Introduzione all'algebra lineare numerica

Laboratorio di Calcolo Numerico

Federico Piazzon

31 Maggio 2022

Outline

1. Soluzioni sistemi triangolari
2. Soluzione sistemi quadrati (metodi diretti)
3. Condizionamento di sistemi lineari

Soluzioni sistemi triangolari

Sistemi triangolari superiori

Sostituzione all'indietro

Se U è matrice $n \times n$ triangolare superiore invertibile, allora la soluzione di $Ux = b$ può essere calcolata come

$$\begin{cases} x_n = b_n / U_{n,n} \\ x_{n-k} = \frac{b_{n-k} - \sum_{j=1}^{k} U_{n-k,n-k+j} x_{n-k+j}}{U_{n-k,n-k}} \quad , k = 1, 2, \ldots, n-1 \end{cases}$$

NB: U triangolare quadrata è invertibile se e solo se $U_{i,i} \neq 0$ per ogni $i = 1, 2, \ldots, n$

Sistemi triangolari inferiori

Sostituzione in avanti

Se U è matrice $n \times n$ triangolare inferiore invertibile, allora la soluzione di $Ux = b$ può essere calcolata come

$$\begin{cases} x_1 = b_1 / U_{1,1} \\ x_{k+1} = \frac{b_{k+1} - \sum_{j=1}^{k} U_{k+1,k+1-j} x_{k+1-j}}{U_{k+1,k+1}} & , k = 1, 2, \ldots, n-1 \end{cases}$$

implementazione Matlab

Gli algoritmi di sostituzione possono essere implementati rispettivamente come

```
n=size(U,1);
x=zeros(n,1);
x(n)=b(end)/U(n,n);
for k=1:n-1
    x(n-k)=(b(n-k)-U(n-k,:)*x)./U(n-k,n-k);
end
```

```
n=size(L,1);
x=zeros(n,1);
x(1)=b(1)/L(1,1);
for k=1:n-1
    x(k+1)=(b(k+1)-L(k+1,:)*x)./L(k+1,k+1);
end
```

Soluzione sistemi quadrati (metodi diretti)

Fattorizzazione LU

L'*eliminazione Gaussiana* (con piv.) è implementata in Matlab nella function `lu`.

`lu` in Matlab

Sia A matrice quadrata invertibile. Il comando `[L U P]=lu(A)` calcola le matrici

- `L` triangolare inferiore
- `U` triangolare superiore
- `P` di permutazione

tali che `L*U=P*A`.

NB:

- se A è invertibile la fattorizzazione esiste
- la presenz della matrice di permutazione è dovuta al pivoting
- `[L U]=lu(A)` fa comunque il pivoting e in generale $LU \neq A$

Soluzione sistemi quadrati con `lu` e sostituzione

Grazie ad `lu` possiamo risolvere sistemi quadrati invertibili con il seguente algoritmo

1. calcolo fattorizzazione $LU = PA$ con `lu`
2. soluzione di $Ly = Pb$ con *sostituzione avanti*
3. soluzione di $Ux = y$ con *sostituzione all'indietro*

(??) *Perchè Matlab non prevede LU senza pivoting?*

- solo sotto opportune condizioni sulla matrice quadrata invertibile A, è *teoricamente* possibile calcolare la fattorizzazione $LU = A$.
- l'algoritmo senza pivoting parziale è potenzialmente instabile: quando A è mal condizionata le matrici L ed U calcolate in aritmetica finita potrebbero essere "sensibilmente" non triangolari.

Vediamo un esempio sotto forma di esercizio.

LU senza pivoting in function I

Scarichiamo la function `LUnoPiv.m`

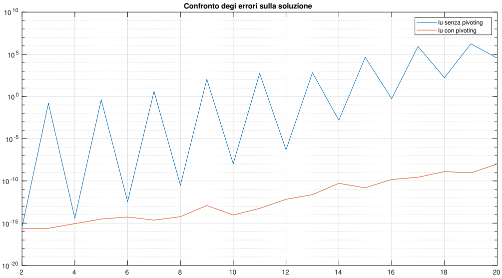
```
function [L, U] = LUnoPiv(A)
n = size(A,1);
L = eye(n);
for k = 1:n
    for i = k+1:n
        L(i,k) = A(i,k)/A(k,k);
        for j = k:n
            A(i,j) =  A(i,j) -L(i,k)*A(k,j);
        end
    end
end
U = A;
```

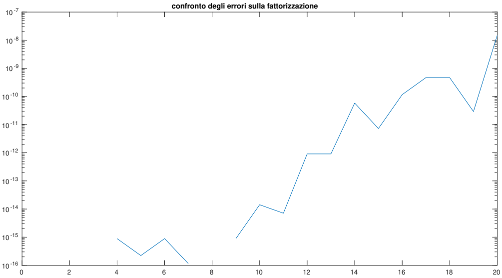
LU senza pivoting in function II

Esercizio 1

Per $n = 2, 3 \ldots, 20$ si creino n punti equispaziati z in $[-1, 1]$, si calcoli la matrice di Vandermonde nella base canonica `V=vander(z)`, si ponga $A = V + \epsilon\mathbb{I}$ con $\epsilon = 10^{-15}$, e si risolva il sistema lineare $Ax = b$ con b creato ad-hoc tramite $b = A * (1, 1, \ldots, 1)^t$ sia con `lu` che con `LUnoPiv`. Per ogni n si calcolino gli errori delle due soluzioni memorizzandole in un vettore e l'errore `norm(U-triu(U))`. Si plottino due figure (grafici semilogaritmici) per il confronto dei metodi.

Risultati


Confronto degli errori sulla soluzione
lu senza pivoting
lu con pivoting
10^10
10^5
10^0
10^-5
10^-10
10^-15
10^-20
2
4
6
8
10
12
14
16
18
20


confronto degli errori sulla fattorizzazione
10^-7
10^-8
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
0
2
4
6
8
10
12
14
16
18
20

Matlab backslash

E' possibile risolvere sistemi lineari $Ax = b$ con il comando Matlab detto backslash che prevede le due sintassi equivalenti

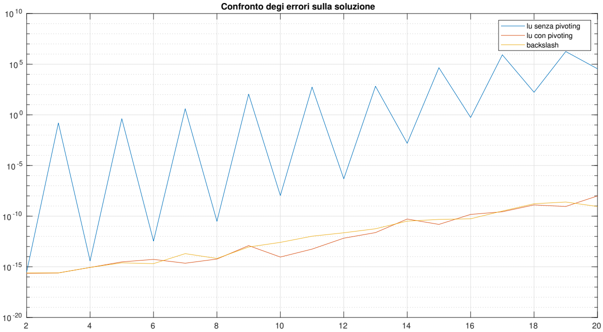
```
x=A\b

x=mldivide(A,b)
```

- Il comando backslash tenta la soluzione del sistema lineare $Ax = b$ ma non è disegnato solo per questo tipo di problemi.
- Non abbiamo controllo sull'algoritmo scelto da Matlab
- Backslash è in grado di risolvere generalizzazioni del problema $Ax = b$ e potrebbe decidere di reinterpretare il problema in questo senso. Ciò accade quasi certamente in presenza di problemi fortemente mal-condizionati.

Confronto con `lu`

Esercizio 2

Si modifichi lo script dell'Esercizio 1 in modo che venga anche calcolata la soluzione con il metodo backslash e plottato l'errore della soluzione.



Condizionamento di sistemi lineari

Richiamo teorico: norme matriciali

Fissata una norma $\|\cdot\|$ su $\mathbb{R}^n$ resta definita la *norma indotta* sulle matrici $M_{n\times n}(\mathbb{R})$:

$$\|A\| := \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|}.$$

Ad esempio:

- $\|A\|_2 = \max\{|\lambda|^{1/2} : \lambda \in \sigma(A^t A)\}$,`norm(A,2)`
- $\|A\|_1 = \max_j \sum_{i=1}^n |A_{i,j}|$, `norm(A,1)`
- $\|A\|_\infty = \max_i \sum_{j=1}^n |A_{i,j}|$, `norm(A,Inf)`

Richiamo teorico: condizionamento

Il numero di condizionamento di una matrice invertibile A *rispetto ad una norma* $\|\cdot\|$ è definito come

$$\kappa(A) := \|A\|\|A^{-1}\|.$$

Tale quantità può essere stimata in Matlab con il comando `cond(A,p)`, dove p può valere `1,2,Inf`.

Stime fondamentali

Se $\tilde{x} = x + \delta x$ risolve $A\tilde{x} = \tilde{b} := b + \delta b$ e $Ax = b$, si ha

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta b\|}{\|b\|}.$$

Se $\tilde{x} = x + \delta x$ risolve $(A + \delta A)\tilde{x} = \tilde{b} := b + \delta b$ e $Ax = b$, si ha

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

Assumiamo residuo piccolo. E l'errore?

In pratica noi calcoleremo sempre $\tilde{x} = x + \delta x \sim x$ e possiamo calcolare il residuo $\|A\tilde{x} - b\| = \|\delta b\|$, dalla precedente diseguaglianza abbiamo la *stima a posteriori*

$$err_{rel}(\tilde{x}) := \frac{\|\tilde{x} - x\|}{\|x\|} \leq \kappa(A) \frac{\|A\tilde{x} - b\|}{\|b\|}.$$

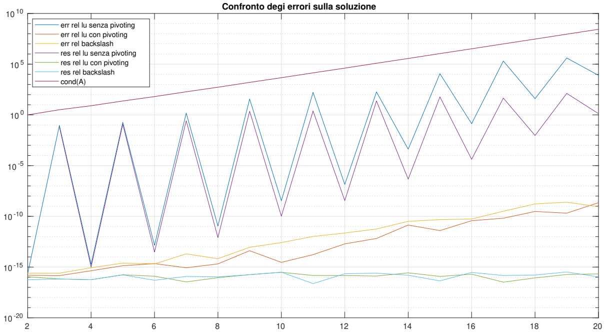
Non è difficile vedere anche che

$$err_{ass}(\tilde{x}) := \|\tilde{x} - x\| \leq \|A^{-1}\| \|A\tilde{x} - b\| = \|A^{-1}\| \|res(\tilde{x})\|.$$

Confronto errori-residui

Esercizio 3

Partendo dallo script dell'Esercizio 2, si modifichi il programma per ottenere uno script che, per ogni n e per ogni metodo, calcoli anche $err_{rel}(\tilde{x})$ e $\frac{\|A\tilde{x}-b\|}{\|b\|}$. Si produca una unica figura con $\kappa(A)$ e queste quantità per i tre metodi considerati.



Calcolo inversa

Per calcolare l'inversa della matrice invertibile A possiamo (non efficiente) risolvere n sistemi lineari $Ax^{(k)} = e_k$, con $e_1, e_2, \ldots, e_n$ base canonica di $\mathbb{R}^n$ ed ottenere $A^{-1} = [x^{(1)}, x^{(2)}, \ldots, x^{(n)}]$.

Esercizio 4

Scrivere una function `A1=myInv(A)` che implementi l'algoritmo sopra esposto, risolvendo i sitemi lineari con `lu` e le function di sostituzione.

Esercizio 4.1

E' possibile, con un piccolo sforzo di programmazione, modificare (senza introdurre cicli!) l'algoritmo di sostituzione in avati (risp. indietro) affinchè risolva sistemi *matriciali* del tipo

$$LX = B, \text{ (risp. } UX = B)$$

Ove cioè la j-esima colonna X_j di X risolve $LX_j = B_j$ (risp. $UX_j = B_j$), dove B_j è la j-esima colonna di B.
Scrivere una function `A1=Inv(A)` (si rispetti la lettera maiuscola, `inv` è una built in function) che, nel calcolo della matrice inversa, implementi tale algoritmo.

Test calcolo inversa

Esercizio 5

Si riprenda l'Esercizio 3 e lo si modifichi opportunamente per calcolare errori e residui *assoluti* anzichè errori e residui relativi e $\|A^{-1}\|$ anzichè $\kappa(A)$, producendo una figura analoga. Per il calcolo dell'inversa si utilizzi `myInv` o `Inv`.