mathbf{x}+\mathbf{d}) - \begin{pmatrix} 1 & 1 \end{pmatrix} \mathbf{x} \right\| \\ &= \left\| \begin{pmatrix} 1 & 1 \end{pmatrix} \mathbf{d} \right\| \leq \left\| \begin{pmatrix} 1 & 1 \end{pmatrix} \right\| \cdot \|\mathbf{d}\| = \sqrt{2}\,\|\mathbf{d}\| \end{aligned}$$

Since we found a direction $\mathbf{d} = \delta \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ *that matches the upper bound of the conditioning, we get that* $\kappa_{abs} = \sqrt{2}$.

In general, whenever we are computing a change of something, it is better to normalize it to the value of the change.

Definition 3.1.4 (Relative error)**.** *The* ***relative error*** *of an approximation* $\widetilde{\mathbf{x}}$ *to a quantity* $\mathbf{x}$ *is the quantity*

$$\frac{\|\widetilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|}$$

Definition 3.1.5 (Relative condition number)**.** *The* ***relative condition number*** *of a function* $f : \mathbb{R}^n \to \mathbb{R}$ *is defined as*

$$\kappa_{rel}(f, \mathbf{x}) = \lim_{\delta \to 0} \sup_{\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \delta} \frac{\frac{\|f(\tilde{\mathbf{x}}) - f(\mathbf{x})\|}{\|f(\mathbf{x})\|}}{\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|}} = \kappa_{abs}(f, \mathbf{x}) \frac{\|\mathbf{x}\|}{\|f(\mathbf{x})\|},$$

where $\widetilde{\mathbf{x}} = \mathbf{x} + \mathbf{d}$. *With respect to the absolute condition number, we replaced the absolute error* $\|\tilde{\mathbf{x}} - \mathbf{x}\|$ *with the relative error.*

Here are some examples of good and bad accuracy:

- $\frac{|\tilde{x}-x|}{|x|} \approx 1$: **very bad** accuracy; it's just a number with the same order of magnitude.
- $\frac{|\tilde{x}-x|}{|x|} \approx 10^{-3}$: about 3 correct significant digits.
- $\frac{|\tilde{x}-x|}{|x|} \approx 10^{-16}$: about 16 correct digits; we **can't do better** typically (with `double` precision arithmetic).

3.1.1 Conditioning of linear systems

Do you recall?

Let $A \in M(n,m,\mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^n$. A **linear system** expresses the problem of finding $\mathbf{x} \in \mathbb{R}^m$ such that

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

Computing the condition number of a linear system means computing the condition number of the function $f(A,\mathbf{b}) = A^{-1}\mathbf{b}$, perturbing the inputs A and $\mathbf{b}$, one at a time.

Perturbing $\mathbf{b}$ We want to compute the limit of the relative error $\frac{\left\|f(A,\widetilde{\mathbf{b}})-f(A,\mathbf{b})\right\|}{\|f(A,\mathbf{b})\|}$, so we set $\mathbf{x} = A^{-1}\mathbf{b}$ and $\tilde{\mathbf{x}} = A^{-1}\tilde{\mathbf{b}}$, and we estimate *output error* $= \frac{\left\|\widetilde{\mathbf{x}}-\mathbf{x}\right\|}{\|\mathbf{x}\|} =?$

1.
$$\begin{aligned}\|\widetilde{\mathbf{x}} - \mathbf{x}\| &= \left\|A^{-1}\widetilde{\mathbf{b}} - A^{-1}\mathbf{b}\right\| \\ &= \left\|A^{-1}(\widetilde{\mathbf{b}} - \mathbf{b})\right\| \\ &\leq \left\|A^{-1}\right\| \cdot \left\|\widetilde{\mathbf{b}} - \mathbf{b}\right\| \\ &= \left\|A^{-1}\right\| \cdot \|\mathbf{b} + \mathbf{d} - \mathbf{b}\| \\ &= \left\|A^{-1}\right\| \cdot \|\mathbf{d}\|\end{aligned} \tag{3.1.1}$$

2. $\|\mathbf{b}\| = \|A\mathbf{x}\| \leq \|A\| \cdot \|\mathbf{x}\|$

At this point we are ready to make the whole computation:

$$\underbrace{\frac{\|\widetilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|}}_{\text{rel. out. change}} \leq \frac{\left\|A^{-1}\right\| \cdot \|\mathbf{d}\|}{\frac{\|\mathbf{b}\|}{\|A\|}} = \underbrace{\|A\| \cdot \left\|A^{-1}\right\|}_{\kappa_{\text{rel}(f,\mathbf{b})}} \cdot \underbrace{\frac{\|\mathbf{d}\|}{\|\mathbf{b}\|}}_{\text{rel. in. change}}$$

In the end, since *input error* $= \frac{\left\|\widetilde{b}-b\right\|}{\|b\|}$ we obtain

$$\kappa_{rel}(f,x) = \lim_{\varepsilon\to 0} \frac{\textit{output error}}{\textit{input error}} \leq \lim_{\varepsilon\to 0} \left\|A^{-1}\right\| \cdot \|A\| = \left\|A^{-1}\right\| \cdot \|A\|$$

We denote $\kappa(A) = \left\|A^{-1}\right\| \cdot \|A\|$ the **condition number of** A;

Perturbing A Given $A\mathbf{x} = \mathbf{b}$ we obtain $(A + \Delta_A) \cdot (\mathbf{x} + \Delta_\mathbf{x}) = \mathbf{b}$, where $\widetilde{A} = A + \Delta_A$ and $\widetilde{\mathbf{x}} = \mathbf{x} + \Delta_\mathbf{x}$. Then we can expand as follows

$\cancel{A\mathbf{x}} + \Delta_A \cdot \mathbf{x} + A \cdot \Delta_\mathbf{x} + \Delta_A \cdot \Delta_\mathbf{x} = \cancel{\mathbf{b}}$

We can stop taking into account $\Delta_A \cdot \Delta_{\mathbf{x}}$, since it's a sort of second order term ($\Delta_A \cdot \Delta_{\mathbf{x}} = o(\|\Delta_A\| \cdot \|\Delta_{\mathbf{x}}\|)$), so we get the following

$$\Delta_A \cdot \mathbf{x} + A \cdot \Delta_{\mathbf{x}} = 0$$

$$\Delta_{\mathbf{x}} = -A^{-1} \cdot \Delta_A \cdot \mathbf{x}$$

then $\|\Delta_{\mathbf{x}}\| \leq \left\|A^{-1}\right\| \cdot \|\Delta_A\| \cdot \|\mathbf{x}\|$, which implies $\frac{\|\Delta_{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \left\|A^{-1}\right\| \cdot \frac{\|\Delta_A\|}{\|A\|}$.

We obtain that *relative output error* $\leq \kappa(A) \cdot$ *relative input error*.

We only proved an inequality, but it turns out that it is tight: for every A and $\mathbf{b}$ there is a possible choice of the perturbation $\tilde{\mathbf{x}}$ that attains equality.

Terminology

We call **ill-conditioned** problems, those which have a high condition number. Conversely, we term **well-conditioned** problems those which have a small condition number.

Condition number, SVD, and distance to singularity

Fact 3.1.1. *Let $A \in M(n, m, \mathbb{R})$. The conditioning of A ($\kappa(A)$) is the ratio between the smallest and the largest singular value. Formally, $\kappa(A) = \frac{\sigma_1}{\sigma_n}$.*

Proof. Let $A = USV^T$, then $\|A\| = \left\|U \cdot S \cdot V^T\right\| = \|S\| = \sigma_1$, since

$$S = \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & \sigma_n \end{pmatrix} \text{ and } \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n.$$

It's also true that $\left\|A^{-1}\right\| = \left\|(USV)^{-1}\right\| = \left\|VS^{-1}U^T\right\| = \left\|S^{-1}\right\| = \sigma_n$, since

$$S = \begin{pmatrix} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & \frac{1}{\sigma_n} \end{pmatrix} \text{ and } \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n.$$

In the end $\kappa(A) = \frac{\|A\|}{\|A^{-1}\|} = \frac{\sigma_1}{\sigma_n}$ □

Notice that we cannot use the other definition $\kappa(A) = \|A\| \left\|A^{-1}\right\|$, since A^{-1} does not exist for a non-square A. However, one can verify that $\|A\| \cdot \left\|A^{\dagger}\right\| = \frac{\sigma_1}{\sigma_n} = \kappa(A)$, where $A^{\dagger}$ is the pseudo-inverse.

Fact 3.1.2. *Let $A \in M(n, m, \mathbb{R})$. The relative distance between A and the closest singular matrix is computed as $\frac{1}{\kappa(A)}$.*

Do you recall?

As stated in **Eckart-Young theorem** (Theorem 1.6.6), let $A \in M(n,m,\mathbb{R})$ and let $A = U\Sigma V^T$ be its singular value decomposition. The closest matrix to A that has rank $r = n-1$ is the solution of $\min_{rk(X)=n-1} \left\| A - \hat{A} \right\|_F$:

$$\hat{A} = \begin{pmatrix} U^1 & U^2 & \cdots & U^{n-1} \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_{n-1} \end{pmatrix} \cdot \begin{pmatrix} V^1 \\ V^2 \\ \vdots \\ V^{n-1} \end{pmatrix}$$
$$= U \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_{n-1} & \\ & & & & 0 \end{pmatrix} \cdot V^T$$

Proof.

$$\begin{aligned} \left\| A - \hat{A} \right\| &= \left\| U \cdot \left(\begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_{n-1} & \\ & & & & \sigma_n \end{pmatrix} - \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_{n-1} & \\ & & & & 0 \end{pmatrix} \right) \cdot V^T \right\| \\ &= \left\| U \cdot \begin{pmatrix} 0 & & & \\ & \ddots & & \\ & & 0 & \\ & & & \sigma_n \end{pmatrix} \cdot V^T \right\| \\ &= \sigma_n \end{aligned} \tag{3.1.2}$$

Thus, $\frac{\left\| A - \hat{A} \right\|}{\|A\|} = \frac{\sigma_n}{\sigma_1} = \frac{1}{\kappa(A)}$. □

Proposition 3.1.2 states that the larger $\kappa(A)$ the closest is A to singularity.

We analyzed the conditioning of linear systems, but the main problem we want to study in this course is least squares problem.

3.1.2 Conditioning of least squares problem

Theorem 3.1.3 (Trefethen, Bau, Theorem 18.1)**.** *Let us be given a matrix $A \in M(m,n,\mathbb{R})$ with full column rank and a vector $\mathbf{b} \in \mathbb{R}^m$. Consider the* *minimum problem*

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\|$$

The relative condition number of f with respect to the input $\mathbf{b}$ satisfies

$$\kappa_{rel}(f,\mathbf{b}) \le \frac{\kappa(A)}{\cos\theta} \tag{3.1.3}$$

and the relative condition number of f with respect to A is such that

$$\kappa_{rel}(f,A) \le \kappa(A) + \kappa(A)^2 \tan\theta, \tag{3.1.4}$$

where θ is the angle such that $\cos\theta = \frac{\|A\mathbf{x}\|}{\|\mathbf{b}\|}$.

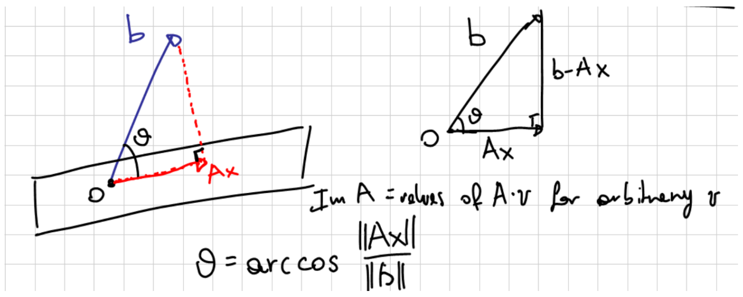

b
θ
O
Ax
b
b-Ax
θ
O
Ax
Im A = values of A·v for arbitrary v
θ = arccos ||Ax|| / ||b||

Figure 3.4: A geometric idea of the meaning of the angle θ.

Observation 3.1.1.

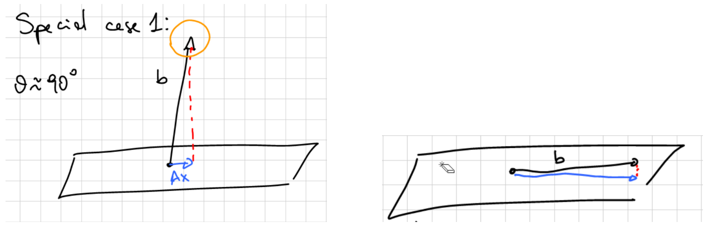
Special case 1: $\theta \approx 90°$ *We can see from the figure that a big change of $\mathbf{b}$ induces a big relative perturbation of $A\mathbf{x}$. No matter what the conditioning of A is, a small (relative) perturbation in $\mathbf{b}$ can cause a huge (relative) perturbation in $A\mathbf{x}$, see Figure 3.5(a).*

Special case 2: $\theta \approx 0°$ *When $\mathbf{b}$ is almost in plane with $Im(x)$. In this case $\kappa_{rel}(f,A) \approx \kappa(A)$ and a big relative change in $\mathbf{b}$ does not impact much $A\mathbf{x}$, see Figure 3.5(b).*

General case: θ far from $0°$ and $90°$ *In the more general case, $\kappa_{rel}(f,A) \approx \kappa(A)^2$.*

3.2 Floating point numbers

Computers work with IEEE arithmetic, where floating point numbers are expressed in base-2 scientific (exponential) notation.



(a) $\theta \approx 90°$ (b) $\theta \approx 0°$

FIGURE 3.5: Two different angles between **b** and the hyperplane.

`double` (64-bit numbers):

$$\pm 1.\underbrace{01001011101\ldots101}_{\text{52 binary digits}} \cdot 2^{\pm \overbrace{}^{}\underbrace{101\ldots01}_{\text{10 binary digits}}}.$$

We use 1 bit for the sign, 52 bits for the "mantissa" and 11 bits for the exponent and its sign.
Some of these combinations of bits are reserved for special numbers, e.g. `Inf` and `NaN`, `-0`.
This system is subject to approximation errors, exactly like the "usual" decimal arithmetic: for example, if we do $\frac{1}{3} = 0.33333\ldots$ and if we do $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 0.99999\ldots \neq 1$.

Moreover, not all numbers are exactly representable, take $x = 0.1_{\text{dec}} = 0.0\overline{0011}_{\text{bin}}$. It is a periodic number when written in binary, hence we cannot represent it exactly as a machine number, hence it should be rounded up to $\widetilde{x}$, the closest floating point number.

Notice that the precision of the numbers that can be stored in floating point arithmetic is variable, in particular there are more numbers close to zero with respect to those that are far from 0, see Figure 3.6.

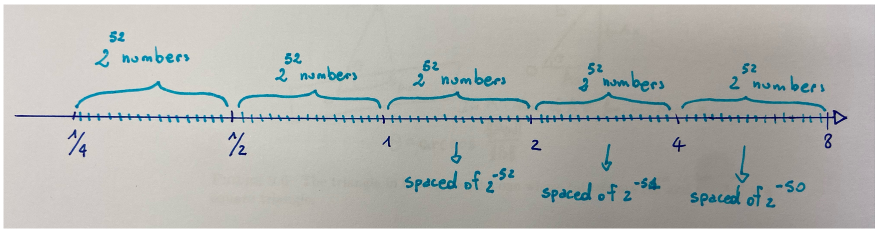


FIGURE 3.6: We have 2^{52} equispaced numbers between $\frac{1}{2}$ and 1 and between 1 and 2, and 2^{52} between 2 and 4 and so on and so forth, so we have the same number of integers, although the interval is getting bigger and bigger.

Definition 3.2.1 (Error bound)**.** *On every machine, for each $x \in \pm[10^{-308}, 10^{308}]$, there is an exactly representable number $\tilde{x}$ that satisfies the **error bound**, namely* *it is closer to x than the constant $\mathsf{u} = 2^{-52} \approx 10^{-16}$. Formally, $\frac{|\tilde{x}-x|}{|x|} \leq \mathsf{u}$. Equivalently, $\tilde{x} = x \cdot (1 \pm \delta)$, where $|\delta| \leq \mathsf{u}$.*

3.3 Stability of algorithms

In this lecture we will try to answer the question: "Is our algorithm going to compute a good approximation of the solution, provided that all the operations are performed with machine precision?"

The concept of *stability* is linked with the particular algorithm used.

Let us assume that we have the best possible algorithm that returns $\tilde{y} = f(\tilde{x})$ which is the best representation of $f(\tilde{x})$, then by definition we have

$$\begin{aligned}\frac{|\tilde{y}-y|}{|y|} &\leq \kappa_{rel}(f,x)\frac{|\tilde{x}-x|}{|x|} + o(\frac{|\tilde{x}-x|}{|x|}) \\ &\leq \kappa_{rel}(f,x)\mathsf{u} + o(\mathsf{u}).\end{aligned}$$

In practice we may ignore $o(u)$, since it has order of magnitude greater than 2^{-16}.

Terminology

From now on, we will denote floating point operations with circled operators, such as $\odot, \oplus$, where

$$a \oplus b = (a+b)(1+\delta), \quad |\delta| \leq \mathsf{u} \iff \frac{|(a \oplus b) - (a+b)|}{|a+b|} = \delta$$

and

$$a \odot b = ab(1+\delta), \quad |\delta| \leq \mathsf{u} \iff \frac{|(a \odot b) - ab|}{|ab|} = \delta$$

Example 3.3.1. *We would like to compute the error on the function $f : \mathbb{R}^3 \to \mathbb{R}$ s.t.*

$$f(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

Step1: computer result

$$\begin{aligned}
\tilde{y} &= a_1 \odot b_1 \oplus a_2 \odot b_2 \oplus a_3 \odot b_3 \\
&= \left[\left[a_1b_1 \cdot (1+\delta) + a_2b_2 \cdot (1+\delta_2)\right] \cdot (1+\delta_4) + a_3b_3 \cdot (1+\delta_3)\right] \cdot (1+\delta_5) \\
&= a_1b_1 \cdot (1+\delta_1) \cdot (1+\delta_4) \cdot (1+\delta_5) + a_2b_2 \cdot (1+\delta_2) \cdot (1+\delta_4) \cdot (1+\delta_5) \\
&+ a_3b_3 \cdot (1+\delta_3) \cdot (1+\delta_5) \\
&= a_1b_1(1+\delta_1+\delta_4+\delta_5+O(u^2)) \\
&+ a_2b_2 \cdot (1+\delta_1+\delta_4+\delta_5+O(u^2)) \\
&+ a_3b_3 \cdot (1+\delta_3+\delta_5+O(u^2)) \\
&\approx a_1b_1(1+\delta_1+\delta_4+\delta_5) + a_2b_2 \cdot (1+\delta_1+\delta_4+\delta_5) + a_3b_3 \cdot (1+\delta_3+\delta_5) \\
&= y + a_1b_1(\delta_1+\delta_4+\delta_5) + a_2b_2(\delta_1+\delta_4+\delta_5) + a_3b_3(\delta_3+\delta_5)
\end{aligned}$$

Where $O(u^2)$ comes from the summation of $\delta_i\delta_j$, for some i, j and allows us to do an approximation up to second order terms of precision.

Step2: abolute error

$$\begin{aligned}
|\tilde{y} - y| &= |\cancel{y} + a_1b_1(\delta_1+\delta_4+\delta_5) + a_2b_2 \cdot (\delta_1+\delta_4+\delta_5) + a_3b_3 \cdot (\delta_3+\delta_5) - \cancel{y}| \\
&\overset{(1)}{\leq} |a_1b_1| \cdot 3u + |a_2b_2| \cdot 3u + |a_3b_3| \cdot 2u \\
&\leq (|a_1b_1| + |a_2b_2| + |a_3b_3|) \cdot 3u
\end{aligned}$$

Where $\overset{(1)}{\leq}$ follows from the observation that $|\delta_i| \leq u$

Let us take $\mathbf{a} = \begin{pmatrix} 1 & -1 & 0 \end{pmatrix}$ *and* $\mathbf{b} = \begin{pmatrix} 10^6+1 \\ 10^6 \\ 1 \end{pmatrix}$*, then*

$$f(\mathbf{a}, \mathbf{b}) = 10^6 + 1 - 10^6 + 1 = 2$$

while

$$|\tilde{y} - y| = 10^6 + 1 + 10^6 + 1 = 2 \cdot 10^6 + 2$$

Theorem 3.3.1. *Let $A \in M(m, n, \mathbb{R})$ and let $B \in M(n, p, \mathbb{R})$, the matrix-matrix product has an error bounded by*

$$\left|\tilde{C}C\right| \leq n \cdot |A| \cdot |B|\mathsf{u} + O(\mathsf{u}^2)$$

3.3.1 Backward stability

Notice that computing the error using this "forward" technique requires a lot of computation also in the case of very simple algorithms. In order to simplify things a little, Wilkinson's trick (from the 60's) may be used. The basic idea of backward stability is to see the computer result as the exact output of an algorithm run on a slightly perturbed input.

Terminology

From now on we will decorate with ˜ floating point numbers, while we will use ˆ for perturbed inputs.

Example 3.3.2. *Let us resort the function of the previous example $f : \mathbb{R}^3 \to \mathbb{R}$ s.t.*

$$f(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

where we can write the computed value as

$$\tilde{y} = a_1 \hat{b_1} + a_2 \hat{b_2} + a_3 \hat{b_3}$$

where

$$\begin{aligned}
\hat{b_1} &= b_1(1+\delta_1)(1+\delta_4)(1+\delta_5) = b_1 + 3ub_1 + o(u), \\
\hat{b_2} &= b_2(1+\delta_2)(1+\delta_4)(1+\delta_5) = b_2 + 3ub_2 + o(u), \\
\hat{b_3} &= b_3(1+\delta_3)(1+\delta_5) = b_3 + 2ub_3 + o(u)
\end{aligned}$$

Let us compute the error on $\hat{b_i}$:

$$\begin{aligned}
\left|\hat{b_i} - b_i\right| &= |b_i(1+\delta_1)(1+\delta_4)(1+\delta_5) - b_i| \\
&= |b_i(\delta_1 + \delta_4 + \delta_5)| \\
&= |b_i| \cdot 3\mathsf{u}
\end{aligned}$$

And the relative error is $\frac{\left\|\hat{\mathbf{b}} - \mathbf{b}\right\|}{\|\mathbf{b}\|} \leq 3u + o(u)$.

Hence, the bound on the result:

$$\frac{\|\tilde{y} - y\|}{\|y\|} \leq \kappa_{rel}(inn.\ prod., b) \frac{\left\|\hat{\mathbf{b}} - \mathbf{b}\right\|}{\|\mathbf{b}\|} \leq \kappa_{rel}(inn.\ prod., b) \cdot 3\mathsf{u}$$

Notice that the theoretical bound is

$$\frac{\|\tilde{y} - y\|}{\|y\|} \leq \kappa_{rel}(inn.\ prod., b) \cdot \mathsf{u}$$

therefore the bound that we found is not bad.

Definition 3.3.1 (Backward stability of an algorithm)**.** *An algorithm that computes $\mathbf{y} = f(\mathbf{x})$ is called **backward stable** if the computed output $\tilde{\mathbf{y}}$ can be written as $\tilde{\mathbf{y}} = f(\hat{\mathbf{x}})$, where $\hat{\mathbf{x}} = \mathbf{x} + O(\mathsf{u}\,\|\mathbf{x}\|)$ (exact function, perturbed input).*

Observation 3.3.1. *In real-life usage, this $O()$ notation often hides polynomial factors in the dimension n. Although this may look an illicit simplification, we observe that these factors are much more harmless than the error that we could make otherwise.*

Theorem 3.3.2. *Backward stable algorithms are as accurate as theoretically possible (given the condition number of a problem), up to some factor that depends only on the dimension (e.g., n, $2n^2 + 18n$, ...).*

Proof.

$$\frac{\|\hat{\mathbf{y}} - \mathbf{y}\|}{\|\mathbf{y}\|} \leq \kappa_{rel}(f, \mathbf{x}) \frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \kappa_{rel}(f, \mathbf{x}) O(\mathsf{u}),$$

while the best attainable accuracy is $\frac{\|\tilde{y} - y\|}{\|y\|} \leq \kappa_{rel}(f, \mathbf{x})\mathsf{u}$. □

We may ask ourselves if it's possible to perturb the input in order to get $\widetilde{y}$ for every possible algorithm and the answer is no. Let us see a counterexample.

Example 3.3.3. *Let us take the functionf* $: \mathbb{R}^3 \to M(3, \mathbb{R})$ *s.t.*

$$f(\mathbf{a}, \mathbf{b}) = \mathbf{a}\mathbf{b}^T = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix} = M$$

Let us consider the machine computation

$$\tilde{M} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \odot \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_1 \odot b_1 & a_1 \odot b_2 & a_1 \odot b_3 \\ a_2 \odot b_1 & a_2 \odot b_2 & a_2 \odot b_3 \\ a_3 \odot b_1 & a_3 \odot b_2 & a_3 \odot b_3 \end{bmatrix}$$

We are looking for $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ such that $\tilde{M} = \hat{\mathbf{a}}^T \cdot \hat{\mathbf{b}}$, but it is impossible to obtain $\tilde{M}$ as $\begin{bmatrix} \hat{a_1} \\ \hat{a_2} \\ \hat{a_3} \end{bmatrix} \cdot \begin{bmatrix} \hat{b_1} & \hat{b_2} & \hat{b_3} \end{bmatrix}$ because the columns of $\tilde{M}$ are all multiples of the same vector.

Backward stability of QR factorization

Theorem 3.3.3. *Let $A \in M(m, n, \mathbb{R})$ and let $Q \in O(m, \mathbb{R})$ be an Householder's reflector. The following holds*

$$Q \odot A = QA + E$$

where $\|E\| \leq \|A\| \cdot O(\mathsf{u})$

Corollary 3.3.4. *As usual, when computing the backward stability, we want to write the error as a perturbation of A:*

$$Q \odot A = QA + E = Q \cdot \underbrace{(A + F)}_{\hat{A}}$$

where $F = Q^{-1}E = Q^T E$ and $\|F\| \leq \underbrace{\|Q\|}_{=1} \cdot \|E\| \leq \|A\| \cdot O(\mathsf{u})$

Do you recall?

A generic step $k \in \mathbb{N}$ of the computation of the QR factorization of a matrix has the following shape:

$$\underbrace{\begin{pmatrix} \mathrm{I}_{k-1,k-1} & \\ & \mathrm{H}_k \end{pmatrix}}_{Q_k} \cdot \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}}_{R_{k-1}} = \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix}}_{R_k}$$

Theorem 3.3.5. *Each step of the QR factorization in backward stable.*

Proof. The computed matrix $\widetilde{R_k}$ is the exact result of applying the transformation to

$$\begin{aligned} \hat{R}_{k+1} &= Q_k \odot R_k \\ &= Q_k R_k + O(\mathsf{u}) \|Q_k\| \|R_k\| \\ &= Q_k R_k + E, \text{ where } \|E\| = O(\mathsf{u}) \|Q_k\| \|R_k\| \\ &= Q_k \cdot (R_k + F), \text{ where } F = {Q_k}^{-1} E \end{aligned}$$

Hence

$$\|F\| \le \left\|{Q_k}^{-1}\right\| \|E\| = O(\mathsf{u}) \underbrace{\|Q_k\|}_{1} \cdot \underbrace{\left\|{Q_k}^{-1}\right\|}_{1} \cdot \underbrace{\|R_k\|}_{\|A\|}$$

$\cdot O(\mathsf{u}) \overset{*}{=} \|A\| \cdot O(\mathsf{u})$, thanks to Theorem 3.3.3. Notice that $\overset{*}{=}$ holds because R_{k-1} is obtained as product between matrix A and orthogonal matrices. □

Theorem 3.3.6 (Backward stability of QR factorization)**.** *QR factorization is* backward stable*: the computed $\tilde{Q}\tilde{R}$ are the exact result of the function $f_{QR}(\hat{A})$, where $\hat{A}$ is such that $\left\|\hat{A} - A\right\| \le O(\mathsf{u}) \cdot \|A\|$.*

Proof. At each step, the computed matrix $\widetilde{R_k}$ is the exact result obtained starting from a perturbed $\hat{A} = A + \underbrace{F_1 + F_2 + \ldots + F_k}_{\text{errors at each step}}$, as stated in Theorem 3.3.5. Provided that $\|F_i\| \le \|A\| \cdot O(\mathsf{u})$ we have the thesis $\|F_1 + F_2 + \ldots + F_k\| = \|A\| \cdot O(\mathsf{u})$. □

Mantra

Orthogonal transformations are the key for stability.

3.3.2 Stability of algorithms for least-squares problems

Let us see how various algorithms to solve LS problems behave with respect to backward stability.

Least squares problem via QR

Step 1: Computing a thin QR (`qr(A, 0);`) $\rightarrow$ backward stable;

Step 2: (`c = Q1'*b;`)$\rightarrow$ backward stable;

Step 3: (`R1 \c;`)$\rightarrow$ backward stable;

Least squares problem via SVD : We never discussed how SVD is computed, hence we state the following

Theorem 3.3.7. *SVD it is a product of orthogonal matrices, hence all the errors are of size $O(\mathtt{u}) \cdot \|A\|$. In particular we get*

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \Big(\kappa_{rel}(f, A) + \kappa_{rel}(f, \mathbf{b})\Big) \cdot O(\mathtt{u})$$

Step 1: Computing a SVD (`svd(A, 0);`) $\rightarrow$ backward stable;

Step 2: (`c = U'*b;`)$\rightarrow$ backward stable;

Step 3: (`d = c ./ diag(S);`)$\rightarrow$ backward stable;

Step 4: (`V*d;`)$\rightarrow$ backward stable;

Least squares problem via Normal Equations : As you might recall from the comparison of the efficiency of algorithms for solving LS problems (Table 2.1), this method does not work properly in the case of "almost singular" matrices A. At this point, we have the theoretical knowledge to fully understand what is happening:

Step 1: `C = A' * A;` $\rightarrow \tilde{C} = (A + E_1)^T \cdot (A + E_1)$

Step 2: `d = A' * b;` $\rightarrow \tilde{\mathbf{d}} = (A + E_2)^T \cdot (\mathbf{b} + \mathbf{e_1})$. This is the first problem, because it is not possible to have the same perturbation of A ($E_1 = E_2$) obtained as backward stability of two different algorithms

Step 3: `x = C \ d;` $\rightarrow$ This implies that the error is $\|C\| = \left\|A^T A\right\| = \left\|A^2\right\| \neq \|A\|$ and this is in contrast with the definition of backward stability, because $\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|}{\|x\|} = \kappa(A)^2\mathtt{u}$

Example 3.3.4. *Let $A \in \mathcal{M}(4, 3, \mathbb{R})$ s.t.*

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 4 \\ 1 & 2 & 3 + 10^{-8} \end{pmatrix}$$

We may observe that A is at distance 10^{-8} from a matrix without full column rank, hence $\kappa(A) \approx 10^8$. On this particular matrix A, the conditioning on QR and SVD is $\kappa(A) \cdot \mathsf{u} \approx 10^8$, while the conditioning of normal equations is $\kappa(A)^2 \cdot \mathsf{u} \approx 10^0$.

3.4 A posteriori checks

So far, we used a “a priori” bound, hence we computed the theoretical stability. Another approach would be to use a “a posteriori” bound, by using the computed value $\tilde{\mathbf{x}}$ for assessing how good the result is.

Definition 3.4.1 (Residual)**.** *Let (P) be a problem and let $\mathbf{x} \in \mathbb{R}^m$ be the solution of that problem. Let $\widetilde{\mathbf{x}}$ be the computed solution of (P). We define* ***residual*** *the value of the problem in $\widetilde{\mathbf{x}}$.*

3.4.1 A posteriori check for linear systems

Definition 3.4.2 (Residual)**.** *Let $A \in \mathcal{M}(m, \mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, and let $\mathbf{x} \in \mathbb{R}^m$ be the solution of $A\mathbf{x} = \mathbf{b}$. Let $\widetilde{\mathbf{x}}$ be the computed solution of $A\mathbf{x} = \mathbf{b}$. We define* ***residual***

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

Theorem 3.4.1. *Let $A \in M(m, \mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, and let $\mathbf{x} \in \mathbb{R}^m$ be the solution of $A\mathbf{x} = \mathbf{b}$ and let $\mathbf{r}$ be the residual of the computed solution $\widetilde{\mathbf{x}}$. The following holds*

$$O(\mathsf{u}) \approx \frac{\|\mathbf{r}\|}{\|A\| \cdot \|\mathbf{x}\|}$$

Proof. Provided that we solve $A\mathbf{x} = \mathbf{b}$ via a backward stable algorithm, obtaining $\tilde{\mathbf{x}}$, then

$$(A + E) \cdot \widetilde{\mathbf{x}} = \mathbf{b} + \mathbf{f}$$

with $\frac{\|E\|}{\|A\|} = O(\mathsf{u})$ and $\frac{\|\mathbf{f}\|}{\|\mathbf{b}\|} = O(\mathsf{u})$.

The norm of the residual is expressed as

$$\Bigg\|\underbrace{A\widetilde{\mathbf{x}} - \mathbf{b}}_{\mathbf{r}}\Bigg\| = \|\mathbf{f} - E\widetilde{\mathbf{x}}\| \leq \|\mathbf{f}\| + \|E\| \cdot \|\widetilde{\mathbf{x}}\| = O(\mathsf{u}) \cdot \|\mathbf{b}\| + O(\mathsf{u}) \cdot \|A\| \cdot \|\widetilde{\mathbf{x}}\|$$

Let us isolate $O(\mathsf{u})$ and we get

$$\begin{aligned} O(\mathsf{u}) &= \frac{\|\mathbf{r}\|}{\|A\| \cdot \|\widetilde{\mathbf{x}}\| + \|\mathbf{b}\|} \\ &\overset{(1)}{\geq} \frac{\|\mathbf{r}\|}{\|A\| \cdot (\|\widetilde{\mathbf{x}}\| + \|\mathbf{x}\|)} \\ &\overset{(2)}{\approx} \frac{\|\mathbf{r}\|}{\|A\| \cdot \|\mathbf{x}\|} \end{aligned}$$

where $\overset{(1)}{\geq}$ holds because $\mathbf{b} = A\mathbf{x} \Rightarrow \|\mathbf{b}\| \leq \|A\| \cdot \|\mathbf{x}\|$ and $\overset{(2)}{\approx}$ holds because $\|\tilde{\mathbf{x}}\|$ and $\|\mathbf{x}\|$ are very close to each-other. □

Theorem 3.4.2. *Let $A \in M(m, \mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, and let $\mathbf{x} \in \mathbb{R}^m$ be the solution of $A\mathbf{x} = \mathbf{b}$.*

For a given $\widetilde{\mathbf{x}}$, the relative error of $\widetilde{\mathbf{x}}$ is bounded by the conditioning of matrix A times the ratio between the norm of the residual and the norm of $\mathbf{b}$. Formally,

$$\frac{\|\mathbf{x} - \widetilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}.$$

This theorem tells us that $\widetilde{x}$ is "close to the solution" apart from a factor which is the conditioning of matrix A.

Proof. Follows from the perturbation results for linear systems. The idea is that $\widetilde{x}$ is the exact solution of the perturbed system

$$A\widetilde{x} \overset{*}{=} b + \mathbf{r} = \widetilde{b}$$

Where $\overset{*}{=}$ follows from the definition of r and $\frac{\|\widetilde{b} - b\|}{\|b\|} = \frac{\|r\|}{\|b\|}$.
A relative perturbation of size $\frac{\mathbf{r}}{b}$ is amplified by $\kappa(A)$. □

It's important to notice that also computing $A \odot x \ominus b$ is an approximated operation. We choose to simplify things and ignore this error.

3.4.2 A posteriori check for Least Squares Problems

We cannot use the same approach used with linear system "a posteriori" check, because for $A \in M(m, n, \mathbb{R})$, with $m \gg n$ and full column rank and $\mathbf{b} \in \mathbb{R}^n$ the quantity $\|A\mathbf{x} - \mathbf{b}\|$ is not small at all, indeed it could be as large as $\mathbf{b}$, as shown in Figure 3.7.

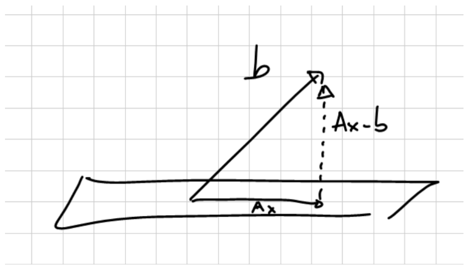


FIGURE 3.7: A can be as large as $\mathbf{b}$ if $\mathbf{b}$ is perfectly orthogonal to the hyperplane identified by A.

Definition 3.4.3 (Backward Error)**.** *We term **backward error** the minimal perturbation of inputs that produces an approximated output.*

Scrivere meglio

Observation 3.4.1. *If you solve LSP via QR* $\|Ax-b\| = \left\| \begin{pmatrix} R_1 x - {Q_1}^T b \\ -{Q_2}^T b \end{pmatrix} \right\|$. *We said that the entries in the second block are fixed irrespective of* x*, but we could make the entries in the first block zero, by choosing* $x = R_1^{-1} Q^T b$*. This information let us infer something about the values of the vectors in Figure 3.7, in particular the minimum of the value that we can get is* $\|{Q_2}^T b\| = \|Ax-b\|$.

With some algebra we may also check that $\|{Q_1}^T b\| = \|Ax\|$.

Since this is a minimum problem, we know that the gradient of the function is small near the optimum value, hence

$$f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x}-\mathbf{b}\|^2 = \frac{1}{2}(A\mathbf{x}-\mathbf{b})^T \cdot (A\mathbf{x}-\mathbf{b}) = \frac{1}{2}(\mathbf{x}^T A^T A\mathbf{x} - \mathbf{b}^T A\mathbf{x} - \mathbf{x}^T A^T\mathbf{b} + \mathbf{b}^T\mathbf{b})$$

$$\nabla f(\mathbf{x}) = A^T A\mathbf{x} - A^T\mathbf{b}$$

Let us define the residual $\mathbf{r} = \nabla f(\mathbf{x})$, therefore $\widetilde{\mathbf{x}}$ solves

$$A^T A\widetilde{\mathbf{x}} - A^T\mathbf{b} - \mathbf{r} = 0 \quad (P_2)$$

hence

$$\frac{\|\widetilde{\mathbf{x}}-\mathbf{x}\|}{\mathbf{x}} \leq \kappa_{\text{rel}}(P_2, A^T\mathbf{b}) \cdot \frac{\|A^T\mathbf{b} + \mathbf{r} - A^T\mathbf{b}\|}{\|A^T\mathbf{b}\|} = \underbrace{\kappa(A^T A)}_{\|A^T A\| \cdot \left\|(A^T A)^{-1}\right\| = (\sigma_1/\sigma_2)^2} \cdot \frac{\|\mathbf{r}\|}{\|A^T\mathbf{b}\|}$$

notice that $\left(\frac{\sigma_1}{\sigma_n}\right)^2$ could be incredibly large.

Theorem 3.4.3. $\frac{\|\widetilde{x}-x\|}{x} \leq (\kappa(A))^2 \cdot \frac{\|A^T Ax - A^T b\|}{\|A^T b\|}$. *Although we might have wanted to have the condition number of the problem, instead of the condition number of A and this could lead to underestimating the error.*

Another idea could be using as error the first entry of the vector obtained via QR (namely ${R_1}^T x - {Q_1}^T b$), by imposing $R_1 x = {Q_1}^T b$

We may observe that this is a truly backward stable measure:

given $r = \|{R_1}^T \widetilde{x} - {Q_1}^T b\|$, there exists $\widetilde{b}$ with $\left\|\widetilde{b} - b\right\| = \|r\|$ such that $\widetilde{x}$ is the exact solution of $\min \left\|Ax - \widetilde{b}\right\|$.

We have proved the following

Fact 3.4.4. *Let* $A \in M(m,n,\mathbb{R})$*, with* $m \gg n$ *and full column rank and* $\mathbf{b} \in \mathbb{R}^n$*. If we denote* $\mathbf{x} \in \mathbb{R}^n$ *the solution of* $\|A\mathbf{x}-\mathbf{b}\|$ *and with* $\widetilde{\mathbf{x}} \in \mathbb{R}^n$ *the computed value, the following holds*

$$\frac{\|\widetilde{\mathbf{x}} - x\|}{\|x\|} \leq \frac{\kappa(A)}{\cos\theta} \cdot \frac{\|\mathbf{r_1}\|}{\|\mathbf{b}\|}$$

Proof. Let us apply the condition number bound on

- modified input: $\mathbf{b} \to \mathbf{b} + Q_1\mathbf{r_1}$
- solution: $\mathbf{x} \to \widetilde{\mathbf{x}}$

$$\begin{aligned}\frac{\|\widetilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} &\leq \kappa_{\text{rel}}(LS, \mathbf{b}) \cdot \frac{\left\|\widetilde{\mathbf{b}} - \mathbf{b}\right\|}{\|\mathbf{b}\|} \\ &= \kappa_{\text{rel}}(LS, \mathbf{b}) \cdot \frac{\|\mathbf{b}\| + Q_1\mathbf{r_1} - \mathbf{b}}{\|\mathbf{b}\|} \\ &\overset{(1)}{\leq} \frac{\kappa(A)}{\cos(\theta)} \cdot \frac{\overbrace{\|Q_1\mathbf{r_1}\|}^{\|\mathbf{r_1}\|}}{\|\mathbf{b}\|}\end{aligned}$$

where $\overset{(1)}{\leq}$ follows from Equation (3.1.3) □

Theorem 3.4.5. *Let $A = Q_1R_1$ be a thin QR factorization. Let $\mathbf{r}_1 = Q_1^T(A\widetilde{\mathbf{x}} - \mathbf{b})$. Then, $\widetilde{\mathbf{x}}$ is the exact solution of the LS problem*

$$\min \|A\mathbf{x} - (\mathbf{b} + Q_1\mathbf{r}_1)\|,$$

so the backward error of $\widetilde{\mathbf{x}}$ is $\|Q_1\mathbf{r}_1\| = \|\mathbf{r}_1\|$.

Proof. Idea: replay the solution of a LS problem with QR factorization, and use ${Q_1}^T T Q_1 = I$. You will get in the first block $R_1x = {Q_1}^T b + \mathbf{r}_1$, i.e., ${Q_1}^T(Ax - b) = \mathbf{r}_1$, which is verified by $\widetilde{x}$.

$$\begin{aligned}\|A\mathbf{y} - (\mathbf{b} + Q_1\mathbf{r_1})\| &= \left\|Q^T \cdot (A\mathbf{y} - \mathbf{b} - Q_1\mathbf{r_1})\right\| \\ &= \left\|\begin{pmatrix}{Q_1}^T \\ {Q_2}^T\end{pmatrix} \cdot (A\mathbf{y} - \mathbf{b} - Q_1\mathbf{r_1})\right\| \\ &= \left\|\begin{pmatrix}R_1\mathbf{y} \\ 0\end{pmatrix} - \begin{pmatrix}{Q_1}^T\mathbf{b} \\ {Q_2}^T\mathbf{b}\end{pmatrix} - \begin{pmatrix}\mathbf{r_1} \\ \mathbf{0}\end{pmatrix}\right\| \\ &= \left\|\begin{pmatrix}R_1\mathbf{y} - {Q_1}^T\mathbf{b} - \mathbf{r_1} \\ -{Q_2}^T\mathbf{b}\end{pmatrix}\right\|\end{aligned}$$

where the first component of the last vector is 0 whenever we choose $\widetilde{\mathbf{x}} = \mathbf{y}$ and the second component does not depend on $\mathbf{y}$. □

Chapter 4

Linear Systems

4.1 LS: the task

In this lecture we address the problem of solving linear systems exactly.

Do you recall?

Solving a linear system means for any given $A \in M(m, \mathbb{R})$, $\mathbf{b} \in \mathbb{R}^m$, find $\mathbf{x} \in \mathbb{R}^n$ such that

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

In this course we provide the following four ways of solving a linear system:

- Gaussian elimination;
- LU factorization;
- QR factorization;
- Cholesky factorization (specialized method for positive definite matrices) Idea: A^TA can be written as $A^TA = R^TR$, where R is a square, upper triangular matrix.

Someone could observe that this subject has already been studied in the numerical linear algebra course, but we are interested in computing the solution to this problem quickly when the dimensions are *large* and the matrix A is *sparse*.
Since the complexity of Gauss method is cubic, this algorithm is unfeasible for large inputs.

Let us first discuss some real life examples, where the matrices are large and sparse.

Local function on graphs: A **local function on graphs** is a function that depend on few nearby vertices. This kind of functions lead to a sparse adjacency matrix A, as can be observed in Figure 4.1;

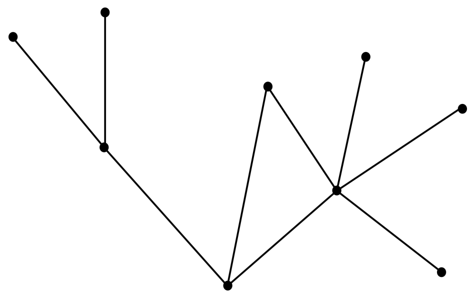


FIGURE 4.1: A local function on graph.

IMAGES: Take an $m \times m$ image and blur it (each pixel is obtained as the average of its neighbours). $T : \mathcal{M}(m, \mathbb{R}) \to \mathcal{M}(m, \mathbb{R})$ such that $T(A)_{ij} = \frac{1}{9} \cdot (A_{i-1j} + A_{i-1j-1} + A_{i-1j+1} + A_{ij} + A_{ij-1} + A_{ij+1} + A_{i+1j} + A_{i+1j-1} + A_{i+1j+1})$. T may be written as a matrix that maps all the m images to a set of m blurred images and has the following shape $T \in \mathcal{M}(m^2, \mathbb{R})$ such that the (i, j)-th row of T has exactly 9 entries with value $\frac{1}{9}$ and all the others are 0. The non zero entries correspond to A_{i-1j}, ..., A_{i+1j+1};

KKT SYSTEMS constrained optimization;

ENGINEERING PROBLEM: To check stability of a bridge, it gets split into small cells. It can be proven that the stress on each of these cells corresponds to the force applied by the neighbours, as shown in Figure 4.2. In the end, this local phenomenon may be represented by a sparse matrix.

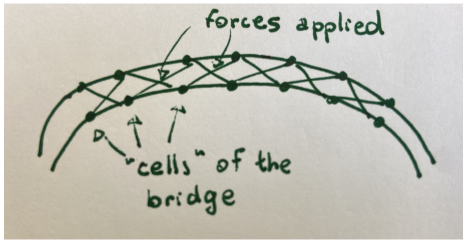


FIGURE 4.2: Graphic idea of a bridge partitioned into small blocks

Something on Matlab . . .

Matlab allows to display sparse matrices in a visual-friendly way, via the command `spy(A)`. Other useful commands are the following

- `nnz(A)` display the non-zero entries of A
- The density of a matrix A can be computed as `density = nnz(A) / (size(A) * size(A))`

Moreover, Matlab efficiently stores sparse matrices storing only the couple (pos, value) of non-zero entries (the so-called *coordinate format*). In order to convert a sparse matrix into a regular matrix there is the command `full(A)`.

4.2 QR Factorization

Thanks to QR factorization, the matrix A can be written as $A = QR$, where $Q \in O(m, \mathbb{R})$ and $R \in T(m, \mathbb{R})$. In this case $\mathbf{x}$ can be computed as

$$\mathbf{x} = A^{-1}\mathbf{b} = R^{-1}Q^{-1}\mathbf{b}$$

and the steps are the following

1. compute $A = QR$, complexity: $4/3m^3 + O(m^2)$
2. compute $c = Q^{-1}\mathbf{b} = Q^T\mathbf{b}$, complexity: $O(m^2)$
3. compute $\mathbf{x} = R^{-1}\mathbf{c}$ via back-substitution, complexity: $O(m^2)$

Hence the total cost is $4/3m^3 + O(m^2)$.

4.3 SVD Factorization

The same trick we used for QR factorization holds for SVD.

4.4 LU Factorization

Gaussian elimination can be seen as a factorization: $A = LU$. The intuition is to proceed iteratively, multiplying each time for a new matrix, just like QR factorization.

Since the idea of Gauss elimination is to add multiples of row 1 to all the rows from 2 to m to kill off $A(2 : end, 1)$ we have that:

STEP 1:

$$\underbrace{\begin{pmatrix} 1 & & & & \\ -a_2 & 1 & & & \\ -a_3 & & 1 & & \\ -a_4 & & & 1 & \\ -a_5 & & & & 1 \end{pmatrix}}_{\text{matrix } L_1} \underbrace{\begin{pmatrix} \textcircled{\times} & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}}_{\text{matrix } A} = \underbrace{\begin{pmatrix} \textcircled{\times} & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}}_{\text{matrix } A_1}$$

Where the $\textcircled{\times}$ is called **pivot** and L_1 is such that

$$a_k = (L_1)_{k1} = \frac{(A)_{k1}}{(A)_{11}}, \quad k = 2, 3, \dots, m.$$

STEP 2: let us multiply A_1 (obtained at the previous step) by a matrix L_2 that has an "identity frame" and inside does the same that L_1 was doing before.

$$\underbrace{\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & -b_3 & 1 & & \\ & -b_4 & & 1 & \\ & -b_5 & & & 1 \end{pmatrix}}_{\text{matrix } L_2} \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \textcircled{\times} & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}}_{\text{matrix } A_1} = \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \textcircled{\times} & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}}_{\text{matrix } A_2}$$

$$b_k = (L_2)_{k2} = \frac{(A_1)_{k2}}{(A_1)_{22}}, \quad k = 3, \dots, m.$$

STEP 3: let us go on and

$$\underbrace{\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & -c_4 & 1 & \\ & & -c_5 & & 1 \end{pmatrix}}_{\text{matrix } L_3} \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \textcircled{\times} & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}}_{\text{matrix } A_2} = \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \textcircled{\times} & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix}}_{\text{matrix } A_3}$$

$$c_k = (L_3)_{k3} = \frac{(A_2)_{k3}}{(A_2)_{33}}, \quad k = 4, \dots, m.$$

STEP 4: one more operation

$$\underbrace{\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & -d_5 & 1 \end{pmatrix}}_{\text{matrix } L_4} \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \textcircled{\times} & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix}}_{\text{matrix } A_3} = \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \textcircled{\times} & \times \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}}_{\text{matrix } A_4}$$

$$d_k = (L_4)_{k4} = \frac{(A_3)_{k4}}{(A_3)_{44}}, \quad k = 5, \dots, m.$$

In the generic case we have $L_{m-1} \cdot L_{m-2} \cdot \dots \cdot L_1 \cdot A = U$, where U is upper triangular, or $A = \underbrace{L_1^{-1} \cdot L_2^{-1} \cdot \dots \cdot L_{m-1}^{-1}}_{=L} \cdot U$, with U upper triangular and L lower triangular.

Theorem 4.4.1. *Let $A \in \mathcal{M}(m, \mathbb{R})$ such that we do not encounter zero pivots in the algorithm. A admits a factorization $A = LU$, where $L \in T(m, \mathbb{R})$ is lower triangular with ones on its diagonal, and $U \in T(m, \mathbb{R})$ is upper triangular.*

Observation 4.4.1 (Stroke of luck)**.** *The product of the L_i^{-1}'s (denoted as L) can be computed for free, since the following holds:*

$$\begin{bmatrix} 1 & & & & \\ -a_2 & 1 & & & \\ -a_3 & & 1 & & \\ -a_4 & & & 1 & \\ -a_5 & & & & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -b_3 & 1 & & \\ & -b_4 & & 1 & \\ & -b_5 & & & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & -c_4 & 1 & \\ & & -c_5 & & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & -d_5 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & \\ a_2 & 1 & & & \\ a_3 & b_3 & 1 & & \\ a_4 & b_4 & c_4 & 1 & \\ a_5 & b_5 & c_5 & d_5 & 1 \end{bmatrix}$$

Algorithm 4.4.1 LU factorization, Matlab implementation.

```
1  function [L, U] = lu_factorization(A)
2    m = size(A, 1);
3    L = eye(m);
4    U = A;
5    for k = 1 : m - 1
6      % compute "multipliers"
7      L(k+1:end, k) = U(k+1:end, k) / U(k, k);
8      % update U
9      U(k+1:end, k) = 0;
10     U(k+1:end, k+1:end) = U(k+1:end, k+1:end) ...
11         - L(k+1:end, k) * U(k, k+1:end);
12   end
```

Fact 4.4.2. *The computational complexity of LU factorization is*

$$\frac{2}{3}m^3 + O(m^2)$$

Proof. The computational complexity of this algorithm is concentrated at lines 10, 11 where at the k-th iteration we need to perform 2 operations for each entry of a $(m-k)\times(m-k)$ sub-matrix of U

$$\begin{pmatrix} \times & & & & \\ 0 & \times & & & \\ 0 & 0 & \times & \dots & \times \\ \vdots & \vdots & \vdots & (m-k) & \vdots \\ 0 & 0 & \times & \dots & \times \end{pmatrix}$$

The total complexity is $2(m-1)^2+2(m-2)^2+\ldots+2^2+1 \approx 2\Big(\frac{m^3}{3}+O(m^2)\Big)$. $\square$

An attentive reader may notice that the cost of LU factorization is half as much as QR factorization.

The algorithm for computing a solution of the linear system would then be

- $\mathbf{c} = L^{-1}\mathbf{b}$, solved through forward substitution
- $\mathbf{x} = U^{-1}\mathbf{c}$, solved through backward substitution

Something on Matlab ...

Implementation of \ in Matlab: We consider important to remark how the operator \ is implemented in Matlab and uses an “automatic algorithm”:

- if A is triangular, back(for)ward substitution is used directly
- otherwise LU factorization is used.

Notice that there is not any check on the orthogonality of the matrix. It seems odd, because in that case $A^{-1} = A^T$, and hence it would be easy to solve the system. The issue is that in order to find out that a matrix is orthogonal we would need to compute A^TA, which is too costly.

4.4.1 Stability of LU

A downside of this approach is that it’s not numerically stable. The intuition is that the conditioning is bad whenever the matrix A has a very small pivot.

Let us see an example:

Example 4.4.1. *Let $A \in M(2,\mathbb{R})$ such that*

$$A = \begin{bmatrix} 10^{-30} & 1 \\ 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ 10^{30} & 1 \end{bmatrix}}_{\text{matrix } L} \underbrace{\begin{bmatrix} 10^{-30} & 1 \\ 0 & 1-10^{30} \end{bmatrix}}_{\text{matrix } U}$$

In this case the LU factorization of A produces two triangular matrices L, U with norm much larger than $\|A\|$.

Partial pivoting

Luckily, we can circumvent this issue multiplying L_is by some permutation matrices (which swap rows in order to have a s pivot the largest value on that column), as follows

$$L_{m-1}P_{m-1}\dots L_2P_2L_1P_1A = U$$

where a permutation matrix P_i has the following shape

$$\begin{pmatrix} \mathrm{I} & & 0 & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & \mathrm{I} \end{pmatrix}$$

Observation 4.4.2 (Stroke of luck 2)**.** *Thanks to another "stroke of luck" we can reorder those factors:*

$$L_{m-1}P_{m-1}\dots L_2P_2L_1P_1 = \widehat{L}_{m-1}\widehat{L}_{m-2}\dots\widehat{L}_1P_{m-1}P_{m-2}\dots P_1$$

where $\widehat{L}_i$ is equal to a modified version of L_i where the entries of the active column are swapped just like P_i.

Fact 4.4.3. *LU factorization is not backward stable.*

Proof.

$$\begin{aligned}\hat{U}_{k+1} &= L_k \odot U_k\\ &= L_kU_k + E \text{ where } \|E\| = O(\mathsf{u})\cdot\|L_k\|\cdot\|U_k\|\\ &= L_k(U_k+F) \text{ where } F = {L_k}^{-1}E\end{aligned}$$

Hence

$$\|F\| \le \left\|{L_1}^{-1}\right\|\cdot\|E\| = O(\mathsf{u})\cdot\|L_k\|\cdot\left\|{L_k}^{-1}\right\|\cdot\|U_k\|$$

□

We can now introduce the following

Theorem 4.4.4. *Let $A \in \mathcal{M}(m,\mathbb{R})$. A admits a factorization $A = PLU$, where P is a permutation matrix, $L \in T(m,\mathbb{R})$ is lower triangular with ones on its diagonal, and $U \in T(m,\mathbb{R})$ is upper triangular.*

It goes without saying that there is an overhead to performing partial pivoting during LU factorization and its cost is $O(m^2)$.
Notice that P is orthogonal ($P^{-1} = P^T$).
In the pivoting case the algorithm is

- $\mathbf{c} = P\mathbf{b}$
- $\mathbf{d} = L^{-1}\mathbf{c}$, through forward substitution

- $\mathbf{x} = U^{-1}\mathbf{d}$, through back substitution

Notice that LU factorization cannot be used to solve Least Squares Problem, because $\|A\mathbf{x} - \mathbf{b}\| \neq \left\|L^{-1}(A\mathbf{x} - \mathbf{b})\right\|$.

Although the elements of the active column of each L_i are bounded by 1 (because $(L_i)_{ki} = -A_{ki}/A_{ii}$ where A_{ii} is chosen as the largest element of the i-th column of A) there is a worst case scenario in which $\|U\| \approx 2^m \cdot \|A\|$. Luckily, this worst case scenario is not statistically relevant in practice.

4.4.2 Gaussian elimination on sparse matrices

Given a sparse matrix

$$A = \begin{pmatrix}
\times & \times & & & \times & \times & & \times & \\
 & \times & & \times & & \times & \times & & \\
\times & & \times & \times & \times & & \times & & \\
\times & & & \times & & & \times & & \\
 & \times & & & & \times & & \times & \times \\
\times & & & \times & & \times & \times & & \\
\times & \times & & & & & \times & & \times \\
\times & \times & \times & \times & & & & & \\
\times & & \times & & & \times & & & \times
\end{pmatrix}$$

Gaussian elimination causes some fill-in, due to the sum of a multiple of the first row, which has non zero entries in different positions:

$$\underbrace{\begin{pmatrix}
\times & \times & & & \times & \times & & \times & \\
 & \times & & \times & & \times & \times & & \\
\times & & \times & \times & \times & & \times & & \\
\times & & & \times & & & \times & & \\
 & \times & & & & \times & & \times & \times \\
\times & & & \times & & \times & \times & & \\
\times & \times & & & & & \times & & \times \\
\times & \times & \times & \times & & & & & \\
\times & & \times & & & \times & & & \times
\end{pmatrix}}_{\text{matrix } A} \to \underbrace{\begin{pmatrix}
\times & \times & & & \times & \times & & \times & \\
 & \times & & \times & & \times & \times & & \\
{\color{red}0} & {\color{red}\times} & \times & \times & \times & {\color{red}\times} & \times & {\color{red}\times} & \\
{\color{red}0} & {\color{red}\times} & & \times & {\color{red}\times} & {\color{red}\times} & \times & {\color{red}\times} & \\
 & \times & & & & \times & & \times & \times \\
{\color{red}0} & {\color{red}\times} & & \times & {\color{red}\times} & \times & \times & {\color{red}\times} & \\
{\color{red}0} & \times & & & {\color{red}\times} & {\color{red}\times} & \times & {\color{red}\times} & \times \\
{\color{red}0} & \times & \times & \times & {\color{red}\times} & {\color{red}\times} & & {\color{red}\times} & \\
{\color{red}0} & {\color{red}\times} & \times & & {\color{red}\times} & \times & & {\color{red}\times} & \times
\end{pmatrix}}_{\text{matrix } A_1}$$

We may observe that the computational complexity of sparse LU is linear in the number of non zero entries of the final matrix, obtained by the algorithm, which is possibly much larger than the number of non zeros in A.

How to circumvent this problem? At each step we may use as pivot row the most sparse one. This computation may be done in a more sofisticate way, considering the "relative" position of non zeros between couples of rows.

Because of this trade-off the choice is made in relation to the needs of the implementation. We won't study any algorithm that deals with sparse matrices, since they are very complicated and make use of heuristics.

There are some lucky cases in which the fill-in is almost none, for example a matrix that only has 5 diagonals which entries are different from 0 (called

tridiagonal). In this particular case L is tridiagonal and lower triangular and U is tridiagonal and upper triangular, as shown below.

$$A = \begin{pmatrix} \times & \times & \times & 0 & 0 & 0 & \cdots & 0 \\ \times & \times & \times & \times & 0 & 0 & \cdots & 0 \\ \times & \times & \times & \times & \times & 0 & \cdots & 0 \\ 0 & \times & \times & \times & \times & \times & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & \times & \times & \times & \times \\ 0 & 0 & 0 & \cdots & \times & \times & \times & \times \\ 0 & 0 & 0 & \cdots & 0 & \times & \times & \times \end{pmatrix}$$

$$L = \begin{pmatrix} \times & & & & & & \\ \times & \times & & & & & \\ \times & \times & \times & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \times & \times & \times & & \\ & & & \times & \times & \times & \\ & & & & \times & \times & \times \end{pmatrix} \quad U = \begin{pmatrix} \times & \times & \times & & & & \\ & \times & \times & \times & & & \\ & & \times & \times & \times & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \times & \times & \times \\ & & & & & \times & \times & \times \\ & & & & & & \times & \times \end{pmatrix}$$

Observation 4.4.3. *We should remark that if we are interested in high-performance computing we need to pay attention to the blocking, because we go from vector-vector operation to matrix-matrix operation and some of these operations may be performed more efficiently. Parallel/multithreaded implementations are available by means of parallel libraries for Matlab.*

4.4.3 Gaussian elimination on symmetric matrices

Do you recall?

In Gaussian elimination we had A and we multiplied it by L_1 in order to get a new matrix with a big chunk of 0s in the first column

$$\underbrace{\begin{pmatrix} 1 & & & & \\ \times & 1 & & & \\ \times & & 1 & & \\ \times & & & 1 & \\ \times & & & & 1 \end{pmatrix}}_{\text{matrix } L_1} \underbrace{\begin{pmatrix} \textcircled{\times} & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}}_{\text{matrix } A} = \underbrace{\begin{pmatrix} \textcircled{\times} & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}}_{\text{matrix } A_1}$$

Let us consider an upgrade of Gaussian elimination in the case of $A \in S(m, \mathbb{R})$.

Let us see what happens if we multiply $L_1 A$ on the right by the transpose of L_1:

$$\left(L_1 A {L_1}^T\right)^T = \left({L_1}^T\right)^T A^T {L_1}^T = L_1 A {L_1}^T \in S(m, \mathbb{R})$$

Rimpicciolire le matrici in larghezza per farcele entrare

Step 1:

$$\underbrace{\begin{pmatrix} 1 & & & & \\ -a_1 & 1 & & & \\ -a_2 & & 1 & & \\ -a_3 & & & 1 & \\ -a_4 & & & & 1 \end{pmatrix}}_{\text{matrix } L_1} \underbrace{\begin{pmatrix} \otimes & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}}_{\text{matrix } A} \underbrace{\begin{pmatrix} 1 & -a_1 & -a_2 & -a_3 & -a_4 \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}}_{\text{matrix } {L_1}^T} = \underbrace{\begin{pmatrix} \otimes & 0 & 0 & 0 \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{pmatrix}}_{\text{matrix } A_1}$$

Step 2:

$$\underbrace{\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & -b_1 & 1 & & \\ & -b_2 & & 1 & \\ & -b_3 & & & 1 \end{pmatrix}}_{\text{matrix } L_2} \underbrace{\begin{pmatrix} \times & 0 & 0 & 0 & 0 \\ 0 & \otimes & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}}_{\text{matrix} A_1} \underbrace{\begin{pmatrix} 1 & & & & \\ & 1 & -b_1 & -b_2 & -b_3 \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}}_{\text{matrix } {L_2}^T} = \underbrace{\begin{pmatrix} \times & 0 & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & \otimes & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}}_{\text{matrix } A_2}$$

Step m:

$$L_{m-1} \cdot L_{m-2} \cdot \ldots \cdot L_1 \cdot A \cdot {L_1}^T \cdot \ldots \cdot L_{m-2}^T \cdot L_{m-1}^T = D,$$

where D is diagonal, or

$$A = \underbrace{L_1 \cdot L_2 \cdot \ldots \cdot L_{m-1}}_{L} \cdot D \cdot \underbrace{L_{m-1}^T \cdot \ldots \cdot L_2^T \cdot L_1^T}_{L^T} = L \cdot D \cdot L^T.$$

Observation 4.4.4 (Stroke of luck)**.** *Notice that the stroke of luck of Observation 4.4.1 holds in this case too, hence we pay nothing to compute both matrices L and L^T.*

$$\begin{bmatrix} 1 & & & & \\ -a_1 & 1 & & & \\ -a_2 & & 1 & & \\ -a_3 & & & 1 & \\ -a_4 & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -b_1 & 1 & & \\ & -b_2 & & 1 & \\ & -b_3 & & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & -c_1 & 1 & \\ & & -c_2 & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & -d_1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ a_1 & 1 & & & \\ a_2 & b_1 & 1 & & \\ a_3 & b_2 & c_1 & 1 & \\ a_4 & b_3 & c_2 & d_1 & 1 \end{bmatrix}$$

Theorem 4.4.5 (Symmetric Gaussian elimination)**.** *Let $A \in S(m, \mathbb{R})$ such that during Gaussian elimination we don't encounter any* 0 *pivot. A admits a factorization $A = LDL^T$, where $L \in T(m, \mathbb{R})$ is lower triangular with ones on its diagonal, and $D \in D(m, \mathbb{R})$.*

A Matlab implementation of symmetric Gaussian elimination is shown in Algorithm 4.4.2.

Algorithm 4.4.2 Symmetric Gaussian factorization, Matlab implementation.

```
function [L, D] = ldl_factorization(A)
m = size(A, 1);
L = eye(m);
D = zeros(m);
for k = 1:m-1
D(k, k) = A(k, k);
L(k+1:end, k) = A(k+1:end, k) / A(k, k);
A(k+1:end, k+1:end) = A(k+1:end, k+1:end) ...
- L(k+1:end, k) * A(k, k+1:end);
end
D(m, m) = A(m, m);
```

Notice that it is possible to make an optimization of this algorithm: since A is supposed to be symmetric, we only need to update the lower triangular part of A, since the rest is mirrored by symmetry, hence the computational complexity is half that of Gaussian elimination, namely $\frac{1}{3}m^3 + O(m^2)$.

This algorithm is not backward stable, exactly like the one on non symmetric matrices. Pivoting may be performed in order to improve stability. It comes without saying that the row swap should be done consistently on the columns to preserve symmetry. This is performed by permuting rows and columns of A, that means swapping elements on the diagonal. In particular, at each step, we work on a 2×2 block diagonal matrix, hence

$$D = \begin{pmatrix} 0 & \times & & & \\ \times & 0 & & & \huge 0 \\ & & \ddots & & \\ \huge 0 & & & 0 & \times \\ & & & \times & 0 \end{pmatrix}$$

Of course there are some matrices (like the ones with all 0s on the diagonal) that cannot be "pivoted". There are workarounds, as splitting matrices into blocks. As an example, Matlab's `[L, D, P] = ldl(A)` produces matrices such that $P^T AP = LDL^T$, where D is not diagonal, but it has 2×2 blocks on the diagonal.

Do you recall?

A matrix $A \in M(m, \mathbb{R})$ is said to be **positive definite** if all its eigenvalues are strictly positive. Formally, A is positive definite if

$$\forall \mathbf{z} \neq 0 \in \mathbb{R}^m \;\; \mathbf{z}^T A \mathbf{z} > \mathbf{0}$$

One can prove that on positive definite matrices we do not need to divide by 0, while performing symmetric Gaussian elimination and this is proved by the following

Lemma 4.4.6. *In the context of positive definite matrices the following holds:*

1. *Let A be a symmetric matrix. A is positive definite if and only if MAM^T is positive definite, for some invertible $M \in GL(m, \mathbb{R})$. Formally,*

$$\forall A \in S(m, \mathbb{R}) \text{ s.t. } A \succ 0 \Leftrightarrow \exists M \in GL(m, \mathbb{R}) \text{ s.t. } MAM^T \succ 0$$

2. *Let A a symmetric positive definite matrix such that $A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right]$, then A_{11} and A_{22} are positive definite too. Formally,*

$$\forall A \in S(m, \mathbb{R}) \text{ s.t. } A \succ 0 \text{ and } A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right] \Rightarrow A_{11} \succ 0 \text{ and } A_{22} \succ 0$$

Proof.

1.

   $\Rightarrow$) $A \in S(m, \mathbb{R})$ and $A \succ 0 \Longrightarrow MAM^T \in S(m, \mathbb{R})$ and $MAM^T \succ 0$.
   Take $z \in \mathbb{R}^m$, $\mathbf{z} \neq \mathbf{0}$ $\mathbf{z}^T MAM^T \mathbf{z} = \mathbf{y}^T A \mathbf{y} > 0$, where we performed a variable change $\mathbf{y} = M^T \mathbf{z}$. Notice that $\mathbf{y} \neq \mathbf{0}$ because M is invertible (and $ker(M) = \{\mathbf{0}\}$). The symmetry of the matrix MAM^T follows from $(MAM^T)^T = M^{T^T} A^T M^T = MAM^T$;

   $\Leftarrow$) $MAM^T \in S(m, \mathbb{R})$ and $MAM^T \succ 0 \Longrightarrow A \in S(m, \mathbb{R})$ and $A \succ 0$.
   This proof follows from the previous arrow, where the substitution is $\mathbf{z} = M^{-1} \mathbf{y}$.

2. $A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right]$ positive definite $\Longrightarrow A_{11}$ and A_{22} are positive definite too.
   Since A is positive definite, its scalar product is greater than zero with all the vectors in $\mathbb{R}^m$.

   A_{11}) Let us take $\mathbf{z} = \left[\begin{smallmatrix} \mathbf{z}_1 \\ 0 \end{smallmatrix}\right]$.
   $\left[\begin{smallmatrix} \mathbf{z}_1{}^T & 0 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} \mathbf{z}_1 \\ 0 \end{smallmatrix}\right] = \mathbf{z_1}^T A_{11} \mathbf{z_1} > 0$, $\forall \mathbf{z}_1 \in \mathbb{R}^{sizeof A_{11}}$

   A_{22}) Let us take $\mathbf{z} = \left[\begin{smallmatrix} 0 \\ \mathbf{z}_2 \end{smallmatrix}\right]$.
   $\left[\begin{smallmatrix} 0 & \mathbf{z}_2{}^T \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 0 \\ \mathbf{z}_2 \end{smallmatrix}\right] = \mathbf{z_2}^T A_{22} \mathbf{z_2} > 0$, $\forall \mathbf{z}_2 \in \mathbb{R}^{sizeof A_{22}}$

□

Corollary 4.4.7. *Let $A \in S(n, \mathbb{R})$ such that A is positive definite. When computing the LDL^T factorization of A, at each step we have $D_{kk} > 0$, hence we need no pivoting technique.*

Proof. From the first point of Lemma 4.4.6 we have that, since $A \succ 0$, $L_1 A {L_1}^T$ is positive definite. Thanks to the second point of the same lemma we have $L_1 A {L_1}^T = \begin{pmatrix} \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}$ and so the first and the second diagonal blocks are positive definite ($D_{11} > 0$ and $D_{22} \succ 0$).

This reasoning can be done recursively, hence the diagonal of $L_1 A {L_1}^T$ is positive. □

Theorem 4.4.8. *The cost of LDL factorization with pivoting is $\frac{1}{3}m^3 + O(m^2)$ (half of LU).*

4.4.4 Cholesky Factorization

Definition 4.4.1 (Cholesky Factorization)**.** *Let $A \in S(m, \mathbb{R})$ such that $A \succ 0$. It can be factored as*

$$A = R^T R$$

where $R \in T(m, \mathbb{R})$ is upper triangular and it as positive elements on the diagonal.

If we rewrite the diagonal matrix of the Gaussian elimination D as product of $D^{1/2}$ times itself we get:

$$\begin{aligned} D &= \begin{pmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{mm} \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} \sqrt{d_{11}} & & & \\ & \sqrt{d_{22}} & & \\ & & \ddots & \\ & & & \sqrt{d_{mm}} \end{pmatrix}}_{\text{matrix } D^{1/2}} \cdot \underbrace{\begin{pmatrix} \sqrt{d_{11}} & & & \\ & \sqrt{d_{22}} & & \\ & & \ddots & \\ & & & \sqrt{d_{mm}} \end{pmatrix}}_{\text{matrix } D^{1/2}} \end{aligned}$$

Therefore the LDL factorization may be rewritten as follows

$$A = LDL^T = LD^{1/2}({D^{1/2}}^T L^T) = CC^T,$$

where $D^{1/2} = \mathrm{diag}(D_{11}^{1/2}, D_{22}^{1/2}, \ldots, D_{mm}^{1/2})$, and C is lower triangular (but not anymore with ones on the diagonal).

Notice that it is guaranteed that the square root of elements on the diagonal exists because they are all positive, as proved by Corollary 4.4.7.

Something on Matlab . . .

In Matlab the Cholesky factorization of a positive definite matrix is performed by the function `chol(A);` and returns C^T.

Observation 4.4.5. *We will not discuss stability further, but Cholesky is always backward stable even without pivoting (since* $\|C\| = \|A\|^{1/2}$*).*

Observation 4.4.6. *In a sparse matrix, we can choose the (symmetric) permutation with the only goal of reducing fill-in. The same considerations about LU factorization hold in this case too.*

Note

We encountered Cholesky factorization in discussing how to solve LSP with QR factorization. Let $A \in M(m, n, \mathbb{R})$ with $m \geq n$. Thanks to thin QR factorization, we can write $A = Q_1 R_1$ such that

$$A^T A = (Q_1 R_1)^T Q_1 R_1 = {R_1}^T R_1$$

where ${R_1}^T$ is lower triangular and R_1 is upper triangular.

4.4.5 Wrap up

We designed a variant of LU factorization for the various kinds of matrices A:

- If A is positive definite, Cholesky factorization (cost: $1/3m^3 + O(m^2)$);
- If A is symmetric, LDL factorization (cost: $1/3m^3 + O(m^2)$);
- Otherwise, standard LU factorization (cost: $2/3m^3 + O(m^2)$).

In any case, if A is sparse (namely, with density < 0.4), it is important to consider sparse variants, but they suffer from the so-called fill-in.

4.5 Krylov's subspace methods

This kind of methods are useful for large matrices, whenever we run out of RAM.

Example 4.5.1. *Let us consider the optimization problem* $\min \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x}$*, where* $A \succ 0$*. We know that the solution to this problem is* $\mathbf{x} = A^{-1}\mathbf{b}$*. As discussed in Optimization course, we can solve this problem via a gradient descent-type method. We know that if* f *is strictly convex there exists a unique minimum. Let us start from* $\mathbf{x_0} = \mathbf{0}$*.*

Step 1:
$\mathbf{x_0} = \mathbf{0}$
$\nabla f(\mathbf{x_0}) = -\mathbf{b}$*;*

Step 2:
$\mathbf{x_1}$ *= some multiple of* $\nabla f(\mathbf{x_0}) \in$ *Span*$(\nabla f(\mathbf{x_0})) \equiv$ *some multiple of* $\mathbf{b}$ $\in$ *Span*$(\mathbf{b})$
$\nabla f(\mathbf{x_1}) = A\mathbf{x_1} - \mathbf{b}$

Step 3:
$\mathbf{x_2}$ *= mult. of* $\mathbf{x_1}$ *+ mul. of* $\nabla f(\mathbf{x_1})$ *+ mult. of* $\mathbf{x_0} = \alpha\mathbf{b} + \beta A\mathbf{b}$, $\mathbf{x_2} \in$ *Span*$(A\mathbf{b}, \mathbf{b})$*;*

Step 4:
$\mathbf{x_3}$ *= mult. of* $\mathbf{x_2}$*+ mult. of* $\nabla f(\mathbf{x_2})$*+ previous iterates* $\cdots$
$A\mathbf{x_2} - \mathbf{b} = A\cdot(\alpha\mathbf{b} + \beta A\mathbf{b}) - \mathbf{b} = \alpha A\mathbf{b} + \beta A^2\mathbf{b} - \mathbf{b} \in$ *span*$(\mathbf{b}, A\mathbf{b}, A^2\mathbf{b})$*,*
where $\alpha, \beta \in \mathbb{R}$*;*

Step 5: $\mathbf{x_4} \in$ *Span*$(\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, A^3\mathbf{b})$*.*

Definition 4.5.1 (Krylov subspace)**.** *Let* $A \in M(m, \mathbb{R})$ *and let* $\mathbf{b} \in \mathbb{R}^m$*. The* ***Krylov subspace*** *of index* n *is*

$$\mathbf{K_n}(\mathbf{A}, \mathbf{b}) = \textit{Span}(\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \dots, A^{n-1}\mathbf{b})$$

Equivalently,

$$\mathbf{v} \in K_n(A, \mathbf{b}) \iff \exists \alpha_1, \dots, \alpha_{n-1} \in \mathbb{R} \text{ s.t. } \mathbf{v} = \alpha_0\mathbf{b} + \alpha_1 A\mathbf{b} + \alpha_2 A^2\mathbf{b} + \cdots + \alpha_{n-1}A^{n-1}\mathbf{b}$$

Equivalently, it is the set of all polynomials p *of degree such that* $deg(p) \leq n-1$

$$(\alpha_0 + \alpha_1 A + \alpha_2 A^2 + \cdots + \alpha_{n-1}A^{n-1})\mathbf{b} = p(A)\mathbf{b}$$

Equivalently,

$$\left\{ \mathbf{v} \ : \ \mathbf{v} = V\mathbf{w}, \ \textit{where } \mathbf{w} \in \mathbb{R}^m \textit{ and } V = \begin{pmatrix} \mathbf{b} & A\mathbf{b} & \cdots & A^{n-1}\mathbf{b} \end{pmatrix} \right\}$$

Fact 4.5.1 (Properties of Krylov's subspaces)**.**
The following holds:

1. $\mathbf{v}, \mathbf{w} \in K_n(A, \mathbf{b}) \Rightarrow \alpha\mathbf{v} + \beta\mathbf{w} \in K_n(A, \mathbf{b})$*;*

2. $\mathbf{v} \in K_n(A, \mathbf{b}) \Rightarrow A\mathbf{v} \in K_{n+1}(A, \mathbf{b})$.
   Proof. *Let us take* $\mathbf{v} = \alpha_0\mathbf{b} + \dots + \alpha_{n-1}\mathbf{b}$, *then* $A\mathbf{v} = A(\alpha_0\mathbf{b} + \dots + \alpha_{n-1}\mathbf{b}) = \alpha_0 A\mathbf{b} + \dots + \alpha_{n-1}A^n\mathbf{b} \in K_{n+1}(A, \mathbf{b})$*;*

3. $\dim(K_n(A, \mathbf{b})) \leq n$. *It is exactly* n *iff* $\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \dots, A^{n-1}\mathbf{b}$ *are linearly independent.*
   Proof. *Let us assume* $dim(K_n(A, \mathbf{b})) \leq n$*. In the second point, if* A^{n-1} *was really necessary* $\alpha_{n-1} \neq 0$ *or* $\mathbf{v} \in K_n(A, \mathbf{b})$, $\mathbf{v} \notin K_{n-1}(A, \mathbf{b})$*, equivalently then* $A^n\mathbf{b}$ *is really necessary to write* $A\mathbf{v}$*, i.e.* $A\mathbf{v} \in K_{n+1}(A, \mathbf{b})$ *but* $A\mathbf{v} \notin K_n(A, \mathbf{b})$.

4. $\dim(K_1(A, \mathbf{b})) < \dim(K_2(A, \mathbf{b})) < \dots < \dim(K_{n_{max}}(A, \mathbf{b})) = \dim(K_{n_{max}+1}(A, \mathbf{b})) = \dots$.

Notice that $K_n(A, \mathbf{b})$ is the set of vectors that we can reach from $\mathbf{b}$ with two operations:

- multiply a vector by matrix A;
- take a linear combination of the vectors obtained.

Idea of the algorithm: first compute the basis of $K_n(A, \mathbf{b}) = Span(\mathbf{b}, A\mathbf{b}, \dots, A^{n-1}\mathbf{b})$, then look for the best solution inside this subspace, namely find the best $\mathbf{x_n} \in K_n(A, \mathbf{b})$ such that

$$\mathbf{x_n} = V\mathbf{y} = \mathbf{b}y_1 + A\mathbf{b}y_2 + \ldots + A^{n-1}\mathbf{b}y_n \text{ and } \min_{\mathbf{x_n} \in K_n(A,\mathbf{b})} \|A\mathbf{x_n} - \mathbf{b}\| = \min_{\mathbf{y}\ in\mathbb{R}^m} \|AV\mathbf{y} - \mathbf{b}\|$$

The method for solving linear systems based on Krylov's subspace hinges on a sort of "oracle function" that performs the matrix-vector multiplication without properly knowing A. Such oracle function may be optimized using the sparse structure of A.

Algorithm 4.5.1 Matlab code for oracle function.

```
function [x] = compute_product_with_A(z)
x = zeros(size(A, 1), 1);
x[i:end] = A[i:end, :] * z;
end
```

Fact 4.5.2. *Let* $A \in M(m, \mathbb{R})$ *and let* $\mathbf{z} \in \mathbb{R}^m$*. The cost of multiplying* A *times* $\mathbf{z}$ *is* $O(nnz(A))$*, where* $nnz(A)$ *is the number of non zero entries of* A*.*

It goes without saying that if A is sparse, these algorithms become particularly fast. Moreover, if we somehow have matrices that are not really sparse, but for which there exists a clever implementation of the matrix-vector product, this class of algorithms will give good results.

Notice that $V = \begin{pmatrix} b & Ab & \cdots & A^{n-1}b \end{pmatrix}$ is a bad basis, because it is ill conditioned. This is due to the fact that, in general, powers of the form $A^*\mathbf{b}$ "converge" (tend to be aligned) to a certain (dominant) eigenvector fo A.
A first way of overcoming this issue would be taking the QR factorization of $\begin{pmatrix} V & 0 \end{pmatrix}$: $\begin{pmatrix} Q_1 & R_1 \end{pmatrix}$ and use Q_1 as a basis of $K_n(A, \mathbf{b})$. This is not enough, because V is already ill-conditioned, hence storing it in memory will already alter $K_n(A, \mathbf{b})$ a lot.

4.6 Arnoldi algorithm

Do you recall?

The Gram-Schmidt algorithm, given a matrix $W \in M(m, n, \mathbb{R})$, such that $W = \begin{pmatrix} \mathbf{w_1} & \mathbf{w_2} & \dots & \mathbf{w_n} \end{pmatrix}$, computes an orthonormal basis for $Im(W) = Span(\mathbf{w_1}, \mathbf{w_2}, \dots, \mathbf{w_n})$.

This algorithm resembles the Gram-Schmidt algorithm for linear algebra, where we take Q with orthonormal columns $Q = \begin{pmatrix} \mathbf{q_1} & \mathbf{q_2} & hdots & \mathbf{q_n} \end{pmatrix} \in M(m, n, \mathbb{R})$, whose columns are a basis of $K_n(A, \mathbf{b})$.
The idea behind this algorithm is to build an orthogonal basis of $K_n(A, \mathbf{b})$ incrementally: at a generic step, it takes an orthogonal basis for $K_n(A, \mathbf{b})$ and adds a vector to produce one of $K_{n+1}(A, \mathbf{b})$.

Step 1: $K_1(A, \mathbf{b}) = Span(\mathbf{b})$, $\mathbf{q_1} = \frac{\mathbf{b}}{\|\mathbf{b}\|}$

Generic j-th step: produce a vector $\mathbf{w}$, belonging to $K_{j+1}(A, \mathbf{b}) \backslash K_j(A, \mathbf{b})$, namely $\mathbf{w} = A\mathbf{q_j}$

1. Compute $\beta_i = \mathbf{q_i}^T \mathbf{w}$ for $i = 1, \dots, j+1$
2. Compute $\mathbf{q_{j+1}} = \frac{\mathbf{w} - \mathbf{q_1}\beta_1 - \mathbf{q_2}\beta_2 - \dots - \mathbf{q_j}\beta_j}{\beta_{j+1}}$

Fact 4.6.1. *Let $A \in M(m, n, \mathbb{R})$, let $\mathbf{b} \in \mathbb{R}^n$, and let $\mathbf{w} = \mathbf{q_1}\beta_1 + \mathbf{q_2}\beta_2 + \dots + \mathbf{q_j}\beta_j + \mathbf{q_{j+1}}\beta_{j+1} \in K_{j+1}(A, \mathbf{b}) \backslash K_j(A, \mathbf{b})$.*

$$\beta_i = \mathbf{q_i}^T \mathbf{w} \text{ for } i = 1, \dots, j+1$$

and

$$\mathbf{q_{j+1}} = \frac{\mathbf{w} - \mathbf{q_1}\beta_1 - \mathbf{q_2}\beta_2 - \dots - \mathbf{q_j}\beta_j}{\beta_{j+1}}$$

Proof.

$$\mathbf{q_i}^T \mathbf{w} = \mathbf{q_i}^T \mathbf{q_1}\beta_1 + \dots + \mathbf{q_i}^T \mathbf{q_j}\beta_j + \mathbf{q_i}^T \mathbf{q_{j+1}}\beta_{j+1} = \mathbf{q_i}^T \mathbf{q_i}\beta_i = \beta_i$$

The second part comes from inverting the formula of $\mathbf{w}$. □

Notice that we are assuming that at each step j the vector $\mathbf{q_j} \notin K_{j-1}(A, \mathbf{b})$.

An implementation of Arnoldi algorithm is shown in Algorithm 4.6.1.

Algorithm 4.6.1 Arnoldi algorithm Matlab implementation.

```
function Q = arnoldi(A, b, n)
Q = zeros(length(b), n); %will be filled in
H = zeros(n+1, m);
Q(:, 1) = b / norm(b);
for j = 1 : n
  w = A * Q(:, j);
  for i = 1:j
  % not what we showed earlier here, but stabler
    betai = Q(:, i)' * w;
    w = w - betai * Q(:, i);
    H(i, j) = betai;
  end
  nrm = norm(w);
  H(j+1, j) = nrm;
  Q(:, j+1) = w / nrm;
end
```

Notice that in the implementation we compute

$$\begin{aligned} \beta_i &= \mathbf{q_1}^T \mathbf{w} \\ \mathbf{w} &\leftarrow \mathbf{w} - \beta_1 \mathbf{q_1} \\ \beta_2 &= \mathbf{q_2}^T \mathbf{w} \\ \mathbf{w} &\leftarrow \mathbf{w} - \beta_2 \mathbf{q_2} \end{aligned}$$

because it is more stable.

Fact 4.6.2. *The complexity of Arnoldi algorithms is $O(n \cdot nnz(A) + m \cdot n)$.*

Proof.

- $n - 1$ products with A $\rightarrow O(n \cdot nnx(A))$
- $\frac{n^2}{2}$ scalar products $\rightarrow O(m \cdot n^2)$
- $\frac{n^2}{2}$ linear combinations of vectors $\rightarrow O(m \cdot n^2)$
- norm

□

4.6.1 Arnoldi as a factorization

Definition 4.6.1 (Hessemberg Matrix). *Let $H \in M(m, \mathbb{R})$ such that $H_{ij} = 0,\ \forall i > j+1$. H is called* ***Hessemberg matrix****.*

$$H = \begin{bmatrix} \times & \times & \times & \cdots & \times \\ \times & \times & \times & \cdots & \times \\ 0 & \times & \times & \cdots & \times \\ 0 & 0 & \times & \cdots & \times \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \times \end{bmatrix}$$

Fact 4.6.3. *Let $A \in M(m, n, \mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^n$.*

$$AQ_n = Q_{n+1}H$$

where $Q_n \in M(m, n, \mathbb{R})$, $Q_{n+1} \in M(m, n+1, \mathbb{R})$ and $H \in M(n+1, n, \mathbb{R})$ such that

$$Q_n = \begin{pmatrix} \mathbf{q_1} & \mathbf{q_2} & \dots & \mathbf{q_n} \end{pmatrix},\ Q_{n+1} = \begin{pmatrix} \mathbf{q_1} & \mathbf{q_2} & \dots & \mathbf{q_{n+1}} \end{pmatrix},\ H = \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} & \cdots & \beta_{1,n} \\ \beta_{2,1} & \beta_{2,2} & \beta_{2,3} & \cdots & \beta_{2,n} \\ 0 & \beta_{3,2} & \beta 3{,}3 & \cdots & \beta_{3,n} \\ 0 & 0 & \beta_{4,3} & \cdots & \beta_{4,n} \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \beta_{n+1,n} \end{bmatrix}$$

Proof. At step j

$$\mathbf{w} = A\mathbf{q_j} = \beta_{1,j}\mathbf{q_1} + \beta_{2,j}\mathbf{q_2} + \cdots + \beta_{j,j}\mathbf{q_j} + \beta_{j+1,j}\mathbf{q_{j+1}} = Q_{n+1}\begin{bmatrix} \beta_{1,j} \\ \beta_{2,j} \\ \vdots \\ \beta_{j+1,j} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

If we collect all $\mathbf{q_j}$ in matrix Q_n and we get the thesis. □

Corollary 4.6.4. *Let $A \in M(m, n, \mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^n$.*

$$AQ_n = \begin{pmatrix} Q_n & \mathbf{q_{n+1}} \end{pmatrix} \cdot \begin{pmatrix} & H_n & \\ \hline 0 & \cdots & 0 \quad \beta_{n+1,n} \end{pmatrix} = Q_nH_n + \mathbf{q_{n+1}}\beta_{n+1,n}\mathbf{e_1}$$

where $Q_n \in M(m, n, \mathbb{R})$, $H \in M(n, n, \mathbb{R})$ is an Hessemberg matrix.

Fact 4.6.5. *For every matrix $A \in M(m, n, \mathbb{R})$ there exists an Arnoldi factorization.*

An attentive reader may have noticed that $AQ_n = Q_{n+1}H$ does not allow a factorization of the matrix A, because Q_n is not invertible, because it is tall, thin hence it does not have an inverse.

4.6.2 Arnoldi Termination

Definition 4.6.2 (Arnoldi breakdown)**.** *Let $A \in M(m, n, \mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^n$. Let us assume that we arrived at step $j \le m$ such that $\mathbf{q_1}, \mathbf{q_2}, \dots, \mathbf{q_j}$ are a basis of $K_{j+1}(A, \mathbf{b}) \subseteq \mathbb{R}^m$. We say that we are encountering **Arnoldi breakdown** and*

$$A\mathbf{q_j} = \beta_1\mathbf{q_1} + \cdots + \beta_m\mathbf{q_j} + 0$$

(without any additional term $\beta_{j+1}\mathbf{q_{j+1}}$)

Fact 4.6.6. *Let $A \in M(m, n, \mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^n$. If we get to m without earlier breakdown*

$$A = Q_m H {Q_m}^T$$

where $Q_m \in O(m, \mathbb{R})$ and $H_m \in M(m, \mathbb{R})$ is a Hessemberg matrix.

Fact 4.6.7. *The QR factorization of an Hessemberg matrix $H \in M(m, \mathbb{R})$ can be computed in $O(m^2)$ operations.*

Thanks to the combination of Proposition 4.6.6 and Proposition 4.6.7, we can easily invert matrix A and get a solution $\mathbf{x}$.

Theorem 4.6.8 (Lucky breakdown)**.** ***'Lucky breakdown'**: if it happens at an early step, we can solve linear systems (or compute some eigenvalues) cheaply: it costs n times a matrix-vector products + $O(mn^2)$.*

Theorem 4.6.9. *Let us assume we have a lucky breakdown at step n, then*

- *for every couple $\lambda, \mathbf{v}$ eigenpair of H_n the couple $\lambda, Q_n\mathbf{v} = \mathbf{w}$ is an eigenpair of A. Formally,*

$$H_n\mathbf{v} = \lambda\mathbf{v} \Rightarrow A\mathbf{w} = \lambda\mathbf{w} \text{ where } \mathbf{w} = Q_n\mathbf{v}$$

- *the solution of $A\mathbf{x} = \mathbf{b}$ is*

$$\mathbf{x} = Q_n \left\|\mathbf{b}\right\| H_n^{-1}\mathbf{e}_1$$

Proof.

$$A = \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} H_n & L \\ 0 & M \end{bmatrix} \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix}^T$$

- $$AQ_n = Q_nH_n + \mathbf{q_{n+1}}\beta_{n+1,n}{\mathbf{e_n}}^T \overset{*}{=} Q_nH_n$$

  where $\overset{*}{=}$ holds since we have a breakdown at step n. Hence we have

$$AQ_n\mathbf{v} = Q_nH_n\mathbf{v} = Q_n\lambda\mathbf{v} = \lambda Q_n\mathbf{v}$$

•

$$\mathbf{x} = A^{-1}\mathbf{b} = Q_m H^{-1} {Q_m}^T \mathbf{b} = Q_m H^{-1} \begin{bmatrix} \mathbf{q_1}^T \\ \mathbf{q_2}^T \\ \vdots \\ \mathbf{q_m}^T \end{bmatrix} \mathbf{b} \overset{(1)}{=} Q_m H^{-1} \begin{bmatrix} \|\mathbf{b}\| \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} {H_n}^{-1} & -{H_n}^{-1} L M^{-1} \\ 0 & M^{-1} \end{bmatrix} \begin{bmatrix} \|\mathbf{b}\| \, \mathbf{e}_1 \\ \mathbf{0} \end{bmatrix}$$

$$= \begin{bmatrix} Q_n & \widehat{Q} \end{bmatrix} \begin{bmatrix} \|\mathbf{b}\| \, H_n^{-1} \mathbf{e}_1 \\ \mathbf{0} \end{bmatrix} = Q_n \|\mathbf{b}\| \, H_n^{-1} \mathbf{e}_1 .$$

□

Something on Matlab . . .

In Matlab a nice way to plot eigenvalues is:

- `ev = eig(A);`
- `plot(real(ev), image(ev), 'x');`

On the x axis, the real component of the eigenvalues, on the y axis the complex component.

Notice that if $\mathbf{b}$ is an eigenvector of A, the breakdown happens already at $n = 1$.

Moreover, how to deal with a division by 0 in $\mathbf{q_{n+1}} = \frac{\mathbf{z}}{\|\mathbf{z}\|}$? We need to change the definition $\beta_{n+1} = \|\mathbf{z}\| = 0$. At this point we do not get a basis of the Krylov space anymore, but we can still go on as "nothing happened", as long as these vectors are orthonormal. We go on until the end we get

$$AQ_m = H_m Q_m,$$

Fact 4.6.10. *Even if no breakdown happens, the eigenpairs of H_n $(\lambda, Q_n\mathbf{v})$ are often close to the eigenpairs of A (see Figure 4.3).*

Definition 4.6.3 (Ritz vectors)**.** *The eigenvalues of H_n are called Ritz vectors of A.*

The Krylov's space Q_n looks a lot like the space generated by the first n eigenvalues. If A is diagonalizable $A = V\Lambda V^{-1}$, $A^k = V\Lambda^k V^{-1}$. Hence $A^k\mathbf{b} = V\Lambda \underbrace{V^{-1}\mathbf{b}}_{\mathbf{c}} = V^1 {\lambda_1}^k c_1 + \ldots + V^m {\lambda_m}^k c_m$. When k gets large, the dominant eigenvalue λ_1 dominates $A^k\mathbf{b}$ gets very close to $Span(V^1, \ldots, V^m)$.

What happens when there is no breakdown? After n steps of Arnoldi,

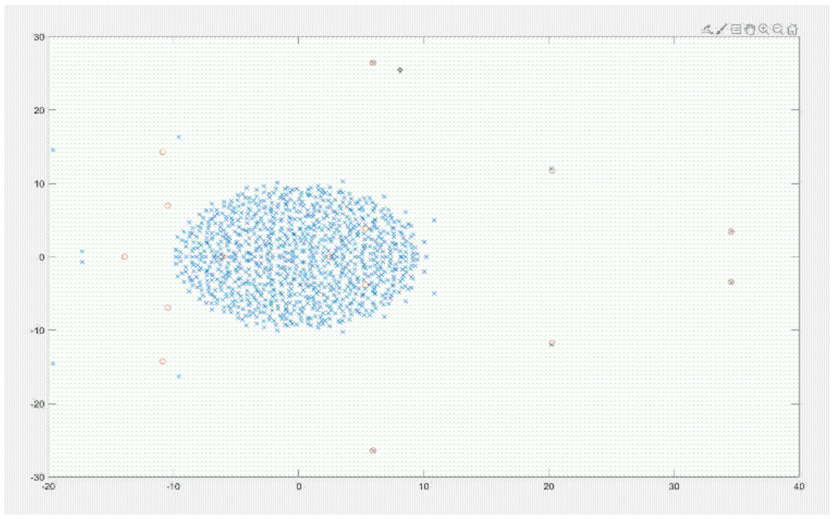


FIGURE 4.3: Indicated as a blue cross the eigenvalues of A, in red circles the eigenvalues of H.

1. if $H_n\mathbf{v} = \lambda\mathbf{v}$ is an eigenpair of H_n. $Q_n\mathbf{v}$, λ is an approximation of an eigenpair of A.

2. $\widetilde{\mathbf{x}} = Q_n {H_n}^{-1} \|\mathbf{b}\| \, \mathbf{e_1}$ is an approximation of the solution $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$

4.6.3 Stability of Arnoldi

Suppose $A\mathbf{w} - \lambda\mathbf{w} = \mathbf{z}$, where $\|\mathbf{z}\| = \varepsilon$. Then $(A + E)\mathbf{w} = \lambda\mathbf{w}$ for some small E, such that $E\mathbf{w} = \mathbf{z}$.

Let us define $\mathbf{q} = \frac{\mathbf{z}}{\|\mathbf{z}\|}$, $E = \frac{\mathbf{w}\mathbf{q}^T}{\|\mathbf{z}\|}$, then

$$E\mathbf{z} = \frac{\mathbf{w}\mathbf{q}^T}{\|\mathbf{z}\|} \cdot \mathbf{z} = \mathbf{w}\mathbf{q}^T \frac{1}{\|\mathbf{z}\|} \cdot \|\mathbf{z}\| \cdot \mathbf{q} = \mathbf{w}$$

Then $\|E\| = \frac{\|\mathbf{w}\|}{\|\mathbf{z}\|}$

We would like to check the backward stability, that means finding the exact solution of a nearby problem. There is an exact eigenvector $\hat{\mathbf{w}}$ of A such that

$$\frac{\|\hat{\mathbf{w}} - \mathbf{w}\|}{\|\mathbf{w}\|} \leq \kappa(eig.A) \cdot \frac{\|E\|}{\|A\|}$$

where $\kappa(eig.A)$ is the condition number of the iegenvalue problem.

The eigenvalues of H_n are eigenvalues of a "nearby matrix" obtained by taking $\widetilde{H}_m$ (result of the full process) and replacing $(\widetilde{H}_m)_{n+1,n}$ with zero.

$$\widetilde{H}_m = \left[\begin{array}{cccc|cccc}
\times & \cdots & \times & \times & \times & \cdots & \cdots & \times \\
\times & \cdots & \times & \times & \times & \cdots & \cdots & \times \\
0 & \ddots & \times & \times & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \times & \times & \times & \cdots & \cdots & \times \\
\hline
 & & & \textcircled{0} & \times & \cdots & \times & \times \\
 & & & & \times & \cdots & \times & \times \\
 & & & & 0 & \ddots & \times & \times \\
 & & & & 0 & 0 & \times & \times
\end{array}\right] \to H_m = \left[\begin{array}{cccc|cccc}
\times & \cdots & \times & \times & \times & \cdots & \cdots & \times \\
\times & \cdots & \times & \times & \times & \cdots & \cdots & \times \\
0 & \ddots & \times & \times & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \times & \times & \times & \cdots & \cdots & \times \\
\hline
 & & & \textcircled{\times} & \times & \cdots & \times & \times \\
 & & & & \times & \cdots & \times & \times \\
 & & & & 0 & \ddots & \times & \times \\
 & & & & 0 & 0 & \times & \times
\end{array}\right]$$

We expect that this change does not lead to a significant change in the eigenvalues, in other words, the eigenvalues of $\widetilde{H}_m$ differ from the eigenvalues of H_m by $|h_{n+1,n}|$. Formally, $\left\|\widetilde{H}_m - H_m\right\| = h_{n+1,n}$.

4.6.4 Using Arnoldi for computing eigenvalues

Arnoldi's method can be used for approximating the *largest eigenvalues* of a large, sparse matrix A.

Fact 4.6.11. *The space* $K_n(A, \mathbf{b}) = \operatorname{span}(\mathbf{b}, A\mathbf{b}, \dots, A^{n-1}\mathbf{b})$ *contains the right "features" to represent the eigenvectors of A associated to the largest eigenvalues.*

Proof. Let us take A diagonalizable: $A = V\Lambda V^{-1}$. Then

$$\begin{aligned}
A^k\mathbf{b} &= (V\Lambda V^{-1}) \cdot \ldots \cdot (V\Lambda V^{-1})\mathbf{b} = V\Lambda^k V^{-1}\mathbf{b} \\
&= \begin{pmatrix} V^1 & V^2 & \cdots & V^m \end{pmatrix} \cdot \begin{pmatrix} {\lambda_1}^k & & & \\ & {\lambda_2}^k & & \\ & & \ddots & \\ & & & {\lambda_m}^k \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} \\
&= V^1\lambda_1^k c_1 + V^2\lambda_2^k c_2 + \cdots + V^m\lambda_m^k c_m
\end{aligned}$$

where $\mathbf{c} = V^{-1}\mathbf{b}$.

$A^k\mathbf{b}$ is a linear combination of the eigenvectors V^i in which those with largest $|\lambda_i|$ are "more prominent", in other words as k increases the components involving the largest $|\lambda_i|$s grow faster. □

Corollary 4.6.12. *This also tells us that* $\operatorname{span}(V^1, V^2, \dots, V^m)$ *(that are the eigenvectors associated to largest eigenvalues in modulus) "represent well"* $K_m(A, b) = \operatorname{span}(\mathbf{b}, A\mathbf{b}, \dots, A^{m-1}\mathbf{b})$.

Example 4.6.1. *It is possible to build a counter-example, that shows that sometimes it is crucial to reach the n-th step to have accurate eigenvalues. Let*

$A \in M(m, \mathbb{R})$, such that $A = \begin{pmatrix} 0 & & & & 1 \\ 1 & 0 & & & \\ & 1 & \ddots & & \\ & & \ddots & & \\ & & & 1 & 0 \end{pmatrix}$

In this case the eigenvalues are 0 *until the last iteration and they get the correct value only at the last step.*

Something on Matlab ...

The command `[V, D] = eigs(A)` does not work on sparse matrices A. In this case we may run Arnoldi method and use the best values obtained by Arnoldi: `[V, D] = eigs(A, n)`, which computes approximations of the top-n (largest in modulus) eigenvalues.
Notice that the Matlab "implementation" (the quotes are because the Arnoldi method *de facto* is not implemented in Matlab) of the Arnoldi method uses some tricks to converge fast.
It is possible to use the command `eigs` and pass to it an **anonymous** (λ) function which computes the matrix-vector product and this is useful when the matrix and the vector have a particular shape: `f= @(x) A*x;`

Lemma 4.6.13. *Let $A \in M(m, \mathbb{R})$ and let $(\lambda_1, \mathbf{v}_1), \dots, (\lambda_k, \mathbf{v}_k)$ be the eigenvalues/vectors of A. The following holds:*

1. *$(\lambda_i + \alpha, \mathbf{v}_i)$ are eigenvalues/vectors of $A + \alpha I$;*
2. *$(\frac{1}{\lambda_i}, \mathbf{v}_i)$ are eigenvalues/vectors of A^{-1};*
3. *$(\lambda_i^k, \mathbf{v}_i)$ are eigenvalues/vectors of A^k;*
4. *$(\frac{1}{\lambda_i - \alpha}, \mathbf{v_i})$ are eigenvalues/vectors of $(A - \alpha I)^{-1}$*

Proof. Let us omit the subscript i to ease notation:

1. $(A + \alpha I)\mathbf{v} = A\mathbf{v} + \alpha\mathbf{v} = \lambda\mathbf{v} + \alpha\mathbf{v} = (\lambda + \alpha)\mathbf{v}$;
2. $(\lambda^{-1}\mathbf{v})$ is an eigenpair of A^{-1}. We need to check that $A^{-1}\mathbf{v} = \lambda^{-1}\mathbf{v}$. If we multiply by λA both sides: $\lambda \cancel{A A^{-1}}\mathbf{v} = \lambda A \lambda^{-1}\mathbf{v} \Leftrightarrow \lambda\mathbf{v} = \cancel{\lambda\lambda^{-1}} A\mathbf{v}$, which is true by definition of eigenvalue/vector of A;
3. (by induction) $A^2\mathbf{v} = A(A\mathbf{v}) = A \cdot \lambda\mathbf{v} = \lambda A\mathbf{v} = \lambda\lambda\mathbf{v} = \lambda^2\mathbf{v}$.

□

Fact 4.6.14. *Arnoldi's algorithm can be used for computing the smallest eigenvalues or also those that are closest to some value $\mu \in \mathbb{R}$.*

Proof. Thanks to the fourth item of Lemma 4.6.13, we get that if $(\lambda, \mathbf{v})$ is an eigenpair of A, then $((\lambda-\mu)^{-1}, \mathbf{v})$ is an eigenpair of $B=(A-\mu I)^{-1}$, since $(\lambda-\mu^{-1})$ is large whenever λ and μ are close.

Following the same reasoning, we can compute the smallest eigenvalues of A by taking the inverse of the largest eigenvalues of A^{-1}, namely $\frac{1}{\lambda_m}, \dots, \frac{1}{\lambda_1}$. □

Notice that such matrix B is not sparse when A is sparse.

Something on Matlab . . .

In Matlab, we can overcome the issue of matrix B not being sparse, avoiding to compute it directly and providing an anonymous function that computes the product $B\mathbf{z} = A-\mu I^{-1}\mathbf{z}$. The idea is to use factorizations: $A-\mu I = LU$, then $B\mathbf{z} = U^{-1}(L^{-1}\mathbf{z})$, that we can compute by back-substitution: `f = @(x) U \(L \x);` Moreover, we can use:

- `fl = eigs(A, 5, mu);` for computing 5 eigenvalues closest to mu
- `fl = eigs(A, 5, 'SM');` for computing 5 eigenvalues with smallest magnitude
- `fl = eigs(A, 5, 'LM');` for computing 5 eigenvalues with largest magnitude

As final observation, the equivalents to Matlab's `eigs` function are `scipy.linalgs.eigs` for Python and `arpack` for C/C++ and Fortran.

4.6.5 GMRES: Using Arnoldi for solving linear systems

In previous sections, we already discussed that solving linear systems using Arnoldi's algorithm is easy in those cases in which there is lucky breakdown.

In the case of no breakdown, we can

- use $\widetilde{\mathbf{x}} = Q_n {H_n}^{-1} \|\mathbf{b}\| \mathbf{e_1}$ as an approximation of the solution $\mathbf{x}$ of $A\mathbf{x}=\mathbf{b}$
- use the **G**eneralized **M**inimum **RES**idual (GMRES) algorithm

In GMRES we want to approximate the solution of a large-scale linear system of the form $A\mathbf{x}=\mathbf{b}$ by looking for "the closest thing to solution" inside $K_n(A,\mathbf{b})$.

Fact 4.6.15. *Let $A \in M(m,n,\mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^m$ such that Arnoldi's algorithm does not lead to a breakdown. The solution of the minimum problem*

$$\min_{\mathbf{x}\in K_n(A,\mathbf{b})} \|A\mathbf{x}-\mathbf{b}\|$$

is expressed as

$$\mathbf{x} = Q_n \cdot {H_n}^{\dagger} \|\mathbf{b}\| \mathbf{e_1}$$

Proof. Let us take $\mathbf{x} = Q_n\mathbf{y}$, then the minimum problem is equivalent to $\min\limits_{\mathbf{y}\in\mathbb{R}^n} \|AQ_n\mathbf{y} - \mathbf{b}\|$.

We can perform some more reductions and:

$$\begin{aligned}
\|AQ_n\mathbf{y} - \mathbf{b}\| &\overset{\mathbf{(1)}}{=} \|Q_{n+1}\underline{H}_n\mathbf{y} - \mathbf{b}\| \\
&\overset{\mathbf{(2)}}{=} \|Q_{n+1}\underline{H}_n\mathbf{y} - Q_{n+1}\|\mathbf{b}\|\,\mathbf{e_1}\| \\
&= \|Q_{n+1}\cdot(\underline{H}_n\mathbf{y} - \|\mathbf{b}\|\,\mathbf{e_1})\| \\
&\overset{\mathbf{(3)}}{=} \|\underline{H}_n\mathbf{y} - \|\mathbf{b}\|\,\mathbf{e_1}\|\,.
\end{aligned}$$

where

- $\overset{\mathbf{(1)}}{=}$ is due to Arnoldi's equivalence $AQ_n = Q_{n+1}H_n$, with $H_n \in M(n+1, n)$
- $\overset{\mathbf{(2)}}{=}$ follows from

$$\begin{aligned}
\mathbf{b} = \mathbf{q_1}\|\mathbf{b}\|\,\mathbf{e_1} &= \begin{pmatrix} \mathbf{q_1} & \cdots & \mathbf{q_n} \end{pmatrix} \cdot \begin{pmatrix} \|\mathbf{b}\| \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
&= Q_n \begin{pmatrix} \|\mathbf{b}\| \\ 0 \\ \vdots \\ 0 \end{pmatrix} = Q_n\mathbf{e_1}\|\mathbf{b}\| = Q_{n+1}\mathbf{e_1}\|\mathbf{b}\|
\end{aligned}$$

- $\overset{\mathbf{(3)}}{=}$ is explained by Proposition 1.4.1 ($\|Q\mathbf{x}\| = \|\mathbf{x}\|$ for Q orthogonal)

□

Corollary 4.6.16. *The cost of Arnoldi's algorithm for solving linear systems is $O(n^3)$ and it does not depend on m.*

Proof. $qr(H)$ can be computed in $O(n^2)$ using the fact that H is 'almost triangular' (Hessemberg matrix), although it is not a big optimization, since n Arnoldi steps need to be computed first.

$$H = \left[\begin{array}{cccc|cccc}
\times & \cdots & \times & \times & \times & \cdots & \cdots & \times \\
\times & \cdots & \times & \times & \times & \cdots & \cdots & \times \\
0 & \ddots & \times & \times & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \times & \times & \times & \cdots & \cdots & \times \\
\hline
 & & & \otimes & \times & \cdots & \times & \times \\
 & & & & \times & \cdots & \times & \times \\
 & \Large 0 & & & 0 & \ddots & \times & \times \\
 & & & & 0 & 0 & \times & \times
\end{array}\right]$$

□

An implementation of Arnoldi's algorithm for solving linear systems is shown in Algorithm 4.6.2.

ALGORITHM 4.6.2 Arnoldi's algorithm for linear systems Matlab implementation.

```
function x = GMRES(A, b, n)
[Q, H] = arnoldi(A, b, n);
v = norm(b) + eye(n+1, 1);
y = H \ v;
x = Q(:, 1:n) * y;
```

Notice that instead of doing a QR at the end, we can compute QRs of $\underline{H}_1, \underline{H}_2, \ldots$ and update them at each step. This allows us to compute at each step a residual $\|A\mathbf{x_n} - \mathbf{b}\|$ that we can use as stopping criterion.

Something on Matlab ...

Matlab has `gmres(A, b)` (and Python has `scipy.sparse.linalg.gmres`).

GMRES Convergence Speed

Fact 4.6.17. *Let $A \in M(m, n, \mathbb{R})$ and let $\mathbf{b} \in \mathbb{R}^m$. If A has k different eigenvalues, there exists a polynomial p with degree $\leq k-1$ such that $p(\lambda_i) = \frac{1}{\lambda_i}$, $\forall i = 1, \ldots, m$. Then GMRES recovers the exact solution in exactly k iterations.*

Proof. Let us resort the definition of Krylov's space as a polynomial:

$$\mathbf{x} \in K_n(A, \mathbf{b}) \iff \mathbf{x} = \alpha_0\mathbf{b} + \alpha_1 A\mathbf{b} + \ldots + \alpha_{n-1}A^{n-1}\mathbf{b} = p(A)\mathbf{b}, \text{ for some } \alpha_i \in \mathbb{R}$$

We can now bound the norm of the residual as

$$\begin{aligned}\min_{\cdots} \|A\mathbf{x} - \mathbf{b}\| &= \min_{\cdots} \|A \cdot p(A) \cdot \mathbf{b} - \mathbf{b}\| \\ &= \min_{\cdots} \|(A \cdot p(A) - I) \cdot \mathbf{b}\| \\ &\leq \min_{\cdots} \|(A \cdot p(A) - I)\| \cdot \|\mathbf{b}\|\end{aligned}$$

We are interested in putting to 0 the quantity $\|(A \cdot p(A) - I)\|$ and this is possible iff

$$V \begin{bmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_m p(\lambda_m) - 1 \end{bmatrix} V^{-1} = 0$$

And this holds iff $\forall i,\ \lambda_i p(\lambda_i) - 1 = 0 \iff p(\lambda_i) = \frac{1}{\lambda_i}$. This is a polynomial interpolation problem and it can be solved in k steps if k is the number of distinct eigenvalues. $\square$

Notice that if A has more than k eigenvalues, but all of them are clustered around k distinct values, then it is possible to find a polynomial that attains $p(\lambda_i) = \frac{1}{\lambda_i}, \forall i = 1, \dots, m$, hence the residual should be small:

$$\begin{aligned}
\|\mathbf{r_n}\| = \|A\mathbf{x_n} - \mathbf{b}\| &= \min_{\substack{p(x) \\ \text{of degree} \leq n-1}} \|(Ap(A) - I)\mathbf{b}\| \\
&\leq \|V\| \cdot \min_{\substack{p(x) \\ \text{of degree} \leq n-1}} \left\| \begin{bmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_m p(\lambda_m) - 1 \end{bmatrix} \right\| \cdot \|V\| \cdot \|\mathbf{b}\| \\
&\leq \|V\| \cdot \left\|V^{-1}\right\| \cdot \|\mathbf{b}\| \min_{\substack{p(x) \\ \text{of degree} \leq n-1}} \max_i \|\lambda_i p(\lambda_i) - 1\|
\end{aligned}$$

Notice that Gauss operations on the rows of any matrix A (e.g. swapping rows or scalar multiplication of a row) change its eigenvalues, without changing the solution.

More generally, given $P \in M(n, \mathbb{R})$ we can change the problem $A\mathbf{x} = \mathbf{b}$ to $PA\mathbf{x} = P\mathbf{b}$. If P is invertible, the two systems have the same solution. However, the spectrum of PA may be much better (in the above sense) than the spectrum of A, leading to a faster solution with GMRES.

In particular, this happens if we manage to find $P \approx A^{-1}$. The perfect choice would be $P = A^{-1}$, but, of course, if we knew A^{-1} we would already have a way to solve linear systems: just compute the matrix multiplication $A^{-1}\mathbf{b}$.

There are various techniques (often problem-dependent) to build effective *preconditioners P*. One comes from approximate LU factorizations of A (in a suitable sense); they are known as *incomplete LU* preconditioners.

4.6.6 MINRES: Modified Arnoldi for symmetric matrices

MINRES algorithm is based on two different modifications of GMRES:

- Improvement of Arnoldi step, due to the symmetry of matrix A (Lanczos)
- Improvement in solving LS problem (specialized GMRES)

Lanczos step

Fact 4.6.18. *Let $A \in M(m, \mathbb{R})$. Then $H_n \in S(n, \mathbb{R}),\ \forall n \leq m$.*

Proof. According to Arnoldi's algorithm, we have that a vector $\mathbf{w} \in K_{j+1}(A, \mathbf{b}) \backslash K_j(A, \mathbf{b})$ is chosen as $\mathbf{w} = A\mathbf{q_j}$. The Hessemberg matrix is such that

$$H_{ij} = \beta_{ij} = \mathbf{q_i}^T \cdot \mathbf{w} = \mathbf{q_i}^T \cdot A\mathbf{q_j},\ \forall i, j$$

Therefore

$$(H_{ij})^T = \mathbf{q_j}^T A^T \mathbf{q_i} = \mathbf{q_j}^T A \mathbf{q_j} = H_{ij}$$

□

Corollary 4.6.19. *A symmetric Hessemberg matrix is also tridiagonal.*

$$\begin{pmatrix} \times & \times & & & \\ \times & \times & \times & \mathit{0} & \\ & \ddots & \ddots & \ddots & \\ & \mathit{0} & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

Notice that a tridiagonal matrix requires a lot less space, hence we can save a lot of computation in the Arnoldi loop, as shown Algorithm 4.6.3.

Algorithm 4.6.3 Lanczos algorithm Matlab implementation.

```
function Q = lanczos(A, b, n)
Q = zeros(length(b), n); %will be filled in
H = zeros(n+1, m);
Q(:, 1) = b / norm(b);
for j = 1 : n
        w = A * Q(:, j);
        for i = j-1:j %second modification
                betai = Q(:, i)' * w;
                w = w - betai * Q(:, i); %performed only on non-zero entries
                H(i, j) = betai;
        end
        nrm = norm(w);
        H(j+1, j) = nrm;
        Q(:, j+1) = w / nrm;
end
```

Fact 4.6.20. *The cost of Lanczos algorithm is* $n \cdot nnz(A) + m \cdot n$*.*

Specialized GMRES

There is an extra saving when solving LS problem $\min \|H_n \mathbf{y} - \mathbf{e_1} \cdot \|\mathbf{b}\|\|$.

4.6.7 CG: Modified Arnoldi for positive definite matrices

Let us take $A = A^T \in M(m, \mathbb{R})$ such that A is positive definite. Then, we can find the solution to $A\mathbf{x} = \mathbf{b}$ by minimizing the (strictly convex) function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$, which means imposing the gradient to 0:

$$\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b} = \mathbf{0} \iff A\mathbf{x} = \mathbf{b}$$

Surprisingly, conjugate gradient on this problem can be interpreted as a Krylov subspace method.

Definition 4.6.4 (A-orthogonal)**.** *Let $A \in M(m, \mathbb{R})$ and let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$. The vectors $\mathbf{x}, \mathbf{y}$ are said to be **A-orthogonal** if $\mathbf{x}^T A\mathbf{y} = 0$.*

Definition 4.6.5 (A-norm)**.** *Let $A \in M(m, \mathbb{R})$ positive definite and let $\mathbf{x} \in \mathbb{R}^m$. We term **A-norm** the square root of the scalar product of $\mathbf{x}$ by A. Formally,*

$$\|\mathbf{x}\|_A = \sqrt{\mathbf{x}^T A\mathbf{x}}, \textit{ where } \mathbf{x}^T A\mathbf{x} \geq 0, \textit{ because } A \succ 0$$

The conjugate method algorithm (Algorithm 4.6.4) works by keeping track of 3 different vectors: $\mathbf{x_j}$ the current iterate, $\mathbf{r_j} = \mathbf{b} - A\mathbf{x_j} = -\nabla f(\mathbf{x_j})$ the residual (that expresses how far we are from the solution), and the search direction $\mathbf{d_j}$.

Algorithm 4.6.4 Pseudocode for the conjugate gradient method.

```
1: procedure CG_ITERATION
2:     $\mathbf{x_0} \leftarrow \mathbf{0}$;
3:     $\mathbf{r_0} \leftarrow \mathbf{b}$;
4:     $\mathbf{d_0} \leftarrow \mathbf{b}$;
5:     for j = 1:n do
6:         $\alpha_j \leftarrow \frac{\mathbf{r_{j-1}}^T \mathbf{r_{j-1}}}{\mathbf{d_{j-1}}^T A \mathbf{d_{j-1}}}$;
7:         $\mathbf{x_j} \leftarrow \mathbf{x_{j-1}} + \alpha_j \mathbf{d_{j-1}}$;
8:         $\mathbf{r_j} \leftarrow \mathbf{r_{j-1}} - \alpha_j A \mathbf{d_{j-1}}$;
9:         $\beta_j \leftarrow \frac{\mathbf{r_j}^T \mathbf{r_j}}{\mathbf{r_{j-1}}^T \mathbf{r_{j-1}}}$;
10:        $\mathbf{d_j} \leftarrow \mathbf{r_j} + \beta_j \mathbf{d_{j-1}}$;
11:    end for
12: end procedure
```

Notice that the search direction (line 10) is modified adding a multiple of the previous search direction (the so-called "deflection") to the residual and β_j is chosen such that $\mathbf{d_j}$ and $\mathbf{d_{j-1}}$ are A-orthogonal (formally, $\mathbf{d_j}^T A\mathbf{d_{j-1}} = 0$).

Conversely, the next point is chosen in order to minimize the objective function $f(\mathbf{x_{j-1}} + \alpha_j \mathbf{d_{k-1}})$.

The update of the residual at line 9 follows from

$$\mathbf{r_j} = \mathbf{b} - A\mathbf{x_j} = \mathbf{b} - A \cdot (\mathbf{x_{j-1}} + \alpha_j \mathbf{d_{j-1}}) = \underbrace{\mathbf{b} - A\mathbf{x_{j-1}}}_{\mathbf{r_{j-1}}} - \alpha_j \mathbf{d_{j-1}}$$

Theorem 4.6.21. *The time complexity of Conjugate Gradient method is $O(n + n \cdot m)$ (that is less than GMRES) and the space complexity is constant (3 vectors).*

Notice that it holds that $\mathbf{r_j}, \mathbf{d_j}$, and $\mathbf{x_{j+1}} \in K_{j+1}$.

Theorem 4.6.22. $K_j(A,\mathbf{b}) = \mathit{span}(\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_j}) = \mathit{span}(\mathbf{d_0},\mathbf{d_1},\ldots,\mathbf{d_{j-1}}) = \mathit{span}(\mathbf{r_0},\mathbf{r_1},\ldots,\mathbf{r_{j-1}})$.

Theorem 4.6.23. *The residuals* $\{\mathbf{r_0},\mathbf{r_1},\ldots,\mathbf{r_{j-1}}\}$ *are orthogonal and the search directions are A-orthogonal. Formally,*

$$\mathbf{r_j}^T\mathbf{r_j} = \mathbf{d_i}^T A\mathbf{d_j} = 0,\ \forall\ i<j$$

Proof. By induction: Let us assume we proved the thesis $\mathbf{r_i}^T\mathbf{r_j} = 0$ for $j-1$ and $i = j-2,\ldots,0$.

We want to prove that

$$\mathbf{r_i}^T\mathbf{r_j} = \mathbf{r_i}^T\cdot(\mathbf{r_{j-1}} - \alpha_j A\mathbf{d_{j-1}}) = \mathbf{r_i}^T\mathbf{r_{j-1}} - \alpha_j\mathbf{r_i}^T A\mathbf{d_{j-1}} = 0$$

Case $i<j-1$**:**

- $\mathbf{r_i}^T\mathbf{r_{j-1}} = 0$ hold by induction
- $\alpha_j\mathbf{r_i}^T A\mathbf{d_{j-1}} = 0$ holds, because $\mathbf{r_i}\in K_{i+1}(A,\mathbf{b}) = \mathit{Span}(\mathbf{d_0},\ldots,\mathbf{d_i})$. Since $i<j-1$ and the vectors spanning the Krylov's space are orthogonal we have the equality with 0 (that proves also the A-orthogonality)

Case $i=j-1$**:**

$$\mathbf{r_i}^T\mathbf{r_j} = 0 \iff \alpha_j = \frac{\mathbf{r_{j-1}}^T\mathbf{r_{j-1}}}{\mathbf{r_{j-1}}^T A\mathbf{d_{j-1}}}$$

and this holds because, since $\mathbf{d_{j-1}} = \mathbf{r_{j-1}} + \beta_{j-1}\mathbf{d_{j-2}}$,

$$\mathbf{d_{j-1}}^T A\mathbf{d_{j-1}} = (\mathbf{r_{j-1}} + \beta_{j-1}\mathbf{d_{j-1}})^T A\mathbf{d_{j-1}} = \mathbf{r_{j-1}}^T A\mathbf{d_{j-1}} + \underbrace{\beta_{j-1}\mathbf{d_{j-1}}^T A\mathbf{d_{j-1}}}_{=0 \text{ ind.}} s$$

and by definition $\alpha_j = \frac{\mathbf{r_{j-1}}^T\mathbf{r_{j-1}}}{\mathbf{d_{j-1}}^T A\mathbf{d_{j-1}}}$

□

Notice that the base of residuals is orthogonal but not orthonormal, we need to re-scale it to obtain an orthonormal one, moreover, $\frac{1}{\|\mathbf{r_i}\|}\mathbf{r_i}$ coincides (up to a sign) with the $\mathbf{q_i}$ obtained with Arnoldi.

Theorem 4.6.24. *Let* $A\in M(m,n,\mathbb{R})$ *and let* $\mathbf{b}\in\mathbb{R}^m$*. The solution of the minimum problem*

$$\min_{\mathbf{x_n}\in K_n(A,\mathbf{b})} \|A\mathbf{x}-\mathbf{b}\|$$

is expressed as

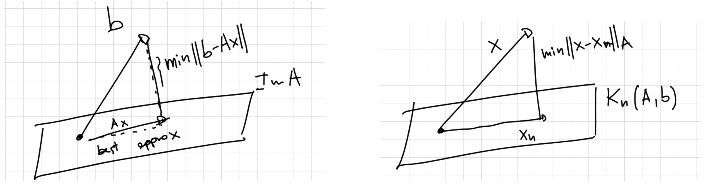
$$\mathbf{x_n} = Q_n\cdot {H_n}^{-1}\|\mathbf{b}\|\,\mathbf{e_1}$$

Proof. Saying that $\mathbf{r_n} = \mathbf{b} - A\mathbf{x_n}$ is orthogonal to all vectors of $K_n(A, \mathbf{b})$ is equivalent to requiring ${Q_n}^T \cdot (\mathbf{b} - A\mathbf{x_n}) = 0$

$$\begin{gathered}{Q_n}^T\mathbf{b} = {Q_n}^T A \underbrace{\mathbf{x_n}}_{Q_n\mathbf{y_n}} \\ \Updownarrow \\ \|\mathbf{b}\|\, \mathbf{e_1} = H_n\mathbf{y_n}\end{gathered}$$

□

In figure Figure 4.4 we can see a comparison between Arnoldi algorithm and the conjugate gradient.



(a) Arnoldi (b) Conjugate gradient

FIGURE 4.4: Traditional orthogonality (Arnoldi) leads to the minimization of the 2-norm, while in the conjugate gradient we impose A-orthogonality and we get a good approximation in several norms.

Theorem 4.6.25. *In Conjugate Gradient method,* $\mathbf{x_j}$ *is the best approximation in* $K_j(A, \mathbf{b})$ *of the exact (and unknown) solution* $\mathbf{x}_*$ *to* $A\mathbf{x}_* = \mathbf{b}$ *in* $K_j(A, \mathbf{b})$ *in the A-norm. Formally*

$$\|\mathbf{x_j} - \mathbf{x}_*\|_A^2 = (\mathbf{x_j} - \mathbf{x}_*)^T A(\mathbf{x_j} - \mathbf{x}_*)$$

Or, equivalently,

$$\mathbf{x_j} = \underset{\mathbf{x}\in K_j(A,\mathbf{b})}{\arg\min} \underbrace{\frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + \ \textit{constant}}_{f(\mathbf{x})}$$

Proof. Let us prove that any $\mathbf{z} \in K_j(A, \mathbf{b})$ is further from $\mathbf{x}_*$ then $\mathbf{x_j}$. We can write $\mathbf{z} \in K_j(A, \mathbf{b})$ as $\mathbf{x_j} + \mathbf{y}$, where $\mathbf{x_j}, \mathbf{y} \in K_j(A, \mathbf{b})$

$$\begin{aligned}(\mathbf{x}_* - \mathbf{z})^T A(\mathbf{x}_* - \mathbf{z}) &= (\mathbf{x}_* - \mathbf{x_j})^T A(\mathbf{x}_* - \mathbf{x_j}) + \underbrace{(\mathbf{x}_* - \mathbf{x_j})^T A\mathbf{y}}_{0} + \underbrace{\mathbf{y}^T A \cdot (\mathbf{x}_* - \mathbf{x_j})}_{0} + \underbrace{\mathbf{y}^T A\mathbf{y}}_{\geq 0} \\ &> (\mathbf{x}_* - \mathbf{x_j})^T A(\mathbf{x}_* - \mathbf{x_j})\end{aligned}$$

□

Convergence of Conjugate Gradient

Notice that the Conjugate Gradient method is a monotonic algorithm.

Theorem 4.6.26. *Let $\lambda_{\max}$, $\lambda_{\min}$ be the maximum/minimum eigenvalue of A; then, CG converges with rate*

$$\|\mathbf{x}_* - \mathbf{x_n}\| \leq \left(\underbrace{\frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}}}_{<1} \right)^n \|\mathbf{x}_* - \mathbf{x_0}\| .$$

We can rewrite it in terms of a more familiar quantity: for a positive definite matrix, eigenvalues and singular values coincide, hence

$$\frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}} = \frac{\sqrt{\sigma_1} - \sqrt{\sigma_m}}{\sqrt{\sigma_1} + \sqrt{\sigma_m}} = \frac{\sqrt{\frac{\sigma_1}{\sigma_m}} - 1}{\sqrt{\frac{\sigma_1}{\sigma_m}} + 1} = \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}.$$

For large values of $\kappa(A)$, this is approximately $1 - \frac{2}{\sqrt{\kappa(A)}}$, while if $\kappa(A) \approx 1$ the convergence speed is very high.

Theorem 4.6.27. *As we already said for GMRES, if A has at most n different eigenvalues, we can find a polynomial p of degree $\leq n-1$ such that $1-p(\lambda_i)\lambda_i = 0$ and therefore we have convergence in n steps.*

Proof.

$$\begin{aligned}
\min_{\mathbf{x_n} \in K_n(A,\mathbf{b})} \|\mathbf{x}_* - \mathbf{x_n}\|_A &= \min_{\substack{p(x) \\ \text{of degree } \leq n-1}} \|\mathbf{x}_* - p(A)\mathbf{b}\|_A \\
&= \|\mathbf{x}_* - p(A)A\mathbf{x}_*\|_A \\
&\leq \|I - p(A)A\| \cdot \|\mathbf{x}_*\|
\end{aligned}$$

But the quantity $\|I - p(A)A\|$ is small, since

$$\|I - p(A)A\| = \left\| V \cdot \begin{bmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_m p(\lambda_m) - 1 \end{bmatrix} \cdot \in V \right\|$$

□

Moreover, if the eigenvalues of A are ‘clustered’, one can construct polynomials such that $p(\lambda)$ attains the clusters, then fast convergence is implied.

Something on Matlab . . .

In Matlab it is possible to write a permutation of the rows of A that tries to put non-zero entries as close as possible to the diagonal: `p = syrcm(A);`