Capitolo 4

Calcolo matriciale: metodi iterativi

4.1 Introduzione

Nella risoluzione di sistemi di equazioni lineari la scelta dell'algoritmo risolutivo più efficace richiede un'analisi delle caratteristiche del sistema. In molte applicazioni i sistemi sono di **grandi dimensioni** (**di ordine elevato**), cioè con un grande numero di equazioni e quindi di incognite, e **sparsi**, cioè i coefficienti di molte incognite sono uguali a zero (questo significa che ogni equazione coinvolge solo un piccolo numero di incognite).

♣ **Esempio 4.1.** Si consideri un oggetto bidimensionale. Si assuma per semplicità che l'oggetto sia quadrato e sia suddiviso in $n = d^2$ quadrati (vedi Figura 4.1, in cui $d = 8$), detti *pixel*, numerati da 1 a n. Una sorgente di raggi-X emette un fascio di raggi che, dopo aver attraversato l'oggetto, è intercettato da un rilevatore che ne misura l'assorbimento totale da parte dell'oggetto. Si vuole calcolare il coefficiente di assorbimento di ogni pixel in cui è stato suddiviso l'oggetto.
Tale tecnica, detta *tomografia*, è utilizzata in medicina diagnostica; l'oggetto è una sezione piana del corpo umano e viene irradiata con raggi-X al fine di scoprire eventuali anomalie organiche (radiografia, TAC, ...).
Per ogni raggio che attraversa l'oggetto l'assorbimento totale del raggio è dato dalla somma degli assorbimenti nei pixel. Indicato con b_i l'assorbimento totale del raggio $i-$mo, misurato dal rilevatore, con x_j, $j = 1, ..., n$, il coefficiente di assorbimento del j-mo pixel (incognite del problema) e con a_{ij} il peso del contributo del j-mo pixel all'assorbimento totale dell'i-mo raggio, si ha:

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i \quad . \tag{4.1}$$

Se n è il numero di raggi proiettati, si ottiene un sistema di n equazioni in n incognite, che nel seguito si chiamerà sistema `PIXEL`, la cui soluzione costituisce l'insieme dei coefficienti di assorbimento dei pixel in cui è stato suddiviso l'oggetto.
Supponendo che i raggi siano proiettati perpendicolarmente (vedi Figura 4.1), ogni raggio attraversa solo d pixel. Di conseguenza, la generica equazione (4.1) del sistema `PIXEL` ha almeno $n - d$ coefficienti a_{ij} uguali a zero, cioè quelli relativi ai pixel non attraversati dall'i-mo raggio e che, quindi, non danno contributo al suo assorbimento. Il numero totale di coefficienti nulli del sistema `PIXEL` è quindi $n\times(n-d)$.
Una misura di "quanto è sparso" il sistema `PIXEL` si ha considerando il rapporto tra il numero di coefficienti nulli e il numero complessivo dei coefficienti $(n \times n)$, cioè:

$$SP = \frac{n(n-d)}{n^2} = 1 - \frac{d}{n}.$$

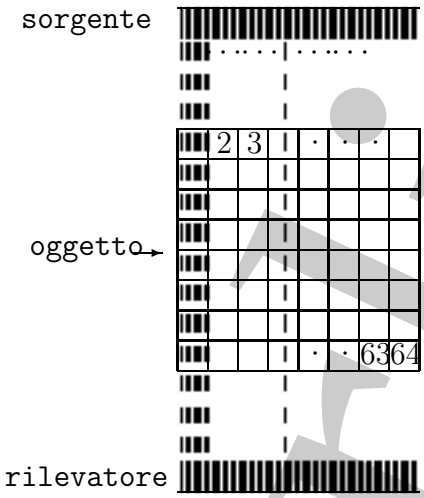


Figura 4.1: Analisi della struttura interna di un oggetto mediante proiezione di raggi-X.

d	n^2	$n(n-d)$	SP
16	65536	61440	0.937
32	1048576	1015808	0.968
64	16777216	16515072	0.984
128	268435456	266338304	0.992

Tabella 4.1: Grado di sparsità del sistema `PIXEL` al variare di d

Nel caso considerato in Figura 4.1 si ha:

$$
\begin{aligned}
&d = 8\\
&n = d^2 = 64\\
&n \times (n-d) = 3584\\
&n \times n = 4096
\end{aligned}
$$

e, quindi, $SP = 0.875$. In altri termini, il numero di coefficienti nulli del sistema `PIXEL` è l'87.5% del numero complessivo di coefficienti. In Tabella 4.1 sono riportati i valori di SP al variare di d tra 16 e 128. Come si aspettava, la "sparsità" del sistema `PIXEL` aumenta al crescere di d e quindi delle sue dimensioni. Si osserva, infine, che, nella pratica, per effettuare un'analisi sufficientemente accurata della struttura interna dell'oggetto, è necessario considerare sia un numero di pixel elevato sia un numero di raggi elevato. In generale $n > 10^5$ e, di conseguenza, il sistema di equazioni risultante è da considerarsi di grandi dimensioni e con sparsità elevata.

Definizione 4.1.1. (Grado di sparsità)
Dato un sistema lineare di n equazioni in n incognite, sia p il numero dei coefficienti diversi da zero. Si definisce **grado di sparsità** *del sistema la quantità:*

$$SP = \frac{n^2 - p}{n^2} = 1 - \frac{p}{n^2}.$$

Si ha $0 \leq SP \leq 1$[1].

Se il sistema è di grandi dimensioni ed ha anche un grado di sparsità elevato è importante utilizzare metodi *ad hoc* che sfruttino, da un punto di vista della complessità computazionale, il fatto che la maggior parte dei coefficienti sono nulli. Per chiarire questa considerazione si supponga di voler risolvere un sistema di 10 equazioni in 10 incognite. Il metodo di eliminazione di Gauss applicato a tale sistema richiede circa 400 operazioni floating-point per ottenere la soluzione. Se il sistema ha solo 2 coefficienti non nulli per ciascuna equazione, il numero totale di coefficenti diversi da zero è 20, cioè la quinta parte del numero complessivo di coefficienti del sistema. Sarebbe, quindi, desiderabile ottenere la soluzione con un numero di operazioni uguale a 400/5. Sfortunatamente, la complessità del metodo di eliminazione di Gauss è, in generale, sempre la stessa, anche se il sistema presenta molti coefficienti uguali a zero. Ciò è dovuto al fatto che il metodo di eliminazione di Gauss modifica i coefficienti del sistema[2], durante il processo di trasformazione del sistema dato in un sistema triangolare equivalente.

[1]Data una matrice A di dimensione $n \times m$, se p è il numero degli elementi non nulli, si ha:

$$SP = \frac{nm - p}{nm} = 1 - \frac{p}{nm}.$$

Se tutti gli elementi di A sono non nulli, allora $p = n \times m$ e si ha $SP = 0$. Se, al contrario, tutti gli elementi di A sono nulli (in tal caso A è la matrice nulla), allora $p = 0$ e $SP = 1$. Quindi, si ha $0 \leq SP \leq 1$.
In particolare, considerando le matrici quadrate ($n = m$) non singolari (cioè con determinante non nullo), una matrice che ha il minor numero di elementi non nulli è quella diagonale. Tale matrice ha, in questa classe di matrici, il massimo grado di sparsità, che è dato da:

$$SP = \frac{n(n-1)}{n^2} = 1 - \frac{1}{n}.$$

Si può affermare che una matrice (o un sistema lineare) ha un elevato grado di sparsità se SP è molto vicino a 1.
Si osservi, infine, che, data una matrice A quadrata di ordine n, se d è il numero di elementi diversi da zero su ciascuna riga (come nella matrice dei coefficienti del sistema PIXEL dell'esempio 4.1), si ha:

$$SP = \frac{n(n-d)}{n^2} = 1 - \frac{d}{n}.$$

[2]A causa di tali modifiche può accadere che coefficienti uguali a zero diventino non nulli. Tale fenomeno è detto **fill-in**.

♣ **Esempio 4.2.** Si consideri il seguente sistema lineare di 3 equazioni in 3 incognite:

$$\begin{cases} 3x_1 & +4x_2 & + x_3 & = 7 \\ 2x_1 & & +4x_3 & = 6 \\ x_1 & +2x_2 & & = 3. \end{cases}$$

Dopo il primo passo del metodo di eliminazione di Gauss tale sistema è trasformato nel sistema equivalente:

$$\begin{cases} 3x_1 & + & 4x_2 & + & x_3 & = 7 \\ & & -8/3x_2 & +10/3x_3 & = 4/3 \\ & & 2/3x_2 & - & 1/3x_3 & = 2/3. \end{cases}$$

Si osserva che i coefficienti a_{22} e a_{33}, inizialmente nulli, sono divenuti non nulli (rispettivamente $-8/3$ e $-1/3$) dopo l'applicazione del primo passo del metodo. ♣

Il metodo di eliminazione di Gauss, almeno nella sua forma classica[3], non "sfrutta" il fatto che il sistema ha molti coefficienti nulli. Tale inefficienza, che è di tutti i metodi diretti, è ancora più evidente per sistemi sparsi di dimensioni maggiori rispetto a quelle del sistema dell'esempio 4.2.
In tal caso, una valida alternativa è costituita dai **metodi iterativi**. Essi consentono di sfruttare l'eventuale sparsità del sistema perché non modificano i suoi coefficienti. Si analizzano, di seguito, le caratteristiche fondamentali dei metodi iterativi e gli aspetti di base per un loro utilizzo efficiente. In particolare, tale analisi è svolta, inizialmente, attraverso l'esame di due tra i più noti ed antichi metodi iterativi, il **metodo di Jacobi** ed il **metodo di Gauss-Seidel**.

4.2 I metodi di Jacobi e Gauss-Seidel

♣ **Esempio 4.3.** Si consideri il seguente sistema lineare di 3 equazioni in 3 incognite:

$$\begin{cases} 8x_1 & - x_2 & +2x_3 & = 56 \\ x_1 & -4x_2 & + x_3 & = -1 \\ x_1 & +2x_2 & -5x_3 & = -37. \end{cases} \tag{4.2}$$

Per risolvere tale sistema si procede nel modo seguente. Si riscrive il sistema (4.2) in modo che a primo membro della i-ma equazione, $i = 1, 2, 3$, compaia solo l'incognita x_i. Questa semplice "riscrittura" può sempre essere fatta[4], eventualmente, modificando l'ordine delle equazioni:

$$\begin{cases} 8x_1 & = & x_2 & -2x_3 & +56 \\ -4x_2 & = -x_1 & - x_3 & - 1 \\ -5x_3 & = -x_1 & -2x_2 & -37. \end{cases} \tag{4.3}$$

[3] È importante ricordare che per alcune classi di sistemi di equazioni lineari sparsi, in cui i coefficienti non nulli sono disposti secondo un preciso schema (in tal caso si parla di **sistemi sparsi strutturati**), esistono opportune versioni del metodo di eliminazione di Gauss o, in generale, opportuni metodi diretti (ad esempio il metodo di Cholesky per sistemi lineari a banda simmetrici definiti positivi), che consentono di minimizzare il fenomeno del fill-in e, di conseguenza, di ridurre la complessità computazionale.

[4] Se il sistema è non singolare.

Si dividono ambo i membri della i-ma equazione del sistema (4.3), $i = 1, 2, 3$, per il coefficiente di x_i. Il sistema (4.2) è quindi trasformato nel sistema equivalente:

$$\begin{cases} x_1 = & \frac{1}{8}(& x_2 & -2x_3 & +56) \\ x_2 = & -\frac{1}{4}(-x_1 & - & x_3 & - 1) \\ x_3 = & -\frac{1}{5}(-x_1 & -2x_2 & & -37). \end{cases} \tag{4.4}$$

Questa rappresentazione del sistema è il punto di partenza per costruire un processo iterativo: infatti, se si scelgono ad arbitrio dei valori per le incognite x_1, x_2 e x_3 utilizzandoli nel secondo membro delle equazioni (4.4), si ottengono nuovi valori per x_1, x_2 e x_3. Indicati con $x_1^{(0)}$, $x_2^{(0)}$ e $x_3^{(0)}$ i valori iniziali scelti si ha:

$$\begin{cases} x_1^{(1)} = & \frac{1}{8}(& x_2^{(0)} & -2x_3^{(0)} & +56) \\ x_2^{(1)} = & -\frac{1}{4}(-x_1^{(0)} & - & x_3^{(0)} & - 1) \\ x_3^{(1)} = & -\frac{1}{5}(-x_1^{(0)} & -2x_2^{(0)} & & -37). \end{cases}$$

Il passo successivo consiste nell'utilizzare nel secondo membro delle equazioni (4.4) i nuovi valori $x_1^{(1)}$, $x_2^{(1)}$ e $x_3^{(1)}$ al posto dei valori iniziali corrispondenti, cioè:

$$\begin{cases} x_1^{(2)} = & \frac{1}{8}(& x_2^{(1)} & -2x_3^{(1)} & +56) \\ x_2^{(2)} = & -\frac{1}{4}(-x_1^{(1)} & - & x_3^{(1)} & - 1) \\ x_3^{(2)} = & -\frac{1}{5}(-x_1^{(1)} & -2x_2^{(1)} & & -37). \end{cases}$$

In generale, se $x_1^{(k)}$, $x_2^{(k)}$ e $x_3^{(k)}$ sono i valori delle incognite x_1, x_2 e x_3 ottenuti dopo k passi, si calcolano nuovi valori $x_1^{(k+1)}$, $x_2^{(k+1)}$ e $x_3^{(k+1)}$ mediante le equazioni:

$$\begin{cases} x_1^{(k+1)} = & \frac{1}{8}(& x_2^{(k)} & -2x_3^{(k)} & +56) \\ x_2^{(k+1)} = & -\frac{1}{4}(-x_1^{(k)} & - & x_3^{(k)} & - 1) \\ x_3^{(k+1)} = & -\frac{1}{5}(-x_1^{(k)} & -2x_2^{(k)} & & -37). \end{cases} \tag{4.5}$$

Il procedimento descritto è, quindi, di tipo iterativo; a partire da valori iniziali arbitrari delle incognite $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)})$, si esegue una sequenza di passi, detti **iterazioni**, in ognuno dei quali si calcolano nuovi valori per le incognite "sostituendo" i valori calcolati al passo precedente. Si noti inoltre che ad ogni iterazione i coefficienti del sistema (4.4) non cambiano e sono semplicemente legati in maniera opportuna ai coefficienti del sistema iniziale (4.2).
Le (4.5) definiscono la generica iterazione del procedimento descritto, noto con il nome di **metodo di Jacobi**. Nella Tabella 4.2 sono riportati i valori[5], arrotondati alla sesta cifra decimale, ottenuti nelle prime 9 iterazioni del metodo di Jacobi, avendo scelto come valori iniziali $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$. La soluzione del sistema (4.2) è $x_1 = 5.000$, $x_2 = 4.000$ e $x_3 = 10.000$. Si osserva che già dopo 5 iterazioni si hanno valori che sono sufficientemente vicini alla soluzione. ♣

Si consideri il seguente sistema lineare di n equazioni in n incognite:

$$\begin{cases} a_{11}x_1 + & a_{12}x_2 & + \ldots + & a_{1n}x_n & = b_1 \\ a_{21}x_1 + & a_{22}x_2 & + \ldots + & a_{2n}x_n & = b_2 \\ \ldots & \ldots & \ldots & \ldots & = \ldots \\ a_{n1}x_1 + & a_{n2}x_2 & + \ldots + & a_{nn}x_n & = b_n. \end{cases} \tag{4.6}$$

[5] Tali valori e tutti i risultati sperimentali riportati in seguito si riferiscono ad elaborazioni effettuate utilizzando il sistema aritmetico standard IEEE.

Per risolvere tale sistema si procede nel modo seguente. Si riscrive il sistema (4.6) in modo che a primo membro della i-ma equazione, $i = 1, \ldots, n$, compaia solo l'incognita x_i. Questa semplice "riscrittura" é sempre possibile, eventualmente modificando l'ordine delle equazioni:

$$\begin{cases} a_{11}x_1 = & -a_{12}x_2 - & a_{13}x_3 & -\ldots+ & b_1 \\ a_{22}x_2 = & -a_{21}x_1 - & a_{23}x_3 & -\ldots+ & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{kk}x_k = & -a_{k1}x_1 - & a_{k2}x_2 & -\ldots+ & b_k \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{nn}x_n = & -a_{n1}x_1 - & a_{n2}x_2 & -\ldots+ & b_n. \end{cases} \tag{4.7}$$

Si dividono ambo i membri della i-ma equazione del sistema (4.7), $i = 1, \ldots, n$, per il coefficiente di x_i. Il sistema (4.6) è quindi trasformato nel sistema equivalente:

$$\begin{cases} x_1 = & -\frac{1}{a_{11}}(a_{12}x_2 + & a_{13}x_3 & +\ldots- & b_1) \\ x_2 = & -\frac{1}{a_{22}}(a_{21}x_1 + & a_{23}x_3 & +\ldots- & b_2) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_k = & -\frac{1}{a_{kk}}(a_{k1}x_1 + & a_{k2}x_2 & +\ldots- & b_k) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_n = & -\frac{1}{a_{nn}}(a_{n1}x_1 + & a_{n2}x_2 & +\ldots- & b_n). \end{cases} \tag{4.8}$$

Questa rappresentazione del sistema è il punto di partenza per costruire un processo iterativo: infatti, se si scelgono ad arbitrio dei valori per le incognite x_1, x_2, ..., x_n, utilizzandoli nel secondo membro delle equazioni (4.8), si ottengono nuovi valori per x_1, x_2, ..., x_n. Indicati con $x_1^{(0)}$, $x_2^{(0)}$, ..., $x_n^{(0)}$ i valori iniziali scelti si ha:

$$\begin{cases} x_1^{(1)} = & -\frac{1}{a_{11}}(a_{12}x_2^{(0)} + & a_{13}x_3^{(0)} & +\ldots- & b_1) \\ x_2^{(1)} = & -\frac{1}{a_{22}}(a_{21}x_1^{(0)} + & a_{23}x_3^{(0)} & +\ldots- & b_2) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_k^{(1)} = & -\frac{1}{a_{kk}}(a_{k1}x_1^{(0)} + & a_{k2}x_2^{(0)} & +\ldots- & b_k) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_n^{(1)} = & -\frac{1}{a_{nn}}(a_{n1}x_1^{(0)} + & a_{n2}x_2^{(0)} & +\ldots- & b_n) \end{cases}$$

Il passo successivo consiste nell'utilizzare, nel secondo membro delle equazioni (4.8), i

nuovi valori $x_1^{(1)}$, $x_2^{(1)}$, ..., $x_n^{(1)}$ al posto dei valori iniziali corrispondenti, cioè:

$$\begin{cases} x_1^{(2)} = -\frac{1}{a_{11}}(a_{12}x_2^{(1)} + a_{13}x_3^{(1)} + \ldots - b_1) \\ x_2^{(2)} = -\frac{1}{a_{22}}(a_{21}x_1^{(1)} + a_{23}x_3^{(1)} + \ldots - b_2) \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ x_k^{(2)} = -\frac{1}{a_{kk}}(a_{k1}x_1^{(1)} + a_{k2}x_2^{(1)} + \ldots - b_k) \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ x_n^{(2)} = -\frac{1}{a_{nn}}(a_{n1}x_1^{(1)} + a_{n2}x_2^{(1)} + \ldots - b_n). \end{cases}$$

In generale, se $x_1^{(k)}$, $x_2^{(k)}$, ..., $x_n^{(k)}$ sono i valori delle incognite x_1, x_2, ..., x_n ottenuti dopo k passi, si calcolano i nuovi valori $x_1^{(k+1)}$, $x_2^{(k+1)}$, ..., $x_n^{(k+1)}$ mediante le equazioni:

$$\begin{cases} x_1^{(k+1)} = -\frac{1}{a_{11}}(a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + \ldots - b_1) \\ x_2^{(k+1)} = -\frac{1}{a_{22}}(a_{21}x_1^{(k)} + a_{23}x_3^{(k)} + \ldots - b_2) \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ x_k^{(k+1)} = -\frac{1}{a_{kk}}(a_{k1}x_1^{(k)} + a_{k2}x_2^{(k)} + \ldots - b_k) \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ x_n^{(k+1)} = -\frac{1}{a_{nn}}(a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \ldots - b_n). \end{cases} \tag{4.9}$$

Il procedimento descritto è, quindi, di tipo iterativo; a partire da valori iniziali arbitrari, delle incognite ($x_1^{(0)}$, $x_2^{(0)}$, ..., $x_n^{(0)}$), si esegue una sequenza di passi, detti **iterazioni**, in ognuno dei quali si calcolano nuovi valori per le incognite "sostituendo" i valori calcolati al passo precedente. Si noti inoltre che ad ogni iterazione i coefficienti del sistema (4.8) non cambiano e sono semplicemente legati in maniera opportuna ai coefficienti del sistema iniziale (4.6).
Le (4.9) definiscono la generica iterazione del procedimento descritto, noto con il nome di **metodo di Jacobi**.
Nella notazione utilizzata, gli indici in alto si riferiscono al numero di iterazioni; in altre parole, $x_1^{(k+1)}$, $x_2^{(k+1)}$, $x_n^{(k+1)}$ sono i valori ottenuti alla $(k+1)$-ma iterazione utilizzando i valori $x_1^{(k)}$, $x_2^{(k)}$ e $x_n^{(k)}$ calcolati nell'iterazione precedente.

♣ **Esempio 4.4.** Riprendiamo l'esempio 4.3. Considerando le (4.5), si osserva che quando $x_1^{(k+1)}$ è stato calcolato, è possibile utilizzare tale valore al posto di $x_1^{(k)}$, per determinare il valore $x_2^{(k+1)}$. Analogamente, per calcolare $x_3^{(k+1)}$ è possibile utilizzare i valori di x_1 e x_2 già calcolati alla $(k+1)$-ma iterazione. Quanto detto conduce al seguente metodo iterativo:

$$\begin{cases} x_1^{(k+1)} = \frac{1}{8}(x_2^{(k)} - 2x_3^{(k)} + 56) \\ x_2^{(k+1)} = -\frac{1}{4}(-x_1^{(k+1)} - x_3^{(k)} - 1) \\ x_3^{(k+1)} = -\frac{1}{5}(-x_1^{(k+1)} - 2x_2^{(k+1)} - 37), \end{cases} \tag{4.10}$$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	7.000000	0.250000	7.400000
2	5.181250	3.850000	8.900000
3	5.256250	3.770312	9.976250
4	4.977227	4.058125	9.959375
5	5.017422	3.984150	10.018700
6	4.993345	4.009029	9.997145
7	5.001842	3.997622	10.002280
8	4.999133	4.001031	9.999417
9	5.000275	3.999638	10.000240
⋮	⋮	⋮	⋮

Tabella 4.2: Valori ottenuti nelle prime 9 iterazioni del metodo di Jacobi applicato al sistema (4.2)

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	7.000000	2.000000	9.600000
2	4.850000	3.862500	9.915000
3	5.004063	3.979766	9.992719
4	4.999291	3.998003	9.999060
5	4.999986	3.999761	9.999902
6	4.999995	3.999974	9.999989
⋮	⋮	⋮	⋮

Tabella 4.3: Valori ottenuti nelle prime 6 iterazioni del metodo di G-S applicato al sistema (4.2)

noto con il nome di **metodo di Gauss-Seidel (G-S)**.
Nella Tabella 4.3 sono riportati i valori ottenuti nelle prime 6 iterazioni del metodo di G-S, scegliendo come valori iniziali $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$.

Si osserva che i valori ottenuti con il metodo di G-S si avvicinano alla soluzione più rapidamente, cioè con un minore numero di iterazioni, di quelli ottenuti con il metodo di Jacobi. Tale fenomeno si verifica spesso, ma non sempre: infatti, per alcuni sistemi di equazioni lineari, il metodo di Jacobi genera valori che si avvicinano alla soluzione del sistema, mentre ciò non accade per il metodo di G-S. ♣

Riprendiamo il sistema (4.9). Si osserva che quando $x_1^{(k+1)}$ è stato calcolato, è possibile utilizzare tale valore al posto di $x_1^{(k)}$, per determinare il valore $x_2^{(k+1)}$. Analogamente,

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	1.000000	-1.000000	-3.000000
2	5.000000	-3.000000	-3.000000
3	1.000000	1.000000	1.000000
4	1.000000	1.000000	1.000000
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabella 4.4: Valori ottenuti nelle prime 4 iterazioni del metodo di Jacobi applicato al sistema (4.12)

per calcolare $x_n^{(k+1)}$ è possibile utilizzare i valori di $x_1^{(k+1)}$, $x_2^{(k+1)}$, ..., $x_{n-1}^{(k+1)}$ già calcolati alla $(k+1)$-ma iterazione. Quanto detto conduce al seguente metodo iterativo:

$$\begin{cases} x_1^{(k+1)} = -\frac{1}{a_{11}}(a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + \ldots - b_1) \\ x_2^{(k+1)} = -\frac{1}{a_{22}}(a_{21}x_1^{(k+1)} + a_{23}x_3^{(k)} + \ldots - b_2) \\ \ldots \quad\quad \ldots \quad\quad \ldots \\ x_n^{(k+1)} = -\frac{1}{a_{nn}}(a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + \ldots - b_n), \end{cases} \tag{4.11}$$

noto con il nome di **metodo di Gauss-Seidel (G-S)**.

♣ **Esempio 4.5.** Si consideri il sistema lineare:

$$\begin{cases} x_1 - 2x_2 + 2x_3 = 1 \\ -x_1 + x_2 - x_3 = -1 \\ -2x_1 - 2x_2 + x_3 = -3, \end{cases} \tag{4.12}$$

la cui soluzione è $x_i = 1$, $i = 1, 2, 3$.
Nelle Tabelle 4.4 e 4.5 sono riportati alcuni valori ottenuti risolvendo tale sistema con i metodi di Jacobi e di G-S. Si nota che i valori relativi al metodo di Jacobi si avvicinano alla soluzione (nel caso particolare i valori sono proprio uguali alla soluzione) già dopo 3 iterazioni, mentre i valori generati dal metodo di G-S si allontanano sempre di più ad ogni iterazione.

♣

Si consideri ora un sistema lineare di n equazioni in n incognite:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \quad\quad \ldots \quad\quad \ldots \quad\quad \ldots = \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n, \end{cases}$$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	1.000000	0.000000	-1.000000
2	3.000000	1.000000	5.000000
3	-7.000000	-3.000000	-23.00000
4	41.00000	17.00000	113.0000
5	-191.0000	-79.00000	-543.0000
6	929.0000	385.0000	2625.000
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabella 4.5: Valori ottenuti nelle prime 6 iterazioni del metodo di G-S applicato al sistema (4.12)

che può essere scritto anche nelle due forme equivalenti:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, \cdots, n, \tag{4.13}$$

$$Ax = b,$$

con $A = [a_{ij}]$[6] matrice di dimensione $n \times n$, $b = [b_i]$ e $x = [x_i]$ vettori di dimensione n. In relazione a tale sistema, le (4.5) si generalizzano come segue:

METODO DI JACOBI

$$x_i^{(k+1)} = \left(b_i - a_{i1}x_1^{(k)} - \cdots - a_{i,i-1}x_{i-1}^{(k)} - a_{i,i+1}x_{i+1}^{(k)} - \cdots - a_{in}x_n^{(k)}\right)/a_{ii} = \\ = \left(b_i - \sum_{j=1,j\neq i}^{n} a_{ij}x_j^{(k)}\right)/a_{ii}, \quad i = 1, \cdots, n; \quad k \geq 0. \tag{4.14}$$

Ovviamente, affinchè le (4.14) siano definite è necessario che gli elementi diagonali di A, a_{ii}, $i = 1, \cdots, n$, siano diversi da zero[7].
In relazione al sistema (4.13), il metodo di G-S è definito da:

[6] Si assume qui e nel seguito che la matrice dei coefficienti $A \in \mathbb{R}^{n\times n}$, del sistema (4.13) sia non singolare, cioè con determinante diverso da zero, e, quindi, che il sistema (4.13) ammetta una ed una sola soluzione.

[7] Se il sistema (4.13) è non singolare, ciò può essere sempre ottenuto scambiando in modo opportuno le equazioni.

METODO DI GAUSS-SEIDEL

$$x_i^{(k+1)} = \left(b_i - a_{i1}x_1^{(k+1)} - \cdots - a_{i,i-1}x_{i-1}^{(k+1)} - a_{i,i+1}x_{i+1}^{(k)} - \cdots - a_{in}x_n^{(k)}\right)/a_{ii} =$$
$$= \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)/a_{ii}, \quad i = 1, \cdots, n; \quad k \geq 0. \tag{4.15}$$

Come per il metodo di Jacobi, si assume che gli elementi diagonali di A siano diversi da zero.

4.2.1 Primi esempi di algoritmi

Un primo esempio di algoritmo per la risoluzione di un sistema di equazioni lineari con il metodo iterativo di Jacobi è illustrato nella **Procedura 4.1**. Essa è scritta nel linguaggio *pascal-like* e prevede come dati di input la dimensione del sistema `n`, la matrice dei coefficienti del sistema, rappresentata dall'array `A`, il vettore dei termini noti, rappresentato dall'array `b`, i valori iniziali contenuti nell'array `xold` ed il numero di iterazioni da effettuare, `MaxIt`. I soli dati di output sono i valori ottenuti dopo l'esecuzione di `MaxIt` iterazioni del metodo di Jacobi, immagazzinati nell'array `x`. Si osservi che, nella procedura in esame, l'array `xold` contiene i valori calcolati all'iterazione precedente, mentre l'array `x` contiene i nuovi valori da calcolare.
Una prima versione dell'algoritmo basato sul metodo di G-S è invece illustrato nella **Procedura 4.2**. Si osservi che in tale procedura, a differenza di quella relativa al metodo di Jacobi, non è necessario utilizzare l'array `xold`, poiché, per calcolare i nuovi valori `x(i)` alla generica iterazione, si utilizzano i valori `x(j)`, `j` $<$ `i`, già calcolati in quella stessa iterazione.

procedure Jacobi1(**in:** $n, A, b, xold, MaxIt$; **out:** x)

/# SCOPO: risoluzione di Ax=b con il metodo di Jacobi

/# SPECIFICHE PARAMETRI:

/# PARAMETRI DI INPUT:

var: n	: **intero**	{*dimensione del sistema*}
var: $A(n,n)$	: **array di reali**	{*coefficienti del sistema*}
var: $b(n)$	: **array di reali**	{*termini noti del sistema*}
var: $xold(n)$	: **array di reali**	{*valori iniziali*}
var: $MaxIt$	: **intero**	{*numero di iterazioni da effettuare*}

/# PARAMETRI DI OUTPUT:

var: $x(n)$	: **array di reali**	{*soluzione calcolata*}

/# VARIABILI LOCALI:

var: i, j, k	: **interi**	{*contatori*}

Procedura 4.1: Algoritmo di Jacobi (prima versione) - continua

```
/# INIZIO ISTRUZIONI:
    for k = 1, MaxIt do                          {ciclo sul numero di iterazioni}
      for i = 1, n do                            {ciclo per il calcolo di x^(k)}
        x(i) := b(i);
        for j = 1, n do                          {ciclo per il calcolo di x_i^(k)}
          if i ≠ j then
            x(i) := x(i) − A(i, j) ∗ xold(j);
          endif
        endfor
        x(i) := x(i)/A(i, i);
      endfor
      for i = 1, n do                            {aggiornamento di xold}
        xold(i) := x(i);
      endfor
    endfor
    return x
  end Jacobi1
```

Procedura 4.1: Algoritmo di Jacobi (prima versione) - fine

procedure G-S1(**in:** $n, A, b, x, MaxIt$; **out:** x)

/# SCOPO: risoluzione di Ax=b con il metodo di Gauss-Seidel

/# SPECIFICHE PARAMETRI:

/# PARAMETRI DI INPUT:

var: n	: **intero**	{*dimensione del sistema*}
var: $A(n,n)$	: **array di reali**	{*coefficienti del sistema*}
var: $b(n)$	: **array di reali**	{*termini noti del sistema*}
var: $x(n)$	: **array di reali**	{*valori iniziali*}
var: $MaxIt$	: **intero**	{*numero di iterazioni da effettuare*}

/# PARAMETRI DI OUTPUT:

var: $x(n)$	: **array di reali**	{*soluzione calcolata*}

/# VARIABILI LOCALI:

var: i, j, k	: **interi**	{*contatori*}

Procedura 4.2: Algoritmo di G-S (prima versione) - continua

```
/# INIZIO ISTRUZIONI:
    for k = 1, MaxIt do                          {ciclo sul numero di iterazioni}
      for i = 1, n do                              {ciclo per il calcolo di x^(k)}
        x(i) := b(i);
        for j = 1, n do                            {ciclo per il calcolo di x_i^(k)}
          if i ≠ j then
            x(i) := x(i) − A(i, j) ∗ x(j);
          endif
        endfor
        x(i) := x(i)/A(i, i);
      endfor
    endfor
    return x
  end G-S1
```

Procedura 4.2: Algoritmo di G-S (prima versione) - fine

4.2.2 Complessità computazionale

Si analizza ora il numero di operazioni floating-point richieste dal metodo di Jacobi. Considerando la generica iterazione del metodo, si osserva che il calcolo di ciascuna componente di $x^{(k+1)}$ richiede $n-1$ moltiplicazioni, $n-2$ addizioni, 1 sottrazione e 1 divisione. Contando una divisione come una moltiplicazione ed una sottrazione come un'addizione, il numero totale di operazioni richieste da un'iterazione del metodo di Jacobi è:

$$n^2\mathcal{M} + n(n-1)\mathcal{A}, \tag{4.16}$$

dove $\mathcal{A}$ denota un'addizione e $\mathcal{M}$ denota una moltiplicazione e, dunque, una stima del numero di flops, T_{jac}, richieste da una iterazione è

$$T_{jac} \simeq n^2. \tag{4.17}$$

Dalla (4.17) si rileva che il costo di ciascuna iterazione del metodo è equivalente a quello del calcolo del prodotto tra una matrice ed un vettore.
Nel derivare la stima (4.17) si è supposto che, per ciascuna componente di $x^{(k+1)}$, tutte le operazioni in (4.14) siano effettivamente eseguite. Se invece il sistema lineare da risolvere ha dei coefficienti a_{ij} nulli, le operazioni nelle quali si utilizzano tali coefficienti possono non essere eseguite e, quindi, non devono essere considerate nel calcolo del numero di flops richieste dal metodo di Jacobi. Per chiarire questa considerazione si assuma che

il sistema abbia solo p coefficienti non nulli, con $p < n^2$. Di conseguenza, il calcolo di $x^{(k+1)}$ richiede un numero di flops circa uguale a p. In tal caso, la (4.17) si generalizza nella:

$$T_{jac} \simeq p = n^2(1 - (1 - p/n^2)) = n^2(1 - SP), \tag{4.18}$$

dove SP è il grado di sparsità del sistema. Si osservi che, quando $SP = 0$ (cioè, tutti i coefficienti sono non nulli), dalla (4.18) si ottiene la (4.17). Inoltre, la (4.18) mostra che se il sistema ha un grado di sparsità elevato (che significa $p << n^2$ e, quindi, SP molto vicino a 1), il costo di un'iterazione del metodo di Jacobi si riduce notevolmente rispetto al costo per un sistema non sparso. Come già detto ciò rappresenta uno dei vantaggi dell'utilizzo dei metodi iterativi nella risoluzione di sistemi di dimensioni elevate. Su tale aspetto si tornerà nei prossimi paragrafi.
Dalla (4.15), si verifica che tutte le considerazioni appena fatte sulla complessità computazionale del metodo di Jacobi si possono applicare anche al metodo di G-S giungendo alle stesse conclusioni, e quindi:

$$T_{gs} \simeq n^2(1 - SP).$$

4.2.3 Interpretazione geometrica

Si consideri il sistema lineare di due equazioni:

$$\begin{cases} 2x_1 - x_2 = 0 \\ x_1 - 3x_2 = 0, \end{cases} \tag{4.19}$$

ciascuna della quali, in un sistema di riferimento di assi cartesiani in cui x_1 è l'asse delle ascisse e x_2 è l'asse delle ordinate, rappresenta una retta del piano; il loro punto di intersezione, l'origine, è la soluzione del sistema (vedi Figura 4.2). Il metodo di Jacobi applicato a tale sistema ha la forma:

$$\begin{cases} x_1^{(k+1)} = \frac{1}{2}x_2^{(k)} \\ x_2^{(k+1)} = \frac{1}{3}x_1^{(k)}. \end{cases}$$

Quindi, se $x^{(0)}$ è il punto iniziale, la prima equazione è risolta rispetto alla variabile x_1 con la variabile x_2 fissata. Dalla Figura 4.2 si evince che geometricamente ciò significa determinare sulla retta definita da tale equazione il punto P che ha la stessa ordinata di $x^{(0)}$. Procedendo allo stesso modo per la seconda equazione, si determina sulla retta ad essa relativa il punto Q che ha la stessa ascissa di $x^{(0)}$. Il nuovo punto $x^{(1)}$ si ottiene geometricamente come estremo del vettore somma dei vettori definiti dai punti

$$(x^{(0)}, P) \quad \text{e} \quad (x^{(0)}, Q)$$

così determinati, puntati in $x^{(0)}$. Nelle iterazioni successive si procede in maniera analoga e si osserva che ci si avvicina sempre di più alla soluzione.

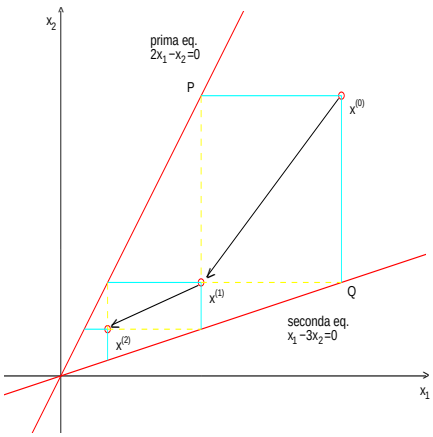


Figura 4.2: Interpretazione geometrica del metodo di Jacobi applicato al sistema (4.19).

Infine, in Figura 4.3 è illustrata l'interpretazione geometrica del metodo di G-S applicato al sistema (4.19):

$$\begin{cases} x_1^{(k+1)} &= \frac{1}{2}x_2^{(k)} \\ x_2^{(k+1)} &= \frac{1}{3}x_1^{(k+1)} \end{cases}.$$

A partire da $x^{(0)}$, il primo passo è identico a quello di Jacobi. Il punto $x^{(1)}$, invece, si ottiene a partire dal punto determinato sulla prima retta e considerando il punto sulla seconda retta che ha la stessa ascissa. Confrontando le Figure 4.2 e 4.3 si può osservare graficamente che per il sistema in esame il metodo di G-S si avvicina alla soluzione più velocemente del metodo di Jacobi.

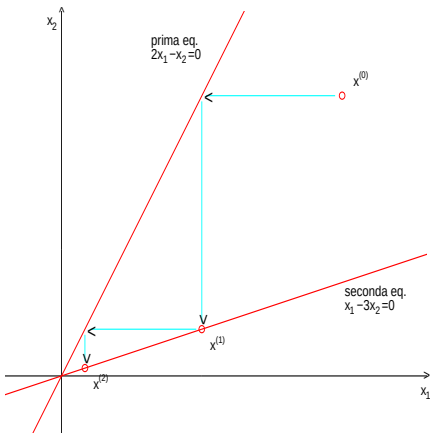


Figura 4.3: Interpretazione geometrica del metodo di G-S applicato al sistema (4.19).

4.3 Convergenza

Si è visto che gli algoritmi relativi ai metodi di Jacobi e di G-S generano n successioni di valori $\{x_i^{(k)}\}$, $i = 1, \cdots, n$, mediante le equazioni (4.14) o (4.15). Quindi, il primo problema relativo all'utilizzo di tali algoritmi è la determinazione di condizioni che assicurino che tali valori, al crescere di n, si avvicinino sempre di più alla soluzione del sistema. In altre parole, è necessario analizzare il problema della **convergenza** dei metodi di Jacobi e G-S, da cui questi algoritmi derivano.

Indicata con $x^* = [x_i^*]$ la soluzione del sistema (4.13), si vogliono determinare condizioni opportune per le quali si abbia:

$$\lim_{k\to\infty} x_i^{(k)} = x_i^*, \quad i = 1, \cdots, n.$$

♣ **Esempio 4.6.** Si consideri il sistema lineare:

$$\begin{cases} x_1 & +2x_2 & & = 1 \\ x_1 & + x_2 & +x_3 & = 0 \\ & 2x_2 & +x_3 & = 1, \end{cases} \tag{4.20}$$

la cui soluzione è $x_1^* = -1/3$, $x_2^* = 2/3$, $x_3^* = -1/3$. Risolvendo tale sistema con i metodi di Jacobi e di G-S, partendo dai valori iniziali $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$, si hanno i risultati riportati nelle Tabelle 4.6 e 4.7: entrambi i metodi generano valori che non si avvicinano alla soluzione.

♣

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	1.000000	0.000000	1.000000
2	1.000000	-2.000000	1.000000
3	5.000000	-2.000000	5.000000
4	5.000000	-10.00000	5.000000
5	21.00000	-10.00000	21.00000
6	21.00000	-42.00000	21.00000
⋮	⋮	⋮	⋮
15	21845.00	-10922.00	21845.00
⋮	⋮	⋮	⋮

Tabella 4.6: Valori ottenuti nelle prime 15 iterazioni del metodo di Jacobi applicato al sistema (4.20).

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	1.000000	-1.000000	3.000000
2	3.000000	-6.000000	13.00000
3	13.00000	-26.00000	53.00000
4	53.00000	-106.0000	213.0000
5	213.0000	-426.0000	853.0000
6	853.0000	-1706.000	3413.000
⋮	⋮	⋮	⋮
15	$2.2 \cdot 10^8$	$-4. \cdot 10^8$	$8.9 \cdot 10^8$
⋮	⋮	⋮	⋮

Tabella 4.7: Valori ottenuti nelle prime 15 iterazioni del metodo di G-S applicato al sistema (4.20).

♣ **Esempio 4.7.** Si consideri il sistema lineare:

$$\begin{cases} x_1 & -3x_2 & = 0 \\ 2x_1 & - \ x_2 & = 0, \end{cases} \tag{4.21}$$

ottenuto dal sistema (4.19) semplicemente scambiando le equazioni e la cui soluzione è $x_1^* = 0,\ x_2^* = 0$. Per tale sistema, partendo dai valori iniziali $x_1^{(0)} = x_2^{(0)} = 0.5$, sia il metodo di Jacobi sia il metodo di G-S non convergono. Dopo 10 iterazioni si ha infatti:

$$\begin{aligned} &(Jacobi) \qquad x_1 = 3888;\ \ x_2 = 3888, \\ &(G-S) \qquad x_1 = 1.51165 \times 10^7;\ \ x_2 = 3.02331 \times 10^7. \end{aligned}$$

Tale fenomeno si evidenzia anche nelle Figure 4.4 e 4.5, dove è illustrata l'interpretazione geometrica dei metodi di Jacobi e G-S applicati al sistema in esame. Ricordando che per il sistema (4.19) entrambi i metodi convergono, si deduce che l'ordine delle equazioni può influire sulla convergenza dei metodi in esame (tale questione verrà ripresa nei prossimi paragrafi).

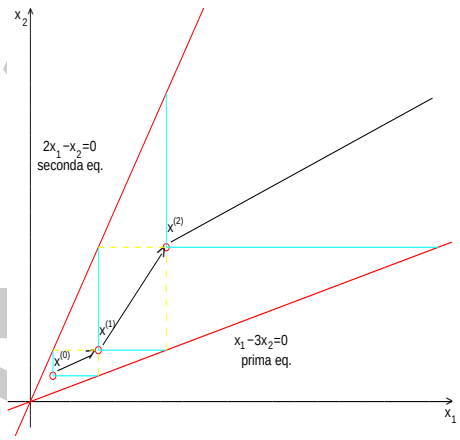


Figura 4.4: Interpretazione geometrica del metodo di Jacobi applicato al sistema (4.21).

♣

Allo scopo di illustrare le idee di base relative allo studio della convergenza dei metodi

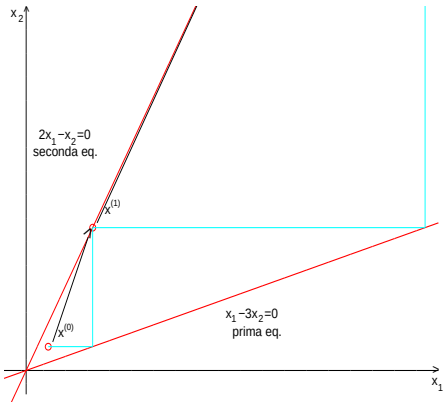


Figura 4.5: Interpretazione geometrica del metodo di G-S applicato al sistema (4.21).

di Jacobi e di G-S si osservi che entrambi i metodi possono essere espressi nella forma:

$$x_i^{(k+1)} = c_{i1}x_1^{(k)} + \cdots + c_{in}x_n^{(k)} + d_i \; =$$

$$= \sum_{j=1}^{n} c_{ij}x_j^{(k)} + d_i, \quad i = 1, \cdots, n, \quad k \geq 0, \tag{4.22}$$

o equivalentemente:

$$x^{(k+1)} = Cx^{(k)} + d, \quad k \geq 0,$$

dove $C = [c_{ij}]$ è una matrice $n \times n$ e $d = [d_i]$ è un vettore di dimensione n.
È immediato verificare, dalla (4.14), che per il metodo di Jacobi si ha:

$$c_{ij} = \begin{cases} 0 & i = j \\ -a_{ij}/a_{ii} & i \neq j \end{cases}, \quad d_i = b_i/a_{ii}. \tag{4.23}$$

Per il metodo di G-S scritto in forma (4.22) [8] si ha:

$$c_{ij} = \begin{cases} -(\sum_{l=1}^{i-1} a_{il}c_{lj})/a_{ii}, & j = 1, 2, \cdots, i \\ -(\sum_{l=1}^{i-1} a_{il}c_{lj} + a_{ij})/a_{ii}, & j = i+1, \cdots, n \end{cases} \tag{4.24}$$

$$d_i = (b_i - \textstyle\sum_{l=1}^{i-1} a_{il}d_l)/a_{ii}, \quad i = 1, \cdots, n.$$

[8] Non è altrettanto immediato scrivere il metodo di G-S nella forma (4.22). Tuttavia ciò può essere fatto con un procedimento per induzione. Si osservi innanzitutto che la (4.15) per $i = 1$ è già in forma (4.22) con $c_{11} = 0$ e $c_{1j} = -a_{1j}/a_{11}$, $j = 2, ..., n$ e $d_1 = b_1/a_{11}$. Si assuma che $x_2^{(k+1)}, ..., x_{i-1}^{(k+1)}$ dati dalla (4.15) abbiano la forma (4.22). Si dimostra ora che anche $x_i^{(k+1)}$ può essere scritto in forma (4.22).

Definizione 4.3.1. (Metodo convergente)
Si dice che un metodo del tipo (4.22) è **convergente** *se, qualunque siano i valori iniziali, le successioni* $\{x_i^{(k)}\}$, $i = 1, ...n$, *convergono, cioè se*

$$\lim_{k\to\infty} x_i^{(k)} = \bar{x}_i, \quad i = 1, \cdots, n, \quad \forall x_i^{(0)} \ , \tag{4.25}$$

oppure, in modo completamente equivalente, se:

$$\lim_{k\to\infty} x^{(k)} = \bar{x}, \quad \forall x^{(0)},$$

dove $x^{(k)}$ *e* $\bar{x}$ *sono i vettori di componenti, rispettivamente,* $(x_1^{(k)}, ..., x_n^{(k)})$ *e* $(\bar{x_1}, ..., \bar{x_n})$.

Si osservi che, se il metodo (4.22) converge, passando al limite in entrambi i membri delle (4.22), si ha:

$$\bar{x_i} = \sum_{j=1}^{n} c_{ij}\bar{x_j} + d_i \ , \quad i = 1, \cdots, n \ . \tag{4.26}$$

che, in forma compatta, può scriversi come:

$$\bar{x} = C\bar{x} + d \ , \ \ o \ \ equivalentemente \ \ [(I - C)\bar{x} = d]$$

Se ora si considera il metodo di Jacobi e si assume che sia convergente, indicato con $\bar{x}$ il limite, dalle (4.23) si ha:

$$\bar{x_i} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}\bar{x_j}\right) = \bar{x_i} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{n} a_{ij}\bar{x_j}\right), \quad i = 1, \cdots, n,$$

Infatti, considerando l'espressione di $x_i^{(k+1)}$ in (4.15), si ha:

$$\begin{aligned}
x_i^{(k+1)} &= \frac{1}{a_{ii}}\left[b_i - \sum_{l=1}^{i-1} a_{il}x_l^{(k+1)} - \sum_{l=i+1}^{n} a_{il}x_l^{(k)}\right] \\
&= \frac{1}{a_{ii}}\left[b_i - \sum_{l=1}^{i-1} a_{il}\left(\sum_{j=1}^{n} c_{lj}x_j^{(k)} + d_l\right) - \sum_{l=i+1}^{n} a_{il}x_l^{(k)}\right] \\
&= \sum_{j=1}^{i}\left(\sum_{l=1}^{i-1}\frac{-a_{il}c_{lj}}{a_{ii}}\right)x_j^{(k)} + \sum_{j=i+1}^{n}\left(\sum_{l=1}^{i-1}\frac{-a_{il}c_{lj}}{a_{ii}} - \frac{a_{ij}}{a_{ii}}\right)x_j^{(k)} \\
&\quad + \frac{1}{a_{ii}}\left(b_i - \sum_{l=1}^{i-1} a_{il}d_l\right)
\end{aligned}$$

da cui segue la (4.24). Si osservi che per il metodo di G-S non è conveniente utilizzare l'equazione (4.22) con i coefficienti c_{ij} dati dalle (4.24). Le equazioni (4.15) risultano molto più "semplici". La sola ragione per cui si è introdotta la forma (4.22) è quella di facilitare una discussione elementare sulla convergenza dei metodi iterativi in esame.

e, di conseguenza:

$$\sum_{j=1}^{n} a_{ij}\bar{x_j} = b_i, \quad i = 1, \cdots, n,$$

cioè, il limite è soluzione del sistema $Ax = b$ e, quindi, $\bar{x}_i = x_i^*$, $i = 1, ..., n$.
Si può perciò affermare:

Proprietà 4.3.1. *Se il metodo di Jacobi converge, allora il limite è la soluzione del sistema (4.13).*

Con un ragionamento analogo, si può verificare che anche il metodo di G-S gode della proprietà suddetta, che è detta **consistenza** del metodo con il sistema (4.13). È evidente che tale proprietà è essenziale per un qualsiasi metodo iterativo del tipo (4.22).

Definizione 4.3.2. (Metodo consistente)
Un metodo del tipo (4.22) è **consistente**[9] *con il sistema (4.13) se, quando converge, il limite è la soluzione del sistema (4.13).*

Ad ogni iterazione di un metodo iterativo del tipo (4.22), il vettore $x^{(k)}$ rappresenta un'approssimazione della soluzione del sistema (4.13).

Definizione 4.3.3. (Errore di troncamento analitico)
Sia $\{x^k\}$ la successione generata dal metodo iterativo:

$$x_i^{(k+1)} = \sum_{j=1}^{n} c_{ij}x_j^{(k)} + d_i, \quad i = 1, \cdots, n, \quad k \geq 0.$$

Si definisce **errore di troncamento analitico alla k-ma iterazione** *del metodo, il vettore:*

$$e^{(k)} = x^{(k)} - x^*. \tag{4.27}$$

In base alla definizione precedente è evidente che la convergenza di un metodo del tipo (4.22) alla soluzione del sistema (4.13) è equivalente alla convergenza della successione degli errori a zero, cioè alla condizione:

$$\lim_{k\to\infty} |e_i^{(k)}| = 0, \quad i = 1, \cdots, n, \tag{4.28}$$

dove $e_i^{(k)} = (x_i^{(k)} - x_i^*)$ è l'i-ma componente di $e^{(k)}$.

[9] Nei prossimi paragrafi il concetto di consistenza verrà ripreso con maggiore approfondimento.

♣ **Esempio 4.8.** Si consideri il sistema

$$\begin{cases} 8x_1 & - & x_2 & +2x_3 & = & 56 \\ x_1 & & -4x_2 & + & x_3 & = - & 1 \\ x_1 & & +2x_2 & & -5x_3 & = -37. \end{cases} \tag{4.29}$$

Nella Tabella che segue sono riportati i valori assoluti delle componenti dell'errore relativo alle prime 9 iterazioni (vedi Tabella 4.2) risolvendo il sistema con il metodo di Jacobi. Si osserva che tali quantità si avvicinano a zero al crescere del numero di iterazioni, come ci si aspetta sempre quando il metodo è convergente alla soluzione.

k	$\lvert e_1^{(k)}\rvert$	$\lvert e_2^{(k)}\rvert$	$\lvert e_3^{(k)}\rvert$
1	2.000000	3.750000	2.600000
2	0.181250	0.150000	1.100000
3	0.256250	0.229688	0.023750
4	0.022773	0.058125	0.040625
5	0.017422	0.015850	0.018700
6	0.006655	0.009029	0.002855
7	0.001842	0.002378	0.002280
8	0.000867	0.001031	0.000583
9	0.000275	0.000362	0.000239

♣

♣ **Esempio 4.9.** Si consideri il sistema (4.29). La matrice C relativa al metodo di Jacobi espresso nella forma (4.22) applicato a tale sistema è:

$$C = \begin{bmatrix} 0 & 0.125 & -0.25 \\ 0.25 & 0 & 0.25 \\ 0.2 & 0.4 & 0 \end{bmatrix}.$$

Si ha quindi $\|C\| = 0.6$. ♣

Una semplice condizione sufficiente per la convergenza dei metodi iterativi della forma (4.22) è data dal seguente Teorema:

Teorema 4.3.1. *Il metodo iterativo:*

$$x_i^{(k+1)} = \sum_{j=1}^{n} c_{ij} x_j^{(k)} + d_i, \quad i = 1, \cdots, n, \;\; k \geq 0,$$

è convergente se $\|C\| < 1$.

Dimostrazione Se $\|C\| < 1$, la matrice $I-C$ è non singolare (vedi **Teoremi B.5** e **B.12** dell'**Appendice B** della **Parte 1**) e, quindi, il sistema lineare $x = Cx + d$ ha un'unica soluzione, $\bar{x}$. Posto:

$$e^{(k)} = x^{(k)} - \bar{x}, \ \ \forall k \geq 0,$$

dalle (4.22) e dalle (4.26) si ha:

$$e_i^{(k+1)} = x_i^{(k+1)} - \bar{x}_i = \sum_{j=1}^{n} c_{ij}(x_j^{(k)} - \bar{x}_j) = \sum_{j=1}^{n} c_{ij} e_j^{(k)} \ , \quad i = 1, \cdots, n \ \ , \tag{4.30}$$

da cui:

$$|e_i^{(k+1)}| \leq \sum_{j=1}^{n} |c_{ij}||e_j^{(k)}| \ , \quad i = 1, \cdots, n. \tag{4.31}$$

Utilizzando le norma infinito per vettori e matrici, dalla (4.31) si ha:

$$|e_i^{(k+1)}| \leq \sum_{j=1}^{n} |c_{ij}|\|e^{(k)}\| \ , \quad i = 1, \cdots, n \ , \tag{4.32}$$

da cui si ottiene:

$$|e_i^{(k+1)}| \leq \|C\|\|e^{(k)}\| \ , \quad i = 1, \cdots, n \ \ ,$$

e quindi:

$$\|e^{(k+1)}\| \leq \|C\|\|e^{(k)}\| \ .$$

Allo stesso modo si può dimostrare che:

$$\|e^{(k)}\| \leq \|C\|\|e^{(k-1)}\| \ ,$$

da cui segue:

$$\|e^{(k+1)}\| \leq \|C\|\|e^{(k)}\| \leq \|C\|^2\|e^{(k-1)}\| \ .$$

Ripetendo altre $k-1$ volte tale procedimento, si ottiene:

$$\|e^{(k+1)}\| \leq \|C\|^{k+1}\|e^{(0)}\| \ . \tag{4.33}$$

Se $\|C\| < 1$, il secondo membro della (4.33) converge a zero per $k \to \infty$, qualunque sia $e^{(0)}$. Si ha quindi:

$$\lim_{k\to\infty} \|e^{(k+1)}\| = 0, \ \ \forall e^{(0)},$$

che equivale a:

$$\lim_{k\to\infty} |e_i^{(k+1)}| = 0, \ \ i = 1, .., n, \ \ \forall e_i^{(0)},$$

da cui:

$$\lim_{k\to\infty} x_i^{(k)} = \bar{x}_i, \ \ i = 1, .., n, \ \ \forall x_i^{(0)},$$

cioè, il metodo è convergente a $\bar{x}$. ■

Osservazione 4.1. *Il Teorema precedente stabilisce una condizione solo sufficiente per la convergenza basata sull'ordine di grandezza dei coefficienti c_{ij} in (4.22). Come già*

detto esistono altre misure della grandezza di tali coefficienti e per tali misure vale il risultato stabilito dal Teorema 4.3.1[10].

Si consideri il metodo di Jacobi espresso nella forma (4.22), dove i coefficienti c_{ij} e d_i sono dati dalle (4.23). Poiché:

$$\|C\| = \max_{1\le i\le n}\left(\sum_{j=1}^{n}|c_{ij}|\right) = \max_{1\le i\le n}\left(\sum_{j=1,j\neq i}^{n}|a_{ij}|/|a_{ii}|\right).$$

Quindi, se:

$$\sum_{j=1,j\neq i}^{n}|a_{ij}|/|a_{ii}| < 1, \quad i = 1,\cdots,n,$$

che è equivalente alla condizione:

$$\sum_{j=1,j\neq i}^{n}|a_{ij}| < |a_{ii}|, \quad i = 1,\cdots,n, \tag{4.35}$$

si ha $\|C\| < 1$. Un sistema di equazioni lineari in cui i coefficienti a_{ij} soddisfano le (4.35) è detto a **diagonale strettamente dominante**.
Si può quindi enunciare il seguente risultato:

Corollario 4.3.1. *Se il sistema lineare (4.13) è a* **diagonale strettamente dominante***, il metodo di Jacobi è convergente.*

Poichè i risultati dimostrati in questo paragrafo forniscono condizioni solo sufficienti per la convergenza, un metodo iterativo del tipo (4.22) può convergere anche se il sistema non è a diagonale strettamente dominante o, in generale, non soddisfa l'ipotesi del Teorema 4.3.1.

[10] Ad esempio, se si considera la **norma del valore assoluto** (o **norma 1**):

$$\|x\|_1 = |x_1| + \cdots + |x_n| = \sum_{i=1}^{n}|x_i|,$$

$$\|A\|_1 = \max_{1\le j\le n}\left(\sum_{i=1}^{n}|a_{ij}|\right), \tag{4.34}$$

si può verificare che vale ancora la relazione (4.33); il Teorema 4.3.1 è quindi vero anche se si considera la norma suddetta.

♣ **Esempio 4.10.** Considerato il sistema lineare:

$$\begin{cases} x_1 & -2x_2 & +2x_3 & = 1 \\ 2x_1 & + x_2 & +2x_3 & = 1 \\ x_1 & + x_2 & + x_3 & = 1, \end{cases}$$

la cui soluzione è $x_1^* = -3$, $x_2^* = 1$, $x_3^* = 3$, è immediato verificare che esso non è a diagonale strettamente dominante. Inoltre, considerato il metodo di Jacobi applicato a tale sistema:

$$\begin{cases} x_1^{(k+1)} = +2x_2^{(k)} - 2x_3^{(k)} + 1 \\ x_2^{(k+1)} = -2x_1^{(k)} - 2x_3^{(k)} + 1 \\ x_3^{(k+1)} = - x_1^{(k)} - x_2^{(k)} + 1 \, , \end{cases}$$

si verifica che i coefficienti c_{ij} del metodo non soddisfano l'ipotesi del Teorema 4.3.1. Tuttavia, il metodo di Jacobi converge alla soluzione del sistema, come mostrano i valori riportati nella seguente Tabella.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	1.000000	1.000000	1.000000
1	1.000000	-3.000000	-1.000000
2	-3.000000	1.000000	3.000000
3	-3.000000	1.000000	3.000000
⋮	⋮	⋮	⋮

♣

4.4 Un semplice criterio di arresto

Un aspetto fondamentale legato all'utilizzo efficace dei metodi iterativi in esame è la scelta di un opportuno criterio per decidere quando arrestare il processo iterativo. In generale, un criterio di arresto soddisfacente deve essere tale che il suo utilizzo conduca ad un risultato sufficientemente accurato, o meglio, consenta di ottenere l'accuratezza desiderata. Appare evidente, quindi, che il criterio più naturale è richiedere che la "distanza" tra $x^{(k)}$ e la soluzione del sistema, e cioè l'errore di troncamento analitico, sia abbastanza piccola. Se si utilizza la norma del massimo per misurare la grandezza dell'errore, una possibile condizione per l'arresto del metodo iterativo è richiedere che l'errore assoluto, $\|x^{(k+1)} - x^*\|$, sia minore di una tolleranza `Tol`:

$$\|x^{(k+1)} - x^*\| \leq \texttt{Tol}, \tag{4.36}$$

che rappresenta l'accuratezza richiesta.

♣ **Esempio 4.11.** Si consideri il sistema lineare:

$$\begin{cases} 4x_1 & + x_2 & - x_3 & = -64 \\ 3x_1 & +5x_2 & + x_3 & = 721 \\ x_1 & -4x_2 & +6x_3 & = 807, \end{cases}$$

la cui soluzione è $x_1^* = 9$, $x_2^* = 99$, $x_3^* = 199$. Nella Tabella che segue sono riportati i valori ottenuti risolvendo tale sistema con il metodo di Jacobi, partendo dai valori iniziali $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$. Inoltre, per ciascuna iterazione, è indicato l'errore assoluto. Si osserva che se si utilizzasse come criterio di arresto la condizione (4.36) con `Tol`$= 10^{-3}$, il metodo si fermerebbe dopo 17 iterazioni e si otterrebbe un'approssimazione della soluzione corretta a 3 cifre decimali. Analogamente, con `Tol`$= 10^{-4}$ si otterrebbe, dopo 21 iterazioni, un'accuratezza fino a 4 cifre decimali.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\|x^{(k)} - x^*\|$
⋮	⋮	⋮	⋮	⋮
10	9.05555	98.91611	199.04701	$.83887\times10^{-1}$
11	9.03272	98.95727	198.93482	$.65183\times10^{-1}$
12	8.99439	98.99340	198.96606	$.33941\times10^{-1}$
13	8.99316	99.01016	198.99654	$.10156\times10^{-1}$
14	8.99660	99.00479	199.00791	$.79100\times10^{-2}$
15	9.00078	99.00046	199.00376	$.37636\times10^{-2}$
16	9.00083	98.99878	199.00018	$.12201\times10^{-2}$
17	9.00035	98.99947	198.99905	$.95100\times10^{-3}$
18	8.99989	98.99998	198.99959	$.41216\times10^{-3}$
19	8.99990	99.00015	199.00000	$.14545\times10^{-3}$
20	8.99996	99.00006	199.00011	$.11333\times10^{-3}$
21	9.00001	99.00000	199.00004	$.44536\times10^{-4}$
22	9.00001	98.99998	199.00000	$.17207\times10^{-4}$
23	9.00000	98.99999	198.99999	$.13391\times10^{-4}$
⋮	⋮	⋮	⋮	⋮

♣

Come mostra l'esempio 4.11, se nella relazione (4.36) si pone `Tol`$= 10^{-m}$, si richiede che il metodo iterativo si arresti quando è stata ottenuta un'approssimazione della soluzione corretta a m cifre decimali. Tuttavia, l'utilizzo della (4.36) presuppone, come criterio di arresto, la conoscenza della soluzione del sistema, che in generale non è nota a priori. È necessario, allora, determinare stime calcolabili dell'errore di troncamento analitico ad ogni iterazione.

Teorema 4.4.1. *Dato il metodo iterativo:*

$$x_i^{(k+1)} = \sum_{j=1}^{n} c_{ij} x_j^{(k)} + d_i, \quad i = 1, \cdots, n, \quad k \geq 0,$$

si assuma che sia consistente con il sistema lineare $Ax = b$. Se $\|C\| < 1$ si ha:

$$\|x^{(k+1)} - x^*\| \leq \frac{\|C\|}{1 - \|C\|}\|x^{(k+1)} - x^{(k)}\|, \quad k \geq 0. \tag{4.37}$$

Dimostrazione Poichè il metodo è consistente, se $\|C\| < 1$ esso converge alla soluzione del sistema, x^*. Valgono quindi le (4.30), con x_i^* al posto di $\bar{x}_i$, che possono essere riscritte nel modo seguente:

$$x_i^{(k+1)} - x_i^* = \sum_{j=1}^{n} c_{ij}(x_j^{(k+1)} - x_j^*) - \sum_{j=1}^{n} c_{ij}(x_j^{(k+1)} - x_j^{(k)}), \quad i = 1, \cdots, n,$$

da cui

$$|x_i^{(k+1)} - x_i^*| \leq \sum_{j=1}^{n} |c_{ij}||x_j^{(k+1)} - x_j^*| + \sum_{j=1}^{n} |c_{ij}||x_j^{(k+1)} - x_j^{(k)}|, \quad i = 1, \cdots, n,$$

e quindi:

$$\|x^{(k+1)} - x^*\| \leq \|C\|\|x^{(k+1)} - x^*\| + \|C\|\|x^{(k+1)} - x^{(k)}\|.$$

Poichè $\|C\| < 1$ per ipotesi, si ottiene:

$$\|x^{(k+1)} - x^*\| \leq \frac{\|C\|}{1 - \|C\|}\|x^{(k+1)} - x^{(k)}\|.$$

■

♣ **Esempio 4.12.** Relativamente alla risoluzione del sistema dell'esempio 4.11, nella Tabella che segue è riportato, per ciascuna iterazione, sia l'errore assoluto, sia la norma del massimo della differenza tra il vettore calcolato in quella iterazione ed il vettore ottenuto nell'iterazione precedente. Si osserva che tale valore costituisce una buona stima dell'errore assoluto.

k	$\|x^{(k)} - x^*\|$	$\|x^{(k)} - x^{(k-1)}\|$
⋮	⋮	⋮
10	$.83887\times10^{-1}$	$.25544\times10^{0}$
11	$.65183\times10^{-1}$	$.11220\times10^{0}$
12	$.33941\times10^{-1}$	$.38338\times10^{-1}$
13	$.10156\times10^{-1}$	$.30478\times10^{-1}$
14	$.79100\times10^{-2}$	$.11373\times10^{-1}$
15	$.37636\times10^{-2}$	$.43332\times10^{-2}$
16	$.12201\times10^{-2}$	$.35861\times10^{-2}$
17	$.95100\times10^{-3}$	$.11285\times10^{-2}$
18	$.41216\times10^{-3}$	$.53885\times10^{-3}$
19	$.14545\times10^{-3}$	$.41671\times10^{-3}$
20	$.11333\times10^{-3}$	$.10878\times10^{-3}$
21	$.44536\times10^{-4}$	$.68793\times10^{-4}$
22	$.17207\times10^{-4}$	$.47863\times10^{-4}$
23	$.13391\times10^{-4}$	$.10962\times10^{-4}$
⋮	⋮	⋮

♣

Il Teorema e l'esempio precedenti mostrano, quindi, che una stima dell'errore di troncamento analitico è data dalla differenza tra i vettori ottenuti in due iterazioni successive del metodo iterativo. Di conseguenza un possibile criterio di arresto "effettivamente utilizzabile", perchè basato su quantità calcolate, è il seguente:

$$\|x^{(k+1)} - x^{(k)}\| \leq \texttt{Tol}. \tag{4.38}$$

Inoltre, in base al risultato stabilito dal Teorema 4.4.1, se si arresta il metodo iterativo quando è verificata la condizione (4.38), per l'errore assoluto di troncamento analitico si ha la maggiorazione:

$$\|x^{(k+1)} - x^*\| \leq \frac{\|C\|}{1 - \|C\|}\texttt{Tol}. \tag{4.39}$$

Se, in particolare, si desidera stimare l'errore relativo di troncamento analitico:

$$\frac{\|x^{(k+1)} - x^*\|}{\|x^*\|},$$

basta osservare che, dalla (4.37), segue:

$$\frac{\|x^{(k+1)} - x^*\|}{\|x^*\|} \leq \frac{\|C\|}{1 - \|C\|}\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^*\|}.$$

Poichè la soluzione x^* non è nota, nella pratica si sostituisce ad essa il valore corrente $x^{(k+1)}$, ottenendo così il seguente criterio di arresto basato sulla distanza relativa tra i valori ottenuti in due iterazioni successive:

$$\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k+1)}\|} \leq \texttt{Tol}. \tag{4.40}$$

♣ **Esempio 4.13.** Si consideri ancora la risoluzione del sistema dell'esempio 4.11. Nella Tabella seguente sono riportati i valori dell'errore relativo e della distanza relativa tra i valori ottenuti in due iterazioni successive del metodo di Jacobi. Si osserva, inanzitutto, che l'errore relativo fornisce informazioni sull'accuratezza in termini di cifre significative. Inoltre la quantità $\|x^{(k)} - x^{(k-1)}\|/\|x^{(k)}\|$ costituisce una stima affidabile dell'errore relativo.

k	$\frac{\|x^{(k)}-x^*\|}{\|x^*\|}$	$\frac{\|x^{(k)}-x^{(k-1)}\|}{\|x^{(k)}\|}$
$\vdots$	$\vdots$	$\vdots$
10	$.42154\times10^{-3}$	$.12833\times10^{-2}$
11	$.32755\times10^{-3}$	$.56398\times10^{-3}$
12	$.17056\times10^{-3}$	$.19269\times10^{-3}$
13	$.51035\times10^{-4}$	$.15316\times10^{-3}$
14	$.39749\times10^{-4}$	$.57151\times10^{-4}$
15	$.18912\times10^{-4}$	$.21774\times10^{-4}$
16	$.61311\times10^{-5}$	$.18021\times10^{-4}$
17	$.47789\times10^{-5}$	$.56707\times10^{-5}$
18	$.20711\times10^{-5}$	$.27078\times10^{-5}$
19	$.73090\times10^{-6}$	$.20940\times10^{-5}$
20	$.56949\times10^{-6}$	$.54663\times10^{-6}$
21	$.22380\times10^{-6}$	$.34569\times10^{-6}$
22	$.86467\times10^{-7}$	$.24052\times10^{-6}$
23	$.67290\times10^{-7}$	$.55086\times10^{-7}$
$\vdots$	$\vdots$	$\vdots$

♣

Il costo computazionale relativo all'utilizzo dei criteri di arresto (4.38) e (4.40) è solo quello del calcolo delle quantità $\|x^{(k+1)} - x^{(k)}\|$ e $\|x^{(k+1)}\|$, in quanto i vettori che si considerano sono già calcolati durante il processo iterativo.

Nella prima versione degli algoritmi relativi ai metodi iterativi di Jacobi e G-S (**Procedure 4.1** e **4.2**) l'unica condizione per l'arresto del processo iterativo è il valore di `MaxIt`, che rappresenta il numero di iterazioni da eseguire. Una seconda versione di tali algoritmi, che include sia un criterio di arresto basato sulla relazione (4.40) sia una condizione di arresto basata su un numero massimo di iterazioni (`MaxIt`), è riportata nelle **Procedure 4.3 e 4.4**. Tali procedure forniscono, in output, anche una stima (`Err`) dell'errore relativo, cioè la quantità:

$$\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k+1)}\|},$$

dove $x^{(k+1)}$ è l'ultimo valore calcolato prima dell'arresto del processo iterativo. Inoltre, altri due parametri di output delle procedure sono: la variabile `Iflag`, che segnala se è stata raggiunta l'accuratezza desiderata e la variabile `It`, che indica il numero di iterazioni che sono state eseguite.

procedure Jacobi2(**in:** $n, A, b, xold, MaxIt, Tol$; **out:** $x, Err, Iflag, It$)

/# SCOPO: risoluzione di un sistema di equazioni lineari
con il metodo iterativo di Jacobi

/# SPECIFICHE PARAMETRI:

/# PARAMETRI DI INPUT:

var: n	: **intero**	{*dimensione del sistema*}
var: $A(n,n)$	: **array di reali**	{*coefficienti del sistema*}
var: $b(n)$	: **array di reali**	{*termini noti del sistema*}
var: $xold(n)$	: **array di reali**	{*valori iniziali*}
var: $MaxIt$	: **intero**	{*numero massimo di iterazioni*}
var: Tol	: **reale**	{*tolleranza usata nel criterio di arresto*}

/# PARAMETRI DI OUTPUT:

var: $x(n)$	: **array di reali**	{*soluzione calcolata*}
var: Err	: **reale**	{*stima dell'errore*}
var: $Iflag$	: **intero**	{*= 0 se la tolleranza non è soddisfatta*}
		{*= 1 se la tolleranza è soddisfatta*}
var: It	: **intero**	{*numero di iterazioni eseguite*}

/# VARIABILI LOCALI:

var: i, j	: **interi**	{*contatori*}
var: $xnorm$	: **reale**	{*norma del vettore all'ultimo passo*}
var: $dnorm$	: **reale**	{*norma della diff. tra i vettori ottenuti*}
		{*in due iterazioni successive*}

Procedura 4.3: Algoritmo di Jacobi (seconda versione) - continua

```
/# INIZIO ISTRUZIONI:
    It := 0;                                   {inizializzazione di It}
    Iflag := 0;                                {inizializzazione di Iflag}
    repeat                                     {ciclo principale con}
      xnorm := 0.;                             {condizioni di arresto}
      dnorm := 0.;
      for i = 1, n do                          {ciclo per il calcolo di x^(k)}
        x(i) := b(i);
        for j = 1, n do                        {ciclo per il calcolo di x_i^(k)}
          if i ≠ j then
            x(i) := x(i) − A(i, j) ∗ xold(j);
          endif
        endfor
        x(i) := x(i)/A(i, i);
        if |x(i)| > xnorm then                 {calcolo di ||x^(k+1)||}
          xnorm := |x(i)|;
        endif
        if |x(i) − xold(i)| > dnorm then       {calcolo di ||x^(k+1) − x^(k)||}
          dnorm := |x(i) − xold(i)|;
        endif
      endfor
      Err := dnorm/xnorm;                      {stima dell'errore}
```

Procedura 4.3: Algoritmo di Jacobi (seconda versione) - continua

```
        if Err ≤ Tol then                    {controllo condizione di arresto}
          Iflag := 1;
        endif
        It := It + 1;                        {aggiornamento del contatore It }
        for i = 1, n do                      {aggiornamento di xold}
          xold(i) := x(i);
        endfor
      until (It = MaxItorIflag = 1)          {verifica condizioni di arresto}
      return x, Err, Iflag, It
end Jacobi2
```

Procedura 4.3: Algoritmo di Jacobi (seconda versione) - fine

procedure G-S2(**in:** $n, A, b, xold, MaxIt, Tol$; **out:** $x, Err, Iflag, It$)

/# SCOPO: risoluzione di un sistema di equazioni lineari
con il metodo iterativo di Gauss-Seidel

/# SPECIFICHE PARAMETRI:

/# PARAMETRI DI INPUT:

var: n	: **intero**	{*dimensione del sistema*}
var: $A(n,n)$	: **array di reali**	{*coefficienti del sistema*}
var: $b(n)$	: **array di reali**	{*termini noti del sistema*}
var: $xold(n)$	: **array di reali**	{*valori iniziali*}
var: $MaxIt$	: **intero**	{*numero massimo di iterazioni*}
var: Tol	: **reale**	{*tolleranza usata nel criterio di arresto*}

/# PARAMETRI DI OUTPUT:

var: $x(n)$	: **array di reali**	{*soluzione calcolata*}
var: Err	: **reale**	{*stima dell'errore*}
var: $Iflag$	: **intero**	{*= 0 se la tolleranza non è soddisfatta*}
		{*= 1 se la tolleranza è soddisfatta*}
var: It	: **intero**	{*numero di iterazioni eseguite*}

/# VARIABILI LOCALI:

var: i, j	: **interi**	{*contatori*}
var: $xnorm$	: **reale**	{*norma del vettore all'ultimo passo*}
var: $dnorm$	: **reale**	{*norma della diff. tra i vettori ottenuti*}
		{*in due iterazioni successive*}

Procedura 4.4: Algoritmo di Gauss-Seidel (seconda versione) - continua

```
/# INIZIO ISTRUZIONI:
    It := 0;                                   {inizializzazione di It}
    Iflag := 0;                                {inizializzazione di Iflag}
    for i = 1, n do                            {inizializzazione di x}
      x(i) := xold(i);
    endfor
    repeat                                     {ciclo principale con}
      xnorm := 0.;                             {condizioni di arresto}
      dnorm := 0.;
      for i = 1, n do                          {ciclo per il calcolo di x^(k)}
        x(i) := b(i);
        for j = 1, n do                        {ciclo per il calcolo di x_i^(k)}
          if i ≠ j then
            x(i) := x(i) − A(i, j) ∗ x(j);
          endif
        endfor
        x(i) := x(i)/A(i, i);
        if |x(i)| > xnorm then                 {calcolo di ||x^(k+1)||}
          xnorm := |x(i)|;
        endif
        if |x(i) − xold(i)| > dnorm then       {calcolo di ||x^(k+1) − x^(k)||}
          dnorm := |x(i) − xold(i)|;
        endif
      endfor
      Err := dnorm/xnorm;                      {stima dell'errore}
      if Err ≤ Tol then                        {controllo condizione di arresto}
        Iflag := 1;
      endif
```

Procedura 4.4: Algoritmo di Gauss-Seidel (seconda versione) - continua

$It := It + 1;$ $\{$*aggiornamento del contatore* It $\}$
for $i = 1, n$ **do** $\{$*aggiornamento di* $xold\}$
$xold(i) := x(i);$
endfor
until $(It = MaxIt \, or \, Iflag = 1)$ $\{$*verifica condizioni di arresto*$\}$
return $x, Err, Iflag, It$
end G-S2

Procedura 4.4: Algoritmo di Gauss-Seidel (seconda versione) - fine

4.5 Un esempio di software matematico per i metodi iterativi

A partire dalle procedure descritte nel precedente paragrafo, si possono costruire elementi di software matematico, per la risoluzione di sistemi di equazioni lineari mediante i metodi di Jacobi e Gauss-Seidel. Uno degli scopi di tale software è rendere trasparente, a chi le utilizza, i dettagli dell'algoritmo su cui si basano e di fornire la soluzione a partire da un insieme minimo di dati di input relativi al sistema da risolvere. Se, ad esempio, si utilizza il linguaggio di programmazione FORTRAN, le testate delle subroutine relative alle procedure `Jacobi2` e `G-S2` possono essere le seguenti:

```
SUBROUTINE JACOBI2(N,A,LDA,B,XOLD,MAXIT,TOL,X,ERR,IFLAG,IT)

SUBROUTINE G-S2(N,A,LDA,B,XOLD,MAXIT,TOL,X,ERR,IFLAG,IT)
```

Di seguito è illustrato un esempio di programma chiamante la subroutine `JACOBI2`, per risolvere il sistema lineare:

$$\begin{cases} 3x_1 & +0.5x_2 & +0.3x_3 & = 3.8 \\ 0.5x_1 & + \ 2x_2 & +0.9x_3 & = 3.4 \\ -0.1x_1 & +0.6x_2 & + \ x_3 & = 1.5, \end{cases}$$

la cui soluzione è $x^* = (1, 1, 1)$, con `MAXIT`=50, `TOL`=0.000001 e $x_i^{(0)} = 0.$, $i = 1, 2, 3$. In tale esempio si è supposto che i dati relativi alla matrice dei coefficienti ed al termine noto del sistema da risolvere, siano forniti da una subroutine (`SISTEMA`), che deve essere sviluppata dall'utente. Se si vuole risolvere un sistema lineare diverso è sufficiente modificare solo tale subroutine.

```
      PROGRAM JACOBI
C
C dichiarazione delle variabili
C
      REAL A(10,10), B(50), XOLD(10), X(10), TOL, ERR
      INTEGER N, LDA, MAXIT, CONV, K, IT, IFLAG, I
C
C inizializzazione delle variabili di input
C
      N=3
      LDA=10
      MAXIT=50
      TOL=0.000001
      DO 10 I=1,N
           XOLD(I)=0.0
 10    CONTINUE
C
C chiamata di SISTEMA per generare A e b
C
      CALL SISTEMA(N,A,LDA,B)
C
C chiamata di JACOBI2
C
      CALL JACOBI2(N,A,LDA,B,XOLD,MAXIT,TOL,
     * X,ERR,IFLAG,IT)
C
C stampa dei risultati
C
      IF (IFLAG .EQ. 1) THEN
           WRITE(*,*) Soluzione        =,(X(I),I=1,N)
           WRITE(*,*) Errore           =,ERR
           WRITE(*,*) Numero Iter.     =,IT
           WRITE(*,*) Iflag            =,IFLAG
      ELSE
```

Esempio di programma chiamante FORTRAN per la subroutine JACOBI2 - continua

```
            WRITE(*,*) Tolleranza non
     * raggiunta
      ENDIF
      STOP
      END


C
C subroutine che genera la matrice
C ed il vettore dei termini noti
C
      SUBROUTINE SISTEMA(N,A,LDA,B)
C
C dichiarazione dei parametri
C
      REAL A(LDA,N), B(N)
      INTEGER N, LDA
C
C definizione di A e di B
C
      A(1,1)=3.
      A(1,2)=0.5
      A(1,3)=0.3
      A(2,1)=0.5
      A(2,2)=2.
      A(2,3)=0.9
      A(3,1)=-0.1
      A(3,2)=0.6
      A(3,3)=1.

      B(1)=3.8
      B(2)=3.4
      B(3)=1.5

      RETURN
      END
```

Esempio di programma chiamante FORTRAN per la subroutine JACOBI2
- fine

Sistema (a)	IT	x_1	x_2	x_3	ERR
Jacobi	48	5.199987	6.799983	3.399991	$0.98{\cdot}10^{-6}$
G-S	26	5.199993	6.799994	3.399997	$0.70{\cdot}10^{-6}$

Tabella 4.8: Schema riassuntivo relativo alla risoluzione del sistema (a) con i metodi di Jacobi e G-S.

L'esecuzione del programma che implementa la versione in singola precisione della subroutine `JACOBI2`, con l'esempio di programma chiamante appena illustrato, fornisce i risultati seguenti:

```
Soluzione          = .9999998 .9999997 .9999995
Errore             = 9.53675E-07
Numero iter.       = 26
Iflag              = 1
```

♣ **Esempio 4.14.** Si risolvano i seguenti sistemi lineari:

$$(a)\quad \begin{cases} 3x_1 - 2x_2 = 2 \\ -x_1 + 2x_2 - x_3 = 5 \\ -x_2 + 2x_3 = 0 \end{cases} \qquad (b)\quad \begin{cases} 4x_1 + 2x_2 + x_3 = 1 \\ x_1 + 3x_2 + x_3 = 0 \\ 2x_2 - 3x_3 = 1 \end{cases}$$

$x_1^* = 5.2,\ x_2^* = 6.8,\ x_3^* = 3.4$ $\qquad$ $x_1^* = 1/3,\ x_2^* = 0,\ x_3^* = -1/3$

$$(c)\quad \begin{cases} 4x_1 - 2x_2 = 0 \\ -2x_1 + 4x_2 - 2x_3 = 2 \\ -2x_2 + 4x_3 = 0 \end{cases}$$

$x_1^* = 0.5,\ x_2^* = 1,\ x_3^* = 0.5$

con i metodi di Jacobi e di Gauss-Seidel.
Modificando opportunamente la subroutine `SISTEMA` in base al sistema da risolvere, sono stati costruiti dei programmi FORTRAN che implementano le subroutines `JACOBI2` e `G-S2` e programmi chiamanti simili all'esempio precedente. In particolare si è utilizzato `MAXIT`=50, `TOL`=10^{-6} e come valori iniziali $x_i^{(0)} = 0,\ i = 1, 2, 3$.
Nelle Tabelle 4.8, 4.9, 4.10 sono riportati i risultati ottenuti dall'esecuzione dei suddetti programmi per risolvere i sistemi in esame. Si osservi che `IT` indica il numero di iterazioni eseguite per soddisfare il criterio di arresto, x_1, x_2, x_3 sono gli ultimi valori calcolati prima dell'arresto del processo iterativo, `ERR` rappresenta la stima finale dell'errore relativo; la soluzione ottenuta è un'approssimazione della soluzione accurata a 7 cifre significative. Infine, dai valori di `IT` si evince che, per i sistemi in esame, il metodo di Gauss-Seidel converge alla soluzione più velocemente rispetto al metodo di Jacobi.

♣

Sistema (b)	IT	x_1	x_2	x_3	ERR
Jacobi	16	0.333333	0.000002	-0.333333	$0.60 \cdot 10^{-6}$
G-S	4	0.333333	0.000000	-0.333333	$0.10 \cdot 10^{-6}$

Tabella 4.9: Schema riassuntivo relativo alla risoluzione del sistema (b) con i metodi di Jacobi e G-S.

Sistema (c)	IT	x_1	x_2	x_3	ERR
Jacobi	38	0.499999	0.999998	0.499999	$0.95 \cdot 10^{-6}$
G-S	20	0.499999	0.999999	0.499999	$0.95 \cdot 10^{-6}$

Tabella 4.10: Schema riassuntivo relativo alla risoluzione del sistema (c) con i metodi di Jacobi e G-S.

4.6 Efficienza

Nel paragrafo 4.2 si è analizzata la complessità di tempo di una iterazione dei metodi di Jacobi e di G-S e si è visto che il numero di flops è circa uguale al numero di coefficienti non nulli del sistema. Il risultato di tale analisi si estende evidentemente ad un generico metodo iterativo della forma:

$$x_i^{(k+1)} = \sum_{j=1}^{n} c_{ij} x_j^{(k)} + d_i, \quad i = 1, \cdots, n, \quad k \geq 0, \tag{4.41}$$

per cui, indicato con $T_{it}(n)$ il relativo numero di flops, si ha:

$$T_{it}(n) \simeq n^2(1 - SP),$$

con SP grado di sparsità del sistema. Da ciò discende che la complessità di tempo totale è data da:

$$T(n) = k \cdot T_{it}(n) \simeq kn^2(1 - SP), \tag{4.42}$$

dove k è il numero di iterazioni che sono eseguite. Dalla (4.42) si evince, pertanto, che uno dei fattori da cui dipende l'efficienza di un metodo iterativo è il numero di iterazioni che sono eseguite e, quindi, la **velocità** con la quale la successione, generata dal metodo, converge alla soluzione del sistema.
Confrontando tale quantità con la complessità di tempo del metodo di eliminazione di Gauss, $T_{gauss} \simeq n^3/3$, si evince che risulta più efficiente utilizzare il metodo di Jacobi o il metodo di G-S, e cioè $T_{it}^{tot} < T_{gauss}$, se si effettua un numero di iterazioni k tale che

$$k < \frac{n}{3(1 - SP)}.$$

Ad esempio, se $SP = 0.99$, cioè se nel sistema, solo l'1% dei coefficienti è non nullo, si ha:

$$T_{it}^{tot}(n) \simeq k \cdot n^2 \cdot 0.01,$$

e, quindi, per un sistema lineare con tale grado di sparsità, è più efficiente utilizzare un metodo iterativo se si esegue un numero di iterazioni minore di circa $33n$.

4.6.1 Velocità di convergenza

Si assume che il metodo iterativo (4.41) sia consistente e che sia $\|C\| < 1$. In tali ipotesi, il metodo è convergente a x^*, dove x^* è la soluzione del sistema $Ax = b$.
Assunto, $e^{(k)} = x^{(k)} - x^*$, l'errore alla k-ma iterazione, il punto di partenza per ottenere una stima della velocità di convergenza del metodo iterativo è considerare la disuguaglianza:

$$\|e^{(k)}\| \leq \|C\|^k \|e^{(0)}\|, \tag{4.43}$$

che si ottiene dalla (4.33), dimostrata nel Teorema 4.3.1, con x^* al posto di $\bar{x}$. La (4.43) mette in relazione l'errore al passo k con l'errore iniziale e mostra che quanto più piccola è la quantità $\|C\|$ tanto più rapidamente converge a zero $\|e^{(k)}\|$ e quindi tanto più basso sarà il numero di iterazioni necessario per rendere la norma dell'errore minore di una prefissata tolleranza.

♣ **Esempio 4.15.** Se $\|C\| = \frac{1}{2}$ e $\|e^{(0)}\| = 1$, dalla (4.43) si ha $\|e^{(k)}\| \leq (\frac{1}{2})^k$; in tal caso, l'errore sarà minore di 10^{-7} dopo 24 iterazioni (infatti $k = 24$ è il più piccolo numero intero positivo per cui $(\frac{1}{2})^k \leq 10^{-7}$). D'altra parte, se $\|C\| = 0.8$, sono necessarie 73 iterazioni per avere $\|e^{(k)}\| \leq 10^{-7}$. ♣

Se le due quantità, $\|e^{(0)}\|$ e $\|C\|$, sono note (o possono essere facilmente calcolate), utilizzando la (4.43) è possibile ottenere una stima del minimo numero di iterazioni necessarie per rendere la norma dell'errore minore di una quantità `Tol` fissata. Infatti, si deve avere che:

$$\|C\|^k \|e^{(0)}\| \leq \texttt{Tol}. \tag{4.44}$$

Applicando la funzione logaritmo naturale (ln) ad entrambi i membri della disuguaglianza (4.44) si ha:

$$k \cdot \ln(\|C\|) + \ln(\|e^{(0)}\|) \leq \ln(\texttt{Tol}). \tag{4.45}$$

Poichè $\|C\| < 1$ per ipotesi, si ha $\ln(\|C\|) < 0$ e, quindi, dalla (4.45) si ricava che una stima del numero minimo di iterazioni per ottenere la stabilita riduzione dell'errore è:

$$k \geq \frac{\ln(\texttt{Tol}) - \ln(\|e^{(0)}\|)}{\ln(\|C\|)}. \tag{4.46}$$

Osservazione 4.2. *Esistono vari problemi legati all'utilizzo della stima (4.46). Il primo è che, non essendo nota in generale la soluzione del sistema, la quantità* $\ln(\|e^{(0)}\|)$ *non è calcolabile. Un altro problema è dovuto al fatto che la stima (4.46) dipende dalla norma che si utilizza, come mostra l'esempio 4.15. A conferma di ciò si consideri il sistema lineare:*

$$\begin{cases} x_1 + 0.4x_2 - 0.1x_3 = 1 \\ 0.3x_1 + x_2 - 0.001x_3 = -2.5 \\ 0.1x_1 + 0.4x_2 + x_3 = 3.0, \end{cases}$$

la cui soluzione è $x_1^* = 2.7315$, $x_2^* = -3.3154$, $x_3^* = 4.0530$. *Si verifica facilmente che, applicando il metodo di Jacobi a tale sistema, partendo da* $x_i^{(0)} = 0$, $i = 1, 2, 3$, *si ha* $\|e^{(0)}\|_\infty = 4.0530$ *e* $\|C\|_\infty = 0.5$, *mentre* $\|e^{(0)}\|_1 = 10.0999$ *e* $\|C\|_1 = 0.8$. *Nel primo caso la (4.46) mostra che sono necessarie circa 25 iterazioni affinchè l'errore sia minore di* 10^{-7}, *mentre nel secondo caso ne sono necessarie circa 82. Anche da tale esempio si evince, quindi, che il numero di iterazioni, che si stima essere necessario per rendere l'errore minore di* `Tol`, *dipende dalla norma che si utilizza per misurare l'ordine di grandezza della matrice* C[11].

4.6.2 Un algoritmo per la memorizzazione dei coefficienti di una matrice ad elevato grado di sparsità

La risoluzione di un sistema lineare di grandi dimensioni comporta, in generale, un'elevata complessità di spazio, oltre che di tempo. Se il sistema è sparso è possibile utilizzare opportune tecniche per memorizzare in forma compatta solo i coefficienti diversi da zero; ciò può consentire una risoluzione più efficiente anche di sistemi lineari di grandi dimensioni.

Uno dei più noti schemi per la memorizzazione di una matrice sparsa è il cosiddetto **schema dei tre vettori**. La matrice $A \in \mathbb{R}^{n\times n}$ è rappresentata utilizzando tre vettori, R, C, J. Il vettore R contiene gli elementi della matrice diversi da zero, riga per riga. L'indice di colonna in A dell'elemento $R(k)$ è dato da $C(k)$, mentre $J(i)$, $i = 1, \cdots, n$, fornisce la posizione, nell'array R, del primo elemento diverso da zero della i-ma riga.

♣ **Esempio 4.16.** Utilizzando lo schema dei tre vettori, la matrice

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

[11] Nei paragrafi successivi si vedrà che esiste un'altra stima della velocità di convergenza, che a differenza della (4.46) non dipende dalla norma utilizzata e quindi risulta ovviamente più affidabile.

è memorizzata nel modo seguente:

$$R = (7, -3, -1, 2, 8, 1, -3, 5, -1, 4, -2, 6);$$
$$C = (1, 3, 5, 1, 2, 3, 1, 4, 2, 5, 4, 6);$$
$$J = (1, 4, 6, 7, 9, 11, 13).$$

Si noti che l'ultimo elemento di J è uguale al numero totale degli elementi di R più uno.

♣

Nel caso in cui la matrice è simmetrica, è possibile utilizzare uno schema di memorizzazione più efficiente di quello appena illustrato, lo **schema del profilo**. Si fa uso di due array, R e M: il primo contiene, per ogni riga della matrice A, gli elementi compresi tra il primo non nullo su quella riga e l'elemento diagonale. Il secondo array, $M(i)$, $i = 1, \cdots, n$, contiene la posizione, in R, dell'elemento diagonale della i-ma riga.

♣ **Esempio 4.17.** Secondo lo schema del profilo, la matrice

$$A = \begin{pmatrix} 25 & 3 & 0 & 0 & 0 \\ 3 & 21 & 2 & 4 & 0 \\ 0 & 2 & 23 & 0 & 0 \\ 0 & 4 & 0 & 22 & 1 \\ 0 & 0 & 0 & 1 & 20 \end{pmatrix}$$

è memorizzata mediante

$$R = (25, 3, 21, 2, 23, 4, 0, 22, 1, 20);$$
$$M = (1, 3, 5, 8, 10).$$

♣

Utilizzando tale schema, si osserva che il generico elemento a_{ij} di A è memorizzato in R se $i - j < M(i) - M(i-1)$. Inoltre $a_{ij} = R(p)$, con $p = M(i) - i + j$.

Negli esempi precedenti, date le dimensioni della matrice A, non appare evidente il vantaggio nell'utilizzo degli schemi descritti rispetto alle tecniche di memorizzazione usuali (**array bidimensionale**); il vantaggio è invece evidente per matrici di grandi dimensioni.

♣ **Esempio 4.18.** Si consideri una matrice quadrata A di dimensione $n = 100$ con $s = 400$ elementi diversi da zero. Se si usa per la sua memorizzazione l'usuale array bidimensionale, la complessità di spazio è $n \times n = 10000$, mentre se si sceglie lo schema dei tre vettori la complessità si riduce a $s + s + n + 1 = 901$, con un risparmio, quindi, di circa il 91%. ♣

Se k è il numero di iterazioni eseguite, la complessità di tempo totale è:

$$T_{it}^{tot}(n) \simeq k \cdot n^2(1 - SP).$$

Nella **Procedura 4.5** è illustrata una nuova versione dell'algorimo relativo al metodo di Jacobi, in cui si assume che la matrice dei coefficienti del sistema sia memorizzata con lo schema dei tre vettori. Nella notazione utilizzata, i tre vettori R, C, J, sono rappresentati rispettivamente dagli array `r`, `c`, `ia`.

procedure Jacobi3(**in:** $n, s, r, c, ia, b, xold, MaxIt, Tol$; **out:** $x, Err, Iflag, It$)

/# SCOPO: risoluzione di un sistema di equazioni lineari
con il metodo iterativo di Jacobi;
la matrice è memorizzata con lo schema dei tre vettori

/# SPECIFICHE PARAMETRI:

/# PARAMETRI DI INPUT:

var: n	: **intero**	{*dimensione del sistema*}
var: s	: **intero**	{*numero coefficienti non nulli*}
var: $r(s)$	: **array di reali**	{*coefficienti non nulli riga per riga*}
var: $c(s)$	: **array di interi**	{*indici di colonna degli elementi di r*}
var: $ia(n)$	: **array di interi**	{*ia(i) = posizione in r del primo*}
		{*elemento non nullo della i-ma riga*}
var: $b(n)$	: **array di reali**	{*termini noti del sistema*}
var: $xold(n)$	: **array di reali**	{*valori iniziali*}
var: $MaxIt$	: **intero**	{*numero massimo di iterazioni*}
var: Tol	: **reale**	{*tolleranza usata nel criterio di arresto*}

/# PARAMETRI DI OUTPUT:

var: $x(n)$	: **array di reali**	{*soluzione calcolata*}
var: Err	: **reale**	{*stima dell'errore*}
var: $Iflag$	: **intero**	{*= 0 se la tolleranza non è soddisfatta*}
		{*= 1 se la tolleranza è soddisfatta*}
var: It	: **intero**	{*numero di iterazioni eseguite*}

/# VARIABILI LOCALI:

var: i, j, l	: **interi**	{*contatori*}
var: $diff$	: **intero**	{*numero degli elementi non nulli di*}
		{*ciascuna riga della matrice dei coefficienti*}
var: $xnorm$	: **reale**	{*norma del vettore calcolato all'ultimo passo*}
var: $dnorm$	: **reale**	{*norma della differenza tra i vettori ottenuti*}
		{*in due iterazioni successive*}
var: d	: **reale**	{*elemento diagonale, riga per riga*}

Procedura 4.5: Algoritmo di Jacobi (terza versione) - continua

```
/# INIZIO ISTRUZIONI:
      It := 0;                                  {inizializzazione di It ed Iflag}
      Iflag := 0;
      repeat                                                {ciclo principale con}
         l := 0;                                          {condizioni di arresto}
         xnorm := 0.;
         dnorm := 0.;
         for i = 1, n do                                {ciclo per il calcolo di x^(k)}
            x(i) := b(i);
            diff := ia(i + 1) − ia(i);
            for j = 1, diff do                        {ciclo per il calcolo di x_i^(k)}
               l := l + 1;
               if c(l) ≠ i then
                  x(i) := x(i) − r(l) ∗ xold(c(l));
               else
                  d := r(l);
               endif
            endfor
            x(i) := x(i)/d;
            if |x(i)| > xnorm then                    {calcolo di ||x^(k+1)||}
               xnorm := |x(i)|;
            endif
```

Procedura 4.5: Algoritmo di Jacobi (terza versione) - continua

```
        if |x(i) − xold(i)| > dnorm then          {calcolo di ||x^(k+1) − x^(k)||}
          dnorm := |x(i) − xold(i)|;
        endif
      endfor
      Err := dnorm/xnorm;                          {calcolo della stima dell'errore}
      if Err ≤ Tol then                            {controllo condizione di arresto}
        Iflag := 1;
      endif
      It := It + 1;                                {aggiornamento del contatore It}
      for i = 1, n do                              {aggiornamento di xold}
        xold(i) := x(i);
      endfor
    until (It = MaxItorIflag = 1)                  {verifica delle condizioni di arresto}
    return x, Err, Iflag, It
end Jacobi3
```

Procedura 4.5: Algoritmo di Jacobi (terza versione) - fine

4.7 Un esempio di programma MATLAB per i metodi iterativi

In questo paragrafo si descrive l'utilizzo del software MATLAB per lo sviluppo di un programma che implementi la **Procedura 4.3** relativa al metodo di Jacobi.

Allo stato attuale non sono presenti funzioni per la risoluzione di sistemi di equazioni lineari con metodi iterativi. Tuttavia, MATLAB fornisce la possibilità di utilizzare un insieme di istruzioni, come un linguaggio di programmazione, per lo sviluppo di funzioni che risolvono problemi specifici.

Un esempio di programma MATLAB per la risoluzione di sistemi lineari con il metodo di Jacobi è riportato nella Funzione 1. Tale funzione ha 4 parametri di output, cioè quelli racchiusi tra parentesi quadre dopo la parola chiave **function**. La funzione ha 5 parametri di input, quelli specificati nelle parentesi tonde dopo il nome della funzione. Il simbolo % si utilizza per inserire i commenti. La funzione **norm(x,inf)** calcola

la norma del massimo del vettore `x`. La funzione **length**`(b)` restituisce, in output, la lunghezza del vettore `b`. Segue un esempio di utilizzo della funzione in esame.

♣ **Esempio 4.19.** Si risolva il sistema lineare:

$$\begin{cases} 3x_1 & +0.5x_2 & +0.3x_3 & = 3.8 \\ 0.5x_1 & + 2x_2 & +0.9x_3 & = 3.4 \\ -0.1x_1 & +0.6x_2 & + x_3 & = 1.5, \end{cases}$$

la cui soluzione è $x^* = (1, 1, 1)$, uilizzando la Funzione 1.
Le seguenti istruzioni definiscono le variabili che servono come parametri di input per la funzione.

```
>> A=[3 0.5 0.3; 0.5 2 0.9; -0.1 0.6 1];
>> b=[3.8; 3.4; 1.5];
>> x=[0; 0; 0;];
>> Tol=0.000001;
>> MaxIt=50;
```

L'istruzione che segue chiama la funzione.

```
>> [x,ERR,IFLAG,IT] = Jacobi(A,b,x,Tol,MaxIt)
```

L'output è:

```
x =
      0.99999984417416
      0.99999968961009
      0.99999966015567

ERR =
      9.221026131790661e-07

IFLAG =
      1

IT =
      26
```

♣

```
function [x,ERR,IFLAG,IT] = Jacobi(A,b,x,Tol,MaxIt);
%
% ------------------------------------------------------------------
%
%       SCOPO
%       =====
%       Calcola la soluzione di un sistema di equazioni lineari
%       mediante il metodo iterativo di Jacobi.
%
%
%       PARAMETRI DI INPUT
%       ==================
%
%       A - Matrice dei coefficienti del sistema.
%
%       b - Vettore dei termini noti.
%
%       x - Valori iniziali.
%
%       Tol - Tolleranza per il criterio di arresto.
%
%       MaxIt - Numero massimo di iterazioni.
%
%
%       PARAMETRI DI OUTPUT
%       ===================
%
%       x - Soluzione.
%
%       ERR - Stima dell'errore relativo.
%
%       IFLAG - Vale 1 se si è raggiunta la tolleranza,
%                    0 se si è raggiunto il numero massimo di iterazioni.
%
%       IT - Numero di iterazioni eseguite.
%
%
% ------------------------------------------------------------------
%
%
```

Funzione 1:
Funzione MATLAB per la risoluzione di un sistema di equazioni lineari con il metodo di Jacobi - continua

```
n=length(b);                                  % lunghezza di b =
                                              % = ordine del sistema
xold=x;                                       % vettore di appoggio
ERR=1.0;                                      % inizializzazione della stima dell'errore
IFLAG=0;                                      % inizializzazione di IFLAG
IT=0;                                         % inizializzazione del numero di iterazioni

while ( ERR>Tol & IT<MaxIt)                   % k-ma iterazione
    for i=1:n
      sum=b(i)-(A(i,[1:i-1,i+1:n]))*...
                      (xold([1:i-1,i+1:n]));
      x(i)=sum/A(i,i);                        % i-ma componente della
                                              % k-ma approssimazione
    end
    ERR=norm(x-xold, inf)/norm(x,inf);        % stima dell'errore relativo
    xold=x;
    IT=IT+1;
end

if (ERR ≤ Tol)                                % controllo sulla stima errore
    IFLAG=1;
end
```

Funzione 1:
Funzione MATLAB per la risoluzione di un sistema di equazioni lineari con il metodo di Jacobi - fine

4.8 I Metodi iterativi stazionari

In questo paragrafo, attraverso una riformulazione in termini matriciali, sono ripresi ed approfonditi i concetti di base discussi nel paragrafo precedente. Inoltre, sono illustrate alcune tecniche classiche per accelerare la convergenza di metodi iterativi. In particolare l'attenzione è rivolta alla classe di metodi noti con il nome di **metodi iterativi stazionari**.

Si consideri il sistema di equazioni lineari:

$$Ax = b, \tag{4.47}$$

con $A = [a_{ij}] \in \mathbb{R}^{n\times n}$ **non singolare**, $x = [x_i]$ e $b = [b_i] \in \mathbb{R}^n$. Sia $x^* = A^{-1}b$ la sua soluzione.

Si possono dare le seguenti definizioni generali:

Definizione 4.8.1. (Metodo iterativo)
Una famiglia di funzioni $\{G_k\}$*:*

$$G_k \ : \ D_k \ \subseteq \ (\mathbb{R}^n)^{k+p} \ = \ \underbrace{\mathbb{R}^n \times ... \times \mathbb{R}^n}_{k+p \text{ volte}} \rightarrow \ \mathbb{R}^n, \quad k = 0, 1, ...$$

definisce un **processo iterativo** *(***metodo iterativo***)* **con** p **punti iniziali** *e con dominio* $D^* \subseteq D_0$*, se* D^* *è non vuoto e se per ogni punto* $(x^{(0)}, ..., x^{(-p+1)}) \in D^*$*, la successione* $\{x^{(k)}\}$ *generata mediante le:*

$$x^{(k+1)} = G_k(x^{(k)}, ..., x^{(-p+1)}), \quad k = 0, 1, ... \tag{4.48}$$

è tale che $(x^{(k)}, ..., x^{(-p+1)}) \in D_k$*,* $\forall k \geq 0$*. Un punto* x^* *tale che* $\lim_{k\rightarrow\infty} x^{(k)} = x^*$ *è detto un* **limite** *del processo iterativo.*

Le seguenti definizioni caratterizzano un processo iterativo del tipo (4.48).

Definizione 4.8.2. (Metodo iterativo a s **passi)**
Un processo iterativo (4.48) è detto un **metodo a** s **passi**, *se* $p = s - k$ *e se*

$$G_k \ : \ D_k \ \subseteq \ (\mathbb{R}^n)^s \rightarrow \ \mathbb{R}^n, \quad k = 0, 1, ...$$

e quindi

$$x^{(k+1)} = G_k(x^{(k)}, ..., x^{(k-s+1)}), \quad k = 0, 1, ...$$

Definizione 4.8.3. (Metodo iterativo stazionario)
Un metodo iterativo del tipo (4.48) a s passi è detto **stazionario** *se* $G_k = G$ *e* $D_k = D$*,* $\forall k \geq 0$*. Quindi, in tal caso si ha:*

$$x^{(k+1)} = G(x^{(k)}, ..., x^{(k-s+1)}), \quad k = 0, 1, ...$$

La funzione G *è detta la* **funzione di iterazione del metodo.**

I metodi di Jacobi e di Gauss-Seidel possono essere scritti nella forma:

$$x^{(k+1)} = G(x^{(k)}) = Cx^{(k)} + d, \quad k = 0, 1, 2, ... , \tag{4.49}$$

dove $C = [c_{ij}] \in \mathbb{R}^{n\times n}$ è detta **matrice di iterazione** e $d = [d_i] \in \mathbb{R}^n$. La (4.49) definisce la generica iterazione di un metodo iterativo **stazionario ad un passo**. Il termine *stazionario* si riferisce al fatto che nel metodo, a parte ovviamente la sequenza $\{x^{(k)}\}$, non ci sono quantità che variano al variare delle iterazioni (la matrice C ed il vettore d sono costanti). L'espressione *ad un passo* sta ad indicare che il vettore alla generica iterazione si ottiene utilizzando solo il vettore calcolato alla iterazione precedente.

♣ **Esempio 4.20.** Altri esempi di metodi iterativi sono i seguenti:

- **metodo non stazionario ad un passo**

$$x^{(k+1)} = C_{k+1}x^{(k)} + d^{(k+1)}, \quad k = 0, 1, \ldots$$

- **metodo stazionario a due passi**

$$x^{(k+1)} = Cx^{(k)} + Bx^{(k-1)} + d, \quad k = 0, 1, \ldots$$

- **metodo non stazionario a due passi**

$$x^{(k+1)} = C_{k+1}x^{(k)} + B_{k+1}x^{(k-1)} + d^{(k+1)}, \quad k = 0, 1, \ldots$$

- **metodo stazionario a s passi**

$$x^{(k+1)} = \sum_{i=1}^{s} Cx^{(k+1-i)} + d, \quad k = 0, 1, \ldots$$

♣

I metodi iterativi concettualmente più semplici, ed in molti casi più utilizzati, sono quelli **stazionari ad un passo** (p=1), cioè quelli del tipo:

$$x^{(k+1)} = G(x^{(k)}), \qquad k = 0, 1, ... \tag{4.50}$$

Affinchè un metodo iterativo della forma (4.49) costituisca una procedura effettivamente utilizzabile per risolvere il sistema (4.47), è indispensabile che esso sia consistente.
Se la successione generata dal metodo iterativo (4.49) converge, indicato con $\bar{x}$ il limite, risulta

$$\bar{x} = \lim_{k\to\infty} x^{(k+1)} = C\left(\lim_{k\to\infty} x^{(k)}\right) + d = C\bar{x} + d,$$

cioè il limite $\bar{x}$ è soluzione del sistema lineare:

$$x = Cx + d\,, \quad o \ \ equivalentemente \ \ (I - C)x = d, \tag{4.51}$$

il quale è detto il **sistema associato** al sistema (4.47). Allora, per i metodi iterativi del tipo (4.49) la definizione di consistenza può essere così riformulata:

Definizione 4.8.4. (Metodo consistente)
Un metodo iterativo del tipo (4.49) si dice consistente con il sistema di equazioni lineari (4.47), se la matrice $(I - C)$ è non singolare e, detta $\bar{x}$ l'unica soluzione del sistema di equazioni lineari $(I - C)x = d$, si ha che $\bar{x}$ è anche la soluzione del sistema $Ax = b$.

4.9 Metodi basati sulla decomposizione della matrice (splitting)

La tecnica più comune per sviluppare un metodo iterativo del tipo (4.49) che sia consistente è quella di considerare una decomposizione (**splitting**) della matrice A della forma:

$$A = M - R, \tag{4.52}$$

dove $M \in \mathbb{R}^{n\times n}$ è una matrice non singolare, detta **matrice di splitting**. Utilizzando tale decomposizione, il sistema lineare (4.47) è trasformato nel sistema:

$$Mx = Rx + b,$$

da cui deriva il metodo iterativo:

$$Mx^{(k+1)} = Rx^{(k)} + b, \quad k = 0, 1, 2, ...$$

che, essendo M non singolare, può essere anche scritto come:

$$x^{(k+1)} = M^{-1}Rx^{(k)} + M^{-1}b, \quad k = 0, 1, 2, ... \tag{4.53}$$

Il metodo (4.53) è un metodo del tipo (4.49), dove in particolare:

$$C = M^{-1}R = I - M^{-1}A, \qquad d = M^{-1}b. \tag{4.54}$$

Inoltre si ha:

Teorema 4.9.1. *Posto $A = M - R$, con M matrice non singolare, il metodo iterativo:*

$$x^{(k+1)} = M^{-1}Rx^{(k)} + M^{-1}b, \quad k = 0, 1, 2, ...$$

è consistente con il sistema $Ax = b$.

Dimostrazione Poichè A e M sono matrici non singolari, in base alla (4.54) anche la matrice $I - C = M^{-1}A$ è non singolare. Sia $\bar{x}$ l'unica soluzione del sistema $x = Cx + d$. Si ha:

$$\bar{x} = C\bar{x} + d = (I - M^{-1}A)\bar{x} + M^{-1}b = \bar{x} - M^{-1}(A\bar{x} - b),$$

da cui discende che $A\bar{x} = b$ e, quindi, $\bar{x}$ è la soluzione del sistema $Ax = b$. In base alla definizione 4.8.4, il metodo è quindi consistente. ■

È importante osservare che la scelta della matrice di splitting M in (4.52), per ragioni di efficienza, non può essere completamente arbitraria. Dalla (4.53) si osserva infatti che, ad ogni iterazione del metodo, è richiesto il calcolo di un vettore del tipo $y = M^{-1}z$. Di

conseguenza, è importante scegliere la matrice M in modo che la risoluzione di sistemi lineari della forma $My = z$ comporti un "ragionevole" costo computazionale. In generale, la scelta della matrice M cade su matrici per le quali il calcolo di $y = M^{-1}z$ richieda un costo computazionale equivalente a quello di un prodotto matrice-vettore (circa n^2 flops).

♣ **Esempio 4.21.** Si consideri la seguente decomposizione della matrice $A = [a_{ij}] \in \mathbb{R}^{n\times n}$:

$$A = D + L + U, \tag{4.55}$$

dove:

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n-1,n-1} & 0 \\ 0 & \cdots & \cdots & 0 & a_{nn} \end{pmatrix}, \quad a_{ii} \neq 0, \quad i = 1, \ldots, n$$

$$L = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ a_{21} & 0 & \cdots & \cdots & 0 \\ a_{31} & a_{32} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 0 \end{pmatrix}, \quad U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{n-2,n-1} & a_{n-2,n} \\ 0 & \cdots & \cdots & 0 & a_{n-1,n} \\ 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}.$$

Utilizzando tale decomposizione e denotando con il simbolo $[Px]_i$ la i-ma componente del prodotto tra una matrice $P \in \mathbb{R}^{n\times n}$ ed un vettore $x \in \mathbb{R}^n$, il metodo di Jacobi:

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}, \quad i = 1, \cdots, n,$$

può essere riscritto nel modo seguente:

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \frac{1}{a_{ii}} [(L+U)x^{(k)})]_i, \quad i = 1, \cdots, n,$$

da cui:

$$x_i^{(k+1)} = [D^{-1}b]_i - [D^{-1}(L+U)x^{(k)})]_i, \quad i = 1, \cdots, n,$$

e, quindi, in forma matriciale:

$$x^{(k+1)} = -D^{-1}(L+U)x^{(k)} + D^{-1}b. \tag{4.56}$$

Dalla (4.56) si evince che il metodo iterativo di Jacobi deriva da uno splitting di A del tipo (4.52), dove in particolare:

$$M_J = D; \quad R_J = -(L+U).$$

Si osservi che la matrice di splitting del metodo di Jacobi è la matrice diagonale i cui elementi sono gli elementi diagonali della matrice A e quindi, poichè tali elementi sono diversi da zero per ipotesi, è non singolare. Di conseguenza, il metodo di Jacobi è consistente (Teorema 4.9.1).
In conclusione, in forma (4.49) il metodo di Jacobi assume la seguente espressione:

METODO DI JACOBI

$$x^{(k+1)} = C_J x^{(k)} + d_J$$
$$C_J = -D^{-1}(L+U) = I - D^{-1}A$$
$$d_J = D^{-1}b.$$

♣

♣ **Esempio 4.22.** Il metodo di G-S :

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, \cdots, n,$$

può essere riscritto come:

$$\frac{1}{a_{ii}}\sum_{j=1}^{i} a_{ij}x_j^{(k+1)} = \frac{b_i}{a_{ii}} - \frac{1}{a_{ii}}\sum_{j=i+1}^{n} a_{ij}x_j^{(k)}, \quad i = 1, \cdots, n,$$

da cui, in base della decomposizione (4.55), si ha:

$$[D^{-1}(D+L)x^{(k+1)}]_i = [D^{-1}b]_i - [D^{-1}Ux^{(k)}]_i, \quad i = 1, \cdots, n,$$

e quindi, in forma matriciale:

$$x^{(k+1)} = -(D+L)^{-1}Ux^{(k)} + (D+L)^{-1}b. \tag{4.57}$$

Dalla (4.57) si osserva che il metodo di G-S deriva da uno splitting del tipo (4.52), dove, in particolare:

$$M_{GS} = D + L; \quad R_{GS} = -U.$$

La matrice di splitting del metodo di G-S è quindi triangolare inferiore e, poichè gli elementi diagonali di A sono diversi da zero, è non singolare, da cui discende che il metodo di G-S è consistente.
In conclusione, in forma (4.49) esso assume la seguente espressione:

METODO DI GAUSS-SEIDEL

$$x^{(k+1)} = C_G x^{(k)} + d_G$$
$$C_G = -(D+L)^{-1}U = I - (D+L)^{-1}A$$
$$d_G = (D+L)^{-1}b$$

♣

4.10 Un cenno ai metodi iterativi non stazionari basati sui sottospazi di Krylov:

Dalla (3.54) segue che un metodo iterativo stazionario ad un passo del tipo:

$$x^{(k+1)} = C\,x^{(k)} + d$$

è esprimibile anche nella forma:

$$x^{(k+1)} = (I - M^{-1}A)x^{(k)} + M^{-1}b = x^{(k)} + M^{-1}(A\,x^{(k)} - b)$$

ovvero, posto $r^{(k)} = b - A\,x^{(k)}$, come:

$$x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)} \tag{4.58}$$

dove M è la matrice derivante dallo splitting (3.52) di A.
La (4.58), nel caso particolare in cui M sia la matrice identica, prende il nome di *iterazione di Richardson.* Si noti che, a partire dall'iterazione di Richardson, segue:

$$b - A\,x^{(k+1)} = b - A\,x^{(k)} - A\,r^{(k)}$$

cioè:

$$r^{(k+1)} = (I - A)r^{(k)} = (I - A)^{(k+1)}r^{(0)}$$

ovvero, il *residuo* $r^{(k)}$ si esprime come il polinomio in A:

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

di grado $k+1$, moltiplicato per il residuo iniziale. Questa è una proprietà comune a molti metodi iterativi, ed è particolarmente utile per derivare tecniche di accelerazione della convergenza. Sostituendo questa espressione del residuo nell'iterazione di Richardson, si ha:

$$x^{(k+1)} = x^{(k)} + r^{(k)} = x^{(k)} + (I - A)^{(k)}r^{(0)}$$

da cui, esprimendo a sua volta ciascuna iterata in funzione della precedente, si ha:

$$x^{(k+1)} = r^0 + r^{(1)} + r^{(2)} + \ldots + r^{(k)} = \sum_{j=0}^{k}(I - A)^j r^{(0)}$$

ovvero, l'iterata al passo $k+1$ appartiene al sottospazio

$$K_k = \{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{(k)}r^{(0)}\}$$

generato da A e da $r^{(0)}$. Tale sottospazio, di dimensione k, generato dai vettori $r^{(0)}, Ar^{(0)}, \ldots, A^{(k)}r^{(0)}$, è detto *sottospazio di Krylov.*

L'iterazione di Richardson genera, quindi, al generico passo k, un'approssimazione della soluzione del sistema $Ax = b$, appartenente al sottospazio di Krylov di dimensione k. In particolare, invece della base standard, costruita a partire dal residuo iniziale, si costruiscono basi ortonormali, perché sono più convenienti da un punto di vista computazionale. L'ortonormalizzazione dei vettori della base, tipicamente basato sul procedimento di Gram-Schmidt modificato, conduce a relazioni di ricorrenza anche su tali vettori, e il metodo iterativo che deriva risulta quindi **non stazionario**, del tipo:

$$x^{(k+1)} = x^{(k)} + \alpha_{k+1} r^{(k)}$$

dove i coefficienti α_{k+1} sono scelti in modo da costruire *in qualche senso* la migliore approssimazione possibile, a ogni passo. A questa classe di metodi appartengono ad esempio i metodi del **Gradiente Coniugato** (CG) e del GMRES (**Generalized Minimal Residual**). Nel Gradiente Coniugato l'iterata al passo k viene calcolata come segue:

$$x^{(k+1)} = x^{(k)} + \alpha_{k+1} p^{(k+1)}$$

dove:

$$p^{(k+1)} = r^{(k+1)} + \beta_{k+1} p^{(k)}, \quad r^{(k+1)} = r^{(k)} - \alpha_{k+1} A p^{(k+1)}$$

con:

$$\alpha_{k+1} = \frac{p^{(k)T} \cdot r^{(k)}}{p^{(k+1)T} A p^{(k+1)}}, \quad \beta_{k+1} = -\frac{r^{(k+1)T} \cdot A p^{(k+1)}}{p^{(k)T} A p^{(k)}}$$

e $p^{(0)} = r^{(0)}$. Con tale scelta di α_{k+1} e di β_{k+1} al passo $k+1$ il residuo $r^{(k+1)}$ risulta ortogonale a tutti quelli precedenti ed inoltre $x^{(k+1)}$ viene individuato come il vettore che nel sottospazio di Krylov costruito al passo $k+1$ minimizza la A-norma[12] dell'errore al passo $k+1$, $e^{(k+1)} = x^{(k+1)} - x^\star$, dove con $x^\star$ abbiamo indicato la soluzione del sistema $Ax = b$.

Se la matrice non è simmetrica e non è neanche definita positiva, l'ortogonalizzazione dei vettori della base al passo $k+1$, porta a relazioni di ricorrenza con i vettori costruiti alle iterazioni precedenti, più articolate[13], come ad esempio nel caso del GMRES, in cui l'approssimazione $x^{(k)}$ viene costruita in modo da minimizzare, tra tutti i vettori appartenenti al sottospazio di Krylov di dimensione $k+1$, la norma euclidea del residuo $r^{(k+1)}$.

4.11 Studio della convergenza

In questo paragrafo si studiano alcune condizioni sotto le quali la successione $\{x^{(k)}\}$ generata da un metodo iterativo del tipo (4.49) converge, qualunque sia la scelta di $x^{(0)}$.

[12] Data una matrice $A \in R^{n\times n}$ simmetrica e definita positiva, ed un vettore $z \in R^n$, si definisce A-norma del vettore z la quantità $z^T A z$.

[13] Ad esempio, si usano tecniche di restart, dopo un prefissato numero di passi

Si analizzano ora le proprietà di convergenza dei metodi iterativi *stazionari ad un passo*. A tal fine si premettono alcune definizioni e risultati [14].

Definizione 4.11.1. (Contrazione)
Una funzione $G : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ *è una* **contrazione** *sull'insieme* $D_0 \subseteq D$, *se esiste un* $0 < \alpha < 1$ *tale che:*

$$\|G(x) - G(y)\| \leq \alpha \|x - y\|, \quad \forall x, y \in D_0.$$

Si verifica immediatamente che se una funzione $G : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ è una contrazione su $D_0 \subseteq D$, allora essa è continua in D_0.

Definizione 4.11.2. (Punto fisso)
Data una funzione $G : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$, *un punto* $x^* \in D$ *è detto un* **punto fisso** *di* G *se è soluzione dell'equazione:*

$$x = G(x).$$

Teorema 4.11.1. *Sia* $G : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ *una contrazione su* $D_0 \subseteq D$. *Allora* G *può avere al più un punto fisso in* D_0.

Dimostrazione Siano x e $y \in D_0$, $x \neq y$, due punti fissi di G. Si ha:

$$\|x - y\| = \|G(x) - G(y)\| < \|x - y\|,$$

che è una contraddizione. Quindi $x = y$. ■

Il Teorema che segue (noto come **Teorema delle Contrazioni**) assicura che, nelle ipotesi in cui G sia una contrazione, la successione $\{x^{(k)}\}$ generata da un metodo stazionario ad un passo converge ad un punto fisso di G, qualunque sia il punto iniziale $x^{(0)}$. Ciò mostra anche che una contrazione ha almeno un punto fisso. Per il Teorema 4.11.1 esso è unico.

Teorema 4.11.2. *Sia* $G : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ *una contrazione su un insieme chiuso* $D_0 \subseteq D$ *e tale che* $G(D_0) \subseteq D_0$. *Allora la successione* $\{x^{(k)}\}$ *generata dal metodo iterativo (4.50) converge ad un punto fisso di* G, *qualunque sia il punto iniziale* $x^{(0)}$.

[14] Tali definizioni e risultati si possono estendere nel caso più generale degli spazi metrici. In particolare, il **Teorema delle Contrazioni** è noto anche come **Teorema di punto fisso di Banach-Caccioppoli** ed è un importante strumento nella teoria degli spazi metrici.

Dimostrazione Sia $x^{(0)}$ un arbitrario punto di D_0. Poichè $G(D_0) \subseteq D_0$, la successione $\{x^{(k)}\}$ generata da $x^{(k+1)} = G(x^{(k)})$, $k = 0, 1, ...$, è ben definita ed appartiene a D_0. Inoltre, dall'ipotesi che G sia una contrazione su D_0 si ha:

$$\|x^{(k+1)} - x^{(k)}\| = \|G(x^{(k)}) - G(x^{(k-1)})\| \leq \alpha \|x^{(k)} - x^{(k-1)}\|.$$

Applicando lo stesso ragionamento a $\|x^{(k)} - x^{(k-1)}\|$ si ottiene:

$$\|x^{(k+1)} - x^{(k)}\| \leq \alpha^2 \|x^{(k-1)} - x^{(k-2)}\|$$

e, quindi, iterando il procedimento, si ha:

$$\|x^{(k+1)} - x^{(k)}\| \leq \alpha^k \|x^{(1)} - x^{(0)}\|. \tag{4.59}$$

La (4.59) è facilmente generalizzabile nella:

$$\|x^{(k+i)} - x^{(k+i-1)}\| \leq \alpha^{i-1} \|x^{(k+1)} - x^{(k)}\|, \;\; i \geq 2, \; \forall k \geq 0. \tag{4.60}$$

Dalle (4.59) e (4.60) segue che:

$$\|x^{(k+p)} - x^{(k)}\| \leq \sum_{i=1}^{p} \|x^{(k+i)} - x^{(k+i-1)}\| \leq (\alpha^{p-1} + \alpha^{p-2} + ... + 1)\|x^{(k+1)} - x^{(k)}\|$$

$$\leq (\alpha^{p-1} + \alpha^{p-2} + ... + 1)\alpha^k \|x^{(1)} - x^{(0)}\| \leq [\alpha^k/(1-\alpha)]\|x^{(1)} - x^{(0)}\|,$$

dall' essere

$$\sum_{j=0}^{p-1} \alpha^j = \frac{1 - \alpha^{p-1}}{1 - \alpha},$$

e poiché $0 < \alpha < 1 \Rightarrow 0 < \alpha^{p-1} < 1$ con $p \geq 1$. Quindi, $\{x^{(k)}\}$ è una successione di Cauchy e converge ad un limite $x^* \in D_0$. Inoltre, dalla continuità di G si ha:

$$x^* = \lim_{k \to \infty} x^{(k+1)} = \lim_{k \to \infty} G(x^{(k)}) = G(x^*),$$

per cui x^* è un punto fisso di G. ■

Il Teorema precedente può essere applicato anche a processi iterativi non stazionari e si generalizza nel seguente:

Teorema 4.11.3. *Sia $G : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ una contrazione su un insieme chiuso $D_0 \subseteq D$ e tale che $G(D_0) \subseteq D_0$. Si assuma che $G_k : D_0 \subseteq \mathbb{R}^n \to \mathbb{R}^n$, $k = 0, 1, \ldots$, siano funzioni tali che $G_k(D_0) \subseteq D_0$, $\forall k \geq 0$, e che:*

$$\lim_{k \to \infty} \|G_k(x) - G(x)\| = 0, \quad \textit{uniformemente rispetto a } x \in D_0 \; . \tag{4.61}$$

Allora la successione $\{x^{(k)}\}$ generata dal metodo non stazionario:

$$x^{(k+1)} = G_k(x^{(k)}), \quad k = 0, 1, ... \tag{4.62}$$

converge all'unico punto fisso x^* di G in D_0, qualunque sia il punto iniziale $x^{(0)}$.

Dimostrazione Sia $x^{(0)}$ un arbitrario punto di D_0. Poichè $G_k(D_0) \subseteq D_0$, $\forall k \geq 0$, la successione $\{x^{(k)}\}$ generata dal metodo (4.62) è ben definita ed appartiene a D_0. Inoltre, dall'ipotesi che G sia una contrazione su D_0 si ha:

$$\|x^{(k+1)} - x^*\| = \|G_k(x^{(k)}) - G(x^*)\| \leq \|G_k(x^{(k)}) - G(x^{(k)})\| + \|G(x^{(k)}) - G(x^*)\| \leq$$

$$\leq \epsilon_k + \alpha\|x^{(k)} - x^*\| \leq \cdots \leq \sum_{j=0}^{k} \alpha^{k-j}\epsilon_j + \alpha^{k+1}\|x^{(0)} - x^*\|, \tag{4.63}$$

con $\epsilon_k = \|G_k(x^{(k)}) - G(x^{(k)})\|$. Per l'ipotesi (4.61) la successione $\{\epsilon_k\}$ converge a zero e, quindi, dato un $\epsilon > 0$, esiste un k_0 tale che $\epsilon_k \leq \epsilon$, $\forall k \geq k_0$. Posto $\gamma_k = \sum_{j=0}^{k} \alpha^{k-j}\epsilon_j$, si ha:

$$\gamma_k \leq \alpha^{k-k_0}\gamma_{k_0} + \sum_{j=k_0+1}^{k} \alpha^{k-j}\epsilon_j \leq \alpha^{k-k_0}\gamma_{k_0} + \epsilon[1/(1-\alpha)],$$

che mostra che la successione $\{\gamma_k\}$ converge a zero. Quindi, dalla (4.63) si ha che $\lim_{k\to\infty} x^{(k)} = x^*$. ■

Una condizione necessaria e sufficiente per la convergenza è stabilita dal seguente:

Teorema 4.11.4. *Il metodo iterativo*

$$x^{(k+1)} = Cx^{(k)} + d, \quad k = 0, 1, 2, \ldots,$$

è convergente se e solo se

$$\rho(C) < 1 \tag{4.64}$$

dove

$$\rho(C) = \max_{i=1,\ldots,n} |\lambda_i|,$$

con λ_i i-mo autovalore di C, è il **raggio spettrale** *della matrice C.*

Dimostrazione Sia $\rho(C) < 1$. In tale ipotesi la matrice $I - C$ è non singolare (vedi **Teorema B.5** dell'**Appendice B**, **Parte 1**). Sia $\bar{x}$ l'unica soluzione del sistema lineare $x = Cx + d$ e sia

$$e^{(k)} = x^{(k)} - \bar{x}, \ \ \forall k \geq 0.$$

Si ha:

$$e^{(k+1)} = x^{(k+1)} - \bar{x} = Cx^{(k)} + d - (C\bar{x} + d) = C(x^{(k)} - \bar{x}) = Ce^{(k)}.$$

Applicando lo stesso ragionamento a $e^{(k)}$ si ottiene:

$$e^{(k+1)} = C(Ce^{(k-1)}) = C^2e^{(k-1)}.$$

Dopo $k+1$ passi si ha:

$$e^{(k+1)} = C^{k+1}e^{(0)}.$$

Poichè $\rho(C) < 1$, dal **Teorema B.17** dell'**Appendice B**, **Parte 1**, ($\lim_{k\to\infty} A^k v = 0$, $\forall v$, $\Leftrightarrow \rho(A) < 1$) segue che:

$$\lim_{k\to\infty} e^k = 0, \ \ \forall e^{(0)},$$

e, quindi, il metodo (4.49) converge a $\bar{x}$ qualunque sia $x^{(0)}$.

Si supponga ora che il metodo sia convergente. Indicato con $\bar{x}$ il limite, si ricorda che $\bar{x}$ è soluzione del sistema associato al metodo, cioè $\bar{x} = C\bar{x} + d$. Si supponga per assurdo che $\rho(C) \geq 1$. Sia $\tilde{\lambda}$ un autovalore di C tale che $|\tilde{\lambda}| = \rho(C)$ e sia $\tilde{x}$ un autovettore ad esso corrispondente. Posto $x^{(0)} = \bar{x} - \tilde{x}$ e $e^{(k)} = x^{(k)} - \bar{x}$, si ha:

$$e^{(k)} = C^k e^{(0)} = \tilde{\lambda}^k e^{(0)},$$

da cui, poiché $|\tilde{\lambda}| \geq 1$, si ricava che la successione $\{e^{(k)}\}$ non converge a zero, e che quindi la successione $\{x^{(k)}\}$ non converge a $\bar{x}$. Ciò contraddice l'ipotesi. ■

♣ **Esempio 4.23.** Nell'esempio 4.3 si è considerato il sistema lineare:

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

per il quale, come si evince dai risultati riportati nelle Tabelle 4.4 e 4.5, il metodo di Jacobi converge, mentre ciò non accade per il metodo di G-S. Si osservi che, per tale sistema, si ha:

$$\begin{aligned} \rho(C_J) &= 0.1020 \cdot 10^{-4} &< 1 \\ \rho(C_G) &= 4.8284 &> 1 \end{aligned}$$

che, in base al Teorema 4.11.4, conferma i risultati ottenuti. Quindi, si può affermare che, in generale, non esiste un legame tra il comportamento (in termini di convergenza) tra i due metodi.
Si consideri ora il sistema lineare:

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

per il quale si ha:

$$\begin{aligned} \rho(C_J) &= 3.2361 &> 1 \\ \rho(C_G) &= 12.3246 &> 1 \end{aligned}$$

e quindi entrambi i metodi divergono. Si osservi che, considerando la decomposizione (4.55) per la matrice dei coefficienti di tale sistema, gli elementi della matrice $-(L+U)$ sono non negativi.
Si consideri ora il seguente sistema lineare:

$$\begin{cases} 5x_1 & -2x_2 & -1x_3 & = 3 \\ -\ x_1 & +4x_2 & -\ x_3 & = 2 \\ -3x_1 & -2x_2 & +7x_3 & = 2, \end{cases}$$

per il quale si ha:

$$\begin{aligned} \rho(C_J) &= 0.5944 &< 1 \\ \rho(C_G) &= 0.3833 &< 1 \end{aligned}$$

e quindi entrambi i metodi convergono. Si osservi che anche per tale sistema gli elementi della matrice $-(L+U)$ sono non negativi.
Il particolare comportamento (entrambi divergono o convergono) dei metodi di Jacobi e di G-S per gli ultimi due sistemi considerati è stabilito dal seguente:

Teorema 4.11.5. [Teorema di Stein-Rosenberg] *Posto $A = D + L + U$, se la matrice $-(L+U)$ è non negativa, allora è vera una e una soltanto delle condizioni seguenti*

- $\rho(C_J) = \rho(C_G) = 0$
- $\rho(C_J) = \rho(C_G) = 1$
- $0 < \rho(C_G) < \rho(C_J) < 1$
- $1 < \rho(C_J) < \rho(C_G)$

Quindi, in base a tale Teorema, per i sistemi lineari che ne soddisfano l'ipotesi, i metodi di Jacobi e di G-S o sono entrambi convergenti o sono entrambi divergenti. ♣

Per sintetizzare quanto detto finora, anche alla luce dei Teoremi 4.9.1 e 4.11.4, si può concludere che, se valgono le condizioni seguenti:

1. A è non singolare, e quindi il sistema lineare $Ax = b$ ha un'unica soluzione;
2. $C = I - M^{-1}A$, $d = M^{-1}b$, con M matrice di splitting non singolare;
3. $\rho(C) < 1$;

allora, qualunque sia $x^{(0)}$, la successione di vettori $\{x^{(k)}\}$ generata dal metodo (4.49) converge all'unica soluzione di $Ax = b$. Infatti, la condizione 2. assicura che il metodo iterativo (4.49) è consistente, cioè, se converge, converge alla soluzione del sistema $Ax = b$. La condizione 3., invece, assicura che il metodo converge qualunque sia il punto iniziale $x^{(0)}$.

Si osservi che la condizione per la convergenza (4.64) stabilita dal Teorema 4.11.4 non è di verifica facile perchè richiede la conoscenza dell'autovalore massimo della matrice di iterazione C, un problema che in generale è computazionalmente molto dispendioso. D'altra parte, nelle applicazioni, è necessario disporre di condizioni facilmente verificabili. In tale ottica, è importante ricordare la relazione seguente (vedi **Teorema B.12** dell'**Appendice B**, **Parte 1**):

$$\rho(A) \leq \|A\|, \tag{4.65}$$

che vale per ogni $A \in \mathbb{R}^{n\times n}$ e per ogni norma matriciale $\|\cdot\|$ compatibile con una norma vettoriale (cfr. **Definizione B.5** dell'**Appendice B**, nella **Parte 1**).
Si riporta inoltre un risultato, che è utile in alcune applicazioni, il quale stabilisce una condizione sufficiente per la convergenza dei metodi iterativi basati sullo splitting di A, nel caso in cui A presenti particolari caratteristiche.

Teorema 4.11.6. *Se la matrice A del sistema (4.47) è* **simmetrica e definita positiva**[15]*, posto $A = M - R$, con M matrice non singolare, il metodo iterativo (4.53)*

[15]Sistemi lineari con tale caratteristica sono spesso ricorrenti nelle applicazioni. Un esempio è la risoluzione numerica di equazioni differenziali alle derivate parziali di tipo ellittico.

converge se la matrice simmetrica:

$$Q = M + M^T - A,$$

è **definita positiva**.

Dimostrazione Siano B e G le due matrici simmetriche definite positive tali che $A = B^2$ e $Q = G^2$ (l'esistenza e l'unicità di tali matrici è assicurata dal **Teorema B.19** dell'**Appendice B**, **Parte 1**). Considerata la matrice di iterazione del metodo, $C = I - M^{-1}A$, si ponga:

$$H = BCB^{-1} = I - BM^{-1}B,$$

da cui, la matrice HH^T, i cui autovalori sono non negativi perchè semidefinita positiva (vedi **Teoremi B.9** e **B.10** dell'**Appendice B**, **Parte 1**), ha la seguente espressione:

$$\begin{aligned} HH^T &= (I - BM^{-1}B)(I - BM^{-T}B) = I - BM^{-T}B - BM^{-1}B + BM^{-1}AM^{-T}B = \\ &= I - BM^{-1}(M + M^T - A)M^{-T}B = I - T, \end{aligned} \tag{4.66}$$

dove:

$$T = BM^{-1}QM^{-T}B.$$

Poichè $Q = G^2$, la matrice T si scrive anche come:

$$T = (BM^{-1}G)(BM^{-1}G)^T.$$

Poichè B, G, e M sono matrici non singolari, la matrice $BM^{-1}G$ è non singolare. Di conseguenza, la matrice T, essendo il prodotto di una matrice non singolare per la sua trasposta, è simmetrica e definita positiva (**Teorema B.10** dell'**Appendice B**, **Parte 1**) e ha gli autovalori positivi. Dalla (4.66) segue che gli autovalori della matrice HH^T appartengono all'intervallo $[0, 1[$ e, quindi, $\rho(HH^T) < 1$. Se si considera la norma matriciale $\|\cdot\|_{2,B}$ (vedi §**B.3** dell'**Appendice B** della **Parte 1**), si ha:

$$\|C\|_{2,B} = \|BCB^{-1}\|_2 = \|H\|_2 = (\rho(HH^T))^{1/2} < 1.$$

Essendo, dunque, $\|C\| < 1$ ed, in particolare,

$$\rho(C) \leq \|C\| < 1,$$

per il Teorema 4.11.4 il metodo è convergente. ■

In generale, per la risoluzione di sistemi di equazioni lineari:

$$Ax = b, \quad A \in \mathbb{R}^{n\times n},\ x, b \in \mathbb{R}^n, \tag{4.67}$$

si utilizzano processi iterativi lineari, cioè ad esempio metodi stazionari ad un passo del tipo (4.50) in cui la funzione di iterazione G è lineare, $G(x) = Cx + d$, dove C è una matrice $\in \mathbb{R}^{n\times n}$ e d è un vettore $\in \mathbb{R}^n$. Il metodo (4.50) assume quindi la forma:

$$x^{(k+1)} = Cx^{(k)} + d, \quad k = 0, 1, ... \tag{4.68}$$

Da osservare che in tal caso il dominio della funzione G è tutto $\mathbb{R}^n$ e, di conseguenza, il metodo (4.68) è sempre ben definito.

Considerati due punti x e $y \in \mathbb{R}^n$, si ha:

$$\|Cx + d - Cy - d\| = \|C(x - y)\| \leq \|C\|\|x - y\|.$$

Segue che la funzione $G(x) = Cx + d$ è una contrazione se $\|C\| < 1$. In tale ipotesi, in base al **Teorema 4.11.2**, il metodo iterativo (4.68) genera una successione $\{x^{(k)}\}$ che converge ad un punto fisso di G, e quindi, ad una soluzione del sistema lineare $(I - C)x = d$, qualunque sia il punto iniziale $x^{(0)}$. Da osservare, inoltre, che nell'ipotesi che $\|C\| < 1$, la matrice $(I - C)$ è non singolare, e quindi il sistema $(I - C)x = d$ ammette una sola soluzione. Se, infine, la matrice C è scelta in modo tale che il sistema $(I - C)x = d$ sia equivalente al sistema da risolvere (4.67), allora si può concludere che nell'ipotesi che $\|C\| < 1$ il metodo (4.68) converge alla soluzione.

4.12 Velocità di convergenza

Allo scopo di analizzare i fattori che ne determinano la velocità di convergenza, si suppone che il metodo iterativo (4.49) sia consistente e convergente. Posto:

$$e^{(k)} = x^{(k)} - x^*,$$

l'errore di troncamento analitico alla k-ma iterazione del metodo, si osserva (vedi dimostrazione del Teorema 4.11.4 con x^* al posto di $\bar{x}$) che vale la relazione:

$$e^{(k)} = C^k e^{(0)}.$$

Denotata con $\|\cdot\|$ una norma vettoriale ed una norma matriciale tra loro compatibili, si ha:

$$\|e^{(k)}\| \leq \|C^k\|\|e^{(0)}\|, \tag{4.69}$$

da cui discende che $\|C^k\|$ rappresenta una misura di quanto si riduce l'errore dopo k iterazioni.
In generale, si desidera arrestare il metodo iterativo quando:

$$\|e^{(k)}\| \leq \texttt{Tol}\|e^{(0)}\|, \tag{4.70}$$

con $\texttt{Tol} < 1$. In base alla (4.69), per soddisfare la (4.70), è sufficiente determinare un numero intero k tale che:

$$\|C^k\| \leq \texttt{Tol}. \tag{4.71}$$

Assumendo che $\|C\| < 1$ (condizione sufficiente per la convergenza), si ha che $\|C^k\| \leq \|C\|^k < 1$, e, applicando la funzione logaritmo naturale ad entrambi i membri della (4.71) e moltiplicandoli per k, si ha:

$$k \geq -\ln(\texttt{Tol})/(-\frac{1}{k}\ln(\|C^k\|)).$$

Si evince quindi che il numero minimo di iterazioni, k, necessario per ottenere la richiesta riduzione (4.70) è inversamente proporzionale alla quantità:

$$-\frac{1}{k}\ln(\|C^k\|).$$

Ciò conduce alla seguente definizione:

Definizione 4.12.1. (Velocità media di convergenza)
Relativamente al metodo iterativo:

$$x^{(k+1)} = Cx^{(k)} + d, \quad k = 0, 1, 2, \ldots,$$

si definisce **velocità media di convergenza** *per k iterazioni la quantità*

$$R_k(C) = -\frac{1}{k}\ln(\|C^k\|).$$

Si osservi che il valore di $R_k(C)$ dipende dalla norma matriciale che si utilizza.
Un'altra definizione di velocità di convergenza, che non presenta tale dipendenza, si ottiene dalla relazione:

$$\lim_{k\to\infty} \|C^k\|^{1/k} = \rho(C),$$

che vale se $\rho(C) < 1$. Si ha dunque la seguente:

Definizione 4.12.2. (Velocità asintotica di convergenza)
Relativamente al metodo iterativo:

$$x^{(k+1)} = Cx^{(k)} + d, \quad k = 0, 1, 2, \ldots,$$

si definisce **velocità asintotica di convergenza**, *o più semplicemente* **velocità di convergenza**, *il numero:*

$$R_\infty(C) = \lim_{k\to\infty} R_k(C) = -\ln(\rho(C)).$$

Il valore di $R_\infty(C)$ è determinato solo dal raggio spettrale della matrice di iterazione e, quindi, è indipendente dalla norma che si utilizza. Inoltre, quanto più piccolo è $\rho(C)$ tanto più grande è $R_\infty(C)$, e quindi tanto più velocemente il metodo converge alla soluzione.
Sulla base della definizione 4.12.2, si ha che una stima del minimo numero di iterazioni necessarie per ridurre l'errore di un fattore `Tol` è data da

$$k \geq \frac{-\ln(\texttt{Tol})}{R_\infty(C)}. \tag{4.72}$$

♣ **Esempio 4.24.** Si considerino i sistemi lineari (a) e (b) dell'esempio 4.14. Per la matrice di iterazione del metodo di Jacobi applicato a tali sistemi si ha:

$$\begin{aligned} &\text{sistema (a)}: \quad \rho(C_J) = 0.7637; \\ &\text{sistema (b)}: \quad \rho(C_J) = 0.3333. \end{aligned}$$

Quindi, il metodo di Jacobi converge più velocemente per il sistema (b). Ciò è confermato dai risultati riportati nelle Tabelle 4.8 e 4.9: il numero di iterazioni effettuate per rendere la norma del massimo dell'errore minore di $\texttt{Tol} = 10^{-6}$ è 48 per il sistema (a) e 16 per il sistema (b).
Posto $\texttt{Tol} = 10^{-6}$ nella (4.72), si ha:

$$\begin{aligned} &\text{sistema (a)}: k \simeq 51; \\ &\text{sistema (b)}: k \simeq 13, \end{aligned}$$

da cui si evince che la (4.72) fornisce una stima attendibile "a priori" del numero di iterazioni necessarie per ottenere la fissata riduzione dell'errore. ♣

♣ **Esempio 4.25.** In relazione al sistema (a) dell'esempio 4.14, per la matrice di iterazione del metodo di G-S si ha $\rho(C_G) = 0.5833$. Quindi, utilizzando la (4.72), si ha che una stima del numero di iterazioni necessarie per ridurre l'errore di $\texttt{Tol} = 10^{-6}$ è $k = 26$. L'attendibilità di tale stima è confermata dai risultati riportati nella Tabella 4.8. Inoltre, come descritto nell'esempio precedente, per la matrice di iterazione del metodo di Jacobi applicato al sistema (a) si ha $\rho(C_J) = 0.7637$ e, quindi, per tale sistema il metodo di G-S converge più velocemente, come confermano di nuovo i risultati riportati nella Tabella 4.8.

♣

Si osservi che, nonostante i risultati mostrati negli esempi precedenti, per alcuni sistemi lineari la stima (4.72) può fornire un valore che risulta sensibilmente inferiore al numero effettivo di iterazioni necessarie. L'utilizzo della seguente stima:

$$k \geq \frac{-\ln(\texttt{Tol})}{R_k(C)},$$

in cui si considera la velocità media di convergenza, può fornire un valore più attendibile.

Osservazione 4.3. *Come illustrato nel §4.12, la velocità di convergenza di un metodo iterativo dipende dalle proprietà spettrali della matrice dei coefficienti; di qui il tentativo di trasformare il sistema lineare in esame in uno equivalente, che abbia, cioè, la stessa soluzione, ma presenti proprietà spettrali più vantaggiose ai fini della convergenza del*

metodo. La matrice che determina tale trasformazione prende il nome di **precondizionatore**. *Ad esempio, se una matrice M approssima, in qualche modo, la matrice dei coefficienti, A, il sistema trasformato*

$$M^{-1}Ax = M^{-1}b$$

ha la stessa soluzione del sistema originale, $Ax = b$, ma le proprietà spettrali della sua matrice dei coefficienti, $M^{-1}A$, potrebbero garantire una più rapida convergenza del metodo iterativo. In generale, comunque, un buon precondizionatore deve migliorare la velocità di convergenza del metodo, ma tanto da bilanciare il costo computazionale richiesto per la sua costruzione e la sua applicazione. Cioè, un efficiente precondizionatore M, deve poter avere le seguenti caratteristiche:

- *M è una buona approssimazione di A in qualche senso;*
- *il costo computazionale della costruzione di M non deve essere proibitivo tanto da annulare i vantaggi dell'aumentata velocità di convergenza;*
- *la matrice M deve essere facile da invertire nel senso che il sistema $My = c$ sia più facile da risolvere del sistema $Ax = b$;*

La individuazione di un opportuno precondizionatore per una data matrice è un attivo ed attuale campo di ricerca. Noi ci limitiamo ad introdurre alcuni classici metodi per l'accelerazione della convergenza.

4.13 Accelerazione della convergenza

4.13.1 Metodi di rilassamento: il metodo SOR

L'analisi effettuata nel paragrafo precedente ha mostrato che la convergenza di un metodo iterativo del tipo (4.49) è tanto più rapida quanto più piccolo è il raggio spettrale della matrice di iterazione del metodo. Di conseguenza, una delle tecniche più naturali per accelerare la convergenza è apportare modifiche al metodo in modo da rendere il più piccolo possibile $\rho(C)$. In generale tale problema é di non facile soluzione: lo si risolve solo in qualche caso particolare, ma rilevante per le applicazioni.
Come primo passo in tale direzione, si riscrive il metodo di G-S [16] :

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, \cdots, n,$$

nella forma (aggiungendo e sottraendo $a_{ii}x_i^{(k)}$):

[16] consideriamo il metodo di G-S perché quando converge, converge piú velocemente del metodo di Jacobi (Teorema 4.11.5).

$$x_i^{(k+1)} = x_i^{(k)} + r_i^{(k)}, \quad i = 1, \cdots, n, \tag{4.73}$$

dove:

$$r_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1, \cdots, n.$$

Definizione 4.13.1. (Metodo SOR)
Considerato un numero reale ω, si modificano le (4.73) come segue:

$$x_i^{(k+1)} = x_i^{(k)} + \omega r_i^{(k)}, \quad i = 1, \cdots, n. \tag{4.74}$$

Le formule (4.74) definiscono un metodo iterativo detto **metodo SOR** *(***S***uccessive* **O***ver* **R***elaxation) ed ω è detto* **parametro di rilassamento**.

Si puó dedurre anche per un metodo SOR una formulazione matriciale. Si ha la seguente:

Proposizione 4.13.1. (Formulazione matriciale del Metodo SOR)
Un metodo SOR é un metodo che deriva da uno splitting di A del tipo (4.52), dove, in particolare:

$$M_\omega = \omega^{-1} D + L; \quad R_\omega = (\omega^{-1} - 1) D - U \ ,$$

con L, D e U definite tramite la decomposizione (4.55).

Dimostrazione Le (4.74) possono essere scritte come:

$$x_i^{(k+1)} + \omega \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k+1)} = (1-\omega) x_i^{(k)} - \omega \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \omega \frac{b_i}{a_{ii}}, \quad i = 1, \cdots, n,$$

da cui si ha:

$$[(I + \omega D^{-1} L) x^{(k+1)}]_i = [(1-\omega) x^{(k)}]_i - [\omega D^{-1} U x^{(k)}]_i + \omega [D^{-1} b]_i \ , \quad i = 1, \cdots, n,$$

e, quindi, in forma matriciale:

$$x^{(k+1)} = (I + \omega D^{-1} L)^{-1} [(1-\omega) I - \omega D^{-1} U] x^{(k)} + (I + \omega D^{-1} L)^{-1} \omega D^{-1} b. \tag{4.75}$$

Osservando che:

$$(I + \omega D^{-1} L) = \omega D^{-1} (\omega^{-1} D + L)$$

e

$$(1-\omega) I - \omega D^{-1} U = \omega D^{-1} [(1-\omega) \omega^{-1} D - U],$$

la (4.75) diventa:

$$x^{(k+1)} = \overbrace{(\omega^{-1} D + L)^{-1}}^{M_\omega^{-1}} \overbrace{[(\omega^{-1} - 1) D - U]}^{R_\omega} x^{(k)} + \overbrace{(\omega^{-1} D + L)^{-1} b}^{M_\omega^{-1} b} \tag{4.76}$$

e quindi la tesi. ■

È immediato verificare che, se $\omega \neq 0$, la matrice di splitting M_ω è non singolare e quindi il metodo SOR è consistente. Inoltre, nel caso $\omega = 1$, il metodo SOR si riduce al metodo di G-S. In conclusione, in forma (4.49) il metodo SOR ha la seguente espressione:

METODO SOR

$$x^{(k+1)} = C_\omega x^{(k)} + d_\omega \quad , \quad C_\omega = M_\omega^{-1} R_\omega \quad , \quad M_\omega = (\omega^{-1} D + L)$$
$$d_\omega = M_\omega^{-1} b \quad , \quad R_\omega = (\omega^{-1} - 1)D - U \quad . \tag{4.77}$$

Per la convergenza del metodo SOR è fondamentale il risultato seguente:

Teorema 4.13.1. *Per la matrice di iterazione del metodo SOR ((4.74)) si ha:*

$$\rho(C_\omega) \geq |\omega - 1|.$$

Dimostrazione Considerate le matrici non singolari $E = D^{-1}L$ e $F = D^{-1}U$, la matrice di iterazione del metodo SOR può scriversi come:

$$C_\omega = (I + \omega E)^{-1}[(1 - \omega)I - \omega F].$$

Poichè $(I + \omega E)$ è un matrice triangolare inferiore con elementi diagonali uguali a 1, lo è anche la sua inversa che ha, quindi, determinante uguale a 1. La matrice $[(1 - \omega)I - \omega F]$ è una matrice triangolare superiore con elementi diagonali tutti uguali a $(1-\omega)$, il cui determinante è quindi $(1-\omega)^n$. Da quanto detto, il determinante di C_ω, $\mathtt{det}(C_\omega)$, è:

$$\mathtt{det}(C_\omega) = \mathtt{det}((I + \omega E)^{-1}) \cdot \mathtt{det}([(1 - \omega)I - \omega F]) = (1 - \omega)^n.$$

Poichè il prodotto degli autovalori di una matrice è uguale al suo determinante, si ha:

$$\rho(C_\omega) \geq (|\omega - 1|^n)^{1/n} = |\omega - 1|$$

e quindi la tesi. ■

Dal risultato del Teorema precedente, si evince che la condizione $0 < \omega < 2$ è necessaria per la convergenza del metodo SOR. Se la matrice A del sistema lineare da risolvere è simmetrica definita positiva si ha il seguente importante risultato, che è una conseguenza dei Teoremi 4.11.6 e 4.13.1:

Corollario 4.13.1. *Se A è* **simmetrica e definita positiva**, *il metodo SOR è convergente se e solo se*

$$0 < \omega < 2.$$

Dimostrazione Che la condizione sia necessaria discende dal Teorema 4.13.1. Si supponga ora che $0 < \omega < 2$. Ricordando che la matrice di splitting del metodo SOR è $M_\omega = \omega^{-1}D + L$, si ha, essendo A simmetrica, per cui è $L^T = U$, che la matrice $Q = M_\omega + M_\omega^T - A = (2\omega^{-1} - 1)D$ è una matrice diagonale, i cui elementi sono dati da $(2\omega^{-1}-1)a_{ii}$, $i = 1, \ldots, n$, dove a_{ii}, $i = 1, \ldots, n$, sono gli elementi diagonali di A. Poichè A è definita positiva, i suoi elementi diagonali sono tutti positivi; di conseguenza, anche la matrice Q, poichè $0 < \omega < 2$, ha tutti gli elementi positivi e, quindi, è definita positiva. In base al Teorema 4.11.6, si ha che il metodo SOR converge. ■

Il problema cruciale relativo all'utilizzo del metodo SOR è la scelta del parametro di rilassamento ω; tale scelta determina la velocità di convergenza del metodo e, quindi, la sua maggiore efficacia rispetto ai metodi di Jacobi e di G-S. Si consideri la seguente:

Definizione 4.13.2. (Valore ottimale)
Relativamente al metodo SOR ((4.74)), si definisce **valore ottimale di** ω *il valore* ω_{opt} *tale che:*

$$\rho(C_{\omega_{opt}}) = \min_{\omega} \rho(C_\omega).$$

In generale, la determinazione del valore ottimale del parametro di rilassamento del metodo SOR è un problema di non facile soluzione. Tuttavia, per alcune importanti classi di sistemi lineari il valore di ω_{opt} è noto o può essere stimato con sufficiente accuratezza.

♣ **Esempio 4.26.** Si consideri il seguente problema al contorno per l'equazione di Poisson:

$$\begin{cases} -u_{xx} - u_{yy} = f(x,y) & (x,y) \in \Omega =]0,1[\times]0,1[\\ u(x,y) = g(x,y) & (x,y) \in \delta\Omega \end{cases} \tag{4.78}$$

dove $\delta\Omega$ è la frontiera del quadrato unitario, $f(x,y)$ e $g(x,y)$ sono funzioni continue rispettivamante su Ω e $\delta\Omega$. La soluzione numerica di tale problema la si può ottenere mediante il *metodo delle differenze finite* utilizzando uno schema a 5 punti. Più precisamente si considera una reticolazione del dominio quadrato $\Omega \cup \delta\Omega$ in quadratini di lato $h = 1/(m+1)$ e si vuole calcolare un'approssimazione dei valori che la funzione incognita $u(x,y)$ assume nei punti interni di tale reticolazione (quelli sui punti di frontiera sono noti, nota la g, per la condizione imposta nella (4.78)), cioè nei punti:

$$(x_i, y_j), \;\; x_i = i \cdot h, \; y_j = j \cdot h, \; i,j = 1, \ldots, m.$$

A tal fine, nei suddetti punti si approssimano le derivate parziali seconde con le usuali differenze centrali in modo che il problema di partenza (4.78) sia approssimato dal problema discreto:

$$\frac{1}{h^2}(4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j+1} - u_{i,j-1}) = f_{i,j}, \; i,j = 1, \ldots, m, \tag{4.79}$$

dove $u_{i,j}$ e $f_{i,j}$ rappresentano i valori approssimati rispettivamente di $u(x_i, y_j)$ e $f(x_i, y_j)$. Numerando i punti interni della reticolazione (a partire da 1) da sinistra a destra e dal basso verso l'alto e portando a termine noto i valori sui punti della frontiera, posto

$$\mathbf{u}^T = (u_{11}, u_{12}, \ldots, u_{1m}, u_{21}, u_{22}, \ldots, u_{2m}, \ldots, u_{m1}, u_{m2}, \ldots, u_{mm}),$$

le (4.79) costituiscono un sistema di n equazioni lineari in n incognite, con $n = m^2$:

$$Au = f, \tag{4.80}$$

la cui soluzione rappresenta i valori approssimati della funzione incognita $u(x, y)$ nei punti interni della reticolazione. La matrice dei coefficienti ha la seguente struttura:

$$A = \begin{pmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -I & T \end{pmatrix},$$

dove I è la matrice identità di ordine m e T è la matrice tridiagonale di ordine m:

$$T = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}.$$

La matrice A è quindi una T-matrice[17] simmetrica, in cui le matrici diagonali, T, sono non singolari. Inoltre, posto:

$$t = \frac{\pi}{m+1} = \pi h, \tag{4.81}$$

17

Definizione 4.13.3. (Matrice tridiagonale a blocchi)
Si definisce **matrice tridiagonale a blocchi** *(nel seguito chiamata* T**-matrice**$)$ *una matrice* $A \in \mathbb{R}^{n\times n}$ *del tipo:*

$$A = \begin{pmatrix} D_1 & F_1 & & & \\ E_2 & D_2 & F_2 & & \\ & \ddots & \ddots & \ddots & \\ & & E_{p-1} & D_{p-1} & F_{p-1} \\ & & & E_p & D_p \end{pmatrix},$$

dove D_i, E_i, F_i *sono matrici di dimensioni ripsettive* $n_i \times n_i$, $n_i \times n_{i-1}$ *e* $n_i \times n_{i+1}$ *con* $\sum_{i=1}^{p} n_i = n$.

Per le T-matrici valgono i seguenti risultati.

Teorema 4.13.2. *Dato un sistema lineare* $Ax = b$, *in cui* A *è una* T*-matrice, con matrici diagonali* D_i *non singolari, si ha:*

$$\rho(C_{GS}) = \rho^2(C_J),$$

e quindi:

$$R_\infty(C_{GS}) = 2R_\infty(C_J).$$

Sulla base del Teorema precedente, si ha che, per sistemi lineari in cui la matrice dei coefficienti è una T-matrice, se il metodo di Jacobi converge, allora converge anche il metodo di G-S con una velocità asintotica doppia.

è noto che gli autovalori della matrice A sono dati da:

$$\lambda_{i,j} = 4\left(\sin^2 i\frac{t}{2} + \sin^2 j\frac{t}{2}\right), \quad i,j = 1,\ldots,m.$$

Dalla conoscenza degli autovalori di A si determinano facilmente anche quelli della matrice di iterazione del metodo di Jacobi applicato al sistema (4.80), che nel caso specifico è data da $C_J = I - D^{-1}A = I - \frac{1}{4}A$. Gli autovalori di C_J sono quindi:

$$\mu_{i,j} = 1 - \frac{1}{4}\lambda_{i,j} = \frac{1}{2}(\cos it + \cos jt), \quad i,j = 1,...,m,$$

da cui, per il raggio spettrale di C_J si ha:

$$\rho(C_J) = \max_{i,j}|\mu_{i,j}| = \cos t < 1. \tag{4.82}$$

In conclusione, per il sistema (4.80) sono verificate tutte le ipotesi del Teorema 4.13.3, per cui il valore ottimale del parametro di rilassamento relativo al metodo SOR è:

$$\omega_{opt} = \frac{2}{1 + (1 - \cos^2 t)^{1/2}} = \frac{2}{1 + \sin t}. \tag{4.83}$$

Come esempio di applicazione del risultato ottenuto, si consideri, in particolare, il caso in cui, in (4.78), si abbia:

$$f(x,y) = 0; \quad g(x,y) = x + y. \tag{4.84}$$

Con facili calcoli si verifica che in tali ipotesi e per $m = 2$ la matrice A ed il vettore f del sistema (4.80) sono:

$$A = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}; \quad f = \begin{pmatrix} 2/3 \\ 2 \\ 2 \\ 10/3 \end{pmatrix}. \tag{4.85}$$

La soluzione di tale sistema è $(2/3, 1, 1, 4/3)$. Inoltre, dalla (4.83) con $t = \frac{\pi}{3}$ si ha che:

$$\omega_{opt} = 1.0718$$

Nella Tabella 4.11 sono riportati i risultati ottenuti risolvendo il sistema (4.80), con A e f dati dalle (4.85), rispettivamente con il metodo di Jacobi, il metodo di G-S ($\omega = 1$) e il metodo SOR con $\omega = \omega_{opt} = 1.0718$. In particolare sono state utilizzate le **Procedure 4.3** e **4.4**, l'ultima opportunamente modificata per introdurre il parametro di rilassamento. Inoltre, si è posto `Tol`=10^{-6} e $x_i^{(0)} = 0$, $i = 1,...,4$. Si osservi che, come si aspettava, la velocità di convergenza del metodo di G-S è quasi il doppio di quella del metodo di Jacobi. Inoltre, il numero di iterazioni (`IT`) richieste dal metodo SOR per soddisfare la tolleranza è inferiore rispetto a quelle richieste dal metodo di G-S.

Teorema 4.13.3. *Dato un sistema lineare $Ax = b$, in cui A è una T-matrice, con matrici diagonali D_i non singolari, se tutti gli autovalori della matrice di iterazione del metodo di Jacobi, C_J, sono reali e se $\rho(C_J) < 1$, allora il metodo SOR converge per $0 < \omega < 2$. Inoltre si ha:*

1) $\omega_{opt} = \frac{2}{1+(1-\rho^2(C_J))^{1/2}}$
2) $\rho(C_{\omega_{opt}}) = \omega_{opt} - 1.$

Metodo	IT	x_1	x_2	x_3	x_4	ERR
Jacobi	20	0.666674	0.999999	0.999999	1.33332	$0.71{\cdot}10^{-6}$
G-S	12	0.666674	0.999999	0.999999	1.33332	$0.49{\cdot}10^{-6}$
SOR	8	0.666674	1.000000	1.000000	1.33333	$0.76{\cdot}10^{-6}$

Tabella 4.11: Schema riassuntivo relativo alla risoluzione del sistema (4.80), con A e f dati dalle (4.85), con i metodi di Jacobi, G-S e SOR ($\omega = 1.0718$).

È da osservare che il miglioramento in velocità di convergenza, usando $\omega = \omega_{opt}$ nel metodo SOR, aumenta al crescere di m e, quindi, della dimensione del sistema. Infatti, per m abbastanza grande, e quindi, dalla (4.81), per $t << 1$, si ha che una stima accurata del raggio spettrale della matrice di iterazione del metodo SOR è data da:

$$\rho(C_{\omega_{opt}}) = \omega_{opt} - 1 = \frac{2}{1+\sin t} - 1 \simeq 1 - 2t.$$

Di conseguenza, una stima della velocità di convergenza del metodo SOR con $\omega = \omega_{opt}$ è:

$$R_{\omega_{opt}} \simeq -\ln(1-2t) \simeq 2t. \tag{4.86}$$

Per il metodo di G-S si ha:

$$\rho(C_G) = \rho^2(C_J) = \cos^2 t \simeq 1 - t^2,$$

da cui la velocità di convergenza del metodo di G-S è:

$$R_G \simeq -\ln(1-t^2) \simeq t^2.$$

Confrontando le velocità di convergenza dei due metodi, si ha che, per m abbastanza grande, il metodo SOR con valore ottimale del parametro di rilassamento richiede, rispetto al metodo di G-S, un numero di iterazioni minore di un fattore dato da:

$$\frac{R_{\omega_{opt}}}{R_G} \simeq \frac{2}{t} \simeq \frac{2m}{\pi}.$$

Tale fattore è proporzionale a m, cioè aumenta al crescere di m. Ad esempio, se $m = 5$ si ha:

$$\frac{R_{\omega_{opt}}}{R_G} \simeq 3,$$

mentre se $m = 10$ si ha:

$$\frac{R_{\omega_{opt}}}{R_G} \simeq 6\ .$$

In pratica, ciò significa che, se $m = 5$, il metodo SOR con parametro ottimale è circa 3 volte più veloce del metodo di G-S, mentre, se $m = 10$, circa 6. In Tabella 4.12 è riportato, al variare di m da 2 a 10, il numero di iterazioni richiesto dai metodi di Jacobi, G-S e SOR con parametro ottimale, per risolvere il sistema (4.80) derivante dal problema differenziale (4.78), con le funzioni f e g date dalle (4.84) ed utilizzando Tol=10^{-6}. I risultati ottenuti confermano le stime teoriche sul guadagno, in termini di velocità di convergenza, che si ha utilizzando il metodo SOR con il valore ottimale del parametro di rilassamento.

♣

m	n	IT (Jacobi)	IT (G-S)	IT (SOR)
2	4	20	12	8
3	9	38	21	12
5	25	84	45	18
7	49	142	77	24
9	81	214	116	30
10	100	254	138	32

Tabella 4.12: Risultati relativi alla risoluzione del sistema (4.80) al variare di m, utilizzando i metodi di Jacobi, G-S e SOR con parametro ottimale.

È importante osservare ancora che l'utilizzazione pratica dell'espressione analitica del valore ottimale del parametro di rilassamento fornita dal Teorema 4.13.3, dipende, tuttavia, dalla possibilità di disporre di una stima del raggio spettrale della matrice di iterazione del metodo di Jacobi. Tale stima, inoltre, a parte qualche caso particolare come quello illustrato nell'esempio 4.26, non è nota, e spesso calcolarla è computazionalmente molto dispendioso[18].

4.13.2 Accelerazione polinomiale

In questo paragrafo si introduce un'altra tecnica generale per accelerare la convergenza dei metodi iterativi del tipo (4.49). Tale tecnica, nota con il nome di **procedura di accelerazione polinomiale**, consiste nel costruire una nuova sequenza di vettori mediante combinazioni lineari dei vettori ottenuti con un fissato metodo iterativo, che è definito **metodo iterativo base** della procedura. Consideriamo il seguente problema.

Problema: formulazione (I)
Sia $\{x^{(k)}\}$ la successione di vettori generati, a partire da un vettore iniziale $x^{(0)}$, da un metodo base:

$$x^{(k+1)} = Cx^{(k)} + d, \quad k \geq 0. \tag{4.87}$$

[18]Per tale motivo, la maggior parte delle principali librerie di software matematico, che contengono routine per la risoluzione di sistemi lineari con il metodo SOR, prevedono procedure *adattative* per il calcolo di ω_{opt}, cioè algoritmi numerici che determinino dinamicamente, ad ogni iterazione, una stima di ω_{opt}.

Si consideri la nuova sequenza di vettori $\{y^{(k)}\}$ *definiti dalle combinazioni lineari:*

$$y^{(k)} = \sum_{j=0}^{k} \alpha_{jk} x^{(j)}, \quad \alpha_{jk} \in \mathbb{R}, \quad k \geq 0. \tag{4.88}$$

Si vogliono determinare i coefficienti α_{jk} *in modo che* $y^{(k)}$ *sia un'approssimazione più accurata della soluzione del sistema rispetto a* $x^{(k)}$.

Indicata con x^* la soluzione del sistema $Ax = b$, si osserva, innanzitutto, che se $x^{(0)} = x^{(1)} = ... = x^{(k)} = x^*$, allora deve risultare $y^{(k)} = x^*$. Sui coefficienti α_{jk} si impone quindi il seguente vincolo:

$$\sum_{j=0}^{k} \alpha_{jk} = 1, \quad \forall k \geq 0. \tag{4.89}$$

Tenendo conto della (4.89), il problema puó essere cosí riformulato

Problema: formulazione (II)
Indicati con

$$e^{(k)} = x^{(k)} - x^* \quad ed \quad \tilde{e}^{(k)} = y^{(k)} - x^*,$$

gli errori associati rispettivamente a $x^{(k)}$ *e ad* $y^{(k)}$, *per le (4.88) e (4.89) si ha:*

$$\tilde{e}^{(k)} = \sum_{j=0}^{k} \alpha_{jk} x^{(j)} - x^* = \sum_{j=0}^{k} \alpha_{jk} (x^{(j)} - x^*) = \sum_{j=0}^{k} \alpha_{jk} e^{(j)}. \tag{4.90}$$

Ricordando che:

$$e^{(j)} = C^j e^{(0)}, \quad j \geq 0,$$

con $e^{(0)} = x^{(0)} - x^*$, *dalla (4.90) si ha:*

$$\tilde{e}^{(k)} = \left(\sum_{j=0}^{k} \alpha_{jk} C^j \right) e^{(0)}, \quad \forall k \geq 0, \tag{4.91}$$

dove $\tilde{e}^{(0)} = e^{(0)}$. *Introdotto il polinomio di grado* k *nella variabile reale* z:

$$p_k(z) = \alpha_{0,k} + \alpha_{1,k} z + ... + \alpha_{k,k} z^k, \quad con \quad p_k(1) = 1,$$

la (4.91) può essere riscritta nella forma:

$$\tilde{e}^{(k)} = p_k(C) \tilde{e}^{(0)}, \quad \forall k \geq 0, \tag{4.92}$$

dove $p_k(C) \in \mathbb{R}^{n \times n}$ *ha la seguente espressione:*

$$p_k(C) = \alpha_{0,k} I + \alpha_{1,k} C + ... + \alpha_{k,k} C^k \ .$$

Il problema fondamentale legato all'utilizzo della procedura di accelerazione descritta è quello di trovare una successione di polinomi

$$\{p_k(z) \ : \ p_k(1) = 1\}$$

in modo da rendere minimo l'errore $\tilde{e}^{(k)}$, $\forall k \geq 0$. *Considerando la norma euclidea, dalla (4.92) si ha:*

$$\|\tilde{e}^{(k)}\|_2 \leq \|p_k(C)\|_2 \|\tilde{e}^{(0)}\|_2, \quad \forall k \geq 0,$$

e, quindi, per rendere piccola la norma dell'errore è necessario determinare una successione di polinomi $\{p_k(z) \ : \ p_k(1) = 1\}$ *tali che* $\|p_k(C)\|_2$ *sia piccola. Ciò conduce al seguente problema:*

$$\mathbf{P2}: \quad min_{p_k(1)=1} \|p_k(C)\|_2, \ \ \forall k \geq 0. \tag{4.93}$$

Si analizza nel prossimo paragrafo una possibile soluzione di tale problema.

4.13.3 Accelerazione polinomiale di Chebyshev

Nel caso in cui la matrice C sia simmetrica si ha una ulteriore formulazione del problema.

Problema: formulazione (III)

Se la matrice di iterazione C *del metodo base (4.87) della procedura é simmetrica, i suoi autovalori* $\{\lambda_i\}_{i=1}^n$ *sono reali e, nell'ipotesi che il metodo base converga* ($\rho(C) < 1$*), si ha:*

$$-1 < a = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n = b < 1, \tag{4.94}$$

con a *e* b*, rispettivamente, il più piccolo ed il più grande tra gli autovalori di* C*. Si osservi che* $\{p_k(\lambda_i)\}_{i=1}^n$ *sono gli autovalori della matrice* $p_k(C)$ *(vedi* **Teorema B.18** *dell'***Appendice B***, della* **Parte 1***). Inoltre, dal fatto che* C *è simmetrica discende che lo è anche* $p_k(C)$*. Si ha quindi:*

$$\|p_k(C)\|_2 = \rho(p_k(C)) = \max_{1 \leq i \leq n} |p_k(\lambda_i)|.$$

Dalla (4.94) si ha:

$$\|p_k(C)\|_2 = \max_{1 \leq i \leq n} |p_k(\lambda_i)| \leq \max_{a \leq z \leq b} |p_k(z)|.$$

Di conseguenza, il problema **P2** *è ricondotto ad un nuovo problema: la determinazione di una successione di polinomi* $\{p_k(z) \ : \ p_k(1) = 1\}$ *che risolvano:*

$$\mathbf{P3}: \quad \min_{p_k(1)=1} \left\{ \max_{a \leq z \leq b} |p_k(z)| \right\}, \ \ \forall k \geq 0. \tag{4.95}$$

La soluzione del problema è unica ed è definita in termini dei **polinomi di Chebyshev**, come mostra il Teorema che segue, la cui dimostrazione (come anche le principali proprietà dei polinomi di Chebyshev) è riportata in Appendice (Teorema C.2.2).

Teorema 4.13.4. *Sia $T_k(z)$ il polinomio di Chebyshev di prima specie di grado k. Si consideri la funzione $w : [a,b] \longrightarrow [-1,1]$ così definita:*

$$w(z) = \frac{2z-(b+a)}{b-a}.$$

Sia $\mathcal{P}_k$ l'insieme dei polinomi $p_k(z)$ di grado k nella variabile reale z tali che $p_k(1) = 1$. Il polinomio $H_k(z)$ definito da:

$$H_k(z) = \frac{T_k(w(z))}{T_k(w(1))}, \quad \forall k \geq 0, \tag{4.96}$$

è l'unico polinomio $\in \mathcal{P}_k$ che risolve il problema (4.95).

Il metodo iterativo base (4.87) e le (4.88), dove i coefficienti α_{jk} sono quelli dei polinomi (4.96), rappresentano la **procedura di accelerazione di Chebyshev**. Affinchè tale procedura costituisca un metodo effettivamente utilizzabile, è necessario tuttavia determinare una formula più efficiente della (4.88) per calcolare i vettori $y^{(k)}$; in tale formula, infatti, $y^{(k)}$ si ottiene a partire dai $k+1$ vettori $x^{(j)}$, $j = 0, ..., k$. Pertanto, l'uso della (4.88) comporta un'elevata complessità sia di tempo che di spazio quando k è grande. Sfruttando la nota relazione di ricorrenza a tre termini dei polinomi di Chebyshev (Teorema C.2.1 dell'Appendice) si può ottenere una formula più conveniente della (4.88) da un punto di vista computazionale. Si ha infatti il seguente Teorema, la cui dimostrazione è riportata in Appendice (Teorema C.2.3):

Teorema 4.13.5. *I polinomi definiti in (4.96) soddisfano la relazione di ricorrenza:*

$$\begin{cases} H_0(z) = 1, \quad H_1(z) = \gamma z + 1 - \gamma \ ; \\ H_{k+1}(z) = \beta_{k+1}(\gamma z + 1 - \gamma)H_k(z) + (1-\beta_{k+1})H_{k-1}(z), \quad k \geq 1, \end{cases} \tag{4.97}$$

dove:

$$\begin{cases} \gamma = 2/(2-(b+a)) \ ; \\ \beta_{k+1} = 2w(1)T_{(k)}(w(1))/T_{(k+1)}(w(1)) \ . \end{cases} \tag{4.98}$$

Dalla relazione (4.97) stabilita dal Teorema precedente segue che anche i valori $y^{(k)}$ della procedura di accelerazione basata sui polinomi $H_{(k)}(z)$ possono essere ottenuti mediante una formula di ricorrenza a tre termini. Si ha infatti il seguente:

Corollario 4.13.2. *I vettori $y^{(k)}$ della procedura di accelerazione di Chebyshev si possono calcolare, a partire da un arbitrario vettore $y^{(0)}$, mediante la formula:*

$$\begin{cases} y^{(1)} = \gamma(Cy^{(0)} + d) + (1-\gamma)y^{(0)} \ ; \\ y^{(k+1)} = \beta_{k+1}[\gamma(Cy^{(k)} + d) + (1-\gamma)y^{(k)}] + (1-\beta_{k+1})y^{(k-1)} \ . \end{cases} \tag{4.99}$$

Dimostrazione Ricordando la (4.88), $y^{(k)} = \sum_{j=0}^{k} \alpha_{jk} x^{(j)}$, si ricava che $y^{(0)} = x^{(0)}$. Inoltre, dalle (4.97) si ha $H_1(x(1)) = \gamma x^{(1)} + 1 - \gamma$. Ma, d'altra parte, $H_1(x(1)) = \alpha_{10} + \alpha_{11} x^{(1)}$, da cui si ha $\alpha_{11} = \gamma$ e $\alpha_{01} = 1 - \gamma$. Poichè $y^{(1)} = \alpha_{01} x^{(0)} + \alpha_{11} x^{(1)}$, e tenuto conto che $x^{(1)} = Cx^{(0)} + d$, si ha la prima uguaglianza delle (4.99).
Si consideri ora l'errore associato al vettore $y^{(k)}$ in (4.88). Dalla (4.92) e dalla (4.97) si ha:

$$\tilde{e}^{(k+1)}(z) = \{\beta_{k+1}[\gamma C + (1-\gamma)I]H_k(C) + (1-\beta_{k+1})H_{k-1}(C)\}\tilde{e}^{(0)},$$

da cui, utilizzando ancora la (4.92), si ha:

$$\tilde{e}^{(k+1)}(z) = \beta_{k+1}[\gamma C + (1-\gamma)I]\tilde{e}^{(k)} + (1-\beta_{k+1})\tilde{e}^{(k-1)}. \tag{4.100}$$

Aggiungendo x^* ad entrambi i termini della (4.100) si ottiene:

$$y^{(k+1)}(z) = \beta_{k+1}[\gamma C + (1-\gamma)I]y^{(k)} + (1-\beta_{k+1})y^{(k-1)} - \beta_{k+1}\gamma(C-I)x^*. \tag{4.101}$$

Poichè x^* è soluzione del sistema associato al metodo di base (4.87), cioè $x^* = Cx^* + d$, si ha $(C-I)x^* = -d$, per cui dalla (4.101) si ottiene la relazione (4.99). ■

Osservazione 4.4. *I coefficienti β_{k+1} della relazione di ricorrenza (4.99) sono determinati dalla formula (4.98). Si osserva, tuttavia, che tali coefficienti possono essere scritti nella forma seguente, più vantaggiosa da un punto di vista computazionale:*

$$\beta_1 = 1,\ \beta_2 = (1 - \tfrac{1}{2}\sigma^2)^{-1}$$
$$\beta_{k+1} = (1 - \tfrac{1}{4}\sigma^2\beta_k)^{-1},\ \ k \geq 2,$$

con:

$$\sigma = 1/(w(1)) = (b-a)/(2-(b+a)).$$

La relazione (4.99) mostra che non è necessario utilizzare un vettore ausiliario per memorizzare i vettori $x^{(k)}$ del metodo base (4.87). Infatti, anche se l'idea di partenza della procedura di accelerazione è quella di ottenere i vettori $y^{(k)}$ come combinazioni dei vettori $x^{(k)}$, la (4.99) consente di determinare direttamente gli $y^{(k)}$ in un modo simile al calcolo dei vettori $x^{(k)}$ nel metodo base. Si nota, tuttavia, che, mentre $x^{(k+1)}$ dipende solo da $x^{(k)}$, il vettore $y^{(k+1)}$ è ottenuto utilizzando i vettori calcolati nelle utlime due iterazioni, $y^{(k)}$ e $y^{(k-1)}$. Per tale motivo la procedura descritta è un **metodo iterativo a due passi**. Per la presenza di coefficienti il cui valore non è costante, ma dipende dall'iterazione (β_k), la procedura, al contrario dei metodi analizzati fino ad ora, appartiene alla classe dei **metodi non stazionari**. Inoltre, essa richiede l'uso di un vettore in più rispetto al metodo base. Ciò può pesare in maniera sensibile quando si vogliono risolvere sistemi lineari di grandi dimensioni[19].

[19] É da osservare, infine, che la procedura di accelerazione in esame richiede anche la conoscenza di a e b che, si ricorda, sono il più piccolo ed il più grande autovalore della matrice di iterazione del metodo base. Nella maggior parte dei casi tali quantità non sono note, ed il loro calcolo è, in generale, computazionalmente dispendioso. Per tale motivo, sono state sviluppate opportune versioni della procedura, nelle quali si utilizzano stime di a e di b.

4.14 Criteri di arresto

Nel paragrafo 4.4 si è analizzato un criterio per arrestare un metodo iterativo, il quale è basato su una stima dell'errore. Si ritorna su tale argomento con l'obiettivo di discutere dei concetti di base per la progettazione di criteri di arresto affidabili, cioè criteri il cui utilizzo consenta di ottenere una desiderata accuratezza.
In generale, l'effettivo utilizzo di un metodo iterativo richiede l'arresto del procedimento quando l'errore è divenuto sufficientemente piccolo. Poichè non è possibile calcolare l'errore in maniera diretta dal momento che la soluzione non è nota, per progettare un criterio di arresto soddisfacente occorre stimare l'errore ad ogni iterazione.
Si premette la seguente definizione:

Definizione 4.14.1. (Residuo)
Dato il sistema lineare $Ax = b$, sia $\bar{x}$ un vettore di dimensione n. Si definisce **residuo** *del sistema in $\bar{x}$ il vettore:*

$$r = b - A\bar{x}.$$

Si osservi che una norma del residuo è una misura di quanto il vettore $\bar{x}$ soddisfa il sistema.

Uno dei criteri di arresto più utilizzato si basa su una stima di quanto il residuo all'iterazione generica si è ridotto rispetto al residuo iniziale, cioè:

$$\frac{\|r^{(k)}\|}{\|r^{(0)}\|} \leq \texttt{Tol}, \tag{4.102}$$

dove $r^{(k)}$ è il residuo del sistema $Ax = b$ in $x^{(k)}$. Il Teorema che segue stabilisce la relazione che sussiste tra l'errore e il residuo alla generica iterazione.

Teorema 4.14.1. *Alla k-ma iterazione del metodo iterativo:*

$$x^{(k+1)} = Cx^{(k)} + d, \;\; k \geq 0,$$

si ha:

$$\|x^{(k)} - x^*\| \leq \|A^{-1}\| \cdot \|r^{(k)}\|, \tag{4.103}$$

dove $r^{(k)}$ è il residuo in $x^{(k)}$. Inoltre, se $x^{(0)}$ è il vettore iniziale, si ha:

$$\frac{\|x^{(k)} - x^*\|}{\|x^{(0)} - x^*\|} \leq \mu(A) \cdot \frac{\|r^{(k)}\|}{\|r^{(0)}\|}, \tag{4.104}$$

dove $\mu(A) = \|A\| \cdot \|A^{-1}\|$ è l'indice di condizionamento della matrice A.

Dimostrazione Da

$$r^{(k)} = b - Ax^{(k)}, \tag{4.105}$$

si ha:

$$x^{(k)} - x^* = -A^{-1}r^{(k)},$$

da cui segue la (4.103). Inoltre, dalla (4.105), per $k = 0$ si ha:

$$r^{(0)} = -A(x^{(0)} - x^*),$$

da cui segue che:

$$\|r^{(0)}\| = \|A(x^{(0)} - x^*)\| \leq \|A\| \cdot \|x^{(0)} - x^*\|.$$

Dall'ultima relazione e dalla (4.103) si ricava che:

$$\frac{\|x^{(k)} - x^*\|}{\|x^{(0)} - x^*\|} \leq \|A\| \cdot \|A^{-1}\| \cdot \frac{\|r^{(k)}\|}{\|r^{(0)}\|}$$

e, quindi, la (4.104). ■

Si osservi, innanzitutto, che la (4.104) mette in relazione la riduzione dell'errore alla generica iterazione con quella del residuo. Inoltre, in base alla (4.103), se si arresta il metodo quando vale la (4.102), per l'errore assoluto si ha la seguente maggiorazione:

$$\|x^{(k)} - x^*\| \leq \|A^{-1}\| \cdot \|r^{(0)}\| \cdot \texttt{Tol}. \tag{4.106}$$

In generale, l'utilizzo del criterio (4.102) richiede ad ogni iterazione il calcolo del prodotto tra una matrice ed un vettore, e quindi un costo computazionale per iterazione circa uguale a n^2 flops. Tuttavia, per alcuni metodi, il residuo ad ogni iterazione è disponibile senza costi aggiuntivi. Ad esempio, per il metodo di Jacobi, il residuo al passo k può essere ottenuto senza dover calcolare un prodotto matrice-vettore; infatti per la i-ma componente del residuo si ha:

$$r_i^{(k)} = b_i - \sum_{j=1}^{n} a_{ij}x_j^{(k)} = b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^{(k)} - a_{ii}x_i^{(k)} = a_{ii}(x_i^{(k+1)} - x_i^{(k)}).$$

Si può verificare che per il metodo di G-S il residuo assume un'analoga espressione.

Osservazione 4.5. *È importante osservare che, anche se criteri di arresto basati sul residuo, come (4.102), sono utilizzati frequentemente, un residuo piccolo non comporta automaticamente un errore piccolo. Ciò è dovuto in generale al fatto che, come stabilito dal Teorema 4.14.1, la relazione tra l'errore e il residuo dipende dal condizionamento della matrice A.*

♣ **Esempio 4.27.** Per illustrare che la grandezza del residuo, $\|r^{(k)}\|$, può non essere un affidabile indicatore della grandezza dell'errore, si consideri il sistema lineare:

$$\begin{cases} 5x_1 + 0.0001x_2 + 0.0001x_3 = 5.0002 \\ 0.0001x_1 + 2.25000001x_2 + 0.00000001x_3 = 2.25010002 \\ 0.0001x_1 + 0.00000001x_2 + 0.00000001x_3 = 0.00010002 \end{cases}$$

la cui soluzione è $x_i^* = 1$, $i = 1, 2, 3$. Nella Tabella che segue sono riportati i valori di $\|r^{(k)}\|$ e $\|x^{(k)} - x^*\|$ nelle prime iterazioni del metodo di G-S (in particolare si è utilizzata la norma euclidea) partendo da $x_i^0 = 0$, $i = 1, 2, 3$. Si evince che ad un valore piccolo di $\|r^{(k)}\|_2$ non necessariamente corrisponde un errore con un comparabile ordine di grandezza. Tale fenomeno è dovuto agli autovalori della matrice A $(0.8 \cdot 10^{-8}, 2.25, 5)$, in particolare al più piccolo, il cui valore è $\lambda_1 \simeq 10^{-8}$. Poichè $\|A^{-1}\|_2 = \lambda_1^{-1}$, dalla (4.103) si ha, per l'errore, la maggiorazione $\|x^{(k)} - x^*\|_2 \leq 10^8 \cdot \|r^{(k)}\|_2$, che giustifica i risultati ottenuti.

k	$\|r^{(k)}\|_2$	$\|x^{(k)} - x^*\|_2$
1	$1.60 \cdot 10^{-4}$	$0.40 \cdot 10^{0}$
2	$3.20 \cdot 10^{-5}$	$8.00 \cdot 10^{-2}$
3	$6.40 \cdot 10^{-6}$	$1.60 \cdot 10^{-2}$
4	$1.28 \cdot 10^{-6}$	$3.20 \cdot 10^{-3}$
5	$2.56 \cdot 10^{-7}$	$6.40 \cdot 10^{-4}$
6	$5.12 \cdot 10^{-8}$	$1.28 \cdot 10^{-4}$
7	$1.02 \cdot 10^{-8}$	$2.56 \cdot 10^{-5}$
8	$2.04 \cdot 10^{-9}$	$5.12 \cdot 10^{-6}$
9	$4.09 \cdot 10^{-10}$	$1.02 \cdot 10^{-6}$
10	$8.19 \cdot 10^{-11}$	$2.04 \cdot 10^{-7}$

♣

Si osservi, infine, che l'affidabilità del criterio di arresto (4.102) dipende anche dalla scelta di $x^{(0)}$. Infatti, se $x^{(0)}$ è "vicino" alla soluzione, la quantità $\|r^{(0)}\|$ può essere molto piccola e si corre il rischio di eseguire iterazioni inutili, cioè che non migliorano l'accuratezza del risultato; viceversa, se il punto iniziale è "distante" dalla soluzione, e quindi $\|r^{(0)}\|$ è un numero grande, il metodo può essere arrestato "troppo presto", ottenendo un'approssimazione poco accurata della soluzione.

♣ **Esempio 4.28.** Si consideri il sistema di equazioni lineari:

$$\begin{cases} 10000x_1 + 1x_2 + 4x_3 = 10005 \\ 0.5x_1 + 1x_2 + 0.4x_3 = 1.9 \\ 0.2x_1 + 0.6x_2 + 0.3x_3 = 1.1 \end{cases}$$

la cui soluzione è $x_i^* = 1$, $i = 1, 2, 3$. Applicando a tale sistema il metodo di Jacobi con $x_i^{(0)} = 0, i = 1, 2, 3$ ed utilizzando come criterio di arresto la (4.102) con `Tol`$= 10^{-6}$, il metodo si ferma dopo 71 iterazioni e gli ultimi valori calcolati, arrotondati alla sesta cifra decimale, sono: $x_1^{(71)} = 1.000000$, $x_2^{(71)} = 1.000380$, $x_3^{(71)} = 1.001114$. Si osserva che per le componenti x_2 e x_3 non si è ottenuta l'accuratezza desiderata. Poichè $\|r^{(0)}\|_2 \simeq 10005$ e $\|A^{-1}\|_2 \simeq 21.119$, dalla (4.106) si ha, per l'errore, la maggiorazione $\|x^{(71)} - x^*\|_2 \leq 0.211$. Si osservi che $\|x^{(71)} - x^*\|_2 = 0.001177$. ♣

4.15 Software matematico disponibile per i metodi iterativi

Descriveremo il software matematico disponibile per i metodi iterativi classificandolo secondo la convenzione introdotta da *J. Rice* in *Numerical methods, Software and Analysis*, Academic Press, 1993 (cfr. **Parte 1, Cap. 3**). Parliamo, dunque, di:

1. routine individuali, o eventualmente raccolte in collezioni, pubblicate su riviste di software specializzate, ad esempio `TOMS` e *Jcam*;

2. packages di routine che risolvono problemi in una specifica area computazionale; approfondiremo, a tal proposito, la libreria `SPARSKIT`, di dominio pubblico, specifica per la risoluzione di problemi di algebra lineare che coinvolgono matrici sparse;

3. packages di routine di base, ad esempio `SLATEC`, di dominio pubblico;

4. librerie di software *general - purpouse*, di tipo commerciale, come le librerie del `NAg` e la `IMSL`, oppure le librerie *open-source*, `PETSc` e `TRILINOS`; queste ultime due, in particolare, contengono moduli mirati alla risoluzione di sistemi lineari mediante i più noti metodi iterativi, per i quali è prevista, inoltre, un'ampia scelta di precondizionatori.

5. Ambienti di risoluzione di problemi (PSE), ad esempio `MATLAB` che mette a disposizione funzioni che implementano i più noti metodi iterativi; esistono, inoltre, pacchetti specifici sviluppati in `MATLAB` e di dominio pubblico, che contengono routine per l'implementazione di metodi iterativi e l'interazione ed il confronto con metodi diretti.

Analizziamo, brevemente, le singole classi.

1. Tra le routine dell'ACM-TOMS, reperibili all'indirizzo WWW di Internet:

   `http://www.netlib.org/toms`,

   ne citiamo solo alcune destinate alla risoluzione numerica di equazioni lineari mediante metodi iterativi. La routine `LSQR`, sviluppata in Fortran nel 1982, è finalizzata alla risoluzione di sistemi lineari sparsi, sottodeterminati o sovradeterminati e problemi di minimi quadrati.
   Fa parte di `CALGO` anche `ITPACK 2C`, un Package FORTRAN per la risoluzione di sistemi lineari sparsi e di grandi dimensioni, mediante metodi iterativi *adattativi*

accelerati (*Solving Large Sparse Linear Systems by Adaptive Accelerated Iterative Methods*). Il package, pubblicato nel 1982, include routine che implementano metodi iterativi classici precondizionati, tra i quali: JCG, JSI, SOR, SSORCG, SSORSI, RSCG e RSSI. Le versioni in singola e doppia precisione sono disponibili anche nella directory `http://www.netlib.org/itpack/` di Netlib.
`PREQN` è una collezione di subroutine, scritte in Fortran 77 e pubblicate nel 2001 sul TOMS, destinate al calcolo di precondizionatori per il metodo del Gradiente Coniugato (CG), quando quest'ultimo è applicato ad un insieme di sistemi lineari con matrici dei coefficienti simmetriche e definite positive.
Ricordiamo, infine, la libreria `sparse BLAS` (*Sparse Basic Linear Algebra Subprograms*) scritta in Fortran 95 e pubblicata nel 2002, che contiene routine per operazioni di base tra vettori e matrici sparse. L'ultima versione del package è reperibile alla pagina web

`http://www.cerfacs.fr/~voemel/SparseBLAS/SparseBLAS.html`.

Anche il *Journal of Computational and Applied Mathematics*, la cui *home page* è:

`http://www.elsevier.com/locate/cam`

ha pubblicato software matematico relativo all'implementazione di metodi iterativi, in particolare a tecniche di precondizionamento e confronto con metodi diretti.

2. `SPARSKIT` è un package per la risoluzione di problemi di algebra lineare con matrice sparsa. Il software, scritto in Fortran, è reperibile dalla pagina web:

   `http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html`

   dove, oltre al manuale in formato *ps* ed alle informazioni generali sul pacchetto e sui diritti di licenza, è possibile reperire il software e la documentazione nell'archivio compresso: `SPARSKIT2.tar.gz`.

Sul sito di Netlib sono disponibili, inoltre, packages finalizzati alla risoluzione di sistemi lineari con matrice dei coefficienti sparsa. Ne citiamo solo alcuni.

slap (*Sparse Linear Algebra Package*) è un package per la risoluzione di sistemi lineari di notevoli dimensioni, sparsi, con matrice dei coefficienti simmetrica o non simmetrica, definita positiva, mediante metodi iterativi precondizionati. Il software è disponibile nella directory di Netlib: `http://www.netlib.org/slap/` ed include routine che implementano metodi iterativi noti, tra i quali il metodo del Gradiente Coniugato (CG) precondizionato, del Gradiente Coniugato per le equazioni normali (CGNE) precondizionato, del Gradiente Bi-Coniugato (BiCG)

precondizionato ed il metodo del minimo residuo generalizzato (GMRES).

Nella directory di Netlib: `http://www.netlib.org/linalg` è disponibile **psblas1.0**, per l'implementazione parallela di solutori iterativi per sistemi lineari sparsi; sono implementati, in particolare, metodi iterativi basati su sottospazi di Krylov, per calcolatori a memoria distribuita. Il software è implementato in Fortran 90/Fortran 77 e C e richiede l'interfaccia BLACS per le comunicazioni in ambiente parallelo (*BLACS message passing*).

Sempre di pubblico dominio è il package **UMFPACK**, un insieme di routine per la risoluzione di sistemi lineari non-simmetrici sparsi, disponibile sulla pagina web:

`http://www.cise.ufl.edu/research/sparse/umfpack/`.

Il software è scritto in ANSI/ISO C, con una interfaccia in `MATLAB` (Versione 6.0 e successive).

3. `SLATEC` è una libreria di dominio pubblico costituita da routine di base, scritte in Fortran 77. In particolare un capitolo, indicato con `D`, è dedicato all'algebra lineare e si articola in sezioni che riguardano:

   `D1` operazioni elementari tra vettori e matrici,

   `D2` soluzione di sistemi di equazioni lineari,

   `D3` calcolo del determinante,

   `D4` autovalori ed autovettori,

   `D5` decomposizione QR ed ortogonalizzazione di Gram-Schmidt,

   `D6` decomposizione ai valori singolari (SVD),

   `D7` aggiornamento di decomposizioni matriciali,

   `D8` risoluzione di sistemi di equazioni sovradeterminati o sottodeterminati e calcolo della pseudo-inversa di una matrice.

   Della sezione `D2` fanno parte le routine di `SLATEC` che implementano metodi iterativi (Jacobi, Gauss-Seidel, GMRES, CG, BiCG), con o senza precondizionatore, da applicare a sistemi lineari con matrice dei coefficienti reale *sparsa* (simmetrica o non simmetrica). Le routine di `SLATEC` sono reperibili all'indirizzo World Wide Web (WWW) di Internet: `http://www.netlib.org/slatec`.

4. La libreria NAg del Numerical Algorithm Group di Oxford è sicuramente una fonte molto ampia di software matematico. Per buona parte delle routine esistono versioni in FORTRAN e C, in singola e doppia precisione, nonché una versione parallela (in Fortran 77).

Il Capitolo 5 della NAg's Fortran 90 Library, è dedicato alla risoluzione numerica di sistemi di equazioni lineari; il modulo `Module` 5.7 : `nag_sparse_lin_sys` contiene procedure che implementano metodi iterativi, con o senza precondizionatore, per la risoluzione di sistemi lineari sparsi con matrice dei coefficienti non simmetrica o complessa non Hermitiana. La routine `nag_sparse_gen_lin_sol` consente la scelta tra quattro metodi iterativi: GMRES(m), CGS, Bi-CGSTAB(l), TFQMR e prevede un'opzione sull'utilizzo del precondizionatore. L'indirizzo World Wide Web (WWW) di Internet del NAg è: `http://www.nag.com`.

La libreria dell'IMSL (*International Mathematics and Statistics Library*) è una collezione di software matematico, di tipo commerciale. La prima versione è stata pubblicata in Fortran nel 1970, seguita da una versione in linguaggio C, originariamente chiamata C/Base nel 1991, da una versione in Java nel 2002 ed, infine, da una versione in C# nel 2004.
La versione `IMSL Fortran Numerical Library Version 6.0` comprende una sezione costituita da routine dedicate alla risoluzione di sistemi lineari mediante metodi iterativi; `PCGRC` risolve un sistema lineare a coefficienti reali, con matrice simmetrica e definita, usando il metodo del Gradiente Coniugato (CG) precondizionato (with reverse communication), `JCGRC` risolve un sistema lineare a coefficienti reali, con matrice simmetrica e definita, usando il metodo del Gradiente Coniugato (CG) con precondizionatore di Jacobi (with reverse communication), infine `GMRES` implementa il metodo omonimo (GMRES with reverse communication), per la risoluzione numerica di un sistema lineare.

La libreria `PETSc` (*Portable Extensible Toolkit for Scientific Computation*), di tipo *open-source*, è una collezione di strutture dati e routine per la soluzione di problemi modellizzati mediante equazioni differenziali alle derivate parziali (PDE) e risolvibili sia in ambiente di calcolo sequenziale che parallelo. `PETSc` è di dominio pubblico e reperibile alla pagina web:

`http://www-unix.mcs.anl.gov/petsc/petsc-2/`.

La libreria `PETSc` è adatta ad essere utilizzata per problemi in larga scala e molti progetti scientifici sono costruiti intorno ai suoi moduli. PETSc include, inoltre, un gran numero di solutori per sistemi di equazioni lineari e non lineari che si interfacciano con software scritto in C, C++ e Fortran e che si basano su librerie di software per il calcolo scientifico ormai consolidate, quali BLAS e LAPACK; i solutori di `PETSc` utilizzano, inoltre, la libreria MPI per le comunicazioni in ambiente parallelo.
Tra i moduli principali che costituiscono la libreria citiamo il `KSP`, che contiene le implementazioni di molti dei più popolari metodi iterativi basati sui sottospazi di Krylov, incluso GMRES, CG, CGS, Bi-CG-Stab e LSQR; a tal proposito vale la pena ricordare, inoltre, il modulo `PC`, una collezione di precondizionatori, implementati sia in sequenziale che in parallelo (che comprende, ad esempio, *ILU(k)*

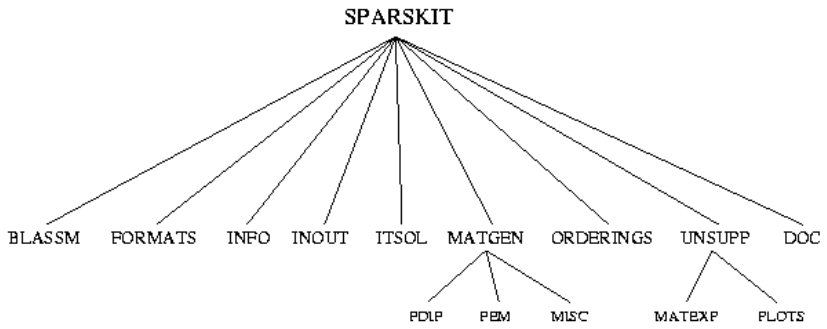


Figura 4.6: **Organizzazione di** `SPARSKIT`

(sequenziale), *LU*, *block Jacobi*, sia in sequenziale che in parallelo, e *ICC(0)*).

Routines destinate alla risoluzione di sistemi di equazioni lineari mediante metodi iterativi, in particolare basati sui sottospazi di Krylov, si possono trovare nella libreria di software matematico `TRILINOS`, di dominio pubblico, reperibile alla pagina web:

`http://trilinos.sandia.gov/`.

`TRILINOS` è composto da una serie di pacchetti di routine sequenziali e parallele, autoconsistenti; tuttavia lo sviluppatore di software che utilizza lo strumento `TRILINOS` può sfruttare la possibile interazione tra i diversi pacchetti nonché quella tra questi ultimi e librerie di software di produzione esterna. I pacchetti di `TRILINOS` si servono anche del supporto delle librerie BLAS e LAPACK ed utilizzano la libreria MPI per le comunicazioni in ambiente parallelo. Essi sono scritti in C++, ma forniscono interfacce di supporto per utenti C e Fortran.

5. Tra gli ambienti di risoluzione di problemi (PSE) quello più utilizzato è sicuramente `MATLAB` il quale mette a disposizione nove funzioni che implementano i principali **metodi iterativi** per sistemi con matrice dei coefficienti sparsa[20]; tra questi:

   `bicg, cgs, gmres, lsqr, minres, symmlq, ...`

4.15.1 La libreria `SPARSKIT`

`SPARSKIT` è una libreria di software per la risoluzione di problemi con matrici sparse. La sua organizzazione è illustrata in figura 4.6 e descritta di seguito.

- Il pacchetto `FORMATS` contiene sia routines per la memorizzazione di matrici (nel file `formats.f`) che routines per eseguire operazioni di base sulle matrici (`unary.f`).

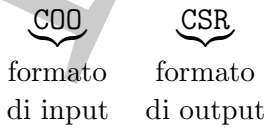
[20] `http://www.mathworks.com/`

- `BLASSM` (*Basic Linear Algebra Subprograms with Sparse Matrices*) comprende due moduli per l'*Algebra Lineare di Base per Matrici Sparse*, `blassm.f` e `matvec.f`.
- `INOUT` contiene le routines per l'input e l'output dei dati (modulo `inout.f`).
- `MATGEN` contiene routines (`FDIF`, `FEM`, `MISC`) per la generazione delle matrici relative ai *problemi test*.
- Nella directory `ITSOL` sono implementati i principali metodi iterativi e precondizionatori.
- `ORDERINGS` contiene le routines per il riordino degli elementi utilizzando tre distinti metodi (`levset.f`, `color.f`, `ccn.f`).
- `UNSUPP` contiene routines e drivers di supporto: `MATEXP`, `PLOTS`, `BLAS1`.
- `INFO` contiene informazioni utili sulle routines.
- `DOC` contiene la documentazione principale del pacchetto.

Analizziamo, brevemente, alcuni pacchetti che costituiscono la libreria.

FORMATS

Utilizzando le procedure contenute nel file `formats.f` è possibile convertire una matrice sparsa da un formato di memorizzazione ad un altro. Il nome delle routine incluse nel file è costituito da sei lettere, le prime tre per il formato di input, le altre tre per il formato di output; ad esempio il nome della routine `COOCSR` fornisce informazioni sul formato della matrice prima e dopo la conversione:



Analizziamo alcune delle sedici memorizzazioni supportate.

1. `DNS` (*Dense format*): è lo schema di memorizzazione in cui sono memorizzati tutti gli elementi della matrice sparsa.
2. `BND` (*Banded Linpack format*): è la memorizzazione utilizzata da Linpack per le matrici a banda.

♣ **Esempio 4.29.** Si consideri una matrice A di dimensione $m \times n$, con ku diagonali superiori e kl diagonali inferiori. Sia $m = n = 6$ e le ampiezze di banda inferiore e superiore rispettivamente $kl = 1$, $ku = 2$.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & 0 \\ 0 & 0 & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} & a_{5,6} \\ 0 & 0 & 0 & 0 & a_{6,5} & a_{6,6} \end{pmatrix}.$$

Dato che la matrice ha solo quattro diagonali *"significative"*, si può considerare un array bidimensionale AB di dimensione $ku + kl + 1 \times n = 4 \times n$ nelle cui righe memorizzare le diagonali della matrice nel modo seguente:

$$AB = \begin{pmatrix} * & * & a_{1,3} & a_{2,4} & a_{3,5} & a_{4,6} \\ * & a_{1,2} & a_{2,3} & a_{3,4} & a_{4,5} & a_{5,6} \\ a_{1,1} & a_{2,2} & a_{3,3} & a_{4,4} & a_{5,5} & a_{6,6} \\ a_{2,1} & a_{3,2} & a_{4,3} & a_{5,4} & a_{6,5} & * \end{pmatrix}$$

dove il simbolo $*$ indica che al corrispondente elemento dell'array AB non è stato associato alcun valore. L'elemento $a_{i,j}$ è dunque memorizzato nella colonna di posto j. L'indice di riga in AB è individuato dalla corrispondenza: $i \rightarrow ku + 1 + i - j$. ♣

In generale, per memorizzare una generica matrice a banda A, di dimensione $m \times n$, con ampiezze di banda ku e kl, si utilizza un array bidimensionale di dimensione $(ku + kl + 1) \times n$, nel modo seguente:

$$AB(ku + 1 + i - j, j) = a_{i,j}$$
$$1 \leq ku + 1 + i - j \leq ku + kl + 1 \Rightarrow \max(1, j - ku) \leq i \leq \min(m, j + kl)\,.$$

Tale schema di memorizzazione è detto *a banda* (in inglese si parla di *band storage*) ed ha una complessità di spazio pari a $(ku + kl + 1)n$. Tale memorizzazione risulta "significativamente" vantaggiosa, rispetto alla usuale memorizzazione mediante un array bidimensionale di dimensione $m \times n$, se $ku, kl << min\{m, n\}$.

3. `CSR` (*Compressed Sparse Row format*) è la rappresentazione di base utilizzata da `SPARSKIT`.

♣ **Esempio 4.30.** Sia assegnata la matrice sparsa $A \in \mathbb{R}^{m \times n}$ con nnz elementi non nulli. Sia $m = n = 6$ e $nnz = 12$. Memorizziamo la matrice A nei tre vettori AA, JA, IA. Il vettore di reali AA contiene gli nnz elementi non nulli di A memorizzati per righe; il vettore di interi JA contiene l'indice di colonna degli elementi a_{ij} di A memorizzati in AA. La dimensione di JA è nnz; il vettore di reali IA contiene il puntatore al primo elemento non nullo di ogni riga di A. Quindi il valore di $IA(i)$ coincide con la posizione in AA (e JA) del primo elemento non nullo della $i-$esima riga. La dimensione di IA è $m + 1$ con $IA(m + 1) = nnz + 1$. Si osserva, inoltre,

che $IA(i+1) - IA(i)$ rappresenta il numero di elementi non nulli della riga i. Utilizzando tale schema la matrice

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

è memorizzata nel modo seguente:

$$AA = (7, -3, -1, 2, 8, 1, -3, 5, -1, 4, -2, 6);$$

$$JA = (1, 3, 5, 1, 2, 3, 1, 4, 2, 5, 4, 6);$$

$$IA = (1, 4, 6, 7, 9, 11, 13).$$

Si noti che l'ultimo elemento di IA è uguale al numero totale degli elementi di AA più uno.

♣

4. `CSC` (*Compressed Sparse Column format*) consiste nella memorizzazione degli elementi non nulli per colonne.

♣ **Esempio 4.31.** Sia $A \in \mathbb{R}^{m\times n}$ la matrice sparsa dell'esempio 4.30, con nnz elementi non nulli, $m = n = 6$ e $nnz = 12$. Memorizziamo la matrice A nei tre vettori AA, JA, IA. Il vettore di reali AA contiene gli nnz elementi non nulli di A memorizzati per colonne; il vettore di interi JA contiene l'indice di riga degli elementi a_{ij} di A memorizzati in AA. La dimensione di JA è nnz; il vettore di reali IA contiene il puntatore al primo elemento non nullo di ogni colonna di A. Quindi il valore di $IA(j)$ coincide con la posizione in AA (e JA) del primo elemento non nullo della $j-$esima colonna. La dimensione di IA è $n+1$ con $IA(n+1) = nnz+1$. Si osserva, inoltre, che $IA(j+1) - IA(j)$ rappresenta il numero di elementi non nulli della colonna j. Utilizzando tale schema la matrice

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

è memorizzata nel modo seguente:

$$AA = (7, 2, -3, 8, -1, -3, 1, 5, -2, -1, 4, 6);$$

$$JA = (1, 2, 4, 2, 5, 1, 3, 4, 6, 1, 5, 6);$$

$$IA = (1, 4, 6, 8, 10, 12, 13).$$

Si noti che l'ultimo elemento di IA è uguale al numero totale degli elementi di AA più uno.

♣

5. `COO` (*Coordinate format*) è la memorizzazione degli elementi non nulli di una matrice sparsa in qualunque ordine.

♣ **Esempio 4.32.** Sia $A \in \mathbb{R}^{m\times n}$ una matrice sparsa con nnz elementi non nulli, $m = n = 5$ e $nnz = 12$. Memorizziamo la matrice A in tre vettori AA, JA, IA. Il vettore di reali AA contiene gli nnz elementi non nulli di A memorizzati *in qualunque ordine*; il vettore di interi JA contiene l'indice di riga degli elementi a_{ij} di A memorizzati in AA. La dimensione di JA è nnz; il vettore di reali IA contiene l'indice di colonna degli elementi a_{ij} di A memorizzati in AA. La dimensione di IA è nnz. Utilizzando tale schema la matrice

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

è memorizzata nel modo seguente:

$$AA = (12, 9, 7, 5, 1, 2, 11, 3, 6, 4, 8, 10);$$

$$JA = (5, 3, 3, 2, 1, 1, 4, 2, 3, 2, 3, 4);$$

$$IA = (5, 5, 3, 4, 1, 4, 4, 1, 1, 2, 4, 3).$$

♣

Tra le trentuno routine contenute in `formats.f` ricordiamo:

1. `CSRDNS`: converte il formato dal `CSR` al `DNS`;
2. `DNSCSR`: converte il formato dal `DNS` al `CSR`;
3. `COOCSR`: converte il formato dal `COO` al `CSR`;
4. `COICSR`: converte il formato dal `COO` al `CSR` effettuandolo *in-place*;
5. `CSRCOO`: converte il formato dal `CSR` al `COO`;
6. `CSRCSC`: converte il formato dal `CSR` al `CSC`, effettuando una trasposizione della matrice iniziale.

Analizziamo la `DNSCSR`; per la descrizione delle altre si rimanda a

`http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html`

subroutine dnscsr(nrow, ncol, nzmax, dns, ndns, a, ja, ia, ierr):

Parametri di input:

- *nrow*: numero di righe della matrice assegnata;
- *ncol*: numero di colonne della matrice assegnata;
- *nzmax*: massimo numero di elementi non zero della matrice assegnata; sarà la dimensione degli array a e ja;
- *dns*: array di dimensione $nrow \times ncol$, contenente, in input, la matrice in formato denso;
- *ndns*: prima dimensione dell'array *dns*.

Parametri di output:

- a: array dei valori non nulli della matrice assegnata nel formato `CSR` (primo vettore della rappresentazione);
- ja: array delle colonne degli elementi non nulli della matrice assegnata nel formato `CSR` (secondo vettore della rappresentazione);
- ia: array di puntatori; terzo vettore della rappresentazione della matrice assegnata nel formato `CSR`;
- *ierr*: indicatore di errore; $ierr = 0$ se l'esecuzione si è conclusa correttamente; $ierr = i$ se l'esecuzione si è fermata nell'analisi della riga i, poiché non c'è spazio sufficiente nei vettori a, ja, ia (secondo quanto stabilito attraverso il valore del parametro di input $nzmax$).

♣ **Esempio 4.33.** Sia assegnata una matrice di reali, *sparsa*, quadrata di dimensione 5, memorizzata in un array *dns*:

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

Convertire A dal formato *denso*, `DNS`, al formato `CSR`. Assegnando in input i parametri: $nrow = 5$, $ncol = 5$, un valore $nzmax \geq 9$, la variabile *dns* e $ndns = 5$, la routine `dnscsr` restituisce, in output, i tre vettori:

$$a = (1, 2, 3, 4, 5, 6, 7, 8, 9);$$

$$ja = (1, 3, 2, 4, 4, 5, 2, 5, 5);$$

$$ia = (1, 3, 5, 7, 9, 10).$$

e la variabile $ierr = 0$. ♣

Alcune delle routine contenute nel file `unary.f`, che eseguono operazioni di base sulle matrici, sono:

1. `SUBMAT`: estrae una sottomatrice quadrata o rettangolare da una matrice sparsa. Sia la matrice in input che quella in output sono nel formato `CSR`. La routine è *in-place*.

2. `COPMAT`: copia una matrice nel formato `CSR` in un'altra anch'essa in formato `CSR`.

3. `GETELM`: è una funzione il cui valore di output è l'elemento a_{ij} per ogni coppia (i, j) assegnata. Come parametro di output fornisce anche l'indirizzo dell'elemento negli array A e JA.

4. `GETDIA`: estrae la diagonale della matrice assegnata. Si può scegliere di non modificare la matrice di input o azzerare tutti i suoi elementi diagonali.

5. `CPERM`: effettua una permutazione delle colonne della matrice assegnata A, ovvero calcola la matrice $B = A \cdot Q$, con Q matrice di permutazione.

6. `RPERM`: effettua una permutazione delle righe della matrice assegnata A, ovvero calcola la matrice $B = P \cdot A$, con P matrice di permutazione.

7. `RETMX`: restituisce l'elemento massimo in valore assoluto per ciascuna riga della matrice assegnata A.

8. `INFDIA`: calcola il numero di elementi non nulli di ciascuna delle $2n - 1$ diagonali della matrice assegnata. Si noti che la prima diagonale considerata è quella denominata $-n$, costituita dal solo elemento di input $a_{n,1}$, mentre l'ultima è quella denominata n, costituita dal solo elemento $a_{1,n}$.

9. `RNRMS`: calcola le norme delle righe della matrice assegnata. Le norme $\|\cdot\|_1$, $\|\cdot\|_2$ e $\|\cdot\|_\infty$ sono supportate.

Analizziamo, ad esempio, la `INFDIA`.

subroutine infdia(n, ja, ia, ind, idiag):

Parametri di input:

- n: dimensione della matrice assegnata;
- ja: array delle colonne degli elementi non nulli della matrice assegnata nel formato `CSR` (secondo vettore della rappresentazione);

- *ia*: array di puntatori; terzo vettore della rappresentazione della matrice assegnata nel formato `CSR`.

Parametri di output:

- *ind*: array di interi di lunghezza $2n-1$. Il Il $k-$esimo elemento del vettore *ind* contiene il numero di elementi non nulli della diagonale k.
- *idiag*: intero, contiene il numero totale di elementi non nulli trovati sulle diagonali della matrice assegnata

♣ **Esempio 4.34.** Sia assegnata una matrice di reali, *sparsa*, quadrata di dimensione $n = 5$:

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

Si supponga A memorizzata nel formato `CSR` di `SPARSKIT`; in particolare, il secondo e terzo vettore della rappresentzione sono, rispettivamente:

$$ja = (2, 3, 1, 4, 5, 2, 3, 5, 4, 1, 4)$$

$$ia = (1, 3, 6, 9, 10, 12).$$

Assegnando, in input, alla routine `infdia` la variabile $n = 5$ ed i vettori ja e ia, essa restituisce il vettore

$$ind = (1, 0, 0, 3, 2, 1, 3, 1, 0)$$

e la variabile $idiag = 11$.

♣

BLASSM

I moduli contenuti in `BLASSM` eseguono operazioni algebriche di base. In particolare il modulo `blassms.f` esegue operazioni che coinvolgono due matrici, quali: $A+B$, $A+\sigma\, B$, con $\sigma \in \mathbb{R}$, $A \cdot B$, etc.

Descriviamo, brevemente, di seguito, le nove routines contenute in `blassms.f`:

1. `AMUB`: calcola il prodotto di due matrici, ovvero $C = A \cdot B$, dove sia A che B sono in formato `CSR`;
2. `APLB`: calcola la somma di due matrici, ovvero $C = A + B$, dove sia A che B sono in formato `CSR`;

3. `APLSB`: calcola $C = A + \sigma B$, dove $\sigma \in \mathbb{R}$ e sia A che B sono matrici in formato `CSR`;
4. `APMBT`: calcola sia la somma $C = A + B^T$ che la differenza $C = A - B^T$;
5. `APLSBT` calcola l'operazione $C = A + \sigma\, B^T$, $\sigma \in \mathbb{R}$;
6. `APLDIA`: calcola la somma di una matrice sparsa e di una matrice diagonale;
7. `DIAMUA`: calcola il prodotto di una matrice diagonale (posta a sinistra) ed una matrice sparsa, ovvero $C = D \cdot A$, con D matrice diagonale ed A matrice sparsa, entrambe memorizzate in formato `CSR`.
8. `AMUDIA`: calcola il prodotto di una matrice sparsa ed una matrice diagonale (posta a destra), ovvero $C = A \cdot D$, con D matrice diagonale ed A matrice sparsa, entrambe memorizzate in formato `CSR`.
9. `APLSCA`: *in-place* somma uno scalare alla diagonale di una matrice sparsa, ovvero esegue $A = A + \sigma I$, dove σ è uno scalare, A la matrice sparsa ed I la matrice identica.

Analizziamo in dettaglio la `AMUB`.

subroutine amub(nrow, ncol, job, a, ja, ia, b, jb, ib, c, jc, ic, nzmax, iw, ierr):

Parametri di input:

- *nrow*: intero, numero di righe della matrice A e della matrice C;
- *ncol*: intero, numero di colonne della matrice B e della matrice C;
- *job*: intero. Se $job = 0$ vengono creati solo i vettori jc e ic, ovvero viene creata solo la struttura della matrice C ma non sono calcolati i valori dei suoi elementi;
- a, ja, ia: array per la memorizzazione della matrice A nel formato `CSR`;
- b, jb, ib: array per la memorizzazione della matrice B nel formato `CSR`;
- *nzmax*: intero, rappresenta la lunghezza dei vettori di output, c e jc;
- *iw*: array di interi, area di lavoro di lunghezza uguale al numero di colonne della matrice A.

Parametri di output:

- c, jc, ic: array per la memorizzazione della matrice prodotto C nel formato `CSR`;

- *ierr*: intero, indicatore di errore; $ierr = 0$ indica che l'esecuzione si è conclusa correttamente mentre $ierr = k > 0$ indica che l'esecuzione è terminata durante il calcolo della k-esima riga della matrice C.

♣ **Esempio 4.35.** Siano assegnate due matrici di reali, A e B, *sparse*, quadrate, di dimensione 5 e si suppongano memorizzate nel formato `CSR` di `SPARSKIT`; in particolare, i vettori della rappresentazione di A sono:

$$a = (1, 4, 9, 2, 8, 5, 7)$$

$$ja = (1, 2, 2, 4, 2, 3, 5)$$

$$ia = (1, 3, 5, 6, 7, 8)$$

mentre quelli della rappresentazione di B sono:

$$b = (1, 2, 4, 2, 1, 3, 1)$$

$$jb = (2, 3, 1, 5, 5, 4, 1)$$

$$ib = (1, 3, 5, 6, 7, 8)$$

Assegnando, in input, alla routine `amub` le variabili $nrow = 5$, $ncol = 5$, $job = 1$, $nmax = 15$ e $iw[ncol]$, oltre che i sei vettori, a, ja, ia e b, jb, ib, essa restituisce, in output, la matrice prodotto, anch'essa memorizzata nel formato `CSR`:

$$c = (16, 1, 2, 8, 36, 6, 18, 32, 16, 5, 7)$$

$$ja = (1, 2, 3, 5, 1, 4, 5, 1, 5, 5, 1)$$

$$ia = (1, 5, 8, 10, 11, 12)$$

e la variabile $ierr = 0$. ♣

Il modulo `matvec.f` esegue operazioni di base che coinvolgono una matrice ed un vettore, ad esempio il prodotto matrice per vettore e la risoluzione di sistemi triangolari. Descriviamo, brevemente, alcune delle quindici routines contenute in `matvec.f`:

1. `AMUX`: esegue il prodotto di una matrice per un vettore ($y = Ax$). La matrice sparsa A è memorizzata nel formato `CSR`.

2. `ATMUX`: esegue il prodotto della trasposta di una matrice per un vettore ($y = A^T x$). La matrice sparsa A deve essere memorizzata nel formato `CSR`. Si noti come questa routine può eseguire anche il prodotto di una matrice sparsa A per un vettore, con A memorizzata nel formato `CSC`.

3. `LSOL`: risolve un sistema la cui matrice è triangolare inferiore ed unitaria (l'inversa coincide con la trasposta coniugata). La matrice è memorizzata nel formato `CSR`.

4. `LDSOL`: risolve un sistema la cui matrice è triangolare inferiore. La matrice è memorizzata nel formato `MSR` (*Modified Sparse Row*). Gli elementi della diagonale sono memorizzati in ordine inverso.

5. `LSOLC`: risolve un sistema la cui matrice è triangolare inferiore ed unitaria. La matrice è memorizzata nel formato `CSC`.

6. `USOL`: risolve un sistema la cui matrice è triangolare superiore ed unitaria. La matrice è memorizzata nel formato `CSR`.

7. `USOLC`: risolve un sistema la cui matrice è triangolare superiore ed unitaria. La matrice è memorizzata nel formato `CSC`.

Le altre routine contenute in `matvec.f` eseguono le stesse operazioni delle routine illustrate ma con la matrice coinvolta nell'operazione memorizzata in altri formati.

Analizziamo in dettaglio la `AMUX`.

subroutine amux(n, x, y, a , ja, ia):

Parametri di input:

- n: intero, numero di righe della matrice A;
- x: array di reali di lunghezza pari al numero di colonne della matrice A;
- a, ja, ia: array per la memorizzazione della matrice A nel formato `CSR`.

Parametri di output:

- y: array di reali di lunghezza n; contiene il prodotto $A \cdot x$.

♣ **Esempio 4.36.** Siano A una matrice di reali, quadrata, di dimensione $n = 5$ e x un vettore reale, anch'esso di dimensione $n = 5$. Sia A memorizzata nel formato `CSR` di `SPARSKIT`; in particolare, i vettori della sua rappresentazione siano:

$$a = (1, 4, 9, 2, 8, 5, 7)$$
$$ja = (1, 2, 2, 4, 2, 3, 5)$$
$$ia = (1, 3, 5, 6, 7, 8)$$

e sia

$$x = (1, 2, 3, 4, 5)$$

Assegnando, in input, alla routine `amux` le variabili $n = 5$, x ed i tre vettori, a, ja, ia, essa restituisce, in output, il vettore prodotto:

$$y = (9, 26, 16, 15, 35)$$

♣

INOUT

`INOUT` comprende routine per la lettura, scrittura e visualizzazione grafica delle strutture delle matrici sparse.
Descriviamo, brevemente, alcune delle undici routine contenute in `inout.f`:

1. `READMT`: legge una matrice nel formato `HB` (*Harwell Boeing Sparse Matrix File Format*) [21] e vettori dei termini noti, *solo in formato pieno*, non sparso.
2. `PRTMT`: crea un file `HB` a partire da un'arbitraria matrice nei formati `CSR` o `CSC`.
3. `PSPLTM`: genera in un file *ps* il *plot* della struttura della matrice A.
4. `PLTMT`: genera un file *pic* per il *plot* della struttura della matrice A (un *pic file* si utilizza per memorizzare immagini in un formato non compresso).
5. `READSK`: legge una matrice nel formato `CSR`.
6. `PRTUNF`: scrive matrici in formato `CSR` in un file non formattato, cioè scritto senza un formato particolare.
7. `READUNF`: legge file non formattati contenenti matrici in formato `CSR`.

Analizziamo in dettaglio la `READMT`.

subroutine readmt(nmax,nzmax,job,iounit,a,ja,ia,rhs,nrhs,guesol,nrow, ncol,nnz,title,key,type,ierr):

Parametri di input:

- $nmax$: intero, massimo numero di colonne per la matrice di input;
- $nzmax$: massimo numero di elementi non nulli per la matrice di input;
- job: intero, contiene informazioni sulla lettura dei dati di input; se $job = 0$ legge i valori di $ncol$, $nrow$, nnz, $title$, key, $type$ e $return$. La matrice non è letta; gli array a, ja, ia ed il vettore dei termini noti rhs non sono modificati. Se $job = 1$ sono letti solo gli arrays ja e ia. Se $job = 2$ sono letti gli arrays a, ja e ia. Se $job = 3$ sono letti gli arrays a, ja e ia ed il vettore dei termini noti rhs.
- $nrhs$: intero. In input contiene la dimensione di rhs.
- $iounit$: variabile logica, fornisce informazioni sulla matrice.

[21] Il formato `HB` memorizza una matrice sparsa in un file. Lo spazio richiesto è ridotto usando un formato di tipo *Compressed Column Storage* (`CCS`=`CSC` o `CSR`). Se la matrice è letta da file si utilizza lo stesso formato per rappresentarla in memoria.

Parametri di output:

- *job*: in output è modificato al valore più alto che può assumere, in base ai dati di input; ad esempio, se, in input, $job = 2$ ma non è fornita una matrice da leggere, il valore della variabile è modificato, in output, in $job = 1$.
- a, ja, ia: vettori della rappresentazione della matrice, nel formato `CSC`;
- *rhs*: array di reali, se determinato, in base al valore di *job*, è di dimensione $nrow + 1$;
- *nrhs*: intero, contiene il numero dei vettori dei termini noti;
- *guesol*: dato di tipo alfanumerico, costituito da due caratteri; questi ultimi segnalano, rispettivamente, se, in coda alle componenti del vettore dei termini noti, sono assegnati, in input, un dato iniziale (*initial guess*) e/o la *soluzione "esatta"*.
- *nrow*: intero, numero di righe della matrice;
- *ncol*: intero, numero di colonne della matrice;
- *nnz*: intero, numero di elementi non nulli di A.
- *title*: dato di tipo alfanumerico, costituito da 72 caratteri; contiene il titolo (descrittivo) della matrice test.
- *key*: dato di tipo alfanumerico, costituito da 8 caratteri; identificativo per la matrice;
- *type*: dato di tipo alfanumerico, costituito da 3 caratteri; descrive il tipo della matrice.
- *ierr*: intero, indicatore di errore; $ierr = 0$ segnala che la matrice è stata letta, $ierr = 1$ che la matrice non può essere letta, perché $ncol + 1 > nmax$, $ierr = 3$ segnala che la matrice non può essere letta, perché $ncol + 1 > nmax$ e $nnz > nzmax$; se $ierr = 4$ vuol dire che il vettore dei termini noti, l'*initial guess* e la soluzione "esatta" non possono essere letti poiché memorizzati in formato *sparso*; infine, se $ierr = 5$ allora i tre vettori (il vettore dei termini noti, l'*initial guess* e la soluzione "esatta") non possono essere letti poiché la dimensione dell'array *rhs*, dichiarata in input nella variabile *nrhs*, non è sufficientemente grande per memorizzarli.

♣ **Esempio 4.37.** Sia assegnato, in input, il puntatore al file in cui è rappresentata la matrice in formato `H/B`:

```
title                                                   key
         5    1    1    3     0
RUA          5    5   12     0
(6I3)     (12I3)     (5E15.8)     (5E15.8)
1    4    6    8   11   13
1    3    5    1    2    2    3    1    4    5    3    5
2.02765219E-01  6.03792479E-01  1.98814268E-01  1.52739270E-02  7.46785677E-01
8.46221418E-01  5.25152496E-01  8.38118445E-01  3.79481018E-01  8.31796018E-01
4.28892365E-01  1.89653748E-01
```

Il file descrive la matrice che, rappresentata in formato denso, è:

$$A = \begin{pmatrix} 2.02765219E-01 & 0 & 6.03792479E-01 & 0 & 1.98814268E-01 \\ 1.52739270E-02 & 7.46785677E-01 & 0 & 0 & 0 \\ 0 & 8.46221418E-01 & 5.25152496E-01 & 0 & 0 \\ 8.38118445E-01 & 0 & 0 & 3.79481018E-01 & 8.31796018E-01 \\ 0 & 0 & 4.28892365E-01 & 0 & 1.89653748E-01 \end{pmatrix}$$

In output, la funzione `readmt` fornisce, tra le altre variabili, i tre vettori:

$$\begin{aligned} a &= (2.02765219E-01, \quad 6.03792479E-01, \quad 1.98814268E-01, \quad 1.52739270E-02, \\ &\quad 7.46785677E-01, \quad 8.46221418E-01, \quad 5.25152496E-01, \quad 8.38118445E-01, \\ &\quad 3.79481018E-01, \quad 8.31796018E-01, \quad 4.28892365E-01 \quad 1.89653748E-01) \\ ja &= (1, 3, 5, 1, 2, 2, 3, 1, 4, 5, 3, 5) \\ ia &= (1, 4, 6, 8, 11, 13) \end{aligned}$$

♣

ITSOL

`ITSOL` comprende due driver principali: `ILUT` e `ITERS`:

- `ILUT` implementa l'algoritmo del GMRES (metodo iterativo per la risoluzione di un sistema lineare) precondizionato.
- `ITERS` implementa nove dei più utilizzati metodi iterativi per la risoluzione di sistemi lineari.

4.16 Risoluzione di sistemi lineari *sparsi* in ambiente `MATLAB`

In ambiente `MATLAB` [22] le funzioni relative a *matrici sparse* sono collocate nella directory `sparfun` e sono organizzate in categorie. Se ne descrivono alcune di seguito.

- Funzioni che consentono la **conversione da un formato di memorizzazione ad un altro** che sia più "vantaggioso" in termini di complessità di spazio. Inoltre, la memorizzazione dei soli elementi non nulli e dei loro indici consente, attraverso

[22] http://www.mathworks.com/

l'utilizzo di opportune funzioni, di evitare operazioni aritmetiche che sarebbero eseguite su elementi nulli, riducendone, così, il costo computazionale richiesto.

♣ **Esempio 4.38.** Sia `S` una matrice sparsa, memorizzata in formato *denso* [23]; si individuino gli indici di riga e colonna degli elementi non nulli di `S`; si memorizzino questi ultimi in un array `s`:

```
S=[ 1.6000    2.7000         0         0         0         0
    0.3000         0         0         0   47.0000         0
    1.1300   71.0000         0         0         0    1.0000
    0.0100         0         0    0.2200    0.0300   84.0000
         0         0         0         0   14.0000   15.0000
         0         0    0.7000    0.0800         0         0 ]
```

```
>>  [i,j,s] = find(S);
```

I vettori di output `i` e `j` contengono, rispettivamente, gli indici di riga e di colonna degli elementi non nulli di `S`.
Costruire una matrice `T`, nel formato sparso di `MATLAB` , che conservi solo le informazioni relative agli elementi non nulli di `S`:

```
>>  [m,n] = size(S);
>>  T= sparse(i,j,s,m,n);
```

L'output visualizzato sarà:

```
T =

   (1,1)       1.6000
   (2,1)       0.3000
   (3,1)       1.1300
   (4,1)       0.0100
   (1,2)       2.7000
   (3,2)      71.0000
   (6,3)       0.7000
   (4,4)       0.2200
   (6,4)       0.0800
   (2,5)      47.0000
   (4,5)       0.0300
   (5,5)      14.0000
   (3,6)       1.0000
   (4,6)      84.0000
   (5,6)      15.0000
```

Lo stesso risultato si ottiene attraverso l'istruzione:

[23] Per *formato denso* si intende la memorizzazione di una matrice in un array bidimensionale contenente tutti gli elementi della matrice, compresi i nulli.

```
>> T=sparse(S)
```

L'output consiste negli elementi non nulli di S con i rispettivi indici di riga e di colonna. Gli elementi sono ordinati per colonna. Con

```
>> full(T)
```

si visualizza T nel formato denso:

```
ans =

    1.6000    2.7000         0         0         0         0
    0.3000         0         0         0   47.0000         0
    1.1300   71.0000         0         0         0    1.0000
    0.0100         0         0    0.2200    0.0300   84.0000
         0         0         0         0   14.0000   15.0000
         0         0    0.7000    0.0800         0         0
```

♣

- Funzioni che **generano** matrici sparse elementari, ad esempio, `speye`, `sprand`, `sprandn`, o che operano su matrici sparse, come `sprandsym`, `spdiags`.

♣ **Esempio 4.39.** Assegnata una matrice *a banda* A, di dimensione $m \times n = 7 \times 4$:

```
>>    A=[ 11     0    13     0
           0    22     0    24
           0     0    33     0
          41     0     0    44
           0    52     0     0
           0     0    63     0
           0     0     0    74 ]
```

e ampiezza di banda $w = 6$, estrarre tutte le diagonali non nulle. In particolare: costruire una matrice B di dimensione $min(m, n) \times \overline{w}$, con $\overline{w} \leq w$, le cui colonne siano le $\overline{w}$ diagonali non nulle di A; costruire, inoltre, un vettore d di lunghezza w, le cui componenti intere individuino la posizione, in A, delle sue diagonali non nulle:

```
>> [B,d]=spdiags(A)

B =

    41    11     0
    52    22     0
    63    33    13
    74    44    24
```

```
d =

    -3
     0
     2
```

Si osserva che, se la $i-$esima componente di d è $d(i) = 0$, allora l'$i-$esima colonna di B è la diagonale principale di A; se $d(i) = k > 0$, l'$i-$esima colonna di B è la $k-$esima diagonale sopra la principale, se, infine, $d(i) = k < 0$, l'$i-$esima colonna di B è la $k-$esima diagonale sotto la principale.

Viceversa, dati gli array $B \in \mathbb{R}^{p \times \overline{w}}$ e $d \in \mathbb{R}^{\overline{w} \times 1}$, con $p = 4$ e $\overline{w} = 3$, costruire la matrice C che abbia $m = 7$ righe, $n = p$ colonne e le $\overline{w}$ colonne di B al posto delle diagonali indicate dalle componenti del vettore d:

```
>> C = spdiags(B,d,m,n)

C =

   (1,1)        11
   (4,1)        41
   (2,2)        22
   (5,2)        52
   (1,3)        13
   (3,3)        33
   (6,3)        63
   (2,4)        24
   (4,4)        44
   (7,4)        74
```

La funzione `spdiags` può, dunque, sia estrarre elementi diagonali da una matrice sparsa, riducendo la complessità di spazio richiesta dalla memorizzazione dell'intera matrice, che, eventualmente, sostituire i suoi elementi diagonali con valori nuovi. ♣

- Funzioni che forniscono **informazioni quantitative** o **grafiche** sulla struttura delle matrici sparse. Ad esempio:
  - informazioni sugli elementi non nulli o sul loro numero sono rese dalle funzioni `nonzeros` e `nnz`, rispettivamente;
  - la visualizzazione grafica della distribuzione degli elementi non nulli di una matrice sparsa è resa possibile dalla funzione `spy` che genera una figura in cui è visualizzata la *struttura* della matrice assegnata in input. Ogni punto del grafico rappresenta la posizione di un elemento non nullo all'interno della matrice.

♣ **Esempio 4.40.** Sia S la matrice sparsa dell'esempio 4.38. Con

```
>> spy(S,'r*')
```

si ottiene il grafico in Fig.4.7. Agli elementi di A, a_{ij}, $i = 1, \ldots, 6$, $j = 1, \ldots, 6$, corrispondono i punti del grafico, di coordinate $x \in \{1, \ldots, 6\}$, $y \in \{1, \ldots, 6\}$.
Nella finestra grafica compare, *di default*, anche il numero, `nz`, di elementi non nulli della matrice.

♣

♣ **Esempio 4.41.** Esempi di matrici che possono risultare più o meno sparse sono le *matrici di adiacenze* di *grafi*.
Un grafo è un insieme di nodi opportunamente connessi tra loro. La sua matrice delle adiacenze è una matrice quadrata, i cui elementi sono uguali a 0 o 1 (binaria). In particolare, l'elemento (i, j) della matrice è 1 se e solo se esiste nel grafo un arco che va dal vertice i al vertice j, altrimenti è 0.
In molti casi ogni nodo è connesso solo a pochi altri nodi, per cui la matrice delle adiacenze risulta sparsa con, in particolare, pochi elementi non nulli per ciascuna riga.
Ad esempio, in `MATLAB` , il grafo della semisfera (con cui si rappresenta la cupola geodesica di Buckminster Fuller) può essere generato attraverso la funzione `bucky` che fornisce la matrice delle adiacenze, B, e la matrice, V, delle coordinate cartesiane xyz dei vertici. Combinando due semisfere si ottiene il grafo di una struttura detta anche *Buckyball*, la cui forma ricorda quella di un pallone da calcio o una molecola di carbonio-60.
La funzione `gplot` applicata alla matrice delle adiacenze fornisce la visualizzazione bidimensionale del grafo (Fig.4.8).

```
>> [B,V] = bucky;
>> % matrice sparsa con elementi nulli
>> H = sparse(60,60);
>> k = 31:60;
>> H(k,k) = B(k,k);

>> gplot(B-H,V,'b-');
>> hold on
>> gplot(H,V,'r-');
>> title('Backyball')
>> axis equal
```

La distribuzione degli elementi non nulli della matrice delle adiacenze della semisfera si può visualizzare attraverso la funzione `spy` che fornisce il grafico della struttura della matrice di input. Si osserva che la matrice B è sparsa, di dimensione 60×60, e simmetrica in quanto il nodo i è connesso al nodo j se, e solo se, il nodo j è connesso al nodo i. Il suo diagramma è riportato in Fig.4.9.

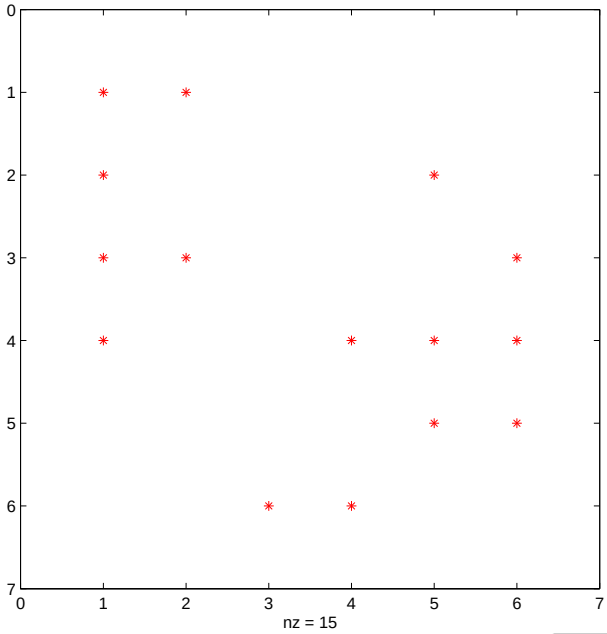


Figura 4.7:

```
>> spy(B)
```

♣

- Funzioni applicabili a matrici sparse che lavorano equivalentemente anche su matrici piene; di esse esiste, inoltre, una versione non specifica per matrici sparse. Tra queste:

  `eigs, svds,`
  `normest,condest,sprank`

  Ad esempio, la function `eigs` consente di calcolare un numero arbitrario di autovalori di una matrice. Opzionalmente consente di calcolare gli autovettori corrispondenti. La matrice di input deve essere quadrata e, preferibilmente, *di grandi dimensioni e sparsa*, sebbene la funzione si possa applicare, equivalentemente, anche a matrici piene.

- Funzioni che implementano le usuali tecniche di fattorizzazione, ad esempio LU, Cholesky e QR, che, se applicate a matrici sparse, tengono conto della sparsità della matrice nell'esecuzione delle operazioni richieste dal metodo implementato:

  `lu, chol, qr`

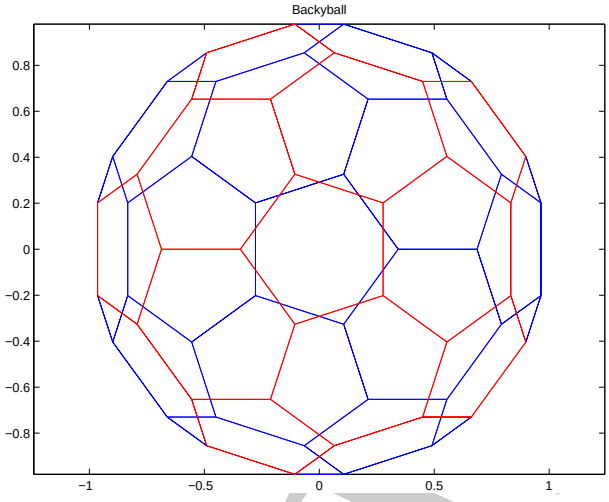


Figura 4.8:

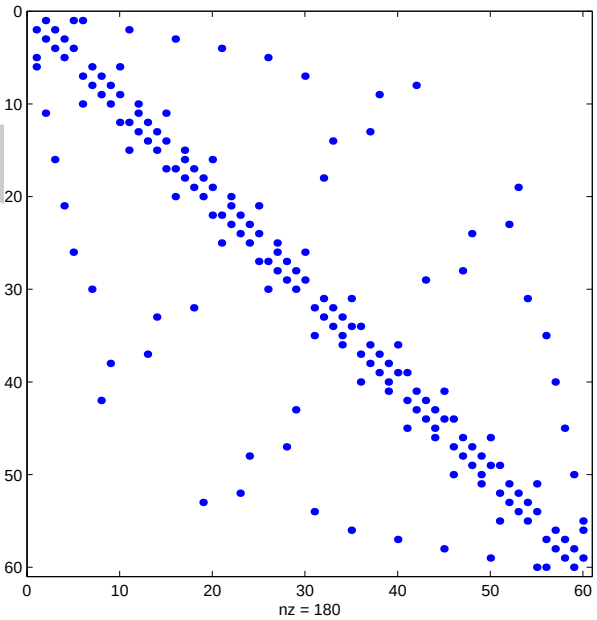


Figura 4.9:

- Infine, `MATLAB` mette a disposizione nove funzioni che implementano i principali **metodi iterativi** per sistemi con matrice dei coefficienti sparsa; tra questi:

  `bicg, cgs, gmres, lsqr, minres, symmlq, ...`

  Approfondimenti relativi a questi ultimi sono reperibili in letteratura (ad esempio in [2] e [11]).

4.17 Esercizi

4.17.1 Quesiti

Quesito 1 Sotto quali condizioni il metodo iterativo di Gauss-Seidel, per la risoluzione di un sistema di equazioni lineari $Ax = b$, converge?

Quesito 2 Il metodo iterativo di Gauss-Seidel è un caso particolare di metodo di tipo **SOR** per la risoluzione di un sistema di equazioni lineari? Perché?

Quesito 3 Quale fattore principale rende poco efficiente l'utilizzo dei metodi diretti basati su fattorizzazione matriciale, nella risoluzione di sistemi di equazioni lineari sparsi e di grandi dimensioni?

Quesito 4 Qual è la formulazione generale di un metodo iterativo *stazionario* per la risoluzione di un sistema di equazioni lineari del tipo $Ax = b$?

Quesito 5

(a) Cosa si intende per *splitting* di una matrice?

(b) Qual è la formulazione del metodo iterativo risultante dallo splitting, per la risoluzione di un sistema di equazioni lineari del tipo $Ax = b$?

(c) Quale condizione imposta sullo splitting garantisce che lo schema iterativo risultante sia convergente?

(d) Assegnata la matrice

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix},$$

qual è la matrice di splitting per il metodo di Jacobi?

(e) Per la stessa matrice in (d), qual è la matrice di splitting per il metodo di Gauss-Seidel?

Quesito 6 Quale dei seguenti metodi iterativi, per la risoluzione di un sistema di equazioni lineari, è *stazionario*?

(a) Jacobi

(b) Gauss-Seidel

(c) SOR

Quesito 7 Descrivere la differenza tra i metodi iterativi di Jacobi e Gauss-Seidel.

(a) Quale dei due metodi converge più rapidamente?

(b) Quale dei due metodi richiede una minore complessità di spazio per la memorizzazione di approssimazioni successive?

Quesito 8 Quali sono i limiti del parametro di rilassamento ω nel metodo SOR?

Quesito 9 Cosa si intende per *precondizionamento*?

4.17.2 Esercizi numerici

Esercizio 1 Fornire vantaggi e svantaggi dei metodi iterativi rispetto ai metodi diretti, nella risoluzione di sistemi di equazioni lineari *sparsi* e di *grandi dimensioni*.

Esercizio 2 Di seguito sono elencate alcune proprietà che caratterizzano i metodi per la risoluzione di sistemi di equazioni lineari. Per ciascuna delle proprietà elencate specificare se essa descrive in maniera più idonea un metodo diretto o un metodo iterativo:

(a) Gli elementi della matrice non sono modificati nel calcolo della soluzione.

(b) Una buona stima a priori per la soluzione è utile.

(c) Gli elementi della matrice sono memorizzati esplicitamente utilizzando uno schema di memorizzazione standard come, ad esempio, un array.

(d) Il costo computazionale richiesto dipende dall'indice di condizionamento del problema.

(e) L'avere risolto un particolare sistema consente di risolvere *facilmente* un altro sistema con la stessa matrice dei coefficienti ma diverso vettore dei termini noti.

(f) Generalmente si utilizzano tecniche di accelerazione o precondizionatori.

(g) Generalmente si ottiene una fattorizzazione della matrice.

(h) La complessità di tempo può essere determinata *a priori*.

Esercizio 3 Sia A una matrice non singolare. Sia L la parte di A *strettamente triangolare inferiore* D la matrice diagonale con elementi uguali a quelli diagonali della matrice A e U la parte di A *strettamente triangolare superiore*.

(a) Esprimere lo schema iterativo di Jacobi in termini di L, D, e U.

(b) Esprimere lo schema iterativo di Gauss-Seidel in termini di L, D, e U.

Esercizio 4 Ordinare i seguenti metodi iterativi per la risoluzione di sistemi di equazioni lineari secondo la loro velocità di convergenza, dal più veloce al più lento:

(a) Gauss-Seidel

(b) Jacobi

(c) SOR con parametro di rilassamento ω ottimale.

Esercizio 5 Dimostrare che il metodo di Jacobi per la risoluzione di un sistema di equazioni lineari $Ax = b$ converge se la matrice dei coefficienti, A, è a diagonale dominante per righe.
[*Suggerimento*: Usare la norma-∞.]

Esercizio 6 Provare che il metodo SOR diverge se ω non appartiene all'intervallo $(0, 2)$.

Esercizio 7 Sia $A \in \mathbb{R}^{n\times n}$ una matrice a diagonale strettamente dominante per righe. Provare che il metodo iterativo di Gauss-Seidel per la risoluzione di un sistema lineare del tipo $Ax = b$ è convergente.

Esercizio 8 Si consideri la matrice *tridiagonale*, del tipo

$$T_\alpha = tridiag(-1, \alpha, -1) = \begin{pmatrix} \alpha & -1 & & & & \\ -1 & \alpha & -1 & & & \\ & -1 & \alpha & -1 & & \\ & & -1 & \alpha & -1 & \\ & & & -1 & \alpha & -1 \\ & & & & -1 & \alpha \end{pmatrix}$$

con $\alpha \in \mathbb{R}$. Gli autovalori di T_α sono esprimibili come:

$$\lambda_j = \alpha - 2\,cos(j\,\theta), \qquad j = 1, \ldots, n$$

dove $\theta = \pi/(n+1)$ e gli autovettori corrispondenti sono:

$$\mathbf{q}_j = [sin(j\,\theta), sin(2\,j\,\theta), \ldots, sin(n\,j\,\theta)]^T$$

Per quali valori di α la matrice è definita positiva?
[*Soluzione*: $\alpha \geq 2$].

Esercizio 9 Si consideri la matrice dell'esercizio precedente. Sia $\alpha = 2$.

- Il metodo iterativo di Jacobi converge se applicato alla risoluzione di un sistema di equazioni con T_2 come matrice dei coefficienti? In caso affermativo, quale è il suo *fattore di convergenza* [24]?

[24] Il *fattore di convergenza* di una sequenza è il limite

$$\rho = \lim_{k\to\infty} \left(\frac{\|d_k\|}{\|d_0\|}\right)^{1/k}$$

Si osserva che ρ coincide con il raggio spettrale della matrice di iterazione di un metodo iterativo stazionario ad un passo.

- Il metodo iterativo di Gauss-Seidel converge se applicato alla risoluzione di un sistema di equazioni con T_2 come matrice dei coefficienti? In caso affermativo, quale sará il suo *fattore di convergenza*?
- Per quali valori di ω il metodo SOR convergerà?

Esercizio 10 Si consideri la matrice pentadiagonale di dimensione n:

$$A = pentadiag_n(-1, -1, 10, -1, -1).$$

Assumiamo che sia: $n = 10$ e $A = M + N + D$, con $D = diag(8, \ldots, 8) \in \mathbb{R}^{10\times 10}$, $M = pentadiag_{10}(-1, -1, 1, 0, 0)$ e $N = M^T$. Si analizzi la convergenza dei metodi iterativi seguenti, nella risoluzione del sistema $Ax = b$.

(a) $(M + D)x^{(k+1)} = -Nx^{(k)} + \mathbf{b}$,

(b) $Dx^{(k+1)} = -(M + N)x^{(k)} + \mathbf{b}$,

(c) $(M + N)x^{(k+1)} = -Dx^{(k)} + \mathbf{b}$.

[*Soluzione*: denotando, rispettivamente con $\rho(a)$, $\rho(b)$ e $\rho(c)$ il raggio spettrale delle matrici di iterazione dei tre metodi, si ha $\rho(a) = 0.1450$, $\rho(b) = 0.5$ e $\rho(c) = 12.2870$, il che implica la convergenza per i metodi (a) e (b) e la divergenza del metodo (c).]

Esercizio 11 Per risolvere il sistema lineare $Ax = b$ con:

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}, \qquad b = \begin{bmatrix} 3 \\ 5 \end{bmatrix},$$

consideriamo il metodo iterativo seguente

$$x^{(k+1)} = B(\theta)x^{(k)} + g(\theta), \quad k \geq 0 \quad \text{e} \quad x^{(0)} \quad \text{assegnata}$$

dove θ è un parametro reale e

$$B(\theta) = \frac{1}{4}\begin{bmatrix} 2\theta^2 + 2\theta + 1 & -2\theta^2 + 2\theta + 1 \\ -2\theta^2 + 2\theta + 1 & 2\theta^2 + 2\theta + 1 \end{bmatrix}, \qquad g(\theta) = \begin{bmatrix} \frac{1}{2} - \theta \\ \frac{1}{2} - \theta \end{bmatrix}.$$

Provare che il metodo è consistente $\forall\, \theta \in \mathbb{R}$. Determinare i valori di θ per cui il metodo è convergente e calcolare il valore *ottimale* di θ (i.e. il valore del parametro per cui la *velocità asintotica di convergenza* risulta massima).
[*Soluzione*: il metodo è convergente se, e solo se, $-1 < \theta < 1/2$ e la *velocità asintotica di convergenza* è massima se $\theta = (1 - \sqrt{3})/2$.]

Esercizio 12 Per risolvere il seguente sistema lineare a blocchi

$$\begin{bmatrix} A_1 & B \\ B & A_2 \end{bmatrix}\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

consideriamo i due metodi

(1) $A_1x^{(k+1)} + By^{(k)} = b_1,\ Bx^{(k)} + A_2y^{(k+1)} = b_2;$

(2) $A_1x^{(k+1)} + By^{(k)} = b_1,\ Bx^{(k+1)} + A_2y^{(k+1)} = b_2.$

Trovare condizioni sufficienti che garantiscano la convergenza dei due schemi, per qualsiasi scelta dei dati iniziali: $x^{(0)}, y^{(0)}$.
[*Soluzione*: il metodo (1) è un sistema (disaccoppiato) nelle incognite $x^{(k+1)}$ e $y^{(k+1)}$. Assumendo che A_1 e A_2 siano invertibili, il metodo (1) converge se $\rho(A_1^{-1}B) < 1$ e $\rho(A_2^{-1}B) < 1$.
A proposito del metodo (2) abbiamo un sistema (accoppiato) da risolvere ad ogni passo, nelle incognite $x^{(k+1)}$ e $y^{(k+1)}$. Risolvendo, formalmente, la prima equazione rispetto a $x^{(k+1)}$ (il che richiede che A_1 sia invertibile) e sostituendo nella seconda, si riscontra che il metodo (2) è convergente se $\rho(A_2^{-1}BA_1^{-1}B) < 1$ (A_2 deve essere invertibile).]

Esercizio 13 Consideriamo il sistema lineare $Ax = b$ con

$$A = \begin{bmatrix} 62 & 24 & 1 & 8 & 15 \\ 23 & 50 & 7 & 14 & 16 \\ 4 & 6 & 58 & 20 & 22 \\ 10 & 12 & 19 & 66 & 3 \\ 11 & 18 & 25 & 2 & 54 \end{bmatrix}, \qquad b = \begin{bmatrix} 110 \\ 110 \\ 110 \\ 110 \\ 110 \end{bmatrix}.$$

Verificare se i metodi di Jacobi e Gauss-Seidel possono essere applicati per risolvere il sistema.
[*Soluzione*: La matrice A non è a diagonale dominante nè simmetrica definita positiva, per cui bisogna calcolare il raggio spettrale delle matrici di iterazione dei metodi di Jacobi e Gauss-Seidel per verificare se sono convergenti. Si riscontra che: $\rho_J = 0.9280$ e che $\rho_{GS} = 0.3066$, il che implica la convergenza per entrambi i metodi.]

Esercizio 14 Consideriamo il sistema lineare $Ax = b$ con

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}, \qquad b = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix}.$$

Analizzare le proprietà di convergenza dei metodi di Jacobi e Gauss-Seidel applicati al sistema nelle loro formulazioni *puntuali* e *a blocchi* (partizionando la matrice A in blocchi 2×2).
[*Soluzione*: entrambi i metodi sono convergenti, essendo, in particolare, la forma a blocchi la più *veloce*. Inoltre: $\rho^2(B_J) = \rho(B_{GS})$.]

Esercizio 15 Per risolvere il sistema lineare $Ax = b$, consideriamo il metodo iterativo

$$Mx^{(k+1)} = Rx^{(k)} + \mathbf{b} \qquad k \geq 0$$

dove le matrici M e R si ottengono mediante uno *splitting* della matrice A: $A = M - R$; sia $M = D + \omega F$ e $R = -\beta F - E$, con ω e β numeri reali. Provare che il metodo è consistente solo se $\beta = 1 - \omega$. In tal caso esprimere gli autovalori della matrice di iterazione in funzione di ω e determinare per quali valori di ω il metodo è convergente. Calcolare, inoltre, il valore ottimale ω_{opt}, assumendo che sia $A = tridiag_{10}(-1, 2, -1)$.
[*Suggerimento*: Sfruttare il risultato dell'**Esercizio 8**.]

Esercizio 16 Sia $A \in \mathbb{R}^{n\times n}$ tale che $A = (1+\omega)P-(N+\omega P)$, con $P^{-1}N$ non singolare e con autovalori reali $1 > \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Determinare i valori del parametro $\omega \in \mathbb{R}$ in corrispondenza dei quali il metodo iterativo seguente:

$$(1+\omega)Px^{(k+1)} = (N+\omega P)x^{(k)} + \mathbf{b}, \qquad k \geq 0,$$

converge, $\forall\, x^{(0)}$, alla soluzione del sistema lineare $Ax = b$. Determinare, inoltre, il valore di ω per cui la velocità asintotica di convergenza risulta massima.
[*Soluzione*: $\omega > -(1+\lambda_n)/2$; $\omega_{opt} = -(\lambda_1 + \lambda_n)/2$.]

Esercizio 17 Si consideri il sistema lineare:

$$\begin{cases} 5x + 3y = 6 \\ 4x - 2y = 8 \end{cases}$$

È possibile utilizzare sia il metodo di Jacobi che quello di Gauss-Seidel per risolvere tale sistema? Perché?

Esercizio 18 L'iterazione di Jacobi può sempre essere usata per calcolare la soluzione del sistema seguente?

$$\begin{cases} 2x + y - 5z = 9 \\ x - 5y - z = 14 \\ 7x - y - 3z = 26 \end{cases}$$

Esercizio 19 Si consideri il seguente sistema lineare *tridiagonale* e si assuma che la matrice dei coefficienti sia *a diagonale dominante*:

$$\begin{cases} d_1x_1 + c_1x_2 & & & & & & = b_1 \\ a_1x_1 + d_2x_2 + c_2x_3 & & & & & & = b_2 \\ \quad a_2x_2 + d_3x_3 + c_3x_4 & & & & & & = b_3 \\ \quad\quad \ddots \quad\quad \ddots & & \ddots & & & & \vdots \\ & & a_{N-2}x_{N-2} + & d_{N-1}x_{N-1} + & c_{N-1}x_N & & = b_{N-1} \\ & & & a_{N-1}x_{N-1} + & d_Nx_N & & = b_N \end{cases}$$

Scrivere un algoritmo che descriva un metodo iterativo per la risoluzione di tale sistema.

Esercizio 20 Dimostrare che, se A è *a diagonale dominante* e se Q è la matrice diagonale i cui elementi diagonali sono gli elementi di A, allora

$$\rho(I - Q^{-1}A) < 1$$

[*Suggerimento*: Basta applicare il **Teorema di Gershgorin** (cfr. **Teorema B.6** dell'**Appendice B**, **Parte 1**) alla matrice $I - Q^{-1}A$ e ricordare la definizione di raggio spettrale.]

Esercizio 21 Dimostrare che il processo iterativo di base, del tipo:

$$Qx^{(k)} = (Q - A)x^{(k-1)} + b \qquad (k \geq 1)$$

è equivalente al seguente:

Assegnata $x^{(k)}$,

calcolare $r^{(k)} = b - Ax^{(k)}$,

risolvere rispetto a $z^{(k)}$ l'equazione $Qz^{(k)} = r^{(k)}$,

definire $x^{(k+1)} = x^{(k)} + z^{(k)}$.

Esercizio 22 Usando la notazione dell'esercizio precedente, dimostrare che

$$r^{(k+1)} = (I - AQ^{-1})r^{(k)}$$
$$z^{(k+1)} = (I - Q^{-1}A)z^{(k)}$$

Esercizio 23 Dimostrare che il metodo di Gauss-Seidel è un caso particolare di metodo SOR.

Esercizio 24 Siano $D = diag(A)$, $-C_L$ la parte strettamente triangolare inferiore di A (ovvero la matrice triangolare inferiore costituita dagli elementi di A sotto la diagonale principale) e $-C_U$ la parte strettamente triangolare superiore di A. Dimostrare che le matrici

$$\mathcal{J} = I - D^{-1}A$$
$$\mathcal{G} = I - (D - C_L)^{-1}A$$
$$\mathcal{S}_\omega = I - \omega(2 - \omega)(D - \omega C_U)^{-1}D(D - \omega C_L)^{-1}A$$

sono le matrici di iterazione per i metodi di Jacobi, Gauss-Seidel e SSOR, rispettivamente.

Esercizio 25 Trovare la forma esplicita della matrice di iterazione $I-Q^{-1}A$ del metodo di Gauss-Seidel quando

$$A = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}$$

Esercizio 26 Caratterizzare la famiglia di tutte le matrici $n \times n$ non singolari, A, per le quali l'algoritmo per il metodo di Gauss-Seidel risolve un sistema $Ax = b$ con *una sola iterazione*, a partire da $x^{(0)} = 0$.

Esercizio 27 Fornire un esempio di matrice, A, che non sia a diagonale dominante, in corrispondenza della quale, tuttavia, il metodo di Gauss-Seidel applicato ad un sistema $Ax = b$ converge.

Esercizio 28 Come si semplifica la tecnica di accelerazione di Chebyshev se il metodo iterativo di base è il metodo di Jacobi?

Esercizio 29 Dimostrare che, se il numero $\delta = \|I - Q^{-1}A\|$ è minore di 1, allora

$$\|x^{(k)} - x\| \leq \frac{\delta}{1-\delta}\|x^{(k)} - x^{(k-1)}\|$$

Esercizio 30 Sia $A \in \Re^{8\times 8}$:

$$A = \begin{bmatrix} 8 & -2 & 0 & 0 & \cdots \\ -2 & 8 & -2 & 0 & \cdots \\ 0 & -2 & 8 & -2 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & -2 & 8 \end{bmatrix}$$

la matrice dei coefficienti di un sistema lineare:

$$Ax = b.$$

1. Verificare che il metodo di Jacobi è convergente.
2. Verificare che il metodo di Gauss-Seidel è convergente.
3. Detta C_J la matrice di iterazione del metodo di Jacobi e C_{GS} quella del metodo di Gauss-Seidel, verificare che tra i raggi spettrali di tali matrici sussiste la relazione:
$$\rho(C_{GS}) = \rho^2(C_J).$$

Esercizio 31 Sia $A \in \Re^{8\times 8}$:

$$A = \begin{bmatrix} 7 & 1 & 0 & 0 & \cdots \\ 1 & 7 & 1 & 0 & \cdots \\ 0 & 1 & 7 & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 1 & 7 \end{bmatrix}$$

la matrice dei coefficienti di un sistema lineare:

$$Ax = b.$$

Supponendo di utilizzare il metodo iterativo SOR:

1. verificare che il metodo è convergente;
2. trovare il valore ottimale del parametro di rilassamento ω_{opt} del metodo SOR;
3. calcolare la matrice di iterazione $C_{\omega_{opt}}$ del metodo SOR.

Esercizio 32 Sia $A \in \Re^{3\times 3}$:

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

la matrice dei coefficienti di un sistema lineare:

$$Ax = b.$$

Supponendo di utilizzare il metodo iterativo di Jacobi:

1. verificare che il metodo è convergente;
2. costruire la matrice di iterazione C_J;
3. determinare il numero k minimo di iterazioni affinché l'errore sulla soluzione sia minore di 10^{-5}.

Esercizio 33 Sia $A \in \Re^{3\times 3}$:

$$A = \begin{bmatrix} 10 & -3 & 1 \\ -3 & 10 & 4 \\ 1 & 4 & 10 \end{bmatrix}$$

la matrice dei coefficienti di un sistema lineare:

$$Ax = b.$$

Supponendo di utilizzare il metodo iterativo di Gauss-Seidel:

1. verificare che il metodo è convergente;

2. costruire la matrice di iterazione C_{GS};
3. determinare il numero k minimo di iterazioni affinché l'errore sulla soluzione sia minore di 10^{-5}.

Esercizio 34 Si consideri un sistema lineare:

$$Ax = b, \qquad b \in \Re^3$$

dove la matrice dei coefficienti è

$$A = \begin{bmatrix} -1 & 0 & a \\ f & -1 & 0 \\ 0 & c & -1 \end{bmatrix},$$

con a, f, e c numeri reali. Studiare la convergenza del metodo iterativo di Gauss-Seidel al variare dei parametri a, f, e c.

4.17.3 Problemi da risolvere con il calcolatore

Problema 1 Implementare almeno due metodi iterativi per la risolvere un sistema lineare con matrice dei coefficienti simmetrica definita positiva. Confrontarne le prestazioni sia in termini di *velocità asintotica di convergenza* che di complessità di tempo totale, risolvendo un campione significativo di problemi test, sia ben-condizionati che mal-condizionati. Confrontare i risultati ottenuti con le stime teoriche per la *velocità asintotica di convergenza*.

Problema 2 Implementare il metodo di Jacobi per risolvere un sistema lineare $Ax = b$, $n \times n$, dove A è tridiagonale con elementi diagonali uguali a 2 e con elementi sub-diagonali and superdiagonali uguali a -1. Si assuma $\mathbf{b} = \mathbf{0}$, così che la soluzione "esatta" sia $\mathbf{x} = \mathbf{0}$ e ad ogni iterazione l'errore sia uguale al valore corrente di $\mathbf{x}$. Come valore inizale si assuma

$$x_j = sin\left(\frac{jk\pi}{n+1}\right), \qquad j = 1, \ldots n.$$

Problema 3 Sviluppare un elemento di software matematico che implementi il metodo iterativo di Gauss-Seidel e testarlo per la risoluzione dei seguenti sistemi:

a.
$$\begin{cases} 3x + y + z = 5 \\ x + 3y - z = 3 \\ 3x + y - 5z = -1 \end{cases}$$

b.
$$\begin{cases} 3x + y + z = 5 \\ 3x + y - 5z = -1 \\ x + 3y - z = 3 \end{cases}$$

Analizzare cosa accade quando i due sistemi sono risolti mediante metodo di eliminazione di Gauss *senza* pivoting.

Problema 4 Applicare il metodo iterativo di Gauss-Seidel al sistema $Ax = b$ con:

$$A = \begin{bmatrix} 0.96326 & 0.81321 \\ 0.81321 & 0.68654 \end{bmatrix} \qquad b = \begin{bmatrix} 0.88824 \\ 0.74988 \end{bmatrix}$$

Utilizzare, come punto iniziale, $(0.33116, 0.7)^T$ e spiegare cosa accade.

Problema 5 Per ciascuno dei sistemi seguenti:

1. $\begin{cases} 4x - y = 15 \\ x + 5y = 9 \\ 3x + y - 5z = -1 \end{cases}$

2. $\begin{cases} 8x - 3y = 10 \\ -x + 4y = 6 \end{cases}$

3. $\begin{cases} -x + 3y = 1 \\ 6x - 2y = 2 \end{cases}$

4. $\begin{cases} 2x + 3y = 1 \\ 7x - 2y = 1 \end{cases}$

5. $\begin{cases} 5x - y + z = 10 \\ 2x + 8y - z = 11 \end{cases}$

6. $\begin{cases} 2x + 8y - z = 11 \\ 5x - y + z = 10 \\ -x + y + 4z = 3 \end{cases}$

7. $\begin{cases} x - 5y - z = -8 \\ 4x + y - z = 13 \\ 2x - y - 6z = -2 \end{cases}$

8. $\begin{cases} 4x + y - z = 13 \\ x - 5y - z = -8 \\ 2x - y - 6z = -2 \end{cases}$

- porre $x^{(0)} = \underline{0}$ ed utilizzare i metodo di Jacobi per calcolare le iterate: $x^{(1)}$, $x^{(2)}$ e $x^{(3)}$. Il metodo converge verso la soluzione?
- Porre $x^{(0)} = \underline{0}$ ed utilizzare i metodo di Gauss-Seidel per calcolare le iterate: $x^{(1)}$, $x^{(2)}$ e $x^{(3)}$. Il metodo converge verso la soluzione?

- Scrivere un elemento di software matematico che implementi i metodi di Jacobi e Gauss-Seidel.

Problema 6 Calcolare la soluzione dei seguenti sistemi *tridiagonali*:

a.
$$\begin{cases} 4m_1+ & m_2 & & = 3 \\ m_1+ & 4m_2+ & m_3 & = 3 \\ m_2+ & 4m_3+ & m_4 & = 3 \\ m_3+ & 4m_4+ & m_5 & = 3 \\ \vdots & \vdots & \vdots & \vdots \\ m_{48}+ & 4m_{49}+ & m_{50} & = 3 \\ & m_{49}+ & 4m_{50} & = 3 \end{cases}$$

b.
$$\begin{cases} 4m_1+ & m_2 & & = 1 \\ m_1+ & 4m_2+ & m_3 & = 2 \\ m_2+ & 4m_3+ & m_4 & = 1 \\ m_3+ & 4m_4+ & m_5 & = 2 \\ \vdots & \vdots & \vdots & \vdots \\ m_{48}+ & 4m_{49}+ & m_{50} & = 1 \\ & m_{49}+ & 4m_{50} & = 2 \end{cases}$$

Problema 7 Utilizzare il metodo iterativo di Gauss-Seidel per risolvere il sistema lineare *a banda* seguente:

$$\begin{cases} 12x_1 & -2x_2 & +x_3 & & & = 5 \\ -2x_1 & +12x_2 & -2x_3 & +x_4 & & = 5 \\ x_1 & -2x_2 & +12x_3 & -2x_4 & +x_5 & = 5 \\ x_2 & -2x_3 & +12x_4 & -2x_5 & +x_6 & = 5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{46} & -2x_{47} & +12x_{48} & -2x_{49} & +x_{50} & = 5 \\ & x_{47} & -2x_{48} & +12x_{49} & -2x_{50} & = 5 \\ & & x_{48} & -2x_{49} & +12x_{50} & = 5 \end{cases}$$

Problema 8 Si consideri un sistema lineare:

$$Ax = b$$

dove

$$A = \begin{bmatrix} 10 & -1 & 2 & 3 \\ 1 & -1 & -1 & 2 \\ 2 & 3 & 20 & -1 \\ 3 & 2 & 1 & 20 \end{bmatrix} \quad \text{e} \quad b = \begin{bmatrix} 14 \\ 12 \\ 24 \\ 26 \end{bmatrix}.$$

Dopo avere discusso la convergenza dei metodi iterativi di Jacobi e di Gauss-Seidel per ciascun metodo, sviluppare una function `MATLAB` per risolvere il sistema lineare. Inoltre,

1. determinare le velocità asintotiche di convergenza per ciascun metodo;
2. verificare i risultati ottenuti con le stime sperimentali dedotte applicando le function `MATLAB` elaborate.

Bibliografia

[1] Axelsson O. - *Iterative Solution Methods* - Cambridge University Press, 1994.

[2] Barlett R. et al. - *Templates for the Solution of Linear Systems. Building Blocks for Iterative Methods* - SIAM, Philadelphia, 1993.

[3] Cheney W., Kincaid D.R. - *Numerical Mathematics and Computing* - 3rd ed., Pacific Grove, CA, Brooks/Cole, 1994.

[4] Greenbaumm A. - *Iterative Methods for Solving Linear Systems* - SIAM, Philadelphia, 1997.

[5] Hageman L.A., Young D.M. - *Applied Iterative Methods* - Academic Press, 1981.

[6] Kelley C.T. - *Iterative Methods for Linear and Nonlinear Equations* - SIAM, Philadelphia, 1995.

[7] Lancaster P., Tismenetsky M. - *The Theory of Matrices* - second edition, Academic Press, 1985.

[8] Ortega J.M., Rheinboldt W.C. - *Iterative Solution of Nonlinear Equations in Several Veriables* - Academic Press, 1970.

[9] Ortega J.M. - *Matrix Theory* - Plenum Press, 1988.

[10] Rice J. - *Matrix Computation and mathematical software*, McGrawHill, 1981.

[11] Saad Y. - *Iterative Methods for Sparse Linear Systems* - PWS Publishing Co, Boston, 2003.

[12] Stewart G.W. - *Afternotes on Numerical Analysis* - SIAM, Philadelphia, 1998.

[13] Trefethen L.N., Bau D. - *Numerical Linear Algebra* - SIAM, Philadelphia, 1997.

[14] Varga R.S. - *Matrix Iterative Analysis* - Prentice Hall, Englewood Cliffs, NJ, 1962.

[15] Van der Vost H.A. - *Iterative Krylov methods for large linear systems,* Cambridge, 2003.

[16] Young D.M. - *Iterative Solution of Large Linear Systems* - Academic Press, NY, 1971.