Elaborato di

Calcolo Numerico

Anno Accademico 2019/2020

Niccolò *Piazzesi* - 6335623 - *niccolo.piazzesi@stud.unifi.it*
Pietro *Bernabei* - 6291312 - *pietro.bernabei@stud.unifi.it*

Contents

List of Figures

List of Tables

1 Capitolo 1

1.1 Esercizio 1

Sia $f(x)$ una funzione sufficientemente regolare e sia $h > 0$ una quantità abbastanza "piccola". Possiamo sviluppare i termini $f(x-h)$ e $f(x+h)$ mediante il polinomio di Taylor:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4)$$

Sostituiamo i termini nell'espressione iniziale:

$$\frac{f(x-h) - 2f(x) + f(x+h)}{h^2} =$$

$$= \frac{f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4) - 2f(x) + f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)}{h^2} =$$

$$= \frac{h^2 f''(x) + O(h^4)}{h^2} = f''(x) + O(h^2)$$

1.2 Esercizio 2

Eseguendo lo script si ottiene $u = 1.1102e-16 = \frac{\epsilon}{2}$, dove ϵ è la precisione di macchina. ϵ è il più piccolo valore di macchina per il quale $a+\epsilon \neq a$ per un qualsiasi numero a. Quando u assume valore $\frac{\epsilon}{2}$ il controllo interno $1 + u == 1$, che corrisponde alla condizione di uscita, risulta vero, perchè u è minore di ϵ.

1.3 Esercizio 3

Quando si esegue $a - a + b$ il risultato è 100 mentre quando si esegue $a + b - a$ si ottiene 0. La differenza dei risultati è dovuta al fenomeno della cancellazione numerica:

- nel primo caso la sottrazione avviene sullo stesso numero $a = 1e20$. Sottrare un numero da se stesso ha sempre risultato esatto 0.
- nel secondo caso la sottrazione avviente tra i termini $a + b = 1e20 + 100$ e $a = 100$. Poichè 1e20 è molto più grande di 100, a+b è "quasi uguale" ad a. La sottrazione amplifica gli errori di approssimazione causati dalla rappresentazione in aritmetica finita dei numeri coinvolti. A causa di questi errori il calcolatore approssima la differenza con 0.

2 Capitolo 2

2.1 Esercizio 4

```
function x1=radn(x, n)
%
% x1=radn(n.x)
% funzione Matlab che implementa il metodo di newton per il calcolo della
% radice n-esima di un numero positivo x
%
format long e
imax=1000;
tolx=eps;
if x<=0
    error('valore in ingresso errato');
end
x1=x/2;
for i=1:imax
   x0=x1;
   fx=x0^n-x;
   fx1=(n)*x0^(n-1);
   x1=x0-fx/fx1;
   if abs(x1-x0)<=tolx
       break
   end

end
if abs(x1-x0)>tolx
    error('metodo non converge')
end
```

2.2 Esercizio 5

- Metodo di bisezione

```
function [x,i] = bisezione(f,a,b,tolx)
%bisez
%[x,i]=bisezione(f, a, b,  tolx, maxit)
%Pre: f continua in [a,b]
%  Applica il metodo di bisezione per il calcolo della
%  radice dell'equazione f(x)=0
%  f          -funzione
%  a, b       - estremi dell'intervallo
%
% tolx        -tolleranza
% restituisce in x l'approssimazione della radice e in i il numero di iterazioni
% VEDI ANCHE: newton, corde, secanti, aitken, newtonmod
    format long e
    fa = feval(f,a);
    fb = feval(f,b);
    if(fa * fb > 0 )
        error('gli estremi  hanno lo stesso segno');
    end
    x0=a;
    imax =  ceil(log2(b-a) - log2(tolx));
    for i = 1:imax
        x = (a+b)/2;
        fx = feval(f,x);
```

```
        if abs(x-x0) <= tolx*(1+abs(x0))
            break
        end
        x0=x;
        if fa*fx<0
            b = x;
            fb = fx;
        else
            a = x;
            fa = fx;
        end
    end

end
```

- Metodo di Newton

```
function [x,i] = newton( f, f1, x0, tolx, maxit )
%newton
%[x,i]=newton(f,f1, x0, tolx, maxit)
%Pre: f derivabile
%  Applica il metodo di newton per il calcolo della
%  radice dell'equazione f(x)=0
%  f          -funzione
%  f1         -derivata di f
%  x0         -approssimazione iniziale
% tolx        -tolleranza
% maxit       -numero massimo di iterazioni(default=100)
% restituisce in x l'approssimazione della radice e in i il numero di iterazioni
% VEDI ANCHE: bisezione, corde, secanti, aitken, newtonmod

       format long e
       if nargin<4
              error('numero argomenti insufficienti');
       elseif nargin==4
               maxit = 100;
       end
       if tolx<eps
              error('tolleranza non idonea');
       end
       x = x0;
       for i = 1:maxit
              fx = feval( f, x );
              f1x = feval( f1, x );
              x = x - fx/f1x;
              if abs(x-x0)<=tolx*(1+abs(x0))
                     break;
              else
                     x0 = x;
              end
       end
       if abs(x-x0) > tolx*(1+abs(x0))
              error('metodo non converge');
       end
end
```

- Metodo delle secanti

```
function [x, i]=secanti(f,x0,x1,tolx,maxit)
%secanti
%[x,i]=secanti(f, x0, x1, tolx, maxit)
%
%  Applica il metodo delle secanti per il calcolo della
%  radice dell'equazione f(x)=0
%  f          -funzione
%  x0         -approssimazione iniziale
%  x1         -seconda approssimazione iniziale
% tolx        -tolleranza
% maxit       -numero massimo di iterazioni(default=100)
% restituisce in x l'approssimazione della radice e in i il numero di iterazioni
% VEDI ANCHE: bisezione, newton, corde, aitken, newtonmod

  format long e
  if nargin<4
    error('numero argomenti insufficienti');
  elseif nargin==4
    maxit = 100;
  end
  i=0;
  f0=feval(f,x0);
  for i=1:maxit
      f1=feval(f,x1);
      df1=(f1-f0)/(x1-x0);
      x=x1-(f1/df1);
      if abs(x1-x0)<=tolx*(1+abs(x0))
        break;
      end
      x0=x1;
      x1=x;
      f0=f1;

  end
  if abs(x-x0) > tolx*(1+abs(x0))
    error('metodo non converge');
  end
end
```

- Metodo delle corde

```
function [x,i] = corde( f, f1, x0, tolx, maxit )
%corde
%[x,i]=corde(f,f1, x0, tolx, maxit)
%Pre: f derivabile
%  Applica il metodo delle corde per il calcolo della
%  radice dell'equazione f(x)=0
%  f          -funzione
%  f1         -derivata di f
%  x0         -approssimazione iniziale
% tolx        -tolleranza
% maxit       -numero massimo di iterazioni(default=100)
% restituisce in x l'approssimazione della radice e in i il numero di iterazioni
% VEDI ANCHE: bisezione, newton, secanti, aitken, newtonmod

    format long e
    if nargin<4
           error('numero argomenti insufficienti');
    elseif nargin==4
```

```
            maxit = 100;
    end
    if tolx<eps
           error('tolleranza non idonea');
    end
    f1x = feval(f1, x0);
    x = x0;
    for i = 1:maxit
           fx = feval( f, x );
           if fx==0
                  break;
           end
           x = x - fx/f1x;
           if abs(x-x0)<=tolx*(1+abs(x0))
                  break;
           else
                  x0 = x;
           end
    end
    if abs(x-x0) > tolx*(1+abs(x0))
       error('metodo non converge');
     end
end
```

2.3 Esercizio 6

Eseguendo lo script es6.m si ottengono i risultati contenuti nella tabella 2 e nella figura 1. Come si può notare, il metodo di newton e il metodo delle secanti convergono molto più rapidamente del metodo di bisezione e del metodo delle corde.

Metodo	tolleranza= 10^{-3}	tolleranza= 10^{-6}	tolleranza= 10^{-9}	tolleranza= 10^{-12}
bisezione	0.739257812500000	0.739085197448730	0.739085133187473	0.739085133215667
newton	0.739085133385284	0.739085133215161	0.739085133215161	0.739085133215161
corde	0.739567202212256	0.739084549575213	0.739085132739254	0.739085133215737
secanti	0.739085133215001	0.739085133215161	0.739085133215161	0.739085133215161

Table 1: valori approssimati

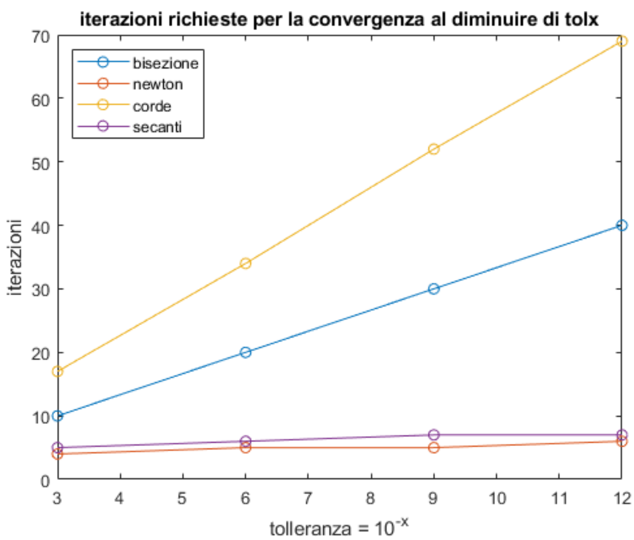


Figure 1: iterazioni richieste

2.4 Esercizio 7

Le nuove funzioni utilizzate in questo esercizio sono:

- Metodo di Newton modificato

```
function [x, i] = newtonmod( f, f1,  x0, m, tolx, maxit )
%NEWTONMOLT
%[x,i]=Newtonmolt(f,f1,x0,m,tolx,maxit)
%  Pre: f derivabile
%  Applica il metodo di Newton per il calcolo della
%  radice (di molteplicita' nota r) dell'equazione f(x)=0
%  f          -funzione
%  f1         -derivata di f
%  x0         -approssimazione iniziale
%  m          -molteplicita' della radice
% tolx        -tolleranza
% maxit       -numero massimo di iterazioni(default=100)
% restituisce in x l'approssimazione della radice e in i il numero di iterazioni
% VEDI ANCHE: bisezione, newton, secanti, corde, aitken

    format long e
    if nargin<5
           error('numero argomenti insufficienti');
    elseif nargin==5
            maxit = 100;
    end
    if tolx<eps
           error('tolleranza non idonea');
    end
    x = x0;
    for i = 1:maxit
           fx = feval( f, x );
           f1x = feval( f1, x );
           if fx==0
                  break;
           end
           x = x - m*(fx/f1x);
           if abs(x-x0)<=tolx*(1+abs(x0))
                  break;
           else
                  x0 = x;
           end
    end

end
```

- Metodo delle accelerazioni di Aitken

```
function [x, i] = aitken( f, f1, x0, tolx, maxit )
%aitken
%[x,i]=aitken(f,f1, x0, tolx, maxit)
%  Pre: f derivabile
%  Applica il metodo di  accelerazione di aitken per il calcolo della
%  radice (di molteplicita' incognita) dell'equazione f(x)=0
%  f          -funzione
%  f1         -derivata di f
%  x0         -approssimazione iniziale
% tolx        -tolleranza
% maxit       -numero massimo di iterazioni(default=100)
```

```
% restituisce in x l'approssimazione della radice e in i il numero di iterazioni
% VEDI ANCHE: bisezione, newton, secanti, corde, newtonmod
        format long e
        if nargin<4
                error('numero argomenti insufficienti');
        elseif nargin==4
                 maxit = 100;
        end
        if tolx<eps
                error('tolleranza non idonea');
        end
        x = x0;
        for i = 1:maxit
                x0 = x;
                fx = feval( f, x0 );
                f1x = feval( f1, x0 );
                x1 = x0 - fx/f1x;
                fx = feval( f, x1 );
                f1x = feval( f1, x1 );
                x = x1 -fx/f1x;
                x = (x*x0-x1^2)/(x-2*x1+x0);
                if abs(x-x0)<=tolx*(1+abs(x0))
                        break;
                end
        end
        if abs(x-x0) > tolx*(1+abs(x0))
                disp('metodo non converge');
        end
end
```

La radice nulla della funzione $f(x) = x^2 tan(x)$ ha molteciplita m = 3, in quanto 0 annulla due volte il termine x^2 e $tan(0) = 0$.

3 Capitolo 3

3.1 Esercizio 8

```
function [LU,p]=palu(A)
% [LU,p]=palu(A)
% funzione che dato in input matrice A restituisce matrice fattorizzata LU
% e il relativo vettore p di permutazione di LU con pivoting parziale di A
% input:
%   A= matrice di cui si vuole calcolare la fattorizzazione lu con pivoting
%   parziale
% output:
%   LU=matrice quadrata di dimensioni n*n, composta dalla matrice
%   triangolare superiore U e la matrice triangolare inferiore a diagonale
%   unitaria L
%   p= vettore di permutazione di dimensione n, generato dalla
%   fattorizzazione di A con pivoting parziale
%

[n,m]=size(A);
if(n~=m)
    error(matrice A non quadrata);
end
LU=A;
p=[1:n];
for i=1:n-1
    [mi,ki]=max(abs(LU(i:n,i)));
    if mi==0
        error('La matrice e'' non singolare')
    end
    ki=ki+i-1;
    if ki>i
        p([i ki])=p([ki i]);
       LU([i ki],:)= LU([ki i],:);
    end
    LU(i+1:n,i)=LU(i+1:n,i)/LU(i,i);
    LU(i+1:n,i+1:n)=LU(i+1:n,i+1:n)-LU(i+1:n,i)*LU(i,i+1:n);
end
return
end
```

3.2 Esercizio 9

```
function x=LUsolve(LU,p,b)
%
% funzione che risolve il sistema lineare LUx=b(p):
%input:
%   LU=matrice quadrata (n*n) fattorizzata LU, ottenuta attrarso la
%   fattorizzazione con pivoting parziale
%   p= vettore di permutazione per b, di dimensione n, con valori da (1 a
%   n)
%   b=vettore dei termini noti
%output:
%   x=vettore delle incognite calcolate
%
%
   [m,n]=size(LU);
```

```
    if(m~=n || n~=length(b)) error('dati incosistenti')
    else if(min(abs(diag(LU)))==0)
            error(fattorizzazione errata);
        end
    end
     x=b(p);
     for i=1:n-1
         x(i+1:n)=x(i+1:n)-(LU(i+1:n,i)*x(i));
     end
        x(n)=x(n)/LU(n,n);
        for i=n-1:-1:1
            x(1:i)=x(1:i)-(LU(1:i,i+1)*x(i+1));
            x(i)=x(i)/LU(i,i);
        end
     return
end
```

3.3 Esercizio 10

Si nota da questo tabella, che sigma e la norma euclidea,tra la differenza di x e xref, sono direttamente proporzionali

Sigma	Norma
10^{-1}	8.9839e-15
10^{1}	1.4865e-14
10^{3}	1.3712e-12
10^{5}	1.2948e-10
10^{7}	5.3084e-09
10^{9}	1.0058e-06
10^{11}	8.5643e-05
10^{13}	0.0107
10^{15}	0.9814
10^{17}	4.1004e+03

Table 2: valori approssimati

3.4 Esercizio 11

```
function QR = myqr(A)
%QR = myqr(A)
% calcola la fattorizzazione QR di Householder della matrice A

    [m,n] = size(A);
    if  n > m
        error('Dimensioni errate');
    end
    QR = A;
    for i = 1:n
        alfa = norm(QR(i:m,i));
        if alfa == 0
            error('la matrice non ha rango massimo');
        end
        if QR(i,i) >= 0
            alfa = -alfa;
        end
        v1 = QR(i,i) -alfa;
        QR(i,i) = alfa;
```

```
        QR(i+1:m,i) = QR(i+1:m,i)/v1;
        beta = -v1/alfa;
        v = [1; QR(i+1:m,i)];
        QR(i:m,i+1:n) = QR(i:m,i+1:n) - (beta * v) * (v' * QR(i:m,i+1:n));
    end
end
```

3.5 Esercizio 12

```
function x = qrsolve(QR, b)
%
%
% x = qrSolve(QR, b)
% risolve il sistema QR*x=b nel senso dei minimi quadrati
%
[m,n] = size(QR);
k = length(b);
if k ~= m
    error('Dati inconsistenti');
end
x=b(:);
for i = 1:n
    v=[1; QR(i+1:m,i)];
    beta = 2/(v'* v);
    x(i:m) = x(i:m) - (beta*(v'*x(i:m))*v);
end
x=x(1:n);
for  j = n:-1:1
    if QR(j,j)==0
        error('Matrice singolare');
    end
    x(j) = x(j) / QR(j,j);
    x(1:j-1) = x(1:j-1) - QR(1:j-1,j)*x(j);
end
return
end
```

4 Capitolo 4

5 Capitolo 5

5.1 Esercizio 21

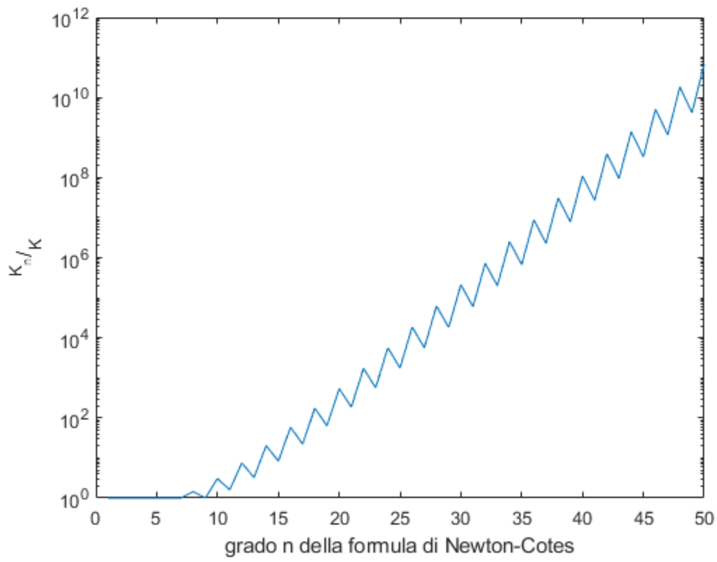
```
function c = ncweights(n)
%
%
%  c = nc-weights(n)
%  calcola i pesi della formula di newton cotes di grado n;
%
if n<=0
    error('grado della formula non positivo');
end
c=zeros(1,floor(n / 2 + 1));
for j = 1:(ceil((n+1)/2))
    temp = (0:n);
    vj = temp(j);
    temp(j)=[];
    f = @(x)(prod(x-temp) /prod(vj-temp));
    c(j) = integral(f, 0, n, 'ArrayValued', true);
end

c = [c flip(c)];
if mod(n,2)==0
    c(n/2+1) = [];
end
return
end
```

5.2 Esercizio 22

Sappiamo che $k = (b-a)$ e $k_n = (b-a)\frac{1}{n}\sum_{i=0}^{n}|c_{in}|$. Il rapporto sarà dunque dato da:

$$\frac{k_n}{k} = \frac{(b-a)\frac{1}{n}\sum_{i=0}^{n}|c_{in}|}{b-a} = \frac{1}{n}\sum_{i=0}^{n}|c_{in}|$$

Calcolando il rapporto per $n = 1, \ldots.., 50$ si ottiene:



5.3 Esercizio 23

```
function y = newtoncotes(f,a, b, n)
%
% y= newtoncotes(f,a,b, n)
% calcola l'approssimazione dell'integrale definito per la funzione f sull'intervallo [a, b],
% utilizzando la formula di newton cotes di grado n.
%

if a > b || n < 0
    error('dati inconsistenti');
end
xi = linspace(a, b, n+1);
fi = feval(f, xi);
h = (b-a) / n;
c = ncweights(n);
y = h*sum(fi.*c);
return
end
```

```
value = log(cos(1)/cos(1.1));
x = zeros(1,9);
errors=zeros(1, 9);
for i = 1:9
    x(i) = newtoncotes(@tan, -1,1.1, i);
    errors(i) = abs(value-x(i));
end
```

RISULTATI:

grado della formula	valore integrale	errore
1	0.428	0.253
2	0.213	0.038
3	0.196	0.021
4	0.180	0.005
5	0.179	0.004
6	0.176	0.001
7	0.176	0.001
8	0.175	0.000
9	0.175	0.000