

# COMPUTATIONAL MATHEMATICS NUMERICAL METHODS

Based on prof. Federico Poloni's lectures

Gemma Martini  
August 28, 2021



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



# Contents

<b>1</b>	<b>20th of September 2018 — F. Poloni</b>	<b>5</b>
1.1	A brief journey in Linear Algebra . . . . .	5
1.1.1	Matrix multiplications: four flavors plus one of looking at it	10
1.2	Formal definitions . . . . .	11
1.3	Solving Linear Systems . . . . .	15
1.4	Orthogonality . . . . .	17
<b>2</b>	<b>26th of September 2018 — F. Poloni</b>	<b>19</b>
2.1	Orthogonality (II) . . . . .	19
2.2	Eigenvalues / Eigenvectors . . . . .	20
2.2.1	Eigenvector: what could possibly go wrong? . . . . .	22
<b>3</b>	<b>28th of September 2018 — F. Poloni</b>	<b>27</b>
3.1	Singular value decomposition (SVD) . . . . .	27
3.1.1	Properties of SVD . . . . .	29
<b>4</b>	<b>4th of October 2018 — F. Poloni</b>	<b>33</b>
<b>5</b>	<b>10th of October 2018 — F. Poloni</b>	<b>35</b>
<b>6</b>	<b>18th of October 2018 — F. Poloni</b>	<b>37</b>
6.1	Least squares problem . . . . .	37
6.1.1	Method of normal equations . . . . .	38
6.2	QR factorization . . . . .	39
<b>7</b>	<b>26th of October 2018 — F. Poloni</b>	<b>43</b>
7.1	How to construct a QR factorization . . . . .	43
7.1.1	Matlab implementation . . . . .	46
7.2	How to use the thin QR factorization to solve a least squares problem . . . . .	47
<b>8</b>	<b>7th of November 2018 — F. Poloni</b>	<b>49</b>
8.1	Least squares problem with SVD . . . . .	50
8.1.1	Behaviour in case of zeros as singular values . . . . .	53
8.2	Truncated SVD . . . . .	54

8.3	Tikhonov regularization / ridge regression . . . . .	54
<b>9</b>	<b>9th of November 2018 — F. Poloni</b>	<b>57</b>
9.1	Conditioning . . . . .	57
9.1.1	Conditioning of linear systems . . . . .	61
9.2	Condition number, SVD, and distance to singularity . . . . .	62
9.3	Conditioning of least squares problem . . . . .	63
<b>10</b>	<b>15th of November 2018 — F. Poloni</b>	<b>67</b>
10.1	Stability of algorithms . . . . .	67
10.2	Backward stability of QR factorization . . . . .	71
10.2.1	Stability of algorithms for least-squares problems . . . . .	72
10.2.2	A posteriori checks . . . . .	74
10.3	A posteriori check for Least Squares Problems . . . . .	75
<b>11</b>	<b>21st of November 2018 — F. Poloni</b>	<b>77</b>
11.1	Gaussian elimination and LU factorization . . . . .	78
11.1.1	Stability of LU . . . . .	81
11.1.2	Gaussian elimination on sparse matrices . . . . .	82
<b>12</b>	<b>23rd of November 2018 — F. Poloni</b>	<b>85</b>
12.1	Gaussian elimination on symmetric matrices . . . . .	85
12.2	Cholesky factorization . . . . .	88
12.3	Krylov subspace methods . . . . .	89
<b>13</b>	<b>29th of November 2018 — F. Poloni</b>	<b>91</b>
13.0.1	Naive idea . . . . .	91
13.0.2	Improvement . . . . .	91
13.1	Arnoldi algorithm . . . . .	92
<b>14</b>	<b>5th of December 2018 — F. Poloni</b>	<b>97</b>
14.1	Convergence of Arnoldi . . . . .	98
14.1.1	Better explanation . . . . .	98
<b>15</b>	<b>7th of December 2018 — F. Poloni</b>	<b>101</b>
<b>16</b>	<b>13th of December 2018 — F. Poloni</b>	<b>105</b>
16.1	Lanczos algorithm . . . . .	105

# Chapter 1

## 20th of September 2018 — F. Poloni

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 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 content has been written also 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.

During the course, some Matlab code will be covered. We do not plan to include a deep introduction to the Matlab language chiefly for two reasons:

1. there is already plenty of stuff online,
2. it is too time consuming to create one.

### 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 equa-

tions 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:

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

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 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.

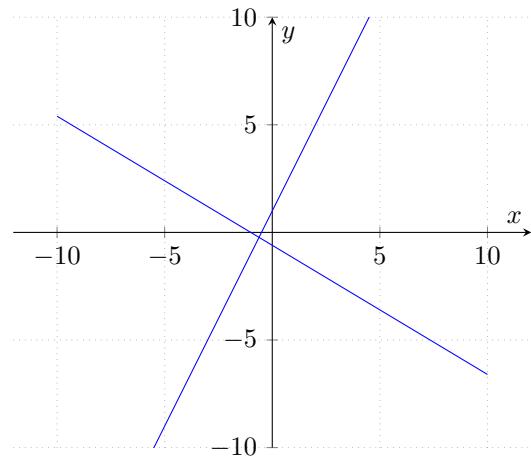


FIGURE 1.1: Linear system

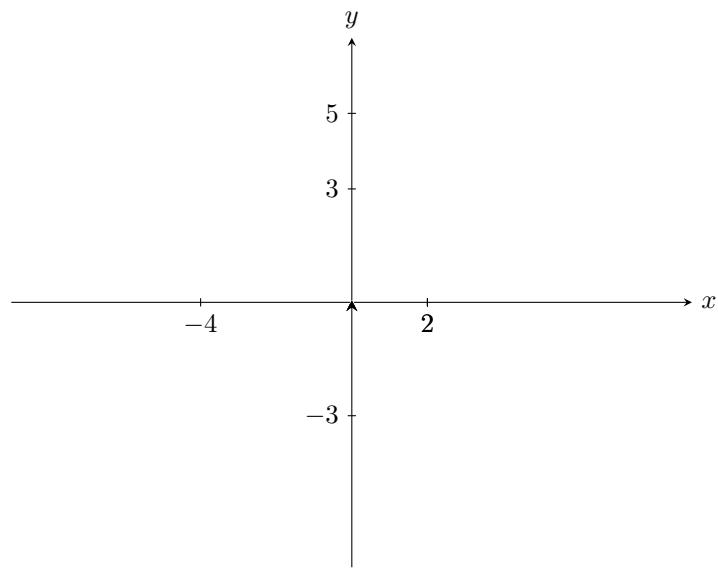


FIGURE 1.2: Linear system 2

**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  $Ax = \mathbf{b}$  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:

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

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:

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

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

$$x_1 [a_{11} \ a_{12} \ \dots \ a_{1n}] + \dots + x_n [a_{n1} \ a_{n2} \ \dots \ a_{nn}] = [b_1 \ b_2 \ \dots \ b_n]$$

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 that 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  $\mathbf{x}^T \mathbf{y} = \sum_{i=1}^n x_i y_i$ . 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  $A \in M(n, m, \mathbb{R})$  and  $\lambda \in \mathbb{R}$  we call the **scalar-matrix product** the following:

$$\lambda A = A\lambda = \begin{pmatrix} \lambda A_{11} & \lambda A_{12} & \dots & \lambda A_{1m} \\ \lambda A_{21} & \lambda A_{22} & \dots & \lambda A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda A_{n1} & \lambda A_{n2} & \dots & \lambda A_{nm} \end{pmatrix}$$

- **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.:  $A(B+C) = AB+AC$ ,  $A(BC) = (AB)C, \dots$

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  $A$  ( $\text{ker}(A)$ ): 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  $A_{ij}^T = A_{ji}$ . The **transpose of a product** (shoe-sock identity) is  $(AB)^T = B^TA^T$ . (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

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

$\mathbf{v}$  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

$$\mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{e}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

and each vector  $\mathbf{w} \in \mathbb{R}^4$  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  $\mathcal{B}_2$ :

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

and

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

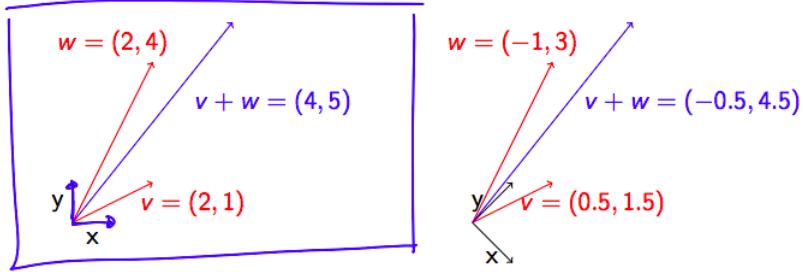


FIGURE 1.3: How a change of basis reflects on the space.

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

## 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:

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

Our goal is to *approximate* such vector  $\mathbf{x}$ , hence resulting in solving a minimum problem:

$$\min \|A\mathbf{x} - \mathbf{b}\|$$

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

**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  $\mathbf{A}=[1, 2, 3; 4, 5, 6]$ ;, where  $[1, 2, 3]$  is the first row of the matrix  $\mathbf{A}$ .

The transpose of a matrix or a vector is denoted by  $\mathbf{A}'$ .

The inverse of a square matrix is denoted by `inv(A)`.

If we are interested in only a part of our matrix  $\mathbf{A}$  we may write  $\mathbf{A}(1:2, 1:3)$  and obtain only the rows of  $\mathbf{A}$  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 1.3.2** (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.

$$\left( \begin{array}{ccc|cc|cc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \end{array} \right) \cdot \left( \begin{array}{ccc|cc|cc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \end{array} \right) = \left( \begin{array}{ccc|cc|cc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \end{array} \right)$$

$$\left( \begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{array} \right) \cdot \left( \begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{array} \right) + \left( \begin{array}{c} \times \\ \times \\ \times \end{array} \right) \cdot (\times \times \times) + \left( \begin{array}{cc} \times & \times \\ \times & \times \end{array} \right) \cdot \left( \begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{array} \right) = \left( \begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{array} \right)$$

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 1.3.1** (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 1.3.2** (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  $A_{ii}$  block;
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 2.2.1.

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 1.4.

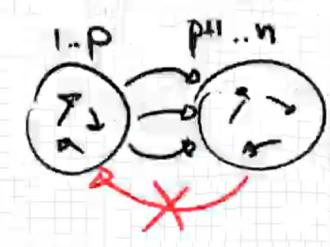


FIGURE 1.4: The adjacency matrix of a bipartite graph has 0s in its bottom left part (Matlab syntax  $A[p+1:n; 1:p]=0$ ), which means that the edges from a connected component and the other are in one direction only.

### ★ 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  $2 \times 2$  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.)

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

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

*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  $1 - \# \text{of } 0 \text{ entries}$ ;

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

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

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  $A \in O(n, \mathbb{R})$ ,  $\forall \mathbf{x} \in \mathbb{R}^n$  we have that  $\|A\mathbf{x}\| = \|\mathbf{x}\|$ .

*Proof.* 1.

$$\|A\mathbf{x}\| = \sqrt{(A\mathbf{x})^T A} = \sqrt{\mathbf{x}^T A^T A \mathbf{x}} = \sqrt{\mathbf{x}^T I_n \mathbf{x}} = \sqrt{\mathbf{x}^T \mathbf{x}} = \|\mathbf{x}\|$$

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

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

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

□

# Chapter 2

26th of September 2018 —  
F. Poloni

## 2.1 Orthogonality (II)

In the previous lecture we introduced some sufficient conditions for matrix orthogonality.

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

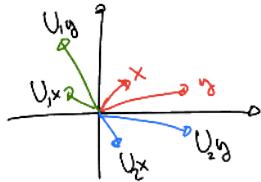


FIGURE 2.1: Matrix  $U_1$  represents a rotation, while  $U_2$  is a symmetry operation.

**Definition 2.1.1** (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,  $\langle \mathbf{x}, \mathbf{y} \rangle = 0$  and  $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$ .*

**Fact 2.1.1.** *Let us take  $U \in \mathbb{R}^{n,n}$ . Then its columns  $U^1, U^2, \dots, U^n$  are **orthonormal** and the same holds for its rows.*

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

and

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

**Fact 2.1.2.** *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.*  $(UV)^T \cdot (UV) = V^T U^T UV = V^T I_n V = V^T V = I_n$   $\square$

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

$$U_1 = (U^1 \quad U^2 \quad \dots \quad U^n) \in M(m, n, \mathbb{R}) \text{ where } m \geq n$$

the following fact may come in handy

**Fact 2.1.3.**  *$\forall U_1 \in M(m, n, \mathbb{R})$  where  $M \geq n$  and the columns of  $U_1$  are orthogonal  $\exists U_2$  s.t.  $(U_1 \quad U_2) \in O(m, \mathbb{R})$ .*

## 2.2 Eigenvalues / Eigenvectors

**Definition 2.2.1** (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**  $\lambda$ .*

**Theorem 2.2.1.** *Let  $A \in T(n, \mathbb{R})$  (real triangular matrix). The eigenvalues of  $A$  are the scalars on the diagonal.*

**Definition 2.2.2** (Diagonalizable matrix). *Some matrices  $A \in M(n, \mathbb{R})$  under some conditions can be decomposed as:*

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

$$A = V\Lambda V^{-1} = \begin{pmatrix} V^1 & V^2 & \dots & V^n \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 & & 0 & \\ & \lambda_2 & & \\ 0 & & \ddots & \\ & & & \lambda_n \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

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

**Fact 2.2.2.**  *$\forall V^i \in \mathbb{R}$  eigenvectors of  $A$  they are still eigenvector of the diagonalized form of  $A = V\Lambda V^{-1}$*

*Proof.*

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

$\square$

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

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

$$\boxed{\mathbf{v}_1} \cdot \boxed{\lambda_1} \cdot \boxed{\mathbf{w}_1^T} + \boxed{\mathbf{v}_2} \cdot \boxed{\lambda_2} \cdot \boxed{\mathbf{w}_2^T} + \dots + \boxed{\mathbf{v}_n} \cdot \boxed{\lambda_n} \cdot \boxed{\mathbf{w}_n^T}$$



### 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  $x$ . This process allow to scale a general vector  $x$ .



### Something on Matlab ...

Let  $A = [1 1; 1 1]$  and  $x = [1 1]$  such that  $\lambda_x = 2$  then  $A*x$  is equal to  $[2 2]$ , and  $A*A*x$  is equal to  $[4 4]$ ,

**Fact 2.2.3.** If  $A \in M(n, \mathbb{R})$  is diagonalizable (aka may be written as  $A = V\Lambda V^{-1}$ ) then  $A^k x = \sum_{i=1}^n \lambda_i^k \alpha_i V^i$ , for some  $\alpha_i \in \mathbb{R}$ .

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

$$x = V^1 \alpha_1 + V^2 \alpha_2 + \dots + V^n \alpha_n$$

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

- ALGEBRAIC VIEW POINT:

$$\begin{aligned} Ax &= A \cdot (V^1 \alpha_1 + V^2 \alpha_2 + \dots + V^n \alpha_n) \\ &= AV^1 \alpha_1 + AV^2 \alpha_2 + \dots + AV^n \alpha_n \\ &= \lambda_1 V^1 \alpha_1 + \lambda_2 V^2 \alpha_2 + \dots + \lambda_n V^n \alpha_n \\ &= V^1 (\lambda_1 \alpha_1) + V^2 (\lambda_2 \alpha_2) + \dots + V^n (\lambda_n \alpha_n) \end{aligned} \tag{2.2.1}$$

Then

$$\begin{aligned}
 A^2x &= A \cdot \left( V^1(\lambda_1\alpha_1) + V^2(\lambda_2\alpha_2) + \cdots + V^n(\lambda_n\alpha_n) \right) \\
 &= AV^1\lambda_1\alpha_1 + AV^2\lambda_2\alpha_2 + \cdots + AV^n\lambda_n\alpha_n \\
 &= \lambda_1^2V^1\alpha_1 + \lambda_2^2V^2\alpha_2 + \cdots + A\lambda_n^2V^n\alpha_n
 \end{aligned} \tag{2.2.2}$$

The thesis follows inductively.

- LINEAR ALGEBRA VIEW POINT:

$$\begin{aligned}
 A^k\mathbf{x} &= A \cdot A \cdot \dots \cdot A \cdot \mathbf{x} \\
 &= V\Lambda V^{-1}V\Lambda V^{-1}\dots V\Lambda V^{-1}\mathbf{x} \\
 &= V\Lambda^k V^{-1}\mathbf{x} \\
 &= V \begin{pmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{pmatrix} V^{-1}\mathbf{x} \\
 &= V \begin{pmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix}
 \end{aligned} \tag{2.2.3}$$

□

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.

### 2.2.1 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 2.2.4.** Let  $A \in M(n, \mathbb{R})$ . If  $|\lambda_i| < 1$  for all eigenvalues  $\lambda_i$  of  $A$  then  $\lim_{k \rightarrow \infty} A^k \mathbf{x} = 0$ ,  $\forall \mathbf{x} \in \mathbb{R}^n$ .

**Theorem 2.2.5.** 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 2.2.6.** Let  $A \in M(n, \mathbb{R})$  be a diagonalizable matrix and let

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

Let us now consider a reordering of  $V$ 's columns and apply the same permutation to the “diagonal vector” of  $\Lambda$  such that

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

$A$  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 2.2.7.** 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  $\tilde{V} = \begin{pmatrix} V^1 + V^2 & V^1 - V^2 & V^3 & \dots & V^n \end{pmatrix}$ .

**Theorem 2.2.8** (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  $[V, D] = \text{eig}(B)$ , matlab will always return an orthogonal matrix  $V$ .

**Definition 2.2.3** (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 2.2.1.** Let  $f_1\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = (x_1 \quad x_2) \begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3x_1^2 + 4x_1x_2 + 4x_2^2$

$$\text{and let } f_2\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = (x_1 \quad x_2) \begin{pmatrix} 3 & 2 \\ 2 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3x_1^2 + 4x_1x_2 - 4x_2^2$$

*For a graphic hint see Figure 2.2.*

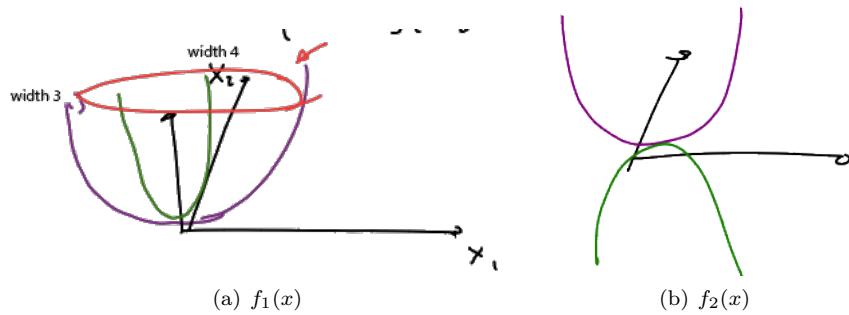


FIGURE 2.2: The plot of functions.

**Fact 2.2.9** (Variational characterization). *Let  $Q \in S(n, \mathbb{R})$  and let  $\mathbf{x} \in \mathbb{R}^n$ . Then*

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

where  $\lambda_{max}$  and  $\lambda_{min}$  are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.

*Proof.* EASY CASE WITH  $Q$  DIAGONAL:

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

It is obvious that this sum is bounded by:

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

The following holds:  $\lambda_{\min} \cdot (x_1^2 + x_2^2 + \dots + x_n^2) = \lambda_{\min} \cdot \mathbf{x}^T \mathbf{x} = \lambda_{\min} \cdot \|\mathbf{x}\|^2$   
 and, on the other hand,  $\lambda_{\max} \cdot (x_1^2 + x_2^2 + \dots + x_n^2) = \lambda_{\max} \cdot \mathbf{x}^T \mathbf{x} =$

$\lambda_{\max} \cdot \|x\|^2$  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.

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

where  $\stackrel{(1)}{=}$  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  $\|U^T \mathbf{x}\|^2 = \|\mathbf{y}\|^2$  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 2.2.10.** *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  $\lambda_{\max}$  and  $\lambda_{\min}$  are respectively the eigenvalue of maximum value and the eigenvalue of minimum value.

**Definition 2.2.4** (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 2.2.5** (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 2.2.11.** *Let  $Q \in S(n, \mathbb{R})$ . The following holds*

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

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  $\geq 0$

( $\Rightarrow$ )  $\forall \mathbf{v}_i$  eigenvector of  $Q$   $0 \leq \mathbf{v}_i^T A \mathbf{v}_i = \mathbf{v}_i^T \cdot (\lambda_i \mathbf{v}_i) = \lambda_i \mathbf{v}_i^T \mathbf{v}_i = \lambda_i \|\mathbf{v}_i\|^2 \Rightarrow \lambda_i \geq 0$

□

**Corollary 2.2.12.** Let  $Q \in S(n, \mathbb{R})$  such that  $Q \succeq 0$ . The following holds:

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

**Fact 2.2.13.** 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 2.2.14.** 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)

---

\*Often denoted with  $A^*$  or  $A^H$  and called **Hermitian matrix**.

# Chapter 3

28th of September 2018 —  
F. Poloni

## 3.1 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 3.1.1** (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 3.1.2** (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.*

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

Where  $\sigma_i$  are called **singular values** and they are sorted such that:

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$$

General fact on singular values:

- Singular values  $\neq$  eigenvalues

- They are always positive and usually more spread apart than the eigenvalues.

$$\sigma_1 \geq |\lambda_1| \text{ and } |\lambda_m| \geq \sigma_m$$

$\sigma_i$  is larger than the largest eigenvalue of a matrix  $A$  and  $\sigma_m$  is smaller than the smallest eigenvalue of a matrix  $A$ .

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

**Definition 3.1.3** (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  $\Sigma$  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 3.1.4** (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.

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



### Something on Matlab ...

In Matlab the SVD decomposition is obtained through the command `svd(A)`, which return value is made of the three matrices  $U, \Sigma, V$ .

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.

- $[U, S, V] = \text{svd}(A, 0)$  (thin) costs  $O(mn^2)$  ops for  $A \in M(m, n, \mathbb{R})$  or  $A \in M(n, m, \mathbb{R})$  with  $m \geq n$
- $[U, S, V] = \text{svd}(A)$  (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).

### 3.1.1 Properties of SVD

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

**Definition 3.1.5** (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 3.1.1.** *A matrix  $A \in M(n, \mathbb{R})$  has rank  $r$  iff all its eigenvalues starting from the  $r+1$ -th are 0, formally iff  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_n = 0$ .*

Thanks to Property 3.1.1, we can somehow talk about an “even thinner” SVD, where all the 0s in the bottom right part of the matrix  $\Sigma$ , 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  $\Sigma$  can be found below.

$$\Sigma = \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_r & \\ \hline & & & & 0 \\ & & & & \ddots \\ \hline & & & & 0 \\ & & & & \hline \mathbf{0} & & & & \end{pmatrix}$$

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

**Fact 3.1.2.** *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  $U_1, U_2, \dots, U_r$ , hence  $\text{rk}(A) = r$ . Moreover,  $\ker(A) = \text{span}(V_{r+1}, V_{r+2}, \dots, V_n)$ , since  $V$  is orthogonal.*

*Proof.*

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

□

**Definition 3.1.6** (Matrix norm). *Let  $A \in M(m, n, \mathbb{R})$ . We define the **matrix norm** of  $A$  as*

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

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 3.1.3.** 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  $A = 0$ );
- $\|\alpha A\| = |\alpha| \|A\|, \forall \alpha \in \mathbb{R}$ ;
- $\|A + B\| \leq \|A\| + \|B\|$ ;
- $\|AB\| \leq \|A\| \|B\|$ ;
- $\|Ax\| \leq \|A\| \|\mathbf{x}\|, \forall \mathbf{x} \in \mathbb{R}^n$ .

**Fact 3.1.4.** Let  $A \in (n, \mathbb{R})$  and let  $U \in O(m, n, \mathbb{R})$  orthogonal, in the case of 2-norm  $\|A\|_2 \stackrel{(1)}{=} \|AU\|_2 \stackrel{(2)}{=} \|UA\|_2$ .

*Proof.*

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

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

$$\begin{aligned} \text{norm } AU_2 &= \max_{\mathbf{x} \in R^n, \mathbf{x} \neq \mathbf{0}} \frac{\|AU\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \\ &\stackrel{(2)}{=} \max_{\mathbf{y} \in R^m, \mathbf{y} \neq \mathbf{0}} \frac{\|A\mathbf{y}\|_2}{\|\mathbf{y}\|_2} = \|A\|_2 \end{aligned}$$

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

**Definition 3.1.7** (Frobenius norm). Let  $A \in M(n, m, \mathbb{R})$ , we term **Frobenius norm** of  $A$   $\|A\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m (A)_{ij}^2}$ .

Notice that all the properties enumerated in Property 3.1.3 hold for the Frobenius norm as well.

**Fact 3.1.5.** 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.  $\|A\|_F = \|\Sigma\|_F = \sum_{i=1}^{\tilde{n}} \sigma_i^2$ , where  $\tilde{n} = \min n, m$

*Proof.* 1. The first equality follows from Proposition 3.1.4, while the second is proved as follows:

$$\begin{aligned}
 \|\Sigma\|_2 &= \max_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq \mathbf{0}} \frac{\|\Sigma \mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq \mathbf{0}} \frac{\left\| \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ 0 & & & \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \right\|_2}{\left\| \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \right\|_2} \\
 &= \max_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq \mathbf{0}} \left\| \begin{pmatrix} \sigma_1 x_1 \\ \sigma_2 x_2 \\ \vdots \\ \sigma_n x_n \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right\|_2 = \frac{\sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2 + \cdots + (\sigma_n x_n)^2}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}} \\
 &\leq \frac{\sqrt{(\sigma_1 x_1)^2 + (\sigma_1 x_2)^2 + \cdots + (\sigma_1 x_n)^2}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}} = \sqrt{\sigma_1^2} \cdot \frac{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}} = \sigma_1
 \end{aligned} \tag{3.1.1}$$

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.

□

**Theorem 3.1.6** (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  $\min_{rk(X) \leq k} \|A - X\|_F$  is given by the truncated SVD:*

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

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

**Fact 3.1.7.** *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  $\sigma_i$  is 0, hence the smaller (namely  $\sigma_n$ ) is not 0.

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

Where  $\stackrel{(1)}{=}$  follows from the orthogonality 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:

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

□

## Chapter 4

# 4th of October 2018 — F. Poloni

This lecture is about practical usage of the singular value decomposition and takes place almost wholly on Matlab.

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 4.1. 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)`

**Definition 4.0.1** (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.*

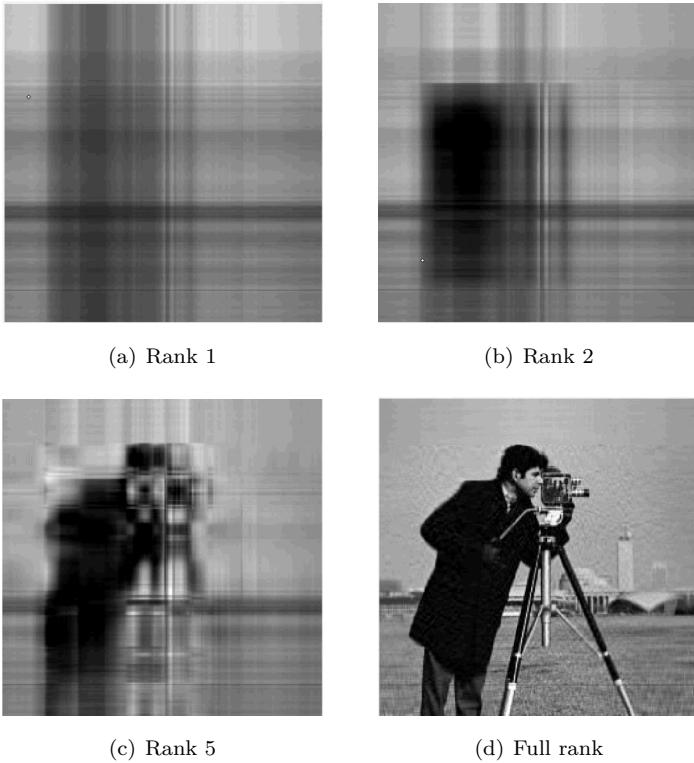


FIGURE 4.1: How the approximation of a matrix changes with respect to the different ranks.

# Chapter 5

## 10th of October 2018 — F. Poloni

This lecture has the goal of introducing the concept of linear combinations.

**Definition 5.0.1** (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  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \in \mathbb{R}^n$  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.

The task of finding such vectors is called **solving a linear system** and it is formally written as  $A\mathbf{x} = \mathbf{b}$ .

**Theorem 5.0.1.** *Let  $A \in M(n, m)$  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.*

We are interested in finding approximate solutions of such systems, where the proximity to the target is expressed in terms as  $\|A\mathbf{x} - \mathbf{b}\|$  that should be close to zero. A geometric intuition is displayed in Figure 5.1.



### Something on Matlab ...

Matlab provides syntactic sugar to solve linear systems.

Before introducing such syntax let us just notice the following  $5 \setminus 2$   
 $(= 2/5) \neq 5/2$ .

The syntax to solve  $A\mathbf{x} = \mathbf{b}$  is  $A \setminus \mathbf{b}$ , 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.

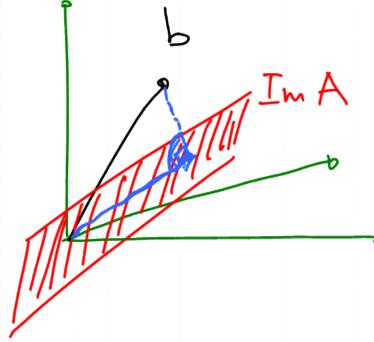


FIGURE 5.1: In this case the image of the matrix  $A$  (in red) does not contain  $\mathbf{b}$  and the best one can do is to obtain a projection of  $\mathbf{b}$  in the plane  $Im(A)$  (drawn in blue).

**Definition 5.0.2** (Least square problem). *Let  $A \in M(n, m, \mathbb{R})$  and let  $\mathbf{b} \in \mathbb{R}^n$ , we term **linearly square problem** the task of computing  $\min_{\mathbf{x} \in \mathbb{R}^m} \|A\mathbf{x} - \mathbf{b}\|_2$ .*



### Something on Matlab ...

In Matlab the syntax `.func` means that function func should be performed entry by entry of the non-scalar variable.

An example of a practical least square problem may be predicting the salary of NBA players, assuming that the income is obtained as a linear combination of some features.

**Definition 5.0.3** (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,  $\text{rk}(A) = n$  or alternatively  $\nexists \mathbf{z} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$  such that  $A\mathbf{z} = \mathbf{0}$ .*

**Fact 5.0.2.** *Let  $A \in M(n, m, \mathbb{R})$ , the least square problem  $\min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|$  has a unique solution iff  $A$  has full column rank.*

**Theorem 5.0.3.** *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  $\iff \|\mathbf{Az}\| \neq 0, \forall \mathbf{z} \in \mathbb{R}^m \setminus \{\mathbf{0}\} \iff \|\mathbf{Az}\|^2 \neq 0, \forall \mathbf{z} \in \mathbb{R}^m \setminus \{\mathbf{0}\} \iff 0 = (\mathbf{Az})^T \mathbf{Az} = \mathbf{z}^T A^T A \mathbf{z}$   $\square$

# Chapter 6

## 18th of October 2018 — F. Poloni

### 6.1 Least squares problem

**Fact 6.1.1.** 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;
- $\text{Ker}(A) = \{0\}$ .

**Fact 6.1.2.** Let  $A \in \mathcal{M}(m, n, \mathbb{R})$ , s.t.  $Q = A^T A \in M(n, \mathbb{R})$  is **positive semidefinite**. The quadratic function  $f(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x} - \mathbf{q}^T \mathbf{x} + \mathbf{b}^T \mathbf{b}$  is **strongly (or strictly) convex**. In other words,  $f(\mathbf{x})$  has a **unique minimum**.

**Fact 6.1.3.** Let  $A \in \mathcal{M}(m, n, \mathbb{R})$ , s.t.  $Q = A^T A \in M(n, \mathbb{R})$  is **positive semidefinite**. The quadratic function  $f(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x} - \mathbf{q}^T \mathbf{x} + \mathbf{b}^T \mathbf{b}$  has gradient

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

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

$$\begin{aligned} f(\mathbf{x} + \mathbf{h}) &= (\mathbf{x} + \mathbf{h})^T A^T A (\mathbf{x} + \mathbf{h}) - 2\mathbf{b}^T A (\mathbf{x} + \mathbf{h}) + \mathbf{b}^T \mathbf{b} \\ &= \mathbf{x}^T A^T A \mathbf{x} + \mathbf{x}^T A^T A \mathbf{h} + \mathbf{h}^T A^T A \mathbf{x} + \mathbf{h}^T A^T A \mathbf{h} - 2\mathbf{b}^T A \mathbf{x} - 2\mathbf{b}^T A \mathbf{h} + \mathbf{b}^T \mathbf{b} \\ &= f(\mathbf{x}) + (2\mathbf{x}^T A^T A \mathbf{h} - 2\mathbf{b}^T A \mathbf{h}) + o(\|\mathbf{h}\|) \\ &= f(\mathbf{x}) + (2A^T A \mathbf{x} - 2A^T \mathbf{b})^T \mathbf{h} + o(\|\mathbf{h}\|) \end{aligned} \tag{6.1.1}$$

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

□

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

**Fact 6.1.4.** Let  $A \in \mathcal{M}(m, n, \mathbb{R})$ , s.t.  $Q = A^T A \in M(n, \mathbb{R})$  is **positive semidefinite**. The quadratic function  $f(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x} - \mathbf{q}^T \mathbf{x} + \mathbf{b}^T \mathbf{b}$  has a unique minimum in

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

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

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:  $A^T A$  can be written as  $A^T A = R^T R$ , where  $R$  is a square, upper triangular matrix.

Why do we need factorization method? Let's compute complexity:

- $A^T A \rightarrow 2mn^2$ , where  $m > n$ ;
- $A^T \mathbf{b} \rightarrow 2mn$ ;
- Solving  $A^T A \mathbf{x} = A^T \mathbf{b}$  with Gaussian elimination has a computational complexity of  $\frac{2}{3}n^3$ ;
- Cholesky factorization  $A^T A = R^T R$  which has a cost of  $\frac{1}{3}n^3$

### 6.1.1 Method of normal equations

This method solves least squares problem and takes his name from the fact that “normal” means orthogonal. The key idea is using symmetry to skip half of the entries of  $A^T A$

If  $A\mathbf{x} = \mathbf{b}$  cannot be solved, since  $A$  is tall and thin, we can multiply on both sides by  $A^T$  and try again, since the matrix is square now.

#### 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

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

**Fact 6.1.5.** 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  $\mathbf{v} \in \text{span}(A)$ . Formally,

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

*Proof.*  $\mathbf{v}^T(A^T A\mathbf{x} - A^T \mathbf{b}) = 0$ , see ??.

□

**Definition 6.1.1** (Moore-Penrose pseudoinverse). Let  $A$  be a matrix in  $M(n, m, \mathbb{R})$ .

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

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

- 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  $A$ :  $A A^\dagger = I$

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.

## 6.2 QR factorization

**Theorem 6.2.1** (QR factorization).  $\forall A \in M(m, \mathbb{R})$ ,  $\exists Q \in O(m, \mathbb{R})$ ,  $\exists R \in \mathcal{T}(m, \mathbb{R})$  upper triangular such that  $A = QR$

**Fact 6.2.2.** Let  $A \in M(n, m, \mathbb{R})$ ,  $\mathbf{b} \in \mathbb{R}^n$ . A solution to the least squares problem

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

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  $\mathbf{y} = Q^T \mathbf{b}$

3. and then  $\mathbf{x} = R^{-1}\mathbf{c}$

And the computational cost is expressed as"

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

**Definition 6.2.1** (Householder 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  $\text{vect } \mathbf{v}_u = \frac{1}{\|\mathbf{v}\|} \mathbf{v}$ .

**Lemma 6.2.3.** Householder reflectors are orthogonal.

*Proof.*

$$\begin{aligned} HH^T &= \left( I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T \right) \cdot \left( I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T \right) \\ &= I \cdot I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T I - I \cdot \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T + \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T \cdot \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T \\ &= I - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T - \frac{2}{\|\mathbf{v}\|^2} \cdot \mathbf{v} \mathbf{v}^T + \frac{4}{\|\mathbf{v}\|^4} \cdot \mathbf{v} \mathbf{v}^T \mathbf{v} \mathbf{v}^T \\ &= I - \frac{4}{\|\mathbf{v}\|^2} \mathbf{v} \mathbf{v}^T + \frac{4}{\|\mathbf{v}\|^4} \mathbf{v} \|\mathbf{v}\|^2 \mathbf{v}^T \\ &= I \end{aligned} \tag{6.2.1}$$

□

**Corollary 6.2.4.** 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 ??

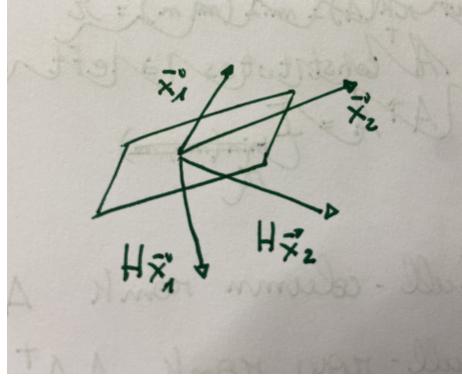


FIGURE 6.1: Reflection with respect to the hyperplane.

**Example 6.2.1.** Let us be given the problem of computing the product  $H\mathbf{x}$  for  $H \in M(m, \mathbb{R})$  and  $\mathbf{x} \in \mathbb{R}^m$ .

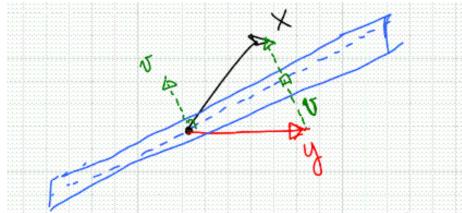
$$H\mathbf{x} = (I - \frac{2}{\|\mathbf{v}\|^2} \mathbf{v}\mathbf{v}^T)\mathbf{x} = \mathbf{x} - \frac{2}{\|\mathbf{v}\|^2} \mathbf{v}(\mathbf{v}^T \mathbf{x})$$

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 6.2).

**Lemma 6.2.5.**  $\forall \mathbf{x}, \mathbf{y}$  s.t  $\|\mathbf{x}\| = \|\mathbf{y}\| \exists H$  s.t.  $H\mathbf{x} = \mathbf{y}$ , choosing  $\mathbf{v} = \mathbf{x} - \mathbf{y}$ .

FIGURE 6.2:  $H$  is the hyperplane that “bisects” the angle between  $\mathbf{x}$  and  $\mathbf{y}$ .

What happens if  $y = \begin{pmatrix} \|\mathbf{x}\| \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ ?

$y = H^T x$ , actually  $H^T = H^{-1} = H$  Let's map  $x$  to  $y$ :

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

where  $s = \|x\|$

#### Matlab syntax for functions

`function [v,s] = householder_vector(x)`, where v and s are the returned values and x is the argument.

# Chapter 7

## 26th of October 2018 — F. Poloni

### 7.1 How to construct a QR factorization

In the previous lecture we introduced the  $QR$  factorization and we defined what an Householder reflector is.

At the end of the lecture we gave a first MatLab implementation of `householder_vector`:

---

ALGORITHM 7.1.1 Householder vector Matlab implementation.

---

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

---

What's the problem of this algorithm? That the subtraction may create a problem with machine numbers, if  $s$  and  $\|x\|$  are very close. If we take  $\|x\| = -s$  the subtraction becomes an 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  $s = -\text{sign}(x(1)) * \text{norm}(x)$ .

**Example 7.1.1.** Given  $A = \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix} \in \mathcal{M}(m, m, \mathbb{R})$ , where  $m = 5$ , we would like to calculate the  $QR$  factorization of  $A$ .



FIGURE 7.1: If  $x$  is oriented as in the plot it's better if we choose  $-\|x\|e_1$  verse, since it's opposite to  $x$ .

STEP 1 : construct a Householder matrix that sends  $A(:, 1)$  (first column of  $A$ )

$$\text{to a multiple of } e_1. \text{ Then we have } H_1 A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}$$

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

$$\begin{aligned} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & & & & \\ 0 & & H_2 & & \\ 0 & & & & \\ 0 & & & & \end{pmatrix} \cdot H_1 A &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & & & & \\ 0 & & H_2 & & \\ 0 & & & & \\ 0 & & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix} \\ &= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \end{aligned} \tag{7.1.1}$$

And we denote  $Q_2 = \begin{pmatrix} I_{1 \times 1} & 0 \\ 0 & H_2 \end{pmatrix}$ ,  $Q_1 = H_1$ ;

STEP 3 : take  $H_3 \in \mathcal{M}(m-2, m-2, \mathbb{R})$  such that  $H_3 A(3 : \text{end}, 3) = \begin{pmatrix} \times \\ 0 \\ 0 \end{pmatrix}$  and

we compute:

$$\begin{aligned}
 Q_3 \cdot (Q_2 Q_1 A) &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & & & \\ 0 & 0 & & H_3 & \\ 0 & 0 & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \\
 &= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix}
 \end{aligned} \tag{7.1.2}$$

$$\text{So, } Q_3 = \begin{pmatrix} I_{2 \times 2} & 0 \\ 0 & H_3 \end{pmatrix};$$

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

$$\begin{aligned}
 Q_4 \cdot (Q_3 Q_2 Q_1 A) &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & & \\ 0 & 0 & & H_4 & \\ 0 & 0 & & & \end{pmatrix} \cdot \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \\
 &= \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}
 \end{aligned} \tag{7.1.3}$$

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

In the end, since  $Q_i$  is an orthogonal matrix 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 7.1.1** (Product of block matrices). Let  $I \in \mathcal{M}(k, k, \mathbb{R})$ , let  $H_i \in \mathcal{M}(m-k, m-k, \mathbb{R})$  and let  $B_i \in \mathcal{M}(k, k, \mathbb{R})$ ,  $C_i \in \mathcal{M}(k, m-k, \mathbb{R})$  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.  $\square$

### 7.1.1 Matlab implementation

---

ALGORITHM 7.1.2 First implementation of QR factorization.

---

```

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

---

**Fact 7.1.2.** *The cost of this implementation when  $A$  is a square matrix is  $O(n^3 + (n-1)^3 + \dots + 1^3)$ . If  $A$  is a rectangular matrix, then the computational complexity is  $O(m \cdot n^2 + (m-1) \cdot (n-1)^2 + \dots + (m-n+1)^3)$ .*

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

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

---

ALGORITHM 7.1.3 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
```

---

Let's suppose that  $A$  is square matrix, partitioned as:  $A = \begin{pmatrix} A_1 & A_2 \\ Q_1 & Q_2 \end{pmatrix} \cdot \begin{pmatrix} R_{1,1} & R_{1,2} \\ 0 & R_{2,2} \end{pmatrix}$  Then we can recover the factorization of  $A_1$  from the factorization of  $A$ , since  $A_1 = Q \cdot R_{11}$ .

**Fact 7.1.3** (Thin QR factorization). *we may replace  $Q \in \mathcal{M}(m, m, \mathbb{R})$  and  $R \in \mathcal{M}(m, m, \mathbb{R})$  with  $Q_1 \in \mathcal{M}(m, n, \mathbb{R})$  and  $R_1 \in \mathcal{M}(n, n, \mathbb{R})$  and the same factorization holds:  $A = QR = Q_1 R_1$ . This is called **thin QR factorization**.*

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

In order to save space we may work in the following way:

$$Q \cdot B = \begin{pmatrix} 1 & \times & \cdots & \times \\ 0 & & & \\ \vdots & I - 2V_1 V_1^T & & \\ 0 & & & \\ 0 & & & \end{pmatrix} \cdot \begin{pmatrix} 1 & \times & \times & \cdots & \times \\ 0 & 1 & \times & \cdots & \times \\ \vdots & 0 & & I - 2V_2 V_2^T & \\ 0 & \vdots & & & \\ 0 & 0 & & & \end{pmatrix} \cdot \dots \cdot \begin{pmatrix} 1 & \times & \times & \cdots & \times \\ 0 & 1 & \times & \cdots & \times \\ \vdots & 0 & 1 & & \\ 0 & \vdots & 0 & I - 2V_n V_n^T & \\ 0 & 0 & 0 & & \end{pmatrix} \cdot B.$$

#### Fun fact

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

## 7.2 How to use the thin QR factorization to solve a least squares problem

We would like to solve  $\|Ax - b\|$   $\forall A \in \mathcal{M}(m, n, \mathbb{R})$  and  $\forall B \in \mathbb{R}^n$  where  $m > n$  (a.k.a  $A$  is a tall, thin matrix), through the  $QR$  factorization. We would like to solve  $\min \|Ax - b\|$  through the  $QR$  factorization.

We may write first the  $QR$  factorization of  $A$ , so  $\forall A \in \mathcal{M}(m, n, \mathbb{R})$ ,  $\exists Q \in \mathcal{M}(m, m, \mathbb{R})$ ,  $\exists R \in \mathcal{M}(m, n, \mathbb{R})$  such that  $A = QR$ , where  $Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$  and  $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$ .

Then,

$$\begin{aligned}
 \|Ax - b\| &= \|Q^T(Ax - b)\| = \|Q^T QRx - Q^T b\| \\
 &= \|Rx - Q^T b\| = \left\| \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x - \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} b \right\| \\
 &= \left\| \begin{pmatrix} R_1 x - Q_1^T b \\ Q_2^T b \end{pmatrix} \right\|
 \end{aligned} \tag{7.2.1}$$

How can we pick  $x$  to minimize the norm of  $Ax - b$ ?

Can we choose  $x$  such that  $R_1 x - Q_1^T b = 0$ ? Yes, we can, since this is a linear square system, so  $x = R_1^{-1} Q_1^T b$

$$\|Ax - b\| = \|Q_2^T b\|$$

We used the fact that  $R_1$  is invertible, but is it always true that  $R_1$  is invertible?

**Lemma 7.2.1.**  $R_1$  is invertible  $\Leftrightarrow A$  has full column rank.

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

Let's compute  $QR^T QR = R^T Q^T QR = R^T R = (R_1^T \ 0) \cdot \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = R_1^T \cdot R_1$ .

So,  $A^T A$  is invertible  $\Leftrightarrow R_1$  is invertible.  $\square$

### Note

$R_1^T R_1$  is the Cholesky factorization of  $A^T A$ .

The computational complexity is asymptotically equal to the one of computing the  $QR$  factorization, since the other operations are cheaper (the product  $Q_1^T b$  costs  $O(mn)$  and solving the triangular linear system by back-substitution costs  $O(n^2)$ ).



## Chapter 8

# 7th of November 2018 — F.Poloni

### 8.1 Least squares problem with SVD

 Do you recall?

**Tall thin SVD:** A matrix  $A$  can be written as  $A = USV^T$ , where  $U$  is orthogonal,  $S$  is a diagonal matrix and  $V$  is orthogonal as well. In the case of a tall, thin  $A$  the decomposition has the following shape:

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

where

$$\Sigma = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix}$$

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

We would like to see how we can solve a least squares problem through *SVD*:

$$\begin{aligned}
 \|Ax - b\| &= \|USV^T x - b\| && \leftarrow \text{Def. of SVD} \\
 &= \|U^T(USV^T x - b)\| && \leftarrow U^T \text{ is orthogonal} \\
 &= \|SV^T x - U^T b\| && \leftarrow \text{Distributivity + orthogonality} \\
 &= \|Sy - U^T b\| && \leftarrow y = V^T x \\
 &= \left\| \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ & & & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} - \begin{pmatrix} U^{1T} b \\ U^{2T} b \\ \vdots \\ U^{nT} b \\ U^{n+1T} b \\ \vdots \\ U^{mT} b \end{pmatrix} \right\| && (8.1.1) \\
 &= \left\| \begin{pmatrix} \sigma_1 y_1 - U^{1T} b \\ \sigma_2 y_2 - U^{2T} b \\ \vdots \\ \sigma_n y_n - U^{nT} b \\ \sigma_{n+1} y_{n+1} - U^{n+1T} b \\ \vdots \\ \sigma_m y_m - U^{mT} b \end{pmatrix} \right\|
 \end{aligned}$$

Where the first  $n$  rows may be assigned to 0 iff  $y_i = -\frac{U^{iT} b}{\sigma_i}$  (if  $\sigma_i \neq 0 \forall i$ ), while the latter  $m - n$  do not depend on  $y$ . This process produces a solution  $y$ , but

the variable change may be inverted, so

$$\begin{aligned}
 x &= Vy && \leftarrow \text{Orthogonality of } V \\
 &= V^1 y_1 + V^2 y_2 + \cdots + V^n y_n \\
 &= V^1 \frac{1}{\sigma_1} U^{1T} b + V^2 \frac{1}{\sigma_2} U^{2T} b + \cdots + V^n \frac{1}{\sigma_n} U^{nT} b \\
 &= V \cdot \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} \cdot U^T \cdot b \tag{8.1.2} \\
 &= V \cdot \begin{pmatrix} \frac{1}{\sigma_1} & & & \\ & \frac{1}{\sigma_2} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n} \end{pmatrix} \cdot U_1^T \cdot b
 \end{aligned}$$

Which depends only on the tall, thin SVD.

**Fact 8.1.1.** *The  $\sigma_i$  are different from 0 iff  $A$  has full column rank.*

*Proof.*  $A$  has full column rank

$$\begin{aligned}
 &\Updownarrow \\
 A^T A \text{ is invertible} &\Updownarrow \\
 &\Updownarrow \\
 (USV^T)^T (USV^T) = VS^T U^T USV^T = V \cdot \begin{pmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_n^2 \end{pmatrix} \cdot V^T \text{ is} \\
 &\text{invertible} \\
 &\Updownarrow \\
 \forall i \ \sigma_i \neq 0
 \end{aligned}$$

□

**Observation 8.1.1.** *This lemma also proves that the factorization is also a QR factorization.*

#### Note on Matlab syntax

`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.

### 8.1.1 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 ordered 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.

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

$$\left\| \begin{pmatrix} \sigma_1 y_1 - U^{1T} b \\ \sigma_2 y_2 - U^{2T} b \\ \vdots \\ \sigma_n y_n - U^{nT} b \\ 0 \cdot y_{n+1} - U^{n+1T} b \\ \vdots \\ 0 \cdot y_m - U^{mT} b \end{pmatrix} \right\|.$$

We also recall the following  $\|Ax - b\| =$

$$\begin{pmatrix} -\frac{U^{1T} b}{\sigma_1} \\ -\frac{U^{2T} b}{\sigma_2} \\ \vdots \\ -\frac{U^{rT} b}{\sigma_r} \\ * \end{pmatrix}$$

No matter what value we choose for  $y_{r+1} \dots y_n$ , the value doesn't change since it's multiplied by 0. Therefore we get infinite solutions of the form  $y =$

We would like to make the solution unique, so we can modify the problem:

- taking the value that minimizes the norm:

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

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

•

The solution of  $P2$  is given by

$$V y = \begin{pmatrix} V^1 & V^2 & \cdots & V^n \end{pmatrix} \cdot \begin{pmatrix} -\frac{U^{1T} b}{\sigma_1} \\ -\frac{U^{2T} b}{\sigma_2} \\ \vdots \\ -\frac{U^{rT} b}{\sigma_r} \\ 0 \end{pmatrix} = V^{1T} \frac{1}{\sigma_1} U^{1T} b + \cdots + V^{rT} \frac{1}{\sigma_r} U^{rT} b$$

What happens when working with machine precision? Let's make an example where  $r = n - 1$ , so only the last singular value is 0. If the check of  $\sigma_n = 0$  fails,  $\frac{1}{\sigma_n}$  becomes very big. A way to circumvent this problem is to find the linear dependencies between the columns, so that the algorithm works correctly.

## 8.2 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 and noise. Note, though, that in the sum  $\sum_{i=1}^n V^i \frac{U^{iT} b}{\sigma_i}$  the small singular values may have a large impact, because  $\sigma_i$  is in the denominator.

We can modify the solution to cope with real world data problems:

$$x = \sum_{i=1}^n V^i \frac{U^{iT} b}{\sigma_i} \longrightarrow x_{trunc} = \sum_{i=1}^k V^i \frac{U^{iT} b}{\sigma_i}$$

for a certain  $k$ , ignoring small singular values.

Another way to modify the problem is the following.

## 8.3 Tikhonov regularization / ridge regression

The Tikhonov regularization is a smoother version of truncated SVD.

$$x_{Tik} = \arg \min_{x \in \mathbb{R}^n} \|Ax - b\|^2 + \alpha^2 \|x\|^2$$

**Fact 8.3.1.** *The Tikhonov regularization is equivalent to*

$$x_{Tik} = \arg \min_{x \in \mathbb{R}^n} \left\| \begin{pmatrix} A \\ \alpha I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|^2 \quad (8.3.1)$$

We show that the two objective functions coincide.

*Proof.*

$$\begin{aligned}
 \left\| \begin{pmatrix} A \\ \alpha I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|^2 &= \left\| \begin{pmatrix} Ax \\ \alpha x \end{pmatrix} - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|^2 \\
 &= \left\| \begin{pmatrix} Ax - b \\ \alpha x \end{pmatrix} \right\|^2 \\
 &= \|Ax - b\|^2 + \|\alpha x\|^2 \\
 &= \|Ax - b\|^2 + \alpha \cdot \|x\|^2
 \end{aligned} \tag{8.3.2}$$

□

**Fact 8.3.2.** *The solution of the Tikhonov regularization is given by the formula  $x_{Tik} = (A^T A + \alpha^2 I)^{-1} A^T b$ .*

*Proof.* We start by writing the explicit solution of Equation (8.3.1) using the pseudoinverse

$$\begin{aligned}
 \begin{pmatrix} A \\ \alpha I \end{pmatrix}^+ \begin{pmatrix} b \\ 0 \end{pmatrix} &= \left( \begin{pmatrix} A \\ \alpha I \end{pmatrix}^T \begin{pmatrix} A \\ \alpha I \end{pmatrix} \right)^{-1} \begin{pmatrix} A \\ \alpha I \end{pmatrix}^T \begin{pmatrix} b \\ 0 \end{pmatrix} \\
 &= \left( (A^T \quad \alpha I) \begin{pmatrix} A \\ \alpha I \end{pmatrix} \right)^{-1} (A^T \quad \alpha I) \begin{pmatrix} b \\ 0 \end{pmatrix} \\
 &= (A^T A + \alpha^2 I)^{-1} A^T b.
 \end{aligned} \tag{8.3.3}$$

□

**Fact 8.3.3.** *We can observe that  $(A^T A + \alpha^2 I)$  is positive definite.*

*Proof.*  $z^T \cdot (A^T A + \alpha^2 I) \cdot z = z^T A^T A z + \alpha^2 z^T z \stackrel{(1)}{=} \alpha^2 z^T z = \alpha^2 \|z\|^2 > 0$ . The equality (1) is obtained because  $z^T A^T A z \geq 0$ , since  $A^T A$  is positive semidefinite.

□

**Exercise 8.3.1.** *Show using the SVD of  $A$  that the Tikhonov / Ridge solution can be written as*

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

When  $\sigma_i \gg \alpha$ ,  $\frac{\sigma_i}{\sigma_i^2 + \alpha^2} \approx \frac{1}{\sigma_i}$ : similar to the ‘true’ solution.

When  $\sigma_i \ll \alpha$ ,  $\frac{\sigma_i}{\sigma_i^2 + \alpha^2} \approx \frac{\sigma_i}{\alpha^2} \approx 0$ : approximately ignoring small singular values.

**Per casa:** fare esercizio.

How can we choose  $k$ ? We don’t know.

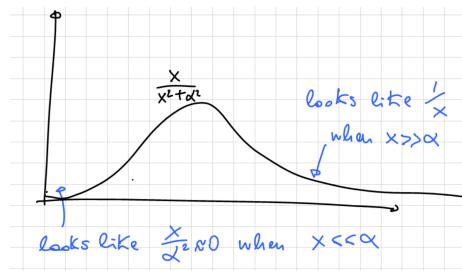


FIGURE 8.1: Here is the shape of the formula for the singular values.

# Chapter 9

## 9th of November 2018 — F. Poloni

### 9.1 Conditioning

Two lectures ago we introduced the QR factorization to solve least squares problems and we noticed that it has a computational complexity which is much worse than the normal equations method.

Why did we introduce the QR factorization to solve the least squares problem, then? Although it's more complex computationally speaking, it's much better than the normal equation for what concerns accuracy. Let's see why through an example:

**Example 9.1.1.** Let  $A \in \mathcal{M}(4, 3, \mathbb{R})$  s.t.

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

In this case the normal equations method doesn't even find the order of magnitude correctly.

At this point we may introduce the problem of **sensitivity**:

**Definition 9.1.1** (Sensitivity). We call **sensitivity** the measure of how much the output of a problem changes when we perturb its input.

As an example, let  $f(x, y) = x + 2y$ . If we perturb the second parameter of  $f$  as follows  $\tilde{y} = y + \delta$ , we can compute the variation in the output value of the function  $v$  as

$$\mathbf{v} = f(x, \tilde{y}) - f(x, y) = x + 2(y + \delta) - (x + 2y) = 2\delta.$$

A good example of this behaviour 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. This function is plotted in Figure 9.1.

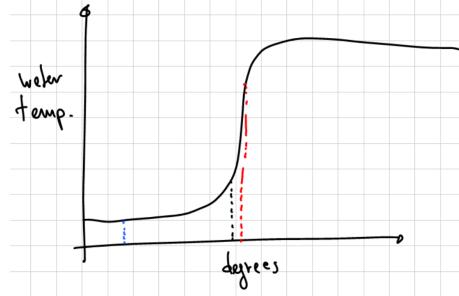


FIGURE 9.1: Geometric idea of temperature of the water in the shower

**Definition 9.1.2** (Absolute condition measure). *The absolute condition number of a function  $f$  is the maximum possible output change / input change ratio in the limit for a small change of the input.*

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

We would like to focus on what this definition means.

Why are we interested in the limit of a very small change?

If we zoom-in a continuous function it gets basically linear (key idea of derivative) and then the ratio between the difference on the outputs and the one of the inputs is approximatively the derivative, as shown in Figure 9.2.

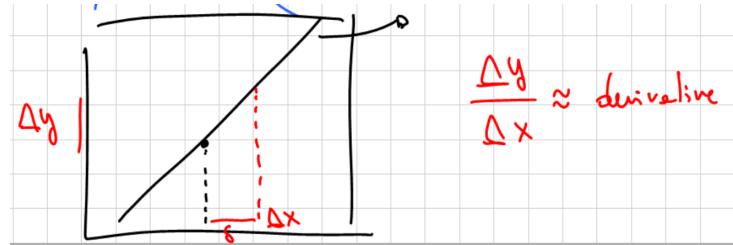


FIGURE 9.2: Geometric idea behind the derivative, as “zoom” of the function in a certain point.

We take a point  $x$  and we consider a ball of radius  $\varepsilon$  and we compute the change in the output over the change in the input, then we take the maximum.

**Example 9.1.2.** Let  $f(x) = x^2$ .

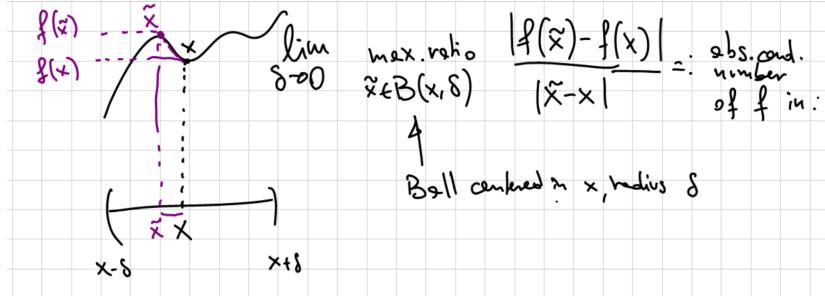


FIGURE 9.3: Geometric idea behind absolute condition number.

If we perturb the input  $x$  to  $x + \delta$ , then  $f(\tilde{x}) = (x + \delta)^2 = x^2 + 2x\delta + \delta^2$ , then we obtain the ratio

$$r = \frac{|f(\tilde{x}) - f(x)|}{|\tilde{x} - x|} = \frac{|2x\delta + \delta^2|}{|\delta|} = |2x + \delta|$$

If we denote  $\varepsilon$  the radius of the ball, we obtain the following

$$\max_{\substack{|\delta| < |\varepsilon| \\ \tilde{x} \in B(x, \varepsilon)}} r = |2x| + |\varepsilon|$$

then

$$\lim_{\varepsilon \rightarrow 0} \max_{|\delta| < |\varepsilon|} r = \lim_{\varepsilon \rightarrow 0} |2x| + |\varepsilon| = |2x|$$

**Example 9.1.3.** It's more interesting to see a multivariate function:

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

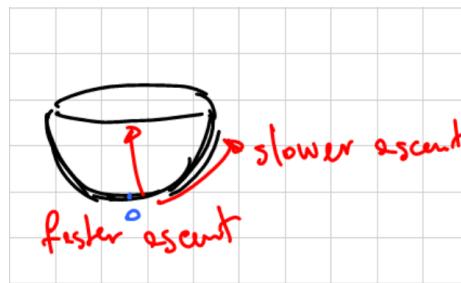


FIGURE 9.4: Paraboloid

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  $\lim_{\varepsilon \rightarrow 0} \max_{\tilde{x} \in B(x, \varepsilon)} \frac{\|f(\tilde{x}) - f(x)\|}{\|\tilde{x} - x\|}$ , 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  $B(x, \varepsilon)$ .

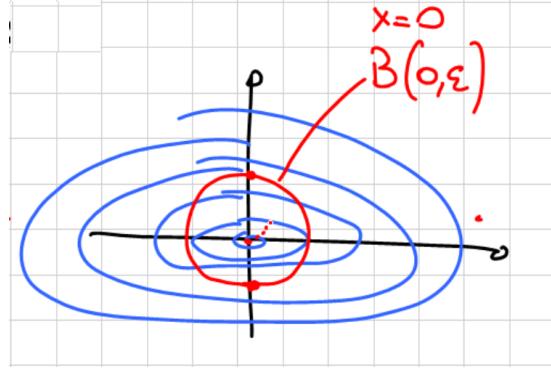


FIGURE 9.5: Level curves of a quadratic function (“seen from above”).

At this point, an observation is mandatory: **any** absolute measure doesn't take into account the values of the function in other points, so we want to define the following

**Definition 9.1.3** (Relative error). *The relative error of an approximation  $\tilde{x}$  to a quantity  $x$  is  $\frac{\|\tilde{x} - x\|}{\|x\|}$ .*

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).

**Definition 9.1.4** (Relative condition number). *The relative condition number of a function  $f$  is defined as*

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

i.e., we replace the absolute error  $\|\tilde{\mathbf{x}} - \mathbf{x}\|$  with the relative error.

### 9.1.1 Conditioning of linear systems

At this point we would like to compute the condition number of solving a linear system, i.e., the condition number of the function  $f(A, b) = A^{-1}b$ , perturbing the inputs  $A$  and  $b$ , one at a time.

PERTURBING  $b$  We want to compute the limit of the relative error  $\frac{\|f(A, \tilde{b}) - f(A, b)\|}{\|f(A, b)\|}$ ,

so we set  $x = A^{-1}b$  and  $\tilde{x} = A^{-1}\tilde{b}$ , and we estimate *output error* =  $\frac{\|\tilde{x} - x\|}{\|x\|} = ?$

1.

$$\begin{aligned} \|\tilde{x} - x\| &= \|A^{-1}\tilde{b} - A^{-1}b\| \\ &= \|A^{-1}(\tilde{b} - b)\| \\ &\leq \|A^{-1}\| \cdot \|\tilde{b} - b\| \end{aligned} \quad (9.1.1)$$

2. Since  $\|b\| = \|Ax\| \leq \|A\| \cdot \|x\|$  we have  $\frac{\|\tilde{x} - x\|}{\|x\|} \leq \|A^{-1}\| \cdot \|A\| \cdot \frac{\|\tilde{b} - b\|}{\|b\|}$

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

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

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

PERTURBING  $A$  Given  $Ax = b$  we obtain  $(A + \Delta_A) \cdot (x + \Delta_x) = b$ , where  $\tilde{A} = A + \Delta_A$  and  $\tilde{x} = x + \Delta_x$ . Then we can expand as follows

$$A\tilde{x} + \Delta_A \cdot x + A \cdot \Delta_x + \Delta_A \cdot \Delta_x = \tilde{b}$$

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

$$\Delta_A \cdot x + A \cdot \Delta_x = 0$$

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

then  $\|\Delta_x\| \leq \|A^{-1}\| \cdot \|\Delta_A\| \cdot \|x\|$ , which implies  $\frac{\|\Delta_x\|}{\|x\|} \leq \|A^{-1}\| \cdot \frac{\|\Delta_A\|}{\|A\|}$ .

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

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

In the end, in both cases, the error in the output is the error in the input (namely  $b$  or  $A$ ) times the condition number.

## 9.2 Condition number, SVD, and distance to singularity

**Fact 9.2.1.**  $\kappa(A) = \frac{\sigma_1}{\sigma_n}$ , i.e.,  $\kappa(A)$  is the ratio between the smallest and the largest singular value.

So we can say that if a matrix is close to a singular matrix, then its condition number is going to be large.

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

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

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

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

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

□

**Fact 9.2.2.** The relative distance between  $A$  and the closest singular matrix is  $\frac{1}{\kappa(A)}$ .

### 💡 Do you recall?

**Eckart-Young theorem:** the closest matrix to  $A$  that has rank  $\leq n-1$  is

$$\hat{A} = U \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ & & & 0 \end{pmatrix} \cdot V^T$$

*Proof.*

bau (9.2.1)

$$\begin{aligned}
\|A - \hat{A}\| &= \left\| U \cdot \left( \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix} - \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_{n-1} & 0 \end{pmatrix} \right) \cdot V^T \right\| \\
&= \left\| U \cdot \begin{pmatrix} 0 & & & \\ & \ddots & & \\ & & 0 & \\ & & & \sigma_n \end{pmatrix} \cdot V^T \right\| \\
&= \sigma_n
\end{aligned} \tag{9.2.2}$$

Thus,  $\|A - \hat{A}\| = \sigma_n$ . We know already that  $\|A\| = \sigma_1$ , so we just need to take the ratio.  $\square$

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

### 9.3 Conditioning of least squares problem

We need two quantities to be able to measure the conditioning of least squares problem:

- ★  $\kappa(A)$  Let  $A \in \mathcal{M}(m, n, \mathbb{R})$ , with  $m > n$  (tall, thin  $A$ ). We define  $\kappa(A) = \frac{\sigma_1}{\sigma_n}$ . We take this as a definition of  $\kappa(A)$ . Note that we cannot use the other definition  $\kappa(A) = \|A\| \|A^{-1}\|$ , since  $A^{-1}$  does not exist for a non-square  $A$ . However, one can verify that  $\|A\| \cdot \|A^\dagger\| = \frac{\sigma_1}{\sigma_n} = \kappa(A)$ , where  $A^\dagger$  is the pseudoinverse.

**Observation 9.3.1.** Note that  $\frac{1}{\kappa(A)}$  is the relative distance to the closest  $\hat{A}$  without full column rank.

- ★  $\theta$  The second quantity needed is the angle between  $Ax$  and  $b$ , see Figure 9.6.  
 $\theta = \arccos \frac{\|Ax\|}{\|b\|}$

Now we can express the theorem:

**Theorem 9.3.1** (Trefethen, Bau, Theorem 18.1). *Consider the linear least squares problem  $\min \|Ax - b\|$ , with  $A \in \mathbb{R}^{m \times n}$  with full column rank. Its relative condition number with respect to the input  $b$  is*

$$\kappa_{rel,b \rightarrow x} \leq \frac{\kappa(A)}{\cos \theta},$$

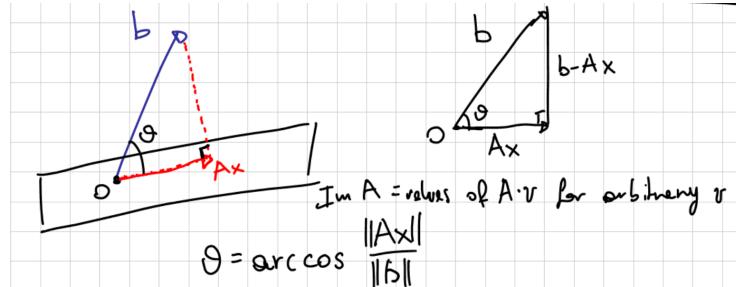


FIGURE 9.6: The triangle in the figure (the one whose cathetus are  $Ax$  and  $b - Ax$ ) is a square triangle.

and with respect to  $A$  it is

$$\kappa_{rel, A \rightarrow x} \leq \kappa(A) + \kappa(A)^2 \tan \theta,$$

where  $\theta$  is the angle such that  $\cos \theta = \frac{\|Ax\|}{\|b\|}$ .

At this point we have two condition numbers and they both depend on  $\kappa(A)$  and  $\theta$ .

### Observation 9.3.2.

SPECIAL CASE 1:  $\theta \approx 90^\circ$  We can see from the figure that a big change of  $b$  induces a small perturbation of  $Ax$ . No matter what the conditioning of  $A$  is, a small (relative) perturbation in  $b$  can change a large (relative) perturbation in  $x$  and  $Ax$ , see Figure 9.7.

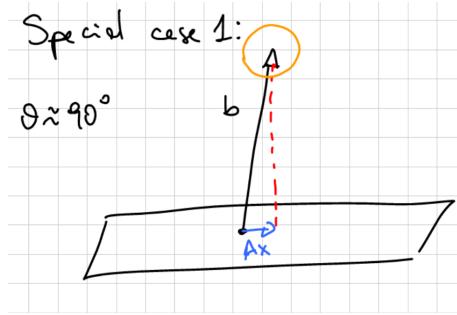


FIGURE 9.7: Special case 1

SPECIAL CASE 2:  $\theta \approx 0^\circ$  When  $b$  is almost in plane with  $\text{Im}(x)$ . In this case  $\text{cond} \approx \kappa(A)$ , see Figure 9.8.

GENERAL CASE:  $\theta$  FAR FROM  $0^\circ$  AND  $90^\circ$  In the more general case,  $\text{cond} \approx \kappa(A)^2$ .

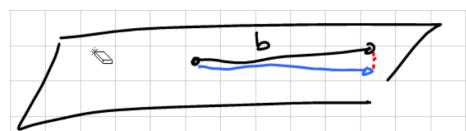


FIGURE 9.8: Special case 2.



# Chapter 10

## 15th of November 2018 — F. Poloni

### 10.1 Stability of algorithms

In this lecture we will try to answer the question: “Is our algorithm (using floating point) going to compute a good approximation of the answer?”

It is related to sensitivity / conditioning but different. Depends on how we perform the computation.

#### 💡 Do you recall?

Computers work with IEEE arithmetic and the basic idea is the following.  
TL;DR: floating point numbers are numbers in base-2 scientific (exponential) notation.

`double` (64-bit numbers):

$$\pm 1.\underbrace{01001011101\dots 101}_{52 \text{ binary digits}} \cdot 2^{\pm \underbrace{101\dots 01}_{10 \text{ binary digits}}}.$$

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  $\frac{1}{3} = 0.33333\dots$  and if we do  $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 0.99999\dots \neq 1$ .

Whenever we store a number on a computer we need to approximate it, using the “representable numbers”, as shown in Figure 10.1.

Not all numbers are exactly representable, take 0.1 (decimal). It’s a periodic

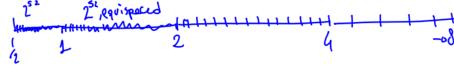


FIGURE 10.1: We have  $2^{52}$  equispaced numbers between  $\frac{1}{2}$  and 1 and between 1 and 2, and also  $2^{52}$  between 2 and 4 and so on and so forth, so we have the same number of integers, although the space is enlarging.

number when written in binary, hence we can't represent it exactly as a machine number.

**Definition 10.1.1** (Error bound). *For each  $x \in \pm[10^{-308}, 10^{308}]$ , there is an exactly representable number  $\tilde{x}$  such that  $\frac{|\tilde{x}-x|}{|x|} \leq u$ , with  $u = 2^{-52} \approx 2 \cdot 10^{-16}$ .*

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

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

In practice we may ignore  $o(u)$ , since it's small o of  $2^{-16}$ .

When we are approximating a result, we are approximating it up to a relative error of order of magnitude  $10^{-16}$ .

$$a \oplus b = (a+b)(1+\delta), \quad |\delta| \leq u.$$

⇓

$$\frac{|(a \oplus b) - (a+b)|}{|a+b|} = \delta$$

We would like to compute the error on the function  $f(a, b) = a^T b$ .

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \text{ so } f(a, b) = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

Denoting  $\odot$  the approximated product and  $\oplus$  the approximated sum on a computer, we can write the following:

$$\begin{aligned}
\text{computer result} &= a_1 \odot b_1 \oplus a_2 \odot b_2 \oplus a_3 \odot b_3 \\
&= \left[ [a_1 b_1 \cdot (1 + \delta) + a_2 b_2 \cdot (1 + \delta_2)] \cdot (1 + \delta_3) + a_3 b_3 \cdot (1 + \delta_4) \right] \cdot (1 + \delta_5) \\
&= a_1 b_1 \cdot (1 + \delta_1) \cdot (1 + \delta_3) \cdot (1 + \delta_5) + a_2 b_2 \cdot (1 + \delta_2) \cdot (1 + \delta_3) \cdot (1 + \delta_5) \\
&\quad + a_3 b_3 \cdot (1 + \delta_4) \cdot (1 + \delta_5) \\
&= a_1 b_1 (1 + \delta_1 + \delta_3 + \delta_5 + O(u^2)) \\
&\quad + a_2 b_2 \cdot (1 + \delta_1 + \delta_3 + \delta_5 + O(u^2)) \\
&\quad + a_3 b_3 \cdot (1 + \delta_4 + \delta_5 + O(u^2)) \\
&\approx a_1 b_1 (1 + \delta_1 + \delta_3 + \delta_5) + a_2 b_2 \cdot (1 + \delta_1 + \delta_3 + \delta_5) + a_3 b_3 \cdot (1 + \delta_4 + \delta_5)
\end{aligned} \tag{10.1.1}$$

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.

The absolute error then is

$$\begin{aligned}
\text{Err}_a &= |a_1 b_1 (1 + \delta_1 + \delta_3 + \delta_5) + a_2 b_2 \cdot (1 + \delta_1 + \delta_3 + \delta_5) + a_3 b_3 \cdot (1 + \delta_4 + \delta_5) + a_1 b_1 + a_2 b_2 + a_3 b_3| \\
&\stackrel{(1)}{\leq} |a_1 b_1| \cdot 3u + |a_2 b_2| \cdot 3u + |a_3 b_3| \cdot 2u \\
&\leq (|a_1 b_1| + |a_2 b_2| + |a_3 b_3|) \cdot 3u
\end{aligned} \tag{10.1.2}$$

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

The result that we can obtain is weaker than we would have expected: if  $a_i b_i \geq 0, \forall i = 1, 2, 3$ , then

$$\frac{|\text{computer result} - (a_1 b_1 + a_2 b_2 + a_3 b_3)|}{a_1 b_1 + a_2 b_2 + a_3 b_3} \leq 3u,$$

which means that the algorithm is stable.

However, if  $a_1 b_1, a_2 b_2$  and  $a_3 b_3$  have different signs, then we can bound the error not with  $a_1 b_1 + a_2 b_2 + a_3 b_3$ , but only with  $|a_1 b_1| + |a_2 b_2| + |a_3 b_3|$ . This might be a lot larger than what we want to compute.

**Example 10.1.1.** Take  $\varepsilon = 10^{-16}$ . Compute  $(1 \quad -1 \quad 0) \cdot \begin{pmatrix} 1 + \varepsilon \\ 1 \\ 1 \end{pmatrix} = (1 + \varepsilon) - 1 + 0 = \varepsilon$ .

In this case we have a subtraction between two very close numbers.

$|\text{computer result}| \leq u \cdot (1 + |1 \cdot (1 + \varepsilon)| + |-1 \cdot 1| + |0 \cdot 1|) = (2 + \varepsilon) \cdot u$ , which means that we have 10 correct digits.

The problem is in the fact that we have a very small result ( $\varepsilon = O(10^{-6})$ ) and a very small error (of the same order of the result), which implies a large relative error.

An attentive reader might have noticed that it's very long to make this computation. Therefore, since we computed it on easy examples, we have to observe that we can't afford it for SVD or other complicated methods.

Wilkinson trick (from the 60's) comes to help us, to simplify this computation for more complicated problems.

**Idea:** see the computer result as the exact output of an algorithm run on a slightly perturbed input.

$$\begin{aligned}\tilde{y} &= \dots \\ &= ((a_1 b_1(1 + \delta_1) + a_2 b_2(1 + \delta_2))(1 + \delta_4) + a_3 b_3(1 + \delta_3))(1 + \delta_5) \quad (10.1.3) \\ &= a_1 \tilde{b}_1 + a_2 \tilde{b}_2 + a_3 \tilde{b}_3\end{aligned}$$

where

$$\begin{aligned}\tilde{b}_1 &= b_1(1 + \delta_1)(1 + \delta_4)(1 + \delta_5) = b_1 + 3ub_1 + o(u), \\ \tilde{b}_2 &= b_2(1 + \delta_2)(1 + \delta_4)(1 + \delta_5) = b_2 + 3ub_2 + o(u), \quad (10.1.4) \\ \tilde{b}_3 &= b_3(1 + \delta_2)(1 + \delta_5) = b_3 + 2ub_3 + o(u),\end{aligned}$$

$$\tilde{a}_i = a_i \quad i = 1, 2, 3.$$

And the relative error is  $\frac{\|\tilde{b} - b\|}{\|b\|} \leq 3u + o(u)$ .  
Hence,

$$\frac{\|\tilde{y} - y\|}{\|y\|} \leq \kappa_{rel,b} \frac{\|\tilde{b} - b\|}{\|b\|} \leq k_{rel,b} \cdot 3u$$

This isn't bad, because it's within a factor 3 of the optimal error for a perfect algorithm.

Computing the conditioning is much easier than making all the calculations of  $\delta$ s.

**Definition 10.1.2** (Backward stability of an algorithm). *An algorithm to compute  $\mathbf{y} = f(\mathbf{x})$  is called **backward stable** if the computed output  $\tilde{\mathbf{y}}$  can be written as  $\tilde{\mathbf{y}} = f(\tilde{\mathbf{x}})$ , where  $\tilde{\mathbf{x}} = \mathbf{x} + O(\mathbf{u} \|\mathbf{x}\|)$  (exact function, perturbed input).*

**Observation 10.1.1.** *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 10.1.1.** *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.*

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

while the best attainable accuracy is  $\kappa_{rel}(f, \mathbf{x})\mathbf{u}$ . □

We may ask ourselves if it's possible to perturb the input in order to get  $\tilde{y}$  for every possible algorithm and the answer is no. Let us see a counterexample, where  $f(a, b) = a^T b$  (vector-vector product in the order that produces a matrix).

We observe that in general it's not true that the computed approximation of  $f(a, b)$  has rank 1.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \cdot \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \\ x_3 y_1 & x_3 y_2 & x_3 y_3 \end{bmatrix} = M$$

While

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \odot \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1 \odot y_1 & x_1 \odot y_2 & x_1 \odot y_3 \\ x_2 \odot y_1 & x_2 \odot y_2 & x_2 \odot y_3 \\ x_3 \odot y_1 & x_3 \odot y_2 & x_3 \odot y_3 \end{bmatrix} = \widetilde{M}$$

And we are looking for  $\tilde{x}$  and  $\tilde{y}$  such that  $\widetilde{M} = \tilde{x}^T \cdot \tilde{y}$

**Example 10.1.2.** Example (with exaggerated errors):

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \cdot [4 \quad 5 \quad 6] = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix}$$

While

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \odot [4 \quad 5 \quad 6] = \begin{bmatrix} 4.01 & 4.99 & 6.01 \\ 7.99 & 10.01 & 12.02 \\ 11.98 & 15.02 & 17.97 \end{bmatrix}$$

and this second matrix doesn't have rank 1.

## 10.2 Backward stability of QR factorization

### 💡 Do you recall?

A generic step of the computation of the QR factorization of a matrix has the following shape:

$$\left( \begin{array}{cc} I & \\ & H \ u_k \end{array} \right) \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix} = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

Each step of the QR factorization is backward stable.

Let us compute the backward stability of QR factorization:

With some tedious computations like the ones above, one can show that *one* step of the QR factorization is backward stable, i.e.,  $\tilde{R}_{k-1} = R_{k-1} + \Delta_{R_{k-1}}$ , where  $\|\Delta_{R_{k-1}}\| \leq O(u) \|R_{k-1}\|$  and this allows us to write the following

$$\frac{\|\tilde{R}_{k-1} - R_{k-1}\|}{\|R_{k-1}\|} \leq O(u)$$

At a generic step  $k$  we have that

$$R_k = Q_k \cdot R_{k-1}$$

$\Updownarrow$

$$\tilde{R}_k = Q_k(R_{k-1} + \Delta_{R_{k-1}})$$

The idea is doing this procedure one step after the other, starting from step 1, where  $R_0 = A$ .

$$\tilde{R}_1 = Q_1 \tilde{R}_0 = Q_1 \cdot (R_0 + \Delta_{R_0}) = Q_1 \cdot (A + \Delta_{R_0})$$

$$\tilde{R}_2 = Q_2 \tilde{R}_1 = Q_2(R_1 + \Delta_{R_1}) = Q_2 Q_1(A + \Delta_{R_0}) + Q_2 \Delta_{R_1} = Q_2 Q_1(A + \Delta_{R_0} + Q_1^T \Delta_{R_1})$$

We go on combining errors like this and we see that all the quantities  $(A + \Delta_{R_j})$  are perturbations of the original matrix  $A$ .

In the end we may observe that the final computed  $R_n$  is the exact result obtained from  $A$  plus  $n$  perturbations.

**Observation 10.2.1.** *We should notice that the norm of each perturbation is small with respect to the norm of  $A$ .*

*In formal terms,  $\|\Delta_{R_0}\| \leq u \cdot \|R_0\| = u \cdot \|A\|$  and  $\|Q_1^T \Delta_{R_1}\| \stackrel{(1)}{=} \|\Delta_{R_1}\| \leq u \cdot \|R_1\| \stackrel{(2)}{=} u \cdot \|A\|$ .*

*Where  $\stackrel{(1)}{=}$  and  $\stackrel{(2)}{=}$  hold because  $Q_i$  are orthogonal.*

### ★ Mantra

Orthogonal transformations are the key for stability.

#### 10.2.1 Stability of algorithms for least-squares problems

Let us see how various algorithms to solve LS problems (implemented in Matlab) behave in relation to backward stability.

##### Least squares problem via QR

STEP 1: Computing a thin QR (`qr(A, 0);`) → backward stable;

STEP 2:  $(Q_1^T * b;)$  → backward stable;

STEP 3:  $(R_1 \ c;)$  → backward stable;

### Least squares problem via SVD

Scrivere  
meglio

STEP 1: Computing a SVD ( $\text{svd}(A, 0;)$ ) → backward stable;

STEP 2:  $(U^T * b;)$  → backward stable;

STEP 3:  $(c ./ \text{diag}(S);)$  → backward stable;

STEP 4:  $(V * d;)$  → backward stable;

### Least squares problem via Normal Equations

STEP 1:  $C = A^T * A;$

STEP 2:  $d = A^T * b;$

STEP 3:  $x = C^{-1} * d;$

We can also prove that some algorithms aren't backward stable, like normal equations. The issue here is that the same input is needed in more than one operation, so it should satisfy more than one equation and this may lead to a solution set which is empty. Also, in the last equation we solve a linear system with matrix  $C$ , and this gives an error of the order of  $\kappa(A)^2$ , never of  $\kappa(A)$ :  $\kappa(C) = \kappa(A^T A) = \kappa(A)^2$ .

Let us take the example of two lectures ago: ??.

**Example 10.2.1.** 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  $\kappa \approx 10^8$ .

What is the condition number of solving this least squares problem? It's about  $10^8$ , since in that problem we generated  $b = A \cdot (3 \ 4 \ 5)$ , so  $b$  lies in the image of  $A$  ( $\text{Im}(A)$ ). The geometric idea in Figure 10.2.

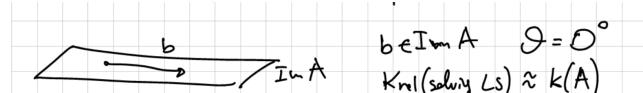


FIGURE 10.2:  $b$  lies (at least almost, because of numerical error) on the plane of  $\text{Im}(A)$ . In this case  $\kappa_{\text{rel}}(\text{solving LS}) \approx \kappa(A)$ .

A brief recap may be found in Table 10.1.

	Normal eqns	QR	SVD
$m \approx n$	$\frac{4}{3}n^3$	$\frac{4}{3}n^3$	$\approx 13n^3$
$m \gg n$	$mn^2$	$2mn^2$	$2mn^2$
Unstable when $cond \approx \kappa(A)$	Backward stable	Backward stable	Backward stable; reveals info on sensitivity, allows regularization

TABLE 10.1: Brief recap of the complexities of the algorithms we studied for solving the least squares problem. The last row takes into account the stability.

### 10.2.2 A posteriori checks

Let's assume that Matlab gives us a solution of a problem. We want to be able to check how good this result is.

**Example 10.2.2.** Suppose we have solved a linear system...

```
>> A = randn(4, 4); b = randn(4, 1);
>> x = A \ b;
>> A*x - b
ans =
    0
-1.3878e-17
    0
  2.2204e-16
```

**Definition 10.2.1** (Residual). Let  $A \in \mathcal{M}(m, \mathbb{R})$  and  $\in \mathbb{R}^m$ , and  $x$  be the solution of  $Ax = b$ . For a given  $\tilde{x}$  we define **residual** the following  $\mathbf{r} = A\tilde{x} - b$ .

assume that  $\|\mathbf{r}\|$  is small; does this mean that  $\tilde{x}$  is close to the exact solution  $x$ ?

**Theorem 10.2.1.** Let  $A \in \mathbb{R}^{m \times m}, \in \mathbb{R}^m$ , and  $x$  be the solution of  $Ax = b$ .

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

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

This theorem tells us that  $\tilde{x}$  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  $\tilde{x}$  is the exact solution of the perturbed system

$$A\tilde{x}^* = b + \mathbf{r} = \tilde{b}$$

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

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.

### 10.3 A posteriori check for Least Squares Problems

Can we make an analogous check for the Least Squares Problem?

Take a LSP  $\min \|Ax - b\|$ , with  $A$  tall thin, with full column rank.

The problem is that  $\|Ax - b\|$  isn't small at all, indeed it could be as large as  $b$ , as you can see from Figure 10.3.

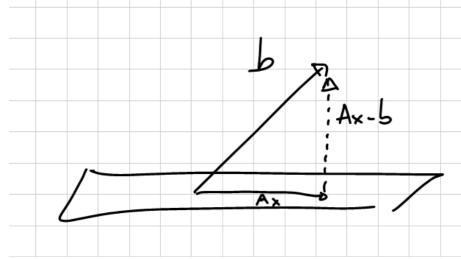


FIGURE 10.3:  $A$  can be as large as  $b$  if  $b$  is perfectly orthogonal.

**Observation 10.3.1.** 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 10.3, 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:  $\min \|Ax - b\|^2 = \min x^T A^T A x - 2b^T A x + b^T b$ .

$$\nabla_{\tilde{x}} f = 2(A^T A x - A^T b) \rightarrow 0$$

**Theorem 10.3.1.**  $\frac{\|\tilde{x} - x\|}{\|x\|} \leq (\kappa(A))^2 \cdot \frac{\|A^T A x - A^T b\|}{\|A^T b\|}$ . 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  $R_1^{-1} x - Q_1^T b$ ), by imposing  $R_1 x = Q_1^T b$

We may observe that this is a truly backward stable measure:  
given  $r = \|R_1^T \tilde{x} - Q_1^T b\|$ , there exists  $\tilde{b}$  with  $\|\tilde{b} - b\| = \|r\|$  such that  $\tilde{x}$  is  
the exact solution of  $\min \|Ax - \tilde{b}\|$ .

We have proved the following

$$\text{Fact 10.3.2. } \frac{\|\tilde{x} - x\|}{\|x\|} \leq \kappa_{rel,LS} \cdot \frac{\|\tilde{b} - b\|}{\|b\|} = \kappa_{rel,LS} \cdot \frac{\|r\|}{\|b\|}.$$

**Theorem 10.3.3.** Let  $A = Q_1 R_1$  be a thin QR factorization. Let  $\mathbf{r}_1 = Q_1^T(A\tilde{x} - b)$ . Then,  $\tilde{x}$  is the exact solution of the LS problem

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

so the backward error of  $\tilde{x}$  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  $R_1 x = Q_1^T b + \mathbf{r}_1$ , i.e.,  
 $Q_1^T(Ax - b) = \mathbf{r}_1$ , which is verified by  $\tilde{x}$ .  $\square$

# Chapter 11

## 21st of November 2018 — F. Poloni

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

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 see 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 11.1;

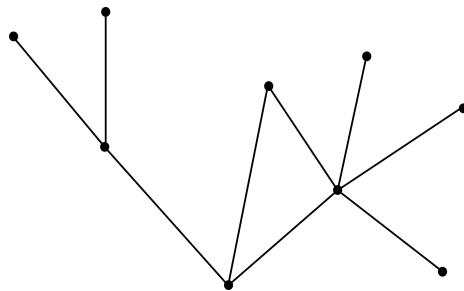


FIGURE 11.1: A local function on graph.

**IMAGES:** Take an  $m \times m$  image and blur it (each pixel is obtained as the average of its neighbours).  $T : \mathcal{M}(m, \mathbb{R}) \rightarrow \mathcal{M}(m, \mathbb{R})$  such that  $T(A)_{ij} = \frac{1}{9} \cdot (A_{i-1,j} + A_{i-1,j-1} + A_{i-1,j+1} + A_{ij} + A_{ij-1} + A_{ij+1} + A_{i+1,j} + A_{i+1,j-1} + A_{i+1,j+1})$ .

$T$  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  $A_{i-1j}, \dots, A_{i+1j+1}$ ;

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. In the end, this local phenomenon may be represented by a sparse matrix.

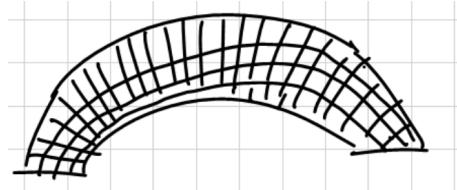


FIGURE 11.2: Graphic idea of a bridge partitioned into small blocks

## 11.1 Gaussian elimination and 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  $n$  to kill off  $A_{2:end,1}$  we have that:

STEP 1:

$$\begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} \textcircled{*} & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix} = \begin{pmatrix} \textcircled{*} & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix},$$

Where the  $\textcircled{*}$  is called **pivot**.

$$L_1 A = A_1.$$

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

STEP 2: we multiply for a matrix that has an “identity frame” and inside does the same  $L_1$  was doing before.

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \end{pmatrix} \begin{pmatrix} * & \odot & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} = \begin{pmatrix} * & \odot & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix}$$

$$L_2 A_1 = A_2,$$

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

STEP 3: we go on and

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & * & 1 & & \\ * & & & 1 & \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \odot & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix} = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \odot & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

$$L_3 A_2 = A_3,$$

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

STEP 4: one more operation

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & \odot & * \\ 0 & 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & \odot & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}$$

$$L_4 A_3 = A_4,$$

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

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

**Theorem 11.1.1.** *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  $L$  is lower triangular with ones on its diagonal, and  $U$  is upper triangular.*

**Observation 11.1.1** (Stroke of luck). *The product of the  $L_i^{-1}$ 's (denoted  $L$ ) can be computed for free, since the following holds:*

$$\left[ \begin{array}{cc} 1 & \\ -a_2 & 1 \\ -a_3 & 1 \\ -a_4 & 1 \\ -a_5 & 1 \end{array} \right]^{-1} \cdot \left[ \begin{array}{cc} 1 & \\ & 1 \\ -b_3 & 1 \\ -b_4 & 1 \\ -b_5 & 1 \end{array} \right]^{-1} \cdot \left[ \begin{array}{cc} 1 & \\ & 1 \\ & -c_4 & 1 \\ & -c_5 & 1 \end{array} \right]^{-1} \cdot \left[ \begin{array}{cc} 1 & \\ & 1 \\ & & 1 \\ & & -d_5 & 1 \end{array} \right]^{-1} = \left[ \begin{array}{ccccc} 1 & & & & \\ a_2 & 1 & & & \\ a_3 & b_3 & 1 & & \\ a_4 & b_4 & c_4 & 1 & \\ a_5 & b_5 & c_5 & d_5 & 1 \end{array} \right]$$

---

ALGORITHM 11.1.1 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

```

---

The idea behind the implementation of Algorithm 11.1.1 is shown in Figure 11.3.

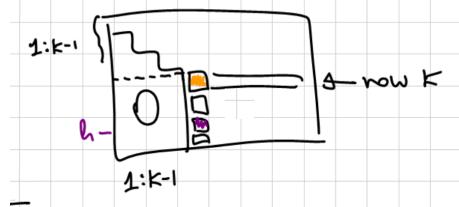


FIGURE 11.3: Assuming that we are at step  $k$ , we have that the  $h$ -th multiplier is expressed as  $\frac{A_{hk}}{A_{kk}}$  and this multiplier goes to the right position in  $L$ .

**Observation 11.1.2.** *The computational complexity of this algorithm is concentrated at lines 10, 11 and the cost of this operation is  $O((m-1)^2 + (m-2)^2 + \dots + 2^2 + 1)$ . So we have that for a dense matrix the computational complexity is  $\frac{2}{3}m^3 + O(m^2)$ , in other words half as much as QR factorization.*



### Something on Matlab ...

**Implementation of \ in Matlab:** We consider important to remark how the operator  $\backslash$  is implemented in Matlab. It works using the LU factorization, making some checks and changing the algorithm as follows:

- upper/lower triangular systems: back-substitution ( $O(n^2)$ );
- non-triangular linear systems: LU with partial pivoting (then throw away the factors);
- symmetric and/or sparse systems: uses appropriate LU variants (will see in the following).
- non-square matrices: solves the system “in the least squares sense  $\min_x \|Ax - b\|_2$ ”.

Why isn't there any check on the orthogonality of the matrix? Because in that case  $A^{-1} = A^T$ , and hence it is easy to solve the system. Well, in order to find out that a matrix is orthogonal we need to compute  $A^T A$ , which is too costly.

**Obs:** LU without pivoting isn't stable, as shown in Section 11.1.1, so Matlab uses pivoting.

missing

#### 11.1.1 Stability of LU

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

Let us see an example:

##### Example 11.1.1.

$$A = \begin{bmatrix} 10^{-30} & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^{30} & 1 \end{bmatrix} \begin{bmatrix} 10^{-30} & 1 \\ 0 & 1 - 10^{30} \end{bmatrix}.$$

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

Luckily, it's easy to circumvent this issue multiplying  $L_i$ s by some permutation matrices (which swap rows in order to keep “large” pivots), as follows

$$L_{m-1} P_{m-1} \dots L_2 P_2 L_1 P_1 A = U.$$

**Observation 11.1.3.** Thanks to another “stroke of luck” we can reorder those factors:

$$L_{m-1} P_{m-1} \dots L_2 P_2 L_1 P_1 = \widehat{L}_{m-1} \widehat{L}_{m-2} \dots \widehat{L}_1 P_{m-1} P_{m-2} \dots P_1$$

where  $\widehat{L}_i$  have the same structure as the  $L_i$ .

We can now introduce the following

**Theorem 11.1.2.** *Let  $A \in \mathcal{M}(m, \mathbb{R})$ .  $A$  admits a factorization  $A = PLU$ , where  $P$  is a permutation matrix,  $L$  is lower triangular with ones on its diagonal, and  $U$  is upper triangular.*

What about the stability of this improvement to LU factorization, called **LU with partial pivoting**?

It's not stable at all, in the worst case it may happen that  $\|U\|/\|A\|$  may grow as  $\approx 2^m$ , although matrices for which this happens are very rare.

### 11.1.2 Gaussian elimination on sparse matrices

Given a sparse matrix

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

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:

$$\begin{pmatrix} * & * & & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{pmatrix} \rightarrow \begin{pmatrix} * & * & & * & * & * \\ * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{pmatrix}$$

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.

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

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

**Observation 11.1.4.** *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.*



# Chapter 12

23rd of November 2018 —  
F. Poloni

## 12.1 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 big chunk of 0s in the first column

$$\begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} \odot & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix} = \begin{pmatrix} \odot & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix},$$

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  $L_1$ :

STEP 1:

$$\begin{pmatrix} 1 & & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix} \begin{pmatrix} 1 & * & * & * & * \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}$$

STEP 2:

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{pmatrix} \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} \begin{pmatrix} 1 & & & & \\ & 1 & * & * & * \\ & & 1 & & * \\ & & & 1 & \\ & & & & 1 \end{pmatrix} = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix}$$

STEP  $m$ :

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

where  $D$  is diagonal, or

$$A = L_1 L_2 \dots L_{m-1} D L_{m-1}^T \dots L_2^T L_1^T = LDL^T.$$

**Observation 12.1.1** (Stroke of luck). Notice that the stroke of luck of Observation 11.1.1 holds in this case too, hence we pay nothing to compute matrix  $L$ .

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

**Theorem 12.1.1** (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  $L$  is lower triangular with ones on its diagonal, and  $D$  is diagonal.

A Matlab implementation of symmetric Gaussian elimination is shown in Algorithm 12.1.1.

---

ALGORITHM 12.1.1 Symmetric Gaussian factorization, Matlab implementation.

---

```

1  function [L, D] = ldl_factorization(A)
2      m = size(A, 1);
3      L = eye(m); D = zeros(m);
4      for k = 1:m-1
5          D(k, k) = A(k, k);
6          L(k+1:end, k) = A(k+1:end, k) / A(k, k);
7          A(k+1:end, k+1:end) = A(k+1:end, k+1:end) ...
8              - L(k+1:end, k) * A(k, k+1:end);
9      end
10     D(m, m) = A(m, m);

```

---

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.

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.

Of course there are some matrices (like the ones with all 0s on the diagonal) that cannot be “pivoted”. There are workarounds, though. As an example, Matlab’s  $[L, D, P] = \text{ldl}(A)$  produces matrices such that  $P^T AP = LDL^T$ , where  $D$  may have  $2 \times 2$  diagonal blocks.

### 💡 Do you recall?

We recall the characterization of **positive definite matrix**  $A \in M(m, \mathbb{R})$ : all its eigenvalues are strictly positive. In other words,  $A$  is positive definite if  $\forall z \neq 0 \in \mathbb{R}^m z^T Az > 0$ .

**Lemma 12.1.2.** *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 so, for some invertible  $M \in M(m, \mathbb{R})$ . Formally,  $\forall A \in S(m, \mathbb{R})$  s.t.  $A\succ 0 \Leftrightarrow \exists M \in GL(m, \mathbb{R})$  s.t.  $MAM^T\succ 0$ ;*
2. *Let  $A$  a symmetric positive definite matrix such that  $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ , then  $A_{11}$  and  $A_{22}$  are, too. Formally,  $\forall A \in S(m, \mathbb{R})$  s.t.  $A\succ 0$  and  $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \Rightarrow A_{11}\succ 0$  and  $A_{22}\succ 0$ .*

*Proof.*

1.

$$\Rightarrow) A \in S(m, \mathbb{R}) \text{ and } A\succ 0 \implies MAM^T \in S(m, \mathbb{R}) \text{ and } MAM^T\succ 0.$$

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

$$\Leftarrow) MAM^T \in S(m, \mathbb{R}) \text{ and } MAM^T\succ 0 \implies A \in S(m, \mathbb{R}) \text{ and } A\succ 0.$$

This proof follows from the previous arrow, where the substitution is  $z = M^{-1}y$ .

$$2. A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \text{ positive definite} \implies A_{11} \text{ and } A_{22} \text{ are positive definite too.}$$

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  $\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ 0 \end{bmatrix}$ .

$$\begin{bmatrix} \mathbf{z}_1^T & 0 \end{bmatrix} \cdot \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{z}_1 \\ 0 \end{bmatrix} = \mathbf{z}_1^T A_{11} \mathbf{z}_1 > 0, \forall \mathbf{z}_1 \in \mathbb{R}^{sizeof A_{11}}$$

$A_{22}$ ) Let us take  $\mathbf{z} = \begin{bmatrix} 0 \\ \mathbf{z}_2 \end{bmatrix}$ .

$$\begin{bmatrix} 0 & \mathbf{z}_2^T \end{bmatrix} \cdot \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \mathbf{z}_2 \end{bmatrix} = \mathbf{z}_2^T A_{22} \mathbf{z}_2 > 0, \forall \mathbf{z}_2 \in \mathbb{R}^{sizeof A_{22}}$$

□

**Corollary 12.1.3.** *Let  $A \in M(n, \mathbb{R})$  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 12.1.2 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

$$L_1 A L_1^T = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} \text{ and so the first and the second diagonal blocks}$$

are positive definite ( $D_{11} > 0$  and  $D_{22} \succ 0$ ).

Notice that, since  $A$  is positive definite  $A_{11} \succ 0$ , but it's a scalar, hence  $A_{11} > 0$  and this implies no breakdown case. □

We may introduce another kind of factorization.

## 12.2 Cholesky factorization

The key idea is to write the diagonal matrix of the Gaussian elimination  $D$  as product of  $D^{1/2}$  times itself:

$$D = \begin{pmatrix} d_{11} & & & & \\ & d_{22} & & & \\ & & \ddots & & \\ & & & d_{mm} & \end{pmatrix} = \begin{pmatrix} \sqrt{d_{11}} & & & & \\ & \sqrt{d_{22}} & & & \\ & & \ddots & & \\ & & & \sqrt{d_{mm}} & \end{pmatrix} \cdot \begin{pmatrix} \sqrt{d_{11}} & & & & \\ & \sqrt{d_{22}} & & & \\ & & \ddots & & \\ & & & \sqrt{d_{mm}} & \end{pmatrix}$$

The LDL factorization may be rewritten as follows

$$A = LDL^T = LD^{1/2}(D^{1/2}L^T) = CC^T,$$

where  $D^{1/2} = \text{diag}(D_{11}^{1/2}, D_{22}^{1/2}, \dots, D_{mm}^{1/2})$ , and  $C$  is lower triangular (but not anymore with ones on the diagonal).

In Matlab the Cholesky factorization of a positive definite matrix is performed by the function `chol(A)`; and returns  $C^T$ .

**Observation 12.2.1.** *We will not discuss stability further, but Cholesky is always backward stable even without pivoting ( $\|C\| = \|A\|^{1/2}$ ).*

**Observation 12.2.2.** *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.*

## 12.3 Krylov subspace methods

In this part we will discuss some different techniques to solved linear systems, inspired from optimization algorithms.

**Example 12.3.1.** Let us consider the optimization problem  $\min \frac{1}{2}x^T Ax + b^T x$ , where  $A \succ 0$ . We know that the solution to this problem is  $x = A^{-1}b$ . Assume we solve this problem via a gradient descent-type method, starting from  $x_0 = 0$ .

STEP 1:  $x_0 = 0$ ,  $\nabla f(x_0) = -b$ ;

STEP 2:  $x_1 = \text{some multiple of } b \in \text{Span}(b)$

$$f(x_1) = Ax_1 - b$$

$$\nabla f(x_1) = \text{some multpiple of } Ab + \text{some multiple of } b \in \text{span}(b);$$

STEP 3:  $x_2 = \text{mult. of } x_1 + \text{mul. of } \nabla f(x_1) + \text{mult. of } x_0 + \dots \in \text{span}(Ab_1, b)$ ;

STEP 4:  $x_3 = \text{mult.of } x_2 + \dots \nabla f(x_2) + \text{previous iterates} \dots$

$$Ax_2 - b = A \cdot (sAb + tb) - b = sA^2b + tAb - b \in \text{span}(b, Ab, A^2b), \text{ where } s \text{ and } t \text{ are scalars};$$

STEP 5:  $x_4 \in \text{span}(b, Ab, A^2b, A^3b)$ .

Notice that we can make some linear combinations of the vectors we have available and  $A(A(sb + tAb) + uAb + vb) + e(sb + tAb) + fAb + gb \in \text{span}(b, Ab, A^2b, A^3b)$ .

**Idea:** first compute the basis of  $\text{span}(b, Ab, A^2b, A^3b)$ , then look for the best solution inside this subspace.

The following family of algorithms “uses” the matrix  $A$  only by computing matrix-vector products.

**Observation 12.3.1.** The cost of multiplying a sparse matrix  $A$  with a vector  $z$  is  $O(nnz(A))$ , where  $nnz(A)$  is the number of non zero entries of  $A$ .

*Proof.* Let us assume the matrix  $A$  is stored as a vector, whose entries are  $(i, j, A_{ij})$ .

The matrix-vector product would then be

1. `x=zeros`
2. `for (i, j, Aij) such that Aij ≠ 0`
3.  $x_i = x_i + A_{ij} * z_j$
4. `end`

□

From now on we will consider to have a function `compute_product_with_A` that we use to compute matrix-vector products with  $A$ . In particular,  $x = \text{compute\_product\_with\_A}(z)$  computes  $x = Az$ , given  $z$ . This function will be the only way in which the matrix  $A$  appears in our algorithm. If  $A$  is sparse, hence, 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.

**Definition 12.3.1** (Krylov subspace). *Let  $A \in M(m, \mathbb{R})$  and let  $b \in \mathbb{R}^m$ . The Krylov subspace of index  $n$  is  $\mathbf{K}_n(\mathbf{A}, \mathbf{b}) = \text{span}(b, Ab, A^2b, \dots, A^{n-1}b)$ .*

*Equivalently,  $k \in K_n(A, b) \iff \exists \alpha_1, \dots, \alpha_{n-1} \in \mathbb{R}^m$  s.t.  $v = \alpha_0 b + \alpha_1 Ab + \alpha_2 A^2b + \dots + \alpha_{n-1} A^{n-1}b$ .*

*Equivalently,  $(\alpha_0 + \alpha_1 A + \alpha_2 A^2 + \dots + \alpha_{n-1} A^{n-1})b = p(A)b$  for a polynomial  $p$  of degree such that  $\deg(p) \leq n - 1$ .*

**Observation 12.3.2** (Properties).

1.  $v, w \in K_n(A, b) \Rightarrow \alpha + \beta W \in K_n(A, b);$
2.  $v \in K_n(A, b) \Rightarrow Av \in K_n(A, b)$ . *Proof.* Let us take  $v = \alpha_0 b + \dots + \alpha_{n-1} b$ , then  $Av = A(\alpha_0 b + \dots + \alpha_{n-1} b) = \alpha_0 Ab + \dots + \alpha_{n-1} A^n b;$
3.  $\dim(K_n(A, b)) \leq n$ . It is exactly  $n$  if  $b, Ab, A^2b, \dots, A^{n-1}b$  are linearly independent.
4. Let us assume  $\dim(K_n(A, b)) \leq n$ . In the second point, if  $A^{n-1}$  was really necessary  $\alpha_{n-1} \neq 0$  or  $v \in K_n(A, b)$ ,  $v \notin K_{n-1}(A, b)$ , equivalently then  $A^n b$  is really necessary to write  $Av$ , i.e.  $Av \in K_{n+1}(A, b)$  but  $Av \notin K_n(A, b)$ .
5. We may observe that  $\dim(K_1(A, b)) < \dim(K_2(A, b)) < \dots < \dim(K_{n_{\max}}(A, b)) = \dim(K_{n_{\max}+1}(A, b)) = \dots$

# Chapter 13

## 29th of November 2018 — F. Poloni

In this lecture we address the problem of designing methods that use Krylov spaces to solve linear systems.

We need alternative methods to Gaussian elimination because matrices are too large.

### 13.0.1 Naive idea

Find first a “good” search subspace, then look for best approximation of the solution of  $Ax = b$  in that space.

Let us assume that our space is the image of a matrix  $V$   $Im(V)$ .  $\forall x \in Im(V)$  such  $x$  may be written as  $x = V^1y_1 + V^2y_2 + \dots + V^ny_n$ , where  $V^i$  are the columns of the matrix  $V$ .

The idea is to find a vector  $y$  that satisfies  $\min_{y \in \mathbb{R}} \|AVy - b\|$ , which is a least squares problem.

### 13.0.2 Improvement

A good search space is  $Im(V) = K_n(A, b) = \text{span}(b, Ab, A^2b, \dots, A^{n-1}b)$ , where  $K_n(A, b)$  is the Krylov space.

The issue here is that  $V = (b \quad Ab \quad \dots \quad A^{n-1}b)$  is a bad basis, because it is ill conditioned.

Working with that  $V$  is problematic: its columns “tend” to be aligned with the dominant eigenvector and this means that  $V$  is close to a rank 1 matrix.

We need a better basis for  $K_n(A, b)$ , in particular an orthogonal basis.

 Do you recall?

An orthogonal basis is a basis in which each couple of vectors are orthogonal

### 13.1 Arnoldi algorithm

The idea behind this algorithm is to build an orthogonal basis of  $K_n(A, b)$  incrementally.

The algorithm at a generic step, takes an orthogonal basis for  $K_n(A, b)$  and adds a vector to produce one of  $K_{n+1}(A, b)$ .

Let us assume that we start with  $\{q_1, q_2, \dots, q_n\}$ , orthogonal basis of  $K_n(A, b)$ .

We also assume that  $q_n = \alpha_0 b + \alpha_1 A b + \dots + \alpha_{n-1} A^{n-1} b = p(A)b$ , where we impose  $\alpha_{n-1} \neq 0$ .

$p(A) = \alpha_0 I + \alpha_1 A + \dots + \alpha_{n-1} A^{n-1}$  has degree exactly 1.

STEP 1:  $K_1(A, b) = \text{span}(b)$ ,  $q_1 = \frac{q_1}{\|q_1\|}$

GENERIC STEP:

1. produce a vector  $K_{n+1}(A, b) - K_n(A, b) : w = Aq_n$
2.  $w \in K_{n+1}(A, b) = 0$   $w = q_1\beta_1 + q_2\beta_2 + \dots + q_n\beta_n + q_{n+1}\beta_{n+1}$ , where  $q_1, q_2, \dots, q_n, q_{n+1}$  is an orthogonal basis of  $K_{n+1}(A, b)$ . In this context  $q_1, q_2, \dots, q_n$  are known, while  $q_{n+1}$  is still to be determined.

Notice that  $\forall i = 1, \dots, n$  holds the following  $q_i^T w = q_i^T q_1\beta_1 + \dots + q_i^T q_n\beta_n + q_i^T q_{n+1}\beta_{n+1} = q_i^T q_i\beta_i = \beta_i$ , because of the orthonormality of the basis. Now we can compute  $q_{n+1}\beta_{n+1} = w - q_1\beta_1 - q_2\beta_2 - \dots - q_n\beta_n = z$ . If we choose  $\beta_{n+1} = \|z\|$  we have that  $q_{n+1} = \frac{z}{\|z\|}$  and this produces a valid choice of  $q_{n+1}$ . We still need to prove that it has norm 1 and it's orthogonal to all the other vectors in the basis.

An implementation of this algorithm is shown in Algorithm 13.1.1.

---

ALGORITHM 13.1.1 Arnoldi algorithm Matlab implementation.

---

```

1 function Q = arnoldi(A, b, n)
2 Q = zeros(length(b), n); %will be filled in
3 H = zeros(n+1, m);
4 Q(:, 1) = b / norm(b);
5 for j = 1 : n
6   w = A * Q(:, j);
7   for i = 1:j
8     % not what we showed earlier here, but stabler
9     betai = Q(:, i)' * w;
10    w = w - betai * Q(:, i);
11    H(i, j) = betai;
12  end
13  nrm = norm(w);
14  H(j+1, j) = nrm;
15  Q(:, j+1) = w / nrm;
16 end

```

---

Notice that we presented an algorithm where  $\beta_i = q_i^T w$  for  $i = 1, \dots, n$ , then  $w \leftarrow w - \beta_1 q_1 - \beta_2 q_2 - \dots - \beta_n q_n$ .

In the implementation we compute  $\beta_i = q_1^T w$ ,  $w \leftarrow w - \beta_1 q_1$ ,  $\beta_2 = q_2^T w$ ,  $w \leftarrow w - \beta_2 q_2$ . Why? It is more stable.

At step  $j$

$$Aq_j = \beta_{1,j}q_1 + \beta_{2,j}q_2 + \dots + \beta_{j,j}q_j + \beta_{j+1,j}q_{j+1} = Q \begin{bmatrix} \beta_{1,j} \\ \beta_{2,j} \\ \vdots \\ \beta_{j+1,j} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_n & q_{n+1} \end{bmatrix}$$

$$AQ = A \begin{bmatrix} q_1 & q_2 & \cdots & q_n & q_n \end{bmatrix} = Q \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} & \cdots & \beta_{1,n} \\ \beta_{2,1} & \beta_{2,2} & \beta_{2,3} & \cdots & \beta_{2,n} \\ 0 & \beta_{3,2} & \beta_{3,3} & \cdots & \beta_{3,n} \\ 0 & 0 & \beta_{4,3} & \cdots & \beta_{4,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \beta_{n+1,n} \end{bmatrix}$$

Hence the values  $q_i$ ,  $\beta_{i,j}$  computed by Arnoldi satisfy  $AQ_n = Q_{n+1}H$ , where  $H \in M(n+1, n, \mathbb{R})$  and looks like a triangular matrix plus a diagonal below,

namely  $H = \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} & \cdots & \beta_{1,n} \\ \beta_{2,1} & \beta_{2,2} & \beta_{2,3} & \cdots & \beta_{2,n} \\ 0 & \beta_{3,2} & \beta_{3,3} & \cdots & \beta_{3,n} \\ 0 & 0 & \beta_{4,3} & \cdots & \beta_{4,n} \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \beta_{n+1,n} \end{bmatrix}$

Notation:  $Q_{n+1} = \begin{bmatrix} q_1 & q_2 & \cdots & q_n & q_{n+1} \end{bmatrix}$ ,  $Q_n = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}$   
such that  $Q_n \in M(m, n, \mathbb{R})$ , while  $Q_{n+1} \in M(m, n+1, \mathbb{R})$ ,  $A \in M(m, \mathbb{R})$  and  $b \in M(m, n)$ .

FIGURE 13.1: We started from a matirx  $A$  which has many entries and it gets factorized by the product of two smaller matrices

For every matrix  $A$  there exist and Arnoldi factorization.

We don't like that we multiply  $A$  by two different matrices on the left and on the right, but we may improve the algorithm by using the same matrix.

$AQ_n = (Q_n|q_{n+1}) \cdot \text{matriToDraw} = Q_nH + q_{n+1} \cdot (0, \dots, 0, *) = Q_nH + q_{n+1} \cdot \text{tre}_{n+1}\beta_{j+1,j}$ , where  $e_i = 0 \dots 010 \dots 0$ .

Last remark:  $AQ_n = Q_{n+1}H$  doesn't allow a factorization of the matrix  $A$ , because  $Q_n$  isn't invertible, because  $Q_n$  is tall, thin and it doesn't have an inverse.

Once we see how Arnoldi works, we would like to see how to use it.

At some point this assumption must become false. For instance, assume we arrive at step  $m = n$ :  $q_1, q_2, \dots, q_m$  are a basis of  $\mathbb{R}^m$ , so

$$Aq_m = \beta_1 q_1 + \cdots + \beta_m q_m + 0$$

(without an additional term  $\beta_{m+1}q_{m+1}$ .)

Arnoldi factorization for  $m = n$  (assuming nothing broke down before):

$AQ_m = Q_mH$  is a factorization into square matrices, formally  $Q_m, H, A \in M(m, \mathbb{R})$ , so we can write something that we couldn't write before:

$A = A_m H Q_m^T$  we recall that  $H$  is upper triangular plus another diagonal before.

**Definition 13.1.1.** Let  $H \in M(m, \mathbb{R})$  such that  $H_{ij} = 0 \forall i > j + 1$ .  $H$  is called **Hessemberg matrix**.

**Fact 13.1.1.** *The QR factorization an Hessemberg matrix  $H \in M(m, \mathbb{R})$  can be computed in  $O(m^2)$  operations.*

This approach may be used, but in practice, since  $A$  is large and sparse we do not want to go until the end.

Let us analyze what happens if at some point  $K_{n+1}(A, b) = K_n(A, b)$ ?

$$Aq_n = \beta_1 q_1 + \cdots + \beta_n q_n + 0$$

for instance, if  $b$  is an eigenvector of  $A$ , it happens already at  $n = 1$ .

In the implementation Algorithm 13.1.1 if we have a breakdown at step  $n + 1$  this means that  $w = Aq_n$  has already enough  $q_i$ s, so  $q_{n+1}\beta_{n+1} = 0$ .

Problem:  $q_{n+1} = \frac{z}{\|z\|}$  division by 0. We need to change the definition  $\beta_{n+1} = \|z\| = 0$ . At this point we don't get a basis of the Krylov space, but we can still go on as "nothing happened", as long as these vectors are orthonormal. We go on until the end we get

$$AQ_m = H_m Q_m,$$

$$H_m = \left[ \begin{array}{cccc|cccccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & 0 & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array} \right].$$

The blocks are square.

This is good news, because it allows us to make a lot of manipulations.

$$A = Q_m H_m Q_m^T = \begin{bmatrix} Q_n & \hat{Q} \end{bmatrix} \begin{bmatrix} H_n & L \\ 0 & M \end{bmatrix} \begin{bmatrix} Q_n & \hat{Q} \end{bmatrix}^T$$

 Do you recall?

We said two things about block triangular matrices:

1. The eigenvalues of the matrix are the eigenvalues of the diagonal blocks, more formally,  $eigh(A) = eigh(H) = eigh(H_n) \cup eigh(M)$
2. We can solve linear systems on block matrices more easily. In this case,

$$\begin{aligned}
 x = A^{-1}b &= Q_m H^{-1} Q_m^T b = Q_m H^{-1} \begin{bmatrix} q_1^T \\ q_2^T \\ \vdots \\ q_m^T \end{bmatrix} b \stackrel{(1)}{=} Q_m H^{-1} \begin{bmatrix} \|b\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\
 &= \begin{bmatrix} Q_n & \hat{Q} \end{bmatrix} \begin{bmatrix} H_n^{-1} & -H_n^{-1}LM^{-1} \\ 0 & M^{-1} \end{bmatrix} \begin{bmatrix} \|b\| \mathbf{e}_1 \\ 0 \end{bmatrix} \\
 &= \begin{bmatrix} Q_n & \hat{Q} \end{bmatrix} \begin{bmatrix} \|b\| H_n^{-1} \mathbf{e}_1 \\ 0 \end{bmatrix} = Q_n \|b\| H_n^{-1} \mathbf{e}_1.
 \end{aligned} \tag{13.1.1}$$

where  $\stackrel{(1)}{=}$  follows from the fact that  $b = \|b\| q_1$  and  $q_1$  is orthogonal to all the other  $q_i$ .

An attentive reader may notice that at step  $j$ , when encountering  $B_{j+1,j} = 0$ , we know the following:

- some eigenvalues of  $A$  (those of  $H_n$ )
- Given an eigencouple  $v, \lambda$  such that  $H_n v = \lambda v$  then  $Q_n v$  is an eigenvector of  $A$  with eigenvalue  $\lambda$
- the solution  $x = Q_n H_n^{-1} \|b\| e_1$  of  $Ax = b$  and this is called lucky breakdown.

The point is that we have what we need to compute the solution at last step before the breakdown.

**Theorem 13.1.2. ‘Lucky breakdown’:** if it happens at an early step, we can solve linear systems (or compute some eigenvalues) cheaply: costs  $n$  matrix-vector products +  $O(mn^2)$ .

What happens when there is no breakdown? After  $n$  steps of Arnoldi,

1. if  $H_n v = \lambda v$  is an eigenpair of  $H_n$ .  $Q_n v, \lambda$  is an approximation of an eigenpair of  $A$ .
2.  $\tilde{x} = Q_n H_n^{-1} \|b\| e_1$  is an approximantion of the solution  $x$  of  $Ax = b$

## Chapter 14

# 5th of December 2018 — F. Poloni

### 💡 Do you recall?

**Arnoldi factorization:** we wanted to build an orthonormal base  $\{q_1, q_2, \dots, q_n\}$  of the Krylov space  $K_n(A, b) = \{b, Ab, A^2b, \dots, A^{n-1}b\}$ . We showed that the matrix  $A$  could be more or less factorized as  $AQ_n = Q_{n+1}H_n = Q_nH_n + q_{n+1}h_{n+1,n}e_{n+1}^T$ .

Moreover, we concluded that we could approximate the eigenvalues of the matrix  $A$  through the eigenvalues of matrix  $H_n$ , while the eigenvectors are obtained as  $Q_nv$ , where  $v$  is an eigenvector of the matrix  $H_n$ .

In the first part of the lecture we run some experiments on MatLab and we observed that the eigenvalues are approximated better and better starting from the boundaries. Notice that the approximation is not always good, but it is good enough if we take into account the complexity of the algorithm.

We are interested in explaining the convergence of Arnoldi method.

## 14.1 Convergence of Arnoldi

The eigenvalues of  $H_n$  are eigenvalues of a “nearby matrix” obtained by taking  $\tilde{H}_m$  (result of the full process) and replacing  $\tilde{H}_{n+1,n}$  with zero.

$$\tilde{H}_m = \left[ \begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & \textcircled{0} & & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array} \right] \rightarrow H_m = \left[ \begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & \textcircled{*} & & * & \cdots & * & * \\ & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array} \right]$$

We expect that this change does not lead to a significative change in the eigenvalues, in other words, the eigenvalues of  $\tilde{H}_m$  differ from the eigenvalues of  $H_m$  by  $|h_{n+1,n}|$ . Formally,  $\|\tilde{H}_m - H_m\| = h_{n+1,n}$ .

### 14.1.1 Better explanation

The space  $K_n(A, b) = \text{span}(b, Ab, \dots, A^{n-1}b)$  contains the right “features” to represent the eigenvectors of  $A$  with largest eigenvalues: if  $A = V\Lambda V^{-1}$  is diagonalizable, then

$$\begin{aligned} A^k b &= (V\Lambda V^{-1}) \cdots (V\Lambda V^{-1}) b \\ &= V\Lambda^k V^{-1} b \\ &= \begin{pmatrix} V^1 & V^2 & \cdots & V^m \end{pmatrix} \cdot \begin{pmatrix} \lambda_1^n & & & \\ & \lambda_2^n & & \\ & & \ddots & \\ & & & \lambda_m^n \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} \quad (14.1.1) \\ &= V^1 \lambda_1^k c_1 + V^2 \lambda_2^k c_2 + \cdots + V^m \lambda_m^k c_m \end{aligned}$$

where  $c = V^{-1}b$ .

$A^k b$  is a linear combination of the eigenvectors  $V^i$  in which those with largest  $|\lambda_i|$  are “more prominent”, in other words as  $n$  increases the components involving the largest  $|\lambda_i|$ s grow faster.

This also tells us that  $\text{span}(V^1, V^2, \dots, V^m)$  (that are the eigenvectors associated to largest eigenvalues in modulus) “represent well”  $K_m(A, b) = \text{span}(b, Ab, \dots, A^{m-1}b)$ .

We cannot provide a formal proof of the convergence, because there are some counter examples.

**Example 14.1.1.** 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. In the last step the eigenvalues become correct.

Notice that in this case the absolute values of the eigenvalues are the same, hence we are not able to do the trick explained above.

**Observation 14.1.1** (Matlab syntax). The command  $[V, D] = \text{eigs}(A)$  does not work on sparse matrices  $A$ . In this case we may run Arnoldi method and use the best values obtained by Arnoldi:  $[V, D] = \text{eigs}(A, n)$ , 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  $\text{eigs}$  and pass to it a  $\lambda$ -function which computes the matrix-vector product and this is useful when the matrix and the vector have a particular shape.

We would like to understand how to compute some eigenvalues that are not the biggest.

**Lemma 14.1.1.** 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  $A^k$ .

*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  $\lambda A$  both sides:  $\lambda A A^{-1}\mathbf{v} = \lambda A \lambda^{-1}\mathbf{v} \Leftrightarrow \lambda\mathbf{v} = \lambda\lambda^{-1}A\mathbf{v}$ , which is true by definition of eigenvalue/vector of  $A$ ;
3. (by induction)  $A^2\mathbf{v} = A(A\mathbf{v}) = A \cdot \lambda\mathbf{v} = \lambda A\mathbf{v} = \lambda\lambda\mathbf{v} = \lambda^2\mathbf{v}$ .

□

This lemma gives us the chance to use Arnoldi algorithm to compute the eigenvalues of a slightly modified matrix  $B = (A - \mu I)^{-1}$ . If  $(\lambda, v)$  is an eigenpair of  $A$ , then  $(\lambda - \mu^{-1}, v)$  is an eigenpair of  $B$ .

When is that  $(\lambda - \mu^{-1})$  is large? When  $\lambda$  and  $\mu$  are close.

If  $A$  has eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_m$ , then the eigenvalues of  $B$  are  $\nu_1 = \frac{1}{\lambda_1 - \mu}, \nu_2 = \frac{1}{\lambda_2 - \mu}, \dots, \nu_m = \frac{1}{\lambda_m - \mu}$ .

The problem is that such matrix  $B$  is not sparse when  $A$  is sparse.

We perform a trick to overcome this issue: we pass the function `eig` a  $\lambda$ -function that computes the product  $Bz = A - \mu I^{-1}z$  without computing  $B$  explicitly. The idea is to use factorizations:  $A - \mu I = LU$ , then  $Bz = U^{-1}(L^{-1}z)$ , that we can compute by back-substitution.

```
% computes 5 eigenvalues closest to mu=2
» f1 = eigs(A, 5, mu);
» f1 = eigs(A, 5, 'SM'); %smallest magnitude
» f1 = eigs(A, 5, 'LM'); %largest magnitude
```

Notice that `eigs(A, 6, 1)` needs to factorize  $A - 1 \cdot I = LU$ , which might induce a lot of fill-in.

As final observation, the equivalents to Matlab's `eigs` function are `scipy.linalg.eigs` for Python and `arpack` for C/C++ and Fortran.

# Chapter 15

## 7th of December 2018 — F. Poloni

In this lecture we are interested in using Arnoldi method to solve linear systems.

We can use the *sparse eigenvalues function* that we saw in last lecture and the **Generalized Minimum RESidual** (GMRES).

Our task is to approximate the solution of a large-scale linear system of the form  $Ax = b$  and our approach is to look for “the closest thing to solution” inside  $K_n(A, b)$ .

Through Arnoldi of  $A$ ,  $b$  and  $n$ , we obtained  $[Q, H]$  and we can approximate the solution  $x$  as  $Q_1y_1 + Q_2y_2 + \dots + Q_ny_n = Qy$ , which is a good approximation of the solution inside  $K_n(A, b)$ , formally

$$\min_{x \in K_n(A, b)} \|Ax - b\|, \quad x = Q_n y.$$

which is equivalent to  $\min_{y \in \mathbb{R}^n} \|AQ_n y - b\|$ .

We can perform some more reductions and:

$$\begin{aligned} \|AQ_n y - b\| &\stackrel{(1)}{=} \|Q_{n+1}\underline{H}_n y - b\| \\ &\stackrel{(2)}{=} \|Q_{n+1}\underline{H}_n y - Q_{n+1}\|b\|e_1\| \\ &= \|Q_{n+1} \cdot (\underline{H}_n y - \|b\|e_1)\| \\ &\stackrel{(3)}{=} \|\underline{H}_n y - \|b\|e_1\|. \end{aligned} \tag{15.0.1}$$

where  $\stackrel{(1)}{=}$  is due to the equivalence  $AQ_n = Q_{n+1}H_n$ , with  $H_n \in M(n+1, n)$ ,  $\stackrel{(2)}{=}$  follows from the fact that  $q_1 = \frac{b}{\|b\|}$  and  $\stackrel{(3)}{=}$  is explained recalling that  $Q_{n+1}$  is an orthogonal rectangular matrix in  $M(mn+1)$  and  $\|z\| = \|Q_{n+1}z\|$ , since  $z^T z = z^T Q_{n+1}^T Q_{n+1} z$ .

We got a LS problem of size  $(n+1) \times n$  (small), where  $y \in \mathbb{R}^n$  and  $e_1 \in \mathbb{R}^{n+1}$ ; moreover  $\underline{H}$  has the following shape

$$H = \left[ \begin{array}{cccc|cccc} * & \cdots & * & * & * & \cdots & \cdots & * \\ * & \cdots & * & * & * & \cdots & \cdots & * \\ 0 & \ddots & * & * & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & * & * & * & \cdots & \cdots & * \\ \hline & & & \textcircled{*} & * & \cdots & * & * \\ 0 & & & & * & \cdots & * & * \\ & & & & 0 & \ddots & * & * \\ & & & & 0 & 0 & * & * \end{array} \right]$$

hence it is quite sparse.

$gr(H)$  can be computed in  $O(n^2)$  using the fact that  $H$  is ‘almost triangular’ (Hessenberg matrix), although it is not a big optimization, since  $n$  Arnoldi steps need to be computed first.

Notice that instead of doing a QR at the end, we can compute QRs of  $H_1, H_2, \dots$  and update them at each step. This allows us to compute at each step a residual  $\|Ax_n - b\|$  that we can use as stopping criterion.



### Something on Matlab ...

Matlab has `gmres(A, b)` (and Python has `scipy.sparse.linalg.gmres`).

To estimate the convergence of GMRES we can see  $x$  as a polynomial ( $x = p(A)b$ , such that  $p(t) = \alpha_0 + \alpha_1 t + \cdots + \alpha_{n-1} t^{n-1}$  is a polynomial of degree  $n-1$ ).

As far as the residual is concerned  $Ax - b = A \cdot p(A) \cdot b - b = A \cdot (\alpha_0 I + \alpha_1 A + \cdots + \alpha_{n-1} A^{n-1}) \cdot b - b = q(A) \cdot b$ , where  $q(t) = t \cdot p(t) - 1$ . If  $A = V \Lambda V^{-1}$

diagonalizable, then  $A^k = V \cdot \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_m^k \end{bmatrix} V^{-1}$  and

$$q(A) = V \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} V^{-1}.$$

All this computation was needed to write the residual GMRES in a clearer form:

$$\begin{aligned}
\min_{x \in K_n(A, b)} \|Ax - b\| &= \min_{\substack{q(x) = xp(x)-1 \\ \text{of degree } \leq n}} \|Ap(A)b - b\| \\
&= \min_{\substack{q(x) = xp(x)-1 \\ \text{of degree } \leq n}} \|q(A)b\| \\
&\leq (\min_{\dots} \|q(A)\|) \cdot \|b\| \\
&= \min_{\dots} \left\| V \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} V^{-1} \right\| \quad (15.0.2) \\
&\leq \min_{\dots} \|V\| \cdot \left\| \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} \right\| \cdot \|V^{-1}\| \\
&\leq K(V) \cdot \left\| \min_{\dots} \begin{bmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_m) \end{bmatrix} \right\|
\end{aligned}$$

If  $A$  has very few distinct eigenvalues ( $k \leq n$  of them), then we can find  $q$  such that  $q(\lambda_i) = 0$  for all  $i$  and  $q(0) = -1$ , hence  $n$  steps of GMRES give us the exact solution.

If  $A$  has eigenvalues clustered in  $n$  points in the plane, we can find a polynomial  $q$  such that  $q(\lambda_i)$  is small for all  $i$ .

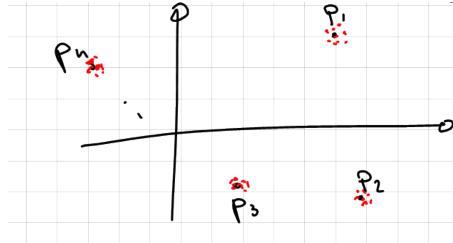


FIGURE 15.1: In this picture the eigenvalues are clustered around  $P_1, P_2, P_3$  and  $P_4$ . We can find a polynomial  $q$  such that  $q(\text{red points}) \approx 0$ .

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  $Ax = b$  to  $PAx = Pb$ . 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  $A^{-1}b$ .

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.

# Chapter 16

## 13th of December 2018 — F. Poloni

### 16.1 Lanczos algorithm

#### 💡 Do you recall?

In last lecture we saw how to factorize a matrix  $A \in M(m, \mathbb{R})$  with Arnoldi (i.e.  $AQ_n = Q_{n+1}\underline{H}_n = Q_n\underline{H}_n + h_{n+1,n}q_{n+1}e_1^T$ ).

If  $A$  is **symmetric**, something special happens:  $\underline{H}_n = Q_m^T A Q_m$  is symmetric as well, so it is a **tridiagonal** matrix. This improves the complexity of the Arnoldi process, because many iterations of the for loop ( $j = 1 : n$ ) are not needed anymore, we need only two iterations.

This symmetric variant of Arnoldi is called *Lanczos algorithm*, and such algorithm reduces the cost to  $n$  matrix products +  $O(mn)$ .

Suppose  $A = A^T$  is positive definite. Then, we can find the solution to  $Ax = b$  by minimizing the (strictly convex) function  $f(x) = \frac{1}{2}x^T Ax - b^T x$ .

Surprisingly, conjugate gradient on this problem can be interpreted as a Krylov subspace method.

The pseudocode of such algorithm can be found in Algorithm 16.1.1, where  $x_k$  is the current iterate,  $r_k = b - Ax_k = -\nabla f(x_k)$  is the residual and  $d_k$  is the search direction.

---

ALGORITHM 16.1.1 Pseudocode for the conjugate gradient method.

---

```

1: procedure CG_ITERATION
2:    $x_0 \leftarrow 0;$ 
3:    $r_0 \leftarrow b;$ 
4:    $d_0 \leftarrow b;$ 
5:   for  $k = 1:n$  do
6:      $\alpha_k \leftarrow (r_{k-1}^T r_{k-1}) / (d_{k-1}^T A d_{k-1});$ 
7:      $x_k \leftarrow x_{k-1} + \alpha_k d_{k-1};$ 
8:      $r_k \leftarrow r_{k-1} - \alpha_k A d_{k-1};$ 
9:      $\beta_k \leftarrow (r_k^T r_k) / (r_{k-1}^T r_{k-1});$ 
10:     $d_k \leftarrow r_k + \beta_k d_{k-1};$ 
11:   end for
12: end procedure

```

---

Notice that the search direction (line 10) is modified adding a multiple of the previous search direction to the residual and  $\beta_k$  is chosen such that  $d_k$  and  $d_{k-1}$  are  $A$ -orthogonal (formally,  $d_k^T A d_{k-1} = 0$ ).

Conversely, the next point is chosen in order to minimize the objective function  $f(x_{k-1} \alpha_k d_{k-1})$ .

As far as the complexity is concerned, the space complexity is constant (three vectors) and the time complexity is  $O(mn)$ .

**Theorem 16.1.1.**  $K_k(A, b) = \text{span}(x_1, x_2, \dots, x_k) = \text{span}(d_0, d_1, \dots, d_{k-1}) = \text{span}(r_0, r_1, \dots, r_{k-1})$ .

**Theorem 16.1.2.** *The residuals are orthogonal and the search directions are  $A$ -orthogonal. Formally,  $r_j^T r_k = d_i^T A d_k = 0, \forall i < k$ .*

*Proof.* By induction: Let us assume we proved the thesis  $r_j^T r_k = 0$  for  $k-1, k-2, \dots, 0$ . Since  $x_k = x_{k-1} \alpha_k d_{k-1}$ , the residual  $r_k = b - Ax_k = b - A(x_{k-1} + \alpha_k d_{k-1}) = b - Ax_{k-1} - \alpha_k A d_{k-1} = r_{k-1} - \alpha_k A d_{k-1}$ .

Let us compute  $r_j^T r_k = r_j^T \cdot (r_{k-1} - \alpha_k A d_{k-1})$ .

- If  $i < k-1$  then  $r_j^T r_{k-1} - r_j^T \alpha_k A d_{k-1} = 0$ , because the first term is 0 from induction hypothesis and  $r_j^T \alpha_k A d_{k-1} = 0$ , because  $r_j \in K_{k-1}(A, b) = \text{span}(d_0, d_1, \dots, d_{k-2})$ .
- If  $i = k-1$ ,  $0 = r_{k-1}^T \cdot (r_{k-1} - \alpha_k A d_{k-1}) = r_{k-1}^T r_{k-1} - \alpha_k r_{k-1}^T A d_{k-1}$  holds if  $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T A d_{k-1}}$ . We are left with proving that  $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{r_{k-1}^T A d_{k-1}} = \alpha_k = \frac{r_{k-1}^T r_{k-1}}{d_{k-1}^T A d_{k-1}}$ .

This is true, since  $d_{k-1} = r_{k-1} + \beta_{k-1} d_{k-2}$ , so  $d_{k-1}^T A d_{k-1} = (r_{k-1} + \beta_{k-1} d_{k-2})^T A d_{k-1} = r_{k-1}^T A d_{k-1} + \beta_{k-1} d_{k-2}^T A d_{k-1}$  and the second part is equal to 0 by induction.

□

Notice that this base is not orthonormal, we need to rescale it to obtain an orthonormal one, moreover,  $\frac{1}{\|r_i\|} r_i$  coincides (up to a sign) with the  $q_i$  obtained with Arnoldi.

We are left with writing the equation we need to solve at each iteration, namely we need to ensure that  $r_k = b - Ax_k$  is orthogonal to all vectors of  $K_k(A, b)$  which is equivalent to requiring  $Q_k^T \cdot (b - Ax_k) = 0$  or, equivalently,  $\|b\| \cdot e_1 = H_k y_k$ .

In figure Figure 16.1 we can see a comparison between Arnoldi algorithm and the conjugate gradient.

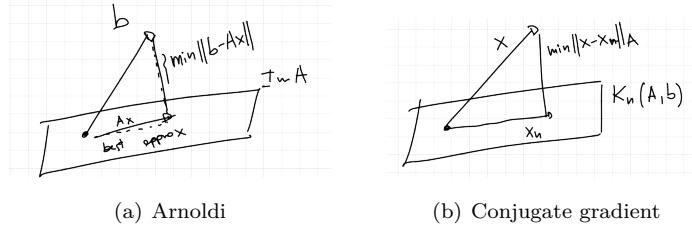


FIGURE 16.1: Traditional orthogonality (Arnoldi) leads to the minimization of the 2-norm, while in the conjugate gradient we impose  $A$ -orthogonality and we get a good approximation in several norms.

As far as convergence speed is concerned,

**Theorem 16.1.3.**  $x_k$  is the best approximation of the exact (and unknown) solution  $x$  to  $Ax = b$  in  $K_k(A, b)$  in the  $A$ -norm, i.e.  $x_k = \arg \min_{z \in K_k(A, b)} (x - z)^T A (x - z)$

**Theorem 16.1.4.** Let  $\lambda_{\max}$ ,  $\lambda_{\min}$  be the maximum/minimum eigenvalue of  $A$ ; then, CG converges with rate

$$\|x - x_k\| \leq \left( \frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}} \right)^k \|x - x_0\|.$$

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.

As for GMRES, if  $A$  has only  $n$  different eigenvalues, then this minimum reaches 0 after  $n$  steps. If the eigenvalues of  $A$  are ‘clustered’, one can construct polynomials such that  $q(\lambda)$  is small for each  $\lambda$  then fast convergence is implied.