



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

1	Capitolo 1	3
1.1	Esercizio 1	3
1.2	Esercizio 2	3
1.3	Esercizio 3	3
2	Capitolo 2	4
2.1	Esercizio 4	4
2.2	Esercizio 5	4
2.3	Esercizio 6	6
3	Capitolo 3	8
3.1	Esercizio 8	8
3.2	Esercizio 11	8
3.3	Esercizio 12	9
4	Capitolo 4	10
5	Capitoli 5/6	11

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:

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

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 $a = 1e20$. Sottrarre un numero da se stesso ha sempre risultato esatto 0.
- nel secondo caso la sottrazione avviene 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

```
1 function x1=radn(x, n)
2 %
3 % x1=radn(n.x)
4 % funzione Matlab che implementa il metodo di newton per il calcolo della
5 % radice n-esima di un numero positivo x
6 %
7 format long e
8 imax=1000;
9 tolx=eps;
10 if x<=0
11     error('valore in ingresso errato');
12 end
13 x1=x/2;
14 for i=1:imax
15     x0=x1;
16     fx=x0^n-x;
17     fx1=(n)*x0^(n-1);
18     x1=x0-fx/fx1;
19     if abs(x1-x0)<=tolx
20         break
21     end
22 end
23 if abs(x1-x0)>tolx
24     error('metodo non converge')
25 end
26 end
```

2.2 Esercizio 5

- Metodo di bisezione

```
1 function [x,i] = bisez(f,a,b,tolx)
2 %
3 % [x, i] = bisez(f, a, b, tolx)
4 % calcola la radice di f(x) utilizzando il metodo di bisezione sull'intervallo [
5 % a, b]
6 format long e
7 fa = feval(f,a);
8 fb = feval(f,b);
9 if(fa * fb > 0 )
10     error('gli estremi hanno lo stesso segno');
11 end
12 imax = ceil(log2(b-a) - log2(tolx));
13 for i = 1:imax
14     x = (a+b)/2;
15     fx = feval(f,x);
16     f1x = abs((fb-fa)/(b-a));
17     if abs(fx) <= tolx*f1x
18         break
19     elseif fa*fx<0
20         b = x;
21         fb = fx;
22     else
23         a = x;
```

```

23         fa = fx;
24     end
25 end
26
27 end

```

- Metodo di Newton

```

1  function [x,i] = newton( f, f1, x0, tolx, maxit )
2      %
3      % [x,i] = newton( f, f1, x0, tolx [, maxit] )
4      %
5      % Metodo di Newton per determinare una approssimazione
6      % della radice di f(x)=0 con tolleranza tolx, a
7      % partire da x0, entro maxit iterazioni (default = 100).
8
9      format long e
10     if nargin<4
11         error('numero argomenti insufficienti');
12     elseif nargin==4
13         maxit = 100;
14     end
15     if tolx<eps
16         error('tolleranza non idonea');
17     end
18     x = x0;
19     for i = 1:maxit
20         fx = feval( f, x );
21         f1x = feval( f1, x );
22         x = x - fx/f1x;
23         if abs(x-x0)<=tolx*(1+abs(x0))
24             break;
25         else
26             x0 = x;
27         end
28     end
29     if abs(x-x0) > tolx*(1+abs(x0))
30         error('metodo non converge');
31     end
32 end

```

- Metodo delle secanti

```

1  function [x, i]=secanti(f,x0,x1,tolx,maxit)
2      %
3      % [x,i] = secanti( f, x0, x1, tolx [, maxit] )
4      %
5      % Metodo delle secanti per determinare una approssimazione
6      % della radice di f(x)=0 con tolleranza tolx, a
7      % partire da x0, entro maxit iterazioni (default = 100).
8
9      format long e
10     if nargin<4
11         error('numero argomenti insufficienti');
12     elseif nargin==4
13         maxit = 100;
14     end
15     i=0;
16     f0=feval(f,x0);
17     for i=1:maxit

```

```

17     f1=feval(f,x1);
18     df1=(f1-f0)/(x1-x0);
19     x=x1-(f1/df1);
20     if abs(x1-x0)<=tolx*(1+abs(x0))
21         break;
22     end
23     x0=x1;
24     x1=x;
25     f0=f1;
26
27 end
28 if abs(x-x0) > tolx*(1+abs(x0))
29     error('metodo non converge');
30 end
31 end

```

- Metodo delle corde

```

1 function [x,i] = corde( f, f1, x0, tolx, maxit )
2 %
3 % [x,i] = corde( f, f1, x0, tolx [, maxit] )
4 %
5 % Metodo delle corde per determinare una approssimazione
6 % della radice di f(x) con tolleranza tolx, a
7 % partire da x0, entro maxit iterazioni (default = 100).
8
9 format long e
10 if nargin<4
11     error('numero argomenti insufficienti');
12 elseif nargin==4
13     maxit = 100;
14 end
15 if tolx<eps
16     error('tolleranza non idonea');
17 end
18 flx = feval(f1, x0);
19 x = x0;
20 for i = 1:maxit
21     fx = feval( f, x );
22     if fx==0
23         break;
24     end
25     x = x - fx/flx;
26     if abs(x-x0)<=tolx*(1+abs(x0))
27         break;
28     else
29         x0 = x;
30     end
31 end
32 if abs(x-x0) > tolx*(1+abs(x0))
33     error('metodo non converge');
34 end
35 end

```

2.3 Esercizio 6

```

1 f = @(x)(x-cos(x));

```

```

2  f1 = @(x)(1+sin(x));
3
4  x0 = 0;
5  x1 = 1;
6  for i=3:3:12
7      [x_bisez, it_bisez] = bisez(f, x0, x1, 10^(-i))
8      [x_newt, it_newt] = newton(f, f1, x0, 10^(-i))
9      [x_corde, it_corde] = corde(f, f1, x0, 10^(-i))
10     [x_secanti, it_secanti] = secanti(f, x0, x1, 10^(-i), 100)
11 end

```

Eseguendo questo codice si ottengono i seguenti risultati:

• tolleranza = 10^{-3}	Metodo	Valore	iterazioni
	Bisezione	0.7382812500000000	8
	Newton	0.7390851333852840	4
	Secanti	0.7390851332150012	5
	Corde	0.7395672022122561	17

• tolleranza = 10^{-6}	Metodo	Valore	iterazioni
	Bisezione	0.7390842437744141	19
	Newton	0.7390851332151607	5
	Secanti	0.7390851332151607	6
	Corde	0.7390845495752126	34

• tolleranza = 10^{-9}	Metodo	Valore	iterazioni
	Bisezione	0.7390851341187954	28
	Newton	0.7390851332151607	5
	Secanti	0.7390851332151607	7
	Corde	0.7390851327392538	52

• tolleranza = 10^{-12}	Metodo	Valore	iterazioni
	Bisezione	0.7390851332147577	39
	Newton	0.7390851332151607	6
	Secanti	0.7390851332151607	7
	Corde	0.7390851332157368	69

3 Capitolo 3

3.1 Esercizio 8

```
1 function [LU,p]=palu(A)
2 % [LU,p]=palu(A)
3 % funzione Matlab che dato in input matrice A restituisce matrice fattorizzata LU
4 % e il relativo vettore p di permutazione di LU con pivoting parziale di A
5
6 [n,m]=size(A);
7 if(n~=m)
8     error(matrice A non quadrata);
9 end
10 LU=A;
11 p=[1:n];
12 for i=1:n-1
13     [mi,ki]=max(abs(LU(i:n,i)));
14     if mi==0
15         error('La matrice e'' non singolare')
16     end
17     ki=ki+i-1;
18     if ki>i
19         LU([i ki],:);
20         p([i ki])=p([ki i]);
21     end
22     LU(i+1:n,1)=LU(i+1:n,i)/LU(i,i);
23     LU(i+1:n,i+1:n)=LU(i+1:n,i+1:n)-LU(i+1:n,i)*LU(i,i+1:n);
24 end
```

3.2 Esercizio 11

```
1 function QR = myqr(A)
2 %QR = myqr(A)
3 % calcola la fattorizzazione QR di Householder della matrice A
4
5 [m,n] = size(A);
6 if n > m
7     error('Dimensioni errate');
8 end
9 QR = A;
10 for i = 1:n
11     alfa = norm(QR(i:m,i));
12     if alfa == 0
13         error('la matrice non ha rango massimo');
14     end
15     if QR(i,i) >= 0
16         alfa = -alfa;
17     end
18     v1 = QR(i,i) - alfa;
19     QR(i,i) = alfa;
20     QR(i+1:m,i) = QR(i+1:m,i)/v1;
21     beta = -v1/alfa;
22     v = [1; QR(i+1:m,i)];
23     QR(i:m,i+1:n) = QR(i:m,i+1:n) - (beta * v) * (v' * QR(i:m,i+1:n));
24 end
25 end
```


3.3 Esercizio 12

```
1 function x = qrsolve(QR, b)
2 %
3 %
4 % x = qrSolve(QR, b)
5 % risolve il sistema  $QR \cdot x = b$  nel senso dei minimi quadrati
6 %
7 [m,n] = size(QR);
8 k = length(b);
9 if k ~= m
10     error('Dati inconsistenti');
11 end
12 x=b(:);
13 for i = 1:n
14     v=[1; QR(i+1:m,i)];
15     beta = 2/(v'* v);
16     x(i:m) = x(i:m) - (beta*(v'*x(i:m))*v);
17 end
18 x=x(1:n);
19 for j = n:-1:1
20     if QR(j,j)==0
21         error('Matrice singolare');
22     end
23     x(j) = x(j) / QR(j,j);
24     x(1:j-1) = x(1:j-1) - QR(1:j-1,j)*x(j);
25 end
26 return
27 end
```

4 Capitolo 4

5 Capitoli 5/6