Laboratorio di Calcolo Numerico: Algebra lineare numerica

Giacomo Elefante

Laboratorio di calcolo numerico
06/11/24

Dato un vettore $x \in \mathbb{R}^n$, le norme vettoriali più usate sono

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|, \qquad \|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}, \qquad \|x\|_\infty = \max_{i=1,\dots,n} |x_i|$$

Mentre nel caso di una matrice $A \in \mathbb{R}^{n \times n}$

$$\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^{n} |a_{i,j}|, \qquad \|A\|_2 = \sqrt{\max_{i=1,\dots,n} |\lambda_i(A^t A)|},$$

$$\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^{n} |a_{i,j}|$$

dove $\lambda_i(B)$ sono gli autovalori della matrice B

Esercizio

In Matlab tali norme sono implementabili facilmente con il comando `norm`, ad esempio `norm(x,1)` restituirà la norma 1 del vettore (o matrice) `x`, per la norma infinito basta sostituire 1 con `inf`/`Inf`/`'Inf'`/`'inf'`.

Esercizio

Si crei una function `norme_varie` che prende in input un vettore o una matrice `x` e un parametro `p` che può prendere solo (altrimenti viene segnalato un errore) i valori 1, 2, 'Inf' (o 'inf'); La function inoltre resituisce come output il valore `s` corrispondente alla norma di `x`
Dato quindi `x`, la function dovrà distinguere il caso esso sia un vettore o una matrice.
Si costruisca la function **senza** usare il comando `norm` di Matlab ma attraverso le definizioni.

Data una matrice $A \in \mathbb{R}^{n \times n}$ e un vettore $b \in \mathbb{R}^n$, in Matlab è possibile ricavare il vettore $x \in \mathbb{R}^n$ soluzione del sistema lineare

$$Ax = b,$$

attraverso il comando backslash \, ovvero indicando
`x = A\b;`

Il condizionamento di una matrice $\kappa_p(A)$ associato ad una certa norma $\|\cdot\|_p$ e definito come

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p$$

e, similmente al comando `norm`, calcolabile in Matlab con il comando `cond`.

Data una matrice quadrata $A \in \mathbb{R}^{n \times n}$, per decomposizione LU si riferisce alla fattorizzazione della matrice A in due matrici $L, U \in \mathbb{R}^{n \times n}$, la prima triangolare inferiore, la seconda triangolare superiore, tali che

$$A = LU.$$

Tale fattorizzazione però non è sempre possibile da fare, come ad esempio nel caso in cui $a_{11} = 0$ e A è una matrice non singolare. Infatti, il primo elemento di A uguale a 0 implica che o $l_{11} = 0$ o $u_{11} = 0$, e, essendo le matrici triangolari, si ha che o L o U risulta essere una matrice singolare, ma tale fatto è impossibile.
Per cui, è preferibile la fattorizzazione LU con pivoting parziale

$$PA = LU.$$

Tale decomposizione può essere effettuata su Matlab attraverso il comando `lu`.
Nel caso si chiedano solo due output, essi saranno due matrici, una la matrice U e l'altra la matrice L permutata, ovvero $P^t L$

Matlab non prevede che possa essere fatta la fattorizzazione LU senza pivoting perchè l'algoritmo senza pivoting parziale è potenzialmente instabile. Infatti, quando A è mal condizionata le matrici L ed U calcolate in aritmetica finita potrebbero essere "sensibilmente" non triangolari.

La decomposizione LU può essere usata per trovare la soluzione di un sistema lineare $Ax = b$, infatti, considerando la decomposizione, è possibile andare a risolvere direttamente due sistemi triangolari.

$$\begin{cases} Ly = Pb \\ Ux = y \end{cases}$$

Esercizio

Esercizio

Si crei la function `lu_solver` che prende in input una matrice A e un vettore colonna b e ricava la soluzione del sistema, facendo la decomposizione $PA = LU$ della matrice.
Si testi in seguito la funzione sulla matrice

$$\texttt{A} = \texttt{magic}(9), \qquad b = \left(910, 1034, 1113, 1264, 1172, 981, 1060, 941, 750,\right)^t$$

Esercizio

Per $n = 2, 4, \ldots, 20$ si creino n punti equispaziati `z` in $[-1, 1]$, si calcoli la matrice di Vandermonde nella base canonica attraverso il comando `V = vander(z)`. Si ponga in seguito `A = V + ϵI` con ϵ=`1e-15` e si risolva il sistema lineare `Ax=b` con `b=A*(1,1,...,1)'` utilizzando sia la decomposizione LU di Matlab attraverso il comando `lu` che la decomposizone senza pivoting della funzione `LUnoPiv`. Per ogni n si calcoli l'errore delle due soluzioni memorizzando i valori in un vettore (uno per tipo di soluzione) e si calcoli anche l'errore `norm(U-triu(U))` memorizzandolo in un vettore. Si facciano due figure in scala semilogaritmica dei vari errori (uno sulla soluzione e uno sulla matrice U).

Metodi iterativi

La fattorizzazione LU con pivoting, il cui costo computazionale è in generale di $\mathcal{O}(n^3/3)$ operazioni diventa proibitivo se n è particolarmente elevato.
L'idea dei metodi iterativi è quello di ottenere una successione di vettori $x^{(k)} \to x^*$ così da avere $x^{(k)} \approx x^*$ per $k << n$.
I metodi saranno del tipo

$$x^{(k+1)} = Ex^{(k)} + q,$$

partendo da un vettore $x^{(0)}$.

Saranno utili i comandi matlab `diag` (o la composizione `diag(diag(·))`), `triu` e `tril`

Decomposizione matriciale

Sia $A \in \mathbb{R}^{n \times n}$, non singolare allora si può decomporre come

$$A = M - N,$$

con M non singolare e facilmente invertibile. Pertanto

$$Ax = b \tag{1}$$
$$Mx - Nx = b \tag{2}$$
$$Mx = Nx + b \tag{3}$$
$$x = M^{-1}Nx + M^{-1}b \tag{4}$$

da cui, ponendo $E = M^{-1}N$ e $q = M^{-1}b$, la soluzione è ricondotta al sistema $x = Px + q$.

Metodo di Richardson

Nel metodo di Richardson si ha

$$M = I, \quad N = I - A,$$

dove I è la matrice identità. Pertanto l'iterazione di Richardson diventa

$$x^{(k+1)} = (I - A)x^{(k)} + b.$$

```
function [x,errs,iter]=richardson(A,x,b,max_iter,tol)

N = eye(size(A)) - A;

for iter=1:max_iter
    x_old = x;
    x     = N*x_old+b;                    % nuova iterazione
    errs(iter) = norm(x-x_old)/norm(x);   % stima errore relativo
    if (errs(iter) <= tol)
        return
    end
end
end
```

Figure: Il codice Matlab

Metodo di Jacobi

Nel metodo di Jacobi si ha

$$M = D, \quad N = F + G,$$

dove D è la matrice diagonale con la diagonale di A, F e G sono l'opposto delle matrici triangolare inferiore e superiore con diagonale nulla. Ovvero $A = D - F - G$. Pertanto l'iterazione di Jacobi diventa

$$x^{(k+1)} = D^{(-1)}(F + G)x^{(k)} + D^{-1}b.$$

```
function [x,errs,iter]=jacobi(A,x,b,max_iter,tol)

M = diag(diag(A));

if det(M) == 0
    error('Metodo di Jacobi non applicabile');
end

N = M - A;

for iter=1:max_iter
    x_old = x;
    x     = M\(N*x_old+b);                % nuova iterazione
    errs(iter) = norm(x-x_old)/norm(x);   % stima errore relativo
    if (errs(iter) <= tol)
        return
    end
end
end
```

Figure: Il codice Matlab

Metodo di Gauss-Seidel

Nel metodo di Gauss-Seidel si ha

$$M = D - F, \quad N = G,$$

dove D è la matrice diagonale con la diagonale di A, F e G sono l'opposto delle matrici triangolare inferiore e superiore con diagonale nulla. Ovvero $A = D - F - G$. Pertanto l'iterazione di Gauss-Seidel diventa

$$x^{(k+1)} = (D - F)^{(-1)} G\, x^{(k)} + (D - F)^{-1} b.$$

Esercizio

Considerando le funzioni date di Richardson e di Jacobi, si costruisca la funzione per il metodo di Gauss-Seidel.

Esercizio

Si assegni ad $\mathtt{A} \in \mathbb{R}^{n \times n}$ la matrice definita come

$$
\begin{aligned}
a_{ij} &= 1, \qquad \text{se } i = j \\
a_{ij} &= -1/3 \qquad \text{se } j = i + 1 \\
a_{ij} &= 1/3 \quad \text{se } j = i - 1,
\end{aligned}
$$

inoltre si consideri come `b` il vettore con le stesse righe di `A` e i cui elementi sono i numeri da 1 a n.
Si costruiscano entrambi gli elementi per $n = 50$. Si confronti quindi la soluzione ottenuta con i metodi di Richardson, Jacobi, Gauss-Seidel, utilizzando come vettore iniziale un vettore `x0` con le stesse righe di `A` e contenente tutti zeri, una tolleranza `tol=1e-8` e 200 iterate massime.
Si stampi quindi a schermo il numero di iterate impiegate dai vari metodi. Inoltre si calcolino i valori massimi (per la convergenza) `|1+eig(A)|`, `|eig(J)|` e `|eig(GS)|`, dove `J` e `GS` sono le matrici di iterazione rispettivamente del metodo di Jacobi e del metodo di Gauss-Seidel.