Determinazione degli zeri di una formula Gaussiana mediante l'algoritmo QR per il calcolo degli autovalori di una matrice.

Andrea Salimbeni 39919

Relatore
Prof. Giuseppe Rodriguez

Indice

Lo scopo della tesina è quello di determinare gli zeri di una formula gaussiana mediante l'algoritmo QR per il calcolo degli autovalori di una matrice.

Premesse

Definendo una generica formula di quadratura come:

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

$$con \quad \alpha_j=\int_a^b L_j(x)\,dx$$

Gli α_j sono i pesi della formula mentre gli x_j sono i nodi della stessa.
Prima di analizzare le caratteristiche di una formula di quadratura gaussiana è necessario definire i polinomi ortogonali.
Si dice ortogonale la famiglia di polinomi i cui elementi hanno mutuo prodotto scalare nullo.
È possibile definire dei polinomi monici ortogonali mediante la formula ricorsiva a tre termini seguente:

$$p_0(x)=1;$$
$$p_{-1}(x)=0;$$
$$p_{k+1}(x)=(x-\alpha_k)\cdot p_k(x)-\beta_k^2\cdot p_{k-1}(x)$$
con
$$\alpha_k=\frac{\langle xp_k, p_k\rangle}{\langle p_k, p_k\rangle} \quad k=0,1,...$$
$$\beta_0=0,$$
$$\beta_k=\frac{\langle xp_k, p_{k-1}\rangle}{\langle p_{k-1}, p_{k-1}\rangle}, \quad k=1,2,...$$

Formule di quadratura Gaussiane

È possibile, a questo punto, definire con chiarezza una formula di quadratura gaussiana come quella formula avente: come nodi gli zeri del polinomio p_{n+1} ortogonale rispetto al prodotto scalare, come pesi

$$\alpha_j=\int_a^b L_j(x)w(x)dx=\int_a^b \left(\frac{p_{n+1}}{c_j\cdot(x_j-x_k)}\right)w(x)\,dx$$
con
$$c_j=\prod_{K=0,k\neq j}^{n}(x_j-x_k).$$

Il problema più complesso risulta quindi la determinazione dei nodi. È possibile però dimostrare che la matrice:

$$J_n = \begin{pmatrix} \alpha_0 & \beta_1 & 0 & 0 & 0 \\ \beta_1 & \alpha_1 & \beta_2 & 0 & 0 \\ 0 & \beta_2 & \alpha_2 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \beta_n \\ 0 & 0 & \cdots & \beta_n & \alpha_n \end{pmatrix}$$

ha come polinomio caratteristico proprio $p_{n+1}()$. Essendo la matrice tridiagonale simmetrica avrà autovalori reali e distinti. L'algoritmo QR si presta bene quindi al loro calcolo.

L'algoritmo QR

L'algoritmo QR permette di determinare gli autovalori di una matrice. Nell'ipotesi che gli autovalori siano tutti distinti in modulo può essere applicato senza problemi. In quei casi, compreso come si vedrà in seguito il nostro, in cui questa condizione non sia verificata si può comunque applicare il metodo modificandolo opportunamente. Posta A_0 la matrice di cui si vogliono calcolare gli autovalori, l'algoritmo QR fattorizza QR la matrice, determina una matrice A_1 data dal prodotto R*Q per poi rifattorizzarla QR e reiterare il processo. Si dimostra che la successione delle A_k converge a una matrice triangolare superiore T avente come elementi della diagonale principale gli autovalori di A_0.
Le matrici A_k infatti sono unitariamente simili:

$$A_{k+1}=R_k\cdot Q_k=Q_k^T\cdot Q_k\cdot R_k\cdot Q_k=Q_k^T\cdot A_k\cdot Q_k \;\Rightarrow\; A_{k+1} \;\text{è simile ad}\; A_k$$

Come precedentemente annunciato, nel caso in cui la matrice A_0 di partenza non abbia autovalori distinti in modulo. Il metodo non converge a una matrice T ma ad una matrice così strutturata:

$$\tilde{T} = \begin{pmatrix} \# & \# & * & \cdots & * \\ \# & \# & * & \cdots & * \\ 0 & 0 & \lambda_3 & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & * \\ 0 & 0 & 0 & \cdots & \lambda_n \end{pmatrix} \; con \;\; |\lambda_1|=|\lambda_2|$$

Il blocco 2x2 non si stabilizza ma ha comunque autovalori pari a λ_1 e λ_2, determinabili anche con l'algoritmo classico del calcolo degli zeri del polinomio caratteristico. All'algoritmo QR possono essere associati vari metodi per accelerarne la convergenza.
La fattorizzazione QR della matrice A può essere fatta per mezzo di matrici di Householder o di Givens e in questo lavoro la scelta è ricaduta su Householder.

Matrici elementari di Householder

Una matrice elementare di Householder può essere così definita:

$$H = I - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} \quad con \quad I = matrice\ identità$$
$$\|\boldsymbol{w}\|_2 = 1 \quad , \quad \boldsymbol{w} \in \mathbb{R}$$

una matrice così definita è simmetrica e ortogonale. Infatti:

$$Simmetria: \quad H = (I - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}})^T = I^T - 2 \cdot (\boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}})^T = I - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} .$$

$$Ortogonalità: \quad H * H^T = (I - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}}) \cdot (I - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}})^T = (I - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}})^2 =$$
$$= I - 4 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} + 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} = I - 4 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} + 4 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} = I$$

Assegnato un generico vettore x si vuole determinare **w** tale che:

$$H \cdot \boldsymbol{x} = k \cdot \boldsymbol{e_1}$$

Per cominciare si nota che:

$$\|H \cdot \boldsymbol{x}\| = \|k \cdot \boldsymbol{e_1}\|$$
$$\|H\| \cdot \|\boldsymbol{x}\| = \|k\| \cdot \|\boldsymbol{e_1}\|$$
$$\|\boldsymbol{x}\| = \|k\|$$
$$da\ cui \quad k = \mp \sigma \qquad con \qquad \sigma = \|\boldsymbol{x}\|$$

Calcolando poi:

$$\|H \cdot \boldsymbol{x}\| = (I - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}}) \cdot \boldsymbol{x} = x\,\boldsymbol{x} - 2 \cdot \boldsymbol{w} \cdot \boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{x} = \boldsymbol{x} - 2 \cdot (\boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{x}) \cdot \boldsymbol{w} = k \cdot \boldsymbol{e_1}$$
$$\boldsymbol{w} = \frac{\boldsymbol{x} - k \cdot \boldsymbol{e_1}}{2 \cdot \boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{x}} = \frac{\boldsymbol{x} - k \cdot \boldsymbol{e_1}}{\|\boldsymbol{x} - k \cdot \boldsymbol{e_1}\|} \qquad dato\ che \|\boldsymbol{w}\| = 1.$$

infine:

$$\boldsymbol{x}^{\boldsymbol{T}} \cdot H \cdot \boldsymbol{x} = \boldsymbol{x}^{\boldsymbol{T}} \cdot \boldsymbol{x} - 2 \cdot (\boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{x})^2 = k \cdot x_1$$
$$da\ cui \qquad (\boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{x})^2 = \frac{\sigma^2 - k \cdot x_1}{2}$$
$$e\ per\ evitare\ la\ cancellazione: \qquad k = -sign(x_1) \cdot \sigma$$
$$e \quad \lambda = 2 \cdot \boldsymbol{w_T} \cdot \boldsymbol{x} = \sqrt{2 \cdot \sigma \cdot (\sigma + |x_1|)}$$

L'algoritmo è stato quindi implementato su matlab nel seguente modo:

```
%Householder
%%
function H = myhousefat(x)
[nn,mm]=size(x);%Determina la dimensione del vettore x
Ih=eye(nn);%costruisce una matrice identita della dimensione di x

e = zeros(nn,1);%crea il versore e1
e(1)=1;

s=norm(x);%determina la norma di x
if x(1) >= 0%ciclo if che sostituisce la funzione segno
   kk = -s;
else
   kk = s;
end
L=(2*s*(s+abs(x(1))))^(1/2);
w = (x-kk*e)/L;
H = Ih-(2*w*w');
```

È stato necessario introdurre un ciclo if in quanto la funzione “sign(x)” di matlab restituisce 0 se x = 0 e non 1 come richiesto dall'algoritmo.

È stato effettuato un test su un vettore casuale per testarne il funzionamento:

```
>> w = rand(8,1)

w =

   0.3111
   0.9234
   0.4302
   0.1848
   0.9049
   0.9797
   0.4389
   0.1111

>> H = myhousefat(w);
>> e = H*w

e =

  -1.7755
   0.0000
   0.0000
   0.0000
   0.0000
   0.0000
   0.0000
        0
```

Fattorizzazione QR con Householder

Azzerando tutti i termini di un vettore tranne il primo, la matrice elementare di Householder può essere efficaciemente utilizzata per fattorizzare QR una qualsiasi matrice.
Scrivendo una generica matrice A come matrice di vettori colonna, al primo passo dell'algoritmo si avrà:

$$A = A^{(1)} = [\,\boldsymbol{a}_1^{(1)} \;\; \boldsymbol{a}_2^{(1)} \;\; \boldsymbol{a}_3^{(1)} \;\; \dots \;\; \boldsymbol{a}_n^{(1)}\,]$$

Applicando l'algoritmo esposto nel paragrafo precedente si possono facilmente azzerare tutti i termini sotto il termine a_1.
Quindi:

$$A^{(2)} = H_1 \cdot A^{(1)} = \begin{bmatrix} k_1 & a_{12}^{(2)} & a_{13}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & \boldsymbol{v}_1^T \\ \mathbf{0} & \hat{A}^{(2)} \end{bmatrix}$$

Al passo successivo occorre considerare selettivamente la matrice $\hat{A}^{(2)}$:

$A^{(2)} = [\,\hat{\boldsymbol{a}}_2^{(2)} \;\; \hat{\boldsymbol{a}}_3^{(2)} \;\; \hat{\boldsymbol{a}}_4^{(2)} \;\; \dots \;\; \hat{\boldsymbol{a}}_n^{(2)}\,]$ e applicare householder solo al vettore colonna $\hat{\boldsymbol{a}}_2^{(2)}$.
Per far questo senza alterare gli zeri già introdotti in A bisogna costruire la H nel seguente modo:

$$H_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \hat{H}_2 & \\ 0 & & & \end{bmatrix}$$

Quindi per il passo "i" si avrà:

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

L'algoritmo termina al passo n-1 se la matrice è quadrata, al passo n se la matrice ha numero di righe maggiore del numero di colonne.
Si ha pertanto:

$$R = A^{(n)} = H_{n-1} \cdot A^{n-1} = H_{n-1} \cdot H_{n-2} \cdot A^{n-2} = \dots = H_{n-1} \cdot H_{n-2} \cdot \dots H_1 \cdot A^1 = Q^T \cdot A$$
$$Q^T \; ortogonale \Rightarrow A = Q \cdot R$$

L'algoritmo di fattorizzazione QR scritto su matlab è il seguente:

```
%Householder
%%
function [Qd,Rd] = miaqrconhouse(X1)
[m1,n1]=size(X1);   %determina la dimensione della matrice in ingresso X1
Q1 = eye(size(X1));%costruisce una matrice Identità della stessa dimensione di X1
if m1==n1% Caso 1(matrice quadrata)
   for i=1:(n1-1)%Essendo la matrice quadrata il cilco si ferma a n-1
      ap = X1(i:n1,i);%prende il vettore da “azzerare” ad eccezione del primo termine
      Hp = myhousefat(ap);%Applica la matrice elementare di householder al vettore ap
      Hs = eye(n1);% costruisce una matrice identità di dimensione n1
      Hs(i:n1,i:n1) = Hp;% sostituisce alla matrice identità gli elementi della matrice di householder appena calcolata
      Q1 = Q1*Hs;%aggiorna la matrice data dal prodotto di tutte le matrici di householder
      X1=Hs*X1;%aggiorna la matrice X1
   end
   Qd=Q1;%restituisce Q
   Rd=X1;%restituisce R
elseif m>n%Caso 2 (matrice rettangolare)
   for i=1:n1
      ap = X1(i:n1,i);
      Hp = myhousefat(ap);
      Hs = eye(n1);
      Hs(i:n1,i:n1) = Hp;
      Q1 = Q1*Hs;
      X1=Hs*X1;
   end
   Qd=Q1;
   Rd=X1;
else
   'Errore'
end
```

Dove come si può notare si è diviso il caso di una matrice quadrata da quello di una matrice rettangolare.
Di seguito invece l'applicazione dell'algoritmo ad una matrice casuale:

```
>> A = rand(8)

A =


   0.2581   0.2967   0.9289   0.5211   0.0377   0.1366   0.8909   0.9047
   0.4087   0.3188   0.7303   0.2316   0.8852   0.7212   0.3342   0.6099
   0.5949   0.4242   0.4886   0.4889   0.9133   0.1068   0.6987   0.6177
   0.2622   0.5079   0.5785   0.6241   0.7962   0.6538   0.1978   0.8594
   0.6028   0.0855   0.2373   0.6791   0.0987   0.4942   0.0305   0.8055
   0.7112   0.2625   0.4588   0.3955   0.2619   0.7791   0.7441   0.5767
   0.2217   0.8010   0.9631   0.3674   0.3354   0.7150   0.5000   0.1829
   0.1174   0.0292   0.5468   0.9880   0.6797   0.9037   0.4799   0.2399
```

```
>> [Q R] = miaqrconhouse(A)

Q =

  -0.2048  -0.1579   0.6411   0.2483  -0.3899   0.4870  -0.0112  -0.2668
  -0.3243  -0.0698   0.2824   0.4766   0.5872  -0.2647   0.4066   0.0541
  -0.4720  -0.0541  -0.2778  -0.0833   0.3690   0.6394  -0.2501   0.2875
  -0.2080  -0.4069  -0.1481  -0.4381   0.1939  -0.0325   0.1744  -0.7129
  -0.4783   0.3562  -0.0816  -0.2994  -0.3847  -0.0202   0.6055   0.1816
  -0.5643   0.2273  -0.1144   0.2146  -0.1887  -0.4176  -0.5505  -0.2401
  -0.1759  -0.7875  -0.0852   0.0242  -0.3127  -0.2460  -0.0103   0.4273
  -0.0932   0.0544   0.6190  -0.6133   0.2193  -0.2181  -0.2678   0.2485


R =

  -1.2604  -0.8027  -1.3708  -1.2471  -1.2088  -1.3342  -1.2289  -1.6186
   0.0000  -0.8378  -0.9994  -0.2827  -0.5735  -0.5045  -0.4700  -0.2816
   0.0000  -0.0000   0.7649   0.6508   0.2567   0.5338   0.6090   0.4546
   0.0000  -0.0000  -0.0000  -0.7897  -0.3757  -0.4353   0.1040  -0.1727
  -0.0000  -0.0000  -0.0000  -0.0000   0.9532   0.1738  -0.0582  -0.0234
   0.0000   0.0000  -0.0000   0.0000   0.0000  -0.7858   0.2467   0.2917
   0.0000  -0.0000   0.0000  -0.0000  -0.0000   0.0000  -0.5392   0.3373
  -0.0000  -0.0000  -0.0000   0.0000  -0.0000   0.0000        0  -0.4980

>> I = Q*Q'

I =

   1.0000   0.0000  -0.0000   0.0000   0.0000  -0.0000   0.0000   0.0000
   0.0000   1.0000  -0.0000  -0.0000   0.0000   0.0000  -0.0000   0.0000
  -0.0000  -0.0000   1.0000  -0.0000  -0.0000  -0.0000   0.0000  -0.0000
   0.0000  -0.0000  -0.0000   1.0000   0.0000   0.0000  -0.0000  -0.0000
   0.0000   0.0000  -0.0000   0.0000   1.0000  -0.0000   0.0000   0.0000
  -0.0000   0.0000  -0.0000   0.0000  -0.0000   1.0000   0.0000   0.0000
   0.0000  -0.0000   0.0000  -0.0000   0.0000   0.0000   1.0000   0.0000
   0.0000   0.0000  -0.0000  -0.0000   0.0000   0.0000   0.0000   1.0000
```

Implementazione dell'algoritmo QR su matlab

Avendo già descritto l'algoritmo in precedenza, non resta a questo punto che implementarlo in matlab in modo da determinare gli autovalori di una matrice assegnata.

```
%Fattorizzazione QR
%%
function lambda = miautovalqr(A)
[m,n]=size(A); %Determina la dimensione di A
N=100;% Fissa il numero di iterazioni
tau=1e-8;%Fissa la soglia sotto la quale un elemento è considerato nullo
k=1;%Inizializza k
if max(diag(A)) == 0 %discrimina il caso in cui la diagonale principale abbia tutti elementi nulli
   simm = 1;
else
   simm = 0;
end

if m==n
      while k<N
      [Q,R] = miaqrconhouse(A); %applica l'algoritmo QR alla matrice A
      A=R*Q;%Costruisce la matrice Ak
      k=k+1;%Aggiorna k
   end
end
if simm %Caso con diagonale avente elementi della diagonale principale tutti nulli
   lambda = [];%Inizializza il vettore lambda contenente gli autovalori come vettore vuoto
   for i = 2:2:n%ciclo for con i che incrementa di 2 per ogni passo per evitare di considerare 2 volte lo stesso autovalore
      if abs(A(i,i-1)) > tau%verifica che l'elemento A(i,i-1) sia diverso da zero, se lo è si hanno due autovalori uguali in
                            %modulo
         lambda = [lambda; eig(A(i-1:i,i-1:i))];%Calcola i due autovalori con eig.
      end
   end
   if size(lambda,1) == n-1%Se la dimensione di lambda è inferiore a n aggiunge l'elemento zero come spiegato in
                           %seguito
      lambda = [lambda; 0];
   elseif size(lambda,1) == n
      lambda = lambda;
   end
else
   lambda = diag(A);%Se gli elementi della diagonale principale sono diversi da zero coincidono con gli autovalori
end
lambda = sort(lambda);
```

Nell'algoritmo sono presenti alcuni accorgimenti che verranno spiegati di seguito. Nel caso in cui la matrice abbia autovalori uguali in modulo a due a due (cosa che come si vedrà in seguito accade con legendre e hermite) dopo le N iterazioni previste assume una struttura di questo tipo:

$$J_n = \begin{pmatrix} 0 & * & 0 & \cdots & 0 & 0 \\ * & 0 & * & \cdots & 0 & 0 \\ 0 & * & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & * & \ddots & * & 0 \\ 0 & 0 & 0 & \ddots & 0 & * \\ 0 & 0 & 0 & \cdots & * & 0 \end{pmatrix}$$

Si è quindi introdotta una condizione if che discrimini proprio questa situazione agendo sulla variabile simm. Quando simm è vera e cioè si è nella situazione esposta sopra, non abbiamo gli autovalori nella diagonale principale. Si inizializza quindi il vettore lambda come vettore vuoto e si controllano gli elementi della sottodiagonale tramite un ciclo for con passo 2. L'indice i incrementa di due ad ogni passo per considerare solo gli elementi 2x2 utili per la determinazione degli autovalori.

$$J_n = \begin{pmatrix} \# & \# & 0 & 0 & 0 & 0 \\ \# & \# & * & 0 & 0 & 0 \\ 0 & * & \# & \# & 0 & 0 \\ 0 & 0 & \# & \# & * & 0 \\ 0 & 0 & 0 & * & \# & \# \\ 0 & 0 & 0 & 0 & \# & \# \end{pmatrix}$$

Nella matrice di sopra sono evidenziati i quadrati 2x2 da considerare tramite dei “#” e gli elementi della sottodiagonale da saltare tramite dei “*”. Considerarli in una matrice 6x6 come quella dell'esempio comporterebbe l'avere in uscita un vettore con 8 autovalori invece di 6. Il ciclo for quindi controlla, come visto con passo pari a due, gli elementi sottodiagonali: nel caso in cui siano diversi da zero calcola gli autovalori delle sottomatrici 2x2 associate tramite la funzione eig di matlab e aggiorna il vettore di uscita lambda; nel caso in cui siano uguali a zero siamo in presenza di un autovalore non accoppiato, nel caso di laguerre e legendre sempre pari a zero, che viene saltato dal ciclo e aggiunto successivamente “manualmente” con la condizione if che verifica la compatibilità della dimensione di lambda.
L'efficacia nella determinazione degli autovalori è stata testata su una matrice simmetrica ottenendo i seguenti risultati:

```
>> B = rand(8);
>> A = B * B'

A =

   1.4415   1.9650   1.1204   1.2465   1.6987   1.1384   1.8757   0.7472
   1.9650   4.0660   2.3912   3.0689   3.4834   2.1653   3.3453   2.2743
   1.1204   2.3912   2.3849   2.2521   2.2597   1.6847   2.2206   1.6974
   1.2465   3.0689   2.2521   3.4631   2.9193   1.8167   2.2064   2.1898
   1.6987   3.4834   2.2597   2.9193   3.8049   2.4242   3.1991   2.4919
   1.1384   2.1653   1.6847   1.8167   2.4242   2.0589   2.1928   1.2502
   1.8757   3.3453   2.2206   2.2064   3.1991   2.1928   3.3550   1.9581
   0.7472   2.2743   1.6974   2.1898   2.4919   1.2502   1.9581   2.2941
```

```
>> Autovalori = miautovalqr(A)

Autovalori =

   0.0133
   0.0601
   0.2259
   0.6368
   0.8551
   0.8558
   1.7044
  18.5170

>> Verifica = eig(A)

Verifica =

   0.0133
   0.0601
   0.2259
   0.6368
   0.8434
   0.8675
   1.7044
  18.5170
```

Famiglie di polinomi di Legendre, Hermite, Laguerre.

Si è deciso di determinare gli zeri dei polinomi appartenenti a tre delle più comuni famiglie: Legendre, Hermite e Laguerre.
Il polinomio di Legendre ha peso pari a uno e intervallo di integrazione [-1;1].
Viene definito come:

$$P_0(x)=1,\quad P_1(x)=x,$$

$$P_{n+1}(x)=x\cdot P_n(x)-\frac{n^2}{4\cdot n^2-1}\cdot P_{n-1}(x),\qquad con\, n\geq 1$$

da cui riferendosi alla forma generale

$$P_{n+1}(x)=(x-\alpha_n)\cdot P_n(x)-\beta_n^2\cdot P_{n-1}(x)$$

si estrapola :

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

Analogamente si hanno i polinomi di Hermite: $peso\, w(x)=e^{-x^2},\quad [a,b]=(-\infty,\infty)$

$$H_0(x)=1,\quad H_{-1}(x)=0,$$

$$H_{n+1}(x)=x\cdot H_n(x)-\frac{n}{2}\cdot H_{n-1}(x),\qquad con\quad n\geq 0$$

da cui riferendosi alla forma generale

$$P_{n+1}(x)=(x-\alpha_n)\cdot P_n(x)-\beta_n^2\cdot P_{n-1}(x)$$

si estrapola :

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

E per Laguerre: $peso\, w(x)=e^{-x}, \quad [a,b]=(0,\infty)$

$L_0(x)=1, \quad L_{-1}(x)=0,$

$L_{n+1}(x)=(x-2\cdot n-1)\cdot L_n(x)-n^2\cdot L_{n-1}(x), \qquad con \quad n\geq 0$

da cui riferendosi alla forma generale

$P_{n+1}(x)=(x-\alpha_n)\cdot P_n(x)-\beta_n^2\cdot P_{n-1}(x)$

si estrapola :

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

L'algoritmo per la determinazione degli zeri prevede quindi, caso per caso, il calcolo degli α e β, la costruzione della matrice Jn e l'applicazione dell'algoritmo QR modificato a quest'ultima.

```
%Polinomi
%%
clc
'Calcolo dei nodi di una formula Gaussiana di integrazione'
met = input('Scegliere la famiglia di polinomi: Legendre(1), Laguerre(2), Hermite(3) ');
np = input('Inserisci il grado del polinomio: ');
if met == 1
   n1 = [1:(np-1)]';%np-1 in quanto sono presenti n-1 beta e n alfa nella matrice Jn
   beta1 = n1./sqrt(4*n1.*n1-1);%Determina i beta
   Jn1 = diag(beta1,1)+diag(beta1,-1)%costruisce Jn
   l1 = miautovalqr(Jn1);% determina gli autovalori
   'Le radici del polinomio sono'
   l1
elseif met==2
   n2a = [1:np]';
   n2b = [1:(np-1)]';
   alfa2 = n2a.*2+1;
   beta2 =sqrt(n2b);
   Jn2 = diag(alfa2)+diag(beta2,-1)+diag(beta2,1);
   l2 = miautovalqr(Jn2);
   'Le radici del polinomio sono'
   l2
elseif met == 3
   n3 = [1:(np-1)]';
   beta3 = sqrt(n3./2);
   Jn3 = diag(beta3,1)+diag(beta3,-1)
   l3 = miautovalqr(Jn3);
   'Le radici del polinomio sono'
   l3
else
   error('errore')

end
```

Di seguito, e nell'ordine, un esempio di applicazione dell'algoritmo con legendre, laguerre ed hermite per “n” pari a sette:

Legendre

```
Calcolo dei nodi di una formula Gaussiana di integrazione

Scegliere la famiglia di polinomi: Legendre(1), Laguerre(2), Hermite(3) 1
Inserisci il grado del polinomio: 7

ans =

Le radici del polinomio sono


Radici =

   -0.9491
   -0.7415
   -0.4058
         0
    0.4058
    0.7415
    0.9491

>> Jn1

Jn1 =

         0    0.5774         0         0         0         0         0
    0.5774         0    0.5164         0         0         0         0
         0    0.5164         0    0.5071         0         0         0
         0         0    0.5071         0    0.5040         0         0
         0         0         0    0.5040         0    0.5025         0
         0         0         0         0    0.5025         0    0.5017
         0         0         0         0         0    0.5017         0

>> beta1

beta1 =

    0.5774
    0.5164
    0.5071
    0.5040
    0.5025
    0.5017
```

Laguerre

```
Calcolo dei nodi di una formula Gaussiana di integrazione

Scegliere la famiglia di polinomi: Legendre(1), Laguerre(2), Hermite(3) 2
Inserisci il grado del polinomio: 7

ans =

Le raici del polinomio sono


Radici =

    2.5000
    4.5000
    6.5010
    8.5210
   10.6860
   13.3244
   16.9676

>> Jn2

Jn2 =

    3.0000    1.0000         0         0         0         0         0
    1.0000    5.0000    1.4142         0         0         0         0
         0    1.4142    7.0000    1.7321         0         0         0
         0         0    1.7321    9.0000    2.0000         0         0
         0         0         0    2.0000   11.0000    2.2361         0
         0         0         0         0    2.2361   13.0000    2.4495
         0         0         0         0         0    2.4495   15.0000

>> beta2

beta2 =

    1.0000
    1.4142
    1.7321
    2.0000
    2.2361
    2.4495
```

Hermite

```
Calcolo dei nodi di una formula Gaussiana di integrazione

Scegliere la famiglia di polinomi: Legendre(1), Laguerre(2), Hermite(3) 3
Inserisci il grado del polinomio: 7

ans =

Le radici del polinomio sono


Radici =

   -2.6520
   -1.6736
   -0.8163
         0
    0.8163
    1.6736
    2.6520

>> Jn3

Jn3 =

         0    0.7071         0         0         0         0         0
    0.7071         0    1.0000         0         0         0         0
         0    1.0000         0    1.2247         0         0         0
         0         0    1.2247         0    1.4142         0         0
         0         0         0    1.4142         0    1.5811         0
         0         0         0         0    1.5811         0    1.7321
         0         0         0         0         0    1.7321         0

>> beta3

beta3 =

    0.7071
    1.0000
    1.2247
    1.4142
    1.5811
    1.7321
```