

# **Statistical and Mathematical Methods for Artificial Intelligence**

Academic Year 2023 – 2024  
Alma Mater Studiorum · University of Bologna

# Contents

<b>1</b>	<b>Finite numbers</b>	<b>1</b>
1.1	Sources of error . . . . .	1
1.2	Error measurement . . . . .	1
1.3	Representation in base $\beta$ . . . . .	1
1.4	Floating-point . . . . .	2
1.4.1	Numbers distribution . . . . .	2
1.4.2	Number representation . . . . .	2
1.4.3	Machine precision . . . . .	3
1.4.4	IEEE standard . . . . .	3
1.4.5	Floating-point arithmetic . . . . .	3
<b>2</b>	<b>Linear algebra</b>	<b>5</b>
2.1	Vector space . . . . .	5
2.1.1	Basis . . . . .	5
2.1.2	Dot product . . . . .	5
2.2	Matrix . . . . .	6
2.2.1	Invertible matrix . . . . .	6
2.2.2	Kernel . . . . .	6
2.2.3	Similar matrices . . . . .	6
2.3	Norms . . . . .	6
2.3.1	Vector norms . . . . .	6
2.3.2	Matrix norms . . . . .	7
2.4	Symmetric, positive definite matrices . . . . .	7
2.5	Orthogonality . . . . .	8
2.6	Projections . . . . .	9
2.6.1	Projection onto general subspaces . . . . .	9
2.7	Eigenvectors and eigenvalues . . . . .	9
2.7.1	Diagonalizability . . . . .	10
<b>3</b>	<b>Linear systems</b>	<b>11</b>
3.1	Square linear systems . . . . .	11
3.2	Direct methods . . . . .	11
3.2.1	Gaussian factorization . . . . .	12
3.2.2	Gaussian factorization with pivoting . . . . .	12
3.2.3	Cholesky factorization . . . . .	12
3.3	Iterative methods . . . . .	13
3.3.1	Stopping criteria . . . . .	13
3.4	Condition number . . . . .	13
3.5	Linear least squares problem . . . . .	14
<b>4</b>	<b>Matrix decomposition</b>	<b>15</b>
4.1	Eigendecomposition . . . . .	15

4.2	Singular value decomposition . . . . .	15
4.2.1	Singular values and eigenvalues . . . . .	15
4.2.2	Singular values and 2-norm . . . . .	16
4.2.3	Application: Matrix approximation . . . . .	16
4.2.4	Application: Linear least squares problem . . . . .	17
4.2.5	Application: Polynomial interpolation . . . . .	18
4.3	Eigendecomposition vs SVD . . . . .	18
<b>5</b>	<b>Vector calculus</b>	<b>19</b>
5.1	Gradient of real-valued multivariate functions . . . . .	19
5.1.1	Partial differentiation rules . . . . .	19
5.2	Gradient of vector-valued multivariate functions . . . . .	21
5.3	Backpropagation . . . . .	21
5.4	Automatic differentiation . . . . .	22
<b>6</b>	<b>Gradient methods</b>	<b>25</b>
6.1	Minimum of a function . . . . .	25
6.1.1	Optimality conditions . . . . .	25
6.2	Descent methods . . . . .	25
6.2.1	Choice of the search direction . . . . .	26
6.2.2	Choice of the step length . . . . .	26
6.2.3	Stopping condition . . . . .	27
6.2.4	Problems . . . . .	27
6.3	Convex functions . . . . .	28
6.3.1	Properties . . . . .	29
6.3.2	Quadratic functions . . . . .	29
6.4	Gradient descent with momentum . . . . .	29
6.5	Stochastic gradient descent (SGD) . . . . .	30

# 1 Finite numbers

## 1.1 Sources of error

**Measure error** Precision of the measurement instrument.

Measure error

**Arithmetic error** Propagation of rounding errors in each step of an algorithm.

Arithmetic error

**Truncation error** Approximating an infinite procedure to a finite number of iterations.

Truncation error

**Inherent error** Caused by the finite representation of the data (floating-point).

Inherent error

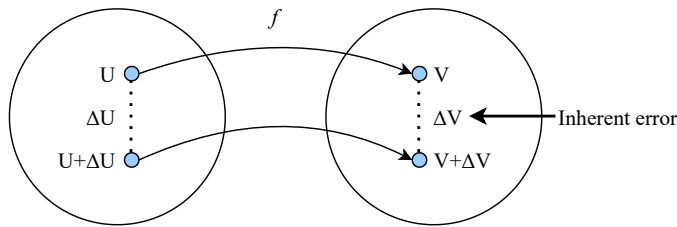


Figure 1.1: Inherent error visualization

## 1.2 Error measurement

Let  $x$  be a value and  $\hat{x}$  its approximation. Then:

**Absolute error**

$$E_a = \hat{x} - x$$

Absolute error

Note that, out of context, the absolute error is meaningless.

**Relative error**

$$E_r = \frac{\hat{x} - x}{x}$$

Relative error

## 1.3 Representation in base $\beta$

Let  $\beta \in \mathbb{N}_{>1}$  be the base. Each  $x \in \mathbb{R} \setminus \{0\}$  can be uniquely represented as:

$$x = \text{sign}(x) \cdot (d_1\beta^{-1} + d_2\beta^{-2} + \dots + d_n\beta^{-n})\beta^p \quad (1.1)$$

where:

- $0 \leq d_i \leq \beta - 1$
- $d_1 \neq 0$
- starting from an index  $i$ , not all  $d_j$  ( $j \geq i$ ) are equal to  $\beta - 1$

Equation (1.1) can be represented using the normalized scientific notation as:

Normalized scientific notation

$$x = \pm(0.d_1d_2\dots)\beta^p$$

where  $0.d_1d_2\dots$  is the **mantissa** and  $\beta^p$  the **exponent**.

Mantissa  
Exponent

## 1.4 Floating-point

A floating-point system  $\mathcal{F}(\beta, t, L, U)$  is defined by the parameters:

Floating-point

- $\beta$ : base
- $t$ : precision (number of digits in the mantissa)
- $[L, U]$ : range of the exponent

Each  $x \in \mathcal{F}(\beta, t, L, U)$  can be represented in its normalized form:

$$x = \pm(0.d_1d_2\dots d_t)\beta^p \quad L \leq p \leq U \quad (1.2)$$

We denote with  $\text{fl}(x)$  the representation of  $x \in \mathbb{R}$  in a given floating-point system.

**Example.** In  $\mathcal{F}(10, 5, -3, 3)$ ,  $x = 12.\bar{3}$  is represented as:

$$\text{fl}(x) = +0.12333 \cdot 10^2$$

### 1.4.1 Numbers distribution

Given a floating-point system  $\mathcal{F}(\beta, t, L, U)$ , the total amount of representable numbers is:

$$2(\beta - 1)\beta^{t-1}(U - L + 1) + 1$$

Representable numbers are more sparse towards the exponent upper bound and more dense towards the lower bound. It must be noted that there is an underflow area around 0.

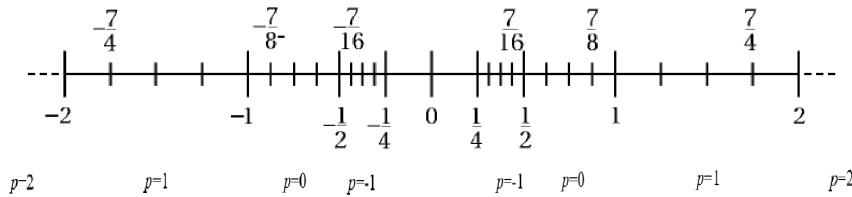


Figure 1.2: Floating-point numbers in  $\mathcal{F}(2, 3, -1, 2)$

### 1.4.2 Number representation

Given a floating-point system  $\mathcal{F}(\beta, t, L, U)$ , the representation of  $x \in \mathbb{R}$  can result in:

**Exact representation** if  $p \in [L, U]$  and  $d_i = 0$  for  $i > t$ .

**Approximation** if  $p \in [L, U]$  but  $d_i$  may not be 0 for  $i > t$ . In this case, the representation is obtained by truncating or rounding the value.

Truncation  
Rounding

**Underflow** if  $p < L$ . In this case, the value is approximated to 0.

Underflow

**Overflow** if  $p > U$ . In this case, an exception is usually raised.

Overflow

### 1.4.3 Machine precision

Machine precision  $\varepsilon_{\text{mach}}$  determines the accuracy of a floating-point system. Depending on the approximation approach, machine precision can be computed as:

**Truncation**  $\varepsilon_{\text{mach}} = \beta^{1-t}$

**Rounding**  $\varepsilon_{\text{mach}} = \frac{1}{2}\beta^{1-t}$

Therefore, rounding results in more accurate representations.

$\varepsilon_{\text{mach}}$  is the smallest distance among the representable numbers (Figure 1.3).

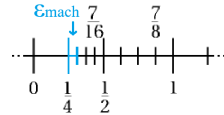


Figure 1.3: Visualization of  $\varepsilon_{\text{mach}}$  in  $\mathcal{F}(2, 3, -1, 2)$

In alternative,  $\varepsilon_{\text{mach}}$  can be defined as the smallest representable number such that:

$$\text{fl}(1 + \varepsilon_{\text{mach}}) > 1.$$

### 1.4.4 IEEE standard

IEEE 754 defines two floating-point formats:

**Single precision** Stored in 32 bits. Represents the system  $\mathcal{F}(2, 24, -128, 127)$ . float32

1 (sign)	8 (exponent)	23 (mantissa)
----------	--------------	---------------

**Double precision** Stored in 64 bits. Represents the system  $\mathcal{F}(2, 53, -1024, 1023)$ . float64

1 (sign)	11 (exponent)	52 (mantissa)
----------	---------------	---------------

As the first digit of the mantissa is always 1, it does not need to be stored. Moreover, special configurations are reserved to represent **Inf** and **NaN**.

### 1.4.5 Floating-point arithmetic

Let:

- $+$  :  $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  be a real numbers operation.
- $\oplus$  :  $\mathcal{F} \times \mathcal{F} \rightarrow \mathcal{F}$  be the corresponding operation in a floating-point system.

To compute  $x \oplus y$ , a machine:

1. Calculates  $x + y$  in a high precision register (still approximated, but more precise than the floating-point system used to store the result)
2. Stores the result as  $\text{fl}(x + y)$

A floating-point operation causes a small rounding error:

$$\left| \frac{(x \oplus y) - (x + y)}{x + y} \right| < \varepsilon_{\text{mach}}$$

However, some operations may be subject to the **cancellation** problem which causes information loss.

Cancellation

**Example.** Given  $x = 1$  and  $y = 1 \cdot 10^{-16}$ , we want to compute  $x + y$  in  $\mathcal{F}(10, 16, U, L)$ .

$$\begin{aligned}
 z &= \mathbf{fl}(x) + \mathbf{fl}(y) \\
 &= 0.1 \cdot 10^1 + 0.1 \cdot 10^{-15} \\
 &= (0.1 + 0.\overbrace{0 \dots 0}^{16 \text{ zeros}}1) \cdot 10^1 \\
 &= 0.1\overbrace{0 \dots 0}^{15 \text{ zeros}}1 \cdot 10^1
 \end{aligned}$$

Then, we have that  $\mathbf{fl}(z) = 0.1\overbrace{0 \dots 0}^{15 \text{ zeros}}1 \cdot 10^1 = 1 = x$ .

## 2 Linear algebra

### 2.1 Vector space

A **vector space** over  $\mathbb{R}$  is a nonempty set  $V$ , whose elements are called vectors, with two operations:

Vector space

$$\begin{array}{ll} \text{Addition} & + : V \times V \rightarrow V \\ \text{Scalar multiplication} & \cdot : \mathbb{R} \times V \rightarrow V \end{array}$$

A vector space has the following properties:

1. Addition is commutative and associative
2. A null vector exists:  $\exists \bar{\mathbf{0}} \in V$  s.t.  $\forall \mathbf{u} \in V : \bar{\mathbf{0}} + \mathbf{u} = \mathbf{u} + \bar{\mathbf{0}} = \mathbf{u}$
3. An identity element for scalar multiplication exists:  $\forall \mathbf{u} \in V : 1\mathbf{u} = \mathbf{u}$
4. Each vector has its opposite:  $\forall \mathbf{u} \in V, \exists \mathbf{a} \in V : \mathbf{a} + \mathbf{u} = \mathbf{u} + \mathbf{a} = \bar{\mathbf{0}}$ .  
 $\mathbf{a}$  is denoted as  $-\mathbf{u}$ .
5. Distributive properties:

$$\forall \alpha \in \mathbb{R}, \forall \mathbf{u}, \mathbf{w} \in V : \alpha(\mathbf{u} + \mathbf{w}) = \alpha\mathbf{u} + \alpha\mathbf{w}$$

$$\forall \alpha, \beta \in \mathbb{R}, \forall \mathbf{u} \in V : (\alpha + \beta)\mathbf{u} = \alpha\mathbf{u} + \beta\mathbf{u}$$

6. Associative property:

$$\forall \alpha, \beta \in \mathbb{R}, \forall \mathbf{u} \in V : (\alpha\beta)\mathbf{u} = \alpha(\beta\mathbf{u})$$

A subset  $U \subseteq V$  of a vector space  $V$  is a **subspace** iff  $U$  is a vector space.

Subspace

#### 2.1.1 Basis

Let  $V$  be a vector space of dimension  $n$ . A basis  $\beta = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$  of  $V$  is a set of  $n$  linearly independent vectors of  $V$ .

Basis

Each element of  $V$  can be represented as a linear combination of the vectors in the basis  $\beta$ :

$$\forall \mathbf{w} \in V : \mathbf{w} = \lambda_1 \mathbf{v}_1 + \dots + \lambda_n \mathbf{v}_n \text{ where } \lambda_i \in \mathbb{R}$$

The canonical basis of a vector space is a basis where each vector represents a dimension  $i$  (i.e. 1 in position  $i$  and 0 in all other positions).

Canonical basis

**Example.** The canonical basis  $\beta$  of  $\mathbb{R}^3$  is  $\beta = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$

#### 2.1.2 Dot product

The dot product of two vectors in  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$  is defined as:

Dot product

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_{i=1}^n x_i \cdot y_i$$



## 2.2 Matrix

This is a (very formal definition of) **matrix**:

Matrix

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

### 2.2.1 Invertible matrix

A matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is invertible (non-singular) if:

Non-singular matrix

$$\exists \mathbf{B} \in \mathbb{R}^{n \times n} : \mathbf{AB} = \mathbf{BA} = \mathbf{I}$$

where  $\mathbf{I}$  is the identity matrix.  $\mathbf{B}$  is denoted as  $\mathbf{A}^{-1}$ .

### 2.2.2 Kernel

The null space (kernel) of a matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  is a subspace such that:

Kernel

$$\text{Ker}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \bar{\mathbf{0}}\}$$

**Theorem 2.2.1.** A square matrix  $\mathbf{A}$  with  $\text{Ker}(\mathbf{A}) = \{\bar{\mathbf{0}}\}$  is non singular.

### 2.2.3 Similar matrices

Two matrices  $\mathbf{A}$  and  $\mathbf{D}$  are **similar** if there exists an invertible matrix  $\mathbf{P}$  such that:

Similar matrices

$$\mathbf{D} = \mathbf{P}^{-1}\mathbf{AP}$$

## 2.3 Norms

### 2.3.1 Vector norms

The norm of a vector is a function:

Vector norm

$$\|\cdot\| : \mathbb{R}^n \rightarrow \mathbb{R}$$

such that for each  $\lambda \in \mathbb{R}$  and  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ :

- $\|\mathbf{x}\| \geq 0$
- $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \bar{\mathbf{0}}$
- $\|\lambda\mathbf{x}\| = |\lambda| \cdot \|\mathbf{x}\|$
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$

Common norms are:

$$\textbf{2-norm} \quad \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$$

$$\textbf{1-norm} \quad \|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$$

**$\infty$ -norm**  $\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$

In general, different norms tend to maintain the same proportion. In some cases, unbalanced results may be obtained when comparing different norms.

**Example.** Let  $\mathbf{x} = (1, 1000)$  and  $\mathbf{y} = (999, 1000)$ . Their norms are:

$$\begin{aligned} \|\mathbf{x}\|_2 &= \sqrt{1000001} & \|\mathbf{y}\|_2 &= \sqrt{1998001} \\ \|\mathbf{x}\|_\infty &= 1000 & \|\mathbf{y}\|_\infty &= 1000 \end{aligned}$$

### 2.3.2 Matrix norms

The norm of a matrix is a function:

Matrix norm

$$\|\cdot\| : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$$

such that for each  $\lambda \in \mathbb{R}$  and  $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ :

- $\|\mathbf{A}\| \geq 0$
- $\|\mathbf{A}\| = 0 \iff \mathbf{A} = \mathbf{0}$
- $\|\lambda \mathbf{A}\| = |\lambda| \cdot \|\mathbf{A}\|$
- $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$

Common norms are:

**2-norm**  $\|\mathbf{A}\|_2 = \sqrt{\rho(\mathbf{A}^T \mathbf{A})}$ ,  
where  $\rho(\mathbf{X})$  is the largest absolute value of the eigenvalues of  $\mathbf{X}$  (spectral radius).

**1-norm**  $\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{i,j}|$  (i.e. max sum of the columns in absolute value)

**Frobenius norm**  $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{i,j}^2}$

## 2.4 Symmetric, positive definite matrices

**Symmetric matrix** A square matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is symmetric  $\iff \mathbf{A} = \mathbf{A}^T$

Symmetric matrix

**Positive semidefinite matrix** A symmetric matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is positive semidefinite iff

Positive semidefinite matrix

$$\forall \mathbf{x} \in \mathbb{R}^n \setminus \{0\} : \mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$$

**Positive definite matrix** A symmetric matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is positive definite iff

Positive definite matrix

$$\forall \mathbf{x} \in \mathbb{R}^n \setminus \{0\} : \mathbf{x}^T \mathbf{A} \mathbf{x} > 0$$

It has the following properties:

1. The null space of  $\mathbf{A}$  has the null vector only:  $\text{Ker}(\mathbf{A}) = \{\mathbf{0}\}$ .  
Which implies that  $\mathbf{A}$  is non-singular (Theorem 2.2.1).
2. The diagonal elements of  $\mathbf{A}$  are all positive.

## 2.5 Orthogonality

**Angle between vectors** The angle  $\omega$  between two vectors  $\mathbf{x}$  and  $\mathbf{y}$  can be obtained from: Angle between vectors

$$\cos \omega = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \cdot \|\mathbf{y}\|_2}$$

**Orthogonal vectors** Two vectors  $\mathbf{x}$  and  $\mathbf{y}$  are orthogonal ( $\mathbf{x} \perp \mathbf{y}$ ) when: Orthogonal vectors

$$\langle \mathbf{x}, \mathbf{y} \rangle = 0$$

**Orthonormal vectors** Two vectors  $\mathbf{x}$  and  $\mathbf{y}$  are orthonormal when: Orthonormal vectors

$$\mathbf{x} \perp \mathbf{y} \text{ and } \|\mathbf{x}\| = \|\mathbf{y}\| = 1$$

**Theorem 2.5.1.** The canonical basis of a vector space is orthonormal.

**Orthogonal matrix** A matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is orthogonal if its columns are orthonormal vectors. It has the following properties: Orthogonal matrix

1.  $\mathbf{A}\mathbf{A}^T = \mathbf{I} = \mathbf{A}^T\mathbf{A}$ , which implies  $\mathbf{A}^{-1} = \mathbf{A}^T$ .
2. The length of a vector is unchanged when mapped through an orthogonal matrix:

$$\|\mathbf{A}\mathbf{x}\|^2 = \|\mathbf{x}\|^2$$

3. The angle between two vectors is unchanged when both are mapped through an orthogonal matrix:

$$\cos \omega = \frac{(\mathbf{A}\mathbf{x})^T(\mathbf{A}\mathbf{y})}{\|\mathbf{A}\mathbf{x}\| \cdot \|\mathbf{A}\mathbf{y}\|} = \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$$

Note: an orthogonal matrix represents a rotation.

**Orthogonal basis** Given a  $n$ -dimensional vector space  $V$  and a basis  $\beta = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$  of  $V$ .  $\beta$  is an orthogonal basis if: Orthogonal basis

$$\mathbf{b}_i \perp \mathbf{b}_j \text{ for } i \neq j \text{ (i.e. } \langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0)$$

**Orthonormal basis** Given a  $n$ -dimensional vector space  $V$  and an orthogonal basis  $\beta = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$  of  $V$ .  $\beta$  is an orthonormal basis if: Orthonormal basis

$$\|\mathbf{b}_i\|_2 = 1 \text{ (or } \langle \mathbf{b}_i, \mathbf{b}_i \rangle = 1)$$

**Orthogonal complement** Given a  $n$ -dimensional vector space  $V$  and a  $m$ -dimensional subspace  $U \subseteq V$ . The orthogonal complement  $U^\perp$  of  $U$  is a  $(n - m)$ -dimensional subspace of  $V$  such that it contains all the vectors orthogonal to every vector in  $U$ : Orthogonal complement

$$\forall \mathbf{w} \in V : \mathbf{w} \in U^\perp \iff (\forall \mathbf{u} \in U : \mathbf{w} \perp \mathbf{u})$$

Note that  $U \cap U^\perp = \{\bar{\mathbf{0}}\}$  and it is possible to represent all vectors in  $V$  as a linear combination of both the basis of  $U$  and  $U^\perp$ .

The vector  $\mathbf{w} \in U^\perp$  s.t.  $\|\mathbf{w}\| = 1$  is the **normal vector** of  $U$ . Normal vector

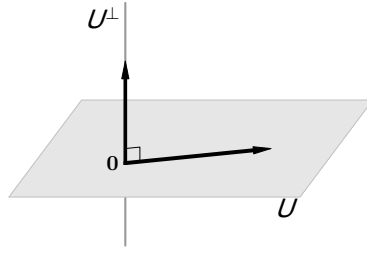


Figure 2.1: Orthogonal complement of a subspace  $U \subseteq \mathbb{R}^3$

## 2.6 Projections

Projections are methods to map high-dimensional data into a lower-dimensional space while minimizing the compression loss.

Let  $V$  be a vector space and  $U \subseteq V$  a subspace of  $V$ . A linear mapping  $\pi : V \rightarrow U$  is a (orthogonal) projection if:

Orthogonal  
projection

$$\pi^2 = \pi \circ \pi = \pi$$

In other words, applying  $\pi$  multiple times gives the same result (i.e. idempotency).

$\pi$  can be expressed as a transformation matrix  $P_\pi$  such that:

$$P_\pi^2 = P_\pi$$

### 2.6.1 Projection onto general subspaces

To project a vector  $\mathbf{x} \in \mathbb{R}^n$  into a lower-dimensional subspace  $U \subseteq \mathbb{R}^n$ , it is possible to use the basis of  $U$ .

Projection onto  
subspace basis

Let  $m = \dim(U)$  be the dimension of  $U$  and  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_m) \in \mathbb{R}^{n \times m}$  an ordered basis of  $U$ . A projection  $\pi_U(\mathbf{x})$  represents  $\mathbf{x}$  as a linear combination of the basis:

$$\pi_U(\mathbf{x}) = \sum_{i=1}^m \lambda_i \mathbf{b}_i = \mathbf{B}\lambda$$

where  $\lambda = (\lambda_1, \dots, \lambda_m)^T \in \mathbb{R}^m$  are the new coordinates of  $\mathbf{x}$  and is found by minimizing the distance between  $\pi_U(\mathbf{x})$  and  $\mathbf{x}$ .

## 2.7 Eigenvectors and eigenvalues

Given a square matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,  $\lambda \in \mathbb{C}$  is an eigenvalue of  $\mathbf{A}$  with corresponding eigenvector  $\mathbf{x} \in \mathbb{R}^n \setminus \{\bar{\mathbf{0}}\}$  if:

Eigenvalue  
Eigenvector

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

It is equivalent to say that:

- $\lambda$  is an eigenvalue of  $\mathbf{A} \in \mathbb{R}^{n \times n}$
- $\exists \mathbf{x} \in \mathbb{R}^n \setminus \{\bar{\mathbf{0}}\}$  s.t.  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$   
Equivalently the system  $(\mathbf{A} - \lambda\mathbf{I}_n)\mathbf{x} = \bar{\mathbf{0}}$  is non-trivial ( $\mathbf{x} \neq \bar{\mathbf{0}}$ ).
- $\text{rank}(\mathbf{A} - \lambda\mathbf{I}_n) < n$

- $\det(\mathbf{A} - \lambda \mathbf{I}_n) = 0$  (i.e.  $(\mathbf{A} - \lambda \mathbf{I}_n)$  is singular (i.e. not invertible))

Note that eigenvectors are not unique. Given an eigenvector  $\mathbf{x}$  of  $\mathbf{A}$  with eigenvalue  $\lambda$ , we can prove that  $\forall c \in \mathbb{R} \setminus \{0\} : c\mathbf{x}$  is an eigenvector of  $\mathbf{A}$ :

$$\mathbf{A}(c\mathbf{x}) = c(\mathbf{A}\mathbf{x}) = c\lambda\mathbf{x} = \lambda(c\mathbf{x})$$

**Theorem 2.7.1.**  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is symmetric positive definite  $\iff$  its eigenvalues are all positive. Eigenvalues and positive definiteness

**Eigenspace** Set of all the eigenvectors of  $\mathbf{A} \in \mathbb{R}^{n \times n}$  associated to an eigenvalue  $\lambda$ . This set is a subspace of  $\mathbb{R}^n$ . Eigenspace

**Eigenspectrum** Set of all eigenvalues of  $\mathbf{A} \in \mathbb{R}^{n \times n}$ . Eigenspectrum

**Geometric multiplicity** Given an eigenvalue  $\lambda$  of a matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ . The geometric multiplicity of  $\lambda$  is the number of linearly independent eigenvectors associated to  $\lambda$ . Geometric multiplicity

**Theorem 2.7.2.** Given a matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ . If its  $n$  eigenvectors  $\mathbf{x}_1, \dots, \mathbf{x}_n$  are associated to distinct eigenvalues, then  $\mathbf{x}_1, \dots, \mathbf{x}_n$  are linearly independent (i.e. they form a basis of  $\mathbb{R}^n$ ). Linearly independent eigenvectors

**Defective matrix** A matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is defective if it has less than  $n$  linearly independent eigenvectors. Defective matrix

**Theorem 2.7.3** (Spectral theorem). Given a symmetric matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ . Its eigenvectors form an orthonormal basis and its eigenvalues are all in  $\mathbb{R}$ . Spectral theorem

### 2.7.1 Diagonalizability

A matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is diagonalizable if it is similar to a diagonal matrix  $\mathbf{D} \in \mathbb{R}^{n \times n}$ : Diagonalizable matrix

$$\exists \mathbf{P} \in \mathbb{R}^{n \times n} \text{ s.t. } \mathbf{P} \text{ invertible and } \mathbf{D} = \mathbf{P}^{-1} \mathbf{A} \mathbf{P}$$

**Theorem 2.7.4.** Similar matrices have the same eigenvalues.

**Theorem 2.7.5.** A symmetric matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is always diagonalizable. Symmetric matrix diagonalizability

## 3 Linear systems

A linear system:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = b_m \end{cases}$$

can be represented as:

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

where:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times n} \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m$$

### 3.1 Square linear systems

A square linear system  $\mathbf{A}\mathbf{x} = \mathbf{b}$  with  $\mathbf{A} \in \mathbb{R}^{n \times n}$  and  $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$  has an unique solution iff one of the following conditions is satisfied: Square linear system

1.  $\mathbf{A}$  is non-singular (invertible)
2.  $\text{rank}(\mathbf{A}) = n$  (full rank)
3.  $\mathbf{A}\mathbf{x}$  admits only the solution  $\mathbf{x} = \bar{\mathbf{0}}$

The solution can be algebraically determined as

Algebraic solution to linear systems

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

However, this approach requires to compute the inverse of a matrix, which has a time complexity of  $O(n^3)$ .

### 3.2 Direct methods

Direct methods compute the solution of a linear system in a finite number of steps. Compared to iterative methods, they are more precise but more expensive. Direct methods

The most common approach consists in factorizing the matrix  $\mathbf{A}$ .

### 3.2.1 Gaussian factorization

Given a square linear system  $\mathbf{Ax} = \mathbf{b}$ , the matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is factorized into  $\mathbf{A} = \mathbf{LU}$  such that:

Gaussian  
factorization  
(LU decomposition)

- $\mathbf{L} \in \mathbb{R}^{n \times n}$  is a lower triangular matrix
- $\mathbf{U} \in \mathbb{R}^{n \times n}$  is an upper triangular matrix

The system can be decomposed to:

$$\begin{aligned}\mathbf{Ax} = \mathbf{b} &\iff \mathbf{LUx} = \mathbf{b} \\ &\iff \mathbf{y} = \mathbf{Ux} \text{ \& } \mathbf{Ly} = \mathbf{b}\end{aligned}$$

To find the solution, it is sufficient to solve in order:

1.  $\mathbf{Ly} = \mathbf{b}$  (solved w.r.t.  $\mathbf{y}$ )
2.  $\mathbf{y} = \mathbf{Ux}$  (solved w.r.t.  $\mathbf{x}$ )

The overall complexity is  $O(\frac{n^3}{3}) + 2 \cdot O(n^2) = O(\frac{n^3}{3})$ .

$O(\frac{n^3}{3})$  is the time complexity of the LU factorization.  $O(n^2)$  is the complexity to directly solving a system with a triangular matrix (forward or backward substitutions).

### 3.2.2 Gaussian factorization with pivoting

During the computation of  $\mathbf{A} = \mathbf{LU}$  (using Gaussian elimination<sup>1</sup>), a division by 0 may occur. A method to prevent this problem (and to lower the algorithmic error) is to change the order of the rows of  $\mathbf{A}$  before decomposing it. This is achieved by using a permutation matrix  $\mathbf{P}$ , which is obtained as a permutation of the identity matrix.

Gaussian  
factorization with  
pivoting

The permuted system becomes  $\mathbf{PAx} = \mathbf{Pb}$  and the factorization is obtained as  $\mathbf{PA} = \mathbf{LU}$ . The system can be decomposed to:

$$\begin{aligned}\mathbf{PAx} = \mathbf{Pb} &\iff \mathbf{LUx} = \mathbf{Pb} \\ &\iff \mathbf{y} = \mathbf{Ux} \text{ \& } \mathbf{Ly} = \mathbf{Pb}\end{aligned}$$

An alternative formulation (which is what SciPy uses) is defined as:

$$\mathbf{A} = \mathbf{PLU} \iff \mathbf{P}^T \mathbf{A} = \mathbf{LU}$$

It must be noted that  $\mathbf{P}$  is orthogonal, so  $\mathbf{P}^T = \mathbf{P}^{-1}$ . The solution to the system ( $\mathbf{P}^T \mathbf{Ax} = \mathbf{P}^T \mathbf{b}$ ) can be found as above.

### 3.2.3 Cholesky factorization

Given a symmetric definite positive matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ . It is possible to decompose  $\mathbf{A}$  as:

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

where  $\mathbf{L}$  is lower triangular.

A square system where  $\mathbf{A}$  is symmetric definite positive can be solved as above using the Cholesky factorization. This method has time complexity  $O(\frac{n^3}{6})$ .

<sup>1</sup>[https://en.wikipedia.org/wiki/LU\\_decomposition#Using\\_Gaussian\\_elimination](https://en.wikipedia.org/wiki/LU_decomposition#Using_Gaussian_elimination)

### 3.3 Iterative methods

Iterative methods solve a linear system by computing a sequence that converges to the exact solution. Compared to direct methods, they are less precise but computationally faster and more adapt for large systems.

Iterative methods

The overall idea is to build a sequence of vectors  $\mathbf{x}_k$  that converges to the exact solution  $\mathbf{x}^*$ :

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

Generally, the first vector  $\mathbf{x}_0$  is given (or guessed). Subsequent vectors are computed w.r.t. the previous iteration as  $\mathbf{x}_k = g(\mathbf{x}_{k-1})$ .

The two most common families of iterative methods are:

**Stationary methods** compute the sequence as:

Stationary methods

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

where  $B$  is called iteration matrix and  $\mathbf{d}$  is computed from the  $\mathbf{b}$  vector of the system. The time complexity per iteration is  $O(n^2)$ .

**Gradient-like methods** have the form:

Gradient-like methods

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

where  $\alpha_{k-1} \in \mathbb{R}$  and the vector  $\mathbf{p}_{k-1}$  is called direction.

#### 3.3.1 Stopping criteria

One or more stopping criteria are needed to determine when to truncate the sequence (as it is theoretically infinite). The most common approaches are:

Stopping criteria

**Residual based** The algorithm is terminated when the current solution is close enough to the exact solution. The residual at iteration  $k$  is computed as  $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ . Given a tolerance  $\varepsilon$ , the algorithm may stop when:

- $\|\mathbf{r}_k\| \leq \varepsilon$  (absolute)
- $\frac{\|\mathbf{r}_k\|}{\|\mathbf{b}\|} \leq \varepsilon$  (relative)

**Update based** The algorithm is terminated when the difference between iterations is very small. Given a tolerance  $\tau$ , the algorithm stops when:

$$\|\mathbf{x}_k - \mathbf{x}_{k-1}\| \leq \tau$$

Obviously, as the sequence is truncated, a truncation error is introduced when using iterative methods.

### 3.4 Condition number

Inherent error causes inaccuracies during the resolution of a system. This problem is independent from the algorithm and is estimated using exact arithmetic.

Given a system  $A\mathbf{x} = \mathbf{b}$ , we perturbate  $A$  and/or  $\mathbf{b}$  and study the inherited error. For instance, if we perturbate  $\mathbf{b}$ , we obtain the following system:

$$A\tilde{\mathbf{x}} = (\mathbf{b} + \Delta\mathbf{b})$$



After finding  $\tilde{\mathbf{x}}$ , we can compute the inherited error as  $\Delta \mathbf{x} = \tilde{\mathbf{x}} - \mathbf{x}$ .  
 By comparing  $\left\| \frac{\Delta \mathbf{x}}{\mathbf{x}} \right\|$  and  $\left\| \frac{\Delta \mathbf{b}}{\mathbf{b}} \right\|$ , we can compute the error introduced by the perturbation.  
 It can be shown that the distance is:

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

Finally, we can define the **condition number** of a matrix  $\mathbf{A}$  as:

Condition number

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

A system is **ill-conditioned** if  $K(\mathbf{A})$  is large (i.e. a small perturbation of the input causes a large change of the output). Otherwise it is **well-conditioned**.

Ill-conditioned

Well-conditioned

### 3.5 Linear least squares problem

See Section 4.2.4.

## 4 Matrix decomposition

### 4.1 Eigendecomposition

Given a matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ . If the eigenvectors of  $\mathbf{A}$  form a basis of  $\mathbb{R}^n$ , then  $\mathbf{A} \in \mathbb{R}^{n \times n}$  can be decomposed into: Eigendecomposition

$$\mathbf{A} = \mathbf{P} \mathbf{D} \mathbf{P}^{-1}$$

where  $\mathbf{P} \in \mathbb{R}^{n \times n}$  contains the eigenvectors of  $\mathbf{A}$  as its columns and  $\mathbf{D}$  is a diagonal matrix whose diagonal contains the eigenvalues of  $\mathbf{A}$ .

### 4.2 Singular value decomposition

Given a matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  of rank  $r \in [0, \min\{m, n\}]$ . The singular value decomposition (SVD) of  $\mathbf{A}$  is always possible and has form: Singular value decomposition

$$\begin{aligned} \mathbf{A} &= \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \\ &= \left( \begin{pmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_m \end{pmatrix} \right) \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{\min\{m,n\}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_n \end{pmatrix} \end{aligned}$$

where:

- $\mathbf{U} \in \mathbb{R}^{m \times m}$  is an orthogonal matrix with columns  $\mathbf{u}_i$  called left-singular vectors.
- $\mathbf{V} \in \mathbb{R}^{n \times n}$  is an orthogonal matrix with columns  $\mathbf{v}_i$  called right-singular vectors.
- $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$  is a matrix with  $\Sigma_{i,j} = 0$  (i.e. diagonal if it was a square matrix) and the singular values  $\sigma_i, i = 1 \dots \min\{m, n\}$  on the diagonal. By convention  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r \geq 0$ . Note that singular values  $\sigma_j = 0$  for  $(r+1) \leq j \leq \min\{m, n\}$  (i.e. singular values at indexes after  $\text{rank}(\mathbf{A})$  are always 0).

We can also represent SVD as a **singular value equation**, which resembles the eigenvalue equation: Singular value equation

$$\mathbf{A} \mathbf{v}_i = \sigma_i \mathbf{u}_i \text{ for } i = 1, \dots, r$$

This is derived from:

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \iff \mathbf{A} \mathbf{V} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \mathbf{V} \iff \mathbf{A} \mathbf{V} = \mathbf{U} \mathbf{\Sigma}$$

#### 4.2.1 Singular values and eigenvalues

Given  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , we can obtain the eigenvalues and eigenvectors of  $\mathbf{A}^T \mathbf{A}$  and  $\mathbf{A} \mathbf{A}^T$  through SVD. Eigendecomposition of  $\mathbf{A}^T \mathbf{A}$  and  $\mathbf{A} \mathbf{A}^T$

For  $\mathbf{A}^T \mathbf{A}$ , we can compute:

$$\begin{aligned}\mathbf{A}^T \mathbf{A} &= (\mathbf{U} \mathbf{\Sigma} \mathbf{V}^T)^T (\mathbf{U} \mathbf{\Sigma} \mathbf{V}^T) \text{ using } (\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T \\ &= (\mathbf{V} \mathbf{\Sigma}^T \mathbf{U}^T) (\mathbf{U} \mathbf{\Sigma} \mathbf{V}^T) \\ &= \mathbf{V} \mathbf{\Sigma}^T \mathbf{\Sigma} \mathbf{V}^T \\ &= \mathbf{V} \mathbf{\Sigma}^2 \mathbf{V}^T\end{aligned}$$

As  $\mathbf{V}$  is orthogonal ( $\mathbf{V}^T = \mathbf{V}^{-1}$ ), we can apply the eigendecomposition theorem:

- The diagonal of  $\mathbf{\Sigma}^2$  (i.e. the square of the singular values of  $\mathbf{A}$ ) are the eigenvalues of  $\mathbf{A}^T \mathbf{A}$
- The columns of  $\mathbf{V}$  (right-singular vectors) are the eigenvectors of  $\mathbf{A}^T \mathbf{A}$

The same process holds for  $\mathbf{A} \mathbf{A}^T$ . In this case, the columns of  $\mathbf{U}$  (left-singular vectors) are the eigenvectors.

### 4.2.2 Singular values and 2-norm

Given a symmetric matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , we have that  $\mathbf{A}^T \mathbf{A} = \mathbf{A}^2 = \mathbf{A} \mathbf{A}^T$  (as  $\mathbf{A}^T = \mathbf{A}$ ). The eigenvalues of  $\mathbf{A}^2$  are  $\lambda_1^2, \dots, \lambda_n^2$ , where  $\lambda_i$  are eigenvalues of  $\mathbf{A}$ . Alternatively, the eigenvalues of  $\mathbf{A}^2$  are the squared singular values of  $\mathbf{A}$ :  $\lambda_i^2 = \sigma_i^2$ . Moreover, the eigenvalues of  $\mathbf{A}^{-1}$  are  $\frac{1}{\lambda_1}, \dots, \frac{1}{\lambda_n}$ .

We can compute the 2-norm as:

2-norm using SVD

$$\|\mathbf{A}\|_2 = \sqrt{\rho(\mathbf{A}^T \mathbf{A})} = \sqrt{\rho(\mathbf{A}^2)} = \sqrt{\max\{\sigma_1^2, \dots, \sigma_r^2\}} = \sigma_1$$

$$\|\mathbf{A}^{-1}\|_2 = \sqrt{\rho((\mathbf{A}^{-1})^T (\mathbf{A}^{-1}))} = \sqrt{\rho((\mathbf{A} \mathbf{A}^T)^{-1})} = \sqrt{\rho((\mathbf{A}^2)^{-1})} = \sqrt{\max\{\frac{1}{\sigma_1^2}, \dots, \frac{1}{\sigma_r^2}\}} = \frac{1}{\sigma_r}$$

Furthermore, we can compute the condition number of  $\mathbf{A}$  as:

$$K(\mathbf{A}) = \|\mathbf{A}\|_2 \cdot \|\mathbf{A}^{-1}\|_2 = \sigma_1 \cdot \frac{1}{\sigma_r}$$

### 4.2.3 Application: Matrix approximation

Given a matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and its SVD decomposition  $\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$ , we can construct a rank-1 matrix (dyad)  $\mathbf{A}_i \in \mathbb{R}^{m \times n}$  as:

Dyad

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

where  $\mathbf{u}_i \in \mathbb{R}^m$  is the  $i$ -th column of  $\mathbf{U}$  and  $\mathbf{v}_i \in \mathbb{R}^n$  is the  $i$ -th column of  $\mathbf{V}$ . Then, we can compose  $\mathbf{A}$  as a sum of dyads:

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

By considering only the first  $k < r$  singular values, we can obtain a rank- $k$  approximation of  $\mathbf{A}$ :

Rank- $k$  approximation

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

**Theorem 4.2.1** (Eckart-Young). Given  $\mathbf{A} \in \mathbb{R}^{m \times n}$  of rank  $r$ . For any  $k \leq r$  (this theorem is interesting for  $k < r$ ), the rank- $k$  approximation is:

$$\hat{\mathbf{A}}(k) = \arg \min_{\mathbf{B} \in \mathbb{R}^{m \times n}, \text{rank}(\mathbf{B})=k} \|\mathbf{A} - \mathbf{B}\|_2$$

In other words, among all the possible projections,  $\hat{\mathbf{A}}(k)$  is the closer one to  $\mathbf{A}$ . Moreover, the error of the rank- $k$  approximation is:

$$\|\mathbf{A} - \hat{\mathbf{A}}(k)\|_2 = \left\| \sum_{i=1}^r \sigma_i \mathbf{A}_i - \sum_{j=1}^k \sigma_j \mathbf{A}_j \right\|_2 = \left\| \sum_{i=k+1}^r \sigma_i \mathbf{A}_i \right\|_2 = \sigma_{k+1}$$

### Image compression

Each dyad requires  $1 + m + n$  (respectively for  $\sigma_i$ ,  $\mathbf{u}_i$  and  $\mathbf{v}_i$ ) numbers to be stored. A rank- $k$  approximation requires to store  $k(1 + m + n)$  numbers. Therefore, the compression factor is given by:

Compression factor

$$c_k = 1 - \frac{k(1 + m + n)}{mn}$$

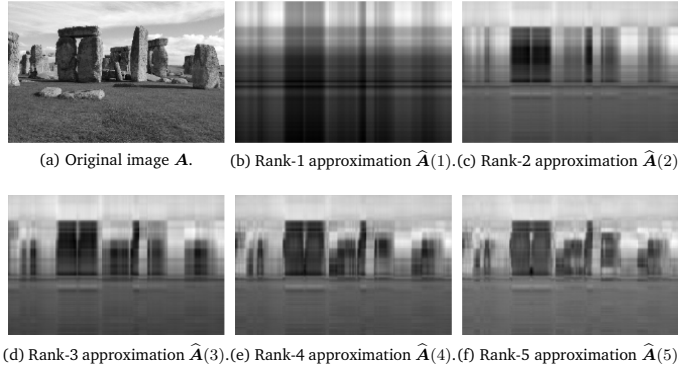


Figure 4.1: Approximation of an image

### 4.2.4 Application: Linear least squares problem

A system  $\mathbf{A}\mathbf{x} = \mathbf{b}$  with  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $m > n$  does not generally have a solution. Therefore, instead of finding the exact solution, it is possible to search for a  $\tilde{\mathbf{x}}$  such that:

Linear least squares

$$\mathbf{A}\tilde{\mathbf{x}} - \mathbf{b} \approx \bar{\mathbf{0}}$$

In other words, we aim to find a  $\tilde{\mathbf{x}}$  that is close enough to solve the system. This problem is usually formulated as:

$$\tilde{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

It always admits a solution and, depending on  $\text{rank}(\mathbf{A})$ , there two possible cases:

**rank**( $\mathbf{A}$ ) =  $n$  The solution is unique for each  $\mathbf{b} \in \mathbb{R}^m$ . It is found by solving the normal equation:

Normal equation

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

$\mathbf{A}^T \mathbf{A}$  is symmetric definite positive and the system can be solved using the Cholesky factorization.

**rank(A) < n** The system admits infinite solutions. Of all the solutions  $S$ , we are interested in the one with minimum norm:

Least squares using  
SVD

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in S} \|\mathbf{x}\|_2$$

This problem can be solved using SVD:

$$\mathbf{x}^* = \sum_{i=1}^{\text{rank}(\mathbf{A})} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

#### 4.2.5 Application: Polynomial interpolation

Given a set of  $m$  data  $(x_i, y_i), i = 1, \dots, m$ , we want to find a polynomial of degree  $n$  ( $m > n$ ) that approximates it. In other words, we want to find a function:

Polynomial  
interpolation

$$f(x) = c_0 + c_1x + c_2x^2 + \dots + c_nx^n$$

that minimizes the residual vector  $\mathbf{r} = (r_1, \dots, r_m)$ , where  $r_i = |y_i - f(x_i)|$ . We can formulate this as a linear system:

$$\mathbf{r} = \mathbf{y} - \mathbf{A}\mathbf{c} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} - \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix}$$

that can be solved as a linear least squares problem:

$$\min_{\mathbf{c} \in \mathbb{R}^n} \|\mathbf{y} - \mathbf{A}\mathbf{c}\|_2^2$$

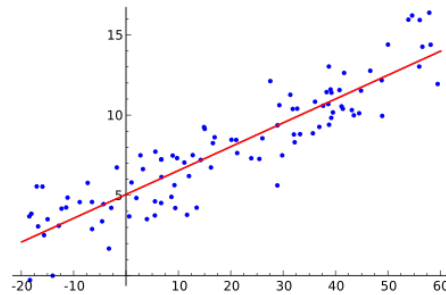


Figure 4.2: Interpolation using a polynomial of degree 1

### 4.3 Eigendecomposition vs SVD

Eigendecomposition $\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1}$	SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$
Only defined for square matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$ with eigenvectors that form a basis of $\mathbb{R}^n$	Always exists
$\mathbf{P}$ is not necessarily orthogonal	$\mathbf{U}$ and $\mathbf{V}$ are orthogonal
The elements on the diagonal of $\mathbf{D}$ may be in $\mathbb{C}$	The elements on the diagonal of $\mathbf{\Sigma}$ are all non-negative reals
For symmetric matrices, eigendecomposition and SVD are the same	

# 5 Vector calculus

## 5.1 Gradient of real-valued multivariate functions

**Gradient** Given a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , the gradient is a row vector containing the partial derivatives of  $f$ : Gradient

$$\nabla f(\mathbf{x}) = \left( \frac{\partial f(\mathbf{x})}{\partial x_1} \quad \frac{\partial f(\mathbf{x})}{\partial x_2} \quad \dots \quad \frac{\partial f(\mathbf{x})}{\partial x_n} \right) \in \mathbb{R}^{1 \times n}$$

**Hessian** Given a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , the Hessian matrix  $\mathbf{H} \in \mathbb{R}^{n \times n}$  contains the second derivatives of  $f$ : Hessian matrix

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \dots & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

In other words,  $H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ . Moreover,  $\mathbf{H}$  is symmetric.

### 5.1.1 Partial differentiation rules

**Product rule** Let  $f, g : \mathbb{R}^n \rightarrow \mathbb{R}$ : Product rule

$$\frac{\partial}{\partial \mathbf{x}}(f(\mathbf{x})g(\mathbf{x})) = \frac{\partial f}{\partial \mathbf{x}}g(\mathbf{x}) + f(\mathbf{x})\frac{\partial g}{\partial \mathbf{x}}$$

**Sum rule** Let  $f, g : \mathbb{R}^n \rightarrow \mathbb{R}$ : Sum rule

$$\frac{\partial}{\partial \mathbf{x}}(f(\mathbf{x}) + g(\mathbf{x})) = \frac{\partial f}{\partial \mathbf{x}} + \frac{\partial g}{\partial \mathbf{x}}$$

**Chain rule** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $\mathbf{g}$  a vector of  $n$  functions  $g_i : \mathbb{R}^m \rightarrow \mathbb{R}$ : Chain rule

$$\frac{\partial}{\partial \mathbf{x}}(f \circ \mathbf{g})(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}}(f(\mathbf{g}(\mathbf{x}))) = \frac{\partial f}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$$

More precisely, considering a  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  of two variables  $g_1(t), g_2(t) : \mathbb{R} \rightarrow \mathbb{R}$  that are functions of  $t$ . The gradient of  $f$  with respect to  $t$  is:

$$\frac{df}{dt} = \begin{pmatrix} \frac{\partial f}{\partial g_1} & \frac{\partial f}{\partial g_2} \end{pmatrix} \begin{pmatrix} \frac{\partial g_1}{\partial t} \\ \frac{\partial g_2}{\partial t} \end{pmatrix} = \frac{\partial f}{\partial g_1} \frac{\partial g_1}{\partial t} + \frac{\partial f}{\partial g_2} \frac{\partial g_2}{\partial t}$$

In other words, the first matrix represents the gradient of  $f$  w.r.t. its variables and the second matrix contains in the  $i$ -th row the gradient of  $g_i$ .

Therefore, if  $g_i$  are in turn multivariate functions  $g_1(s, t), g_2(s, t) : \mathbb{R}^2 \rightarrow \mathbb{R}$ , the chain rule can be applies as:

$$\frac{df}{d(s, t)} = \begin{pmatrix} \frac{\partial f}{\partial g_1} & \frac{\partial f}{\partial g_2} \end{pmatrix} \begin{pmatrix} \frac{\partial g_1}{\partial s} & \frac{\partial g_1}{\partial t} \\ \frac{\partial g_2}{\partial s} & \frac{\partial g_2}{\partial t} \end{pmatrix}$$

**Example.** Let  $f(x_1, x_2) = x_1^2 + 2x_2$ , where  $x_1 = \sin(t)$  and  $x_2 = \cos(t)$ .

$$\begin{aligned}\frac{df}{dt} &= \frac{\partial f}{\partial x_1} \frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2} \frac{\partial x_2}{\partial t} \\ &= (2x_1)(\cos(t)) + (2)(-\sin(t)) \\ &= 2\sin(t)\cos(t) - 2\sin(t)\end{aligned}$$

**Example.** Let  $h : \mathbb{R} \rightarrow \mathbb{R}$  be defined as  $h(t) = (f \circ g)(t)$  where:

$$f : \mathbb{R}^2 \rightarrow \mathbb{R} \text{ is defined as } f(\mathbf{x}) = \exp(x_1 x_2^2)$$

$$g : \mathbb{R} \rightarrow \mathbb{R}^2 \text{ is defined as } \mathbf{g}(t) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} t \cos(t) \\ t \sin(t) \end{pmatrix}$$

The gradient of  $h$  with respect to  $t$  can be computed as:

$$\begin{aligned}\frac{dh}{dt} &= \frac{\partial f}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial t} = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{pmatrix} \begin{pmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{pmatrix} \\ &= (\exp(x_1 x_2^2) x_2^2 \quad 2 \exp(x_1 x_2^2) x_1 x_2) \begin{pmatrix} \cos(t) + (-t \sin(t)) \\ \sin(t) + t \cos(t) \end{pmatrix}\end{aligned}$$

**Example** (Gradient of a least squares loss). Given a linear model defined on  $\theta$ :

Least squares loss  
gradient

$$\mathbf{y} = \Phi \theta$$

with  $\theta \in \mathbb{R}^D$ ,  $\Phi \in \mathbb{R}^{N \times D}$  and  $\mathbf{y} \in \mathbb{R}^N$ . We can define the least squares loss function as:

$$\begin{aligned}L(\mathbf{e}) &= \|\mathbf{e}\|_2^2 \\ \mathbf{e}(\theta) &= \mathbf{y} - \Phi \theta\end{aligned}$$

It must be noted that:

$$L(\mathbf{e}) = \|\mathbf{e}\|_2^2 = \mathbf{e}^T \mathbf{e} = \sum_{i=1}^N \mathbf{e}_i^2$$

To compute the gradient of  $L$  with respect to  $\theta$ , we can use the chain rule:

$$\begin{aligned}\nabla L(\theta) &= \frac{\partial L}{\partial \mathbf{e}} \frac{\partial \mathbf{e}}{\partial \theta} = (2\mathbf{e}^T)(-\Phi) \\ &= -2(\mathbf{y}^T - \theta^T \Phi^T) \Phi \\ &= -2(\mathbf{y}^T \Phi - \theta^T \Phi^T \Phi)\end{aligned}$$

Note that if we enforce  $\nabla L(\theta) = \bar{\mathbf{0}}$ , we obtain the normal equation of Section 4.2.4:

$$\begin{aligned}\nabla L = 0 &\iff -2(\mathbf{y}^T \Phi - \theta^T \Phi^T \Phi) = \bar{\mathbf{0}} \\ &\iff \mathbf{y}^T \Phi - \theta^T \Phi^T \Phi = \bar{\mathbf{0}} \\ &\iff \Phi^T \mathbf{y} - \Phi^T \Phi \theta = \bar{\mathbf{0}}\end{aligned}$$

## 5.2 Gradient of vector-valued multivariate functions

**Vector-valued function** Function  $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  with  $n \geq 1$  and  $m > 1$ . Given  $\mathbf{x} \in \mathbb{R}^n$ , the output can be represented as:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{pmatrix} \in \mathbb{R}^m$$

where  $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ .

**Jacobian** Given  $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , the Jacobian matrix  $\mathbf{J} \in \mathbb{R}^{m \times n}$  contains the first-order derivatives of  $\mathbf{f}$ : Jacobian matrix

$$\mathbf{J} = \nabla \mathbf{f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

In other words,  $J_{i,j} = \frac{\partial f_i}{\partial x_j}$ . Note that the Jacobian matrix is a generalization of the gradient in the real-valued case.

## 5.3 Backpropagation

Backpropagation is used to tune the parameters of a neural network. A neural network can be seen as a composition of many functions: Backpropagation

$$\mathbf{y} = (\mathbf{f}_K \circ \mathbf{f}_{K-1} \circ \cdots \circ \mathbf{f}_1)(\mathbf{x}) = \mathbf{f}_K(\mathbf{f}_{K-1}(\cdots \mathbf{f}_1(\mathbf{x}) \cdots))$$

Each  $\mathbf{f}_i$  takes as input the output of the previous layer  $\mathbf{x}_{i-1}$  and has the form:

$$\mathbf{f}_i(\mathbf{x}_{i-1}) = \sigma_i(\mathbf{A}_{i-1}\mathbf{x}_{i-1} + \mathbf{b}_{i-1})$$

where  $\sigma_i$  is an activation function<sup>1</sup> (a function to add nonlinearity), while  $\mathbf{A}_{i-1}$  (linear mapping) and  $\mathbf{b}_{i-1}$  (biases) are the parameters of  $\mathbf{f}_i$ .

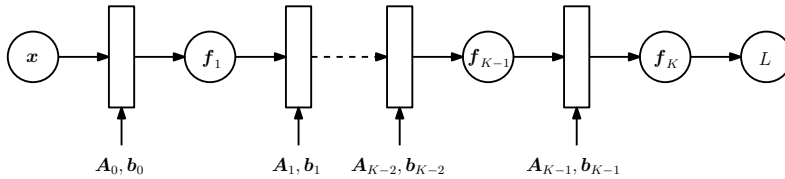


Figure 5.1: Forward pass

We can more compactly denote a neural network with input  $\mathbf{x}$  and  $K$  layers as:

$$\begin{aligned} \mathbf{f}_0 &= \mathbf{x} \\ \mathbf{f}_i &= \sigma_i(\mathbf{A}_{i-1}\mathbf{f}_{i-1} + \mathbf{b}_{i-1}) \quad i = 1, \dots, K \end{aligned}$$

Given the ground truth  $\mathbf{y}$ , we want to find the parameters  $\mathbf{A}_j$  and  $\mathbf{b}_j$  that minimizes the squared loss:

$$L(\theta) = \|\mathbf{y} - \mathbf{f}_K(\theta, \mathbf{x})\|^2$$

<sup>1</sup>[https://en.wikipedia.org/wiki/Activation\\_function](https://en.wikipedia.org/wiki/Activation_function)



where  $\theta = \{\mathbf{A}_0, \mathbf{b}_0, \dots, \mathbf{A}_{K-1}, \mathbf{b}_{K-1}\}$  are the parameters of each layer. This can be done by using the chain rule to compute the partial derivatives of  $L$  with respect to the parameters  $\theta_j = \{\mathbf{A}_j, \mathbf{b}_j\}$ :

$$\begin{aligned}
\frac{\partial L}{\partial \theta_{K-1}} &= \overbrace{\frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \theta_{K-1}}}^{\text{New}} \\
\frac{\partial L}{\partial \theta_{K-2}} &= \overbrace{\frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \theta_{K-2}}}^{\text{Known} \quad \text{New}} \\
\frac{\partial L}{\partial \theta_{K-3}} &= \overbrace{\frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \mathbf{f}_{K-2}} \frac{\partial \mathbf{f}_{K-2}}{\partial \theta_{K-3}}}^{\text{Known} \quad \text{New}} \\
&\vdots \\
\frac{\partial L}{\partial \theta_i} &= \overbrace{\frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \dots \frac{\partial \mathbf{f}_{i+2}}{\partial \mathbf{f}_{i+1}} \frac{\partial \mathbf{f}_{i+1}}{\partial \theta_i}}^{\text{Known} \quad \text{New}}
\end{aligned}$$

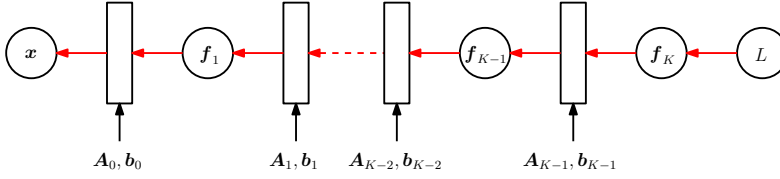


Figure 5.2: Backward pass

## 5.4 Automatic differentiation

Starting from the example below first is recommended.

Automatic differentiation allows to numerically compute the gradient of complex functions using elementary functions, intermediate variables and the chain rule through a computation graph. When the gradient has many components, it also allows to compute it more efficiently.

Automatic  
differentiation

Let  $f$  be a function,  $x_1, \dots, x_d$  the input variables of  $f$ ,  $x_{d+1}, \dots, x_{D-1}$  the intermediate variables and  $x_D$  the output variable. The computation graph can be expressed as:

$$\forall i \in \{d+1, \dots, D\} : x_i = g_i(x_{\text{Pa}(x_i)})$$

where  $g_i$  are elementary functions and  $x_{\text{Pa}(x_i)}$  are the parent nodes of  $x_i$  in the graph. In other words, each intermediate variable is expressed as an elementary function of its preceding nodes. The derivatives of  $f$  can then be computed step-by-step going backwards as:

$$\begin{aligned}
\frac{\partial f}{\partial x_D} &= 1, \text{ as by definition } f = x_D \\
\frac{\partial f}{\partial x_i} &= \sum_{\forall x_j : x_i \in \text{Pa}(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i} = \sum_{\forall x_j : x_i \in \text{Pa}(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial g_j}{\partial x_i}
\end{aligned}$$

where  $\text{Pa}(x_j)$  is the set of parent nodes of  $x_j$  in the graph. In other words, to compute the partial derivative of  $f$  w.r.t.  $x_i$ , we apply the chain rule by first computing the partial derivative of  $f$  w.r.t. the variables following  $x_i$  in the graph (as the computation goes backwards).

Automatic differentiation is applicable to all functions that can be expressed as a computational graph and when the elementary functions are differentiable. Note that backpropagation is a special case of automatic differentiation.

**Example.** Given the function:

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$$

and the elementary functions  $\{(\cdot)^2, \exp(\cdot), +, \sqrt{\cdot}, \cos(\cdot)\}$ ,  $f$  can be decomposed in the following intermediate variables:

$$a = x^2$$

$$b = \exp(a)$$

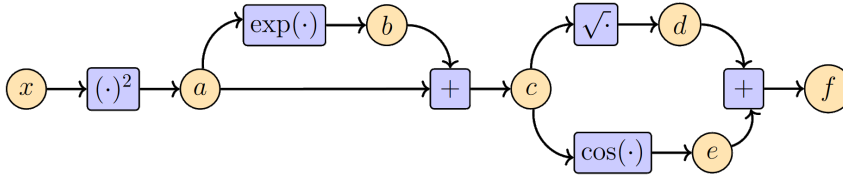
$$c = a + b$$

$$d = \sqrt{c}$$

$$e = \cos(c)$$

$$f = d + e$$

Which corresponds to the following computation graph:



We can then compute the derivatives of the intermediate variables w.r.t. their inputs (i.e. inbound edges):

$$\frac{\partial a}{\partial x} = 2x$$

$$\frac{\partial b}{\partial a} = \exp(a)$$

$$\frac{\partial c}{\partial a} = 1$$

$$\frac{\partial c}{\partial b} = 1$$

$$\frac{\partial d}{\partial c} = \frac{1}{2\sqrt{c}}$$

$$\frac{\partial e}{\partial c} = -\sin(c)$$

$$\frac{\partial f}{\partial d} = 1$$

$$\frac{\partial f}{\partial e} = 1$$

Finally, we can compute  $\frac{\partial f}{\partial x}$  by going backward from the output ( $f$ ) to the input ( $x$ ):

$$\frac{\partial f}{\partial d} = \text{already known (previous step)}$$

$$\frac{\partial f}{\partial e} = \text{already known (previous step)}$$

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} + \frac{\partial f}{\partial e} \frac{\partial e}{\partial c}$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial b}$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} \frac{\partial b}{\partial a} + \frac{\partial f}{\partial c} \frac{\partial c}{\partial a}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial x}$$

In other words, to compute the partial derivative of  $f$  w.r.t. a variable  $x_i$ , all variables  $w_j$  that follows  $x_i$  in the graph are considered.

Now, by substituting we obtain:

$$\frac{\partial f}{\partial c} = 1 \cdot \frac{1}{2\sqrt{c}} + 1 \cdot (-\sin(c))$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \cdot 1$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} \cdot \exp(a) + \frac{\partial f}{\partial c} \cdot 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \cdot 2x$$

## 6 Gradient methods

### 6.1 Minimum of a function

Let  $f : \mathbb{R}^N \rightarrow \mathbb{R}$  be continuous and differentiable in  $\mathbb{R}^N$ .

**Stationary point**  $\mathbf{x}^*$  is a stationary point of  $f$  iff:

$$\nabla f(\mathbf{x}^*) = \bar{\mathbf{0}}$$

Stationary point

**Local minimum**  $\mathbf{x}^* \in \mathbb{R}^N$  is a local minimum of  $f$  iff:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^N : \|\mathbf{x} - \mathbf{x}^*\| < \varepsilon$$

Local minimum

**Strict local minimum**  $\mathbf{x}^* \in \mathbb{R}^N$  is a strict local minimum of  $f$  iff:

$$f(\mathbf{x}^*) < f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^N : \|\mathbf{x} - \mathbf{x}^*\| < \varepsilon$$

Strict local minimum

**Global minimum**  $\mathbf{x}^* \in \mathbb{R}^N$  is a global minimum of  $f$  iff:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^N$$

Global minimum

**Strict global minimum**  $\mathbf{x}^* \in \mathbb{R}^N$  is a strict global minimum of  $f$  iff:

$$f(\mathbf{x}^*) < f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^N$$

Strict global minimum

Note that  $\max f(x) = \min -f(x)$ .

#### 6.1.1 Optimality conditions

**First order condition** Let  $f : \mathbb{R}^N \rightarrow \mathbb{R}$  be continuous and differentiable in  $\mathbb{R}^N$ .

First order condition

$$\text{If } \mathbf{x}^* \text{ local minimum of } f \Rightarrow \nabla f(\mathbf{x}^*) = \bar{\mathbf{0}}$$

**Second order condition** Let  $f : \mathbb{R}^N \rightarrow \mathbb{R}$  be continuous and twice differentiable.

Second order condition

$$\text{If } \nabla f(\mathbf{x}^*) = \bar{\mathbf{0}} \text{ and } \nabla^2 f(\mathbf{x}^*) \text{ positive definite} \Rightarrow \mathbf{x}^* \text{ strict local minimum of } f$$

As the second order condition requires to compute the Hessian matrix, which is expensive, in practice only the first order condition is checked.

### 6.2 Descent methods

Descent methods are iterative methods that have the property:

Descent methods

$$f(\mathbf{x}_k) < f(\mathbf{x}_{k+1})$$

The iteration is defined as:

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

where  $\mathbf{p}_{k-1} \in \mathbb{R}^N$  is the search direction and  $\alpha_{k-1} \in \mathbb{R}$  is the step length.

Search direction  
Step length

Note: descent methods usually converge to a local minimum.

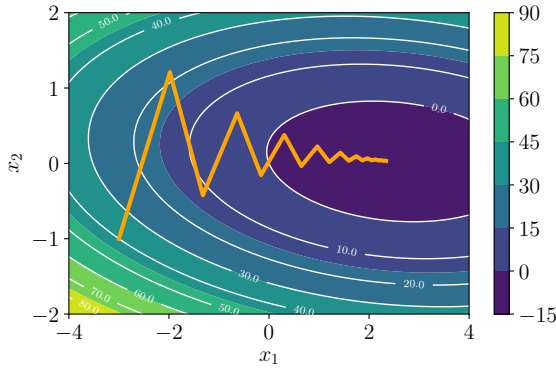


Figure 6.1: Descent method steps in  $\mathbb{R}^2$  (i.e. moving across contour lines)

### 6.2.1 Choice of the search direction

**Descent direction**  $\mathbf{p} \in \mathbb{R}^N$  is a descent direction of  $f$  in  $\mathbf{x}$  if:

Descent direction

$$\exists \bar{\alpha} > 0, \forall \alpha \in [0, \bar{\alpha}] : f(\mathbf{x} + \alpha \mathbf{p}) < f(\mathbf{x})$$

**Theorem 6.2.1.** Let  $\mathbf{p} \in \mathbb{R}^N$ ,  $\mathbf{p} \neq \bar{\mathbf{0}}$ .

If  $\mathbf{p}^T \nabla f(\mathbf{x}) < 0 \Rightarrow \mathbf{p}$  descent direction of  $f$  in  $x$

**Theorem 6.2.2.** For all  $\mathbf{x}$ ,  $\mathbf{p} = -\nabla f(\mathbf{x})$  is a descent direction of  $f$  in  $x$ .

*Proof.*

$$\begin{aligned} \mathbf{p}^T \nabla f(\mathbf{x}) < 0 &\iff -(\nabla f(\mathbf{x}))^T \nabla f(\mathbf{x}) < 0 \\ &\iff -\|\nabla f(\mathbf{x})\|_2^2 < 0 \end{aligned}$$

This holds as the norm is always positive. □

**Gradient-like methods** Gradient-like methods are descent methods that use  $-\nabla f$  as step.

Gradient-like methods

### 6.2.2 Choice of the step length

**Constant** In machine learning, it is common to set a constant value for the step (learning rate), but it can be proved that this does not guarantee convergence.

**Backtracking procedure**  $\alpha_k$  is chose such that it respects the Wolfe condition<sup>1</sup>:

Backtracking procedure

```
def backtracking( $\tau$ ,  $c_1$ ):
     $\alpha_k = 1$  # Initial guess
    while  $f(x_k + \alpha_k \nabla f(\mathbf{x}_k)) > f(\mathbf{x}_k) + c_1 \alpha_k \nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)$ :
         $\alpha_k = \alpha_k / \tau$ 
    return  $\alpha_k$ 
```

It can be proved that, by using the backtracking procedure, gradient methods converge to a local minimum.

<sup>1</sup>[https://en.wikipedia.org/wiki/Wolfe\\_conditions](https://en.wikipedia.org/wiki/Wolfe_conditions)

### 6.2.3 Stopping condition

We can stop iterating when  $\mathbf{x}_k \approx \mathbf{x}^*$ , that is,  $\nabla f(\mathbf{x}_k) \approx \bar{\mathbf{0}}$ . We can verify this by checking the norm of the gradient against a tolerance  $\tau$ :

Stopping condition

**Absolute condition**  $\|\nabla f(x_k)\|_2 < \tau$

**Relative condition**  $\frac{\|\nabla f(x_k)\|_2}{\|\nabla f(x_0)\|_2} < \tau$

A generic gradient-like method can then be defined as:

```
def gradientMethod(f, x0):
    k = 0
    while stoppingCondition(f, xk, x0):
        pk = -∇f(xk)
        αk = backtracking(...)
        xk+1 = xk + αk pk
        k = k + 1
    return xk
```

### 6.2.4 Problems

**Choice of the initialization point** The starting point of an iterative method is a user defined parameter. For simple problems, it is usually chosen randomly in  $[-1, +1]$ .

Initialization point

For complex problems, the choice of the initialization point is critical as it may cause numerical instabilities or bad results. Heuristics can be used to select an adequate starting point.

**Flag regions and local optima** Flat regions slow down the learning speed, while a local optima causes the method to converge at a poor solution.

Flag regions and local optima

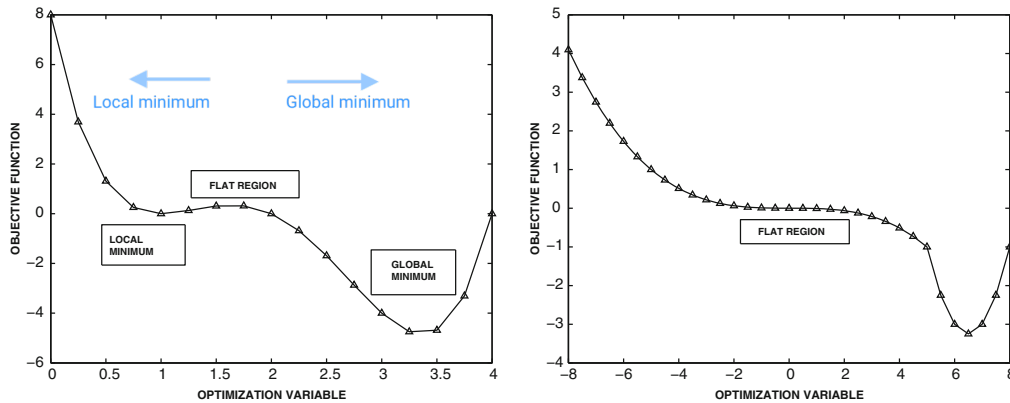


Figure 6.2: Flat regions and local minima

**Differential curvature** Different magnitudes of the partial derivatives may cause the problem of vanishing and exploding gradient. This causes the learning process to require more iterations to correct the direction.

Vanishing gradient  
Exploding gradient

In practice, as the gradient of complex functions is only an instantaneous direction of best decrease and does not represent the direction to the minimum in the long term, many updates are required for a gradient method to converge.

A method to mitigate this issue is to use feature normalization techniques.

**Non-differentiable objective function** If the objective function has a small number of non-differentiable points, the gradient descent method can be applied with minor modifications.

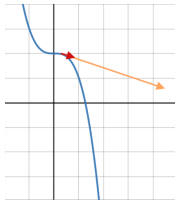
If lots of points are non-differentiable, the gradients will not be informative enough to determine a decrease direction.

**Difficult topologies** A cliff in the objective function causes problems when evaluating the gradient at the edge. With a small step size, there is a slow down in convergence. With a large step size, there is an overshoot that may cause the algorithm to diverge.

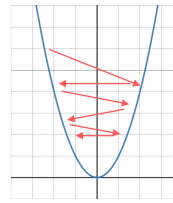
A valley in the objective function causes a gradient method to bounce between the sides to a point where no significant progress can be made.

Cliff

Valley



(a) Cliff region

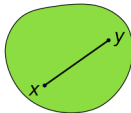


(b) Ping pong tournament in a valley

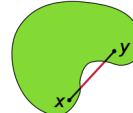
## 6.3 Convex functions

**Convex set** Informally, a set is convex if, for any two points of the set, the points laying on the segment connecting them are also part of the set.

Convex set



(a) Convex set



(b) Non-convex set

**Convex function** Let  $\Omega \subseteq \mathbb{R}^n$  be a convex set and  $f : \Omega \rightarrow \mathbb{R}$ .  $f$  is convex if:

Convex function

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in \Omega, \forall t \in [0, 1] : f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2)$$

In other words, the segment connecting two points of the function lays above the graph.

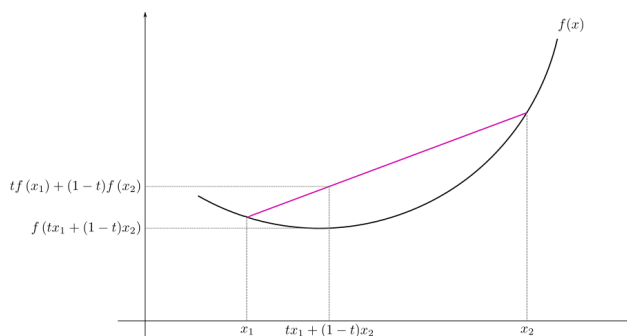


Figure 6.5: Convex function

**Strictly convex function** Let  $\Omega \subseteq \mathbb{R}^n$  be a convex set and  $f : \Omega \rightarrow \mathbb{R}$ .  $f$  is strictly convex if:

Strictly convex function

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in \Omega, \forall t \in [0, 1] : f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) < tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2)$$

### 6.3.1 Properties

Convex properties

- if  $f$  convex  $\Rightarrow$  any local minimum of  $f$  is also global
- if  $f$  strictly convex  $\Rightarrow$  the global minimum of  $f$  is unique
- if  $f$  convex and differentiable  $\Rightarrow$  any stationary point of  $f$  is a global minimum

### 6.3.2 Quadratic functions

A quadratic function has form:

Quadratic function

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

where  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,  $\mathbf{b} \in \mathbb{R}^n$  and  $c \in \mathbb{R}$ .

**Theorem 6.3.1.** If  $f$  is a quadratic form with  $\mathbf{A} \in \mathbb{R}^{n \times n}$  symmetric positive semidefinite, then  $f$  is convex.

**Theorem 6.3.2.** If  $f$  is a quadratic form with  $\mathbf{A} \in \mathbb{R}^{n \times n}$  symmetric positive definite, then  $f$  is strictly convex.

**Theorem 6.3.3.** The least squares problem  $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$  is a quadratic function.

Least squares quadratic function

*Proof.*

$$\begin{aligned} (\mathbf{A}\mathbf{x} - \mathbf{b})^T (\mathbf{A}\mathbf{x} - \mathbf{b}) &= (\mathbf{x}^T \mathbf{A}^T - \mathbf{b}^T) (\mathbf{A}\mathbf{x} - \mathbf{b}) \\ &= \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{A} \mathbf{x} - \mathbf{x}^T \mathbf{A}^T \mathbf{b} + \mathbf{b}^T \mathbf{b} \end{aligned}$$

As  $\mathbf{b}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T \mathbf{A}^T \mathbf{b}$ , we have:

$$\mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} - 2\mathbf{x}^T \mathbf{A}^T \mathbf{b} + \mathbf{b}^T \mathbf{b}$$

Let  $\mathbf{B} = \mathbf{A}^T \mathbf{A}$ ,  $\mathbf{q} = \mathbf{A}^T \mathbf{b}$  and  $c = \mathbf{b}^T \mathbf{b}$ , we have the quadratic form:

$$\mathbf{x}^T \mathbf{B} \mathbf{x} - 2\mathbf{x}^T \mathbf{q} + c$$

$\mathbf{B}$  is symmetric positive semidefinite (i.e.  $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$  is convex). Moreover, when  $\mathbf{A}$  is full-rank,  $\mathbf{B}$  is symmetric positive definite (i.e. strictly convex).  $\square$

## 6.4 Gradient descent with momentum

The momentum is an additional term to keep track of previous iterations:

Momentum

$$\Delta \mathbf{x}_k = \mathbf{x}_k - \mathbf{x}_{k-1} = \gamma \Delta \mathbf{x}_{k-1} - \alpha_{k-1} \nabla f(\mathbf{x}_{k-1})$$

where  $\gamma \in [0, 1]$ . An iteration is therefore defined as:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha_{k-1} \nabla f(\mathbf{x}_{k-1}) + \gamma \Delta \mathbf{x}_{k-1}$$



## 6.5 Stochastic gradient descent (SGD)

SGD is a stochastic approximation of gradient descent that uses an approximation of the gradient. Given  $N$  data points, the loss can be defined as the sum of the individual losses:

Stochastic gradient descent

$$L(\mathbf{x}) = \sum_{n=1}^N L_n(\mathbf{x})$$

where  $\mathbf{x}$  is the vector of parameters. The corresponding gradient can be computed as:

$$\nabla L(\mathbf{x}) = \sum_{n=1}^N \nabla L_n(\mathbf{x})$$

SGD reduces the amount of computation by approximating the gradient with a subset (mini-batch)  $B$  of  $\nabla L_n$ :

Mini-batch

$$\nabla L(\mathbf{x}) = \sum_{i \in B} \nabla L_i(\mathbf{x})$$

**Theorem 6.5.1.** Under some assumptions and with an appropriate decrease in learning rate, SGD is guaranteed to converge to a local minimum.

Different sizes of the mini-batch result in different behavior:

**Large mini-batches** accurate estimates of the gradient.

**Small mini-batches** faster computation.