Fondamenti di Informatica, A.A. 2013-2014 - Compito A

30/07/2014

Prova Pratica

L'integrale definito di una funzione continua su un intervallo chiuso e limitato può essere calcolato con la regola dei trapezi composita, che si definisce come segue:

1. Dato un intervallo $[a, b]$ lo si divide in N sottointervalli $[x_i, x_{i+1}]$, $i = 0, 1, 2, \ldots$, dove

$$a = x_0\,, b = x_N, \quad x_{i+1} = x_i + h, \quad h = \frac{b-a}{N};$$

2. Su ciascun intervallo si approssima l'area sottesa dalla curva con l'area del trapezio definito sul segmento $[x_i, x_{i+1}]$, dalla retta che congiunge i punti $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$; l'area del trapezio elementare così ottenuto vale

$$\frac{f(x_i) + f(x_{i+1})}{2}(x_{i+1} - x_i) = \frac{h}{2}(f(x_i) + f(x_{i+1}))$$

3. L'integrale viene calcolato sommando tutti i contributi così ottenuti;

Per valutare la qualità della stima si possono calcolare approssimazioni successive con valori di N crescenti ed arrestandosi quando i valori stimati sono sufficientemente vicini a fronte di un congruo aumento di N. Si definisca una funzione Matlab/Octave che data una funzione, un intervallo e una tolleranza, calcoli l'integrale definito della funzione, stimando un errore che rispetti la tolleranza.

Si usi il codice per valutare

$$\int_0^1 \log(x^2 + 1).$$

(Bonus: nell'aumentare il numero di punti, usare $N_{k+1} = 2N_k$, e riutilizzare i valori $f(x_i)$ della iterazione precedente).

Soluzione

La soluzione del problema proposto può essere organizzata in maniera semplice definendo una funzione interna che applica la regola dei trapezi composita su N intervalli, ed in una esterna che verifica la convergenza.

Per la funzione interna indichiamo con `n` il numero di punti di valutazione da usare, ricordando che $n = N + 1$; usando il comando `linspace` otteniamo le ascisse necessarie. Notiamo inoltre che vale

$$\sum_{i=0}^{N-1} \frac{h}{2}(f(x_i) + f(x_{i+1})) = \frac{h}{2}(f(x_0) + f(x_N)) + h\sum_{i=1}^{N-1} f(x_i)$$

da cui

```
function val=trapezi_in(f,a,b,n)
  if (n>1)
   x=linspace(a,b,n);
   y=f(x);
   h=(b-a)/(n-1);
   val=h*(0.5*(y(1)+y(end))+sum(y(2:end-1)));
  end
end
```

La funzione esterna incrementa N e confronta i valori ottenuti:

```
function  val=trapezi(f,a,b,tol)

  if (a>b)
    c=a; a=b; b=c;
  end
  maxn=1025;
  n=1;
  valp=trapezi_in(f,a,b,n+1);
  n=2
  val=trapezi_in(f,a,b,n+1);
  while ((abs(val-valp)>tol)&&(n<=maxn))
    valp=val;
    n=2*n;
    val=trapezi_in(f,a,b,n+1);
  end

end
```

La soluzione presentata è certamente compatta, ma potrebbe non essere efficiente, nel senso che tende a ricalcolare i valori di $f(x)$ in punti già visitati.

Una soluzione più efficiente si può ottenere osservando che se ad ogni passo raddoppiamo il numero di intervalli N, questo corrisponde ad inserire nell'insieme $\{x_i\}$ i punti medi degli intervalli già esistenti; i valori della funzione già calcolati possono essere riutilizzati, e bisogna calcolare i nuovi valori solo nei punti medi. Con queste modifiche si ottiene la versione

```
function  val=trapezi(f,a,b,tol)

  if (a>b)
    c=a; a=b; b=c;
  end
  maxn=10;
  n=1;
  h=b-a;
  x0=[a,b];
  y0=f(x0);
  val0=h*sum(y0)/2;
  valp=val0;
```

```
xm=(a+b)/2;
h2=h/2;
y=f(xm);
val=h2*(0.5*sum(y0)+sum(y));
x=[a,xm,b];
while ((abs(val-valp)>tol)&&(n<=maxn))
   valp=val;
   h=h2;
   % Calcola i punti medi
   xm = (x(1:end-1)+x(2:end))/2;
   h2=h/2;
   % valuta F nei punti medi
   ym = f(xm);
   y=[y,ym];
   %Aggiorna l'integrale
   val = h2*(0.5*sum(y0)+sum(y));
   % Costruisci il nuovo insieme di punti X
   xn=zeros(1,length(x)+length(xm));
   xn(1:2:end)    = x;
   xn(2:2:end)    = xm;

   x=xn;
   n=n+1;
end
```

Con il codice proposto si ottiene

$$\int_0^1 \log(x^2+1) \approx 0.26394.$$

Fondamenti di Informatica, A.A. 2013-2014 - Compito B

30/07/2014

Prova Pratica

Data una generica equazione non lineare

$$f(x) = 0,$$

si usano spesso dei metodi iterativi che a partire da un tentativo iniziale x_0 producono una successione di punti $x_0, x_1, \ldots, x_k$, che sotto opportune condizioni approssimano sempre meglio la soluzione x^*. Tra questi abbiamo il metodo di Newton che si applica quando sia nota la derivata $f'(x)$:

Dati $(x_0, f(x_0), f'(x_0))$, e $i \leftarrow 0$;
while Se il criterio di arresto non è verificato da $x_i, f(x_i)$ **do**
$x_{i+1} \leftarrow x_i - \frac{f(x_i)}{f'(x_i)}$
$i \leftarrow i + 1$;

end while

Per verificare la convergenza di solito si controlla sia che il valore della funzione $f(x_i)$ sia sufficientemente piccolo, sia che due approssimazioni x_i, x_{i+1} siano sufficientemente vicine. Si definisca una funzione Matlab/Octave che, dati in ingresso una funzione, la sua derivata, un punto iniziale, una tolleranza ed un massimo numero di iterazioni ammissibili, restituisca una stima della radice della funzione calcolata con il metodo appena visto.

Si usi la funzione Matlab/Octave sviluppata per trovare lo zero della funzione

$$f(x) = 2 - x^2 - cos(x) \qquad x \in [0, \frac{\pi}{2}]$$

usando come punto iniziale $x_0 = \frac{\pi}{2}$.

Soluzione

La soluzione riportata impone che la tolleranza non sia eccessivamente stringente, ed interpreta la distanza tra due iterazioni successive secondo l'errore relativo, avendo cura di evitare possibili divisioni per numeri molto piccoli. Una caratteristica particolare che non era contenuta nel testo, ma che in un metodo siffatto andrebbe considerata, è che il codice controlla che il valore x_{i+1} sia comunque contenuto nell'intervallo iniziale $[a, b]$.

```
function [r,it]=newton(f,f1,x0,tol,maxit)
%************************************************************
%
%          Metodo di Newton
%                  x_{i+1}=x_i-f(x_i)/f1(x_i)
% per l'approssimazione di  una radice di un'equazione f(x)=0
% data la funzione f e la sua derivata f1
%
%************************************************************
  if (nargin > 4)
    n_max_iter=maxit;
  else
    n_max_iter=40;
  end
  if (nargin > 3)
    tau=abs(tol);
  else
    tau=1e-5;
  end
  tau = max(tau,eps);

  x=x0;
  x1=x;
  for i=1:n_max_iter
    y=feval(f,x);
    if (abs(y) < tau)
      % Exit condition on the residual
```

```
    % Here x=x1
    break
  end
  y1=feval(f1,x);
  % x1 now gets x_{i+1}
  x1 = x - y / y1;
  %
  if (abs(x1-x)< tau*max(abs(x),abs(x1)))
    % Exit condition on convergence of X
    break
  end
  x=x1;
end

r=x1;
if(nargout>1)
  it=i;
end
```

Applicando il metodo alla ricerca proposta si ottiene

$$x^* \approx 1.3256.$$