Università degli Studi di Cagliari
Corso di Calcolo Numerico 2

Algoritmo QR e zeri di una formula Gaussiana

Docente
Prof. Rodriguez

Tesina di
Gianluca Pisano
Anna Concas

Introduzione

Lo scopo di questa tesina è quello di analizzare il problema del calcolo dei pesi e dei nodi di formule Gaussiane mediante l'algoritmo QR, di cui si da una trattazione teorica nella prima parte di questo lavoro.
Tutti gli algoritmi sono stati implementati con il software commerciale MATLAB©.

Indice

Capitolo 1

Fattorizzazione QR

1.1 Matrici elementari di Householder

Ogni matrice A $m \times n$ può essere fattorizzata nella forma:

$$A = QR$$

dove Q è una matrice quadrata di ordine m, ortogonale cioè tale che $Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I$, e R matrice triangolare superiore avente le stesse dimensioni di A.
La fattorizzazione QR di una matrice può essere calcolata in vari modi, in questa tesina effettueremo tale fattorizzazione mediante le matrici elementari di Householder.
Una matrice elementare di Householder H è una matrice reale della forma

$$H = I - 2ww^{\mathrm{T}}$$

dove $w \in \mathbb{R}^n$ ha norma euclidea unitaria; la matrice $2ww^{\mathrm{T}}$ avente ordine n di rango 1.
Le matrici di Householder sono simmetriche (H=H^{T}), ortogonali ($H^{\mathrm{T}}H = HH^{\mathrm{T}} = I$) e hanno l'importante proprietà di annullare determinati elementi di una matrice o di un vettore.
Per costruire una matrice di Householder H si pone il problema, assegnato un vettore $\mathbf{x}$, di determinare $\mathbf{w}$ in modo che risulti

$$H\mathbf{x} = k\mathbf{e}_1 \tag{1.1}$$

in cui $k \in \mathbb{R}^n$ e $\mathbf{e}_1$ è il primo versore della base canonica di $\mathbb{R}^n$.
Vediamo come costruire la matrice H.
Considerando la (1.1), essendo H ortogonale, si ha che

$$\|H\mathbf{x}\| = \|\mathbf{x}\| = |k|$$

pertanto risulta

$$k = \pm\sigma, \text{ con } \sigma = \|\mathbf{x}\|.$$

Applicando la definizione di H alla (1.1) otteniamo

$$H\mathbf{x} = \mathbf{x} - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}\mathbf{x} = \mathbf{x} - 2(\mathbf{w}^{\mathrm{T}}\mathbf{x})\mathbf{w} = k\mathbf{e}_1 \ ,$$

da cui si ha

$$\mathbf{w} = \frac{\mathbf{x}-k\mathbf{e}_1}{2\mathbf{w}^{\mathrm{T}}\mathbf{x}} = \frac{\mathbf{x}-k\mathbf{e}_1}{\|\mathbf{x}-k\mathbf{e}_1\|}$$

essendo $\mathbf{w}$ di norma unitaria. Resta quindi determinato il vettore w tale che $H\mathbf{x} = k\mathbf{e}_1$; vediamo come poter ridurre il carico computazionale richiesto per la normalizzazione del suddetto vettore.
Prendendo ora in esame la relazione

$$\mathbf{x}^{\mathrm{T}}H\mathbf{x} = \mathbf{x} - 2(\mathbf{w}^{\mathrm{T}}\mathbf{x})^2 = kx_1$$

essendo $H\mathbf{x} = k\mathbf{e}_1$; dalla precedente otteniamo quindi

$$(\mathbf{w}^{\mathrm{T}}\mathbf{x})^2 = \frac{\sigma^2 - kx_1}{2}.$$

L'espressione trovata in precedenza per $\mathbf{w}$ implica che

$$\|\mathbf{x} - k\mathbf{e}_1\| = 2(\mathbf{w}^{\mathrm{T}}\mathbf{x})^2 = \sqrt{2(\sigma^2 - kx_1)}.$$

L'utilizzo di questa espressione permette di ridurre il carico computazionale richiesto per la normalizzazione del vettore $\mathbf{w}$. Per evitare eventuali rischi di cancellazione, è opportuno scegliere il segno di k in modo che $-kx_1$ sia positivo.
Definendo sign(y) pari a $+1$ o -1 a seconda che $y \geq 0$ oppure $y < 0$, si ha

$$k\text{= -sign}(x_1)\sigma, \ \|\mathbf{x} - k\mathbf{e}_1\| = \sqrt{2\sigma(\sigma + |x_1|)}.$$

L'algoritmo, per la costruzione di una matrice di Householder, è stato implementato in Matlab nel seguente modo:

```
function [H] = house_vec (x)
[n, m]= size (x);
sigma = norm(x);
if x(1)>=0
    k = -sigma;
else
    k=sigma;
```

```
end
lambda = sqrt( 2*sigma*(sigma+abs(x(1))));
e1 = eye(n,1);
w = (x - k*e1)/lambda;
I = eye(n);
H = I-2*(w*w');
end
```

Osserviamo che non è indispensabile costruire esplicitamente la matrice H, dato che la conoscenza di $\mathbf{w}$ o di $\mathbf{v}$ determina univocamente tale matrice. Quando utilizzeremo le matrici di Householder per effettuare la fattorizzazione QR di una matrice A, dovremmo applicare la matrice H ad un vettore $\mathbf{z}$. Questo può essere fatto senza dover costruire espilicitamente la matrice H con complessità $O(n)$, come segue

$$H\mathbf{z} = (I - 2\mathbf{w}\mathbf{w}^{\mathrm{T}})\mathbf{z} = \mathbf{z} - \gamma\mathbf{w}, \text{ dove } \gamma = 2\mathbf{w}^{\mathrm{T}}\mathbf{z},$$

Oppure si può utilizzare una seconda rappresentazione della matrice H, costruita senza l'uso della radice quadrata e della normalizzazione di $\mathbf{w}$ (il che consente una riduzione degli errori introdotti dall'algoritmo), data da

$$H = I - \tfrac{1}{\beta}\mathbf{v}\mathbf{v}^{\mathrm{T}}, \quad \mathbf{v} = \mathbf{x} - k\mathbf{e}_1, \quad \beta = \sigma(\sigma + |x_1|.$$

In tal caso applicando H a $\mathbf{z}$ semza costruirla esplicitamente, otteniamo

$$H\mathbf{z} = (I - \tfrac{1}{\beta}\mathbf{v}\mathbf{v}^{\mathrm{T}})\mathbf{z} = \mathbf{z} - \delta\mathbf{v}, \quad \text{con } \delta = \tfrac{1}{\beta}\mathbf{v}^{\mathrm{T}}\mathbf{z}$$

Abbiamo implementato anche la versione modificata dell'algoritmo di Householder, che permette di applicare la matrice H ad un vettore senza costruirla in maniera esplicita, migliorando la complessità computazionale.

```
function [House] = house_vec2 (x)
[n, m]= size (x);
sigma = norm(x);
if x(1)>=0
    k = -sigma;
else
    k=sigma;
end
lambda = sqrt( 2*sigma*(sigma+abs(x(1))));
e1 = eye(n,1);
w = (x - k*e1)/lambda;
gamma=2*w'*x;
House=x-(gamma*w);  %House rappresenta il prodotto H*x
end
```

1.2 Fattorizzazione QR di Householder

Le matrici elementari di Householder possono essere utilizzare per determinare la fattorizzazione QR di una matrice A. Consideriamo dapprima il caso in cui A sia quadrata di ordine n. Nel corso dell'algoritmo si genererá una successione di matrici A_i per $i = 1, \ldots, n$, tali che A_n sia triangolare superiore (e quindi coincida con la matrice R della fattorizzazione).
Al **passo 1** dell'algoritmo, scriviamo la matrice A in termini delle sue colonne

$$A = A^{(1)} = \begin{bmatrix} \mathbf{a}_1^{(1)} & \mathbf{a}_2^{(1)} & \ldots & \mathbf{a}_n^{(1)} \end{bmatrix}$$

Costruiamo poi, utilizzando l'algoritmo Householder, la matrice elementare di Householder H_1 tale che

$$H_1 \mathbf{a}_1^{(1)} = k_1 \mathbf{e}_1$$

Moltiplichiamo tale matrice a sinistra per A_1, cos da generare la seconda matrice della successione

$$A^{(2)} = H_1 A^{(1)} = \begin{bmatrix} \mathbf{a}_1^{(2)} & \mathbf{a}_2^{(2)} & \ldots & \mathbf{a}_n^{(2)} \end{bmatrix},$$

in cui

$$\mathbf{a}_1^{(2)} = k_1 \mathbf{e}_1, \qquad \mathbf{a}_j^{(2)} = H_1 \mathbf{a}_j^{(1)}, j = 2, \ldots, n.$$

Quindi la matrice $A^{(2)}$ ha la seguente struttura

$$A^{(2)} = \begin{bmatrix} k_1 & a_{12}^{(2)} & \ldots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{2n}^{(2)} & \ldots & a_{nn}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & \mathbf{v}_1^{\mathrm{T}} \\ \mathbf{0} & \hat{A}^{(2)} \end{bmatrix}.$$

Nella rappresentazione compatta della matrice $A^{(2)}$, $\mathbf{0}$ denota un vettore colonna, avente tutte le $(n-1)$ componenti nulle e $\hat{A}^{(2)}$ è una sottomatrice di dimensione $n-1$. Nel **passo 2** prendiamo in esame la sottomatrice $\hat{A}^{(2)}$ scritta in termini delle sue colonne

$$\hat{A}^{(2)} = \begin{bmatrix} \hat{\mathbf{a}}_2^{(2)} & \hat{\mathbf{a}}_3^{(2)} & \ldots & \hat{\mathbf{a}}_n^{(2)} \end{bmatrix}$$

e costriuamo la matrice elementare di Householder $\hat{H}_2$ di dimensione $n-1$ tale che

$$\hat{H}_2 \hat{\mathbf{a}}_2^{(2)} = k_2 \mathbf{e}_1$$

dove $\mathbf{e}_1$ è il primo elemento della base canonica di $\mathbb{R}^{n-1}$.
Occorre ora orlare la matrice $\hat{H}_2$, con una riga e una colonna della matrice identitá, perchè raggiunga dimensione n per poter essere cosí applicata a sinistra alla matrice $A^{(2)}$ e iterare cosí il procedimento.
Quindi orlando $\hat{H}_2$ otteniamo

$$H_2 = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & \hat{H}_2 & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \hat{H}_2 \end{bmatrix}$$

moltiplicandola ora a sinistra per $A^{(2)}$ otteniamo

$$A^{(3)} = H_2 A^{(2)} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \hat{H}_2 \end{bmatrix} \begin{bmatrix} k_1 & \mathbf{v}_1^{\mathrm{T}} \\ \mathbf{0} & \hat{A}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & \mathbf{v}_1^{\mathrm{T}} \\ \mathbf{0} & \hat{H}_2 \hat{A}^{(2)} \end{bmatrix},$$

ovvero

$$A^{(3)} = \begin{bmatrix} k_1 & * & \dots & \dots & * \\ 0 & k_2 & * & \dots & * \\ \vdots & 0 & & & \\ \vdots & \vdots & \hat{A}^{(3)} & & \\ 0 & 0 & & & \end{bmatrix}$$

Consideriamo ora il generico **passo i** dell'algoritmo e osserviamo che l'iterazione precedente ha prodotto la matrice

$$A^{(i)} = \begin{bmatrix} k_1 & * & \dots & \dots & \dots & * \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & k_{i-1} & * & \dots & * \\ \vdots & & 0 & & & \\ \vdots & & \vdots & & \hat{A}^{(i)} & \\ 0 & \dots & 0 & & & \end{bmatrix} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{A}^{(i)} \end{bmatrix},$$

in cui gli asterischi denotano il fatto che quegli elementi della matrice non verranno piú modificati nei successivi passi dell'algoritmo, mentre la sottomatrice $\hat{A}^{(i)}$ ha dimensione $n - i + 1$ e scritta in termini della sue colonne è del tipo

$$\hat{A}^{(i)} = \begin{bmatrix} \hat{\mathbf{a}}_i^{(i)} & \hat{\mathbf{a}}_{i+1}^{(i)} & \dots & \hat{\mathbf{a}}_n^{(i)} \end{bmatrix}.$$

Analogamente a quanto fatto in precedenza, costruiamo una matrice elementare di Householder $\hat{H}_i$ di dimensione $n-i+1$ che sia tale che

$$\hat{H}_i \hat{\mathbf{a}}_i^{(i)} = k_i \mathbf{e}_1,$$

orliamo ora la matrice $\hat{H}_i$, con i-1 righe e colonne della matrice identità, per far si che abbia dimensione n e poterla quindi applicare alla sinistra alla matrice $A^{(i)}$ in modo da otterene

$$A^{(i+1)} = H_i A^{(i)} = \begin{bmatrix} I_{i-1} & 0 \\ 0 & \hat{H}_i \end{bmatrix} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{A}^{(i)} \end{bmatrix} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{H}_i \hat{A}^{(i)} \end{bmatrix}.$$

Ovvero la matrice generata nel generico i-esimo passo sarà del tipo

$$A^{(i+1)} = \begin{bmatrix} k_1 & * & \dots & \dots & \dots & * \\ 0 & \ddots & \ddots & & & * \\ \vdots & \ddots & k_i & * & \dots & * \\ \vdots & & 0 & & & \\ \vdots & & \vdots & & \hat{A}^{(i+1)} & \\ 0 & \dots & 0 & & & \end{bmatrix}.$$

Se la matrice presa in esame è quadrata di ordine n, l'algoritmo termina all' $(n-1)$-esimo passo e l'ultima matrice della successione prodotta sarà la matrice triangolare superiore R della fattorizzazione QR di A

$$A^{(n)} = H_{n-1} A^{(n-1)} = \begin{bmatrix} k_1 & * & \dots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & * & \\ 0 & \dots & 0 & k_n \end{bmatrix} = R.$$

Riformulando l'intero algoritmo in termini matriciali otteniamo

$$R = A_{(n)} = H_{n-1} A^{(n-1)} = H_{n-1} H_{n-2} A^{(n-2)} = \dots = H_{n-1} H_{n-2} \dots H_1 A^{(1)} = Q^{\mathrm{T}} A.$$

La matrice

$$Q = H_1 H_2 \dots H_{n-1}$$

è ortogonale essendo prodotto di n matrici di Householder, quindi si ha

$$A = QR.$$

Resta quindi determinata la fattorizzazione QR della matrice A utilizzando il metodo di Householder.
Possiamo determinare la fattorizzazione QR anche di una matrice A rettangolare di dimensione mxn con $m > n$; l'algoritmo viene implementato nella stessa maniera tramite prodotti a sinistra per matrici elementari di Householder ma in tal caso esso non termina dopo $n-1$ passi, bensí occorre un ulteriore passo per azzerare gli elementi dell'ultima colonna di A (l'n-esima). Dopo n passi si otterrà la matrice

$$R = A^{(n+1)} = \begin{bmatrix} k_1 & * & \dots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ \vdots & & \ddots & k_n \\ \vdots & & & 0 \\ \vdots & & & \vdots \\ 0 & \dots & & 0 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

in cui R_1 è una matrice triangolare superiore di dimensione n che sarà non singolare nel caso in cui A sia a rango pieno.
Analogamente a quanto fatto nel caso precedente, riformuliamo l'intero algoritmo in forma matriciale evidenziando il legame tra R e A

$$R = H_n H_{n-1} \dots H_2 H_1 A,$$

da cui si ha

$$A = QR \text{ , con } Q = H_1 H_2 \dots H_n.$$

Il seguente algoritmo implementato in Matlab permette di costruire la fattorizzazione QR di una matrice A ad entrate casuali:

```
function [Q, R] = QR_fact( A )
[m, n]= size(A);
Q=eye(m); %Inizializzo Q
if m>n%matrice rettangolare alta
    q=n;
else
    q=n-1;%matrice quadrata o rettangolare bassa
end
for i= 1:q

    H_i=house_vec(A(i:m,i));
```

```
    H=eye(m);
    H(i:m,i:m)=H_i;
    A=H*A;
    Q=Q*H;


end

R=A;
```

Questo algoritmo, applicabile a matrici rettangolari o quadrate, determina la fattorizzazione QR di una matrice A (verificando in quale dei due casi si rientra); questa viene ottenuta attraverso la costruzione delle sottomatrici di Householder H_i, che verranno orlate per raggiungere la dimensione opportuna per generare le matrici $A^{(i)}$.

Capitolo 2

L'algoritmo QR

2.1 Algoritmo QR per il calcolo degli autovalori

L'algoritmo QR è un metodo iterativo che permette di approssimare tutti gli autovalori di una matrice. Se la matrice non ha una forma particolare, il suo costo computazionale è elevato per questo motivo viene solitamente implementato con alcune tecniche che consentono .di accelerarne la convergenza. Infatti viene generalmente applicato dopo aver trasformato la matrice, mediante trasformazioni di Householder o di Givens, ad una forma tridiagonale o di Hessenberg. L'algoritmo consiste nel, data una matrice quadrata A di ordine n, porre per inizializzare il metodo $A_0 = A$; nella generica iterazione si calcola la fattorizzazione QR della matrice $A_k \Rightarrow Ak = Q_k R_k$, usando ad esempio il metodo di Householder, e si calcola poi la matrice al passo successivo rimoltiplicando i fattori in ordine inverso $\Rightarrow A_{k+1} = R_k Q_k$.
Possiamo innanzittutto osservare che tutte le matrici della sucessione risultano unitariamente simili, infatti si ha

$$A_{k+1} = R_k Q_k = Q_k^{\mathrm{T}} Q_k R_k Q_k = Q_k^{\mathrm{T}} A_k Q_k$$

pertanto avranno tutte gli stessi autovalori (coincidenti appunto con quelli della matrice iniziale A). Sotto l'ipotesi che gli autovalori de A siano distinti in modulo, la successione di matrici A_k converge ad una matrice triangolare superiore

$$T = \begin{bmatrix} \lambda_1 & * & \dots & * \\ & \lambda_2 & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda_n \end{bmatrix}$$

che conterrà gli autovalori della matrice di partenza A, essendo simile a questa, nella diangonale principale.
Nel caso in cui la matrice di partenza sia simmetrica, essendo $A_{k+1}^{\mathrm{T}} = Q_k^{\mathrm{T}} A_k^{\mathrm{T}} Q_k$, saranno simmetriche tutte le matrici della successione e quindi necessariamente la matrice T (a cui esse convergono) coinciderà con una matrice diagonale

$$T = D = diag(\lambda_1, \ldots, \lambda_n).$$

Osserviamo che la fattorizzazione QR formnisce anche un'approssimazione della forma canonica di Schur della matrice A; infatti si ha

$$A_{k+1} = Q_k^{\mathrm{T}} A_k Q_k = Q_k^{\mathrm{T}} Q_{k-1}^{\mathrm{T}} A_{k-1} Q_{k-1} Q_k = \ldots$$

$$= Q_k^{\mathrm{T}} \ldots Q_0^{\mathrm{T}} A Q_0 \ldots Q_k = U_k^{\mathrm{T}} A U_k,$$

in cui abbiamo indicato con $U_k = Q_0 \ldots Q_k$; pertanto risulta

$$\lim_{k\to\infty} A_k = T = U^{\mathrm{T}} A U, \text{ con } U = \prod_{i=1}^{\infty} Q_i.$$

Nela caso in cui A sia simmetrica otteniamo la sua fattorizzazione spettrale

$$A = U D U^{\mathrm{T}},$$

in cui U è la matrice le cui colonne sono gli autovalori di A.
Abbiamo affermato in precendenza che il metodo risulta essere convergente sotto l'ipotesi che gli autovalori siano tutti distinti in modulo

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|;$$

tuttavia anche se questa ipotesi non fosse soddisfatta il metodo può essere modificato in modo tale da approssimare comunque gli autovalori.
Infatti nel caso la matrice abbia due autovalori con stesso modulo $\lambda_1 = \overline{\lambda_2}$, in tal caso sarà presente un blocco diagonale 2×2 avente l'elemento sottodiagonale non nullo ma gli autovalori tenderanno a λ_1 e λ_2. In un caso del genere la successione della matrici A_k non convergerà ad una matrice triangolare superiore ma sarà sufficiente calcolare separatamente gli autovalori dei blocchi diagonali che non si stabilizzano applicando l'algoritmo classico, ovvero calcolando gli zeri del polinomio caratteristico corrispondente.

2.2 Passaggio in forma di Hessenberg

L'algoritmo QR è il metodo piú diffuso per il calcolo degli autovalori di matrici di dimensione non troppo elevata; per diminuire il numero di iterazioni e quindi la complessità computazionale, viene spesso applicato insieme a delle tecniche che ne accelerano appunto la convergenza.
Una tecnica spesso utilizzata consiste nel traformare la matrice A, di cui vogliamo calcolare gli autovalori con l'algoritmo QR, in una matrice in forma di Hessenberg cioè avente la seguente struttura

$$\tilde{A} = \begin{bmatrix} * & \dots & \dots & * \\ * & \ddots & & \vdots \\ & \ddots & \ddots & \vdots \\ & & * & * \end{bmatrix}$$

in cui cioè $\tilde{a_{ij}} = 0$ per $i > j + 1$.
Questa tecnica è molto utilizzata per l fatto che la forma di Hessenberg è invariante per trasformazioni QR, ovvero se la matrice A a cui si deve applicare l'algoritmo QR è in forma di Hessenberg avranno questa struttura anche tutte le matrici della successione generata dal metodo. Questo accelera la convergenza per il fatto che l'algortimo in tal caso dovrà azzerare solo gli $n-1$ elementi sottodiagonali (per ottenere la matrice triangolare superiore T). Per azzerare i suddetti $n-1$ elementi sottodiagonali sono sufficienti $n-1$ trasformazioni di Givens e pertanto nel calcolare la fattorizzazione QR di una matrice in forma di Hessenberg si ha una notevole diminuzione del costo computazionale di ogni iterazione.
Per effettuare il passaggio in forma di Hessenberg sono sufficienti $n-2$ trasformazioni di Householder applicate contemporaneamente alla destra e alla sinistra della matrice A così da conservare la similitudine. Noi, nell'algoritmo QR, abbiamo effettuato il passaggio in forma di Hessenberg applicando alla matrice A la funzione `hess` predefinita in Matlab.

```
function[lambda]=QR_eig(A)
n=size(A);
A=hess(A);
tau=1e-8;
Nmax=200;

lambda(:,1)=diag(A);
for i=(1:n-1)

    for k=(1:Nmax)
```

```
        [Q,R]=QR_fact(A);
        A=R*Q;
        lambda(:,k+1)=diag(A);
        if norm(diag(A,-1), inf)<tau
            break
        end


    end
    lambda=diag(A);
end
```

Nell'implementazione, dopo aver ridotto in forma di Hessenberg e aver assegnato una certa tolleranza e il numero massimo di iterazioni, utilizziamo l'algoritmo `QR_fact` per fattorizzare la matrice A e come criterio di stop utilizziamo il seguente

$$|a_{i+1,1}| \leqslant \tau,\ i = 1, \ldots, n-1,$$

verificando che gli elementi sottodiagonali siano inferiori in modulo alla tolleranza assegnata.

Capitolo 3

Zeri di una formula Gaussiana

Il metodo più utilizzato per il calcolo degli zeri di un polinomio si basa sulla risoluzione di un problema agli autovalori. Vedremo quindi come applicare l'algoritmo QR per il calcolo degli zeri di una formula Gaussiana.

3.1 Formule di quadratura e polinomi ortogonali

Le formule di quadratura sono delle formule utilizzate per il calcolo di integrali definiti; consistono nel determinare il valore numerico di un integrale definito mediante una combinazione lineare di valori assunti dalla funzione integranda.
Supponendo di voler calcolare l'integrale definito

$$I(f) = \int_a^b f(x)\,dx$$

una formula di quadratura è un'espressione del tipo

$$I_n(f) = \sum_{j=0}^{n} \alpha_j f(x_j),$$

ovvero una combinazione lineare di valori assunti dalla funzione integranda sugli $n+1$ punti x_j, detti **nodi** della formula di quadratura, con coefficienti α_j chiamati **pesi** della stessa. Le formule Gaussiane sono delle particolari formule di quadratura, aventi precisione ottimale (diciamo che una formula di quadratura ha precisione algebrica n se essa integra esattamente un polinomio di grado n).
Per esaminare le caratteristiche di una formula di quadratura Gaussiana occorre definire i polinomi ortogonali e per fare una trattazione pi generale

considereremo il calcolo dell'integrale pesato

$$I(f) = \int_a^b w(x)f(x)\,dx \tag{3.1}$$

per un'opportuna funzione peso $w(x)$.
I **polinomi ortogonali** sono una famiglia di polinomi aventi mutuo prodotto scalare pesato

$$\langle f, g\rangle = \textstyle\int_a^b w(x)f(x)g(x)\,dx$$

nullo. Osserviamo che è possibile definire dei polinomi monici ortogonali rispetto al prodotto scalare precedente mediante la seguente formula ricorsiva a tre termini

$$\begin{aligned} &p_0(x) = 1,\ p_{-1}(x) = 0,\\ &p_{k+1}(x) = (x-\alpha_k)p_k(x) - \beta_k^2 p_{k-1}(x) \end{aligned}$$

in cui

$$\alpha_k = \frac{\langle xp_k, p_k\rangle}{\langle p_k, p_k\rangle},$$

$$\beta_0 = 0,$$

$$\beta_k^2 = \frac{\langle xp_k, p_{k-1}\rangle}{\langle p_{k-1}, p_{k-1}\rangle} = \frac{\langle p_k, p_k\rangle}{\langle p_{k-1}, p_{k-1}\rangle}, \qquad k = 1, 2, \ldots$$

3.2 Formule di quadratura Gaussiane

Una formula di quadratura Gaussiana per il calcolo dell'integrale (3.1)è una formula di quadratura avente come nodi gli zeri del polinomio p_{n+1} ortogonale allo spazio Π_n rispetto al prodotto scalare pesato e i cui nodi sono scelti in modo da renderla interpolatoria

$$\alpha_j = \int_a^b L_j(x)w(x)\,dx = \int_a^b \frac{p_{n+1}(x)}{c_j(x-x_j)}w(x)\,dx,$$

in cui $L_j(x) = \prod_{k=0,k\neq j}^n \frac{(x-x_k)}{(x_j-x_k)}$ sono i polinomi caratteristici di Lagrange e $c_j = \prod_{k=0,k\neq j}^n (x_j - x_k)$.
Nella costruzione di una formula Gaussiana l problema più complesso è quello del calcolo dei nodi della stessa, ovvero degli zeri del polinomio ortogonale

p_{n+1} che risultano semplici, reali e contenuti nell'intervallo aperto (a, b).
Osservimo che tuttavia si può dimostrare che $p_{n+1}(\lambda)$ è il polinomio caratteristico della matrice tridiagonale simmetrica

$$J_n = \begin{bmatrix} \alpha_0 & \beta_1 & & & \\ \beta_1 & \alpha_1 & \beta_2 & & \\ & \beta_2 & \alpha_2 & \ddots & \\ & & \ddots & \ddots & \beta_n \\ & & & \beta_n & \alpha_n \end{bmatrix}$$

le cui componenti non nelle risultano essere i coefficienti della formula ricorsiva che definisce i polinomi monici ortogonali.
Gli zeri del polinomio p_{n+1} sono quindi gli autovalori della matrice J_n; essendo questi zeri, come abbiamo detto, reali distinti e contenuti nell'intervallo (a, b), si utilizza in maniera efficace l'algoritno QR per la loro determinazione (osserviamo che J_n è tridiagonale simmetrica e come tale avrà autovalori reali e distinti).

3.3 Esempi di famiglie di polinomi ortogonali

Abbiamo preso in esame alcune famiglie di polinomi ortogonali di cui si vogliono calcolare gli zeri che saranno appunto i nodi di una formula Gaussiana. Abbiamo considerato i polinomi di Legendre, i polinomi di Hermite e quelli di Laguerre. I **polinomi di Legendre** hanno peso $w(x) = 1$ e l'intervallo d'integrazione è $[a, b] = [-1, 1]$.
Mediante a formula ricorsiva vista in precedenza è definito da

$$\begin{cases} P_0(x) = 1, \quad P_1(x) = x, \\ P_{n+1}(x) = xP_n(x) - \frac{n^2}{4n^2-1} P_{n-1}(x), \quad n \geq 1. \end{cases}$$

Se quindi facciamo riferimento ala formula ricorsiva generale si evince che

$$\alpha_n = 0, \quad \beta_n = \sqrt{\frac{n^2}{4n^2 - 1}}.$$

I **polinomi di Hermite**, aventi funzione peso $w(x) = e^{-x^2}$ e intervallo d'integrazione coincidente con$\mathbb{R}$ sono definiti da

$$\begin{cases} H_0(x) = 1, \quad H_{-1}(x) = 0, \\ H_{n+1}(x) = xH_n(x) - \frac{1}{2} nH_{n-1}(x), \quad n \geq 0. \end{cases}$$

In tal caso

$$\alpha_n = 0, \quad \beta_n = \sqrt{\frac{n}{2}}.$$

Osserviamo che in questo caso, così come per i polinomi di Laguerre, essendo la funzione peso non unitaria occorre modificare l'integrale per far comparire la funzione peso in modo da poter applicare la formula Gaussiana all'integrale pesato.
Analogamente i **polinomi di Laguerre**, con peso e^{-x} e intervallo d'integrazione $\mathbb{R}^+$, sono definiti da

$$\begin{cases} L_0(x) = 1, \quad L_{-1}(x) = 0, \\ L_{n+1}(x) = (x - 2n - 1)L_n(x) - n^2 L_{n-1}(x), \quad n \geq 0. \end{cases}$$

In questo caso i coefficienti della formula ricorsiva sono dati da

$$\alpha_n = 2n + 1, \quad \beta_n = n.$$

Gli algoritmi implementati in Matlab, per la determinazione degli zeri dei suddetti polinomi, consistono nel calcolare i coefficienti α e β e costruire quindi la matrice J_n calcolandone gli autovalori tramite l'algoritmo QR. Verranno presentati nella prossima sezione, nella quale discuteremo alcuni test effettuati.

3.4 Test

Il primo algoritmo implementato è per il calcolo dei nodi e dei pesi della formula gaussiana relativa ai polinomi di Legendre:

```
function [pesi, nodi, integrale,J]= Legendre(fun)
 tic
n=input('Inserire␣il␣grado␣del␣polinomio:␣\n');
k= [1:n]';
beta=k./sqrt(4*k.*k-1);
J=diag(beta,1)+diag(beta,-1);
nodi=QR_eig(J);
A=zeros(n+1,1);
A(:,1)=ones(n+1,1);
A(:,2)=nodi;
for i=2:n
    A(:,i+1)=nodi.*A(:,i)-beta(i-1)^2*A(:,i-1);
end
```

```
e1=[1;zeros(n,1)];
pesi=A'\(2*e1);
integrale= pesi'*fun(nodi);
toc
```

Abbiamo costruito la matrice J_n, di cui abbiamo calcolato gli autovalori (nodi della formula gaussiana) con l'algoritmo `QR_eig` e i pesi risolvendo il sistema lineare:

$$A^{\mathrm{T}}\alpha = \|p_0\|^2 e_1.$$

Analogamente, per i polinomi di Laguerre:

```
function [pesi, nodi, integrale,J]= Laguerre(fun)
n=input('Inserire il grado del polinomio: \n');
k1= [1:n+1]';
k2=[1:n]';
beta=sqrt(k2);
alfa=k1.*2+1;

J=diag(alfa)+diag(beta,1)+diag(beta,-1);
nodi=QR_eig(J);
A=zeros(n+1,1);
A(:,1)=ones(n+1,1);
A(:,2)=nodi;
for i=2:n
    A(:,i+1)=(nodi-alfa).*A(:,i)-beta(i-1)^2*A(:,i-1);
end
e1=[1;zeros(n,1)];
pesi=A'\(2*e1);
w=exp(nodi);
fun1=fun(nodi).*w;
integrale= pesi'*fun1;
```

e di Hermite:

```
function [pesi, nodi, integrale,J]= Hermite(fun)
n=input('Inserire␣il␣grado␣del␣polinomio:␣\n');
k=[1:n]';
beta=sqrt(k)/2;
J=diag(beta,1)+diag(beta,-1);
nodi=QR_eig(J);
A=zeros(n+1,1);
A(:,1)=ones(n+1,1);
A(:,2)=nodi;
for i=2:n
    A(:,i+1)=nodi.*A(:,i)-beta(i-1)^2*A(:,i-1);
end
e1=[1;zeros(n,1)];
nodes=nodi.^2;
pesi=A'\(2*e1);
w=exp(nodes);
fun1=fun(nodi).*w;
integrale= pesi'*fun1;
```

Osserviamo che i pesi risultano essere i quadrati delle prime componenti degli autovettori di J_n. Possiamo controllare questo fatto calcolando la fattorizzazione spettrale di J_n, che è simmetrica:

$$J_n = UDU^{\mathrm{T}},$$

in cui U la matrice degli autovettori di J_n.
Abbiamo effettuato alcune prove confrontando i pesi calcolati tramite la risoluzione del sistema lineare e quelli ottenuti con la fattorizzazione spettrale, variando il grado del polinomio.
In generale abbiamo riscontrato che al crescere del grado n del polinomio la differenza tra i pesi calcolati nei due suddetti modi risulta essere minore, denotando una migliore approssimazione.
D'altra parte il tempo necessario per l'elaborazione aumenta al crescere di n, come osservato utilizzando `tic toc`.
Infatti abbiamo utilizzato la formula di Legendre, applicandola a polinomi di grado compreso tra 1 e 50, con l'obiettivo di determinare l'errore tra i pesi calcolati attraverso il sistema

$$A^{\mathrm{T}}\alpha = \|p_0\|^2 e_1$$

e, per l'appunto, le prime componenti della matrice degli autovettori che abbiamo trovato tramite la forma di Schur. La prima prova l'abbiamo effettuata utilizzando l'algoritmo QR mostrato precedentemente, per il calcolo dei nodi della formula Gaussiana .
Questo algoritmo, ottimizzato per matrici simmetriche, si è dimostrato non idoneo per l'applicazione alla matrice J_n, che nel caso di Legendre ha diagonale principale avente elementi tutti nulli. Di conseguenza, l'algoritmo mostrava come risultati nodi, pesi ed approssimazione dell'integrale come forme indeterminate. Questo risultato è spiegabile notando che la matrice J_n ha autovalori non distinti in modulo e quindi non viene rispettata la condizione di convergenza dell'algoritmo QR classico. Sappiamo però che con una semplice modifica possiamo fare in modo che vengano calcolati, utilizzando il polinomio caratteristico corrispondente (di grado 2), gli autovalori dei blocchetti diagonali 2×2 che non si stabilizzano.
Con l'implementazione dell'algoritmo QR modificato siamo riusciti ad effettuare il calcolo dei nodi in maniera corretta. Abbiamo inoltre calcolato i pesi della formula Gaussiana, come precedentemente spiegato, e l'approssimazione di un integrale noto. In particolare abbiamo preso in esame l'integrale della funzione $f(x) = x^4$, ed abbiamo trovato che la formula Gaussiana fornisce un'approssimazione sufficientemente accurata dell'integrale (per n sufficientemente grande risulta essere esatta).

Riguardo al confronto fra i pesi della formula e le prime componenti delle colonne della matrice U, abbiamo invece riscontrato un errore. Abbiamo considerato, per ogni n (da 1 a 50), la norma infinito dell'errore così definita:

$$E_n = \|u_j - \alpha_j\|_\infty, \quad j = 1, \ldots, 50,$$

dove gli u_j sono le prime componenti al quadrato delle colonne di U, mentre gli α_j sono ovviamente i pesi.
Abbiamo trovato, come ipotizzato, che al crescere di n l'errore tende a diminuire significativamente e ciò fa ipotizzare che, facendo crescere ulteriormente n, l'errore possa tendere a zero.
Riportiamo nella seguente tabella i risultati numerici per l'errore E_n, e successivamente ne mostriamo l'andamento grafico.

n	errore	n	errore
1	0.5000	26	0.0976
2	0.6111	27	0.1055
3	0.4782	28	0.0912
4	0.3602	29	0.0978
5	0.3823	30	0.0789
6	0.3171	31	0.0903
7	0.2631	32	0.0901
8	0.2896	33	0.0878
9	0.2622	34	0.0852
10	0.2054	35	0.0826
11	0.2099	36	0.0637
12	0.1579	37	0.0790
13	0.1680	38	0.0678
14	0.1326	39	0.0752
15	0.1759	40	0.0727
16	0.1674	41	0.0613
17	0.1439	42	0.0695
18	0.1430	43	0.0683
19	0.1439	44	0.0666
20	0.0942	45	0.0550
21	0.1056	46	0.0632
22	0.1163	47	0.0553
23	0.1116	48	0.0614
24	0.1165	49	0.0605
25	0.1130	50	0.0591

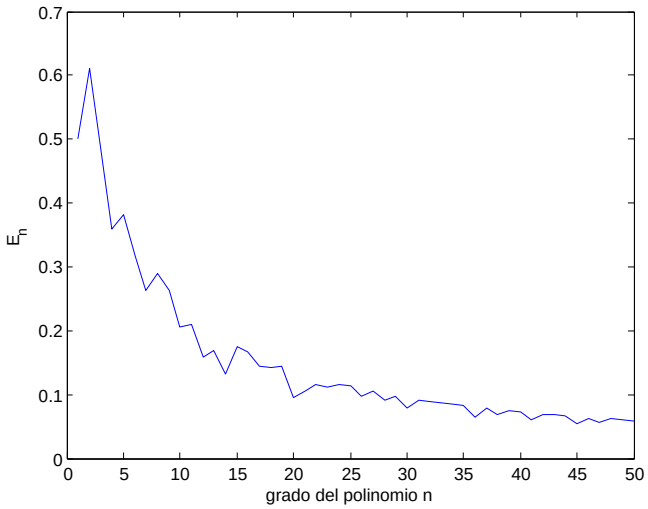

Figura 3.1: Andamento dell'errore al crescere di n

Bibliografia

[1] G. Rodriguez. *Algoritmi Numerici*, Pitagora Editrice Bologna, 2008.

[2] G.H. Golub, C.F. Van Loan. *Matrix Computations*, The John Hopkins University Press, Baltimore, third edition 1996.

[3] D. Bini, M. Capovani, O. Menchi . *Metodi Numerici per l' Algebra Lineare*, Zanichelli, Bologna 1987.