

# Contents

<b>1 Numeri finiti</b>	<b>3</b>
1.1 Numeri binari ed in base N . . . . .	3
<b>2 hklklk</b>	<b>4</b>
2.1 Che cos'è il floating point . . . . .	4
2.2 Definizione formale . . . . .	4
2.3 Gap . . . . .	4
2.4 floating point nei calcolatori . . . . .	5
2.5 troncamento . . . . .	6
2.6 Errore relativo di arrotondamento . . . . .	6
2.6.1 definizione . . . . .	6
2.7 aritmetica floating point . . . . .	7
2.7.1 operazione floating point . . . . .	7
<b>3 Robe</b>	<b>9</b>
3.1 Ricerca della radice tramite bisezione :) . . . . .	9
3.1.1 Primo algoritmo di bisezione: n-iterativo . . . . .	9
3.1.2 Secondo algoritmo di bisezione: tolleranza d'errore . . . . .	9
3.2 Autovalore e autovettori . . . . .	11
3.2.1 Definizione . . . . .	11
3.2.2 decomposizione agli autovalori . . . . .	11
3.2.3 Vettori ortogonali e ortonormali . . . . .	12
3.2.4 Matrici ortogonali . . . . .	12
3.2.5 Fattorizzazione in Valori Singolar (SVD) . . . . .	12
3.2.6 numero di condizione . . . . .	14
3.3 Problema dei minimi quadrati . . . . .	14
3.3.1 Equazioni normali . . . . .	15
3.3.2 Decomposizione in valori singolari (SVD) . . . . .	16
3.3.3 Matrice pseudoinversa . . . . .	17
<b>4 Algoritmi per il calcolo di sistemi lineari</b>	<b>19</b>
4.1 Metodi diretti . . . . .	20
4.1.1 algortimo delle sostituzioni . . . . .	21
4.1.2 Fattorizzazione LU . . . . .	22
4.1.3 Eliminazione di Gauss . . . . .	24
4.1.4 algoritmo di gauss rivisto come metodo di fattorizzazione LU . . . . .	24
<b>5 Problema di interpolazione</b>	<b>26</b>
5.1 Esistenza dell'unicità . . . . .	26
5.2 Calcolo del polinomio di interpolazione . . . . .	27
5.2.1 Metodo classico . . . . .	27
5.2.2 Metodo del polinomio interpolatore di Lagrange . . . . .	28

5.3	Errore di un polinomio interpolatore di una funzione . . . . .	28
5.4	Condizionamento dell'interpolazione polinomiale . . . . .	30
<b>6</b>	<b>Problemi diretti</b>	<b>32</b>
<b>7</b>	<b>Problemi inversi</b>	<b>32</b>
7.1	Problemi inversi lineari con matrici mal condizionate . . . . .	33
7.2	Perturbazione dell'errore . . . . .	33
7.2.1	Condizioni di Picard . . . . .	34
7.3	Metodi di regolarizzazione . . . . .	36
7.3.1	Decomposizione in Valori Singolari Troncata . . . . .	36
7.3.2	Regolarizzazione di Tikhonov . . . . .	36
7.3.3	Principio di massima discrepanza . . . . .	37
<b>8</b>	<b>Estensione di una matrice</b>	<b>41</b>
<b>9</b>	<b>Point Spread Function</b>	<b>44</b>
9.1	PSF con errori non deterministici . . . . .	44
9.1.1	Soluzione naive . . . . .	45
9.2	Metriche di valutazione . . . . .	45
9.2.1	errore relativo . . . . .	45
9.2.2	Rapporto Segnale-Rumore di Picco . . . . .	46
9.2.3	Indice di Similarità Strutturale (SSIM) . . . . .	46
9.2.4	Roba . . . . .	47
9.3	Regolarizzazione di Tikhonov applicata alle immagini . . . . .	47
9.3.1	Regolarizzazione con Variazione totale . . . . .	47

# 1 Numeri finiti

## 1.1 Numeri binari ed in base N

I numeri hanno diversi modi per essere rappresentati, tra cui quello più comune che è la **rappresentazione decimale**, dove ogni numero è formato da una lista di caratteri compresi tra  $[0, 9]$  dove ad ogni cifra è associata una potenza del 10

**Esempio:**  $147.3 = 1 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0 + 3 \cdot 10^{-1}$

In generale, comunque, per convertire un numero da una base  $n$  a base 10:

teorema

Dato un numero in base  $n$   $(a_k a_{k-1} \dots a_1 a_0)_n$ , la sua conversione in base 10 è data dalla formula:

$$\sum_{i=0}^k a_i \cdot n^i$$

dove  $a_i$  sono le cifre in base  $n$  e  $n$  è la base.

Per forza di cose i calcolatori operano con i numeri *in base 2*, le cui cifre vengono chiamate **bit**, esempio 101101 (base 2). Per convertire un numero dalla base 10 alla base 2 bisogna dividerlo in potenze di due

$$37 \text{ (base 10)} = 32 + 4 + 1 = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 100101 \text{ (base 2)}$$

In generale occorre dividerlo per due finché non si riduce ad uno ad esempio:

$$(35)_{10} = (100011)_2$$

$$\begin{array}{l} 35 : 2 = 17 \quad \text{resto } 1 \\ 17 : 2 = 8 \quad \text{resto } 1 \\ 8 : 2 = 4 \quad \text{resto } 0 \\ 4 : 2 = 2 \quad \text{resto } 0 \\ 2 : 2 = 1 \quad \text{resto } 0 \\ 1 : 2 = 0 \quad \text{resto } 1 \end{array}$$

## 2 hklklk

### 2.1 Che cos'è il floating point

Dato che in informatica non è possibile implementare il concetto di infinito (a differenza della matematica) non si è in grado di rappresentare tutto l'insieme dei reali; pertanto occorre, per forza di cose, approssimarlo. Il metodo che oggi è comune per il calcolo scientifico è noto come sistema **floating point** che permette una rappresentazione di ampio intervallo della retta reale con una distribuzione uniforme degli errori

### 2.2 Definizione formale

Si definisce **insieme dei numeri numeri macchina** (floating point) con  $t$  cifre significative, con base  $\beta$ , con  $p \in [L, U]$  e con le cifre  $d_i$  dove  $0 \leq d_i \leq \beta - 1$ ,  $i = 1, 2, \dots$  e  $d_1 \neq 0$

$$\mathbb{F}(\beta, t, L, U) = \{0\} \cup \{x \in \mathbb{R} = \text{sign}(x)\beta^p \sum_{i=1}^t d_i \beta^{-i}\}$$

Usualmente  $U$  è positivo e  $L$  negativo

Questo insieme è, quindi, un'approssimazione all'intervallo preciso di numeri  $[min, max] \subseteq \mathbb{R}$  che dipendono da  $U$  e  $L$ . Se voglio rappresentare un numero maggiore/minore di questo intervallo avrò un errore chiamato **overflow/underflow**

### 2.3 Gap

con questa notazione non è impossibile scrivere ogni numero presente nella retta dei numeri Reali  $\mathbb{R}$ , ma tra due numeri vi è sempre una certa distanza quindi è pertanto vero che  $\mathbb{F} \subseteq \mathbb{R}$ . Questo concetto introduce la definizione di Gap:

Chiamiamo **gap** la distanza da un numero al suo successivo

È, inoltre, vero che il gap rimane costante quando l'esponente non cambia, mentre se nell'insieme  $\mathbb{F}$  il successivo di un numero ha un esponente diverso il suo gap sarà diverso di quello del suo precedente con l'esponente uguale. Facciamo un esempio per rendere il tutto più chiaro:

Sia  $\beta = 2, t = 3, L = -1, U = 2$  allora

$$\mathbb{F}(2, 3, -1, 2) = \{0\} \cup \{0.100 \times 2^p, 0.101 \times 2^p, 0.110 \times 2^p, 0.111 \times 2^p, p = -1, 0, 1, 2\}$$

Questo insieme rappresenta 33 numeri compreso lo 0:

	.101(-1)	.110(-1)	.111(-1)
	$\frac{5}{16}$	$\frac{6}{16}$	$\frac{7}{16}$
.100(-1)	.101(0)	.110(0)	.111(0)
$\frac{1}{4}$	$\frac{5}{8}$	$\frac{6}{8}$	$\frac{7}{8}$
.100(0)	.101(1)	.110(1)	.111(1)
$\frac{1}{2}$	$\frac{5}{4}$	$\frac{6}{4}$	$\frac{7}{4}$
.100(1)	.101(2)	.110(2)	.111(2)
1	$\frac{5}{2}$	$\frac{6}{2}$	$\frac{7}{2}$
.100(2)			
2			

La rappresentazione di  $\mathbb{F}$  nella retta dei numeri reali è questa



retta\_reale.png

Risulta quindi evidente l'aumento dell'ampiezza degli intervalli definiti dal valore  $p$ , in ognuno dei quali vengono localizzati le  $t$  cifre della mantissa; ne

risulta di conseguenza una diradazione delle suddivisioni verso gli estremi sinistro e destro della retta reale.

Il numero  $t$ , quindi, fissa la precisione di rappresentazione di ogni numero; infatti la retta viene suddivisa in intervalli  $[\beta^p, \beta^{p+1}]$  di ampiezza crescente e in questi intervalli viene vengono rappresentati lo stesso numero  $(\beta - 1)\beta^{t-1}$  di valori, generando ,così, una suddivisione molto densa per valori vicini allo 0 e più rada per valori grandi in valore assoluto. Di conseguenza si ha che **maggiore è il numero  $t$  di cifre della mantissa, minore sarà l'ampiezza dei intervalli e quindi migliore l'approssimazione introdotta**

## 2.4 floating point nei calcolatori

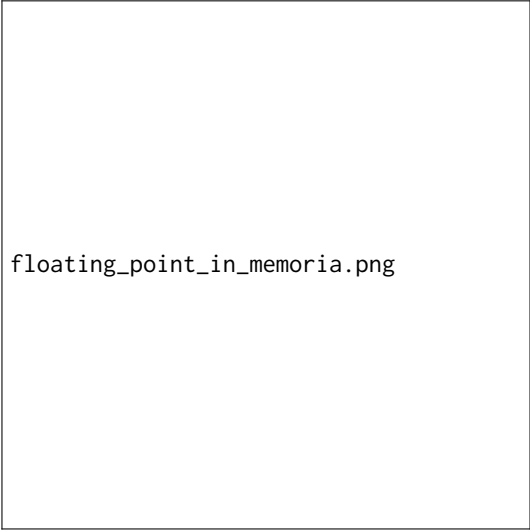
La rappresentazione più comune del sistema nei calcolatori è definita dallo Standard IEEE 754 il cui insieme è  $\mathbb{F}(2, 52, 1023, 1024)$  ed è descritta dalla formula

$$n = (-1)^s \cdot 2^{e-1023} \cdot (1 + f)$$

Dove:

- **s**: è un bit che determina il segno del numero. Se  $s = 0$  il numero è positivo, se  $s = 1$  il numero è negativo.
- **e**: è l'esponente del numero due, ed è un intero di 11 bit tale che  $e \in [0, 2047]$ .
- **f**: il valore detto **mantissa** è una serie di 52 bit che rappresenta la parte frazionaria del numero, nel nostro caso  $f \in [0, 1]$ .
- **1023**: è detto **bias** ed è usato per codificare esponenti negativi e positivi.

In memoria si traduce in un array contenente questi bit:



floating\_point\_in\_memoria.png

Per trovare la codifica del numero in floating point  $1\ 10000000010\ 1000$  in base 10 occorre:

1. Trovare l'esponente decimale, che è  $1 \times 2^{10} + 1 \times 10^1 - 1023 = 3$ .
2. Trovare la parte frazionaria, che è  $1 \times \frac{1}{2^1} = 0,5$ .

Quindi  $n = (-1)^1 \times 2^3 \times (1 + 0,5) = -12,0$ .

- [illegible]

[illegible]

## 2.5 troncamento

Sia l'insieme  $\mathbb{F}(\beta, t, L, U) = \{\{\emptyset\} \cup n = \pm\beta^p \cdot d_0 d_1 \dots d_t\}$  con  $-L \leq p \leq U$  l'insieme dei numeri floating point caratterizzati dai valori  $\beta, t, L, U$

Inoltre sia  $x \in \mathbb{R}$  dove  $x = \pm \beta^p \cdot d_0 d_1 \dots d_t, d_{t+1}, \dots$  con  $L \leq p \leq U$ , nel caso in cui  $x \notin \mathbb{F}$  rappresento  $x$  con un elemento di  $\mathbb{F}$  che indico con  $fl(x) = \pm \beta^p \cdot d_0 d_1 \dots d_t$ . Questa operazione si chiama **troncamento**

## 2.6 Errore relativo di arrotondamento

Il sistema floating point dato che è un'approssimazione è ovvio che la precisione di rappresentazione che offre non è perfetta, in quanto i numeri infiniti (come il  $\pi$ ) vengono delineati con una serie finita di numeri finiti. La differenza tra la sua approssimazione e il suo valore reale è detta **errore di arrotondamento**

### 2.6.1 definizione

klkjklkjklkj

Sia  $x \in \mathbb{R}$  un numero reale e sia  $fl(x)$  (dove  $fl()$  è la funzione arrotondamento) il numero  $x$  arrotondato, allora l'errore relativo di arrotondamento è definito dalla seguente formula:

$$\frac{|fl(x) - x|}{|x|} \leq \frac{1}{2} \beta^{1-t}$$

Sia  $x \in \mathbb{R}$  un numero reale e sia  $fl(x)$  (dove  $fl()$  è la funzione arrotondamento) il numero  $x$  arrotondato, allora l'errore relativo di arrotondamento è definito dalla seguente formula:

$$\frac{|fl(x) - x|}{|x|} \leq \frac{1}{2} \beta^{1-t}$$

La quantità  $esp = \frac{1}{2}\beta^{1-t}$  è detta **precisione macchina** nel sistema floating point, ed è il più piccolo numero macchina positivo tale che:

La quantità  $esp = \frac{1}{2}\beta^{1-t}$  è detta **precisione macchina** nel sistema floating point, ed è il più piccolo numero macchina positivo tale che:

$$fl(1 + esp) > 1$$

## 2.7 artimetica floating point

Il calcolatore esegue operazioni solo con numeri rappresentabili da calcolatore stesso, quindi in questo caso dai numeri scritti in forma floating point (per i

numeri reali), il punto è che **le usuali operazioni aritmetiche non sono chiuse nell'insieme  $\mathbb{F}$** , quindi presi 2 numeri dall'insieme  $\mathbb{F}$  e computati tramite operazioni aritmetiche, il risultato di tale computazione potrebbe appartenere nell'insieme  $\mathbb{R}$ . Pertanto **i risultati delle varie computazioni dovranno essere sempre arrotondati ad un numero che stia in  $\mathbb{F}$**

### 2.7.1 operazione floating point

L'operazione floating point (o di macchina) è definita

$$\odot : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

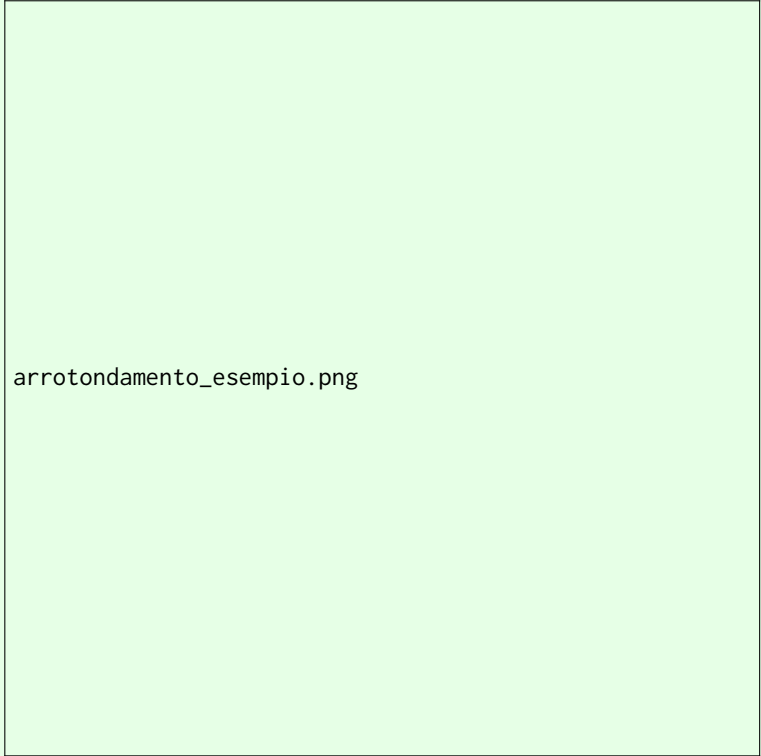
Ogni operazione provoca un piccolo errore detto **errore di arrotondamento**:

$$\left| \frac{(x \odot y) - (x \cdot y)}{x \cdot y} \right| < eps$$

l'operazione floating point nei calcolatori viene strutturata in due fasi:

1. **Esegui l'operazione esatta:** (es.  $z = x + y$ ) dove viene fatta utilizzando più bit utilizzati di quelli utilizzati per memorizzare il risultato, non è accessibile all'utente ma viene memorizzato in un registro interno alla cpu
2. **Trocamento del risultato:** (es.  $x \oplus y = fl(z)$ ), dove una volta che il calcolo è stato fatto in precisione estesa il risultato viene arrotondato alla precisione del risultato





arrotondamento\_esempio.png

## 3 Robe

### 3.1 Ricerca della radice tramite bisezione :)

Prima di tutto si analizzi il calcolo della radice di una equazione (anche non lineare)

$$f(x) = 0$$

Esempio:

Ci si chiede se esistono tali implicazioni:

- Esistenza e unicità della soluzione
- Esistenza di algoritmi per calcolare soluzione

Innanzitutto si tenga conto di tale teorema

teorema

**Teorema di Bolzano:**

Sia  $F : [a, b] \in \mathbb{R} \rightarrow \mathbb{R}$  continua in  $[a, b]$  e  $f(a) \cdot f(b) < 0 \Rightarrow \exists x^* : F(x^*) = 0$

Per la dimostrazione si vedano gli appunti/libri di analisi

#### 3.1.1 Primo algoritmo di bisezione: n-iterativo

Intuitivamente l'algoritmo di bisezione ha due fasi:

1. se  $f(a) \cdot f(c) = 0$  la radice è nell'intervallo  $[a, c]$  che diventa la nuova versione dell'intervallo  $[a, b]$
2. altrimenti, se  $f(c) \cdot f(b) = 0$  la radice è nell'intervallo  $[c, b]$  che diventa la nuova versione dell'intervallo  $[a, b]$

Ecco qui descritto l'algoritmo di bisezione in pseudocodice:

Questo tipo di soluzione per la bisezione ha un numero  $n$  di iterazioni per approssimare la soluzione, questa soluzione è molto semplice ma non la migliore. Vi è tuttavia una soluzione migliore

#### 3.1.2 Secondo algoritmo di bisezione: tolleranza d'errore

Nella pratica è molto più utile definire una costante  $\epsilon$  che vuole rappresentare l'errore di approssimazione massimo, di modo che un'approssimazione  $\bar{r}$  della radice reale  $r$  garantisca che  $|r - \bar{r}| \leq \epsilon$ . In altre parole si vuole garantire che  $r \in [\bar{r} - \epsilon, \bar{r} + \epsilon]$ .

Nel nostro caso  $\bar{r}$  è il valore di  $c = (b - a)/2$  mentre impostiamo  $\epsilon = (b - a)/2$ . Si imposti inoltre un **errore di tolleranza**  $e_{tol}$  per iterare le operazioni fino a che non si incontrino. Di seguito è riportato l'algoritmo:

---

**Algorithm 1:** primo algoritmo di Bisezione (Function f, int a, int b, int n)

---

**Data:** Funzione continua  $f(x)$ , continua in  $a$  e  $b$  con  $f(a) \cdot f(b) < 0$ , un numero  $N$  di iterazioni

**Result:** Approssimazione della radice

```

1 for 1 to n do
2    $c \leftarrow \frac{a+b}{2}$ ;
3   if  $f(c) = 0$  then
4     return  $c$  ; // Trovata la radice esatta
5   if  $f(a) \cdot f(c) < 0$  then
6      $b \leftarrow c$  ; // La radice è nell'intervallo  $[a, c]$ 
7   else
8      $a \leftarrow c$  ; // La radice è nell'intervallo  $[c, b]$ 
9 return  $c$  ; // Approssimazione della radice

```

---



---

**Algorithm 2:** Algoritmo di Bisezione

---

**Data:** Funzione continua  $f(x)$ , estremi  $a$  e  $b$  con  $f(a) \cdot f(b) < 0$ , tolleranza  $\epsilon$

**Result:** Approssimazione della radice  $x^*$  con  $|f(x^*)| < \text{Toccalarivoluzioneeperlesighedelchea2euro}\epsilon$

```

1 while  $|b - a| > \epsilon$  do
2    $c \leftarrow \frac{a+b}{2}$ ; if  $f(c) = 0$  then
3     return  $c$  ; // Trovata la radice esatta
4   if  $f(a) \cdot f(c) < 0$  then
5      $b \leftarrow c$  ; // La radice è nell'intervallo  $[a, c]$ 
6   else
7      $a \leftarrow c$  ; // La radice è nell'intervallo  $[c, b]$ 
8 return  $\frac{a+b}{2}$  ;
; // Approssimazione della radice

```

---

## 3.2 Autovalore e autovettori

### 3.2.1 Definizione

Data una matrice  $A$  di dimensione  $n \times n$  se vale:

$$Ax = \lambda x$$

con  $\lambda \in \mathbb{R}$  e  $x \in \mathbb{R}^n$ ,  $\lambda$  si dice autovalore di  $A$  e  $x$  autovettore di  $A$  associato a  $\lambda$

Data questa definizione, e richiamando i vari concetti dell'algebra lineare, abbiamo che se vale  $Av_i = \lambda_i v_i, \forall i \in 1, \dots, n$  allora vale anche:

$$[Av_1, Av_2, \dots, Av_n] = [\lambda_1 v_1, \lambda_2 v_2, \dots, \lambda_n v_n]$$

$$\text{quindi: } A[v_1, v_2, \dots, v_n] = [v_1, v_2, \dots, v_n] \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

Questa bellissima scrittura si può tranquillamente riassumere come:

$$AV = DV$$

### 3.2.2 decomposizione agli autovalori

sia  $A \in M^{n \times n}$ , allora una decomposizione agli autovalori di  $A$  è una fattorizzazione (ovvero un prodtto) di questo tipo:

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

Dove  $V = [v_1, v_2, \dots, v_n]$  è la matrice con colonne gli autovettori  $v_i$  di  $A$  e  $D$  è una matrice diagonale che ha sulla diagonale gli autovalori di  $A$  associati alle colonne di  $V$

Se  $A$  ha una decomposizione agli autovalori allora si dice che  $A$  è **diagonalizzabile**; in caso contrario  $A$  si dice **non diagonalizzabile** (o defettiva).

Se  $A$  ha  $n$  autivalori distinti, allora  $A$  è diagonalizzabile

### 3.2.3 Vettori ortogonali e ortonormali

Siano  $v_1, v_2, \dots, v_m$  vettori di  $R^n$ , si dicono **ortognonali** se:

$$v_i^T v_j = 0, \quad \text{se } i \neq j$$

Si può scrivere anche:

$$\langle v_i, v_j \rangle = 0, \quad \text{se } i \neq j$$

Ovvero il prodotto scalare

Questa relazione di vettori si collega molto bene con i vettori ortonormali:

Siano  $v_1, v_2, \dots, v_m$  vettori di  $R^n$ , si dicono **ortonormali** se sono ortogonali w di lunghezza unitaria, ovvero  $v_i^T v_j = \|v_i\| = \delta_{ij}$  dove

$$\delta_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases}$$

### 3.2.4 Matrici ortogonali

Una matrice  $W$  di dimensione  $n \times n$  si dice ortogonale se le sue colonne sono vettori ortonormali

Un esempio semplice:

$$W = \begin{pmatrix} \frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & 1 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & 0 & 0 \end{pmatrix}$$

le matrici ortogonali hanno diverse proprietà:

1.  $W^T = W^{-1}$
2.  $\forall x \in R^n, \|x\|_2 = \|Wx\|_2$

### 3.2.5 Fattorizzazione in Valori Singolar (SVD)

La fattorizzazione in valori singolari non è altro che uno strumento che permette di rappresentare una matrice  $A \in M^{m \times n}$  attraverso una matrice diagonale.

Vediamo come:

teorema

Sia  $A \in M^{m \times n}$  una matrice di rango  $k$  con  $k \leq n \leq m$ . Allora esistono:

- una matrice ortogonale  $U \in M^{m \times m}$
- una matrice ortogonale  $V \in M^{n \times n}$
- una matrice diagonale  $\Sigma \in M^{m \times n}$

tali che:

$$A = U\Sigma V^T, \quad \Sigma = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

- Gli elementi  $\sigma_1 \geq \sigma_2 \geq \dots \sigma_n \geq 0$  sono i cosiddetti **valori singolari di  $A$**
- Mentre il rango  $k$  è uguale al numero di valori singolari positivi, cioè  $k = \text{rango}(A) \iff (\sigma_1 \geq \sigma_2 \geq \dots \sigma_k > 0 \wedge \sigma_{k+1} = \dots = \sigma_n = 0)$
- Le colonne di  $U = (u_1, u_2, \dots, u_m)$  e di  $V = (v_1, v_2, \dots, v_n)$  sono, rispettivamente, i **vettori singolari sinistri e destri di  $A$**  associati ai valori singolari  $\sigma_i$  e formano una base ortonormale di  $R^m$  e  $R^n$ , rispettivamente, poichè vale la relazione:

$$Au_i = \sigma_i v_i$$

Questo teorema porta a un'implicazione piuttosto interessante, infatti si ha:

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

dato che  $U$  è ortogonale si "annulla" con la sua trasposta, quindi continua con

$$V\Sigma^T \Sigma V^T = V \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2 \end{pmatrix} V^T$$

Inoltre:

$$\sigma_i = \sqrt{\lambda_i}, i = \dots, n$$

Dove  $\lambda_i$  non è che l'autovalore di  $A^T A$ , si ha quindi:

- $\sigma_1 = \sqrt{\lambda_{max}} = \sqrt{\rho} = \|A\|$
- $\sigma_k = \sqrt{\lambda_{min}} \Rightarrow \|A^{-1}\| = \frac{1}{\sigma_k}$

### 3.2.6 numero di condizione

Si dice **numero di condizione** di  $A$  tale valore:

$$K_2(A) = \frac{\sigma_1}{\sigma_k}$$

Dove  $\sigma_k$  è il valore singolare minimo di  $A$  e  $k$  è il rango della matrice, quindi  $k \leq \min(m, n)$

Un'altra definizione di questo valore è:

$$K_2(A) = \|A\| \cdot \|A^{-1}\|$$

Un **numero di condizionamento basso** (vicino a 1) indica che la matrice è ben condizionata. Ciò significa che piccole variazioni negli input producono piccole variazioni negli output, viceversa un numero di condizionamento alto indica che la matrice è mal condizionata. Ciò significa che piccole variazioni negli input possono causare grandi variazioni negli output

### 3.3 Problema dei minimi quadrati

Si consideri il seguente sistema:

$$Ax = b$$

Dove  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$ ,  $x \in \mathbb{R}^n$  e  $m > n$ . È facile verificare che risulta indeterminato, dato che il numero di equazioni è superiore al numero di incognite, non ammettendo, quindi, alcuna soluzione.

Consci di questo fatto alquanto deprecabile e accettata la sua inalterabilità, si vuole però cercare la  $x$  tale che l'espressione  $Ax - b$  non sia 0 ma "quasi", ovvero il più piccolo valore possibile. Si è così giunti al concetto di problema ai minimi quadrati:

Si definisce il **problema ai minimi quadrati (LSQ)**, che consiste nel determinare il vettore  $x \in \mathbb{R}^n$  che minimizzi il vettore dei residui  $r = Ax - b$ :

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2 = \min_{x \in \mathbb{R}^n} \|r\|_2^2$$

dove  $r = (r_1, r_2, \dots, r_n) = (Ax_1 - b_1, \dots, Ax_n - b_n)$

Si giunge così al seguente teorema:

#### teorema

Sia  $A \in \mathbb{R}^{m \times n}$  con  $m > n$  e  $rg(A) = k \leq n$ , allora il problema

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2 = \min_{x \in \mathbb{R}^n} \|r\|_2^2$$

**Ammette sempre almeno una soluzione**, inoltre:

- $k = n$  (**quindi ha rango massimo**): il problema ha una ed una sola soluzione
- $k < n$ : il problema ha infinite soluzioni e tali soluzioni formano un sottospazio di  $\mathbb{R}^n$  di dimensione  $n - k$

### 3.3.1 Equazioni normali

Come si fa, quindi, a minimizzare questo di residuo? Bhe bisogna trovare un "punto di minimo" che ovviamente va trovato con le derivate (nel nostro caso dato che siamo in  $\mathbb{R}^n$  il gradiente). Intanto sappiamo che:

$$\begin{aligned}\|Ax - b\|_2^2 &= (Ax - b)^T (Ax - b) \\ &= (-b^T + x^T A^T) (Ax - b) \\ &= -b^T Ax + b^T b + x^T A^T Ax \\ &\quad \text{(sommando i termini simili)} \\ &\quad b^T Ax = x^T A^T b \\ &= x^T A^T Ax - 2x^T Ab + b^T b\end{aligned}$$

Si pone:

$$f(x) = x^T A^T Ax - 2x^T Ab + b^T b$$

Questa è un'equazione  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ , condizione necessaria (ma non sufficiente) che un punto  $x^*$  sia di minimo è che  $\nabla f(x^*) = 0$  dove:

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)$$

Dato che  $f$  non è altro che una sommatoria di tre addendi, non mi resta che fare il gradiente di questi per poi addizionarli. Quindi:

- $\nabla(x^T Ax) = 2A^T Ax$
- $\nabla(2x^T Ab) = 2A^T b$
- $\nabla(b^T b) = 0$

Allora:

$$\nabla f(x) = 2A^T Ax - 2A^T b$$

Imponendo che il gradiente sia nullo si ottiene il cosiddetto **Sistema delle equazioni normali**

$$A^T Ax = A^T b$$



Se  $A^T A$  è simmetrica e definita positiva allora  $A$  ha rango massimo, è possibile quindi risolvere il sistema con un opportuno metodo adatto alla matrice  $A^T A$ , ad esempio utilizzando la fattorizzazione di Cholesky, si ottiene una matrice  $L$  t.c.  $A^T A = LL^T$  e si risolvono nell'ordine i due sistemi:

$$\begin{cases} Ly = A^T b \\ L^T x = y \end{cases}$$

Tuttavia, il calcolo di  $A^T A$  può rendere il problema eccessivamente mal condizionato poiché  $K(A^T A) = K(A)^2$ , ovvero il numero di condizionamento della matrice  $A^T A$  è approssimativamente il quadrato del numero di condizionamento della matrice  $A$ . Ricordiamo che un problema mal condizionato è più difficile da risolvere numericamente, poiché anche piccoli errori di arrotondamento nei calcoli possono causare errori significativi nella soluzione finale

### 3.3.2 Decomposizione in valori singolari (SVD)

Se il rango  $k < \min(m, n)$  allora vi sono infinite di soluzioni, ma ne esiste una sola  $\bar{x}$  di norma minima. Tale soluzione si può ottenere tramite SVD

#### teorema

Sia  $A \in \mathbb{R}^{m \times n}$ , con  $rg(A) = k \leq n$  e sia  $A = U\Sigma V^T$  la sua decomposizione in valori singolari, allora il vettore:

$$x^* = \sum_i^k \frac{u_i^T b}{\sigma_i} v_i$$

è la soluzione di minima norma al problema:

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$$

per la proprietà del vettore ortogonale di non alterare le norme di vettori e in corrispondenza di tale soluzione e si ottiene

$$\|Ax - b\| = \|U^T Ax - U^T b\| = \|U^T A V V^T x - U^T b\|$$

Posto  $y = V^T x \in \mathbb{R}^n$  e  $g = U^T b \in \mathbb{R}^m$  si ha:

$$\|r\| = \|Ax^* - b\| = \sum_{i=k+1}^n (u_i^T b)^2$$

Moltiplicando  $U^T$  a destra e a sinistra si ottiene:

$$\|Ax - b\| = \|\Sigma y - g\| = \sum_{i=1}^k (\sigma_i y_i - g_i)^2 + \sum_{i=k+1}^n g_i^2$$

Per minimizzare tale quantità è sufficiente scegliere:

$$y_i = \frac{g_i}{\sigma_i} = \frac{U^T b}{\sigma_i}$$

poichè  $x^* = Vy$  risulta:

$$x^* = \sum_{i=1}^k \frac{u_i^T b}{\sigma_i} v_i$$

Allora in corrispondenza di tale soluzione, la norma del residuo è:

$$\|r\| = \sum_{i=k+1}^n (g_i)^2 = \sum_{i=k+1}^n (u_i^T b)^2$$

### 3.3.3 Matrice pseudoinversa

Qui viene riportata la definizione di una matrice con proprietà molto utili:

Sia  $A \in \mathbb{R}^{m \times n}$ , con  $rg(A) = k \leq n$  e sia  $A = U\Sigma V^T$  la sua decomposizione in valori singolari. Si definisce **pseudoinversa** la matrice:

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

Dove:

$$(\Sigma)_{ij} = \begin{cases} \frac{1}{\sigma_i} & \text{se } i = j \wedge i \leq k \\ 0 & \text{altrimenti} \end{cases}$$

Ovvero:

$$\Sigma^+ = \begin{pmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \cdots & 0 \\ 0 & 0 & \cdots & \frac{1}{\sigma_k} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & \cdots & 0 \end{pmatrix}_{m \times n}$$

Questa giga matrice gode delle seguenti proprietà:

1.  $AA^+A = A$
2.  $A^+AA^+ = A^+$
3.  $(AA^+)^T = AA^+$
4.  $(A^+A)^T = A^+A$

La pseudoinversa di una matrice rettangolare  $A$  permette di scrivere la soluzione del problema dei minimi quadrati in modo simile alla soluzione  $x = A^{-1}b$  di un sistema lineare quadrato, cioè

$$x^* = V\Sigma U^T b \implies x^* = A^+ b$$

Tuttavia sebbene la pseudoinversa sia teoricamente potente per analizzare il problema dei minimi quadrati e comprendere le proprietà delle soluzioni, il calcolo diretto della pseudoinversa di una matrice è spesso inefficiente e computazionalmente oneroso, soprattutto per matrici di grandi dimensioni. Questo è dovuto principalmente al costo della decomposizione ai valori singolari (SVD), che richiede un notevole sforzo computazionale. Pertanto non è uno strumento adeguato per calcolarle poiché computazionalmente costoso.

Si può inoltre definire il numero di condizionamento di  $A \in \mathbb{R}^{m \times n}$  in termini di pseudoinversa:

$$K(A) = \|A\| \|A^+\|$$

È il numero di condizionamento in norma -2 (o condizionamento spettrale):

$$K(A)_2 = \|A\|_2 \cdot \|A^+\|_2 = \begin{cases} \frac{\sigma_1}{\sigma_n} & k = vg(A) \\ \frac{\sigma_1}{\sigma_k} & k = rg(A) \end{cases}$$

Dove:

- **k=vg(A)**: quando  $k$  è uguale al numero di colonne di  $A$ , si sta considerando la situazione in cui la matrice ha tutte le colonne linearmente indipendenti
- **k=rg(A)**: quando  $k$  è uguale al rango effettivo di  $A$ , ovvero il numero massimo di colonne linearmente indipendenti

Il parametro di regolarizzazione, come già visto nel caso dei problemi inversi 1D, può essere scelto in modo euristico oppure applicando il criterio di Massima Discrepanza. Per verificare sperimentalmente il parametro “ottimale” nel caso in cui si abbia la soluzione esatta (cioè in un problema test) è quello di verificare quale parametro rispetto a quelli scelti produce l’errore minore rispetto alla “ground truth”  $x_{GT}$ , utilizzando per esempio l’errore relativo già definito in precedenza:

$$ER = \frac{\|x - x_{GT}\|_2^2}{\|x_{GT}\|}$$

## 4 Algoritmi per il calcolo di sistemi lineari

$$\begin{cases} a_{21}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_2 \\ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ \vdots \\ a_{n1}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_n \end{cases}$$

Con  $m$  equazioni ed  $n$  incognite. E sia  $m = n$ , corrispondente ad un sistema quadrato

Sia  $Ax = b$  con

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

e

$$x \in \mathbb{R}^n = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, b \in \mathbb{R}^n = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Allora si ha che il sistema  $Ax = b$  ha **una ed una sola soluzione se e solo se  $A$  non è singolare** ( $\det A \neq 0$ )

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

Dove **il Calcolo di  $A^{-1}$  complessità computazionale  $O(n^3)$**

A questo proposito ci vengono in aiuto diversi metodi per il calcolo:

- **Metodi Diretti:** la soluzione viene calcolata in un numero finito di passi modificando la matrice del problema in modo da rendere più agevole il calcolo della soluzione.

Es.

- Matrici triangolari: Metodi di Sostituzione
- Fattorizzazione LU
- fattorizzazione di Cholesky,

- **Metodi Iterativi:** Calcolo di una soluzione come limite di una successione di approssimazioni  $x_k$ , senza modificare la struttura della matrice  $A$ . Adatti per sistemi di grandi dimensioni con matrici sparse (pochi elementi non nulli)

## 4.1 Metodi diretti

Iniziamo con la definizione di matrici triangolari:

Una matrice  $M \in \mathbb{R}^{n \times n}$  è detta **triangolare inferiore** se:

$$L = \begin{pmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{pmatrix}$$

Dove  $l_{ij}$  per  $i < j$ , ossia tutti gli elementi sopra la diagonale (con  $j > i$ ) sono nulli

Da cui discende la definizione di **sistema triangolare inferiore**:

Un **sistema triangolare inferiore** è un sistema lineare di equazioni in cui la matrice dei coefficienti è una matrice triangolare inferiore, e si presenta nella seguente forma:

$$\begin{pmatrix} a_{1,1} & 0 & 0 & \dots & 0 \\ a_{2,1} & l_{2,2} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \dots & a_{n-1,n-1} & 0 \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

e in un sistema lineare:

$$\begin{cases} a_{1,1}x_1 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 = b_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 = b_3 \\ \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \dots + a_{n,n}x_n = b_n \end{cases}$$

Adesso c'è contrapposto con il **sistema triangolare superiore**

Una matrice  $M \in \mathbb{R}^{n \times n}$  è detta triangolare superiore se:

$$L = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ 0 & 0 & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n,n} \end{pmatrix}$$

Dove  $u_{ij} = 0$  per  $i > j$ , ossia tutti gli elementi sotto la diagonale (con  $i > j$ ) sono nulli.

Da cui discende la definizione di **sistema triangolare superiore**

Un **sistema triangolare superiore** è un sistema lineare di equazioni in cui la matrice dei coefficienti è una matrice triangolare superiore, e si presenta nella seguente forma:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ 0 & 0 & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

e in un sistema lineare:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,2}x_2 + a_{2,3}x_3 + \cdots + a_{2,n}x_n = b_2 \\ a_{3,3}x_3 + \cdots + a_{3,n}x_n = b_3 \\ \vdots \\ a_{n,n}x_n = b_n \end{cases}$$

#### 4.1.1 algoritmo delle sostituzioni

Quando si ha a che fare con matrici triangolari la soluzione può essere facilmente calcolata attraverso l'**algoritmo delle sostituzioni**, che si differenzia nel caso di matrici triangolari inferiori o superiori

- Nel caso di matrici triangolari inferiori l'algoritmo prende il nome di **sos-**

**tituzioni all'indietro** ed è descritto dal sistema qui sotto:

$$\begin{cases} x_n = \frac{b_n}{a_{nn}} \\ x_i = \frac{b_i - \sum_{k=i+1}^n a_{ik} x_k}{a_{ii}} \end{cases} \quad i = n-1, n-2, \dots, 1$$

- Nel caso di matrici triangolari superiori l'algoritmo prende il nome di **sostituzioni in avanti** ed è descritto dal sistema qui sotto:

$$\begin{cases} x_1 = \frac{b_1}{a_{11}} \\ x_i = \frac{b_i - \sum_{k=1}^{i-1} a_{ik} x_k}{a_{ii}} \end{cases} \quad i = 2, 3, \dots, n$$

In entrambi i casi il numero di operazioni richiesto è  $\simeq \frac{n(n+1)}{2} = O(n^2)$

#### 4.1.2 Fattorizzazione LU

Considerata una matrice  $A \in \mathbb{R}^{n \times n}$ , una fattorizzazione  $LU$  fattorizza la matrice  $A$  nella forma

$$A = LU$$

con

- $L$  matrice  $n \times n$  triangolare inferiore
- $U$  matrice  $n \times n$  triangolare superiore

Per trasformare una matrice  $A$  in una matrice  $LU$ , è possibile utilizzare l'algoritmo di Gauss per ridurla a una matrice triangolare superiore  $U$  ed occorrerà inserire le operazioni di eliminazione nella matrice  $L$  (inizializzata a una matrice identità). È possibile dimostrare che  $A = L \times U$

Ecco l'algoritmo in pseudocodice, assumendo che la matrice  $A$  sia ben condizionata e pertanto non si necessita uno scambio di righe:

---

**Algorithm 3:** Fattorizzazione LU

---

**Input:** Intero  $n$ , Matrice  $A[1..n, 1..n]$   
**Output:** Array  $B[1,2]$  contenente le matrici  $L$  e  $U$

```
1 Let  $B$  be a new Array[2];
2 Let  $L$  be a new Matrix[1..n, 1..n];
3 Let  $U$  be a new Matrix[1..n, 1..n];
4 for  $i = 1$  to  $n$  do
5   for  $j = 1$  to  $n$  do
6     if  $i == j$  then
7        $L[i, j] \leftarrow 1$ ;
8     else
9        $L[i, j] \leftarrow 0$ ;
10 Let  $p$  be a new Real; // dichiaro il pivot
    // colonne per pivot
11 for  $i = 1$  to  $n$  do
12    $p \leftarrow A[i, i]$ ; // aggiorno il pivot
    // righe
13   for  $j = i + 1$  to  $n$  do
14      $L[j, i] \leftarrow \frac{A[j, i]}{p}$ ; // fattore moltiplicativo in L
15   for  $k = i$  to  $n$  do
16      $A[j, k] \leftarrow A[j, k] - L[j, i] \times A[i, k]$ ;
17  $B[1] \leftarrow L$ ;
18  $B[2] \leftarrow A$ ;
19 return  $B$ ;
```

---

Ecco un esempio pratico:

Supponiamo di voler fattorizzare la matrice:

$$A = \begin{pmatrix} 2 & 3 \\ 4 & 7 \end{pmatrix}$$

Poi inizializzo  $L$  come una matrice identità e  $U = A$

$$L = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 2 & 3 \\ 4 & 7 \end{pmatrix}$$

Poi eseguo l'algoritmo di Gauss in  $U$  avendo come pivot 2, in questo caso occorre solo modificare solo la seconda in quanto la matrice ha solo 2 righe. Il fattore moltiplicativo nella riga 2 e colonna 1 è  $l_{2,1} = \frac{4}{2} = 2$ , si può procedere così con l'eliminazione Gaussiana ottenendo:

$$U = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$$



Si inserisca, inoltre,  $l_{2,1}$  nella matrice  $L$ :

$$L = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$$

La fattorizzazione, così, è finita e si ha  $A = L \times U$ :

$$\begin{pmatrix} 2 & 3 \\ 4 & 7 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$$

#### 4.1.3 Eliminazione di Gauss

Si eliminano le incognite in modo sistematico per trasformare il sistema lineare in uno equivalente con matrice a struttura triangolare superiore, più precisamente si introduce una successione di matrici tale che:

$$A^{(k)} = (a_{ij}^{(k)}), \quad k = 1, \dots, n$$

Con  $A^{(1)} = A$  e  $A^{(n)} = U$  e dove la matrice  $A^{(k+1)}$  ha tutti gli elementi  $\{a_{ij}^{(k+1)}, j+1 \leq i \leq n, 1 \leq j \leq k\}$  nulli. Ovvero essa differisce da  $A^{(k)}$  per avere **gli elementi sottodiagonali della colonna k-esima nulli**. Risparmio ulteriori spiegazioni di come funziona e passo direttamente alla formula:

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \\ a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}$$

Per portare a termine

Inoltre è possibile che bisogna scambiare le righe, ad esempio quando  $\det(A_k) = 0$  e pertanto il  $a_{kk} = 0$ , Perciò occorre scambiarli con altre righe (ad esempio la riga  $p$ ). Si noti che lo scambio di righe  $i, j$  di una matrice  $A$  equivale a moltiplicarla per una matrice  $P^{ij}$  tale che:

$$(P^{ij})_{km} = \begin{cases} \delta_{km} & k \neq i, j \\ 0 & k = m = i, j \\ 1 & k = i, m = j \\ 1 & k = j, m = i \end{cases}$$

La quale verrà chiamata **matrice di permutazione**

#### 4.1.4 algoritmo di gauss rivisto come metodo di fattorizzazione LU

Si può dimostrare che l'algoritmo di Gauss realizza la seguente fattorizzazione della matrice  $A$ :

$$A = LU$$

Dove:

- $U = A^{(n)}$  è una matrice triangolare superiore
- $L$  è la matrice triangolare inferiore dei moltiplicatori, ovvero:

$$L = \begin{cases} L_{ij} = m_{ij} & i < j \\ L_{ij} = 1 & i = j \\ L_{ij} = 0 & j < i \end{cases}$$

#### dimostrazione

Sia  $M_k = I - m_k e_k^T$ , dove:

- $(e_k)_j = \delta_{kj}$  quindi  $e_k^T = [0, 0, \dots, 1, \dots, 0]$  Dove 1 è alla posizione

- $m_k = \begin{pmatrix} 0 \\ \dots \\ m_{k+1,k} \\ m_{k+2,k} \\ \dots \\ m_{nk} \end{pmatrix}$

Che equivale in termini di componenti è uguale a:

$$(M_k)_{ip} = \delta_{ip} - (m_k e_k^T)_{ip} = \delta_{ip} - m_{ik} \delta_{kp}$$

Dalle formule dell'algoritmo di gauss si ha:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ij}^{(k)} a_{ij}^{(k)} = a_{ij}^{(k)} - m_{ik} \delta_{kk} a_{kj}^{(k)} = \sum_p (\delta_{ip} - m_{ik} \delta_{kp}) a_{pj}^{(k)}$$