Quadratura numerica

12 dicembre 2007

1 Introduzione

Un classico problema dell'analisi numerica è quello di calcolare l'*integrale definito* di una funzione f nell'intervallo $[a,b]$ cioè

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

dove supporremo per il momento che

1. $[a,b] \subset [\alpha,\beta] \subseteq \mathbb{R}$ sia un intervallo chiuso e limitato;
2. la funzione $f : [\alpha,\beta] \to \mathbb{R}$ sia sufficientemente regolare.

Successivamente indeboliremo queste condizioni.
La nostra intenzione è di di approssimare $I(f)$ come

$$I(f) \approx Q_N(f) := \sum_{i=1}^{N} w_i f(x_i) \tag{1}$$

I termini w_i e $x_i \in [\alpha,\beta]$ sono detti rispettivamente pesi e nodi. Notiamo che consideriamo $[a,b] \subset [\alpha,\beta] \subseteq \mathbb{R}$: questo perchè in alcune formule si possono considerare nodi esterni al dominio di integrazione.

1.1 Formule di Newton-Cotes

Le prime *regole* di questo tipo sono dette di *Newton-Cotes* chiuse (cf. [11, p.336]) e si ottengono integrando l'interpolante di f in nodi equispaziati

$$x_i = a + \frac{(i-1)\,(b-a)}{N-1},\ i = 1,\ldots,N.$$

Alcune classiche regole sono:

1. *regola del trapezio*

$$I(f) \approx S_1(f) := S_1(f,a,b) := \frac{(b-a)\,(f(a)+f(b))}{2}$$

Figure 1: Cavalieri e Simpson.

avente *grado di precisione* 1, cioè esatta per polinomi di grado inferiore o uguale a 1; si può dimostrare (con un po' di fatica) dal teorema del resto per l'interpolazione polinomiale (cf. [2, p.132]) che l'errore della regola del trapezio [20] è

$$E_1(f) := I(f) - S_1(f) = \frac{-h^3}{12} f^{(2)}(\xi)$$

per qualche $\xi \in (a,b)$;

2. *regola di Cavalieri-Simpson*

$$I(f) \approx S_3(f) := S_3(f,a,b) := \frac{b-a}{6}\left[f(a) + 4f(\frac{a+b}{2}) + f(b)\right]$$

avente *grado di precisione* 3, cioè esatta per polinomi di grado inferiore o uguale a 3; si può dimostrare (non facile!) che l'errore della regola di Cavalieri-Simpson [21] è

$$E_3(f) := I(f) - S_3(f) = \frac{-h^5}{90} f^{(4)}(\xi),\ h = \frac{b-a}{2}$$

per qualche $\xi \in (a,b)$;

Per ulteriori dettagli si confronti [1, p.252-258], [11, p.333-336]. Qualora le funzioni da integrare non siano sufficientemente derivabili, una stima dell'errore viene fornita dalle formule dell'errore via nucleo di Peano ([1, p.259]). Ricordiamo che per $N > 8$ le formule di Newton-Cotes chiuse hanno pesi di segno diverso e sono instabili dal punto di vista della propagazione degli errori (cf. [2, p.196]).

1.2 Formule di Newton-Cotes composte

Si suddivida l'intervallo (chiuso e limitato) $[a,b]$ in N subintervalli $T_j = [x_j, x_{j+1}]$ tali che $x_j = a + jh$ con $h = (b-a)/N$. Dalle proprietà dell'integrale

$$\int_a^b f(x)\,dx = \sum_{j=0}^{N-1} \int_{x_j}^{x_{j+1}} f(x)\,dx \approx \sum_{j=0}^{N-1} S(f, x_j, x_{j+1}) \tag{2}$$

dove S è una delle regole di quadratura finora esposte. Le formule descritte in (2) sono dette *composte*. Due casi particolari sono

1. *formula composta dei trapezi*

$$S_1^{(c)} := h\left[\frac{f(x_0)}{2} + f(x_1) + \ldots + f(x_{N-1}) + \frac{f(x_N)}{2}\right] \tag{3}$$

il cui errore è

$$E_1^{(c)}(f) := I(f) - S_1^{(c)}(f) = \frac{-(b-a)}{12} h^2 f^{(2)}(\xi),\ h = \frac{(b-a)}{N}$$

per qualche $\xi \in (a,b)$;

2. *formula composta di Cavalieri-Simpson* fissati il numero N di subintervalli e i punti $x_k = a + kH/2$ dove

$$H = \frac{b-a}{N}$$

sia

$$I(f) \approx S_3^{(c)}(f) := \frac{H}{6}\left[f(x_0) + 2\sum_{r=1}^{N-1} f(x_{2r}) + 4\sum_{s=0}^{N-1} f(x_{2s+1}) + f(x_{2N})\right]; \tag{4}$$

il cui errore è

$$E_3^{(c)}(f) := I(f) - S_3^{(c)}(f) = \frac{-(b-a)}{180}\left(\frac{H}{2}\right)^4 f^{(4)}(\xi)$$

per qualche $\xi \in (a,b)$.

1.3 Implementazione Matlab di alcune formule composte

Mostreremo di seguito un'implementazione in Matlab/Octave della formula composta dei trapezi e di Cavalieri-Simpson.

```
function [x,w]=trapezi_composta(N,a,b)

% FORMULA DEI TRAPEZI COMPOSTA.

% INPUT:
% N: NUMERO SUBINTERVALLI.
% a, b: ESTREMI DI INTEGRAZIONE.

% OUTPUT:
% x: NODI INTEGRAZIONE.
% w: PESI INTEGRAZIONE (INCLUDE IL PASSO!).

h=(b-a)/N;              % PASSO INTEGRAZIONE.
x=a:h:b; x=x';          % NODI INTEGRAZIONE.
```

```
w=ones(N+1,1);          % PESI INTEGRAZIONE.
w(1)=0.5; w(N+1)=0.5;
w=w*h;
```

La funzione `trapezi_composta` appena esposta calcola i nodi e i pesi della omonima formula composta.
L'unica difficoltà del codice consiste nel calcolo dei pesi `w`. Essendo per loro definizione

$$I(f) \approx S_1^c(f) := \sum_{i=0}^{N} w_i f(x_i) \tag{5}$$

come pure per (3)

$$S_1^{(c)} := h\left[\frac{f(x_0)}{2} + f(x_1) + \ldots + f(x_{N-1}) + \frac{f(x_N)}{2}\right] \tag{6}$$

deduciamo che $w_0 = w_N = h/2$ mentre $w_1 = \ldots = w_{N-1} = h$, cosa che giustifica le ultime linee della function `trapezi_composta`.

Si potrebbe usare il comando Matlab `trapz` nella sua implementazione

```
>> help trapz

 TRAPZ  Trapezoidal numerical integration.
    Z = TRAPZ(Y) computes an approximation of the integral of Y via
    the trapezoidal method (with unit spacing).  To compute the integral
    for spacing different from one, multiply Z by the spacing increment.

    For vectors, TRAPZ(Y) is the integral of Y.
    ... ... ...
```

e sostituire la parte relativa al calcolo dei pesi con

```
I=h*trapz(fx);
```

Vediamone i dettagli in Matlab (versione 6.1) per il calcolo di

$$\int_0^1 \sin(x)dx = -\cos(1) - (-\cos(0)) = -\cos(1) + 1 \approx 0.45969769413186.$$

sia utilizzando la funzione `trapezi_composta` che `trapz`

```
>> format long;
>> [x,w]=trapezi_composta(10,0,1);
>> fx=sin(x);
>> I_trapezi_composta=w'*fx
I_trapezi_composta =
   0.45931454885798
```

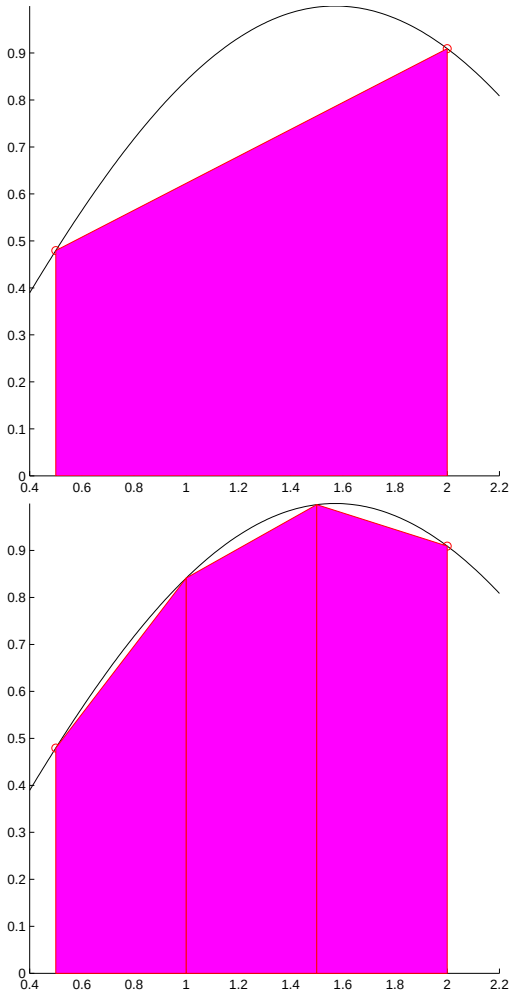

Figure 2: Formula dei trapezi e dei trapezi composta per il calcolo di $\int_{0.5}^{2} \sin(x)\,dx$ (area in magenta).

```
>> h=(1-0)/10;
>> I_trapz=h*trapz(fx)
I_trapz =
   0.45931454885798
>>
```

Di conseguenza per implementare la regola è del tutto equivalente usare la function `trapezi_composta` o `trapz`.
Evitiamo il diretto utilizzo di `trapz` perchè non presente in alcune vecchie versioni di Octave (ma non nella più recente 2.1.73).

Per quanto riguarda la formula di Cavalieri-Simpson composta

```
function [x,w]=simpson_composta(N,a,b)

% FORMULA DI SIMPSON COMPOSTA.

% INPUT:
% N: NUMERO SUBINTERVALLI.
% a, b: ESTREMI DI INTEGRAZIONE.

% OUTPUT:
% x: INTEGRAZIONE.
% w: PESI INTEGRAZIONE (INCLUDE IL PASSO!).

h=(b-a)/N;              % AMPIEZZA INTERVALLO.
x=a:(h/2):b; x=x';      % NODI INTEGRAZIONE.

w=ones(2*N+1,1);            % PESI INTEGRAZIONE.
w(3:2:2*N-1,1)=2*ones(length(3:2:2*N-1),1);
w(2:2:2*N,1)=4*ones(length(2:2:2*N),1);
w=w*h/6;
```

Similmente alla routine per il calcolo dei nodi e i pesi di `trapezi_composta`, le ultime righe sono le più difficili da capire, ma un confronto con (5) e (4) ne spiega il significato.

Una volta noti il vettore (colonna) x dei nodi e w dei pesi di integrazione, se la funzione f è richiamata da un m-file *f.m*, basta

```
fx=f(x);                  % VALUT. FUNZIONE.
I=w'*fx;                  % VALORE INTEGRALE.
```

per calcolare il risultato fornito dalla formula di quadratura composta.
Ricordiamo che se $\mathbf{w} = (\mathbf{w}_k)_{k=0,\dots,N} \in \mathbb{R}^{N+1}$, $\mathbf{fx} = (f(x_k))_{k=0,\dots,N} \in \mathbb{R}^{N+1}$ sono due vettori colonna allora il prodotto scalare

$$\mathbf{w} * \mathbf{fx} := \sum_{k=0}^{N} \mathbf{w}_k \cdot \mathbf{fx}_k := \sum_{k=0}^{N} \mathbf{w}_k \cdot f(x_k)$$

si scrive in Matlab/Octave come `w'*fx`. Osserviamo che dimensionalmente il prodotto di un vettore $1\times(N+1)$ con un vettore $(N+1)\times 1$ dà uno scalare (cioè un vettore 1×1).

Calcoliamo numericamente utilizzando le formule composte sopra citate

$$\int_{-1}^{1} x^{20}dx = (1^{21}/21) - (-1)^{21}/21 = 2/21 \approx 0.09523809523810.$$

A tal proposito scriviamo il seguente codice `demo_composte`.

```
N=11;                                %SCEGLIERE DISPARI.
a=-1; b=1;

N_trap=N-1;
[x_trap,w_trap]=trapezi_composta(N_trap,a,b);
fx_trap=x_trap.^20;                  % VALUT. FUNZIONE.
I_trap=w_trap'*fx_trap;              % TRAPEZI COMPOSTA.

N_simpson=(N-1)/2;
[x_simp,w_simp]=simpson_composta(N_simpson,a,b)
fx_simp=x_simp.^20;                  % VALUT. FUNZIONE.
I_simp=w_simp'*fx_simp;              % SIMPSON COMPOSTA.

fprintf('\n \t [TRAPEZI COMPOSTA] [PTS]: %4.0f', length(x_trap));
fprintf('\n \t [TRAPEZI COMPOSTA] [RIS]: %14.14f', I_trap);

fprintf('\n \t [SIMPSON COMPOSTA] [PTS]: %4.0f', length(x_simp));
fprintf('\n \t [SIMPSON COMPOSTA] [RIS]: %14.14f', I_simp);
```

ottenendo

```
[TRAPEZI COMPOSTA] [PTS]:   11
[TRAPEZI COMPOSTA] [RIS]: 0.20462631505024
[SIMPSON COMPOSTA] [PTS]:   11
[SIMPSON COMPOSTA] [RIS]: 0.13949200364447
```

Si può vedere che usando formule di tipo gaussiano (cf. [18], [19]) o di tipo Clenshaw-Curtis (cf. [16], [13], [14]) a parità di valutazioni della funzione f avremmo ottenuto

```
[GAUSS-LEGENDRE  ]: 0.095238095238095649
[CLENSHAW-CURTIS ]: 0.094905176204004307
```

col costo aggiuntivo di dover calcolare tramite complicati algoritmi i pesi e i nodi di Gauss o i nodi di Clenshaw-Curtis via un IFFT [17].

1.4 Facoltativo: Formule gaussiane

Nelle formule interpolatorie di Newton-Cotes (come ad esempio la regola del Trapezio o di Cavalieri-Simpson) i nodi $x_1,\ldots,x_N$ sono equispaziati e il grado di precisione δ è al massimo uguale al numero di nodi N. Ci si domanda se sia possibile

scegliere nodi $x_1, \dots, x_N$ e pesi $w_1, \dots, w_N$ per cui le relative formule di quadratura abbiano grado di precisione $\delta > N$ (come ad esempio per la regola di Cavalieri-Simpson).

D'altro canto, se $w : [a,b] \to \mathbb{R}$ (non necessariamente limitato) è una funzione peso, cioè (cf. [1, p.206, p.270])

1. w è nonnegativa in (a,b);
2. w è integrabile in $[a,b]$;
3. esista e sia finito
$$\int_a^b |x|^n w(x)\,dx$$
per ogni $n \in \mathbb{N}$;
4. se
$$\int_a^b g(x)\,w(x)\,dx$$
per una qualche funzione nonnegativa g allora $g \equiv 0$ in (a,b).

Tra gli esempi più noti

1. *Legendre*: $w(x) \equiv 1$ in $[a,b]$ limitato;
2. *Jacobi*: $w(x) = (1-x)^\alpha (1+x)^\beta$ in $(-1,1)$ per $\alpha, \beta \geq -1$;
3. *Chebyshev*: $w(x) = \frac{1}{\sqrt{1-x^2}}$ in $(-1,1)$;
4. *Laguerre*: $w(x) = \exp(-x)$ in $[0,\infty)$;
5. *Hermite*: $w(x) = \exp(-x^2)$ in $(-\infty,\infty)$;

Si supponga ora di dover calcolare per qualche funzione $f : (a,b) \to \mathbb{R}$

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

Il problema è evidentemente più generale di quello in (1), visto che l'integranda $f\,w$ non é necessariamente continua in $[a,b]$ (si consideri ad esempio il peso di Chebyshev) oppure può succedere che l'intervallo sia illimitato come nel caso del peso di Laguerre o Hermite.

Esistono nuovamente $x_1, \dots, x_N$ e pesi $w_1, \dots, w_N$ (detti di *Gauss-nome funzione peso*) per cui le relative formule di quadratura abbiano grado di precisione $\delta > N$, cioè calcolino esattamente
$$\int_a^b p_m(x)\,w(x)\,dx$$
per $m > N$?

La risposta è affermativa, come si può vedere in [1, p.272]. Per ogni $N \leq 1$ esistono e sono unici dei nodi $x_1, \dots, x_N$ e pesi $w_1, \dots, w_N$ per cui il grado di precisione sia $2N-1$. Eccetto che in pochi casi (come ad esempio per la funzione peso di Chebyshev), non esistono formule esplicite per l'individuazione di tali nodi e pesi. Una volta venivano tabulati, oggi si consiglia di applicare del software che si può trovare ad esempio nella pagina di Walter Gautschi

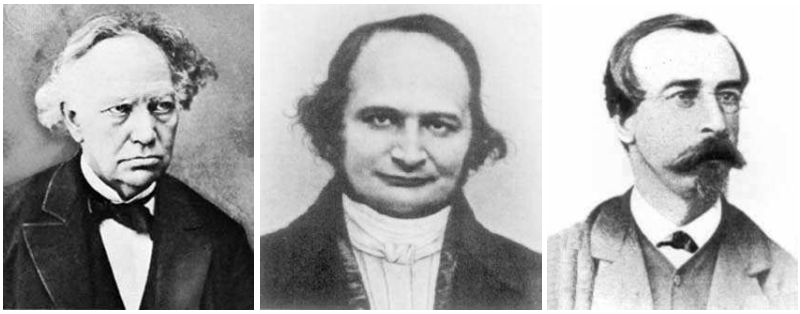

Figure 3: Hermite, Jacobi e Laguerre.

http://www.cs.purdue.edu/people/wxg

Fissato un peso, ad esempio quello di Jacobi, si cercano per prima cosa i *coefficienti di ricorrenza*, che possono essere calcolati con l' m-file *r_jacobi.m* nel sito di W. Gautschi. La sintassi è la seguente

```
ab=r_jacobi(N,a,b)
```

Come si vede dai commenti al file

```
% R_JACOBI Recurrence coefficients for monic Jacobi polynomials.
%
%    ab=R_JACOBI(n,a,b) generates the first n recurrence
%    coefficients for monic Jacobi polynomials with parameters
%    a and b. These are orthogonal on [-1,1] relative to the
%    weight function w(t)=(1-t)^a(1+t)^b. The n alpha-coefficients
%    are stored in the first column, the n beta-coefficients in
%    the second column, of the nx2 array ab. The call ab=
%    R_JACOBI(n,a) is the same as ab=R_JACOBI(n,a,a) and
%    ab=R_JACOBI(n) the same as ab=R_JACOBI(n,0,0).
%
%    Supplied by Dirk Laurie, 6-22-1998; edited by Walter
%    Gautschi, 4-4-2002.
```

a, b corrispondono rispettivamente all'α e β delle formule di Jacobi (e non agli estremi di integrazione!). I coefficienti di ricorrenza sono immagazzinati nella variabile *ab*. A questo punto si chiama la funzione *gauss.m* (reperibile nuovamente presso lo stesso sito di W. Gautschi)

```
% GAUSS Gauss quadrature rule.
%
%    Given a weight function w encoded by the nx2 array ab of the
%    first n recurrence coefficients for the associated orthogonal
```

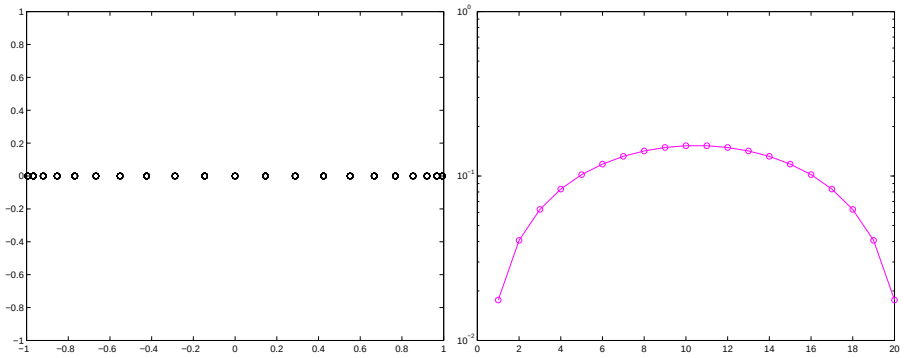

Figure 4: Grafico che illustra la distribuzione dei 20 nodi e i 20 pesi di Gauss-Legendre nell'intervallo $[1,1]$

.

```
%    polynomials, the first column of ab containing the n alpha-
%    coefficients and the second column the n beta-coefficients,
%    the call xw=GAUSS(n,ab) generates the nodes and weights xw of
%    the n-point Gauss quadrature rule for the weight function w.
%    The nodes, in increasing order, are stored in the first
%    column, the n corresponding weights in the second column, of
%    the nx2 array xw.
%
function xw=gauss(N,ab)
N0=size(ab,1); if N0<N, error('input array ab too short'), end
J=zeros(N);
for n=1:N, J(n,n)=ab(n,1); end
for n=2:N
  J(n,n-1)=sqrt(ab(n,2));
  J(n-1,n)=J(n,n-1);
end
[V,D]=eig(J);
[D,I]=sort(diag(D));
V=V(:,I);
xw=[D ab(1,2)*V(1,:)'.^2];
```

che per un certo N, uguale al massimo al numero di righe della matrice di coefficienti di ricorrenza ab, fornisce nodi x e pesi w immagazzinati in una matrice che ha quale prima colonna x e quale seconda colonna w. La descrizione di perchè tale software fornisca il risultato desiderato è complicata ma può essere trovata nella monografia di W. Gautschi sui polinomi ortogonali, per Acta Numerica.

Conseguentemente per trovare i nodi e pesi relativi alla formula di Gauss-Legendre in (a,b), che è una formula di tipo Gauss-Jacobi per $\alpha = 0$ e $\beta = 0$, nonchè calcolare con essi degli integrali di una funzione f, si procede come segue.

1.5 Facoltativo: Una demo di Gauss-Jacobi

Salviamo nel file `integrazione_gauss_jacobi.m`

```
function [I_jac,x_jac,w_jac]=integrazione_gauss_jacobi(N,ajac,bjac,a,b,f)

% INPUT:
%
% N: GRADO DI GAUSS-JACOBI. SE SI VUOLE CALCOLARE I(f) CON
%                 I(f)=int_a^b f(x) dx
%    PORRE ajac=0, bjac=0 (FUNZIONE PESO DI LEGENDRE w(x)=1).
% ajac, bjac: PARAMETRI DI GAUSS-JACOBI, CIOE' SI INTEGRA:
%         I(fw)=int_a^b f(x) w(x) dx
%    CON w(x)=(1-x)^ajac (1+x)^bjac.
% a,b: ESTREMI INTERVALLO INTEGRAZIONE.
% f: FUNZIONE DA INTEGRARE.
%
% OUTPUT:
%
% I_jac: VALORE DI I(fw) APPROSSIMATO DA gauss_jacobi. PER ajac=0,
%        bjac=0 CORRISPONDE A I(f).
% x_jac: NODI GAUSS-JACOBI.
% w_jac: PESI GAUSS-JACOBI.

% ROUTINES ESTERNE: r_jacobi, gauss.


ab_jac=r_jacobi(N,ajac,bjac);        % TERM. RICORSIVI.
xw_jac=gauss(N,ab_jac);              % NODI E PESI IN MATRICE.
x_jac=xw_jac(:,1);                   % NODI GAUSS-LEGENDRE [-1,1].
x_jac_ab=((a+b)/2)+((b-a)/2)*x_jac;  % NODI GAUSS-LEGENDRE [a,b].
w_jac=xw_jac(:,2);                   % PESI GAUSS-LEGENDRE [-1,1].
w_jac_ab=((b-a)/2)*w_jac;            % PESI GAUSS-LEGENDRE [a,b].
fx_jac_ab=feval(f,x_jac_ab);         % VALUTAZIONE FUNZIONE.
I_jac=w_jac_ab'*fx_jac_ab;           % VALORE INTEGRALE.
```

e nel file *f.m* delle funzioni su cui effettueremo dei test. Un esempio è

```
function fx=f(x)

fx=x.^20;

% ALCUNE FUNZIONI CHE FANNO PARTE DEL SET STUDIATO NELL'ARTICOLO:
% "IS GAUSS QUADRATURE BETTER THAN CLENSHAW-CURTIS?"
% DI L.N. TREFETHEN.
% fx=exp(x);

% fx=exp(-x.^2);
% fx=1./(1+16*(x.^2));
```

```
% fx=exp(-x.^(-2));
% fx=abs(x); fx=fx.^3;
% fx=x.^(0.5);
% fx=exp(x).*(sqrt(1-x));
```

Altre funzioni test [13] sono

```
fx = x.^20
fx = exp(x)
fx = exp(-x.^2)
fx = 1./(1+16*(x.^2))
fx = exp(-x.^(-2))
fx = (abs(x)).^3;
fx=x.^(0.5);
fx=exp(x).*(sqrt(1-x));
```

con $a = -1$, $b = 1$ e per cambiare il tipo di funzione, basta modificare la posizione dei caratteri %.

Scaricando dalla pagina web del corso i files `integrazione_gauss_jacobi.m`, `f.m` e dalla pagina di W. Gautschi `r_jacobi.m` e `gauss.m` possiamo fare alcuni esperimenti.
Una lista di possibili tests è la seguente:

$$\int_{-1}^{1} x^2 0\,dx = 2/21 \approx 0.095238095238096801 \tag{7}$$

$$\int_{-1}^{1} e^x\,dx = e - e^{-1} \approx 2.3504023872876032 \tag{8}$$

$$\int_{-1}^{1} e^{-x^2}\,dx = \mathrm{erf}(1)\cdot\sqrt{(\pi)} \approx 1.4936482656248538 \tag{9}$$

$$\int_{-1}^{1} 1/(1+16\cdot x^2)\,dx = 1/2\cdot\mathrm{atan}(4) \approx 0.66290883183401628 \tag{10}$$

$$\int_{-1}^{1} e^{-1/x^2}\,dx = 2\cdot e^{-1} + 2\cdot e^{-1}\cdot\mathrm{erf}(1)\cdot\pi^{1/2}\cdot e^{-1} - 2\cdot\pi^{1/2} \tag{11}$$

$$\approx 0.17814771178156086 \tag{12}$$

$$\int_{-1}^{1} |x|^3\,dx = 1/2 \tag{13}$$

$$\int_{-1}^{1} \sqrt{x}\,dx = 2/3 \tag{14}$$

$$\int_{-1}^{1} e^x\cdot\sqrt{(1-x)}\,dx = -2^{1/2}\cdot e^{-1} + 1/2\cdot e * \pi^{1/2} * \mathrm{erf}(2^{1/2}) \tag{15}$$

$$\approx 1.7791436546919095 \tag{16}$$

Alcune note prima di cominciare.

1. Si osservi che per ottenere i nodi e pesi di Gauss-Legendre in (a,b) da quelli di Gauss-Jacobi in $(-1,1)$ abbiamo effettuato uno *scaling*: se $x_{i,[-1,1]}$ sono i nodi di Gauss-Jacobi in $[-1,1]$ allora i nodi di Gauss-Jacobi $x_{i,[a,b]}$ in $[a,b]$ sono

$$x_{i,[a,b]} = ((a+b)/2) + ((b-a)/2)\dot{x}_{i,[-1,-1]};$$

2. se $w_{i,[-1,1]}$ sono i pesi di Gauss-Jacobi in $[-1,1]$ allora i pesi di Gauss-Jacobi $w_{i,[a,b]}$ in $[a,b]$ sono

$$w_{i,[a,b]} = ((b-a)/2) \cdot w_{i,[-1,1]}.$$

   L'idea è la seguente. Dovendo le formule essere esatte per le costanti come la funzione $f \equiv 1$, nel caso della funzione peso di Legendre $w\ equi v 1$

$$\sum_i w_{i,[a,b]} = \int_a^b w(x)\,dx = b-a$$

   mentre nel caso di $a=-1$, $b=1$ derivante dalla funzione peso di Jacobi abbiamo

$$\sum_i w_{i,[-1,1]} = \int_{-1}^1 w(x)\,dx = 2;$$

   si ha quindi l'intuizione che i pesi in (a,b) siano quelli in $(-1,1)$ moltiplicati per $\frac{b-a}{2}$.

3. Nonostante l'introduzione riguardante nodi e pesi in $[a,b]$ non necessariamente uguale a $[-1,1]$, alla fine eseguiremo test esclusivamente in quest'ultimo intervallo. Qualora necessario, basta aggiungere nuove funzioni matematiche al file `f.m`, modificare adeguatamente a, b, fornire il relativo risultato esatto in `exact_results.m` e testare la formula gaussiana. Per avere il risultato con alta precisione si usi la funzione `quadl.m` o `quad8.m` di Matlab oppure `integrazione_gauss_jacobi.m` con $N = 500$;

4. l'intervallo tipico di Gauss-Legendre è $[a,b]$ (chiuso), mentre per Gauss-Jacobi l'intervallo è tipicamente (a,b) (aperto),poichè in generale gli esponenti della funzione peso di Jacobi possono essere negativi; per Gauss-Legendre il problema non sussiste, visto che la funzione peso è continua, e i punti a, b sono un insieme *trascurabile*.

Se ora testiamo il primo esempio, cioè il calcolo di

$$\int_{-1}^1 x^{20}\,dx \approx 0.09523809523809523300$$

otteniamo

```
>> format long;
>> N=11; ajac=0; bjac=0; a=-1; b=1;
```

```
>> [I_jac,x_jac,w_jac]=integrazione_gauss_jacobi(N,ajac,bjac,a,b,@f);
>> I_jac
I_jac =
   0.09523809523810
>> length(x_jac)
ans =
    11
>> fprintf('\n \t [GAUSS-LEGENDRE]: \%15.20f',I_jac);

          [GAUSS-LEGENDRE]: 0.09523809523809564900
>> 0.09523809523809523300-0.09523809523809564900
ans =
   -4.163336342344337e-016
>>
```

1.6 Facoltativo: Una implementazione di Clenshaw-Curtis

La quadrature di tipo Clenshaw-Curtis si basa su un'espansione dell'integranda in termini di polinomi ortogonali di Chebyshev (cf. [16], [13], [14]).

Usiamo una implementazione di Waldvogel [14]

```
function [x,w]=clenshaw_curtis(n,a,b)

N=[1:2:n-1]'; l=length(N); m=n-l; K=[0:m-1]';

g0=-ones(n,1); g0(1+l)=g0(1+l)+n; g0(1+m)=g0(1+m)+n;
g=g0/(n^2-1+mod(n,2));
end_N=length(N);
v0=[2./N./(N-2); 1/N(end_N); zeros(m,1)];
end_v0=length(v0);
v2=-v0(1:end_v0-1)-v0(end_v0:-1:2);
wcc=ifft(v2+g); weights=[wcc;wcc(1,1)];
k=0:n; nodes=(cos(k*pi/n))';

x=(a+b)/2+((b-a)/2)*nodes;
w=weights*((b-a)/2);
```

Noti i nodi e i pesi di Clenshaw-Curtis, definiamo la function `integrazione_clenshaw_curtis`

```
function [I_cc,x_cc,w_cc]=integrazione_clenshaw_curtis(N_cc,a,b,f)

% INTEGRAZIONE DI CLENSHAW-CURTIS.

% INPUT:
% N_cc: GRADO DI CLENSHAW-CURTIS, CIOE' NUMERO DI NODI MENO UNO.
% a, b: ESTREMI DI INTEGRAZIONE.
% f   : FUNZIONE DA INTEGRARE.
```

```
% OUTPUT:
% I_cc: APPROSSIMAZIONE DELL'INTEGRALE CON FORMULA DI CLENSHAW-CURTIS.
% x_cc: NODI DI CLENSHAW-CURTIS.
% w_cc: PESI DI CLENSHAW-CURTIS.

% ROUTINES ESTERNE: clenshaw_curtis.

[x_cc,w_cc]=clenshaw_curtis(N_cc,a,b);  % NODI E PESI DI CL.-CURTIS.
fx_cc=feval(f,x_cc);                    % VALUTAZIONE FUNZIONE.
I_cc=w_cc'*fx_cc;                       % VALORE INTEGRALE.
```

Applichiamola ora il calcolo di

$$\int_{-1}^{1} x^{20}\,dx \approx 0.09523809523809523300.$$

```
>> N_cc=10; a=-1; b=1;
>> [I_cc,x_cc,w_cc]=integrazione_clenshaw_curtis(N_cc,a,b,@f);
>> fprintf('\n \t [CLENSHAW-CURTIS]: %15.20f',I_cc);

          [CLENSHAW-CURTIS]: 0.09490517620400430700
>> 0.09490517620400430700-2/21
ans =
   -3.329190340909255e-004
>> length(x_cc)
ans =
    11
>>
```

La formula ha 11 nodi e pesi, ed ha un'errore assoluto di circa $3.32\cdot 10^{-4}$. Osserviamo il passaggio della funzione f attraverso `@f`.

Dalla figura con i 6 grafici di errore, si vedono le performances delle formule sopracitate quando applicate a calcolare

$$\int_{-1}^{1} x^2 0\,dx \quad \text{(a sinistra, prima riga)} \tag{17}$$

$$\int_{-1}^{1} e^{x}\,dx \quad \text{(a destra, prima riga)} \tag{18}$$

$$\int_{-1}^{1} e^{-x^2}\,dx \quad \text{(a sinistra, seconda riga)} \tag{19}$$

$$\int_{-1}^{1} 1/(1+16\cdot x^2)\,dx \quad \text{(a destra, seconda riga)} \tag{20}$$

$$\int_{-1}^{1} e^{-1/x^2}\,dx \quad \text{(a sinistra, terza riga)} \tag{21}$$

$$\int_{-1}^{1} |x|^3\,dx \quad \text{(a destra, terza riga)} \tag{22}$$

Da osservare che fissato N, le formule composte hanno N nodi mentre le gaussiane e Clenshaw-Curtis ne hanno $N-1$.

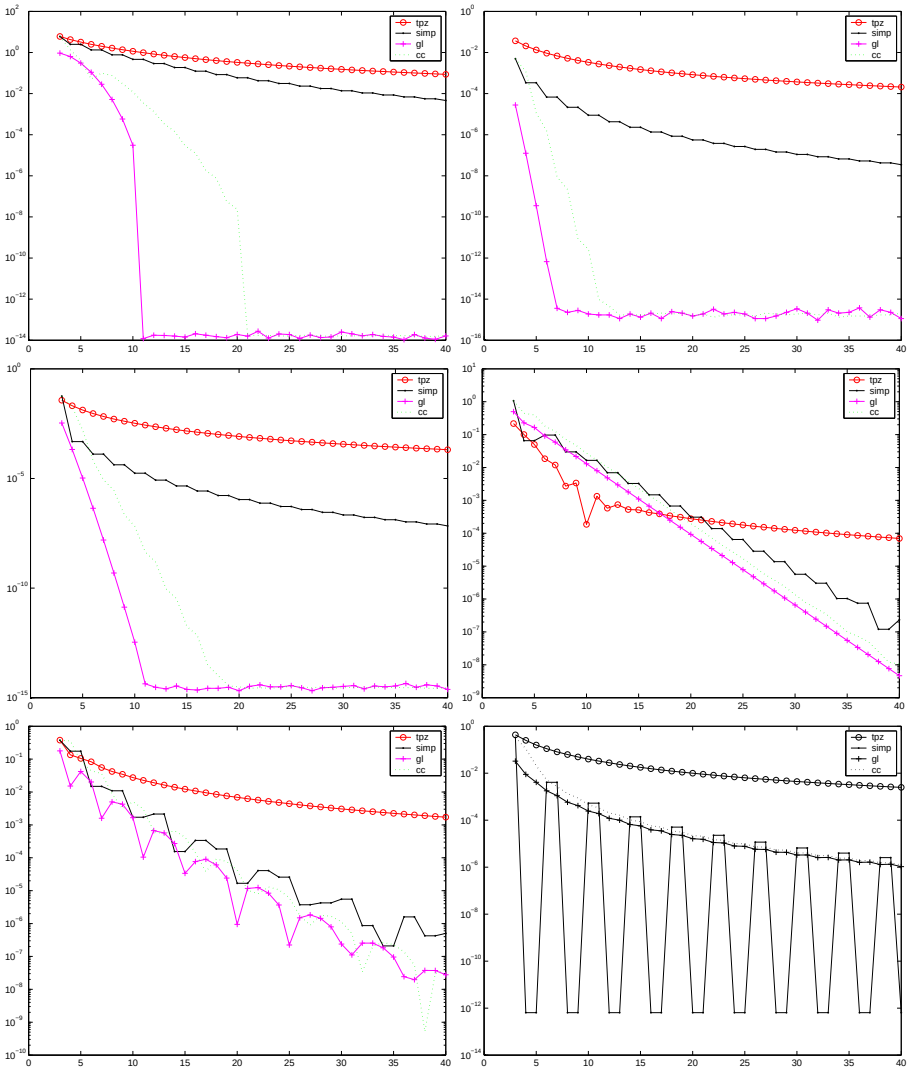

Figure 5: Grafico che illustra l'errore delle formule di quadratura dei trapezi composta, Cavalieri-Simpson composta, Gauss-Legendre e Clenshaw-Curtis su 6 funzioni test.

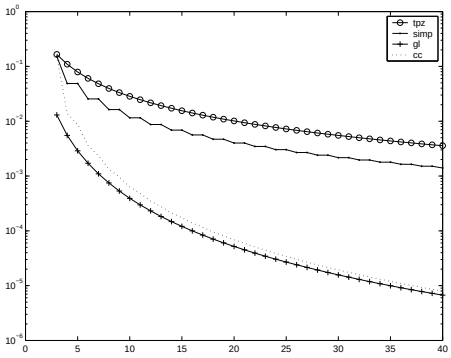

Figure 6: Grafico che illustra l'errore delle formule di quadratura dei trapezi composta, Cavalieri-Simpson composta, Gauss-Legendre e Clenshaw-Curtis sulla funzione test (23).

1.7 Facoltativo: Un'integrale con funzione peso

Consideriamo ora l'integrale

$$\int_{-1}^{1} \exp(x)\sqrt{1-x}\,dx = 1.77914365469190979259117902999 41. \tag{23}$$

Tale risultato è stato ottenuto usando il comando simbolico di Matlab 6.5 (non funziona in Octave, vedere in alternativa il programma Maxima!!)

```
>> syms x
>> int('(exp(x)) * ( (1-x)^(0.5) )',-1,1)
ans =
1.77914365469190979259117902999 41
```

Si capisce che

1. `syms x` rende la variabile x di tipo simbolico (e non numerico!);
2. il termine

   ```
   int('(exp(x)) * ( (1-x)^(0.5) )',-1,1)
   ```

   calcola simbolicamente l'integrale

$$\int_{-1}^{1} \exp(x)\sqrt{1-x}\,dx.$$

Dal grafico in figura 23, si vede come pure la formula di Gauss-Legendre non abbia una grande performance.

E' immediato osservare che $w(x) = \sqrt{1-x}$ è un peso di Gauss-Jacobi

$$w(t) = (1-t)^{\alpha}(1+t)^{\beta}$$

per $\alpha = 1/2$ e $\beta = 0$.

Infatti se $w(x) = \sqrt{1-x}$, allora $g(x) = \exp(x)\, w(x)$ il che corrisponde a usare Gauss-Jacobi con $f(x) = \exp(x)$. Quindi paragoniamo le formule gaussiane (il codice funziona in Matlab 6.1 come pure nella release 2.1.73 di Octave

```
>> a=-1; b=1;
>> [I_jac,x_jac,w_jac]=integrazione_gauss_jacobi(10,1/2,0,a,b,@exp);
>> fprintf('\n \t [GAUSS-JACOBI]: %15.20f',I_jac);

          [GAUSS-JACOBI]: 1.77914365469190930000
>> 1.77914365469190979259117902999 41-1.77914365469190930000
ans =
    4.440892098500626e-016
>> length(x_jac)
ans =
    10
>> [I_jac,x_jac,w_jac]=integrazione_gauss_jacobi(10,0,0,...
      a,b,inline('exp(x).*sqrt(1-x.^2)'));
>> fprintf('\n \t [GAUSS-LEGENDRE]: %15.20f',I_jac);

          [GAUSS-LEGENDRE]: 1.77661983094052260000
>> 1.77914365469190979259117902999 41-1.77661983094052260000
ans =
   0.00252382375139
>> length(x_jac)
ans =
    10
>>
```

Entrambe le formule hanno lo stesso numero di nodi (e pesi), come si vede dalla riga di comando

```
>> length(x_jac)
ans =
    10
```

ma offrono risultati diversi, con un errore assoluto di circa $4.44 \cdot 10^{-16}$ per Gauss-Jacobi con $a = 1/2$, $b = 0$ e di $2.52 \cdot 10^{-3}$ per Gauss-Legendre (cioè Gauss-Jacobi con $a = 0$, $b = 0$).

2 Online

Alcuni link interessanti sono i seguenti:

http://www.cs.purdue.edu/people/wxg
http://en.wikipedia.org/wiki/Clenshaw-Curtis_quadrature
http://en.wikipedia.org/wiki/Fast_Fourier_transform
http://en.wikipedia.org/wiki/Gaussian_quadrature

http://it.wikipedia.org/wiki/Quadratura_di_Gauss
http://it.wikipedia.org/wiki/Regola_del_trapezio
http://it.wikipedia.org/wiki/Regola_di_Cavalieri-Simpson

Per le biografie di alcuni matematici menzionati in questa lezioni si considerino

http://it.wikipedia.org/wiki/Bonaventura_Cavalieri
http://it.wikipedia.org/wiki/Roger_Cotes
http://www-gap.dcs.st-and.ac.uk/~history/Biographies/Cavalieri.html
http://www-history.mcs.st-andrews.ac.uk/Biographies/Cotes.html
http://www-history.mcs.st-andrews.ac.uk/Biographies/Hermite.html
http://www-history.mcs.st-andrews.ac.uk/Biographies/Jacobi.html
http://www-history.mcs.st-andrews.ac.uk/Biographies/Laguerre.html
http://www-history.mcs.st-andrews.ac.uk/Biographies/Simpson.html

Quale nota storica osserviamo che da [9]

> Simpson is best remembered for his work on interpolation and numerical methods of integration. However the numerical method known today as "Simpson's rule", although it did appear in his work, was in fact due to Newton as Simpson himself acknowledged. By way of compensation, however, the Newton-Raphson method for solving the equation f(x) = 0 is, in its present form, due to Simpson. Newton described an algebraic process for solving polynomial equations which Raphson later improved. The method of approximating the roots did not use the differential calculus. The modern iterative form $x_{n+1} = x_n - f(x_n)/f'(x_n)$ is due to Simpson, who published it in 1740.

References

[1] K. Atkinson, *Introduction to Numerical Analysis*, Wiley, 1989.

[2] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.

[3] S.D. Conte e C. de Boor, *Elementary Numerical Analysis, 3rd Edition*, Mc Graw-Hill, 1980.

[4] W. Gautschi, personal homepage, `http://www.cs.purdue.edu/people/wxg`.

[5] Mac Tutor, Cavalieri, `http://www-gap.dcs.st-and.ac.uk/ history/Biographies/Cavalieri.html`.

[6] Mac Tutor, Cotes, `http://www-history.mcs.st-andrews.ac.uk/Biographies/Cotes.html`.

[7] Mac Tutor, Hermite, `http://www-history.mcs.st-andrews.ac.uk/Biographies/Hermite.html`.

[8] Mac Tutor, Jacobi,
http://www-history.mcs.st-andrews.ac.uk/Biographies/Jacobi.html.

[9] Mac Tutor, Simpson,
http://www-history.mcs.st-andrews.ac.uk/Biographies/Simpson.html.

[10] The MathWorks Inc., *Numerical Computing with Matlab*,
http://www.mathworks.com/moler.

[11] A. Quarteroni e F. Saleri, *Introduzione al calcolo scientifico*, Springer Verlag, 2006.

[12] A. Suli e D. Mayers, *An Introduction to Numerical Analysis*, Cambridge University Press, 2003.

[13] L.N. Trefethen, *Is Gauss quadrature better than Clenshaw-Curtis?*, SIAM Reviews, to appear (2007),
http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/CC_trefethen_revised2.pdf.

[14] J. Waldvogel, *"Fast Construction of the Fejer and Clenshaw-Curtis Quadrature Rules"*, BIT 46 (2006), no. 1, 195–202,
http://www.math.ethz.ch/~waldvoge/Papers/fejer.pdf.

[15] Wikipedia, Cavalieri,
http://it.wikipedia.org/wiki/Bonaventura_Cavalieri.

[16] Wikipedia, Clenshaw-Curtis quadrature,
http://en.wikipedia.org/wiki/Clenshaw-Curtis_quadrature.

[17] Wikipedia, Fast Fourier Transform,
http://en.wikipedia.org/wiki/Fast_Fourier_transform.

[18] Wikipedia, Gaussian quadrature,
http://en.wikipedia.org/wiki/Gaussian_quadrature.

[19] Wikipedia, Quadratura gaussiana,
http://it.wikipedia.org/wiki/Quadratura_di_Gauss.

[20] Wikipedia, Regola del trapezio,
http://it.wikipedia.org/wiki/Regola_del_trapezio.

[21] Wikipedia, Regola di Cavalieri-Simpson,
http://it.wikipedia.org/wiki/Regola_di_Cavalieri-Simpson.