Corso di Metodi Numerici per l'Ingegneria

Progetto numerico al calcolatore

Parte I

Implementazione del metodo del Gradiente Coniugato Modificato (GCM) per la soluzione di sistemi lineari sparsi, simmetrici e definiti positivi

Indice

1 **Richiami dello schema del GCM** 1

2 **Memorizzazione di una matrice in forma compatta** 4

3 **Memorizzazione compatta di una matrice in Matlab** 6

3.1 Formato CSC 6

3.2 Principali comandi per gestire matrici sparse 7

3.3 Creare matrici sparse 8

4 **Calcolo e applicazione del precondizionatore** 10

4.1 Fattorizzazione incompleta di Cholesky 11

4.2 La fattorizzazione incompleta di Cholesky in Matlab 12

1 Richiami dello schema del GCM

Il metodo del Gradiente Coniugato Modificato (GCM) si basa sulla tecnica del Gradiente Coniugato (GC) sviluppata nei primi anni '50 da Hestenes e Stiefel per la soluzione di sistemi lineari con matrice simmetrica e definita positiva. Secondo il metodo del GC, la soluzione del generico sistema:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

si ottiene costruendo una successione di approssimazioni successive $\boldsymbol{x}_k$ a partire da un vettore iniziale arbitrario $\boldsymbol{x}_0$ secondo la formula ricorrente:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \tag{2}$$

Lo scalare α_k viene scelto in modo da minimizzare una opportuna misura Φ dell'errore commesso assumendo $\boldsymbol{x}_{k+1}$ come soluzione del sistema (1). Adottando come misura dell'errore la seguente forma quadratica:

$$\Phi(\boldsymbol{x}_{k+1}) = \frac{1}{2}\boldsymbol{e}_{k+1}^T A \boldsymbol{e}_{k+1} = \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}_{k+1}\right)^T A \left(\boldsymbol{x} - \boldsymbol{x}_{k+1}\right) \tag{3}$$

si ottiene:

$$\alpha_k = \frac{\boldsymbol{p}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \tag{4}$$

dove $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ è il residuo associato alla k-esima iterazione. Il nuovo residuo relativo alla soluzione $\boldsymbol{x}_{k+1}$ si ricava facilmente dalla (2):

$$\boldsymbol{r}_{k+1} = \boldsymbol{b} - A\left(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k\right) = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k \tag{5}$$

La direzione di ricerca $\boldsymbol{p}_{k+1}$ utilizzata per calcolare la nuova approssimazione della soluzione viene determinata A-ortogonalizzando $\boldsymbol{p}_{k+1}$ rispetto a $\boldsymbol{p}_k$ secondo la formula ricorrente:

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k \tag{6}$$

Imponendo la condizione di A-ortogonalità fra i vettori $\boldsymbol{p}_k$ e $\boldsymbol{p}_{k+1}$:

$$\boldsymbol{p}_{k+1}^T A \boldsymbol{p}_k = 0 \tag{7}$$

si ottiene la relazione che definisce lo scalare β_k:

$$\beta_k = -\frac{\boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \tag{8}$$

Lo schema del GC va inizializzato scegliendo un vettore soluzione arbitrario $\boldsymbol{x}_0$, il corrispondente residuo $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ e ponendo $\boldsymbol{p}_0 = \boldsymbol{r}_0$. Si applicano quindi in successione le formule ricorrenti (2), (5) e (6) mediante le (4) e (8) finché una opportuna norma del residuo, ad esempio la norma euclidea, risulta inferiore a una prefissata tolleranza. Si osservi che è generalmente molto più conveniente fissare

la tolleranza in funzione della norma del termine noto. Il residuo, quindi, su cui viene effettivamente eseguito il controllo sarà quello relativo:

$$r_r = \frac{\|\boldsymbol{r}_{k+1}\|}{\|\boldsymbol{b}\|} < \varepsilon \tag{9}$$

dove ε è un parametro adimensionale scelto dall'utente. L'ultima approssimazione $\boldsymbol{x}^*$ del vettore $\boldsymbol{x}$ così ottenuta viene assunta come soluzione del sistema lineare (1). Va osservato che, specialmente in problemi mal-condizionati, gli errori di arrotondamento che si accumulano nel calcolo del residuo mediante la formula ricorrente (5) possono far sì che quest'ultimo sia in realtà anche molto diverso dal residuo vero e che, di conseguenza, $\boldsymbol{x}^*$ sia lontano dalla soluzione esatta del sistema (1). Per evitare tale inconveniente, è opportuno calcolare al termine della procedura iterativa del GC il residuo relativo vero r_r^*:

$$r_r^* = \frac{\|\boldsymbol{b} - A\boldsymbol{x}^*\|}{\|\boldsymbol{b}\|} \tag{10}$$

e verificare che sia effettivamente dell'ordine della tolleranza prefissata.

Si può dimostrare che il vettore $\boldsymbol{r}_{k+1}$ risulta ortogonale ai precedenti $\boldsymbol{r}_0, \ldots, \boldsymbol{r}_k$ e che, pertanto, il residuo n-esimo, avendo indicato con n la dimensione del sistema (1), deve necessariamente essere nullo. A causa degli errori di arrotondamento del calcolatore, tuttavia, ciò non si verifica e il GC, pur essendo teoricamente un metodo diretto, va catalogato fra le tecniche iterative. In conseguenza a questo aspetto, che incide negativamente sulla performance del solutore, il GC, se non opportunamente accelerato, non offre alcun vantaggio rispetto ai metodi diretti classici.

La convergenza del GC risulta particolarmente accelerata qualora gli autovalori della matrice A siano raggruppati attorno a un unico valore non nullo, il che corrisponde a interpretare geometricamente l'iper-ellissoide associato alla matrice A come quasi-sferico. Si può, pertanto, cercare di risolvere un nuovo sistema:

$$B\boldsymbol{y} = \boldsymbol{c} \tag{11}$$

equivalente al precedente, in cui, tuttavia, la matrice B presenta i propri autovalori raggruppati attorno all'unità. Il nuovo sistema (11), detto *sistema precondizionato*, è collegato al sistema originale (1) mediante le seguenti relazioni:

$$B = X^{-1}AX^{-1} \qquad \boldsymbol{y} = X\boldsymbol{x} \qquad \boldsymbol{c} = X^{-1}\boldsymbol{b} \tag{12}$$

La matrice $K^{-1} = X^{-1}X^{-1}$ è detta *matrice di precondizionamento*. Se si sceglie $K^{-1} = A^{-1}$ si osserva facilmente che la matrice B coincide con l'identità e la soluzione di (11) è immediata. È ovvio che tale opportunità è del tutto teorica, in quanto se si conoscesse già l'inversa di A non servirebbe ricorrere ad alcun metodo per risolvere il sistema (1). Tuttavia, questa osservazione ci permette di concludere che la matrice di precondizionamento prescelta sarà tanto più efficace quanto più il prodotto AK^{-1} si avvicinerà da un punto di vista spettrale all'identità.

Riscrivendo le equazioni del GC per il sistema (11) e successivamente ripristinando le variabili originali, si ottiene lo schema del Gradiente Coniugato Modificato (GCM):

$$
\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k \\
\boldsymbol{p}_{k+1} &= K^{-1} \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k
\end{aligned}
\tag{13}
$$

dove:

$$
\alpha_k = \frac{\boldsymbol{p}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \qquad \beta_k = -\frac{\boldsymbol{r}_{k+1}^T K^{-1} A \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \tag{14}
$$

Lo schema del GCM va inizializzato partendo da un vettore arbitrario $\boldsymbol{x}_0$, con il relativo residuo $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, e dalla direzione $\boldsymbol{p}_0 = K^{-1}\boldsymbol{r}_0$. La soluzione iniziale può essere migliorata con alcune (poche) iterazioni effettuate con lo schema delle Correzioni Residue (CR), che coincide con lo schema di Richardson applicato al sistema (11) assumendo che il coefficiente di rilassamento α sia unitario:

$$
\begin{aligned}
\boldsymbol{y}_0 &= \boldsymbol{c} \\
\boldsymbol{y}_1 &= (I - B)\, \boldsymbol{y}_0 + \boldsymbol{c} \\
&\vdots \\
\boldsymbol{y}_{k+1} &= (I - B)\, \boldsymbol{y}_k + \boldsymbol{c}
\end{aligned}
\tag{15}
$$

Espresso nelle variabili originali, si verifica facilmente che lo schema (15) risulta:

$$
\begin{aligned}
\boldsymbol{x}_0 &= K^{-1} \boldsymbol{b} \\
\boldsymbol{x}_1 &= \boldsymbol{x}_0 + K^{-1} \boldsymbol{r}_0 \\
&\vdots \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + K^{-1} \boldsymbol{r}_k
\end{aligned}
\tag{16}
$$

Lo schema del GCM si colloca, allo stato attuale, fra i metodi più moderni ed efficienti per la soluzione di sistemi lineari sparsi, simmetrici e definiti positivi, quali tipicamente si possono derivare dall'applicazione del metodo degli elementi finiti in svariati problemi di ingegneria civile, ambientale, meccanica, ecc. Uno dei punti vincenti del GCM sta nella semplicità di implementazione e nel ridotto costo computazionale rischiesto a ogni iterazione, essendo le operazioni più dispendiose limitate a un prodotto matrice-vettore ($A\boldsymbol{p}_k$) e una applicazione del precondizionatore ($K^{-1}\boldsymbol{r}_{k+1}$). Il GCM può essere implementato mediante il seguente algoritmo:

001 Input: $\boldsymbol{x}$, K^{-1}, ε, $k_{\max}$

002 $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$

003 $\boldsymbol{p} = K^{-1}\boldsymbol{r}$

004 $k = 0$

005 $\tau = \|\boldsymbol{r}\|$

006 **Finché** $\tau > \varepsilon\|\boldsymbol{b}\|$ e $k < k_{\max}$ **esegui**

007 $k \leftarrow k + 1$

008 $\boldsymbol{t} = A\boldsymbol{p}$

009 $\gamma = \boldsymbol{p}^T \boldsymbol{t}$

010 $\alpha = \boldsymbol{p}^T \boldsymbol{r}/\gamma$

011 $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha\boldsymbol{p}$

012 $\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha\boldsymbol{t}$

013 $\boldsymbol{v} = K^{-1}\boldsymbol{r}$

014 $\beta = -\boldsymbol{v}^T \boldsymbol{t}/\gamma$

015 $\boldsymbol{p} \leftarrow \boldsymbol{v} + \beta\boldsymbol{p}$

016 $\tau = \|\boldsymbol{r}\|$

017 **Fine Finché**

È, quindi, sufficiente memorizzare 5 vettori ($\boldsymbol{x}$, $\boldsymbol{r}$, $\boldsymbol{p}$, $\boldsymbol{t}$ e $\boldsymbol{v}$) e costruire una function in grado di applicare il precondizionatore al vettore residuo. Come output, può essere utile conservare anche tutti i valori di τ calcolati a ogni iterazione, in modo da poter costruire l'andamento di $r_r = \|\boldsymbol{r}\|/\|\boldsymbol{b}\|$ in scala semilogaritmica rispetto al numero di iterazioni. Tale diagramma è generalmente indicato come *profilo di convergenza* del GCM. Tuttavia, per sfruttare appieno da un punto di vista computazionale le potenzialità del GCM è opportuno memorizzare la matrice del sistema non in forma tabellare *densa*, bensì in forma compatta *sparsa*.

2 Memorizzazione di una matrice in forma compatta

Le matrici tipicamente ottenute dalla discretizzazione di problemi ingegneristici sono molto sparse, con una percentuale di elementi nulli (grado di sparsità) spesso ben superiore al 99%. Risulta pertanto molto più conveniente da un punto di vista computazionale memorizzare la matrice A in forma compatta, escludendo, cioè, tutti i coefficienti nulli. Oltre a un ragguardevole risparmio di memoria, questa tecnica di memorizzazione consente anche di trascurare tutti i prodotti in cui uno dei fattori è a priori nullo e il cui risultato è ovviamente già noto.

Esistono varie convenzioni per memorizzare una matrice sparsa, la cui convenienza dipende soprattutto dall'hardware su cui un algoritmo viene implementato. A titolo esemplificativo, vediamo come memorizzare una matrice sparsa secondo il formato Compressed Sparse Row (CSR). Data una

generica matrice A quadrata di ordine n, la tecnica di memorizzazione CSR prevede la generazione di 3 vettori:

1. `COEF`: vettore di numeri reali contenente gli nt coefficienti non nulli della matrice A memorizzati in successione per righe;

2. `JA`: vettore di numeri interi contenente gli nt indici di colonna dei corrispondenti elementi memorizzati in `COEF`;

3. `TOPOL`: vettore di numeri interi con $n+1$ componenti, contenente le posizioni in cui si trova in `COEF` il primo elemento di ciascuna riga di A.

Il vettore `TOPOL` è anche chiamato "vettore topologico". L'uso congiunto di `TOPOL` e `JA` consente di individuare qualsiasi elemento non nullo a_{ij} memorizzato in `COEF`. Infatti, l'elemento a_{ij} si troverà in una posizione k del vettore `COEF` compresa nell'intervallo $\texttt{TOPOL}(i) \leq k \leq \texttt{TOPOL}(i+1)-1$ e tale per cui $\texttt{JA}(k) = j$. Queste due condizioni permettono di determinare univocamente l'indice k per cui $\texttt{COEF}(k) = a_{ij}$. Si noti che l'occupazione di memoria si riduce da n^2 numeri reali, generalmente in doppia precisione, a nt numeri reali e $nt+n+1$ numeri interi, con $nt \ll n^2$.

Facciamo un esempio numerico per rendere più chiara la tecnica CSR. Si consideri la seguente matrice A di dimensione 6×6, sparsa e non simmetrica:

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

Gli elementi non nulli sono $nt = 18$, mentre $n^2 = 36$, pertanto il grado di sparsità di A è il 50%. Trascurando i coefficienti nulli, la matrice A diventa:

$$A' = \begin{bmatrix} 4 & & 7 & & & \\ 3 & 4 & & 2 & & \\ & 2 & 4 & & 4 & \\ & & 7 & 4 & & 1 \\ 1 & & & 3 & 4 & \\ 1 & 1 & & & 3 & 4 \end{bmatrix}$$

Il vettore `COEF` ha dimensione 18 e risulta:

$$\overbrace{4 \quad 7 \quad 3 \quad 4 \quad 2 \quad 2 \quad 4 \quad 4 \quad 7 \quad 4 \quad 1 \quad 1 \quad 3 \quad 4 \quad 1 \quad 1 \quad 3 \quad 4}^{nt}$$

Il vettore `JA` ha la stessa dimensione di `COEF` e per ogni coefficiente di quest'ultimo individua l'indice di colonna:

$$\overbrace{1 \quad 3 \quad 1 \quad 2 \quad 4 \quad 2 \quad 3 \quad 5 \quad 3 \quad 4 \quad 6 \quad 1 \quad 4 \quad 5 \quad 1 \quad 2 \quad 5 \quad 6}^{nt}$$

Il vettore `TOPOL`, infine, individua la posizione in `COEF` del primo coefficiente di ciascuna riga:

$$\overbrace{1 \quad 3 \quad 6 \quad 9 \quad 12 \quad 15 \quad 19}^{n+1}$$

Si noti che che le componenti di `TOPOL` sono necessariamente ordinate in senso strettamente crescente e che $\texttt{TOPOL}(n+1) = nt+1$. Ciò si deduce immediatamente dal fatto che per individuare un elemento a_{ij} dell'ultima riga di A si deve cercare l'indice k della componente in `COEF` nell'intervallo $\texttt{TOPOL}(n) \leq k \leq \texttt{TOPOL}(n+1) - 1$.

3 Memorizzazione compatta di una matrice in Matlab

In Matlab esistono due formati di memorizzazione delle matrici: denso (`full`) o sparso (`sparse`). Quindi, due variabili `A` e `B` possono avere diverse classi di memorizzazione, ma rappresentare la stessa matrice (stesso determinante, stessi autovalori, ecc.). Le due variabili, tuttavia, occupano una differente quantità di memoria. Il formato di memorizzazione compatta di una matrice sparsa si basa sulla convenzione Compressed Sparse Column (CSC), che, di fatto, corrisponde al formato CSR in cui si ordinano gli elementi della matrice per colonne anziché per righe.

3.1 Formato CSC

Nonostante sia scritto principalmente in C, Matlab si basa interamente sulla libreria Linpack, scritta in Fortran negli anni '70, che costituisce il linguaggio vincolante per la memorizzazione delle matrici. Questo spiega perché Matlab memorizzi le matrici dense per colonne. Le matrici sparse seguono lo stesso formalismo, quindi una matrice sparsa è una concatenazione di vettori sparsi che rappresentano le colonne della matrice. I non zeri sono raccolti in un vettore di reali in virgola mobile. Un secondo vettore memorizza gli indici di riga e completa le informazioni necessarie. Per accedere alla singola colonna, infine, un terzo vettore indica dove si trova il primo termine di ogni colonna nei due vettori precedentemente definiti. Di conseguenza, una matrice $m \times n$ di reali in doppia precisione, con nt non zeri occupa $12nt + 4n$ bytes, dove si assume di lavorare con interi a 4 bytes. Se i valori sono complessi, serve un secondo vettore di reali per memorizzare la parte immaginaria. Da notare che m, ovvero il numero di righe, è sostanzialmente irrilevante ai fini sia della memorizzazione che del calcolo, e il suo unico scopo consiste nell'effettuare un controllo sul range degli indici.

In Matlab nessuna matrice sparsa viene creata senza un'esplicita istruzione da parte dell'utente. Quindi, operazioni tra matrici dense produrranno sempre matrici dense. Appena si introduce la

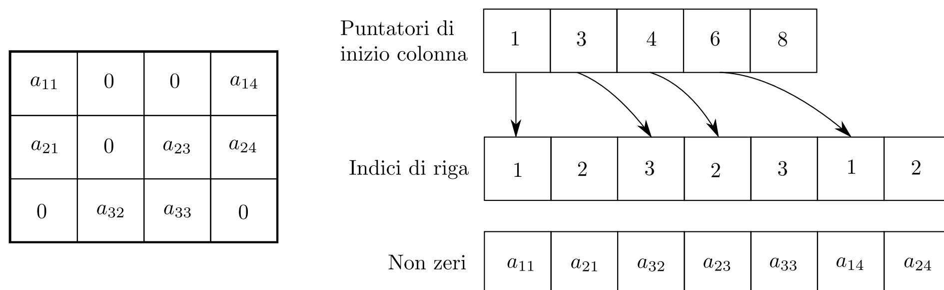

FIGURA 1: *Matrice sparsa memorizzata in formato CSC.*

sparsità, però, questa si mantiene, ovvero operazioni tra matrici sparse producono matrici sparse. Operazioni miste tra matrici sparse e dense producono un risultato sparso nei limiti del possibile. Per esempio, la somma di una matrice densa e una sparsa produce una matrice densa. Invece, l'inserimento di un blocco denso in una matrice sparsa conserva la sparsità. L'operazione più diffusa in assoluto con le matrici sparse è il prodotto matrice vettore: se il vettore `v` è denso, il risultato `A*v` è ancora denso.

3.2 Principali comandi per gestire matrici sparse

Ci sono due comandi MATLAB molto utili quanto si lavora con le matrici sparse: `full` e `sparse`. Per ogni matrice `A`, `full(A)` memorizza la matrice in formato denso. Se `A` è già densa, nulla cambia, ma se `A` è sparsa, i "vuoti" vengono riempiti da zeri e vengono memorizzati. Viceversa, `sparse(A)` rimuove tutti gli elementi nulli e trasforma `A` da formato denso a CSC. Da notare che, in generale, se `A` è grande, non c'è sufficiente spazio in memoria per salvare la versione densa. Inoltre, il comando `sparse` elimina tutti e soli gli elementi che sono esattamente zero. Dato che MATLAB adotta lo standard IEEE-754, il minimo numero reale rappresentabile, prima di incorrere in un errore di underflow, è $2.225073858507201 \cdot 10^{-308}$. Ogni altro numero inferiore, in modulo, a questo, risulta pari a 0.

Il comando `nnz(A)` restituisce il numero di non zeri di una matrice sparsa, mentre il comando `spy(A)` ne rappresenta la struttura, ossia la posizione dei non zeri (sparsity pattern). Talvolta, in fase di costruzione è conveniente allocare uno spazio maggiore rispetto a quello effettivamente utilizzato dalla matrice. Per ottenere tale informazione, si usa il comando `nzmax(A)`. Questi tre comandi possono essere eseguiti anche con matrici memorizzate in formato denso, ma il risultato è banale, in quanto $nt =$ `nzmax(A)` $= m \times n$ e tutte le posizioni hanno dei non zeri.

Infine, il comando `issparse(A)` è molto utile per gestire matrici memorizzate in formati diversi. Questo comando restituisce un valore logico: `true` se la matrice è sparsa, `false` se è densa. Questo è l'unico comando che offre un risultato diverso se applicato alla stessa matrice, a seconda del formato di memorizzazione scelto. Tutti gli altri comandi od operazioni non cambiano risultato al variare della

memorizzazione utilizzata per le matrici.

3.3 Creare matrici sparse

Per creare una matrice sparsa in Matlab si usa l'istruzione:

$$\texttt{S = sparse(row,col,v,m,n,nt)} \tag{17}$$

in cui i vettori `row`, `col` e `v` contengono rispettivamente gli indici di riga, di colonna e il valore per ogni termine non nullo della matrice, che ha `m` righe, `n` colonne e `nt` non zeri. I tre vettori `row`, `col` e `v` devono avere la stessa lunghezza, altrimenti la funzione `sparse` restituisce un messaggio d'errore. In formula, si può scrivere:

$$\texttt{S(row(k),col(k)) = v(k)} \qquad \forall \texttt{k} \in [1, \texttt{length(v)}] \tag{18}$$

Le variabili `m`, `n` e `nt` possono essere omesse. In tal caso, Matlab calcola automaticamente le dimensioni della matrice: il numero di righe è il massimo termine di `row`, il numero di colonne è il massimo termine di `col` e il numero di non zeri è la lunghezza dei tre vettori. Se nei vettori `row` e `col` ci sono coppie di indici uguali, nella matrice `S` si memorizza la somma dei rispettivi valori in `v`. Da notare che non è richiesto alcun ordine nei vettori `row`, `col` e `v`, se non che siano tra loro coerenti.

Per creare una matrice vuota, ovvero per allocare uno spazio in memoria, si usa lo stesso comando:

$$\texttt{S = sparse(m,n,nt)} \tag{19}$$

senza i tre vettori con le informazioni su indici e valore dei termini non nulli. Nuovamente, `nt` può essere omesso. In tal caso, Matlab assume di dover memorizzare un solo termine.

Il comando inverso rispetto a `sparse` è `find`:

$$\texttt{[row,col,v] = find(S)} \tag{20}$$

che, data la matrice sparsa, fornisce i tre vettori `row`, con gli indici di riga, `col`, con gli indici di colonna, e `v`, con i termini non nulli, di lunghezza `nnz(S)`. Di default, `col` è ordinato in ordine crescente e gli altri vettori di conseguenza. Questa convenzione è legata all'ordine con cui Matlab memorizza le informazioni al suo interno, ovvero per colonne. Infine, se si vuole risalire anche al numero di righe e colonne con il comando da usare è:

$$\texttt{[m,n] = size(S)} \tag{21}$$

come per le matrici dense.

Per leggere una matrice sparsa da file, è sufficiente leggere i tre vettori `row`, `col` e `v`, e utilizzare il comando `sparse`. Per creare una matrice identità sparsa, si usa il comando:

$$\texttt{I = speye(m,n)} \tag{22}$$

che, in generale, crea una matrice sparsa $m \times n$, con 1 nelle posizioni in cui l'indice di riga coincide con l'indice di colonna. Se la matrice è quadrata, questa diventa la matrice identità di ordine n. Per creare una matrice diagonale sparsa con termini scelti dall'utente, si usa il comando:

$$\texttt{D = spdiags(v,d,m,n)} \tag{23}$$

in cui `v` è il vettore contenente $\min(m, n)$ valori, `m` e `n` sono, al solito, le dimensioni e `d` indica quale diagonale si va a popolare. Nel caso più semplice, ossia di diagonale principale, `d = 0`. Se `d` è positivo si intende la `(d)`-esima diagonale sopra quella principale, se, invece, è negativo, si intende la `(-d)`-esima diagonale sotto quella principale. `D` è una matrice rettangolare che sulla diagonale principale (se `d = 0`) ha i termini del vettore `v`. In generale, $m = n =$ `length(v)`. Lo stesso comando consente di estrarre la diagonale di una matrice:

$$\texttt{v = spdiags(S)} \tag{24}$$

Il vettore `v` contiene i non zeri della diagonale principale di `S`. Da notare che `v` è un *vettore sparso*. Infatti, il concetto di vettore sparso è lo stesso di una matrice, con la peculiarità di avere una sola colonna.

Per creare una matrice di numeri random, si usa il comando **`sprand`**, che ha la stessa sintassi del comando **`rand`** per matrici dense, con un ulteriore parametro, ovvero il grado di sparsità desiderato, inteso come il rapporto tra il numero di non zeri e il numero di termini che la matrice avrebbe se fosse piena, ovvero $m \times n$:

$$\texttt{R = sprand(m,n,density)} \tag{25}$$

`R` è una matrice random di dimensioni $m \times n$, sparsa, con un numero di non zeri circa pari a `density*m*n`. I coefficienti sono uniformemente distribuiti tra 0 e 1. Se l'utente è interessato a una matrice con entrate distribuite secondo la normale a media 0 e deviazione standard 1, il comando da usare è:

$$\texttt{R = sprandn(m,n,density)} \tag{26}$$

Se la sparsità scelta è eccessivamente bassa, non ci sono garanzie che la matrice prodotta da uno dei due precedenti comandi abbia tutte le righe/colonne non nulle. In generale, la matrice `R`, quindi, è singolare. In alcune applicazioni, può essere utile una matrice a coefficienti random non singolare. In tal caso, serve aggiungere un parametro ai precedenti comandi. Per comodità si riporta l'esempio solo per **`sprand`**, avendo **`sprandn`** la stessa sintassi. Si deve introdurre **`rc`**, ovvero il reciproco del numero di condizionamento spettrale della matrice:

$$\texttt{R = sprand(m,n,density,rc)} \tag{27}$$

Le matrici sparse random così ottenute non sono simmetriche. Per avere una matrice simmetrica, si una un'altra istruzione:

$$\texttt{R = sprandsym(n,density,rc,kind)} \tag{28}$$

Come si può facilmente intuire, per matrici simmetriche basta il numero di righe (e di colonne) `n`. I parametri `density` e `rc` hanno lo stesso significato di prima. Il nuovo parametro `kind`, se definito, permette di ottenere una matrice simmetrica e definita positiva (SPD). Il valore di `kind` può essere 1 o 2 e definisce l'algoritmo con cui MATLAB crea la matrice random SPD. In generale, con `kind = 1`, la matrice risultante ha un densità leggermente inferiore.

4 Calcolo e applicazione del precondizionatore

Una delle chiavi del successo del GCM nella soluzione efficiente di sistemi lineari sparsi, simmetrici e definiti positivi sta nella possibilità di ottenere formidabili accelerazioni della convergenza mediante l'uso di opportune matrici di precondizionamento. La scelta di K^{-1} deve soddisfare alle seguenti caratteristiche:

- deve essere tale che il prodotto AK^{-1} abbia lo spettro, cioè l'insieme degli autovalori, raggruppato attorno all'unità, o comunque un numero di condizionamento spettrale molto inferiore a quello di A;
- il calcolo deve essere semplice e poco costoso per non appesantire lo schema;
- l'occupazione di memoria deve essere paragonabile a quella della matrice del sistema allo scopo di non vanificare lo sforzo computazionale effettuato per la memorizzazione compatta.

Spesso le suddette caratteristiche confliggono fra loro e necessariamente la scelta di K^{-1} diventa il risultato di un compromesso. Per paradosso, la matrice che soddisfa completamente la prima richiesta è ovviamente A^{-1}, il cui costo e occupazione di memoria (si ricordi che l'inversa di una matrice sparsa è generalmente una matrice piena) tuttavia rendono del tutto incalcolabile.

Una discreta accelerazione del metodo del GC è ottenuta scegliendo:

$$K^{-1} = F^{-1} \tag{29}$$

dove F è la matrice diagonale contenente gli elementi diagonali di A. Tale matrice è indicata anche come *precondizionatore di Jacobi*. In questo caso, il calcolo della matrice di precondizionamento e la sua applicazione nello schema del GCM sono banali e senza un apprezzabile costo computazionale aggiuntivo rispetto al GC. Il raggruppamento degli autovalori attorno all'unità di AF^{-1}, tuttavia, può essere limitato, specialmente se A presenta coefficienti extra-diagonali grandi rispetto agli elementi diagonali.

Risultati assai più significativi sono invece ottenuti assumendo:

$$K^{-1} = \left(\tilde{L}\tilde{L}^T\right)^{-1} \tag{30}$$

dove $\tilde{L}$, matrice triangolare bassa, è calcolata mediante la fattorizzazione incompleta di A, cioè la decomposta di Cholesky a cui viene assegnato il medesimo schema di sparsità di A. L'uso di questa matrice di precondizionamento, proposto alla fine degli anni '70 prima da Meijerink e Van der Vorst e poi da Kershaw, si rivela generalmente un buon compromesso fra le contrapposte esigenze precedentemente discusse. Il calcolo di K^{-1} secondo la (30) e la sua applicazione nell'algoritmo del GCM verranno esaminati nei due paragrafi seguenti.

4.1 Fattorizzazione incompleta di Cholesky

Il calcolo del fattore incompleto di Cholesky si basa sulla fattorizzazione triangolare di matrici simmetriche che viene di seguito brevemente richiamata.

Supponiamo che A sia una matrice 3×3 piena di cui svolgiamo per esteso la fattorizzazione:

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{31}$$

Si noti che nella (31) si è sfruttata la simmetria di A. Sviluppiamo il prodotto (31) procedendo secondo la successione delle colonne di A. Per la prima colonna vale:

$$\begin{aligned} a_{11} = l_{11}^2 \quad &\Rightarrow \quad l_{11} = \sqrt{a_{11}} \\ a_{21} = l_{11} l_{21} \quad &\Rightarrow \quad l_{21} = \frac{a_{21}}{l_{11}} \\ a_{31} = l_{11} l_{31} \quad &\Rightarrow \quad l_{31} = \frac{a_{31}}{l_{11}} \end{aligned} \tag{32}$$

Si può osservare che i coefficienti posti nella parte triangolare alta di A vengono di volta in volta già utilizzati nel calcolo. Si procede, pertanto, con le colonne 2 e 3 considerando i soli termini relativi alla triangolare bassa:

$$\begin{aligned} a_{22} = l_{21}^2 + l_{22}^2 \quad &\Rightarrow \quad l_{22} = \sqrt{a_{22} - l_{21}^2} \\ a_{32} = l_{21} l_{31} + l_{22} l_{32} \quad &\Rightarrow \quad l_{32} = \frac{1}{l_{22}} \left(a_{23} - l_{21} l_{31} \right) \\ a_{33} = l_{31}^2 + l_{32}^2 + l_{33}^2 \quad &\Rightarrow \quad l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} \end{aligned} \tag{33}$$

Dalle (32) e (33) si può facilmente generalizzare l'algoritmo di calcolo del fattore completo di Cholesky per una matrice di ordine n qualsiasi:

001 $\quad l_{11} := \sqrt{a_{11}}$

002 $\quad$ **Per** $i = 2, n$

003 $\quad\quad l_{i1} := a_{i1} / l_{11}$

004 **Fine Per**

005 **Per** $j = 2, n$

006 $u_{jj} := \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$

007 **Per** $i = j + 1, n$

008 $l_{ij} := \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} \, l_{jk} \right) / l_{jj}$

009 **Fine Per**

010 **Fine Per**

Il passaggio concettuale dal fattore completo di Cholesky a quello incompleto risulta a questo punto banale, in quanto all'algoritmo precedente è sufficiente aggiungere l'assegnazione:

$$a_{ij} = 0 \Rightarrow l_{ij} := 0 \tag{34}$$

Poiché A è memorizzata in modo compatto, e così anche $\tilde{L}$, si deduce immediatamente che, in virtù dell'assegnazione (34), i vettori `row` e `col` descrivono, per quanto riguarda la parte bassa di A, anche la topologia di $\tilde{L}$.

Si deve infine ricordare che l'uso generalizzato delle (32) e (33) comporta l'estrazione di radici quadrate il cui argomento, nel caso di una fattorizzazione incompleta, non è più garantito essere positivo. In questo caso l'elemento diagonale in questione verrà posto pari a un numero positivo arbitrario (ad esempio, l'ultimo coefficiente diagonale di $\tilde{L}$ non nullo). Si dimostra comunque teoricamente che se la matrice A è di tipo M, cioè diagonalmente dominante con elementi extra-diagonali di segno opposto rispetto a quelli diagonali, come ad esempio si verifica nella discretizzazione agli elementi finiti dell'equazione ellittica di Laplace, tutte le radici da calcolare nel fattore incompleto esistono nel campo reale.

4.2 La fattorizzazione incompleta di Cholesky in Matlab

L'implementazione efficiente del calcolo della fattorizzazione incompleta di Cholesky in algebra sparsa non è banale. Poiché questo tipo di precondizionatore è oramai diventato uno standard in molti problemi, esiste una function intrinseca di Matlab che ne permette il calcolo e la memorizzazione in formato sparso. L'istruzione di Matlab per il calcolo di $\tilde{L}$ è la seguente:

`L=ichol(A,opts)` (35)

dove `A` è la matrice memorizzata in formato sparso e `opts` è una variabile record contenente al piú 5 campi in cui è possibile specificare delle opzioni aggiuntive nel calcolo di $\tilde{L}$. La variabile `opts` contiene i seguenti campi:

- `opts.type`: la decomposta incompleta viene calcolata solamente sul pattern di non-zeri di A (`'nofill'`) oppure su un pattern più ampio (`'ict'`). Il valore di default è `'nofill'`;

- `opts.droptol`: è la tolleranza rispetto alla norma-1 della riga corrente della matrice A al di sotto della quale un coefficiente di $\tilde{L}$ viene trascurato. Questo campo va valorizzato solamente se `opts.type='ict'` e il default è 0;

- `opts.michol`: se `'on'` introduce una compensazione sui valori diagonali di $\tilde{L}$ per tener conto dei coefficienti trascurati. Il valore di default è `'off'`;

- `opts.diagcomp`: calcola la decomposta incompleta della matrice stabilizzata $A' = A + \alpha \text{diag}(A)$, dove α è lo scalare assegnato alla variabile in questione. Il valore di default per α è 0;

- `opts.shape`: calcola il fattore incompleto triangolare basso $\tilde{L}$ (`'lower'`) oppure quello traingolare alto $\tilde{L}^T$ (`'upper'`). Il valore di default è `'lower'`.

L'uso di `opts` è opzionale. In caso non venga assegnato, sono utilizzati i valori di default.

La funzione intrinseca `ichol` non calcola esplicitamente la matrice di precondizionamento K^{-1}, ma solamente il fattore incompleto $\tilde{L}$. È quindi necessario sviluppare un algoritmo che permetta di calcolare il prodotto $K^{-1}\boldsymbol{r}$ senza generare esplicitamente K^{-1}.

Sia $\boldsymbol{v}$ il vettore in $\mathbb{R}^n$ risultato del prodotto $K^{-1}\boldsymbol{r}$. Per definizione di K^{-1} si ha:

$$\boldsymbol{v} = \left(\tilde{L}\tilde{L}^T\right)^{-1} \boldsymbol{r} \tag{36}$$

cioè, premoltiplicando per K ambo i membri:

$$\left(\tilde{L}\tilde{L}^T\right) \boldsymbol{v} = \boldsymbol{r} \tag{37}$$

Il calcolo di $\boldsymbol{v}$ viene quindi ricondotto alla soluzione di un sistema lineare la cui matrice è K. Poiché K è fattorizzabile nel prodotto di due matrici triangolari, il sistema (37) è facilmente risolubile tramite sostituzioni in avanti e all'indietro. Posto:

$$\tilde{L}^T \boldsymbol{v} = \boldsymbol{z} \tag{38}$$

la (37) diventa:

$$\tilde{L}\boldsymbol{z} = \boldsymbol{r} \tag{39}$$

con $\boldsymbol{z}$ ricavabile tramite sostituzioni in avanti. Iniziando dalla prima componente, ricavata in modo immediato come:

$$z_1 = \frac{r_1}{l_{11}} \tag{40}$$

si ottiene con semplici calcoli la formula ricorrente:

$$z_i = \frac{1}{l_{ii}} \left(r_i - \sum_{j=1}^{i-1} l_{ij} z_j \right) \qquad i = 2, \ldots, n \tag{41}$$

Ottenuto il vettore $\boldsymbol{z}$ si può infine calcolare $\boldsymbol{v}$ risolvendo il sistema (38) tramite sostituzioni all'indietro. La formula ricorrente si ricava in modo del tutto analogo a quanto fatto nelle equazioni (40) e (41) partendo in questo caso dalla componente n-esima:

$$v_n = \frac{z_n}{l_{nn}} \tag{42}$$

$$v_i = \frac{1}{l_{ii}} \left(z_i - \sum_{j=i+1}^{n} l_{ji} v_j \right) \qquad i = n-1, \dots, 1 \tag{43}$$

La maniera più semplice ed efficiente per risolvere in Matlab i sistemi (38) e (39) con $\tilde{L}$ memorizzata in formato sparso consiste nell'usare il comando \ ("backslash"). Tale istruzione analizza le caratteristiche della matrice e vi applica il miglior metodo diretto disponibile. Nel nostro caso, Matlab riconoscerà che `L` calcolata con la (35) è una matrice sparsa triangolare bassa e utilizzerà sostituzioni in avanti e all'indietro. La sequenza di istruzioni in Matlab per implementare l'applicazione della decomposta incompleta di Cholesky sarà dunque:

```
z=L\ r
v=L'\ z
```
(44)