Calcolo Numerico
Appunti

Alex Bastianini

Contents

Chapter 1

I numeri finiti

1.1 Base-N and Binary

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

Example 1.1.1

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:

Theorem 1.1.1

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 alle base bisogna dividerlo in potenze di due

Example 1.1.2

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

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

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

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

1.1.1 Addizione e moltiplicazione in binario

1.1.2 Conversione fra basi

1.1.3 Floating point

Data la finitezza della memoria di un calcolatore, e' impossibile rappresentare esattamente un numero che ha precisione infinita. Per riuscire a rappresentare in modo piu' efficente possibile si usano i **floating point** numbers:

Definition 1.1.1: Floating Point

Alloca:

- Un indicatore per il segno (s)
- L'esponente (e)
- La mantissa (f)

Nello standard **IEEE754** double precision, che occupa 64 bit, un numero e' rappresentato nel seguente modo:

no puede soportar este sufrimento

In generale, ogni numero reale $n \in \mathbb{R}$ e' definito da:

no puede soportar este sufrimento

Quindi l'insieme dei numeri floating point caratterizzati dai valori $\beta, t, +L, U$ sono:

no puede soportar este sufrimento

Se un numero $x \in \mathbb{R}$ ha un numero di cifre decimali maggiore di t, allora $x \notin \mathcal{F}$ e deve essere approssimato (solitamente col troncamento).

1.2 Errori di rappresentazione

Dato che stiamo usando lo stesso numero di bit, il numero di valori rappresentabili rimane uguale, ma la codifica *floating point* fa in modo che il **gap**, ovvero la differenza, fra due numeri successivi (che esistono sempre dato che lavoriamo con valori discreti) sia relativo al valore assoluto dei numeri rappresentati. Questo e' utile dato che riusciamo ad essere molto precisi con numeri piccoli, riuscendo comunque a rappresentare valori enormi con un errore che in confronto rimane minuscolo.

no puede soportar este sufrimento

Abbiamo due modi per indicare l'accuratezza con la quale possiamo rappresentare un valore $x \in \mathbb{R}$:

- L'errore **assoluto**, che e' piu' grande piu' e' grande l'ordine di x

$$|fl(x) - x| < beta^{p-t}$$

- E quello **relativo**, che dipende solo da t

$$\frac{|fl(x) - x|}{|x|} < \frac{1}{2}\beta^{1-t} = \textbf{eps}$$

Il valore **eps** e' detto **precisione macchina**, ed e' il numero macchina piu' piccolo positivo tale che

$$fl(1 + \text{eps}) > 1$$

Quindi, definite due operazioni generiche, una per i reali e l'altra per i floating-point:

- $\cdot : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- $\circ : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$

$$x \circ y = fl(x \cdot y)$$

Abbiamo che ogni operazione fra floating-point genera un errore (di **arrotondamento**):

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

Nel caso di somma di numeri di grandezza molto diversa, le cifre piu' significative del numero piu' piccolo possono essere perse. Per una sola operazione questo non causa un errore troppo grande, ma se l'operazione viene eseguita molte volte l'errore si **amplifica** e puo' diventare catastrofico.

Chapter 2

Risoluzione di equazioni

2.1 Calcolo dello zero di una funzione

Come primo metodo per risolvere (approssimativamente) equazioni, usiamo il metodo della **bisezione**. Questo sfrutta il teorema degli zeri per trovare uno dei possibili zeri di una funzione continua su un intervallo chiuso con valori discordi di segno agli estremi. Come esempio, prendiamo questa equazione:

$$x = cosx$$

Che possiamo riscrivere come:

$$(f(x) = x - cosx), \quad x \in \mathbb{R}.f(x) = 0$$

Dato l'intervallo $[-1, 1]$, otteniamo il seguente grafico:

no puede soportar este sufrimento

2.1.1 Algoritmo di bisezione

L'algoritmo di bisezione usa un approccio simile alla ricerca binaria per avvicinarsi sempre di piu' al valore dello zero. Dato in input la funzione continua $f : \mathbb{R} \to \mathbb{R}$ e un intervallo $[a, b].f(a)f(b) < 0$, viene calcolato il valore della funzione nel punto di mezzo c. L'intervallo viene poi ristretto, prendendo c come uno dei limiti e scegliendo l'altro fra a e b in modo che i valori della funzione agli estremi abbiano sempre segno opposto. In pseudocodice:

Algorithm 1: Bisezione semplice

Input: $f : \mathbb{R} \to \mathbb{R}$ continua su $[a, b] \subset \mathbb{R}$, tale che $f(a)f(b) < 0$. $N > 0$ (numero di iterazioni)
Output: Valore approssimato di uno zero di f su $[a, b]$

```
1  c ← 0;
2  for N times do
3  |  c ← (a+b)/2;
4  |  if f(a)f(c) < 0 then
5  |  |  b ← c
6  |  else
7  |  |  a ← c
8  |  end
9  end
10 return c
```

Se volgiamo assicurarci che il risultato r' che otteniamo abbia al massimo un errore E_{tol} rispetto al valore reale r della radice, basta porre una condizione d'arresto nel ciclo di bisezione. Sapendo che ad ogni ciclo lo zero si trova fra a e b, scegliendo r' come punto di mezzo rende l'errore massimo $E_{max} = \frac{b-a}{2}$, quindi:

Algorithm 2: Bisezione con errore

Input: $f : \mathbb{R} \to \mathbb{R}$ continua su $[a,b] \subset \mathbb{R}$, tale che $f(a)f(b) < 0$. E_{tol}
Output: Valore approssimato di uno zero di f su $[a,b]$ con errore assoluto al massimo E_{tol}

```
c ← 0;
E_max = (b−a)/2;
while E_max > E_tol do
    c ← (a+b)/2;
    if f(a)f(c) < 0 then
        b ← c
    else
        a ← c
    end
    E_max = (b−a)/2
end
return c
```

2.2 Iterazione di punto fisso

Presa la funzione f di cui vogliamo trovare lo zero, possiamo definire una funzione $g(x) = x - w(x)f(x)$ dove w e' una funzione peso *positiva* e *limitata*. Quindi:

$$f(x) = 0 \iff g(x) = x$$

no puede soportar este sufrimento Quindi siamo passati dal cercare uno zero di una funzione a trovare il **punto fisso**:

Definition 2.2.1: Punto fisso

Data $g : D \to C$, si chiamano **punti fissi** tutti i punti $x \in D$:

$$g(x) = x$$

Per trovare il punto fisso, bisogna quasi sempre usare un metodo *iterativo*, ovvero di approssimazioni successive, seguendo questa procedura:

- Iniziare con una prima approssimazione x_0
- Iterare con la seguente formula: $x_{k+1} = g(x_k)$

Proposition 2.2.1

Data una funzione $g : D \to C$ continua e una successione $x_n \in D . x_{k+1} = g(x_k)$ tale che $x_n \xrightarrow{n \to +\infty} p$, allora:

$$g(p) = p$$

Ovvero p e' un **punto fisso** di g.

Dimostrazione: Per proprieta' delle successioni e per ipotesi, sappiamo che $\lim_{k \to +\infty} x_k = p$. Quindi, per continuita' di g:

$$\lim_{k \to +\infty} g(x_k) = g(p)$$

Inoltre, per costruzione della successione x_n, sappiamo che $g(x_k) = x_{k+1}$, quindi:

$$\lim_{k \to +\infty} x_{k+1} = \lim_{k \to +\infty} g(x_k)$$

$$p = g(p)$$

😄

2.2.1 Mappe contrattive

Ma come facciamo ad assicurarci che la successione converga? Ecco che entrano in gioco le **mappe contrattive**.

Definition 2.2.2: Mappa

Una **mappa** e' una funzione $g : D \to D$ dove D e' un intervallo chiuso $[a, b]$.

Proposition 2.2.2

Presa una mappa continua su un intervallo $[a, b]$, allora:

$$\exists p \in [a, b].p \text{ e' un punto fisso}$$

Dimostrazione: Consideriamo la funzione $f(x) = x - g(x)$, dove g e' una mappa continua sull'intervallo $[a, b]$. Quindi $f(a) = a - g(a) \leqslant 0$ e $f(b) = b - g(b) \geqslant 0$, dato che per definizione di mappa $g(a), g(b) \in [a, b]$. Quindi per teorema degli zeri $\exists p \in [a, b].f(p) = 0$. Quindi $g(p) = p$ e' un punto fisso. 😄

no puede soportar este sufrimento

Come si vede dal grafico della mappa continua $g(x)$ su $[-4, 4]$, e' possibile che una mappa abbia piu' di un punto fisso. Per assicurarci l'esistenza di una soluzione unica, dobbiamo aggiugere un'ultima condizione:

Definition 2.2.3: Mappa contrattiva

Una mappa $g : D \to D$ si dice **contrattiva** se esiste $C < 1$ tale che:

$$\forall x, y \in D.|g(x) - g(y)| \leqslant C|x - y|$$

C e' detta costante contrattiva

Questo significa che per ogni due punti che prendiamo dal dominio, la loro distanza sara' sempre maggiore (o uguale) alla distanza fra i due punti corrispondenti nel codominio. Graficamente (in $\mathbb{R}^2$):

no puede soportar este sufrimento

Theorem 2.2.1

Data una mappa contrattiva g su un intervallo chiuso D, esiste un solo punto fisso p a cui converge la successione x_n dove $x_{k+1} = g(x_k)$, scegliendo x_0 come qualunque punto in D.

Dimostrazione (singolo punto fisso): Assumiamo che g abbia piu' di un punto fisso e dimostriamo l'assurdo. Consideriamo il seguente grafico:

no puede soportar este sufrimento

Prendendo due dei punti fissi, p_1, p_2, notiamo che $\frac{|g(p_1)-g(p_2)|}{|p_1-p_2|} = \frac{|p_1-p_2|}{|p_1-p_2|} = 1$, ma dato che la costante C deve essere minore di 1, abbiamo dimostrato l'assurdo. 😄

La seconda parte della dimostrazione la vediamo piu' avanti.

Come controllare se una funzione differenziabile e' una mappa contrattiva

Nella dimostrazione precedente e' apparsa una forma che somigliava molto a una derivata. Infatti, se la funzione g e' derivabile possiamo usare il seguente teorema:

Theorem 2.2.2

Se una funzione $g : [a, b] \to [a, b]$ e' differenziabile e esiste una costante $C < 1$ tale che $\forall x \in [a, b].|g'(x)| < C$, allora g e' una **mappa contrattiva**

Dimostrazione: Per il teorema di Lagrange, $\forall x < y \in [a,b].\exists c \in [x,y].g(y) - g(x) = g'(c)(y-x)$. Quindi, aggiungendo valori assoluti, $|g(y) - g(x)| = |g'(c)| \cdot |y - x| \leqslant C \cdot |y - x|$ per ipotesi. Quindi, per definizione, g e' una mappa contrattiva. 😄

2.2.2 Velocita' del miglioramento dell'approssimazione

Chiamiamo l'errore assoluto $|E_k|$ la differenza assoluta fra x_k e il valore effettivo del punto fisso p.

Proposition 2.2.3

$|E_{k+1}| \leqslant C|E_k|$, quindi

$$|E_k| = C^k|E_0|$$

Dove $E_0 = x_0 - p$.

Quindi per ogni iterazione, l'errore iniziale diminuisce almeno di un fattore C (che ricordo e' sempre < 1 quindi e' una cosa buona). Cio' implica che piu' e' piccolo C, piu' ci avviciniamo in fretta a p.

Dimostrazione: Per definizione di errore $E_{k+1} = x_{k+1} - p$, che possiamo riscrivere come $g(x_k) - g(p)$, quindi per definizione di mappa contrattiva $|E_{k+1}| = |g(x_k) - g(p)| \leqslant C|x_k - p| = C|E_k|$. Se sostituiamo $k = 0$ abbiamo il caso base:

$$|E_1| = C|x_0 - p|$$

Da cui ricorsivamente:

$$|E_k| = C \cdot C \cdot \ldots \cdot C|x_0 - p| = C^k|x_0 - p|$$

😄

Ora siamo in grado di finire la dimostrazione di prima:

Diostrazione (convergenza): Usando la proposizione precedente, $\lim_{k\to+\infty} |E_k| = \lim_{k\to+\infty} C^k|x_0 - p| = 0$. Quindi:

$$x_k \xrightarrow{k\to+\infty} p$$

😄

2.2.3 Metodo di Newton

Vediamo ora il metodo di Newton, che rispetto alla bisezione e' generalmente piu' veloce (ma non abbiamo la garanzia). Puo' essere dimostrato come una specifica mappa contrattiva o come approssimazione a linee tangenti, vediamo la prima:

Prendiamo una funzione differenziabile f e costruiamole una mappa contrattiva g. Per migliorare l'efficenza, vogliamo fare in modo che la costante C abbia valore minimo dato un intorno della radice r di f, come facciamo? Dato che C assume il valore della derivata massima della mappa, vogliamo fare in modo che la derivata di g sia minima per quell'intorno. Un modo per fare cio' e' assicurarci che $g'(r) = 0$, vediamo come:

$$g'(r) = 1 - w'(r)f(r) - w(r)f'(r) = 1 - w(r)f'(r)$$

dato che $f(r) = 0$ per definizione, quindi:

$$w(r) = \frac{1}{f'(r)}$$

Dato che non sappiamo il valore di r, poniamo $w(x) = \frac{1}{f'(x)}$ per ogni x nel dominio. Da notare che $w(x)$ non esiste quando $f'(x) = 0$. Facendo cosi', la formula di iterazione diventa:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

che e' la formula per il metodo di Newton.

2.3 Risolvere sistemi lineari

- Metodi Diretti: +accurati, -efficenti
- Metodi Iterativi: -accurati, +efficenti

2.3.1 Fattorizzazione LU

Per risolvere sistemi lineari, il metodo diretto piu' efficente risulta essere la **fattorizzazione LU**. Purche' abbia un numero maggiore di operazioni floating-point (flops) rispetto al Gaussian-Jordan, permette di calcolare multiple soluzioni di un sistema $Ax = b$ decomponendo una sola volta la matrice A.

Algoritmo

Dobbiamo fattorizzare la matrice A e ottenere due matrici triangolari, una superiore (U) e l'altra inferiore (L). In piu', L deve avere tutti 1 sulla diagonale principale.

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

In modo da essere fattorizzabile, A non deve essere *singolare* ($detA \neq 0$) e deve avere i minori principali diversi da 0. Per fare cio', moltiplichiamo la matrice A per $n-1$ matrici $E_1, E_2, ..., E_{n-1}$ (che per costruzione sono tutte lower) in modo che A diventi triangolare superiore (sara' il valore di U), mentre L sara' l'inversa del prodotto $E_{n-1} \cdot ... \cdot E_2 \cdot E_1$ (dato che $L^{-1}A = U$). Quindi per prop dell'inversa $L = E_1^{-1} \cdot E_2^{-1} \cdot ... \cdot E_{n-1}^{-1}$:

no puede soportar este sufrimento

Dato che $E_1, E_2, ..., E_{n-1}$ sono triangolari con solo 1 sulla diagonale, per trovare l'inversa basta cambiare di segno i loro valori (tranne gli 1 diagonali). Per evitare di dividere per numeri troppo piccoli durante la costruzione delle matrici lower, e' possibile usare una matrice P di cambio di righe:

$$PA = LU \implies A = P^{-1}LU \implies A = PLU$$

($P = P^{-1} = P^T$). Infine ci resta risolvere il sistema:

$$Ax = b \implies LUx = b \implies \begin{cases} Ux = y \\ Ly = b \end{cases}$$

Per cui usiamo il metodo di sostituzione avanti, dato che sono sistemi triangolari:

no puede soportar este sufrimento

2.3.2 Fattorizzazione di Cholesky

Si puo' usare solo nel caso speciale in cui A sia **definita semi-positiva**:

Definition 2.3.1: Matrice semi-definita positiva

A $n \times n$ simmetrica e' **semi-definita positiva** se $\forall x \in \mathbb{R}^n$:

$$x^T Ax \geqslant 0$$

Per cui vale:

Proposition 2.3.1 Autovalori di matrici semi-definite positive

Data una matrice A semi-definita positiva, ogni autovalore di A e' positivo o nullo

Grazie a queste proprieta', possiamo dire che $\exists L$:

$$A = L \cdot L^T$$

e trovarlo e' circa due volte piu' efficente rispetto a LU.

2.4 Approssimazione

2.4.1 SVD

Prima di iniziare ad introdurre metodi di approssimazione, e' necessario esplorare uno dei teoremi finali dell'algebra lineare: la **SVD** (Single Value Decomposition). Questo teorema sara' fondamentale per descrivere vari modelli di ottimizzazione e approssimazione per motivi che vedremo dopo.

Theorem 2.4.1 Single Value Decomposition

Data una qualunque matrice reale A $m \times n$ (per comodita' stabiliamo $m \geqslant n$) di rango k, allora esistono:

- $\mathcal{U}$ ortonormale $m \times m$
- V ortonormale $n \times n$
- Σ rettangolare diagonale $m \times n$

Tali che:

$$A = \mathcal{U}\Sigma V^T$$

Quindi **qualunque** matrice puo' essere espressa come una matrice della stessa dimensione diagonale, moltiplicata per due matrici ortonormali (di cui una trasposta). Proviamo a vedere perche':

: 😄

Chapter 3

Sistemi lineari

3.1 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

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

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 su ha che il sistema $Ax = b$ ha una ed una sola soluzione sse A non è singolare ($detA \neq 0$)

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

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 piú 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)

3.1.1 Metodi diretti

Iniziamo con la definizione di matrici triangolari:

Definition 3.1.1: Matrice triangolare inferiore

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**:

Definition 3.1.2: 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**

Definition 3.1.3: matrice triangolare superiore

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

$$L = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ 0 & 0 & a_{3,3} & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{n,n} \end{pmatrix}$$

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**

Definition 3.1.4: 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:

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

algortimo 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

Theorem 3.1.1

- Nel caso di matrici triangolari inferiori l'algoritmo prende il nome di **sostituzioni all'indietro** ed è descritto dal sistema qui sotto:

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

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

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

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

Eliminazione di Gauss

Si eliminano le incognite in modo sistematico per trasformare i 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 $\{a_{ij}^{(k+1)}, j+1 \leqslant i \leqslant n, 1 \leqslant j \leqslant k\}$ 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:

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

Inoltre è possibile che bisogni 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:

$$(P^{(ik)})_{km} = \begin{cases} \delta_{km} & \text{se } k \neq i \text{ e } k \neq j, \\ 0 & \text{se } k = m \text{ e } (k = i \text{ o } k = j), \\ 1 & \text{se } k = i \text{ e } m = j, \\ 1 & \text{se } k = j \text{ e } m = i. \end{cases}$$

La quale verrà chiamata **matrice di permutazione**

algoritmo di gauss rivisto come mettodo di fattorizzazione LU

Si può dimostrare che l'algortimo 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:

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

Theorem 3.1.2

Sia $A \in \mathbb{R}\mathfrak{n} \times \mathfrak{n}$ una matrice, è possibile fattorizzarla come spiegato poc'anizi

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

- $(e_k)_j = \delta_{kj}$ quindi $e_k^T = [0, 0, \ldots, 1, \ldots, 0]$ Dove 1 è alla posizione
- $m_k = \begin{pmatrix} 0 \\ \cdots \\ m_{k+1,k} \\ m_{k+2,k} \\ \cdots \\ m_{nk} \end{pmatrix}$

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'algortimo di gauss si ha:

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

😄

Example 3.1.1

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 $A = L \times U$:

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