

Vedremo come una fattorizzazione analoga a quella vista nelle lezioni precedenti, si possa ottenere con minor costo per matrici tridiagonali

Caso particolare: cerco di risolvere un sistema lineare

$A \underline{x} = b$ con A matrice tridiagonale

$$A = \begin{pmatrix} a_1 & c_1 & & & 0 \\ b_2 & a_2 & c_2 & & \\ & b_3 & a_3 & \ddots & \\ 0 & & & b_n & a_n \end{pmatrix} = LU$$

matrici NORD-OVEST

matrice A stessa

$$L = \begin{pmatrix} 1 & & & & 0 \\ l_2 & 1 & & & \\ & l_3 & 1 & & \\ 0 & & l_4 & \ddots & \\ & & & l_n & 1 \end{pmatrix} \quad U = \begin{pmatrix} u_1 & w_1 & & & 0 \\ & u_2 & w_2 & & \\ & & u_3 & \ddots & \\ 0 & & & u_{n-1} & \\ & & & & u_n \end{pmatrix}$$

Quello che vedremo ora, è un algoritmo che ci permette oltre che di ricavare queste matrici L e U , ci permette di calcolare i determinanti dei minori di nord ovest

$d_0 = 1$ serve solo per far tornare l'algoritmo

$$d_1 = \text{Det} \begin{pmatrix} a_1 \end{pmatrix} = a_1$$

$$d_2 = \text{Det} \begin{pmatrix} a_1 & c_1 \\ b_2 & a_2 \end{pmatrix} = a_1 a_2 - b_2 c_1 = d_1 a_2 - b_2 c_1 d_0$$

$$d_3 = \text{Det} \begin{pmatrix} a_1 & c_1 & \\ b_2 & a_2 & c_2 \\ & b_3 & a_3 \end{pmatrix} = d_2 a_3 - b_3 c_2 d_1$$

...

$$\star d_k = d_{k-1} \cdot a_k - b_k c_{k-1} d_{k-2} \quad k = 2, \dots, n$$

$$L = \begin{pmatrix} 1 & & & & 0 \\ b_2 \frac{d_0}{d_1} & 1 & & & \\ & b_3 \frac{d_1}{d_2} & 1 & & \\ 0 & & b_4 \frac{d_2}{d_3} & \ddots & \\ & & & b_n \frac{d_{n-2}}{d_{n-1}} & 1 \end{pmatrix} \quad U = \begin{pmatrix} \frac{d_1}{d_0} c_1 & & & & 0 \\ & \frac{d_2}{d_1} c_2 & & & \\ & & \frac{d_3}{d_2} c_3 & & \\ 0 & & & \ddots & \\ & & & & \frac{d_n}{d_{n-1}} \end{pmatrix}$$

Facendo il prodotto riga-colonna LU , data quella definizione di determinanti appena trovata, si può verificare che questa fattorizzazione restituisce la matrice di partenza.

Ma quanto ci costa l'algoritmo dei determinanti?

$$\text{somme} = n-1$$

$$\text{prodotti} = 3(n-1)$$

E il costo di L ?

$$\text{somme} = 0$$

$$\text{prodotti} = n-1$$

$$\text{divisioni} = n-1$$

E per il calcolo di U ?

$$\text{divisioni} = n-1$$

Quindi, globalmente, questo algoritmo quanto costa? $O(n)$

Essendo una matrice più sparsa, ma organizzata con una particolare struttura, il costo di fattorizzazione di una matrice tridiagonale è dell'ordine di n . [che rispetto all' n^3 delle matrici piene, è

molto più basso].

C'è una questione: io ho una fattorizzazione che mi costa LU e poi avrei un algoritmo di sostituzione in avanti/ all'indietro che mi dovrebbe costare n^2 , vediamo se in questo particolare caso, anche il costo delle risoluzioni dei sistemi lineari si sgonfia:

Supponiamo di dover risolvere il nostro sistema

Risoluzione sistema $A \underline{x} = \underline{t}$ con A matrice tridiagonale

$$A = LU \quad LU \underline{x} = \underline{t}$$

$$L \underline{y} = \underline{t} \quad \text{con L matrice bidiagonale inferiore}$$

$$\begin{cases} y_1 = t_1 \\ b_2 \frac{d_0}{d_1} y_1 + y_2 = t_2 \\ b_3 \frac{d_1}{d_2} y_2 + y_3 = t_3 \\ \vdots \\ b_n \frac{d_{n-2}}{d_{n-1}} y_{n-1} + y_n = t_n \end{cases}$$

Di fatto rimane una matrice triangolare inferiore, la risolverò quindi:

$$\begin{cases} y_1 = t_1 \\ y_2 = t_2 - b_2 \frac{d_0}{d_1} y_1 \\ y_3 = t_3 - b_3 \frac{d_1}{d_2} y_2 \\ \vdots \\ y_n = t_n - b_n \frac{d_{n-2}}{d_{n-1}} y_{n-1} \end{cases}$$

Costo computazionale: (n-1) somme
(n-1) prodotti
Totale = $O(n)$

Non abbiamo risolto il nostro sistema lineare di partenza, dobbiamo tornare indietro e risolvere

$$U \underline{x} = \underline{y}$$

$$\begin{cases} \frac{d_1}{d_0} \cdot x_1 + c_1 x_2 = y_1 \\ \frac{d_2}{d_1} \cdot x_2 + c_2 x_3 = y_2 \\ \frac{d_3}{d_2} \cdot x_3 + c_3 x_4 = y_3 \\ \vdots \\ \frac{d_{n-1}}{d_{n-2}} \cdot x_{n-1} + c_{n-1} x_n = y_{n-1} \\ \frac{d_n}{d_{n-1}} \cdot x_n = y_n \end{cases}$$

Da risolvere con algoritmo di sostituzione all'indietro

$$\begin{cases} x_n = y_n \frac{d_{n-1}}{d_n} \\ x_{n-1} = y_{n-1} - c_{n-1} x_n \cdot \frac{d_{n-2}}{d_{n-1}} \\ \vdots \\ x_2 = y_2 - c_2 x_3 \cdot \frac{d_1}{d_2} \\ x_1 = y_1 - c_1 x_2 \cdot \frac{d_0}{d_1} \end{cases}$$

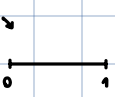
ce li siamo già calcolati = Sono la sottodiagonale di L

Costo computazionale : (n-1) Somme
 $2(n-1)+1$ prodotti
TOTALE: $O(n)$

Su Matlab, la struttura di una matrice ci permette di risparmiare costo computazionale e creare degli algoritmi ad hoc per la struttura particolare.

Vediamo quando una struttura di questo tipo può emergere per farci risolvere un problema.

Supponiamo di avere l'equazione del calore:

$$\begin{cases} \frac{d^2 u}{dx^2} = f(x) & x \in (0,1) \\ u(0) = a \\ u(1) = b \end{cases}$$


sorgente di calore

u è l'incognita e alla fine mi dovrà dire come è distribuita la temperatura lungo la barretta ab .

Bisogna ricordarsi lo sviluppo di Taylor

$$F(x+h) = F(x) + F'(x)h + F''(x)\frac{h^2}{2} + F'''(x)\frac{h^3}{3!} + F^{IV}(x)\frac{h^4}{4!} + F^V(x)\frac{h^5}{5!} + \dots$$

$$F(x-h) = F(x) - F'(x)h + F''(x)\frac{h^2}{2} - F'''(x)\frac{h^3}{3!} + F^{IV}(x)\frac{h^4}{4!} - F^V(x)\frac{h^5}{5!} + \dots$$

$$\begin{aligned} F(x+h) - F(x-h) &= F(x) - F(x) + F'(x)h - (-F'(x)h) + F''(x)\frac{h^2}{2} - F''(x)\frac{h^2}{2} \dots \\ &= 2F'(x)h + 2F'''(x)\frac{h^3}{3!} + 2F^V(x)\frac{h^5}{5!} + 2F^{VII}(x)\frac{h^7}{7!} + \dots \end{aligned}$$

Potrei andare avanti, ma mi fermo, posso trovare in questo modo un'approssimazione della derivata prima di una funzione in un punto, come?

Se io ignoro tutti i termini della derivata prima, posso dire che

$$F'(x) \cong \frac{F(x+h) - F(x-h)}{2h}$$

Allora posso ottenere un'approssimazione di una derivata prima di una funzione, come differenza della funzione in due punti centrati attorno al punto in cui voglio valutare la derivata prima.

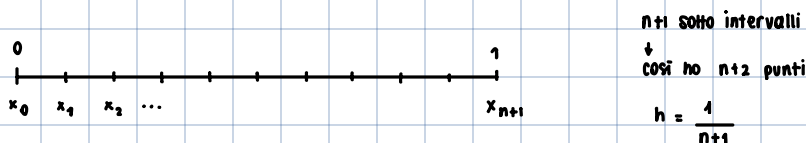
Purtroppo a noi serve la derivata seconda, come posso ottenere da quelle due espressioni $F(x+h)$ e $F(x-h)$, la derivata seconda togliendomi dalle scatole la derivata prima? Invece che sottrarle, posso sommarle.

$$F(x+h) + F(x-h) = 2F(x) + 2F''(x)\frac{h^2}{2} + 2F^{IV}(x)\frac{h^4}{4} + \dots$$

$$F''(x) \cong \frac{-2F(x) + F(x+h) + F(x-h)}{h^2}$$

Come faccio a sfruttare questa equazione per risolvere il sistema di partenza?

Devo creare una griglia:



Andiamo ad approssimare la derivata seconda

$$f(x_i) = u''(x_i) \cong \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1}))}{h^2} \quad i = 1, \dots, n$$

Questa è un'approssimazione che io do, ed la do in ogni punto che va da i a n , di fatto è come se creassi n equazioni

$$\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1}))}{h^2} \cong f(x_i) \quad i = 1, \dots, n$$

Se risolvessi queste n equazioni, non otterrei proprio la soluzione esatta, quindi dico che la mia soluzione approssimata, la chiamo

$$u_i \approx u(x_i)$$

$$\begin{cases} \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f(x_i) & i = 1, \dots, n \\ u(x_0) = u_0 = a \\ u(x_{n+1}) = u_{n+1} = b \end{cases}$$

Come è fatto questo sistema lineare? Traduciamolo in forma matriciale

$$\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \begin{pmatrix} -2 & 1 & & 0 \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & \ddots & \ddots \\ 0 & & & 1 & -2 \end{pmatrix} = \begin{pmatrix} f(x_1) \cdot h^2 - a \\ f(x_2) \cdot h^2 \\ \vdots \\ f(x_n) \cdot h^2 - b \end{pmatrix}$$

nota, quindi la porto al secondo membro

$$\begin{cases} i=1 & u_2 - 2u_1 + u_0 = f(x_1) \cdot h^2 \rightarrow u_2 - 2u_1 = f(x_1) \cdot h^2 - u_0 \\ i=2 & u_3 - 2u_2 + u_1 = f(x_2) \cdot h^2 \\ i=3 & u_4 - 2u_3 + u_2 = f(x_3) \cdot h^2 \\ \vdots & \\ i=n-1 & u_n - 2u_{n-1} + u_{n-2} = f(x_{n-1}) \cdot h^2 \\ i=n & u_{n+1} - 2u_n + u_{n-1} = f(x_n) \cdot h^2 \rightarrow -2u_n + u_{n-1} = f(x_n) \cdot h^2 - u_{n+1} \end{cases}$$

Questi che abbiamo visto, sono metodi diretti e hanno i corrispettivi comandi su Matlab:

$$A \cdot x = b$$

Risolvere un sistema lineare in Matlab: $A \setminus b$ oppure $\text{mldivide}(A, b)$

Fattorizzazione LU: $[L, U] = \text{lu}(A)$ oppure $[L, U, P] = \text{lu}(A) \rightarrow$ return unit lower triangular matrix L, upper triangular matrix U, and permutation matrix P such that $P^*A = L^*U$.

METODI ITERATIVI

Perché uno dovrebbe essere interessato a risolvere un sistema lineare con un metodo che non ci assicura la soluzione esatta in un numero finito di passi?

I metodi iterativi si contrappongono ai metodi diretti perché la soluzione del sistema lineare $Ax=b$ viene ottenuta "formalmente" dopo un numero infinito di passi.

Come abbiamo visto, anche i metodi diretti non danno un risultato preciso e accurato e ci impiegano molto tempo, i metodi iterativi mi danno un risultato approssimato ma in meno tempo.

Quindi la scelta sta nel "quanto tempo ho per ottenere un risultato?" e "quanto accurato voglio che sia il risultato?"

I metodi iterativi costruiscono una successione di vettori che mi aspetto tendano alla soluzione esatta

$$\{x^{(k)}\} \quad x^{(k)} \in \mathbb{R}^n \\ x^{(k)} \rightarrow x \quad \text{soluzione esatta di } Ax = b$$

Def: Sia $x^{(k)}$ una successione di vettori di \mathbb{R}^n , essa converge al vettore $x \in \mathbb{R}^n$ se esiste una norma vettoriale per cui:

$$\lim_{k \rightarrow +\infty} \|x - x^{(k)}\| = 0$$

Oss: la convergenza si ha componente per componente

$$|x_i - x_i^{(k)}| \leq \|x - x^{(k)}\|_\infty \leq c \|x - x^{(k)}\|_1 \longrightarrow 0 \quad i=1, \dots, n$$

Noi accetteremo metodi iterativi consistenti

Def: un metodo iterativo si dice consistente se:

$$\underline{x}^{(k)} \equiv \underline{x} \longrightarrow \underline{x}^{(k+1)} \equiv \underline{x} \quad i=1, \dots$$

↓
coincide con la soluzione esatta.

(= se io ad un certo passo del mio algoritmo, trovo la soluzione esatta, anche in tutti i passi successivi continuerò a trovare quella stessa soluzione esatta. Rimarrò bloccata.)