Dispense di

Laboratorio di Calcolo Numerico 2

Prof. Marco Caliari

a.a. 2017/18

Questi appunti non hanno nessuna pretesa di completezza. Sono solo alcune note ed esercizi che affiancano l'insegnamento di Laboratorio di Calcolo Numerico 2. Sono inoltre da considerarsi in perenne "under revision" e pertanto possono contenere discrepanze, inesattezze o errori. Si userà il termine MATLAB per indicare il linguaggio di programmazione comune a GNU Octave e Matlab®.

Questa è la versione del 20 febbraio 2018. La versione più aggiornata si trova all'indirizzo

http://profs.scienze.univr.it/caliari/aa1718/calcolo_numerico2/dispense.pdf

Indice

Capitolo 1

Metodi iterativi per sistemi lineari

I metodi iterativi per sistemi lineari si usano quando si vogliono sfruttare alcune proprietà della matrice (per esempio la sparsità), oppure quando la soluzione del sistema è richiesta a meno di una certa tolleranza. I fattori da invertire nel calcolo delle matrici di iterazione sono "facilmente invertibili", cioè il costo dell'inversione è proporzionale, al massimo, al costo di un prodotto matrice-vettore (dunque quadratico nell'ordine della matrice) e non cubico come nell'applicazione di un metodo diretto per matrici piene.

Potrebbe essere necessario un pivoting parziale preventivo per poter applicare i metodi iterativi, come si vede nel caso di una matrice

$$A = \begin{bmatrix} 0 & 1 & 4 & 1 \\ 1 & 4 & 1 & 0 \\ 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

Nel seguito useremo la decomposizione $D + L + U = A$, ove D è la diagonale di A e L and U sono, rispettivamente, la parte triangolare bassa e la parte triangolare alta di A, private della diagonale (si possono ottenere con `tril (A, -1)` e `triu (A, 1)`).

1.1 Memorizzazione di matrici sparse

Sia A una matrice di ordine n con m elementi diversi da zero. La si definisce *sparsa* se $m = \mathcal{O}(n)$ invece che $m = \mathcal{O}(n^2)$. Esistono molti formati di memorizzazione di matrici sparse. Quello usato da GNU Octave e Matlab® 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 elemento i-esimo ($i < n+1$) è la posizione dentro `data` del primo elemento della colonna i e l'elemento $(n+1)$-esimo è il numero totale di elementi diversi da zero incrementato di uno. 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

$$\begin{aligned} \texttt{data} &= [1, 4, 2, 3, 5, 6, 7] \\ \texttt{ridx} &= [1, 3, 2, 2, 3, 3, 4] \\ \texttt{cidx} &= [1, 3, 4, 6, 8] \end{aligned}$$

Talvolta, soprattutto in linguaggi di calcolo con array che iniziano dall'indice 0, gli array `ridx` e `cidx` hanno elementi decrementati di uno.

In MATLAB, il formato CCS e l'implementazione del prodotto matrice-vettore sono automaticamente usati dalla function `sparse` e dall'operatore `*`, rispettivamente.

1.1.1 Alcuni comandi per matrici sparse

- Il comando `speye(n)` genera la matrice identità di ordine n.

- Il comando `spdiags(v,0,n,n)`, ove v è un vettore colonna, genera la matrice diagonale di ordine n avente v in diagonale. Se la dimensione di v è minore di n, la diagonale viene riempita con zeri posti dopo il vettore v. Se invece la dimensione di v è maggiore di n, vengono usate solo le prime n componenti di v.

  Sia V la matrice

$$V = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ \vdots & \vdots & \vdots \\ v_{n1} & v_{n2} & v_{n3} \end{pmatrix}$$

Il comando `spdiags(V,-1:1,n,n)` genera la matrice

$$\begin{pmatrix} v_{12} & v_{23} & 0 & 0 & \dots & 0 \\ v_{11} & v_{22} & v_{33} & 0 & \dots & 0 \\ 0 & v_{21} & v_{32} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & v_{n-2\,1} & v_{n-1\,2} & v_{n3} \\ 0 & \dots & \dots & 0 & v_{n-1\,1} & v_{n2} \end{pmatrix}$$

1.1.2 Metodo di Jacobi

Nel metodo di Jacobi, la matrice di iterazione J_{J} è

$$J_{\text{J}} = -D^{-1}(L+U) = -D^{-1}(-D+A) = (I - D^{-1}A)$$

e dunque

$$x^{(k+1)} = J_{\text{J}} x^{(k)} + D^{-1}b \tag{1.1}$$

La matrice diagonale D può essere costruita, direttamente in formato sparso, con il comando

```
D = spdiags(diag(A),0,size(A,1),size(A,2))
```

È possibile scrivere il metodo di Jacobi per componenti: si ha

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}, \quad i = 1, 2, \dots, n$$

Come test di arresto si può usare

$$\frac{\|r^{(k+1)}\|}{\|b\|} \leq \text{tol}$$

Infatti, siccome

$$e^{(k+1)} = x^{(k+1)} - x = A^{-1}Ax^{(k+1)} - A^{-1}b = -A^{-1}(b - Ax^{(k+1)}) = -A^{-1}r^{(k+1)}$$

e

$$\|b\| \leq \|A\|\,\|x\|$$

si ha

$$\frac{\|e^{(k+1)}\|}{\|x\|} \leq \|A^{-1}\|\,\|A\|\,\frac{\|r^{(k+1)}\|}{\|b\|} \leq \text{cond}(A) \cdot \text{tol}$$

Il metodo di Jacobi è parallelizzabile, cioè ad ogni iterazione il calcolo di $x_i^{(k+1)}$ può essere fatto indipendentemente dal calcolo di $x_j^{(k+1)}$, se $i \neq j$.

Metodo di Jacobi per blocchi

Nella discretizzazione di equazioni alle derivate parziali in dimensione maggiore di uno è facile trovare sistemi lineari di tipo

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & \dots & A_{2N} \\ \vdots & \vdots & \dots & \vdots \\ A_{N1} & A_{N2} & \dots & A_{NN} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix}$$

ove ogni A_{ij} è una matrice di ordine n. Si può applicare il metodo di Jacobi per blocchi

$$A_{II} X_I^{(k+1)} = \left(B_I - \sum_{J=1}^{I-1} A_{IJ} X_J^{(k)} - \sum_{J=I+1}^{N} A_{IJ} X_J^{(k)} \right), \quad I = 1, 2, \dots, N$$

ove per ogni componente I è richiesta la soluzione di un sistema lineare di matrice (piccola) A_{II}. Ognuno di questi N sistemi lineari potrebbe essere risolto indipendentemente dagli altri. Poiché ad ogni iterazione le matrici di questi N sistemi lineari non cambiano, esse possono essere fattorizzate una volta e per tutte.

1.1.3 Metodo di Gauss–Seidel

Nel metodo di Gauss–Seidel, la matrice di iterazione J_{GS} è

$$J_{\mathrm{GS}} = -(D+L)^{-1}U = -(D+L)^{-1}(-(D+L)+A) = (I-(D+L)^{-1}A)$$

e dunque

$$x^{(k+1)} = J_{\mathrm{GS}} x^{(k)} + (D+L)^{-1} b$$

La matrice $D+L$ è triangolare inferiore: dunque la matrice di iterazione J_{GS} può essere calcolata risolvendo il sistema lineare

$$(D+L) B_{\mathrm{GS}} = -U$$

per mezzo dell'algoritmo delle sostituzioni in avanti. L'operatore \ di GNU Octave e Matlab® applica automaticamente questa tecnica quando la matrice del sistema è triangolare. È possibile scrivere il metodo di Gauss–Sidel per componenti: si ha

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}, \quad i = 1, 2, \dots, n$$

Il metodo di Gauss–Sidel può essere implementato usando un solo vettore `x`.

Per matrici simmetriche e definite positive il metodo di Gauss–Sidel converge. Se inoltre sono tridiagonali, allora convere anche il metodo di Jacobi, ma più lentamente, poiché

$$\rho(J_{\mathrm{GS}}) = \rho(J_{\mathrm{J}})^2.$$

1.1.4 Metodo del gradiente a parametro ottimale

L'idea è quella di risolvere il sistema lineare

$$Ax = b$$

con A simmetrica e definita positiva ricorrendo ad un metodo iterativo che minimizzi ad ogni passo il funzionale $E\colon \mathbb{R}^n \to \mathbb{R}$

$$E(x^{(k)}) = (x^{(k)} - x)^T A(x^{(k)} - x)$$

il cui gradiente rispetto ad $x^{(k)}$ si annulla proprio per $Ax^{(k)} - b = 0$. Si ha infatti

$$\nabla E(x^{(k)}) = \nabla \left(\left(x^{(k)}\right)^T Ax^{(k)} - 2x^T Ax^{(k)} + x^T Ax \right) = 2 \left(x^{(k)}\right)^T A - 2b^T$$

Definito l'errore al passo k come $x^{(k)} - x$, $E(x^{(k)})$ risulta essere una norma al quadrato dell'errore. Nel metodo del gradiente si costruisce una nuova iterazione $x^{(k+1)}$ a partire da $x^{(k)}$ aggiungendo un'opportuna scalatura della direzione di massima discesa data dalla direzione del gradiente, con verso opposto e dunque

$$x^{(k+1)} = x^{(k)} + \alpha_k(-Ax^{(k)} + b) = x^{(k)} + \alpha_k r^{(k)}$$

Fissata tale direzione, si tratta di scegliere il valore effettivo di α_k. Si può prendere il parametro *ottimale*, cioè quello che realizza

$$\alpha_k = \arg\min_{\alpha} E(x^{(k)} + \alpha r^{(k)})$$

La funzione $\alpha \mapsto E(x^{(k)} + \alpha r^{(k)})$ è una parabola rivolta verso l'alto e dunque il suo minimo è raggiunto in corrispondenza del vertice, per cui

$$\alpha_k = \frac{(r^{(k)}, r^{(k)})}{(Ar^{(k)}, r^{(k)})}$$

ove $(\cdot, \cdot)$ indica il prodotto scalare in $\mathbb{R}^n$. Tenuto conto che

$$r^{(k+1)} = b - Ax^{(k+1)} = b - A(x^{(k)} + \alpha_k r^{(k)}) = \left(b - Ax^{(k)}\right) - \alpha_k Ar^{(k)} = r^{(k)} - \alpha_k Ar^{(k)}$$

l'iterazione di base del metodo del gradiente a parametro ottimale può essere scritta

$$\begin{aligned} w^{(k)} &= Ar^{(k)} \\ \alpha_k &= \frac{(r^{(k)}, r^{(k)})}{(w^{(k)}, r^{(k)})} \\ x^{(k+1)} &= x^{(k)} + \alpha_k r^{(k)} \\ r^{(k+1)} &= r^{(k)} - \alpha_k w^{(k)} \end{aligned}$$

Il metodo richiede un solo prodotto matrice-vettore per ogni iterazione e viene usato tipicamente quando la matrice è sparsa. Il valore α_k risulta essere non negativo. Tuttavia, monitorare la positività di tale valore risulta utile per capire se il metodo viene erroneamente usato per matrici non definite positive.

Calcolo di gradienti di funzioni vettoriali

Il calcolo di $\nabla E(x^{(k)}$ può essere eseguito scrivendo $(x^{(k)} - x)^T A(x^{(k)} - x)$ per componenti. Si può anche calcolare

$$\lim_{\varepsilon \to 0} \frac{E\left(x^{(k)} + \varepsilon v\right) - E\left(x^{(k)}\right)}{\varepsilon} = 2\left(x^{(k)}\right)^T Av - 2b^T v$$

da cui discende il risultato. Quest'ultima formula ha il vantaggio che è possibile testarla in MATLAB

```
> A = rand (4);
> A = A * A'; % matrice simmetrica definita positiva
> b = rand (4, 1);
> x = A \ b;
> E=@(xk) (xk - x)' * A * (xk - x);
> nablaE = @(xk) 2 * xk' * A - 2 * b';
> v = rand (4, 1);
> xk = rand (4, 1);
> nablaE (xk) * v
  ans =  3.5302 % valore da testare
> epsilon = 1e-4; % sufficientemente piccolo ma non troppo
> (E(xk + epsilon * v) - E(xk)) / epsilon
  ans =  3.5304 % approssimazione del limite
```

1.2 Esercizi

1. Si implementi il metodo SOR con una function `sor.m`.

2. Dato `m`, si costruisca la matrice

```
kron(toeplitz([2,-1,zeros(1,m-2)]),eye(m))+...
kron(eye(m),toeplitz([2,-1,zeros(1,m-2)]))
```

   e si applichi il metodo di Jacobi a blocchi, usando un opportuno solutore per i sistemi lineari di ogni blocco.

3. Si considerino le matrici

$$A_1 = \begin{bmatrix} 1 & -2 & 2 \\ -1 & 1 & -1 \\ -2 & -2 & 1 \end{bmatrix}, A_2 = \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{bmatrix}, A_3 = \begin{bmatrix} 1 & 4 & 0 & -2 \\ 0 & 2 & 5 & 1 \\ 4 & 0 & 1 & -1 \\ -2 & 1 & 0 & 5 \end{bmatrix}$$

   e si discutano le proprietà delle matrici di iterazioni dei metodi di Jacobi e Gauss–Sidel. È possibile effettuare degli scambi di righe sulla matrice A_3 tali da rendere convergente il metodo di Jacobi?

4. Si implementi una function `p = scambiarighe (A)` che restituisce un vettore `p` in modo che si abbia $|a_{p(i),i}| = \max_{i \le j \le n} |a_{j,i}|$, per $i = 1, 2, \ldots, n$. Si provi ad implementarla senza scambiare effettivamente le righe di A.

5. Si implementi il metodo di Jacobi con pivoting parziale delle righe, senza scambiare effettivamente le righe della matrice.

6. Si dimostri che $E(x^{(k)})$ è una norma al quadrato e che il valore ottimale di α è proprio α_k.

Capitolo 2

Fattorizzazioni

2.1 Fattorizzazione QR

La fattorizzazione QR di una matrice $A \in \mathbb{R}^{m\times n}$ è

$$A = QR \tag{2.1}$$

con $Q \in \mathbb{R}^{m\times m}$ matrice ortogonale ($Q^TQ = I_m$) e $R \in \mathbb{R}^{m\times n}$ matrice triangolare superiore. In MATLAB la fattorizzazione QR è disponibile attraverso il comando `[Q, R] = qr (A)`. L'algoritmo usato si basa sulle trasformazioni elementari di Householder e costa $\mathcal{O}(n^2(m - n/3)$.

In MATLAB è possibile costruire anche la fattorizzazione QR con *pivoting*. Con il comando `[Q, R, P] = qr (A)` si ottiene la fattorizzazione $QR = AP$, ove P è matrice di permutazione (delle colonne) ed R ha elementi diagonali decrescenti in modulo.

2.1.1 Matrici di Householder

Siano x e w due vettori (colonna). Si vuole calcolare il vettore x' ottenuto per riflessione di x attraverso l'iperpiano π ortogonale a w. Si ha

$$x' = x - 2(w^Tx)w/\|w\|_2^2 = \left(I - \frac{2}{\|w\|_2^2}ww^T\right)x = Q_wx$$

(si veda Figura 2.1). La matrice Q_w è banalmente simmetrica ed anche ortogonale: infatti

$$Q_w^TQ_w = Q_wQ_w = \left(I - \frac{2}{\|w\|_2^2}ww^T\right)^2 = I - \frac{4}{\|w\|_2^2}ww^T + \frac{2}{\|w\|_2^2}ww^T\frac{2}{\|w\|_2^2}ww^T =$$
$$= I - \frac{4}{\|w\|_2^2}ww^T + \frac{4}{\|w\|_2^4}w\|w\|_2^2w^T = I.$$

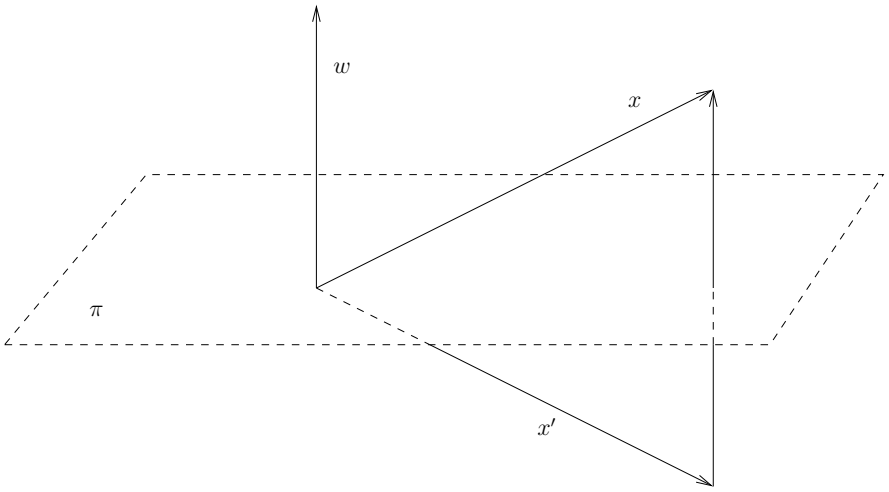

Figura 2.1: Vettore x' riflessione di x attraverso π ortogonale a w.

Data una matrice

$$A = \begin{bmatrix} x_1 & \dots & \dots & \dots \\ x_2 & \dots & \dots & \dots \\ \vdots & \dots & \dots & \dots \\ x_m & \dots & \dots & \dots \end{bmatrix}$$

cerchiamo una matrice di Householder $Q_{w^{(1)}}$ tale che

$$Q_{w^{(1)}} A = \begin{bmatrix} \alpha x_1 & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots \\ \vdots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots \end{bmatrix}$$

Si ha $w_1^{(1)} = x_1 \pm \sqrt{\sum_{j=1}^m x_j^2} = x_1 \pm \|x\|_2$ e $w_j^{(1)} = x_j$, $j = 2, 3, \dots, m$. Infatti si ha

$$\|w^{(1)}\|_2^2 = x_1^2 + \sum_{j=1}^m x_j^2 \pm 2x_1 \sqrt{\sum_{j=1}^m x_j^2} + \sum_{j=2}^m x_j^2 = 2\sum_{j=1}^m x_j^2 \pm 2x_1 \sqrt{\sum_{j=1}^m x_j^2}$$

e

$$w^{(1)^T} x = x_1^2 \pm x_1 \sqrt{\sum_{j=1}^m x_j^2} + \sum_{j=2}^m x_j^2 = \sum_{j=1}^m x_j^2 \pm x_1 \sqrt{\sum_{j=1}^m x_j^2} = \frac{\|w^{(1)}\|_2^2}{2}$$

e pertanto

$$Q_{w^{(1)}}x = x - \frac{2}{\|w^{(1)}\|_2^2} w^{(1)} \frac{\|w^{(1)}\|_2^2}{2} = x - w^{(1)}$$

da cui si vede che

$$\alpha x_1 = \mp \|x\|_2.$$

Solitamente si disambigua $\pm$ scegliendo $\text{sign}(x_1)$. Si continua poi scegliendo

$$Q_{w^{(k)}} = \begin{bmatrix} I_k & 0 \\ 0 & I - \frac{2}{\|w^{(k)}\|_2^2} w^{(k)} w^{(k)T} \end{bmatrix}$$

analogamente, con $w^{(k)} \in \mathbb{R}^{m+1-k}$. A questo punto, il prodotto $Q_{w^{(\min\{m,n\}-1)}} \dots Q_{w^{(1)}} A$ è la matrice triangolare superiore R e dunque $Q = Q_{w^{(1)}} \dots Q_{w^{(\min\{m,n\}-1)}}$.

2.2 Decomposizione SVD

La decomposizione SVD di $A \in \mathbb{R}^{m\times n}$ è $A = USV^T$ ove $U \in \mathbb{R}^{m\times m}$ e $V \in \mathbb{R}^{n\times n}$ sono matrici ortogonali e $S \in \mathbb{R}^{m\times n}$ "diagonale" con elementi σ_i, $i = 1, 2, \dots, \min\{m, n\}$. Gli elementi σ_i soddisfano $\sigma_1 \geq \sigma_2 \geq, \dots, \sigma_{\min\{m,n\}} \geq 0$ e r di loro sono strettamente positivi se $r = \text{rank}(A)$. I valori σ_i si chiamano valori singolari di A e coincidono con le radici degli autovalori di A^TA. La decomposizione SVD si ottiene in MATLAB con il comando `[U, S, V] = svd (A)`.

2.3 Uso delle fattorizzazioni

2.3.1 Sistemi sovradeterminati

Consideriamo un sistema lineare sovradeterminato

$$Ax = b, \quad A \in \mathbb{R}^{m\times n},\ m > n,\ \text{rank}(A) = n,\ \text{rank}(A|b) = n + 1$$

Poiché non esiste *la* soluzione, si può cercare la soluzione *ai minimi quadrati*, cioè quella che minimizza la norma euclidea del residuo

$$b - Ax$$

Si ha

$$\|b - Ax\|_2^2 = b^Tb - 2x^TA^Tb - x^TA^TAx$$

il cui gradiente è $2A^Tb - 2A^TAx$. Pertanto, la soluzione di

$$A^TAx = A^Tb$$

(chiamato sistema delle equazioni normali) minimizza la norma euclidea del residuo. La matrice A^TA è definita positiva (infatti i suoi n autovalori sono i valori singolari di A che, essendo A di rango n, sono positivi). Pertanto, è teoricamente possibile utilizzare la fattorizzazione di Cholesky per risolvere il sistema. In maniera più efficiente, si può usare la fattorizzazione $A = QR$, con $Q \in \mathbb{R}^{n\times n}$ ortogonale e $R \in \mathbb{R}^{n\times m}$ triangolare superiore, senza il bisogno di calcolare esplicitamente la matrice A^TA. Si ha infatti

$$A^TAx = A^Tb \Leftrightarrow R^TQ^TQRx = R^TQ^Tb \Leftrightarrow R^T(Rx - Q^Tb) = 0.$$

Poiché le ultime $m - n$ colonne di R^T sono nulle, basta considerare le prime n righe del sistema $Rx = Q^Tb$, cioè

$$(I_{n,m}R)x = (I_{n,m}Q^T)b$$

ove

$$I_{n,m} = \begin{bmatrix} I_n & 0 \end{bmatrix} \in \mathbb{R}^{n\times m}$$

In ogni caso, il numero condizionamento della matrice A^TA potrebbe essere molto elevato. Si può ricorrere alla decomposizione SVD. Sia

$$A = USV^T, \quad U \in \mathbb{R}^{m\times m},\ S \in \mathbb{R}^{m\times n},\ V \in \mathbb{R}^{n\times n}$$

con U e V matrici ortogonali e

$$S = \begin{bmatrix} \sigma_1 & 0 & \dots & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \sigma_{n-1} & 0 \\ 0 & \dots & \dots & 0 & \sigma_n \\ 0 & \dots & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & 0 \end{bmatrix}$$

La soluzione ai minimi quadrati è quella che minimizza $\|b - Ax\|_2$. Per l'ortogonalità di U e V, si ha

$$\|b - Ax\|_2 = \|U^T(b - AVV^Tx)\|_2 = \|d - Sy\|_2$$

con $d = U^Tb$ e $y = V^Tx$. Il minimo di $\|d - Sy\|_2$ si ha per $y_i = d_i/\sigma_i$, $i = 1, \dots, n$, da cui poi si ricava $x = Vy$.

Il comando `x = A\b` di MATLAB usa automaticamente la decomposizione SVD per la risoluzione ai minimi quadrati di un sistema sovradeterminato.

2.3.2 Sistemi sottodeterminati

Consideriamo il caso di un sistema *sottodeterminato*

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

con $\text{rank}(A) = m$. Tra le infinite soluzioni, possiamo considerare quella di norma euclidea minima. Consideriamo ancora la decomposizione SVD, ove

$$S = \begin{bmatrix} \sigma_1 & 0 & \dots & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 & \vdots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & 0 & \sigma_{m-1} & 0 & \vdots & \dots & \vdots \\ 0 & \dots & \dots & 0 & \sigma_m & 0 & \dots & 0 \end{bmatrix}$$

Da $Ax = b$ si ricava $SV^T x = U^T b$ e, per l'ortogonalità di V, minimizzare la norma euclidea di x equivale a minimizzare la norma euclidea di $y = V^T x$. Il sistema $Sy = d$, ove $d = U^T b$, ammette come soluzione di norma euclidea minima il vettore $y_i = d_i/\sigma_i$, $i = 1, \dots, m$, $y_i = 0$, $i = m+1, \dots, n$, da cui poi si ricava $x = Vy$. Il comando `x = A\b` di GNU Octave usa automaticamente la decomposizione SVD per trovare la soluzione di norma euclidea minima di un sistema sottodeterminato, anche in questo caso chiamata soluzione ai minimi quadrati.

Nel caso di sistemi sottoderminati, un'altra soluzione interessante è quella con un numero di elementi uguali a zero pari a $n - m$. Si considera la fattorizzazione QR con pivoting $AP = QR$. Si trova

$$A^T Ax = A^T b \Leftrightarrow PR^T(Ry - Q^T b) = 0$$

ove $y = P^T x$. Il sistema $Ry = Q^T b$ è sottodeterminato, dunque si possono scegliere $y_i = 0$, $i = m + 1, \dots, n$. Poi si ricava $x = Py$ che, essendo una permutazione di y, mantiene la stessa sparsità. Un'implementazione efficiente dal punto di vista dell'occupazione di memoria è la seguente:

```
[Q, R, p] = qr (A, 0);
x = R(:,1:size (A, 1)) \ (Q' * b);
x(size (A, 2)) = 0;
x(p) = x;
```

Il comando `x = A\b` di Matlab® usa automaticamente la fattorizzazione QR con pivoting per trovare la soluzione più sparsa di un sistema sottodeterminato.

2.3.3 Altri sistemi lineari

Il bestiario dei sistemi lineari non è purtroppo finito. Prendiamo ad esempio

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}$$

È sottodeterminato, ci sono infinite soluzioni, GNU Octave resitutisce $[x_1, x_2] = [0.2, 0.4]$ e Matlab® $[x_1, x_2] = [0, 0.5]$. Se però consideriamo il sistema equivalente

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

questo è un sistema quadrato singolare, GNU Octave restituisce la medesima soluzione (con un warning sulla singolarità della soluzione) e Matlab® restituisce un vettore di `NaN`. Se aggiungiamo un'altra riga equivalente otteniamo i risulati iniziali. Ci sono poi sistemi come

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

che GNU Octave considera sovradeterminati e per cui restituisce $[x_1, x_2] = [0.3, 0.6]$ e Matlab® singolari e per cui restituisce un vettore di `Inf`. Se però aggiungiamo una riga equivalente

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

con un po' di fatica si scopre che GNU Octave applica lo schema

```
[U, S, V] = svd (A);
d = U' * b;
y(1) = d(1) / S(1, 1);
y(2) = 0; % S(2, 2) e' nullo
x = V * y;
```

mentre Matlab®

```
[Q, R, p] = qr (A, 0);
d = Q' * b;
x = R(1, :) \ d(1); % sistema sottodeterminato
x(p) = x;
```

Altri casi interessanti delle differenze tra GNU Octave e Matlab® si possono trovale nel web[1].

[1] https://lists.gnu.org/archive/html/octave-maintainers/2008-04/msg00160.html

2.4 Pseudoinversa

2.5 Esercizi

1. Si generi una matrice random A di dimensione 4×3 e si calcoli $w^{(1)}$ tale che $Q_{w^{(1)}} A$ abbia la prima colonna nulla eccetto l'elemento in posizione $(1,1)$. Si verifichi che tale colonna coincide con la prima colonna della matrice R ottenuta per fattorizzazione QR di A. Inoltre, si verifichi che la prima colonna di $Q_{w^{(1)}}$ coincide con la prima colonna di Q. Perché?

2. Si provi a scrivere l'algoritmo che calcola la fattorizzazione QR di una matrice, usando le matrici di riflessione di Householder.

3. Si consideri una matrice quadrata A. Si costruisca una matrice Q tale che l'elemento in posizione $(1,1)$ di QA coincida con $a_{1,1}$, l'elemento in posizione $(2,1)$ sia diverso da zero e tutti gli altri elementi della prima colonna siano nulli. Quanti elementi nulli ha la matrice QAQ? E se A è simmetrica? Quali sono gli autovalori di QAQ?

4. Si consideri una matrice quadrata A di dimensione $m \times m$. Si costruisca $Q_{w^{(1)}}$ come sopra. Si costruisca adesso $P_{z^{(1)}}$ tale che

$$Q_{w^{(1)}} A P_{z^{(1)}} = \begin{bmatrix} * & * & 0 & \dots & 0 \\ 0 & * & * & \dots & * \\ 0 & * & * & \dots & * \\ \vdots & * & * & \dots & * \\ 0 & * & * & \dots & * \end{bmatrix}$$

   Quali sono i valori singolari di $Q_{w^{(1)}} A P_{z^{(1)}}$? Procedendo in questo modo, qual è la struttura che raggiunge la matrice $Q_{w^{(m-1)}} \dots Q_{w^{(1)}} A P_{z^{(1)}} \dots P_{z^{(n-2)}}$?

5. Si consideri una matrice random di dimensione 4×3. Si verifichi che MATLAB calcola la soluzione dei sistemi lineari sovradeterminati con questa matrice come descritto in sezione 2.3.1.

6. Si consideri una matrice random di dimensione 3×4. Si verifichi che GNU Octave/Matlab® calcola la soluzione dei sistemi lineari sottodeterminati con questa matrice come descritto in sezione 2.3.2.

7. Si verifichino le soluzioni della sezione 2.3.3.

8. Se x è vettore complesso di lunghezza m, con $x_1 = re^{i\theta}$, si dimostri che si può scegliere $w_1^{(1)} = x_1 \pm e^{i\theta} \|x\|_2$, $w_j = x_j$, $j = 2, 3, \dots, m$.

Capitolo 3

Sistemi non lineari

3.1 Metodo delle secanti in due dimensioni

Si vuole trovare la soluzione di

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$$

Date tre coppie di punti (x_1, y_1), (x_2, y_2), (x_3, y_3), si possono costruire i due piani $a_1 + a_2 x + a_3 y$ e $b_1 + b_2 x + b_3 y$ passanti per $(x_1, y_1, f(x_1, y_1))$, $(x_2, y_2, f(x_2, y_2))$ e $(x_3, y_3, f(x_3, y_3))$ e $(x_1, y_1, g(x_1, y_1))$, $(x_2, y_2, g(x_2, y_2))$ e $(x_3, y_3, g(x_3, y_3))$, rispettivamente. I coefficienti a_i e b_i devono soddisfare

$$\begin{bmatrix} 1 & x_3 & y_3 \\ 1 & x_2 & y_2 \\ 1 & x_1 & y_1 \end{bmatrix} \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix} = \begin{bmatrix} f_3 & g_3 \\ f_2 & g_2 \\ f_1 & g_1 \end{bmatrix}$$

o, equivalentemente,

$$\begin{bmatrix} 1 & x_3 & y_3 \\ 0 & \Delta X^T \end{bmatrix} \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix} = \begin{bmatrix} f_3 & g_3 \\ \Delta F^T \end{bmatrix}$$

ove

$$\Delta X^T = \begin{bmatrix} x_2 - x_3 & y_2 - y_3 \\ x_1 - x_3 & y_1 - y_3 \end{bmatrix}, \quad \Delta F^T = \begin{bmatrix} f_2 - f_3 & g_2 - g_3 \\ f_1 - f_3 & g_1 - g_3 \end{bmatrix}$$

da cui

$$\begin{bmatrix} a_2 & b_2 \\ a_3 & b_3 \end{bmatrix} = \Delta X^{-T} \Delta F^T$$

e quindi

$$\begin{bmatrix} a_2 & a_3 \\ b_2 & b_3 \end{bmatrix} = \Delta F \Delta X^{-1}$$

Siccome

$$\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} f_3 \\ g_3 \end{bmatrix} - \begin{bmatrix} a_2 & a_3 \\ b_2 & b_3 \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}$$

il punto comune di intersezione dei due piani con il piano xOy soddisfa

$$\begin{bmatrix} a_2 & a_3 \\ b_2 & b_3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = - \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} = - \begin{bmatrix} f_3 \\ g_3 \end{bmatrix} + \begin{bmatrix} a_2 & a_3 \\ b_2 & b_3 \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}$$

da cui

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} - \Delta X \Delta F^{-1} \begin{bmatrix} f_3 \\ g_3 \end{bmatrix} = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} - \begin{bmatrix} x_2 - x_3 & x_1 - x_3 \\ y_1 - y_3 & y_1 - y_3 \end{bmatrix} \begin{bmatrix} f_2 - f_3 & f_1 - f_3 \\ g_1 - g_3 & g_1 - g_3 \end{bmatrix}^{-1} \begin{bmatrix} f_3 \\ g_3 \end{bmatrix}$$

Si vede che

1. il metodo coincide con il classico metodo delle secanti nel caso unidimensionale
2. il metodo è un'approssimazione del metodo di Newton

Per ulteriori informazioni, si veda [4].

3.2 Metodo di Newton

Per trovare lo zero di $F\colon \mathbb{R}^m \to \mathbb{R}^m$ si può procedere con lo sviluppo in serie di Taylor

$$F(x) = F(x_n) + J_F(x_n)(x - x_n) + \mathcal{O}(\|x - x_n\|^2)$$

e definire x_{n+1} come la soluzione di

$$F_n(x) = F(x_n) + J_F(x_n)(x - x_n) = 0$$

da cui

$$x_{n+1} = x_n - J_F(x_n)^{-1} F(x_n)$$

Ovviamente non si inverte la matrice jacobiana $J_F(x_n)$, ma si procede iterativamente in questo modo (dopo aver scelto un guess iniziale x_0)

$$\begin{aligned} J_F(x_n)\delta_n &= -F(x_n) \\ x_{n+1} &= x_n + \delta_n \end{aligned}$$

Solitamente si usa $\|\delta_n\|$ come stima per l'errore $\|x_n - \bar{x}\|$, ove $F(\bar{x}) = 0$. Poiché però il calcolo di δ_n è la parte costosa dell'algoritmo (risoluzione di un sistema lineare), spesso si calcola anche x_{n+1}.

3.2.1 Metodi di Newton modificati

Ogni iterazione del metodo di Newton richiede la soluzione di un sistema lineare. Tale operazione può risultare costosa. La valutazione stessa della matrice jacobiana può essere poco praticabile. Ogni volta che invece della matrice jacobiana esatta $J_F(x_n)$ si usa qualcos'altro, si parla di metodo di Newton modificato. Ci sono molte possibilità:

- calcolare solo la matrice jacobiana $J_F(x_0)$ ed usarla per tutte le iterazioni successive, fattorizzandolo dunque in L, U e P una volta e per tutte;
- calcolare la matrice jacobiana ogni k iterazioni ed usarla, in forma fattorizzata, per quel numero di iterazioni;
- approssimare con differenze finite o altre formule la matrice jacobiana (in questi tre casi si parla di metodi di Newton *inesatti*);
- calcolare una approssimazione della matrice jacobiana (come nel caso del metodo delle secanti: in questo caso si parla di metodi *quasi*-Newton).

Metodo di Broyden

Il metodo di Broyden è un metodo quasi-Newton che generalizza il metodo delle secanti in una dimensione. Data l'iterazione x_n, il punto successivo è lo zero del modello lineare

$$F_n(x) = F(x_n) + B_n(x - x_n).$$

La matrice B_n è determinata in modo che

$$F_n(x_{n-1}) = F(x_{n-1})$$

da cui si ricava

$$B_n(x_n - x_{n-1}) = F(x_n) - F(x_{n-1}) \tag{3.1}$$

Nel caso in cui $m = 1$, si vede che B_n è univocamente determinato ed il metodo risulta essere quello delle secanti. Altrimenti, si tratta di trovare una matrice che realizzi l'uguaglianza. È possibile fissare arbitrariamente le prime $m - 1$ colonne di B_n: pertanto serve una strategia per il calcolo di B_n. Una possibile soluzione è quella che calcola B_n a partire da B_{n-1} con un basso costo computazionale. Si può dunque chiedere la quantità $F_n(x) - F_{n-1}(x)$ sia piccola. Si trova facilmente (usando (3.1)) che

$$F_n(x) - F_{n-1}(x) = (B_n - B_{n-1})(x - x_{n-1})$$

Per ogni $x \in \mathbb{R}^m$, si può considerare l'iperpiano passante per i vettori $x - x_{n-1}$ e $x_n - x_{n-1}$ e considerare su di esso un vettore $t \in \mathbb{R}^m$ ortogonale a $x_n - x_{n-1}$. Sarà dunque possibile scegliere α e β tali per cui $x - x_{n-1} = \alpha(x_n - x_{n-1}) + \beta t$. Perciò, la quantità da minimizzare, al variare di B_n, diventa

$$(B_n - B_{n-1})(x - x_{n-1}) = \alpha(B_n - B_{n-1})(x_n - x_{n-1}) + \beta(B_n - B_{n-1})t$$

Non si può agire sul primo membro, poiché la condizione (3.1) dice già quanto deve fare $(B_n - B_{n-1})(x_n - x_{n-1})$. Si può però chiedere che il secondo membro sia nullo per ogni t ortogonale a $x_n - x_{n-1}$. Risulta che $B_n - B_{n-1}$ deve avere la forma $u(x_n - x_{n-1})^T$ (cioè B_n è un update di B_{n-1} di rango 1), con $u \in \mathbb{R}^m$ e siccome

$$(B_n - B_{n-1})(x_n - x_{n-1}) = F(x_n) - F(x_{n-1}) - B_{n-1}(x_n - x_{n-1}) = u\|x_n - x_{n-1}\|_2^2$$

si ricava

$$u = \frac{F(x_n) - F(x_{n-1}) - B_{n-1}(x_n - x_{n-1})}{\|x_n - x_{n-1}\|_2^2}$$

da cui, finalmente,

$$B_n = B_{n-1} + \frac{(F(x_n) - F(x_{n-1}) - B_{n-1}(x_n - x_{n-1}))(x_n - x_{n-1})^T}{\|x_n - x_{n-1}\|_2^2}$$

Come matrice B_0 si può considerare la matrice identità. Per ulteriori informazioni si veda [3].

3.2.2 Approssimazione di matrici jacobiane

La matrice jacobiana di una funzione $F\colon \mathbb{R}^m \to \mathbb{R}^m$ soddisfa

$$J_F(x)y = \lim_{\varepsilon \to 0} \frac{F(x + \varepsilon y) - F(x)}{\varepsilon}, \quad \forall\ y \in \mathbb{R}^m$$

L'espressione da mandare al limite può essere usata sia come controllo del corretto calcolo della matrice jacobiana che come approssimazione della stessa (scegliendo come y i vettori della base canonica). Ovviamente la scelta di ε è cruciale: troppo grande porta ad una cattiva approssimazione del limite, troppo piccolo ad errori di cancellazione numerica. Si possono usare anche formule più accurate, come

$$J_F(x)y = \lim_{\varepsilon \to 0} \frac{F(x + \varepsilon y) - F(x - \varepsilon y)}{2\varepsilon}, \quad \forall\ y \in \mathbb{R}^m \tag{3.2}$$

Esiste una formula che usa i numeri complessi ed approssima la matrice jacobiana di funzioni F analitche senza introdurre cancellazione numerica. È definita dalla formula

$$J_F(x)y = \lim_{\varepsilon \to 0} \frac{\Im(F(x + \mathrm{i}\varepsilon y))}{\varepsilon}$$

Si ottiene considerando lo sviluppo in serie di Taylor (per semplicità qui riportato per una funzione da $\mathbb{R}$ in $\mathbb{R}$)

$$f(x + \mathrm{i}\varepsilon y) = f(x) + \mathrm{i}\varepsilon f'(x)y - \frac{\varepsilon^2}{2} f''(x)y - \mathrm{i}\frac{\varepsilon^3}{6} f'''(x)y + \dots$$

Si veda [5].

3.3 Esercizi

1. Si generalizzi il metodo delle secanti di sezione 3.1 al caso m-dimensionale.

2. Si implementi il metodo di Newton inesatto che usa la stessa matrice jacobiana per k iterazioni, k parametro di input.

3. Si confrontino le velocità di convergenza (in un grafico semilogaritmico, numero di iterazioni in ascissa e stima dell'errore in ordinata) del metodo delle secanti, di Newton e di Broyden per un esempio visto.

4. L'inversa di una matrice che è somma di una matrice data A e di una matrice di rango 1 uv^T esiste la *formula di Sherman–Morrison*

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \tag{3.3}$$

   (se $1 + v^TA^{-1}u \neq 0$). Si dimostri la formula e si scriva una function che risolve il sistema lineare $(A + uv^T)x = b$ usando la fattorizzazione LU di A (senza calcolare A^{-1}).

5. Si dimostri che la formula dentro il limite di (3.2) è un'approssimazione di $J_F(x)$ a meno di $\mathcal{O}(\varepsilon^2)$.

6. Si consideri il problema di calcolare le radici del polinomio $x^2 + x - 1$. Esse soddisfano

$$\begin{cases} r_1 + r_2 = -1 \\ r_1 r_2 = -1 \end{cases}$$

Si analizzi il comportamento dei due metodi di punto fisso

$$\begin{cases} r_1^{n+1} = -1 - r_2^n \\ r_2^{n+1} = -\dfrac{1}{r_1^n} \end{cases}$$

e

$$\begin{cases} r_2^{n+1} = -1 - r_1^n \\ r_1^{n+1} = -\dfrac{1}{r_2^n} \end{cases}$$

a partire da $r_1^0 = -0.1$ e $r_2^0 = 0.1$.

Capitolo 4

Autovalori

4.1 Matrici e autovalori

Una matrice $A \in \mathbb{C}^{n\times n}$ è detta *normale* se $AA^* = A^*A$, ove A^* denota la trasposta coniugata. Esempi di matrici normali sono le hermitiane (simmetriche se reali) $A^* = A$, le antihermitiane (antisimmetriche se reali) $A^* = -A$ e le unitarie (ortogonali se reali) $A^* = A^{-1}$. Un esempio di matrice normale non hermitiana, non antihermitiana, non unitaria è

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Vale il seguente

Teorema 1 (Teorema spettrale)**.** *Una matrice A è normale se e solo se si può decomporre come*

$$A = U\Lambda U^*$$

ove U è una matrice unitaria e Λ la matrice diagonale degli autovalori di A.

In particolare, gli autovettori di una matrice normale (le colonne di U) relativi ad autovalori distinti sono tra loro ortogonali e dunque formano una base per $\mathbb{C}^n$. Le matrici unitarie U hanno norma euclidea pari ad 1. Infatti

$$\|U\|_2 = \sup_{x\neq 0} \frac{\|Ux\|_2}{\|x\|_2} = \sup_{x\neq 0} \frac{\sqrt{x^*U^*Ux}}{\|x\|_2} = \sup_{x\neq 0} \frac{\sqrt{x^*x}}{\|x\|_2} = 1$$

È possibile in generale calcolare le matrici U e Λ con il comando `[U, Lambda] = eig (A)`.

Teorema 2 (Bauer–Fike)**.** *Sia A una matrice diagonalizzabile, cioè $A = U\Lambda U^{-1}$, con Λ matrice diagonale degli autovalori. Sia μ un autovalore di $A + E$. Allora*

$$\min_{\lambda \in \sigma(A)} |\lambda - \mu| \leq \mathrm{cond}_2(U)\|E\|_2$$

ove

$$\mathrm{cond}_2(U) = \|U\|_2\|U^{-1}\|_2$$

Siccome le matrici normali sono diagonalizzabili mediante una matrice unitaria, ne segue che il problema degli autovalori è stabile per esse, poiché $\mathrm{cond}_2(U) = 1$. Il teorema di Bauer–Fike è un esempio di *bacward error analysis*.

4.2 Metodo QR e QR con shift

L'iterazione generale del metodo QR per la ricerca degli autovalori è

$$A_k = Q_k R_k, \quad A_{k+1} = R_k Q_k$$

con $A_0 = A$. Si ha

$$A_{k+1} = R_k Q_k = (Q_k^* Q_k) R_k Q_k = Q_k^* A_k Q_k$$

e dunque A_{k+1} è ortogonalmente simile a A_k. Solitamente, si riduce prima A in forma di Hessenberg superiore (o tridiagonale) mediante riflessioni di Housholder (vedi esercizio 3 del capitolo 2). In tal modo, le matrici A_k preservano la forma. Sotto determinate ipotesi e se gli autovalori di A soddisfano

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n| > 0$$

allora la successione $\{A_k\}_k$ tende ad una forma triangolare superiore $T = (t_{ij}$, con $t_{ii} = \lambda_i$. Se gli autovalori non sono separati in modulo, la successione tende ad uan forma triangolare per blocchi.

Nel metodo con shift, al fine di accelerare la convergenza verso un autovalore λ, si definisce

$$A_k - \mu I = Q_k R_k, \quad A_{k+1} = R_k Q_k + \mu I$$

ove μ è una stima di λ. Infatti, la velocità di convergenza del metodo QR dipende da

$$\max_{1 \leq i \leq n-1} \left| \frac{\lambda_{i+1}}{\lambda_i} \right|$$

e partendo dalla matrice $A - \mu I$ si spera di ottenere rapporti

$$\left| \frac{\lambda_{i+1} - \mu}{\lambda_i - \mu} \right|$$

più favorevoli. In generale conviene scegliere $\mu \approx \lambda_n$, altrimenti il vale assoluto potrebbe avere effetto contrario sui rapporti. Siccome in generale è difficile avere una stima degli autovalori, si può scegliere il valore di μ in maniera adattativa

$$\mu_k = (A_k)_{n,n}$$

4.3 Esercizi

1. Si verifichi il teorema di Bauer–Fike usando matrici simmetriche, antisimmetriche, ortogonali e normali in generale.

2. Si considerino le matrici A_1

$$A_1 = X\Lambda X^{-1}, \quad \Lambda = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 2 \end{bmatrix},$$

$$A_2 = XJX^{-1}, \quad J = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   e se ne calcolino gli autovalori con il comando `eig`. Si ripeta l'esercizio con la matrice

$$X = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

   Quali considerazioni si possono fare?

3. La scelta adattativa dello shift $\mu_k = (A_k)_{n,n}$ può fallire in caso di matrici simmetriche. Si consideri il *Wilkinson shift*

$$\mu_k = (A_k)_{n,n} - \frac{\operatorname{sign}(\delta)(A_k)_{n-1,n}^2}{|\delta| + \sqrt{\delta^2 + (A_k)_{n-1,n}^2}}, \quad \delta = \frac{(A_k)_{n-1,n-1} - (A_k)_{n,n}}{2}$$

   (se $\delta = 0$, si sceglie $\operatorname{sign}(\delta) = 1$). Si dimostri che è l'autovalore di

$$\begin{bmatrix} (A_k)_{n-1,n-1} & (A_k)_{n-1,n} \\ (A_k)_{n-1,n} & (A_k)_{n,n} \end{bmatrix}$$

più vicino a $(A_k)_{n,n}$. Si testi il metodo QR con shift adattativo e Wilkinson shift sulla matrice

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

[Sugg.: gli autovalori sono dati da $(A_k)_{n,n} + \delta \pm \sqrt{\delta^2 + ((A_k)_{n-1,n})^2}$ *e dunque il più vicino a* $(A_k)_{n,n}$ *è quello che minimizza in modulo* $\delta \pm \sqrt{\cdots}$*. Poi occorre razionalizzare per stabilizzare. . .]*

Capitolo 5

Approssimazione

5.1 Algoritmo di Thomas per sistemi tridiagonali

Per un sistema triadiagonale di matrice A

$$\begin{bmatrix} a_1 & c_1 & 0 & \dots & 0 \\ b_2 & a_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & \dots & 0 & b_n & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

l'algoritmo dell'eliminazione di Gauss (senza pivoting) prende la forma di *algoritmo di Thomas*. Si ha

$$A = \begin{bmatrix} 1 & 0 & \dots & 0 \\ \beta_2 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \beta_n & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 & c_1 & \dots & 0 \\ 0 & \alpha_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & c_{n-1} \\ 0 & \dots & 0 & \alpha_n \end{bmatrix}$$

con

$$\alpha_1 = a_1$$
$$\beta_k = \frac{b_k}{\alpha_{k-1}}, \quad \alpha_k = a_k - \beta_k c_{k-1}, \quad k = 2, 3, \dots, n$$

da cui

$$y_1 = d_1, \quad y_k = d_k - \beta_k y_{k-1} \quad k = 2, 3, \dots, n$$
$$x_n = \frac{y_n}{\alpha_n}, \quad x_k = \frac{y_k - c_k x_{k+1}}{\alpha_k}, \quad k = n-1, n-2, \dots, 1$$

Tale algoritmo (di costo $\mathcal{O}(n)$) si può usare quando l'eliminazione di Gauss non richiede pivoting (per esempio per matrici a dominanza diagonale stretta o simmetriche e definite positive).

5.2 Interpolazione polinomiale a tratti

Data una funzione $f\colon [a,b] \to \mathbb{R}$ e un'insieme $\{x_i\}_{i=1}^n \subset [a,b]$ di nodi ordinati ($x_{i-1} < x_i$), consideriamo l'interpolante polinomiale a tratti $L_{k-1}^c f$ di grado $k-1$. Su ogni intervallo $[x_i, x_{i+1}]$ di lunghezza $h_i = x_{i+1} - x_i$ essa coincide con il polinomio di grado $k-1$

$$a_{i,1}(x-x_i)^{k-1} + a_{i,2}(x-x_i)^{k-2} + \ldots + a_{i,k-1}(x-x_i) + a_{i,k}. \tag{5.1}$$

Dunque, l'interpolante polinomiale a tratti è completamente nota una volta noti i nodi e i coefficienti di ogni polinomio.

5.2.1 Strutture in MATLAB: `pp`

In MATLAB è possibile definire delle *strutture*, cioè degli insiemi (non ordinati) di oggetti. Per esempio, le istruzioni

```
S.a = 1;
S.b = [1,2];
```

generano la struttura `S`

```
S =
{
  a =  1
  b =

     1   2

}
```

L'interpolazione polinomiale a tratti è definita mediante una struttura solitamente chiamata `pp` *(piecewise polynomial)*, che contiene gli oggetti `pp.form` (stringa `pp`), `pp.breaks` (vettore riga dei nodi di interpolazione), `pp.coefs` (matrice dei coefficienti), `pp.pieces` (numero di intervalli, cioè numero di nodi meno uno), `pp.order` (grado polinomiale più uno) e `pp.dim` (numero di valori assunti dai polinomi). La matrice A dei coefficienti ha dimensione $n \times k$. Nota una struttura `pp`, è possibile valutare il valore dell'interpolante in un generico target $\bar{x}$ con il comando `ppval (pp, xbar)`.

È possibile definire una struttura per l'interpolazione polinomiale a tratti attraverso il comando `mkpp (breaks, coefs)`.

5.2.2 Splines cubiche

Le splines cubiche sono implementate da MATLAB con il comando `spline` che accetta in input il vettore dei nodi e il vettore dei valori e restituisce la struttura associata. La spline cubica costruita è nota come *not-a-knot*, ossia viene imposta la continuità della derivata terza (generalmente discontinua) nei nodi x_2 e x_{n-1}. Lo stesso comando permette di generare anche le splines *vincolate*: è sufficiente che il vettore dei valori abbia due elementi in più rispetto al vettore dei nodi. Il primo e l'ultimo valore verranno usati per imporre il valore della derivata alle estremità dell'intervallo. Se si usa un ulteriore vettore di input `xbar`, il comando restituisce il valore dell'interpolante sui nodi target `xbar`. Dunque, il comando

```
spline (x, y, xbar)
```

è equivalente ai comandi

```
pp = spline (x, y); ppval (pp, xbar)
```

Implementazione di splines cubiche in MATLAB

Con le notazioni usate fino ad ora, si può costruire una spline cubica S a partire dalla sua derivata seconda nell'intervallo generico $[x_i, x_{i+1}]$

$$S''_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{h_i}(x - x_i) + m_i, \quad i = 1, \ldots, n-1 \tag{5.2}$$

ove $m_i = S''(x_i)$ sono incogniti. Integrando due volte la (5.2), si ottiene

$$S'_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{2h_i}(x - x_i)^2 + m_i(x - x_i) + a_i \tag{5.3}$$

$$S_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{6h_i}(x - x_i)^3 + \frac{m_i}{2}(x - x_i)^2 + a_i(x - x_i) + b_i \tag{5.4}$$

ove le costanti a_i e b_i sono da determinare. Innanzitutto, richiedendo la proprietà di interpolazione, cioè $S_{[x_i,x_{i+1}]}(x_j) = y_j$, $j = i, i+1$, si ottiene

$$\begin{aligned} b_i &= y_i, \\ a_i &= \frac{y_{i+1} - y_i}{h_i} - (m_{i+1} - m_i)\frac{h_i}{6} - m_i\frac{h_i}{2} = \\ &= \frac{y_{i+1} - y_i}{h_i} - m_{i+1}\frac{h_i}{6} - m_i\frac{h_i}{3} \end{aligned}$$

A questo punto, richiedendo la continuità della derivata prima nel nodo x_i, cioè $S'_{[x_{i-1},x_i]}(x_i) = S'_{[x_i,x_{i+1}]}(x_i)$ per $i = 2, \ldots, n-1$, si ottiene

$$\frac{h_{i-1}}{6}m_{i-1} + \frac{h_{i-1}+h_i}{3}m_i + \frac{h_i}{6}m_{i+1} = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \,. \tag{5.5}$$

Risulta chiaro che ci sono $n-2$ equazioni e n incognite m_i.

Splines cubiche *naturali* Si impone che il valore della derivata seconda agli estremi dell'intervallo sia 0. Da (5.2), si ricava dunque $m_1 = m_n = 0$. Il sistema lineare (5.5) diventa allora

$$\begin{bmatrix} 1 & 0 & \ldots & \ldots & \ldots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \ldots & \ldots & \ldots & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

con $d_1 = d_n = 0$ e $d_i = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}$, $i = 2, \ldots, n-1$.

Ovviamente il sistema protrebbe essere semplificato e ridotto a dimensione $n-2$. In tal modo risulta tridiagonale, simmetrico e definito positivo e a dominanza diagonale stretta. Si può usare l'algoritmo di Thomas per risolverlo.

Splines cubiche *vincolate* Si impongono due valori y'_1 e y'_n per la derivata $S'(x_1)$ e $S'(x_n)$, rispettivamente. Da (5.3) si ricava dunque

$$\begin{aligned} a_1 &= y'_1 \\ \frac{m_n - m_{n-1}}{2h_{n-1}}(x_n - x_{n-1})^2 + m_{n-1}(x_n - x_{n-1}) + a_{n-1} &= y'_n \end{aligned}$$

da cui

$$\begin{aligned} \frac{h_1}{3}m_1 + \frac{h_1}{6}m_2 &= \frac{y_2 - y_1}{h_1} - y'_1 \\ \frac{h_{n-1}}{6}m_{n-1} + \frac{h_{n-1}}{3}m_n &= y'_n - \frac{y_n - y_{n-1}}{h_{n-1}} \end{aligned}$$

Il sistema lineare da risolvere diventa dunque

$$\begin{bmatrix} \frac{h_1}{3} & \frac{h_1}{6} & 0 & \dots & \dots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \dots & \dots & 0 & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

con $d_1 = \frac{y_2-y_1}{h_1} - y'_1$ e $d_n = y'_n - \frac{y_n-y_{n-1}}{h_{n-1}}$.

Anche in questo caso il sistema è tridiagonale, simmetrico e definito positivo, a dominanza diagonale stretta.

Splines cubiche *periodiche* Si impone la periodicità della derivata prima e seconda, cioè $S'(x_1) = S'(x_n)$ e $S''(x_1) = S''(x_n)$. Da (5.2) e (5.3) si ricava dunque

$$\begin{aligned} m_1 &= m_n \\ a_1 &= \frac{m_n - m_{n-1}}{2} h_{n-1} + m_{n-1} h_{n-1} + a_{n-1} \end{aligned}$$

da cui

$$\begin{aligned} m_1 - m_n &= 0 \\ \frac{h_1}{3} m_1 + \frac{h_1}{6} m_2 + \frac{h_{n-1}}{6} m_{n-1} + \frac{h_{n-1}}{3} m_n &= \frac{y_2 - y_1}{h_1} + \\ &\quad - \frac{y_n - y_{n-1}}{h_{n-1}} \end{aligned}$$

Il sistema lineare da risolvere diventa dunque

$$\begin{bmatrix} 1 & 0 & \dots & \dots & \dots & 0 & -1 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & \dots & 0 \\ 0 & \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ \frac{h_1}{3} & \frac{h_1}{6} & 0 & \dots & 0 & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

con $d_1 = 0$ e $d_n = \frac{y_2-y_1}{h_1} - \frac{y_n-y_{n-1}}{h_{n-1}}$.

In questo caso non è possibile ridurre il sistema a tridiagonale. Se però si elimina m_1 si ottiene

$$\begin{bmatrix} \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & 0 & \frac{h_1}{6} \\ \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ \frac{h_1}{6} & 0 & \dots & 0 & \frac{h_{n-1}}{6} & \frac{h_1}{3}+\frac{h_{n-1}}{3} \end{bmatrix} \begin{bmatrix} m_2 \\ m_3 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

Questo sistema è *tridiagonale ciclico*, simmetrico e definito positivo, a dominanza diagonale stretta e può comunque essere risolto in $\mathcal{O}(n)$ operazioni.

Splines cubiche *not-a-knot* Si impone la continuità della derivata terza in x_2 e x_{n-1}. Derivando (5.2) si ricava dunque

$$\frac{m_2 - m_1}{h_1} = \frac{m_3 - m_2}{h_2}$$
$$\frac{m_{n-1} - m_{n-2}}{h_{n-2}} = \frac{m_n - m_{n-1}}{h_{n-1}}$$

da cui

$$\frac{1}{h_1}m_1 - \left(\frac{1}{h_1} + \frac{1}{h_2}\right) m_2 + \frac{1}{h_2}m_3 = 0$$
$$\frac{1}{h_{n-2}}m_{n-2} - \left(\frac{1}{h_{n-2}} + \frac{1}{h_{n-1}}\right) m_{n-1} + \frac{1}{h_{n-1}}m_n = 0$$

Il sistema lineare da risolvere diventa dunque

$$\begin{bmatrix} \frac{1}{h_1} & -\frac{1}{h_1}-\frac{1}{h_2} & \frac{1}{h_2} & 0 & \dots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \dots & 0 & \frac{1}{h_{n-2}} & -\frac{1}{h_{n-2}}-\frac{1}{h_{n-1}} & \frac{1}{h_{n-1}} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

con $d_1 = d_n = 0$.

Consideriamo le prime due righe

$$\frac{1}{h_1}m_1 - \left(\frac{1}{h_1} + \frac{1}{h_2}\right) m_2 + \frac{1}{h_2}m_3 = 0$$
$$\frac{h_1}{6}m_1 + \frac{h_1 + h_2}{3}m_2 + \frac{h_2}{6}m_3 = d_2$$

ricaviamo m_1 dalla prima e sostituiamo nella seconda

$$\left(\frac{h_1}{6}+\frac{h_1^2}{6h_2}+\frac{h_1+h_2}{3}\right)m_2+\left(-\frac{h_1^2}{6h_2}+\frac{h_2}{6}\right)m_3=d_2$$

e moltiplichiamo tutto per $h_2/(h_1+h_2)$ ottenendo

$$\frac{h_1+2h_2}{6}m_2+\frac{h_2-h_1}{6}m_3=\frac{h_2}{h_1+h_2}d_2.$$

Procedendo analogamente per le ultime due righe si ottiene

$$\frac{h_{n-2}-h_{n-1}}{6}m_{n-2}+\frac{2h_{n-2}+h_{n-1}}{6}m_{n-1}=\frac{h_{n-2}}{h_{n-2}+h_{n-1}}d_{n-1}.$$

Pertanto il sistema può essere riscritto (ridotto)

$$\begin{bmatrix} \frac{h_1+2h_2}{6} & \frac{h_2-h_1}{6} & 0 & \dots & \dots & 0 \\ \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{h_{n-3}}{6} & \frac{h_{n-3}+h_{n-2}}{3} & \frac{h_{n-2}}{6} \\ 0 & \dots & \dots & 0 & \frac{h_{n-2}-h_{n-1}}{6} & \frac{2h_{n-2}+h_{n-1}}{6} \end{bmatrix} \begin{bmatrix} m_2 \\ m_3 \\ \vdots \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \begin{bmatrix} \frac{h_2}{h_1+h_2}d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_{n-2} \\ \frac{h_{n-2}}{h_{n-2}+h_{n-1}}d_{n-1} \end{bmatrix}$$

Dunque è un sistema a dominanza diagonale stretta. Infine, scalando la prima e l'ultima riga, può essere reso simmetrico. I valori m_1 ed m_n possono essere ricavati dalle condizioni sulla derivata terza.

5.3 Curve di Bézier

Definiamo il polinomio di Bernstein di grado n come

$$B_i^n(t)=\binom{n}{i-1}t^{i-1}(1-t)^{n-i+1},\quad i=1,2,\dots,n+1$$

ove $t\in[0,1]$. Da notare che $B_1^0\equiv 1$. Si può mostrare che

$$B_i^n(t)=(1-t)B_i^{n-1}(t)+tB_{i-1}^{n-1}(t),\quad n>0,\quad i=1,2,\dots,n+1 \tag{5.6}$$

dove $B_0^n\equiv 0$ per $n>0$ e$B_i^n\equiv 0$ per $i>n+1$. Una curva di *Bézier* di grado n è

$$B\colon[0,1]\to\mathbb{R}^d,\quad t\mapsto\sum_{i=1}^{n+1}b_iB_i^n(t)$$

ove i $b_i\in\mathbb{R}^d$ sono chiamati *punti di controllo*. Dalla definizione, si vede che $B_1^n(0)=1$ e $B_i^n(0)=0$ per $i>1$. Inoltre, $B_i^n(1)=0$ per $i<n+1$ e $B_{n+1}^n(1)=1$. Pertanto, una curva di Bézier passa sempre per b_1 e b_{n+1}.

5.3.1 Valutazione di una curva di Bézier

Data una curva di Bézier di grado n e punti di controllo $\{b_i\}_{i=1}^{n+1}$, definiamo $b_i^0(t) \equiv b_i$ e

$$b_i^r(t) = (1-t)b_i^{r-1}(t) + tb_{i+1}^{r-1}(t), \quad r = 1, 2, \ldots, n, \quad i = 1, 2, \ldots, n-r+1.$$

Questo procedimento prende il nome di *algoritmo di De Casteljau*. Si ha

$$B(t) \equiv b_1^n(t) = \sum_{i=1}^{n+1} b_i B_i^n(t). \tag{5.7}$$

5.3.2 Invarianza affine

Se una mappa affine

$$\Phi(x) = Ax + c$$

dove $x, c \in \mathbb{R}^d$, $A \in \mathbb{R}^{d\times d}$ è applicata ad una curva di Bézier, allora il risultato è equivalente alla curva di Bézier dell'immagine affine dei punti di controllo, cioè

$$\Phi(B(t)) = \sum_{i=1}^{n+1} \Phi(b_i) B_i^n(t) \tag{5.8}$$

5.4 Esercizi

1. Si implementino le function `spline_naturale.m`, `spline_vincolata.m` e `spline_periodica.m`, riducendo i sistemi lineari da risolvere a tridiagonali o tridiagonali ciclici.

2. Si modifichi `spline_notaknot.m` in modo che riproduca il comportamento di MATLAB nel caso di 1, 2 o 3 nodi di interpolazione.

3. A partire da una struttura `pp` che identifica una spline cubica, si costruisca la struttura che identifica la sua derivata prima e seconda. Si verifichi che esse risultano continue.

4. Si implementi una function `intlintrat` che implementa l'interpolante lineare a tratti mediante l'uso delle strutture.

5. Si consideri la matrice tridiagonale ciclica

$$A = \begin{bmatrix} a_1 & c_1 & 0 & \dots & 0 & b_1 \\ b_2 & a_2 & c_2 & \ddots & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & \ddots & b_{n-1} & a_{n-1} & c_{n-1} \\ c_n & 0 & \dots & 0 & b_n & a_n \end{bmatrix}$$

ed i vettori

$$u = \begin{bmatrix} b_1 & 0 & 0 & \dots & 0 & c_n \end{bmatrix}^T$$
$$v = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}^T$$

Allora A può essere scritta come $B + uv^T$, con B tridiagonale. Allora il sistema lineare $Ax = b$ può essere risolto applicando la formula di Shermann–Morrison (3.3) e l'algoritmo di Thomas in sezione 5.1. Si implementi una function che implementi quanto descritto.

6. Dati i punti di controllo $b_1 = (0, 0)$, $b_2 = (0, 1)$, $b_3 = (1, 1)$, $b_4 = (1, 0)$, $b_5 = (1/2, 0)$ e $b_6 = (1/2, 1/2)$, si verifichi l'equivalenza (5.8) dopo aver applicato una rototraslazione scalata alla corrispondente curva di Bézier.

7. Si può passare facilmente da una curva di Bézier di grado n ad una di grado $n + 1$ che disegnino la stessa curva. Data la curva di Bézier associata ai punti di controllo $\{b_i\}_{i=1}^{n+1}$, la stessa curva è una curva di Bézier di grado $n + 1$ associata ai punti di controllo

$$\begin{aligned} \hat{b}_1 &= b_1 \\ \hat{b}_i &= \frac{i-1}{n+1} b_{i-1} + \frac{n-i+2}{n+1} b_i, \quad i = 2, 3, \dots, n+1 \\ \hat{b}_{n+2} &= b_{n+1} \end{aligned}$$

Si costruiscano le due curve di Bézier e si verifichi con un esempio che sono la stessa curva.

8. Siano $b_i^r(t)$ le curve associate all'algoritmo di De Casteljau, con $b_i^0(t) \equiv b_i$. Si prenda $t_0 \in (0, 1)$ e si definiscano i punti di controllo

$$\begin{aligned} c_i &= b_1^{i-1}(t_0), \qquad i = 1, 2, \dots, n+1 \\ d_i &= b_i^{n+1-i}(t_0), \quad i = 1, 2, \dots, n+1 \end{aligned}$$

Si mostri con un esempio che la curva di Bézier

$$\sum_{i=1}^{n+1} b_i B_i^n(t)$$

è equivalente all'unione delle due curve di Bézier disgiunte

$$\sum_{i=1}^{n+1} c_i B_i^n(t) \quad \text{e} \quad \sum_{i=1}^{n+1} d_i B_i^n(t).$$

Più precisamente

$$\sum_{i=1}^{n+1} b_i B_i^n(t) = \begin{cases} \displaystyle\sum_{i=1}^{n+1} c_i B_i^n\left(\frac{t}{t_0}\right), & \text{se } t \leq t_0 \\ \displaystyle\sum_{i=1}^{n+1} d_i B_i^n\left(\frac{t-t_0}{1-t_0}\right), & \text{se } t > t_0 \end{cases}$$

Capitolo 6

Quadratura gaussiana

6.1 Gauss–Legendre

Le formule di quadratura di Gauss–Legendre si usano per approssimare integrali del tipo

$$\int_{-1}^{1} f(x)\mathrm{d}x$$

I polinomi di Legendre, ortogonali nell'intervallo $[-1, 1]$ rispetto alla funzione peso $w(x) \equiv 1$ soddisfano la relazione di ricorrenza

$$\begin{aligned} P_0(x) &= 1, \quad P_1(x) = x \\ (n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x), \quad n \geq 1. \end{aligned}$$

Sono polinomi ortogonali, ma non *ortonormali*, in quanto

$$\int_{-1}^{1} P_i(x)P_j(x)\mathrm{d}x = \frac{2}{2n+1}\delta_{ij}.$$

I primi pesi e nodi di quadratura sono facili da calcolare: infatti

$$\begin{aligned} P_1(x_i) = 0 \Longleftrightarrow & x_0 = 0 \\ & w_0 = \int_{-1}^{1} 1 \mathrm{d}x = 2 \\ P_2(x_i) = \frac{3x^2 - 1}{2} = \Longleftrightarrow & x_0 = -\frac{1}{\sqrt{3}} \\ & x_1 = \frac{1}{\sqrt{3}} \\ & w_0 = \int_{-1}^{1} \frac{x - x_1}{x_0 - x_1} \mathrm{d}x = 1 \\ & w_1 = \int_{-1}^{1} \frac{x - x_0}{x_1 - x_0} \mathrm{d}x = 1 \end{aligned}$$

I valori sono noti analiticamente fino a cinque nodi di quadratura. Per alcuni ordini fino a 96 i valori approssimati si possono trovare su [1, pag. 916]. Routines[1] in MATLAB per il calcolo di nodi e pesi di ogni ordine sono associate a [2]. Per esempio, considerando i polinomi di Jacobi, ortogonali in $[-1, 1]$ rispetto alla funzione peso $w(x) = (1-x)^a(1+x)^b$, si possono calcolare nodi e pesi di quadratura Gauss–Legendre di ordine n con il comando

```
xw = gauss (n + 1, r_jacobi (n + 1, 0, 0));
x = xw(:, 1);
w = xw(:, 2);
```

Qualora l'integrale da approssimare sia definito nell'intervallo generico $[a, b]$, è sufficiente il cambio di variabile

$$\int_a^b f(y) \mathrm{d}y = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2} x + \frac{a+b}{2}\right) \mathrm{d}x$$

da cui

$$\int_a^b f(y) \mathrm{d}y \approx \sum_{i=0}^{n} \frac{b-a}{2} w_i f\left(\frac{b-a}{2} x_i + \frac{a+b}{2}\right) = \sum_{i=0}^{n} z_i f(y_i)$$

Per la quadratura di Gauss–Legendre esiste la formula di rappresentazione dell'errore

$$\int_a^b f(x) \mathrm{d}x = \sum_{i=0}^{n} w_i f(x_i) + \frac{(b-a)^{2n+3}((n+1)!)^4}{((2n+2)!)^3(2n+3)} f^{(2n+2)}(c), \quad c \in (a, b)$$

se $f \in \mathcal{C}^{2n+2}(a, b)$.

[1] Disponibili su https://www.cs.purdue.edu/archives/2002/wxg/codes/OPQ.html

6.1.1 Formula di Gauss–Legendre adattativa

Prendiamo $n = 1$ e indichiamo con $\mathrm{GL}(a, b)$ la corrispondente formula di Gauss–Legendre. Si ha

$$\begin{aligned}
\int_a^b f(x)\mathrm{d}x =& \mathrm{GL}(a,b) + \frac{(b-a)^5 2^4}{(4!)^3 5} f^{(4)}(c) \\
\int_a^b f(x)\mathrm{d}x =& \mathrm{GL}(a,(a+b)/2) + \frac{(b-a)^5 (2)^4}{2^5 (4!)^3 5} f^{(4)}(c_1) + \\
& \mathrm{GL}((a+b)/2,b) + \frac{(b-a)^5 2^4}{2^5 (4!)^3 5} f^{(4)}(c_2) = \\
& \mathrm{GL}(a,(a+b)/2) + \mathrm{GL}((a+b)/2,b) + \frac{(b-a)^5 2^4}{2^4 (4!)^3 5} f^{(4)}(\bar{c})
\end{aligned}$$

ove $c \in (a, b)$, $c_1 \in (a, (a + b)/2)$, $c_2 \in ((a + b)/2, b)$ e $\bar{c} \in (a, b)$ per il teorema del valor medio applicato alla funzione continua $f^{(4)}$. *Supponiamo* che $f^{(4)}(c) = f^{(4)}(\bar{c})$. Si ricava

$$\left(1 - \frac{1}{2^4}\right) \frac{(b-a)^5 2^4}{(4!)^3 5} f^{(4)}(c) = \mathrm{GL}(a,(a+b)/2) + \mathrm{GL}((a+b)/2,b) - \mathrm{GL}(a,b)$$

da cui

$$\begin{aligned}
\int_a^b f(x)\mathrm{d}x =& \mathrm{GL}(a,(a+b)/2) + \mathrm{GL}((a+b)/2,b) + \\
& \frac{1}{2^4 - 1} \left(\mathrm{GL}(a,(a+b)/2) + \mathrm{GL}((a+b)/2,b) - \mathrm{GL}(a,b)\right)
\end{aligned}$$

Pertanto, se

$$|\mathrm{GL}(a,(a+b)/2) + \mathrm{GL}((a+b)/2,b) - \mathrm{GL}(a,b)| \leq 15 \cdot \mathrm{tol} \tag{6.1}$$

allora

$$\left| \int_a^b f(x)\mathrm{d}x - (\mathrm{GL}(a,(a+b)/2) + \mathrm{GL}((a+b)/2,b)) \right| \leq \mathrm{tol}.$$

Se la disuguaglianza (6.1) non è soddisfatta, si può ripetere il ragionamento ricorsivamente in ogni sotto intervallo $[a, (a+b)/2]$ e $[(a+b)/2, b]$, chiedendo per ognuno una tolleranza tol/2.

6.2 Gauss–Chebyshev

Le formule di quadratura di Gauss–Chebyshev si usano per approssimare integrali del tipo

$$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} \mathrm{d}x.$$

I polinomi di Chebyshev (di prima specie) sono definiti esplicitamente da

$$T_n(x) = \cos(n \arccos x)$$

e soddisfano la relazione di ricorrenza

$$\begin{aligned} T_0(x) &= 1, \quad T_1(x) = x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x), \quad n \geq 1. \end{aligned}$$

I primi $n+1$ nodi di quadratura sono gli zeri del polinomio $T_{n+1}(x)$ e pertanto sono soluzione di

$$(n+1) \arccos x = \frac{\pi}{2} + i\pi$$

da cui

$$x_i = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right), \quad i = 0, 1, \ldots, n.$$

Per avere numericamente l'antisimmetria dei nodi ($x_i = -x_{n+1-i}$), conviene calcolarli come

$$x_i = \sin\left(\frac{(n-2i)\pi}{2(n+1)}\right), \quad i = 0, 1, \ldots, n.$$

I pesi risultano essere tutti uguali tra loro e valgono $w_i = \pi/(n+1)$. Pertanto

$$\int_{-1}^{1} \frac{f(x)\mathrm{d}x}{\sqrt{1-x^2}} \approx \frac{\pi}{n+1} \sum_{i=0}^{n} f\left(\sin\left(\frac{(n-2i)\pi}{2(n+1)}\right)\right).$$

6.3 Altre formule

È possibile calcolare nodi e pesi di quadratura gaussiana con le routines associate a [2]. La Tabella 6.1 ne riporta alcuni.

nome	(a, b)	$w(x)$	routine
Jacobi	$(-1, 1)$	$(1-x)^a(1+x)^b$	`r_jacobi.m`
Jacobi	$(0, 1)$	$(1-x)^a x^b$	`r_jacobi01.m`
Laguerre	$(0, +\infty)$	$x^a \exp(-x)$	`r_laguerre.m`
Hermite	$(-\infty, \infty)$	$\lvert x\rvert^{2\mu} \exp(-x^2)$	`r_hermite.m`

Tabella 6.1: Polinomi ortogonali in diversi intervalli e per diverse funzioni peso e routine per calcolare nodi e pesi di quadratura gaussiana associati.

6.4 Esercizi

1. Si costruisca la famiglia di polinomi $\hat{P}_i(x)$ *ortonormali* nell'intervallo $[-1, 1]$ rispetto alla funzione peso $w(x) = 1$ e si calcoli la loro formula di ricorrenza a tre termini.

2. Si verifichi il corretto comportamento della formula dell'errore

$$\frac{(b-a)^{2n+3}((n+1)!)^4}{((2n+2)!)^3(2n+3)} f^{(2n+2)}(c)$$

   per la quadratura di Gauss–Legendre di

$$\int_0^1 \mathrm{e}^x \sin(x)\mathrm{d}x$$

3. Si implementi una function `integralgcab (fun, n, a, b)` per l'approssimazione di

$$\int_a^b \frac{f(x)\mathrm{d}x}{\sqrt{(x-a)(b-x)}}$$

   mediante una formula di Gauss–Chebyshev.

4. Si implementi una function per la quadratura adattativa usando la formula di Gauss–Legendre con $n = 2$.

5. Si confrontino le formule di quadratura di Gauss–Legendre e Gauss–Chebyshev di ordine $n = 10$ per l'approssimazione di

$$\int_0^1 \frac{\mathrm{e}^x \mathrm{d}x}{\sqrt[4]{1-x^2}}.$$

Bibliografia

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions*, volume 55 of *Applied Mathematics Series*. NBS, 10th printing edition, 1972.

[2] W. Gautschi. *Orthogonal Polynomials: Computation and Approximation*. Numerical mathematics and scientific computation. Oxford science publications, 2004.

[3] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*, volume 16 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1995.

[4] M. Podisuk. Modified multivariate secant method. *Univ. Iagel. Acta Math.*, 31:265–271, 1994.

[5] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Rev.*, 40(1):110–112, 1998.