

Formule di Quadratura

Data una funzione continua $f(x)$ definita su di un intervallo $[a, b]$, le formule di quadratura si utilizzano per risolvere il problema del calcolo dell'integrale definito

$$S = \int_a^b f(x)dx \Rightarrow S_{n+1} = \sum_{i=0}^n w_i f(x_i)$$

dove gli $n + 1$ punti distinti $x_i \in [a, b]$ per $i = 0, 1, \dots, n$ sono i **nod**i della formula e gli $n + 1$ valori w_i per $i = 0, 1, \dots, n$ sono i **pesi** della formula.

Una **formula di quadratura interpolatoria** si costruisce sostituendo alla $f(x)$ il suo polinomio di interpolazione $p_n(x)$ sugli $n + 1$ nodi x_i

$$p_n(x) = \sum_{i=0}^n L_i(x) f(x_i), \quad L_i(x) = \prod_{\substack{k=0, n \\ k \neq i}} \frac{(x - x_k)}{(x_i - x_k)}, \quad i = 0, 1, \dots, n$$

e pertanto si ottiene

$$S_{n+1} = \int_a^b p_n(x)dx \Rightarrow w_i = \int_a^b L_i(x)dx.$$

Formule di Newton-Cotes : sono formule costruite con nodi equidistanti; in questo caso è possibile scrivere le formule di quadratura nella forma

$$S_{n+1} = h \sum_{i=0}^n \alpha_i f(x_i),$$

dove i pesi α_i sono dati da

$$\alpha_i = \int_0^n \prod_{\substack{k=0, n \\ k \neq i}} \frac{t - k}{i - k} dt$$

e dipendono solo dagli indici i ed n e non anche dall'intervallo di integrazione. In questo modo è possibile tabulare i pesi e ricavare tutte le formule di quadratura di Newton-Cotes.

n	α_0	α_1	α_2	α_3	α_4	α_5	α_6	α_7	Nome
1	$\frac{1}{2}$	$\frac{1}{2}$							Regola del trapezio
2	$\frac{1}{3}$	$\frac{4}{3}$	$\frac{1}{3}$						Formula di Simpson
3	$\frac{3}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	$\frac{3}{8}$					Formula di Simpson 3/8
4	$\frac{14}{45}$	$\frac{64}{45}$	$\frac{24}{45}$	$\frac{64}{45}$	$\frac{14}{45}$				Formula di Bode
5	$\frac{95}{288}$	$\frac{375}{288}$	$\frac{250}{288}$	$\frac{250}{288}$	$\frac{375}{288}$	$\frac{95}{288}$			
6	$\frac{41}{140}$	$\frac{216}{140}$	$\frac{27}{140}$	$\frac{272}{140}$	$\frac{27}{140}$	$\frac{216}{140}$	$\frac{41}{140}$		
7	$\frac{5257}{17280}$	$\frac{25039}{17280}$	$\frac{9261}{17280}$	$\frac{20923}{17280}$	$\frac{20923}{17280}$	$\frac{9261}{17280}$	$\frac{25039}{17280}$	$\frac{5257}{17280}$	

Resto della formula di quadratura: permette di valutare la bontà dell'approssimazione fornita da una formula di quadratura

$$r_{n+1} = S - S_{n+1}$$

Una formula di quadratura ha **grado di precisione** k se risulta $r_{n+1} = 0$ se $f(x) = x^j$ per $j = 0, 1, \dots, k$ e $r_{n+1} \neq 0$ se $f(x) = x^{k+1}$.

Osservazione: essendo sia l'integrale che la formula di quadratura degli operatori lineari, se una formula di quadratura ha grado di precisione k allora calcola esattamente l'integrale definito di qualsiasi polinomio di grado $\leq k$.

Le formule di quadratura di tipo interpolatorio hanno la seguente importante caratteristica.

Teorema 1. *Una formula di quadratura S_{n+1} di tipo interpolatorio ha grado di precisione almeno n .*

In particolare per le formule di Newton-Cotes è possibile dare una espressione piuttosto semplice per il resto.

Teorema 2. Sia S_{n+1} una formula di quadratura di Newton-Cotes, allora

1. se n è pari e $f(x) \in \mathcal{C}^{n+2}([a, b])$, allora esiste un punto $\xi \in]a, b[$ tale che

$$r_{n+1} = \frac{f^{(n+2)}(\xi)}{(n+2)!} \int_a^b x \pi_n(x) dx ,$$

2. se n è dispari e $f(x) \in \mathcal{C}^{n+1}([a, b])$, allora esiste un punto $\xi \in]a, b[$ tale che

$$r_{n+1} = \frac{f^{(n+1)}(\xi)}{(n+1)!} \int_a^b \pi_n(x) dx ,$$

dove $\pi_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$.

Sostituendo alla funzione $f(x)$ i polinomi x^j possiamo studiare il grado di precisione delle formule di quadratura. Riepiloghiamo questi risultati nella tabella che segue.

n	Nome	Resto	Grado di precisione
1	Regola del trapezio	$-\frac{1}{12}h^3 f^{(2)}(\xi)$	1
2	Formula di Simpson	$-\frac{1}{90}h^5 f^{(4)}(\xi)$	3
3	Formula di Simpson 3/8	$-\frac{3}{80}h^5 f^{(4)}(\xi)$	3
4	Formula di Bode	$-\frac{8}{945}h^7 f^{(6)}(\xi)$	5
5		$-\frac{275}{12096}h^7 f^{(6)}(\xi)$	5
6		$-\frac{9}{1400}h^9 f^{(8)}(\xi)$	7
7		$-\frac{8183}{518400}h^9 f^{(8)}(\xi)$	7

Osserviamo che le formule di quadratura di Newton-Cotes S_{n+1} con n pari hanno grado di precisione $n + 1$, mentre le formule con n dispari hanno grado di precisione n .

Date quindi due formule di quadratura successive, una corrispondente ad un n pari ed un'altra al successivo $n + 1$ dispari, esse avranno entrambe grado di precisione $n + 1$ e pertanto è da preferire la formula con n pari in quanto a parità del grado di precisione utilizza meno nodi e quindi riduce il costo di calcolo.

```

% NewtonCotes
% -----
% Algoritmo per il calcolo dell'integrale definito di una funzione
% mediante formule interpolatorie di Newton-Cotes
% -----
% S = NewtonCotes(n,funzione,a,b)
% -----
% Parametri di input
% n = grado della formula di quadratura
% funzione = nome della funzione da integrare
% x0, x1 = estremi dell'intervallo di integrazione
% -----
% Parametri di output
% S = Valore calcolato dell'integrale della funzione in x0, x1
% -----
function S = NewtonCotes(n,funzione,x0,x1) ;

% Coefficienti della formula
switch n
case 1
    a = [1 1] / 2;
case 2
    a = [1 4 1] / 3;
case 3
    a = [3 9 9 3] / 8;
case 4
    a = [14 64 24 64 14] / 45 ;
case 5
    a = [95 375 250 250 375 95] / 288 ;
case 6
    a = [41 216 27 272 27 216 41] / 140 ;
case 7
    a = [5257 25039 9261 20923 20923 9261 25039 5257] / 17280 ;
end

% Calcolo dei nodi e distanza dei nodi
h = (x1 - x0)/n ;
x = linspace(x0,x1,n+1) ;

% Calcolo dell'integrale
S = 0 ;
for i = 1 : n+1
    S = S + a(i)*feval(funzione,x(i)) ;
end
S = S * h ;

```

```

% NewtonCotesConfronto
% -----
% Effettua il confronto tra gli integrali calcolati mediante differenti
% formule di quadratura di Newton-Cotes
% -----
% S = NewtonCotesConfronto(funzione,a,b)
% -----
% Parametri di input
% funzione = nome della funzione da integrare
% a, b = estremi dell'intervallo di integrazione
% -----
% Parametri di output
% S = Ultimo valore calcolato dell'integrale della funzione in x0, x1
% -----
function S = NewtonCotesConfronto(nomefunzione,a,b) ;

% Calcolo integrale esatto mediante la primitiva
S = feval(nomefunzione,b,1) - feval(nomefunzione,a,1) ;

% Scrittura titolo dei risultati
disp(sprintf(' | N | Val. Esatto | Val. Calc. | Err. Relat. | ')) ;

% Calcolo integrale con differenti formula di Newton-Cotes
for n = 1 : 7
    Sn = NewtonCotes(n,nomefunzione,a,b) ;
    disp(sprintf(' | %d | %13.9f | %13.9f | %13.6e | ',n,S,Sn,abs(S-Sn)/S)) ;
end

```

Un modo per migliorare i risultati consiste nell'applicare le formule di quadratura non su tutto l'intervallo $[a, b]$ ma su sottointervalli. Considerati pertanto $N + 1$ nodi in $[a, b]$

$$a = z_0 < z_1 < z_2 < \dots < z_N = b$$

grazie alla proprietà di addittività degli integrali abbiamo che

$$\int_a^b f(x)dx = \sum_{k=0}^{N-1} \int_{z_k}^{z_{k+1}} f(x)dx$$

e pertanto possiamo applicare una formula di quadratura $S_{n+1}^{(k)}$ su ciascun intervallo $[z_k, z_{k+1}]$ ed ottenere la **formula composta**

$$J_{n+1}^{(N)} = \sum_{k=0}^{N-1} S_{n+1}^{(k)}.$$

Ad esempio, applicando la formula dei trapezi su ciascun intervallo $[z_k, z_{k+1}]$, e supponendo che i nodi z_0, z_1, \dots, z_N siano equidistanti con passo $h = (b - a)/N$, abbiamo

$$S_2^{(k)} = \frac{z_{k+1} - z_k}{2} [f(z_k) + f(z_{k+1})] = \frac{b - a}{2N} [f(z_k) + f(z_{k+1})]$$

e pertanto la formula composta è data da

$$J_2^{(N)} = \sum_{k=0}^{N-1} S_2^{(k)} = \frac{b - a}{2N} \sum_{k=0}^{N-1} [f(z_k) + f(z_{k+1})]$$

e quindi

$$J_2^{(N)} = \frac{b - a}{2N} \left[f(z_0) + 2 \sum_{k=1}^{N-1} f(z_k) + f(z_N) \right].$$

Dal punto di vista geometrico, l'applicazione di questa formula equivale a calcolare l'integrale come somma delle aree dei trapezi di coordinate $(z_k, 0)$, $(z_k, f(z_k))$, $(z_{k+1}, f(z_{k+1}))$, $(z_{k+1}, 0)$.

```

% TrapeziComposta
% -----
% Algoritmo per il calcolo dell'integrale definito di una funzione
% mediante la formula composta dei Trapezi
% -----
% [S,Err] = TrapeziComposta(N,funzione,a,b)
% -----
% Parametri di input
% N = numero di sottointervalli della formula composta
% funzione = nome della funzione da integrare
% a, b = estremi dell'intervallo di integrazione
% -----
% Parametri di output
% Sn = Valore calcolato dell'integrale della funzione in a, b
% Err = Errore assoluto rispetto all'integrale esatto
% -----
function [Sn,Err] = TrapeziComposta(N,funzione,a,b) ;

% Calcolo degli estremi dei sottointervalli
z = linspace(a,b,N+1) ;

% Calcolo dell'integrale
Sn = feval(funzione,z(1)) + feval(funzione,z(N+1)) ;
for k = 1 : N
    Sn = Sn + 2*feval(funzione,z(k)) ;
end
Sn = Sn * (b-a)/(2*N) ;

% Calcolo dell'integrale esatto mediante primitiva e dell'errore assoluto
S = feval(funzione,b,1) - feval(funzione,a,1) ;
Err = abs(S-Sn) ;

% Visualizzazione grafica della formula dei trapezi composta
ngrid = 200 ;
t = linspace(a,b,ngrid) ;
for i = 1 : ngrid
    f(i) = feval(funzione,t(i)) ;
end
plot(t,f) ;
hold on
for k = 1 : N
    fk = feval(funzione,z(k)) ;
    fk1 = feval(funzione,z(k+1)) ;
    plot([z(k),z(k),z(k+1),z(k+1),z(k)], [0,fk,fk1,0,0]) ;
end
hold off

```

```

% TrapeziCompostaConfronto
% -----
% Effettua il confronto tra gli integrali calcolati mediante la formula
% dei trapezi composta con differenti suddivisioni dell'intervallo
% -----
% S = TrapeziCompostaConfronto(funzione,a,b,Nmax)
% -----
% Parametri di input
% funzione = nome della funzione da integrare
% a, b = estremi dell'intervallo di integrazione
% Nmax = numero massimo di suddivisioni dell'intervallo
% -----
% Parametri di output
% S = Ultimo valore calcolato dell'integrale della funzione in x0, x1
% -----
function S = TrapeziCompostaConfronto(funzione,a,b,Nmax) ;

% Calcolo integrale esatto mediante la primitiva
S = feval(funzione,b,1) - feval(funzione,a,1)

% Scrittura formato di stampa e titolo dei risultati
formato = ' | %7d | %12.8f | %12.8f | %10.3e | ' ;
disp(' | Interv. | Val. Esatto | Val. Calcolato | Err. Assol. | ' ) ;

% Calcolo dell'integrale con un numero crescente di sottointervalli
for N = 1 : Nmax
    Sn = TrapeziCompostaCalcolo(N,funzione,a,b) ;
    Errore = abs(S-Sn) ;
    disp(sprintf(formato,N,S,Sn,Errore)) ;
end

```

```

% TrapeziCompostaCalcolo
% -----
% Algoritmo per il calcolo dell'integrale definito di una funzione
% mediante la formula dei Trapezi composta
% -----
% Sn = TrapeziCompostaCalcolo(N,funzione,a,b)
% -----
% Parametri di input
% N = numero di sottointervalli della formula composta
% funzione = nome della funzione da integrare
% a, b = estremi dell'intervallo di integrazione
% -----
% Parametri di output
% Sn = Valore calcolato dell'integrale della funzione in a, b
% -----
function Sn = TrapeziCompostaCalcolo(N,funzione,a,b) ;

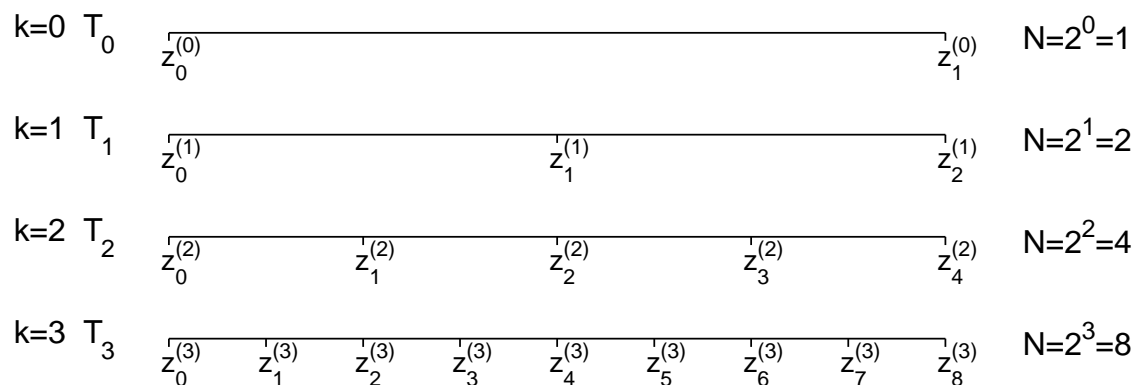
% Calcolo degli estremi dei sottointervalli
z = linspace(a,b,N+1) ;

% Calcolo dell'integrale
Sn = feval(funzione,z(1)) + feval(funzione,z(N+1)) ;
for k = 2 : N
    Sn = Sn + 2*feval(funzione,z(k)) ;
end
Sn = Sn * (b-a)/(2*N) ;

```

Nelle applicazioni reali viene fissata una tolleranza ϵ e si richiede che l'errore commesso nel calcolare l'integrale sia al di sotto di questa tolleranza, ossia che $|S - J_{n+1}^{(N)}| < \epsilon$.

Non avendo a disposizione il valore teorico S dell'integrale non è possibile effettuare direttamente questo controllo, e pertanto risulta particolarmente utile disporre di un modo per **stimare l'errore** commesso. Per fare ciò si costruisce una sequenza di approssimazioni, ciascuna su di un numero di intervalli doppio rispetto al precedente



per cui in generale abbiamo

$$k \rightarrow T_k = J_{n+1}^{(N)}, \quad \text{con } N = 2^k, \quad h_k = \frac{b-a}{2^k}$$

e, indicati con $y_i^{(k+1)}$ i punti di mezzo della suddivisione al passo k , ossia i punti da aggiungere alla suddivisione al passo k per ottenere la suddivisione al passo $k+1$

$$y_i^{(k+1)} = \frac{z_i^{(k)} + z_{i+1}^{(k)}}{2}, \quad i = 0, 1, \dots, 2^k - 1$$

nel caso della formula del trapezio risulta che

$$T_{k+1} = \frac{T_k}{2} + \frac{h_k}{2} \sum_{i=0}^{2^k-1} f(y_i^{(k+1)})$$

che permette di calcolare l'integrale al passo $k+1$ sfruttando i calcoli già effettuati al passo k ed effettuando nuovi calcoli solo per i punti introdotti al passo $k+1$.

Sappiamo che se la funzione $f(x)$ è sufficientemente regolare, allora una stima dell'errore al passo k è data da

$$|S - T_{k+1}| < \frac{1}{3} |T_{k+1} - T_k|$$

che possiamo utilizzare per il confronto.

```

% TrapeziCompostaStimaErrore
% -----
% Algoritmo per il calcolo dell'integrale definito di una funzione
% mediante la formula dei Trapezi composta su una successione di
% intervalli, con stima dell'errore
% -----
% [S,Err] = TrapeziCompostaStimaErrore(K,funzione,a,b)
% -----
% Parametri di input
% K = numero massimo di suddivisioni dell'intervallo da utilizzare
% funzione = nome della funzione da integrare
% a, b = estremi dell'intervallo di integrazione
% -----
% Parametri di output
% Sn = Valore calcolato dell'integrale della funzione in a, b
% Err = Errore assoluto rispetto all'integrale esatto
% -----
function [Sn,Err] = TrapeziCompostaStimaErrore(K,funzione,a,b) ;

% Scrittura formato di stampa e titolo dei risultati
formato = ' | %7d | %12.8f | %12.8f | %12.5e | %12.5e | ' ;
disp(' | Interv. | Val. Esatto | Val. Calcolato | Err. Assoluto | Err. Stimato | ' ) ;

% Calcolo dell'integrale esatto mediante primitiva
S = feval(funzione,b,1) - feval(funzione,a,1) ;

% Calcolo della prima approssimazione
zk(1) = a ; zk(2) = b ;
hk = (b-a) ;
Tk = hk/2 * (feval(funzione,zk(1)) + feval(funzione,zk(2))) ;

for k = 0 : K

    % Calcolo dei nuovi punti per la suddivisione k+1
    for i = 0 : 2^k - 1
        y(i+1) = (zk(i+1)+zk(i+2))/2 ;
    end

    % Calcolo della approssimazione al passo k+1
    Tk1 = Tk/2 ;
    for i = 0 : 2^k-1
        Tk1 = Tk1 + hk/2 * feval(funzione,y(i+1)) ;
    end

    % Aggiunta alla suddivisione al passo k dei nuovi punti
    zk1 = [zk(1)] ;
    for i = 0 : 2^k-1 ;
        zk1 = [zk1 y(i+1) zk(i+2)] ;
    end
end

```

```
end

% Visualizzazione risultati
Errore = abs(S-Tk1) ;
ErroreStimato = abs(Tk1-Tk)/3 ;
disp(sprintf(formato,2^(k+1),S,Tk1,Errore,ErroreStimato)) ;

% Aggiornamento valori
hk = hk/2 ;
Tk = Tk1 ;
zk = zk1 ;

end
```

Per verificare praticamente questi risultati, calcoliamo numericamente con le diverse formule alcuni integrali e confrontiamo il risultato con quello teorico.

Funzione	Primitiva
$f(x) = \sin x \cos x$	$F(x) = \frac{1}{2} \sin^2 x$
$f(x) = e^x \cos x$	$F(x) = \frac{1}{2} e^x \cos x + \frac{1}{2} e^x \sin x$
$f(x) = 2x^3 - 4x^2 + x - 1$	$F(x) = \frac{1}{2} x^4 - \frac{4}{3} x^3 + \frac{1}{2} x^2 - x$
$f(x) = x^4 - 2$	$F(x) = \frac{1}{5} x^5 - 2x$

```

% y = IntegrFunz1(x,flag)
% -----
% Parametri di input
% x = valore in cui calcolare la funzione
% flag = flag la cui presenza indica di calcolare la primitiva della
%       funzione invece che la funzione stessa
% -----
% Parametri di output
% y = valore della funzione o della primitiva
% -----
function y = IntegrFunz1(x,flag) ;
    if nargin == 1
        y = sin(x)*cos(x) ; % Valore funzione
    else
        y = sin(x)^2/2 ; % Valore primitiva
    end
% -----

```

```

% y = IntegrFunz2(x,flag)
% -----
% Parametri di input
% x = valore in cui calcolare la funzione
% flag = flag la cui presenza indica di calcolare la primitiva della
%       funzione invece che la funzione stessa
% -----
% Parametri di output
% y = valore della funzione o della primitiva
% -----
function y = IntegrFunz2(x,flag) ;
    if nargin == 1
        y = exp(x)*cos(x) ; % Valore funzione
    else
        y = (1/2)*exp(x)*cos(x) + (1/2)*exp(x)*sin(x) ; % Valore primitiva
    end
% -----

```

```

% y = IntegrFunz3(x,flag)
% -----
% Parametri di input
% x = valore in cui calcolare la funzione
% flag = flag la cui presenza indica di calcolare la primitiva della
%       funzione invece che la funzione stessa
% -----
% Parametri di output
% y = valore della funzione o della primitiva
% -----
function y = IntegrFunz3(x,flag) ;
    if nargin == 1
        y = 2*x^3 - 4*x^2 + x - 1 ; % Valore funzione
    else
        y = (1/2)*x^4 - (4/3)*x^3 + (1/2)*x^2 - x ; % Valore primitiva
    end
% -----

```

```

% y = IntegrFunz4(x,flag)
% -----
% Parametri di input
% x = valore in cui calcolare la funzione
% flag = flag la cui presenza indica di calcolare la primitiva della
%       funzione invece che la funzione stessa
% -----
% Parametri di output
% y = valore della funzione o della primitiva
% -----
function y = IntegrFunz4(x,flag) ;
    if nargin == 1
        y = x^4 - 2 ; % Valore funzione
    else
        y = 1/5*x^5 - 2*x ; % Valore primitiva
    end
% -----

```