Laboratorio di Calcolo Numerico: Algebra lineare numerica

Giacomo Elefante

Laboratorio di calcolo numerico
09/11/23

Scopo di oggi

Analizzare algoritmi per l'algebra lineare numerica:

- Norme vettoriali e matriciali
- Risoluzione di sistemi lineari
- Condizionamento
- Decomposizione LU
- Metodi iterativi

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'; 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`.

Esercizio

Esercizio

Ricordando la stima, vista a lezione, per l'errore relativo della soluzione di un sistema lineare perturbato $\widetilde{A}\widetilde{x} = \widetilde{b}$

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right),$$

valida se $\kappa(A)\frac{\|\delta A\|}{\|A\|} < 1$ e dove $\widetilde{A} = A + \delta A$, $\widetilde{x} = x + \delta x$, $\widetilde{b} = b + \delta b$, si crei una funzione `sistema_perturbato` senza output e che prenda come input i valori `A`, `b`, `Aap`, `bap` ovvero le matrici e vettori $A, b, \widetilde{A}, \widetilde{b}$.
Internamente alla funzione si calcoli le soluzioni `x`, `xap` relative, rispettivamente, al sistema lineare e al sistema lineare perturbato attraverso il comando backslash.
In seguito, dopo aver eseguito un controllo per $\kappa(A)\frac{\|\delta A\|}{\|A\|} < 1$ (e definito un comando di errore nel caso contrario), si controlli la stima calcolando il valore a sinistra e a destra della disuguaglianza, stampandoli a schermo in formato esponenziale con 1 cifra prima della virgola e due dopo.

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 $P^t L$

Esercizio

La decomposizione 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

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

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

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

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

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

Saranno utili i comandi matlab `diag` (o meglio 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 $P = M^{-1}N$ e $q = M^{-1}b$, la soluzione è ricondotta al sistema $x = Px + q$.

Metodo di Jacobi

Nel metodo di Jacobi si ha

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

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

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

```
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

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

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

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

```
function [x,errs,iter]=gauss_seidel(A,x0,b,max_iter,tol)


M = diag(diag(A))+tril(A,-1);

if det(M) == 0
    error('Metodo di Gauss-Seidel non applicabile');
end

N = -triu(A,1);

for iter=1:max_iter
    if iter ==1
        x_old = x0;
    else
    x_old = x(:,iter-1);
    end
    x(:,iter)     = 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 SOR

Nel metodo SOR (successive over relaxation) si ha

$$M = D - \omega E, \quad N = (1-\omega)D + \omega F;,$$

dove D è la matrice diagonale con la diagonale di A, E e F sono l'opposto delle matrici triangolare inferiore e superiore con diagonale nulla, ovvero $A = D - E - F$, e $0 < \omega < 2$. Pertanto l'iterazione SOR diventa

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

Si noti che con $\omega = 1$, si ottiene l'iterazione di Gauss-Seidel.

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

D = diag(diag(A));

if det(D) == 0
    error('Metodo SOR non applicabile');
end

E = -tril( A, -1 );
F = -triu( A,  1 );
M =  D - w*E;
N = (1-w)*D+w*F;

for iter=1:max_iter
    x_old = x;
    x     = M\(N*x_old+w*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

SOR con omega ottimale

La convergenza del metodo SOR dipende dalla matrice di iterazione

$$H(\omega) = (D - \omega E)^{(-1)}((1-\omega)D + \omega F)$$

e in particolare è possibile ottenere un valore ω^* ottimale per la convergenza.
È possibile infatti dimostrare che se la matrice di iterazione del metodo di Jacobi, $P = I - D^{(-1)}A$, ha solo autovalori reali e con raggio spettrale $\rho(P) < 1$, allora esiste un valore ottimale ω^* per cui si ha che

$$\rho(H(\omega^*)) = \min_{0<\omega<2} \rho(H(\omega)).$$

Esercizio

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

$$\begin{aligned} a_{ij} &= i + 2, && \text{se } i = j \\ a_{ij} &= (-1)^i && \text{se } j = i + 1 \\ a_{ij} &= (-1)^{i+1} && \text{se } j = i - 1, \end{aligned}$$

inoltre si consideri come `b` il vettore con le stesse righe di `A` e i cui elementi sono tutti 1. Si costruiscano entrambi gli elementi per $n = 50$. Si confronti quindi la soluzione ottenuta con i metodi di Jacobi, Gauss-Seidel e SOR con parametro ottimale $\omega^* = 1.045454545454545$, 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.

Esercizio per casa

Esercizio

Si crei una function `SOROmegaOttimale` che prende in input la matrice A del sistema lineare e, dopo aver discretizzato l'intervallo $\mathcal{I} = (0, 2)$, calcola il raggio spettrale della matrice $H(\omega)$ per $\omega \in \mathcal{I}$ e in seguito sceglie il valore ottimale ω^*, che viene immagazzinato nella unica variabile di output `omega_ott`, che corrisponde al valore che minimizza questi raggi spettrali.