Capitolo 2

Calcolo matriciale: metodi diretti

2.1 Introduzione al calcolo numerico matriciale

L'obiettivo ultimo della Matematica Numerica è quello di fornire strumenti software per la risoluzione effettiva di problemi concreti. Nel processo di risoluzione di un problema, la fase di risoluzione numerica del modello matematico che descrive il problema in esame, gioca un ruolo fondamentale, intendendo con ció la fase che va dalla formulazione del modello numerico fino allo sviluppo di software. I modelli matematici piú semplici ed i piú utilizzati sono di tipo lineare e sono frequentemente rappresentati da sistemi di equazioni lineari. Essi sono di fondamentale importanza in quanto l'analisi e lo studio di problemi concreti, anche estremamente complessi, sono in primo luogo effettuati su un modello lineare semplificato. Tali sistemi derivano direttamente dall'applicazione di leggi che regolano il fenomeno in esame come accade in molti campi della scienza e della tecnica, oppure compaiono nel processo di risoluzione numerica di alcuni classici problemi, quali interpolazione, fitting, ottimizzazione e approssimazione di problemi differenziali. In particolare, è stato stimato che il calcolo della soluzione di un sistema lineare costituisce il nucleo principale del processo di risoluzione di almeno il 70% di tutti i problemi scientifici. Ció motiva la necessità di avere a disposizione metodi, algoritmi e software affidabili ed efficienti per la risoluzione di tali problemi.

2.2 Introduzione ai sistemi di equazioni lineari

♣ **Esempio 2.1.** Assegnate due rette del piano, rispettivamente di equazione:

$$4x - y = -10;$$
$$2x - y = -20,$$

si vogliono determinare le coordinate del punto di intersezione. Se P è un tale punto, allora le coordinate (x, y) di P devono soddisfare entrambe le equazioni ovvero **costituiscono una soluzione del sistema di equazioni lineari formato dalle equazioni delle due rette**. Il sistema coinvolge **due** equazioni in **due** incognite e viene perció detto **sistema di ordine** $n = 2$.

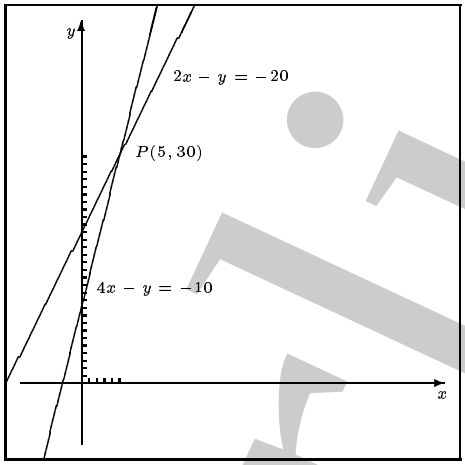

Figura 2.1: Sistema lineare di ordine $n = 2$: interpretazione geometrica.

È chiaro che solo alcune coppie di valori (x, y) soddisfano la prima equazione (ad es. la coppia (2,18) soddisfa la prima equazione mentre la coppia (3,20) non la soddisfa) e che soltanto alcune coppie (x, y) soddisfano la seconda equazione. Il problema che ci si pone è determinare se esiste almeno una coppia di valori (x, y) che soddisfi ambedue le equazioni e, quindi, il sistema di due equazioni in due incognite formato dalle equazioni delle due rette.

Tale sistema puó essere risolto sottraendo la seconda equazione dalla prima, ottenendo

$$2x = 10$$

da cui si ricava

$$x = 5;$$

e sostituendo il valore ottenuto per x nella prima equazione, cioè:

$$20 - y = -10$$

da cui si ricava

$$y = 30.$$

Cosí la coppia di valori (5,30) soddisfa il sistema di equazioni e quindi il punto P del piano di coordinate (5,30) è punto di intersezione delle due rette, come mostrato in Figura 2.1, rappresentando nel piano (x, y) le due rette assegnate. ♣

La Figura 2.1 mostra il significato geometrico di un sistema di equazioni lineari di ordine $n = 2$. Molti problemi, appartenenti ai più svariati settori applicativi e quindi in apparenza completamente differenti, sono descritti in modo naturale da un modello matematico basato su un sistema di equazioni lineari.

♣ **Esempio 2.2.** Si considerino 2 industrie con produzioni diverse, ma con la condizione che ciascuna debba disporre di una certa quantità del prodotto dell'altra. Ad esempio un'industria automobilistica

compra pneumatici da un'industria di pneumatici, mentre l'industria di pneumatici acquista camion dall'industria automobilistica per il trasporto del suo prodotto.

Si supponga che l'industria automobilistica per produrre un camion necessiti di 5 pneumatici, mentre l'industria di pneumatici abbia bisogno di 1 camion per ogni 50 pneumatici prodotti. Indicate rispettivamente con x e y la quantità di camion e la quantità di pneumatici prodotte nelle 2 industrie al termine del ciclo di produzione, si ha che per soddisfare almeno le esigenze di entrambe le industrie, deve verificarsi

$$\begin{array}{rcl} x & \geq & \frac{1}{50}y \\ y & \geq & 5x \end{array}$$

È chiaro che le due industrie devono soddisfare anche le esigenze dei consumatori, includendo, tra questi, eventuali altre industrie.

Supponendo che siano richiesti, dai consumatori, 3000 camion e 30000 pneumatici, si ha che la quantità totale di camion da produrre è:

$$x = \frac{1}{50}y + 3000$$

e la quantità totale di pneumatici da realizzare è:

$$y = 5x + 30000$$

Quindi, per determinare i valori x e y in modo che siano soddisfatte le esigenze dei consumatori e dei processi di produzione, è necessario risolvere il seguente sistema di equazioni lineari:

$$\begin{cases} x - \frac{1}{50}y = 3000 \\ -5x + y = 30000 \end{cases}$$

Moltiplicando per 50 la prima equazione si ottiene il sistema:

$$\begin{cases} 50x - y = 150000 \\ -5x + y = 30000 \end{cases}$$

Addizionando le due equazioni si ha:

$$45x = 180000$$

da cui si ricava:

$$x = 180000/45 = 4000;$$

sostituendo il valore ottenuto per x nella seconda equazione si ha:

$$-5 \times 4000 + y = 30000$$

da cui si ottiene:

$$y = 50000.$$

♣

♣ **Esempio 2.3.** La determinazione di un possibile schema di reazione chimica è il punto di partenza della maggior parte degli studi cinetici. Piú in dettaglio il problema è il seguente: in un esperimento sono identificate alcune specie di molecole (intendendo per specie di molecola *una* molecola di un particolare composto); si vuole determinare quali possono essere le molecole reagenti e quali quelle prodotte e le relative quantità, tenendo presente che il vincolo imposto nella reazione chimica è costituito dalla *Legge di conservazione della massa* (Legge di Lavoisier).

L'applicazione di tale legge conduce ad un sistema di equazioni lineari. In tale sistema ciascun coefficiente rappresenta il numero di atomi di una certa sostanza presente in una delle molecole identificate ($a_{i,j}$ indica quanti atomi della sostanza i sono presenti nella molecola j) e ciascuna incognita rappresenta un coefficiente (detto *stechiometrico*) che sarà il coefficiente associato ad una molecola nell'equazione finale (da ricavare) che descrive la trasformazione totale. In particolare il valore assoluto del coefficiente stechiometrico fornisce il numero di molecole di un certo composto che entra nella reazione finale ed il segno indica se la molecola associata ha il ruolo di reagente (segno -) o di prodotto (segno +).

Supponendo che in un esperimento campione siano state identificate le seguenti specie di molecole:

$$CH_2Cl_2,\ CH_4,\ CH_3Cl$$

si vuole determinare quali di queste molecole possono essere reagenti e quali prodotto e in che quantità ciascuna molecola interviene nella reazione (ovvero qual'è una possibile equazione chimica del tipo: $\sigma_1 CH_2Cl_2 + \sigma_2 CH_4 + \sigma_3 CH_3Cl = 0$).
I coefficienti del sistema lineare sono riportati nella seguente tabella:

	molecole		
atomi	CH_2Cl_2	CH_4	CH_3Cl
C	1	1	1
H	2	4	3
Cl	2	0	1

Le incognite da determinare sono quindi, in numero di 3 (essendo 3 le molecole); σ_1 per CH_2Cl_2, σ_2 per CH_4, σ_3 per CH_3Cl. Il sistema si scrive tenendo presente la *Legge di Lavoisier* e quindi

$$\begin{cases} \sigma_1 & +\sigma_2 & +\sigma_3 & = 0 \\ 2\sigma_1 & +4\sigma_2 & +3\sigma_3 & = 0 \\ 2\sigma_1 & & +\sigma_3 & = 0 \end{cases}$$

In questo caso si è in presenza di 3 equazioni (quanti sono gli atomi) in 3 incognite. Supponendo, ad esempio, che $\sigma_1 = -1$ che equivale a sapere che nella reazione CH_2Cl_2 è una molecola reagente, si ottengono i seguenti valori per le rimanenti incognite: $\sigma_2 = -1$, $\sigma_3 = 2$. Pertanto la reazione finale, nell'ipotesi considerata, è:

$$CH_2Cl_2 + CH_4 = 2CH_3Cl$$

♣

♣ **Esempio 2.4.** Un circuito elettrico è una combinazione di conduttori progettato per il trasporto di energia elettrica o la trasmissione di segnali elettrici. In un circuito RLC, ohmico, induttivo e capacitivo (resistenza R, induttanza L, capacità C collegate in serie e attraversate da corrente alternata) trascorsa una fase transitoria, si rileva il passaggio di una corrente di tipo sinusoidale avente la stessa frequenza della forza elettromotrice (f.e.m).

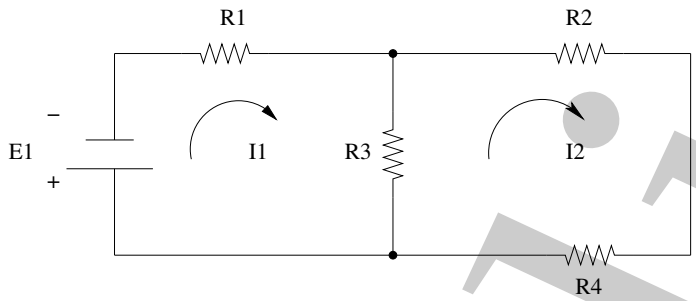

Figura 2.2: Primo circuito elettrico.

Si consideri, ad esempio, il circuito riportato in Figura 2.2 in cui sono presenti 4 resistenze (R_1, R_2, R_3, R_4) ed un generatore di f.e.m. (E_1). Si supponga che la f.e.m. sia costante e che siano state raggiunte condizioni di stazionarietà nel circuito cosicché le correnti (I_1, I_2) siano anch' esse costanti. Generalmente il problema consiste nel trovare le correnti in funzione delle f.e.m. e delle resistenze. Per risolvere questo problema si applicano le *Leggi di conservazione della carica elettrica e dell'energia*[1].
L'applicazione di tali leggi richiede di chiarire alcune convenzioni. Si considerino

- positive le correnti che si allontanano dal nodo, negative quelle dirette verso il nodo;
- positiva una caduta di potenziale attraverso una resistenza se ci si muove nello stesso verso della corrente, negativa se ci si muove nel verso opposto;
- positiva la caduta di potenziale se si passa attraverso una sorgente di f.e.m., nella direzione in cui agisce la sorgente (aumento di potenziale), negativa se si passa nella direzione opposta (diminuzione di potenziale).

L'applicazione delle *Leggi di Kirchhoff* al circuito in Figura 2.2 permette di determinare le correnti I_1 e I_2. In particolare la *seconda Legge* applicata alle due maglie in cui circolano le correnti, entrambe orientate in senso orario, fornisce il sistema:

$$\begin{cases} -E_1 = R_1 \cdot I_1 + R_3 \cdot I_1 - R_3 \cdot I_2 \\ 0 = R_3 \cdot I_2 + R_2 \cdot I_2 + R_4 \cdot I_2 - R_3 \cdot I_1 \end{cases}$$

da cui è possibile ricavare le due correnti in funzione delle resistenze e della f.e.m. presenti nel circuito:

$$\begin{cases} I_1 = \frac{-E_1 \cdot R_3}{R_1 \cdot (R_2+R_3+R_4)+R_3 \cdot (R_2+R_4)} \\ I_2 = \frac{-E_1 \cdot (R_2+R_3+R_4)}{R_1 \cdot (R_2+R_3+R_4)+R_3 \cdot (R_2+R_4)} \end{cases}$$

Nel caso in cui le correnti siano negative si deduce che il loro verso effettivo è opposto a quello da noi scelto.

[1] Si tratta delle due *Leggi di Kirchhoff* secondo cui:

1. la somma di tutte le correnti in un nodo di un circuito vale zero;
2. la somma di tutte le differenze di potenziale lungo un qualsiasi percorso chiuso nel circuito vale zero.

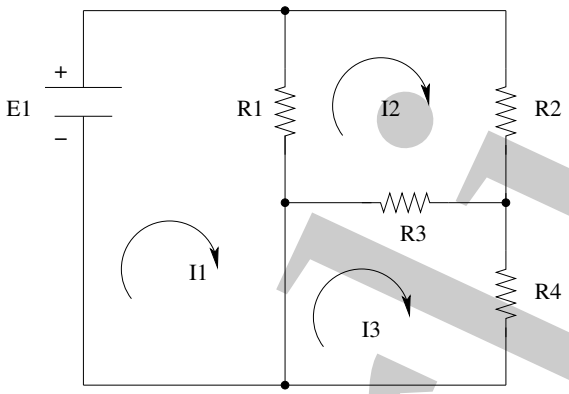

Figura 2.3: Secondo circuito elettrico.

Si consideri, ora, il circuito riportato in Figura 2.3.
L'applicazione delle *Leggi di Kirchhoff* permette di determinare le tre correnti presenti nel circuito: I_1, I_2, I_3; anche in questo esempio supponiamo che siano tutte orientate in senso orario. In particolare, la *seconda Legge* applicata alle tre maglie in cui rimane suddiviso il circuito conduce al sistema:

$$\begin{cases} E_1 = R_1 \cdot I_1 - R_1 \cdot I_2 \\ 0 = R_1 \cdot I_2 + R_2 \cdot I_2 + R_3 \cdot I_2 - R_3 \cdot I_3 - R_1 \cdot I_1 \\ 0 = R_3 \cdot I_3 + R_4 \cdot I_3 - R_3 \cdot I_2 \end{cases}$$

Dalla prima equazione si ricava:

$$I_1 = \frac{E_1}{R_1} + I_2$$

mentre dalla terza si ricava:

$$I_3 = \frac{R_3}{R_4 + R_3} \cdot I_2$$

Sostituiendo tali espressioni nella seconda equazione del sistema, si ricava:

$$I_2 = \frac{(R_3 + R_4) \cdot E_1}{R_2 \cdot R_3 + R_2 \cdot R_4 + R_3 \cdot R_4}$$

I valori di I_1 e di I_3 si ricavano, quindi, dalla loro espressione in funzione di I_2.

♣

Gli esempi ora descritti mostrano che problemi di natura completamente diversa possono essere descritti in modo naturale da sistemi lineari e la loro risoluzione consiste nella risoluzione di sistemi di equazioni lineari.

2.3 Introduzione alla risoluzione numerica di sistemi lineari

♣ **Esempio 2.5.** Si voglia risolvere il sistema di equazioni lineari:

$$\begin{cases} 2x & + & 5y & = 31 \\ 3x & + & 2y & = 19 \end{cases}$$

che puó essere indicato nella forma compatta:

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

con

$$A = \begin{pmatrix} 2 & 5 \\ 3 & 2 \end{pmatrix}; \quad \mathbf{z} = \begin{pmatrix} x \\ y \end{pmatrix}; \quad \mathbf{b} = \begin{pmatrix} 31 \\ 19 \end{pmatrix}.$$

Una possibilità per la risoluzione del sistema assegnato è l'utilizzo della regola di Cramer. Tale metodo, applicato al sistema in esame, consiste nei passi seguenti:

1. calcolo del determinante di A:

$$det(A) = \begin{vmatrix} 2 & 5 \\ 3 & 2 \end{vmatrix} = 2 \times 2 - 5 \times 3 = -11$$

2. calcolo di x e y:

$$x = \frac{\begin{vmatrix} 31 & 5 \\ 19 & 2 \end{vmatrix}}{-11} = \frac{31 \times 2 - 5 \times 19}{-11} = 3;$$

$$y = \frac{\begin{vmatrix} 2 & 31 \\ 3 & 19 \end{vmatrix}}{-11} = \frac{2 \times 19 - 31 \times 3}{-11} = 5$$

Le operazioni aritmetiche effettuate sono:

- passo 1 2 moltiplicazioni e 1 sottrazione;
- passo 2 4 moltiplicazioni, 2 divisioni e 2 sottrazioni.

Quindi, il calcolo della soluzione di un sistema 2×2 mediante il metodo di Cramer richiede, in totale

$$11 \ flop.$$

avendo indicato con *flop* una delle quattro operazioni aritmetiche floating point[2]. ♣

[2]Invero, in generale, nello studio del condizionamento di un problema di algebra lineare, si assume la seguente

Definizione 2.1. (Flop)

♣ **Esempio 2.6.** Si consideri il sistema lineare seguente:

$$\begin{cases} 10x_1 + x_2 - 5x_3 = 1 \\ -20x_1 + 3x_2 + 20x_3 = 2 \\ 5x_1 + 3x_2 + 5x_3 = 6 \end{cases}$$

La risoluzione di tale sistema mediante il metodo di Cramer consiste dei passi seguenti:

1. calcolo del determinanate della matrice del sistema:

$$\begin{vmatrix} 10 & 1 & -5 \\ -20 & 3 & 20 \\ 5 & 3 & 5 \end{vmatrix} = 10 \times \begin{vmatrix} 3 & 20 \\ 3 & 5 \end{vmatrix} + 20 \times \begin{vmatrix} 1 & -5 \\ 3 & 5 \end{vmatrix} + 5 \times \begin{vmatrix} 1 & -5 \\ 3 & 20 \end{vmatrix} =$$

$$= 10 \times (3 \times 5 - 20 \times 3) + 20 \times (1 \times 5 + 5 \times 3) + 5 \times (1 \times 20 + 5 \times 3) = 125$$

2. calcolo di x_1, x_2 ed x_3:

$$x_1 = \frac{\begin{vmatrix} 1 & 1 & -5 \\ 2 & 3 & 20 \\ 6 & 3 & 5 \end{vmatrix}}{125} = 1;$$

$$x_2 = \frac{\begin{vmatrix} 10 & 1 & -5 \\ -20 & 2 & 20 \\ 5 & 6 & 5 \end{vmatrix}}{125} = -2;$$

$$x_3 = \frac{\begin{vmatrix} 10 & 1 & 1 \\ -20 & 3 & 2 \\ 5 & 3 & 6 \end{vmatrix}}{125} = 1.4.$$

Si definisce **flop** *(***fl***oating-point* **op***eration) l'operazione in virgola mobile costituita da una moltiplicazione e da un'addizione.*

Il concetto di flop appena definito è oramai ampiamente utilizzato in Analisi Numerica quando si analizza la complessità computazionale degli algoritmi. Ciò è dovuto al fatto che, nei calcoli su vettori e matrici, l'usuale operazione di base è rappresentata da una moltiplicazione seguita da una addizione. Si pensi ad esempio al calcolo del prodotto tra una matrice ed un vettore, cioè al calcolo di $y = Ax$, con x e y vettori di dimensione n ed A matrice quadrata di dimensione n. L'operazione fondamentale per il calcolo della i-ma componente del vettore y è $y_i = a_{ij} \cdot x_j + y_i$, che deve essere eseguita per tutti i valori di j da 1 a n. Tale operazione è costituita, quindi, da una moltiplicazione e da un'addizione (tipicamente indicata come **saxpy**, "scalar" "α" times vector "x", "plus" vector "y").

Le operazioni aritmetiche effettuate sono:

- passo 1 14 *flop* per il calcolo di un determinante di ordine 3;
- passo 2 45 *flop*.

In totale, sono richieste:

$$59\ flop$$

Si osservi che per calcolare la soluzione del sistema 3×3 in esame è stato necessario valutare 4 determinanti di ordine 3, ciascuno dei quali richiede a sua volta la valutazione di 3 determinanti di ordine 2.

♣

La complessità computazionale di un algoritmo dipende, essenzialmente, da alcuni parametri, tra i quali ricordiamo $\mathcal{T}(n)$ e $\mathcal{S}(n)$ che rappresentano, rispettivamente, la complessità di tempo e di spazio dell'algoritmo in funzione di n, dimensione del problema. La complessità di tempo si misura attraverso il numero di operazioni floating point, *flops*, eseguite dall'algoritmo. La complessitá di spazio, invece, dipende dal numero di variabili utilizzate dall'algoritmo. In generale, sia $\mathcal{T}(n)$ sia $\mathcal{S}(n)$ sono espresse come funzioni **asintotiche** di n, ed a tal fine si usa il *simbolo di Landau*, $\mathcal{O}$[3]

[3]Per il simbolo di Landau si assume la seguente definizione

Definizione 2.2. (Simbolo di Landau: $\mathcal{O}$)
Date due funzioni $f(x)$ e $g(x)$, tali che:

$$\lim_{x\to\infty} f(x) = \infty$$

e:

$$\lim_{x\to\infty} g(x) = \infty$$

si pone:

$$f(x) = \mathcal{O}(g(x))$$

se:

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = C \quad (C = cost \neq 0)$$

In altre parole, nel dare l'andamento asintotico di $\mathcal{T}(n)$ e $\mathcal{S}(n)$ si evidenzia il termine dominante in $\mathcal{T}(n)$ e $\mathcal{S}(n)$, al crescere di n.

Nel risolvere un sistema di equazioni lineari di ordine n con il metodo di Cramer è necessario valutare $n+1$ determinanti di ordine n, ciascuno dei quali richiede la somma di n termini ciascuno costruito mediante $1+(n-1)!$ moltiplicazioni.

Quindi, il numero totale di moltiplicazioni richieste per la valutazione degli $n+1$ determinanti di ordine n è dato da:

$$T_{Cramer}(n) = (n+1)n(n-1)! + (n+1)n.$$

Da ciò si ricava che la complessità computazionale del metodo di Cramer applicato ad un sistema di ordine n è:

$$T_{Cramer}(n) = O((n+1)!). \tag{2.1}$$

Nella Tabella 2.1 sono riportati, relativamente ad un sistema lineare di ordine n (con $n = 5$, $n = 10$ e $n = 20$), il numero di operazioni richiesto per la sua risoluzione mediante il metodo di Cramer ed il tempo di calcolo necessario per eseguire tali operazioni su calcolatori aventi velocità operative differenti. In particolare sono presi in esame: un personal computer (Intel Pentium 4 a 2.2 GHz), una workstation (IBM pseries 655 a 1.3 GHz), un mainframe (HP server rx5670) ed un supercomputer (CRAY C90 a 16 processori). Inoltre, si suppone che nell'esecuzione dell'algoritmo la velocità operativa massima di ciascun calcolatore (riportata nella tabella), misurata in milioni di operazioni floating-point al secondo (Mflops), sia effettivamente raggiunta[4].

Si osservi che, considerato un sistema lineare di ordine 20, disponendo di un calcolatore che effettua oltre 15 miliardi di operazioni floating-point al secondo, il tempo di calcolo richiesto per la sua risoluzione mediante il metodo di Cramer è di 106 anni!! La Tabella 2.1 mette in luce quindi la impraticabilità del metodo di Cramer a causa

---

[4]

Definizione 2.3. (Mflops, Gflops, Tflops)
1 Mflops *(1 milion of floating-point operations per second) = 1 milione di operazioni floating-point al secondo,*

è una delle unità di misura utilizzate per indicare la velocità operativa di un calcolatore. Altre unità sono:

1 Gflops = *1 miliardo di operazioni floating-point al secondo;*
1 Tflops = *1000 miliardi di operazioni floating-point al secondo.*

Ad esempio, 100 Mflops era, negli anni Novanta, la velocità operativa tipica di una workstation *IBM RISC/6000 43P modello 240* con processore *166MHz Power PC 640e.*
L'evoluzione ha condotto, inevitabilmente, allo sviluppo di processori con più elevate velocità operative, alcuni dei quali sono elencati in tabella con, in particolare, dettagli relativi al numero di operazioni floating point eseguite al secondo.
(I dati riportati in tabella sono aggiornati agli inizi del 2001 e reperibili dal sito internet

`http://web.maths.unsw.edu.au/~rsw/MATH5315/processors.shtml.`)

n	# op.	Pers. Comp. 4400 Mflops	Workstation 5200 Mflops	Mainframe 4000 Mflops	Supercomp. 15238 Mflops
5	720	1.6×10^{-7} sec.	1.4×10^{-7} sec.	1.8×10^{-7} sec.	4.7×10^{-8} sec.
10	3.99×10^{7}	9.1×10^{-3} sec.	7.7×10^{-3} sec.	9.9×10^{-3} sec.	2.6×10^{-3} sec.
20	5.1×10^{19}	367 anni	310 anni	404 anni	106 anni

Tabella 2.1: Tempo richiesto per risolvere un sistema di equazioni con il metodo di Cramer su calcolatori con differenti velocità operative.

dell'elevata complessità computazionale[5].
Appare evidente che il metodo di Cramer non costituisce uno strumento effettivo per la risoluzione di un sistema lineare, ovvero non consente di ottenere una risposta al problema in esame, in quanto l'espressione della soluzione fornita dal metodo non può essere valutata nella pratica. Di conseguenza anche la valutazione del determinante della matrice dei coefficienti per stabilire l'unicità della soluzione non è applicabile nella pratica.

In conclusione, la regola di Cramer fornisce una rappresentazione della soluzione e

Processore	**Marca**	**Clock** (MHz)	**Peak** (Mflops)
Alpha 21264b	Compaq	833	1666
UltraSPARC III	SUN	900	1800
MIPS R12000	SGI	400	800
PA-8700	HP	750	3000
Power3-II	IBM	450	1800
Pentium IV	Intel	1700	1700
Pentium III	Intel	1000	1000
Athalon	AMD	1330	1330
PowerPC G4	Apple	733	2932

Nella tabella si è indicato, in particolare, con **peak**, il *peak performance* (picco di prestazione) di un processore, ovvero il numero massimo operazioni floating point al secondo, che il processore può raggiungere in circostanze "ideali", mentre **clock** indica la *frequenza di clock*.
Nuovi processori sono, costantemente, in evoluzione; per aggiornamenti si può fare riferimento, ad esempio, ai siti:

`http://www.aceshardware.com`

`http://www.a1-electronics.co.uk/index.html`

`http://www.chipanalyst.com/`

[5]In particolare si parla di complessità non polinomiale (NP)

non uno strumento effettivo di calcolo! Ció che invece si sta cercando è un metodo *"costruttivo"*, ovvero che consenta di risolvere in maniera effettiva un sistema lineare, intendendo con questo che:

- si possa fare uso, nella sua applicazione, degli strumenti di calcolo disponibili (calcoli a mano, calcolatrici, calcolatori, ...);
- abbia una complessità di tempo *"accettabile"*, ovvero coerente con lo strumento di calcolo utilizzato ed adeguata sia al problema che si sta risolvendo sia allo scopo per cui si sta risolvendo il problema;
- richieda una complessità di spazio *"ammissibile"*, sempre in relazione allo strumento di calcolo.

Per arrivare a determinare un tale metodo, si può seguire un ragionamento che parte dalla individuazione di metodi per la risoluzione di sistemi di equazioni lineari *"semplici"*.

2.4 Metodi di back e forward substitution

Quali sistemi sono "semplici da risolvere"?

♣ **Esempio 2.7.** Si vuole risolvere il sistema:

$$\begin{cases} 7x_1 & & & = 3 \\ & 6.5x_2 & & = 2 \\ & & -8x_3 & = 1.4 \end{cases}$$

In questo caso la forma del sistema suggerisce in modo evidente il metodo di risoluzione che consiste nel:

1. calcolo di x_1 dalla I equazione
$$x_1 = \frac{3}{7};$$
2. calcolo di x_2 dalla II equazione
$$x_2 = \frac{2}{6.5} = \frac{4}{13};$$
3. calcolo di x_3 dalla III equazione
$$x_3 = \frac{1.4}{-8} = -\frac{7}{40}.$$

♣

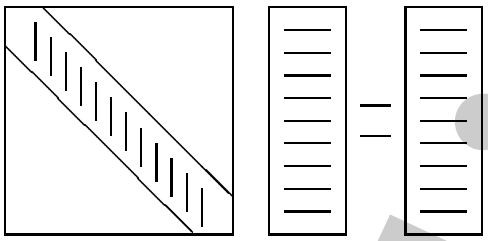

Figura 2.4: Sistema diagonale

Definizione 2.4. (Sistema Diagonale)
Un sistema lineare del tipo:

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

in cui $a_{i,j} = 0$, se $i \neq j$ e con $i, j = 1, \ldots, n$, definisce un sistema diagonale.

La soluzione di un sistema diagonale è data da:

$$x_i = b_i/a_{i,i}, \quad i = 1, \ldots, n$$

se $a_{i,i} \neq 0$ per $i = 1, \ldots, n$. Il numero di operazioni richieste per il calcolo della soluzione è, evidentemente,

$$T_{diag}(n) = n \; flop$$

ovvero:

$$T_{diag}(n) = O(n)$$

Un sistema diagonale può essere rappresentato graficamente nel modo rappresentato in Figura 2.4.

Si osservi che una matrice avente la struttura della matrice dei coefficienti di un sistema diagonale, è detta *matrice diagonale*. In generale, la struttura di una matrice diagonale è la seguente:

$$\begin{pmatrix} a_{1,1} & 0 & \ldots & \ldots & 0 \\ 0 & a_{2,2} & \ldots & \ldots & 0 \\ 0 & \ldots & a_{3,3} & \ldots & 0 \\ \vdots & \ldots & \ldots & \ddots & \vdots \\ 0 & \ldots & \ldots & \ldots & a_{n,n} \end{pmatrix}.$$

Ad esempio la matrice:

$$A = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & -7.4 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \qquad A = diag(5, 2, -7.4, 1)$$

è una matrice diagonale.

Vi sono altri sistemi *"semplici"* da risolvere.

♣ **Esempio 2.8.** Si voglia risolvere il sistema di equazioni lineari:

$$\begin{cases} 2x_1 + 2x_2 + 4x_3 = 5 \\ 7x_2 + 11x_3 = 8 \\ 2x_3 = 2 \end{cases}$$

Anche in questo caso la forma del sistema suggerisce direttamente un metodo di risoluzione che consiste nei passi seguenti:

1. calcolo di x_3 dalla III equazione
$$x_3 = 2/2 = 1;$$
2. calcolo di x_2 dalla II equazione (utilizzando per x_3 il valore calcolato al passo 1)
$$x_2 = \frac{8 - 11x_3}{7} = \frac{8 - 11 \times 1}{7} = \frac{-3}{7};$$
3. calcolo di x_1 dalla prima equazione (utilizzando per x_3 e x_2 i valori calcolati ai passi precedenti)
$$x_1 = \frac{5 - 2x_2 - 4x_3}{2} = \frac{5 - 2 \times (-3/7) - 4 \times 1}{2} = \frac{13}{14}.$$

Tale metodo è detto *metodo di sostituzione all'indietro* (*back-substitution*) in virtú del modo e dell'ordine in cui le incognite sono calcolate. Il numero di operazioni effettuate per risolvere il sistema in esame è:

- passo 1 $\quad 1 flop$
- passo 2 $\quad 3 flop$
- passo 3 $\quad 5 flop$

e quindi, in totale:

$$9\ flop.$$

In maniera analoga il sistema:

$$\begin{cases} 2x_1 = 4 \\ 3x_1 + 2x_2 = 5 \\ x_1 + 2x_2 - 3x_3 = 1 \end{cases}$$

puó essere risolto applicando il metodo seguente:

1. calcolo di x_1 dalla I equazione

$$x_1 = \frac{4}{2} = 2;$$

2. calcolo di x_2 dalla II equazione (utilizzando per x_1 il valore calcolato al passo 1)

$$x_2 = \frac{5 - 3x_1}{2} = \frac{5 - 3 \times 2}{2} = -\frac{1}{2};$$

3. calcolo di x_3 dalla III equazione (utilizzando per x_1 e x_2 i valori calcolati ai passi precedenti)

$$x_3 = \frac{1 - x_1 - 2x_2}{-3} = \frac{1 - 2 - 2 \times (-1/2)}{-3} = 0.$$

Il metodo precedente è detto *metodo di sostituzione in avanti* (*forward-substitution*). Anche in questo caso il numero totale di operazioni effettuate è:

$$9 \; flop.$$

♣

In generale è possibile dare la seguente:

Definizione 2.5. (Sistema triangolare superiore)
Un sistema lineare del tipo:

$$\begin{cases} u_{1,1}x_1 + u_{1,2}x_2 + u_{1,3}x_3 + \ldots + u_{1,n}x_n = b_1 \\ \qquad\quad u_{2,2}x_2 + u_{2,3}x_3 + \ldots + u_{2,n}x_n = b_2 \\ \qquad\qquad\quad u_{3,3}x_3 + \ldots + u_{3,n}x_n = b_3 \\ \qquad\qquad\qquad\quad \ddots \qquad\quad \vdots \qquad\quad \vdots \\ \qquad\qquad\qquad\qquad\quad u_{n,n}x_n = b_n \end{cases} \tag{2.2}$$

in cui $u_{i,j} = 0$ *se* $i > j$, *con* $i, j = 1, \ldots, n$, *è detto* triangolare superiore.

Un sistema lineare triangolare superiore del tipo (2.2) può essere anche indicato nella forma compatta:

$$Ux = b$$

con:

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \ldots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \ldots & u_{2,n} \\ & & u_{3,3} & \ldots & u_{3,n} \\ & & & \ddots & \vdots \\ & & & & u_{n,n} \end{pmatrix} \tag{2.3}$$

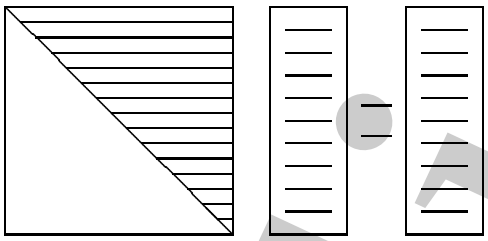

Figura 2.5: Sistema triangolare superiore

Un sistema triangolare superiore è rappresentato graficamente in Figura 2.5.

Una matrice del tipo (2.3), in cui sono nulli tutti gli elementi del triangolo inferiore, ovvero:

$$u_{i,j} = 0 \text{ se } i > j$$

è detta *triangolare superiore*.
Ad esempio la matrice:

$$U = \begin{pmatrix} 1 & 3 & 7 & -4 \\ 0 & 5.3 & 8 & 2.6 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

è triangolare superiore.

La soluzione di un sistema triangolare superiore puó essere ottenuta mediante il metodo di *back-substitution* che calcola:

- dalla n-ma equazione

$$x_n = \frac{b_n}{u_{n,n}};$$

- dalla i-ma equazione, per $i = n-1, n-2, \ldots, 1$

$$x_i = (b_i - u_{i,i+1} \cdot x_{i+1} - u_{i,i+2} \cdot x_{i+2} - \ldots - u_{i,n} \cdot x_n)/u_{i,i} =$$

$$= (b_i - \textstyle\sum_{k=i+1}^{n} u_{i,k} \cdot x_k)/u_{i,i}.$$

È evidente che il metodo di *back-substitution* risulta applicabile se $u_{i,i} \neq 0$, per $i = 1, \ldots, n$.
Il numero di operazioni effettuate è:

$1\ flop$	per il calcolo di x_n
$3\ flop$	per il calcolo di x_{n-1}
$5\ flop$	per il calcolo di x_{n-2}
$2(n-i)+1\ flop$	per il calcolo di $x_i,\ i = n-3, \ldots, 1$

e quindi, sommando tutti i precedenti termini si ha:

$$1 + 3 + 5 + \ldots + 2(n-1) + 1 \; flop = n^2 \; flop$$

da cui possiamo concludere che la complessità computazionale del metodo di *back-substitution*, applicato ad un sistema triangolare superiore, è:

Complessità di tempo $T_{bs}(n) = n^2$

Complessità di spazio $S_{bs}(n) = n^2 + 2n$

o, in termini asintotici:

$$T_{bs}(n) = O\left(n^2\right) \tag{2.4}$$

$$S_{bs}(n) = O\left(n^2\right)$$

Analogamente è possibile dare la seguente:

Definizione 2.6. (Sistema triangolare inferiore)
Un sistema lineare del tipo:

$$\begin{cases} l_{1,1}x_1 & & & & & & & = b_1 \\ l_{2,1}x_1 & + & l_{2,2}x_2 & & & & & = b_2 \\ l_{3,1}x_1 & + & l_{3,2}x_2 & + & l_{3,3}x_3 & & & = b_3 \\ & & & & & \ddots & & \vdots \\ l_{n,1}x_1 & + & l_{n,2}x_2 & + & l_{n,3}x_3 & + \ldots + & l_{n,n}x_n & = b_n \end{cases} \tag{2.5}$$

in cui $l_{i,j} = 0$ *se* $i < j$*, con* $i, j = 1, \ldots, n$*, è detto* triangolare inferiore.

Analogamente, considerato un sistema triangolare inferiore del tipo (2.5), esso può essere scritto nella forma:

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

con:

$$L = \begin{pmatrix} l_{1,1} & & & & \\ l_{2,1} & l_{2,2} & & & \\ l_{3,1} & l_{3,2} & l_{3,3} & & \\ \vdots & \ldots & & \ddots & \\ l_{n,1} & l_{n,2} & l_{n,3} & \ldots & l_{n,n} \end{pmatrix} \tag{2.6}$$

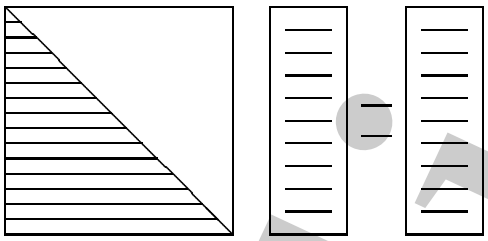

Figura 2.6: Sistema triangolare inferiore

Un sistema triangolare inferiore è rappresentato graficamente in Figura 2.6.

Una matrice del tipo (2.6), in cui sono nulli tutti gli elementi del triangolo superiore, ovvero

$$l_{i,j} = 0 \quad \text{se} \quad i < j$$

è detta *triangolare inferiore*.
Ad esempio la matrice:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 6 & 3.3 & 0 & 0 \\ 2.4 & -3 & 6 & 0 \\ -1 & 3.5 & 2 & 0 \end{pmatrix}$$

è una matrice triangolare inferiore.

La soluzione di un sistema triangolare inferiore puó essere calcolata mediante il metodo di *forward-substitution* che calcola:

- dalla I equazione

$$x_1 = \frac{b_1}{l_{1,1}};$$

- dalla i-ma equazione, per $i = 2, 3, \ldots, n$

$$x_i = (b_i - l_{i,1} \cdot x_1 - l_{i,2} \cdot x_2 - \ldots - l_{i,i-1} \cdot x_{i-1})/l_{i,i} =$$

$$= (b_i - \textstyle\sum_{k=1}^{i-1} l_{i,k} \cdot x_k)/l_{i,i}.$$

Anche in questo caso il metodo è applicabile nell'ipotesi che $l_{i,i} \neq 0$ per $i = 1, 2, \ldots, n$.

Con un procedimento analogo a quello utilizzato per valutare il numero di operazioni aritmetiche richieste dal metodo di *back-substitution*, si ricava che il metodo di *forward-substitution*, applicato ad un sistema triangolare inferiore di ordine n, ha una complessità computazionale:

Complessità di tempo $T_{fs}(n) = n^2$

Complessità di spazio $S_{fs}(n) = n^2 + 2n$

o, in termini asintotici

$$T_{fs}(n) = O\left(n^2\right)$$

$$S_{fs}(n) = O\left(n^2\right)$$

Gli algoritmi di *back-substitution* e *forward-substitution* sono riportati rispettivamente nella Procedura 2.1 e nella Procedura 2.2.

```
procedure backsub (in: u, b, n; out: x)

/# SCOPO: Risoluzione di un sistema triangolare superiore.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n      : intero   { dimensione del sistema }
var: u(n,n) : reale    { matrice del sistema }
var: b(n)   : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: x(n)   : reale    { vettore delle soluzioni }
                       { del sistema }
/# VARIABILI LOCALI:
var: i, j   : interi

/# INIZIO ISTRUZIONI:
x(n) := b(n)/u(n,n)
for i = n - 1 to 1 step -1
   x(i) := b(i)
   for j = i + 1 to n
      x(i) := x(i) - u(i,j) * x(j)
   end for
   x(i) := x(i)/u(i,i)
end for
end backsub
```

Procedura 2.1: Backward-substitution - schema generale

```
procedure forwsub (in: l, b, n; out: x)
/# SCOPO: Risoluzione di un sistema triangolare inferiore.
/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n      : intero   { dimensione del sistema }
var: l(n,n) : reale    { matrice del sistema }
var: b(n)   : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: x(n)   : reale    { vettore delle soluzioni }
                       { del sistema }
/# VARIABILI LOCALI:
var: i, j   : interi
/# INIZIO ISTRUZIONI:
x(1) := b(1)/l(1,1)
for i = 2 to n
   x(i) := b(i)
   for j = 1 to i - 1
      x(i) := x(i) - l(i,j) * x(j)
   end for
   x(i) := x(i)/l(i,i)
end for
end forwsub
```

Procedura 2.2: Forward-substitution - schema generale

Si osservi che sia il metodo di *back -substitution* sia quello di *forward-substitu- tion* richiedono, per la valutazione dell'incognita generica x_i, rispettivamente per $i = n - 1, \ldots, 1$ e per $i = 2, \ldots, n$, il calcolo di una somma di termini, ciascuno dei quali è il prodotto di un coefficiente della i-ma equazione per una incognita il cui valore è stato già calcolato. Il risultato di tale somma costituisce il prodotto scalare di due vettori. In particolare:

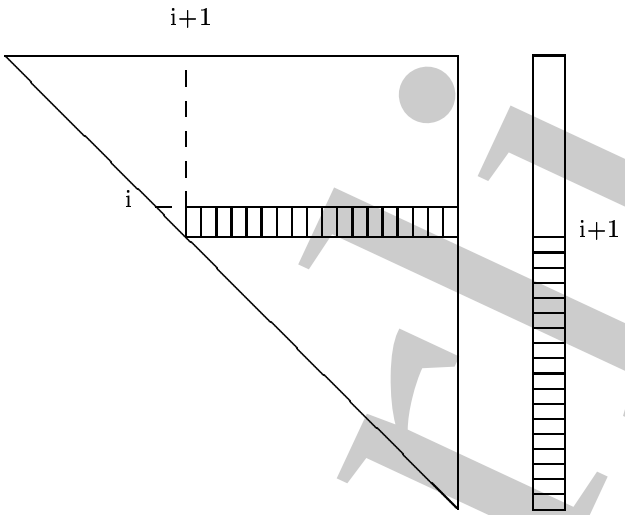

Figura 2.7: Sistema triangolare inferiore

- per il metodo di *back-substitution* (Figura 2.7) si ha:

$$\sum_{k=i+1}^{n} u_{i,k}x_k = (u_{i,i+1}\ u_{i,i+2}\ \ldots\ u_{i,n})(x_{i+1}\ x_{i+2}\ \ldots\ x_n)^T$$

- per il metodo di *forward-substitution* (Figura 2.8) si ha:

$$\sum_{k=1}^{i-1} l_{i,k}x_k = (l_{i,1}\ l_{i,2}\ \ldots\ l_{i,i-1})(x_1\ x_2\ \ldots\ x_{i-1})^T$$

Quindi, il prodotto scalare di due vettori costituisce l'operazione di calcolo matriciale alla base dei metodi di *back* e *forward-substitution.*

L'applicazione dei metodi di *back* e *forward-substitution* ad un sistema triangolare, rispettivamente superiore ed inferiore, richiede, ad ogni passo, una divisione per un elemento della diagonale, per cui è necessario chiedersi se tali elementi siano tutti distinti da zero. La risposta a tale quesito si ottiene osservando che il determinante di una matrice $T = (t_{i,j})$, *con* $i, j = 1, \ldots, n$, triangolare (inferiore o superiore) è espresso da: $det(T) = t_{1,1} \cdot t_{2,2} \cdots t_{n,n}$.
Poiché, assegnata una matrice T di dimensione n, si ha:

$$T \text{ non singolare} \iff det(T) \neq 0$$

se T è una matrice triangolare, si ottiene:

$$T \text{ triangolare non singolare} \iff t_{i,i} \neq 0 \ \ \forall i = 1, \ldots, n.$$

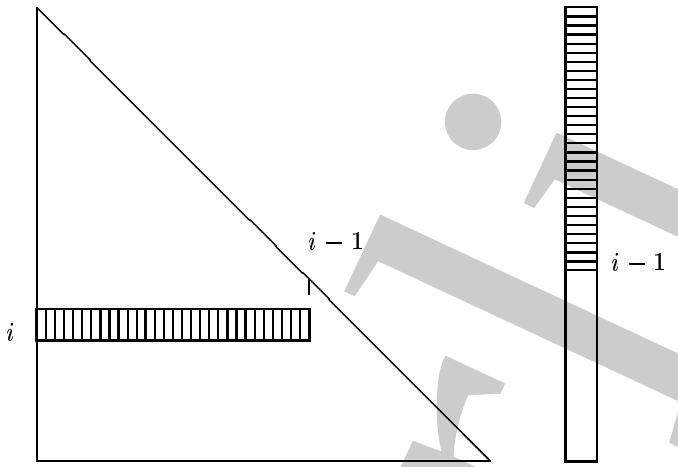

Figura 2.8: Sistema triangolare inferiore

Quindi, dato un sistema di ordine n, $Tx = b$, con T triangolare non singolare, si ha che gli elementi diagonali di T sono non nulli, e ciò permette di applicare il metodo di *back* o *forward-substitution*[6].

Si supponga, ora, di avere un sistema $Tx = b$ di ordine n, con T triangolare superiore singolare. In tal caso si ha che $det(T) = t_{1,1}t_{2,2}\ldots t_{n,n} = 0$ e pertanto T ha almeno un elemento diagonale nullo, ovvero si ha almeno per un indice i tale che $t_{i,i} = 0$. Allora la $i-$ma equazione del sistema risulta:

$$0x_i + t_{i,i+1}x_{i+1} + t_{i,i+2}x_{i+2} + \ldots + t_{i,n}x_n = b_i$$

Da ció si ricava che:

- se il sistema è compatibile ma indeterminato, e quindi ammette infinite soluzioni, si ha:

$$r_i = b_i - \sum_{k=i+1}^{n} t_{i,k}x_k = 0$$

  (r_i si dice *resto della i-ma equazione*) cioè la $i-$ma equazione è verificata per qualsiasi valore di x_i. Una delle infinite soluzioni si ottiene assegnando un valore a scelta a x_i (ad esempio $x_i = 0$) e proseguendo nell'applicazione del metodo di sostituzione;

- se il sistema è incompatibile risulta:

$$r_i \neq 0$$

  e l'applicazione del metodo di sostituzione termina.

[6] Analoghe considerazioni sussistono per un sistema diagonale.

Occorre osservare, comunque, che la prima proprietà descritta, non si inverte. Più in dettaglio, può verificarsi che la matrice sia singolare, con, in particolare, $t_{i,i} = 0$ e $r_i = 0$, per un generico i, ma il sistema risulti incompatibile piuttosto che compatibile ma indeterminato. Ad esempio, se

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & \mathbf{0} & 0 & 0 & 0 \\ 4 & 3 & 2 & 1 & 0 & 0 \\ 5 & 4 & 3 & 2 & 1 & 0 \\ 6 & 5 & 4 & 3 & 2 & \mathbf{0} \end{pmatrix} \quad \text{e} \quad b = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 10 \\ 15 \\ 10000 \end{pmatrix}$$

il sistema

$$Tx = b$$

è caratterizzato da T singolare, ed inoltre, per $i = 3$, $t_{3,3} = 0$ e $r_3 = 0$; scegliendo $x_3 = 0$, procedendo nei conti si trova

$$\begin{aligned} 1 \cdot x_1 &= 1 \\ 2 \cdot x_1 + x_2 &= 3 \Rightarrow x_2 = 1 \\ x_1 + x_2 + 0 \cdot x_3 &= 2 \Rightarrow r_3 = 0; \\ &\vdots \\ 6x_1 + 5x_2 + 4x_3 + 3x_4 + 2x_5 + 0 \cdot x_6 &= 10000 \Rightarrow 0 \neq 10000 \end{aligned}$$

il sistema non ha soluzione!!
Quindi, concludendo, le condizioni $t_{i,i} = 0$ e $r_i = 0$ non sono sufficienti per affermare che il sistema è compatibile ma indeterminato. Per queste considerazioni, allora, risulta preferibile terminare, utilizzando un opportuno indicatore di errore, l'esecuzione dell'algoritmo, per la risoluzione di un sistema che presenta tali caratteristiche, potendo, quest'ultimo, risultare incompatibile.

Pertanto, l'applicazione di un metodo di sostituzione non consente di stabilire se un sistema triangolare risulta compatibile ma indeterminato oppure incompatibile. Ci si aspetta, quindi, di ricavare una soluzione solo nel caso in cui questa sia unica.

Per quanto descritto è possibile modificare la Procedura 2.1 ottenendo la Procedura 2.3 che effettua un controllo sulla singolarità del sistema 2.2. Considerazioni analoghe sussistono per il sistema triangolare inferiore (2.6), per il quale è possibile modificare la Procedura 2.2 nella Procedura 2.4.

```
procedure backsub (in: u, b, n ; out: x, comp)

/# SCOPO: Risoluzione di un sistema triangolare superiore.

/# SPECIFICHE DEI PARAMETRI:

/# PARAMETRI DI INPUT:

var: n        : intero   { dimensione del sistema }
var: u(n,n)   : reale    { matrice del sistema }
var: b(n)     : reale    { vettore dei termini noti }

/# PARAMETRI DI OUTPUT:

var: x(n)     : reale    { vettore delle soluzioni }
                         { del sistema }
var: comp     : logical  { vera se il sistema è }
                         { determinato }
                         { falsa se il sistema è, }
                         { o potrebbe essere, incompatibile }
                         { o compatibile ma indeterminato }

/# VARIABILI LOCALI:

var: i, j     : interi
var: r        : reale

/# VARIABILI LOCALI:

var: i, j     : interi
var: r        : reale

/# INIZIO ISTRUZIONI:

comp := true                                  {sistema compatibile}
if (u(n,n) ≠ 0) then
   x(n) := b(n)/u(n,n)                        {calcolo di x(n)}
else if (b(n) = 0) then
   x(n) := 0      {il sistema potrebbe essere incompatibile}
   comp := false  {o compatibile ma indeterminato}
else
```

Procedura 2.3: Backward-substitution con controllo della singolarità - continua

```
    comp := false                          {sistema incompatibile}
  end if
  i := n − 1
  while (i ≠ 0 and comp)                   {ciclo sugli x(i)}
    r := b(i)
    for j = i + 1 to n                     {calcolo del resto}
      r := r − u(i, j) ∗ x(j)
    end for
    if (u(i, i) ≠ 0) then
      x(i) := r/u(i, i)                    {calcolo di x(i)}
    else if (r = 0) then
    x(i) := 0  {il sistema potrebbe essere incompatibile}
    comp := false     {o compatibile ma indeterminato}
    else
      comp := false                        {sistema incompatibile}
    end if
    i := i − 1
  end while
  end backsub
```

Procedura 2.3: Backward-substitution con controllo della singolarità - fine

```
procedure forwsub(in: l, b, n ; out: x, comp)

/# SCOPO: Risoluzione di un sistema triangolare inferiore.

/# SPECIFICHE DEI PARAMETRI:

/# PARAMETRI DI INPUT:

var: n      : intero   { dimensione del sistema }
var: l(n,n) : reale    { matrice del sistema }
var: b(n)   : reale    { vettore dei termini noti }

/# PARAMETRI DI OUTPUT:

var: x(n)   : reale    { vettore delle soluzioni }
                       { del sistema }
var: comp   : logical  { vera se il sistema è }
                       { compatibile }
                       { falsa se il sistema è }
                       { o potrebbe essere, incompatibile }
                       { o compatibile ma indeterminato }

/# VARIABILI LOCALI:

var: i, j   : interi
var: r      : reale

/# INIZIO ISTRUZIONI:
comp := true                                  {sistema compatibile}
if (l(1,1) ≠ 0) then
   x(1) := b(1)/l(1,1)                        {calcolo di x(1)}
else if (b(1) = 0) then
   x(1) := 0          {il sistema potrebbe essere incompatibile}
   comp := false            {o compatibile ma indeterminato}
else
   comp := false                              {sistema incompatibile}
end if
i := 2
while (i ≤ n and  comp)                       {ciclo sugli x(i)}
   r := b(i)
```

Procedura 2.4: Forward-substitution con controllo della singolarità - continua

```
    for j = 1 to i − 1                          {calcolo del resto}
       r := r − l(i, j) ∗ x(j)
    end for
    if (l(i, i) ≠ 0) then
       x(i) := r/l(i, i)                        {calcolo di x(i)}
    else if (r = 0) then
    x(i) := 0  {il sistema potrebbe essere incompatibile}
    comp := false     {o compatibile ma indeterminato}
    else
       comp := false                            {sistema incompatibile}
    end if
    i := i + 1
 end while
 end forwsub
```

Procedura 2.4: Forward-substitution con controllo della singolarità - fine

2.5 Metodo di Eliminazione di Gauss

2.5.1 Derivazione elementare del metodo di Gauss

Nel paragrafo precedente si è visto che per la risoluzione di un sistema diagonale o triangolare superiore o inferiore si dispone di algoritmi *"semplici"*. Purtroppo, nella maggior parte dei casi è necessario risolvere un sistema lineare che non è triangolare superiore o inferiore.
Un approccio, frequentemente utilizzato in Analisi Numerica, per la risoluzione di un dato problema è di trasformarlo in uno o piú problemi equivalenti di piú semplice risoluzione.
Nel nostro caso l'idea è di operare sul sistema in modo da trasformarlo in un sistema triangolare superiore (o inferiore) equivalente (ovvero che ammette la stessa soluzione).

♣ **Esempio 2.9.** Si consideri il sistema di equazioni:

$$\begin{cases} 2x & - & y & = & -1 \\ 3x & + & y & = & 3 \end{cases} \tag{2.7}$$

La soluzione di tale sistema è data da $x = 0.4, y = 1.8$, coordinate del punto di intersezione delle due rette (Figura 2.9) . Come mostrato in Figura 2.10, il punto P(0.4,1.8) è anche punto di intersezione

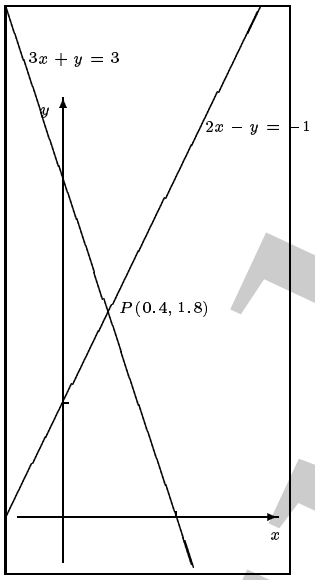

Figura 2.9: Rappresentazione grafica del sistema (2.7): la soluzione è data da $x = 0.4, y = 1.8$, coordinate del punto di intersezione delle due rette

di altre due rette, ovvero è soluzione di un sistema equivalente al (2.7):

$$\begin{cases} 2x - y = -1 \\ y = 9/5, \end{cases} \tag{2.8}$$

che è un sistema triangolare! Ci si chiede quindi con quale metodo, a partire dal sistema (2.7) sia possibile arrivare ad un sistema triangolare equivalente (2.8).

Lo scopo di tale metodo è l'eliminazione dell'incognita x dalla II equazione, ovvero la sostituzione della II equazione con un'altra nella sola incognita y. Ciò puó essere ottenuto combinando opportunamente le due equazioni date. Per realizzare ciò si eseguono le operazioni seguenti:

1. si moltiplica per $3/2$ la I equazione:

$$\frac{3}{2}(2x - y) = \frac{3}{2} \cdot (-1) \Longrightarrow 3x - \frac{3}{2}y = -\frac{3}{2};$$

   in tal modo si ottengono due equazioni aventi ugual coefficiente della x;

2. si sottrae la II equazione dalla I e l'equazione che così si ottiene,

$$-\frac{5}{2}y = -\frac{9}{2},$$

   è una equazione nella sola y che si considera come II equazione del sistema in luogo di quella data.

Pertanto al termine delle operazioni suddette si ottiene il sistema:

$$\begin{cases} 2x - y = -1 \\ y = 9/5; \end{cases}$$

che è un sistema triangolare superiore equivalente a quello dato; a tale sistema è applicabile l'algoritmo di back-substitution.

♣

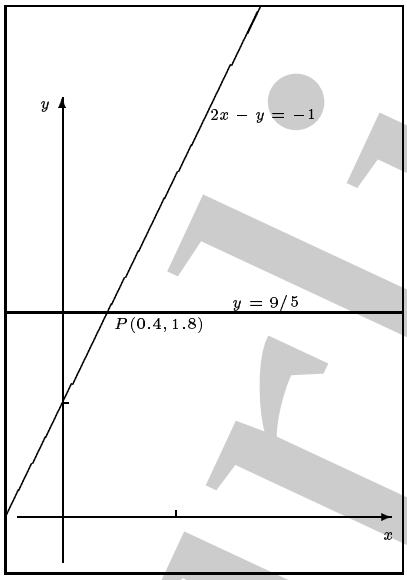

Figura 2.10: Rappresentazione grafica del sistema (2.8), equivalente al (2.7)

Per illustrare l'idea alla base del metodo di Gauss, si consideri l'esempio che segue.

♣ **Esempio 2.10.** Dato il sistema di equazioni:

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ x_1 - x_2 + 5x_3 = 0 \\ 4x_1 + x_2 - 2x_3 = 2 \end{cases}$$

si ha:

passo 1.

scopo:

eliminare l'incognita x_1 dalla II e III equazione (ovvero trasformare le ultime due equazioni che contengono tre incognite, in due equazioni che contengono due incognite).

Per realizzare ció si sottrae da ciascuna di tali equazioni la I moltiplicata per un opportuno fattore. In particolare si opera nel seguente modo:

- si moltiplica per 1/2 la prima equazione ottenendo:

$$x_1 + 2x_2 - x_3 = 3$$

e si sottrae tale equazione dalla seconda ottenendo

$$0x_1 - 3x_2 + 6x_3 = -3;$$

- si moltiplica per 2 la prima equazione ottenendo:

$$4x_1 + 8x_2 - 4x_3 = 12$$

e si sottrae tale equazione dalla terza ottenendo

$$0x_1 - 7x_2 + 2x_3 = -10.$$

Pertanto al termine del passo 1. si ottiene il seguente sistema equivalente a quello assegnato:

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ \quad - 3x_2 + 6x_3 = -3 \\ \quad - 7x_2 + 2x_3 = -10 \end{cases}$$

passo 2.

scopo:

eliminare x_2 dalla III equazione del sistema ottenuto (ovvero trasformare l'ultima equazione che contiene due incognite, in una equazione che contiene una sola incognita).
Per realizzare ciò si applica al sistema costitutito dalle ultime due equazioni, lo stesso procedimento effettuato al passo 1. sull'intero sistema. Pertanto:

- si moltiplica per 7/3 la seconda equazione, ottenendo:

$$-7x_2 + 14x_3 = -7,$$

e si sottrae questa equazione dalla terza ottenendo

$$0x_2 - 12x_3 = -3.$$

Al termine di tale passo si ottiene, quindi, il seguente sistema equivalente ai precedenti:

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ \quad - 3x_2 + 6x_3 = -3 \\ \quad\quad - 12x_3 = -3. \end{cases}$$

Il sistema ottenuto è triangolare superiore e quindi puó essere risolto applicando l'algoritmo di back-substitution.

Si osservi che i fattori 1/2, 2 e 7/3, in virtú dell'uso che di essi viene fatto, sono detti *moltiplicatori* e sono dati dal rapporto tra il coefficiente dell'incognita che deve essere eliminata in una certa equazione ed il coefficiente della stessa incognita nell'equazione che viene utilizzata per effettuare il procedimento di eliminazione (nell'esempio, la I equazione al passo 1., la II equazione al passo 2.).
Chiaramente, la fase di trasformazione del sistema eseguita mediante il metodo di eliminazione di Gauss richiede un certo numero di operazioni. Nell'esempio considerato le operazioni effettuate sono le seguenti:

- passo 1.

  2 *flop* per il calcolo dei moltiplicatori;
  6 *flop* per trasformare la II equazione;
  6 *flop* per trasformare la III equazione.

- passo 2.

  1 *flop* per il calcolo del moltiplicatore;
  4 *flop* per trasformare la III equazione.

In totale quindi, il metodo di eliminazione di Gauss, applicato ad un sistema di ordine 3, richiede:

19 *flop*.

Per ottenere la soluzione del sistema bisogna aggiungere, a tali operazioni, quelle richieste dal metodo di back-substitution che, in particolare, per l'esempio in esame, sono:

$$9\ flop.$$

Pertanto, per la risoluzione del sistema assegnato mediante il metodo di Gauss e di back-substitution sono effettuate in totale:

$$19\ flop\ +9\ flop\ =\ 28\ flop.$$

La risoluzione dello stesso sistema con il metodo di Cramer richiederebbe $20\ A + 39\ M$, ciò evidenzia i vantaggi computazionali derivanti dall'applicazione del metodo di Gauss, già per sistemi di ordine $n = 3$. ♣

2.5.2 Descrizione generale del metodo di Gauss

Si consideri il caso generale della risoluzione del sistema:

$$Ax = b$$

con A matrice dei coefficienti del sistema:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \dots & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix}$$

$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ vettore delle incognite; $b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$ vettore dei termini noti.

Lo scopo del metodo di eliminazione di Gauss è quindi quello di trasformare il sistema di partenza, attraverso opportune combinazioni lineari delle equazioni, in un sistema triangolare superiore (oppure triangolare inferiore)[7].
Il metodo opera sulla matrice A e sul vettore b e consiste nei passi seguenti:

passo 1.

scopo:

[7] Si illustra qui il caso di sistema triangolare superiore perchè è di uso prevalente nelle librerie di software matematico.

annullamento di tutte le componenti della I colonna di A, tranne l'elemento $a_{1,1}$.

procedimento:

si sottrae dalla riga i-ma, per $i = 2, 3, \ldots, n$, la I riga della matrice A moltiplicata per:

$$m_{i,1} = \frac{a_{i,1}}{a_{1,1}},$$

dove il numeratore costituisce il coefficiente dell'incognita da eliminare ed il denominatore l'elemento diagonale della colonna su cui si opera l'eliminazione. In tal modo si ottiene

$$A^{(1)} = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ 0 & a_{2,2} - \frac{a_{2,1}}{a_{1,1}} a_{1,2} & \ldots & a_{2,n} - \frac{a_{2,1}}{a_{1,1}} a_{1,n} \\ \vdots & \vdots & \ldots & \vdots \\ 0 & a_{n-1,2} - \frac{a_{n-1,1}}{a_{1,1}} a_{1,2} & \ldots & a_{n-1,n} - \frac{a_{n-1,1}}{a_{1,1}} a_{1,n} \\ 0 & a_{n,2} - \frac{a_{n,1}}{a_{1,1}} a_{1,2} & \ldots & a_{n,n} - \frac{a_{n,1}}{a_{1,1}} a_{1,n} \end{pmatrix} =$$

$$= \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ 0 & a_{2,2}^{(1)} & \ldots & a_{2,n}^{(1)} \\ \vdots & \vdots & \ldots & \vdots \\ 0 & a_{n-1,2}^{(1)} & \ldots & a_{n-1,n}^{(1)} \\ 0 & a_{n,2}^{(1)} & \ldots & a_{n,n}^{(1)} \end{pmatrix}$$

dove, relativamente agli elementi di $A^{(1)}$, l'indice in alto indica che essi hanno subito una prima trasformazione; in particolare

$$a_{i,j}^{(1)} = a_{i,j} - m_{i,1} a_{1,j}, \quad i, j = 2, 3, \ldots, n.$$

Analoga operazione va effettuata sul vettore dei termini noti. Pertanto si ha:

$$b^{(1)} = (b_1 \;\; b_2^{(1)} \;\; \ldots \;\; b_{n-1}^{(1)} \;\; b_n^{(1)})^T =$$

$$= (b_1 \;,\; b_2 - \frac{a_{2,1}}{a_{1,1}} b_1 \;,\; \ldots \;,\; b_{n-1} - \frac{a_{n-1,1}}{a_{1,1}} b_1 \;,\; b_n - \frac{a_{n,1}}{a_{1,1}} b_1)^T,$$

ovvero:

$$b_i^{(1)} = b_i - m_{i,1} b_1, \quad i = 2, 3, \ldots, n.$$

A partire dalla matrice così ottenuta si ripete il procedimento.

passo 2.

scopo:

annullamento di tutte le componenti della II colonna di $A^{(1)}$ *al di sotto dell'elemento* $a^{(1)}_{2,2}$.

procedimento:

si sottrae dalla riga i-ma, per $i = 3, 4 \ldots, n$, la II riga della matrice $A^{(1)}$ moltiplicata per:

$$m_{i,2} = \frac{a^{(1)}_{i,2}}{a^{(1)}_{2,2}}.$$

Fatto ciò si ottiene la matrice:

$$A^{(2)} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \ldots & a_{1,n} \\ 0 & a^{(1)}_{2,2} & a^{(1)}_{2,3} & \ldots & a^{(1)}_{2,n} \\ 0 & 0 & a^{(2)}_{3,3} & \ldots & a^{(2)}_{3,n} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & a^{(2)}_{n-1,3} & \ldots & a^{(2)}_{n-1,n} \\ 0 & 0 & a^{(2)}_{n,3} & \ldots & a^{(2)}_{n,n} \end{pmatrix}$$

dove:

$$a^{(2)}_{i,j} = a^{(1)}_{i,j} - m_{i,2} a^{(1)}_{2,j}, \quad i, j = 3, 4, \ldots, n.$$

Operando allo stesso modo sul vettore $b^{(1)}$ si ha:

$$b^{(2)} = (b_1 \;\; b^{(1)}_2 \;\; b^{(2)}_3 \;\; \ldots \;\; b^{(2)}_{n-1} \;\; b^{(2)}_n)^T,$$

dove:

$$b^{(2)}_i = b^{(1)}_i - m_{i,2} b^{(1)}_2, \quad i = 3, 4, \ldots, n.$$

Procedendo analogamente, al generico passo k si ottiene la matrice:

$$A^{(k)} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,k} & a_{1,k+1} & \dots & a_{1,n} \\ 0 & a_{2,2}^{(1)} & \dots & a_{2,k}^{(1)} & a_{2,k+1}^{(1)} & \dots & a_{2,n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & a_{k,k}^{(k-1)} & a_{k,k+1}^{(k-1)} & \dots & a_{k,n}^{(k-1)} \\ 0 & 0 & \dots & 0 & a_{k+1,k+1}^{(k)} & \dots & a_{k+1,n}^{(k)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 & a_{n-1,k+1}^{(k)} & \dots & a_{n-1,n}^{(k)} \\ 0 & 0 & \dots & 0 & a_{n,k+1}^{(k)} & \dots & a_{n,n}^{(k)} \end{pmatrix}$$

ed il vettore

$$b^{(k)} = (b_1 \;\; b_2^{(1)} \;\; \dots \;\; b_k^{(k-1)} \;\; b_{k+1}^{(k)} \;\; \dots \;\; b_{n-1}^{(k)} \;\; b_n^{(k)})^T$$

dove

$$a_{i,j}^{(k)} = a_{i,j}^{(k-1)} - m_{i,k} a_{k,j}^{(k-1)}, \quad i,j = k+1, \dots, n \quad (a_{i,j}^{(0)} = a_{i,j}) \tag{2.9}$$

$$b_i^{(k)} = b_i^{(k-1)} - m_{i,k} b_k^{(k-1)}, \quad i = k+1, \dots, n$$

con:

$$m_{i,k} = \frac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}}, \quad i = k+1, \dots, n. \tag{2.10}$$

Il procedimento si ripete fino ad ottenere la matrice

$$A^{(n-1)} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ 0 & a_{2,2}^{(1)} & a_{2,3}^{(1)} & \dots & a_{2,n}^{(1)} \\ 0 & 0 & a_{3,3}^{(2)} & \dots & a_{3,n}^{(2)} \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 0 & a_{n,n}^{(n-1)} \end{pmatrix}.$$

e, di conseguenza, il vettore

$$b^{(n-1)} = (b_1 \;\; b_2^{(1)} \;\; b_3^{(2)} \;\; \dots \;\; b_n^{(n-1)})^T,$$

con $A^{(n-1)}$ matrice triangolare superiore.

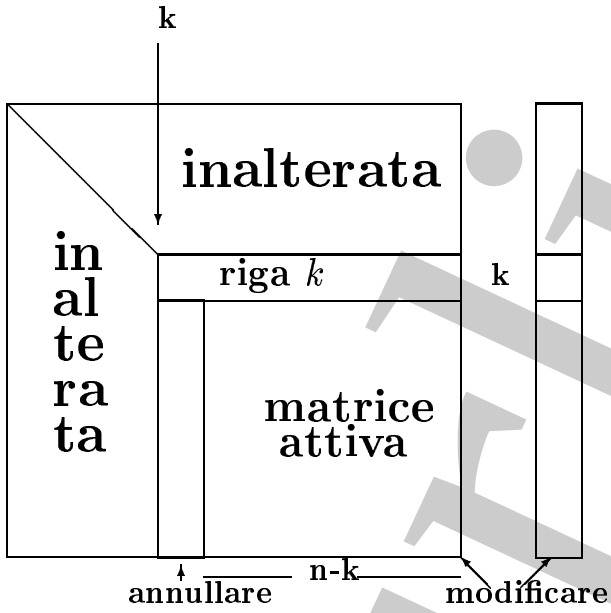

Figura 2.11: Algoritmo di eliminazione di Gauss: rappresentazione della matrice al generico passo k

In Figura 2.11 è mostrata una rappresentazione della matrice dei coefficienti e del vettore dei termini noti al passo k.

Si osserva che gli elementi della diagonale principale di $A^{(n-1)}$ sono, nell'ordine, gli $a_{k,k}^{(k-1)}$ di tutti i passi di eliminazione. Pertanto il metodo di eliminazione di Gauss per un sistema di equazioni di ordine n termina dopo $n-1$ passi. Il sistema triangolare superiore $A^{(n-1)}x = b^{(n-1)}$ ottenuto all'ultimo passo è equivalente al sistema di partenza e può quindi essere risolto mediante l'algoritmo di back-substitution. Un primo raffinamento dell'algoritmo di eliminazione di Gauss è riportato nella Procedura 2.5.

procedure gauss (**in:** a, b, n ; **out:** a)
var integer: k, n
var real: $a[n,n]$, $b[n]$
for $k = 1$ **to** $n-1$
    passo di eliminazione
    in colonna k
end for
end procedure gauss

Procedura 2.5: Algoritmo di eliminazione di Gauss - I raffinamento

Per effettuare *il passo di eliminazione in colonna k* nella Procedura 2.5 è necessario calcolare, per ogni riga i, i moltiplicatori $m_{i,k}$ (equazione (2.10)) e trasformare la riga i-ma della matrice $A^{(k-1)}$ (equazione (2.9)). Quindi un raffinamento dell'algoritmo nella Procedura 2.5 è dato dall'algoritmo nella Procedura 2.6.

procedure gauss (**in:** a, b, n; **out:** a)
var integer: i, k, n
var real: $a[n, n]$, $b[n]$
for $k = 1$ **to** $n - 1$
  for $i = k + 1$ **to** n
    calcolo del moltiplicatore $m_{i,k}$
    trasformazione $i-$**ma riga di** $A^{(k-1)}$
    trasformazione $i-$**mo elemento di** $b^{(k-1)}$
  end for
end for
end procedure gauss

Procedura 2.6: Algoritmo di eliminazione di Gauss - II raffinamento

A questo punto si osservi che, per effettuare le operazioni al passo k (costruzione di $A^{(k)}$), il metodo esposto richiede solo il risultato del passo precedente (la matrice $A^{(k-1)}$). Discende da ciò che l'algoritmo può operare su una unica struttura dati che al passo k accoglierà solo $A^{(k)}$. Questa proprietà dell'algoritmo, importante dal punto di vista della complessità di spazio, viene sintetizzata dicendo che l'algoritmo è *in place*. In pratica, gli elementi modificati al passo k, ovvero gli $a_{i,j}^{(k)}$ con $i, j = k+1, \ldots, n$ e $k = 1, \ldots, (n-1)$, sono memorizzati al posto degli elementi corrispondenti $a_{i,j}^{(k-1)}$, in quanto, questi ultimi sono utilizzati solo per il calcolo degli $a_{i,j}^{(k)}$. Inoltre, tenendo presente che al passo k gli elementi della k-ma colonna al di sotto della diagonale vengono annullati, è possibile memorizzare su tali elementi i moltiplicatori $m_{i,k}$, $i = k + 1, \ldots, n$, calcolati al passo k. La versione finale dell'algoritmo di eliminazione di Gauss è quindi riportata nella Procedura 2.7.

```
procedure gauss (in: a, b, n; out: a, b)

/# SCOPO: trasforma il sistema assegnato in uno
          triangolare superiore equivalente.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n       : intero   { dimensione del sistema }
var: a(n,n)  : reale    { matrice del sistema }
var: b(n)    : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: a(n,n)  : reale    { matrice del sistema }
                        { triangolare superiore }
var: b(n)    : reale    { vettore dei termini noti }
                        { del sistema triangolare }
                        { superiore }
/# VARIABILI LOCALI:
var: i, j, k : interi

/# INIZIO ISTRUZIONI:
for k = 1 to n - 1                          {ciclo sui passi}
   for i = k + 1 to n
      a(i,k) := a(i,k)/a(k,k)               {calcolo moltiplicatori}
      for j = k + 1 to n
         a(i,j) := a(i,j) - a(i,k) * a(k,j)
                                            {trasf. mat. attiva}
      end for
      b(i) := b(i) - a(i,k) * b(k)          {trasf. termini noti}
   end for
end for
end gauss
```

Procedura 2.7: Algoritmo di eliminazione di Gauss - versione finale

Si osservi infine che la modifica della i-ma riga è effettuata mediante un'operazione tra vettori del tipo *saxpy*

$$y = y + \alpha x$$

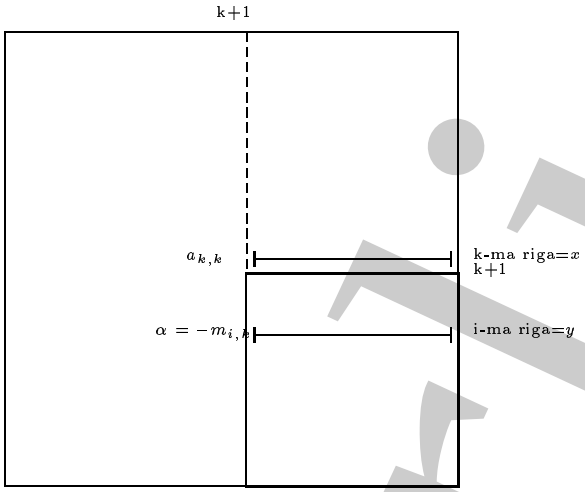

Figura 2.12: Operazione *saxpy* tra la k-ma e la i-ma riga di $A^{(k)}$

Nel caso in esame il vettore y è costituito dagli ultimi $(n-k)$ elementi della $i-$ma riga di $A^{(k-1)}$, il vettore x dagli ultimi $(n-k)$ elementi della $k-$ma riga di $A^{(k-1)}$ e α è il moltiplicatore $m_{i,k}$ cambiato di segno, come illustrato nella Figura 2.12.

2.5.3 Complessità di tempo e di spazio dell'algoritmo di eliminazione di Gauss

Il metodo di eliminazione di Gauss consente, come si è visto, di trasformare il sistema di partenza in un sistema triangolare superiore di più semplice risoluzione.

L'applicazione di tale metodo realizza quindi una fase di *preprocessing* del problema, intendendo con ciò che la sua applicazione ha lo scopo di trasformare il problema dato, mediante opportune manipolazioni, in uno o più problemi più semplici[8]. È necessario, quindi, in generale, valutare il costo computazionale della fase di preprocessing e il costo richiesto dal metodo (o dai metodi) successivamente applicati ai problemi più semplici ottenuti, in modo da verificare sia la validità dell'utilizzo del preprocessing, sia la convenienza dell'utilizzo di tale strategia considerando anche lo scopo per il quale è richiesta la soluzione del problema assegnato.

Per stimare la complessità di tempo dell'algoritmo di eliminazione di Gauss si cominci con l'osservare, ad esempio, che al passo 1. per annullare gli elementi della I colonna di A, escluso $a_{1,1}$, bisogna effettuare le operazioni seguenti:

$$m_{i,1} = a_{i,1}/a_{1,1}, \quad i = 2, \ldots, n;$$

[8]Tale tecnica si è sempre più diffusa per la risoluzione di una gran parte di problemi dell'Analisi Numerica.

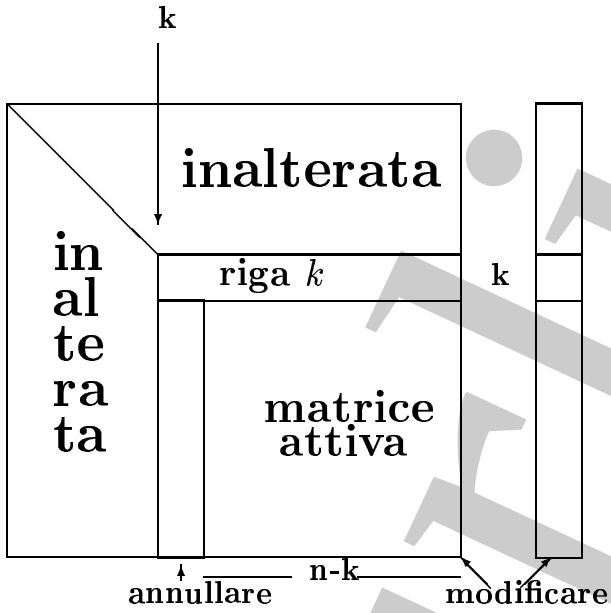

Figura 2.13: Schema della matrice A dopo k passi dell'algoritmo di Gauss.

$$a_{i,j} = a_{i,j} - m_{i,1}a_{1,j}, \quad i = 2, \ldots, n, \quad j = 2, \ldots, n;$$

$$b_i = b_i - m_{i,1}b_1, \quad i = 2, \ldots, n.$$

Pertanto il passo 1. dell'algoritmo di Gauss richiede:

- $(n-1)$ $flop$ per valutare i moltiplicatori;
- $2n(n-1)$ $flop$ per modificare la sottomatrice ed il vettore dei termini noti.

I passi successivi dell'algoritmo eseguono le stesse operazioni ma su matrici (e vettori) che diventano via via di dimensioni sempre più piccole (vedi Figura 2.13).

In particolare (vedi la Procedura 2.7), al passo k è necessario calcolare:

1. $m_{i,k} = a_{i,k}/a_{k,k}, \quad i = k+1, \ldots, n;$
2. $a_{i,j} = a_{i,j} - m_{i,k}a_{k,j}, \quad i = k+1, \ldots, n, \ j = k+1, \ldots, n;$
3. $b_i = b_i - m_{i,k}b_k, \quad i = k+1, \ldots, n.$

Posto, quindi,:

$$l = n - k$$

si ha che al passo k sono richieste:

- l $flop$ per il calcolo dei moltiplicatori;
- $l[2(l+1)\ flop]$ per il calcolo degli $a_{i,j}$ e dei b_i.

Pertanto la complessità di tempo dell'intero algoritmo di eliminazione di Gauss si ottiene sommando il numero di operazioni aritmetiche richieste a ciascun passo k, per $k = 1, \ldots, n-1$, e quindi:

$$\sum_{k=1}^{n-1}[(l+2l(l+1))\ flop] = \sum_{k=1}^{n-1}[(l+2(l^2+l))\ flop].$$

Essendo

$$\sum_{k=1}^{n-1} l\ flop = \sum_{k=1}^{n-1}(n-k)\ flop = \sum_{i=1}^{n-1} i\ flop = \frac{n(n-1)}{2}\ flop = \frac{n^2-n}{2}\ flop,$$

e

$$\sum_{k=1}^{n-1} 2(l^2+l)\ flop = \sum_{i=1}^{n-1} 2(i^2+i)\ flop =$$

$$= 2\left[\frac{n(n-1)(2n-1)}{6} + \frac{n^2-n}{2}\right]\ flop = \frac{2(n^3-n)}{3}\ flop$$

si ha che la complessità di tempo dell'algoritmo di eliminazione di Gauss, applicato ad un sistema di ordine n è:

$$T_{Gauss}(n) = \frac{2(n^3-n)}{3}\ flop + \frac{n^2-n}{2}\ flop = O\left(n^3\right).$$

La risoluzione del sistema triangolare superiore che si ottiene al termine dell'algoritmo di eliminazione di Gauss richiede l'applicazione della back-substitution che ha complessità di tempo $O\left(n^2\right)$. In definitiva, dunque, la risoluzione di un sistema di equazioni lineari di ordine n mediante l'algoritmo di eliminazione di Gauss e di back-substitution ha una complessità di tempo:

$$T_{Gauss+Back}(n) = O\left(n^3\right).$$

Il confronto tra la complessità del metodo ora esposto e del metodo di Cramer mostra la sostanziale differenza esistente tra i due metodi in termini di numero di operazioni e, quindi di tempo di esecuzione richiesto.

Per quanto riguarda la stima dello spazio di memoria richiesto dall'algoritmo di eliminazione di Gauss, si è già osservato, nel paragrafo precedente, che gli elementi calcolati al passo k possono essere memorizzati sugli elementi di ugual posto ottenuti al passo $k-1$. Pertanto la complessità di spazio dell'algoritmo di eliminazione di Gauss, per una matrice di ordine n è:

$$S_{Gauss}(n) = n(n+1) = O(n^2).$$

$n = 100$	$\#oper. = 681550$	
	Tempo di esec.	Mflops ottenuti
Pers. Comp.	3.5×10^{-4} sec.	1911 (4400)
Workstation	2.3×10^{-4} sec.	2899 (5200)
Mainframe	1.9×10^{-4} sec.	3534 (4000)
Supercomp.	6.3×10^{-5} sec.	10780 (15238)

Tabella 2.2: Tempi di esecuzione e Mflops (10^6 oper. floating-point al sec.) ottenuti per la risoluzione di un sistema di equazioni lineari di ordine $n = 100$ con il metodo di Gauss e di back-substitution su vari calcolatori. Il numero di operazioni effettuate è 681550.

Infine, nella Tabella 2.2 sono riportati, per il calcolo della soluzione di un sistema lineare di ordine 100 mediante l'algoritmo di eliminazione di Gauss e l'algoritmo di *back-substitution* implementati nella libreria di software matematico *Linpack*[9] il tempo di esecuzione richiesto e la velocità operativa raggiunta sui quattro calcolatori già considerati nella Tabella 2.1 (per una valutazione dell'efficienza del metodo di Cramer). Si noti che la massima velocità operativa in realtà è puramente teorica (o indicativa delle potenziali prestazioni del calcolatore) e nella pratica solo in alcuni casi ci si avvicina ad essa. Si ricordi che il numero totale di operazioni richiesto dall'implementazione degli algoritmi suddetti è :

$$2 \cdot \frac{n^3}{3} + 3 \cdot \frac{n^2}{2} - 7 \cdot \frac{n}{6}\ flop$$

I tempi riportati nella tabella precedente confermano che, a differenza del metodo di *Cramer*, il metodo di *eliminazione di Gauss* unitamente al metodo di *back-substitution* costituisce uno strumento effettivo, dal punto di vista computazionale, per la risoluzione di un sistema di equazioni lineari.

[9]Benché tale libreria si sia evoluta nella libreria LAPACK alla luce delle nuove architetture vettoriali e piú in generale a memoria gerarchica, essa rimane comunque un punto di riferimento per misurare le prestazioni dei calcolatori, al punto che le routine della libreria Linpack sono ancora utilizzati per misurare le prestazioni dei calcolatori. Tali test sono noti come *Linpack benchmark* e sono diventati uno vero e proprio standard de facto. Un'ulteriore evoluzione ha condotto alla libreria ScaLAPACK (or Scalable LAPACK), che include un sottoinsieme di routines LAPACK ridisegnate per macchine parallele a memoria distribuita MIMD.

2.5.4 Strategia del pivoting nell'eliminazione di Gauss

Un problema da affrontare nell'algoritmo di eliminazione di Gauss riguarda la possibilità che ad un certo passo k, per $k = 1, \ldots, n-1$, l'elemento $a_{k,k}^{(k-1)}$ sia nullo. Se una tale eventualità si verifica non risulta possibile definire i moltiplicatori $m_{i,k}$ in quanto ci si troverebbe ad effettuare una divisione per zero.

♣ **Esempio 2.11.** Si consideri il seguente sistema:

$$\begin{cases} -2x_1 + 4x_2 - 2x_3 - 6x_4 = 4 \\ 3x_1 - 6x_2 + 6x_3 + 10x_4 = -1 \\ -2x_1 + 6x_2 - x_3 + x_4 = 1 \\ 2x_1 - 5x_2 + 4x_3 + 8x_4 = -3 \end{cases}$$

la cui soluzione è il vettore $x = (1 \;\; 1 \;\; 2 \;\; -1)$.
Applicando il passo 1. dell'algoritmo di eliminazione di Gauss al sistema in esame si ottiene:

- $m_{2,1} = -3/2$; $m_{3,1} = 1$; $m_{4,1} = -1$, e

$$A^{(1)} = \begin{pmatrix} -2 & 4 & -2 & -6 \\ 0 & 0 & 3 & 1 \\ 0 & 2 & 1 & 7 \\ 0 & -1 & 2 & 2 \end{pmatrix}; \quad b^{(1)} = \begin{pmatrix} 4 \\ 5 \\ -3 \\ 1 \end{pmatrix}$$

Si osservi che l'elemento $a_{2,2}^{(1)}$ vale zero, pertanto non è possibile definire i moltiplicatori $m_{3,2}$ e $m_{4,2}$ che richiederebbero una divisione per un elemento nullo. Quindi anche se il sistema assegnato ammette soluzione (la matrice A è non singolare), essa non può essere calcolata applicando l'algoritmo di eliminazione di Gauss nella formulazione proposta nel precedente paragrafo. Come si può ovviare a tale inconveniente? La *"naturale"* strategia consiste nello scambiare la II riga di $A^{(1)}$ con la III o con la IV riga in modo che l'elemento diagonale della II riga sia diverso da zero ed applicare il passo 2. dell'algoritmo di eliminazione di Gauss alla matrice così ottenuta; ciò è lecito in quanto scambiare le righe equivale a scrivere le equazioni in ordine differente e questo, ovviamente, non altera la soluzione del sistema. Infatti, scambiando la II e la III riga si ottiene:

$$A^{(1)} = \begin{pmatrix} -2 & 4 & -2 & -6 \\ 0 & 2 & 1 & 7 \\ 0 & 0 & 3 & 1 \\ 0 & -1 & 2 & 2 \end{pmatrix}$$

Il nuovo elemento diagonale della seconda riga è ora diverso da zero, e quindi si può applicare il passo 2 dell'algoritmo di eliminazione di Gauss ottenendo:

- $m_{3,2} = 0$; $m_{4,2} = -1/2$, e

$$A^{(2)} = \begin{pmatrix} -2 & 4 & -2 & -6 \\ 0 & 2 & 1 & 7 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 5/2 & 11/2 \end{pmatrix}; \quad b^{(2)} = \begin{pmatrix} 4 \\ -3 \\ 5 \\ -1/2 \end{pmatrix}$$

Poiché $a_{3,3}^{(2)} = 3 \neq 0$ si esegue l'ultimo passo dell'algoritmo di eliminazione di Gauss che conduce a:

- $m_{4,3} = 5/6$;

$$A^{(3)} = \begin{pmatrix} -2 & 4 & -2 & -6 \\ 0 & 2 & 1 & 7 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 14/3 \end{pmatrix}; \quad b^{(3)} = \begin{pmatrix} 4 \\ -3 \\ 5 \\ -14/3 \end{pmatrix}$$

Pertanto al termine del procedimento ora descritto si ottiene il sistema triangolare superiore equivalente a quello dato

$$\begin{cases} -2x_1 + 4x_2 - 2x_3 - 6x_4 = 4 \\ 2x_2 + x_3 + 7x_4 = -3 \\ 3x_3 + x_4 = 5 \\ 14/3x_4 = -14/3 \end{cases}$$

La soluzione di tale sistema, e quindi del sistema di partenza, calcolata con l'algoritmo di *back-substitution* è $x = (1 \ 1 \ 2 \ -1)$. ♣

L'esempio considerato mostra che è sempre *"possibile"* e qualche volta *"necessario"* scambiare le equazioni del sistema quando si applica l'algoritmo di eliminazione di Gauss.

Una questione fondamentale da affrontare nella valutazione della effettiva applicabilità di un algoritmo, oltre alla sua efficienza computazionale, riguarda il suo comportamento rispetto alla propagazione degli errori di round-off derivanti dall'utilizzo di un sistema aritmetico floating-point a precisione finita. In altre parole, si pone il problema di determinare se l'algoritmo è *stabile* (o *instabile*), ovvero se le operazioni che lo costituiscono *non amplificano* (o *amplificano*) gli *inevitabili* errori di round-off della computazione.

♣ **Esempio 2.12.** Il sistema:

$$\begin{cases} 0.0001 \ x + y = 1 \\ x + y = 2, \end{cases} \tag{2.11}$$

ha la seguente soluzione (arrotondata a 5 cifre significative):

$$x = 1.0001; \quad y = .99990.$$

Utilizzando un sistema aritmetico floating-point caratterizzato dai parametri:

$$\beta = 10; \ \ t = 3; \ \ E_{min} = -6; \ \ E_{max} = 6,$$

l'algoritmo di eliminazione di Gauss applicato al sistema dato fornisce il risultato

$$x = 0 \ ; \ y = 1$$

che è evidentemente errato!

Si osservi che se si scambiano le equazioni del sistema, ottenendo

$$\begin{cases} x + y = 2 \\ 0.0001 \ x + y = 1, \end{cases}$$

e si applica l'algoritmo di eliminazione di Gauss, utilizzando il sistema aritmetico floating-point a precisione finita precedente, si ha il risultato:

$$x = 1 \ ; \ \ y = 1,$$

che è una approssimazione accettabile della soluzione.
Per spiegare il verificarsi di tali situazioni si osservi che nel sistema aritmetico floating-point a precisione finita considerato, l'applicazione del passo 1. dell'algoritmo di eliminazione di Gauss al sistema (2.11) conduce a:

- $m_{2,1} = a_{2,1}/a_{1,1} = (.100 \times 10^1)/(.100 \times 10^{-3}) = .100 \times 10^5$;
- $a^{(1)}_{2,2} = a_{2,2} - m_{2,1}a_{1,2} = .100 \times 10^1 - (.100 \times 10^5) \times (.100 \times 10^1) = .000\underline{01} \times 10^5$ $- .100 \times 10^5 \simeq -.100 \times 10^5$;
- $b^{(1)}_2 = b_2 - m_{2,1}b_1 = .200 \times 10^1 - (.100 \times 10^5) \times (.100 \times 10^1) = .000\underline{02} \times 10^5 - .100 \times 10^5 \simeq$ $-.100 \times 10^5$.

Quindi:

$$A^{(1)} = \begin{pmatrix} .100 \times 10^{-3} & .100 \times 10^1 \\ 0 & -.100 \times 10^5 \end{pmatrix}; \ b^{(1)} = \begin{pmatrix} .100 \times 10^1 \\ -.100 \times 10^5 \end{pmatrix}.$$

Si noti che il sistema $A^{(1)}x = b^{(1)}$ non è equivalente al sistema (2.11), infatti la sua soluzione è $x = 0$, $y = .100 \times 10^1$. Se si scambiano le equazioni del sistema

$$\begin{cases} x + y = 2 \\ 0.0001 \ x + y = 1 \end{cases}$$

e si applica l'algoritmo di eliminazione di Gauss si ha:

- $m_{2,1} = a_{2,1}/a_{1,1} = (.100 \times 10^{-3})/(.100 \times 10^1) = .100 \times 10^{-3}$;
- $a^{(1)}_{2,2} = a_{2,2} - m_{2,1}a_{1,2} = .100 \times 10^1 - (.100 \times 10^{-3}) \times (.100 \times 10^1) = .100 \times 10^1 - .000\underline{01} \times$ $10^1 \simeq .100 \times 10^1$;
- $b^{(1)}_2 = b_2 - m_{2,1}b_1 = .100 \times 10^1 - (.100 \times 10^{-3}) \times (.200 \times 10^1) = .100 \times 10^1 - .000\underline{02} \times 10^1 \simeq .100 \times 10^1$;

quindi

$$A^{(1)} = \begin{pmatrix} .100 \times 10^1 & .100 \times 10^1 \\ 0 & .100 \times 10^1 \end{pmatrix}, \ b^{(1)} = \begin{pmatrix} .200 \times 10^1 \\ .100 \times 10^1 \end{pmatrix}$$

e la soluzione di $A^{(1)}x = b^{(1)}$ è $x = 1$, $y = 1$, ed è un'approssimazione accettabile della soluzione del sistema(2.11).

♣

Si noti anche che nella matrice $A^{(1)}$, prodotta dall'applicazione dell'algoritmo di eliminazione di Gauss al sistema dato, gli elementi sono di ordini di grandezza diversi tra loro, mentre nella matrice $A^{(1)}$, generata dall'algoritmo di eliminazione di Gauss applicato al sistema dopo lo scambio delle equazioni, essi sono omogenei tra loro, evitando così moltiplicatori molto grandi in modulo. Inoltre, l'elemento di massimo modulo di quest'ultima matrice è molto piú piccolo di quello della prima matrice $A^{(1)}$ considerata.

Da quanto detto, quindi, si può concludere che l'algoritmo di eliminazione di Gauss è *instabile* a causa della possibilità di avere divisori *"molto piccoli"*.

L'esempio precedente mostra che l'algoritmo di eliminazione di Gauss *"fallisce"* anche quando un elemento diagonale è *"molto piccolo"* (nel sistema floating-point considerato) rispetto agli altri elementi della matrice; infatti l'algoritmo in presenza di elementi diagonali *"piccoli"* amplifica gli errori di round-off. Tale affermazione può essere derivata, in maniera intuitiva[10], dall'osservazione che nell'espressione per il calcolo degli $a_{i,j}^{(k)}$,

$$a_{i,j}^{(k)} = a_{i,j}^{(k-1)} - m_{i,k} a_{k,j}^{(k-1)}, \quad i = k+1, \ldots, n, \ j = k+1, \ldots, n$$

viene effettuata una moltiplicazione tra il moltiplicatore ed un elemento della matrice ottenuta al passo precedente su cui sono state già effettuate $k-1$ trasformazioni. Più è grande il moltiplicatore, ovvero piú è piccolo l'elemento diagonale che lo ha generato, maggiore è l'amplificazione dell'errore di round-off presente nell'elemento $a_{k,j}^{(k-1)}$. Ciò si può constatare dall'esempio considerato, non scambiando le equazioni, in cui il moltiplicatore $m_{2,1}$ è uguale a $.100 \times 10^5$, in virtú della divisione per $a_{1,1} = .100 \times 10^{-3}$, viene moltiplicato per $a_{1,2} = 1$ e per la sua grandezza provoca un fenomemo di perdita di cifre nella sottrazione con $a_{2,2}$.

L'esempio ora considerato suggerisce anche una strategia per controllare l'amplificazione degli errori di round-off nell'algoritmo di Gauss.

L'obiettivo è avere ad ogni passo moltiplicatori $m_{i,k}$ che siano in valore assoluto minori di uno, in maniera tale che nella moltiplicazione di ciascun moltiplicatore per il relativo $a_{k,j}^{(k-1)}$ non si abbia amplificazione dell'errore di round-off presente in questo elemento. L'idea di base è quindi riorganizzare, ad ogni passo k, la matrice $A^{(k)}$ in modo da avere un elemento diagonale *"non nullo"*[11] e tale che i moltiplicatori $m_{i,k}$ che si calcolano mediante esso siano tutti, in valore assoluto, minori o uguali ad uno.

La strategia generale ora descritta viene detta *pivoting*. Il problema, a questo punto, è stabilire come selezionare al passo k un elemento che soddisfi i requisiti suddetti. Tale elemento è detto *pivot*.

♣ **Esempio 2.13.** Si consideri il seguente sistema:

$$\begin{cases} 0.001x_1 & - & 7x_2 & & & = & 7 \\ -3x_1 & + & 2.099x_2 & + & 6x_3 & = & 3.901 \\ 5x_1 & - & x_2 & + & 5x_3 & = & 6 \end{cases}$$

la cui soluzione esatta è:

$$x_1 = 0; \ x_2 = -1; \ x_3 = 1.$$

Se si risolve il sistema dato mediante l'algoritmo di eliminazione di Gauss e di back-substitution utilizzando un sistema aritmetico floating-point caratterizzato dai seguenti parametri:

$$\beta = 10, \ t = 5, \ E_{min} = -6, \ E_{max} = 6,$$

[10] Una trattazione rigorosa della stabilità dell'algoritmo di eliminazione di Gauss verrà data nei paragrafi successivi.

[11] Se A è non singolare esiste sempre un $a_{i,k}^{(k-1)}$, per $i = k, \ldots, n$, non nullo.

si ha la soluzione calcolata:

$$x_1 = 0.1,\ x_2 = -0.99986,\ x_3 = 0.40008,$$

evidentemenente inaccettabile. È necessario, quindi, applicare la strategia di *pivoting*.
Per stabilire quale elemento selezionare al primo passo, ovvero qual è il pivot al primo passo, si ricordi che l'espressione dei moltiplicatori è la seguente:

$$m_{i,1} = a_{i,1}/pivot,\ \text{per}\ \ i = 2, 3$$

Poiché lo scopo della strategia di *pivoting* è riorganizzare le equazioni in modo che $|m_{i,1}| \leq 1$ per $i = 2, 3$, il pivot va selezionato in maniera tale che i numeratori delle frazioni siano tutti in valore assoluto minori o uguali al valore assoluto del pivot. Ciò si puó ottenere scegliendo come elemento pivot il più grande in valore assoluto tra gli elementi della prima colonna. Quindi, nel caso in esame, l'elemento pivot così selezionato è dato da $a_{3,1} = 5$. Poichè tale elemento si trova nella III equazione del sistema bisogna scambiare la I e la III equazione, ottenendo:

$$\begin{cases} 5x_1 & - & x_2 & + & 5x_3 & = 6 \\ -3x_1 & + & 2.099x_2 & + & 6x_3 & = 3.901 \\ 0.001x_1 & - & 7x_2 & & & = 7 \end{cases}$$

e a tale sistema si applica il passo 1. dell'algoritmo di Gauss.
Quindi:

- $m_{2,1} = -0.6;\ \ m_{3,1} = 0.0002$, e

$$A^{(1)} = \begin{pmatrix} 5 & -1 & 5 \\ 0 & 1.499 & 9 \\ 0 & -6.9998 & -0.001 \end{pmatrix};\ \ b^{(1)} = \begin{pmatrix} 6 \\ 7.501 \\ 6.9988 \end{pmatrix}.$$

Prima di applicare il passo 2. dell'algoritmo bisogna selezionare il pivot sempre con l'obiettivo che il moltiplicatore

$$m_{3,2} = a_{3,2}^{(1)}/pivot$$

sia in valore assoluto minore di uno. Analogamente a prima ciò si verifica scegliendo come pivot il più grande in valore assoluto tra l'elemento $a_{2,2}^{(1)} = 1.499$ e $a_{3,2}^{(1)} = -6.9998$. La scelta, quindi, cade sull'elemento $a_{3,2}^{(1)}$ ed è richiesto di conseguenza uno scambio tra la II e III riga di $A^{(1)}$, ottenendo

$$\begin{pmatrix} 5 & -1 & 5 \\ 0 & -6.9998 & -0.001 \\ 0 & 1.499 & 9 \end{pmatrix};\ \begin{pmatrix} 6 \\ 6.9988 \\ 7.501 \end{pmatrix}.$$

Effettuando il passo 2. dell'algoritmo di Gauss si ha:

- $m_{3,2} = 1.499/(-6.9998) = -0.21415$ e

$$A^{(2)} = \begin{pmatrix} 5 & -1 & 5 \\ 0 & -6.9998 & -0.001 \\ 0 & 0 & 8.9998 \end{pmatrix},\ \ b^{(2)} = \begin{pmatrix} 6 \\ 6.9988 \\ 8.9998 \end{pmatrix}$$

Applicando la back-substitution al sistema così ottenuto, si ha:

$$x_1 = 0;\ \ x_2 = -1;\ \ x_3 = 1.$$

♣

La strategia di *pivoting* seleziona, come elemento pivot al passo k, l'elemento di massimo modulo tra quelli della k-ma colonna a partire dall'elemento diagonale. Tale strategia di *pivoting*, detta **pivoting parziale** si puó descrivere nel modo seguente: (vedi anche Figura 2.14):

- seleziona r, il piú piccolo intero tale che

$$|a_{r,k}^{(k-1)}| = \max_{k \le i \le n} |a_{i,k}^{(k-1)}|,$$

- scambia la riga k con la riga r.

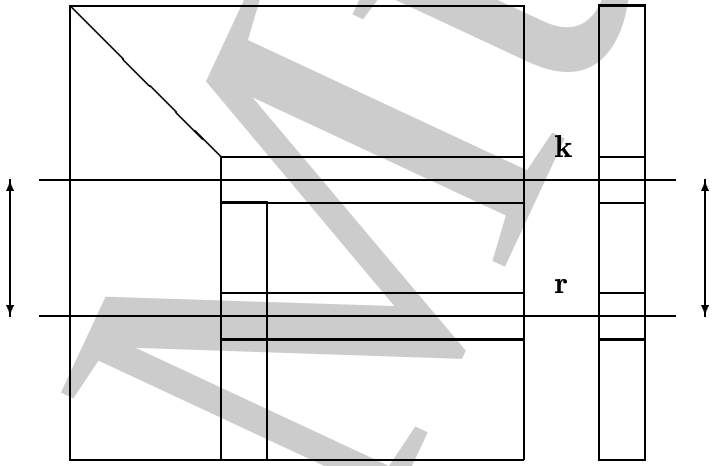

Figura 2.14: Il pivoting parziale nell'algoritmo di eliminazione di Gauss

In tal modo nel calcolo di $m_{i,k}$ si divide per il massimo elemento in valore assoluto della k colonna e quindi risulta:

$$|m_{i,k}| \le 1,$$

raggiungendo così l'obiettivo posto (riduzione dell'amplificazione dell'errore di round-off). Una procedura per effettuare sulla matrice A il pivoting al passo k dell'algoritmo di eliminazione di Gauss è data nella Procedura 2.8, mentre l'algoritmo di eliminazione di Gauss con pivoting parziale è dato nella Procedura 2.9.

procedure pivoting (**in:** a, b, n, k; **out:** a, b)

/# SCOPO: Effettua il pivoting parziale lungo le righe
/# del sistema assegnato.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:

var:	n	: **intero**	{ *dimensione del sistema* }
var:	k	: **intero**	{ *indice della prima riga* }
			{ *da scambiare* }
var:	$a(n,n)$	: **reale**	{ *matrice del sistema* }
var:	$b(n)$	: **reale**	{ *vettore dei termini noti* }

/# PARAMETRI DI OUTPUT:

var:	$b(n)$	: **reale**	{ *vettore dei termini noti* }
			{ *modificato* }
var:	$a(n,n)$	: **reale**	{ *matrice del sistema* }
			{ *modificata* }

Procedura 2.8: Procedura di pivoting per l'algoritmo di eliminazione di Gauss - continua

```
/# VARIABILI LOCALI:
var: i, j, r   : interi
var: t         : reale

/# INIZIO ISTRUZIONI:
mas := abs(a(k, k))
r := k
for i = k + 1 to n                    {ricerca del massimo}
   if (abs(a(i, k)) > mas) then
      mas := abs(a(i, k))
      r := i
   end if
endfor
if (r ≠ k) then
   for j = k to n                     {scambio delle righe}
      t := a(r, j)
      a(r, j) := a(k, j)
      a(k, j) := t
   endfor
   t := b(r)                          {scambio dei termini noti}
   b(r) := b(k)
   b(k) := t
endif
end pivoting
```

Procedura 2.8: Procedura di pivoting per l'algoritmo di eliminazione di Gauss - fine

```
procedure gausspiv (in: a, b, n; out: a, b)
/# SCOPO: applica il metodo di eliminazione di Gauss.
/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n      : intero   { dimensione del sistema }
var: a(n,n) : reale    { matrice del sistema }
var: b(n)   : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: b(n)   : reale    { vettore dei termini noti }
                       { modificato }
var: a(n,n) : reale    { matrice del sistema }
                       { modificata }
/# VARIABILI LOCALI:
var: i, j, k : interi
/# INIZIO ISTRUZIONI:
for k = 1 to n - 1                                  {ciclo sui passi}
   pivoting(in: a, b, n, k; out: a, b)              {pivoting}
   for i = k + 1 to n
      a(i,k) := a(i,k)/a(k,k)                       {calcolo moltiplicatori}
      for j = k + 1 to n
         a(i,j) := a(i,j) - a(i,k) * a(k,j)         {trasf. mat. attiva}
      end for
      b(i) := b(i) - a(i,k) * b(k)                  {trasf. termini noti}
   end for
end for
end gausspiv
```

Procedura 2.9: Algoritmo di eliminazione di Gauss con pivoting parziale

Si osservi che, al passo k, se nella fase di *pivoting* parziale risulta

$$\max_{k \le i \le n} |a_{i,k}^{(k-1)}| = 0,$$

gli elementi della $k-$ma colonna a partire dall'elemento diagonale sono già nulli, cioè:

$$A^{(k-1)} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,k-1} & a_{1,k} & a_{1,k+1} & \dots & \dots & a_{1,n} \\ 0 & a_{2,2}^{(1)} & \dots & a_{2,k-1}^{(1)} & a_{2,k}^{(1)} & a_{2,k+1}^{(1)} & \dots & \dots & a_{2,n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & a_{k-1,k-1}^{(k-2)} & a_{k-1,k}^{(k-2)} & a_{k-1,k+1}^{(k-2)} & \dots & \dots & a_{k-1,n}^{(k-2)} \\ 0 & 0 & \dots & 0 & a_{k,k}^{(k-1)} & a_{k,k+1}^{(k-1)} & \dots & \dots & a_{k,n}^{(k-1)} \\ 0 & 0 & \dots & 0 & a_{k+1,k}^{(k-1)} & a_{k+1,k+1}^{(k-1)} & \dots & \dots & a_{k+1,n}^{(k-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{n-1,k}^{(k-1)} & a_{n-1,k+1}^{(k-1)} & \dots & \ddots & a_{n-1,n}^{(k-1)} \\ 0 & 0 & \dots & 0 & a_{n,k}^{(k-1)} & a_{n,k+1}^{(k-1)} & \dots & \dots & a_{n,n}^{(k-1)} \end{pmatrix}.$$

Pertanto, l'algoritmo puó proseguire con l'annullamento degli elementi della $(k+1)-$ma colonna al di sotto dell'elemento diagonale. In tal caso al termine dell'algoritmo si otterrà una matrice triangolare superiore U dove $a_{k,k}^{(k-1)} = 0$, e poiché, in questo caso,

$$det(U) = a_{1,1}^{(0)} \cdot a_{2,2}^{(1)} \cdots a_{n,n}^{(n-1)} = 0$$

la matrice è singolare e il sistema è incompatibile o indeterminato. Al sistema triangolare superiore ottenuto con l'algoritmo di eliminazione di Gauss con pivoting parziale è infine possibile applicare l'algoritmo descritto nella Procedura 2.3 di `back substitution` con controllo della singolarità.

Riesaminando i passi del metodo di eliminazione di Gauss con pivoting, se, al passo k, si seleziona, come pivot, l'elemento di massimo modulo *in tutta la matrice attiva*, piuttosto che *nella k-esima colonna*, si realizza la tecnica denominata **pivoting totale**. Tale strategia si puó descrivere, più in dettaglio, come segue: (vedi anche Figura 2.15):

- si scelgono r ed s come gli interi più piccoli per cui

$$\left|a_{r,s}^{(k-1)}\right| = \max_{k \le i,j \le n} \left|a_{i,j}^{(k-1)}\right|,$$

- si scambiano le righe k e r e le colonne k e s.

♣ **Esempio 2.14.** Si consideri il sistema

$$\begin{cases} 0.00141x_1 + 0.04004x_2 = 0.01142 \\ 0.20000x_1 + 4.91200x_2 = 1.428 \end{cases}$$

la cui soluzione è il vettore:

$$s = (1, 0.25)^T$$

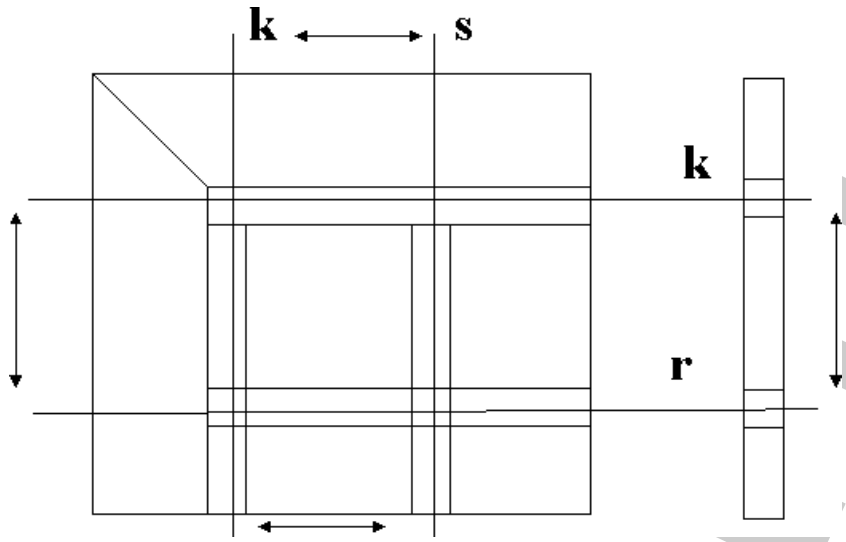

Figura 2.15: Il pivoting totale nell'algoritmo di eliminazione di Gauss

In un sistema aritmetico floating point a precisione finita:

$$\beta = 10, \quad t = 4, \quad t_{reg} = 8$$

si applichi l'algoritmo di eliminazione di Gauss con pivoting *parziale*. Individuato 0.2 come elemento massimo in modulo, nella prima colonna, si scambiano la prima e la seconda riga:

$$\begin{pmatrix} 0.2000 \times 10^0 & 0.4912 \times 10^1 \\ 0.1410 \times 10^{-2} & 0.4004 \times 10^{-1} \end{pmatrix}, \quad \begin{pmatrix} 0.1428 \times 10^1 \\ 0.1142 \times 10^{-1} \end{pmatrix}.$$

I passi dell'algoritmo di eliminazione di Gauss risultano:

$$\begin{aligned} m_{2,1} &= \frac{a_{2,1}}{a_{1,1}} = \frac{0.1410 \times 10^{-2}}{0.2000 \times 10^0} = \\ &= 0.00705 = 0.7050 \times 10^{-2} \end{aligned}$$

$$\begin{aligned} a_{2,2}^{(1)} &= a_{2,2} - m_{2,1} a_{1,2} = \\ &= 0.4004 \times 10^{-1} - (0.7050 \times 10^{-2})(0.4912 \times 10^1) = \\ &= 0.4004 \times 10^{-1} - 0.3463 \times 10^{-1} = 0.5410 \times 10^{-2} \end{aligned}$$

$$\begin{aligned} b_2^{(1)} &= b_2 - m_{2,1} b_1 = \\ &= 0.1142 \times 10^{-1} - (0.7050 \times 10^{-2})(0.1428 \times 10^1) = \\ &= 0.1142 \times 10^{-1} - 0.1007 \times 10^{-1} = 0.1350 \times 10^{-2} \end{aligned}$$

Quindi la matrice dei coefficienti ed il vettore dei termini noti diventano, rispettivamente:

$$A^{(1)} = \begin{pmatrix} 0.2000 \times 10^0 & 0.4912 \times 10^1 \\ 0 & 0.5410 \times 10^{-2} \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} 0.1428 \times 10^1 \\ 0.1350 \times 10^{-2} \end{pmatrix}.$$

Applicando la back-substitution si ha:

$$\begin{aligned}
\tilde{x}_2 &= \frac{b_2^{(1)}}{a_{2,2}^{(1)}} = \frac{0.1350 \times 10^{-2}}{0.5410 \times 10^{-2}} = 0.2495 \times 10^0 = \boxed{0.2495} \\
\tilde{x}_1 &= \frac{b_1^{(1)} - a_{1,2}^{(1)} \tilde{x}_2}{a_{1,1}^{(1)}} = \\
&= \frac{0.1428 \times 10^1 - (0.4912 \times 10^1)(0.2495 \times 10^0)}{0.2000 \times 10^0} = \\
&= \frac{0.1428 \times 10^1 - 0.1226 \times 10^1}{0.2000 \times 10^0} = \frac{0.0202 \times 10^1}{0.2000 \times 10^0} = \\
&= \frac{0.2020 \times 10^0}{0.2000 \times 10^0} = 0.1010 \times 10^1 = \boxed{1.010}
\end{aligned}$$

Quindi:

$$\tilde{s} = (\tilde{x}_1, \tilde{x}_2)^T = (1.010, 0.2495)^T \quad \text{mentre} \quad s = (1, 0.25)^T$$

da cui

$$E' = \frac{\|s - \tilde{s}\|_\infty}{\|s\|_\infty} = 0.01 = 0.1 \times 10^{-1}$$

ovvero l'errore relativo risulta dell' 1% !!! Applicando allo stesso esempio il metodo di eliminazione di Gauss con *pivoting totale* si individua, come elemento massimo in modulo, 0.4912×10^1 e si effettua, di conseguenza, lo scambio tra la prima e la seconda riga e tra la prima e la seconda colonna, ottenendo, dunque:

$$\begin{pmatrix} 0.4912 \times 10^1 & 0.2000 \times 10^0 \\ 0.4004 \times 10^{-1} & 0.1410 \times 10^{-2} \end{pmatrix}, \quad \begin{pmatrix} 0.1428 \times 10^1 \\ 0.1142 \times 10^{-1} \end{pmatrix}.$$

Applicando l'algoritmo di eliminazione di Gauss si calcolano:

$$\begin{aligned}
m_{2,1} &= \frac{a_{2,1}}{a_{1,1}} = \frac{0.4004 \times 10^{-1}}{0.4912 \times 10^1} = 0.8151 \times 10^{-2} \\
a_{2,2}^{(1)} &= a_{2,2} - m_{2,1} a_{1,2} = \\
&= 0.1410 \times 10^{-2} - (0.8151 \times 10^{-2})(0.2000 \times 10^0) = \\
&= 0.1410 \times 10^{-2} - 0.1630 \times 10^{-2} = -0.2200 \times 10^{-3} \\
b_2^{(1)} &= b_2 - m_{2,1} b_1 = \\
&= 0.1142 \times 10^{-1} - (0.8151 \times 10^{-2})(0.1428 \times 10^1) = \\
&= 0.1142 \times 10^{-1} - 0.1164 \times 10^{-1} = -0.2200 \times 10^{-3}
\end{aligned}$$

Quindi la matrice dei coefficienti ed il vettore dei termini noti diventano, rispettivamente:

$$A^{(1)} = \begin{pmatrix} 0.4912 \times 10^1 & 0.2000 \times 10^0 \\ 0 & -0.2200 \times 10^{-3} \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} 0.1428 \times 10^1 \\ -0.2200 \times 10^{-3} \end{pmatrix}.$$

Applicando la back-substitution si ha:

$$\begin{aligned}
\tilde{x}_2 &= \frac{b_2^{(1)}}{a_{2,2}^{(1)}} = 0.1000 \times 10^1 = \boxed{1.000} \\
\tilde{x}_1 &= \frac{b_1^{(1)} - a_{1,2}^{(1)}\tilde{x}_2}{a_{1,1}^{(1)}} = \\
&= \frac{0.1428 \times 10^1 - (0.2000 \times 10^0)(0.1000 \times 10^1)}{0.4912 \times 10^1} = \\
&= \frac{0.1428 \times 10^1 - 0.2000 \times 10^0}{0.4912 \times 10^1} = 0.2500 \times 10^0 = \\
&= \boxed{0.25}
\end{aligned}$$

Quindi:

$$\tilde{s} = (\tilde{x}_1, \tilde{x}_2)^T = (1, 0.25)^T \quad \text{e} \quad s = (1, 0.25)^T$$

ovvero la soluzione può ritenersi accurata. ♣

Si sottolinea, a proposito della tecnica del pivoting totale, che il

$$(k-1)-\text{esimo pivot:} \qquad \max_{k \le i,j, \le n} \left| a_{i,j}^{(k-1)} \right|$$

viene determinato come l'elemento di massimo modulo *tra tutti gli elementi della matrice attiva*, per cui, dividendo opportunamente per esso, si ottiene il moltiplicatore $|m_{i,k}|$ che risulta minore o uguale del moltiplicatore ottenuto col pivoting parziale; si può dedurre, allora, che **rispetto al pivoting parziale, il pivoting totale garantisce una maggiore riduzione dell'errore di round off**. In effetti, con il pivoting parziale, l'errore relativo di round off nella soluzione è, in generale, al più il doppio rispetto a quello risultante dal pivoting totale. Per quanto riguarda il *costo del pivoting totale*, si effettuano

$(n-k+1)^2$ confronti ad ogni, generico, passo k $(k = 1, \ldots, n-1)$,

per un totale di

$$O\left(\frac{n^3}{3}\right) \quad \text{confronti}$$

e questo è il costo, in termini di numero di confronti, che il pivoting totale aggiunge alla complessià computazionale dell'algoritmo di eliminazione di Gauss. Il *costo del pivoting parziale*, invece, richiede

$(n-k+1)$ confronti ad ogni, generico, passo k $(k = 1, \ldots, n-1)$,

per un totale di

$$O\left(\frac{n^2}{2}\right) \quad \text{confronti}$$

e questo è il costo, in termini di numero di confronti, che il pivoting parziale aggiunge alla complessià computazionale dell'algoritmo di eliminazione di Gauss. Sostanzialmente si può concludere che il pivoting parziale risulta caratterizzato da:

- errore relativo di round off sulla soluzione accettabile;
- minor numero di confronti del pivoting totale

ed è per queste ragioni che conviene, in generale, utilizzare la strategia del pivoting parziale, piuttosto che quella del pivoting totale.

2.6 Fattorizzazione LU

♣ **Esempio 2.15.** Si consideri il sistema

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ x_1 - x_2 + 5x_3 = 0 \\ 4x_1 + x_2 - 2x_3 = -2 \end{cases}$$

o, equivalentemente, in forma compatta:

$$Ax = b$$

con

$$A = \begin{pmatrix} 2 & 4 & -2 \\ 1 & -1 & 5 \\ 4 & 1 & -2 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad b = \begin{pmatrix} 6 \\ 0 \\ -2 \end{pmatrix}$$

L'algoritmo di eliminazione di Gauss senza pivoting trasforma il sistema in uno triangolare superiore:

$$Ux = c, \qquad (\, A^{(n-1)}x = b^{(n-1)}\,),$$

con

$$U = \begin{pmatrix} 2 & 4 & -2 \\ 0 & -3 & 6 \\ 0 & 0 & -12 \end{pmatrix}, \quad c = \begin{pmatrix} 6 \\ -3 \\ -7 \end{pmatrix}$$

I moltiplicatori sono:

- al passo 1: $m_{2,1} = 1/2$, $m_{3,1} = 2$;
- al passo 2: $m_{3,2} = 7/3$.

Si consideri, ora, la matrice:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ m_{2,1} & 1 & 0 \\ m_{3,1} & m_{3,2} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 2 & 7/3 & 1 \end{pmatrix};$$

L è una matrice triangolare inferiore con:

- elementi diagonali uguali ad 1;

- elementi della k-ma colonna al di sotto dell'elemento diagonale uguali ai moltiplicatori calcolati al passo k ($k = 1, .., n-1$);

Se si calcola il prodotto LU si ha:

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 2 & 7/3 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 0 & -3 & 6 \\ 0 & 0 & 12 \end{pmatrix} =$$

$$= \begin{pmatrix} 2 & 4 & -2 \\ 1 & -1 & 5 \\ 4 & 1 & -2 \end{pmatrix} = A$$

ovvero:

$$LU = A$$

♣

Il risultato dell'esempio precedente sussiste in generale, ovvero l'algoritmo di eliminazione di Gauss può essere visto come un *metodo di fattorizzazione* della matrice A nel prodotto di due matrici:

$$A = LU$$

con:

- L triangolare inferiore,
- U triangolare superiore.

In particolare, per un sistema di dimensione n, abbiamo:

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ m_{2,1} & 1 & 0 & \dots & 0 \\ m_{3,1} & m_{3,2} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{n,1} & m_{n,2} & m_{n,3} & \dots & 1 \end{pmatrix}$$

cioé L ha:

- elementi diagonali uguali ad 1;
- elementi della I colonna al di sotto dell'elemento diagonale uguali ai moltiplicatori calcolati al passo 1;

- elementi della II colonna al di sotto dell'elemento diagonale uguali ai moltiplicatori calcolati al passo 2, e così via.

$$U = A^{(n-1)} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ 0 & a_{2,2}^{(1)} & a_{2,3}^{(1)} & \dots & a_{2,n}^{(1)} \\ 0 & 0 & a_{3,3}^{(2)} & \dots & a_{3,n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & a_{n,n}^{(n-1)} \end{pmatrix}$$

cioè U è la matrice ottenuta all'ultimo passo dell'algoritmo di eliminazione di Gauss.

♣ **Esempio 2.16.** Si riprenda l'esempio precedente e si osservi che, definita la matrice:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}$$

costruita a partire dai moltiplicatori relativi al primo passo dell'algoritmo di eliminazione di Gauss, si può osservare che:

$$M_1 A = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 1 & -1 & 5 \\ 4 & 1 & -2 \end{pmatrix} = \begin{pmatrix} 2 & 4 & -2 \\ 0 & -3 & 6 \\ 0 & -7 & 2 \end{pmatrix} = A^{(1)}$$

Cioè la matrice che si ottiene da A dopo un passo dell'algoritmo di eliminazione di Gauss può essere ottenuta moltiplicando a sinistra tale matrice per una opportuna matrice M_1. Analogamente al secondo passo, definendo la matrice:

$$M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{7}{3} & 1 \end{pmatrix}$$

a partire dall'unico moltiplicatore $m_{3,2}$, è possibile ottenere:

$$M_2 M_1 A = M_2 A^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{7}{3} & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 0 & -3 & 6 \\ 0 & -7 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 4 & -2 \\ 0 & -3 & 6 \\ 0 & 0 & -12 \end{pmatrix} = A^{(2)}$$

da cui si ha che $L = M_1^{-1} M_2^{-1}$ e $U = A^{(2)}$, cioè la matrice che si ottiene all'ultimo passo dell'algoritmo di eliminazione di Gauss.

♣

In generale sussiste la seguente:

Definizione 2.7. (Matrice triangolare unitaria elementare inferiore) *Dato un vettore m del tipo:*

$$m^T = (0, 0, .., m_{k+1}, .., m_n)$$

la matrice quadrata M_k di ordine n definita come:

$$M_k = I - me_k^T \qquad dove \qquad e_k^T = (\underbrace{0, 0, \ldots, 0}_{k-1}, 1, 0, \ldots, 0)$$

è detta **triangolare unitaria elementare inferiore** *di indice k.*

La matrice M_k è del tipo:

$$M_k = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & -m_n & 0 & \cdots & 1 \end{pmatrix}$$

È possibile ora dimostrare il seguente:

Teorema 2.1. *Sia x un vettore di ordine n:*

$$x = (\xi_1\ , \xi_2\ , ..., \xi_n)^T \qquad \xi_k \neq 0$$

esiste una e una sola matrice M_k triangolare elementare unitaria inferiore di indice k tale che:

$$M_k x = (\xi_1,\ \xi_2,\ ..., \xi_k,\ 0, .., 0)^T$$

Dimostrazione Posto $M_k = I - me_k^T$, affinché tale matrice sia triangolare inferiore di indice k deve essere:

$$m_i = 0 \quad i = 1, .., k \tag{2.12}$$

Inoltre, da:

$$M_k x = (I - me_k^T)x = x - (e_k^T x)m$$

le ultime $n - k$ componenti di $M_k x$ sono nulle ovvero:

$$\xi_i - \xi_k m_i = 0 \quad i = k + 1, .., n$$

ma, poiché $\xi_k \neq 0$ si ha:

$$m_i = \frac{\xi_i}{\xi_k} \quad i = k+1, .., n \tag{2.13}$$

La matrice M_k è quindi univocamente determinata. ■

Tale teorema assicura l'esistenza di una matrice triangolare unitaria inferiore di indice k univocamente determinata da (2.12) e (2.13), che annulla le ultime $n-k$ componenti di un vettore x, lasciando inalterate le prime k componenti. Se, inoltre $\xi_k = 0$ la matrice M_k non esiste.
Quindi, al generico passo k dell'algoritmo di eliminazione di Gauss, detta:

$$x = \left(a_{1,k}^{(0)}, \dots, a_{k-1,k}^{(k-1)}, \dots, a_{n,k}^{(k-1)}\right)^T$$

la $k-$ma colonna della matrice $A^{(k-1)}$, dalle (2.12) e (2.13) il vettore m che definisce la matrice M_k è:

$$m^T = \left(0, \dots, 0, \frac{a_{k+1,k}^{(k-1)}}{a_{k,k}^{(k-1)}}, \dots, \frac{a_{n,k}^{(k-1)}}{a_{k,k}^{(k-1)}}\right) = (0, \dots, 0, m_{k+1,k}, \dots, m_{n,k})$$

costituito cioè dai moltiplicatori calcolati nel $k-$mo passo del processo di eliminazione.

Per annullare le ultime $n-k$ componenti della $k-$ma colonna della matrice lasciando inalterate le prime k, è sufficiente moltiplicare a sinistra la matrice $A^{(k-1)}$ per la matrice elementare unitaria inferiore M_k, cioè:

$$A^{(k)} = M_k A^{(k-1)}$$

Si osservi inoltre che le matrici M_k esistono se e solo se gli elementi $a_{k,k}^{(k-1)} \neq 0$. L'esistenza delle matrici M_k equivale quindi alla condizione di applicabilità dell'algoritmo di eliminazione di Gauss.

Da quanto detto si ha che le $n-1$ trasformazioni dell'algoritmo di eliminazione di Gauss equivalgono a moltiplicare a sinistra la matrice A per le matrici elementari trangolari inferiori M_k con $k = 1, \dots, n-1$, cioè:

$$A^{(n-1)} = M_{n-1} M_{n-2} \cdots M_1 A$$

da cui:

$$A = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} A^{(n-1)}$$

e quindi, definendo $L = M_1^{-1} M_2^{-1} \cdots M_{k-1}^{-1}$ e $U = A^{(n-1)}$ si ottiene che $A = LU$.

Si osservi infine che, data la matrice triangolare elementare unitaria inferiore $M_k = I - me_k^T$ si ha:

$$M_k^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & m_{k+1,k} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & m_n & 0 & \cdots & 1 \end{pmatrix} = I + me_k^T$$

e che:

$$L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ m_{2,1} & 1 & 0 & \cdots & 0 \\ m_{3,1} & m_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & m_{n,3} & \cdots & 1 \end{pmatrix}$$

cioè L è una matrice triangolare inferiore, costituita da tutti i moltiplicatori calcolati durante il processo di eliminazione con, inoltre, $l_{k,k} = 1$ per $k = 1, \ldots, n$.

Quanto detto assicura quindi che l'algoritmo di eliminazione di Gauss calcola una fattorizzazione della matrice A del tipo $A = LU$, con L e U rispettivamente matrici triangolari inferiore e superiore tali che:

$$l_{i,k} = \begin{cases} 1 & \text{se } i = k \\ m_{i,k} = a_{i,k}^{(k-1)} / a_{k,k}^{(k-1)} & \text{se } i > k \end{cases} \qquad e \quad u_{i,k} = a_{i,k}^{(k-1)} \quad i \leq k$$

Ci si chiede ora se esistano altre fattorizzazioni con le medesime proprietà. La risposta a tale quesito è data dal seguente:

Teorema 2.2. *Sia data una matrice quadrata A di ordine n, e siano $A_k = (a_{i,j})$ $i, j = 1, .., k$ i minori principali di ordine k. Se tali matrici A_k sono non singolari esiste una unica matrice triangolare inferiore L e un'unica matrice triangolare superiore U tali che:*

$$A = LU$$

con $diag(L) = (1, 1, \ldots, 1)$ e $det(A_k) = u_{1,1} u_{2,2} \cdots u_{k,k}$.

Dimostrazione La dimostrazione procede per induzione. Per $n = 1$ si ha banalmente che la fattorizzazione $A = LU$ risulta essere $a_{1,1} = 1 u_{1,1}$. Si supponga quindi il teorema vero per $n = k - 1$. Si ha quindi che esiste una fattorizzazione $A_{k-1} = L_{k-1} U_{k-1}$. Siano ora le matrice A_k, L_k e U_k così partizionate:

$$A_k = \begin{pmatrix} A_{k-1} & b \\ c^T & a_{k,k} \end{pmatrix} \quad L_k = \begin{pmatrix} L_{k-1} & \underline{0} \\ m^T & 1 \end{pmatrix} \quad U_k = \begin{pmatrix} U_{k-1} & u \\ \underline{0} & u_{k,k} \end{pmatrix} \tag{2.14}$$

dove b e c sono vettori costituiti rispettivamente dalle prime $k-1$ componenti della k colonna e dalla k riga di A, mentre l'elemento diagonale $u_{k,k}$ e i vettori m e u, di ordine $k-1$, sono da determinare.

Imponendo che $A_k = L_k U_k$ si ottengono le relazioni seguenti:

$$\begin{array}{ll} 1 & A_{k-1} = L_{k-1} U_{k-1} \\ 2 & L_{k-1} u = b \\ 3 & m^T U_{k-1} = c \\ 4 & m^T u + u_{k,k} = a_{k,k} \end{array} \tag{2.15}$$

Inoltre si ha che $det(A_k) = det(L_k)det(U_k)$. La prima delle (2.15) è vera per l'ipotesi di induzione. La seconda è invece un sistema di equazioni lineari con matrice triangolare inferiore non singolare, in quanto $det(L_{k-1}) = 1$ la cui soluzione è il vettore u. Analogamente la terza delle (2.15) è un sistema triangolare superiore con matrice non singolare, in quanto $det(U_{k-1}) = det(A_{k-1}) = u_{1,1} \cdots u_{k-1,k-1} \neq 0$ e che ha soluzione m. Dalla quarta relazione è possibile poi determinare $u_{k,k}$. Infine, dalle posizioni fatte in (2.14) si ha anche che $diag(L_k) = (1, 1, \ldots, 1)$ e $det(A_k) = u_{1,1}u_{2,2} \cdots u_{k,k}$. ■

Tale teorema assicura l'esistenza di un'unica fattorizzazione del tipo $A = LU$ con le proprietà della fattorizzazione calcolata dall'algoritmo di eliminazione di Gauss. È possibile quindi affermare che l'algoritmo di eliminazione di Gauss è il metodo costruttivo per il calcolo dell'unica fattorizzazione $A = LU$ tale che:

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ m_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & 1 \end{pmatrix} \quad U = \begin{pmatrix} a_{1,1}^{(0)} & a_{1,2}^{(0)} & \cdots & a_{1,n}^{(0)} \\ 0 & a_{2,2}^{(1)} & \cdots & a_{2,n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n}^{(n-1)} \end{pmatrix}$$

Una volta calcolati, mediante l'algoritmo di Gauss, i fattori L e U di A (ovvero la *"fattorizzazione LU"* della matrice A), risolvere il sistema

$$Ax = b$$

è equivalente a risolvere il sistema:

$$LUx = b$$

Per avere la soluzione di tale sistema, si risolve dapprima il sistema

$$Ly = b \quad (\text{posto } Ux = y)$$

mediante la *forward-substitution*, e poi il sistema

$$Ux = y$$

mediante *la back-substitution*.

Si osservi, inoltre, che risolvere il sistema $Ly = b$ equivale ad effettuare sul vettore b gli $(n-1)$ passi dell'algoritmo di Gauss, ovvero $y = b^{(n-1)}$, come mostrato nell'esempio che segue:

♣ **Esempio 2.17.** Se si risolve il sistema:

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

con la *forward substitution* si ha:

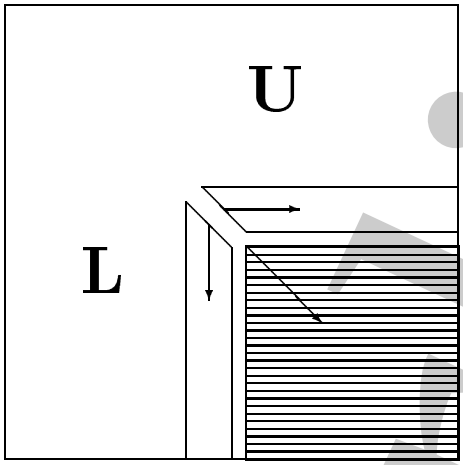

Figura 2.16: Rappresentazione della fattorizzazione LU della matrice A

- $y_1 = b_1$;
- $y_2 = b_2 - m_{2,1}y_1 = b_2^{(1)}$;
- $y_3 = b_3 - m_{3,1}y_1 - m_{3,2}y_2 = (b_3 - m_{3,1}y_1) - m_{3,2}b_2^{(1)} = b_3^{(1)} - m_{3,2}b_2^{(1)} = b_3^{(2)}$;

ovvero:

$$y = (b_1 \quad b_2^{(1)} \quad b_3^{(2)})^T = b^{(2)}.$$

♣

In conclusione, quindi, si ha che l'algoritmo di Gauss è equivalente a:

- fattorizzazione LU di A (triangolarizzazione di A),
- risoluzione del sistema $Ly = b$ (calcolo del vettore $b^{(n-1)}$).

e che tale equivalenza sussiste anche dal punto di vista della complessità computazionale.

La possibilità di fattorizzare la matrice A nella forma $A = LU$ risulta particolarmente vantaggiosa, per la risoluzione di quei problemi nei quali si devono risolvere piú sistemi aventi la stessa matrice dei coefficienti e diversi termini noti.

Problema 1: risoluzione di più sistemi lineari con uguale matrice dei coefficienti.

Molte applicazioni richiedono la risoluzione di piú sistemi lineari del tipo:

$$Ax_1 = b_1, \quad Ax_2 = b_2, \quad \ldots, Ax_m = b_m,$$

cioè sistemi lineari aventi la stessa matrice dei coefficienti ma termini noti differenti che, ad esempio, non sono dati a priori ma dipendono dalla soluzione di uno dei sistemi precedenti (ad es. b_2 dipende in qualche maniera da x_1). In tal caso applicando l'algoritmo

di eliminazione di Gauss e la *back-substitution* a ciascun sistema, il costo computazionale sarebbe uguale a:

$$m(T_{Gauss}(n) + T_{bs}(n)) \simeq O\left(mn^3\right)$$

Tale costo computazionale è invece notevolmente ridotto utilizzando la fattorizzazione LU di A. Infatti, una volta calcolati i fattori L ed U basta risolvere le coppie di sistemi:

$$\begin{array}{ll} Ly_1 = b_1, & Ux_1 = y_1; \\ Ly_2 = b_2, & Ux_2 = y_2; \\ Ly_3 = b_3, & Ux_3 = y_3; \\ \vdots & \vdots \\ Ly_m = b_m, & Ux_m = y_m; \end{array}$$

In sintesi per risolvere il problema in esame basta:

- calcolare **una sola volta** la fattorizzazione LU di A, con un costo computazionale di $O(\frac{n^3}{3})$;
- risolvere m coppie di sistemi triangolari (uno inferiore ed uno superiore), con un costo computazionale di $O(2\frac{n^2}{2})$ per ciascuna di esse.

Di conseguenza il costo computazionale totale è ora

$$O\left(\frac{n^3}{3} + mn^2\right)$$

Problema 2: calcolo dell'inversa di una matrice.

Per calcolare l'inversa A^{-1} di una assegnata matrice A di dimensione n, occorre risolvere n sistemi lineari:

$$Ax_i = e_i, \quad i = 1, \ldots, n,$$

dove x_i è la $i-$ma colonna di A^{-1} e e_i è l' i_mo vettore unitario.
Questo problema è analogo al *Problema 1*, pertanto, utilizzando l'approccio descritto precedentemente si ha che la complessità computazionale del calcolo dell'inversa è:

$$O\left(\frac{n^3}{3}\right) + 2nO\left(\frac{n^2}{2}\right) = O\left(\frac{4}{3}n^3\right)$$

Problema 3: calcolo del determinante di una matrice.

Un altro problema in cui la fattorizzazione LU risulta notevolmente vantaggiosa è il calcolo del determinante di una matrice A di dimensione n. Infatti, dalla relazione

$$A = LU$$

si ricava che:

$$det(A) = det(LU) = det(L) \cdot det(U).$$

Poiché L ed U sono matrici triangolari, il loro determinante è dato dal prodotto dei rispettivi elementi diagonali. Pertanto, poiché L ha tutti gli elementi diagonali uguali ad 1 si ha che:

$$det(A) = det(U) = u_{1,1} \cdot u_{2,2} \cdots u_{n,n}$$

In conclusione, la complessità computazionale del calcolo del determinante di A, utilizzando la fattorizzazione LU, è:

$$O\left(\frac{n^3}{3}\right)$$

Ricordando che la complessità computazionale del calcolo del determinante, è $O(n!)$, risulta evidente il vantaggio computazionale ottenuto.
Si consideri ora il seguente:

♣ **Esempio 2.18.** Si applichi l'algoritmo di eliminazione di Gauss con pivoting parziale al sistema:

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ x_1 - x_2 + 5x_3 = 0 \\ 4x_1 + x_2 - 2x_3 = -2. \end{cases}$$

- passo 1
  - pivoting in I colonna: $\max_{1\leq i\leq 3} |a_{i,1}| = 4 = a_{1,3}$, per cui si scambiano la I e la III equazione:

$$\begin{cases} 4x_1 + x_2 - 2x_3 = -2 \\ x_1 - x_2 + 5x_3 = 0 \\ 2x_1 + 4x_2 - 2x_3 = 6; \end{cases}$$

  - calcolo moltiplicatori: $m_{2,1} = 1/4$ e $m_{3,1} = 1/2$;
  - eliminazione in I colonna:

$$\begin{cases} 4x_1 + x_2 - 2x_3 = -2 \\ -\frac{5}{4}x_2 + \frac{11}{2}x_3 = \frac{1}{2} \\ \frac{7}{2}x_2 - x_3 = 7; \end{cases}$$

- passo 2
  - pivoting in II colonna $\max_{2\leq i\leq 3} |a_{i,2}^{(1)}| = 7/2 = a_{3,2}^{(1)}$, per cui si scambiano la II e la III equazione:

$$\begin{cases} 4x_1 + x_2 - 2x_3 = -2 \\ \frac{7}{2}x_2 - x_3 = 7 \\ -\frac{5}{4}x_2 + \frac{11}{2}x_3 = \frac{1}{2}; \end{cases}$$

  - calcolo moltiplicatore: $m_{3,2} = (-\frac{5}{4})/(\frac{7}{2}) = -\frac{5}{14}$;

– eliminazione in II colonna:

$$\begin{cases} 4x_1 + x_2 - 2x_3 = -2 \\ \frac{7}{2}x_2 - x_3 = 7 \\ \frac{36}{7}x_3 = 3 \end{cases}$$

Se si considerano ora, le due matrici:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & -5/14 & 1 \end{pmatrix}; \quad U = \begin{pmatrix} 4 & 1 & -2 \\ 0 & 7/2 & -1 \\ 0 & 0 & 36/7 \end{pmatrix}$$

dove L è la matrice triangolare inferiore avente elementi diagonali uguali ad 1 ed elementi della $i-$ma colonna al di sotto dell'elemento diagonale uguali ai moltiplicatori calcolati al passo i[12], ed U è la matrice triangolare superiore ottenuta all'ultimo passo dell'algoritmo, e si calcola il prodotto LU, si ha:

$$LU = \begin{pmatrix} 4 & 1 & -2 \\ 2 & 4 & -2 \\ 1 & -1 & 5 \end{pmatrix}$$

Confrontando tale matrice con la matrice A si ha che LU ha le stesse righe di A ma *in ordine diverso*. Ciò deriva dal fatto che l'applicazione del pivoting parziale modifica l'ordine delle righe. Si effettuino ora su A gli scambi fatti durante le fasi di pivoting:

- al passo 1 è stato effettuato uno scambio tra la I e la III riga, cioè:

$$\begin{pmatrix} 2 & 4 & -2 \\ 1 & -1 & 5 \\ 4 & 1 & -2 \end{pmatrix} \Longrightarrow \begin{pmatrix} 4 & 1 & -2 \\ 1 & -1 & 5 \\ 2 & 4 & -2 \end{pmatrix};$$

- al passo 2 è stato effettuato uno scambio tra la II e la III riga, cioè:

$$\begin{pmatrix} 4 & 1 & -2 \\ 1 & -1 & 5 \\ 2 & 4 & -2 \end{pmatrix} \Longrightarrow \begin{pmatrix} 4 & 1 & -2 \\ 2 & 4 & -2 \\ 1 & -1 & 5 \end{pmatrix} = LU$$

♣

L'esempio precedente mostra che l'algoritmo di eliminazione di Gauss con pivoting parziale calcola i fattori L ed U di una matrice avente le stesse righe di A ma nell'ordine determinato dagli scambi di righe effettuati nell'applicazione del pivoting parziale, ovvero A si ottiene da LU operando su essa gli scambi di righe effettuati nell'applicazione dell'algoritmo di eliminazione di Gauss con pivoting parziale.

♣ **Esempio 2.19.** Nell'esempio 2.18 a partire dalla matrice:

$$A = \begin{pmatrix} 2 & 4 & -2 \\ 1 & -1 & 5 \\ 4 & 1 & -2 \end{pmatrix}$$

[12]Si osservi che sulla II riga c'è il moltiplicatore relativo alla riga che è stata modificata una sola volta, mentre sulla III riga ci sono i due moltiplicatori relativi alla riga modificata due volte.

si è calcolata la matrice:

$$LU = \begin{pmatrix} 4 & 1 & -2 \\ 2 & 4 & -2 \\ 1 & -1 & 5 \end{pmatrix} = \overline{A}.$$

Si consideri la seguente matrice:

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

calcolando il prodotto di P per A si ha

$$PA = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 1 & -1 & 5 \\ 4 & 1 & -2 \end{pmatrix} =$$

$$= \begin{pmatrix} 4 & 1 & -2 \\ 2 & 4 & -2 \\ 1 & -1 & 5 \end{pmatrix} = \overline{A} = LU$$

♣

L'effetto della moltiplicazione di P per A è stato di modificare l'ordine delle righe di A. Per tale motivo P è detta matrice di *permutazione*. La matrice P considerata è un esempio di matrice di permutazione (cfr. §A.1.1, Appendice A).
Si puó, quindi, concludere che i fattori L ed U prodotti dall'algoritmo sono i fattori della matrice PA ovvero:

$$PA = LU$$

dove P è una matrice di permutazione che *"conserva"* l'informazione degli scambi effettuati. Una volta effettuata la fattorizzazione $PA = LU$, ed osservando che $PAx = Pb$, la risoluzione del sistema $Ax = b$ puó essere effettuata mediante la risoluzione del sistema triangolare inferiore $Ly = Pb$ mediante la *forward substitution* e del sistema triangolare superiore $Ux = y$ mediante la *backward substitution*. Si osservi comunque che la risoluzione del sistema $Ly = Pb$ richiede che vengano effettuate sul vettore dei termini noti b tutte le permutazioni di righe fatte sulla matrice A nel corso dell'algoritmo di eliminazione di Gauss con pivoting parziale.

♣ **Esempio 2.20.** Nell'esempio 2.18

- al passo 1 è stato effettuato uno scambio tra la I e III riga di A. A questo punto si osservi che se si effettuano gli stessi scambi sulla matrice I, si ottiene la matrice:

$$P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

mediante la quale è possibile ottenere:

$$\begin{pmatrix} 4 & 1 & -2 \\ 1 & -1 & 5 \\ 2 & 4 & -2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 1 & -1 & 5 \\ 4 & 1 & -2 \end{pmatrix} = P_1 A$$

cioè la fase di pivoting è equivalente a moltiplicare a sinistra la matrice A per la matrice P_1. La successiva fase di trasformazione della matrice produce poi la matrice:

$$A^{(1)} = M_1 P_1 A$$

- Analogamente al passo 2: se si effettua uno scambio tra la II e III riga della matrice I si ottiene:

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

mediante la quale è possibile ottenere:

$$\begin{pmatrix} 4 & 1 & -2 \\ 0 & \frac{7}{2} & -1 \\ 0 & -\frac{5}{4} & \frac{11}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & -\frac{5}{4} & \frac{11}{2} \\ 0 & \frac{7}{2} & -1 \end{pmatrix} = P_2 A^{(1)}$$

cioè la fase di pivoting è equivalente a moltiplicare a sinistra la matrice $A^{(1)} = M_1 P_1 A$ per la matrice P_2. La successiva fase di trasformazione della matrice produce poi la matrice:

$$U = A^{(2)} = M_2 P_2 A^{(1)} = M_2 P_2 M_1 P_1 A = M_2 P_2 M_1 P_2^T P_2 P_1 A$$

Posto quindi $P = P_2 P_1$ si ha:

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

cioé la matrice P si ottiene applicando a I tutti gli scambi di righe effettuati durante l'algoritmo di eliminazione di Gauss.

♣

In generale sussiste il

Teorema 2.3. *Data una matrice A non singolare, l'algoritmo di eliminazione di Gauss con pivoting parziale calcola le matrici L e U tali che:*

$$PA = LU$$

dove:

$$P = P_{n-1} \cdots P_1$$
$$\overline{M}_k = P_{n-1} \cdots P_{k+1} M_k P_{k+1} \cdots P_{n-1} \quad per \quad k \leq n-2$$
$$\overline{M}_{n-1} = M_{n-1}$$
$$L = \overline{M}_1^{-1} \cdot \overline{M}_2^{-1} \cdots \overline{M}_{n-1}^{-1}$$
$$U = \overline{M}_{n-1} \cdot \overline{M}_{n-2} \cdots \overline{M}_1 A$$

Dimostrazione: posto $A^{(0)} = A$, al generico passo k dell'algoritmo di eliminazione di Gauss si ha:

$$A^{(k)} = M_k P_k A^{(k-1)} \quad con \quad k = 1, \cdots, n-1$$

dove P_k e M_k sono le matrici che effettuano la fase di pivoting e di trasformazione della matrice attiva. All'ultimo passo si ha quindi:

$$U = A^{(n-1)} = M_{n-1} P_{n-1} M_{n-2} P_{n-2} \cdots M_2 P_2 M_1 P_1 A \tag{2.16}$$

Ora, poiché $\forall k$ si ha $P_k^T P_k = I$ è possibile scrivere:

$$\begin{aligned} M_{n-2} &= M_{n-2} P_{n-1}^T P_{n-1} \\ M_{n-3} &= M_{n-3} P_{n-2}^T P_{n-1}^T P_{n-1} P_{n-2} \\ &\vdots \\ M_k &= M_k P_{k+1}^T \cdots P_{n-1}^T P_{n-1} \cdots P_{k+1} \quad k \geq 1 \end{aligned}$$

e quindi dalla (2.16) si ottiene:

$$U = M_{n-1} P_{n-1} M_{n-2} P_{n-1}^T P_{n-1} P_{n-2} M_{n-3} P_{n-2}^T P_{n-1} \cdots P_{n-1} P_{n-2} \cdots P_1 A$$

Posto, infine:

$$P = P_{n-1} P_{n-2} \cdots P_1 \qquad \overline{M}_k = P_{n-1} \cdots P_{k+1} M_k P_{k+1}^T \cdots P_{n-1}^T$$

si ha la tesi. ■

Poiché $PA = LU$, una volta calcolati, mediante l'algoritmo di eliminazione di Gauss con pivoting parziale, i fattori L ed U della matrice PA si ha

$$LUx = (PA)x = P(Ax) = Pb$$

ovvero prima di procedere nella applicazione della *back* e *forward-substitution* per il calcolo di x è necessario effettuare sul vettore b gli stessi scambi effettuati sulla matrice A.

♣ **Esempio 2.21.** Nell'esempio 2.18, considerato

$$b = (6 \;\; 0 \;\; 2)^T;$$

- al passo 1 viene effettuato uno scambio tra la I e III riga di A. È necessario, quindi operare tale scambio sul vettore b ottenendo:

$$(2 \;\; 0 \;\; 6)^T;$$

- al passo 2 viene effettuato uno scambio tra la II e III riga della matrice ottenuta dopo il passo 1. Tale scambio va operato anche sul vettore dei termini noti e, quindi si ha il vettore:

$$(2 \;\; 6 \;\; 0)^T.$$

Tale vettore si può ottenere anche moltiplicando la matrice di permutazione P per il vettore b. Infatti:

$$Pb = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 6 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \\ 0 \end{pmatrix}$$

♣

Pertanto per risolvere il sistema $Ax = b$ mediante l'algoritmo di eliminazione di Gauss con pivoting parziale:

1. si applica l'algoritmo di eliminazione di Gauss con pivoting parziale alla matrice A (Procedura 2.10) e si ottengono i fattori L ed U della matrice PA, dove P è la matrice di permutazione che si ottiene dalla matrice identica effettuando su essa gli scambi di righe che sono derivati dall'applicazione del pivoting parziale;

2. si effettuano sul vettore b gli stessi scambi (moltiplicando P per b);

3. si risolve il sistema $LUx = Pb$ mediante *forward* e *back substitution*.

2.6.1 Aspetti implementativi

Si è visto, nel paragrafo precedente, che, per operare sul vettore b dei termini noti gli scambi effettuati durante il processo di eliminazione con pivoting sulle righe, bisogna conoscere la matrice P. In realtà *non è necessario costruire* P, ma è sufficiente costruire un vettore che *ricordi gli scambi effettuati*. Come si costruisce tale vettore?

♣ **Esempio 2.22.** Sia

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

Si costruisca un vettore, indicato con *ipiv*, di dimensione 4, che contiene inizialmente gli indici di riga in ordine naturale, ovvero:

$$ipiv_i = i, \quad i = 1, \ldots, 4$$

$$A = \begin{pmatrix} 1 & 3 & 5 & 7 \\ -1 & 4 & 0 & 2 \\ 8 & 3 & 0 & 5 \\ 4 & 2 & 0 & 1 \end{pmatrix} \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = ipiv.$$

Se si opera uno scambio tra la II e IV riga di A, tale scambio è *"memorizzato"* in *ipiv*, scambiando il contenuto di $ipiv_2$ ed $ipiv_4$:

$$\begin{pmatrix} 1 & 3 & 5 & 7 \\ 4 & 2 & 0 & 1 \\ 8 & 3 & 0 & 5 \\ -1 & 4 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 4 \\ 3 \\ 2 \end{pmatrix} = ipiv.$$

Analogamente, se si opera su tale matrice uno scambio tra la II e III riga, esso è memorizzato in *ipiv*

scambiando il contenuto di $ipiv_2$ ed $ipiv_3$:

$$\begin{pmatrix} 1 & 3 & 5 & 7 \\ 8 & 3 & 0 & 5 \\ 4 & 2 & 0 & 1 \\ -1 & 4 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 3 \\ 4 \\ 2 \end{pmatrix} = ipiv$$

Verificando il contenuto degli elementi del vettore *ipiv* è possibile stabilire come è stato alterato l'ordine delle righe di A (nell'esempio si ha che la I riga è rimasta al suo posto, la II riga di A costituisce ora la IV, la III riga di A è ora la II e la IV è la III).
Il vettore *ipiv* è un *vettore di puntatori*, nel senso che l'elemento di posto i di *ipiv* contiene (*"punta a"*) l'indice della riga di A che ora costituisce la i-ma riga ($ipiv_i = j \Longrightarrow$ la $j-$ma riga della matrice originaria A è ora la $i-$ma riga).

Una volta costruito il vettore *ipiv*, durante il processo di eliminazione con pivoting sulla matrice A, per effettuare gli opportuni scambi sul vettore b dei termini noti, basta *"consultarlo"*.
Sia quindi:

$$b = \begin{pmatrix} 3.4 \\ 0 \\ 1.5 \\ 8 \end{pmatrix}; \ ipiv = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 2 \end{pmatrix}$$

di conseguenza per effettuare gli opportuni scambi su b, bisogna spostare la III componente al posto della II, la II al posto della IV e la IV al posto della III, ottenendo:

$$\begin{pmatrix} 3.4 \\ 1.5 \\ 8 \\ 0 \end{pmatrix}$$

♣

Il vettore *ipiv* è un *vettore di puntatori*, nel senso che l'elemento di posto i di *ipiv* contiene (*"punta a"*) l'indice della riga di A che ora costituisce la i-ma riga ($ipiv_i = j \Longrightarrow$ la $j-$ma riga della matrice originaria A è ora la $i-$ma riga).

L'utilizzo che si puó fare del vettore *ipiv* nell'ambito di una procedura per la risoluzione di un sistema lineare $Ax = b$ basata sull'algoritmo di fattorizzazione LU con pivoting parziale e di *back* e *forward-substitution* è in realtà piú ampio. Infatti, oltre ad essere necessario per effettuare gli opportuni scambi sul vettore b, esso puó essere utilizzato anche **per non scambiare fisicamente** le righe di A e gli elementi di b.

♣ **Esempio 2.23.**

$$b = \begin{pmatrix} 2.4 \\ 0 \\ 6 \\ 5 \end{pmatrix}; \ P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

e quindi:

$$Pb = \begin{pmatrix} 0 \\ 6 \\ 2.4 \\ 5 \end{pmatrix}$$

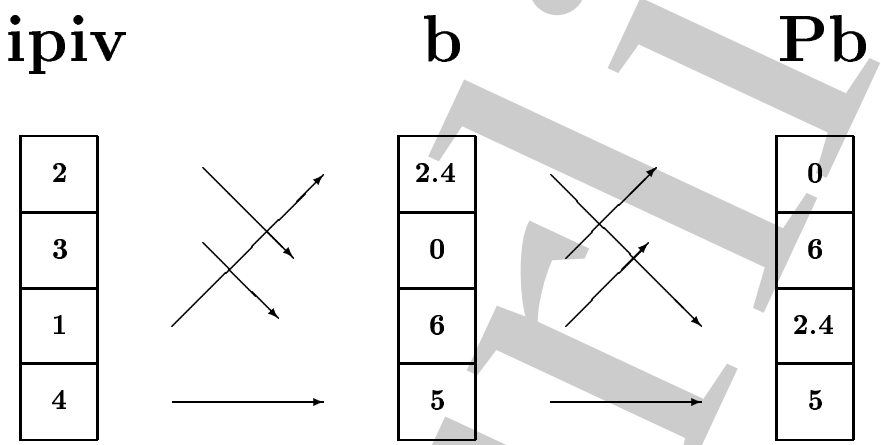

Figura 2.17: Utilizzo dell'array ipiv per l'accesso alle componenti di b.

Il vettore $ipiv$, che memorizza gli scambi operati su b, è:

$$ipiv = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 4 \end{pmatrix}$$

Si osservi che

- I componente di $Pb = 0$ e $b(ipiv(1)) = b_2 = 0$;
- II componente di $Pb = 6$ e $b(ipiv(2)) = b_3 = 6$;
- III componente di $Pb = 2.4$ e $b(ipiv(3)) = b_1 = 2.4$;
- IV componente di $Pb = 5$ e $b(ipiv(4)) = b_4 = 5$.

Ciò significa che:

$$b(ipiv(i)) = (Pb)_i, \quad i = 1, 2, \ldots, 4,$$

ovvero $b(ipiv(i))$ è il valore contenuto nell'elemento di posto i del vettore Pb.

♣

Quindi, per accedere correttamente alle componenti del vettore Pb, anziché scambiare fisicamente gli elementi di b, si possono utilizzare le componenti del vettore $ipiv$ come indici del vettore b (Procedura 2.11), ovvero si può accedere alle componenti del vettore Pb attraverso b:

$$b(ipiv(i)) = (Pb)_i, \quad i = 1, \ldots, n.$$

In maniera analoga si puó operare per la matrice A come mostrato nella Procedura 2.10.

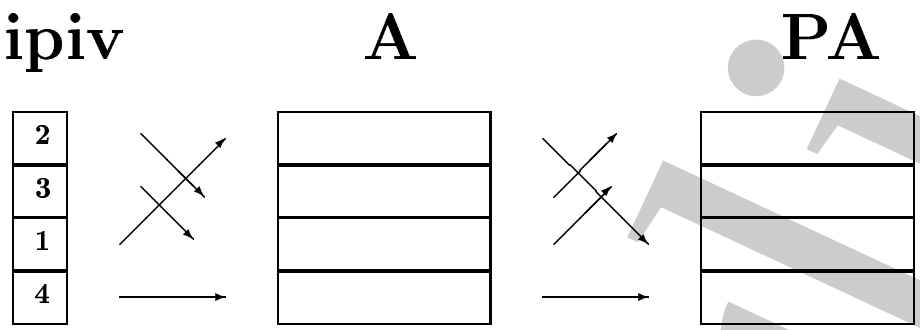

Figura 2.18: Utilizzo dell'array ipiv per l'accesso alle componenti di A.

♣ **Esempio 2.24.**

$$A = \begin{pmatrix} 1 & 3 & 5 & 7 \\ -1 & 4 & 0 & 2 \\ 8 & 3 & 0 & 5 \\ 4 & 2 & 0 & 1 \end{pmatrix}; \quad P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

e quindi:

$$PA = \begin{pmatrix} -1 & 4 & 0 & 2 \\ 8 & 3 & 0 & 5 \\ 1 & 3 & 5 & 7 \\ 4 & 2 & 0 & 1 \end{pmatrix}$$

da cui:

$$ipiv = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 4 \end{pmatrix}$$

Osservando la Figura 2.24 si nota che gli elementi

$$a(ipiv(i), j), \quad j = 1, 2, \ldots, 4,$$

sono gli elementi della $i-$ma riga di PA. Di conseguenza per accedere correttamente alle righe (e quindi agli elementi) di PA , anziché scambiare fisicamente la righe di A, basta utilizzare le componenti del vettore *ipiv* come indici di riga della matrice A.

♣

In conclusione,

- nell'algoritmo di fattorizzazione $PA = LU$ con pivoting parziale (Procedura 2.10) l'utilizzo di un vettore di puntatori che memorizzi, come illustrato, gli scambi richiesti sulla matrice A consente di:

  1. non costruire la matrice P per memorizzare gli scambi;

2. non effettuare fisicamente lo scambio di righe.

- nell'algoritmo di forward substitution per la risoluzione di $Ly = Pb$ (Procedura 2.11) l'utilizzo del vettore di puntatori consente di:

1. conoscere gli scambi da effettuare su b;

2. non effettuare fisicamente gli scambi.

Si osservi infine che, data l'equivalenza dell'algoritmo di eliminazione di Gauss con pivoting parziale con la fase di fattorizzazione $PA = LU$ seguita dalla risoluzione di $Ly = Pb$, nell'algoritmo di *back-substitution* per la risoluzione di $Ux = y$ non è necessario l'utilizzo dell'array *ipiv*, in quanto la fase di *forward substitution* già ha predisposto gli elementi di y nell'ordine opportuno.

procedure paeqlu (**in:** $a, ipiv, n$; **out:** $a, ipiv$)

/# SCOPO: fattorizzazione LU con pivoting parziale.

/# SPECIFICHE DEI PARAMETRI:

/# PARAMETRI DI INPUT:

var:	n	: **intero**	{ *dimensione del sistema* }
var:	$ipiv(n)$	: **intero**	{ *vettore delle permutazioni* }
var:	$a(n, n)$	: **reale**	{ *matrice del sistema* }
			{ *da fattorizzare* }

Procedura 2.10: Algoritmo di fattorizzazione $PA = LU$ con pivoting parziale - continua

```
/# PARAMETRI DI OUTPUT:
var:  ipiv(n)    : intero   { vettore delle permutazioni }
                            { effettuate }
var:  a(n,n)     : reale    { matrice del sistema fattorizzata }

/# VARIABILI LOCALI:
var: i,j,k,t,r   : interi
var: mas         : reale

/# INIZIO ISTRUZIONI:
for i = 1 to n                              {inizializzazione array ipiv}
   ipiv(i) := i
end for
for k = 1 to n - 1                                    {ciclo sui passi}
   mas := abs(a(ipiv(k),k))                                  {pivoting}
      r := k
      for i = k + 1 to n                          {ricerca del massimo}
         if (abs(a(ipiv(i),k)) > mas) then
            mas := abs(a(ipiv(i),k))
            r := i
         end if
      endfor
      if (r ≠ k) then
         t := ipiv(r)                        {scambio virtuale delle righe}
         ipiv(r) := ipiv(k)
         ipiv(k) := t
      endif
      for i = k + 1 to n                {calcolo degli elementi di L e U}
         a(ipiv(i),k) := a(ipiv(i),k)/a(ipiv(k),k)
         for j = k + 1 to n
            a(ipiv(i),j) := a(ipiv(i),j) - a(ipiv(i),k) * a(ipiv(k),j)
         end for
      end for
   end for
   end paeqlu
```

Procedura 2.10: Algoritmo di fattorizzazione $PA = LU$ con pivoting parziale - fine

```
procedure lyeqpb (in: l, b, n, ipiv ; out: x, comp)

/# SCOPO: Risoluzione di un sistema triangolare inferiore.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n        : intero   { dimensione del sistema }
var: ipiv(n)  : intero   { vettore delle permutazioni }
var: l(n,n)   : reale    { matrice del sistema }
                         { da fattorizzare }
var: b(n)     : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: y(n)     : reale    { vettore delle soluzioni }
var: comp     : logical  { è vera se il sistema è }
              :          { compatibile }
                         { falsa se il sistema è, }
                         { o potrebbe essere, incompatibile }
                         { o compatibile ma indeterminato }
/# VARIABILI LOCALI:
var: i, j     : interi
var: r        : reale

/# INIZIO ISTRUZIONI:
comp := true                                  {sistema compatibile}
if (l(ipiv(1),1) ≠ 0) then
   y(1) = b(ipiv(1))/l(ipiv(1),1)             {calcolo di y(1)}
else if (b(ipiv(1)) = 0) then
   y(1) := 0      {il sistema potrebbe essere incompatibile}
   comp := false  {o compatibile ma indeterminato}
else
   comp := false                              {sistema incompatibile}
end if
i := 2
```

Procedura 2.11: Forward-substitution per $Ly = Pb$ - continua

```
while (i ≤ n and comp)                          {ciclo sugli y(i)}
   r := b(ipiv(i))
   for j = 1 to i − 1                           {calcolo del resto}
      r := r − l(ipiv(i), j) ∗ y(j)
   end for
   if (l(ipiv(i), i) ≠ 0) then
      y(i) := r/l(ipiv(i), i)                   {calcolo di y(i)}
   else if (r = 0) then
   y(i) := 0 {il sistema potrebbe essere incompatibile}
   comp := false    {o compatibile ma indeterminato}
   else
      comp := false                             {sistema incompatibile}
   end if
   i := i + 1
end while
end lyeqpb
```

Procedura 2.11: Forward-substitution per $Ly = Pb$ - fine

2.7 Condizionamento di sistemi lineari

È noto che il *condizionamento* è una caratteristica del problema e non dell'algoritmo risolutivo. Pertanto, la risoluzione di un sistema mal condizionato risulta inaccurata anche utilizzando algoritmi stabili.
È chiaro che, essendo interessati alla risoluzione di sistemi lineari mediante calcolatore, è necessario analizzare il condizionamento del sistema in esame, in quanto i dati sono affetti almeno dagli errori di round-off.

♣ **Esempio 2.25.** Il sistema di equazioni lineari:

$$\begin{cases} 2.1x + 3.5y = 8 \\ 4.19x + 7y = 15, \end{cases} \tag{2.17}$$

ha soluzione $x = 100$, $y = -57.714\cdots$. Si consideri ora, il sistema:

$$\begin{cases} 2.1x + 3.5y = 8 \\ 4.192x + 7y = 15, \end{cases} \tag{2.18}$$

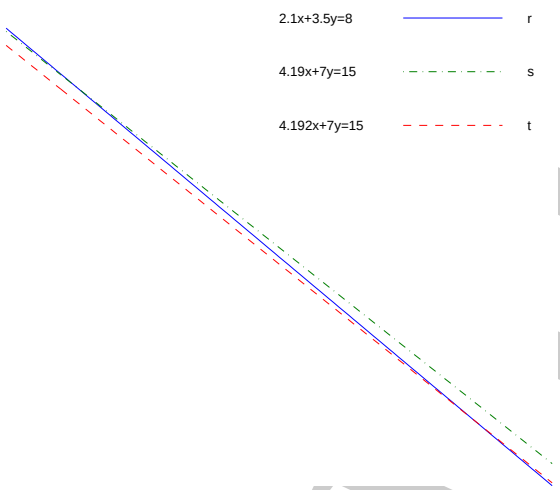

Figura 2.19: Rappresentazione grafica del sistema dell'esempio 2.25

che può essere riguardato come una perturbazione del sistema (2.17), essendo stato introdotto un errore sulla quarta cifra significativa del coefficiente della x nella seconda equazione (un errore relativo di 10^{-3}). La soluzione del sistema (2.18) è $x = 125$, $y = -72.714....$ Se si considera la soluzione di (2.18) come la soluzione del sistema (2.17) perturbato, si osserva che l'errore introdotto ha condotto ad un soluzione completamente diversa. Il motivo per cui si è verificato tale fenomeno si desume facilmente dalla rappresentazione geometrica dei due sistemi (Figura 2.19)

La perturbazione introdotta corrisponde ad una modifica della inclinazione della retta rappresentata dalla seconda equazione del sistema (2.17). Tale modifica provoca una grande perturbazione sulla soluzione in quanto le due rette relative al sistema (2.17) sono *"quasi parallele"* e, quindi una lieve perturbazione del coefficiente angolare di una delle due rette determina un notevole spostamento del punto di intersezione. Il sistema (2.17), pertanto, è tale che ad una piccola variazione nei dati corrisponde una grande perturbazione nella soluzione. Il sistema (2.17) è un sistema mal condizionato.

♣

♣ **Esempio 2.26.** Il sistema:

$$\begin{cases} x + y = 2 \\ x + 1.0001y = 2.0001 \end{cases} \tag{2.19}$$

ha soluzione $x = 1$, $y = 1$. Utilizzando un sistema artimetico floating-point con base $\beta = 10$ e precisione $t = 4$, si ottiene il sistema:

$$\begin{cases} x + y = 2 \\ x + y = 2, \end{cases} \tag{2.20}$$

che è singolare!
Quindi l'errore di round-off nei dati ha alterato il coefficiente angolare della retta rappresentata dalla seconda equazione del sistema (2.19) in maniera tale da rendere coincidenti le due rette (Figura 2.20) e di conseguenza, singolare il relativo sistema.

♣

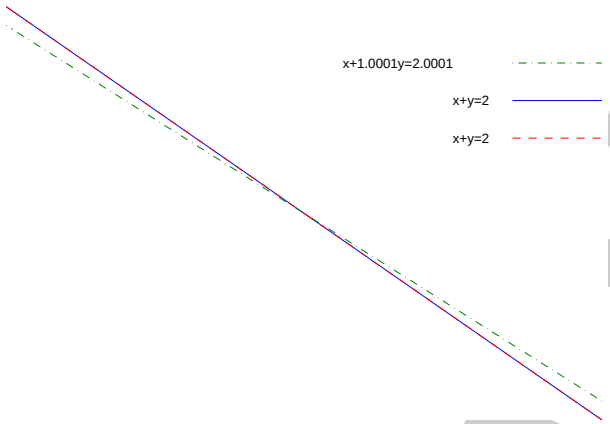

Figura 2.20: Rappresentazione grafica del sistema dell'esempio 2.26

Come si può stimare quantitativamente il condizionamento di un sistema lineare? Dato un sistema lineare:

$$Ax = b, \tag{2.21}$$

$$A = (a_{i,j}), \quad i,j = 1,\ldots,n, \quad b = (b_i) \quad i = 1,\ldots,n,$$

i coefficienti ed il termine noto sono affetti almeno dall'errore di round-off di rappresentazione. Ciò vuol dire che in luogo del sistema assegnato si risolve *sempre* un sistema perturbato; ad esempio, utilizzando il sistema aritmetico indicato nell'esempio 2.26, in luogo del sistema (2.19) si risolve il sistema (2.20).

In generale, si risolve un sistema con coefficienti perturbati:

$$a_{i,j} + \Delta a_{i,j}, \qquad i,j = 1,\ldots,n,$$

con $\Delta a_{i,j}$ = errore di round-off nella rappresentazione di $a_{i,j}$ e componenti del termine noto perturbate:

$$b_i + \Delta b_i, \qquad i = 1,\ldots,n,$$

con Δb_i = errore di round-off nella rappresentazione di b_i.
Naturalmente anche la soluzione, x, del sistema (2.21), risulterà perturbata. Pertanto, indicata con:

$$\Delta A = (\Delta a_{i,j})_{i,j=1,\ldots,n}$$

la matrice delle perturbazioni relativa ai coefficienti del sistema, con:

$$\Delta b = (\Delta b_i)_{i=1,\dots,n}$$

il vettore delle perturbazioni relative alle componenti del termine noto e, detta Δx la perturbazione introdotta sulla soluzione, in luogo del sistema (2.21) si risolverà il sistema perturbato:

$$(A + \Delta A)(x + \Delta x) = b + \Delta b \tag{2.22}$$

♣ **Esempio 2.27.** Utilizzando un sistema aritmetico floating-point con base $\beta = 10$ e precisione relativa $t = 3$, con il metodo dell'arrotondamento, in luogo del sistema

$$\begin{cases} 4.124x & + & 3.02y & = & 5.246 \\ 1.34x & + & 2.563y & = & 7.31, \end{cases}$$

si risolve il sistema:

$$\begin{cases} 4.12x & + & 3.02y & = & 5.25 \\ 1.34x & + & 2.56y & = & 7.31 \end{cases}$$

Pertanto,

$$A = \begin{pmatrix} 4.124 & 3.02 \\ 1.34 & 2.563 \end{pmatrix}; \quad b = \begin{pmatrix} 5.246 \\ 7.31 \end{pmatrix}$$

e

$$\Delta A = \begin{pmatrix} -0.004 & 0. \\ 0. & -0.003 \end{pmatrix}; \quad \Delta b = \begin{pmatrix} 0.004 \\ 0. \end{pmatrix}$$

♣

Prima di risolvere il sistema perturbato è necessario valutare come gli errori ΔA e Δb influenzeranno la soluzione, ovvero determinare una stima dell'errore Δx in funzione degli errori nei dati.

2.7.1 Indice di Condizionamento

Per stimare il condizionamento è necessario valutare l'errore relativo nella soluzione del sistema:

$$\frac{\|\Delta x\|}{\|x\|},$$

in funzione degli errori relativi nei dati:

$$\frac{\|\Delta A\|}{\|A\|} \quad e \quad \frac{\|\Delta b\|}{\|b\|},$$

cioè bisogna determinare quanto l'errore nei dati si amplifichi nella soluzione.

Inizialmente, per semplicità, si supponga che ci sia una perturbazione solo sul termine noto del sistema, ovvero anziché

$$Ax = b, \tag{2.23}$$

si sta risolvendo il sistema perturbato:

$$A(x + \Delta x) = b + \Delta b, \tag{2.24}$$

nella ipotesi che Δb sia *"piccolo"*[13] rispetto a b.
Per stimare Δx in funzione di Δb, si osservi che da (2.24) discende che:

$$Ax + A\Delta x = b + \Delta b, \quad \text{con } Ax = b$$

e quindi:

$$A\Delta x = \Delta b$$

da cui:

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

nell'ipotesi che A sia invertibile.
Introdotta, dunque, una norma matriciale compatibile con una vettoriale (cfr. Definizione A.5), la soluzione soddisfa la disuguaglianza

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

Questa relazione fornisce informazioni sul *condizionamento assoluto* del problema; è possibile, inoltre, ottenere un'espressione analoga, per il *condizionamento relativo*. Valutando l'errore relativo nella soluzione si ha:

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

dove l'ultima eguaglianza si ottiene tenendo presente che:

$$\|b\| = \|Ax\| \leq \|A\|\|x\|$$

da cui:

$$\frac{1}{\|x\|} \leq \frac{1}{\|b\|/\|A\|}.$$

In conclusione si è ottenuto che:

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

Quindi la quantità $\|A\|\|A^{-1}\|$ stima l'amplificazione dell'errore nei dati sulla soluzione.

[13] Per *"piccolo"* si intende dell'ordine dell'errore di round-off che si commette nel rappresentare b in un assegnato sistema aritmetico floating-point a precisione finita.

♣ **Esempio 2.28.** Il sistema di equazioni lineari:

$$\left\{ \begin{array}{rcrcl} x_1 & + & .99x_2 & = & 1.989903 \\ .99x_1 & + & .98x_2 & = & 1.970106 \end{array} \right. \tag{2.25}$$

ha soluzione $x = (3.0000\ ,\ -1.0203)^T$.
Utilizzando un sistema aritmetico floating-point con base $\beta = 10$ e precisione relativa $t = 3$ con il metodo di arrotondamento, il sistema (2.25) diventa

$$\left\{ \begin{array}{rcrcl} x_1 & + & .99x_2 & = & 1.99 \\ .99x_1 & + & .98x_2 & = & 1.97, \end{array} \right. \tag{2.26}$$

la cui soluzione è $x + \Delta x = (1\ ,\ 1)^T$.
Pertanto, una perturbazione

$$\Delta b = \begin{pmatrix} -0.000097 \\ +0.000106 \end{pmatrix}$$

sul termine noto, dovuta alla sua rappresentazione nel sistema aritmetico considerato, ha condotto ad una soluzione completamente diversa. In particolare,

$$\Delta x = \begin{pmatrix} 2.0000 \\ 2.0203 \end{pmatrix}$$

Si osservi che:

$$\|\Delta x\|_\infty = 2.0203 \ \ e \ \ \|x\|_\infty = 3.0000$$

da cui:

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} = \frac{2.0203}{3.0000} \simeq 0.67 \ \text{ arrotondato a 2 cifre}$$

Inoltre,

$$\|\Delta b\|_\infty = .106 \times 10^{-3} \ \ e \ \ \|b\|_\infty = 1.989903$$

e quindi

$$\frac{\|\Delta b\|_\infty}{\|b\|_\infty} = \frac{.106 \times 10^{-3}}{1.989903} \simeq 0.53 \times 10^{-4} \ \text{ arrotondato a 2 cifre.}$$

Da ció discende che

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} \leq 2 \times 10^4 \frac{\|\Delta b\|_\infty}{\|b\|_\infty}$$

ovvero il fattore di amplificazione dell'errore nei dati sulla soluzione è dell'ordine di 10^4.
D'altra parte, poiché

$$A^{-1} = \begin{pmatrix} -9800 & 9900 \\ 9900 & -10000 \end{pmatrix}$$

si ha che:

$$\|A\|_\infty = 1.99 \ \ e \ \ \|A^{-1}\|_\infty = 19900$$

da cui:

$$\|A\|\|A^{-1}\| = 1.99 \times 19900 \simeq 4 \times 10^4$$

si ottiene cioè, a meno di una costante, il fattore di amplificazione dell'errore nei dati sulla soluzione.
♣

Sussiste la seguente:

Definizione 2.8. (Indice di condizionamento relativo)
La quantità

$$\mu(A) = \|A\|\|A^{-1}\|, \tag{2.27}$$

è detta **indice di condizionamento relativo** *del sistema di equazioni* $Ax = b$.

Dall'espressione di $\mu(A)$, si deduce che il condizionamento di un sistema lineare è una proprietà intrinseca della matrice dei coefficienti (e non dipende dagli errori di cui è affetto il vettore dei termini noti). Inoltre, è sempre $\mu(A) \geq 1$, in quanto:

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

L'indice di condizionamento misura il massimo fattore di amplificazione dell'errore relativo sulla soluzione, rispetto all'errore relativo sui dati. Un sistema di equazioni è, quindi, *mal condizionato* se $\mu(A) >> 1$, mentre è *ben condizionato* se $\mu(A) \simeq 1$. Tale definizione richiede alcune considerazioni. Innanzitutto si deve osservare che, il valore dell'indice di condizionamento è, ovviamente, influenzato dalla norma matriciale adottata e, quindi, dal modo in cui si misura l'errore. Si osserva, a tal proposito, che, posto $\mu_\alpha(A) = \|A\|_\alpha\|A^{-1}\|_\alpha$ con $\alpha = 1, 2, \infty$, sussitono le seguenti relazioni:

$$\frac{1}{n}\mu_2(A) \leq \mu_1(A) \leq n\mu_2(A)$$

$$\frac{1}{n}\mu_\infty(A) \leq \mu_2(A) \leq n\mu_\infty(A)$$

$$\frac{1}{n^2}\mu_1(A) \leq \mu_\infty(A) \leq n^2\mu_1(A)$$

da cui si ricava che, se n è grande (ad esempio $n = 10^6$) e $\mu_\infty(A) = O(n)$, il sistema risulterà mal condizionato utilizzando $\mu_\infty(A)$, mentre potrebbe risultare ben condizionato utilizzando $\mu_2(A)$; viceversa, peró, se n è piccolo rispetto a $\mu(A)$, si verifica che, se $\|A\|\|A^{-1}\|$ è grande in una norma, lo sarà in tutte le altre. Per tale motivo si è soliti anche definire mal condizionato un sistema di equazioni in cui è almeno $\mu(A) > O(n)$.

L'ipotesi che il solo termine noto sia perturbato, è, in realtà, poco realistica, dato che la stessa matrice A può essere affetta da errori (in generale è sempre affetta almeno dagli errori di round-off di rappresentazione). È tuttavia possibile ottenere una relazione per il condizionamento anche nel caso più generale, in cui si verifica che l'amplificazione dell'errore presente sui dati continua ad essere influenzata, in modo essenziale, dall'indice di condizionamento $\mu(A)$, con A matrice dei coefficienti del sistema. Supponendo, allora, che i dati costituiti dai coefficienti e dal vettore dei termini noti, siano affetti da errore, per studiare il caso generale, è necessario dimostrare il seguente:

Lemma 2.1. *Sia $\|\cdot\|$ una norma matriciale submoltiplicativa (cioè $\|AB\| \leq \|A\|\|B\|$) e compatibile con una norma vettoriale (cioè $\|Ax\| \leq \|A\|\|x\| \quad \forall x \neq 0$). Se una matrice A è tale che $\|A\| < 1$, si ha:*

1. *$I + A$ è non singolare;*
2. *$(1 + \|A\|)^{-1} \leq \|(I + A)^{-1}\| \leq (1 - \|A\|)^{-1}$*

Dimostrazione Per dimostrare la 1. si consideri un vettore $x \neq 0$. Allora si ha:

$$\|(I + A)x\| = \|x + Ax\| \geq \|x\| - \|Ax\| \geq$$

$$\|x\| - \|x\|\|A\| = \|x\|(1 - \|A\|) > 0$$

Ciò vuol dire che il vettore $(I + A)x \neq 0 \; \forall x \neq 0$, e quindi il sistema di equazioni $(I + A)x = 0$ ammette solo la soluzione banale e quindi la matrice $(I + A)$ è non singolare.

Per dimostrare la 2. si osservi che:

$$1 = \|I\| = \|(I + A)(I + A)^{-1}\| \leq \|I + A\| \cdot \|(I + A)^{-1}\| \leq (1 + \|A\|) \cdot \|(I + A)^{-1}\|$$

da cui si ha:

$$\|(I + A)^{-1}\| \geq \frac{1}{1 + \|A\|}$$

e quindi la prima disuguaglianza è dimostrata. Inoltre si ha:

$$I = (I + A)(I + A)^{-1} = (I + A)^{-1} + A(I + A)^{-1}$$

da cui:

$$(I + A)^{-1} = I - A(I + A)^{-1}$$

Passando alle norme:

$$\|(I + A)^{-1}\| = \|I - A(I + A)^{-1}\| \leq 1 + \|A\| \cdot \|(I + A)^{-1}\|$$

e quindi:

$$\|(I + A)^{-1}\|(1 - \|A\|) \leq 1$$

Ma, poiché l'ipotesi $\|A\| \leq 1 \Longrightarrow (1 - \|A\|) > 0$, si ha:

$$\|(I + A)^{-1}\| \leq \frac{1}{1 - \|A\|}$$

e quindi anche la seconda disuguaglianza della 2. è dimostrata. ■

Si è ora in grado di dimostrare il teorema seguente, che fornisce una stima dell'indice di condizionamento relativo, per un sistema lineare.

Teorema 2.4. [Teorema del condizionamento]
Sia $\|\cdot\|$ una norma matriciale submoltiplicativa (cioè $\|AB\| \leq \|A\|\|B\|$) e compatibile con una norma vettoriale (cioè $\|Ax\| \leq \|A\|\|x\| \quad \forall x \neq 0$). Sia inoltre il sistema $Ax = b$ (con A non singolare) e si consideri il sistema perturbato:

$$(A + \Delta A)(x + \Delta x) = (b + \Delta b) \tag{2.28}$$

Se

$$\|\Delta A\| < \frac{1}{\|A^{-1}\|} \tag{2.29}$$

posto $\mu(A) = \|A\|\|A^{-1}\|$ si ha:

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\mu(A)}{1 - \mu(A)\frac{\|\Delta A\|}{\|A\|}} \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right) \tag{2.30}$$

Dimostrazione Posto $C = A^{-1}\Delta A$ si ha:

$$\|C\| = \|A^{-1}\Delta A\| \leq \|A^{-1}\| \cdot \|\Delta A\| < 1$$

per cui applicando il Lemma alla matrice C si ha che la matrice $(I+C) = (I+A^{-1}\Delta A)$ è non singolare. Poiché inoltre $(A + \Delta A) = A(I + A^{-1}\Delta A)$ si ha che anche $(A + \Delta A)$ è non singolare. Dal lemma si ha anche che:

$$\|(I + C)^{-1}\| = \|(I + A^{-1}\Delta A)^{-1}\| \leq \frac{1}{1 - \|A^{-1}\Delta A\|} \leq \frac{1}{1 - \|A^{-1}\| \cdot \|\Delta A\|} \tag{2.31}$$

Moltiplicando quindi per A^{-1} il sistema perturbato (2.28) si ha:

$$A^{-1}(A + \Delta A)(x + \Delta x) = (I + A^{-1}\Delta A)(x + \Delta x) = A^{-1}b + A^{-1}\Delta b$$

da cui:

$$(I + A^{-1}\Delta A)x + (I + A^{-1}\Delta A)\Delta x = x + A^{-1}\Delta b$$

e quindi:

$$(I + A^{-1}\Delta A)\Delta x = x + A^{-1}\Delta b - x - A^{-1}\Delta A x$$

Isolando a primo membro Δx e passando alle norme si ha:

$$\|\Delta x\| \leq \|(I + A^{-1}\Delta A)^{-1}\| \cdot \|A^{-1}\|(\|\Delta b\| + \|\Delta A\| \cdot \|x\|)$$

da cui, utilizzando anche la (2.31), si ha

$$\|\Delta x\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\| \cdot \|\Delta A\|}(\|\Delta b\| + \|\Delta A\| \cdot \|x\|)$$

e, dunque,

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

Ma, poiché da $Ax = b$ si ha $\|x\| \geq \|b\|/\|A\|$, dalla precedente espressione, moltiplicando e dividendo per $\|A\|$, si ottiene:

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

e quindi, posto $\mu(A) = \|A\| \cdot \|A^{-1}\|$, si ha la tesi. ■

Si osservi che da $\|A\|\|A^{-1}\| \geq 1$, l'ipotesi (2.29), implica che $\|\Delta A\| \leq \|A\|$, la quale è una ipotesi realistica, in quanto afferma che l'ordine di grandezza delle perturbazioni è inferiore all'ordine di grandezza degli elementi di A. Se cosí non fosse le perturbazioni altererebbero completamente i dati. Inoltre, la stessa ipotesi (2.29) garantisce che:

$$0 < \mu(A) \frac{\|\Delta A\|}{\|A\|} < 1$$

e quindi il denominatore al secondo membro della (2.30) sarà sempre strettamente positivo.

♣ **Esempio 2.29.** Considerata la matrice dei coefficienti del sistema nell'esempio 2.26

$$A = \begin{pmatrix} 1. & 1. \\ 1. & 1.0001 \end{pmatrix}$$

si ha che:

$$A^{-1} = \begin{pmatrix} 10001 & -10000 \\ -10000 & 10000 \end{pmatrix}$$

e quindi:

$$\|A\|_\infty = 2.0001 \;\; e \;\; \|A^{-1}\|_\infty = 20001$$

da cui:

$$\mu(A) \simeq 4 \times 10^4$$

ed il sistema (2.19), come visto, è mal condizionato.

Si osservi inoltre, che nel sistema aritmetico floating-point considerato nell'esempio 2.26, la matrice A diventa

$$A + \Delta A = \begin{pmatrix} 1. & 1. \\ 1. & 1. \end{pmatrix}$$

che è una matrice singolare. Ció sembra indicare che una matrice mal condizionata è una matrice *"quasi singolare"*, nel senso che una piccola perturbazione sui suoi elementi conduce ad una matrice singolare. Se si calcola ora la *"distanza"* della matrice A dalla matrice singolare $A + \Delta A$, ovvero l'errore relativo $\frac{\|\Delta A\|}{\|A\|}$, si ha che:

$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} = \frac{0.0001}{2.0001} \simeq 5 \times 10^{-5}$$

e, d'altra parte,

$$\frac{1}{\mu(A)} \simeq 2.5 \times 10^{-5}$$

♣

In generale, $1/\mu(A)$ è una stima della distanza della matrice A dalla matrice singolare piú vicina, cioè:

$$\frac{1}{\mu(A)} = \min \left\{ \frac{\|A - B\|}{\|A\|} \ : \ B \ \text{ singolare} \right\}$$

Pertanto quanto piú è grande l'indice di condizionamento $\mu(A)$ tanto piú la matrice A è vicina (in norma) ad una matrice singolare, con l'assunzione che una matrice singolare ha indice di condizionamento $\mu(A) = \infty$.

♣ **Esempio 2.30.** Si consideri la matrice A con la sua inversa:

$$A = \begin{pmatrix} 10^{-1} & 0 & \cdots & 0 \\ 0 & 10^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 10^{-1} \end{pmatrix} \quad \Longrightarrow \quad A^{-1} = \begin{pmatrix} 10 & 0 & \cdots & 0 \\ 0 & 10 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 10 \end{pmatrix}$$

Tale matrice ha determinante $det(A) = 10^{-n}$ e l'indice di condizionamento è $\mu(A) = 1$ e quindi è ben condizionata. ♣

In generale non c'è alcun legame tra indice di condizionamento e determinante di una matrice, nel senso che, avendo posto $\mu(A) = \infty$ per una matrice singolare (che ha $det(A) = 0$), si potrebbe essere portati a pensare che un determinante piccolo sia caratteristico delle matrici mal condizionate e che il determinante di una matrice sia legato, in qualche senso, all'inverso dell'indice di condionamento.

♣ **Esempio 2.31.** Si considerino i punti $x_i = i$ per $i = 1, 2, .., 10$ e si consideri la matrice di Vandermonde:

$$V = \begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^{9} \\ 1 & x_2 & \cdots & x_2^{9} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{10} & \cdots & x_{10}^{9} \end{pmatrix}$$

È noto che una matrice di Vandermonde è malcondizionata (in questo caso è $\mu(V) = 3.3064 \times 10^{12}$), mentre il determinante è $det(V) = +1.8... \times 10^{21}$.

♣

Stabilito che $\mu(A)$ permette di sapere di quanto l'errore nei dati si amplifica sulla soluzione, il problema che si pone, assegnata una matrice A, è calcolarne l'indice di condizionamento. Il calcolo di $\mu(A)$ coinvolge la valutazione di A^{-1}, che si è visto essere un problema di elevata complessità computazionale (o comunque di pari complessità rispetto alla risoluzione del sistema). Per tale motivo sono stati sviluppati algoritmi per stimare $\mu(A)$, che non richiedono il calcolo dell'inversa di A.
Elementi di software matematico basati su tali algoritmi sono presenti nelle piú importanti librerie di software matematico, quali *LAPACK, NAg e IMSL.*

2.8 Accuratezza della soluzione di un sistema di equazioni lineari

In questo paragrafo si vogliono approfondire alcuni aspetti correlati all'accuratezza della soluzione di sistemi di equazioni lineari di ordine n del tipo[14]:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots a_{1,n}x_n = b_1 \\ \vdots \qquad\qquad\qquad\qquad \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots a_{n,n}x_n = b_n \end{cases} \iff Ax = b \tag{2.32}$$

calcolata mediante la fattorizzazione $A = LU$ della matrice dei coefficienti seguita dagli algoritmi di *Forward Substitution* e di *Back Substitution.*

Ricordiamo che la fattorizzazione $A = LU$ e la successiva risoluzione del sistema triangolare inferiore $Ly = b$ mediante la *Forward Substitution* sono equivalenti all'algoritmo di eliminazione di Gauss, nel senso che, posto $A^{(0)} = A$ e $b^{(0)} = b$, l'algoritmo di eliminazione di Gauss genera una sequenza di sistemi equivalenti:

$$A^{(1)}x = b^{(1)}, \quad A^{(2)}x = b^{(2)}, \quad \cdots \quad A^{(n-1)}x = b^{(n-1)}$$

tali che $U = A^{(n-1)}$ e L è costituita da tutti i moltiplicatori calcolati nel corso dell'algoritmo di eliminazione di Gauss (ponendo inoltre $l_{i,i} = 1, \quad i = 1, .., n$). Si puó osservare inoltre che $b^{(n-1)}$ è la soluzione del sistema triangolare inferiore $Ly = b$.

Pertanto, al fine della valutazione dell'accuratezza della soluzione del sistema (2.32), analizzeremo la stabilità degli algoritmi di eliminazione di Gauss e di Forward e Back substitution, cioè come si propagano gli errori di round-off in tali algoritmi.

[14]Nel seguito, tutti i sistemi di equazioni considerati sono ben condizionati tranne dove specificato diversamente.

2.8.1 Analisi dell'errore di round-off nell'algoritmo di Gauss

♣ **Esempio 2.32.** La soluzione del sistema di equazioni lineari:

$$\begin{cases} 4.4409 \times 10^{-17} x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases}$$

è:

$$x_1 = 1.000\,000\,000\,000\,000\,044... \quad x_2 = 0.999\,999\,999\,999\,999\,955...$$

ma la soluzione calcolata utilizzando un calcolatore con aritmetica IEEE in doppia precisione, mediante:

- fattorizzazione $A = LU$ (mediante algoritmo di Gauss senza pivoting)
- risoluzione di $Ly = b$ (mediante forward Substitution)
- risoluzione di $Ux = y$ (mediante backward Substitution)

è:

$$\widehat{x}_1 = 0 \quad \widehat{x}_2 = 1$$

Tale soluzione è inaccettabile in quanto non verifica la seconda equazione. È possibile osservare inoltre che l'errore relativo commesso è:

$$\frac{||\Delta x||_\infty}{||x||_\infty} = \frac{\max(|\widehat{x}_1 - x_1|, |\widehat{x}_2 - x_2|)}{\max(|x_1|, |x_2|)} \simeq 1$$

♣

♣ **Esempio 2.33.** Si risolva il sistema di equazioni $Ax = b$, con:

$$A = \begin{pmatrix} 0.001 & 1 \\ 1 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{2.33}$$

nel sistema aritmetico floating-point, caratterizzato dai parametri:

$$\mathcal{F} = \{\beta = 10 \quad t = 3 \quad E_{min} = -6 \quad E_{max} = 6\}$$

il quale ha massima accuratezza relativa $u = 0.5 \times 10^{-2}$. La soluzione del sistema è:

$$x_1 = 1.0010... \quad x_2 = 0.999....$$

Fattorizzazione LU. L'algoritmo di eliminazione di Gauss per la fattorizzazione LU in questo caso esegue un solo passo. Eseguendo le operazioni nel sistema aritmetico considerato si ottiene:

- $\widehat{m}_{2,1} = fl(a^{(0)}_{2,1}/a^{(0)}_{1,1}) = (.100 \times 10^1)/(.100 \times 10^{-2}) = .100 \times 10^4$
- $\widehat{a}^{(1)}_{2,2} = fl(a^{(0)}_{2,2} - m_{2,1}a^{(0)}_{1,2}) = .100 \times 10^1 - (.100 \times 10^4) \times (.100 \times 10^1) =$
  $.100 \times 10^1 - .100 \times 10^4 = -.100 \times 10^4$

Si osservi che nella sottrazione presente nel calcolo di $\widehat{a}_{2,2}^{(1)}$ è stato commesso un errore relativo di round-off:

$$|\delta_s| = \frac{.100 \times 10^1}{.100 \times 10^4} = .100 \times 10^{-2}$$

Si ottiene in definitiva che:

$$\widehat{L} = \begin{pmatrix} .100 \times 10^1 & 0 \\ .100 \times 10^4 & .100 \times 10^1 \end{pmatrix} \tag{2.34}$$

$$\widehat{A}^{(1)} = \widehat{U} = \begin{pmatrix} .100 \times 10^{-2} & .100 \times 10^1 \\ 0 & -.100 \times 10^4 \end{pmatrix}$$

Forward Substitution. Applicando la Forward Substitution al sistema $\widehat{L}y = b$ si ottiene:

- $\widehat{y}_1 = b_1 = .100 \times 10^1$
- $\widehat{y}_2 = fl(b_2 - m_{2,1}\widehat{y}_1) = .200 \times 10^1 - (.100 \times 10^4) \times (.100 \times 10^1) =$
  $.200 \times 10^1 - .100 \times 10^4 = -.100 \times 10^4$

Back Substitution. Infine applicando la Back Substitution al sistema $\widehat{U}x = \widehat{y}$ si ottiene:

- $\widehat{x}_2 = fl(\widehat{y}_2 / a_{2,2}^{(1)}) = (-.100 \times 10^4)/(-.100 \times 10^4) = 0.100 \times 10^1$
- $\widehat{x}_1 = fl(\widehat{y}_1 - a_{1,2}^{(1)}\widehat{x}_2) = .100 \times 10^1 - (.100 \times 10^1) \times (.100 \times 10^1) = 0.0$

La soluzione calcolata anche in questo caso è inaccettabile. L'errore relativo commesso è:

$$\frac{||\Delta x||_\infty}{||x||_\infty} = \frac{\max(|\widehat{x}_1 - x_1|, |\widehat{x}_2 - x_2|)}{\max(|x_1|, |x_2|)} \simeq 1$$

A questo punto è utile osservare che, relativamente all'esempio 2.33, la fase di fattorizzazione della matrice A ha generato le matrici $\widehat{L}$ e $\widehat{U}$ della (2.34) tali che:

$$\widehat{L}\widehat{U} = A' = \begin{pmatrix} 0.001 & 1 \\ 1 & 0 \end{pmatrix}$$

Il prodotto $\widehat{L}\widehat{U}$ ricostruisce quindi una matrice A' che puó essere vista come la matrice A in cui l'elemento $a_{2,2}$ è stato modificato mediante una perturbazione dell'ordine $O(10^0)$, cioè:

$$A' = \begin{pmatrix} 0.001 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0.001 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & e_{2,2} \end{pmatrix} = A + E \qquad |E| = O(10^0) \tag{2.35}$$

avendo indicato $|E| = (|e_{i,j}|)_{i,j=1,..,n}$. In definitiva, quindi, a causa del sistema aritmetico a precisione finita utilizzato, nella fase di fattorizzazione si è di fatto fattorizzata la matrice A' della (2.35) in luogo della matrice A della (2.33).

♣

Relativamente ad un generico sistema di equazioni di ordine $n = 2$ si ottiene:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 = b_2 \end{cases} \iff Ax = b$$

nella fase di fattorizzazione della matrice, si ha che:

$$\begin{aligned}\widehat{m}_{2,1} &= fl(a^{(0)}_{2,1}/a^{(0)}_{1,1}) = a^{(0)}_{2,1}(1+\delta_d)/a^{(0)}_{1,1} = m_{2,1}(1+\delta_d)\\ \widehat{a}^{(1)}_{2,2} &= (a^{(0)}_{2,2} - m_{2,1}(1+\delta_d)a^{(0)}_{1,2}(1+\delta_m))(1+\delta_s)\end{aligned} \tag{2.36}$$

dove $|\delta_d| \leq u$, $|\delta_m| \leq u$ e $|\delta_s| \leq u$ sono gli errori relativi di round-off commessi rispettivamente nella divisione, moltiplicazione e sottrazione. Dalla (2.36) si ha:

$$\begin{aligned}\widehat{a}^{(1)}_{2,2} &= a^{(0)}_{2,2} - m_{2,1}a^{(0)}_{1,2} + a^{(0)}_{2,2}\delta_s - m_{2,1}a^{(0)}_{1,2}(\delta_d+\delta_m+\delta_s) + O(u^2) =\\ &= a^{(1)}_{2,2} + a^{(0)}_{2,2}\delta_s - m_{2,1}a^{(0)}_{1,2}(\delta_d+\delta_m+\delta_s) + O(u^2)\end{aligned}$$

Siano:

$$\widehat{L} = \begin{pmatrix} 1 & 0 \\ \widehat{m}_{2,1} & 1 \end{pmatrix} \quad \widehat{U} = \begin{pmatrix} a^{(0)}_{1,1} & a^{(0)}_{1,2} \\ 0 & \widehat{a}^{(1)}_{2,2} \end{pmatrix}$$

Se si effettua in aritmetica esatta il prodotto $\widehat{L}\widehat{U} = A'$ si ottiene la matrice:

$$A' = \begin{pmatrix} a^{(0)}_{1,1} & a^{(0)}_{1,2} \\ a^{(0)}_{2,1}(1+\delta_d) & a^{(0)}_{2,2} + e_{2,2} \end{pmatrix}$$

con:

$$e_{2,2} = a^{(0)}_{2,2}\delta_s - m_{2,1}a^{(0)}_{1,2}(\delta_m+\delta_s) + O(u^2)$$

Dall'espressione precedente si nota che il moltiplicatore $m_{2,1}$ rappresenta un fattore di amplificazione per gli errori relativi di round-off δ_d, δ_m, δ_s commessi nelle operazioni aritmetiche. Di fatto il prodotto $\widehat{L}\widehat{U}$ ricostruisce una matrice A' che, per effetto di tale moltiplicatore, puó essere molto distante dalla matrice A.

Nell'esempio 2.33 si è avuto $\widehat{m}_{2,1} = 0.1 \times 10^4$, per cui l'errore relativo di round-off $|\delta_s| \leq u = 0.5 \times 10^{-2}$ si è amplificato di circa 1000 volte.

Teorema 2.5. *Sia A una matrice di ordine n. Se ad ogni passo k della fattorizzazione LU si ha che $a^{(k)}_{k,k} \neq 0$ (cioè non si incontrano pivot nulli), allora i fattori calcolati $\widehat{L}$ e $\widehat{U}$ soddisfano:*

$$\begin{aligned}\widehat{L}\widehat{U} &= A' = A + E\\ |E| &\leq 3(n-1)u(|A| + |\widehat{L}||\widehat{U}|) + O(u^2)\end{aligned} \tag{2.37}$$

dove $|E| = (|e_{i,j}|)_{i,j=1,..,n}$.

Dimostrazione La dimostrazione procede per induzione su k. Poiché il teorema è ovviamente vero per $k = 1$, si supponga che la (2.37) sia vera per $k-1$. Sia quindi la matrice:

$$A = \begin{pmatrix} \alpha & w^T \\ v & B \end{pmatrix}$$

dove B è una matrice di ordine $k-1$, v e w due vettori di ordine $k-1$ e α uno scalare e se ne effettui la fattorizzazione LU. Allora, nel primo passo di tale fattorizzazione, si ha:

$$\widehat{z} = fl(v/\alpha) = \frac{v}{\alpha} + f \qquad |f| \leq u\left|\frac{v}{\alpha}\right|$$

e

$$\widehat{A}_1 = fl(B - \widehat{z}w^T) = B - \widehat{z}w^T + F \qquad |F| \leq 2u(|B| + |\widehat{z}||w|^T) + O(u^2) \tag{2.38}$$

I passi successivi effettuano la fattorizzazione della matrice $\widehat{A}_1$, di ordine $k-1$. Per l'ipotesi di induzione, per tale matrice vale la (2.37), cioè:

$$\widehat{L}_1\widehat{U}_1 = \widehat{A}_1 + E_1$$
$$|E_1| \leq 3(k-2)u(|\widehat{A}_1| + |\widehat{L}_1||\widehat{U}_1|) + O(u^2)$$

Posto allora:

$$\widehat{L}\widehat{U} = \begin{pmatrix} 1 & 0 \\ \widehat{z} & \widehat{L}_1 \end{pmatrix}\begin{pmatrix} \alpha & w^T \\ 0 & \widehat{U}_1 \end{pmatrix} = A + E$$

dalla (2.38) si ha qundi che

$$|\widehat{A}_1| \leq (1+2u)(|B| + |\widehat{z}||w|^T) + O(u^2)$$

e quindi usando le ipotesi di induzione sulla matrice $\widehat{A}_1$ si ha:

$$|E_1 + F| \leq 3(k-1)u(|B| + |\widehat{z}||w|^T + |\widehat{L}_1||\widehat{U}_1|) + O(u^2)$$

Poiché $|\alpha f| \leq u|v|$ si ha:

$$|E| \leq 3(k-1)u\left[\begin{pmatrix} |\alpha| & |w^T| \\ |v| & |B| \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ |\widehat{z}| & |\widehat{L}_1| \end{pmatrix}\begin{pmatrix} |\alpha| & |w^T| \\ 0 & |\widehat{U}_1| \end{pmatrix}\right] + O(u^2)$$

cioè la tesi. ■

La (2.37) asserisce che, effettuando la fattorizzazione $A = LU$ di una matrice in un sistema aritmetico a precisione finita, in pratica si fattorizza una matrice A', i cui elementi differiscono da quelli di A di una quantità:

$$|a_{i,j} - a'_{i,j}| = |e_{i,j}| \leq 3(n-1)u(|a_{i,j}| + \sum_{k=1}^{n} |\widehat{l}_{i,k}||\widehat{u}_{k,j}|) + O(u^2)$$

cioè:

$$\frac{|a_{i,j} - a'_{i,j}|}{|a_{i,j}|} = \frac{|e_{i,j}|}{|a_{i,j}|} \leq 3(n-1)u\left(1 + \frac{\sum_{k=1}^{n} |\widehat{l}_{i,k}||\widehat{u}_{k,j}|}{|a_{i,j}|}\right) + O(u^2) \tag{2.39}$$

Analizzando la (2.39), si nota che la perturbazione relativa $|e_{i,j}|/|a_{i,j}|$ dipende da quanto grandi possono diventare gli elementi[15] $|l_{i,k}|$ e $|u_{k,j}|$. Poichè la matrice L è costituita da tutti i moltiplicatori $l_{i,k} = a_{i,k}/a_{k,k}$ incontrati nel corso dell'algoritmo di eliminazione di Gauss, la presenza di elementi $a_{k,k}$ piccoli rispetto agli elementi $a_{i,k}$ della stessa colonna, comporta che $|l_{i,k}| >> 1$ e quindi la possibilità di grosse perturbazioni $|e_{i,j}|$ nei dati.

La soluzione a tale problema deve avere quindi come obiettivo quello di rendere gli elementi $|l_{i,k}| = |a_{i,k}|/|a_{k,k}| \leq 1$ in modo da non amplificare gli errori di round-off. È noto che una possibile soluzione risiede nella tecnica del pivoting (parziale o totale) che consiste, ad ogni passo k, in una riorganizzazione delle righe e/o delle colonne di A in modo da avere l'elemento di massimo modulo sempre sulla diagonale. Si ricordi inoltre che l'utilizzo di tale tecnica comporta che ad essere fattorizzata è la matrice A opportunamente moltiplicata per una matrice di permutazione. Ad esempio, relativamente alla tecnica del pivoting parziale si ha:

$$PA = LU \quad \Leftrightarrow \quad A = P^T LU$$

dove P è una matrice di permutazione, P^T la sua trasposta ed è $P^T P = I$.

A questo punto è necessario effettuare nuovamente l'analisi dell'errore di round-off della fattorizzazione $A = P^T LU$ ottenuta utilizzando la tecnica del pivoting parziale. A tal fine è utile notare che nella fase di pivoting, cioè nella fase dello scambio delle righe della matrice A, non vengono effettuate operazioni aritmetiche e quindi non vengono commessi errori di round-off. Il teorema che segue, la cui dimostrazione è analoga a quella del Teorema 2.5, fornisce l'analisi dell'errore di round-off richiesta:

Teorema 2.6. *Sia A una matrice di ordine n. Se ad ogni passo k della fattorizzazione LU con pivoting parziale si ha che $a_{k,k}^{(k)} \neq 0$ (cioè non si incontrano pivot nulli), allora i fattori calcolati $\widehat{L}$ e $\widehat{U}$ soddisfano:*

$$\begin{gathered} \widehat{L}\widehat{U} = A + E \\ |E| \leq 3(n-1)u(|A| + |\widehat{P}^T||\widehat{L}||\widehat{U}|) + O(u^2) \end{gathered} \tag{2.40}$$

dove $|E| = (|e_{i,j}|)$, $i, j = 1, .., n$ e P è la matrice di permutazione derivante dall'utilizzo del pivoting parziale nell'algoritmo di eliminazione di Gauss.

Dalla (2.40) si ricava che:

$$\frac{|a_{i,j} - a'_{i,j}|}{|a_{i,j}|} = \frac{|e_{i,j}|}{|a_{i,j}|} \leq 3(n-1)u \left(1 + \frac{\sum_{p_k=1}^{n} |\widehat{l}_{i,p_k}||\widehat{u}_{p_k,j}|}{|a_{i,j}|}\right) + O(u^2)$$

[15] Il fattore $n-1$ non è di importanza significativa e può essere trascurato in questa analisi. Tale fattore è la conseguenza del fatto che nel Teorema 2.5 si sono utilizzati i valori assoluti degli errori di round-off $|\delta|$ non tenendo conto che essi, essendo con uguale probabilità positivi e negativi, tendono a compensarsi a vicenda. Come sottolineato da Wilkinson lo scopo di tale analisi è di individuare le potenziali instabilità degli algoritmi per migliorarli e non di ottenere una precisa maggiorazione dell'errore di round-off.

dove $p_k = 1, .., n$ rappresenta una permutazione degli indici $k = 1, .., n$. Inoltre, poichè $||\widehat{P}^T|| = 1$ e $||\widehat{L}|| = n$, utilizzando le norme si ha:

$$\frac{||E||_\infty}{||A||_\infty} \leq 3(n-1)u \left(1 + n\frac{||\widehat{U}||_\infty}{||A||_\infty}\right) + O(u^2) \tag{2.41}$$

che mostra la grandezza della perturbazione relativa sulla matrice A nel caso di pivoting parziale.

A questo punto è opportuno chiedersi quanto grande possa diventare il rapporto $||\widehat{U}||_\infty/||A||_\infty$ presente nella (2.41), in quanto anche esso può rappresentare un fattore di amplificazione per gli errori di round-off.

♣ **Esempio 2.34.** Si risolva mediante la fattorizzazione $A = LU$ con pivoting parziale e gli algoritmi di *Forward Substitution* e *Back Substitution* il sistema di equazioni:

$$Ax = b \quad \text{con } A = \begin{cases} -1 & \text{se } i > j \\ 1 & \text{se } i = j \text{ o } j = n \\ 0 & \text{altrimenti} \end{cases} \quad e \quad b_i = \sum_{j=1}^{n} a_{i,j}$$

La soluzione di tale sistema è $x_i = 1 \quad i = 1, .., n$, ma, per un sistema di ordine $n = 100$, si ha che la soluzione calcolata con un calcolatore con aritmetica standard IEEE in doppia precisione è:

$$x_i = 1 \quad i = 1, .., 53 \qquad x_i = 0 \quad i = 54, .., 99 \qquad x_{100} = 1$$

con un errore relativo $||\Delta x||/||x|| = 1$. La soluzione è quindi inaccettabile.

Per capire il perché di tale inaccuratezza, nonostante l'utilizzo della tecnica del pivoting parziale, si osservi che la matrice U che si ottiene nella fase di fattorizzazione è la seguente:

$$U = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 1 \\ 0 & 1 & 0 & \cdots & 2 \\ 0 & 0 & 1 & \cdots & 2^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 2^{n-1} \end{pmatrix}$$

e quindi $||\widehat{U}||_\infty/||A||_\infty = O(2^{n-1}/n)$.

♣

È possibile dare la seguente:

Definizione 2.9. (Fattore di crescita) *Relativamente all'algoritmo di eliminazione di Gauss, si definisce il* **fattore di crescita** *nel modo seguente:*

$$\rho = \max_{i,j,k} \frac{|\widehat{a}_{i,j}^{(k)}|}{||A||_\infty}$$

Il fattore di crescita, quindi, misura quanto grande diventano gli elementi $|\widehat{a}_{i,j}^{(k)}|$ nel corso dell'algoritmo di eliminazione di Gauss rispetto alla matrice A. Poiché $||\widehat{U}||_\infty/||A||_\infty \leq n\rho$, dalla (2.41) si ottiene:

$$\frac{||E||_\infty}{||A||_\infty} \leq 3(n-1)u\left(1 + n^2\rho\right) + O(u^2) \tag{2.42}$$

L'esempio 2.34 mostra che il fattore di crescita puó essere, in alcuni casi, un fattore di amplificazione non trascurabile dell'errore di round-off nell'algoritmo di eliminazione di Gauss.

♣ **Esempio 2.35.** Il programma `MATLAB` :

```
n=100;   rho=0;

% ciclo sul numero di matrici
for nmat=1:100
   a=10*rand(n);   norma=norm(a,inf);   rapp=0;

   % fattorizzazione LU
   for k=1:n-1
      % pivoting
      [y,r] = max(abs(a(k:n,k)));  r=r+k-1;
      % scambio delle righe
      t=a(k,k:n);    a(k,k:n)=a(r,k:n);     a(r,k:n)=t;
      % calcolo  moltiplicatori
      a(k+1:n,k)=a(k+1:n,k)/a(k,k);
      % trasformazione matrice attiva
      a(k+1:n,k+1:n)=a(k+1:n,k+1:n)-a(k+1:n,k)*a(k,k+1:n);
      %
      rt=max(max(abs(a(k:n,k:n))));
      if rt>rapp     rapp=rt;     end
   end

   % calcolo di rho
   rapp=rapp/norma;
   if rapp>rho    rho=rapp;     end

end
disp('massimo rho='); disp(rho)
```

calcola il massimo fattore di cresita ottenuto da 100 matrici casuali di ordine $n = 100$ i cui elementi sono tali che $0 \leq a_{i,j} \leq 10$. Tale programma fornisce come risultato:

```
massimo rho=
    0.1879
```

♣

Anche se è possibile definire matrici con fattore di crescita $\rho = O(2^n)$, nella pratica tale valore rimane limitato e non costituisce un effettivo fattore di amplificazione per gli errori di round-off nell'algoritmo di eliminazione di Gauss con pivoting parziale.

Tale risultato, associato alla considerazione che il costo computazionale del pivoting totale è sensibilmente maggiore di quello del pivoting parziale, permette di concludere

che l'algoritmo di eliminazione di Gauss con pivoting parziale è un algoritmo praticamente stabile e può essere utilizzato nello sviluppo di software matematico accurato.

2.8.2 Analisi dell'errore di round-off nella Forward e Back Substitution

Per completare l'analisi dell'accuratezza della soluzione di un sistema di equazioni lineari $Ax = b$ è opportuno effettuare la backward error analysis anche degli algoritmi di *Forward Substitution* e di *Back Substitution*. Vista l'analogia tra i due algoritmi, nel seguito verrà effettuata solo l'analisi dell'errore di round-off della *Forward Substitution*, in quanto quella della *Back Substitution* può essere condotta allo stesso modo. Sia quindi un sistema di equazioni triangolare inferiore $Lx = b$:

In generale la componente x_i viene calcolata mediante:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} l_{i,j} x_j}{l_{i,i}} \qquad i = 1, ..., n \tag{2.43}$$

Relativamente a tale espressione è possibile dimostrare il seguente:

Teorema 2.7. *Sia L una matrice triangolare inferiore di ordine n. Se ad ogni passo i dell'algoritmo di Forward Substitution si ha che $l_{i,i} \neq 0$, allora la soluzione calcolata $\widehat{x}$ è soluzione del sistema:*

$$(L + E)\widehat{x} = b \qquad |E| \leq (n-1)|L|u + O(u^2) \tag{2.44}$$

Dimostrazione Relativamente alla (2.43) si ha che:

$$\widehat{x}_1 = \frac{b_1}{l_{1,1}(1+\delta_d)} \quad \Rightarrow \quad l_{1,1}(1+\delta_d)\widehat{x}_1 = b_1 \qquad |\delta_d| \leq u$$

con δ_d errore di round-off commesso nella divisione. Il calcolo di x_1 avviene quindi come se l'elemento $l_{1,1}$ fosse stato perturbato della quantità:

$$|e_{1,1}| = |l_{1,1}\delta_d| \leq |l_{1,1}|u$$

Proseguendo nella *Forward Substitution* si ha:

$$\widehat{x}_2 = \frac{b_2 - l_{2,1}\widehat{x}_1(1+\delta_m)}{l_{2,2}(1+\delta_d)(1+\delta_s)} \quad \Longrightarrow \quad l_{2,1}(1+\delta_m)\widehat{x}_1 + l_{2,2}(1+\delta_s)(1+\delta_d)\widehat{x}_2 = b_2$$

con δ_s e δ_m rispettivamente errori di round-off nella sottrazione e nella moltiplicazione. Il calcolo di x_2 avviene quindi come se $l_{2,1}$ e $l_{2,2}$ fossero stati perturbati rispettivamente da:

$$|e_{2,1}| = |l_{2,1}\delta_m| \leq |l_{2,1}|u \quad e \quad |e_{2,2}| = |l_{2,2}(\delta_d + \delta_s + \delta_d\delta_s)| \leq 2|l_{2,2}|u + O(u^2)$$

Per la componente x_i si osservi che $\sum_{j=1}^{i-1} l_{i,j}x_j$ è il prodotto scalare tra le prime $i-1$ componenti della i_ma riga di L e del vettore $\widehat{x}$. Quindi si ha:

$$fl\left(\sum_{j=1}^{i-1} l_{i,j}x_j\right) = l_{i,1}\widehat{x}_1(1+\theta_{i-1}) + l_{i,2}\widehat{x}_2(1+\theta_{i-2}) + ... + l_{i,i-1}\widehat{x}_{i-1}(1+\theta_1)$$

con

$$|\theta_i| \leq iu + O(u^2)$$

da cui si ha:

$$\widehat{x}_i = \frac{b_i - (l_{i,1}\widehat{x}_1(1+\theta_{i-1}) + \cdots + l_{i,i-1}\widehat{x}_{i-1}(1+\theta_1))}{l_{i,i}(1+\delta_s)(1+\delta_d)} \quad \Longrightarrow$$

$$l_{i,1}\widehat{x}_1(1+\theta_{i-1}) + \cdots + l_{i,i}(1+\delta_d)(1+\delta_s)\widehat{x}_i = b_i$$

cioè x_i viene calcolato come se gli elementi $l_{i,j}$ fossero stati perturbati da:

$$\begin{aligned}
|e_{i,1}| = |l_{i,1}\theta_{i-1}| &\leq (i-1)|l_{i,1}|u + O(u^2) \\
|e_{i,j}| = |l_{i,j}\theta_{i-j}| &\leq (i-j)|l_{i,j}|u + O(u^2) \quad j = 2, .., i-1 \\
|e_{i,i}| = |l_{i,i}(\delta_d + \delta_s + \delta_d\delta_s)| &\leq 2|l_{i,i}|u + O(u^2)
\end{aligned}$$

da cui la tesi. ■

La soluzione calcolata $\widehat{x}$ è la soluzione esatta del sistema di equazioni perturbato $(L+E)\widehat{x} = b$. Si osservi però che le perturbazioni $e_{i,j}$ sono piccole rispetto agli elementi della matrice $l_{i,j}$, in quanto:

$$\frac{|e_{i,j}|}{|l_{i,j}|} \leq nu + O(u^2)$$

per cui il sistema perturbato si discosta di poco dal sistema originario $Lx = b$. In definitiva l'algoritmo di *Forward Substitution* (come anche quello di *Back Substitution*) può essere considerato stabile.

2.9 Sistemi lineari con matrici strutturate

L'algoritmo di fattorizzazione LU è stato applicato a matrici generiche, intendendo con ciò che esso non tiene del fatto che la matrice possa avere particolari caratteristiche, ad esempio *"pochi"* elementi non nulli disposti secondo un certo schema, oppure elementi

simmetrici rispetto alla diagonale principale uguali. I paragrafi che seguono sono dedicati alla risoluzione di sistemi lineari con matrice dei coefficienti *"speciale"*; si vedrà infatti come sia possibile sfruttare eventuali proprietà della matrice dei coefficienti per ridurre la complessità di tempo e di spazio richiesta da tale risoluzione. Si noti, a tal proposito, che risolvere un problema cercando di trarre vantaggio dalle particolari caratteristiche del problema stesso costituisce un principio generale dell'Analisi Numerica.

Nei due esempi che seguono sono presentati alcuni problemi descritti da sistemi lineari la cui matrice dei coefficienti ha particolari caratteristiche.

♣ **Esempio 2.36.** Si consideri una rete di resistori uguali tra loro, disposti come in Fig. 2.21. Si indichi con R la resistenza di ciascuno di essi e con v_k il potenziale nel k-mo nodo della rete e si supponga che v_0 e v_{n+1} siano noti.

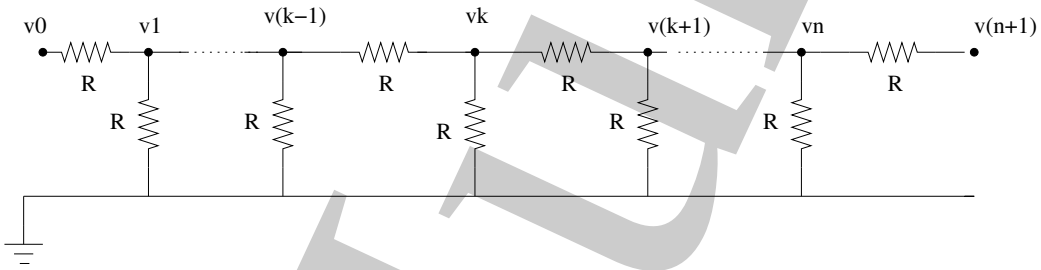

Figura 2.21: Rete di resistori.

Applicando la prima legge di Kirchoff e la legge di Ohm[16], si ha, per ogni nodo k, la seguente equazione:

$$\frac{(v_{k-1} - v_k)}{R} - \frac{(v_k - v_{k+1})}{R} = \frac{v_k}{R},$$

ovvero

$$-v_{k-1} + 3v_k - v_{k+1} = 0. \tag{2.45}$$

L'equazione (2.45), scritta per $k = 1, \ldots, n$, dà luogo al seguente sistema lineare nelle incognite v_k:

$$\begin{pmatrix} 3 & -1 & & & \\ -1 & 3 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 3 & -1 \\ & & & -1 & 3 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} v_0 \\ 0 \\ \vdots \\ 0 \\ v_{n+1} \end{pmatrix}. \tag{2.46}$$

[16]La *legge di Ohm* afferma che l'intensità della corrente elettrica che percorre un filo metallico a temperatura costante è direttamente proporzionale alla differenza di potenziale esistente tra gli estremi del filo. Indicata con $v_A - v_B$ la differenza di potenziale tra gli estremi A e B del filo e con I l'intensità della corrente che percorre il filo dal punto A al punto B, la legge di Ohm si può esprimere come $I = (v_A - v_B)/R$, dove la costante R è la resistenza del filo.

La matrice del sistema (2.46) ha molti elementi nulli e gli elementi non nulli si trovano sulla diagonale principale e sulle due diagonali immediatamente al di sotto ed al di sopra di questa. Tale matrice è inoltre simmetrica, cioè, detto $a_{i,j}$ il suo generico elemento di posto (i,j), risulta $a_{i,j} = a_{j,i}$. ♣

♣ **Esempio 2.37.** Si consideri l'equazione differenziale ordinaria (equazione di convezione-diffusione stazionaria in una dimensione):

$$u\frac{dc}{dx}(x) - \alpha\frac{d^2c}{dx^2}(x) = 0\,, \quad x \in [0,1]\,, \tag{2.47}$$

con le condizioni al contorno:

$$c(0) = 0\,, \quad c(1) = 1\,. \tag{2.48}$$

Per risolvere numericamente il problema (2.47)-(2.48) si può procedere nel modo seguente. Posto $\Delta x = 1/n$, con n intero positivo, si considerino i punti

$$x_0, x_1, \ldots, x_n \in [0,1]$$

tali che

$$x_k = k\Delta x, \quad k = 0, \ldots, n\,,$$

e si approssimino le derivate di $c(x)$ in tali punti con le espressioni seguenti:[17]

$$\begin{aligned}
\frac{dc}{dx}(x_k) &\simeq \frac{1}{2\Delta x}\left(c_{k+1} - c_{k-1}\right) + \frac{1}{6\Delta x}\left(c_{k-2} - 3c_{k-1} + 3c_k - c_{k+1}\right) \\
&= \frac{1}{\Delta x}\left(\frac{1}{6}c_{k-2} - c_{k-1} + \frac{1}{2}c_k + \frac{1}{3}c_{k+1}\right)\,, \\
\frac{d^2c}{dx^2}(x_k) &\simeq \frac{1}{\Delta x^2}\left(c_{k-1} - 2c_k + c_{k+1}\right)\,,
\end{aligned}$$

dove c_k è un'approssimazione di $c(x_k)$. Sostituendo le espressioni precedenti nell'equazione (2.47), si ha:

$$\frac{u}{6\Delta x}c_{j-2} - \left(\frac{u}{\Delta x} + \frac{\alpha}{(\Delta x)^2}\right)c_{j-1} + \left(\frac{u}{2\Delta x} + \frac{2\alpha}{(\Delta x)^2}\right)c_j + \left(\frac{u}{3\Delta x} - \frac{\alpha}{(\Delta x)^2}\right)c_{j+1} = 0\,,$$

ovvero, posto $R = u/\Delta x$:

$$\frac{R}{6}c_{j-2} - (R+1)c_{j-1} + \left(\frac{R}{2} + 2\right)c_j + \left(\frac{R}{3} - 1\right)c_{j+1} = 0.$$

Posto $c_{-1} = 0$ e tenuto conto delle condizioni (2.48), si perviene al seguente sistema lineare nelle incognite c_k, $k = 1, \ldots, n-1$:

$$\begin{pmatrix}
\frac{R}{2}+2 & \frac{R}{3}-1 & & & \\
R+1 & \frac{R}{2}+2 & \frac{R}{3}-1 & & \\
\frac{R}{6} & R+1 & \frac{R}{2}+2 & \frac{R}{3}-1 & \\
\ddots & \ddots & \ddots & \ddots & \\
 & \frac{R}{6} & R+1 & \frac{R}{2}+2 & \frac{R}{3}-1 \\
 & & \frac{R}{6} & R+1 & \frac{R}{2}+2
\end{pmatrix}
\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix}
=
\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{2.49}$$

La matrice dei coefficienti del sistema (2.49) ha gli elementi non nulli disposti solo su quattro diagonali, la diagonale principale, le prime due inferiori e la prima superiore. ♣

[17] Non ci si sofferma, in questa sede, su tali approssimazioni delle derivate. Ci si limita solo ad osservare che esse possono essere ottenute utilizzando opportunamente lo sviluppo in serie di Taylor della funzione $c(x)$ e che sono effettivamente utilizzate nelle applicazioni (Si veda, ad esempio, [7]).

Le matrici considerate negli esempi precedenti hanno gli elementi disposti secondo uno schema preciso e si dicono quindi *strutturate*. In particolare, nella matrice dell'esempio 2.36, gli elementi non nulli sono localizzati sulla diagonale principale e sulle due diagonali ad essa adiacenti; nella matrice dell'esempio 2.37, gli elementi non nulli appartengono alla diagonale principale e ad altre diagonali ad essa vicine.

Definizione 2.10. (Matrice tridiagonale)
Una matrice $A = (a_{i,j})$ si dice tridiagonale se

$$a_{i,j} = 0 \ \ per \ \ |i - j| > 1 \, .$$

Definizione 2.11. (Matrice a banda)
Una matrice $A = (a_{i,j})$ si dice a banda, con ampiezza di banda inferiore p ed ampiezza di banda superiore q, se

$$a_{i,j} = 0 \ \ per \ \ i > j + p \ \ e \ \ per \ \ j > i + q \, .$$

Una rappresentazione grafica di una matrice a banda è fornita in Fig. 2.22. Si osservi che una matrice tridiagonale è una particolare matrice a banda, con ampiezze di banda $p = q = 1$. Si osservi inoltre che la definizione di matrice a banda non implica che gli elementi della diagonale principale, delle p diagonali inferiori e delle q diagonali superiori siano tutti non nulli, ma solo che siano nulli gli elementi che non appartengono a queste diagonali.

Le matrici degli esempi 2.36 e 2.37 sono matrici a banda. La prima ha entrambe le ampiezze di banda uguali ad 1, ovvero è tridiagonale; la seconda ha ampiezza di banda inferiore uguale a 2 ed ampiezza di banda superiore uguale ad 1.

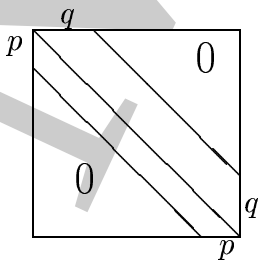

Figura 2.22: Matrice a banda.

In maniera analoga alla matrice tridiagonale si definisce la matrice *pentadiagonale* ($p = q = 2$), *eptadiagonale* ($p = q = 3$), etc. Anche una matrice *diagonale* si può vedere come una particolare matrice a banda, in cui $p = q = 0$. Nel seguito si considerano matrici a banda quadrate, anche se le precedenti definizioni si applicano anche a matrici con un numero di righe diverso dal numero di colonne.

Sono matrici strutturate anche le matrici *triangolari*, i cui elementi non nulli sono appunto disposti secondo una struttura a triangolo. Tali matrici possono essere anche

a banda, con una delle due ampiezze di banda necessariamente nulla. In tal caso, si parla semplicemente di ampiezza di banda della matrice, facendo riferimento alla banda di ampiezza non nulla. Un esempio di matrice triangolare inferiore a banda ed uno di matrice triangolare superiore a banda è fornito di seguito.

♣ **Esempio 2.38.**

Si considerino le seguenti matrici:

$$L = \begin{pmatrix} 1 & & & \\ -2 & 4 & & \\ & 5 & 1 & \\ & & -7 & 2 \end{pmatrix}, \quad U = \begin{pmatrix} 5 & -3 & & \\ & 1 & -2 & \\ & & -4 & 1 \\ & & & 6 \end{pmatrix}.$$

Tali matrici, oltre ad essere triangolari, sono matrici a banda; la prima con ampiezza di banda (inferiore) $p = 1$ e la seconda con ampiezza di banda (superiore) $q = 1$. ♣

In generale, si dà la seguente definizione:

Definizione 2.12. (Matrice bidiagonale)
Una matrice $A = (a_{i,j})$ si dice bidiagonale inferiore se

$$a_{i,j} = 0 \ \ per \ \ j > i \ \ e \ \ per \ \ i > j + 1\,,$$

e si dice bidiagonale superiore se

$$a_{i,j} = 0 \ \ per \ \ i > j \ \ e \ \ per \ \ j > i + 1\,.$$

2.10 Memorizzazione di matrici strutturate

Memorizzare una matrice strutturata in un array bidimensionale non è in generale conveniente dal punto di vista della complessità di spazio, in quanto si memorizzano anche elementi nulli la cui posizione è nota. Una memorizzazione efficiente deve tener conto della struttura della matrice, evitando di conservare informazioni inutili. D'altra parte, la struttura stessa della matrice suggerisce schemi di memorizzazione piú appropriati.

Si inizi col considerare una matrice diagonale, di dimensione $n \times n$:

$$A = \begin{pmatrix} a_{1,1} & & & \\ & a_{2,2} & & \\ & & \ddots & \\ & & & a_{n,n} \end{pmatrix}.$$

Se si memorizza tale matrice in un array di dimensione $n \times n$ si ha una complessità di spazio pari a n^2, anche se gli elementi che bisogna effettivamente conservare, cioè quelli

della diagonale, sono solo n. Tale osservazione conduce in maniera naturale ad utilizzare un array monodimensionale, AD, di lunghezza n, per memorizzare solo tali elementi:

$$AD = (a_{1,1}, a_{2,2}, \ldots, a_{n,n}) ,$$

ovvero

$$AD(i) = a_{i,i} , \quad i = 1, \ldots, n .$$

La complessità di spazio si riduce, in tal modo, di un ordine di grandezza.

Nel caso di una matrice tridiagonale di dimensione $n \times n$, gli elementi che bisogna effettivamente conservare sono quelli delle tre diagonali, cioè sono $3n - 2$.

♣ **Esempio 2.39.** Si consideri la matrice tridiagonale del sistema (2.46) (esempio 2.36). Si vuole determinare uno schema di memorizzazione efficiente, che non conservi cioè gli elementi al di fuori delle tre diagonali. Una scelta naturale è utilizzare tre array monodimensionali, uno per ciascuna diagonale. Detti AD, AU ed AL tali array, si ha quindi:

$$AD = \underbrace{(\ 3,\ 3,\ \ldots\ ,\ 3)}_{n \text{ volte}},$$
$$AL = \underbrace{(-1, -1, \ldots, -1)}_{n-1 \text{ volte}},$$
$$AU = \underbrace{(-1, -1, \ldots, -1)}_{n-1 \text{ volte}} .$$

Si noti che la matrice considerata è anche simmetrica e si può quindi evitare di memorizzare la diagonale inferiore o quella superiore, utilizzando così solo due array monodimensionali, con un ulteriore risparmio dello spazio di memoria. ♣

In generale, data una matrice tridiagonale A di dimensione $n \times n$,

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ & & & a_{n,n-1} & a_{n,n} \end{pmatrix} ,$$

si utilizza un array monodimensionale, di lunghezza n, per memorizzare la diagonale principale e due array monodimensionali, di lunghezza $n - 1$, per memorizzare la diagonale inferiore e quella superiore:

$$\begin{aligned} AD(i) &= a_{i,i} , & i &= 1, \ldots, n , \\ AL(i+1) &= a_{i+1,i} , & i &= 1, \ldots, n-1 , \\ AU(i) &= a_{i,i+1} , & i &= 1, \ldots, n-1 . \end{aligned}$$

Si noti che al primo elemento dell'array AL è stato assegnato l'indice 2; in tal modo l'indice del generico elemento dell'array risulta essere uguale all'indice di riga del corrispondente elemento della diagonale inferiore. La scelta di assegnare l'indice 1 al primo

elemento di AL è chiaramente equivalente. Tale schema di memorizzazione ha una complessità di spazio pari a $3n-2$, cioè $O(n)$, di un ordine di grandezza inferiore rispetto alla usuale memorizzazione mediante un array bidimensionale.

Se la matrice tridiagonale è simmetrica si può evitare di memorizzare la diagonale inferiore o quella superiore, riducendo la complessità di spazio a $2n-1$. Si osservi che la memorizzazione con due array monodimensionali è adatta anche a matrici bidiagonali.

Per una matrice a banda A, di dimensione $n \times n$, con ampiezza di banda inferiore p ed ampiezza di banda superiore q, si potrebbero utilizzare $p+q+1$ array monodimensionali, contenenti le p diagonali inferiori, la diagonale principale e le q diagonali superiori. In tal modo, però, il numero di array da dichiarare in un programma che opera su matrici a banda sarebbe variabile con la dimensione della matrice.

♣ Esempio 2.40.

Si consideri la matrice a banda del sistema lineare (2.49) (esempio 2.37), che ha ampiezza di banda inferiore $p = 2$ ed ampiezza di banda superiore $q = 1$. Dato che la matrice ha solo quattro diagonali *"significative"*, si può considerare un array bidimensionale AB di dimensione $4 \times n$, e si possono memorizzare le diagonali della matrice nelle righe di tale array, nel modo seguente:

$$AB = \begin{pmatrix} * & \frac{R}{3}-1 & \cdots & \frac{R}{3}-1 & \frac{R}{3}-1 & \frac{R}{3}-1 \\ \frac{R}{2}+2 & \frac{R}{2}+2 & \cdots & \frac{R}{2}+2 & \frac{R}{2}+2 & \frac{R}{2}+2 \\ R+1 & R+1 & \cdots & R+1 & R+1 & * \\ \frac{R}{6} & \frac{R}{6} & \cdots & \frac{R}{6} & * & * \end{pmatrix},$$

dove il simbolo $*$ indica che al corrispondente elemento dell'array AB non è stato associato alcun valore. ♣

Nell'esempio 2.40 le diagonali della matrice sono memorizzate, una dopo l'altra, a partire dalla diagonale superiore, nelle righe dell'array. Inoltre, per ogni j, la j-ma colonna della matrice si trova nella j-ma colonna dell'array.

♣ Esempio 2.41.

Si consideri una matrice A di dimensione 6×6, con ampiezze di banda $p = 1$ e $q = 2$:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & & \\ & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & \\ & & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} \\ & & & a_{5,4} & a_{5,5} & a_{5,6} \\ & & & & a_{6,5} & a_{6,6} \end{pmatrix}.$$

Utilizzando per la matrice A lo schema di memorizzazione introdotto nell'esempio 2.40, si ha:

$$AB = \begin{pmatrix} * & * & a_{1,3} & a_{2,4} & a_{3,5} & a_{4,6} \\ * & a_{1,2} & a_{2,3} & a_{3,4} & a_{4,5} & a_{5,6} \\ a_{1,1} & a_{2,2} & a_{3,3} & a_{4,4} & a_{5,5} & a_{6,6} \\ a_{2,1} & a_{3,2} & a_{4,3} & a_{5,4} & a_{6,5} & * \end{pmatrix}.$$

L'elemento $a_{i,j}$ è dunque memorizzato nella colonna di posto j e nella riga di posto $3+i-j = q+1+i-j$. ♣

In generale, per memorizzare una generica matrice a banda A, di dimensione $n \times n$, con ampiezze di banda p e q, si utilizza un array bidimensionale di dimensione $(p+q+1) \times n$, nel modo seguente:

$$AB(q+1+i-j, j) = a_{i,j}\,, \quad \max(1, j-q) \leq i \leq \min(n, j+p)\,.$$

Tale schema di memorizzazione è detto *a banda* (in inglese si parla di *band storage*) ed ha una complessità di spazio pari a $(p+q+1)n$. E' chiaro che la memorizzazione a banda è "significativamente" vantaggiosa, rispetto alla usuale memorizzazione mediante un array bidimensionale di dimensione $n \times n$, se $p, q << n/2$.

Analogamente alle matrici a banda, si definiscono schemi di memorizzazione opportuni, per le matrici triangolari.

♣ Esempio 2.42.

Si consideri la matrice triangolare inferiore

$$L = \begin{pmatrix} 1 & & & & & \\ -3 & 10 & & & & \\ 2 & 8 & -1 & & & \\ 4 & -5 & 1 & 9 & & \\ 2 & 6 & 4 & -3 & 2 & \\ 14 & -8 & 6 & -4 & 7 & 11 \end{pmatrix},$$

di dimensione 6×6. Il triangolo superiore della matrice, senza la diagonale principale, è costituito da 15 elementi nulli; se per memorizzare tale matrice si utilizza un array bidimensionale con 6 righe e 6 colonne, si "sprecano" 15 su 36 componenti dell'array. Una scelta piú efficiente, suggerita in maniera naturale dalla struttura della matrice, è utilizzare un array monodimensionale, LP, contenente, una dopo l'altra, le righe, oppure le colonne, di L (si considera, ovviamente, solo la parte "significativa" di L):

$$LP = (1, -3, 10, 2, 8, -1, 4, -5, 1, 9, 2, 6, 4, -3, 2, 14, -8, 6, -4, 7, 11)\,,$$

oppure

$$LP = (1, -3, 2, 4, 2, 14, 10, 8, -5, 6, -8, -1, 1, 4, 6, 9, -3, -4, 2, 7, 11)\,.$$

♣

Per una matrice triangolare inferiore, di dimensione $n \times n$,

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

si può utilizzare un array monodimensionale di lunghezza $n(n+1)/2$, contenente, una dopo l'altra, le righe oppure le colonne del triangolo inferiore:

$$LP = (l_{1,1}, l_{2,1}, l_{2,2}, \ldots, l_{n,1}, l_{n,2}, \ldots, l_{n,n})\,,$$

oppure

$$LP = (l_{1,1}, l_{2,1}, \ldots, l_{n,1}, l_{2,2}, \ldots, l_{n,2}, \ldots, l_{n,n}) .$$

Si consideri ad esempio la memorizzazione per righe. La posizione di $l_{i,j}$, $i \geq j$, in LP si può determinare col seguente ragionamento. L'elemento $l_{i,j}$, appartenendo alla i-ma riga di L, è preceduto in LP dagli elementi delle $i-1$ righe precedenti, che sono

$$1 + 2 + \ldots + (i-1) = \sum_{k=1}^{i-1} k = \frac{i(i-1)}{2} ;$$

essendo inoltre il j-mo elemento della i-ma riga, è preceduto in LP da altri $j-1$ elementi. L'elemento $l_{i,j}$ è quindi preceduto in LP da

$$\frac{i(i-1)}{2} + j - 1$$

elementi, ovvero si ha

$$LP(i(i-1)/2 + j) = l_{i,j} , \quad i \geq j .$$

Analogamente si verifica che, se si usa la memorizzazione per colonne, risulta

$$LP(i + (2n-j)(j-1)/2) = l_{i,j} , \quad i \geq j .$$

Seguendo gli stessi criteri, si ottengono schemi di memorizzazione adatti ad una matrice triangolare superiore

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ & u_{2,2} & \cdots & u_{2,n} \\ & & \ddots & \vdots \\ & & & u_{n,n} \end{pmatrix} .$$

Anche in questo caso si utilizza un array monodimensionale UP, di lunghezza $n(n+1)/2$, per memorizzare le righe di U:

$$UP = (u_{1,1}, u_{1,2}, \ldots, u_{1,n}, u_{2,2}, \ldots, u_{2,n}, \ldots, u_{n,n}) ,$$

oppure le colonne:

$$UP = (u_{1,1}, u_{1,2}, u_{2,2}, \ldots, u_{1,n}, u_{2,n}, \ldots, u_{n,n}) .$$

Nel primo caso risulta

$$UP((2n-i)(i-1)/2 + j) = u_{i,j} , \quad i \leq j ;$$

nel secondo

$$UP(i + j(j-1)/2) = u_{i,j} , \quad i \leq j .$$

Gli schemi di memorizzazione per matrici triangolari sopra descritti sono detti *packed* (si parla, in inglese, di *packed storage*) e sono utilizzabili anche per la memorizzazione di matrici simmetriche. La scelta di una memorizzazione per righe o per colonne dipende dall'algoritmo che agisce sulla matrice; se l'algoritmo accede agli elementi della matrice per righe è opportuno utilizzare una memorizzazione per righe in modo da avere accessi a locazioni di memoria consecutive e quindi un utilizzo piú efficiente della cache memory.

L'uso di schemi di memorizzazione particolari comporta una *"riorganizzazione"* degli algoritmi che operano sulle corrispondenti matrici. Alcuni esempi al riguardo sono forniti dalle Procedure 2.12 e 2.13. La prima risolve un sistema lineare con matrice triangolare inferiore memorizzata secondo uno schema *packed*, utilizzando l'algoritmo di back substitution particolarizzato per lo schema di memorizzazione in uso. La complessità di tempo è quindi invariata, mentre la complessità di spazio è quasi dimezzata, risultando $O(n^2/2)$ anziché $O(n^2)$. La Procedura 2.13 calcola il prodotto di una matrice tridiagonale per un vettore, utilizzando per la matrice una memorizzazione con tre array monodimensionali. Essa differisce da una procedura per il prodotto di una matrice generica per un vettore non solo per il modo in cui si fa riferimento agli elementi della matrice, ma anche per il fatto che non si eseguono le operazioni che coinvolgono gli elementi (nulli) al di fuori delle tre diagonali. Ciò è una conseguenza immediata dello schema di memorizzazione utilizzato per la matrice tridiagonale. La complessità di tempo della Procedura 2.13 risulta quindi $O(n)$ operazioni aritmetiche floating-point, mentre nel caso generale è $O(n^2)$. Analogamente, la complessità di spazio di tale procedura è $O(n)$, invece di $O(n^2)$.

procedure trisol (**in:** n, lp, b; **out:** $b, ierr$)

/# SCOPO: risolve un sistema lineare con matrice
triangolare inferiore in formato packed

/# SPECIFICHE DEI PARAMETRI:

/# PARAMETRI DI INPUT:

var: n	: **intero**	{ *dimensione del sistema* }
var: $lp(n(n+1)/2)$	: **reale**	{ *matrice del sistema* }
var: $b(n)$	: **reale**	{ *vettore dei termini noti* }

/# PARAMETRI DI OUTPUT:

var: $b(n)$	: **reale**	{ *vettore soluzione* }
var: $ierr$	: **intero**	{ *indice del primo elemento* }
		{ *diagonale nullo* }

Procedura 2.12: Risoluzione di un sistema lineare
con matrice triangolare inferiore in formato packed - continua

```
/# VARIABILI LOCALI:
var: i, j, idiag, irow   : interi

/# INIZIO ISTRUZIONI:
ierr := 0
i := 1
repeat                                   { controllo della singolaritá }
   idiag := i * (i - 1)/2 + i - 1
   if  (lp(idiag) = 0) then
      ierr := i
   endif
   i := i + 1
until  (ierr ≠ 0 or i = n + 1)
if ( ierr ≠ 0 ) then                     { risoluzione del sistema }
   b(1) := b(1)/lp(1)
   for i = 2 to n
      irow := i(i - 1)/2
      for j = 1 to i - 1
         b(i) := b(i) - lp(irow + j) * b(j)     { calcolo della soluzione }
                                                { memorizzata in b}
      endfor
      b(i) := b(i)/lp(irow + i)
  endfor
endif
end trisol
```

Procedura 2.12: Risoluzione di un sistema lineare
con matrice triangolare inferiore in formato packed - fine

```
procedure tridvet (in: n, ad, al, au, x; out: y)

 /# SCOPO: calcola il prodotto di una matrice A,
           tridiagonale quadrata, per un vettore x.
           La matrice è memorizzata in tre vettori.

 /# SPECIFICHE DEI PARAMETRI:
 /# PARAMETRI DI INPUT:
 var: n          : intero  { dimensione del vettore }
 var: ad(n)      : reale   { diagonale principale }
                           { della matrice A}
 var: al(n-1)    : reale   { diagonale inferiore }
                           { della matrice A}
 var: au(n-1)    : reale   { diagonale superiore }
                           { della matrice A}
 var: x(n)       : reale   { secondo fattore del prodotto}
 /# PARAMETRI DI OUTPUT:
 var: y(n)       : reale   { vettore soluzione }
 /# VARIABILI LOCALI:
 var: i          : interi

 /# INIZIO ISTRUZIONI:
   for i = 2 to n - 1                  {componenti i = 2, ... , n - 1 }
     y(i) := al(i) * x(i - 1) + ad(i) * x(i) + au(i) * x(i + 1)
   endfor
   y(1) := ad(1) * x(1) + au(1) * x(2)          {componente i=1 }
   y(n) := al(n) * x(n - 1) + ad(n) * x(n)      {componente i=n }
end tridvet
```

Procedura 2.13: Calcolo del prodotto di una matrice tridiagonale per un vettore. La matrice è memorizzata usando tre array monodimensionali

2.11 Risoluzione di sistemi lineari con matrice a banda

Si consideri un sistema lineare $Ax = b$, dove A è una matrice a banda. Per risolvere tale sistema si può eseguire la fattorizzazione LU di A ed applicare la back e la forward substitution ai sistemi lineari triangolari associati. Si vuole vedere se le matrici L ed U derivanti dalla fattorizzazione sono dotate di qualche struttura, dalla quale si possa trarre vantaggio in termini di complessità computazionale.

Nei prossimi paragrafi si analizza dapprima la fattorizzazione LU senza pivoting e poi quella con pivoting parziale.

2.11.1 Fattorizzazione LU senza pivoting di una matrice tridiagonale e risoluzione dei sistemi lineari associati

L'esempio che segue mostra in che modo la fattorizzazione LU senza pivoting *"agisce"* su una matrice tridiagonale.

♣ **Esempio 2.43.**

Si vuole eseguire la fattorizzazione LU senza pivoting della matrice tridiagonale

$$A = \begin{pmatrix} 5 & -3 & & \\ 1 & 4 & -2 & \\ & -1 & 3 & 1 \\ & & 2 & 5 \end{pmatrix}.$$

Tale fattorizzazione si può ottenere con l'algoritmo di eliminazione di Gauss, cioè con i passi seguenti:

passo 1:

- calcolo dei moltiplicatori relativi alla prima colonna: $m_{2,1} = 1/5$ ($m_{3,1} = m_{4,1} = 0$);
- trasformazione della sottomatrice attiva:

$$A^{(1)} = \begin{pmatrix} 5 & -3 & & \\ 0 & 23/5 & -2 & \\ & -1 & 3 & 1 \\ & & 2 & 5 \end{pmatrix};$$

passo 2:

- calcolo dei moltiplicatori relativi alla seconda colonna: $m_{3,2} = -5/23$ ($m_{4,2} = 0$);
- trasformazione della sottomatrice attiva:

$$A^{(2)} = \begin{pmatrix} 5 & -3 & & \\ 0 & 23/5 & -2 & \\ & 0 & 59/23 & 1 \\ & & 2 & 5 \end{pmatrix};$$

passo 3:

- calcolo del moltiplicatore relativo alla quarta colonna: $m_{4,3} = 46/59$;
- trasformazione della sottomatrice attiva:

$$A^{(3)} = \begin{pmatrix} 5 & -3 & & \\ 0 & 23/5 & -2 & \\ & 0 & 59/23 & 1 \\ & & 0 & 249/59 \end{pmatrix}.$$

Si ha quindi:

$$A = LU$$

con

$$L = \begin{pmatrix} 1 & & & \\ 1/5 & 1 & & \\ & -5/23 & 1 & \\ & & 46/59 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 5 & -3 & & \\ & 23/5 & -2 & \\ & & 59/23 & 1 \\ & & & 249/59 \end{pmatrix},$$

cioè L e U sono matrici *bidiagonali*. ♣

In generale, i fattori L ed U di una matrice tridiagonale (che ammette la fattorizzazione LU) sono bidiagonali. Tale risultato si può dimostrare in maniera indipendente dall'algoritmo utilizzato per calcolare L ed U.

♣ Esempio 2.44.

Sia A una matrice tridiagonale non singolare di dimensione 4×4:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & & \\ a_{2,1} & a_{2,2} & a_{2,3} & \\ & a_{3,2} & a_{3,3} & a_{3,4} \\ & & a_{4,3} & a_{4,4} \end{pmatrix},$$

e siano

$$L = \begin{pmatrix} 1 & & & \\ l_{2,1} & 1 & & \\ l_{3,1} & l_{3,2} & 1 & \\ l_{4,1} & l_{4,2} & l_{4,3} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ & u_{2,2} & u_{2,3} & u_{2,4} \\ & & u_{3,3} & u_{3,4} \\ & & & u_{4,4} \end{pmatrix}$$

le matrici ottenute applicando la fattorizzazione LU senza pivoting ad A. Calcolando il prodotto LU si ottiene la matrice

$$LU = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ l_{2,1}u_{1,1} & l_{2,1}u_{1,2} + u_{2,2} & l_{2,1}u_{1,3} + u_{2,3} & l_{2,1}u_{1,4} + u_{2,4} \\ l_{3,1}u_{1,1} & l_{3,1}u_{1,2} + l_{3,2}u_{2,2} & l_{3,1}u_{1,3} + l_{3,2}u_{2,3} + u_{3,3} & l_{3,1}u_{1,4} + l_{3,2}u_{2,4} + u_{3,4} \\ l_{4,1}u_{1,1} & l_{4,1}u_{1,2} + l_{4,2}u_{2,2} & l_{4,1}u_{1,3} + l_{4,2}u_{2,3} + l_{4,3}u_{3,3} & l_{4,1}u_{1,4} + l_{4,2}u_{2,4} + l_{4,3}u_{3,4} + u_{4,4} \end{pmatrix}$$

e quindi, uguagliando gli elementi di LU con i corrispondenti elementi di A, si ricava:[18]

$$\begin{array}{lcl} & & u_{1,3}=0; \\ & & u_{1,4}=0; \\ l_{2,1}u_{1,4}+u_{2,4}=0 & \Longrightarrow & u_{2,4}=0; \\ l_{3,1}u_{1,1}=0 & \Longrightarrow & l_{3,1}=0; \\ l_{4,1}u_{1,1}=0 & \Longrightarrow & l_{4,1}=0; \\ l_{4,1}u_{1,2}+l_{4,2}u_{2,2}=0 & \Longrightarrow & l_{4,2}=0. \end{array}$$

Risulta dunque:

$$L=\begin{pmatrix} 1 & & & \\ l_{2,1} & 1 & & \\ 0 & l_{3,2} & 1 & \\ 0 & 0 & l_{4,3} & 1 \end{pmatrix}, \quad U=\begin{pmatrix} u_{1,1} & u_{1,2} & 0 & 0 \\ & u_{2,2} & u_{2,3} & 0 \\ & & u_{3,3} & u_{3,4} \\ & & & u_{4,4} \end{pmatrix}.$$

ovvero L e U sono bidiagonali. Inoltre, uguagliando gli elementi delle tre diagonali di LU con i corrispondenti elementi delle tre diagonali di A e ricordando che sono nulli gli elementi $u_{i,j}$ con $i>j$ e gli elementi $l_{i,j}$ con $i<j$, si ricavano le espressioni dei rimanenti elementi di L e di U:

$$\begin{array}{lcl} & & u_{1,1}=a_{1,1}; \\ & & u_{1,2}=a_{1,2}; \\ l_{2,1}u_{1,1}=a_{2,1} & \Longrightarrow & l_{2,1}=a_{2,1}/u_{1,1}; \\ l_{2,1}u_{1,2}+u_{2,2}=a_{2,2} & \Longrightarrow & u_{2,2}=a_{2,2}-l_{2,1}u_{1,2}=a_{2,2}-l_{2,1}a_{1,2}; \\ l_{2,1}\underbrace{u_{1,3}}_{=0}+u_{2,3}=a_{2,3} & \Longrightarrow & u_{2,3}=a_{2,3}; \\ \underbrace{l_{3,1}}_{=0}u_{1,2}+l_{3,2}u_{2,2}=a_{3,2} & \Longrightarrow & l_{3,2}=a_{3,2}/u_{2,2}; \\ \underbrace{l_{3,1}}_{=0}\underbrace{u_{1,3}}_{=0}+l_{3,2}u_{2,3}+u_{3,3}=a_{3,3} & \Longrightarrow & u_{3,3}=a_{3,3}-l_{3,2}u_{2,3}=a_{3,3}-l_{3,2}a_{2,3}; \\ \underbrace{l_{3,1}}_{=0}\underbrace{u_{1,4}}_{=0}+l_{3,2}\underbrace{u_{2,4}}_{=0}+u_{3,4}=a_{3,4} & \Longrightarrow & u_{3,4}=a_{3,4}; \\ \underbrace{l_{4,1}}_{=0}\underbrace{u_{1,3}}_{=0}+\underbrace{l_{4,2}}_{=0}u_{2,3}+l_{4,3}u_{3,3}=a_{4,3} & \Longrightarrow & l_{4,3}=a_{4,3}/u_{3,3}=0; \\ \underbrace{l_{4,1}}_{=0}\underbrace{u_{1,4}}_{=0}+\underbrace{l_{4,2}}_{=0}\underbrace{u_{2,4}}_{=0}+l_{4,3}u_{3,4}+u_{4,4}=a_{4,4} & \Longrightarrow & u_{4,4}=a_{4,4}-l_{4,3}u_{3,4}=a_{4,4}-l_{4,3}a_{3,4}. \end{array}$$

In particolare, gli elementi della diagonale superiore di U sono uguali a quelli della diagonale superiore di A. ♣

Il ragionamento dell'esempio precedente si può applicare ad una generica matrice tridiagonale A di dimensione, $n\times n$:

$$A=\begin{pmatrix} d_1 & f_1 & & & \\ c_2 & d_2 & f_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & d_{n-1} & f_{n-1} \\ & & & c_n & d_n \end{pmatrix}. \tag{2.50}$$

[18] Si noti che $u_{1,1}\neq 0$ e $u_{2,2}\neq 0$, altrimenti A risulterebbe singolare.

Siano

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

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & \dots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \dots & \dots & u_{2,n} \\ 0 & 0 & u_{3,3} & \dots & \dots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 0 & u_{n,n} \end{pmatrix} \tag{2.51}$$

le matrici ottenute applicando la fattorizzazione LU senza pivoting ad A. Calcolando il prodotto LU si ottiene la matrice

$$LU = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & u_{1,n} \\ l_{2,1}u_{1,1} & l_{2,1}u_{1,2} + u_{2,2} & l_{2,1}u_{1,3} + u_{2,3} & \dots & l_{2,1}u_{1,n} + u_{2,n} \\ l_{3,1}u_{1,1} & l_{3,1}u_{1,2} + l_{3,2}u_{2,2} & l_{3,1}u_{1,3} + l_{3,2}u_{2,3} + u_{3,3} & \dots & l_{3,1}u_{1,n} + l_{3,2}u_{2,n} + u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n,1}u_{1,1} & l_{n,1}u_{1,2} + l_{n,2}u_{2,2} & l_{n,1}u_{1,3} + l_{n,2}u_{2,3} + l_{n,3}u_{3,3} & \dots & l_{n,1}u_{1,n} + \dots + u_{n,n} \end{pmatrix}$$

e quindi, uguagliando gli elementi di LU con i corrispondenti elementi di A, si

ricava:[19]

$$
\begin{array}{lll}
 & & u_{1,3}=0;\\
 & & u_{1,4}=0;\\
 & & \vdots\\
 & & u_{1,n}=0;\\
l_{2,1}u_{1,4}+u_{2,4}=0 & \Longrightarrow & u_{2,4}=0;\\
\vdots & & \vdots\\
l_{2,1}u_{1,n}+u_{2,n}=0 & \Longrightarrow & u_{2,n}=0;\\
l_{3,1}u_{1,1}=0 & \Longrightarrow & l_{3,1}=0;\\
\vdots & & \vdots\\
l_{3,1}u_{1,5}+l_{3,2}u_{2,5}+u_{3,5}=0 & \Longrightarrow & u_{3,5}=0;\\
\vdots & & \vdots\\
l_{n,1}u_{1,1}=0 & \Longrightarrow & l_{n,1}=0;\\
\vdots & & \vdots\\
l_{n,1}u_{1,n-2}+l_{n,2}u_{2,n-2}+\ldots+l_{n,n-2}u_{n-2,n-2}=0 & \Longrightarrow & l_{n,n-2}=0.
\end{array}
$$

Risulta dunque:

$$
L=\begin{pmatrix}
1 & 0 & \ldots & \ldots & \ldots & 0\\
l_{2,1} & 1 & 0 & \ldots & \ldots & 0\\
0 & l_{3,2} & 1 & \ldots & \ldots & 0\\
0 & 0 & l_{4,3} & 1 & \ldots & 0\\
\vdots & \vdots & \vdots & \ddots & \ddots & \vdots\\
0 & \ldots & \ldots & \ldots & l_{n,n-1} & 1
\end{pmatrix},
$$

$$
U=\begin{pmatrix}
u_{1,1} & u_{1,2} & 0 & \ldots & \ldots & 0\\
0 & u_{2,2} & u_{2,3} & 0 & \ldots & 0\\
0 & 0 & u_{3,3} & u_{3,4} & 0 & 0\\
\vdots & \vdots & \vdots & \ddots & \ddots & \vdots\\
0 & \ldots & \ldots & \ldots & u_{n-1,n-1} & u_{n-1,n}\\
0 & \ldots & \ldots & \ldots & \ldots & u_{n,n}
\end{pmatrix}.
$$

ovvero L e U sono bidiagonali.

Inoltre, uguagliando gli elementi delle tre diagonali di LU con i corrispondenti elementi delle tre diagonali di A e ricordando che sono nulli gli elementi $u_{i,j}$ con $i>j$ e gli elementi $l_{i,j}$ con $i<j$, si ricavano le espressioni dei rimanenti elementi di L e di U.

[19] Si noti che $u_{1,1}\neq 0$ e $u_{2,2}\neq 0$, altrimenti A risulterebbe singolare.

Riscrivendo le due matrici bidiagonali come:

$$L = \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-1} & 1 & \\ & & & l_n & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_1 & f_1 & & & \\ & u_2 & f_2 & & \\ & & \ddots & \ddots & \\ & & & u_{n-1} & f_{n-1} \\ & & & & u_n \end{pmatrix}; \tag{2.52}$$

si avrà:

$$\begin{array}{ll} d_1 = u_1, & c_2 = l_2 u_1, \\ d_2 = l_2 f_1 + u_2, & c_3 = l_3 u_2, \\ \vdots & \vdots \\ d_n = l_n f_{n-1} + u_n, & c_n = l_n u_{n-1}, \end{array}$$

da cui si ricavano le seguenti espressioni per u_i e l_i:

$$\begin{array}{lll} u_1 = d_1, & u_i = d_i - l_i f_{i-1} & (i = 2, \ldots, n), \\ & l_i = c_i / u_{i-1} & (i = 2, \ldots, n). \end{array}$$

In particolare si osservi che gli elementi della diagonale superiore di U coincidono con quelli della diagonale superiore di A.
Le espressioni degli elementi di L e di U si possono ricavare anche applicando ad A l'algoritmo di eliminazione di Gauss formulato per una generica matrice e tenendo conto del fatto che gli elementi di L al di sotto della prima diagonale inferiore e gli elementi di U al di sopra della prima diagonale superiore sono necessariamente nulli. L'algoritmo per la fattorizzazione LU di una matrice tridiagonale consiste quindi nel calcolare, al generico passo i, la i-ma componente della diagonale inferiore di L e la i-ma componente della diagonale principale di U, come di seguito mostrato nella Procedura 2.14.

```
procedure lutrid (in: n, d, c, f; out: l, u)

/# SCOPO: effettua la fattorizzazione LU senza
           pivoting di una matrice A tridiagionale.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n      : intero  { prima dimensione }
                      { della matrice }
var: d(n)   : reale   { diagonale principale }
                      { della matrice A}
var: c(n)   : reale   { diagonale inferiore }
                      { della matrice A}
var: f(n)   : reale   { diagonale superiore }
                      { della matrice A}
/# PARAMETRI DI OUTPUT:
var: l(n)   : reale   { matrice triangolare}
                      { inferiore }
var: u(n)   : reale   { matrice triangolare }
                      { superiore }

/# VARIABILI LOCALI:
var: i      : interi

/# INIZIO ISTRUZIONI:
u(1) := d(1)                  { calcolo primo elemento}
for i = 2 to n                { fattorizzazione LU }
   l(i) := c(i)/u(i − 1)
   u(i) := d(i) − l(i) ∗ f(i − 1)
endfor
end lutrid
```

Procedura 2.14: Fattorizzazione LU senza pivoting di una matrice tridiagonale

Si osservi che il precedente algoritmo non altera la struttura della matrice, nel senso che non trasforma elementi nulli in elementi non nulli.

Nella Procedura 2.14 si eseguono $2n-2$ moltiplicazioni o divisioni floating-point e $n-1$ addizioni floating-point; la complessità di tempo è quindi

$$T_{LUtrid}(n) = 3n - 3 = O(n) \ \ flop\,,$$

dove $flop$ indica la generica operazione aritmetica floating-point. La struttura della matrice consente dunque di ottenere un notevole vantaggio computazionale rispetto alla fattorizzazione di una matrice generica, la cui complessità di tempo è $O(n^3)$ $flop$.

Per quanto riguarda la complessità di spazio, si osservi che è possibile memorizzare gli elementi l_i ed u_i nelle stesse variabili utilizzate rispettivamente per c_i e d_i, in quanto, per ogni i, c_i interviene solo nel calcolo di l_i, e d_i solo in quello di u_i. Con tale strategia, la complessità di spazio dell'algoritmo di fattorizzazione LU risulta essere:

$$S_{LUtrid}(n) = 3n - 2 = O(n).$$

Si consideri ora il sistema lineare $Ax = b$, dove A è una matrice tridiagonale a cui è stata applicata la fattorizzazione LU. Per calcolare la soluzione di tale sistema, bisogna risolvere i sistemi triangolari bidiagonali $Ly = b$ ed $Ux = y$. Da $Ly = b$ si ricava:

$$\begin{aligned} y_1 &= b_1; \\ y_2 &= b_2 - l_2 y_1; \\ &\vdots \\ y_n &= b_n - l_n y_{n-1}; \end{aligned}$$

mentre da $Ux = y$ si ricava:

$$\begin{aligned} x_n &= y_n/u_n; \\ x_{n-1} &= (y_{n-1} - f_{n-1}x_n)/u_{n-1}; \\ &\vdots \\ x_1 &= (y_1 - f_1 x_2)/u_1. \end{aligned}$$

Gli algoritmi di back e forward substitution applicati alle matrici bidiagonali (2.52) si particolarizzano quindi nel modo illustrato nelle Procedure 2.15 e 2.16.

```
procedure fstrid (in: n, l, b; out: y)

/# SCOPO: effettua la forward substitution di un sistema
           la cui matrice è triangolare inferiore bidiagonale.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n     : intero   { dimensione del sistema }
var: l(n)  : reale    { contiene gli elementi non nulli }
                      { della matrice del sistema }
var: b(n)  : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: y(n)  : reale    { vettore risultante }
/# VARIABILI LOCALI:
var: i     : interi

/# INIZIO ISTRUZIONI:
y(1) := b(1)
for i = 2 to n
   y(i) := b(i) - l(i) * y(i - 1)
endfor
end fstrid
```

Procedura 2.15: Forward substitution per un sistema con matrice triangolare inferiore bidiagonale, derivante dalla fattorizzazione LU senza pivoting di una matrice tridiagonale

```
procedure bstrid (in: n, u, b; out: y)

/# SCOPO: effettua la backward substitution di un sistema
           la cui matrice è triangolare superiore bidiagonale.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n      : intero   { dimensione del sistema }
var: u(n)   : reale    { contiene gli elementi non nulli }
                       { della matrice del sistema }
var: b(n)   : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: y(n)   : reale    { vettore risultante }
/# VARIABILI LOCALI:
var: i      : interi

/# INIZIO ISTRUZIONI:
x(n) := y(n)/u(n)
for i = n - 1 to 1 by -1
   x(i) := (y(i) - f(i) * x(i+1))/u(i)
endfor
end bstrid
```

Procedura 2.16: Backward substitution per un sistema con matrice triangolare superiore bidiagonale, derivante dalla fattorizzazione LU senza pivoting di una matrice tridiagonale

Nella Procedura 2.15 si eseguono $n-1$ moltiplicazioni ed $n-1$ addizioni floating-point, mentre nella Procedura 2.16 si eseguono $2n-1$ moltiplicazioni o divisioni floating-point e $n-1$ addizioni. La complessità di tempo dei due algoritmi è quindi:

$$T_{Ffstrid}(n) = T_{Bbstrid}(n) = O(n) \ \ flop.$$

Nei due algoritmi è inoltre possibile memorizzare la soluzione sovrascrivendo il vettore dei termini noti, senza dover ricorrere all'uso di un'area di memoria aggiuntiva. La complessità di spazio è dunque

$$S_{fstrid}(n) = S_{bstrid}(n) = O(n) \ \ flop.$$

In conclusione, le complessità di spazio e di tempo di un algoritmo per la risoluzione (mediante fattorizzazione LU e back e forward substitution) di un sistema lineare con matrice tridiagonale di dimensione $n \times n$ sono:

$$T_{RIStrid}(n) = O(n),$$
$$S_{RIStrid}(n) = O(n).$$

2.11.2 Fattorizzazione LU senza pivoting di una matrice a banda e risoluzione dei sistemi lineari associati

I risultati ottenuti per una matrice tridiagonale si estendono al caso di una generica matrice a banda.

♣ **Esempio 2.45.** Si esegua la fattorizzazione LU senza pivoting della matrice a banda:

$$A = \begin{pmatrix} 1 & 2 & & & \\ 2 & -1 & 1 & & \\ 3 & 1 & 3 & 1 & \\ & 1 & 2 & -2 & 1 \\ & & -1 & 2 & 1 \end{pmatrix},$$

che ha ampiezza di banda superiore $q = 1$ e ampiezza di banda inferiore $p = 2$. Si ha:

passo 1:

- calcolo dei moltiplicatori: $m_{2,1} = 2$, $m_{3,1} = 3$ ($m_{4,1} = m_{5,1} = 0$);
- trasformazione della sottomatrice attiva:

$$A^{(1)} = \begin{pmatrix} 1 & 2 & & & \\ & -5 & 1 & & \\ & -5 & 3 & 1 & \\ & 1 & 2 & -2 & 1 \\ & & -1 & 2 & 1 \end{pmatrix};$$

passo 2:

- calcolo dei moltiplicatori: $m_{3,2} = 1$, $m_{4,2} = -1/5$ ($m_{5,2} = 0$);
- trasformazione della sottomatrice attiva:

$$A^{(2)} = \begin{pmatrix} 1 & 2 & & & \\ & -5 & 1 & & \\ & & 2 & 1 & \\ & & 11/5 & -2 & 1 \\ & & -1 & 2 & 1 \end{pmatrix};$$

passo 3:

- calcolo dei moltiplicatori: $m_{4,3} = 11/10$, $m_{5,3} = -1/2$;
- trasformazione della sottomatrice attiva:

$$A^{(3)} = \begin{pmatrix} 1 & 2 & & & \\ & -5 & 1 & & \\ & & 2 & 1 & \\ & & & -31/10 & 1 \\ & & & 5/2 & 1 \end{pmatrix};$$

passo 4:

- calcolo del moltiplicatore: $m_{5,4} = -25/31$;
- trasformazione della sottomatrice attiva:

$$A^{(4)} = \begin{pmatrix} 1 & 2 & & & \\ & -5 & 1 & & \\ & & 2 & 1 & \\ & & & -31/10 & 1 \\ & & & & 56/31 \end{pmatrix}.$$

Si ha quindi:

$$A = LU$$

con

$$L = \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ 3 & 1 & 1 & & \\ & -1/5 & 11/10 & 1 & \\ & & -1/2 & -25/31 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 2 & & & \\ & -5 & 1 & & \\ & & 2 & 1 & \\ & & & -31/10 & 1 \\ & & & & 56/31 \end{pmatrix};$$

ovvero L ha ampiezza di banda $p = 2$ ed U ha ampiezza di banda $q = 1$. ♣

In generale sussiste il risultato seguente che include il caso delle matrici tridiagonali:

Teorema 2.8. *Sia A una matrice a banda di dimensione $n \times n$, con ampiezza di banda inferiore p ed ampiezza di banda superiore q. Se A ammette la fattorizzazione LU, allora L è una matrice triangolare inferiore a banda, con ampiezza di banda inferiore p, ed U è una matrice triangolare superiore a banda, con ampiezza di banda superiore q.*

Dimostrazione

La dimostrazione procede per induzione su n. Per $n = 1$ la tesi è banalmente vera. Si suppenga quindi $n > 1$. Scrivendo la fattorizzazione nella forma:

$$A = \begin{pmatrix} \alpha & w^T \\ v & B \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ v/\alpha & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & B - vw^T/\alpha \end{pmatrix} \begin{pmatrix} \alpha & w^T \\ 0 & I_{n-1} \end{pmatrix}$$

È facile osservare che $B - vw^T/\alpha$ ha ampiezza di banda superiore q e ampiezza di banda inferiore p. Sia quindi L_1U_1 la fattorizzazione LU della matrice $B - vw^T/\alpha$. Utilizzando le ipotesi di induzione e osservando che le prime q componenti di w e le prime p di v sono nulle, segue che:

$$L = \begin{pmatrix} 1 & 0 \\ v/\alpha & L_1 \end{pmatrix} \quad U = \begin{pmatrix} \alpha & w^T \\ 0 & U_1 \end{pmatrix}$$

hanno ampiezza di banda rispettivamente p e q e sono tali che $A = LU$. ■

Come già osservato per le matrici tridiagonali, al medesimo risultato si giunge se si applica l'algoritmo di eliminazione di Gauss formulato per una generica matrice e si osserva che la struttura a banda della matrice da fattorizzare implica che siano nulli gli elementi di L al di sotto della p-ma diagonale inferiore e quelli di U al di sopra della q-ma diagonale superiore (cfr. esempio 2.44).

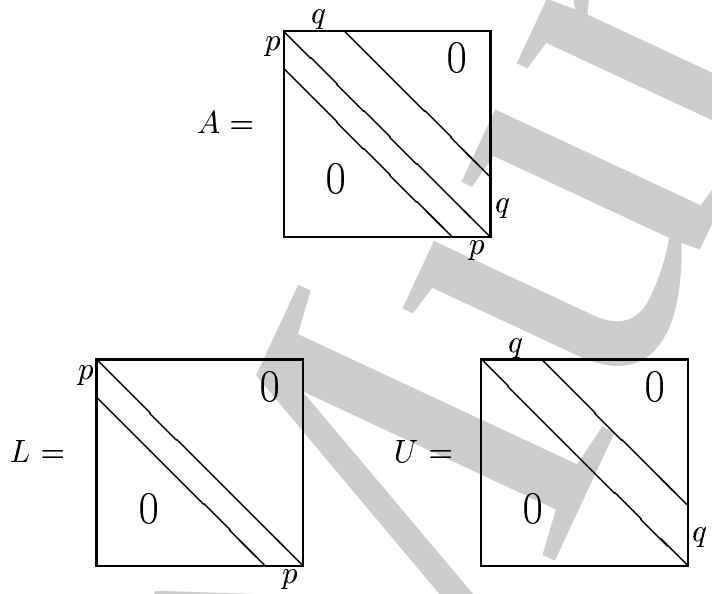

Figura 2.23: Fattorizzazione LU senza pivoting di una matrice a banda.

La fattorizzazione LU senza pivoting di una matrice a banda produce quindi fattori che conservano le ampiezze di banda della matrice di partenza (una rappresentazione grafica di ciò è fornita in Fig. 2.23). Al generico passo k dell'eliminazione di Gauss si calcolano al piú p moltiplicatori, uno per ciascun elemento della banda inferiore nella k-ma colonna, e si trasformano gli elementi della sottomatrice attiva che appartengono alle righe $k+1, \ldots, \min(k+p, n)$ ed alle colonne $k+1, \ldots, \min(k+q, n)$. Tale algoritmo può essere eseguito utilizzando per la matrice A lo schema di memorizzazione a banda di cui è parlato nel §2.10 e scrivendo gli elementi di L e di U sui corrispondenti elementi di A (algoritmo *in place*), come mostrato[20] nella Procedura 2.17. Gli elementi della

[20]Si noti che gli indici delle componenti dell'array AB possono essere scritti in maniera piú "compatta", riducendo il numero di operazioni aritmetiche intere e quindi migliorando l'efficienza dell'algoritmo. Ad esempio, facendo variare l'indice del ciclo su i nel modo seguente:

$$\texttt{for } i := q+2 \texttt{ to } \min(q+p+1, q+n-k+1) \ ,$$

diagonale principale di L non sono, ovviamente, memorizzati.

procedure bandlu (**in:** n, AB, p, q; **out:** AB)

/# SCOPO: effettua la fattorizzazione LU di una matrice A a banda memorizzata secondo lo schema band storage.

/# SPECIFICHE DEI PARAMETRI:

/# PARAMETRI DI INPUT:

var:	n	: **intero**	{ *prima dimensione della matrice A* }
var:	$AB(p+q+1, n)$	: **reale**	{ *matrice da fattorizzare* }
var:	p	: **intero**	{ *ampiezza della banda inferiore di A* }
var:	q	: **intero**	{ *ampiezza della banda superiore di A* }

/# PARAMETRI DI OUTPUT:

var:	$AB(p+q+1, n)$	: **reale**	{ *matrice fattorizzata* }

/# VARIABILI LOCALI:

var: i, j, k : **interi**

Procedura 2.7: Fattorizzazione LU senza pivoting di una matrice a banda memorizzata secondo lo schema *band storage* - continua

i moltiplicatori si possono calcolare con l'istruzione

$$AB(i,k) := AB(i,k)/AB(q+1,k)\,;$$

si può inoltre precalcolare il valore di $q+1$, con un'istruzione del tipo $r := q+1$, e quindi scrivere:

$$AB(i,k) := AB(i,k)/AB(r,k)\,.$$

```
/# INIZIO ISTRUZIONI:
for k = 1 to n − 1
   for i = k + 1 to min(k + p, n)                    { calcolo moltipl.}
      AB(q + 1 + i − k, k) := AB(q + 1 + i − k, k)/AB(q + 1, k)
   endfor

   for j = k + 1 to min(k + q, n)
      for i = k + 1 to min(k + p, n)
         AB(q + 1 + i − j, j) := AB(q + 1 + i − j, j)+
               −AB(q + 1 + i − k, k) ∗ AB(q + 1 + k − j, j)
      endfor
   endfor
endfor
end bandlu
```

Procedura 2.17: Fattorizzazione LU senza pivoting di una matrice a banda memorizzata secondo lo schema *band storage* - fine

Si osservi che nell'algoritmo precedente la trasformazione della sottomatrice attiva è eseguita accedendo agli elementi per colonne, cioè si trasformano una dopo l'altra le colonne di tale matrice (il ciclo sull'indice di riga i è all'interno del ciclo sull'indice di colonna j). Scambiando i cicli su i e j si ottiene un algoritmo che accede alla sottomatrice attiva per righe, analogo a quello per la fattorizzazione LU di una matrice generica. I due algoritmi sono equivalenti, cioè producono gli stessi risultati, ed hanno la stessa complessità computazionale; la scelta dell'uno o dell'altro dipende essenzialmente dal linguaggio in cui tale algoritmo è implementato. Se, ad esempio, si usa il Fortran 77, in cui gli array sono memorizzati per colonne, l'algoritmo che accede agli elementi dell'array AB per colonne consente un utilizzo piú efficiente della memoria.

♣ Esempio 2.46.

Sia A una matrice a banda di dimensione 10×10, con ampiezza di banda inferiore $p = 2$ ed ampiezza di banda superiore $q = 4$:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & & & & \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} & & & \\ & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & a_{4,8} & & \\ & & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & a_{5,8} & a_{5,9} & \\ & & & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} & a_{6,9} & a_{6,10} \\ & & & & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} & a_{7,9} & a_{7,10} \\ & & & & & a_{8,6} & a_{8,7} & a_{8,8} & a_{8,9} & a_{8,10} \\ & & & & & & a_{9,7} & a_{9,8} & a_{9,9} & a_{10,10} \\ & & & & & & & a_{10,8} & a_{10,9} & a_{10,10} \end{pmatrix}.$$

Contiamo il numero di operazioni aritmetiche floating-point che si eseguono applicando la Procedura 2.17 a tale matrice.

L'algoritmo considerato consta di $n-1$ passi, individuati dal contatore k; il numero di operazioni varia, al variare di k, nel modo seguente:

- per $k = 1, \ldots, 6 \;\; (6 = n - q)$ si ha:

$$\min(k+p, n) = \min(k+2, 10) = k+2\,,$$
$$\min(k+q, n) = \min(k+3, 10) = k+3$$

  e quindi al passo k si eseguono

  - $p = 2$ divisioni per calcolare i moltiplicatori,
  - $pq = 8$ addizioni ed altrettante moltiplicazioni per trasformare la sottomatrice attiva,

  per un totale di

$$6(2 + 2 \cdot 8) = 108 \quad flop\,;$$

- per $k = 7, 8 \;\; (7 = n - q + 1, \;\; 8 = n - p)$ si ha:

$$\min(k+p, n) = \min(k+2, 10) = k+2\,,$$
$$\min(k+q, n) = \min(k+3, 10) = 10$$

  e quindi al generico passo k si eseguono

  - $p = 2$ divisioni per calcolare i moltiplicatori,
  - $p(n-k) = 2(10-k)$ addizioni ed altrettante moltplicazioni per trasformare la sottomatrice attiva,

  per un totale di

$$\sum_{k=7}^{8} (2 + 2 \cdot 2 \cdot (10-k)) = 14 + 10 = 24 \quad flop;$$

- per $k = 9 \;\; (9 = n - p + 1 = n - 1)$ si ha:

$$\min(k+p, n) = \min(k+2, 10) = 10\,,$$
$$\min(k+q, n) = \min(k+3, 10) = 10$$

  e quindi si eseguono

  - $n - k = 1$ divisione per calcolare i moltiplicatori,
  - $(n-k)(n-k) = 1$ addizione ed 1 moltplicazione per trasformare la sottomatrice attiva,

  per un totale di

$$3 \quad flop.$$

Nell'algoritmo si eseguono dunque

$$108 + 24 + 3 = 135 \;\; flop\,.$$

Si osservi che tale numero è significativamente inferiore rispetto al numero di operazioni che si eseguono applicando l'algoritmo di eliminazione di Gauss formulato per una generica matrice, che è $(4n^3 - 3n^2 - 7n)/6 = 615$. ♣

Consideriamo una matrice a banda A, di dimensione $n \times n$, con ampiezza di banda inferiore p ed ampiezza di banda superiore q:

$$A = \begin{pmatrix} a_{1,1} & \cdots & \cdots & \cdots & a_{1,q+1} & & & & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & & & \\ a_{p+1,1} & \cdots & \cdots & \cdots & \cdots & \cdots & a_{p+1,q+1+p} & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & a_{n-q,n} \\ & & & & \ddots & \ddots & \ddots & \ddots & \ddots & a_{n-q+1,n} \\ & & & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & & & & \ddots & \ddots & \ddots & \vdots \\ & & & & & & & a_{n,n-p} & \cdots & a_{n,n} \end{pmatrix}.$$

Si suppone che $p < q$, e quindi che $n - q < n - p$; l'algoritmo consta di $n - 1$ passi, individuati dal contatore k; il numero di operazioni varia, al variare di k, nel modo seguente:

- per $k = 1, \ldots, n - q$ risulta

  $$\min(k + p, n) = k + p,$$
  $$\min(k + q, n) = k + q$$

  e quindi al passo k si eseguono

  - p divisioni per calcolare i moltiplicatori,
  - pq addizioni ed altrettante moltiplicazioni per trasformare la sottomatrice attiva,

  per un totale di

  $$(n - q)(p + 2pq) = np + 2npq - pq - 2pq^2 \quad flop;$$

- per $k = n - q + 1, \ldots, n - p$ risulta

  $$\min(k + p, n) = k + p,$$
  $$\min(k + q, n) = n$$

  e quindi al passo k si eseguono

  - p divisioni per calcolare i moltiplicatori,

– $p(n-k)$ addizioni ed altrettante moltplicazioni per trasformare la sottomatrice attiva,

per un totale di

$$\sum_{k=n-q+1}^{n-p}(p+2p(n-k)) = (q-p)p \;+\; 2p\sum_{k=n-q+1}^{n-p}(n-k) =$$
$$= (q-p)p \;+\; 2p\sum_{i=p}^{q-1} i = (q-p)p \;+\; 2p\left(\sum_{i=1}^{q-1} i - \sum_{i=1}^{p-1} i\right) =$$
$$= (q-p)p + p(q(q-1) - p(p-1)) = pq^2 - p^3 \quad flop;$$

- per $k = n-p+1, \ldots, n-1$ risulta

$$\min(k+p, n) = n\,,$$
$$\min(k+q, n) = n$$

e quindi al passo k si eseguono

– $n-k$ divisioni per calcolare i moltiplicatori,

– $(n-k)(n-k)$ addizioni ed altrettante moltplicazioni per trasformare la sottomatrice attiva,

per un totale di

$$\sum_{k=n-p+1}^{n-1}\left((n-k)+2(n-k)^2\right) = \sum_{i=1}^{p-1} i + 2\sum_{i=1}^{p-1} i^2 =$$
$$= \frac{p(p-1)}{2} + \frac{p(p-1)(2p-1)}{3} = \frac{4p^3-3p^2-p}{6} \quad flop.$$

Sommando i valori precedentemente ottenuti, si ha:

$$np + 2npq - pq - 2pq^2 + pq^2 - p^3 + \frac{4p^3-3p^2-q}{6} =$$
$$= 2npq + np - pq^2 - \frac{2p^3+3p^2+p}{6} - pq \quad flop\,.$$

e quindi la complessità di tempo dell'algoritmo descritto nella Procedura 2.17 è :

$$T_{LUband}(n,p,q) = O(npq) \quad flop\,.$$

Si noti che al crescere di p e q la complessità di tempo si avvicina a $O(n^3)$ *flop*, cioè a quella dell'algoritmo per una matrice di dimensione $n \times n$, e che per $p, q \simeq n$, essa è proprio $O(n^3)$ *flop*. L'algoritmo è dunque "significativamente vantaggioso" se $p, q << n$.

La complessità di spazio è quella richiesta dallo schema di memorizzazione a banda (cfr. § 2.10), ovvero

$$S_{LUband}(n, p, q) = (p + q + 1)n = O((p + q + 1)n) .$$

Si consideri ora la risoluzione dei sistemi lineari aventi come matrici dei coefficienti i fattori L ed U calcolati con la Procedura 2.17. Con un ragionamento analogo a quello fatto per le matrici L ed U derivanti da una matrice tridiagonale, si hanno gli algoritmi di back e forward substitution riportati nella Procedura 2.18 e 2.19. Si osservi che gli algoritmi sono in place, cioè il vettore soluzione è riscritto sul vettore dei termini noti. Inoltre, in analogia con l'algoritmo descritto nella Procedura 2.17, l'accesso agli elementi dell'array AB è per colonne; al generico passo j si ottiene il valore della j-ma incognita e si aggiornano i valori delle incognite che devono ancora essere calcolate.

```
procedure bandfs (in: n, AB, p, q, b; out: b)

/# SCOPO : risolve un sistema con matrice a banda A,
           derivante da fattorizzazione LU, utilizzando
           la forward substitution.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n              : intero  { dimensione del sistema }
var: AB(p+q+1, n)   : reale   { matrice da fattorizzare }
var: p              : intero  { ampiezza della banda }
                              { inferiore di A }
var: q              : intero  { ampiezza della banda }
                              { superiore di A }
var: b(n)           : reale   { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: b(n)           : reale   { vettore risultante }
/# VARIABILI LOCALI:
var: i, j           : interi

/# INIZIO ISTRUZIONI:
for j = 1 to n - 1
   for i = j + 1 to min(j + p, n)
      b(i) = b(i) - AB(q + 1 + i - j, j) * b(j)
   endfor
endfor
end bandfs
```

Procedura 2.18 : Forward substitution per un sistema con matrice triangolare inferiore derivante dalla fattorizzazione LU senza pivoting di una matrice a banda

```
procedure bandbs (in: n, AB, p, q, b; out: b)

/# SCOPO: risolve un sistema con matrice a banda A,
           derivante da fattorizzazione LU, utilizzando
           la back substitution.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n              : intero   { dimensione del sistema }
var: AB(p+q+1, n)   : reale    { matrice da fattorizzare }
var: p              : intero   { ampiezza della banda }
                               { inferiore di A }
var: q              : intero   { ampiezza della banda }
                               { superiore di A }
var: b(n)           : reale    { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: b(n)           : reale    { vettore risultante }
/# VARIABILI LOCALI:
var: i, j           : interi

/# INIZIO ISTRUZIONI:
   for j = n to 2 by -1
      b(j) := b(j)/AB(q+1, j)
      for i = max(j-q, 1) to j-1
         b(i) := b(i) - AB(q+1+i-j, j) * b(j)
      endfor
   endfor
   b(1) := b(1)/AB(q+1, 1)
end bandbs
```

Procedura 2.19: Back substitution per un sistema con matrice triangolare superiore derivante dalla fattorizzazione LU senza pivoting di una matrice a banda

Si verifica che la complessità di tempo della Procedura 2.18 è:

$$T_{FORWband}(n,p) = O(np) \quad flop\,,$$

mentre quella della Procedura 2.19 è :

$$T_{BACKband}(n,q) = O(nq) \quad flop\,.$$

Dato che gli elementi delle matrici L ed U sono memorizzati sui corrispondenti elementi della matrice di partenza A, la complessità di spazio è la stessa della fattorizzazione LU, cioè

$$S_{FORWband}(n,p) = S_{BACKband}(n,q) = O((p+q)n)\,.$$

2.11.3 Fattorizzazione LU con pivoting parziale di una matrice a banda e risoluzione dei sistemi lineari associati

Gli algoritmi di fattorizzazione LU considerati nei paragrafi precedenti non utilizzano alcuna strategia di pivoting; in generale, essi risultano quindi instabili. Si vuole allora vedere se la fattorizzazione LU con pivoting parziale genera delle matrici L ed U dotate di qualche struttura. Dato che il pivoting comporta scambi di righe, ci si aspetta che la struttura a banda della matrice di partenza non venga conservata. Gli esempi che seguono mostrano in che modo il pivoting modifica tale struttura.

♣ **Esempio 2.47.**

Si esegua la fattorizzazione LU con pivoting parziale della matrice tridiagonale:

$$A = \begin{pmatrix} 1 & 1 & & \\ 2 & 1 & 1 & \\ & 1 & 1 & 2 \\ & & 2 & 1 \end{pmatrix}.$$

Si ha:

passo 1:

- pivoting in prima colonna: $max_{1\le i\le 4}|a_{i,1}| = 2 = a_{2,1}$ e quindi si scambiano la prima e la seconda riga di A, ottenendo

$$\tilde{A} = \begin{pmatrix} 2 & 1 & 1 & \\ 1 & 1 & & \\ & 1 & 1 & 2 \\ & & 2 & 1 \end{pmatrix};$$

- calcolo dei moltiplicatori: $m_{2,1} = 1/2$ ($m_{3,1} = m_{4,1} = 0$), ovvero

$$L^{(1)} = \begin{pmatrix} 1 & & & \\ 1/2 & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix},$$

  dove $L^{(i)}$ indica la matrice triangolare unitaria inferiore che contiene, nella prime i colonne, i moltiplicatori calcolati nei primi i passi, ed ha tutti gli elementi subdiagonali delle colonne successive alla i-ma uguali a 0;[21]

- trasformazione della sottomatrice attiva:

$$A^{(1)} = \begin{pmatrix} 2 & 1 & 1 & \\ & 1/2 & -1/2 & \\ & 1 & 1 & 2 \\ & & 2 & 1 \end{pmatrix};$$

passo 2:

- pivoting in seconda colonna: $max_{2 \le i \le 4} |a^{(1)}_{i,2}| = 1 = a^{(1)}_{3,2}$ e quindi si scambiano la seconda e la terza riga di $A^{(1)}$ ed $L^{(1)}$, ottenendo

$$\tilde{A}^{(1)} = \begin{pmatrix} 2 & 1 & 1 & \\ & 1 & 1 & 2 \\ & 1/2 & -1/2 & \\ & & 2 & 1 \end{pmatrix}, \quad \tilde{L}^{(1)} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ 1/2 & & 1 & \\ & & & 1 \end{pmatrix};$$

- calcolo dei moltiplicatori: $m_{3,2} = 1/2$ ($m_{4,2} = 0$), ovvero

$$\tilde{L}^{(2)} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ 1/2 & 1/2 & 1 & \\ & & & 1 \end{pmatrix};$$

- trasformazione della sottomatrice attiva:

$$A^{(2)} = \begin{pmatrix} 2 & 1 & 1 & \\ & 1 & 1 & 2 \\ & & -1 & -1 \\ & & 2 & 1 \end{pmatrix};$$

passo 3:

- pivoting in terza colonna: $max_{3 \le i \le 4} |a^{(2)}_{i,3}| = 1 = a^{(2)}_{4,2}$ e quindi si scambiano la terza e la quarta riga di $A^{(2)}$ ed $L^{(2)}$, ottenendo

$$\tilde{A}^{(2)} = \begin{pmatrix} 2 & 1 & 1 & \\ & 1 & 1 & 2 \\ & & 2 & 1 \\ & & -1 & -1 \end{pmatrix}, \quad \tilde{L}^{(2)} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1/2 & 1/2 & & 1 \end{pmatrix};$$

[21] Tale matrice è stata introdotta solo per mostrare in che modo il pivoting agisce sulla matrice L che si ottiene con la fattorizzazione LU.

- calcolo del moltiplicatore: $m_{4,2} = -1/2$, ovvero

$$L^{(3)} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1/2 & 1/2 & -1/2 & 1 \end{pmatrix};$$

- trasformazione della sottomatrice attiva:

$$A^{(3)} = \begin{pmatrix} 2 & 1 & 1 & \\ & 1 & 1 & 2 \\ & & 2 & 1 \\ & & & -1/2 \end{pmatrix};$$

Eseguendo la fattorizzazione LU con pivoting della matrice A si ottiene quindi

$$PA = LU,$$

con

$$L = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1/2 & 1/2 & -1/2 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & 1 & \\ & 1 & 1 & 2 \\ & & 2 & 1 \\ & & & -1/2 \end{pmatrix},$$

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Le matrici L ed U non sono matrici bidiagonali. U è una matrice a banda con ampiezza di banda uguale a 2, cioè uguale alla somma delle ampiezze di banda di A, mentre L è una generica matrice triagolare. Si noti però che ciascuna colonna di L ha al piú un elemento non nullo al di fuori della diagonale principale. ♣

♣ **Esempio 2.48.** Si esegua la fattorizzazione LU con pivoting parziale della matrice a banda, che ha ampiezza di banda inferiore $p = 2$ ed ampiezza di banda superiore $q = 1$:

$$A = \begin{pmatrix} 1 & 1 & & & \\ 2 & -1 & 1 & & \\ 3 & 1 & 3 & 1 & \\ & 1 & 3 & -2 & 1 \\ & & -2 & 1 & 1 \end{pmatrix}.$$

Si ha:

passo 1:

- pivoting in prima colonna: $max_{1 \le i \le 5} |a_{i,1}| = 3 = a_{3,1}$ e quindi si scambiano la prima e la terza riga di A, ottenendo

$$\tilde{A} = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ 2 & -1 & 1 & & \\ 1 & 1 & & & \\ & 1 & 3 & -2 & 1 \\ & & -2 & 1 & 1 \end{pmatrix};$$

- calcolo dei moltiplicatori: $m_{2,1} = 2/3$, $m_{3,1} = 1/3$ ($m_{4,1} = m_{5,1} = 0$), ovvero

$$L^{(1)} = \begin{pmatrix} 1 & & & & \\ 2/3 & 1 & & & \\ 1/3 & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix};$$

- trasformazione della sottomatrice attiva:

$$A^{(1)} = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ & -5/3 & -1 & -2/3 & \\ & 2/3 & -1 & -1/3 & \\ & 1 & 3 & -2 & 1 \\ & & -2 & 1 & 1 \end{pmatrix};$$

passo 2:

- pivoting in seconda colonna: $max_{2 \le i \le 5} |a_{i,2}| = 5/3 = |a_{2,2}|$ e quindi non si eseguono scambi;
- calcolo dei moltiplicatori: $m_{3,2} = -2/5$, $m_{4,2} = -3/5$ ($m_{5,2} = 0$), ovvero

$$L^{(2)} = \begin{pmatrix} 1 & & & & \\ 2/3 & 1 & & & \\ 1/3 & -2/5 & 1 & & \\ & -3/5 & & 1 & \\ & & & & 1 \end{pmatrix};$$

- trasformazione della sottomatrice attiva:

$$A^{(2)} = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ & -5/3 & -1 & -2/3 & \\ & & -7/5 & -3/5 & \\ & & 12/5 & -12/5 & 1 \\ & & -2 & 1 & 1 \end{pmatrix};$$

passo 3:

- pivoting in terza colonna: $max_{3 \le i \le 5} |a_{i,3}| = 12/5 = |a_{4,3}|$ e quindi si scambiano la terza e la quarta riga di $A^{(2)}$ ed $L^{(2)}$, ottenendo

$$\tilde{A}^{(2)} = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ & -5/3 & -1 & -2/3 & \\ & & 12/5 & -12/5 & 1 \\ & & -7/5 & -3/5 & \\ & & -2 & 1 & 1 \end{pmatrix}, \quad \tilde{L}^{(2)} = \begin{pmatrix} 1 & & & & \\ 2/3 & 1 & & & \\ & -3/5 & 1 & & \\ 1/3 & -2/5 & & 1 & \\ & & & & 1 \end{pmatrix};$$

- calcolo dei moltiplicatori: $m_{4,3} = -7/12$, $m_{5,3} = -5/6$, ovvero

$$L^{(3)} = \begin{pmatrix} 1 & & & & \\ 2/3 & 1 & & & \\ & -3/5 & 1 & & \\ 1/3 & -2/5 & -7/12 & 1 & \\ & & -5/6 & & 1 \end{pmatrix};$$

- trasformazione della sottomatrice attiva:

$$A^{(3)} = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ & -5/3 & -1 & -2/3 & \\ & & 12/5 & -12/5 & 1 \\ & & & -2 & 7/12 \\ & & & -1 & 11/6 \end{pmatrix};$$

passo 4:

- pivoting in quarta colonna: $max_{4\le i\le 5}|a_{i,4}| = 2 = |a_{4,4}|$ e quindi non si eseguono scambi;
- calcolo del moltiplicatore: $m_{5,4} = 1/2$, ovvero

$$L^{(4)} = \begin{pmatrix} 1 & & & & \\ 2/3 & 1 & & & \\ & -3/5 & 1 & & \\ 1/3 & -2/5 & -7/12 & 1 & \\ & & -5/6 & 1/2 & 1 \end{pmatrix};$$

- trasformazione della sottomatrice attiva:

$$A^{(4)} = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ & -5/3 & -1 & -2/3 & \\ & & 12/5 & -12/15 & 1 \\ & & & -2 & 7/12 \\ & & & & 37/24 \end{pmatrix}.$$

In conclusione, si ha:

$$PA = LU\,,$$

con

$$L = \begin{pmatrix} 1 & & & & \\ 2/3 & 1 & & & \\ & -3/5 & 1 & & \\ 1/3 & -2/5 & -7/12 & 1 & \\ & & -5/6 & 1/2 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ & -5/3 & -1 & -2/3 & \\ & & 12/5 & -12/5 & 1 \\ & & & -2 & 7/12 \\ & & & & 37/24 \end{pmatrix},$$

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

La matrice U ha ampiezza di banda $3 = p + q$; L ha ampiezza di banda 3, ma ha in ciascuna colonna al piú 2 elementi non nulli oltre a quello diagonale. ♣

Teorema 2.9. *Sia A una matrice a banda di dimensione $n \times n$, con ampiezza di banda inferiore p ed ampiezza di banda superiore q. Eseguendo la fattorizzazione LU con pivoting parziale di A, si ottiene:*

$$PA = LU\,,$$

dove U è una matrice triangolare superiore a banda con ampiezza di banda uguale a $p+q$, L è una matrice triangolare inferiore, con al più p elementi non nulli in ciascuna colonna oltre all'elemento diagonale, e P è una matrice di permutazione.

Dimostrazione Sia $PA = LU$ la fattorizzazione calcolata dall'algoritmo di eliminazione di Gauss con pivoting parziale e si ricordi che $P = P_{n-1} \cdots P_1$. Si ponga $P^T = (e_{s_1}, ..., e_{s_n})$ dove $(s_1, ..., s_n)$ è una permutazione degli indici $(1, ..., n)$. Se $s_i > i + p$ allora si ha che il minore principale di ordine i della matrice PA è singolare, poiché $(PA)_{i,j} = a_{s_i,j} \quad j = 1, .., s_i - p - 1$ con $s_i - p - 1 \geq i$. Ció comporta l'assurdo che A e U sono singolari. Quindi $s_i \leq i + p$ e di conseguenza PA ha ampiezza di banda $p+q$. Dal teorema 2.8 si ha allora che U ha ampiezza di banda superiore $p+q$. ■

Il pivoting distrugge la struttura a banda della matrice A, nel senso che l'ampiezza di banda superiore di U diventa maggiore di quella di A, mentre nulla si può dire sulla banda di L (Fig. 2.24). Se si vuole eseguire una fattorizzazione in place non si può quindi memorizzare A in un array di dimensione $(p + q + 1) \times n$, perché non c'è spazio sufficiente per conservare tutta la banda di U. Dato che L ha al piú p elementi non nulli in ciascuna colonna, oltre a quello diagonale, lo spazio utilizzato per memorizzare la banda inferiore di A è adatto a contenere tali elementi; bisogna però "ricordare" quali righe sono state scambiate durante la fattorizzazione, in modo da poter ricostruire la posizione effettiva degli elementi di L. Per eseguire la fattorizzazione LU in place, si può dunque utilizzare un array di dimensione $(2p + q + 1) \times n$, in cui la matrice A è memorizzata secondo lo schema a banda, ma supponendo che l'ampiezza di banda superiore sia $p + q$, in modo da riservare spazio per gli elementi della banda di U.

♣ **Esempio 2.49.**

Si considerino nuovamente la matrice A dell'esempio 2.48, di dimensione $n \times n = 5 \times 5$ e con ampiezze di banda $p = 2$ e $q = 1$:

$$A = \begin{pmatrix} 1 & 1 & & & \\ 2 & -1 & 1 & & \\ 3 & 1 & 3 & 1 & \\ & 1 & 3 & -2 & 1 \\ & & -2 & 1 & 1 \end{pmatrix},$$

e le matrici L, U e P ottenute mediante la fattorizzazione LU di A con pivoting:

$$L = \begin{pmatrix} 1 & & & & \\ 2/3 & 1 & & & \\ & -3/5 & 1 & & \\ 1/3 & -2/5 & -7/12 & 1 & \\ & & -5/6 & 1/2 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 3 & 1 & 3 & 1 & \\ & -5/3 & -1 & -2/3 & \\ & & 12/5 & -12/5 & 1 \\ & & & -2 & 7/12 \\ & & & & 37/24 \end{pmatrix},$$

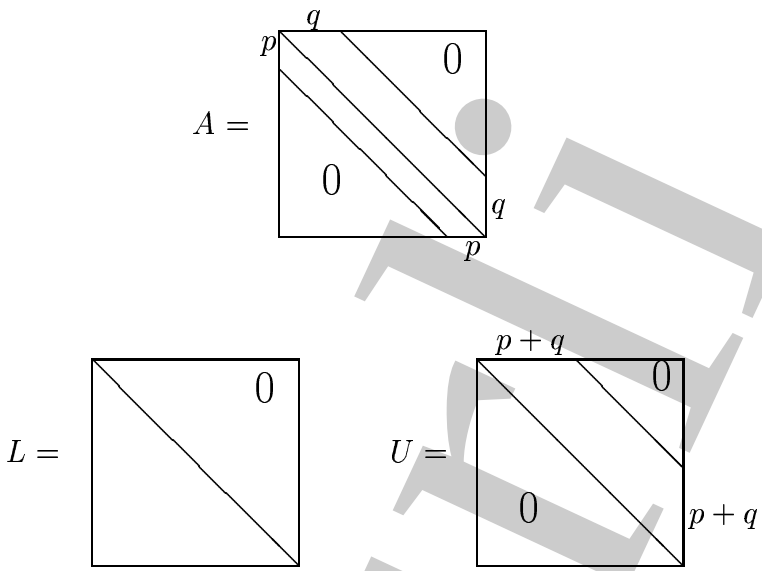

Figura 2.24: Fattorizzazione LU con pivoting parziale di una matrice a banda.

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Se si considera un array AB, di dimensione $(2p+q+1) \times n = 6 \times 5$, e si memorizza A in AB nel modo seguente:[22]

$$AB = \begin{pmatrix} * & * & * & + & + \\ * & * & + & + & + \\ * & 1 & 1 & 1 & 1 \\ 1 & -1 & 3 & -2 & 1 \\ 2 & 1 & 3 & 1 & * \\ 3 & 1 & -2 & * & * \end{pmatrix},$$

si può utilizzare tale array per memorizzare anche L ed U:

$$AB = \begin{pmatrix} * & * & * & 1 & 0 \\ * & * & 3 & -2/3 & 1 \\ * & 1 & -1 & -12/5 & 7/12 \\ 3 & -5/3 & 12/5 & -2 & 37/24 \\ 2/3 & -3/5 & -7/12 & 1/2 & * \\ 1/3 & -2/5 & -5/6 & * & * \end{pmatrix}$$

(i simboli $*$ e $+$ indicano rispettivamente le componenti di AB che non sono proprio usate e quelle che sono usate da U, ma non da A).

[22] In questo caso lo schema di memorizzazione a banda non è conveniente, in quanto il numero di componenti dell'array AB risulta maggiore del numero di elementi della matrice A; ciò è dovuto al fatto che p e q sono *"prossimi"* a $n/2$. Lo schema a banda, infatti, fornisce un effettivo vantaggio in termini di complessità computazionale, rispetto all'usuale schema di memorizzazione di una matrice, se $p, q << n/2$.

La matrice P indica la permutazione che bisogna applicare alle righe di A per ottenere una matrice uguale al prodotto LU, e quindi indica la posizione di ciascuna riga al termine dell'algoritmo di fattorizzazione. Come già visto nel caso di matrici generiche, tale matrice non viene effettivamente costruita; si costruisce piuttosto un vettore di puntatori che contiene in maniera piú compatta le stesse informazioni di P. ♣

Tenendo conto delle considerazioni precedenti si ottiene l'algoritmo descritto nella Procedura 2.20, che esegue in place la fattorizzazione LU con pivoting di una matrice a banda, memorizzando A ed i fattori L ed U secondo lo schema a banda, in un array di dimensione $(2p + q + 1) \times n$.

```
procedure bandlupiv (in: n, p, q, AB; out: AB)

/# SCOPO: effettua la fattorizzazione LU
           con pivoting di una matrice a banda A.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n                  : intero  { prima dimensione della }
                                  { matrice da fattorizzare }
var: AB(p+q+1, n)       : reale   { matrice da fattorizzare }
var: p                  : intero  { ampiezza della banda }
                                  { inferiore di A }
var: q                  : intero  { ampiezza della banda }
                                  { superiore di A }
var: b(n)               : reale   { vettore dei termini noti }
/# PARAMETRI DI OUTPUT:
var: AB(p+q+1, n)       : reale   { matrice fattorizzata }
/# VARIABILI LOCALI:
var: i, j, k, s         : interi
var: t                  : reale
/# INIZIO ISTRUZIONI:
s := p + q
for k = 1 to n - 1
   ip := min {r : |AB(s + r, k)| = max_{1≤i≤p+1} |AB(s + i, k)|}
   ipiv(k) := ip + k - 1
   if (ip ≠ 1)
      for j = k to min(k + q + ip - 1, n)
         t := AB(s + 1 + k - j, j)
         AB(s + 1 + k - j, j) := AB(s + ip + k - j, j)
         AB(s + ip + k - j, j) := t
```

Procedura 2.20: Fattorizzazione LU con pivoting parziale di una matrice a banda - continua

```
      endfor
    endif
    for i = k + 1 to min(k + p, n)
      AB(s + 1 + i − k, k) := AB(s + 1 + i − k, k)/AB(s + 1, k)
    endfor
    for j = k + 1 to min(k + s, n)
      for i = k + 1 to min(k + p, n)
    AB(s + 1 + i − j, j) := AB(s + 1 + i − j, j)+
            −AB(s + 1 + i − k, k) ∗ AB(s + 1 + k − j, j)
      endfor
    endfor
  endfor
  end bandlupiv
```

Procedura 2.20: Fattorizzazione LU con pivoting parziale di una matrice a banda - fine

Si osservi che $ipiv(k) = l$ indica che al passo k la riga k-ma è scambiata con la riga l-ma; l'array $ipiv$ è dunque usato diversamente rispetto al caso generale, in cui $ipiv(k) = l$ indica che al termine dell'algoritmo di fattorizzazione la riga k-ma si trova nella l-ma posizione. Per comprendere meglio la differenza tra i due modi di usare $ipiv$, consideriamo la fattorizzazione LU nell'esempio 2.48. Se si usa $ipiv$ per ricordare lo scambio di righe eseguito al generico passo k, si ha:

$$ipiv = (3, 2, 4, 4);$$

mentre, in generale $ipiv$ è:

$$ipiv = (3, 2, 4, 1, 5).$$

Nel primo caso, ad esempio, $ipiv(4) = 4$ indica che al quarto passo dell'algoritmo la quarta riga non è stata scambiata con alcuna riga, mentre, nel secondo caso $ipiv(4) = 1$ indica che la prima riga di A è diventata la quarta al termine della fattorizzazione. Inoltre, dato che l'algoritmo di fattorizzazione consta di 4 passi, il primo array $ipiv$ ha una componente in meno.

Si osservi inoltre che, al passo k, lo scambio di righe è eseguito solo sulla sottomatrice attiva, lasciando quindi inalterata la posizione degli elementi di L appartenenti alle colonne precedenti la k-ma, in accordo col fatto che $ipiv(k)$ contiene una informazione relativa al solo passo k. Tale scelta è coerente anche con la memorizzazione degli elementi subdiagonali di L, che sono posti nelle ultime p righe dell'array AB, senza tener conto della riga alla quale essi appartengono.

Si noti, infine, che nella Procedura 2.20 si suppone che siano state inizialmente azzerate le componenti delle prime p righe di AB riservate alle p diagonali superiori U generate dalla fattorizzazione; se così non fosse, in seguito agli scambi di righe, si potrebbero avere istruzioni di assegnazione il cui secondo membro contiene variabili non definite.

La Procedura 2.20 si applica, ovviamente, anche ad una matrice A tridiagonale. Se però si assume che A sia memorizzata mediante tre array monodimensionali, si può considerare un ulteriore array monodimensionale per gli elementi della seconda diagonale superiore di U e si possono usare i tre array di A per gli elementi extradiagonali di L e per la diagonale principale e la prima diagonale superiore di U. In questo caso, l'algoritmo va modificato sostituendo opportunamente ad AB i quattro vettori suddetti.

Con un procedimento analogo a quello della fattorizzazione LU senza pivoting, si verifica che la Procedura 2.20 ha la complessità di spazio:

$$T_{LUband+piv}(n,p,q) = O(np(p+q+1)) \quad flop\,;$$

dato che la fattorizzazione è eseguita in place, lo spazio di memoria è essenzialmente quello richiesto dall'array AB, e quindi la complessità di spazio è:

$$S_{LUband+piv}(n,p,q) = O((2p+q+1)n)\,.$$

Si consideri ora la risoluzione dei sistemi lineari $Ly = Pb$ ed $Ux = y$, derivanti dalla fattorizzazione LU con pivoting della matrice a banda A, ottenuta con la Procedura 2.20.

L'algoritmo di forward substitution per la risoluzione di $Ly = Pb$ deve tener conto della effettiva posizione degli elementi di L e deve associare ad ogni riga di L il termine noto giusto, utilizzando le informazioni contenute in $ipiv$.

♣ Esempio 2.50.

Si vuole risolvere il sistema $Ly = Pb$, dove L e P sono le matrici derivanti dalla fattorizzazione LU della matrice A dell'esempio 2.48 e b è il seguente vettore:

$$b = (1, 2, -1, -2, 1)^T.$$

Se si suppone che L sia stata calcolata eseguendo la Procedura 2.20, si ha che gli elementi di L che non appartengono alla diagonale principale sono memorizzati nelle ultime due righe di un array AB, di dimensione 6×5, nel modo seguente:

$$AB = \begin{pmatrix} * & * & * & + & + \\ * & * & + & + & + \\ * & + & + & + & + \\ + & + & + & + & + \\ 2/3 & -2/5 & -7/12 & +1/2 & * \\ 1/3 & -3/5 & -5/6 & * & * \end{pmatrix},$$

dove il simbolo + indica un elemento del fattore U, e la matrice P è rappresentata in maniera compatta dal vettore

$$ipiv = (3, 2, 4, 1).$$

la soluzione del sistema si può calcolare con i passi seguenti:

passo 1:

- scambio delle componenti di b e determinazione di y_1: $ipiv(1) = 3$ indica che al passo 1 dell'algoritmo di fattorizzazione LU la prima riga di a è stata scambiata con la terza; bisogna quindi eseguire lo stesso scambio su b:

$$b = y^{(0)} = \begin{pmatrix} 1 \\ 2 \\ -1 \\ -2 \\ 1 \end{pmatrix} \longrightarrow \overline{y}^{(0)} = \begin{pmatrix} -1 \\ 2 \\ 1 \\ -2 \\ 1 \end{pmatrix}$$

  e risulta

$$y_1 = \overline{y}_1^{(0)} = -1\,;$$

- aggiornamento del vettore soluzione, utilizzando il valore calcolato di y_1:

$$y^{(1)} = \begin{pmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \\ y_4^{(1)} \\ y_5^{(1)} \end{pmatrix},$$

  con

$$\begin{aligned} y_1^{(1)} &= \overline{y}_1^0 = -1\,, \\ y_2^{(1)} &= \overline{y}_2^{(0)} - AB(5,1)y_1^{(1)} = 2 + \frac{2}{3} = \frac{8}{3}\,, \\ y_3^{(1)} &= \overline{y}_3^{(0)} - AB(6,1)y_1^{(1)} = 1 + \frac{1}{3} = \frac{4}{3}\,, \\ y_4^{(1)} &= \overline{y}_4^{(0)} = -2\,, \\ y_5^{(1)} &= \overline{y}_5^{(0)} = 1\,; \end{aligned}$$

passo 2:

- scambio delle componenti di $y^{(1)}$ e determinazione di y_2: $ipiv(2) = 2$ e quindi non si esegue alcuno scambio, ovvero

$$\overline{y}^{(1)} = y^{(1)} = \begin{pmatrix} -1 \\ 8/3 \\ 4/3 \\ -2 \\ 1 \end{pmatrix},$$

  e risulta

$$y_2 = \overline{y}_2^{(1)} = \frac{8}{3}\,;$$

- aggiornamento del vettore soluzione, utilizzando il valore calcolato di y_2:

$$y^{(2)} = \begin{pmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \\ y_4^{(2)} \\ y_5^{(2)} \end{pmatrix},$$

con

$$\begin{aligned} y_1^{(2)} &= y_1 = -1\,, \\ y_2^{(2)} &= y_2 = \frac{8}{3}\,, \\ y_3^{(2)} &= \overline{y}_3^{(1)} - AB(5,2)y_2^{(2)} = \frac{4}{3} + \frac{2}{5}\cdot\frac{8}{3} = \frac{12}{5}\,, \\ y_4^{(2)} &= \overline{y}_4^{(1)} - AB(6,2)y_2^{(2)} = -2 + \frac{3}{5}\cdot\frac{8}{3} = -\frac{2}{5}\,, \\ y_5^{(2)} &= \overline{y}_5^{(1)} = 1\,; \end{aligned}$$

passo 3:

- scambio delle componenti di $y^{(2)}$ e determinazione di y_3: $ipiv(3) = 4$ e quindi si scambiano la terza e la quarta riga di $y^{(2)}$, ovvero

$$y^{(2)} = \begin{pmatrix} -1 \\ 8/3 \\ 12/5 \\ -2/5 \\ 1 \end{pmatrix} \longrightarrow \overline{y}^{(2)} = \begin{pmatrix} -1 \\ 8/3 \\ -2/5 \\ 12/5 \\ 1 \end{pmatrix}$$

e risulta

$$y_3 = \overline{y}_3^{(2)} = -\frac{2}{5}\,;$$

- aggiornamento del vettore soluzione, utilizzando il valore calcolato di y_3:

$$y^{(3)} = \begin{pmatrix} y_1^{(3)} \\ y_2^{(3)} \\ y_3^{(3)} \\ y_4^{(3)} \\ y_5^{(3)} \end{pmatrix}\,,$$

con

$$\begin{aligned} y_1^{(3)} &= y_1 = -1\,, \\ y_2^{(3)} &= y_2 = \frac{8}{3}\,, \\ y_3^{(3)} &= y_3 = -\frac{2}{5}\,, \\ y_4^{(3)} &= \overline{y}_4^{(2)} - AB(5,3)y_3^{(3)} = \frac{12}{5} - \frac{7}{12}\cdot\frac{2}{5} = \frac{13}{6}\,, \\ y_5^{(3)} &= \overline{y}_5^{(2)} - AB(6,3)y_3^{(3)} = 1 + \frac{5}{6}\cdot\left(-\frac{2}{5}\right) = -\frac{2}{3}\,; \end{aligned}$$

passo 4:

- scambio delle componenti di $y^{(3)}$ e determinazione di y_4: $ipiv(4) = 4$ e quindi non si esegue alcuno scambio, ovvero

$$\overline{y}^{(3)} = y^{(3)} = \begin{pmatrix} -1 \\ 8/3 \\ -2/5 \\ 13/6 \\ 2/3 \end{pmatrix}\,,$$

e risulta

$$y_4 = \overline{y}_4^{(3)} = \frac{13}{6};$$

- aggiornamento del vettore soluzione, utilizzando il valore calcolato di y_4:

$$y^{(4)} = \begin{pmatrix} y_1^{(4)} \\ y_2^{(4)} \\ y_3^{(4)} \\ y_4^{(4)} \\ y_5^{(4)} \end{pmatrix},$$

con

$$\begin{aligned} y_1^{(4)} &= y_1 = -1, \\ y_2^{(4)} &= y_2 = \frac{8}{3}, \\ y_3^{(4)} &= y_3 = -\frac{2}{5}, \\ y_4^{(4)} &= y_4 = \frac{13}{6}, \\ y_5^{(4)} &= \overline{y}_5^{(3)} - AB(5,4)y_4^{(4)} = \frac{2}{3} - \frac{1}{2} \cdot \frac{13}{6} = -\frac{5}{12}. \end{aligned}$$

Si osservi che risulta

$$y_5 = y_5^{(4)} = -\frac{5}{12}.$$

♣

In generale, la soluzione del sistema $Ly = Pb$ si può calcolare con una back substitution per colonne, in cui al generico passo j si "aggiornano" le componenti del vettore soluzione a partire dalla j-ma e si scambia la componente di posto j del vettore aggiornato con quella di posto $ipiv(j)$, ottenendo la j-ma componente del vettore soluzione[23], come mostrato nella Procedura 2.21.

[23] Gli elementi diagonali della matrice L sono uguali ad 1.

procedure forsub3d (**in:** $n, b, AB, p, q, ipiv$; **out:** b)

/# SCOPO : risolve un sistema con matrice a banda A,
derivante da fattorizzazione LU con pivotin
parziale, utilizzando la forward substitution.

/# SPECIFICHE DEI PARAMETRI:

/# PARAMETRI DI INPUT:

var:	n	: **intero**	{ *dimensione del sistema* }
var:	$AB(p+q+1, n)$	: **reale**	{ *matrice da fattorizzare* }
var:	p	: **intero**	{ *ampiezza della banda* } { *inferiore di A* }
var:	q	: **intero**	{ *ampiezza della banda* } { *superiore di A* }
var:	$b(n)$	: **reale**	{ *vettore dei termini noti* }
var:	$ipiv(n)$	: **reale**	{ *vettore di permutazione* }

/# PARAMETRI DI OUTPUT:

var:	$b(n)$	: **reale**	{ *vettore risultante* }

/# VARIABILI LOCALI:

var:	i, j, s	: **interi**
var:	t	: **reale**

/# INIZIO ISTRUZIONI:

$s := p + q$
for $j = 1$ **to** $n - 1$

Procedura 2.21: Forward substitution per un sistema con matrice triangolare inferiore derivante dalla fattorizzazione LU con pivoting parziale di una matrice a banda - continua

```
    if (j ≠ ipiv(j)) then
      t := b(j)
      b(j) := b(ipiv(j))
      b(ipiv(j)) := t
    endif
    for i = j + 1 to min(j + p, n)        { aggiornamento componenti }
      b(i) := b(i) − AB(q + 1 + i − j, j) ∗ b(j)
    endfor
  endfor
  end forsub3d
```

Procedura 2.21: Forward substitution per un sistema con matrice triangolare inferiore derivante dalla fattorizzazione LU con pivoting parziale di una matrice a banda - fine

Tale algoritmo ha la stessa complessità di tempo della Procedura 2.19 che risolve il sistema $Ly = b$ derivante dalla fattorizzazione senza pivoting:

$$T_{FORWband+piv}(n,p) = O(np) \quad flop\,.$$

Per quanto riguarda il sistema $Ux = y$, dato che gli scambi sulla matrice U sono stati effettivamente eseguiti durante la fattorizzazione, la soluzione si può calcolare con la Procedura 2.20, considerando $q+p$ al posto di p. La complessità di tempo in questo caso è

$$T_{BACKband+piv}(n,p+q) = O(n(p+q)) \quad flop\,.$$

La complessità di spazio dei due algoritmi è

$$S_{FORWband+piv}(n,p) = S_{BACKband+piv}(n,p+q) = O(n(p+q))\,,$$

in quanto entrambi considerano l'intero array AB, anche se operano solo su parte di esso.

2.12 Fattorizzazione di matrici simmetriche

2.12.1 Fattorizzazione LDL^T

Le matrici simmetriche, come quelle a banda, sono dotate di una struttura, ed è naturale chiedersi se si possa trarre qualche vantaggio da essa nell'esecuzione della fattorizzazione LU.

♣ **Esempio 2.51.**

Si calcoli la fattorizzazione LU della matrice simmetrica

$$A = \begin{pmatrix} 2 & 1 & 3 & 1 \\ 1 & 1 & 2 & 1 \\ 3 & 2 & -2 & 3 \\ 1 & 1 & 3 & 4 \end{pmatrix}.$$

Applicando l'algoritmo di eliminazione di Gauss senza pivoting, si ha:

passo 1:

- calcolo dei moltiplicatori relativi alla prima colonna: $m_{2,1} = 1/2$, $m_{3,1} = 3/2$, $m_{4,1} = 1/2$;
- trasformazione della sottomatrice attiva:

$$A^{(1)} = \begin{pmatrix} 2 & 1 & 3 & 1 \\ & 1/2 & 1/2 & 1/2 \\ & 1/2 & -13/2 & 3/2 \\ & 1/2 & 3/2 & 7/2 \end{pmatrix}$$

passo 2:

- calcolo dei moltiplicatori relativi alla seconda colonna: $m_{3,2} = 1$, $m_{4,2} = 1$;
- trasformazione della sottomatrice attiva:

$$A^{(2)} = \begin{pmatrix} 2 & 1 & 3 & 1 \\ & 1/2 & 1/2 & 1/2 \\ & & -7 & 1 \\ & & 1 & 3 \end{pmatrix};$$

passo 3:

- calcolo del moltiplicatore relativo alla terza colonna: $m_{4,3} = -1/7$;
- trasformazione della sottomatrice attiva:

$$A^{(3)} = U \begin{pmatrix} 2 & 1 & 3 & 1 \\ & 1/2 & 1/2 & 1/2 \\ & & -7 & 1 \\ & & & 22/7 \end{pmatrix}.$$

Si noti che, ad ogni passo, la sottomatrice trasformata, sulla quale si deve operare al passo successivo, è simmetrica; è quindi possibile calcolarne solo gli elementi diagonali e quelli che si trovano al di sopra, o al di sotto, della diagonale principale. Ciò consente di dimezzare approssimativamente la complessità di tempo dell'algoritmo di eliminazione di Gauss.

L'algoritmo precedente calcola i fattori L ed U:

$$L = \begin{pmatrix} 1 & & & \\ 1/2 & 1 & 1 & \\ 3/2 & 1 & 1 & \\ 1/2 & 1 & -1/7 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & 3 & 1 \\ & 1/2 & 1/2 & 1/2 \\ & & -7 & 1 \\ & & & 22/7 \end{pmatrix}.$$

Posto

$$D = \mathrm{diag}(2, 1/2, -7, 22/7),$$

si ha:

$$D^{-1}U = \begin{pmatrix} 1 & 1/2 & 3/2 & 1/2 \\ & 1 & 1 & 1 \\ & & 1 & -1/7 \\ & & & 1 \end{pmatrix} = L^T;$$

la matrice A risulta quindi fattorizzata nel modo seguente:

$$A = LDL^T,$$

dove L è la matrice unitaria triangolare inferiore ottenuta con la fattorizzazione LU senza pivoting e D è una matrice diagonale, i cui elementi sulla diagonale principale coincidono con i corrispondenti elementi della matrice U, risultante dalla fattorizzazione suddetta.

♣

Applicando ad una matrice simmetrica l'algoritmo di eliminazione di Gauss senza pivoting, si ottengono sottomatrici attive simmetriche. Inoltre, a partire dalla fattorizzazione LU di una matrice simmetrica A, si è pervenuti ad un'altra fattorizzazione, $A = LDL^T$, che tiene conto della proprietà di simmetria. I risultati ottenuti valgono in generale per le matrici simmetriche che ammettono la fattorizzazione LU.

Teorema 2.10. *Sia $A \in \Re^{n\times n}$ una matrice simmetrica. Allora, se A ammette la fattorizzazione LU, si ha:*

$$a_{i,j}^{(k)} = a_{j,i}^{(k)}, \quad k = 0, \ldots, n-1, \quad i, j = k+1, \ldots, n,$$

dove $a_{i,j}^{(k)}$ è il generico elemento della matrice $A^{(k)}$ ottenuta al k-mo passo dell'algoritmo di eliminazione di Gauss, e $A^{(0)} = A$.

Dimostrazione
La dimostrazione è basata sul principio di induzione.
Per $k = 0$ la tesi è vera in quanto $A^{(0)} = A$. Si supponga ora che la tesi sia vera per $k = s-1$. Sfruttando tale ipotesi, per $i, j = s+1, \ldots, n$ si ha:

$$a_{i,j}^{(s)} = a_{i,j}^{(s-1)} - \frac{a_{i,s}^{(s-1)}}{a_{s,s}^{(s-1)}} a_{s,j}^{(s-1)} = a_{j,i}^{(s-1)} - \frac{a_{s,j}^{(s-1)}}{a_{s,s}^{(s-1)}} a_{i,s}^{(s-1)} = a_{j,i}^{(s-1)} - \frac{a_{j,s}^{(s-1)}}{a_{s,s}^{(s-1)}} a_{s,i}^{(s-1)} = a_{j,i}^{(s)},$$

e quindi la tesi è vera anche per $k = s$. La tesi è dunque vera per $k = 0, \ldots, n-1$. ■

Teorema 2.11. [**Fattorizzazione** LDL^T]
Sia $A \in \Re^{n\times n}$ una matrice simmetrica tale che tutte le sue sottomatrici principali sono non singolari. Esistono allora una ed una sola matrice $L \in \Re^{n\times n}$ triangolare unitaria inferiore ed una ed una sola matrice $D \in \Re^{n\times n}$ diagonale tali che

$$A = LDL^T.$$

Inoltre, L coincide con la matrice triangolare inferiore della fattorizzazione LU di A e gli elementi diagonali di D coincidono con i corrispondenti elementi diagonali della matrice triangolare superiore della fattorizzazione LU di A, ovvero $L^T = D^{-1}U$.

Dimostrazione

Per ipotesi, A ammette una ed una sola fattorizzazione LU, con $L \in \Re^{n\times n}$ triangolare unitaria inferiore ed $U \in \Re^{n\times n}$ triangolare superiore non singolare. Posto

$$D = \mathrm{diag}(u_{1,1}, \ldots, u_{n,n}),$$

dove $u_{i,i}$ è l'i-mo elemento diagonale di U, e

$$M^T = D^{-1}U,$$

si ha:

$$A = LDM^T, \tag{2.53}$$

con M triangolare unitaria inferiore.

Si vuole ora dimostrare che risulta $L = M$. Moltiplicando ambo i membri di (2.53) a sinistra per M^{-1} ed a destra per $\left(M^T\right)^{-1}$, si ha:

$$M^{-1}A\left(M^T\right)^{-1} = M^{-1}LD.$$

Dato che $M^{-1}A\left(M^T\right)^{-1}$ è una matrice simmetrica e $M^{-1}LD$ è una matrice triangolare inferiore, dalla relazione precedente discende che $M^{-1}L$ è una matrice diagonale; inoltre, $M^{-1}L$ è una matrice unitaria, in quanto prodotto di matrici triangolari inferiori unitarie, e quindi $M^{-1}L = I$, con $I \in \Re^{n\times n}$ matrice identica, ovvero

$$M = L.$$

L'unicità delle matici L e D discende dall'unicità della fattorizzazione LU. Si supponga infatti che esistano $L_1, L_2 \in \Re^{n\times n}$ triangolari unitarie inferiori e $D_1, D_2 \in \Re^{n\times n}$ diagonali tali che

$$A = L_1D_1L_1^T = L_2D_2L_2^T.$$

Posto $U_1 = D_1L_1^T$ e $U_2 = D_2L_2^T$, si ha:

$$A = L_1U_1 = L_2U_2,$$

con U_1 e U_2 triangolari superiori, e quindi, per l'unicità della fattorizzazione LU risulta

$$L_1 = L_2, \quad U_1 = U_2$$

e quindi

$$L_1D_1 = L_2D_2.$$

Poiché $L_1 = L_2$, risulta anche $D_1 = D_2$. ■

I risultati precedenti suggeriscono che il calcolo della fattorizzazione LDL^T si può eseguire modificando opportunamente l'algoritmo di eliminazione di Gauss, utilizzato per ottenere la fattorizzazione LU. Inoltre, grazie alla simmetria della matrice A e delle sottomatrici attive, ad ogni passo, dopo avere calcolato i moltiplicatori, si può operare solo sul triangolo superiore della sottomatrice attiva corrente. L'algoritmo si può eseguire *in place*, memorizzando gli elementi del triangolo superiore di A al posto dei corrispondenti elementi di L^T e gli elementi diagonali di D al posto di quelli di A; gli elementi diagonali di L, essendo tutti uguali ad 1, non sono memorizzati. Una descrizione di tale algoritmo è fornita di seguito.

```
procedure ldlt (in: n, a; out: a)
 /# SCOPO: effettua la fattorizzazione LDL^T
 /# SPECIFICHE DEI PARAMETRI:
 /# PARAMETRI DI INPUT:
 var: n       : intero   { dimensione del sistema }
 var: a(n,n)  : reale    { matrice del sistema }
                         { da fattorizzare }
 /# PARAMETRI DI OUTPUT:
 var: a(n,n)  : reale    { matrice del sistema }
                         { fattorizzata }
 /# VARIABILI LOCALI:
 var: i,j,k   : interi
 var: m       : reale
 /# INIZIO ISTRUZIONI:
for k = 1 to n - 1
   for i = k + 1 to n
      m := a(k,i)/a(k,k)
      for j = i to n
         a(i,j) := a(i,j) - m * a(k,j)
      endfor
      a(k,i) := m
   endfor
endfor
end ldlt
```

Procedura 2.22: Fattorizzazione LDL^T di una matrice tridiagonale - fine

L'algoritmo precedente consta di $n-1$ passi, individuati dal contatore k. Al k-passo si eseguono:

- $n-k$ divisioni floating-point per calcolare i moltiplicatori, m, ovvero gli elementi subdiagonali della k-ma colonna di L;

- $\sum_{i=k+1}^{n}(n-i+1) = \sum_{i=1}^{n-k} i = (n-k)(n-k+1)/2$ addizioni ed altrettante moltiplicazioni floating-point per trasformare il triangolo superiore della sottomatrice attiva.

L'algoritmo richiede in totale

$$\sum_{k=1}^{n-1}\left(n-k+2\frac{(n-k)(n-k+1)}{2}\right) = \sum_{k=1}^{n-1}k + \sum_{k=1}^{n-1}k(k+1) =$$

$$= 2\frac{n(n-1)}{2} + \frac{n(n-1)(2n-1)}{6} = n^2 - n + \frac{2n^3-3n^2+n}{6} =$$

$$= \frac{2n^3+3n^2-5n}{6}$$

operazioni aritmetiche floating-point. La complessità di tempo è dunque circa la metà di quella dell'algoritmo utilizzato per costruire i fattori L ed U di una matrice quadrata:

$$T_{LDL^T}(n) = O\left(\frac{n^3}{6}\right) \; flop\,.$$

La complessità di spazio dell'algoritmo è quella dell'algoritmo di eliminazione di Gauss per matrici generiche, cioè $O(n^2)$, a meno che non si utilizzi uno schema di memorizzazione *packed* (cfr. §2.10); in quest'ultimo caso la complessità di spazio è

$$S_{LDL^T}(n) = O\left(\frac{n^2}{2}\right).$$

Si osservi che la Procedura 2.22 costruisce esplicitamente il triangolo superiore della matrice L^T e lo memorizza sul triangolo superiore della matrice di partenza. Una variante di tale algoritmo, che costruisce direttamente la matrice L, memorizzandola sul triangolo inferiore di A, si può ricavare imponendo l'uguaglianza tra gli elementi della matrice LDL^T ed i corrispondenti elementi di A, con un ragionamento analogo a quello utilizzato, ad esempio, per l'algoritmo di fattorizzazione LU di una matrice tridiagonale.

La risoluzione di un sistema lineare $Ax = b$, con matrice dei coefficienti simmetrica fattorizzata come $A = LDL^T$, si ottiene risolvendo due sistemi triangolari ed uno diagonale:

$$Ly = b, \quad Dw = y, \quad L^T x = w\,;$$

si noti che la risoluzione del sistema $Ly = b$ può essere eseguita senza trasporre effettivamente la matrice L^T costruita con l'algoritmo precedente.

L'algoritmo di eliminazione di Gauss è stato utilizzato per costruire la fattorizzazione LDL^T senza applicare alcuna strategia di pivoting e risulta, in generale, instabile. Dunque, anche se una matrice simmetrica A è dotata di fattorizzazione LDL^T, l'esecuzione della Procedura 2.22 in un sistema aritmetico a precisione finita, può condurre a risultati inaccettabili. Si vuole dunque studiare come si comporta l'algoritmo di eliminazione di Gauss con pivoting parziale quando viene eseguita su una matrice simmetrica.

♣ **Esempio 2.52.**

Si applichi l'algoritmo di eliminazione di Gauss con pivoting parziale alla matrice:

$$A = \begin{pmatrix} 1/2 & 1 \\ 1 & 2 \end{pmatrix}.$$

L'esecuzione del pivoting parziale al primo passo dell'algoritmo genera la matrice

$$A' = \begin{pmatrix} 1 & 2 \\ 1/2 & 1 \end{pmatrix},$$

che non è piú simmetrica. ♣

In generale, applicando ad una matrice simmetrica l'algoritmo di eliminazione di Gauss con pivoting parziale, si distrugge la simmetria della matrice, perdendo in tal modo i vantaggi, in termini di complessità computazionale, derivanti proprio dalla simmetria. Esistono però alcune classi di matrici che non richiedono l'esecuzione del pivoting.

2.12.2 Alcune matrici che non richiedono l'esecuzione del pivoting

♣ **Esempio 2.53.**

Si applichi l'algoritmo di eliminazione di Gauss con pivoting parziale alla matrice:

$$A = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{pmatrix}.$$

Si ha:

passo 1:

- pivoting in prima colonna: $\max_{1 \le i \le 3} |a_{i,1}| = 4 = a_{1,1}$ e quindi non si eseguono scambi;
- calcolo dei moltiplicatori: $m_{2,1} = 1/4$, $m_{3,1} = 1/4$;
- trasformazione della sottomatrice attiva:

$$A^{(1)} = \begin{pmatrix} 4 & 1 & 1 \\ 0 & 11/4 & 3/4 \\ 0 & 3/4 & 15/4 \end{pmatrix}.$$

passo 2:

- pivoting in seconda colonna: $\max_{2 \le i \le 3} |a^{(1)}_{i,2}| = 11/4 = a^{(1)}_{2,2}$ e quindi non si eseguono scambi;
- calcolo del moltiplicatore: $m_{3,2} = 3/11$;

- trasformazione della sottomatrice attiva:

$$A^{(1)} = \begin{pmatrix} 4 & 1 & 1 \\ 0 & 11/4 & 3/4 \\ 0 & 0 & 39/11 \end{pmatrix}.$$

Si noti che, in entrambi i passi, l'elemento di massimo modulo della colonna da annullare nella sottomatrice attiva si trova sulla diagonale e quindi non bisogna eseguire scambi di righe. ♣

La matrice dell'esempio precedente non richiede l'applicazione del pivoting nell'esecuzione dell'algoritmo di eliminazione di Gauss. Ciò è dovuto al fatto che tale matrice, oltre ad essere simmetrica, è a diagonale strettamente dominante.

Definizione 2.13. (Matrice a diagonale dominante)
Una matrice $A \in \Re^{n\times n}$ si dice a diagonale dominante se

$$|a_{i,i}| \geq \sum_{\substack{i,\,j=1 \\ j\neq i}}^{n} |a_{i,j}|, \quad i = 1, 2, \ldots, n. \tag{2.54}$$

In particolare, se in (2.54) vale il segno $>$, la matrice si dice a diagonale strettamente dominante.

Per le matrici simmetriche a diagonale strettamente dominante sussiste il

Teorema 2.12. *Sia $A \in R^{n\times n}$ una matrice a diagonale strettamente dominante. Allora la matrice A è non singolare e, applicando l'algoritmo di eliminazione di Gauss senza pivoting, si ottiene, ad ogni passo, una sottomatrice attiva a diagonale strettamente dominante.*

Dimostrazione La sottomatrice attiva al k-mo passo dell'algoritmo di eliminazione di Gauss, $\bar{A}^{(k-1)}$, è formata dalle ultime $n-k-1$ righe e $n-k-1$ colonne della matrice trasformata, $A^{(k-1)}$, ottenuta al passo $k-1$:

$$\bar{A}^{(k-1)} = \begin{pmatrix} a_{k,k}^{(k-1)} & \ldots & a_{k,n}^{(k-1)} \\ \vdots & \vdots & \vdots \\ a_{n,k}^{(k-1)} & \ldots & a_{n,n}^{(k-1)} \end{pmatrix}.$$

Per dimostrare che $\bar{A}^{(k-1)}$ è a diagonale dominante per ogni $k = 1, \ldots, n$, si può applicare il principio di induzione.

Per $k = 1$ risulta $\bar{A}^{(k-1)} = A$ e quindi la tesi è vera. Si osservi che, essendo A a diagonale dominante, si ha:

$$|a_{1,1}| > \sum_{\substack{i=1 \\ i\neq j}}^{n} |a_{i,j}| \geq 0\,.$$

e quindi $a_{1,1} \neq 0$.

Si supponga ora che $\bar{A}^{(k-2)}$, $k \geq 2$, sia a diagonale strettamente dominante. Per ogni $i = k, \ldots, n$ si ha:

$$\begin{aligned} \sum_{\substack{j=k \\ j \neq i}}^{n} \left| a_{i,j}^{(k-1)} \right| &= \sum_{\substack{j=k \\ j \neq i}}^{n} \left| a_{i,j}^{(k-2)} - \frac{a_{i,k-1}^{(k-2)}}{a_{k-1,k-1}^{(k-2)}} a_{k-1,j}^{(k-2)} \right| \leq \\ &\leq \sum_{\substack{j=k \\ j \neq i}}^{n} \left| a_{i,j}^{(k-2)} \right| + \frac{\left| a_{i,k-1}^{(k-2)} \right|}{\left| a_{k-1,k-1}^{(k-2)} \right|} \sum_{\substack{j=k \\ j \neq i}}^{n} \left| a_{k-1,j}^{(k-2)} \right| , \end{aligned} \tag{2.55}$$

dove $a_{k-1,k-1}^{(k-2)} \neq 0$ in quanto $\bar{A}^{(k-2)}$ è a diagonale strettamente dominante. In virtú di questa proprietà risulta inoltre, per $i = k, \ldots, n$,

$$\begin{aligned} \sum_{\substack{j=k \\ j \neq i}}^{n} \left| a_{i,j}^{(k-2)} \right| + \left| a_{i,k-1}^{(k-2)} \right| &< \left| a_{i,i}^{(k-2)} \right| , \\ \sum_{\substack{j=k \\ j \neq i}}^{n} \left| a_{k-1,j}^{(k-2)} \right| + \left| a_{k-1,i}^{(k-2)} \right| &< \left| a_{k-1,k-1}^{(k-2)} \right| , \end{aligned}$$

e quindi, da (2.55), si ottiene:

$$\begin{aligned} \sum_{\substack{j=k \\ j \neq i}}^{n} \left| a_{i,j}^{(k-1)} \right| &< \left(\left| a_{i,i}^{(k-2)} \right| - \left| a_{i,k-1}^{(k-2)} \right| \right) + \frac{\left| a_{i,k-1}^{(k-2)} \right|}{\left| a_{k-1,k-1}^{(k-2)} \right|} \left(\left| a_{k-1,k-1}^{(k-2)} \right| - \left| a_{k-1,i}^{(k-2)} \right| \right) = \\ &= \left| a_{i,i}^{(k-2)} \right| - \frac{\left| a_{i,k-1}^{(k-2)} \right|}{\left| a_{k-1,k-1}^{(k-2)} \right|} \left| a_{k-1,i}^{(k-2)} \right| \leq \left| a_{i,i}^{(k-2)} - \frac{a_{i,k-1}^{(k-2)}}{a_{k-1,k-1}^{(k-2)}} a_{k-1,i}^{(k-2)} \right| \leq \left| a_{i,i}^{(k-1)} \right| . \end{aligned}$$

La matrice $\bar{A}^{(k-1)}$ è dunque a diagonale strettamente dominante ed è cosí dimostrato che tutte le sottomatrici attive sono a diagonale strettamente dominante.

Per dimostrare che A è non singolare, basta osservare che, al generico passo k dell'algoritmo di eliminazione di Gauss, risulta

$$a_{k,k}^{(k-1)} \neq 0 .$$

■

Una conseguenza dei teoremi 2.12 e 2.10 è che, se una matrice è simmetrica a diagonale dominante, tutte le sottomatrici attive hanno come elemento di massimo modulo della colonna pivot l'elemento diagonale di tale colonna, e quindi la strategia di pivoting è inutile. La fattorizzazione LDL^T si può dunque calcolare con la Procedura 2.22.

2.12.3 Fattorizzazione di Cholesky

Definizione 2.14. (Matrice definita positiva)
Una matrice $A \in \Re^{n\times n}$, simmetrica, si dice definita positiva se, per ogni $x \in \Re^n$, $x \neq 0$, risulta

$$x^T Ax > 0\,.$$

Si dimostra che, se una matrice è simmetrica definita positiva, l'errore di roundoff nell'esecuzione dell'algoritmo di eliminazione di Gauss senza pivoting, in un sistema aritmetico floating-pont a precisione finita, si mantiene limitato in maniera paragonabile a quanto avviene con l'applicazione del pivoting [8]. Non ci si sofferma però su tale argomento, in quanto alle matrici simmetriche definite positive è possibile applicare un algoritmo di fattorizzazione *ad hoc*, come spiegato nel prossimo paragrafo.

Si è visto, nei paragrafi precedenti, che una matrice A simmetrica, in opportune ipotesi, ha una fattorizzazione LDL^T. Si vuole ora vedere se è possibile ottenere una fattorizzazione del tipo $A = LL^T$, dove L è una matrice triangolare inferiore (con elementi diagonali in generale diversi da 1).[24]

♣ **Esempio 2.54.** Si provi a fattorizzare la matrice simmetrica

$$A = \begin{pmatrix} 16 & 4 & 20 \\ 4 & 10 & 2 \\ 20 & 2 & 30 \end{pmatrix},$$

nel prodotto LL^T, dove L è una matrice reale triangolare inferiore:

$$L = \begin{pmatrix} l_{1,1} & & \\ l_{2,1} & l_{2,2} & \\ l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix}.$$

Moltiplicando L per la sua trasposta, si ha:

$$LL^T = \begin{pmatrix} l_{1,1}^2 & l_{1,1}l_{2,1} & l_{1,1}l_{3,1} \\ l_{1,1}l_{2,1} & l_{2,1}^2 + l_{2,2}^2 & l_{3,1}l_{2,1} + l_{2,2}l_{3,2} \\ l_{1,1}l_{3,1} & l_{3,1}l_{2,1} + l_{2,2}l_{3,2} & l_{3,1}^2 + l_{3,2}^2 + l_{3,3}^2 \end{pmatrix},$$

che è una matrice simmetrica. Uguagliando gli elementi del triangolo inferiore (superiore) di A con i corrispondenti elementi di L, si ricava

$$\begin{aligned}
l_{1,1} &= \sqrt{a_{1,1}} = \sqrt{16} = 4; \\
l_{2,1} &= a_{2,1}/l_{1,1} = 4/4 = 1; \\
l_{3,1} &= a_{3,1}/l_{1,1} = 20/4 = 5; \\
l_{2,2} &= \sqrt{a_{2,2} - l_{2,1}^2} = \sqrt{10-1} = 3; \\
l_{3,2} &= (a_{3,2} - l_{3,1}l_{2,1})/l_{2,2} = (2-5)/3 = -1; \\
l_{3,3} &= \sqrt{a_{3,3} - l_{3,1}^2 - l_{3,2}^2} = \sqrt{30-25-1} = 2;
\end{aligned}$$

[24] Anche se la matrice L considerata in $A = LL^T$ differisce dalla matrice L in LDL^T, si continuerà ad indicare entrambe con la lettera L, a meno che ciò non generi confusione, in quanto questa è la notazione solitamente utilizzata.

ovvero

$$L = \begin{pmatrix} 4 & & \\ 1 & 3 & \\ 5 & -1 & 2 \end{pmatrix}.$$

L'esistenza di L dipende dal fatto che gli argomenti delle radici quadrate sono sempre non negativi; ciò garantisce non solo che gli elementi diagonali $l_{i,i}$ esistano in $\Re$, ma anche che non vengano effettuate divisioni per 0 nel calcolo degli altri elementi di L.
La matrice A è definita positiva. Ciò si può verificare applicando il criterio di Sylvester (Teorema A.22, Appendice A.9), ovvero controllando che il determinante di ciascuna sottomatrice principale di A sia maggiore di 0. Si ha infatti:

$$\begin{aligned} det(A_1) &= det\,(16) = 16, \\ det(A_2) &= det \begin{pmatrix} 16 & 4 \\ 4 & 10 \end{pmatrix} = 144, \\ det(A_3) &= det \begin{pmatrix} 16 & 4 & 20 \\ 4 & 10 & 2 \\ 20 & 2 & 30 \end{pmatrix} = 576 \end{aligned}$$

e quindi la matrice è definita positiva. ♣

Teorema 2.13. [Fattorizzazione di Cholesky]
Sia $A \in \Re^{n\times n}$ simmetrica definita positiva. Allora esiste una ed una sola matrice triangolare inferiore $L \in \Re^{n\times n}$, con $l_{k,k} > 0$ per $k = 1, \ldots, n$, tale che

$$A = LL^T.$$

Dimostrazione Se $A \in \Re^{n\times n}$ è definita positiva, per il criterio di Sylvester tutte le sottomatrici principali di A sono non singolari e quindi A ammette una ed una sola fattorizzazione del tipo

$$A = GDG^T,$$

con $G \in \Re^{n\times n}$ triangolare unitaria inferiore e $D = diag\,(d_1, \ldots, d_n) \in \Re^{n\times n}$ non singolare. Dato che $det(A_k) = d_1 d_2 \ldots d_k$, per il criterio di Sylvester risulta anche

$$d_k > 0\,, \quad k = 1, 2, \ldots, n\,.$$

La matrice

$$D^{\frac{1}{2}} = diag\left(\sqrt{d_1}, \ldots, \sqrt{d_n},\right)$$

è quindi reale e non singolare e, posto

$$L = GD^{\frac{1}{2}}\,,$$

risulta

$$A = LL^T\,,$$

con L triangolare inferiore tale che $l_{k,k} > 0$ per $k = 1, \ldots, n$. L'unicità della fattorizzazione LL^T è una conseguenza dell'unicità della fattorizzazione GDG^T. Si supponga, infatti, che esistano $L, \bar{L} \in \Re^{n \times n}$, tali che

$$l_{k,k}, \bar{l}_{k,k} > 0, \quad k = 1, \ldots, n,$$

e

$$A = LL^T = \bar{L}\bar{L}^T.$$

Posto

$$B = diag\,(l_{1,1}, \ldots, l_{n,n}), \quad G = LB^{-1}, \quad D = (B^2)^{-1},$$
$$\bar{B} = diag\,(\bar{l}_{1,1}, \ldots, \bar{l}_{n,n}), \quad \bar{G} = \bar{L}\bar{B}^{-1}, \quad \bar{D} = (\bar{B}^2)^{-1},$$

si ha:

$$A = GDG^T = \bar{G}\bar{D}\bar{G}^T,$$

da cui discende

$$D = (B^2)^{-1} = \bar{D} = (\bar{B}^2)^{-1}, \quad G = LB^{-1} = \bar{G} = \bar{L}\bar{B}^{-1},$$

e quindi

$$L = \bar{L}.$$

■

La dimostrazione precedente è basata sul Teorema 2.11, riguardante l'esistenza e l'unicità della fattorizzazione LDL^T di una matrice. Una dimostrazione indipendente da tale teorema è riportata in nota[25].
La fattorizzazione LL^T di una matrice $A \in \Re^{n \times n}$ simmetrica definita positiva è detta *fattorizzazione di Cholesky* di A e la matrice L è detta *fattore di Cholesky* di A. La dimostrazione del Teorema 2.13 fornisce un procedimento per il calcolo della matrice L

[25] **Dimostrazione 2** Si procede per induzione. Per $n = 1$ basta porre $l_{1,1} = \sqrt{a_{1,1}}$ e la tesi è vera. Si supponga ora che la tesi sia vera per per $n = k - 1 > 1$, cioè tale che, data $A_{k-1} \in \Re^{(k-1)\times(k-1)}$ simmetrica definita positiva, esista $L_{k-1} \in \Re^{(k-1)\times(k-1)}$ triangolare inferiore, con tutti gli elementi diagonali positivi, tale che

$$A_{k-1} = L_{k-1}L_{k-1}^T.$$

Si considerino le seguenti matrici $A_k, L_k \in \Re^{k \times k}$, la prima simmetrica definita positiva e la seconda triangolare inferiore:

$$A_k = \begin{pmatrix} A_{k-1} & y \\ y^T & a_{k,k} \end{pmatrix}, \quad L_k = \begin{pmatrix} L_{k-1} & \underline{0} \\ w^T & l_{k,k} \end{pmatrix},$$

con $y, w \in \Re^{(k-1)}$ vettori colonna e $l_{k,k} > 0$. Imponendo

$$A_k = L_k L_k^T,$$

a partire dalla fattorizzazione GDG^T. Un algoritmo differente per il calcolo di L si ottiene ragionando come nell'esempio 2.54: si considera una matrice $L = (l_{i,j})$ triangolare inferiore, si calcola il prodotto LL^T e si impone l'uguaglianza tra gli elementi di A ed i corrispondenti elementi di LL^T, ovvero

$$\begin{aligned} a_{j,j} &= \sum_{k=1}^{j} l_{j,k} l_{k,j}^T = \sum_{k=1}^{j} l_{j,k}^2, \quad j = 1, \ldots, n, \\ a_{i,j} &= \sum_{k=1}^{j} l_{i,k} l_{k,j}^T = \sum_{k=1}^{j} l_{i,k} l_{j,k} \quad j = 1, \ldots, n-1, \quad i = j+1, \ldots, n. \end{aligned}$$

Da ciò si ricava:

$$\begin{aligned} l_{j,j} &= \sqrt{a_{j,j} - \sum_{k=1}^{j-1} l_{j,k}^2}, \quad j = 1, \ldots, n, \\ l_{i,j} &= \frac{a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} l_{j,k}}{l_{j,j}}, \quad j = 1, \ldots, n-1, \quad i = j+1, \ldots, n. \end{aligned} \tag{2.58}$$

Gli elementi di L si possono quindi determinare in n passi, calcolando, al passo j, prima l'elemento diagonale della colonna j-ma e poi gli altri elementi di tale colonna che

---

si ha:

$$\begin{aligned} &A_{k-1} = L_{k-1} L_{k-1}^T, \\ &y = L_{k-1} w, \\ &y^T = w^T L_{k-1}^T, \\ &a_{k,k} = w^T w + l_{kk}^2. \end{aligned} \tag{2.56}$$

La prima relazione in (2.56) è vera per ipotesi di induzione; dalla seconda (ovvero dalla terza) si ricava univocamente w, in quanto L_{k-1}, avendo tutti gli elementi diagonali positivi, è non singolare; dalla quarta relazione, infine, si ricava:

$$l_{k,k} = \sqrt{a_{k,k} - w^T w}. \tag{2.57}$$

Per concludere la dimostrazione dell'esistenza di L_k occorre quindi dimostrare che l'argomento della radice quadrata in (2.57) è positivo. Considerati i seguenti vettori:

$$z = A_{k-1}^{-1} y\,, \quad x = \begin{pmatrix} z \\ 1 \end{pmatrix},$$

si ha:

$$\begin{aligned} x^T A_k x &= z^T A_{k-1} z - 2 z^T y + a_{k,k} = z^T y - 2 z^T y + a_{k,k} = \\ &= -z^T y + a_{k,k} = a_{k,k} - y^T A_{k-1}^{-1} y = a_{k,k} - y^T (L_{k-1} L_{k-1}^T)^{-1} y = \\ &= a_{k,k} - (L_{k-1}^{-1} y)^T (L_{k-1}^{-1} y) = a_{k,k} - w^T w\,; \end{aligned}$$

dato che $x^T A_k x > 0$ (A_k è definita positiva), risulta $a_{k,k} - w^T w > 0$, e quindi l'esistenza di L_k è dimostrata. L'unicità di L_k discende dall'unicità di L_{k-1}, di w e di $l_{k,k}$. Il teorema è cosí dimostrato.

si trovano al di sotto di quello diagonale. Si noti che i calcoli possono essere eseguiti in place, come mostrato nella Proceura 2.23[26]. Per costruire $l_{i,j}$ si utilizzano, oltre ad $a_{i,j}$, gli elementi della i-ma e della j-ma riga di L che appartengono alle prime $j-1$ colonne, come schematizzato in Fig. 2.25.

[26]Le istruzioni `for` $i = j+1$ `to` n e `for` $k = 1$ `to` $j-1$ si possono scambiare tra loro, lasciando inalterato il valore finale di $a_{i,j}$. In questo caso l'algoritmo, anziché calcolare definitivamente, uno dopo l'altro, gli $n-j$ elementi extradiagonali della colonna j-ma, esegue $j-1$ aggiornamenti parziali di tali elementi, ottenendo la colonna j-ma di L al termine del $(j-1)$-mo aggiornamento. Le due versioni dell'algoritmo differiscono per il modo in cui si accede agli elementi dellla parte di L già calcolata, per ottenere una nuova colonna.

```
procedure cholesky (in: n, a; out: a)

/# SCOPO: effettua la fatttorizzazione
          di Cholesky di una matrice A.

/# SPECIFICHE DEI PARAMETRI:
/# PARAMETRI DI INPUT:
var: n       : intero   { dimensione della matrice A }
var: a(n,n)  : reale    { matrice da fattorizzare }
/# PARAMETRI DI OUTPUT:
var: a(n,n)  : reale    { matrice fattorizzata }
/# VARIABILI LOCALI:
var: i, j, k : interi

/# INIZIO ISTRUZIONI:
for j = 1 to n                { Calcolo degli elementi diagonali}
   for k = 1 to j - 1
      a_{j,j} := a_{j,j} - a_{j,k}^2
   endfor
   a_{j,j} := sqrt(a_{j,j})
   for i = j + 1 to n     { Calcolo degli elementi del triangolo}
                                  { inferiore non diagonali}
      for k = 1 to j - 1
         a_{i,j} := a_{i,j} - a_{i,k} a_{j,k}
      endfor
      a_{i,j} := a_{i,j} / a_{j,j}
   endfor
endfor
end cholesky
```

Procedura 2.23: Algoritmo di Cholesky

Calcoliamo la complessità di tempo della Procedura 2.23, al generico passo j,

- il calcolo dell'elemento diagonale $l_{j,j}$ richiede $j-1$ moltiplicazioni, altrettante addizioni ed una radice quadrata, ovvero

$$2(j-1)\ \ flop\ \ +\ \ 1\ \ radice\ \ quadrata\,;$$

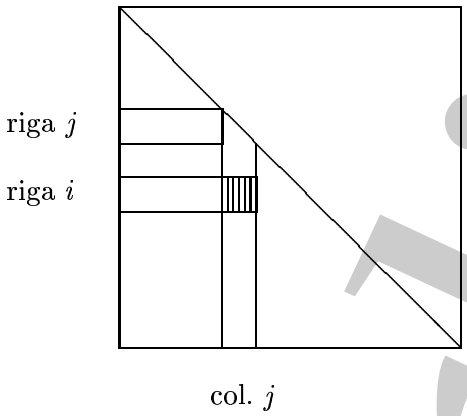

Figura 2.25: Rappresentazione grafica dell'algoritmo di Cholesky.

- il calcolo di un elemento extradiagonale $l_{i,j}$, $i > j$, richiede $j-1$ moltiplicazioni, altrettante addizioni ed una divisione, e quindi il calcolo degli $n-j$ elementi extradiagonali della colonna j-ma richiede

$$(n-j)(2j-1) \ \ flop\,.$$

In totale, l'algoritmo richiede

$$\begin{aligned}
&\sum_{j=1}^{n} (2(j-1) + (n-j)(2j-1)) = \\
= \ & 2\sum_{j=1}^{n-1} j + 2n\sum_{j=1}^{n-1} j - 2\sum_{j=1}^{n-1} j^2 - n^2 + \sum_{j=1}^{n-1} j + n \\
= \ & (2n+3)\frac{n(n-1)}{2} - 2\frac{n(n-1)(2n-1)}{6} - n^2 - n(n-1) \\
= \ & \frac{2n^3 + 3n^2 - 5n}{6}
\end{aligned}$$

e

$$n \ \ radici \ \ quadrate.$$

La complessità di tempo dell'algoritmo è dunque

$$T_{Chol}(n) = O\left(\frac{n^3}{6}\right) \quad flop.$$

Si noti che, come era prevedibile, il numero di operazioni aritmetiche floating-point dell'algoritmo di Cholesky è circa la metà di quello dell'algoritmo di eliminazione di Gauss.

Se per memorizzare la matrice A, e quindi la matrice L, si utilizza un array bidimensionale di dimensione $n \times n$, la complessità di spazio dell'algoritmo di Cholesky è:

$$S_{Chol}(n) = n^2;$$

se invece si utilizza uno schema di memorizzazione packed, la complessità di spazio diventa

$$S_{Chol}^{packed}(n) = \frac{n(n+1)}{2} = O\left(\frac{n^2}{2}\right)$$

e quindi risulta dimezzata. L'algoritmo relativo a quest'ultimo si può ottenere dalla Procedura 2.23, sostituendo ad $a_{i,j}$ l'elemento corrispondente dell'array LP utilizzato per la memorizzazione packed.

Analogamente all'algoritmo di eliminazione di Gauss, l'algoritmo di Cholesky, richiede l'esecuzione di istruzioni del tipo[27]

$$\begin{aligned} a_{i,j} &:= a_{i,j} - a_{i,k}a_{j,k}\,, \\ a_{i,j} &:= a_{i,j}/a_{j,j}\,; \end{aligned} \tag{2.59}$$

è quindi naturale chiedersi se sia necessaria qualche strategia di pivoting per evitare che, al generico passo j, l'elemento $a_{j,j}$ sia "troppo grande" e quindi gli $a_{i,j}$, che risultano essere gli $a_{i,k}$ dei passi successivi, crescano in modo da amplificare significativamente l'errore.

♣ Esempio 2.55.

Applichiamo l'algoritmo di Cholesky alla matrice simmetrica definita positiva

$$A = \begin{pmatrix} .0001 & .01 \\ .01 & 100 \end{pmatrix},$$

utilizzando un sistema aritmetico floating-point con base $\beta = 10$, precisione $t = 2$ e massima accuratezza dinamica. In tale sistema, A ha la seguente rappresentazione:

$$\tilde{A} = A = \begin{pmatrix} .10 \times 10^{-3} & .10 \times 10^{-1} \\ .10 \times 10^{-1} & .10 \times 10^{3} \end{pmatrix}$$

si ha inoltre:

$$\begin{aligned} \tilde{l}_{1,1} &= \sqrt{a_{1,1}} = .10 \times 10^{-1}, \\ \tilde{l}_{2,1} &= a_{2,1}/\tilde{l}_{1,1} = (.10 \times 10^{-1})/(.10 \times 10^{-1}) = .10 \times 10^{1}, \\ \tilde{l}_{2,2} &= \sqrt{a_{2,2} - \tilde{l}_{2,1}^2} = \sqrt{.10 \times 10^{3} - (.10 \times 10)^2} = \\ &= \sqrt{.10 \times 10^{3} - .1 \times 10} = \sqrt{.10 \times 10^{3}} = .10 \times 10^{2}, \end{aligned}$$

[27] Si noti che, per $i = j$, la prima delle (2.59) corrisponde alla seguente istruzione per il calcolo dell'elemento diagonale j-mo:

$$a_{j,j} := a_{j,j} - a_{j,k}^2.$$

e quindi

$$\tilde{L} = \begin{pmatrix} .10 \times 10^{-1} & \\ .10 \times 10^{1} & .10 \times 10^{2} \end{pmatrix}.$$

Eseguendo il prodotto $\tilde{L}\tilde{L}^T$ nel sistema aritmetico considerato, si ha:

$$\tilde{L}\tilde{L}^T = \begin{pmatrix} .10 \times 10^{-3} & .1 \times 10^{-1} \\ .1 \times 10^{-1} & .1 \times 10^{3} \end{pmatrix} = A.$$

Anche se $a_{1,1}$ è due ordini di grandezza più piccolo di $a_{2,1}$, non si ha una crescita *"eccessiva"* degli elementi di $\tilde{L}$, come invece ci si potrebbe aspettare, e quindi una crescita dell'errore di roundoff tale da invalidare il calcolo. ♣

Dalla prima relazione in (2.58) si ricava che

$$l_{j,1}^2 + l_{j,2}^2 + \ldots + l_{j,j}^2 = a_{j,j}, \quad j = 1, \ldots, n\,,$$

dove tutti i termini a primo membro sono non negativi, da cui si ottiene la limitazione a priori per gli elementi di $\tilde{L}$:

$$|\tilde{l}_{j,k}| \leq \sqrt{a_{j,j}}\,, \quad k = 1, \ldots, j\,, j = 1, \ldots, n\,.$$

Ogni elemento di $\tilde{L}$ è dunque limitato dalla radice quadrata dell'elemento diagonale di A che si trova sulla medesima riga, indipendentemente dalla grandezza degli elementi *"pivot"* che compaiono al denominatore nei calcoli e dei conseguenti *"moltiplicatori"*. In altri termini, gli elementi della matrice L non possono crescere in maniera tale da invalidare il calcolo. L'algoritmo di Cholesky risulta quindi stabile.

Teorema 2.14.
Dato un sistema di equazioni $Ax = b$ con $A \in R^{n\times n}$ simmetrica definita positiva. Se $\widehat{L}$ è la matrice triangolare inferiore calcolata dall'algoritmo di Cholesky e $\widehat{x}$ la soluzione calcolata del sistema, allora si ha:

$$(A + \Delta A)\widehat{x} = b \quad con \quad ||\Delta A||_2 \leq c_n u ||A||_2 + o(u^2)$$

con $c_n = o(n^2)$ costante dipendente solo da n.

Dimostrazione Si cominci col dimostrare che, detta u la massima accuratezza relativa e posto $\gamma_i = iu/(1-iu)$, per ogni fissato j si ha:

$$|a_{i,j} - \sum_{k=1}^{i} \widehat{l}_{i,k}\widehat{l}_{j,k}| \leq \gamma_i \sum_{k=1}^{i} |\widehat{l}_{i,k}||\widehat{l}_{j,k}| \quad i = j+1, .., n \tag{2.60}$$

Ció puó essere facilmente provato che detto:

$$\widehat{s} = fl\left(a_{i,j} - \sum_{k=1}^{i} \widehat{l}_{i,k}\widehat{l}_{j,k}\right)$$

e ricordando che $fl(a\ op\ b) = (a\ op\ b)(1+\delta)$ con $|\delta| \leq u$, si ha:

$$\widehat{s} = a_{i,j}(1+\delta_1)\cdots(1+\delta_i) - \sum_{k=1}^{i} \widehat{l}_{i,k}\widehat{l}_{j,k}(1+\epsilon_k)(1+\delta_k)\cdots(1+\delta_i)$$

con $|\epsilon_i|\ |\delta_i| \leq u$. Dividendo per $(1+\delta_1)\cdots(1+\delta_i)$ si ottiene:

$$\frac{\widehat{s}}{(1+\delta_1)\cdots(1+\delta_i)} = a_{i,j} - \sum_{k=1}^{i} \widehat{l}_{i,k}\widehat{l}_{j,k}\frac{(1+\epsilon_k)}{(1+\delta_1)\cdots(1+\delta_{k-1})}$$

Ricordando infine che $\prod_{k=1}^{i}(1+\delta_k)^{\rho_k} = 1+\theta_i$ con $\rho_k = \pm 1$ e $|\theta_i| \leq \gamma_i$ si ha $\widehat{s}(1+\theta_i) = a_{i,j} - \sum_{k=1}^{i} \widehat{l}_{i,k}\widehat{l}_{j,k}(1+\theta_k)$, da cui la (2.60). Analogamente si puó provare che

$$|a_{j,j} - \sum_{k=1}^{j} \widehat{l}_{j,k}^2| \leq \gamma_j \sum_{k=1}^{j} |\widehat{l}_{i,k}^2| \tag{2.61}$$

Dalle (2.60) e (2.61) si ha quindi che, se si suppone $nu < 1$, si ottiene:

$$\widehat{L}\widehat{L}^T = A + \Delta A \quad \text{con} \quad \gamma_{n+1}|\widehat{L}||\widehat{L}^T| \leq nu|\widehat{L}||\widehat{L}^T| + o(u^2) \tag{2.62}$$

Poiché per la risoluzione di sistemi triangolari valgono[28]:

$$\begin{array}{ll} (\widehat{L}+\Delta L)\widehat{y} = b & |\Delta L| \leq nu|\widehat{L}| + o(u^2) \\ (\widehat{U}+\Delta U)\widehat{x} = \widehat{y} & |\Delta U| \leq nu|\widehat{U}| + o(u'2) \end{array}$$

allora si ha:

$$b = (\widehat{L}+\Delta L)(\widehat{L}^T+\Delta L^T)\widehat{x} = (A + \Delta A + \widehat{L}\Delta L^T + \Delta L\widehat{L}^T + \Delta L\Delta L^T)\widehat{x} = (A+\Delta A') \tag{2.63}$$

dove $|\Delta A'| \leq c_n nu|\widehat{L}||\widehat{L}^T| + o(u^2)$.
Si osservi che

$$||\,|L||L^T|\,||_2 = ||\,|L|\,||_2^2 \leq nn||L||_2^2 = n||A||$$

e, analogamente, dalla (2.62) si ha:

$$||\,|\widehat{L}||\widehat{L}^T|\,||_2 \leq n(1-n\gamma_n)^{-1}||A||_2$$

da cui, utilizzando la (2.63) si ottiene

$$||\Delta A'||_2 \leq ||\,|\Delta A|\,||_2 \leq c_n u||A||_2 + o(u^2)$$

cioé la tesi. ■

Il teorema precedente asserisce quindi che risolvere un sistema di equazioni lineari con matrice simmetrica definita positiva, equivale alla risoluzione di un sistema perturbato, dove le perturbazioni sui coefficienti rimangono limitate.

Si osservi, però, che alcune *"difficoltà numeriche"* si possono comunque presentare.

[28] E ricordando che L^T è triangolare superiore

♣ **Esempio 2.56.**

Si consideri la matrice simmetrica definita positiva:

$$A = \begin{pmatrix} 100 & 9.71 \times 10^{-2} \\ 9.71 \times 10^{-2} & 9.43 \times 10^{-5} \end{pmatrix},$$

il cui fattore di Cholesky, arrotondato a quattro cifre decimali significative, è

$$L = \begin{pmatrix} 10 & \\ 9.710 \times 10^{-3} & 9.662 \times 10^{-3} \end{pmatrix}$$

Se si calcola L, mediante l'algoritmo di Cholesky, in un sistema aritmetico floating-point con base $\beta = 10$ e precisione $t = 3$, dotato di massima accuratezza dinamica, si ha:

$$\begin{aligned} \tilde{l}_{1,1} &= \sqrt{a_{1,1}} = .100 \times 10^{2}, \\ \tilde{l}_{2,1} &= a_{2,1}/\tilde{l}_{1,1} = (.971 \times 10^{-1})/(.100 \times 10^{2}) = .971 \times 10^{-2}, \\ \tilde{l}_{2,2} &= \sqrt{a_{2,2} - \tilde{l}_{2,1}^2} = \sqrt{.943 \times 10^{-3} - .943 \times 10^{-4}} = 0\,, \end{aligned}$$

ovvero:

$$\tilde{L} = \begin{pmatrix} 10 & \\ 9.71 \times 10^{-4} & 0 \end{pmatrix},$$

che ha il secondo elemento diagonale uguale a 0. Eseguendo il prodotto $\tilde{L}\tilde{L}^T$, nel sistema aritmetico floating-point considerato, si ha:

$$\tilde{L}\tilde{L}^T = \begin{pmatrix} 100 & 9.71 \times 10^{-2} \\ 9.71 \times 10^{-2} & 0 \end{pmatrix},$$

che non è una matrice definita positiva. Ciò dipende dal fatto che la matrice A è *"lontanamente"* definita positiva; i suoi autovalori, arrotondati a tre cifre significative, sono infatti $\lambda_1 = 100$ e $\lambda_2 = 9.34 \times 10^{-5}$, e quindi uno di essi è prossimo a zero. ♣

In generale, nell'esecuzione dell'algoritmo di Cholesky si posso avere problemi di instabilità numerica nel caso in cui la matrice A non è "sufficientemente" definita positiva, ovvero se uno o piú autovalori sono prossimi a zero.

2.13 Software matematico per la risoluzione di sistemi lineari

Come già osservato nei paragrafi precedenti, uno dei package piú utilizzati per la risoluzione di problemi di algebra lineare con matrice piena o a banda è *LAPACK* [1]. Tale package è il risultato di una intensa attività rivolta allo sviluppo di software matematico

efficiente, accurato e portabile per la risoluzione dei problemi suddetti[29] ed è disponibile, come software di dominio pubblico, al sito web `http://www.netlib.org/lapack`.

LAPACK è scritto in Fortran 77 e contiene routine per la risoluzione di sistemi di equazioni lineari e problemi di autovalori e valori singolari, basate essenzialmente su metodi di fattorizzazione della matrice dei coefficienti del problema. Le routine sono disponibili in singola ed in doppia precisione, per problemi reali e complessi. Esistono inoltre interfacce a LAPACK o conversioni/traduzioni di esso in Fortran 95, C, C++ e Java.

LAPACK contiene, tra l'altro, routine per la risoluzione di sistemi lineari con matrice speciale (tridiagonale, a banda, simmetrica definita positiva, simmetrica definita positiva a banda, simmetrica definita positiva tridiagonale, simmetrica non definita, triangolare a banda). Il package include inoltre routine che eseguono operazioni connesse con la risoluzione di un sistema, quali, ad esempio, la stima dell'indice di condizionamento della matrice dei coefficienti o dell'errore nella soluzione calcolata.

Per le matrici simmetriche, hermitiane o triangolari è usata sia la memorizzazione convenzionale, nel triangolo superiore o inferiore di un array bidimensionale, sia la memorizzazione di tipo *packed*. Per matrici a banda è utilizzato lo schema di memorizzazione a banda, mentre per quelle tridiagonali è utilizzata la memorizzazione mediante tre vettori.

Le routine di LAPACK sono suddivise in tre classi:

- routine *driver*, ciascuna delle quali risolve un problema completo, quale, ad esempio, il calcolo della soluzione di un sistema lineare;
- routine *computazionali*, ciascuna delle quali risolve un singolo problema computazionale, quale, ad esempio, il calcolo della fattorizzazione LU di una matrice o la stima dell'indice di condizionamento;
- routine *ausiliarie*, ciascuna delle quali risolve un singolo sottoproblema che si presenta nei problemi computazionali suddetti.

Le routine driver chiamano, in generale, routines computazionali, le quali chiamano a loro volta routine ausiliarie.

Le routine driver e computazionali hanno un nome a cinque o a sei caratteri[30], rispettivamente del tipo XYYZZ o XYYZZZ, in cui

[29] LAPACK costituisce l'evoluzione e l'estensione, per i calcolatori moderni a memoria gerarchica, dei package *LINPACK* [3] ed *EISPACK* [10], sviluppati negli anni '70 per la risoluzione di sistemi lineari e di problemi lineari di minimi quadrati e per il calcolo di autovalori ed autovettori, utilizzando essenzialmente metodi di fattorizzazione. Gli algoritmi implementati sono stati riorganizzati in termini di operazioni tra matrici, quali, ad esempio, prodotti matrice-matrice, eseguite mediante *Basic Linear Algebra Subprograms (BLAS)* [9, 5, 4], per tener conto della gerarchia di memoria della macchina in uso e quindi minimizzare il traffico di dati da e verso tale memoria, con conseguente aumento dell'efficienza. Si noti che il raggiungimento di elevate prestazioni richiede una implementazione ottimizzata di BLAS, fornita da numerose case costruttrici di computer o generabile in maniera automatica utilizzando il software ATLAS

[30] In LAPACK sono presenti due tipi di routine driver: i *driver semplici*, che calcolano la soluzione di un problema completo, ed i *driver esperti* che, oltre al calcolo della soluzione, sono in grado di

- X indica il tipo di dato su cui la routine agisce (S = REAL, D = DOUBLE PRECISION, C = COMPLEX, Z = COMPLEX*16
  o DOUBLE COMPLEX);

- YY indica il tipo di matrice (GB = generale a banda, GT = generale tridiagonale, PO = simmetrica o Hermitiana definita positiva - memorizzazione convenzionale, PP = simmetrica o Hermitiana definita positiva - memorizzazione packed, SB = simmetrica a banda, etc.);

- ZZ o ZZZ indica il tipo di operazione effettuata (SV = risoluzione di un sistema - driver semplice, TRF = fattorizzazione, TRS = risoluzione del sistema fattorizzato, CON = stima dell'indice di condizionamento, etc.).

Ad esempio, le routine SPOSV e SPPSV risolvono entrambe un sistema lineare reale con matrice dei coefficienti simmetrica definita positiva, operando su dati in singola precisione; SPOSV richiede che la matrice sia memorizzata in un array bidimensionale, mentre SPPSV assume una memorizzazione packed. Analogamente, DGBTRF esegue la fattorizzazione *LU* di una matrice reale a banda e DGBTRS risolve i sistemi lineari triangolari a banda associati a tale fattorizzazione; entrambe le routine eseguono le operazioni aritmetiche in doppia precisione. Maggiori dettagli sono forniti in [1].

Numerose routine per la risoluzione di sistemi lineari con matrice speciale sono presenti nella Libreria Fortran 77 del NAg versione *Mark 19*. In particolare, per la risoluzione di sistemi lineari con matrice generale o speciale, oltre alle routine dei capitoli F01 e F04, la libreria contiene, nel capitolo F07, le routine di LAPACK[31] (si veda, a tal proposito, la pagina web `http://www.nag.co.uk/numeric/fl/manual/html/FLlibrarymanual.asp`).

Routine di LAPACK, infine, sono utilizzate da funzioni di `MATLAB` per risolvere problemi del tipo suddetto. Ad esempio, la funzione *chol*, di cui si parla nel prossimo paragrafo, è basata sulle routine DPOTRF e ZPOTRF, che eseguono la fattorizzazione di Cholesky rispettivamente di una matrice reale in doppia precisione e di una complessa in doppia precisione.

2.14 Risoluzione di sistemi lineari in ambiente `MATLAB`

2.14.1 Elementi di base di `MATLAB`

In ambiente `MATLAB` la memorizzazione di un vettore viene effettuata racchiudendo tra parentesi quadre una sequenza di numeri, separati da uno spazio bianco o da una virgola; ad esempio

eseguire altre operazioni, quali, ad esempio, la stima dell'indice di condizionamento o il raffinamento (miglioramento dell'accuratezza) della soluzione calcolata e la stima dell'errore corrispondente. Il nome di un driver semplice ha cinque caratteri; quello di un driver esperto ne ha sei ed è ottenuto aggiungendo una X alla fine del nome dell'analogo driver semplice.

[31] La libreria Fortran 77 del NAg include anche le routine di LAPACK per i problemi di minimi quadrati e di autovalori, nel capitolo F08.

```
>>  x=[1 2 3]

x =

     1     2     3
```

oppure

```
>>  x=[1, 2, 3];
```

Il punto e virgola alla fine dell'istruzione consente di memorizzare, ma non visualizzare sullo schermo, il risultato di un'operazione.

Analogamente, per definire una matrice si separano gli elementi delle righe con spazi o virgole, mentre le diverse righe con punti e virgola (oppure andando a capo ad ogni nuova riga).

```
>> A=[1 2 3 ; 4 5 6 ; 7 8 9 ]

A =

     1     2     3
     4     5     6
     7     8     9

>> A=[1 2 3
      4 5 6
      7 8 9];
```

L'operatore di trasposizione in MATLAB è il carattere *apice* ', quindi sono analoghe le istruzioni

```
>> y=[1; 4; 7]

y =

     1
     4
     7

>> y=[1 4 7]'

y =
```

```
     1
     4
     7
```

L'accesso agli elementi di una matrice si ottiene tramite indici; l'elemento a_{mn} è indirizzato come $A(m, n)$; ad esempio

```
>> A(2,3)

ans =

     6
```

cioè il comando $A(2, 3)$ fa riferimento all'elemento di posto (2,3) della matrice A. Più in generale, per prelevare l'elemento sulla riga i e sulla colonna j della matrice A e memorizzarlo nella variabile a, si utilizza l'istruzione

```
>> a=A(i,j)
```

La riga m-esima è indirizzata come A(m,:), dove tutte le colonne sono indicate con due punti:

```
>> A(2,:)

ans =

     4     5     6
```

mentre la colonna n-esima è indirizzata come A(:,n), dove tutte le righe sono indicate con due punti:

```
>> A(:,3)

ans =

     3
     6
     9
```

2.14.2 Operazioni matriciali

Le operazioni algebriche tra matrici si eseguono con gli usuali operatori $+, -, *$; in particolare, date due matrici A e B, di dimensioni opportune, si possono definire le seguenti operazioni:

```
>> S=A+B; % somma di due matrici
>> P=A*B; % prodotto righe per colonne
>> At=A'; % trasposta di una matrice
```

Altre funzioni operanti su matrici (e, quindi, su vettori riga o colonna) sono:

```
max, min,
sort,
sum, prod.
```

Esistono, poi, particolari operatori (.∗, . /,. ^) che permettono di effettuare operazioni su matrici, elemento per elemento, senza ricorrere a cicli. Ad esempio, se A e B sono due matrici, per moltiplicarle elemento per elemento basta eseguire l'istruzione

```
>> C=A.*B;
```

Tra le funzioni che operano essenzialmente su matrici si ricordano:

- `det`, che restituisce il determinante di una matrice

```
>> det(A)

ans =

     0
```

- `size`, che restituisce le dimensioni di una matrice

```
>> size(A)

ans =

     3     3
```

- `rank`, che restituisce il rango di una matrice

```
>> rank(A)

ans =

     2
```

Si riporta un elenco di alcune funzioni predefinite, per la creazione di matrici particolarmente usate:

`eye(n)`	: matrice identica di ordine n
`zeros(m,n)`	: matrice di 0, con m righe e n colonne
`ones(m,n)`	: matrice di 1, con m righe e n colonne
`rand(m,n)`	: matrice generata in maniera casuale (*random*), costituita da valori compresi tra 0 e 1
`diag(X)`	: se X è un vettore con n elementi, costruisce una matrice quadrata, diagonale, di dimensione $n \times n$, con gli elementi di X sulla diagonale. Se X è una matrice quadrata di dimensione $n \times n$, produce un vettore di n elementi, uguali a quelli sulla diagonale di X.

`MATLAB` presenta, inoltre, un `help` in linea, con informazioni sulla sintassi di tutte le funzioni disponibili; ad esempio, per avere informazioni relative all'utilizzo di una particolare funzione *fun*, è sufficiente digitare

```
>> help fun
```

Il solo comando

```
>> help
```

elenca tutte le categorie di funzioni disponibili (i **toolbox**), mentre, ad esempio,

```
>> help signal
```

fornisce informazioni su tutte le fiunzioni presenti nella categoria `signal`.

♣ **Esempio 2.57.** Calcolare il prodotto scalare

$$s = u * v^T$$

tra i vettori

$$u = (5, 3, -2, -4, -1) \qquad v = (2, -1, 0, -7, 2)$$

Memorizzando i due vettori mediante i comandi

```
>> u=[5 3 -2 -4 -1]

u =

     5     3    -2    -4    -1
```

```
>> v=[2 -1 0 -7 2]

v =

     2    -1     0    -7     2
```

si determina il risultato, come output generato dal comando

```
>> s=u*v'

s =

    33
```

♣

♣ **Esempio 2.58.** Assegnata la matrice

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

si vogliono calcolare

1. A1=A*A;
2. A2=A'*A;
3. A3=.*A;
4. d1=diag(A);
5. d2=diag(A,1);
6. e=exp(A);
7. sq=sqrt(A);
8. el=exp(log(A+7));
9. m=max(A);
10. sn=sign(A);

A tal fine si memorizza, dapprima, la matrice A come

```
>> A=[5 3 -6; 7 2 0; -4 8 1]

A =

    5     3    -6
    7     2     0
   -4     8     1
```

Si ottengono, dunque, i risultati mediante le istruzioni

```
>> A1=A*A

A1 =

    70   -27   -36
    49    25   -42
    32    12    25

>> A2=A'*A

A2 =

    90    -3   -34
    -3    77   -10
   -34   -10    37

>> A3=A.*A

A3 =

    25     9    36
    49     4     0
    16    64     1

>> d1=diag(A)

d1 =

     5
     2
     1

>> d2=diag(A,1)

d2 =

     3
     0

>> e=exp(A)

e =

   1.0e+03 *

    0.1484    0.0201    0.0000
    1.0966    0.0074    0.0010
    0.0000    2.9810    0.0027

>> sq=sqrt(A)

sq =
```

```
   2.2361             1.7321                  0 + 2.4495i
   2.6458             1.4142                  0
        0 + 2.0000i   2.8284             1.0000
```

```
>> el=exp(log(A+7))

el =

   12.0000   10.0000    1.0000
   14.0000    9.0000    7.0000
    3.0000   15.0000    8.0000

>> m=max(A)

m =

     7     8     1

>> sn=sign(A)

sn =

     1     1    -1
     1     1     0
    -1     1     1
```

Con l'istruzione `save` è possibile salvare, in un file con estensione `.mat`, ad esempio *ristest.mat*, i risultati ottenuti:

```
>> save ristest A1 A2 A3 d1 d2 e sq el m sn
```

Con l'istruzione `clear`

```
>> clear
```

si può liberare la memoria dai risultati di tutte le operazioni eseguite; con il comando `load` è, poi, possibile caricare il file di risultati creato

```
>> load ristest
```

In tal modo, in memoria, saranno nuovamente presenti le variabili salvate nel file *ristest*. ♣

2.14.3 Metodo di eliminazione di Gauss e fattorizzazione LU

Si supponga di voler risolvere in ambiente `MATLAB` un sistema lineare del tipo

$$Ax = B$$

ovvero di voler determinare il vettore x come

$$x = A^{-1}B.$$

MATLAB consente di risolvere il problema eseguendo le istruzioni

```
>> x=A\B;
```

o, analogamente,

```
>> x=inv(A)*B;
```

con

```
>> inv(A); % inversa della matrice A
```

Più in generale la divisione tra matrici in MATLAB può essere eseguita con due diversi simboli:

- $x = A\backslash B$ esegue $x = A^{-1} \cdot B$, risolvendo il sistema $Ax = B$; in tal caso si parla anche di *divisione a sinistra.*
- $x = D/C$ esegue $D \cdot C^{-1}$, risolvendo $x \cdot C = D$; in tal caso si parla anche di *divisione a destra.*

Nel secondo caso si possono utilizzare i due comandi equivalenti:

```
>> x=D/C;
```

oppure

```
>> x=D*inv(C);
```

Riassumendo

- **slash** / esegue la divisione con la matrice posta a destra;
- **backslash** \ esegue la divisione con la matrice posta a sinistra.

MATLAB determina la fattorizzazione LU di una matrice A mediante l'istruzione

```
>> [L,U]=lu(A);
```

che restituisce, in output, le matrici L, triangolare inferiore e U, triangolare superiore. In realtà, nel caso in cui il metodo di eliminazione di Gauss preveda l'applicazione della tecnica di pivoting e, dunque, uno scambio di righe, la matrice L, restituita in output, è la matrice risultante dal prodotto del fattore triangolare inferiore della decomposizione LU, con la matrice di permutazione, P, tale che

$$PA = LU;$$

l'espressione esplicita delle tre matrici, L,U e P, si può, invece, ottenere, mediante l'istruzione

```
>> [L,U,P]=lu(A);
```

♣ **Esempio 2.59.** Risolvere il seguente sistema lineare

$$\begin{cases} 2x_1 & -4x_2 & +7x_3 & +4x_4 & = & 5 \\ 9x_1 & +3x_2 & +2x_3 & -7x_4 & = & -1 \\ 5x_1 & +2x_2 & -3x_3 & +x_4 & = & -3 \\ 6x_1 & -5x_2 & +4x_3 & -3x_4 & = & 2 \end{cases}$$

e determinare i fattori della decomposizione LU della matrice dei coefficienti.

Memorizzando la matrice A mediante l'istruzione

```
>> A=[2 -4 7 4 ; 9 3 2 -7; 5 2 -3 1; 6 -5 4 -3 ]

A =

     2    -4     7     4
     9     3     2    -7
     5     2    -3     1
     6    -5     4    -3
```

ed il vettore dei termini noti

```
>> b=[5 -1 -3 2]'

b =

     5
    -1
    -3
     2
```

la soluzione del sistema si ottiene come output generato dall'istruzione

```
>> x=A\b

x =

   -0.1704
   -0.1137
    0.6614
    0.0640
```

I fattori della decomposizione LU di A si deducono mediante l'istruzione

```
>> [L,U]=lu(A);
```

che genera il seguente output:

```
>> [L,U]=lu(A)

L =

    0.2222    0.6667    1.0000         0
    1.0000         0         0         0
    0.5556   -0.0476   -0.8339    1.0000
    0.6667    1.0000         0         0


U =

    9.0000    3.0000    2.0000   -7.0000
         0   -7.0000    2.6667    1.6667
         0         0    4.7778    4.4444
         0         0         0    8.6744
```

In particolare, l'istruzione

```
>> [L,U,P]=lu(A);
```

restituisce

```
L =

    1.0000         0         0         0
    0.6667    1.0000         0         0
    0.2222    0.6667    1.0000         0
    0.5556   -0.0476   -0.8339    1.0000


U =

    9.0000    3.0000    2.0000   -7.0000
         0   -7.0000    2.6667    1.6667
         0         0    4.7778    4.4444
         0         0         0    8.6744
```

```
P =

     0     1     0     0
     0     0     0     1
     1     0     0     0
     0     0     1     0
```

♣

2.14.4 MATLAB e le matrici speciali

In ambiente MATLAB non sono presenti funzioni predefinite specifiche per la risoluzione di sistemi lineari con matrice a banda o simmetrica non definita. Per i sistemi lineari con matrice simmetrica definita positiva è, invece, disponibile la funzione `chol`, che esegue la fattorizzazione di Cholesky di una matrice A,

$$A = R^T R,$$

dove R è una matrice triangolare superiore, con elementi diagonali positivi, calcolando il *fattore di Cholesky*, R, mediante l'istruzione

```
>> R=chol(A)
```

♣ **Esempio 2.60.**

Si vuole risolvere il sistema linerare $Ax = b$, con

$$A = \begin{pmatrix} 22 & 2 & 12 & 15 \\ 2 & 47 & 18 & -10 \\ 12 & 18 & 25 & 21 \\ 15 & -10 & 21 & 47 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 3 \\ -2 \end{pmatrix}.$$

La matrice A è simmetrica definita positiva. Per verificare che essa gode di quest'ultima proprietà si può applicare il criterio di Sylvester (Teorema A.22, Appendice A.9), utilizzando la funzione *det*.

Le istruzioni

```
>> A = [ 22    2   12   15
        2   47   18  -10
       12   18   25   21
       15  -10   21   47];
>> d2 = det(A(1:2,1:2))
>> d3 = det(A(1:3,1:3))
>> d4 = det(A)
```

generano il seguente output:

```
d2 =

        1030
d3 =

       12718
d4 =

      137641
```

da cui si ricava che la matrice è definita positiva (il determinante della sottomatrice principale di A di ordine 1 è, banalmente, 22). Poiché una matrice simmetrica è definita positiva se e solo se i suoi autovalori sono positivi (cfr. § A.9), la precedente verifica si può eseguire anche utilizzando la funzione `eig`, che calcola gli autovalori (ed anche gli autovettori) di una matrice quadrata. L'istruzione

```
>> lambda = eig(A)
```

fornisce, in output, un vettore, memorizzato nella variabile `lambda`, di componenti gli autovalori di A:

```
lambda =

   14.1028
    2.5464
   56.4276
   67.9232
```

che sono, appunto, reali positivi. Utilizzando la routine `chol` si ottiene:

```
>> R=chol(A)

R =

    4.6904    0.4264    2.5584    3.1980
         0    6.8424    2.4712   -1.6608
         0         0    3.5139    4.8158
         0         0         0    3.2898
```

Naturalmente, eseguendo il prodotto

```
>> C = R'*R
```

si ottiene

```
C =

   22.0000    2.0000   12.0000   15.0000
    2.0000   47.0000   18.0000  -10.0000
   12.0000   18.0000   25.0000   21.0000
   15.0000  -10.0000   21.0000   47.0000
```

che coincide con la matrice A.

La soluzione del sistema $Ax = b$ si può, dunque, eseguire con le istruzioni

```
>> w = R'\b
>> x = R \w
```

che corrispondono ad una forward e ad una back substitution e producono i seguenti risultati:

```
>> w = R'\b

w =

    0.2132
    0.1329
    0.6051
   -1.6339

>> x = R \w

x =

   -0.0439
   -0.4092
    0.8529
   -0.4967
```

Si osservi che il calcolo della soluzione del sistema $Ax = b$ si può eseguire con l'istruzione

```
>> x = A \ b
```

che fornisce, in output, direttamente il vettore x:

```
x =

   -0.0439
   -0.4092
    0.8529
   -0.4967
```

In questo caso, prima di procedere alla risoluzione del sistema, `MATLAB` controlla se la matrice è simmetrica e se ha tutti gli elementi diagonali positivi; in tal caso prova ad eseguire la fattorizzazione di Cholesky. Se la matrice è definita positiva, come in questo caso, la fattorizzazione di Cholesky termina senza errore ed i sistemi triangolari ad essa associati sono risolti con i relativi algoritmi di sostituzione. [32] ♣

[32] Se l'algoritmo di Cholesky fallisce, il sistema è risolto mediante la fattorizzazione LU.

2.15 Operazioni di Calcolo Matriciale

2.15.1 Operazioni tra vettori e matrici

1. Calcolare $\alpha\underline{x}$, $\underline{x}^T\underline{y}$, $\underline{x}\underline{y}^T$ e $\underline{x}+\underline{y}$ per:

   a) $\alpha = 2.5$, $\underline{x} = [6 \;\; 2]^T$, $\underline{y} = [4 \;\; -2]^T$;

   b) $\alpha = -2$, $\underline{x} = \underline{y} = [4 \;\; -2 \;\; 2]^T$.

2. Posto:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}, \quad E = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \end{pmatrix},$$

$$\underline{c} = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \quad \underline{d} = \begin{pmatrix} 2 \\ 3 \end{pmatrix},$$

   calcolare AB e BA. Usa tali risultati per determinare, se definiti:

   a) $2A - 3B$, ABA, A^2B, BA^2, EBA, ABE, BAE;

   b) $\underline{c}^TB$, $E\underline{d}$, $\underline{d}^TA$.

3. Verificare, utilizzando A, B, $\underline{c}$ e $\underline{d}$ dell'esercizio precedente, che:

   a) $(\underline{c}^TA)B = \underline{c}^T(AB)$;

   b) $(A+B)\underline{d} = A\underline{d} + B\underline{d}$;

   c) $A(BA) = (AB)A$.

4. Verificare che per moltiplicare una matrice $r \times s$ per una matrice $s \times t$ sono richieste al piú rstM e $rt(s-1)$A.

5. Scrivere un sottoprogramma FORTRAN $dot(X, Y, N, C)$ che calcoli $c = \underline{x}^T \cdot \underline{y}$, con $\underline{x}, \underline{y} \in R^n$, $c \in R$.

6. Scrivere un sottoprogramma FORTRAN $saxpy(S, X, Y, N, Z)$ che calcoli $\underline{z} = s\underline{x} + \underline{y}$, con $\underline{x}, \underline{yz} \in R^n$, $s \in R$.

7. Scrivere un sottoprogramma FORTRAN $matvet(A, X, M, N, Y)$ che calcoli $\underline{y} = A\underline{x}$, con $\underline{x}, \underline{y} \in R^n$, $A \in R^{m \times n}$.

8. Scrivere un sottoprogramma FORTRAN $add(S, A, B, M, N, C)$ che calcoli $C = A + sB$, con $A \in R^{m \times n}$, $B \in R^{m \times n}$ e $s \in R$.

9. Scrivere un sottoprogramma FORTRAN $mult(A, B, M, N, P, C)$ che calcoli $C = AB$, con $A \in R^{m \times n}$, $B \in R^{n \times p}$.

2.15.2 Il calcolo dell'inversa di una matrice

1. Per le seguenti matrici:

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix},$$

calcolare AB, A^{-1}, B^{-1} e $(AB)^{-1}$.

2. Risolvere, mediante il calcolo dell'inversa della matrice dei coefficienti, i seguenti sistemi:

$$\begin{cases} x_1 + 2x_2 = -3 \\ 4x_1 + 3x_2 = 18 \end{cases} \quad \begin{cases} -2x_1 + x_2 = -1 \\ 3x_1 - x_2 = 3 \end{cases} \quad \begin{cases} 2x_1 - x_2 = 0 \\ 2x_1 + x_2 = 1 \end{cases}$$

3. Determinare quante operazioni aritmetiche sono necessarie per risolvere $A\underline{x} = \underline{b}$ calcolando $A^{-1}\underline{b}$, con $A \in R^{3\times 3}$.

2.15.3 Norme di vettori e matrici

1. Per

$$\underline{x} = \begin{pmatrix} 2 \\ -5 \\ 3 \end{pmatrix}, \quad A = \begin{pmatrix} -1 & 1 & -4 \\ 2 & 2 & 0 \\ 3 & 3 & 2 \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} -2 & 7 & -4 \\ 2 & -5 & 4 \\ 0 & -3 & 2 \end{pmatrix},$$

calcolare $\|\underline{x}\|$, $\|A\|$ e $\|A^{-1}\|$, con $\|\cdot\| = \|\cdot\|_1,\ \|\cdot\|_2,\ \|\cdot\|_\infty$.

2. Per ciascuno dei seguenti vettori $\underline{x}$, calcolare $\|\underline{x}\|_1,\ \|\underline{x}\|_2,\ \|\underline{x}\|_\infty$:

$$[2 \quad -5]^T;\ [2 \quad -5 \quad 5]^T,\ [0 \quad 0 \quad 0]^T.$$

3. Per ciascuna delle seguenti coppie di vettori, calcolare $\underline{x}^T\underline{y}$, $\|\underline{x}\|_2$, $\|\underline{y}\|_2$ e verificare che $|\underline{x}^T\underline{y}| \leq \|\underline{x}\|_2\|\underline{y}\|_2$ (*diseguaglianza di Schwarz*):

   a) $\underline{x} = [1 \quad -3]^T,\ \underline{y} = [2 \quad 1]^T$;

   b) $\underline{x} = [-3 \quad -4]^T,\ \underline{y} = [1 \quad -1]^T$;

   c) $\underline{x} = [2 \quad -3]^T,\ \underline{y} = [-6 \quad 9]^T$.

4. Verificare che $\|\underline{x}\|_\infty \leq \|\underline{x}\|_2 \leq \|\underline{x}\|_1$, $\forall \underline{x} \in R^n$, e che l'uguaglianza puó presentarsi anche per vettori non nulli.

2.15.4 Proprietà di matrici

1. Per la seguente matrice:

$$A = \begin{pmatrix} -3 & 9 & 6 \\ 9 & 3 & -3 \\ -6 & 6 & -3 \end{pmatrix},$$

   a) valutare i minori $M_{1,1}$, $M_{1,3}$, $M_{2,2}$, $M_{3,2}$, $M_{3,3}$;

   b) valutare i minori complementari $A_{1,1}$, $A_{2,2}$, $A_{3,2}$, $A_{3,3}$;

   c) valutare $detA$.

2. Calcolare i determinanti delle seguenti matrici:

$$a) \begin{pmatrix} 2 & -2 & 4 \\ 4 & 2 & 6 \\ 0 & 4 & -2 \end{pmatrix}, \quad b) \begin{pmatrix} 2 & -2 \\ 0 & 4 \end{pmatrix}, \quad c) \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad d) \begin{pmatrix} -7 \end{pmatrix}.$$

3. Verificare che $detL$ con L triangolare inferiore è uguale al prodotto degli elementi diagonali.

4. Verificare che $detI_p = 1$, $\forall p$.

5. Verificare che per P matrice di permutazione si ha che $detP = \pm 1$.

6. Verificare che $det(A^n) = (detA)^n$, $\forall n \in N$.

2.16 Esercizi di Algebra Lineare

1. La Pennsyltucky Pharmaceutical prepara una linea specialistica di tre integratori dietetici usando proporzioni diverse di tre sostanze naturali: estratto di broccoli, gingko biloba e olio di ginseng. Il prodotto 1 è realizzato combinando quantità uguali di tutte e tre le sostanze; il prodotto 2 è costituito prevalentemente da gingko biloba, a quattro dosi di esso corrisponde una dose di estratto di broccoli e olio di ginseng; il prodotto 3 impiega sei dosi di olio di ginseng, due di gingko biloba e una di estratto di broccoli. Queste combinazioni possono essere modellizzate da un insieme di equazioni lineari: se indichiamo con x_1 la quantità di estratto di broccoli, con x_2 la quantità di gingko biloba e con x_3 la quantità di olio di ginseng, e denotiamo con p_1, p_2 e p_3 la quantità di ciascun prodotto, allora otteniamo il sistema di equazioni

$$\begin{aligned} x_1 + x_2 + x_3 &= p_1 \\ x_1 + 4x_2 + x_3 &= p_2 \\ x_1 + 2x_2 + 6x_3 &= p_3. \end{aligned}$$

Data la quantità di produzione desiderata per ciascun prodotto p_i, $i = 1, 2, 3$, per ciascuna sostanza possiamo determinare la quantità da comprare risolvendo questo sistema di equazioni. Questo particolare modello è molto semplice, soprattutto per la dimensione piccola del sistema, e non richiede alcuna tecnica specializzata per la sua risoluzione. Tuttavia, il successo di questa linea di integratori ha convinto la Pennsyltucky Pharmaceutical ad espandere sostanzialmente questo settore dei suoi affari, che sta pensando di produrre 50 prodotti fatti da altrettante sostanze naturali. La risoluzione di un sistema lineare 50×50 non è da prendere alla leggera in assenza di un buon software [6].

2.16.1 Esercizi §2.4

1. Risolvere i seguenti sistemi triangolari utilizzando gli algoritmi di back e forward substitution:

$$a) \begin{cases} x & & & = & 6 \\ 4x & + & 6y & = & 0 \\ 8x & + & 10y & + & 12z & = & 8 \end{cases} \qquad b) \begin{cases} 18x & + & 15y & + & 12z & = & 12 \\ & + & 9y & + & 6z & = & 0 \\ & & & & 9z & = & 9 \end{cases}$$

$$c) \begin{cases} 2x & & & & & = & -2 \\ 14x & + & 2y & & & = & -10 \\ 12x & + & 10y & + & 2z & = & 8 \\ 8x & + & 6y & + & 4z & + & 8u & = & -10 \end{cases}$$

$$d) \begin{cases} 6x & + & 3y & + & 3z & + & 3u & = & -3 \\ & & 6y & + & 3z & - & 6u & = & 6 \\ & & & & 3z & + & 3u & = & -3 \\ & & & & & & 6u & = & -6 \end{cases}$$

$$e) \begin{cases} 2x & + & 4y & + & 2z & + & 2u & = & 12 \\ & & 2y & + & 10z & - & 6u & = & -20 \\ & & & & 2z & - & 4u & = & -10 \\ & & & & & & 2u & = & 4 \end{cases}$$

2. Dimostrare che:

   a) se U è una matrice triangolare superiore invertibile, allora U^{-1} è triangolare superiore;

   b) se L è una matrice triangolare inferiore con elementi diagonali tutti uguali ad 1 (*unitaria*), allora L^{-1} è triangolare inferiore unitaria;

c) il prodotto di due matrici triangolari superiori (inferiori) è una matrice triangolare superiore (inferiore).

3. Scrivere un sottoprogramma FORTRAN $bs(U, B, N, X, INDERR)$ per la procedura di Back-substitution.

4. Scrivere un sottoprogramma FORTRAN $fs(L, B, N, X, INDERR)$ per la procedura di Forward-substitution.

2.16.2 Esercizi §2.5

Esercizi §2.5.2

1. Risolvere, mediante l'algoritmo di eliminazione di Gauss e l'algoritmo di back substitution, il seguente sistema:

$$\begin{cases} 4x - 6y = 16 \\ 8x - 10y + 2z = 30 \\ 4x + 8z = 2 \end{cases}$$

2. Risolvere, mediante l'algoritmo di eliminazione di Gauss e l'algoritmo di back substitution, i seguenti sistemi:

$$a) \begin{cases} 6x_1 - 2x_2 + 2x_3 = -2 \\ 18x_1 - 4x_2 + 2x_3 = -18 \\ 6x_1 + 2x_2 - 4x_3 = -18 \end{cases}$$

$$b) \begin{cases} -3x_1 - 3x_2 + 6x_3 = -15 \\ -9x_1 - 3x_2 + 21x_3 = -66 \\ 3x_1 - 9x_2 - 3x_3 = 30 \end{cases}$$

$$c) \begin{cases} -6x_1 - 12x_2 + 18x_3 = 0 \\ 2x_1 + 8x_2 + 2x_3 = 12 \\ 4x_1 + 16x_2 + 6x_3 = 26 \end{cases}$$

$$d) \begin{cases} 2x_1 + 4x_2 - 10x_3 + 2x_4 = 6 \\ 4x_1 - 6x_2 + 8x_3 - 4x_4 = 4 \\ 8x_1 + 2x_2 - 12x_3 + 6x_4 = 22 \\ 10x_1 + 18x_2 - 40x_3 + 2x_4 = 20 \end{cases}$$

3. Calcolare, mediante l'algoritmo di eliminazione di Gauss, l'inversa di:

$$A = \begin{pmatrix} 4 & 6 \\ 6 & 8 \end{pmatrix}$$

4. Applicare l'algoritmo di eliminazione di Gauss per determinare l'inversa destra delle seguenti matrici. Verificare se le matrici ottenute sono anche inverse sinistre (e quindi inverse).

$$a) \begin{pmatrix} 8 & -24 \\ 32 & 48 \end{pmatrix} \quad b) \begin{pmatrix} -2 & 4 & 2 \\ 0 & 2 & -4 \\ 2 & 8 & -2 \end{pmatrix} \quad c) \begin{pmatrix} 6 & 12 & 6 \\ 12 & 6 & -42 \\ 6 & 18 & 33 \end{pmatrix}$$

5. Applicare l'algoritmo di eliminazione di Gauss ai seguenti sistemi. Se, ad un certo passo k, $a_{k,k}^{(k-1)} = 0$, determinare una strategia che consenta comunque di ottenere la soluzione del sistema.

$$a) \begin{cases} 4x_1 + 12x_2 + 4x_3 = 12 \\ 8x_1 + 64x_2 + 8x_3 = -8 \\ 8x_2 + 8x_3 = -24 \end{cases}$$

$$b) \begin{cases} 10x_1 + 10x_2 - 10x_3 = -20 \\ 10x_1 + 10x_2 - 20x_3 = 40 \\ -5x_1 + 5x_2 + 15x_3 = 20 \end{cases}$$

$$c) \begin{cases} 4x_2 + 2x_3 = 6 \\ -2x_1 + 2x_2 = 4 \\ 4x_1 - 2x_2 + 6x_3 = -10 \end{cases}$$

$$d) \begin{cases} 12x_2 + 6x_3 + 24x_4 = -18 \\ 6x_1 - 6x_2 - 12x_3 = 0 \\ 6x_1 + 36x_4 = -12 \\ -6x_1 + 18x_2 + 12x_3 = 0 \end{cases}$$

$$e) \begin{cases} 3x_1 - 3x_2 + 3x_4 = -9 \\ 6x_1 - 9x_2 - 3x_3 + 9x_4 = -18 \\ 9x_1 - 9x_2 + 6x_3 + 15x_4 = 27 \\ 12x_1 - 6x_2 + 6x_3 = -42 \end{cases}$$

6. Risolvere il seguente sistema, mediante l'algoritmo di eliminazione di Gauss e l'algoritmo di back substitution, utilizzando un sistema aritmetico floating-point normalizzato con $t = 2$ e la tecnica dell'arrotondamento:

$$\begin{cases} 0.98x_1 & + & 0.43x_2 & = & 0.91 \\ -0.61x_1 & + & 0.23x_2 & = & 0.48 \end{cases}$$

Confrontare la soluzione ottenuta con la soluzione vera $x_1^* = 0.005946...$, $x_2^* = 2.102727...$, e con la migliore approssimazione a 2 cifre $x_1' = 0.0059$, $x_2' = 2.1$.

7. Scrivere una procedura per l'algoritmo di eliminazione di Gauss utilizzando l'operazione di tipo *saxpy*.

Esercizi §2.5.3

1. Determinare il numero di moltiplicazioni/divisioni necessario per risolvere, mediante l'algoritmo di eliminazione di Gauss e di back substitution, un sistema di equazioni $p \times p$, per $p = 10$; $p = 50$; $p = 70$; $p = 90$.

Esercizi §2.5.4

1. Stabilire, applicando l'algoritmo di eliminazione di Gauss con pivoting parziale, se il seguente sistema ammette soluzioni:

$$\begin{cases} 2x & + & 2y & + & 2z & = & 0 \\ 2x & + & 4y & + & 6z & = & 0 \\ 6x & + & 10z & + & 14z & = & 2 \end{cases}$$

2. Applicare l'algoritmo di eliminazione di Gauss con pivoting parziale ai seguenti sistemi. Discutere l'esistenza di soluzioni.

$$a) \begin{cases} 3x_1 & + & 6x_2 & - & 10x_3 & = & 6 \\ 6x_1 & - & 9x_2 & + & 12x_3 & = & 12 \\ 12x_1 & + & 3x_2 & - & 18x_3 & = & 24 \end{cases}$$

$$b) \begin{cases} 2x_1 & + & 4x_2 & - & 2x_3 & + & 4x_4 & = & 8 \\ 4x_1 & + & 14x_2 & + & 2x_3 & + & 2x_4 & = & 28 \\ 6x_1 & + & 16x_2 & - & 2x_3 & + & 8x_4 & = & 34 \\ 10x_1 & + & 18x_2 & - & 40x_3 & + & 2x_4 & = & 20 \end{cases}$$

3. Verificare, applicando l'algoritmo di eliminazione di Gauss con pivoting parziale, che le seguenti matrici sono singolari:

$$a)\begin{pmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \quad b)\begin{pmatrix} -2 & 4 & 6 \\ 0 & 2 & 8 \\ 4 & -8 & -12 \end{pmatrix} \quad c)\begin{pmatrix} 3 & -3 & -3 & -6 \\ 6 & -9 & 6 & -3 \\ 9 & -15 & 15 & 0 \\ 3 & -6 & 9 & 3 \end{pmatrix}$$

4. Verificare, applicando l'algoritmo di eliminazione di Gauss con pivoting parziale, che il seguente sistema:

$$\begin{cases} 4x_1 - 8x_2 + 12x_3 = 4 \\ 8x_1 + 4kx_2 + 24x_3 = 24 \\ -4x_1 + 12x_2 + 4(k-3)x_3 = 0 \end{cases}$$

   a) per $k = 0$ ha infinite soluzioni;

   b) per uno specifico valore di k, da determinare, non ha soluzioni;

   c) per tutti gli altri valori di k ammette una sola soluzione.

5. Risolvere, mediante l'algoritmo di eliminazione di Gauss con pivoting parziale, il seguente sistema:

$$\begin{cases} 3x + 3y - 3z = 6 \\ 9x + 9y + 3z = 6 \\ 3x + 3z = 0 \end{cases}$$

6. Risolvere, mediante l'algoritmo di eliminazione di Gauss con pivoting parziale, i seguenti sistemi (esercizio 5 di Esercizi §2.5.2):

$$a)\begin{cases} 4x_1 + 12x_2 + 4x_3 = 12 \\ 8x_1 + 64x_2 + 8x_3 = -8 \\ 8x_2 + 8x_3 = -24 \end{cases}$$

$$b)\begin{cases} 10x_1 + 10x_2 - 10x_3 = -20 \\ 10x_1 + 10x_2 - 20x_3 = 40 \\ -5x_1 + 5x_2 + 15x_3 = 20 \end{cases}$$

$$c)\begin{cases} 4x_2 + 2x_3 = 6 \\ -2x_1 + 2x_2 = 4 \\ 4x_1 - 2x_2 + 6x_3 = -10 \end{cases}$$

$$d)\begin{cases} & 12x_2 + 6x_3 + 24x_4 = -18 \\ 6x_1 - 6x_2 - 12x_3 & = 0 \\ 6x_1 & + 36x_4 = -12 \\ -6x_1 + 18x_2 + 12x_3 & = 0 \end{cases}$$

$$e)\begin{cases} 3x_1 - 3x_2 & + 3x_4 = -9 \\ 6x_1 - 9x_2 - 3x_3 + 9x_4 & = -18 \\ 9x_1 - 9x_2 + 6x_3 + 15x_4 & = 27 \\ 12x_1 - 6x_2 + 6x_3 & = -42 \end{cases}$$

7. Risolvere, applicando l'algoritmo di eliminazione di Gauss *i*) senza pivoting parziale e *ii*) con pivoting parziale, i seguenti sistemi utilizzando un sistema aritmetico floating-point con $t = 3$.

$$a)\begin{cases} 0.005x_1 + x_2 + x_3 = 2 \\ x_1 - 2x_2 + x_3 = 4 \\ -3x_1 - x_2 + 6x_3 = 2 \end{cases} \qquad b)\begin{cases} x_1 - x_2 + 2x_3 = 5 \\ 2x_1 - 2x_2 + x_3 = 1 \\ 3x_1 - 2x_2 + 7x_3 = 20 \end{cases}$$

$$c)\begin{cases} 1.19x_1 + 2.37x_2 - 7.31x_3 + 1.75x_4 = 2.78 \\ 2.15x_1 - 9.76x_2 + 1.54x_3 - 2.08x_4 = 6.27 \\ 10.7x_1 - 1.11x_2 + 3.78x_3 + 4.49x_4 = 9.03 \\ 2.17x_1 + 3.58x_2 + 1.70x_3 + 9.33x_4 = 5.00 \end{cases}$$

Confrontare i risultati.

8. Il pivoting parziale non sempre produce buoni risultati. Per verificare ció, applicare l'algoritmo di Gauss con pivoting parziale, utilizzando un sistema aritmetico floating-point normalizzato con t cifre, al seguente sistema:

$$\begin{cases} 2x_1 + x_2 + x_3 = 1 \\ x_1 + \epsilon x_2 + \epsilon x_3 = 2\epsilon \\ x_1 + \epsilon x_2 - \epsilon x_3 = \epsilon \end{cases}$$

con ϵ molto piccolo e noto alla massima accuratezza del sistema aritmetico. Mostrare, inoltre, che ponendo $z_1 = x_1/\epsilon$, $z_2 = x_2$, $z_3 = x_3$, e dividendo la II e la III equazione per ϵ così da ottenere il sistema:

$$\begin{cases} 2\epsilon z_1 + z_2 + z_3 = 1 \\ z_1 + z_2 + z_3 = 2 \\ z_1 + z_2 - z_3 = 1 \end{cases}$$

l'algoritmo di Gauss con pivoting parziale fornisce una soluzione accurata.

9. Scrivere un sottoprogramma FORTRAN $gausspiv(A, B, N, INDERR)$ per il metodo di eliminazione di Gauss con pivoting parziale (Procedura 2.9), utilizzando l'operazione di tipo *saxpy*.

Esercizi §2.6

1. Applicare l'algoritmo di fattorizzazione LU con pivoting parziale, per calcolare il determinante e l'inversa della seguente matrice:

$$A = \begin{pmatrix} 1.723 & -1.421 & 3.784 \\ 0.113 & 4.071 & 1.213 \\ 5.131 & -4.010 & -2.176 \end{pmatrix}$$

2. Calcolare la fattorizzazione LU (con L unitaria) della seguente matrice:

$$\begin{pmatrix} 2 & 8 & 0 \\ 0 & 6 & 2 \\ 0 & 2 & 0 \end{pmatrix}$$

3. Risolvere, calcolando la fattorizzazione LU (con L unitaria), il seguente sistema:

$$\begin{cases} 12x_1 - 15x_2 + 3x_3 = 45 \\ 6x_1 + 12x_3 = 3 \\ 6x_1 - 9x_2 = 24 \end{cases}$$

4. Applicare l'algoritmo di eliminazione di Gauss per costruire L (L unitaria) ed U tale che la seguente matrice $A = LU$:

$$A = \begin{pmatrix} 8 & -4 & 4 \\ 4 & 6 & -2 \\ 8 & -6 & -4 \end{pmatrix}.$$

5. Risolvere, utilizzando l'algoritmo di fattorizzazione LU, i seguenti sistemi:

$$a) \begin{cases} 8x_1 + 64x_2 + 8x_3 = -8 \\ 4x_1 + 12x_2 + 4x_3 = 12 \\ 8x_2 + 8x_3 = -24 \end{cases}$$

$$b) \begin{cases} 10x_1 + 10x_2 - 10x_3 = -20 \\ -5x_1 + 5x_2 + 15x_3 = 20 \\ 10x_1 + 10x_2 - 20x_3 = 40 \end{cases}$$

$$c) \begin{cases} 4x_1 - 2x_2 + 6x_3 = -10 \\ 4x_2 + 2x_3 = 6 \\ -2x_1 + 2x_2 = 4 \end{cases}$$

$$d)\begin{cases} 6x_1 - 6x_2 - 12x_3 = 0 \\ 12x_2 + 6x_3 + 24x_4 = -18 \\ 6x_1 + 36x_4 = -12 \\ -6x_1 + 18x_2 + 12x_3 = 0 \end{cases}$$

$$e)\begin{cases} 12x_1 - 6x_2 + 6x_3 = -42 \\ 6x_1 - 9x_2 - 3x_3 + 9x_4 = -18 \\ 9x_1 - 9x_2 + 6x_3 + 15x_4 = 27 \\ 3x_1 - 3x_2 + 3x_4 = -9 \end{cases}$$

6. Per ciascuna matrice A seguente calcolare la sua fattorizzazione LU (con L unitaria) e verificare che $A = LU$:

$$a)\begin{pmatrix} 4 & 8 \\ 4 & 2 \end{pmatrix} \quad b)\begin{pmatrix} -6 & 8 & 14 \\ 4 & 8 & 6 \\ 2 & 4 & -12 \end{pmatrix} \quad c)\begin{pmatrix} -4 & -8 & 4 & -4 \\ 2 & 10 & 10 & -16 \\ -2 & 0 & 14 & -22 \\ 4 & 14 & 6 & -6 \end{pmatrix}$$

7. Determinare la fattorizzazione LU della seguente matrice A, con L ed U definite come segue, calcolando gli elementi di L e di U in base alla posizione $A = LU$:

$$A = \begin{pmatrix} 6 & 6 \\ 12 & -3 \end{pmatrix}; \quad L = \begin{pmatrix} l_{1,1} & 0 \\ l_{2,1} & l_{2,2} \end{pmatrix}; \quad U = \begin{pmatrix} 1 & u_{1,2} \\ 0 & 1 \end{pmatrix}$$

8. Esprimere la seguente uguaglianza di matrici con 4 equazioni scalari:

$$\begin{pmatrix} a & 0 \\ b & c \end{pmatrix}\begin{pmatrix} d & e \\ 0 & f \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \quad (LU = A)$$

   Dedurre che A puó avere infinite fattorizzazioni LU. Determinare quella per cui L è unitaria.

9. Dimostrare che se una matrice A non singolare ha una fattorizzazione LU in cui L è triangolare inferiore unitaria e U è triangolare superiore, allora L ed U sono uniche (Sugg: Suppondendo che $A = L_1U_1 = L_2U_2$ da cui si ha $L_2^{-1}L_1 = U_2U_1^{-1}$, dimostrare che l'uguaglianza sussiste solo se I e II membro sono uguali alla matrice identica).

10. Verificare che la matrice:

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

    non ha una fattorizzazione LU in cui L è unitaria.

11. Verificare che ogni matrice della forma:

$$\begin{pmatrix} 0 & a \\ 0 & b \end{pmatrix}$$

ha una fattorizzazione LU. Verificare che, anche se L è triangolare inferiore unitaria, la fattorizzazione non è unica.

12. Verificare che ogni matrice della forma:

$$\begin{pmatrix} 0 & 0 \\ a & b \end{pmatrix}$$

ha una fattorizzazione LU. Verificare se ha una fattorizzazione LU in cui L è unitaria.

13. Determinare tutte le fattorizzazioni LU, in cui L è unitaria, della seguente matrice:

$$A = \begin{pmatrix} 1 & 3 \\ 5 & 15 \end{pmatrix}$$

14. Calcolare la fattorizzazione LU, in cui L è triangolare inferiore ed U è triangolare superiore unitaria, della seguente matrice:

$$A = \begin{pmatrix} 3 & 0 & 1 \\ 0 & -1 & 3 \\ 1 & 3 & 0 \end{pmatrix}$$

15. Calcolare la fattorizzazione LU, in cui sia L sia U sono unitarie, della seguente matrice:

$$A = \begin{pmatrix} 1 & 5 \\ 3 & 16 \end{pmatrix}$$

16. Utilizzare la fattorizzazione LU (con L unitaria) per calcolare le soluzioni $\underline{x}_1$, $\underline{x}_2$ e $\underline{x}_3$ di $A\underline{x}_i = \underline{b}_i$, $i = 1, 2, 3$ con:

$$A = \begin{pmatrix} 12 & 8 & -2 \\ -8 & 32 & 20 \\ 4 & -16 & 8 \end{pmatrix} \quad ;$$

$$\underline{b}_1 = \begin{pmatrix} 58 \\ 92 \\ -64 \end{pmatrix} ; \quad \underline{b}_2 = \begin{pmatrix} 34 \\ -4 \\ 20 \end{pmatrix} ; \quad \underline{b}_3 = \begin{pmatrix} 20 \\ 24 \\ -12 \end{pmatrix}$$

17. Verificare che la complessità computazionale richiesta per risolvere k sistemi $A\underline{x}_i = \underline{b}_i$, $1 \leq i \leq k$, con A matrice qualsiasi, $A \in R^{n\times n}$, calcolando la fattorizzazione LU di A, è sempre minore di quella richiesta per calcolare A^{-1} con l'algoritmo di eliminazione di Gauss e quindi calcolare $\underline{x}_i = A^{-1}\underline{b}_i$, $1 \leq i \leq k$.

18. Scrivere una versione dell'algoritmo di fattorizzazione LU (con L unitaria) che calcoli, al k-mo passo, la k-ma riga di L e la k-ma riga di U (conseguentemente al k-mo passo l'ordine degli elementi da calcolare è $l_{k,1}, l_{k,2}, \ldots, l_{k,k-1}, u_{k,k}.u_{k,k+1}, \ldots, u_{k,n}$) (*algoritmo di Doolittle per righe*). Scrivere poi una versione che calcoli, al k-mo passo, la k-ma colonna di U e la k-ma colonna di L (conseguentemente al k-mo passo l'ordine degli elementi da calcolare è $u_{1,k}, u_{2,k}, \ldots, u_{k,k}, l_{k+1,k}.u_{k+2,k}, \ldots, l_{n,k}$) (*algoritmo di Doolittle per colonne o di Crout*).

19. Determinare quale matrice di permutazione P, di dimensione 3×3, ha l'effetto di scambiare la I con la III riga, di una matrice $A \in R^{3\times 3}$.

20. Risolvere, usando la fattorizzazione LU (con L unitaria), *i*) senza e *ii*) con pivoting parziale, i seguenti sistemi:

$$a)\begin{cases} 3x_1 + 21x_2 + 18x_3 = 6 \\ 3x_1 - 6x_2 - 6x_3 = 0 \\ 6x_1 + 6x_2 - 12x_3 = 24 \end{cases} \qquad b)\begin{cases} 2x_1 - 2x_2 + 4x_3 = 10 \\ 4x_1 - 4x_2 + 2x_3 = 2 \\ 6x_1 - 4x_2 + 14x_3 = 40 \end{cases}$$

$$c)\begin{cases} 3x_1 + 6x_2 + 3x_3 - 3x_4 = 3 \\ 3x_1 + 3x_2 + 3x_3 + 3x_4 = 6 \\ -6x_1 + 3x_2 - 6x_3 + 9x_4 = 3 \\ 9x_1 - 3x_2 + 12x_3 - 3x_4 = 0 \end{cases}$$

21. Per ciascuna matrice A seguente, determinare la sua fattorizzazione P^TLU, applicando l'algoritmo di fattorizzazione LU con pivoting parziale.

$$a)\begin{pmatrix} 0 & 4 \\ 4 & 8 \end{pmatrix} \quad b)\begin{pmatrix} 2 & 6 & -4 \\ -4 & -12 & 11 \\ 3 & 14 & -16 \end{pmatrix} \quad c)\begin{pmatrix} 4 & 7.56 & 0.36 & -4.38 \\ -8 & -15.12 & 0.72 & 6.76 \\ 6 & 21 & 4 & 8 \\ 10 & 12 & 2 & -16 \end{pmatrix}$$

22. Utilizzare l'algoritmo di fattorizzazione LU con pivoting parziale, per calcolare il determinante e l'inversa (se esiste) delle seguenti matrici:

$$a)A = \begin{pmatrix} 1 & 7 & 6 \\ 1 & -2 & -1 \\ 2 & 2 & -4 \end{pmatrix} \quad b)A = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 7 & -1 \\ 7 & 8 & -11 \end{pmatrix}$$

Esercizi §2.6.1

1. Risolvere, applicando l'algoritmo di fattorizzazione LU con pivoting parziale con scambio virtuale delle righe, i seguenti sistemi: (esercizio 5 di Esercizi §2.5.2)

$$a)\begin{cases} 4x_1 + 12x_2 + 4x_3 = 12 \\ 8x_1 + 64x_2 + 8x_3 = -8 \\ 8x_2 + 8x_3 = -24 \end{cases}$$

$$b)\begin{cases} 10x_1 + 10x_2 - 10x_3 = -20 \\ 10x_1 + 10x_2 - 20x_3 = 40 \\ -5x_1 + 5x_2 + 15x_3 = 20 \end{cases}$$

$$c)\begin{cases} 4x_2 + 2x_3 = 6 \\ -2x_1 + 2x_2 = 4 \\ 4x_1 - 2x_2 + 6x_3 = -10 \end{cases}$$

$$d)\begin{cases} 12x_2 + 6x_3 + 24x_4 = -18 \\ 6x_1 - 6x_2 - 12x_3 = 0 \\ 6x_1 + 36x_4 = -12 \\ -6x_1 + 18x_2 + 12x_3 = 0 \end{cases}$$

$$e)\begin{cases} 3x_1 - 3x_2 + 3x_4 = -9 \\ 6x_1 - 9x_2 - 3x_3 + 9x_4 = -18 \\ 9x_1 - 9x_2 + 6x_3 + 15x_4 = 27 \\ 12x_1 - 6x_2 + 6x_3 = -42 \end{cases}$$

2. Determinare, per ciascun sistema dell'esercizio precedente, il vettore finale $IPIV$.

3. Scrivere un sottoprogramma FORTRAN $sistlin(A, B, N, X, INDERR)$ per la risoluzione di un generico sistema lineare.

2.16.3 Esercizi §2.7

1. Dimostrare che, qualunque sia la norma utilizzata, si ha che

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

2. Per la seguente matrice

$$A = \begin{pmatrix} 1 & 0 \\ 0 & k \end{pmatrix}, \quad k > 0$$

determinare:

a) $\mu(A)$ in norma infinito;

b) la piú "vicina" matrice singolare in un sistema aritmetico floating-point normalizzato caratterizzato dai seguenti parametri: $\beta = 10; t = 3; E_{min} = -3; E_{max} = 3$, utilizzando la tecnica di troncamento.

3. Calcolare $\mu(A)$ in norma infinito della seguente matrice:

$$A = \begin{pmatrix} 1 & 1+\epsilon \\ 1-\epsilon & 1 \end{pmatrix}, \quad \epsilon > 0.$$

4. Calcolare $\mu(A)$ in norma infinito della seguente matrice:

$$A = \begin{pmatrix} 14 & 16 \\ 18 & 20 \end{pmatrix}$$

5. Data la seguente matrice:

$$A = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}$$

la cui matrice inversa è:

$$A^{-1} = \begin{pmatrix} 1 & -k \\ 0 & 1 \end{pmatrix},$$

si ha che, sia in norma uno sia in norma infinito:

$$\|A\| = \|A^{-1}\| = 1 + k, \quad \forall k \geq 0$$

cosicché $\mu(A) = (1+k)^2$, che è grande per k grande. Se si considera il sistema di equazioni $A\underline{x} = \underline{b}$ con:

$$\underline{b} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

si ha che la soluzione è

$$\underline{x} = \begin{pmatrix} 1-k \\ 1 \end{pmatrix}.$$

Perturbando il termine noto mediante $\delta \neq 0$ a

$$\underline{b} + \delta\underline{b} = \begin{pmatrix} 1+\delta \\ 1+\delta \end{pmatrix},$$

si ha che la variazione $\delta\underline{x}$ nella soluzione è:

$$\delta\underline{x} = \begin{pmatrix} \delta(1-k) \\ \delta_2 \end{pmatrix}.$$

Determinare una maggiorazione di $\|\delta\underline{x}\|/\|\underline{x}\|$ usando la norma infinito, per mostrare che il sistema è ben condizionato, anche avendo A un indice di condizionamento grande.

6. Ripetere l'esercizio precedente ponendo:

$$\underline{b} = \begin{pmatrix} 1 \\ -1/k \end{pmatrix}.$$

Verificare che in questo caso il sistema è mal condizionato.

7. Verificare, introducendo una perturbazione $\delta\underline{b} = (0.,\ 0.00001)^T$ nel termine noto $\underline{b}$, che il seguente sistema, la cui soluzione esatta è $\underline{x} = (1., 1.)^T$, è mal condizionato:

$$\begin{cases} x_1 + x_2 = 2.00000 \\ 1.00001x_1 + x_2 = 2.00001 \end{cases}$$

Calcolare $\mu(A)$ usando la norma infinito.

8. Considerato il seguente sistema:

$$\begin{cases} 2.67x_1 + 1.59x_2 = 1.08 \\ 1.41x_1 + 0.84x_2 = 0.57 \end{cases}$$

che ha soluzione esatta $x_1 = 1.41$, $x_2 = 0.84$, determinare:

a) $\delta\underline{b}$ cosicché, sostituendo $\underline{b}$ con $\underline{b} + \delta\underline{b}$, la soluzione calcolata sia $x_1 = 0.47$, $x_2 = -0.11$;

b) se il sistema è ben o mal condizionato;

c) l'indice di condizionamento usando la norma infinito.

9. Utilizzare `MATLAB` per ottenere una soluzione approssimata del sistema dell'esercizio precedente. Determinare l'errore assoluto tra la soluzione calcolata e la soluzione esatta, in norma infinito.

10. Risolvere il seguente sistema:

$$\begin{cases} 1.34x_1 + 7.21x_2 + 1.04x_3 = 9.60 \\ 3.18x_1 + 4.01x_2 + 0.98x_3 = 8.17 \\ 2.84x_1 - 2.40x_2 - 2.24x_3 = -23.4 \end{cases}$$

applicando l'algoritmo di eliminazione di Gauss con pivoting parziale, utilizzando un sistema aritmetico floating-point con $t = 3$. Perturbare, poi, il termine noto della I equazione a 9.59, e risolvere alla stessa maniera il sistema così ottenuto. Cosa si puó dedurre per questo sistema?

2.17 Esercizi con il MATLAB

2.17.1 Esercizi §2.14.4

1. Definire i seguenti vettori:

   a) a=(1.01 2.3 $\sqrt{2}$ cos (1))

   b) b=(1 2 3 4) mediante l'operatore *colon*

   c) c=(4 3 2 1) mediante l'operatore *colon*

   d) d=(1 1.5 2 2.5 3) mediante l'operatore *colon*.

   Calcolare poi la somma e la differenza dei vettori b e c, infine calcolare il loro prodotto scalare, verificando la relazione:

   $$bc^T = cb^T$$

2. Estrarre la diagonale principale, la prima colonna, la seconda riga e il blocco 2x2 $A_{1,2;1,2}$ della seguente matrice:

   $$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

3. Costruire la matrice C aggiungendo alla matrice A dell'esercizio precedente la riga

   $$a = \begin{pmatrix} 7 & 8 & 9 \end{pmatrix}$$

   calcolare poi la trasposta di C, e la matrice che si ottiene aggiungendo a C la colonna a^T.

4. Data la matrice A dell'esercizio 2 e la matrice:

   $$B = \begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{pmatrix}$$

   provare tutte le possibili combinazioni di A e B e le loro trasposte per costruire una nuova matrice, e riflettere sui risultati ottenuti.

5. Verificare con un esempio le seguenti relazioni:

   $$(C^T)^T = C$$
   $$(C^T + D^T) = (C + D)^T$$
   $$(\lambda C)^T = \lambda C^T$$

   (Si utilizzi la matrice C dell'esercizio 3, λ =0.5, D= λ C).

6. Si calcoli la trasposta della seguente matrice:

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

Si verifichi che E^T-E=0. Perché?

7. Costruire le seguenti matrici:

a) la matrice diagonale 4x4 D con $D_{ii} = i$ per i=1,2,3,4

b) la matrice identica di ordine 4

c) la matrice 4x4 O, tale che $O_{ii} = 1$

d) la matrice nulla 4x4

e) i vettori riga $(1, 1, 1, 1)$ e $(0, 0, 0, 0)$.

8. Date le matrici

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad B = \begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{pmatrix}$$

calcolarne il prodotto. Verificare che $(AB)^T = B^T A^T$, e che le matrici $A^T A, B^T B$ sono quadrate e simmetriche.

9. Date le seguenti matrici:

$$A_{11} = \begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix} \quad A_{12} = \begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix} \quad A_{21} = \begin{pmatrix} 9 & 10 \\ 13 & 14 \end{pmatrix} \quad A_{22} = \begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix}$$

$$B_{11} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad B_{12} = B_{11} \quad B_{21} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \quad B_{22} = B_{21}$$

si consideri

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Si verifichi che

$$A + B = \begin{pmatrix} A_{11} + B_{11} & A_{12} + B_{12} \\ A_{21} + B_{21} & A_{22} + B_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

10. Costruire una m-function che conta gli elementi nulli in una matrice.

11. Dato il seguente sistema lineare:

$$\begin{cases} x_1 + 2x_3 + 3x_4 = 1 \\ -x_1 + 2x_2 + 2x_3 - 3x_4 = -1 \\ x_2 + x_3 + 4x_4 = 2 \\ 6x_1 + 2x_2 + 2x_3 + 4x_4 = 1 \end{cases}$$

si chiede di:

a) Verificare che il sistema è compatibile.

b) Risolvere il sistema mediante l'operatore "\".

c) Risolvere il sistema mediante la fattorizzazione LU della matrice dei coefficienti.

d) Risolvere il sistema calcolando l'inversa della matrice dei coefficienti.

e) Calcolare mediante la funzione *flops* la complessità di tempo dei tre procedimenti.

12. Calcolare la norma uno, la norma euclidea, la norma infinito di:

$$A = \begin{pmatrix} 2 & -3 & 1 \\ 6 & 2 & 0 \\ -1 & 3 & 2 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ -1 \\ 6 \end{pmatrix}.$$

Verificare poi le seguenti relazioni valide in generale:

$$\|A\|_1 = \max_j \sum_i |a_{ij}| \quad \|A\|_\infty = \max_i \sum_j |a_{ij}|$$

$$\|b\|_1 = \sum_j |b_j| \quad \|b\|_1 = \max_j |b_j|$$

13. Data la matrice dell'esercizio precedente calcolare $\mu(A)$ e verificare le seguenti relazioni:

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

b) $\mu(A) = \mu(\alpha A)$ per $\forall \alpha$

c) $\mu(AA^T) = \mu(A)^2$

d) $\mu(A) = \mu(A^T) = \mu(A^{-1})$

14. Risolvere con `MATLAB` i sistemi lineari del secondo esercizio della sezione 2.16.2, e verificare i risultati ottenuti, inoltre valutare il condizionamento delle matrici dei coefficienti sia tramite *cond* che tramite *rcond*.

15. Risolvere con MATLAB i sistemi lineari del quinto esercizio della sezione 2.16.2, e verificare i risultati ottenuti, inoltre valutare il condizionamento delle matrici dei coefficienti sia tramite *cond* che tramite *rcond*.

16. Costruire due m-function una per la backward substitution e una per la forward substitution. Testarle poi con i sistemi del primo esercizio della sezione 2.16.1.

Bibliografia

[1] Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Sorensen D. - *LAPACK Users' Guide* - terza edizione, SIAM, 1999.

[2] Demmel J. - *Applied numerical linear algebra* - SIAM, 1997.

[3] Dongarra J.J., Bunch J.R., Moler C.B., Stewart G.W. - *LINPACK Users' Guide* - SIAM, 1979.

[4] Dongarra J.J., Du Croz J., Duff I.S., Hammarling S. - *A set of Level 3 Basic Linear Algebra Subprograms* - ACM Transactions on Mathematical Software, vol. 16 (1990), pp. 1-17.

[5] Dongarra J.J., Du Croz J., Hammarling S., Hanson R.J. - *An extended set of FORTRAN basic linear algebra subroutines* - ACM Transactions on Mathematical Software, vol. 14 (1988), pp. 1-17.

[6] Epperson J.F. - *Introduzione all'analisi numerica* Teoria, metodi, algoritmi - McGraw-Hill.

[7] Fletcher C.A.J. - *Computational Techniques for Fluid Dynamics* - volume 1, seconda edizione, Springer, 1991.

[8] Golub G.H., Van Loan C.F. - *Matrix Computations* - third ed., The Johns Hopkins University Press, 1997.

[9] Lawson C.L., Hanson R.J., Kincaid D., Krogh F.T. - *Basic linear algebra subprograms for Fortran usage* - ACM Transactions on Mathematical Software, vol. 5 (1979), pp. 308-323.

[10] Smith B.T., Boyle J.M., Dongarra J.J., Garbow B.S., Ikebe Y., Klema V.C., Moler C.B. - *Matrix Eigensystem Routines - EISPACK Guide* - Lecture Notes in Computer Science, vol. 6, Springer-Verlag, 1976.

[11] Trefethen L.N., Bau D. - *Numerical linear algebra* - SIAM, 1997.