Capitolo 3

Sistemi lineari

3.1 Considerazioni generali

Consideriamo il sistema lineare

$$Ax = b, \quad A \in \mathbb{R}^{n\times n},\ b \in \mathbb{R}^{n\times 1}\ .$$

In GNU Octave il comando

```
x = A\b;
```

calcola la soluzione del sistema lineare, usando un opportuno metodo diretto. Nel caso generale, viene eseguita una fattorizzazione LU con pivoting parziale seguita da una coppia di sostituzioni in avanti e all'indietro. Sarebbe ovviamente possibile usare il comando `inv` per il calcolo dell'inversa A^{-1} e calcolare x tramite il comando `x = inv(A)*b`. Tuttavia, tale metodo risulta essere svantaggioso, sia dal punto di vista computazionale che dal punto di vista dell'accuratezza. Infatti, per il calcolo dell'inversa, occorre risolvere gli n sistemi lineari

$$AX = I\ .$$

È necessaria pertanto una fattorizzazione LU e n coppie di sostituzioni in avanti e all'indietro. Dal punto di vista dell'accuratezza, si consideri il sistema lineare formato dall'unica equazione

$$49x = 49\ .$$

L'applicazione del metodo di eliminazione gaussiana porta al comando `x = 49/49`, mentre l'inversione di matrice porta al comando `x = (1/49)*49`. È facile verificare che solo nel primo caso si ottiene esattamente $x = 1$. La regola fondamentale è: *mai calcolare l'inversa di una matrice*, a meno che

non sia esplicitamente richiesta. Dovendo valutare un'espressione del tipo $A^{-1}B$, si possono considerare i sistemi lineari $AX = B$ da risolvere con il comando `X = A\B`. Analogamente, un'espressione del tipo BA^{-1} può essere calcolata con con il comando `B/A`.

3.2 Operazioni vettoriali in GNU Octave

Riportiamo in questo paragrafo alcuni comandi, sotto forma di esempi, utili per la manipolazione di matrici in GNU Octave.

3.2.1 Operazioni su singole righe o colonne

Data una matrice A, le istruzioni

```
for i = 1:n
  A(i,j) = A(i,j)+1;
end
```

possono essere sostituite da

```
A(1:n,j) = A(1:n,j)+1;
```

È possibile scambiare l'ordine di righe o colonne. Per esempio, l'istruzione

```
A = B([1 3 2],:);
```

crea una matrice A che ha per prima riga la prima riga di B, per seconda la terza di B e per terza la seconda di B. Analogamente, l'istruzione

```
A = B(:,[1:3 5:6]);
```

crea una matrice A che ha per colonne le prime tre colonne di B e poi la quinta e la sesta di B.

È possibile concatenare matrici. Per esempio, l'istruzione

```
U = [A b];
```

crea una matrice U formata da A e da un'ulteriore colonna b (ovviamente le dimensioni di A e b devono essere compatibili).

È possibile assegnare lo stesso valore ad una sottomatrice. Per esempio, l'istruzione

```
A(1:3,5:7) = 0;
```

pone a zero la sottomatrice formata dalle righe dalla prima alla terza e le colonne dalla quinta alla settima.

Un altro comando utile per la manipolazione di matrici è `max`. Infatti, nella forma

```
[m i] = max(A(2:7,j));
```

restituisce l'elemento massimo m nella colonna j-esima di A (dalla seconda alla settima riga) e la posizione di tale elemento nel vettore $[a_{2j}, a_{3j}, \ldots, a_{7,j}]^{\mathrm{t}}$.

Tutte le istruzioni vettoriali che sostituiscono cicli `for` sono da preferirsi dal punto di vista dell'efficienza computazionale in GNU Octave.

3.3 Sostituzioni all'indietro

L'algoritmo delle sostituzioni all'indietro per la soluzione di un sistema lineare $Ux = b$, con U matrice triangolare superiore, può essere scritto

```
n = length(b);
x = b;
x(n) = x(n)/U(n,n);
for i = n-1:-1:1
  for j = i+1:n
    x(i) = x(i)-U(i,j)*x(j);
  end
  x(i) = x(i)/U(i,i);
end
```

Per quanto visto nel paragrafo precedente, le istruzioni

```
for j = i+1:n
  x(i) = x(i)-U(i,j)*x(j);
end
x(i) = x(i)/U(i,i);
```

possono essere sostituite da

```
x(i) = (x(i)-U(i,i+1:n)*x(i+1:n))/U(i,i);
```

ove, in questo caso, l'operatore `*` è il prodotto scalare tra vettori. Generalizzando al caso di più termini noti $b_1, b_2, \ldots, b_m$, le istruzioni

```
n = length(B);
X = B;
X(n,:) = X(n,:)./U(n,n);
for i = n-1:-1:1
  X(i,:) = (X(i,:)-U(i,i+1:n)*X(i+1:n,:))/U(i,i);
end
```

risolvono

$$UX = B$$

cioè

$$Ux_i = b_i, \quad i = 1, 2, \ldots, m \ .$$

3.4 Memorizzazione di matrici sparse

Sia A una matrice sparsa di ordine n e con m elementi diversi da zero. Esistono molti formati di memorizzazione di matrici sparse. Quello usato da GNU Octave è il Compressed Column Storage (CCS). Consiste di tre array: un primo, `data`, di lunghezza m contenente gli elementi diversi da zero della matrice, ordinati prima per colonna e poi per riga; un secondo, `ridx`, di lunghezza m contenente gli indici di riga degli elementi di `data`; ed un terzo, `cidx`, di lunghezza $n+1$, il cui primo elemento è 0 e l'elemento $i+1$-esimo è il numero totale di elementi diversi da zero nelle prime i colonne della matrice. Per esempio, alla matrice

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

corrispondono i vettori

$$\begin{aligned} \texttt{data} &= [1, 4, 2, 3, 5, 6, 7] \\ \texttt{ridx} &= [1, 3, 2, 2, 3, 3, 4] \\ \texttt{cidx} &= [0, 2, 3, 5, 7] \end{aligned}$$

Dato un vettore x, il prodotto matrice-vettore $y = Ax$ è implementato dall'algoritmo in Tabella 3.1.

In GNU Octave, il formato CCS e l'implementazione del prodotto matrice-vettore sono automaticamente usati dalla function `sparse` e dall'operatore `*`, rispettivamente. Un utile comando per la creazione di matrici sparse è `spdiags`.

```
n = length(x);
y = zeros(size(x));
for j = 1:n
  for i = cidx(j)+1:cidx(j+1)
    y(ridx(i)) = y(ridx(i))+data(i)*x(j);
  end
end
```

Tabella 3.1: Prodotto matrice-vettore in formato CCS.

3.5 Metodo di Newton per sistemi di equazioni non lineari

Consideriamo il sistema di equazioni non lineari

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

che può essere riscritto, in forma compatta,

$$f(x) = 0 \, .$$

Dato $x^{(0)}$, il metodo di Newton per calcolare $x^{(k+1)}$ è

$$\begin{aligned} J^{(k)} \delta x^{(k)} &= -f(x^{(k)}) \\ x^{(k+1)} &= x^{(k)} + \delta x^{(k)} \end{aligned} \tag{3.1}$$

ove $J^{(k)}$ è la matrice Jacobiana, definita da

$$J_{ij}^{(k)} = \frac{\partial f_i(x^{(k)})}{\partial x_j^{(k)}} \, . \tag{3.2}$$

Il criterio d'arresto solitamente usato è

$$\left\| \delta x^{(k)} \right\| \leq \text{tol} \, .$$

3.5.1 Metodo di Newton modificato

Il metodo di Newton (3.1) richiede il calcolo della matrice Jacobiana e la sua "inversione" ad ogni passo k. Questo potrebbe essere troppo oneroso. Una strategia per ridurre il costo computazionale è usare sempre la stessa matrice Jacobiana $J^{(0)}$, oppure aggiornarla solo dopo un certo numero di iterazioni. In tal modo, per esempio, è possibile usare la stessa fattorizzazione $L^{(k)}U^{(k)}$ per più iterazioni successive.

3.6 Esercizi

1. Si implementi una function `[U b1] = meg(A,b)` per il metodo di eliminazione gaussiana con pivoting per righe.

2. Si risolvano, mediante il metodo di eliminazione gaussiana e l'algoritmo delle sostituzioni all'indietro, i sistemi lineari

$$A_i x_i = b_i, \quad A_i = (A_1)^i, \quad i = 1, 2, 3, 4$$

   con

$$A_1 = \begin{pmatrix} 15 & 6 & 8 & 11 \\ 6 & 6 & 5 & 3 \\ 8 & 5 & 7 & 6 \\ 11 & 3 & 6 & 9 \end{pmatrix}$$

   e b_i scelto in modo che la soluzione esatta sia $x_i = [1, 1, 1, 1]^{\mathrm{t}}$. Per ogni sistema lineare si calcoli l'errore in norma-2 e il numero di condizionamento della matrice (con il comando `cond`). Si produca infine un grafico logaritmico-logaritmico che metta in evidenza la dipendenza lineare dell'errore dal numero di condizionamento.

3. Si implementi una function `X = fs(L,B)` che implementa l'algoritmo delle sostituzioni in avanti generalizzato al caso di più termini noti.

4. Sia A una matrice di ordine 5 generata in maniera casuale. Se ne calcoli l'inversa X, per mezzo del comando `lu` e degli algoritmi delle sostituzioni in avanti e all'indietro generalizzati al caso di più termini

noti, secondo lo schema

$$
\begin{aligned}
AX &= I \\
PAX &= P \\
LUX &= P \\
&\begin{cases} LY = P \\ UX = Y \end{cases}
\end{aligned}
$$

5. Implemetare le functions `[data ridx cidx] = full2ccs(A)` e `[A] = ccs2full(data,ridx,cidx)`.

6. Data una matrice A memorizzata in formato CCS e un vettore x, implementare la seguente formula: $y = A^{\mathrm{T}}x$.

7. Si implementi il metodo di Jacobi con una `function [x iter err] = jacobi(A,b,tol,maxit)` che costruisce la matrice di iterazione per mezzo del comando `spdiags`. Lo si testi per la soluzione del sistema lineare $Ax = b$ con `A = toeplitz([4 1 0 0 0 0])` e b scelto in modo che la soluzione esatta sia $x = [2, 2, 2, 2, 2, 2]^{\mathrm{t}}$.

8. Si risolva il sistema non lineare

$$
\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 = 1 \\ f_2(x_1, x_2) = \sin(\pi x_1/2) + x_2^3 \end{cases}
$$

   con il metodo di Newton (3.1). Si usi una tolleranza pari a 10^{-6}, un numero massimo di iterazioni pari a 150 e un vettore iniziale $x^{(0)} = [1, 1]^{\mathrm{T}}$.

9. Si risolva lo stesso sistema non lineare usando sempre la matrice Jacobiana relativa al primo passo e aggiornando la matrice Jacobiana ogni r iterazioni, ove r è il più piccolo numero che permette di ottenere la soluzione con la tolleranza richiesta calcolando solo due volte la matrice Jacobiana. La risoluzione dei sistemi lineari deve avvenire con il calcolo della fattorizzazione LU solo quando necessaria.