

1 Linear systems

1.1 Introduction to linear systems

A linear system can be written as

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

where \mathbf{A} is a matrix of size $m \times n$ (we suppose $m \geq n$) and \mathbf{x} is a column vector of length n and \mathbf{b} is a column vector of length m . \mathbf{x} represents the unknown solution while \mathbf{b} is a given vector. This form can be expanded as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m \end{aligned}$$

We are interested in:

- Existence and uniqueness of a solution
- Numerical methods to find the solution
- Conditioning of the problem

We separately consider the case of square linear systems ($m = n$) and least squares problem ($m > n$).

1.2 Square linear systems

Proposition 1.1. The solution of a linear system

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

with \mathbf{A} of dimension $n \times n$, \mathbf{x} and \mathbf{b} of dimension n exists and is unique iff one of the following conditions holds:

- \mathbf{A} is non-singular
- $\text{rank}(\mathbf{A}) = n$
- The system $\mathbf{A}\mathbf{x} = \mathbf{0}$ admits only the solution $\mathbf{x} = \mathbf{0}$

The solution can be algebraically computed in the following way

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

The problem with this kind of method is that computing the inverse of \mathbf{A} can be expensive for large values of n .

Another method to compute the solution of a system is *Cramer's rule*:

$$x_i = \frac{\det A_i}{\det A}, \quad i = 1, \dots, n$$

where A_i is obtained from \mathbf{A} by replacing the i -th column by \mathbf{b}

As for the previous one, the computation of determinants can be quite expensive for large values of n .

We will consider two different approaches to find the solution of a linear system:

- *direct methods*. They yield the solution in a finite number of steps; they are more precise but more expensive in terms of computational cost
- *iterative methods*. They require (in principle) an infinite number of steps; they are less precise and less expensive

It is very important to perform also an error analysis on the obtained results. It generally includes two parts:

- *Rounding or arithmetic errors*. They depend on the algorithm steps. An algorithm producing an arithmetic error limited by a constant is called a **stable algorithm**.
- *Inherent errors*. They are due to the errors in the data representation and the DO NOT depend on the algorithm. For this reason, if the inherent error is large, we speak about **ill-posed problem**.

1.3 Direct methods

In the case of direct methods we factorize the matrix \mathbf{A} as in section ?? . Let $\mathbf{A} = \mathbf{LU}$, then the solution of the linear system $\mathbf{Ax} = \mathbf{b}$ can be computed by solving two triangular systems

$$\mathbf{Ly} = \mathbf{b} \text{ (forward substitutions algorithm)}$$

$$\mathbf{Ux} = \mathbf{y} \text{ (backward substitutions algorithm)}$$

Forward substitution.

Esempio 2.1 *Sostituzione in avanti.*

$$\begin{pmatrix} \ell_{1,1} & 0 & 0 \\ \ell_{2,1} & \ell_{2,2} & 0 \\ \ell_{3,1} & \ell_{3,2} & \ell_{3,3} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\begin{array}{ll} y_1 = b_1 / \ell_{1,1} & 1 \text{ operazione} \\ \downarrow & \\ y_2 = (b_2 - \ell_{2,1} * y_1) / \ell_{2,2} & 3 \text{ operazioni} \\ \downarrow & \\ y_3 = (b_3 - \ell_{3,1} * y_1 - \ell_{3,2} * y_2) / \ell_{3,3} & 5 \text{ operazioni} \end{array}$$

Backward substitution.

$$\begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & u_{3,3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$\begin{array}{ll} x_3 = y_3 / u_{3,3} & 1 \text{ operazione} \\ \downarrow & \\ x_2 = (y_2 - u_{2,3} * x_3) / u_{2,2} & 3 \text{ operazioni} \\ \downarrow & \\ x_1 = (y_1 - u_{1,2} * x_2 - u_{1,3} * x_3) / u_{1,1} & 5 \text{ operazioni} \end{array}$$

In the case of **pivoting algorithm**, we have that **PA = LU**. Then the solution of the linear system **PAX = Pb** can be computed by solving two triangular systems

$$\mathbf{Ly} = \mathbf{Pb} \text{ (forward substitutions algorithm)}$$

$$\mathbf{Ux} = \mathbf{y} \text{ (backward substitutions algorithm)}$$

1.4 Iterative methods

Iterative methods are numerical methods that approximate a solution using a procedure involving several (or an infinite amount of) steps.

The basic idea of iterative methods is to construct a sequence of vectors \mathbf{x}_k enjoying the property of *convergence*

$$\mathbf{x}^* = \lim_{k \rightarrow \infty} \mathbf{x}_k$$

where \mathbf{x}^* is the exact solution and the starting guess \mathbf{x}_0 is given.

In general the sequence \mathbf{x}_k is obtained as $\mathbf{x}_k = g(\mathbf{x}_{k-1})$, $k = 1, \dots$ where g is a particular function or set of operations acting on \mathbf{x}_{k-1} . Different classes of iterative methods are defined for different expressions of g . In general, an iterative method converges for matrices with fixed properties (such as on the shape, on the eigenvalues, ...).

Iterative methods are sensitive to both algorithmic and truncation errors.

Examples of iterative methods (the most common):

- *stationary iterative methods*, they take the form

$$\mathbf{x}_{k+1} = \mathbf{B}\mathbf{x}_k + \mathbf{f}$$

where \mathbf{B} is called *iteration matrix* and \mathbf{f} is a vector obtained from \mathbf{b}

- *gradient-like methods*, they take the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

where $\alpha_k \in \mathbb{R}$, \mathbf{p}_k is a vector called *direction*.

If \mathbf{A} is symmetric and positive definite and the vectors \mathbf{p}_k have the *conjugacy* property, i.e:

$$\mathbf{p}_i^T \mathbf{A} \mathbf{p}_k = 0 \quad \text{if } i \neq k$$

then the method is called *Conjugate Gradients*.

All these methods require one matrix-vector multiplication per iteration. Hence, if \mathbf{A} is a *full* matrix the computational complexity is $O(n^2)$ per iteration.

Stopping criteria for iterative methods for linear systems are:

- defined the residual $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ at iteration k , stop the algorithm when
 - $\|\mathbf{r}_k\| \leq \epsilon$ (absolute criterion)
 - $\frac{\|\mathbf{r}_k\|}{\|\mathbf{b}\|} \leq \epsilon$ (relative criterion)
- $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \tau$ (absolute criterion)

Sparse matrices A matrix A is called *sparse* if it has a very small percentage (i.e. $0.05 - 0.06\%$) of non-null elements. In this case, the matrix is not fully stored, but only the non-null elements and their indices are stored. The matrix-vector multiplication requires k multiplications, where k is the number of non-null elements.

1.5 Inherent errors in linear systems

We remind that inherent errors are due to the errors in the data representation. They **do not** depend on the algorithm used for computing the result. Really, the analysis of inherent errors is performed by supposing to use exact arithmetic to compute the result.

Consider now a linear system $A\mathbf{x} = \mathbf{b}$. What happens to the solution \mathbf{x} when A and/or \mathbf{b} slightly change (e.g. due to machine precision)?

Suppose that A doesn't change while \mathbf{b} changes in $\mathbf{b} + \Delta\mathbf{b}$. The linear system becomes

$$A(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}, \quad \Delta\mathbf{x} \text{ inherent error}$$

We want to compare $\left\| \frac{\Delta\mathbf{x}}{\mathbf{x}} \right\|$ and $\left\| \frac{\Delta\mathbf{b}}{\mathbf{b}} \right\|$ to see how much the solution has changed with respect to \mathbf{b} . Let's subtract $A\mathbf{x} = \mathbf{b}$ to $A(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}$

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

Then we have

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

and from the linear system equation

$$\|A\mathbf{x}\| = \|\mathbf{b}\| \implies \|\mathbf{b}\| = \|A\mathbf{x}\| \leq \|A\| \|\mathbf{x}\| \implies \|\mathbf{x}\| \geq \left\| \frac{\mathbf{b}}{A} \right\|$$

Thus we have

$$\left\| \frac{\Delta\mathbf{x}}{\mathbf{x}} \right\| \leq \|A^{-1}\| \cdot \|A\| \cdot \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} = K(A) \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

Definition 1.1. The *condition number* of a matrix $A \in \mathbb{C}^{n \times n}$ is defined as

$$K(A) = \|A\| \|A^{-1}\|$$

where $\|\cdot\|$ is a matrix norm. In general $K(A)$ depends on the choice of the norm, indicated by a subscript.

Notice that $K(A) \geq 1$ since

$$1 = \|AA^{-1}\| \leq \|A\| \|A^{-1}\| = K(A)$$

The condition number measures how sensitive a function is to changes/errors in the input and how much the output changes as a result of the operations performed. A *well-conditioned* system is a system where $K(A)$ is small, while an *ill-conditioned* system is a system where $K(A)$ is large.

If $K(A)$ is very large, the matrix A is near (with respect to a norm) a singular matrix and its columns are quasi-linearly dependent. Regularization techniques can be used in order to reduce $K(A)$.

Example 1.1. The condition number for p-norm with $p = 2$ is

$$\begin{aligned} \|A\|_2 &= \sqrt{\rho(A^T A)} = \sigma_{max} \\ \|A^{-1}\|_2 &= \frac{1}{\sqrt{\rho(A^T A)}} = \frac{1}{\sigma_{min}} \\ K_2(A) &= \|A\| \|A^{-1}\| = \frac{\sigma_{max}}{\sigma_{min}} \end{aligned}$$

where λ_{max} and λ_{min} are the maximum and minimum singular values of A

1.6 Linear least squares

Consider an *overdetermined system*

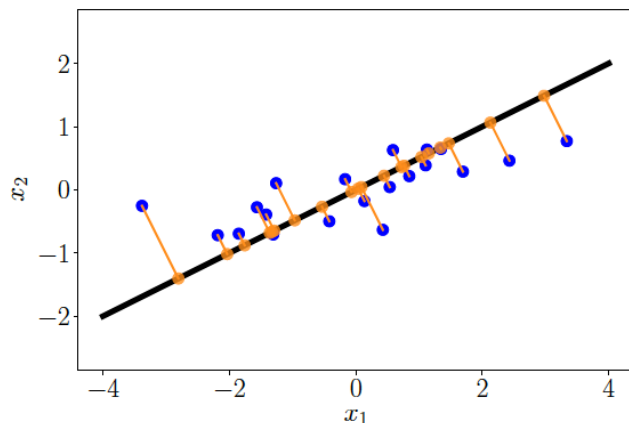
$$A\mathbf{x} = \mathbf{b}$$

where A has dimension $m \times n$ with $m > n$, \mathbf{x} of dimension n and \mathbf{b} of dimension m . Such a system usually has no solution, meaning that \mathbf{b} does not lie on the subspace spanned by the columns of A (denoted by $\text{Col}(A)$ from now on). Since no exact solution exists for this problem, we would like to compute the best approximate solution. We need some tools.

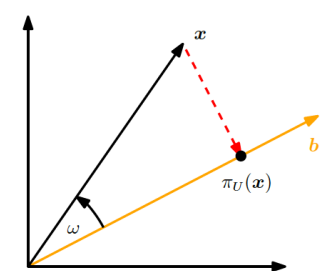
1.6.1 Orthogonality and projections

Projections are an important class of linear transformations. In machine learning one often deals with high-dimensional data which is hard to visualize

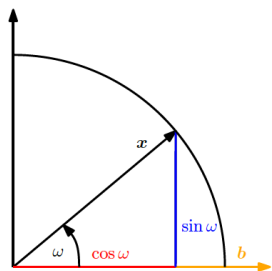
and analyze. Oftentimes, only a few dimensions contain useful information, meaning that other dimensions are not essential to understanding the data. When applying data compression techniques we want to minimize the loss of information by finding the most informative dimensions of the data first.



Orthogonal projection onto a subspace of dimension 1.



(a) Projection of $x \in \mathbb{R}^2$ onto a subspace U with basis vector b .



(b) Projection of a two-dimensional vector x with $\|x\| = 1$ onto a one-dimensional subspace spanned by b .

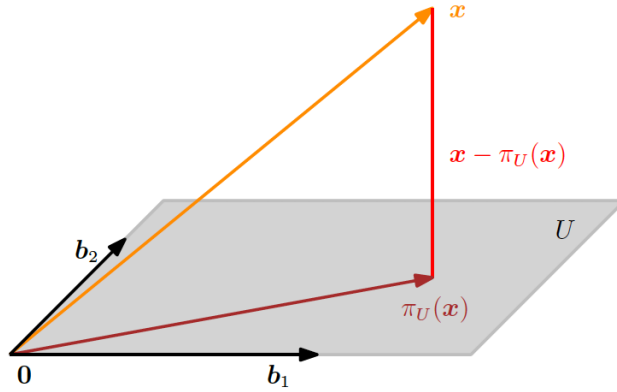
- The projection $z = \Pi_U(\mathbf{x})$ is the closest point to \mathbf{x} of the subspace U generated by \mathbf{b} by considering as the measure of the distance the 2-norm ($\|\cdot\|_2$). The vector $\mathbf{z} - \mathbf{x}$ (the red dashed one in the figure) is orthogonal to the subspace U (and to the vector \mathbf{b} generating U). Hence, the scalar product: $\langle \mathbf{z} - \mathbf{x}, \mathbf{b} \rangle = 0$
- The vector \mathbf{z} is a vector of the subspace generated by \mathbf{b} , hence $\mathbf{z} = \alpha \mathbf{b}$, $\alpha \in \mathbb{R}$.

We can find α in the following way

$$\langle \mathbf{x} - \alpha \mathbf{b}, \mathbf{b} \rangle = 0 \implies \langle \mathbf{x}, \mathbf{b} \rangle - \alpha \langle \mathbf{b}, \mathbf{b} \rangle = 0$$

$$\alpha = \frac{\langle \mathbf{x}, \mathbf{b} \rangle}{\langle \mathbf{b}, \mathbf{b} \rangle} = \frac{\mathbf{x}^T \mathbf{b}}{\|\mathbf{b}\|^2}$$

Example when projecting from \mathbf{R}^3 to \mathbf{R}^2 .



We now generalize to sub-spaces of dimension possibly greater than one.

Theorem 1.1. (*Best approximation theorem*) Let W a subspace with dimension p of \mathbb{R}^n , $\mathbf{y} \in \mathbb{R}^n$ and $\hat{\mathbf{y}} = \text{proj}_W(\mathbf{y})$. Then $\|\mathbf{y} - \hat{\mathbf{y}}\| < \|\mathbf{y} - \mathbf{v}\|$, $\forall \mathbf{v} \in W$, $\mathbf{v} \neq \hat{\mathbf{y}}$.

This means that $\hat{\mathbf{y}}$ is the best approximation of \mathbf{y} in W .

Example 3.10 MML book.

The idea is to use projections to find the vector in $\text{Col}(\mathbf{A})$ (i.e. the subspace generated by set of all the columns of \mathbf{A}) which is closest to \mathbf{b} . As already seen in the previous chapter, this vector is indeed the orthogonal projection of \mathbf{b} onto the subspace $\text{Col}(\mathbf{A})$.

1.6.2 The linear least squares problem

When \mathbf{A} has size $m \times n$ with $m > n$, it is not possible to find a solution of the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. The orthogonal projections allow to find a vector \mathbf{x} that is *near* the system solution (in the sense of the 2-norm).

The problems is then formulated as to compute $\hat{\mathbf{x}}$ so that:

$$\hat{\mathbf{x}} = \operatorname{argmin}_x \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \quad (1)$$

For what concerns existence and uniqueness of the solution of the linear least squares problem, two different cases are possible:

- $\operatorname{rank}(\mathbf{A}) = n$, meaning that every column of \mathbf{A} is linearly independent. A unique solution exists $\forall \mathbf{b} \in \mathbb{R}^m$. Let \mathbf{x}^* be the approximate solution to the linear system, we have

$$\mathbf{A}\mathbf{x}^* = \operatorname{proj}_{\operatorname{Col}(\mathbf{A})}(\mathbf{b})$$

This means that

$$\mathbf{b} - \mathbf{A}\mathbf{x}^* \in [\operatorname{Col}(\mathbf{A})]^\perp \implies \mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}^*) = 0$$

Thus, we obtain the *normal equations*

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

which is a linear system in n equations and n unknowns. $\mathbf{A}^T \mathbf{A}$ is called *Gramian matrix* of \mathbf{A} and is non-singular, positive definite and symmetric. The approximate solution can then be obtained

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

- $\operatorname{rank}(\mathbf{A}) = k$, $k < n$, meaning that there is an infinite number of solutions (∞^{n-k}). We can find the least squares solution to the problem by using the pseudoinverse of \mathbf{A} obtained via SVD

$$\mathbf{A}\mathbf{x} = \mathbf{b} \implies \mathbf{A}^+ \mathbf{A} \mathbf{x}^* = \mathbf{A}^+ \mathbf{b} \implies \mathbf{x}^* = \mathbf{A}^+ \mathbf{b} \implies \mathbf{x}^* = \sum_{i=1}^k \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

It is possible to modify the linear least squares problem (1) by introducing a weight matrix with the aim to give different weights to the components of \mathbf{x} .

$$\hat{\mathbf{x}} = \operatorname{argmin}_x \|\mathbf{W} \mathbf{A} \mathbf{x} - \mathbf{b}\|_2^2 \quad (2)$$

where \mathbf{W} is an invertible matrix. In this case the *weighted normal equations* are:

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

In general the sequence \mathbf{x}_k is obtained as $\mathbf{x}_k = g(\mathbf{x}_{k-1})$, $k = 1, \dots$ where g is a particular function or set of operations acting on \mathbf{x}_{k-1} . Different classes of iterative methods are defined for different expressions of g . In general, an iterative method converges for matrices with fixed properties (such as on the shape, on the eigenvalues, ...).

Iterative methods are sensitive to both algorithmic and truncation errors.

Examples of iterative methods (the most common):

- *stationary iterative methods*, they take the form

$$\mathbf{x}_{k+1} = \mathbf{B}\mathbf{x}_k + \mathbf{f}$$

where \mathbf{B} is called *iteration matrix* and \mathbf{f} is a vector obtained from \mathbf{b}

- *gradient-like methods*, they take the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

where $\alpha_k \in \mathbb{R}$, \mathbf{p}_k is a vector called *direction*.

If \mathbf{A} is symmetric and positive definite and the vectors \mathbf{p}_k have the *conjugacy* property, i.e:

$$\mathbf{p}_i^T \mathbf{A} \mathbf{p}_k = 0 \quad \text{if } i \neq k$$

then the method is called *Conjugate Gradients*.

All these methods require one matrix-vector multiplication per iteration. Hence, if \mathbf{A} is a *full* matrix the computational complexity is $O(n^2)$ per iteration.

Stopping criteria for iterative methods for linear systems are:

- defined the residual $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ at iteration k , stop the algorithm when
 - $\|\mathbf{r}_k\| \leq \epsilon$ (absolute criterion)
 - $\frac{\|\mathbf{r}_k\|}{\|\mathbf{b}\|} \leq \epsilon$ (relative criterion)
- $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \tau$ (absolute criterion)

3.1 Gaussian Elimination Method or LU factorization

Definition 3.1. Let $A \in \mathbb{R}^{n \times n}$. If A is non singular and all the principal minors $A_i, i = 1 \dots n - 1$ are non singular, then there exists two matrices L and U , with L lower triangular with $l_{ii} = 1, i = 1, \dots, n$ and U upper triangular non singular, such that:

$$A = LU$$

We call this an LU-*factorization* of A .

The matrices L and U can be computed in $n - 1$ steps with the so called Gaussian Elimination Method (GEM) having a computational cost of $O(n^3/3)$.

The matrix U can be computed as follows:

1. We define $M^{(k)}$ the matrix of multipliers with elements

$$- m_{ii} = 1, i = 1, \dots, n$$

$$- m_{ik} = -\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, i = k + 1, \dots, n$$

$$- m_{ij} = 0, \text{ otherwise}$$

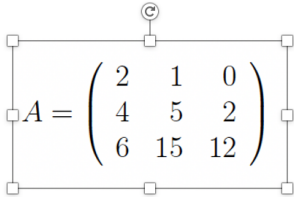
2. We compute $A^{(k+1)} = M^{(k)} A^{(k)}$

3. We repeat steps 1-2 until we obtain $A^{(n)} = U$

We therefore have

$$\begin{aligned} M^{(n-1)} M^{(n-2)} \dots M^{(1)} A &= U \\ A &= (M^{(n-1)} M^{(n-2)} \dots M^{(1)})^{-1} U \\ L &= (M^{(n-1)} M^{(n-2)} \dots M^{(1)})^{-1} \end{aligned}$$

Example



$$A = \begin{pmatrix} 2 & 1 & 0 \\ 4 & 5 & 2 \\ 6 & 15 & 12 \end{pmatrix} \quad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}$$

$$L_1 A = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 0 \\ 4 & 5 & 2 \\ 6 & 15 & 12 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 3 & 2 \\ 0 & 12 & 12 \end{pmatrix}$$

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{pmatrix}$$

$$L_2 (L_1 A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 0 \\ 0 & 3 & 2 \\ 0 & 12 & 12 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 3 & 2 \\ 0 & 0 & 4 \end{pmatrix} = U$$

...

$$L_2 L_1 A = U, \quad A = L_1^{-1} L_2^{-1} U = L U \quad L = L_1^{-1} L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix}.$$

One problem of GEM is that it is *unstable*, i.e. the algorithmic error is not limited. This happens because the elements $a_{kk}^{(k)}$ used to compute the multipliers can be very small or even zero, leading to errors. To avoid this, the *pivoting technique* is employed in order to make the elements $a_{kk}^{(k)}$ as big as possible. This is done by swapping two rows so that $a_{kk}^{(k)}$ is the column element with the maximum absolute value. The swapping can be seen as a left multiplication by a *permutation matrix* P , i.e. an identity matrix with the needed rows swapped.

Example

$$\begin{pmatrix} \times & \times & \times & \dots & \times \\ & \mathbf{x} & \mathbf{x} & \dots & \mathbf{x} \\ & \times & \times & \dots & \times \\ a_{j,k} & \mathbf{x} & \dots & \mathbf{x} \\ & \times & \times & \dots & \times \end{pmatrix} \xrightarrow{P_k} \begin{pmatrix} \times & \times & \times & \dots & \times \\ & a_{j,k} & \mathbf{x} & \dots & \mathbf{x} \\ & \times & \times & \dots & \times \\ & \mathbf{x} & \mathbf{x} & \dots & \mathbf{x} \\ & \times & \times & \dots & \times \end{pmatrix}$$

$$\xrightarrow{L_k} \begin{pmatrix} \times & \times & \times & \dots & \times \\ & a_{j,k} & \mathbf{x} & \dots & \mathbf{x} \\ & 0 & \mathbf{x} & \dots & \mathbf{x} \\ & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

$$\begin{pmatrix} & & 1 & \\ & 1 & & \\ 1 & & & \\ & & & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

$P_1 \qquad \qquad \mathbf{A} \qquad \qquad = \qquad \qquad A_1$

$$\begin{pmatrix} 1 & & & \\ -\frac{1}{2} & 1 & & \\ -\frac{1}{2} & & 1 & \\ -\frac{3}{4} & & & 1 \end{pmatrix} \cdot \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix}$$

$L_1 \qquad \qquad A_1 \qquad \qquad = \qquad \qquad \mathbf{A}_2$

$$\begin{aligned}
\mathbf{A} &= \begin{pmatrix} 8 & 7 & 9 & 5 \\ -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix} \\
\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}_{P_2} \cdot \begin{pmatrix} 8 & 7 & 9 & 5 \\ -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix}_{\mathbf{A}_2} &= \begin{pmatrix} 8 & 7 & 9 & 5 \\ \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{pmatrix}_{A_2} \\
\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}_{L_2} \cdot \begin{pmatrix} 8 & 7 & 9 & 5 \\ \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{pmatrix}_{A_2} &= \begin{pmatrix} 8 & 7 & 9 & 5 \\ \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & -\frac{2}{7} & \frac{4}{7} \\ & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix}_{\mathbf{A}_3}
\end{aligned}$$

$$\begin{aligned}
\mathbf{A} &= \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{2}{7} & \frac{4}{7} \\ & & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix} \\
\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}_{P_3} \cdot \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{2}{7} & \frac{4}{7} \\ & & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix}_{\mathbf{A}_3} &= \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & -\frac{2}{7} & \frac{4}{7} \end{pmatrix}_{A_3} \\
\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}_{L_3} \cdot \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & -\frac{2}{7} & \frac{4}{7} \end{pmatrix}_{A_3} &= \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & & \frac{2}{3} \end{pmatrix}_{\mathbf{R}}
\end{aligned}$$

$$L_3 \cdot P_3 \cdot L_2 \cdot P_2 \cdot L_1 \cdot P_1 \cdot \mathbf{A} = \mathbf{R}$$

$$L_3 \cdot P_3 \cdot L_2 \cdot P_2 \cdot L_1 \cdot P_1 = L'_3 \cdot L'_2 \cdot L'_1 \cdot P_3 \cdot P_2 \cdot P_1$$

infatti:

$$\underbrace{L_3}_{L'_3} \underbrace{(P_3 L_2 P_3^{-1})}_{L'_2} \underbrace{(P_3 P_2 L_1 P_2^{-1} P_3^{-1})}_{L'_1} P_3 P_2 P_1 =$$

$$\begin{pmatrix} & 1 & & \\ & & 1 & \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{pmatrix} \mathbf{A} = \begin{pmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{2} & -\frac{2}{7} & 1 & \\ \frac{1}{4} & -\frac{3}{7} & -\frac{1}{3} & 1 \end{pmatrix} \cdot \mathbf{R}$$

Proposition 3.1. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be symmetric positive semi-definite. Then it is always possible to LU-factorize it without using pivoting. In this case we have

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

and the factorization has a computational complexity of $O(n^3/6)$ (using Cholesky algorithm).

Example

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \mathbf{L}\mathbf{L}^\top = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}.$$

Multiplying out the right-hand side yields

$$\mathbf{A} = \begin{bmatrix} l_{11}^2 & l_{21}l_{11} & l_{31}l_{11} \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & l_{31}l_{21} + l_{32}l_{22} \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{bmatrix}.$$

The elements l_{ij} of the matrix \mathbf{L} are computed by comparing the left and right hand sides of the previous matrix equation.

$$l_{11} = \sqrt{a_{11}}, \quad l_{22} = \sqrt{a_{22} - l_{21}^2}, \quad l_{33} = \sqrt{a_{33} - (l_{31}^2 + l_{32}^2)}.$$

$$l_{21} = \frac{1}{l_{11}}a_{21}, \quad l_{31} = \frac{1}{l_{11}}a_{31}, \quad l_{32} = \frac{1}{l_{22}}(a_{32} - l_{31}l_{21}).$$

The Cholesky decomposition is a very useful tool in numerical linear algebra for Machine Learning, where we often encounter symmetric positive definite matrices. For example, the covariance matrix of a multivariate Gaussian variable is symmetric positive definite.

Moreover, the Cholesky decomposition can be used to efficiently compute the determinant of a symmetric positive definite matrix. We know that

$$\det(A) = \det(L)\det(L^T) = \det(L^2) = \prod_i l_{ii}^2.$$

3.2 Singular Value Decomposition (SVD)

We introduce here a decomposition that involve a diagonal matrix instead of Triangular matrices.

There are mainly two matrix decompositions involving a diagonal matrix: the so called *eigendecomposition* and the *Singular Value Decomposition*. We will go quickly through the first one and we will analyse in detail the second.

Proposition 3.2. *Eigendecomposition.* Let A a square matrix of size $n \times n$. Hence A can be decomposed as:

$$A = PDP^{-1}$$

where P is a non singular matrix of size $n \times n$ and D is a diagonal matrix with diagonal elements d_{ii} corresponding to the eigenvalues of A iff the eigenvectors of A are linear independent and form a basis of \mathbf{R}^n .

Example 4.11 MML book.

Observation. The eigenvalue decomposition can be applied only to square matrices and with particular properties on their spectrum.

We now introduce the Singular Value Decomposition (SVD) which can be applied to **all** the matrices. We first need some definitions and concepts on *orthogonality*.

Definition 3.2. Let $\mathbf{u}, \mathbf{v} \in \mathbf{R}^n$. \mathbf{u} and \mathbf{v} are *orthogonal* if

$$\langle \mathbf{u}, \mathbf{v} \rangle = 0$$

Definition 3.3. $\mathbf{u} \in \mathbb{R}^n$ is a *unit vector* if $\|\mathbf{u}\| = 1$.

Proposition 3.3. (Normalization) $\forall \mathbf{u} \in \mathbb{R}^n$, $\hat{\mathbf{u}} = \frac{\mathbf{u}}{\|\mathbf{u}\|}$ is a unit vector.

Example 3.1.

$$\begin{aligned}\mathbf{u} &= [1, 2, 3] \\ \|\mathbf{u}\|_2 &= \sqrt{14} \\ \hat{\mathbf{u}} &= \frac{\mathbf{u}}{\|\mathbf{u}\|} = \left[\frac{1}{\sqrt{14}}, \frac{2}{\sqrt{14}}, \frac{3}{\sqrt{14}} \right]\end{aligned}$$

Definition 3.4. The set $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p\}$, $\mathbf{u}_i \in \mathbb{R}^n$, $i = 1, \dots, p$ is an *orthogonal set* if $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \mathbf{u}_i^T \mathbf{u}_j = 0$, $\forall i \neq j$

Example 3.2.

$$\mathbf{u}_1 = [3, 1, 1]^T, \mathbf{u}_2 = [-1, 2, 1]^T, \mathbf{u}_3 = \left[-\frac{1}{2}, -2, \frac{7}{2} \right]^T$$

is an orthogonal set because

$$\langle \mathbf{u}_1, \mathbf{u}_2 \rangle = 0, \langle \mathbf{u}_1, \mathbf{u}_3 \rangle = 0, \langle \mathbf{u}_2, \mathbf{u}_3 \rangle = 0,$$

Proposition 3.4. If $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p \in \mathbb{R}^n$ are orthogonal then they are linearly independent. The set $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p\}$, $\mathbf{u}_i \in \mathbb{R}^n$, $i = 1, \dots, p$ is an *orthogonal basis* for $U = \text{span}\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p\}$.

Definition 3.5. The set $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p\}$, $\mathbf{u}_i \in \mathbb{R}^n$, $i = 1, \dots, p$ is an *orthonormal set* if it is an orthogonal set of unit vectors. A basis of orthonormal vectors is called *orthonormal basis*.

Definition 3.6. Let $U \in \mathbb{R}^{m \times n}$. U is an *orthogonal matrix* iff $U^T U = I$. If $m = n$ then $U^T = U^{-1}$.

Proposition 3.5. If $U^{m \times n}$ is orthogonal then

- $\|U\mathbf{x}\|_2 = \|\mathbf{x}\|_2$, $\forall \mathbf{x} \in \mathbb{R}^n$
- $\langle U\mathbf{x}, U\mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle$, $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
- $\langle U\mathbf{x}, U\mathbf{y} \rangle = 0 \iff \langle \mathbf{x}, \mathbf{y} \rangle = 0$, $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

Transformations by orthogonal matrices preserve both length and angles.

We present the SVD in the case $m \geq n$ but it can be easily extended to the case $m < n$ (see for example MML book).

Proposition 3.6. *singular value decomposition (SVD).* Any matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$ with $\text{rank}(A) = k$, $k \leq n$ can be written as

$$A = U \Sigma V^T$$

where

- $U \in \mathbb{R}^{m \times m}$ is an orthogonal matrix with orthogonal vectors \mathbf{u}_i
- $V \in \mathbb{R}^{n \times n}$ is an orthogonal matrix with orthogonal vectors \mathbf{v}_i
- $\Sigma \in \mathbb{R}^{m \times n}$ is a matrix whose diagonal entries are the *singular values* σ_i of A and with extra-diagonal entries equal to 0.

$$\overset{n}{\boxed{A}} = \overset{m}{\boxed{U}} \overset{n}{\boxed{\Sigma}} \overset{n}{\boxed{V^T}}$$

The singular values enjoy the following property

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_k > \sigma_{k+1} = \sigma_{k+2} = \dots = \sigma_n = 0$$

where $k = \text{rank}(A)$. The singular matrix Σ is unique, while the orthogonal matrices U and V aren't.

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n \\ 0 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & 0 \end{bmatrix}.$$

From the geometric point of view, the SVD performs two changes of basis, via the orthogonal matrices U and V and a scaling operation via the matrix Σ . The columns of U and V are sets of orthonormal basis of \mathbf{R}^m and \mathbf{R}^n , respectively. See Example 4.12 MML book.

Results

- Since the following relation holds:

$$A\mathbf{v}_i = \lambda_i\mathbf{v}_i$$

\mathbf{v}_i are called *right singular vectors* and \mathbf{u}_i are called *left singular vectors*.

- We have the following relation between the singular values of A and the eigenvalues of $A^T A$.

$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T$$

$$A^T A = V\Sigma^T \Sigma V^T = V \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n^2 \end{bmatrix} V^T.$$

Hence

and

$$\sigma_i = \sqrt{\lambda_i(A^T A)}, \quad i = \dots, n$$

In particular:

$$\sigma_1 = \sqrt{\lambda_{\max}(A^T A)} = \sqrt{\rho(A^T A)} = \|A\|_2$$

$$\sigma_n = \sqrt{\lambda_{\min}(A^T A)} \implies \|A^{-1}\| = \frac{1}{\sigma_n}$$

Thus,

$$K(A) = \frac{\sigma_1}{\sigma_n}$$

- The left singular vectors of A are the eigenvectors of $A^T A$
- The right singular vectors of A are the eigenvectors of AA^T
- For symmetric matrices A the eigenvalue decomposition and the SVD are one and the same.

Moore-Penrose inverse. The SVD can be used to compute the *Moore-Penrose inverse* (or simply *pseudoinverse*) of a matrix $A \in \mathbb{R}^{m \times n}$

$$A^+ = V\Sigma^+U^T$$

where $\Sigma^+ \in \mathbb{R}^{n \times m}$ is the pseudoinverse of Σ , computed by taking the reciprocal of every non-zero diagonal element, leaving the zeros in place and transposing the matrix

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_n \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \Sigma^+ = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 & \dots & 0 \\ 0 & 0 & \ddots & 0 & \dots & 0 \\ 0 & 0 & 0 & \frac{1}{\sigma_n} & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

3.3 Matrix approximation using SVD

Given the SVD of a matrix $A \in \mathbb{R}^{m \times n}$

$$A = U\Sigma V^T$$

we can use it to represent (or approximate) the matrix A as a sum of low-rank matrices $A_i \in \mathbb{R}^{m \times n}$ with $\text{rank}(A_i) = 1$ such that

$$A_i = \mathbf{u}_i \mathbf{v}_i^T$$

The matrix A of $\text{rank}(A) = k$ can then be written as

$$A = \sum_{i=1}^k \sigma_i A_i = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

To obtain a rank- p approximation A_p ($p < k$) of A we can truncate the sum at index $i = p$. The error introduced with this approximation can be computed as

$$\|A - A_p\|_2 = \left\| \sum_{i=p+1}^k \sigma_i A_i \right\|_2 = \sigma_{p+1}$$

This means that if σ_{p+1} is small we have a good approximation of the original matrix A .

Theorem 3.1. Given $A \in \mathbb{R}^{m \times n}$, $\text{rank}(A) = k$, let $A = U\Sigma V^T$ be its singular value decomposition. Let $A_p = \sum_{i=1}^p \sigma_i \mathbf{u}_i \mathbf{v}_i^T$ be the rank- p approximation of A . Then

$$\forall B \in \mathbb{R}^{m \times n}, \text{rank}(B) = p, \|A - A_p\|_2 \leq \|A - B\|_2$$

So A_p is the best rank- p approximation of A obtained via singular value decomposition.

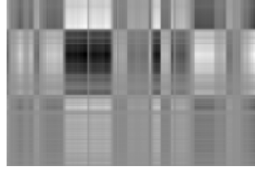
Example. Rank- p approximation of an image matrix.



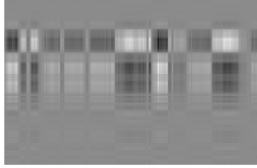
(a) Original image A .



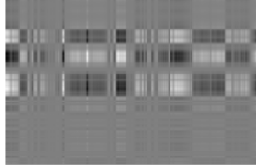
(b) A_1 , $\sigma_1 \approx 228,052$.



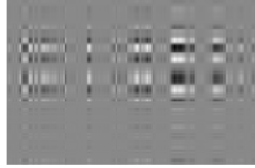
(c) A_2 , $\sigma_2 \approx 40,647$.



(d) A_3 , $\sigma_3 \approx 26,125$.



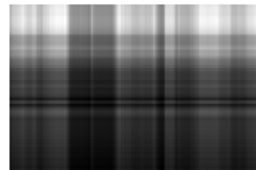
(e) A_4 , $\sigma_4 \approx 20,232$.



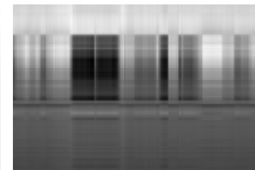
(f) A_5 , $\sigma_5 \approx 15,436$.



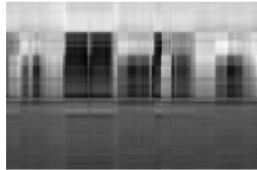
(a) Original image A .



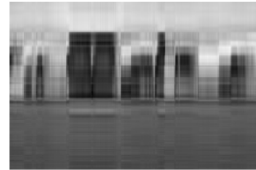
(b) Rank-1 approximation $\hat{A}(1)$.



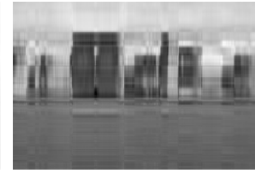
(c) Rank-2 approximation $\hat{A}(2)$.



(d) Rank-3 approximation $\hat{A}(3)$.



(e) Rank-4 approximation $\hat{A}(4)$.



(f) Rank-5 approximation $\hat{A}(5)$.

The matrix associated to the original image has size $m \times n = 1432 \times 1910$, hence it requires 2735120 data. If we consider the rank- p approximation, it requires $(m + n + 1) \times p$ data. For example, the rank-5 approximation

requires $5 \times (1432 + 1910 + 1) = 16715$ data, corresponding to about 0.6% of the original one. Hence we can interpret the rank-p SVD approximation of a matrix A as a lossy compression. This approximation appears in different machine learning applications, such as image processing, noise filtering and regularization of ill-posed problems. Moreover, it is a fundamental tool in the PCA analysis.

We can see the rank-p approximation obtain through SVD as the projection of A onto a subspace of matrices of rank p . Among all the possible projections, rank-p SVD approximation minimizes the error in the 2-norm between A and any rank-p approximation of A .

Examples 4.14 and 4.15 MML book *finding structure in Movie Ratings and Customers*

3.4 Conclusions

We can summarize many tools and ideas in the following figure, where we can see the relationship between different types of matrices.

