III° Esercitazione – Secondo semestre
Analisi Numerica – Calcolo Numerico
A.A. 2014– 2015

Esercizio 1. I polinomi di Bernstein definiti da

$$b_{n,k}(t) = \frac{n!}{k!(n-k)!} t^k (1-t)^{n-k}$$

possono essere calcolati ricorsivamente utilizzando la seguente formula

$$\begin{cases} b_{0,0}(t) = 1 \\ b_{n,k}(t) = (1-t)b_{n-1,k}(t) + tb_{n-1,k-1}(t) \qquad k = 0, \cdots, n \;\; t \in [0,1] \end{cases}$$

si può dimostrare che il polinomio interpolatore di Bernstein definito come:

$$p_n(t) = \sum_{i=0}^{n} f(t_i) b_{n,i}(t) \qquad t \in [0,1], \quad t_i = i/n$$

converge uniformemente ad f per $n \longrightarrow \infty$.
Il seguente programma valuta i polinomi di Bernstein per $t \in [0,1]$ noti k ed n.

```
%-----------------------------------------------------------------
 function [ber]=bernstein(k,n)
  i=0;
if k == 0
    coef=1;
else
    coef=prod([1:n])/(prod([1:k])*prod([1:n-k]));
end for t=0:0.01:1
    i=i+1;
    ber(i)=coef*t^k*(1-t)^(n-k);
end
%-----------------------------------------------------------------
```

• Fissato un valore di n rappresentare in un grafico i polinomi $b_{n,k}$ per $k = 0, \cdots, n$.
• Data la funzione $\sqrt{x+1}$ con $x \in [0,1]$ e scelti 4 punti dell'intervallo costruire il polinomio interpolatore e il polinomio approssimante mediante i polinomi di Bernstein. Rappresentare in un grafico la funzione, i nodi scelti e i due polinomi ottenuti.

Esercizio 2. Data una curva in forma parametrica, ogni sua componente viene da una funzione spline nel modo seguente. Prendiamo una curva nel piano in forma parametrica:

$$p(t) = (x(t), y(t)) \qquad t \in [0, T],$$

e l'insieme dei punti del piano di coordinate $P_i = (x_i, y_i)$, $i = 0, \cdots, n$ ed introduciamo una decomposizione dell'intervallo $[0, T]$:

$$\Delta = \{0 = t_0 < t_1 < \cdots < t_n = T\}.$$

Utilizzando i due insiemi di valori

$$\{t_i,\, x_i\} \qquad \{t_i, y_i\}$$

come dati di interpolazione si ottengono due spline: $S_{k,\Delta,x}(t)$ e $S_{k,\Delta,y}(t)$ rispetto alla variabile t, interpolanti rispettivamente $x(t)$ e $y(t)$.
La curva parametrica:

$$S_{k,\Delta}(t) = (S_{k,\Delta,x}(t), S_{k,\Delta,y}(t))$$

è chiamata curva spline parametrica. Ovviamente diverse parametrizzazioni dell'intervallo $[0, T]$ forniscono spline diverse.
Supponiamo ora di avere $n+1$ punti ordinati nel piano: $P_0, P_1, \cdots, P_n$ una scelta ragionevole del parametro t fa uso della lunghezza dei singoli segmenti $P_{i-1}P_i$.
Indicata con

$$\ell_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \qquad i = 1, \cdots, n$$

la lunghezza di tali segmenti, scegliendo:

$$t_0 = 0 \qquad e \qquad t_i = \sum_{k=1}^{i} \ell_k \qquad i = 1, \cdots, n.$$

In tale caso ogni t_i rappresenta la lunghezza cumulativa della spezzata congiungente i punti P_i. La curva che si ottiene è detta *spline parametrica di lunghezza cumulativa.*

```
%----------------------------------------------------------------
function[xi,yi]=par_spline(x,y) t(1)=0; for i=1:length(x)-1
    t(i+1)=t(i)+sqrt((x(i+1)-x(i))^2+(y(i+1)-y(i))^2);
end
z=[t(1):(t(length(t))-t(1))/100:t(length(t))];
xi=spline(t,x,z); yi=spline(t,y,z);
%----------------------------------------------------------------
```

- Costruire e rappresentare insieme ai nodi usati la spline parametrica di lunghezza cumulativa utilizzando i vettori:

x	1	4	5	8	10	13	11	13	14	13
y	1	3	5	2	4	7	9	11	10	9

x	−2	−2	−2	−1	0	1	2	2	2	2	2	1	0	−1	−2	−2	−2
y	0	1	2	2	2	2	2	1	0	−1	−2	−2	−2	−2	−2	−1	0

x	−2	−2	−2	−1	0	1	2	2	2	2	2	1	0	−1	−2	−2	−2
y	0	1	2	2	2	2	2.5	1	0	−1	−2	−2	−2	−2	−2	−1	0

x	3	10	10	5	5	9	9	6	6	8	8	7	7
y	2	2	9	9	4	4	8	8	5	5	7	7	6

- Descrivere la costruzione di una spline interpolante quadratica con assegnata derivata prima nel primo nodo.

Esercizio 3. Formule di quadratura per integrali doppi
Fissati due interi m e n e scelte due formule di quadratura monodimensionali per il calcolo dell'integrale di una funzione $g(x)$

$$Q_m(g) = \sum_{i=0}^{m} w_i g(x_i) \quad \text{e} \quad Q_n(g) = \sum_{i=0}^{n} z_i g(\bar{x}_i),$$

una formula di quadratura per l'integrale in due dimensioni

$$\int_a^b \int_c^d f(x,y)dxdy$$

si ottiene come "prodotto" di Q_m e Q_n nel modo seguente

$$Q_{m,n}(f) = [Q_m \times Q_n](f) = \sum_{i=0}^{m} \sum_{j=0}^{n} w_i z_j f(x_i, y_j),$$

dove $x_i \in [a,b]$, $i = 0, \ldots, m$ e $y_j \in [c,d]$, $j = 0, \ldots, n$.

- Scrivere la formula prodotto di due formule dei trapezi composite e implementarla in MatLab.
- Utilizzare tale function per l'approssimazione dell'integrale

  $$\int_0^1 \int_0^1 \frac{1}{1+x+y}\, dxdy$$

  prendendo $m = n$, per diversi valori di m, e confrontare con il risultato esatto.
- **(Solo per gli studenti di Matematica)** La stima del resto per la formula prodotto di due formule dei trapezi composite

  $$r = -\frac{(b-a)(d-c)}{12}\left[\frac{(b-a)^2}{m^2}\frac{\partial^2 f}{\partial x^2}(\xi_1, \eta_1) + \frac{(d-c)^2}{n^2}\frac{\partial^2 f}{\partial y^2}(\xi_2, \eta_2)\right].$$

  con $(\xi_1, \eta_1), (\xi_2, \eta_2) \in [a,b] \times [d,c]$.
  Per l'integrale del punto precedente, prendendo $m = n$ dire per quale valore di m l'errore assoluto risulta in modulo minore di $0.5 \cdot 10^{-3}$. Confrontare con i risultati numerici.

Esercizio 4. Metodo di Rilassamento (SOR)
Si studi, al variare del parametro di rilassamento ω in un opportuno intervallo, la convergenza del metodo SOR applicato alle matrici

$$A_1 = \begin{pmatrix} -3 & 3 & -6 \\ -4 & 7 & -8 \\ 5 & 7 & -9 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 7 & 4 & -7 \\ 4 & 5 & -3 \\ -7 & -3 & 8 \end{pmatrix},$$

specificando per entrambe il valore ω_0 che rende minimo il raggio spettrale della matrice di iterazione.
Applicare il metodo di rilassamento per risolvere i sistemi $A_i\mathbf{x} = b_i$, dove

$$b_1 = [-6, -5, 3]^T, \qquad b_2 = [4, 6, -2]^T,$$

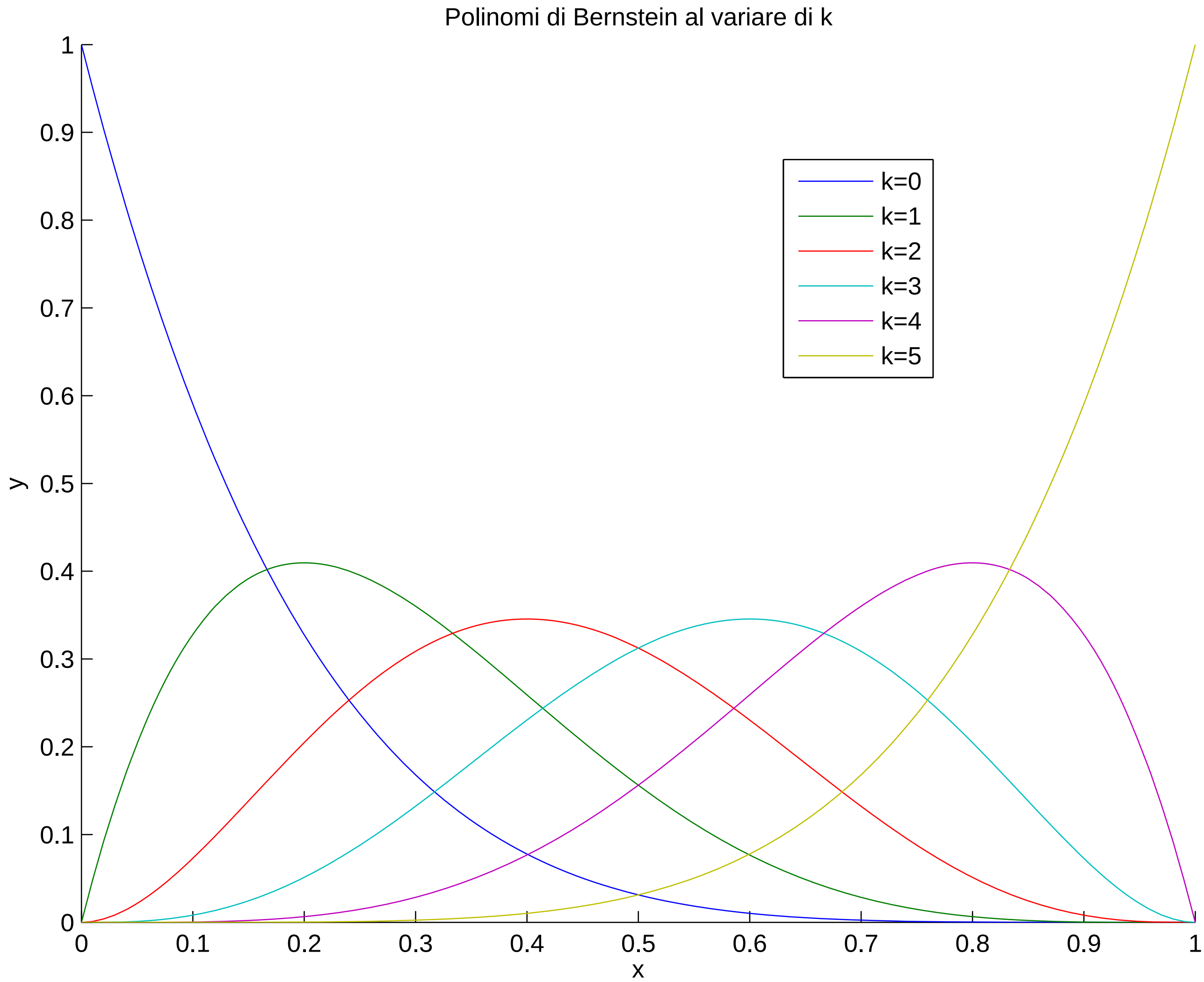
a partire dal punto iniziale $\mathbf{x}^{(0)} = \mathbf{0}$. Confrontare il numero di iterate impiegate per raggiungere una precisione prefissata per due diversi valori ω (ω_0 e 0.1).

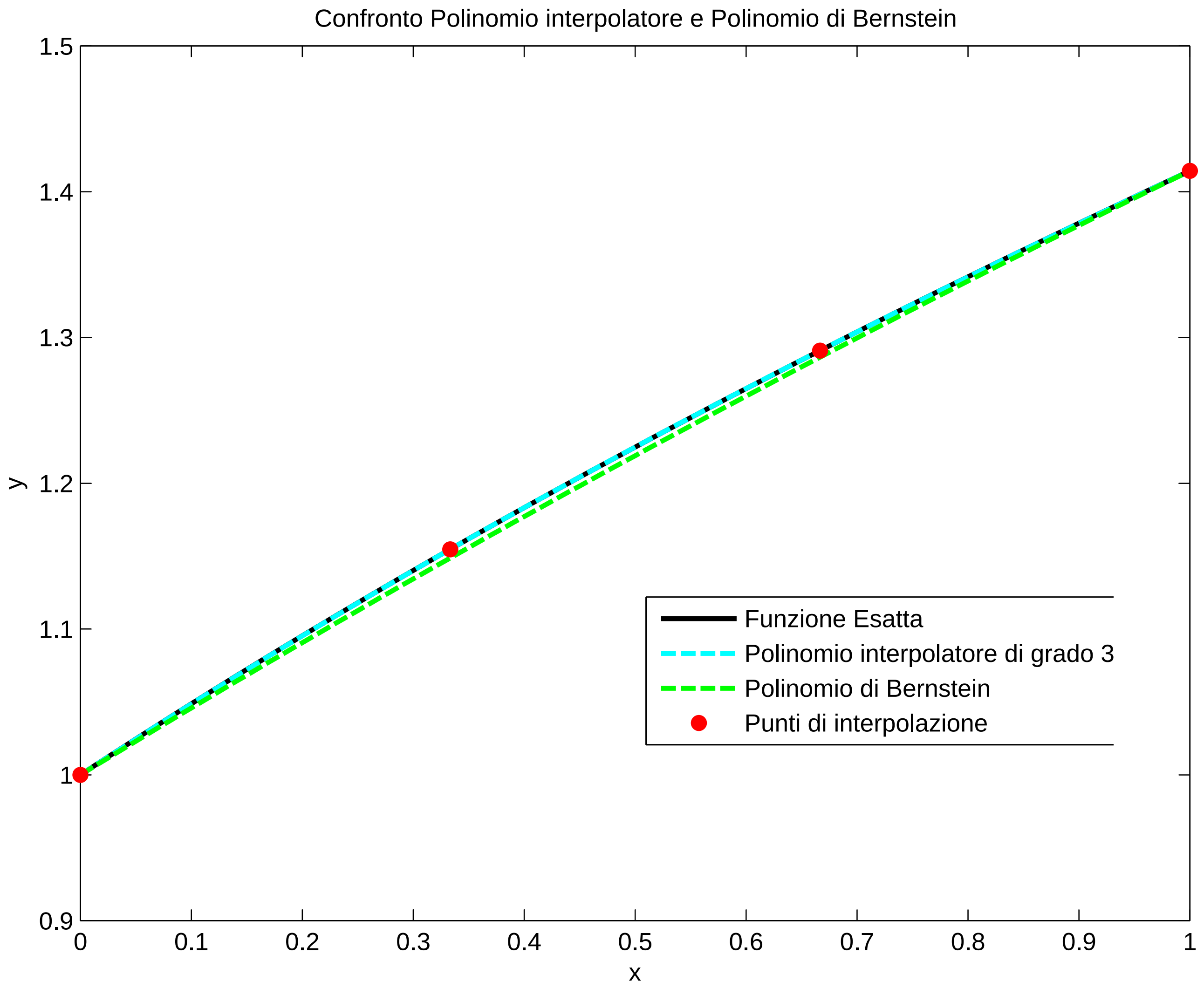
```
% Esercizio 1 - Esercitazione 9
clear all
close all
clc

n=5;
% intervallo di punti per costruire la funzione e i polinomi. intervallo
% noto dalla funzione bernstein
xx = 0:0.01:1;
% calcoliamo i polinomi di Bernstein per k=0...n
for k= 0:n
    yb = bernstein(k,n);
    hold all;
    plot(xx,yb);
end
% numero 4 punti nell'intervallo [0,1]
x = linspace(0,1,4);
% funzione data valutata nei 4 punti nell'intervallo [0,1]
y = sqrt(x+1);
% costruzione polinomio interpolatore di terzo grado sui 4 nodi e
% valutazione nei punti xx
p_inter = polyfit(x,y,3);
yy_inter = polyval(p_inter,xx);
% costruzione polinomio approssimante mediante i polinomi di Bernstein
yy_bern = zeros(101,1);
for i=0:3
    yy_bern = yy_bern + y(i+1)*bernstein(i,3)';
end
% GRAFICO
figure
box on
plot(xx,sqrt(xx+1), xx,yy_inter, xx,yy_bern, x,y)
```

FUNZIONE BERNSTEIN

```
function [ber] = bernstein(k,n)
    i = 0;
    if k == 0
        coef = 1;
    else
        % prod: produttoria degli elementi del array
        coef = prod([1:n])/(prod([1:k])*prod([1:n-k]));
    end
    for t = 0:0.01:1
        i = i+1;
        ber(i) = coef*t^k*(1-t)^(n-k);
    end
end
```

Grafico Esercizio 1 - Esercitazione 9





```
% Esercizio 2 - Esercitazione 9
clear all
close all
clc

x1 = [1 4 5 8 10 13 11 13 14 13];
y1 = [1 3 5 2  4  7  9 11 10  9];
x2 = [-2 -2 -2 -1  0  1  2  2  2  2  2  1  0 -1 -2 -2 -2];
y2 = [ 0  1  2  2  2  2  2  1  0 -1 -2 -2 -2 -2 -2 -1  0];
x3 = [-2 -2 -2 -1  0  1   2  2  2  2  2  1  0 -1 -2 -2 -2];
y3 = [ 0  1  2  2  2  2 2.5  1  0 -1 -2 -2 -2 -2 -2 -1  0];
x4 = [3 10 10  5  5  9  9  6  6  8  8  7  7];
y4 = [2  2  9  9  4  4  8  8  5  5  7  7  6];

% S_(k,delta)(t) = (S_(k,delta,x)(t),S_(k,delta,y)(t))
% tramite le spline delle x e y costruiamo la spline parametrica
subplot(2,2,1);
[x, y] = par_spline(x1,y1);
plot(x,y,x1,y1);

subplot(2,2,2);
[x, y] = par_spline(x2,y2);
plot(x,y,x2,y2);

subplot(2,2,3);
[x, y] = par_spline(x3,y3);
plot(x,y,x3,y3);

subplot(2,2,4);
[x, y] = par_spline(x4,y4);
plot(x,y,x4,y4);
```

FUNZIONE SPLINE PARAMETRICA

```
function [xi, yi] = par_spline(x, y)
% Spline parmetrica
% ------------INPUT----------------
% t : vettore contenente la lunghezza cumulativa per ogni tratto
% (intervallo tra ogni coppia di punti)
% sqrt((x(i+1)-x(i))^2+(y(i+1)-y(i))^2) : lunghezza di ogni segmento (l)
% --------------------------------
    t(1) = 0;
    for i=1:length(x)-1
       t(i+1)=t(i)+sqrt((x(i+1)-x(i))^2+(y(i+1)-y(i))^2);
    end
    % punti su cui interpolare la spline
    z = [t(1):(t(length(t))-t(1))/100:t(length(t))];
    % spline cubica rispetto alla variabile t, interpolante z
    xi = spline(t,x,z);
    % spline cubica rispetto alla variabile t, interpolante z
    yi = spline(t,y,z);
end
```

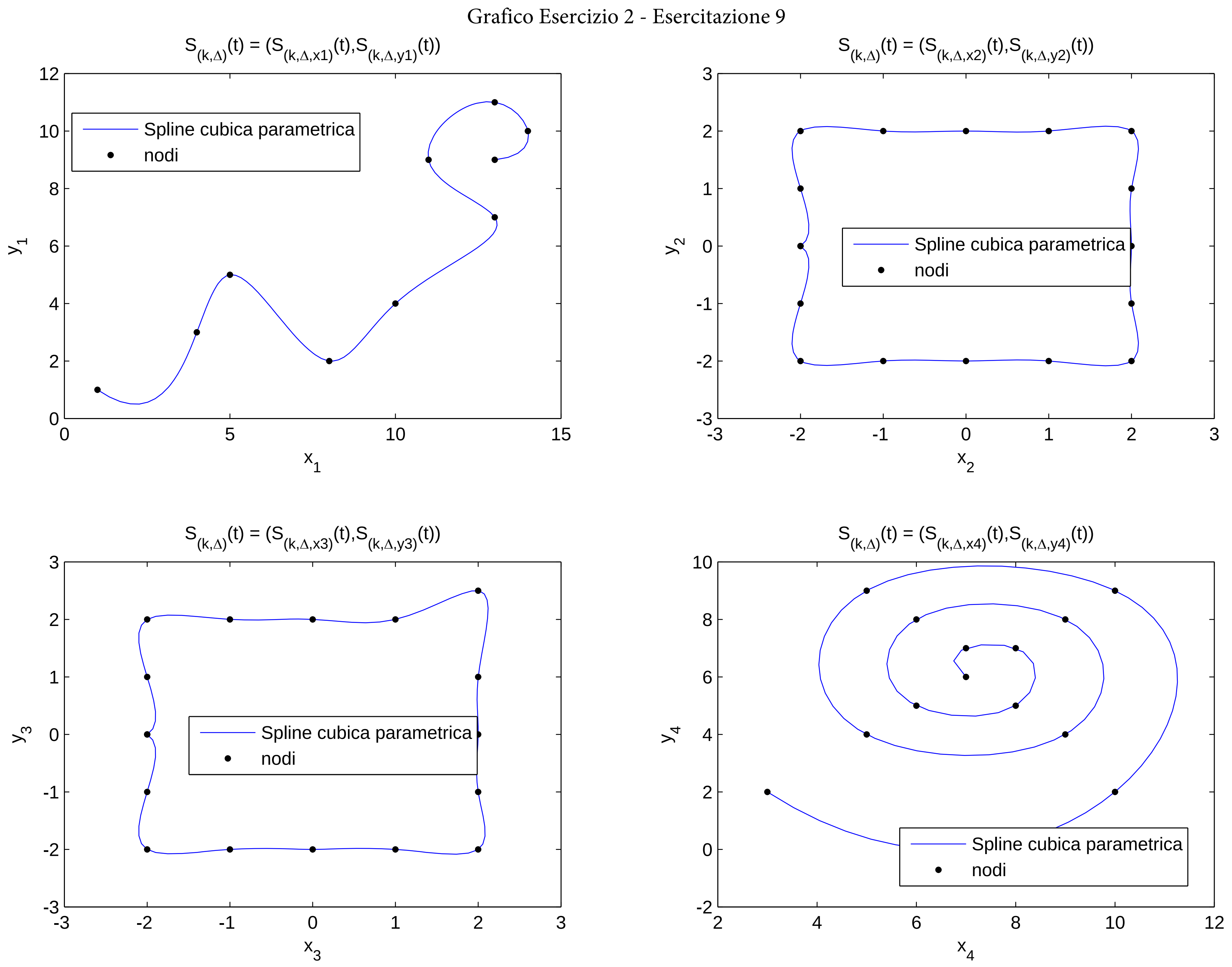


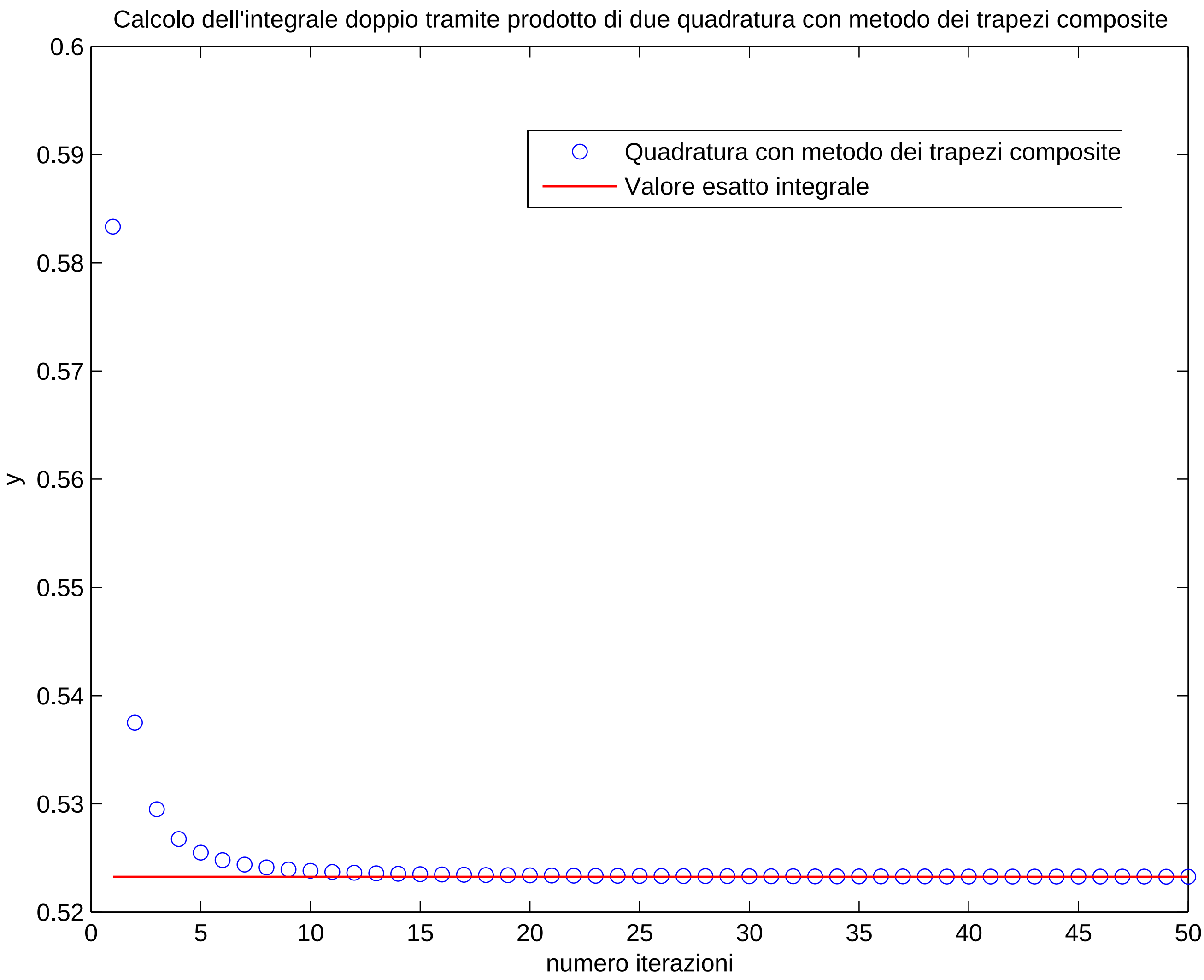
Grafico Esercizio 2 - Esercitazione 9
S(k,Δ)(t) = (S(k,Δ,x1)(t),S(k,Δ,y1)(t))
Spline cubica parametrica
nodi
x1
y1
S(k,Δ)(t) = (S(k,Δ,x2)(t),S(k,Δ,y2)(t))
Spline cubica parametrica
nodi
x2
y2
S(k,Δ)(t) = (S(k,Δ,x3)(t),S(k,Δ,y3)(t))
Spline cubica parametrica
nodi
x3
y3
S(k,Δ)(t) = (S(k,Δ,x4)(t),S(k,Δ,y4)(t))
Spline cubica parametrica
nodi
x4
y4

```
% Esercizio 3 - Esercitazione 9
clear all
close all
clc
% quadratura dell'integrale da 0 a 1 dell'integrale da 0 a 1 della funzione
% 1/(1+x+y) tramite il metodo dei trapezi compositi
% estremi dell'integrale di x
a=0;
b=1;
% estremi dell'integrale di y
c=0;
d=1;
% calcolo del valore esatto dell'integrale doppio
funct1 = @(x,y) 1./(1+x+y);
val_esa = integral2(funct1,0,1,0,1);
% iterazioni, cioè suddivisioni degli intervalli degli integrali
iter = 50;
% quadr_trap_comp: vettore contenente il risultato degli integrali con i
% iterazioni
for i=1:iter
   n=i;
   % poniamo m=n come richiesto dall'esercizio
   m=n;
   quadr_trap_comp(m) = trap_comp_integr_dop(a,b,c,d,m,n, funct1 );
end
plot(1:iter,quadr_trap_comp, 1:iter, val_esa*ones(iter,1))
```

FUNZIONE TRAPEZI COMPOSITE PER INTEGRALI DOPPI

```
function quadr = trap_comp_integr_dop(a,b,c,d,m,n,funct)
% Calcolo dell'integrale doppio tramite prodotto di due quadratura
% con metodo dei trapezi composite.
% ------------INPUT----------------
% a,b: Estremi integrazione su x
% c,d: Estremi integrazione su y
% m: iterate della formula dei trapezi per l'integrale su x, cioè
%    sottintervalli in dividere l'intervallo di x
% n: iterate della formula dei trapezi per l'integrale su y, cioè
%    sottintervalli in dividere l'intervallo di y
% funct: Funzione da integrare su x,y
% ---------------------------------
   quadr = 0;
   % passo dei sottintervalli riferiti all'integrale di x
   H = (b-a)/m;
   % passo dei sottintervalli riferiti all'integrale di y
   K = (d-c)/n;
   % punti dei sottointervalli
   x = linspace(a,b,m+1);
   y = linspace(c,d,n+1);
   % sommatoria di tutti i valori delle quadrature calcolate nei
   % sottointervalli
   for i = 1:m
      for j = 1:n
         quadr = quadr + funct(x(i),y(j)) + funct(x(i),y(j+1)) + funct(x(i+1),y(j)) + funct(x(i+1),y(j+1)) ;
      end
   end
   % complietiamo la formula dei trapezi composita con h/2 e k/2
   quadr = (H/2)*(K/2)*quadr;
end
```

Grafico Esercizio 3 - Esercitazione 9



```
% Esercizio 4 - Esercitazione 9
diary esercizio_4.txt
clear all
close all
clc
A1 = [-3  3 -6; -4  7 -8; 5   7 -9];
A2 = [ 7  4 -7; 4  5 -3; -7 -3  8];
b1 = [-6 -5 3]';
b2 = [4, 6, -2]';
% vettori soluzione
x=zeros(3,1);
% numero massimo di iterazioni
iter_max=50;
% tolleranza  1*10^-1
toll = 0.1;

% errore metodo Jacobi per il sistema lineare A_1*x=b_1 e vettore soluzione
% dell'ultima iterazione
disp('il metodo di Jacobi per il sistema lineare A_1*x=b_1 è ');
[err_A1,x_final1] = Jacobi(A1,b1,x,iter_max,toll);
x_final1
% errore metodo Jacobi per il sistema lineare A_2*x=b_2 e vettore soluzione
% dell'ultima iterazione
disp('il metodo di Jacobi per il sistema lineare A_2*x=b_2 è ');
[err_A2,x_final2] = Jacobi(A2,b2,x,iter_max,toll);
x_final2
% GRAFICI
subplot(2,1,1)
plot(1:length(err_A1), err_A1)
subplot(2,1,2)
plot(1:length(err_A2), err_A2)
% possiamo affermare che la risoluzione tramite il metodo di Jacobi
% del sistema lineare A_1*x=b_1 è convergente, cioè maggiori sono le
% iterazioni minore è l'errore commesso dal metodo, invece per il sistema
% lineare A_2*x=b_2 è non convergente.
diary off
```

COMMAND WINDOW

```
il metodo di Jacobi per il
sistema lineare A_1*x=b_1 è
Convergente

in
i =  9
iterazioni

x_final1 =

   0.9640
   1.0122
   0.9955
```

```
il metodo di Jacobi per il

sistema lineare A_2*x=b_2 è

 Non convergente

x_final2 =

   1.0e+06 *

    5.2138
    4.8977
   -4.5437
```

FUNZIONE JACOBI

```
function [err, xnew] = Jacobi(A, b, x0, iter_max, toll)
% valutiamo l'errore del metodo di Jacobi iterando fino a raggiungere una
% determinata tolleranza(toll).
% ------------INPUT----------------
% iter_max : numero massimo iterazioni
% toll : tolleranza
% ---------------------------------

    D = diag(diag(A));
    B = D-tril(A);
    C = D-triu(A);

    invD = diag(1./diag(A));
    % matrice di iterazione
    J = invD*(B+C);
    % esaminiamo la convergenza del metodo valutando la norma infinita
    % degli autovalori di J
    rag_spet = norm(eig(J),inf);
    if ( rag_spet < 1 )
        disp('Convergente');
    else
        disp('Non convergente');
    end

    xold = x0;
    i = 1;
    test = 1;
    err = zeros(1, iter_max);
    while i < iter_max && test > toll
        % ricaviamo il vettore soluzione di quella spefica iterazione i
        xnew = J*xold+invD*b;
        err(i) = norm(xnew-xold);
        test = err(i);
        xold = xnew;
        i = i+1;
    end
    err = err(1:i-1);
    % numero di iterazioni effettuate
    if ( rag_spet < 1 )
        disp(' in '); i
        disp(' iterazioni');
    end
end
```

Grafico Esercizio 4 - Esercitazione 9