

Relazioni di Calcolo Numerico

Lorenzo Pattarini

1 dicembre 2015

Indice

1	Esercitazione 1	3
1.1	Esercizio 1	3
1.2	Esercizio 2	3
1.3	Esercizio 3	4
1.4	Esercizio 4	4
1.5	Esercizio 5	5
1.6	Esercizio 6	5
2	Esercitazione 2	6
2.1	Esercizio 1	6
2.2	Esercizio 2	6
2.3	Esercizio 3	6
2.4	Esercizio 4	8
2.5	Esercizio 5	9
2.6	Esercizio 6	10
2.7	Esercizio 7	11
2.8	Esercizio 8	
	Sviluppo di una funzione matlab per il calcolo delle radici di un'equazione di quarto grado	12
2.9	Esercizio 9	13
3	Esercitazione 3	14
3.1	Esercizio 1	14
3.2	Esercizio 2	
	Calcolo di $\lim_{x \rightarrow 0} \frac{(x+1)^2-1}{x}$ e $\lim_{x \rightarrow 0} x + 2$	15
3.3	Esercizio 3	
	Sviluppo di Taylor per il calcolo di e^x	17
3.4	Esercizio 4	
	Approssimazioni della derivata di e^x nel punto $x = 1$	19
3.5	Esercizio 5	20
4	Esercitazione 4	21
4.1	Esercizio 1	
	Interpolazione con Vandermonde e perturbazione dei dati	21
4.2	Esercizio 2	24
5	Esercitazione 5	25
5.1	25
5.2	Esercizio 2	28
5.3	Esercizio 3	29
5.4	Esercizio 4	
	Risoluzione di sistemi lineari a matrice tridiagonale	31
5.5	Esercizio 5	33
5.6	Esercizio 6	34
5.7	Esercizio 7	35
5.8	Esercizio 8	
	Approssimazione di π mediante troncata di una serie	36

6	Esercitazione 6	38
6.1	Esercizio 1	38
6.2	Esercizio 2	40
6.3	Esercizio 3	42
6.4	Esercizio 4	43
6.5	Esercizio 5	45
6.6	Esercizio 6	46
6.7	Esercizio 7	48
6.8	Esercizio 8	50
7	Esercitazione 7	51
7.1	Esercizio 1	
	Costruzione di una formula di quadratura per il calcolo dell' integrale definito $I = \int_0^1 e^{-x^2} dx$. .	51
7.2	Esercizio 2	
	Costruzione di un grafico per la rappresentazione dell'errore relativo nel calcolo delle formule di Newton-Cotes approssimanti l'integrale definito $I = \int_0^1 e^{-x^2} dx$	53
7.3	Esercizio 3	
	Costruzione di un grafico per la rappresentazione degli errori relativi nel calcolo dell'integrale definito $\int_0^1 \sin(x) dx$ mediante le formule dei trapezi e di Cavalieri-Simpson iterate	56
8	Esercitazione 8	58
8.1	Esercizio 1	
	Utilizzo della tecnica del pivoting per la fattorizzazione LU	58
8.2	Esercizio 2	
	Implementazione dei metodi di Jacobi e di Gauss-Seidel per la soluzione approssimata di sistemi lineari1	63

1 Esercitazione 1

1.1 Esercizio 1

```
1 a = 1.12;
2 b = 2.34;
3 c = 0.72;
4 d = 0.81;
5 e = 3;
6 f = 19.83;
7 g = 20;
8
9 x = 1+a/b+c/(f^2);
10 s = (b-a)/(d-c);
11 z = (1-(1/(e^5)))^(-1);
12 r = 1/(1/a+1/c+1/c+1/d);
13 y = a*b*(1/c)*(f^2/2);
14 t = 7*(g^(1/3))+4*g^(0.58);
15
16 format short
17 x s z r y t
18
19 format long
20 x s z r y t
```

1.2 Esercizio 2

```
1 % Ricerca degli errori sintattici
2
3 a = 2y+(((3+1)9);
4
5 % Manca una parentesi e la variabile y non risulta inizializzata
6
7 b == 2*sin[3];
8
9 %{
10     L'operatore "==" \'{e} un operatore di testing non di
11     assegnamento, ma non essendo la variabile b inizializzata
12     non ha senso testarne l'uguaglianza.
13     Il secondo errore \'{e} l'utilizzo delle parentesi quadre in quanto,
14     per riferirsi all'argomento di una funzione vanno usate le
15     parentesi tonde.
16 %}
17
18 c = e^0.5;
19
20 %{
21     Il modo corretto per ottenere la radice del numero di nepero
22     \'{e} c = exp(0.5).
23 %}
24
25 d = log(4-8/4*2)
26
27 %{
28     In questo modo non essendo certi della precedenza degli operatori
29     rischiamo di tentare di calcolare log(0), occorre aggiungere le
30     parentesi e riscrivere la formula come
31     d = log(4-8/(4*2))
32 %}
```

1.3 Esercizio 3

```
1 r1 = 5
2 r1 = 5
3 V1 = 4/3*pi*r1^2
4 V1 = 104.72
5 V2 = V1 + (30/100)*V1
6 V2 = 136.14
7 r2 = ((V2/pi)*(3/4))^(1/3)
8 r2 = 10.833
9
10 ex = inline(" 1+x ");
11 ex2 = inline(" 1+x+(x^2)/2 ");
12 % errore assoluto
13 ea1 = abs(e^(0.1)-ex(0.1))
14 ea1 = 0.0051709
15 ea2 = abs(e^(0.1)-ex2(0.1))
16 ea2 = 1.7092e-04
17 % errore relativo
18 er1 = (abs(e^(0.1)-ex(0.1)))/e^(0.1)
19 er1 = 0.0046788
20 er2 = (abs(e^(0.1)-ex2(0.1)))/e^(0.1)
21 er2 = 1.5465e-04
22
23
24 % Radici di un polinomio
25 % uso il vettore dei coefficienti
26 p1 = [ 2 -4 -1 ];
27 p2 = [ 1 0 2 0 -3 ];
28 p3 = [ 1 0 0 2197];
29 s1 = roots(p1)
30 s1 =
31
32     2.22474
33    -0.22474
34
35 s2 = roots(p2)
36 s2 =
37
38    -0.00000 + 1.73205 i
39    -0.00000 - 1.73205 i
40    -1.00000 + 0.00000 i
41     1.00000 + 0.00000 i
42
43 s3 = roots(p3)
44 s3 =
45
46   -13.0000 + 0.0000 i
47    6.5000 + 11.2583 i
48    6.5000 - 11.2583 i
```

1.4 Esercizio 4

```
1 x = 1:10;
2 y = [10:-1:1]';
3
```

```

4 xy = x*y;
5
6 xs = linspace(0,1,11);
7 z = sin(xs);

```

1.5 Esercizio 5

```

1 x = 25:3:91;
2 y = [100:-1:10]';
3 z = linspace(-15,-10,33);

```

1.6 Esercizio 6

```

1 z = 1+i;
2
3 d_m_zn = sqrt(2)^60*(cos(60*(pi/4))+i*sin(60*(pi/4)));
4
5 mat_zn = z^60;

```

2 Esercitazione 2

2.1 Esercizio 1

```
1 %%%% Parte a %%%%
2
3 x = [ -3, 5, 8, 0, 1, 5, -2, 4 ];
4 x(6) = 100;
5 x(1:3) = [ 5, 6, 7];
6 x(4) = [];
7 x(4:7) = [];
8 x = [1,2,3,x];
9 x = [x, 10, 11, 12];
10
11 %%%% Parte b %%%%
12
13 A = eye(4);
14 A(1,1) = A(3,4);
15 A = [ ones(4,1), A];
16 A = [ A , ones(4,1) ];
17 A = [ 4*ones(1,6) ; A ]
18 A = [ A ; 4*ones(1,6) ]
19 A(3,:) = [];
20 A(:,3) = [];
```

2.2 Esercizio 2

```
1 x = [1 : -0.1 : 0];
2
3 % Estrazione degli elementi di indice 1 4 3 del vettore x
4 x([1 4 3]);
5
6 % Sostituzione degli elementi di indice 1 3 5 7 con 0.5
7 % dell'elemento di indice 10 con il valore -0.3
8 x([1:2:7 10] = [ 0.5*ones(1,2) -0.3 ]);
9
10 % \'{e} l'equivalente della scrittura y =fliplr(x):
11 y = x(end:-1:1);
```

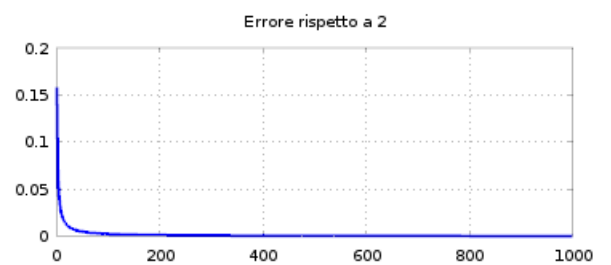
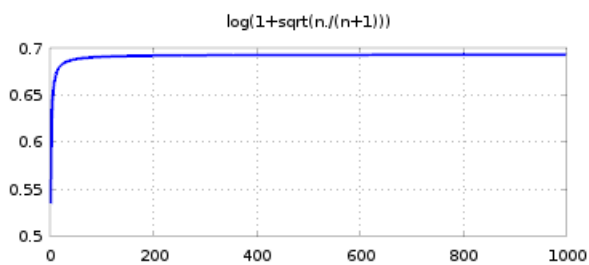
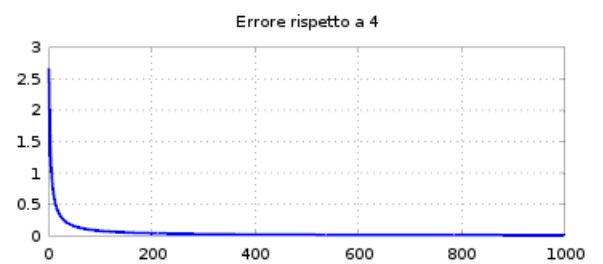
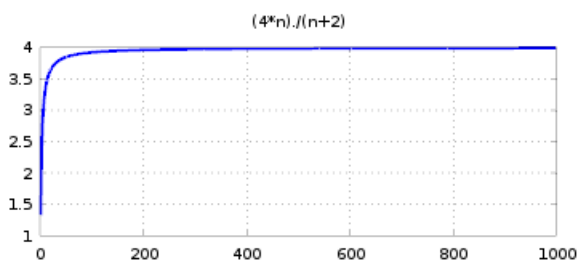
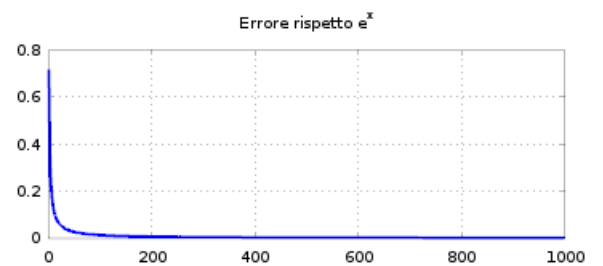
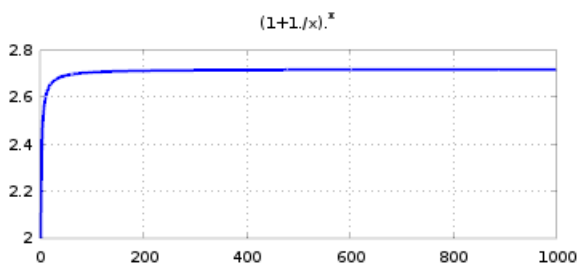
2.3 Esercizio 3

```
1 f1 = inline("(1+1./x).^x");
2 f2 = inline("(4*n)./(n+2)");
3 f3 = inline("log(1+sqrt(n./(n+1)))");
4
5 x = 1:2:1000;
6
7 y1 = f1(x);
8 y2 = f2(x);
9 y3 = f3(x);
10
11 n = size(x);
12
13 err1 = abs(ones(1,n)*exp(1) - y1);
```

```

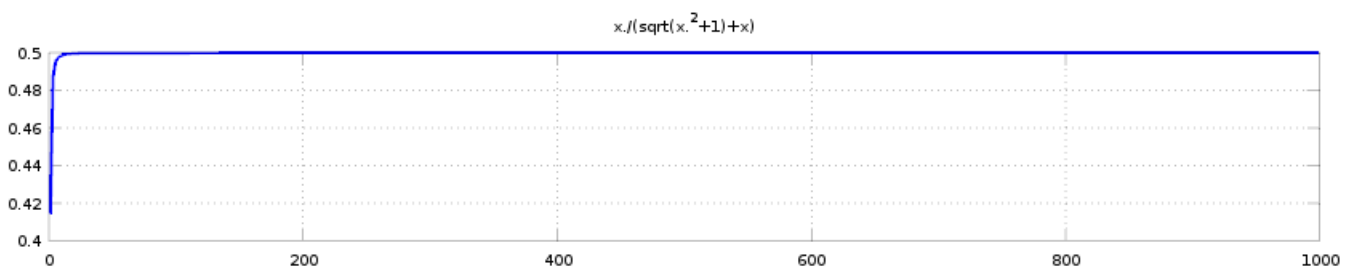
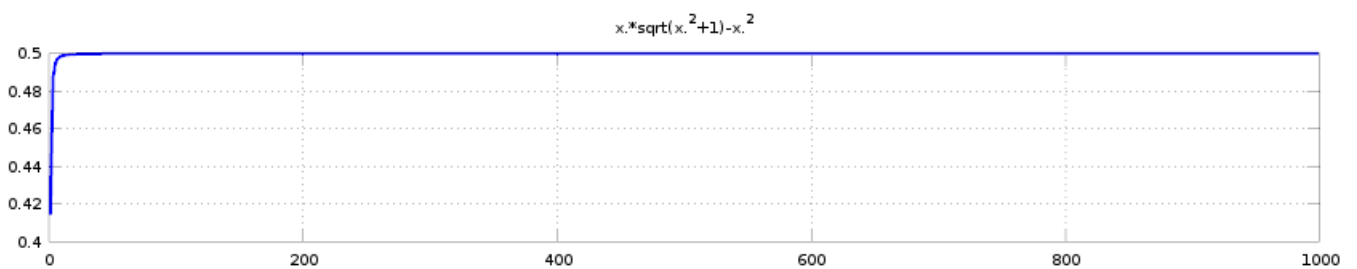
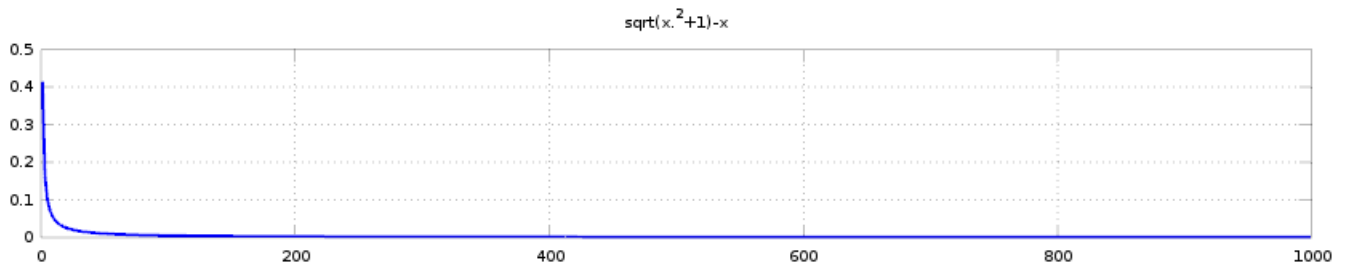
14 err2 = abs(ones(1,n)*4 - y2);
15 err3 = abs(ones(1,n)*log(2) - y3);
16
17 subplot(3,2,1)
18     plot(x,y1,'linewidth',2);
19 subplot(3,2,2)
20     plot(x,err1,'linewidth',2);
21 subplot(3,2,3)
22     plot(x,y2,'linewidth',2);
23 subplot(3,2,4)
24     plot(x,err2,'linewidth',2);
25 subplot(3,2,5)
26     plot(x,y3,'linewidth',2);
27 subplot(3,2,6)
28     plot(x,err3,'linewidth',2);

```



2.4 Esercizio 4

```
1 f1 = inline("sqrt(x.^2+1)-x");
2 f2 = inline("x.*sqrt(x.^2+1)-x.^2");
3 f3 = inline("x./(sqrt(x.^2+1)+x)");
4
5 x = 1:2:1000;
6
7 y1 = f1(x);
8 y2 = f2(x);
9 y3 = f3(x);
10
11
12 subplot(3,1,1)
13     plot(x,y1,'linewidth',2);
14     title("sqrt(x.^2+1)-x");
15     grid on
16 subplot(3,1,2)
17     plot(x,y2,'linewidth',2);
18     title("x.*sqrt(x.^2+1)-x.^2");
19     grid on
20 subplot(3,1,3)
21     plot(x,y3,'linewidth',2);
22     title("x./(sqrt(x.^2+1)+x)");
23     grid on
```



2.5 Esercizio 5

```
1 d0 = -2*ones(1,9);  
2 d1 = ones(1,8);  
3  
4 A = diag(d0) + diag(d1,1) + diag(d1,-1);  
5 A([3,6], :) = A([6,3], :);  
6 A(:, [1,4]) = A(:, [4,1]);
```

2.6 Esercizio 6

```
1 x = [1:12];
2 A = [x([1:4]);x([5:8]);x([9:12])] ;
3 size(A)
4 B = A.*A
5 B = A*A
6 B = A'*A
7 A(1:2,4)
8 A(:,3)
9 A(1:2,:)
10 A(:,[2 4])
11 A([2 3 3])
12 A(3,2)=A(1,1)
13 A(1:2,4) = zeros(2,1);
14 A(2,:) = A(2,:)-A(2,1)/A(1,1)*A(1,:);
```

2.7 Esercizio 7

```
1  a = ones(1,8)
2  A = [];
3  for b= 1:8
4      A(b,:) = b*a;
5  endfor
6
7  S = triu(A)
8  L = tril(A)
9
10 for i=1:size(S)
11     S(i,i) = 0;
12     L(i,i) = 1;
13 endfor
14
15 d0 = diag(A);
16 d1 = diag(A,1);
17 d_1 = diag(A,-1);
18
19 B1 = diag(d0) + diag(d1,1) + diag(d_1,-1);
20 B2 = diag(d0) + diag(d1,1);
21 B3 = diag(d0) + diag(d_1,-1);
```

2.8 Esercizio 8

Sviluppo di una funzione matlab per il calcolo delle radici di un'equazione di quarto grado

```
1 function X = lp_root(alpha)
2
3     b = (1+10^(2*alpha))/10^alpha;
4     t1 = (b + sqrt(b^2-4))/2;
5     t2 = (b - sqrt(b^2-4))/2;
6
7     x1 = -sqrt(t1);
8     x2 = -sqrt(t2);
9     x3 = sqrt(t1);
10    x4 = sqrt(t2);
11
12    X = [ x1, x2, x3, x4 ];
13
14 endfunction
15
16
17 A = [];
18
19 for i=1:10
20     A = [ A ; lp_root(i) ];
21 endfor
22
23
24 B = [];
25
26 for i=1:10
27     B = [ B ; roots([ 1, 0, -((1+10^(2*i))/10^i), 0, 1])' ];
28 endfor
29
30
31 for i=1:10
32     A(i,:) = sort(A(i,:));
33     B(i,:) = sort(B(i,:));
34 endfor
35
36 E1 = abs(A-B)./A;
37 E2 = abs(A-B)./B;
```

2.9 Esercizio 9

3 Esercitazione 3

3.1 Esercizio 1

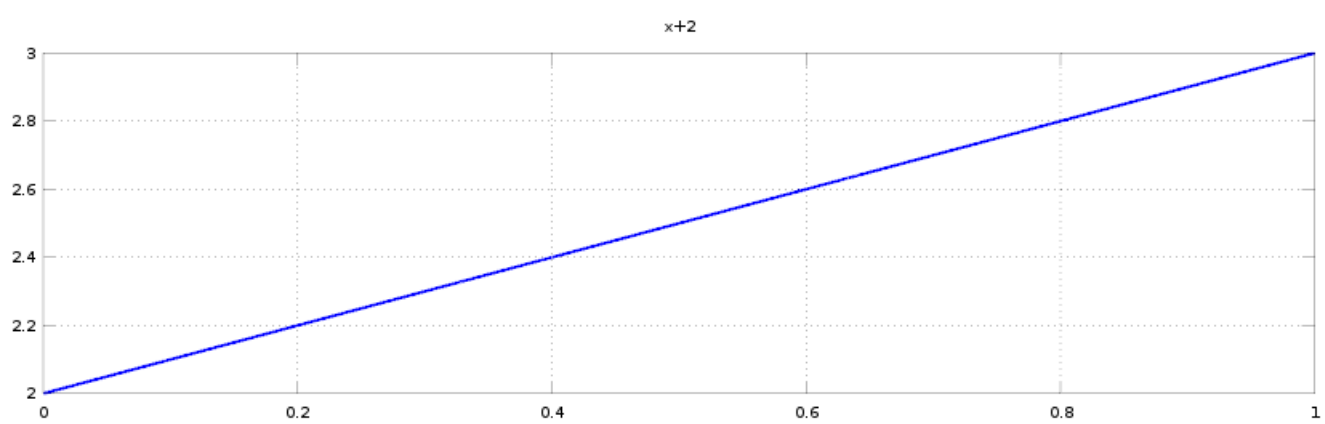
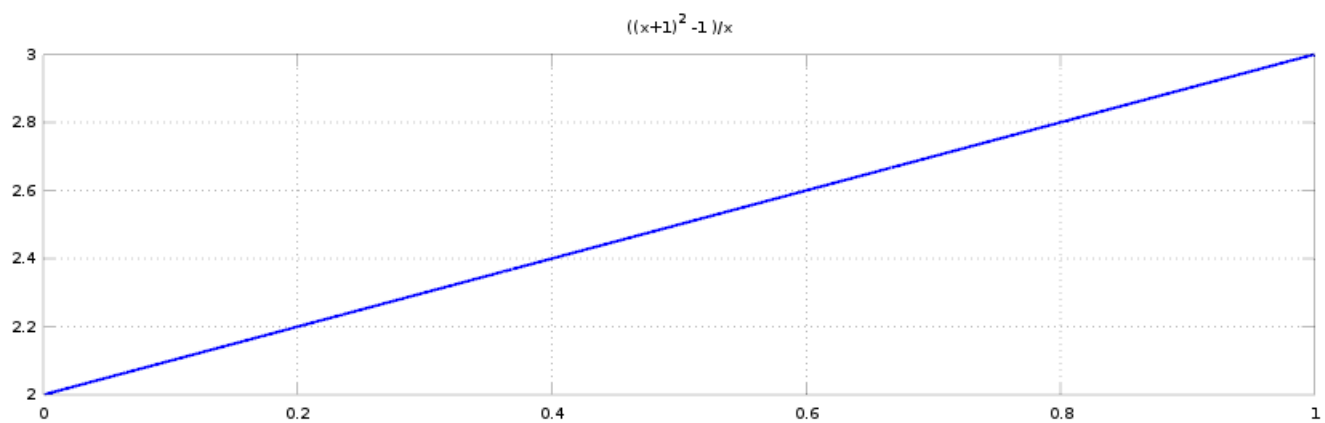
```
1 x = -5:1:9;
2 M = max(x)
3 M = 9
4 m = min(x)
5 m = -5
6 abM = max(abs(x))
7 abM = 9
8 abm = min(abs(x))
9 abm = 0
10 S = sum(x)
11 S = 30
12 abS = sum(abs(x))
13 abS = 60
```

3.2 Esercizio 2

Calcolo di $\lim_{x \rightarrow 0} \frac{(x+1)^2-1}{x}$ e $\lim_{x \rightarrow 0} x+2$

```
1  f1 = inline (" ((x+1).^2 -1)./x ");
2  f2 = inline (" x+2 ");
3
4  x = 1:-0.001:0;
5
6  y1 = f1(x);
7  y2 = f2(x);
8
9  subplot(2,1,1)
10     plot(x,y1,'linewidth',2);
11     title(" ((x+1)^2 -1 )/x ");
12     grid on
13 subplot(2,1,2)
14     plot(x,y2,'linewidth',2);
15     title(" x+2 ")
16     grid on

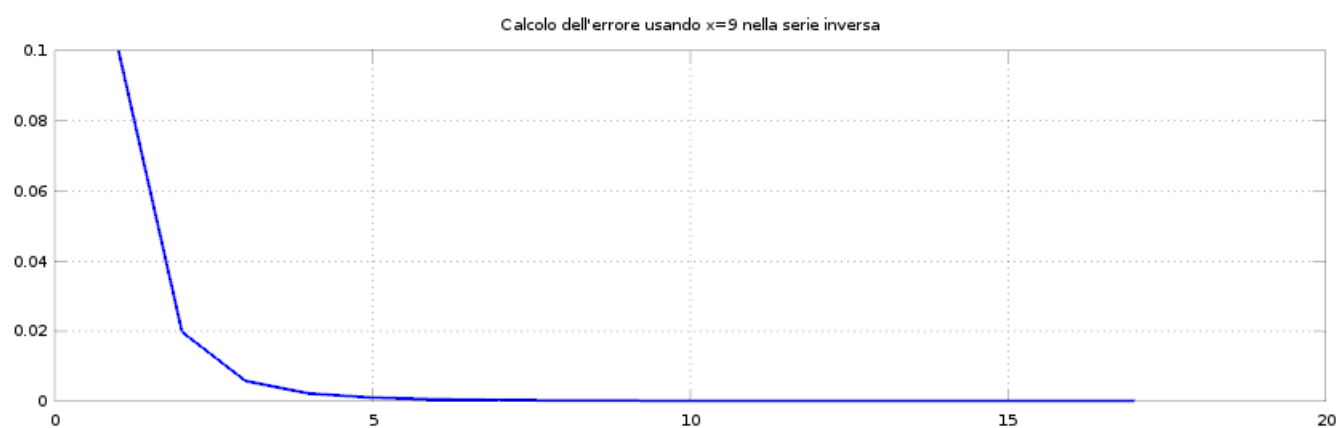
1  f1 = inline (" ((x+1).^2 -1)./x ");
2  f2 = inline (" x+2 ");
3
4  x = 1:-0.001:0;
5
6  y1 = f1(x);
7  y2 = f2(x);
8
9  subplot(2,1,1)
10     plot(x,y1,'linewidth',2);
11     title(" ((x+1)^2 -1 )/x ");
12 subplot(2,1,2)
13     plot(x,y2,'linewidth',2);
14     title(" x+2 ")
```



3.3 Esercizio 3

Sviluppo di Taylor per il calcolo di e^x

```
1 % e^x = 1 + x^2/2! + x^3/3! + ....
2 % Calcolare e^-9 stabilizzando la 6 cifra dopo la virgola
3
4
5 x1 = [];
6 x2 = [];
7
8 err = 10^-6;
9
10 my_e1 = 1;
11 x = -9;
12 i = 1;
13 while ( abs(e^-9 - my_e1) > err )
14     my_e1 = my_e1 + x^i/factorial(i);
15     i = i+1;
16     x1 = [ x1 , my_e1 ];
17 endwhile
18
19 e2 = 1;
20 my_e2 = 1;
21 x = 9;
22 i = 1;
23 while ( abs(e^-9 - my_e2) > err )
24     e2 = e2 + x^i/factorial(i);
25     my_e2 = 1/e2;
26     i = i+1;
27     x2 = [ x2, my_e2 ];
28 endwhile
29
30
31
32
33
34 % Calcolo degli errori relativi
35 % Calcolati sulla somma dei primi n termini
36
37
38 err_rel1 = [];
39 for i=1:size(x1)(2)
40     err_rel1 = [ err_rel1 , abs(x1(i) - exp(-9))/exp(-9) ];
41 endfor
42
43 err_rel2 = [];
44 for i=1:size(x2)(2)
45     err_rel2 = [ err_rel2 , abs(x2(i) - exp(-9))/exp(-9) ];
46 endfor
47
48
49 a1 = 1:size(x1)(2);
50 a2 = 1:size(x2)(2);
51
52 subplot(2,1,1)
53 plot(a1,x1,'linewidth',2);
54
55 subplot(2,1,2)
56 plot(a2,x2,'linewidth',2);
```



3.4 Esercizio 4

Approssimazioni della derivata di e^x nel punto $x = 1$

```

1  h = [];
2
3  for a=1:1:20
4      h = [ h, 10^(-a) ];
5  endfor
6
7  x = 1;
8  app1 = (exp(x+h)-exp(x))./h;
9  app2 = (exp(x+h)-exp(x-h))./(2*h);
10
11  analitic1 = abs((app1 - exp(1)));
12  analitic2 = abs((app2 - exp(1)));
13
14  tab = [ h', analitic1', analitic2' ]
15
16  #{
17      1.0000e-01    1.4056e-01    4.5327e-03
18      1.0000e-02    1.3637e-02    4.5305e-05
19      1.0000e-03    1.3596e-03    4.5305e-07
20      1.0000e-04    1.3592e-04    4.5306e-09
21      1.0000e-05    1.3591e-05    5.8587e-11
22      1.0000e-06    1.3590e-06    1.6346e-10
23      1.0000e-07    1.3995e-07    5.8587e-11
24      1.0000e-08    6.6028e-09    6.6028e-09
25      1.0000e-09    2.1544e-07    6.6028e-09
26      1.0000e-10    1.5477e-06    6.7274e-07
27      1.0000e-11    3.2634e-05    1.0429e-05
28      1.0000e-12    4.3231e-04    2.1027e-04
29      1.0000e-13    4.5586e-04    4.5586e-04
30      1.0000e-14    9.3376e-03    9.3376e-03
31      1.0000e-15    3.9034e-01    1.6830e-01
32      1.0000e-16    2.7183e+00    2.7183e+00
33      1.0000e-17    2.7183e+00    2.7183e+00
34      1.0000e-18    2.7183e+00    2.7183e+00
35      1.0000e-19    2.7183e+00    2.7183e+00
36      1.0000e-20    2.7183e+00    2.7183e+00
37  #}

```

3.5 Esercizio 5

```
1 H = hilb(10);
2 x = ones(10,1);
3
4 b = H*x;
5
6 x1 = H\b;
7
8 z = [ 0.001, zeros(1,size(b)-1)]';
9
10 c = b+z ;
11
12 y = H\c;
13
14 err = norm(x-y)/norm(x)
15
16 err = 4852.2
```

4 Esercitazione 4

4.1 Esercizio 1

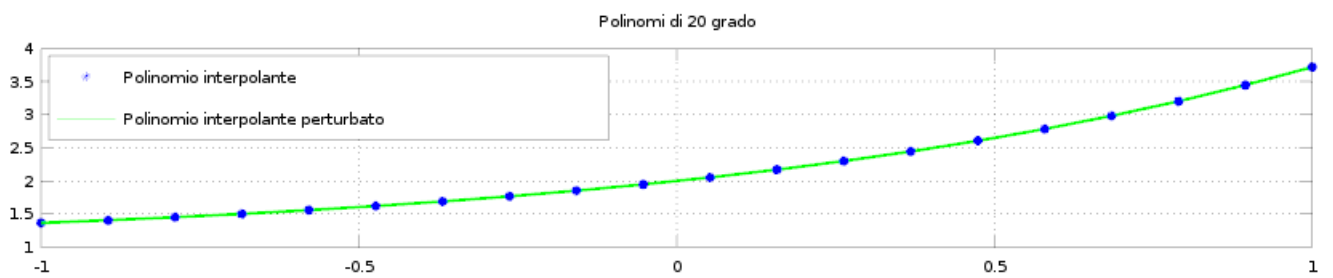
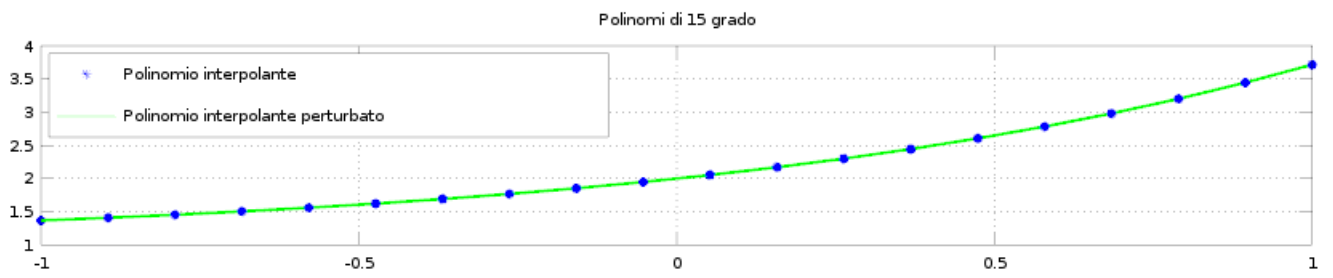
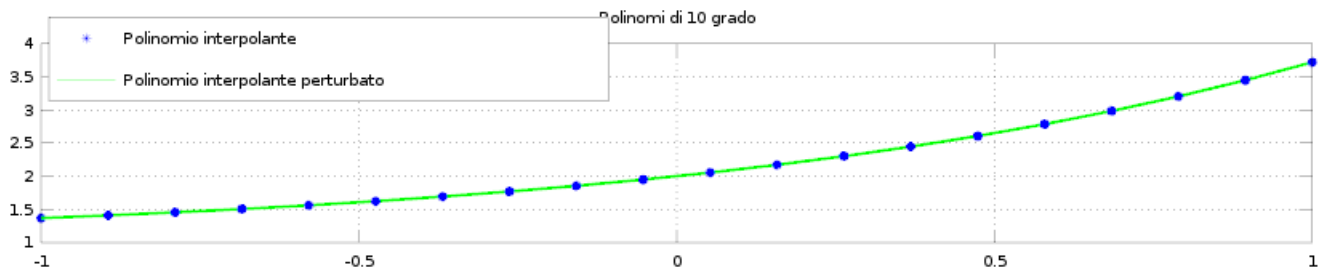
Interpolazione con Vandermonde e perturbazione dei dati

```
1 x10 = linspace(-1,1,10);
2 x15 = linspace(-1,1,15);
3 x20 = linspace(-1,1,20);
4
5 V10 = vander(x10);
6 V15 = vander(x15);
7 V20 = vander(x20);
8
9 f = inline(" exp(x)+1 ");
10
11 % I bi sono i coefficienti della funzione calcolati nei nodi
12 % mi servono ad imporre le condizioni di interpolazione.
13 b10 = f(x10)';
14 b15 = f(x15)';
15 b20 = f(x20)';
16
17 % I sistemi sarebbero genericamente  $Vx = b$ , devo trovare x
18 % che per comodit'\{a\} chiamer'\{o\} a, il vettore dei coefficienti.
19
20
21 % Coefficienti del polinomio
22 a10 = V10\b10;
23 a15 = V15\b15;
24 a20 = V20\b20;
25
26
27 eps = [];
28 for i=1:20
29     eps = [ eps, (-1)^(i)*10^(-5) ];
30 endfor
31
32 b10_1 = b10 + eps([1:10])';
33 b15_1 = b15 + eps([1:15])';
34 b20_1 = b20 + eps([1:20])';
35
36
37 % Coefficienti polinomio perturbato
38 a10_1 = V10\b10_1;
39 a15_1 = V15\b15_1;
40 a20_1 = V20\b20_1;
41
42
43
44
45 % Ricordiamo che Un polinomio in Matlab 'e dato da un vettore che contiene
46 % i suoi coefficienti ordinati da an fino ad a0.
47
48
49
50 a10 = fliplr(a10);
51 a15 = fliplr(a15);
52 a20 = fliplr(a20);
53 a10_1 = fliplr(a10_1);
54 a15_1 = fliplr(a15_1);
55 a20_1 = fliplr(a20_1);
56
```

```

57
58 % Uso x20 per valutare i polinomi
59 y10 = polyval(a10,x20);
60 y15 = polyval(a15,x20);
61 y20= polyval(a20,x20);
62 y10_1 = polyval(a10_1,x20);
63 y15_1 = polyval(a15_1,x20);
64 y20_1 = polyval(a20_1,x20);
65
66 , 'linewidth',2,
67
68
69 subplot(3,1,1)
70 plot(x20,y10, 'linewidth',2, '*',x20,y10_1, 'linewidth',2, 'g');
71 grid on
72 legend("Polinomio interpolante","Polinomio interpolante perturbato");
73 title("Polinomi di 10 grado");
74 subplot(3,1,2)
75 plot(x20,y15, 'linewidth',2, '*',x20,y15_1, 'linewidth',2, 'g');
76 grid on
77 legend("Polinomio interpolante","Polinomio interpolante perturbato");
78 title("Polinomi di 15 grado");
79 subplot(3,1,3)
80 plot(x20,y20, 'linewidth',2, '*',x20,y20_1, 'linewidth',2, 'g');
81 grid on
82 legend("Polinomio interpolante","Polinomio interpolante perturbato");
83 title("Polinomi di 20 grado");
84
85 maxa10 = max(abs(a10 - a10_1));
86 maxa15 = max(abs(a15 - a15_1));
87 maxa20 = max(abs(a20 - a20_1));
88
89 t = linspace(-1,1,101);
90
91 pt10 = polyval(a10,t);
92 pt10_1 = polyval(a10_1,t);
93 pt15 = polyval(a15,t);
94 pt15_1 = polyval(a15_1,t);
95 pt20 = polyval(a20,t);
96 pt20_1 = polyval(a20_1,t);
97
98
99 maxp10 = max(abs(pt10-pt10_1));
100 maxp15 = max(abs(pt15-pt15_1));
101 maxp20 = max(abs(pt20-pt20_1));

```



4.2 Esercizio 2

```
1 A = hilb(1000);
2 B = rand(1000);
3
4 x = ones(1000,1);
5 y = ones(1000,1);
6
7 % Ax = b;
8 % By = c;
9
10 b = A*x;
11 c = B*y;
12
13 x1 = A\x;
14 y1 = B\y;
15
16 % Calcolare gli errori relativi tra b,c b1 e c1
17 % rapportandoli al numero di condizionamento di A e B
18
19 Ka = cond(A);
20 Kb = cond(B);
21
22
23 err_relx = norm(x-x1)/norm(x);
24 err_rely = norm(y-y1)/norm(y);
```


5 Esercitazione 5

5.1

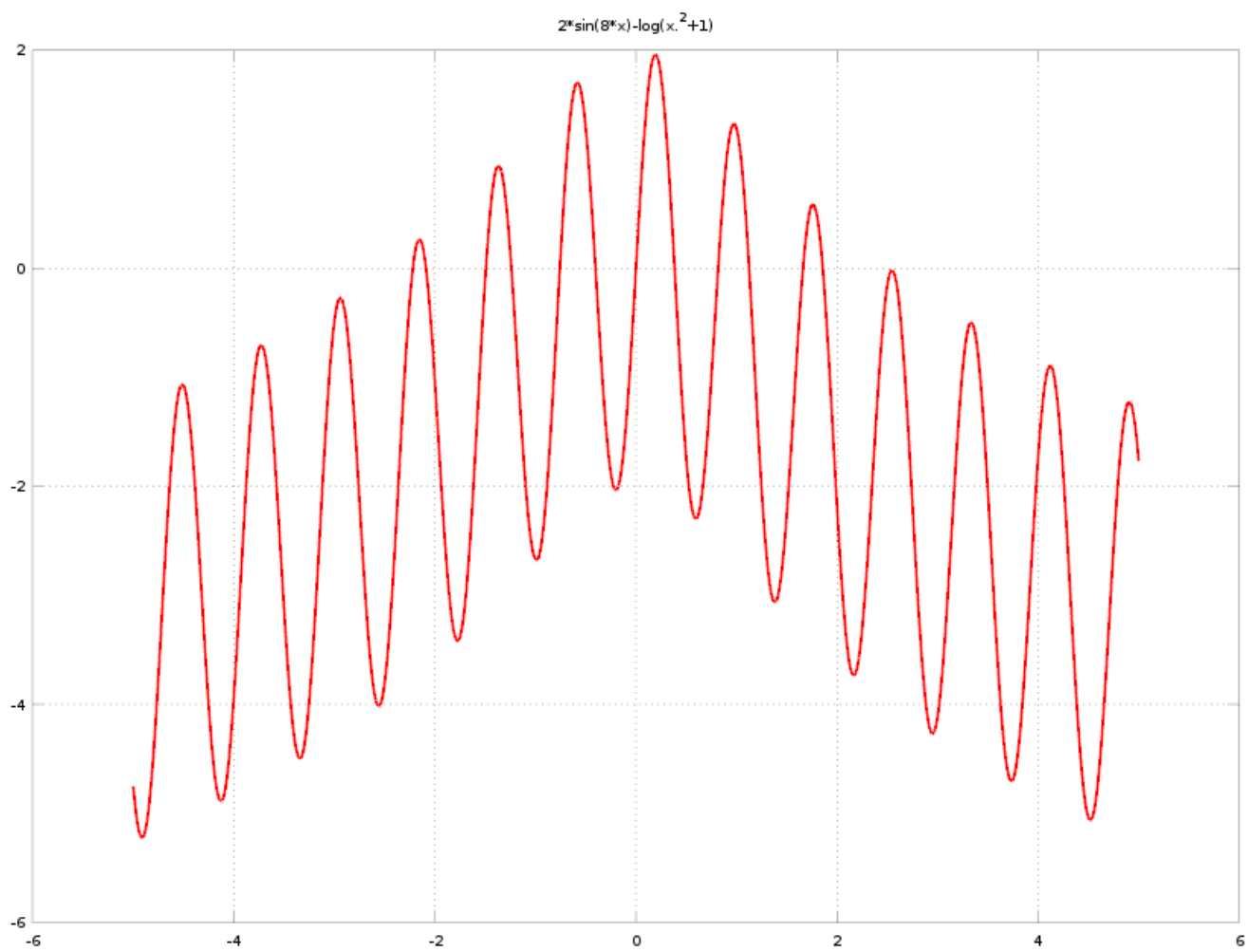
Esercizio 1

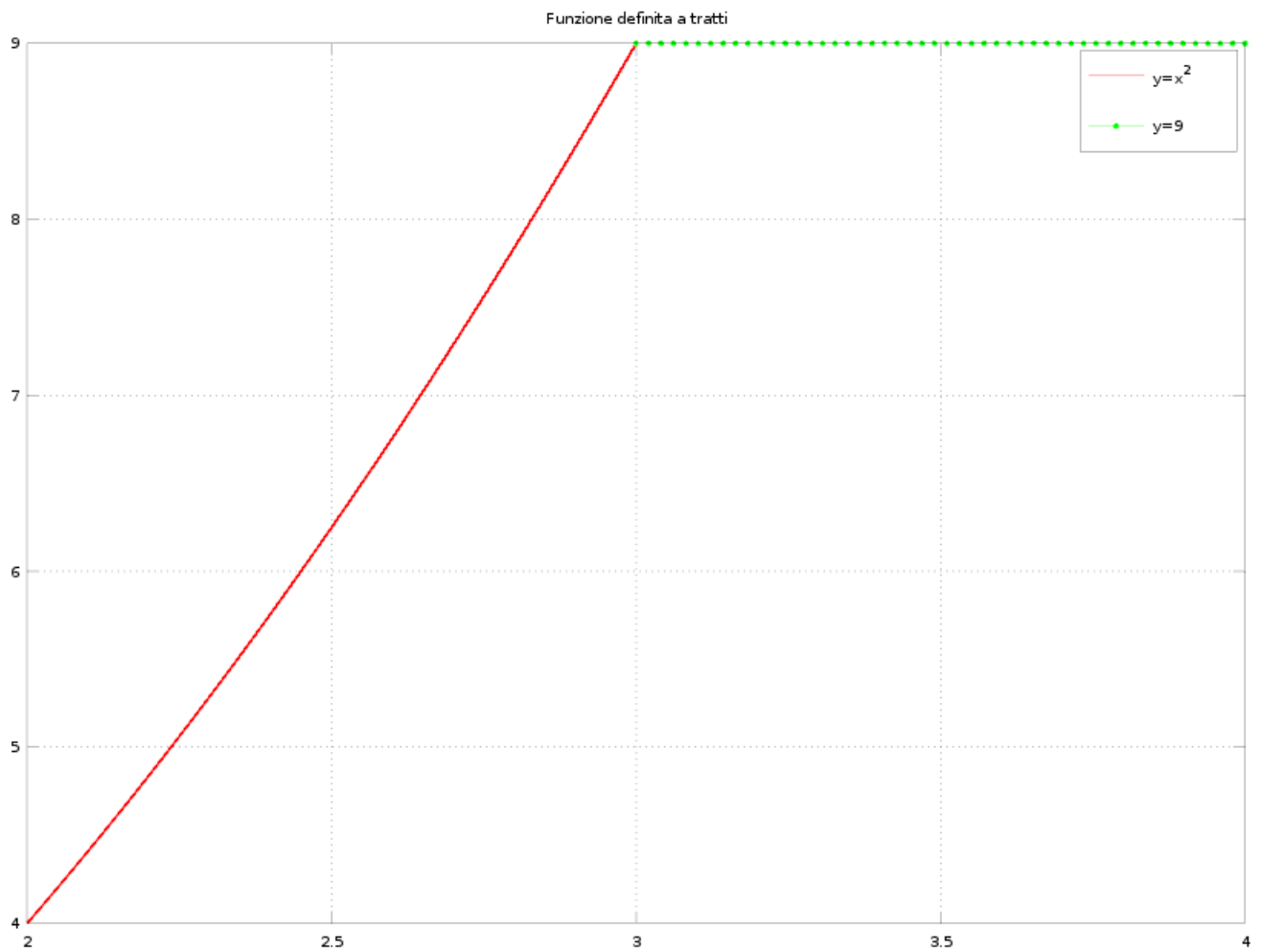
Rappresentazione grafica di funzioni

$$f(x) = 2\sin(8x) - \ln(x^2 + 1)$$

$$g(x) = \begin{cases} x^2 & x < 3 \\ 9 & x > 3 \end{cases}$$

```
1  %%%%%%%%% PARTE a %%%%%%%%%
2  function plot_f(a,b)
3
4      f = inline(" 2*sin(8*x) - log(x.^2+1) ");
5
6      if ( a < b )
7          c = a;
8          a = b;
9          b = c;
10     endif
11
12     x = linspace(a,b,(a-b)*100);
13
14     y = f(x);
15
16     plot(x,y,'r');
17     xlabel(" decomposition ");
18     ylabel(" 2*sin(8*x) - ln(x.^2+1) ");
19     grid on;
20
21 endfunction
22
23 plot_f(-6,6)
24
25 %%%%%%%%% PARTE a %%%%%%%%%
26
27 x1 = linspace(2,3,50);
28 x2 = linspace(3,4,50);
29
30 g1 = inline('x.^2');
31 plot(x1,g1(x1));
32
33 hold on
34
35 g2 = 9;
36 plot(x2,g2);
```

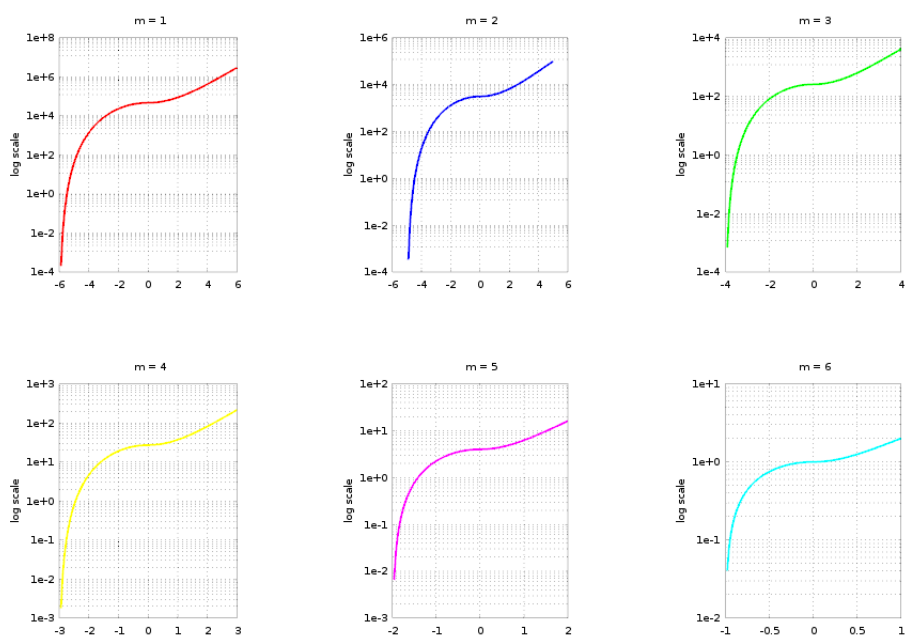
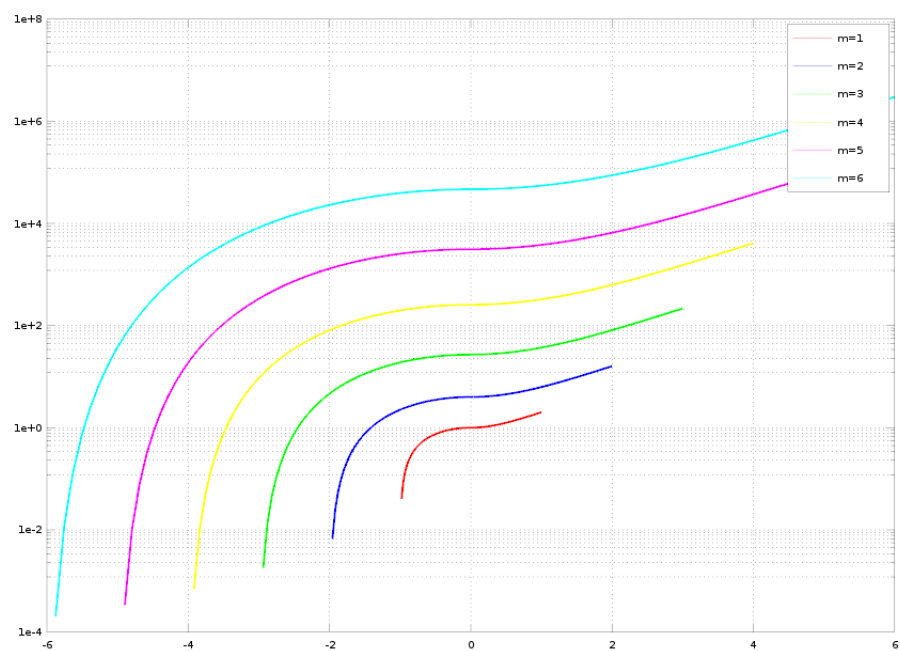




5.2 Esercizio 2

5.3 Esercizio 3

```
1 function [x,y] = m_calc(m)
2     m = abs(m);
3     x1 = linspace(-m,0,50);
4     x2 = linspace( 0,m,50);
5
6     y1 = (m-(x1.^2)./m).^m ;
7     y2 = ((x2.^2)./m+m).^m ;
8
9     x = [ x1 x2 ];
10    y = [ y1 y2 ];
11
12 endfunction
13
14 x=[];
15 y=[];
16
17 A = [];
18 for i=1:6
19     [ x,y ] = m_calc(i);
20     A = [ [ x, y ] ; A];
21     semilogy(x,y);
22     %legend([ " m = " num2str(i) ] );
23     hold on
24 endfor
25
26
27
28 figure 2
29 for i=1:6
30     subplot(2,3,i)
31     semilogy(A(i,1:100),A(i,101:200));
32     title([ " m = " num2str(i) ] );
33     ylabel(" log scale ");
34 endfor
```



5.4 Esercizio 4

Risoluzione di sistemi lineari a matrice tridiagonale

```
1 % Risoluzione di sistemi lineari con matrici tridiagonali
2 % Utilizzo la fattorizzazione LU per la matrice A e poi
3 % risolvo il sistema che ne consegue.
4
5
6 A = diag(ones(1,5));
7 A = A + diag(ones(1,4),1)*3;
8 A = A + diag(ones(1,4),-1)*5;
9
10 % Fattorizzazione LU per matrici tridiagonali.
11 % conditio sine qua non A deve essere tridiagonale.
12 function [ L, U ] = tridiag_fatt(A)
13     a = diag(A);
14     c = diag(A,1);
15     b = diag(A,-1);
16
17 % Calcolo dei determinanti dei minori
18 % NB d(n) = det(A)
19
20 d = ones(1,size(a)+1);
21 d(1) = 1;
22 d(2) = a(1);
23 for i=2:size(a)
24     d(i+1) = (a(i)*d(i) - b(i-1)*c(i-1)*d(i-1));
25 endfor
26
27
28 % diagonale inferiore di L
29 l = ones(1,size(a)-1);
30 % diagonale principale di U
31 u = ones(1,size(a));
32
33 for i=1:size(a)-1
34     l(i) = l(i)*( b(i)*d(i)/d(i+1) );
35 endfor
36
37 for i=1:size(a)
38     u(i) = d(i+1)/d(i);
39 endfor
40
41 % Ora posso calcolare le due matrici L ed U
42
43 L = diag(ones(1,size(a))) + diag(l,-1);
44 U = diag(u) + diag(c,1);
45
46 endfunction
47
48
49 [ L,U ] = tridiag_fatt(A);
50 b = [ 1,2,3,4,5 ]';
51 %% Algoritmo di risoluzione
52 % Ax = b
53 % A = LU
54 % LUx = b
55 %
56 % Ly = b
57 % Ux = y
58 %
```

```

59 %
60 %   x \{'e} la mia soluzione finale
61
62
63 % Diagonale inferiore di L
64   bl = diag(L,-1);
65
66 % Diagonale principale di U
67   au = diag(U);
68
69 % Diagonale superiore di U
70   cu = diag(U,1);
71
72   n = size(L)(1);
73
74
75   x_sure = A\b
76
77   y = ones(1,n);
78   x = ones(1,n);
79
80
81 % Risoluzione di Ly = b
82   y(1) = b(1);
83   for i=2:n
84       y(i) = b(i) - bl(i-1)*y(i-1);
85   endfor
86
87
88 % Risoluzione di Ux = y
89   x(n) = y(n)/au(n);
90   for i=n-1:-1:1
91       x(i) = (y(i-1)-cu(i)*x(i+1))/cu(i);
92   endfor

```


5.5 Esercizio 5

```

1 % funzione per matrice bidiagonale del tipo
2 %   A = diag(ones(1,n)) + diag(2*ones(1,n-1),1);
3
4 function [ inv_A ] = inv_bidiag(A);
5
6 % Calcolo l'inversa con il metodo di Gauss Giordan
7
8
9 % n = numero di righe di A
10 n = size(A)(1);
11
12
13 AI = [ A , eye(size(A)(2)) ];
14
15
16 for r=n:-1:2
17     AI(r-1,:) = AI(r-1,:)-2*AI(r,:);
18 endfor
19
20 inv_A = AI(:,size(AI)(2)/2+1:size(AI)(2));
21
22
23 endfunction
24
25
26 a1=1:10;
27 c1=11:19;
28
29 A = diag(a1) + diag(c1,1) + diag(c1,-1);
30
31 % A deve essere tridiagonale
32 function [ B ] = tri_fact(A)
33
34     n = size(A)(2);
35     p = ones(1,n);
36     q = ones(1,n-1);
37     a = diag(A);
38     c = diag(A,-1);
39
40     p(1) = sqrt(a(1));
41
42     for i=2:n
43         q(i-1) = c(i-1)./p(i-1);
44         p(i) = sqrt(a(i)-q(i-1).^2);
45     endfor
46
47     B = diag(p) + diag(q,-1);
48
49 endfunction
50
51 B = tri_fact(A);
52
53 A1 = B*B';

```

5.6 Esercizio 6

```
1 t = 1:6;
2 y = [ 0.5,0.8,0.7,0.3,0.1,0.4 ];
3
4 f = inline(' 1 + sin(2*pi*x/6) + cos(2*pi*x/6) ');
5
6
7 A = [];
8 for i=1:6
9     A = [ A ; [ 1, sin(2*pi*i/6), cos(2*pi*i/6) ] ];
10 endfor
11
12 a = A\y';
13
14 a = fliplr(a);
15
16 p = polyval(a,t);
17
18 A
```

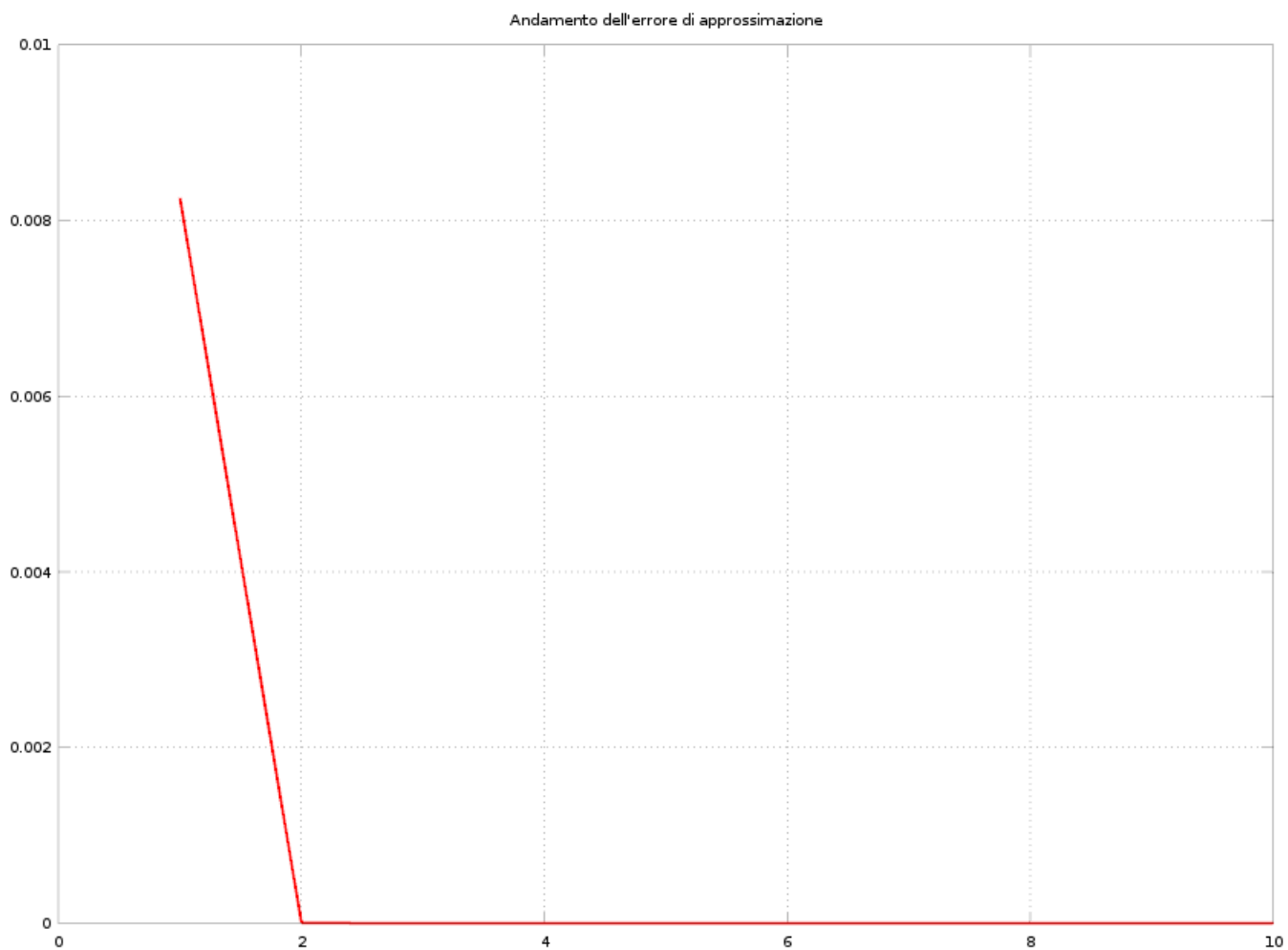
5.7 Esercizio 7

```
1  a1=1:10;
2  c1=11:19;
3
4  A = diag(a1) + diag(c1,1) + diag(c1,-1);
5
6  % A deve essere tridiagonale
7  function [ B ] = tri_fact(A)
8
9      n = size(A)(2);
10     p = ones(1,n);
11     q = ones(1,n-1);
12     a = diag(A);
13     c = diag(A,-1);
14
15     p(1) = sqrt(a(1));
16
17     for i=2:n
18         q(i-1) = c(i-1)./p(i-1);
19         p(i) = sqrt(a(i)-q(i-1).^2);
20     endfor
21
22     B = diag(p) + diag(q,-1);
23
24 endfunction
25
26 B = tri_fact(A);
27
28 A1 = B*B';
```

5.8 Esercizio 8

Approssimazione di π mediante troncata di una serie

```
1  tr = inline("16^(-x)*(4/(8*x+1) - 2/(8*x+4) - 1/(8*x+5) - 1/(8*x+6))");
2
3  p0 = (4-1/2-1/5-1/6);
4
5  p_i = p0*ones(1,100);
6
7
8  for a=2:100
9      for b=1:a
10         p_i(a) = p_i(a) + tr(b);
11     endfor
12 endfor
13
14 % converge velocemente a pi
15
16 real_pi = ones(1,100)*pi;
17 err = abs(real_pi - p_i);
18
19 % alla decima iterazione l'errore in format long risulta essere nullo
20
21 plot(1:10,err(1:1:10),'linewidth',2,'r');
22 grid on
23 title("Andamento dell'errore di approssimazione");
```

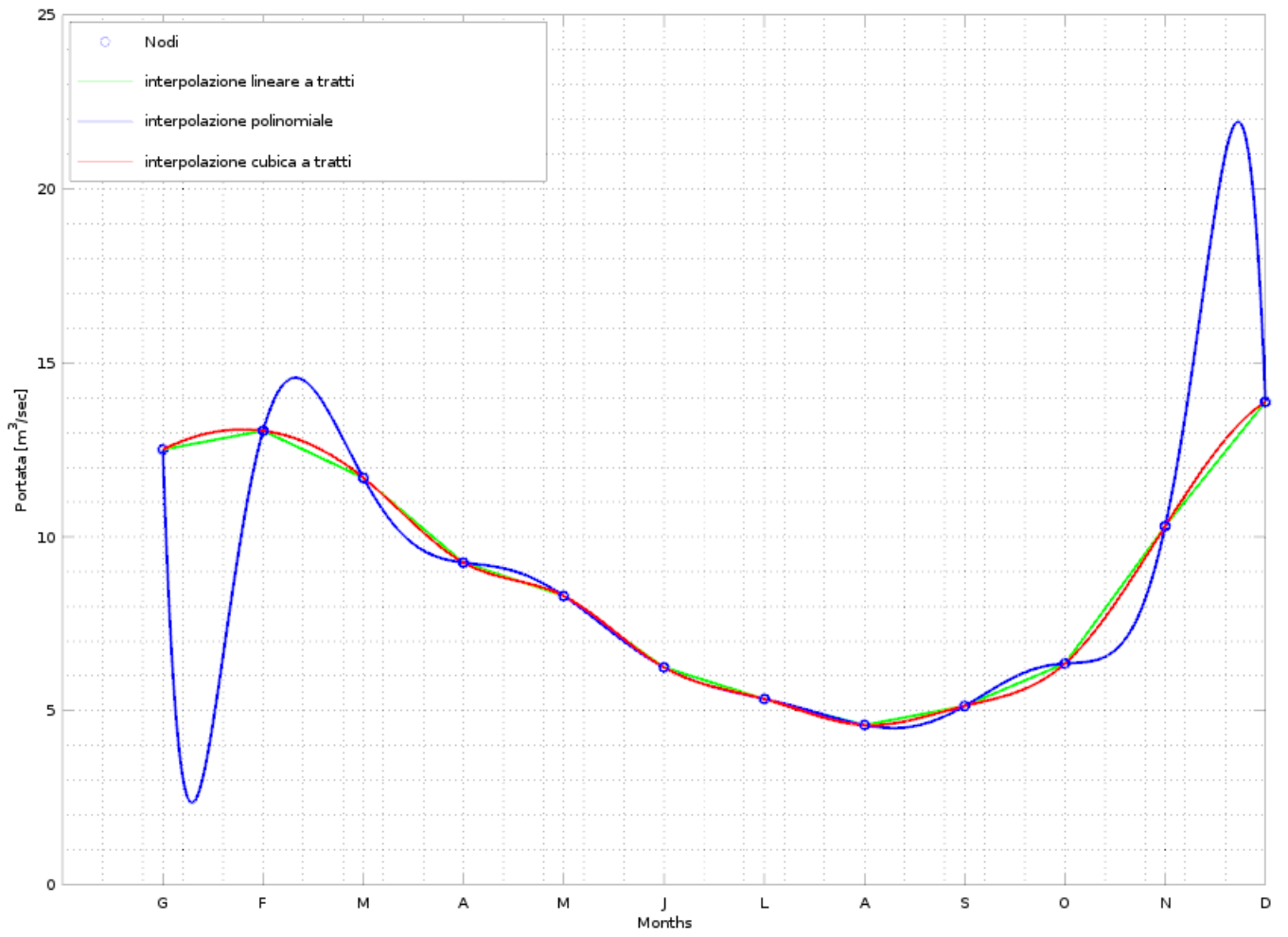


6 Esercitazione 6

6.1 Esercizio 1

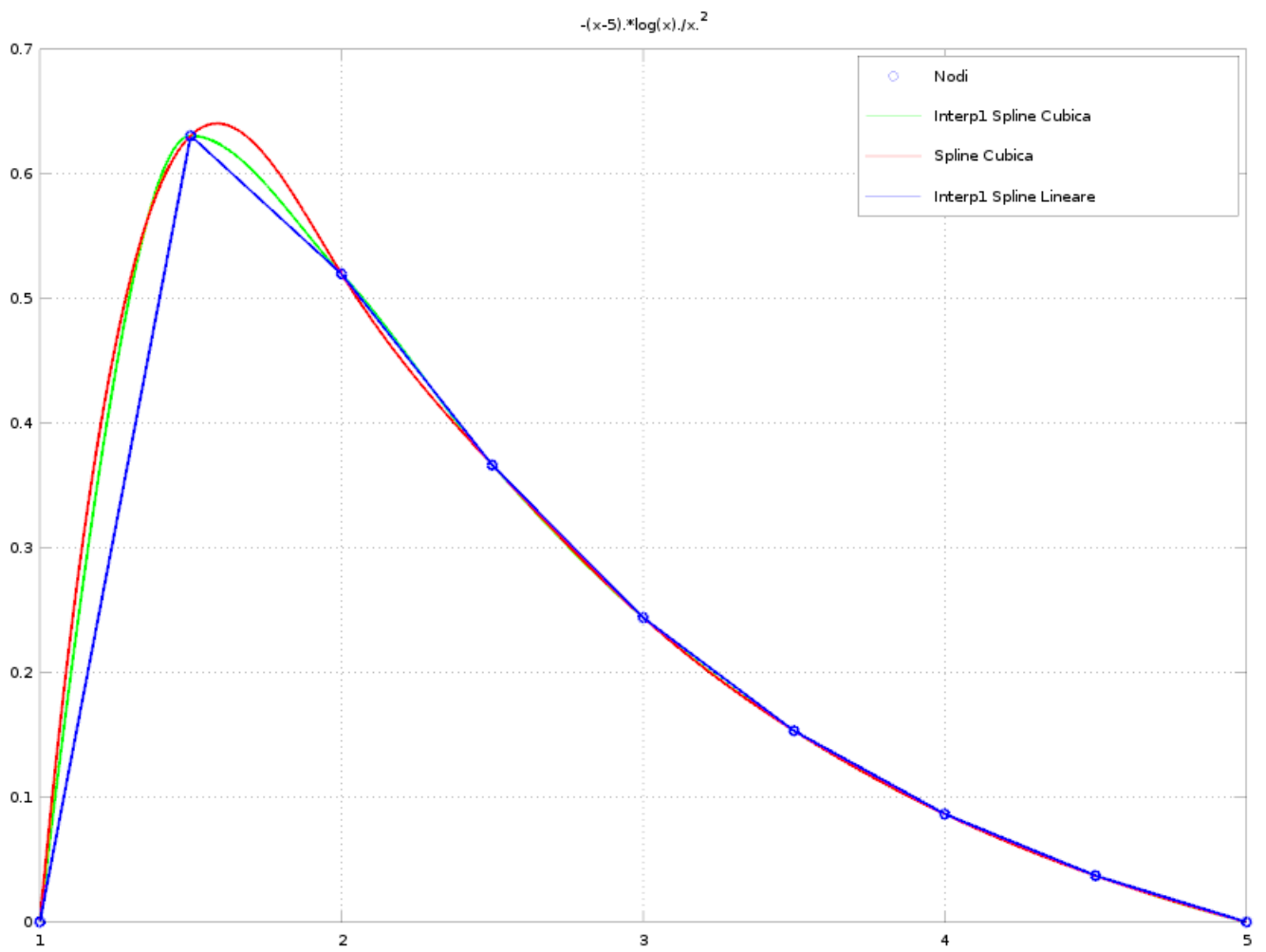
```
1  %months = [ 'G', 'F', 'M', 'A', 'M', 'J', 'L', 'A', 'S', 'O', 'N', 'D']
2  %set(gca,'XtickLabel',months)
3  x=1:12;
4  y=[12.51, 13.05, 11.7, 9.26, 8.3, 6.25, 5.34, 4.59, 5.14, 6.36, 10.31, 13.88]
5
6
7  xx=linspace(1,12,1201);
8
9
10 % spline lineare
11 % se il campo " metodo " viene omissso di default viene usata
12 % l'interpolazione lineare, ovvero le spline lineari
13
14 yyl = interp1(x,y,xx);
15
16
17 % interpolazione polinomiale
18 % comando polifit
19
20 coeff_yyp = polyfit(x,y,11);
21 yyp = polyval(coeff_yyp,xx);
22
23 % spline cubica
24
25 yyc = interp1(x,y,xx,'spline');
26
27
28 % plotting
29
30 plot(x,y,'o',xx,yyl,'g',xx,yyp,'b',xx,yyc,'r');
31 grid on
32 ylabel("Portata [m^3/sec]");
33 xlabel("Months");
34 title("Confronto calcolo della portata tramite tre metodi di interpolazione");
35
36 legend(" Nodi "," interpolazione lineare a tratti "," interpolazione
    polinomiale "," interpolazione cubica a tratti ");
```

Confronto calcolo della portata tramite tre metodi di interpolazione



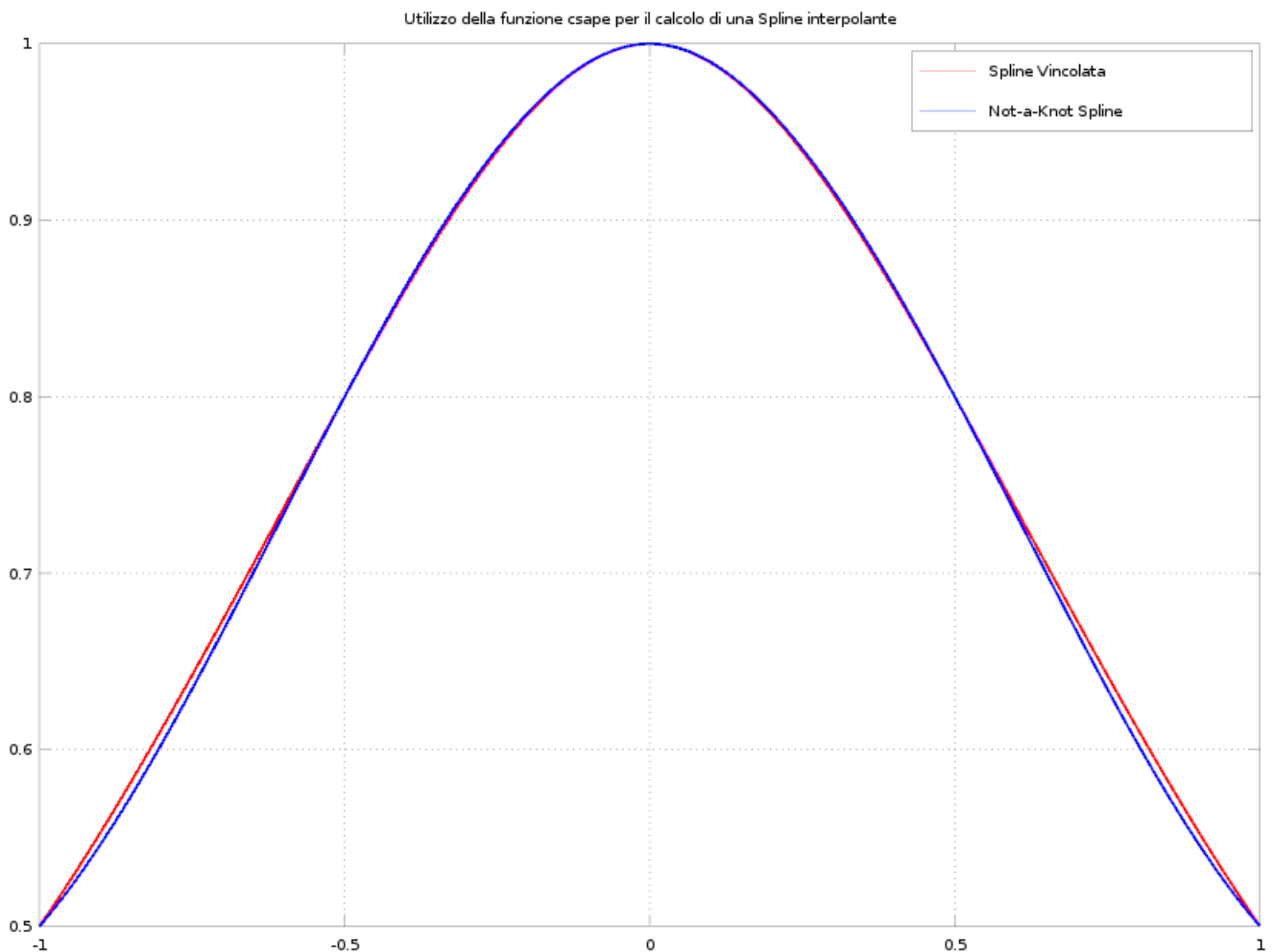
6.2 Esercizio 2

```
1 f = inline("-(x-5).*log(x)./x.^2");
2 x = 1:0.5:5;
3 y = f(x);
4
5
6 xx = linspace(1,5,500);
7
8 % comando interp1 per interpolazione a tratti lineare
9
10 yyl = interp1(x,y,xx,'linear');
11
12 % comando interp1 per interpolazione a tratti cubica
13
14
15 yyi = interp1(x,y,xx,'cubic');
16
17
18 % comando spline per interpolazione a tratti
19
20 pps = spline(x,y);
21 yys = ppval(pps,xx);
22
23
24 % plotting
25
26 plot(x,y,'linewidth',2,'o',xx,yyi,'linewidth',2,'g',xx,yys,'linewidth',2,'r',xx,
      yyn,'linewidth',2,'b');
27 legend(" Nodi ", " Interp1 Spline Cubica ", " Spline Cubica ", " Interp1 Spline
      Lineare");
28 grid on
29
30 % Errori
31 % Valori della funzione nei punti xx
32
33 yyreal = f(xx);
34
35 % confrontiamo i valori della funzione con quelli approssimati
36
37 err_interp_lineare = abs(yyreal-yyl);
38 err_interp_cubica = abs(yyreal-yys);
39 err_interp_spline = abs(yyreal-yys);
40
41
42 figure 2
43
44 plot(xx,err_interp_cubica,'linewidth',2,'g',xx,err_interp_spline,'linewidth',2,'
      o',xx,err_interp_lineare,'linewidth',2,'r');
45 legend(" Interp1 Spline Cubica ", " Spline Cubica ", " Interp1 Spline Lineare");
46 grid on
47
48
49 % La migliore funzione interpolante risulta essere, all'analisi degli errori, la
      spline cubica.
50 % Gli errori della spline cubica calcolati con la funzione spline e quelli
      calcolati con la funzione
51 % interp1(xx,yy,'spline') risultano essere praticamente sovrapposti nel grafico
      , possiamo concludere quindi che siano
52 % entrambi metodi appropriati per il calcolo della funzione interpolante
```

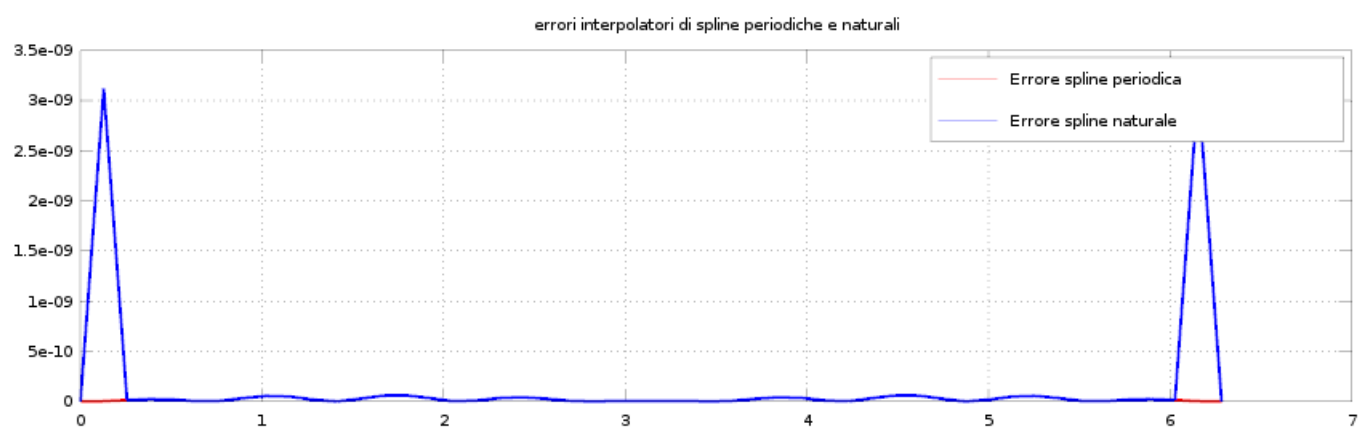
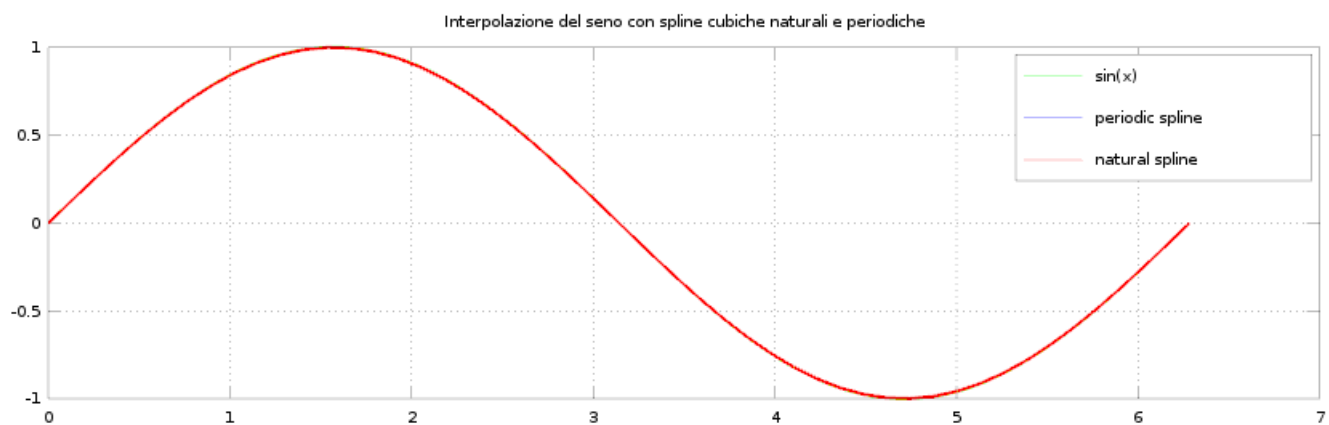
6.3 Esercizio 3

```
1 load csape.m
2
3 x = [-1, -0.5, 0, 0.5, 1];
4 y = [0.5, 0.8, 1, 0.8, 0.5];
5
6 xx = linspace(-1,1,1000);
7
8 % "Complete" condition imposes that the spline's values have to be clamped
9 % at the extreme knots with values specified in 2nd argument vector
10 pp_cl = csape(x,y,'complete',[1/2 -1/2]);
11
12 % Not-a-knot condition imposes the continuity of third derivative between first
13 % and second and
14 % next-to-last and last knot-
15 pp_nk = csape(x,y,'not-a-knot');
16
17 yy_cl = ppval(pp_cl,xx);
18 yy_nk = ppval(pp_nk,xx);
19
20 plot(xx,yy_cl,'linewidth',2,'r',xx,yy_nk,'linewidth',2,'b');
21 grid on
22 legend(" Spline Vincolata ", " Not-a-Knot Spline ");
```



6.4 Esercizio 4

```
1 load csape.m
2
3 x = linspace(0,2*pi,500);
4 y = sin(x);
5
6 xx = linspace(0,2*pi,50);
7 yy = sin(xx);
8
9 % Coefficients Periodic Spline
10 % S'(x0) = S'(xn)
11 % S''(x0) = S''(xn)
12
13 cp = csape(x,y, 'periodic');
14
15
16
17
18 % Coefficients Natural Spline
19 % S'''(x0)=S'''(xn) = 0
20
21 cn = csape(x,y, 'complete', [ 0 0 ]);
22
23
24
25 yp = ppval(cp,xx);
26 yn = ppval(cn,xx);
27
28 err_periodic = abs(yy-yp);
29 err_natural = abs(yy-yn);
30
31
32 subplot(2,1,1)
33 plot(x,y, 'linewidth',2, 'g', xx,yp, 'linewidth',0.5, 'b', xx,yn, 'linewidth',2, 'r'
34 )
35 title(" Interpolazione del seno con spline cubiche naturali e periodiche ");
36 grid on
37 legend(" sin(x) ", " periodic spline ", " natural spline ");
38 subplot(2,1,2)
39 plot(xx,err_periodic, 'linewidth',2, 'r', xx,err_natural, 'linewidth',2, 'b')
40 title(" errori interpolatori di spline periodiche e naturali ");
41 grid on
42 legend( " Errore spline periodica ", " Errore spline naturale ");
43
44
45 % Si noti che data una discretizzazione iniziale piccola ( 5 punti ) come questa
46 % gli errori risultano rilevanti per entrambe le tecniche interpolatorie , per
47 % quanto per la
48 % spline naturale risultino gi\{a} maggiori.
49 % Se aumentiamo notevolmente invece il numero di intervalli notiamo che entrambi
50 % gli errori
51 % tendono a diminuire drasticamente all'interno dell'intervallo , mentre agli
52 % estremi la
53 % spline naturale continua a mantenere un errore notevole.
```



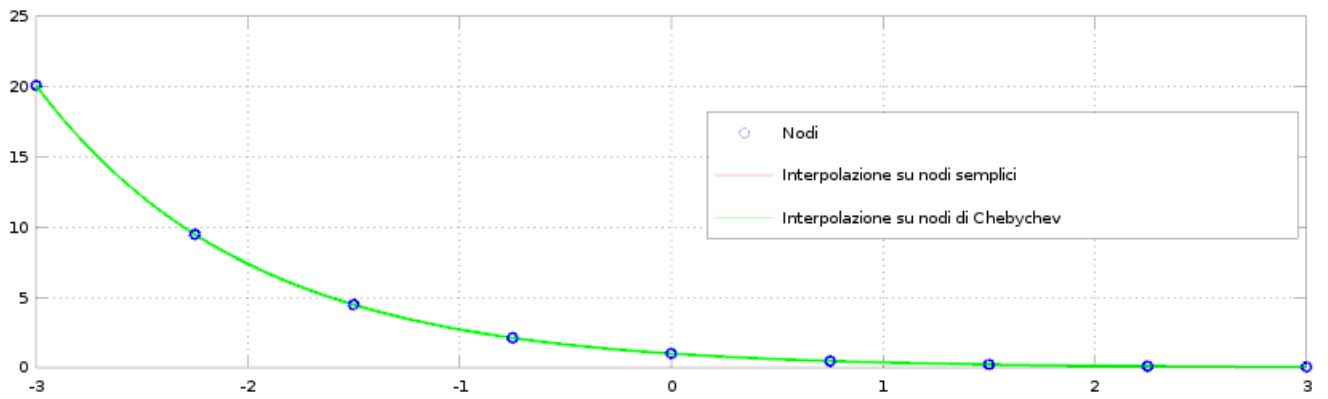
6.5 Esercizio 5

```
1 p1 = [ 2, -3, 4, -7 ];
2 p2 = [ 3, -2, -5 ];
3
4 % Somma di Polinomi
5
6 psum = polyadd(p1,p2);
7
8 % Sottrazione di Polinomi
9
10 psub = polyadd(p1,-p2);
11
12 % Moltiplicazione
13
14 pmul = conv(p1,p2);
15
16 % Divisione
17
18 pdiv = deconv(p1,p2);
```

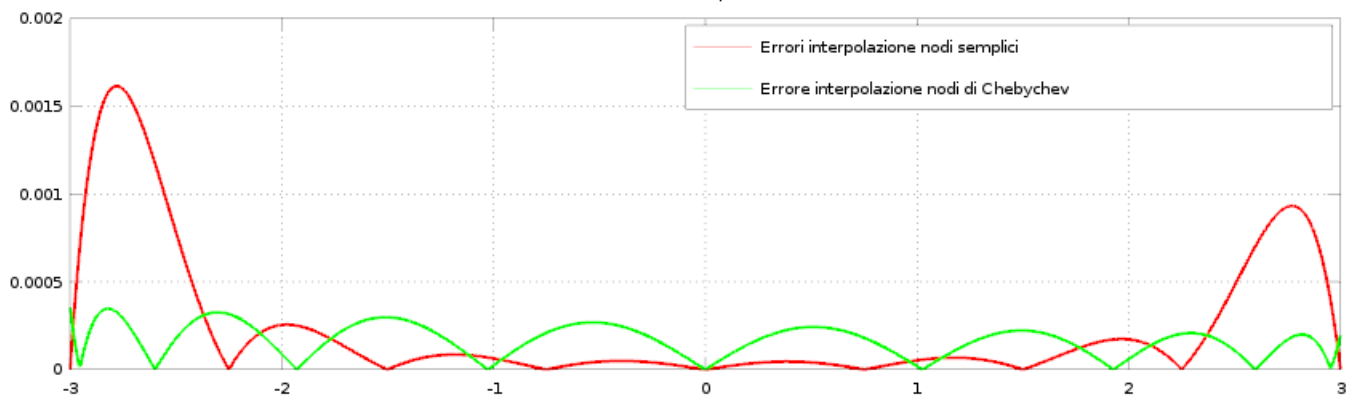
6.6 Esercizio 6

```
1  f = inline(" e.^-x ");
2
3  x = linspace(-3,3,9);
4
5  y = f(x);
6
7  % Polinomio di grado minimo
8
9  cp = polyfit(x,y,8);
10
11  xx = linspace(-3,3,1000);
12
13  % Valori del Polinomio
14
15  yp = polyval(cp,xx);
16
17
18  % Ricerca dei Nodi di Chebychev
19
20
21  cheb = inline("1/2(-3+3) + 1/2(3+3)*cos((2*x-1)/(2*9))");
22
23  xc = zeros(1,9);
24  for i=1:9
25      xc(i) = 1/2*(-3+3)+1/2*(3+3).*(cos(((2.*i-1)./18).*pi));
26  endfor
27
28
29  yc = f(xc);
30
31  % Polinomio di grado minimo con
32  Chebychev
33
34  cc = polyfit(xc,yc,8);
35
36  % Valori del Polinomio di grado minimo
37
38  ypc = polyval(cc,xx);
39
40
41  % Errori
42
43  y_ex = f(xx);
44  err_pol = abs(y_ex-yp);
45  err_cheb = abs(y_ex-ypc);
46
47
48  % Plotting
49
50  subplot(2,1,1)
51  plot(x,y,'linewidth',2,'o',xx,yp,'linewidth',2,'r',xx,ypc,'linewidth',2,'g');
52  legend("Nodi", "Interpolazione su nodi semplici", "Interpolazione su nodi di
    Chebychev");
53
54  subplot(2,1,2)
55  plot(xx,err_pol,'linewidth',2,'r',xx,err_cheb,'linewidth',2,'g');
56  legend("Errori interpolazione nodi semplici","Errore interpolazione nodi di
    Chebychev");
```

Confronto di polinomi interpolatori su nodi equispaziati e su nodi di Chebyshev

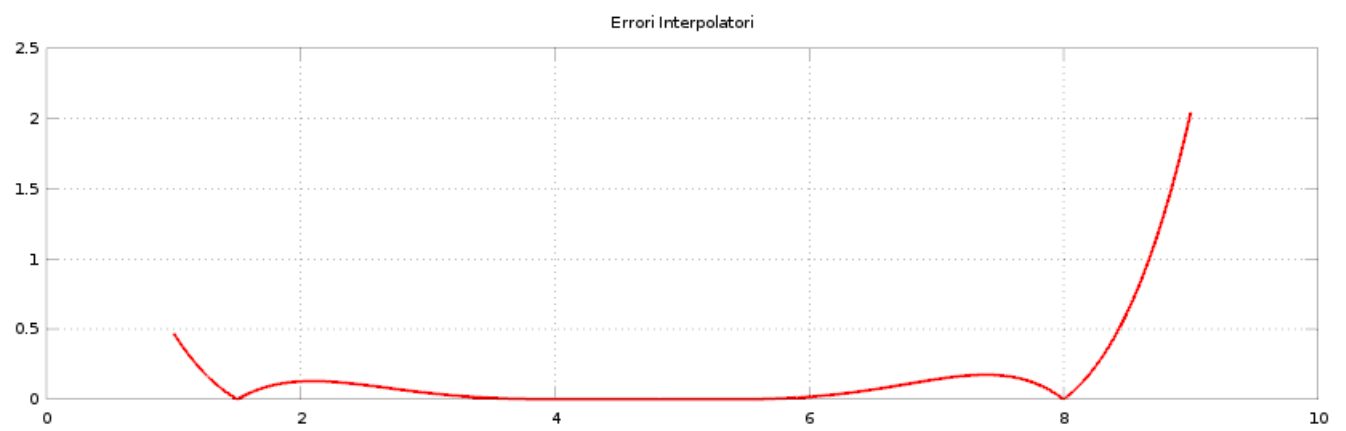
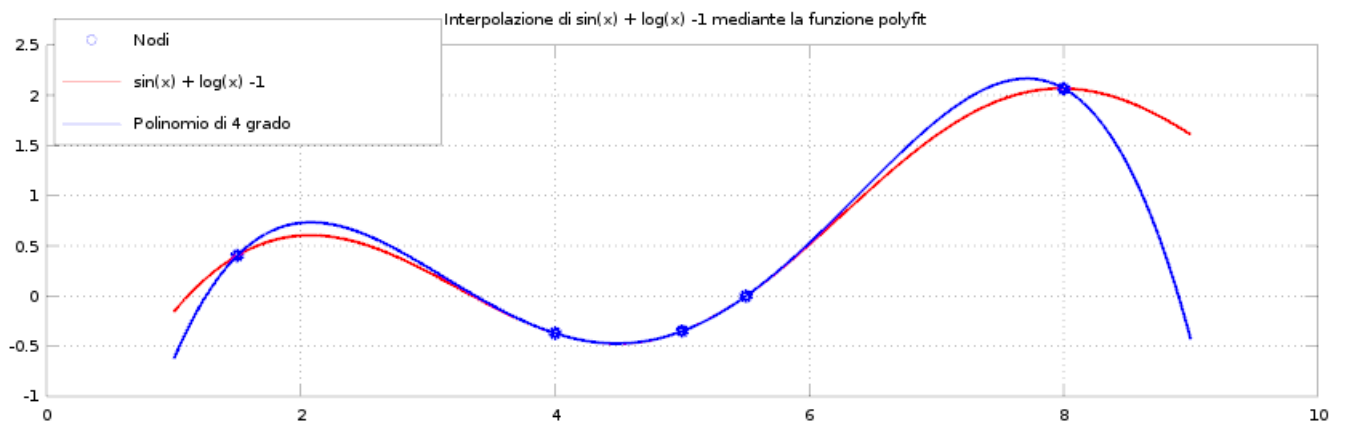


Errori interpolatori



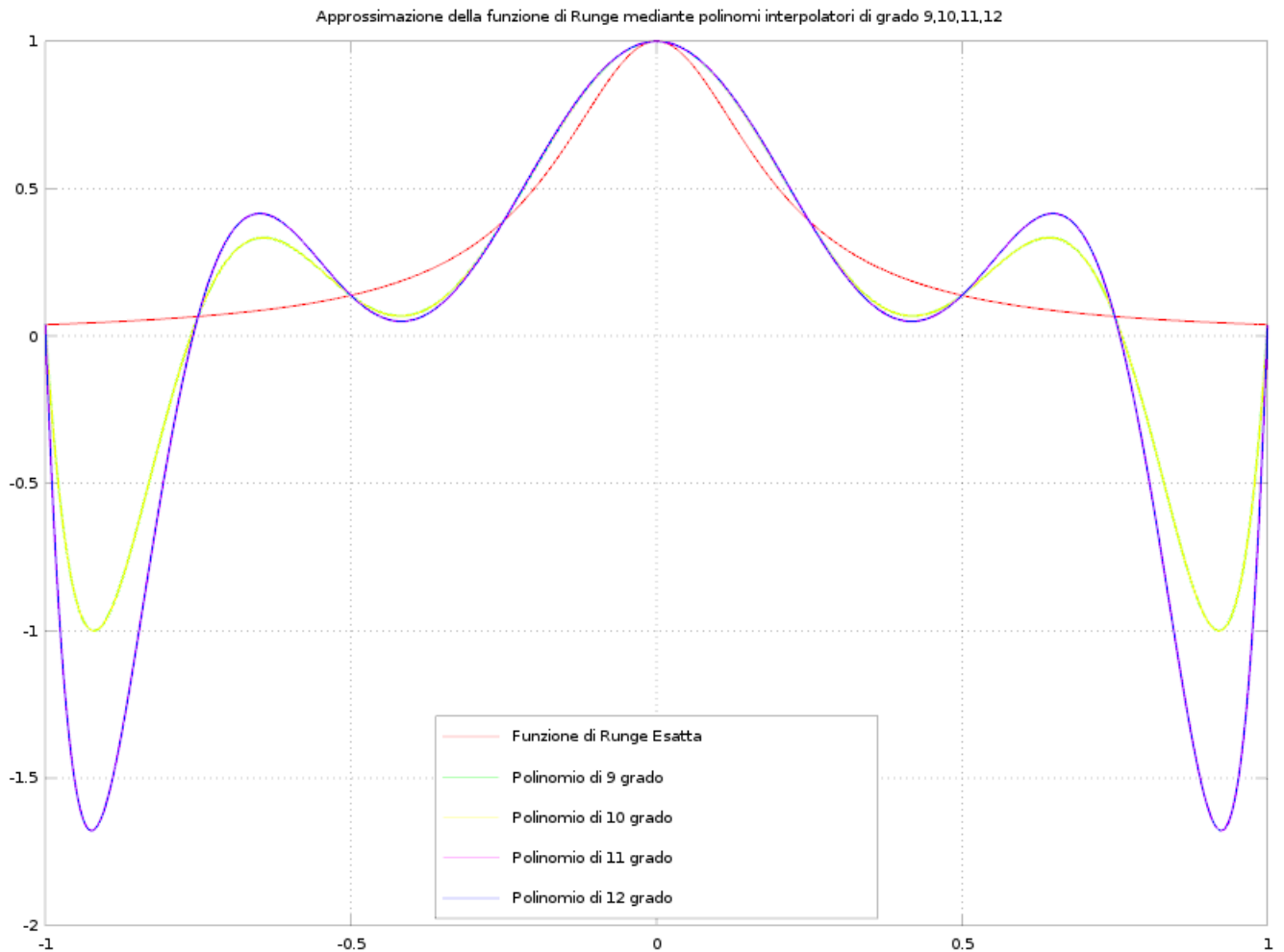
6.7 Esercizio 7

```
1 f = inline("sin(x) + log(x) -1 ");
2 x = [1.5, 4, 5, 5.5, 8];
3 y = f(x);
4
5 xx = linspace(1,9,1000);
6
7 c = polyfit(x,y,4);
8
9 yy = polyval(c,xx);
10
11 y_ex = f(xx);
12
13
14 err = abs(y_ex-yy);
15
16
17 subplot(2,1,1)
18 plot(x,y,'linewidth',3,'o',xx,y_ex,'linewidth',2,'r',xx,yy,'linewidth',2,'b');
19 legend(" Nodi ", " sin(x) + log(x) -1 ", " Polinomio di 4 grado ","location","
    northwest");
20 grid on
21 ylabel(" sin(x) + log(x) -1 ");
22
23 subplot(2,1,2)
24 plot(xx,err,'linewidth',2,'r');
25 title(" Errori Interpolatori ")
26 grid on
```

6.8 Esercizio 8

```
1  f = inline("1./(1+25*x.^2)");
2
3  x9 = linspace(-1,1,9);
4  x10 = linspace(-1,1,10);
5  x11 = linspace(-1,1,11);
6  x12 = linspace(-1,1,12);
7
8  y9 = f(x9);
9  y10 = f(x10);
10 y11 = f(x11);
11 y12 = f(x12);
12
13
14 c9 = polyfit(x9,y9,8);
15 c10 = polyfit(x9,y9,9);
16 c11 = polyfit(x9,y9,10);
17 c12 = polyfit(x9,y9,11);
18
19 xx = linspace(-1,1,1000);
20
21 y_ex = f(xx);
22 yp9 = polyval(c9,xx);
23 yp10 = polyval(c10,xx);
24 yp11 = polyval(c11,xx);
25 yp12 = polyval(c12,xx);
26
27
28 plot(xx,y_ex,'r',xx,yp9,xx,yp10,xx,yp11,xx,yp12);
29 legend(" Funzione di Runge Esatta ", " Polinomio di 9 grado ", " Polinomio di 10
    grado ", " Polinomio di 11 grado ", " Polinomio di 12 grado ","location","
    south");
30 grid on
31 title(" Approssimazione della funzione di Runge mediante polinomi interpolatori
    di grado 9,10,11,12");
```



7 Esercitazione 7

7.1 Esercizio 1

Costruzione di una formula di quadratura per il calcolo dell' integrale definito

$$I = \int_0^1 e^{-x^2} dx$$

```

1 %% Newton Cotes con n = 4 %%
2
3
4 % expr0= ((x-1)/(0-1))*((x-2)/(0-2))*((x-3)/(0-3))*((x-4)/(0-4));
5 % expr2= ((x-0)/(2-0))*((x-1)/(2-1))*((x-3)/(2-3))*((x-4)/(2-4));
6
7
8 % Codice per il Calcolo dei Coefficienti in Matlab
9 syms x;
10
11 alpha0 = int(((x-1)/(0-1))*((x-2)/(0-2))*((x-3)/(0-3))*((x-4)/(0-4)),x, 0,
12     4);
13 alpha2 = int(((x-0)/(2-0))*((x-1)/(2-1))*((x-3)/(2-3))*((x-4)/(2-4)),x, 0,
14     4);
15 alpha1 = (4-(2*alpha0)*-alpha2);
16 alpha1 = alpha1/2;
17 alpha3 = alpha1;

```

```

18  alpha4 = alpha0;
19
20  #{
21  % Coefficienti precalcolati
22  alpha0 = 14/15;
23  alpha1 = 64/15;
24  alpha2 = 24/25;
25  alpha3 = alpha1;
26  alpha4 = alpha0;
27  #}
28
29  % alpha0+alpha1+alpha2+alpha3+alpha4 = 4 ? Propriet\'{a} delle costanti di
    Newton
30  % se non vale i calcoli sono errati
31
32  alpha = [ alpha0, alpha1, alpha2, alpha3, alpha4 ]
33
34
35  % La formula di NewtonCotes chiusa vuole che I = h*[alpha0*f(x(0))+alpha1*f(x)
    ...]
36
37  a=0;
38  b=1;
39  h = (b-a)/4;
40
41  x = [];
42  x = [ x, 0 ];
43  for i=1:4
44      x = [ x, a+i*h ];
45  endfor
46
47
48  f = inline("e^(-x.^2)");
49  fun = @(x) e^(-x.^2);
50  y = f(x);
51
52  I = 0;
53  for i=1:5
54      I = I + y(i)*alpha(i);
55  endfor
56
57      I = h*I;
58
59  RealI = integrate(fun,0,1);
60
61  err = abs(RealI-I);

```

7.2 Esercizio 2

Costruzione di un grafico per la rappresentazione dell'errore relativo nel calcolo delle formule di Newton-Cotes approssimanti l'integrale definito $I = \int_0^1 e^{-x^2} dx$

```

1  % Andamento dell'errore relativo nel calcolo dell'integrale definito della
    funzione e^(-x.^2)
2  % mediante l'uso delle formule di Newton-Cotes
3
4  % Inizio calcolando il valore "esatto" dell'integrale
5
6
7  f = @(x) e.^(-x.^2);
8
9
10 %RealI = integrate(fun,0,1);
11 %err = abs(RealI-I);
12
13 RealI = 0.746824;
14 RealI_v = RealI*ones(1,7);
15
16 % Matrice dei Coefficienti delle formule di Newton-Cotes
17 NC\_coeff =
    (
        1/2      1/2      0      0      0      0      0      0
        1/3      4/3      1/3      0      0      0      0      0
        3/8      9/8      9/8      3/8      0      0      0      0
        14/45     64/45     24/45     64/45     14/45     0      0      0
        95/288     375/288    250/288    250/288    375/288    95/288     0      0
        41/140     216/140     27/140     272/140     27/140     216/140    41/140     0
        5257/17208 25039/17208 9261/17208 20923/17208 20923/17208 9261/17208 25039/17208 5257/17208
    )

18 % Ora cerco i 7 valori dell'integrale approssimato
19
20 a=0;
21 b=1;
22 n=7;
23
24 I = zeros(1,n);
25
26 for i=1:n
27     h = (b-a)/i;
28     for j=1:i+1
29         I(i) = I(i) + NC\_coeff(i,j)*f(a+h*(j-1));
30     endfor
31     I(i) = h*I(i);
32 endfor
33
34
35 err = abs(RealI_v - I);
36
37 plot([1:7], err, 'linewidth',2,'r')
38 grid minor
39 title("Andamento dell'errore di integrazione al crescere della decomposizione \n
    calcolato come differenza tra RealI = 0.746824 e il valore dato dalla
    formula per un n dato ");
40 xlabel("Indice della decomposizione")
41
42

```

```

43
44 % Calcolo ora una stima dell'errore per eccesso mediante le formule alle
    derivate
45
46 f2 = @(x) e.^(-x.^2).*(4*x-2);
47 f4 = @(x) 4*e.^(-x.^2).*(4*x.^3-2*x.^2-6*x+1);
48 f6 = @(x) 8*e.^(-x.^2).*(8*x.^5-4*x.^4-40*x.^3+12*x.^2+30*x-3);
49 f8 = @(x) 16*e.^(-x.^2).*(16*x.^7-8*x.^6-168*x.^5+60*x.^4+420*x.^3-90*x.^2-210*x
    +15);
50
51
52 a = 0;
53 b = 1;
54 h1 = (b-a);
55 h2 = (b-a)/2;
56 h3 = (b-a)/3;
57 h4 = (b-a)/4;
58 h5 = (b-a)/5;
59 h6 = (b-a)/6;
60 h7 = (b-a)/7;
61
62
63 % coefficienti dell'errore
64 coeff_err = [ (-1/12)*h1^3, (-1/90)*h2^5, (-3/80)*h3^5, (-8/945)*h4^7,
    (-275/12096)*h5^7, (-9/1400)*h6^9, (-8183/518400)*h7^9 ];
65
66
67 x = 0:0.01:1;
68
69
70 ve1 = f2(x).*coeff_err(1);
71 ve2 = f4(x).*coeff_err(2);
72 ve3 = f4(x).*coeff_err(3);
73 ve4 = f6(x).*coeff_err(4);
74 ve5 = f6(x).*coeff_err(5);
75 ve6 = f8(x).*coeff_err(6);
76 ve7 = f8(x).*coeff_err(7);
77
78 % Cerco il massimo di ogni vettore per poter fare una stima per eccesso
79
80 Me1 = max(abs(ve1));
81 Me2 = max(abs(ve2));
82 Me3 = max(abs(ve3));
83 Me4 = max(abs(ve4));
84 Me5 = max(abs(ve5));
85 Me6 = max(abs(ve6));
86 Me7 = max(abs(ve7));
87
88
89 Me=[Me1, Me2, Me3, Me4, Me5, Me6, Me7];
90
91
92
93 figure 2
94 plot(1:7, Me, 'linewidth', 2, 'g');
95 grid minor
96 title("Andamento dell'errore interpolatorio calcolato al crescere della
    decomposizione \n calcolato mediante le tabelle degli errori ");

```

Andamento dell'errore di integrazione al crescere della decomposizione
calcolato come differenza tra Reall = 0.746824 e il valore dato dalla formula per un n dato



Andamento dell'errore interpolatorio calcolato al crescere della decomposizione
calcolato mediante le tabelle degli errori



7.3 Esercizio 3

Costruzione di un grafico per la rappresentazione degli errori relativi nel calcolo dell'integrale definito $\int_0^1 \sin(x) dx$ mediante le formule dei trapezi e di Cavalieri-Simpson iterate

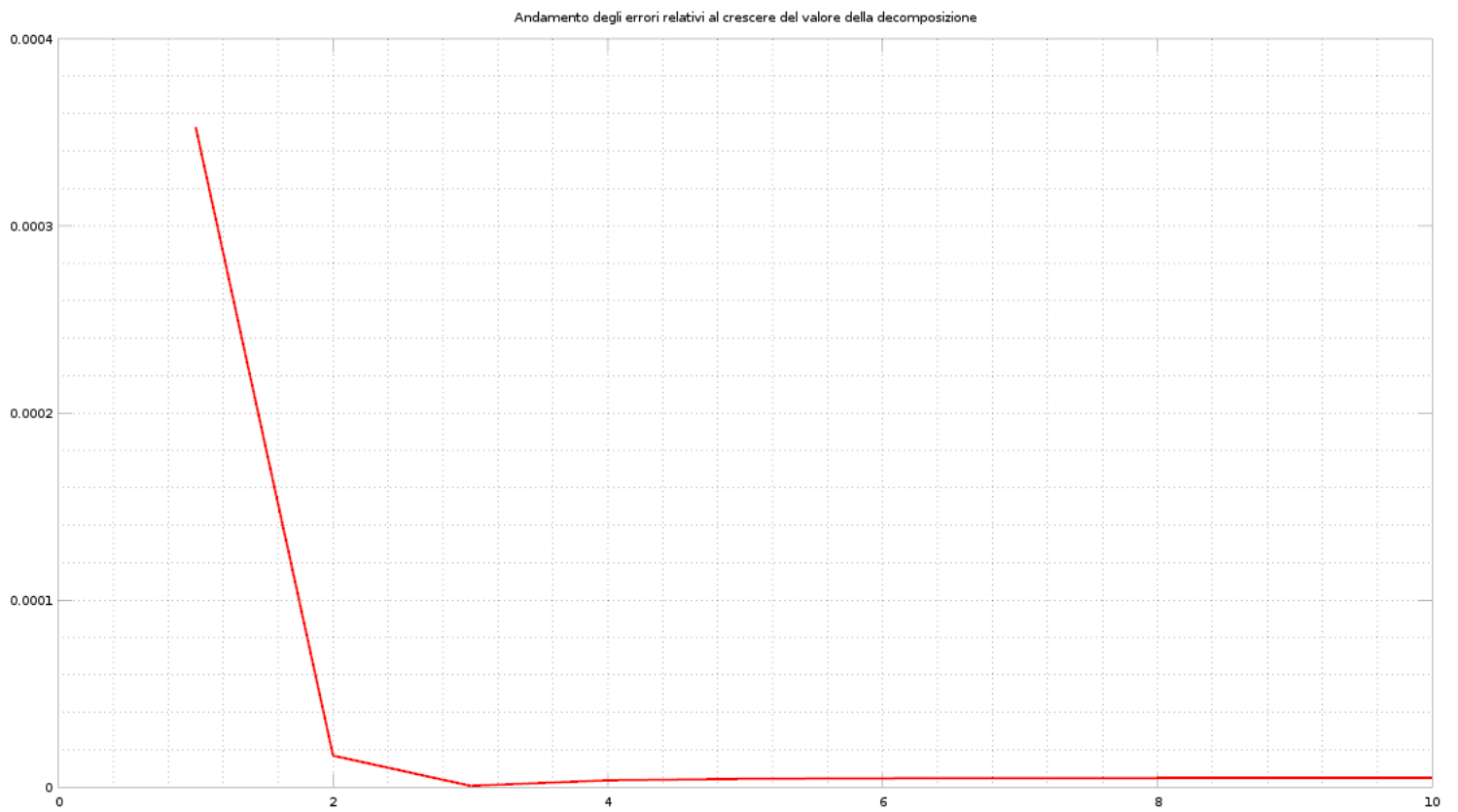
```
1 % Errori relativi al calcolo dell'integrale definito tra 0 e 1
2 % della funzione sin(x) con le formule di Cavalieri Simpson
3 % al crescere dei sottointervalli della decomposizione
4
5
6 % RealI = integrate(0,1,@(x) sin(x));
7 % Calcolato Online risulta
8 RealI = 0.45970;
9
10 CS_coeff = [ 1/3, 4/3, 1/3 ];
11
12 b0 = 1;
13 a0 = 0;
14
15 b = @(i,j) a0 + j*((b0-a0)/i),
16 a = @(i,j) (j-1)/i;
17
18 % Voglio dividere l'intervallo in 2m sottointervalli di modo da poter
19 % applicare la formula di CavalieriSimpson sottointervalli sulle coppie di
20 % sottointervalli
21 % Lo far\ '{o} per n=1:10
22
23 function I = CavalieriSimpson(x,y,f)
24     I = ((y-x)/2)*(1/3*f(x) + 4/3*f((x+y)/2) + 1/3*f(y));
25 endfunction
26
27 I = zeros(10);
28
29 for i=1:10
30     for j=1:i
31         I(i,j) = CavalieriSimpson(a(i,j),b(i,j),@(x) sin(x));
32     endfor
33 endfor
34
35
36
37 % sommando i termini della stessa riga della matrice I ottengo un'
38 % approssimazione ( in teoria sempre migliore )
39 % del valore dell'integrale del seno di x tra 0 e 1
40
41 Integralen = zeros(1,10);
42
43 for i=1:10
44     for j=1:i
45         Integralen(i) = Integralen(i) + I(i,j);
46     endfor
47 endfor
48
49
50 RealIv = ones(1,10)*RealI;
51
52 err_rel = abs(RealIv-Integralen)./RealIv;
53
```



```

54 plot([1:10],err_rel,'linewidth',2,'r');
55 title(" Andamento degli errori relativi al crescere del valore della
    decomposizione ");
56 grid minor

```



8 Esercitazione 8

8.1 Esercizio 1

Utilizzo della tecnica del pivoting per la fattorizzazione LU

```
1 function [A, P] = pivoting(A)
2     P = eye(length(A));
3     for i = 1: length(A)
4         if(A(i, i) == 0)
5             [m, j] = max(abs(A(i:length(A), i)));
6             j = i + j - 1;
7             A([i, j], :) = A([j, i], :);
8             P([i, j], :) = P([j, i], :);
9         end
10    end
11 end
12
13
14 % fattorizzazione LU
15
16 function [ L, U, P ] = lu_fact(A)
17     [ A, P ] = pivoting(A);
18     n = size(A);
19     L = eye(n);
20     U = zeros(n);
21
22     for k=1:n-1
23         for i=k+1:n
24             L(i, k) = A(i, k)/A(k, k);
25             A(i, :) = A(i, :) - L(i, k)*A(k, :);
26         endfor
27     endfor
28
29     U = A;
30 endfunction
31
32
33
34 A1 =
35     1, 0, 0, 1;
36     1, 0, 0, 1;
37     0, 0, 1, 1;
38     2, 3, 4, 5;
39
40 A2 =
41     4, 0, 2, 4;
42     0, 1, 1, 1;
43     2, 2, 11, 9;
44     4, 1, 9, 25;
```

```

45 [ l1 , u1 , p1 ] = lu_fact(A1);
46
47 [ l2 , u2 , p2 ] = lu_fact(A2);
48
49
50 [ ll1 , uu1 , pp1 ] = lu(A1);
51 [ ll2 , uu2 , pp2 ] = lu(A2);
52
53 % Fattorizzazione con lu_fact di A1
54 l1 =
55
56     1     0     0     0
57     2     1     0     0
58     0     0     1     0
59     1     0     0     1
60
61 u1 =
62
63     1     0     0     1
64     0     3     4     3
65     0     0     1     1
66     0     0     0     0
67
68 p1 =
69
70     1     0     0     0
71     0     0     0     1
72     0     0     1     0
73     0     1     0     0
74
75 p1*(l1*u1) =
76
77     1     0     0     1
78     1     0     0     1
79     0     0     1     1
80     2     3     4     5
81
82 A1 =
83
84     1     0     0     1
85     1     0     0     1
86     0     0     1     1
87     2     3     4     5
88
89
90 % Fattorizzazione con lu_fact di A2
91
92 l2 =
93
94     1.00000    0.00000    0.00000    0.00000
95     0.00000    1.00000    0.00000    0.00000
96     0.50000    2.00000    1.00000    0.00000
97     1.00000    1.00000    0.75000    1.00000
98
99 u2 =
100
101     4.00000    0.00000    2.00000    4.00000
102     0.00000    1.00000    1.00000    1.00000
103     0.00000    0.00000    8.00000    5.00000
104     0.00000    0.00000    0.00000    16.25000
105

```

```

106 p2 =
107
108 % Essendo la matrice di permutazione diagonale si pu' dedurre che nessuno
    dei minori
109 % di A2 abbia determinante nullo, quindi non necessita di una
    premoltiplicazione per una
110 % matrice di permutazione.
111
112 Diagonal Matrix
113
114     1     0     0     0
115     0     1     0     0
116     0     0     1     0
117     0     0     0     1
118
119 p2*(l2*u2) =
120
121     4     0     2     4
122     0     1     1     1
123     2     2    11     9
124     4     1     9    25
125
126
127
128 A2 =
129
130     4     0     2     4
131     0     1     1     1
132     2     2    11     9
133     4     1     9    25
134
135
136
137 % Fattorizzazione con lu di matlab di A1
138 l11 =
139
140     1.00000     0.00000     0.00000     0.00000
141     0.50000     1.00000     0.00000     0.00000
142     0.00000    -0.00000     1.00000     0.00000
143     0.50000     1.00000     0.00000     1.00000
144
145 uul =
146
147     2.00000     3.00000     4.00000     5.00000
148     0.00000    -1.50000    -2.00000    -1.50000
149     0.00000     0.00000     1.00000     1.00000
150     0.00000     0.00000     0.00000     0.00000
151
152 ppl =
153
154 Permutation Matrix
155
156     0     0     0     1
157     0     1     0     0
158     0     0     1     0
159     1     0     0     0

```

```

160 pp1*(ll1*uu1) =
161
162     1     0     0     1
163     1     0     0     1
164     0     0     1     1
165     2     3     4     5
166
167 A1 =
168
169     1     0     0     1
170     1     0     0     1
171     0     0     1     1
172     2     3     4     5
173
174
175 % Fattorizzazione con lu di matlab di A2
176
177 ll2 =
178
179     1.00000    0.00000    0.00000    0.00000
180     0.50000    1.00000    0.00000    0.00000
181     0.00000    0.50000    1.00000    0.00000
182     1.00000    0.50000   -0.50000    1.00000
183
184 uu2 =
185
186     4.00000    0.00000     2.00000     4.00000
187     0.00000    2.00000    10.00000     7.00000
188     0.00000    0.00000   -4.00000   -2.50000
189     0.00000    0.00000     0.00000    16.25000

```

```

190 pp2 =
191 Permutation Matrix
192
193     1     0     0     0
194     0     0     1     0
195     0     1     0     0
196     0     0     0     1
197
198 octave:172> pp2*(ll2*uu2)
199 ans =
200
201     4     0     2     4
202     0     1     1     1
203     2     2    11     9
204     4     1     9    25
205
206 A2 =
207
208     4     0     2     4
209     0     1     1     1
210     2     2    11     9
211     4     1     9    25
212
213
214 %Le matrici di permutazione restituite dall'algoritmo implementato nel
215 % linguaggio non sono le stesse che ottengo dal mio algoritmo.
216 % Evidentemente l'implementazione interna utilizza un algoritmo differente
217 % da quello di eliminazione gaussiana.

```

8.2 Esercizio 2

Implementazione dei metodi di Jacobi e di Gauss-Seidel per la soluzione approssimata di sistemi lineari1

```
1 A1
2     -3, 3, -6
3     -4, 7, -8
4     5, 7, -9
5
6 b1 = -6; -5; 3
7
8
9 A2 =
10     4, 1, 1
11     2, -9, 0
12     0, -8, -6
13
14 b2 = 6; -7; -14
15
16
17 function [ xnew, err, ro ] = Jacobi(A,b,x0,Nmax,toll)
18
19 D = diag(A);
20 E = D-tril(A);
21 F = D-triu(A);
22
23 invD = diag(1./diag(A));
24 Pj = invD*(E+F); % Matrice di iterazione
25 ro = max(abs(eig(Pj)));
26
27 xold = x0;
28 i=1;
29 test = 1;
30 err = zeros(1,Nmax);
31
32 while i<Nmax && test>toll
33     xnew = Pj*xold + invD*b; % Da definizione
34     err(i) = norm(xnew-xold);
35     test = err(i);
36     xold = xnew;
37     i = i+1;
38 endwhile
39
40 err = err(1:i-1);
41
42 endfunction
43
44
45
46 function [ xnew, err, ro ] = GaussSeidel(A,b,x0,Nmax,toll)
47
48 D = diag(A);
49 E = D-tril(A);
50 F = D-triu(A);
51
52 invDE = inv(D+E);
53 Pgs = invDE*F; % Matrice di iterazione
54 ro = max(abs(eig(Pgs)));
55
56 xold = x0;
```

```

57     i=1;
58     test = 1;
59     err = zeros(1,Nmax);
60
61     while i<Nmax && test>toll
62         xnew = Pgs*xold + invDE*b;    % Da definizione
63         err(i) = norm(xnew-xold);
64         test = err(i);
65         xold = xnew;
66         i = i+1;
67     endwhile
68
69     err = err(1:i-1);
70
71 endfunction
72
73
74
75
76 [x1j, err1j, ro1j]= Jacobi(A1, b1, zeros(3, 1), 100, 10^-3);
77 [x2j, err2j, ro2j]= Jacobi(A2, b2, zeros(3, 1), 100, 10^-3);
78
79
80 [x1g, err1g, ro1g]= GaussSeidel(A1, b1, zeros(3, 1), 100, 10^-3);
81 [x2g, err2g, ro2g]= GaussSeidel(A2, b2, zeros(3, 1), 100, 10^-3);
82
83
84
85 hold on
86 subplot(2,2,1)
87 semilogy(1:length(err1j), err1j)
88 title([ "Jacobi(A1x1=b1), ro(P1j) = " num2str(ro1j) ] )
89 grid on
90 subplot(2,2,2)
91 semilogy(1:length(err2j), err2j)
92 title([ "Jacobi(A2x2=b2), ro(P2j) = " num2str(ro2j) ] )
93 grid on
94 subplot(2,2,3)
95 semilogy(1:length(err1g), err1g)
96 title([ "GaussSidel(A1x1=b1), ro(P1g) = " num2str(ro1g) ] )
97 grid on
98 subplot(2,2,4)
99 semilogy(1:length(err2g), err2g)
100 title([ "GaussSidel(A2x2=b2), ro(P2g) = " num2str(ro2g) ])
101 grid on
102
103 [firstnumber=last]
104 %%% fprintf degli errori %%%
105
106 e1j = [ 1:length(err1j); err1j ];
107 e2j = [ 1:length(err2j); err2j ];
108
109 e1g = [ 1:length(err1g); err1g ];
110 e2g = [ 1:length(err2g); err2g ];
111
112
113 file_errori = fopen('errori1j.txt','w');
114 fprintf(file_errori, '%6s %56s\n', 'x', 'errore sistema 1 con Jacobi')
115 fprintf(file_errori, '%6.2f %56.8f\n', e1j);
116
117 file_errori = fopen('errori2j.txt','w');

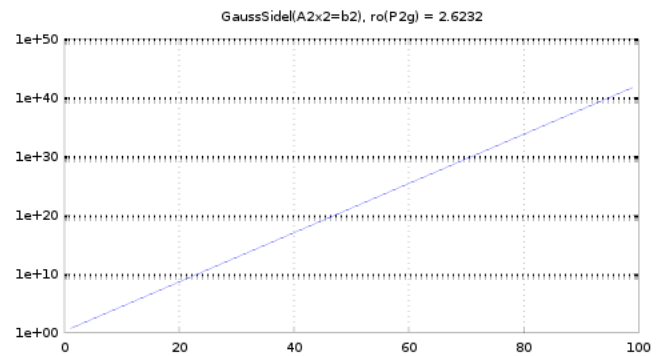
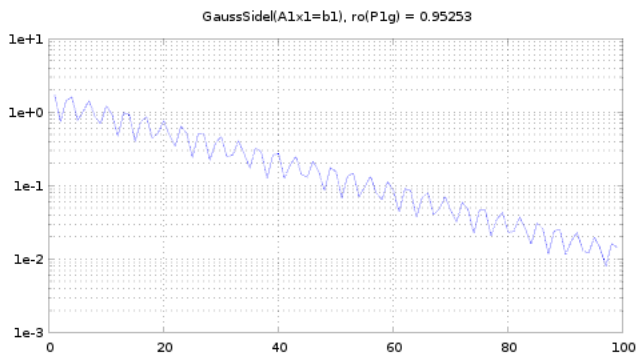
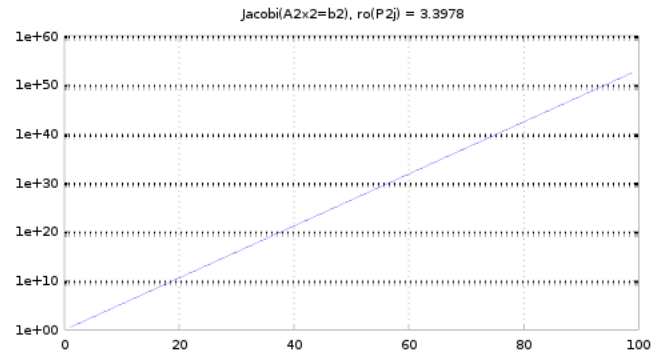
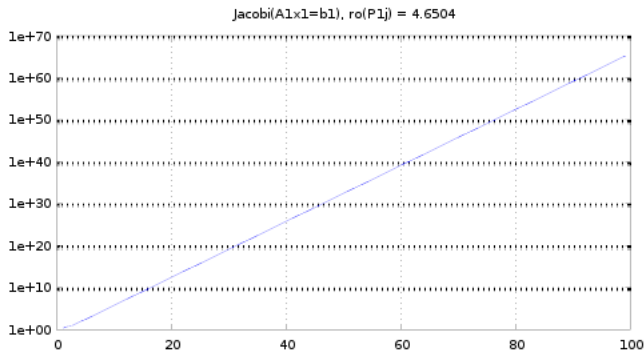
```



```

118 fprintf(file_errori, '%6s %56s\n', 'x', 'errore sistema 2 con Jacobi')
119 fprintf(file_errori, '%6.2f %56.8f\n', e2j);
120
121 file_errori = fopen('errori1g.txt','w');
122 fprintf(file_errori, '%6s %56s\n', 'x', 'errore sistema 1 con Gauss Sidel')
123 fprintf(file_errori, '%6.2f %56.8f\n', e1g);
124
125 file_errori = fopen('errori2g.txt','w');
126 fprintf(file_errori, '%6s %56s\n', 'x', 'errore sistema 2 con Gauss Sidel')
127 fprintf(file_errori, '%6.2f %56.8f\n', e2g);

```



Si noti che l'unica successione convergente è quella che usa il metodo di Gauss Sidel sul primo dei sistemi, la matrice d'iterazione associata infatti ha raggio spettrale minore di 1. Abbiamo un teorema che ci dice che questa è una condizione necessaria e sufficiente, quindi è coerente con la teoria avere come unica successione convergente quella che ha matrice associata d'iterazione con raggio spettrale minore di 1