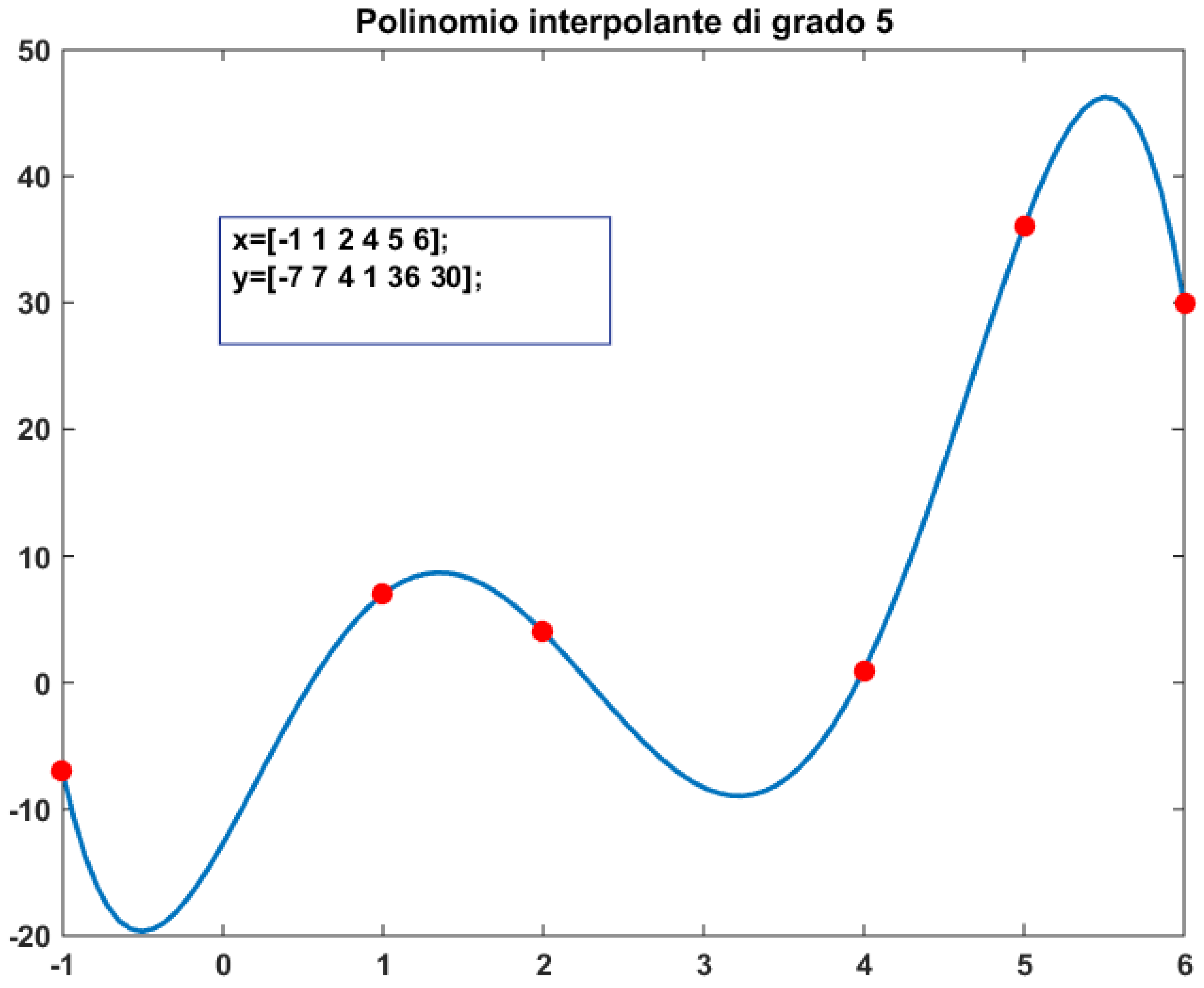
Interpolazione polinomiale

n=6, grado del polinomio n-1



polyfit

Polynomial curve fitting

Description

`p = polyfit(x,y,n)` returns the coefficients for a polynomial `p(x)` of degree `n`

coef=polyfit(x,y,g)

x= vettore di ascisse lungo n
y= vettore di valori noti lungo n
g=grado del polinomio
coef=coefficienti del polinomio

Matlab mette a disposizione la routine polyval per la valutazione del polinomio e

polyval

Polynomial evaluation

Description

e:

`y = polyval(p,x)` evaluates the polynomial `p` at each point in `x`. The argument `p` is a vector of length `n+1` whose elements are the coefficients (in descending powers) of an `n`th-degree polynomial:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}.$$

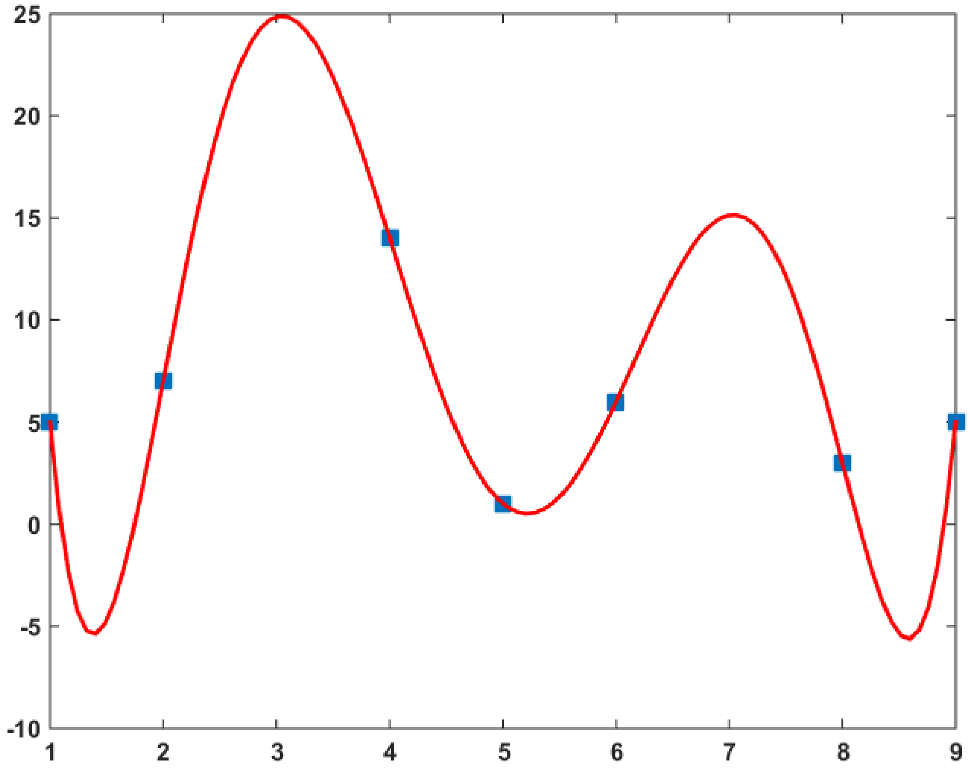
f=polyval(coef,t)

coef=coefficienti del polinomio
t=vettore di punti in cui valutare il polinomio
f= vettore di valori del polinomio nei punti t

```
x=[1 2 4 6 8 9 5];
y=[5 7 14 6 3 5 1];
t=linspace(1,9);
p=polyfit(x,y,6);
f=polyval(p,t);
plot(x,y,'s',t,f)
```

n-1

```
>> n=length(x)
n=
   7
```



```
function [t]=interpol(x,y,s)
    %----------------------------
--------------------
% Interpolazione
%
% In input:
% x: nodi.
% y: valori nei nodi.
% s: nodi su cui calcolare
l'interpolante.
%
% In output:
% t: valori dell'interpolante o
approssimante.
%--------------------------------
----------------
m=length(x)-1;
coeff=polyfit(x,y,m);
t=polyval(coeff,s);
end
```

```
% esempio di utilizzo:
% % definisco gli estremi dell'intervallo
 estremoA = 2;
 estremoB = 5;
% % definisco il numero di nodi da interpolare
nodi = 4;
% % utilizzo la funzione linspace per generare un
vettore x
% %   con 'nodi' elementi compresi tra 'estremoA' e
'estremoB'
 x = linspace(estremoA,estremoB,nodi);
% % definisco una funzione da applicare alle x
 y = sin(x);
% % genero un vettore z con tutti i punti
dell'intervallo
 z = estremoA:0.01:estremoB;
% % genero il polinomio di lagrange
 for i=1:length(z)
t(i)=interpol(x,y,z(i));
 end
% % disegno il grafico che mi mostra la funzione, il
% % polinomio interpolatore e i nodi
 ydis =sin(z);
 plot(xdis,ydis,'r',z,t,'b-.',x,y,'r*')
```

Grafico
Con 4 nodi

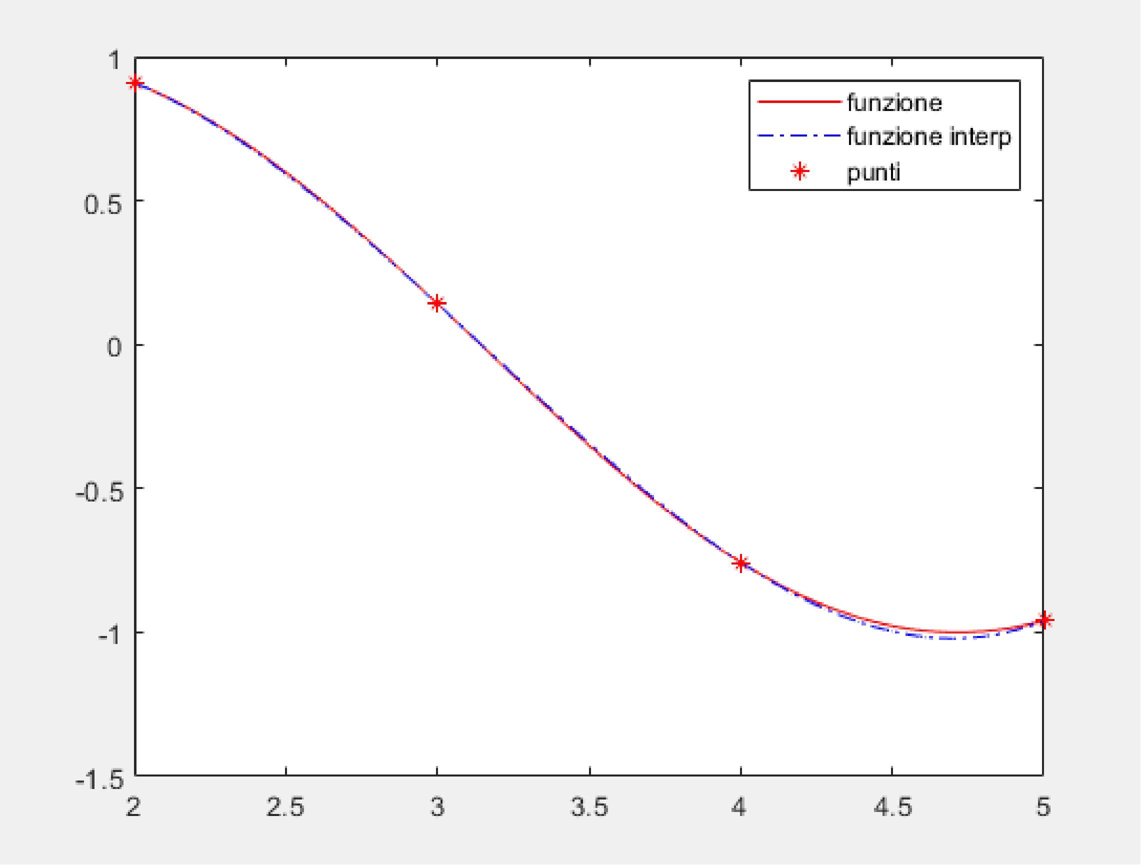


Grafico intervallo [-1,1] funzione seno e interpolazione polinomiale

n=4;

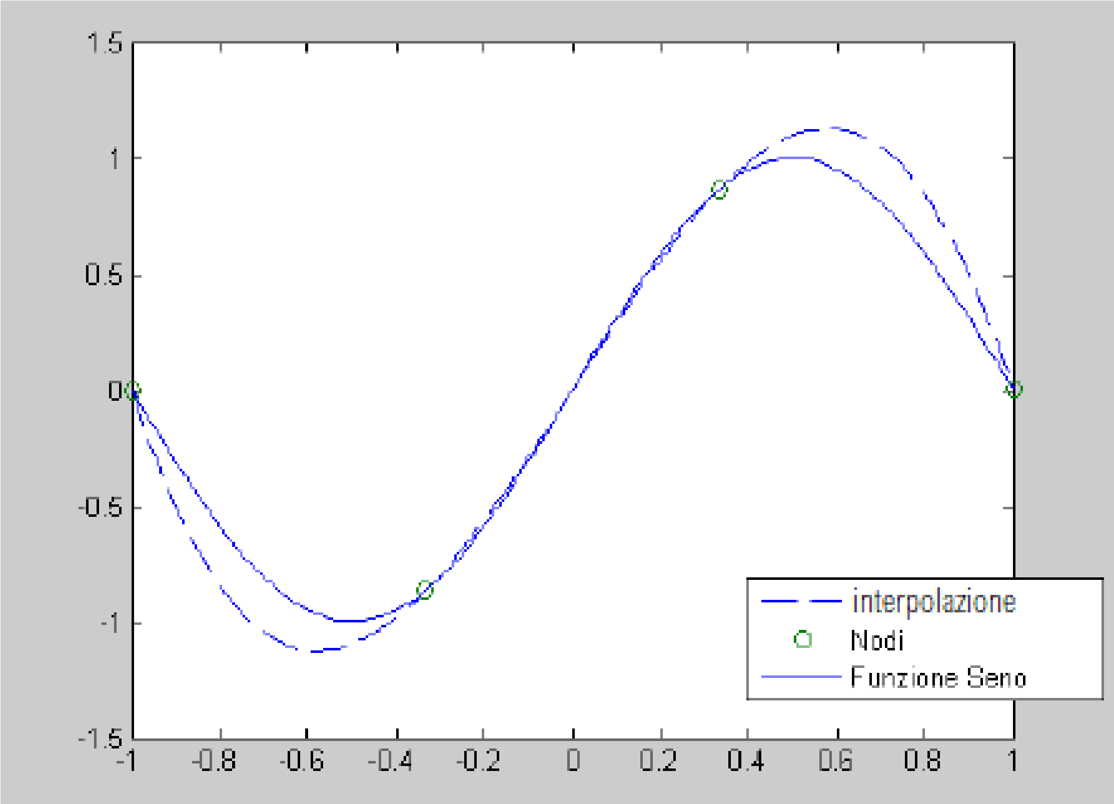


Grafico intervallo [-1,1] funzione seno e interpolazione polinomiale

n=12;

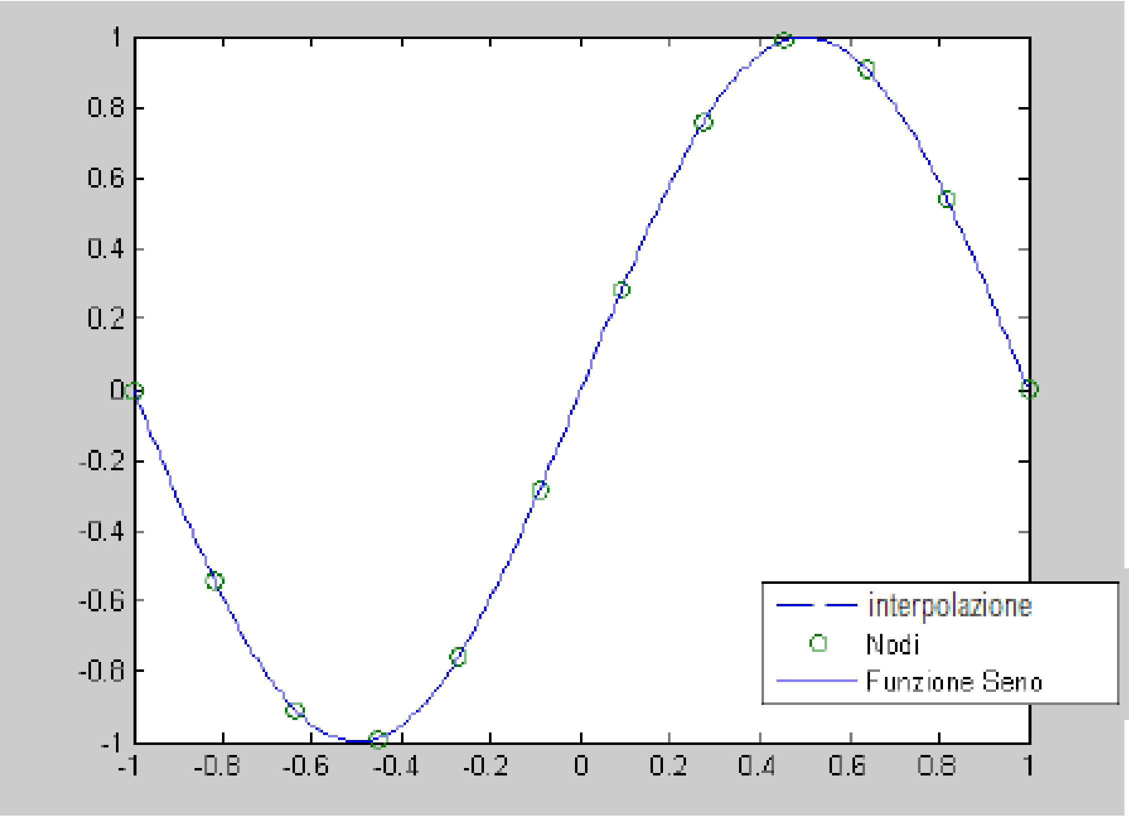


Grafico intervallo [-1,1] funzione seno e interpolazione polinomiale

n=50;

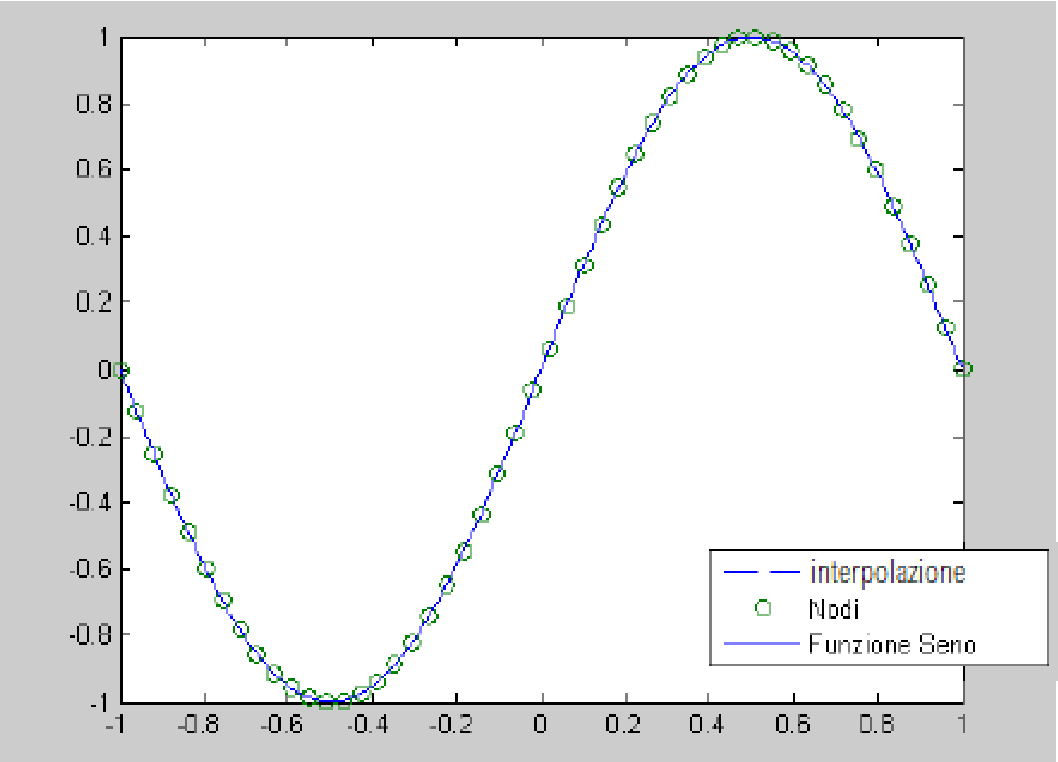


Grafico intervallo [-1,1] funzione seno e interpolazione polinomiale

Al crescere di n aumentano le oscillazione

n=70;

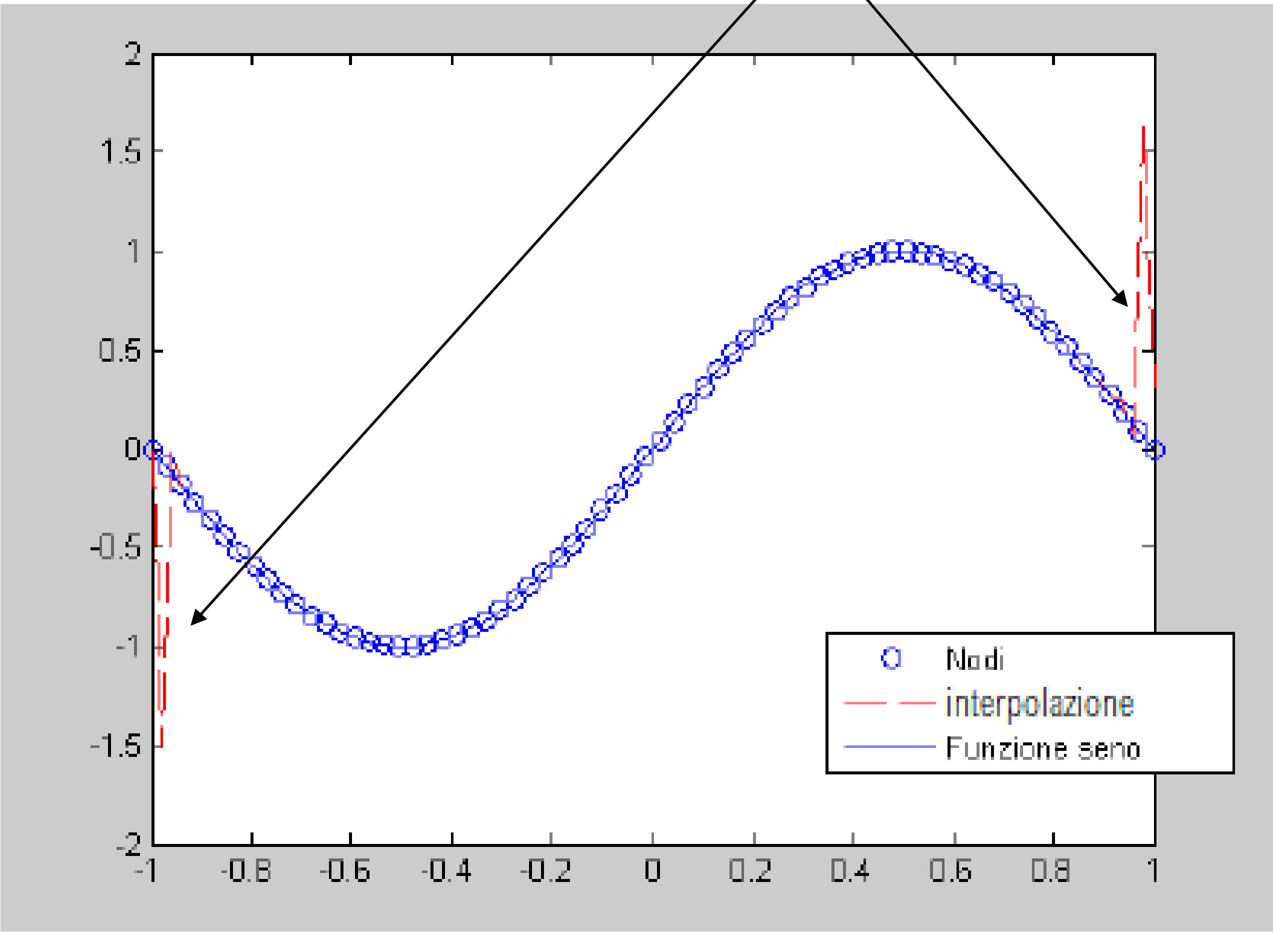
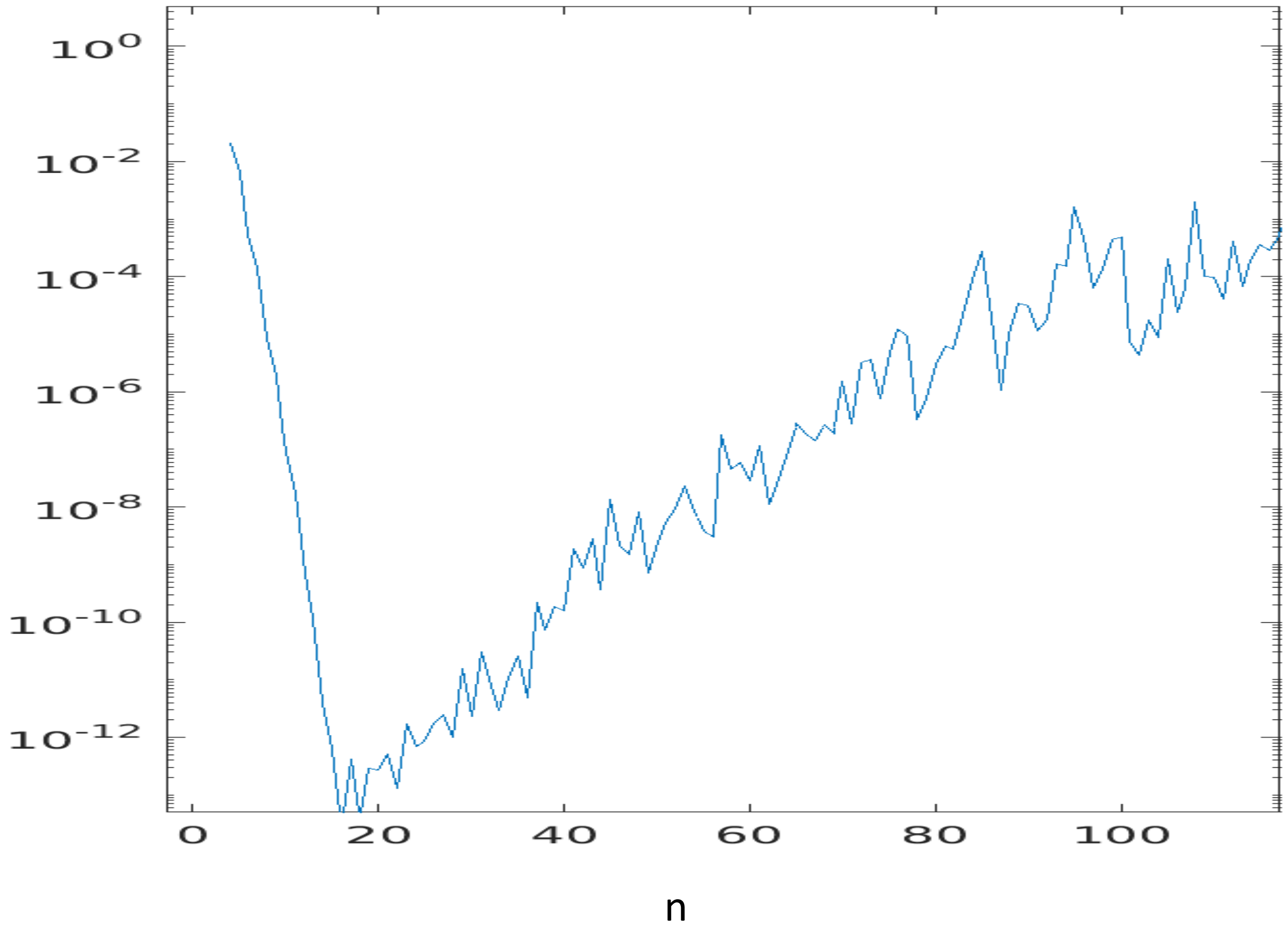


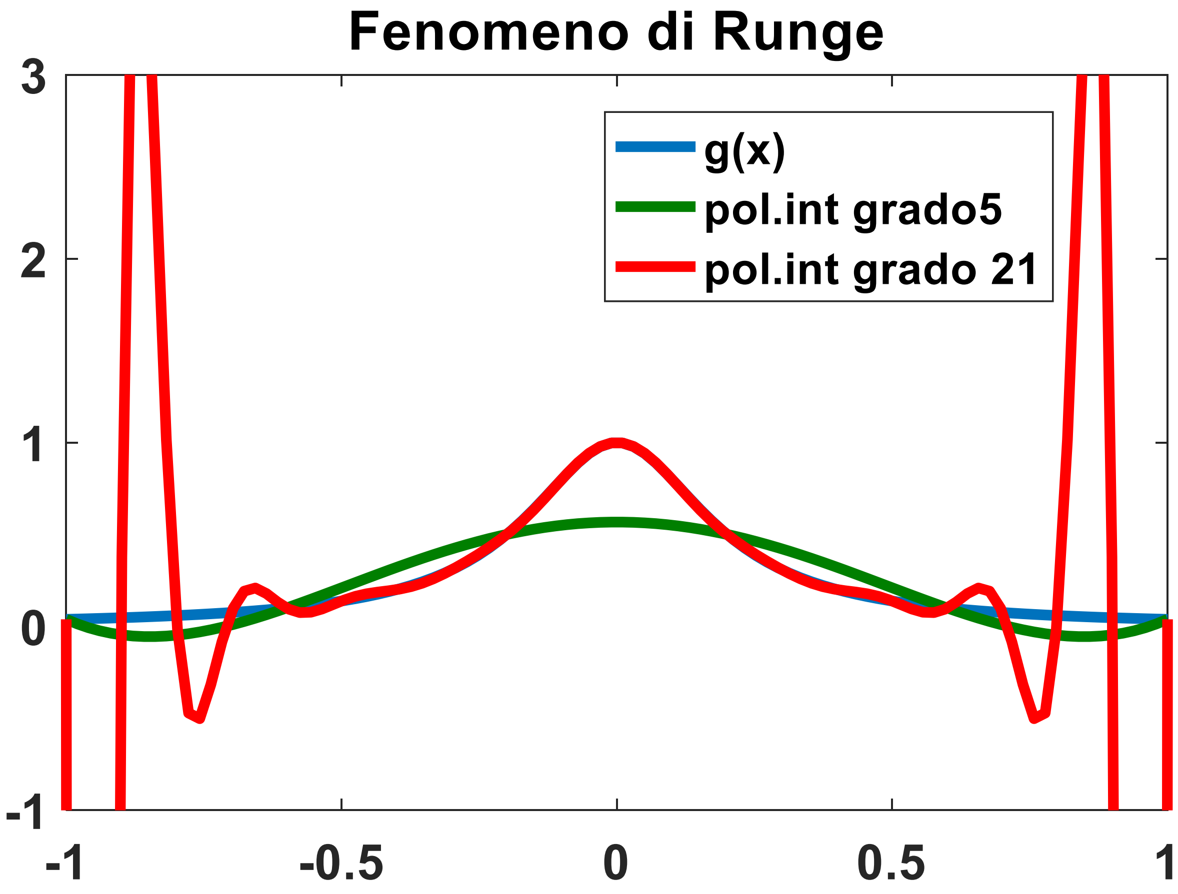
Grafico errore in norma infinito



Limiti dell’interpolazione polinomiale
esempio di Runge

fenomeno di Runge

$g(x)=1/(1+25x^2)$ $x\varepsilon$ [-1,1]



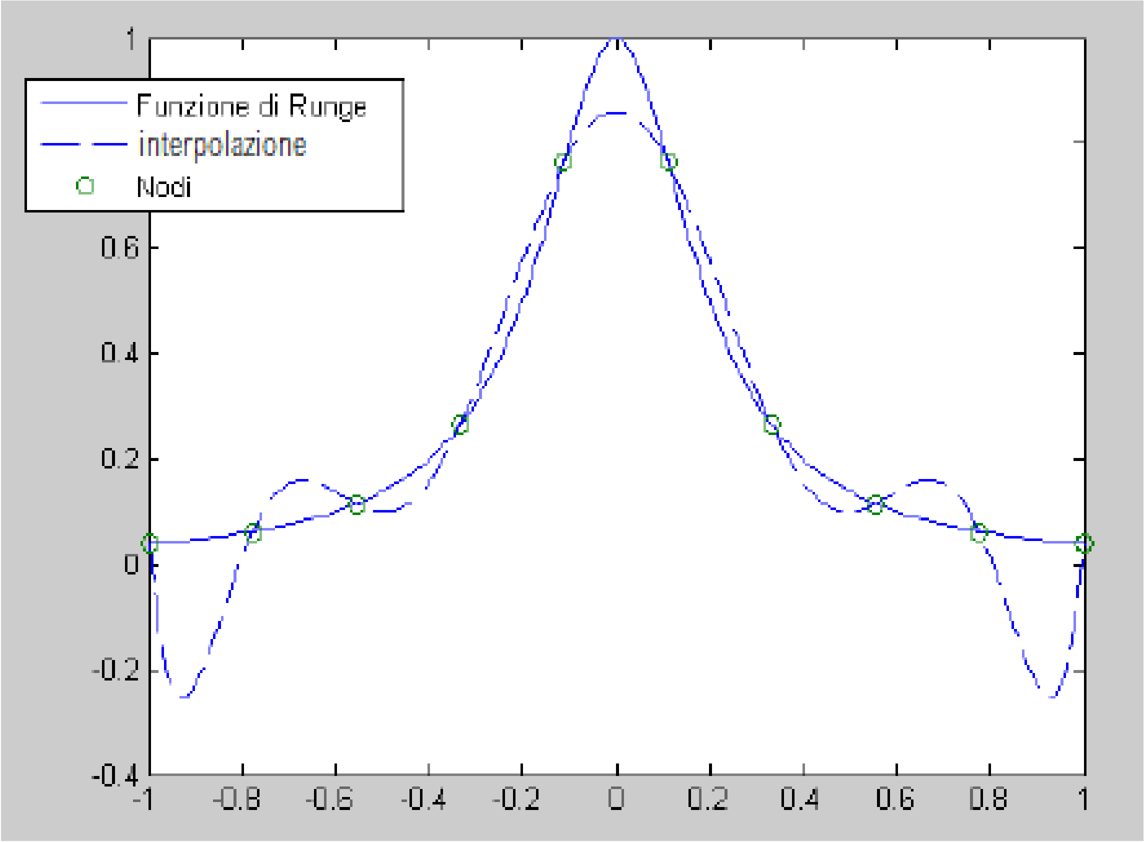
```
% esempio di utilizzo:
% % definisco gli estremi dell'intervallo
 estremoA = -1;
 estremoB = 1;
% % definisco il numero di nodi da
interpolare
nodi = 4;
% % utilizzo la funzione linspace per
generare un vettore x
% %   con 'nodi' elementi compresi tra
'estremoA' e 'estremoB'
 x = linspace(estremoA,estremoB,nodi);
% % definisco una funzione da applicare alle
x
 y = runge(x);
% % genero un vettore z con tutti i punti
dell'intervallo
 z = estremoA:0.01:estremoB;
% % genero il polinomio di lagrange
 for i=1:length(z)
t(i)=interpol(x,y,z(i));
 end
```

```
function [y] = runge(x)

y=1./(25*x.^2+1);
end
```

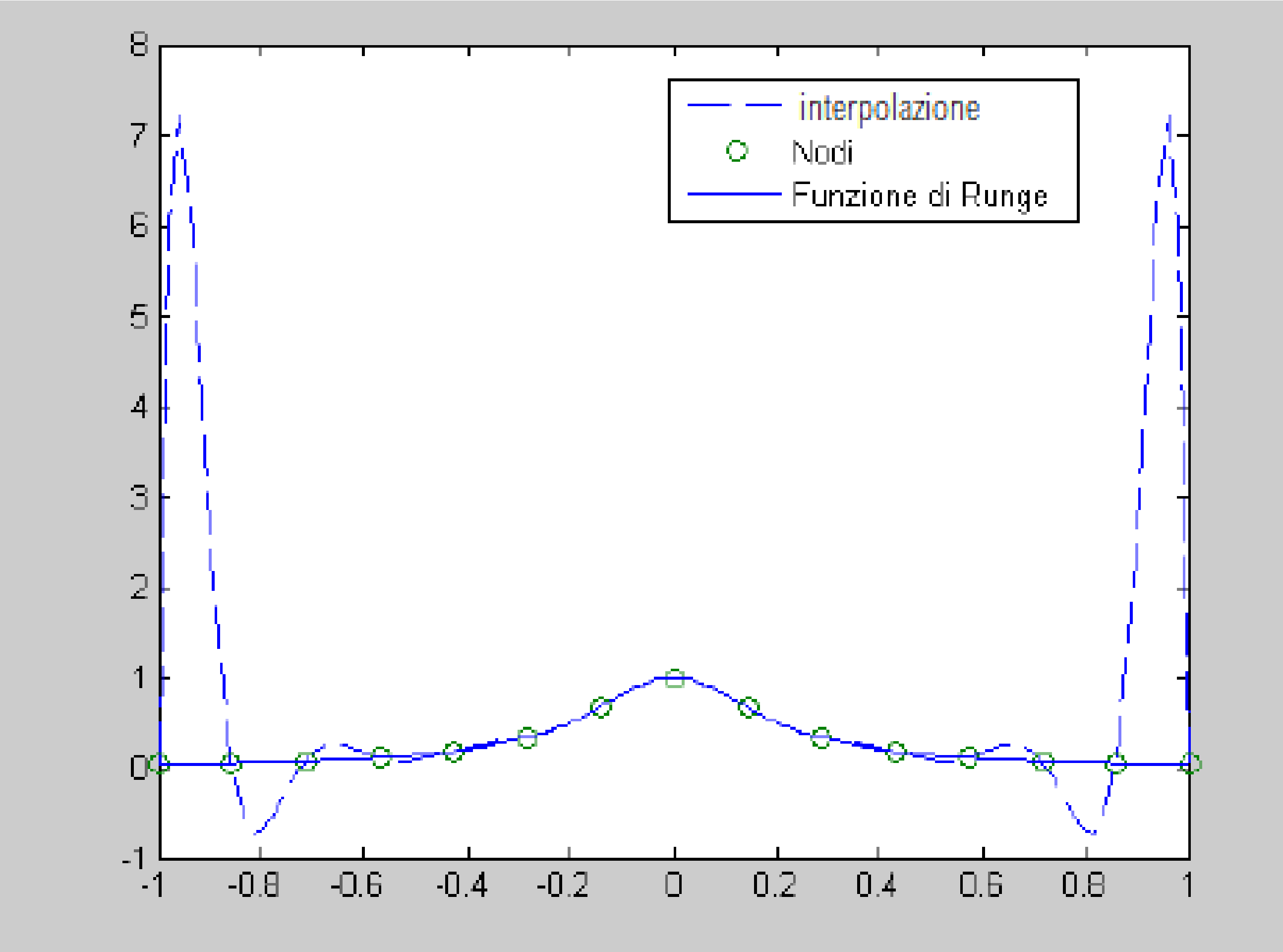
fenomeno di Runge

n=10



fenomeno di Runge

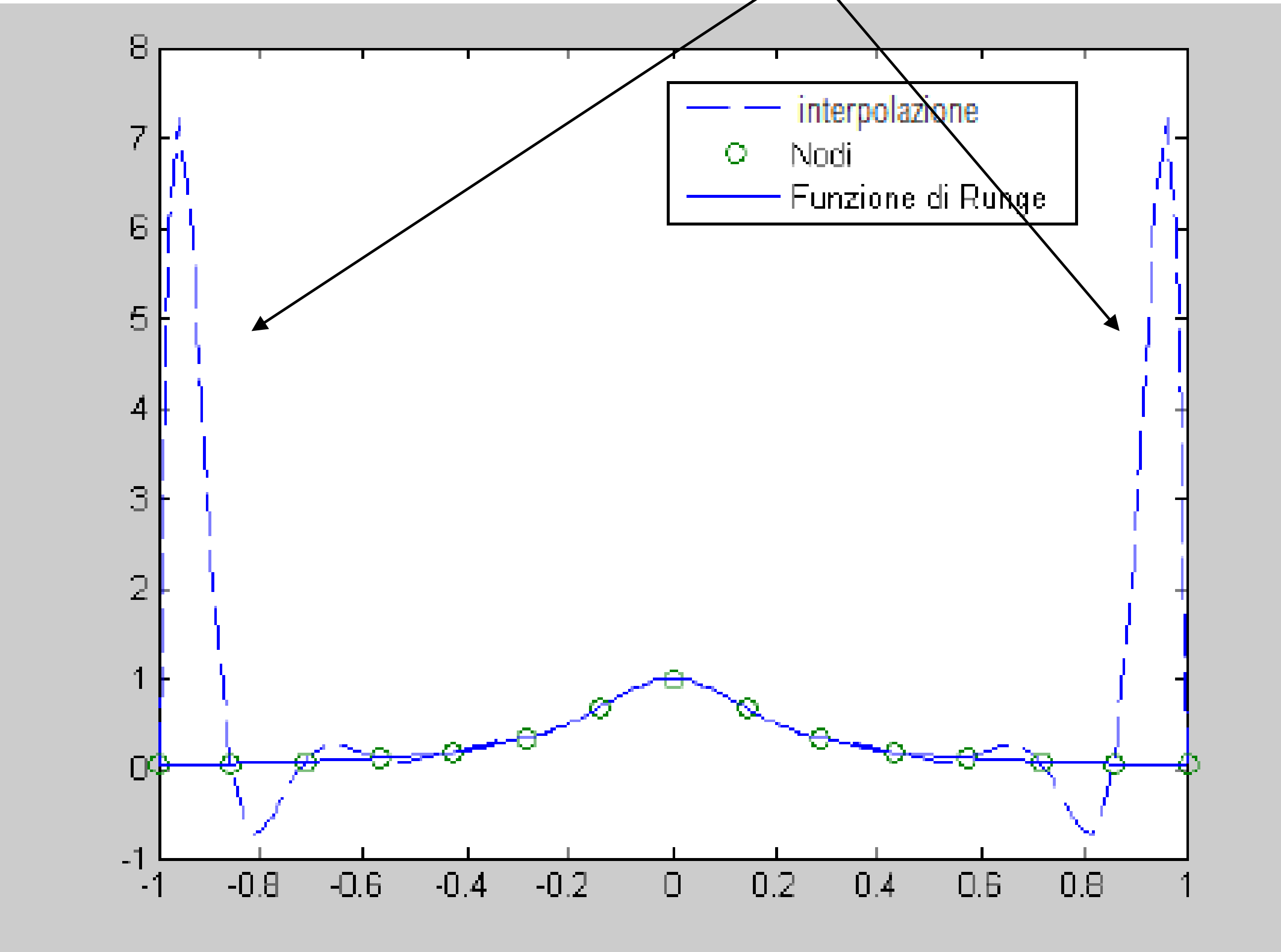
n=15



fenomeno di Runge

Al crescere di n aumentano le oscillazione

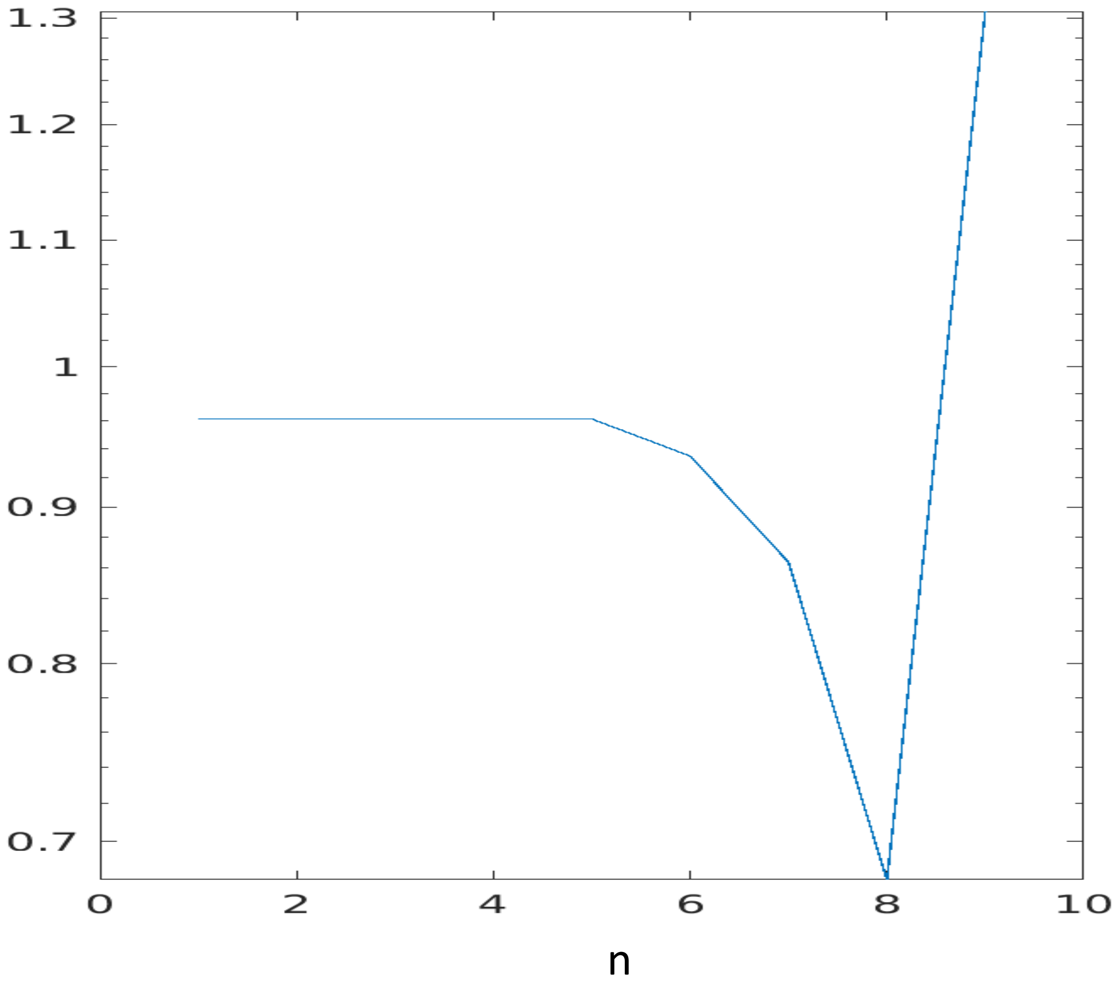
n=15



fenomeno di Runge,

$g(x)=1/(1+25x^2) \quad x \varepsilon [-1,1]$

grafico errore polyfit



Considerimo i seguenti nodi (di Chebychev) non equidistanti

$$x_k = \cos\left(\frac{2k+1}{n+1}\frac{\pi}{2}\right), \qquad k = 0, \ldots, n$$

Grafico 30 nodi
Con punti equidistanti e non equidistanti (chebychev)

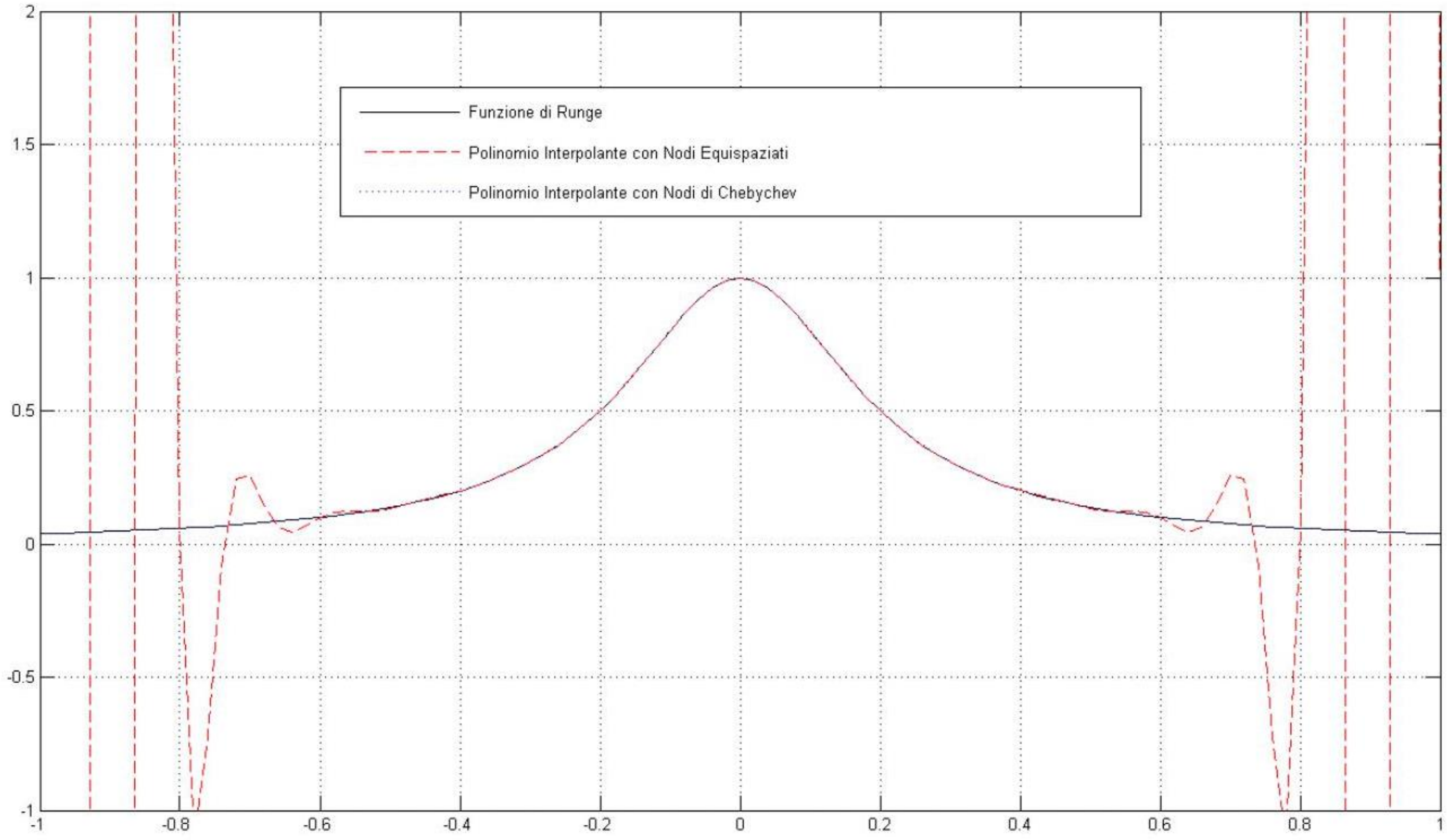
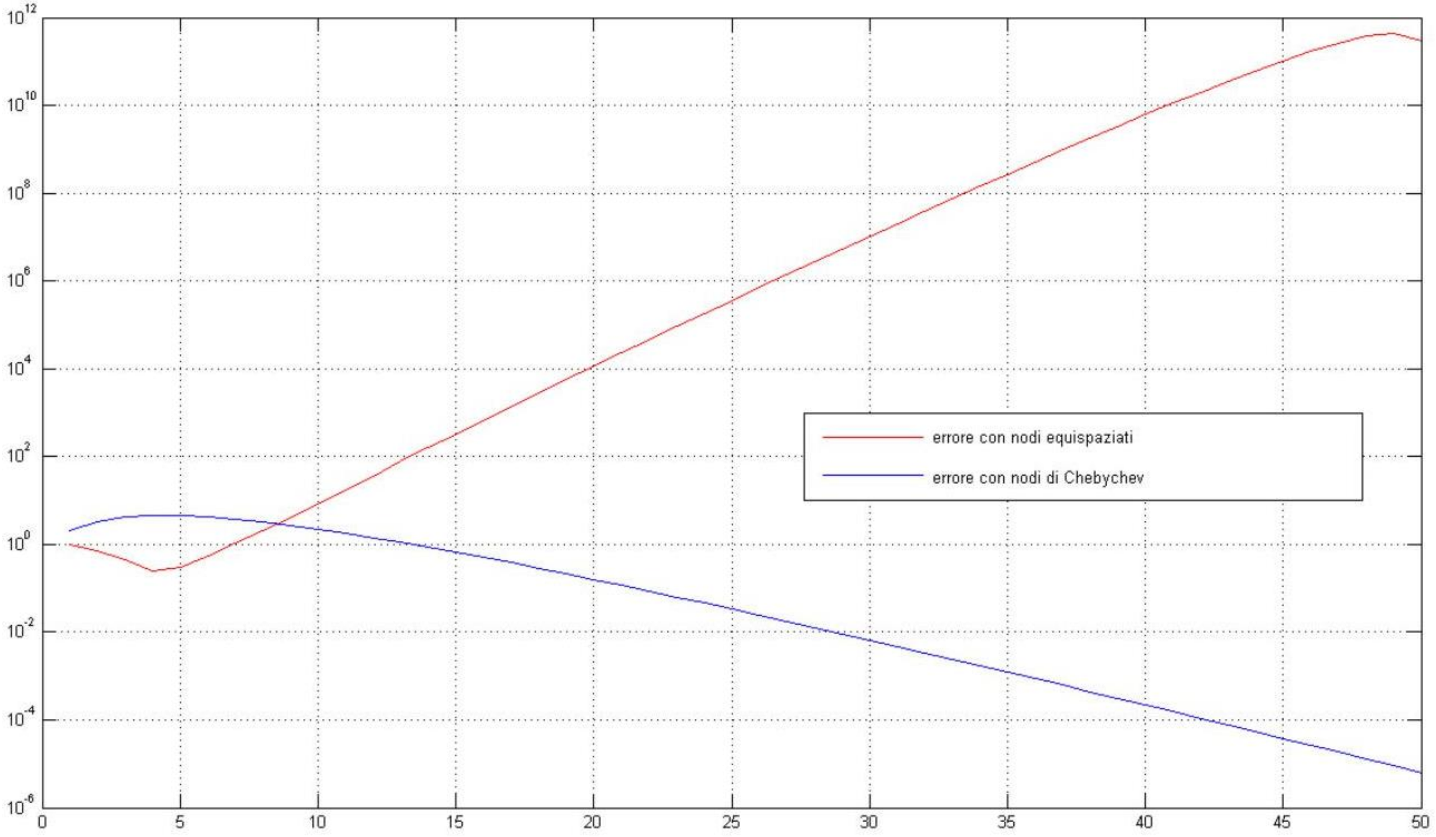


Grafico errore con 30 nodi
Con punti equidistanti e non equidistanti (chebychev)



Interpolazione Polinomiale a tratti

matlab

z=interp1(x,y,t) interpolante a tratti lineare

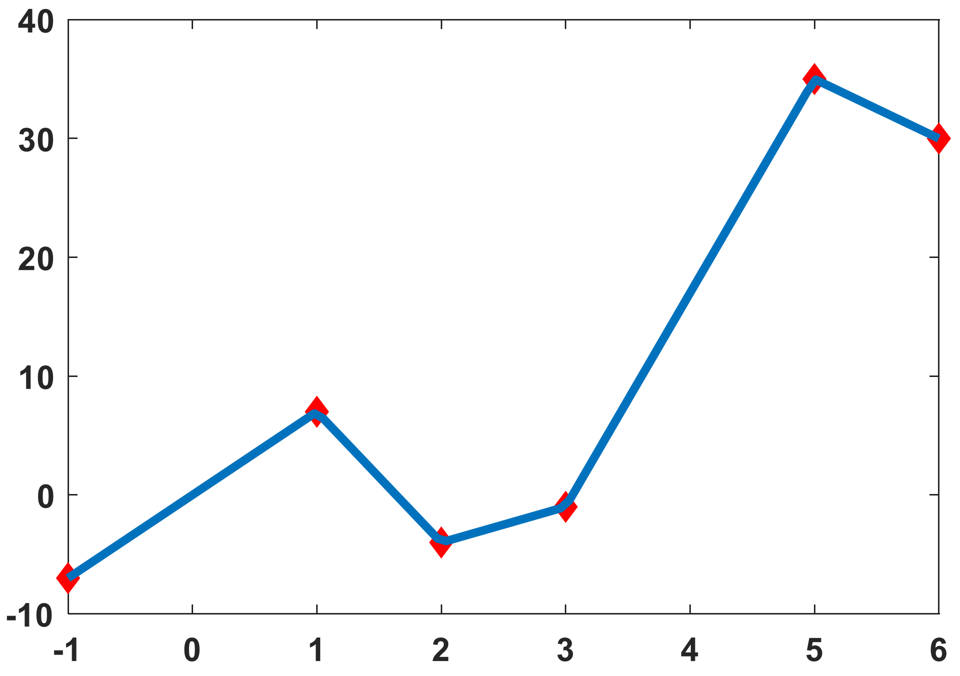
x= vettore di ascisse lungo n

y= vettore di valori noti lungo n

t=vettore di punti in cui valutare il polinomio

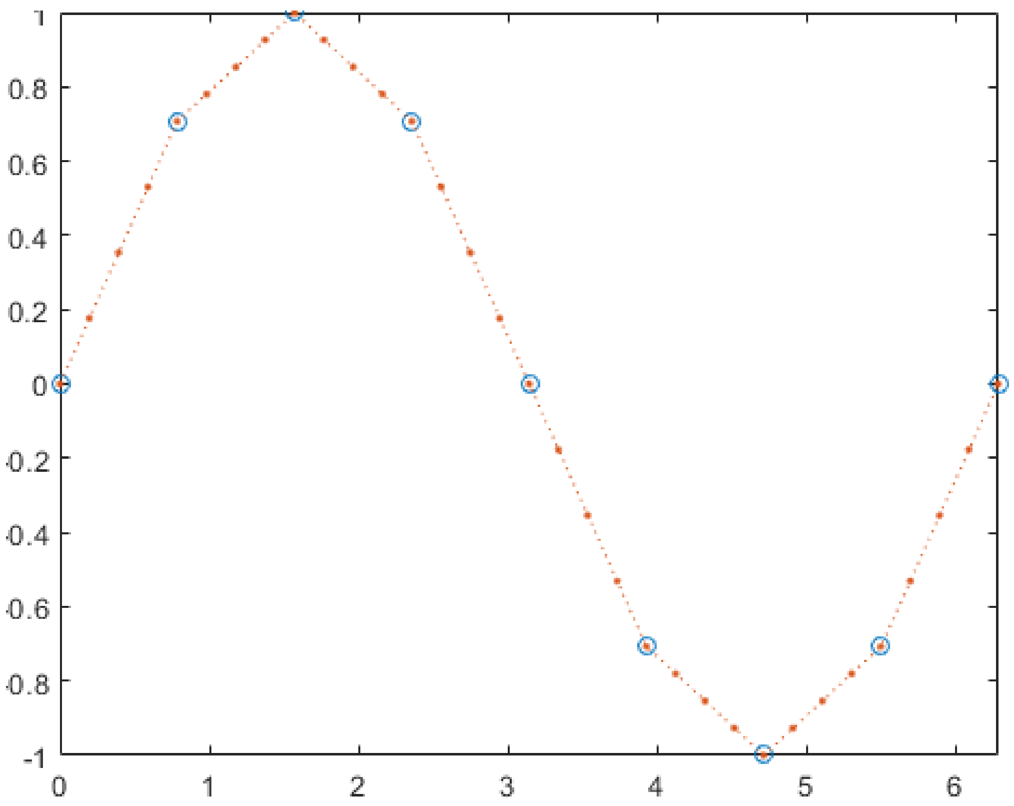
z= vettore di valori del polinomio nei punti t

```
>> x=[-1 1 2 3 5 6];
>> y=[-7 7 -4 -1 35 30];
>> t=linspace(-1,6);
>> pt=interp1(x,y,t);
>> plot(x,y,'d',t,pt)
```



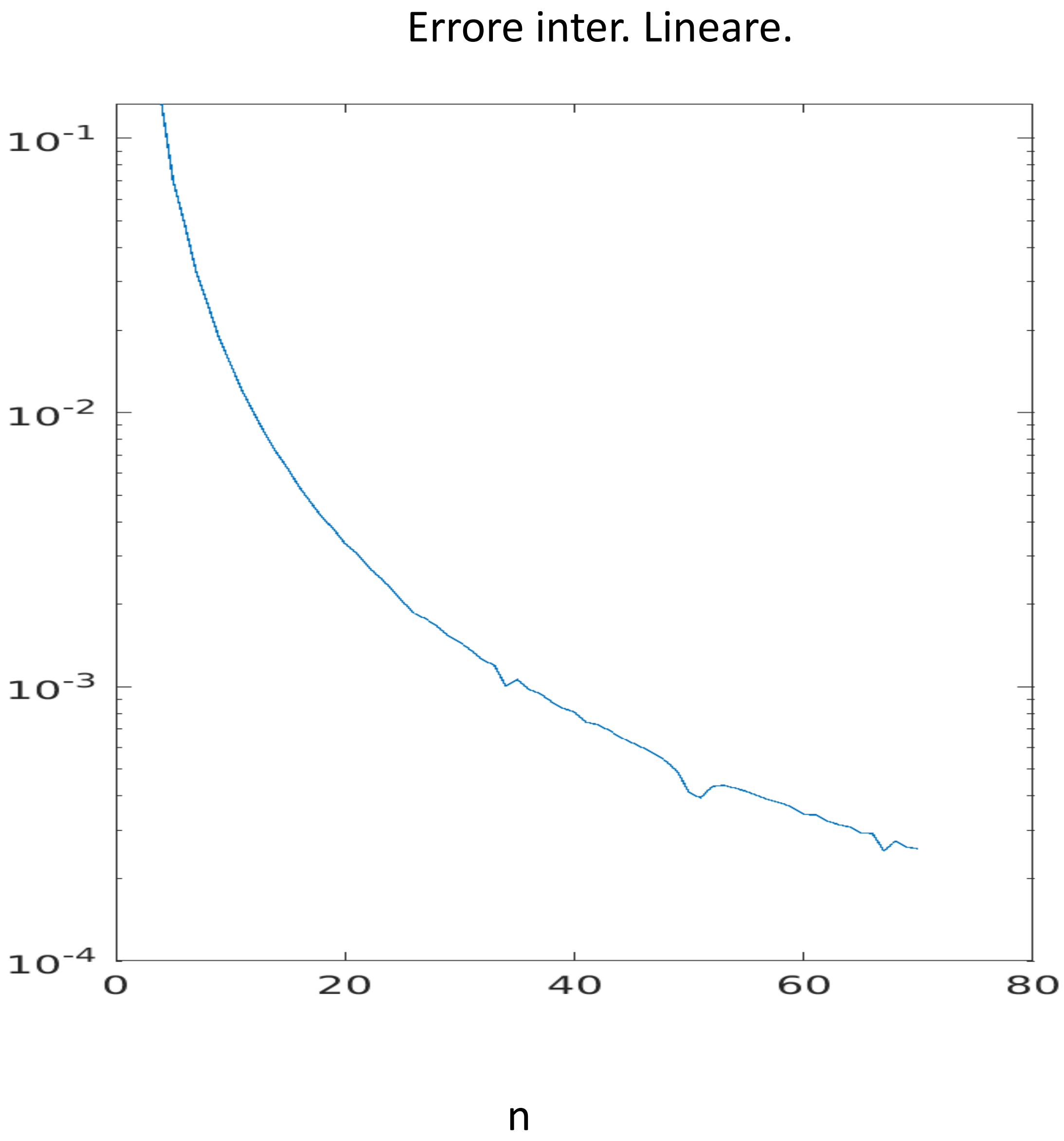
Interpolazione lineare a tratti

```
>>x = 0:pi/4:2*pi;
 >>v = sin(x);
>>xq = 0:pi/16:2*pi;
>>vq1 = interp1(x,v,xq);
>>plot(x,v,'o',xq,vq1,':.’);
>>title('Interpolazione lineare');
```



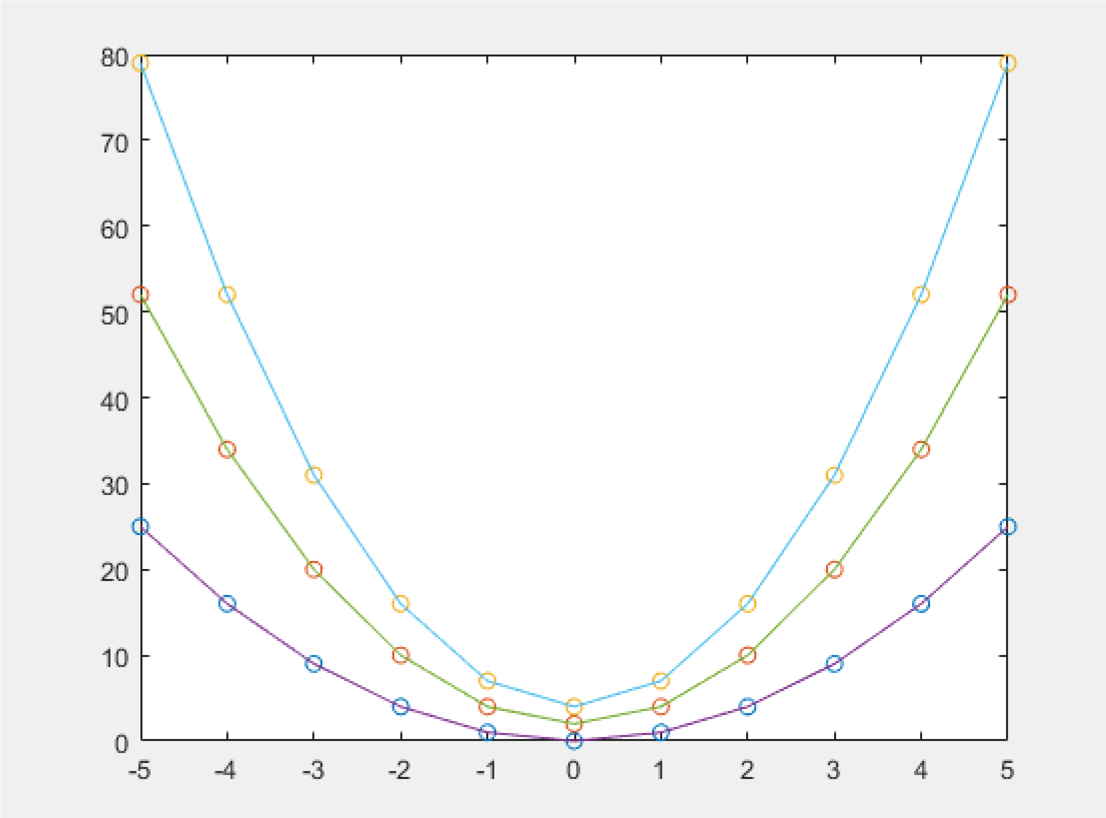
Interpolazione lineare a tratti

```
>>x = 0:pi/4:2*pi;
 >>v = sin(x);
>>xq = 0:pi/16:2*pi;
>>vq1 = interp1(x,v,xq);
>>plot(x,v,'o',xq,vq1,':.');
>>title('Interpolazione lineare');
```



Interpolazione di un set di dati

```
>> x = (-5:5)';
>> v1 = x.^2;
>>v2 = 2*x.^2 + 2;
>>v3 = 3*x.^2 + 4;
>> v = [v1 v2 v3];
>> xq = -5:0.1:5;
>> vq = interp1(x,v,xq);
plot(x,v,'o',xq,vq);
```



Funzioni Spline

Matlab

spline

Cubic spline data interpolation

collaps

Syntax

```
s = spline(x,y,xq)
```

Description

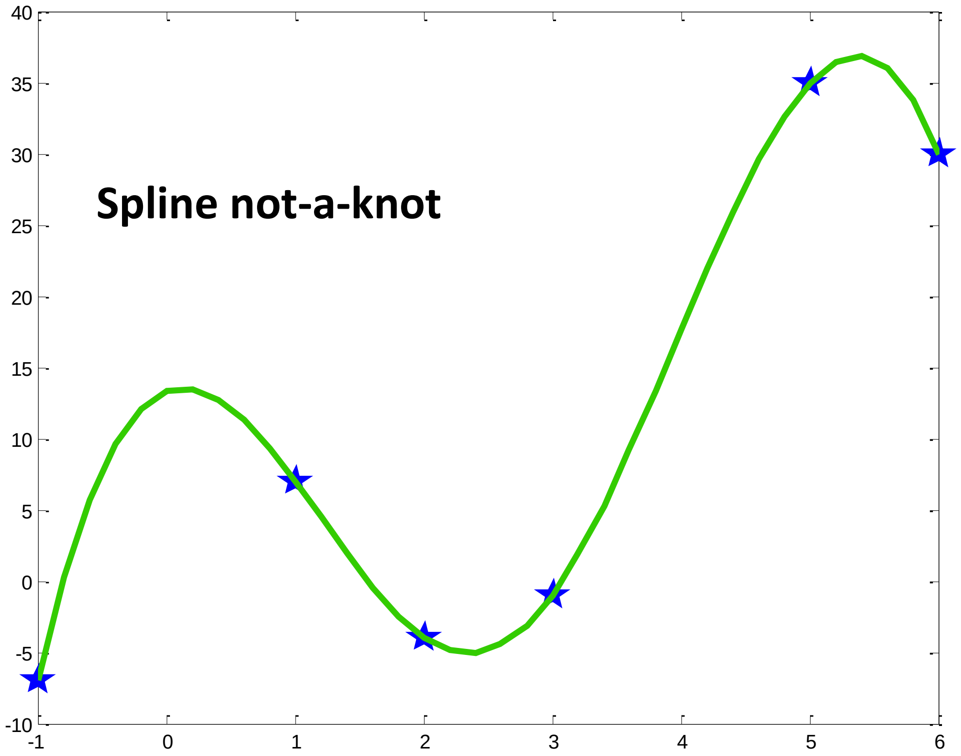
`s = spline(x,y,xq)` returns a vector of interpolated values `s` corresponding to the query points in `xq`. The values of `s` are determined by cubic spline interpolation of `x` and `y`.

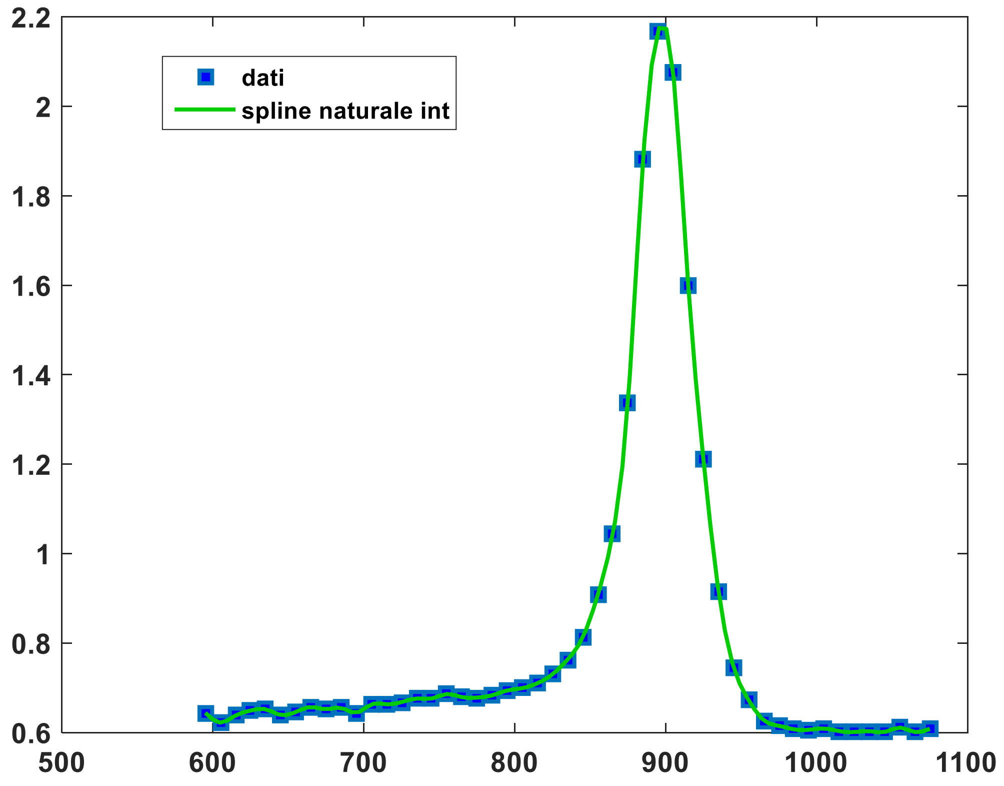
Matlab

f=spline(x,y,t)

- ✓ **x= vettore di ascisse lungo n**
- ✓ **y= vettore di valori noti lungo n**
- ✓ **t=vettore di punti in cui valutare la spline**
- ✓ **f= vettore di valori della spline cubica not-a-knot nei punti t**

```
>>x=[-1 1 2 3 5 6];
>>y=[-7 7 -4 -1 35 30];
>>t=-1:0.2:6;
>> yi=spline(x,y,t);
>> plot(x,y,'*',t,yi)
```

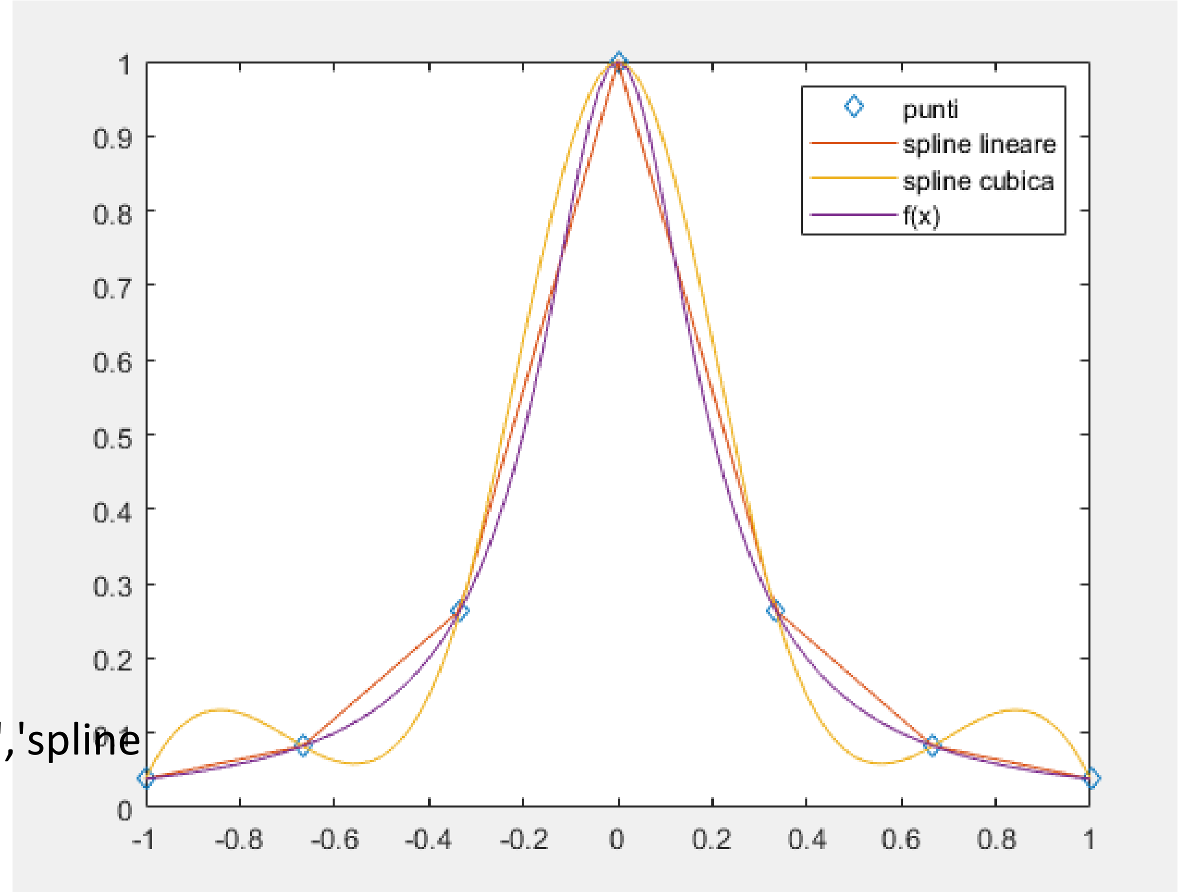




2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
500
600
700
800
900
1000
1100
dati
spline naturale int

Spline lineare e cubica

```
>> x = linspace(-1,1,7);
>> y = 1./(1+25.*x.^2);
>> xx = linspace(-1,1,101);
>> z=interp1(x,y,xx);
>> plot(x,y,'d',xx,z)
>> cs=spline(x,y,xx);
>> hold on
>> plot(xx,cs)
>> y= 1./(1+25*xx.^2);
>> hold on
>> plot(xx,y)
>> legend('punti','spline lineare','spline cubica','f(x)')
```



Spline lineare e cubica: stima dell'errore

Spline lineare

$$|f(x) - s_{1,\Delta}(x)| \leq h_{i,\Delta}^2 \frac{M_{i,\Delta}}{8}, \quad x \in [x_i, x_{i+1}]$$

dove $$M_{i,\Delta} := \max_{x\in[x_i,x_{i+1}]} |f^{(2)}(x)| \leq \max_{x\in[a,b]} |f^{(2)}(x)| := M.$$

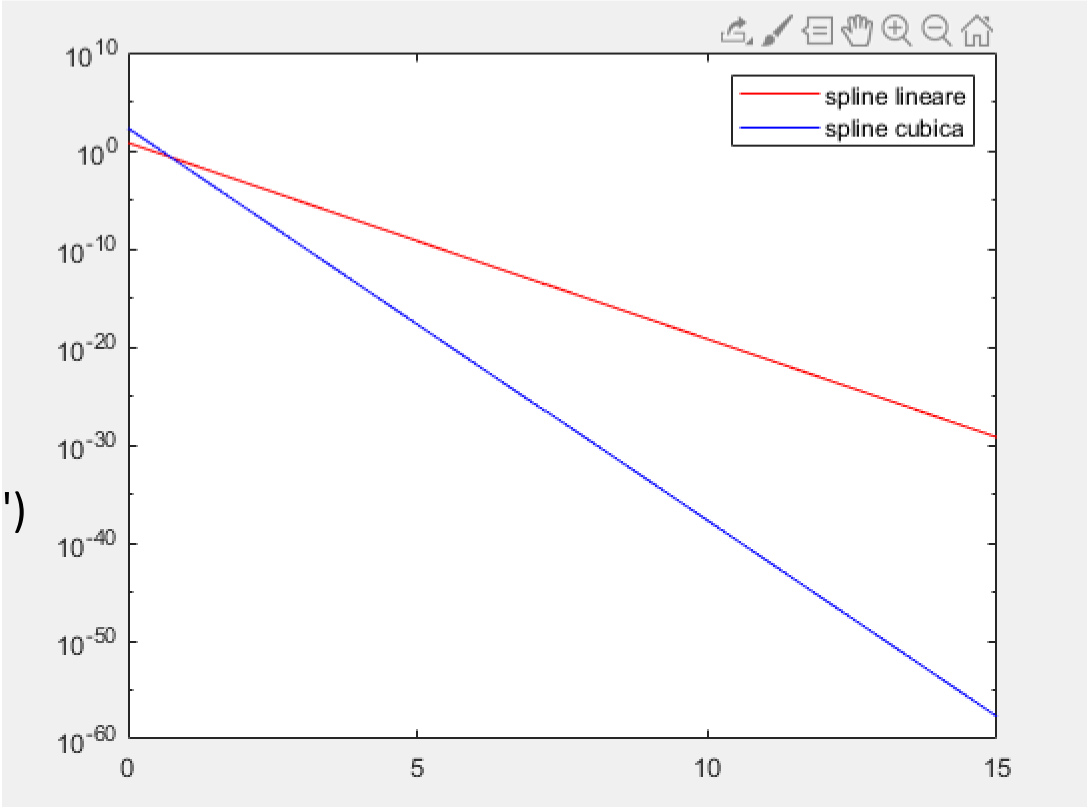
Spline cubica

$$||f - s_{3,\Delta}||_\infty \leq c_0\, h_\Delta^4 ||f^{(4)}||_\infty$$

dove $||f||_\infty = \max_{x\in[a,b]} |f(x)|$ **e** $c_0 = 5/384,$

Grafici errori

```
>> n=0:0.2:15;
>> h=10.^(-n);
>> e1=50/8*h.^2;
>> e2=(5/384)*15000*h.^4;
>> semilogy(n,e1,'-r',n,e2,'-b')
>> legend('spline lineare','spline cubica')
```



Confronto interpolazione lineare e spline cubica

```
xi=linspace(0,pi);
for n=4:50
x=linspace(0,pi,n);% xi vettore di punti su cui
interpolo
y=sin(x);
yes=sin(xi);
% x deve contenere elementi crescenti
yl=interp1(x,y,xi);%interpolazione lineare dei
dati (x,y) nei punti xi
ys=spline(x,y,xi);%interpolazione con spline
cubiche

err1(n)=norm(yl-yes,inf);
errs(n)=norm(ys-yes,inf);
end
plot(xi,yh)
hold on
fplot(@(x) sin(x))
hold on
plot (x,y,'o')
title('Confronto tra differenti interpolazioni');
axis([0 pi 0 1]);
legend('interp Hermite','funzione','punti')
```

Confronto tra diverse interpolazini

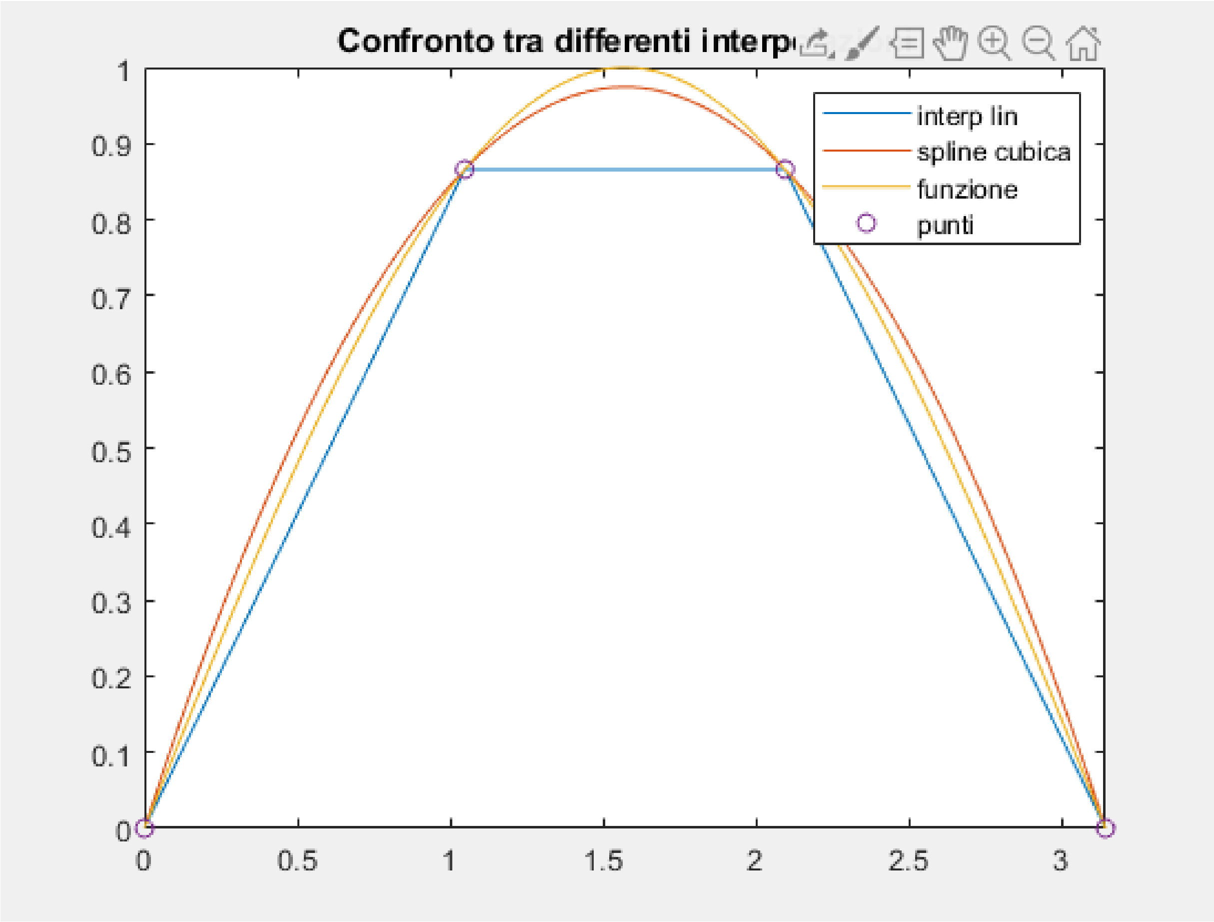
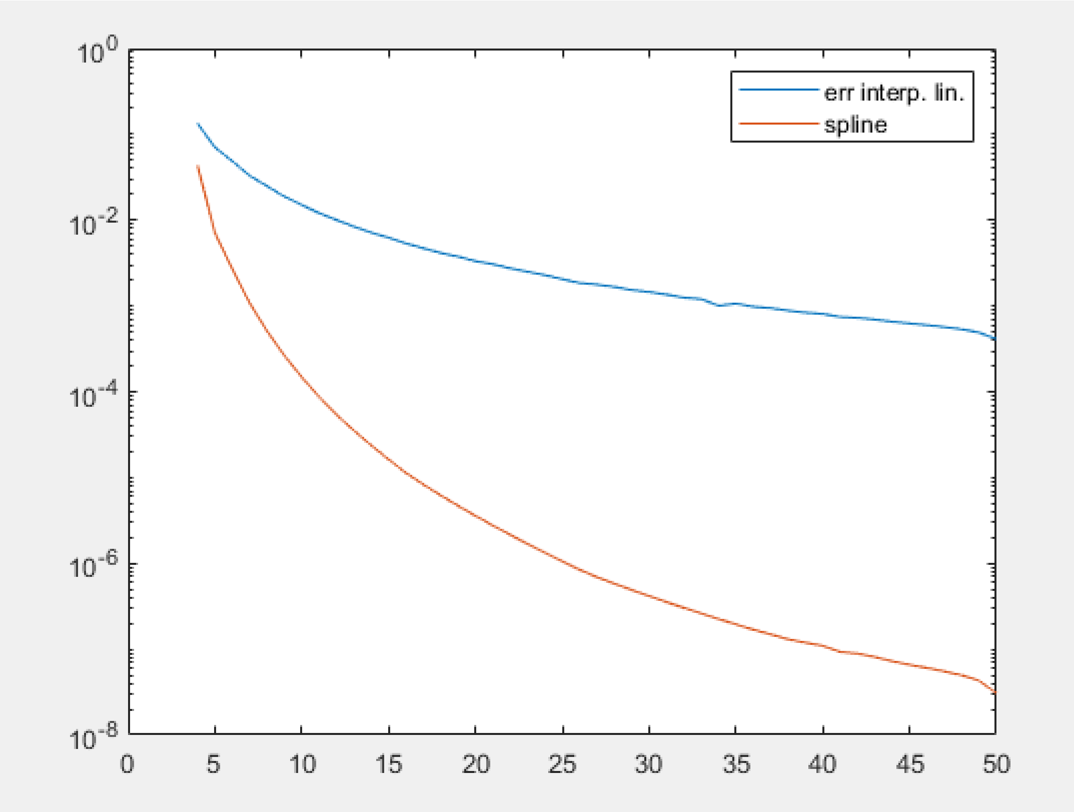


Grafico errore

Considerando 4<=n<=50



n (nodi)

Matlab

spline

Cubic spline data interpolation

collaps

Syntax

```
s = spline(x,y,xq)
pp = spline(x,y)
```

Description

`s = spline(x,y,xq)` returns a vector of interpolated values `s` corresponding to the query points in `xq`. The values of `s` are determined by cubic spline interpolation of `x` and `y`.

`pp = spline(x,y)` returns a piecewise polynomial structure for use by `ppval` and the spline utility `unmkpp`.

pp=spline(x,y)

- ✓x= vettore di ascisse lungo n
- ✓y= vettore di valori noti lungo n
- ✓pp= pp-form ,variabile struct, rappresentazione polinomiale a tratti della spline cubica not-a-knot

la pp-form contiene:

- ✓ breaks = vettore dei nodi
- ✓ coefs = matrice l×k dei coefficienti della spline
  coefficienti del j-mo polinomio = j-ma riga
- ✓ pieces=numero intervalli
- ✓ order=numero coefficienti nell'intervallino

$$s_j(x)=C_{j,4}+C_{j,3}(x-x_j)+C_{j,2}(x-x_j)^2+C_{j,1}(x-x_j)^3$$

```
>> x=1:7;y=rand(1,7);
>> pp=spline(x,y)
pp =
  form: 'pp'
  breaks: [1 2 3 4 5 6 7]     nodi
  coefs: [6x4 double]         coefficienti
  pieces: 6                   intervalli
 order: 4    grado dei polinomi nei tratti +1
 dim: 1
```

ppval

Evaluate piecewise polynomial

Syntax

```
v = ppval(pp,xq)
```

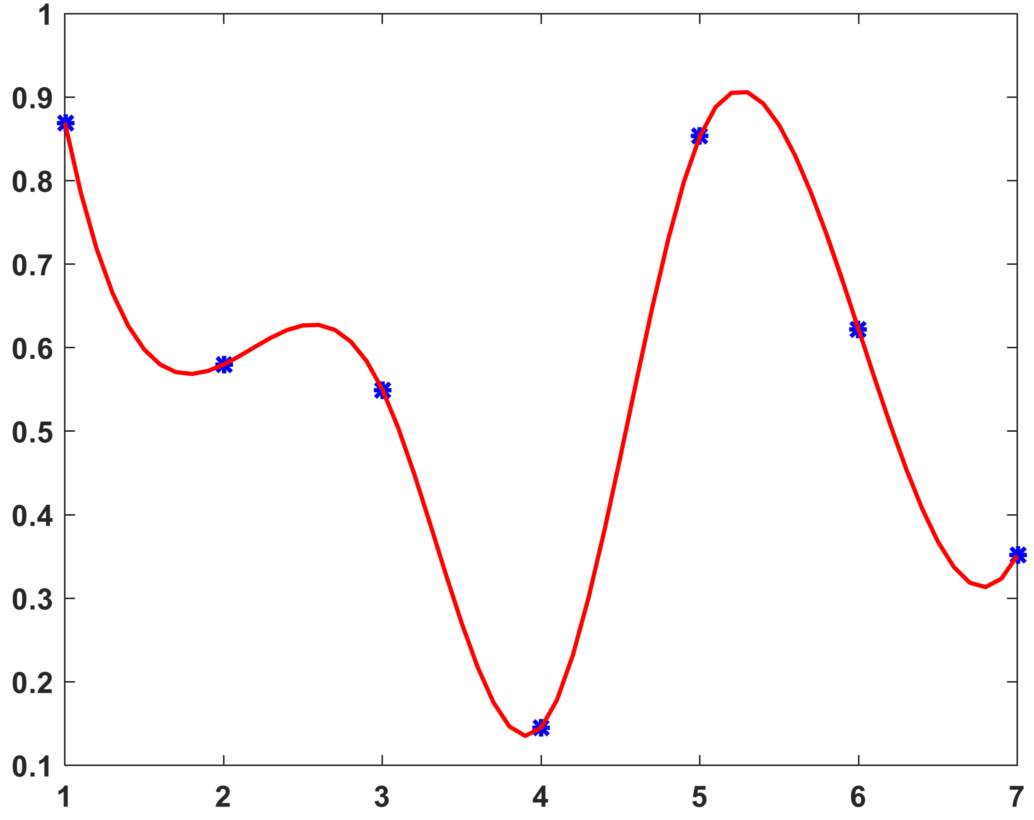
Description

`v = ppval(pp,xq)` evaluates the piecewise polynomial `pp` at the query points `xq`.

ys=ppval(pp,xx)

- ✓ **pp= pp-form della spline (not-a-knot,naturale o clamped)**
- ✓ **xx= vettore di valori in cui valutare la spline**
- ✓ **ys=vettore di valori della spline nei punti xx**

```
>>x=1:7;
>>y=rand(1,7);
>>pp=spline(x,y)
>> t=1:0.1:7;
>> ys=ppval(pp,t);
>> plot(x,y,'*',t,ys)
```



Perché usare la pp-form?

Si può manipolare per calcolare derivate, integrali..

pder=fnder(pp,order)

- ✓ **pp= pp-form**
- ✓ **order= ordine della derivata**
- ✓ **pder= pp-form della derivata di ordine order**

fnder

R2

Differentiate function

collapse all

Syntax

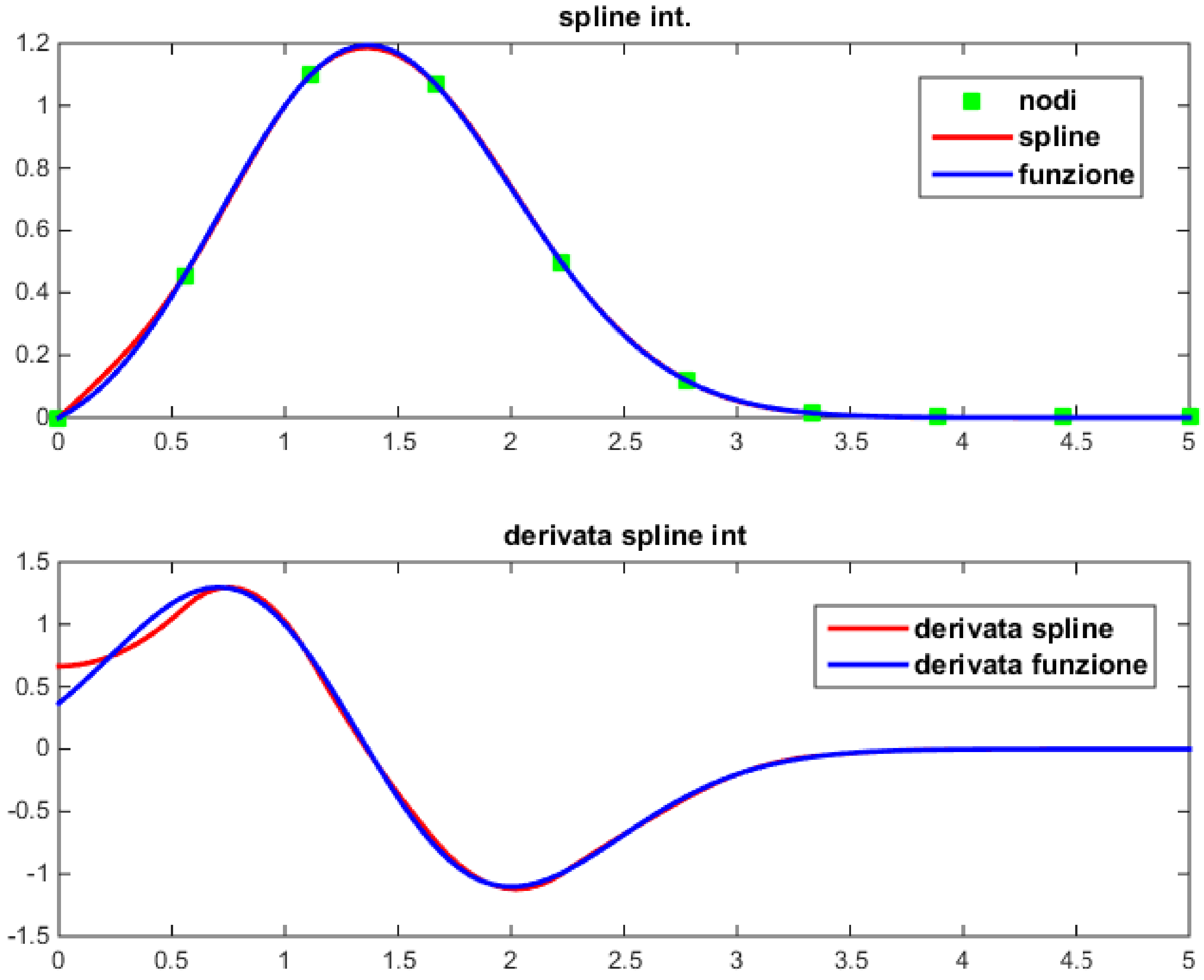
```
fprime = fnder(f,dorder)
fnder(f)
```

Description

e

`fprime = fnder(f,dorder)` returns the `dorder`-th derivative of the function in `f`. The default value of `dorder` is 1. For negative `dorder`, the particular |`dorder`|-th indefinite integral is returned that vanishes |`dorder`|-fold at the left endpoint of the basic interval.

```
%spline e sua derivata
x=linspace(0,5,10);
xx=linspace(0,5);
%valutazione funzione  test
y=x.*exp(-(x-1).^2);
yy=xx.*exp(-(xx-1).^2);
%valutazione derivata funzione test
yd=exp(-(xx-1).^2).*(1-2*xx.*(xx-1));
% pp-form della spline interpolante e della sua derivata
pp=spline(x,y);
sder=fnder(pp,1);
%valutazione spline e sua derivata
sp=ppval(pp,xx);
sd=ppval(sder,xx);
plot(x,y,'*',xx,sp,xx,yy,‘:')
title('spline int.')
plot(xx,sd,xx,yd,':')
title('derivata spline int')
```


spline int.
nodi
spline
funzione
derivata spline int
derivata spline
derivata funzione

csape

R20

Cubic spline interpolation with end conditions

collapse all in

Syntax

```
pp = csape(x,y)
pp = csape(x,y,conds)
pp = csape(x,[e1,y,e2],conds)
pp = csape({x1,...,xn}, __ )
```

Description

`pp = csape(x,y)` returns the cubic spline interpolation to the given data `(x,y)` in ppform form. The function applies Lagrange end conditions to each end of the data, and matches the spline endslopes to the slope of the cubic polynomial that fits the last four data points at each end. Data values at the same site are averaged.

pp=csape(x,y,’second’)

- ✓ **x= vettore di ascisse lungo n**
- ✓ **y= vettore di valori noti lungo n**
- ✓ **pp= pp-form della spline cubica naturale**

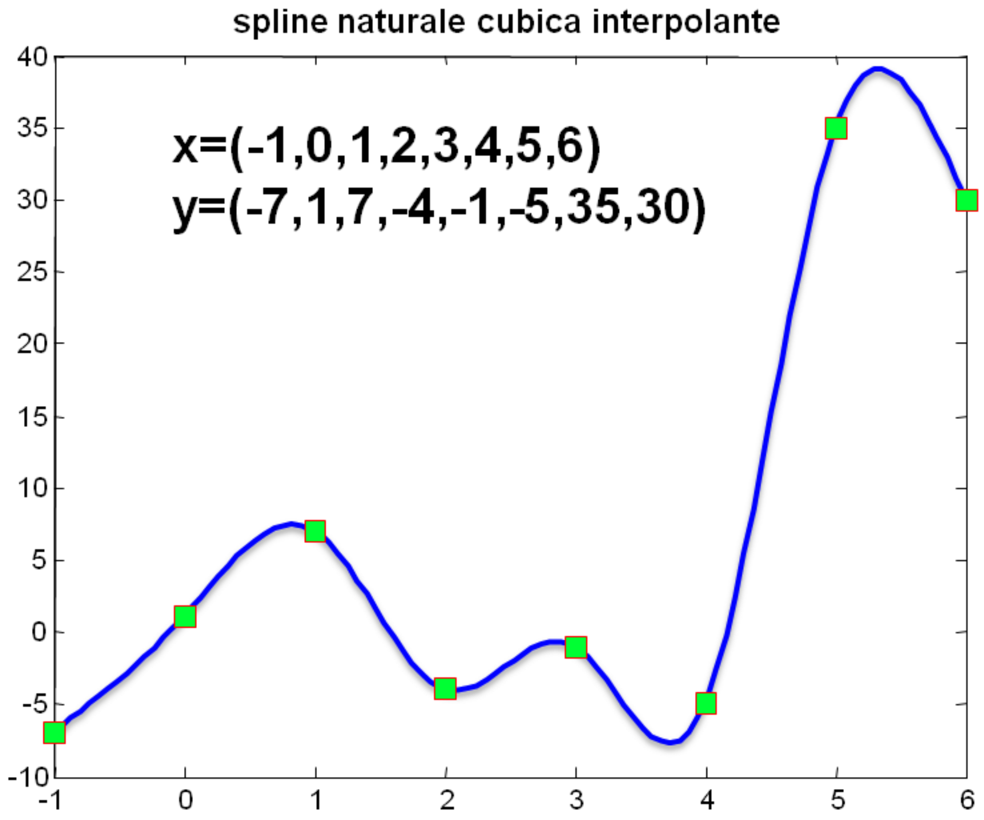
pp=csape(x,[r y s],’clamped’)

Se si conosce o si può stimare la pendenza della curva

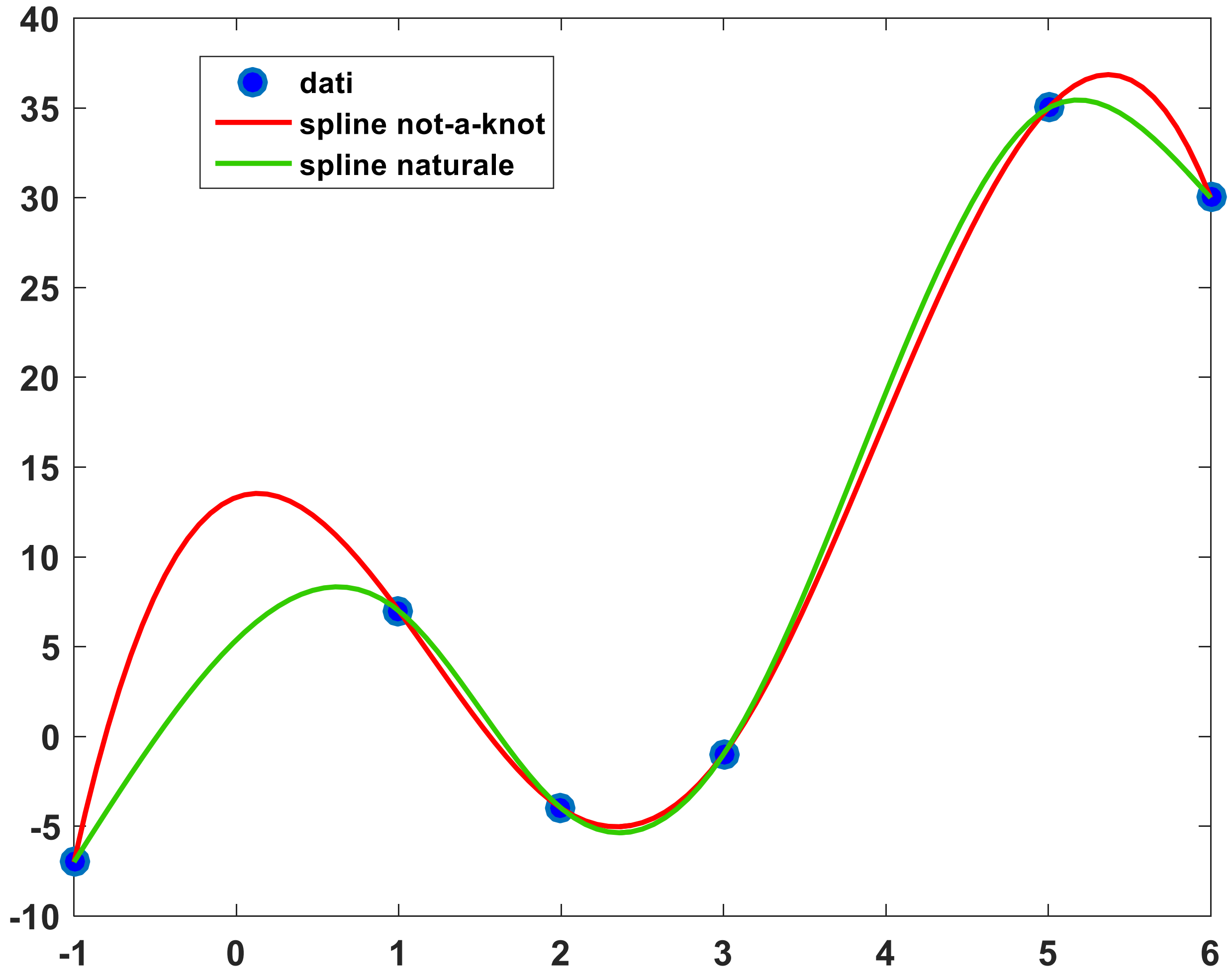
- ✓ **x= vettore di ascisse lungo n**
- ✓ **y= vettore di valori noti lungo n**
- ✓ **r= valore della derivata in x_1**
- ✓ **s= valore della derivata in x_n**
- ✓ **pp= pp-form della clamped spline**

la spline cubica naturale

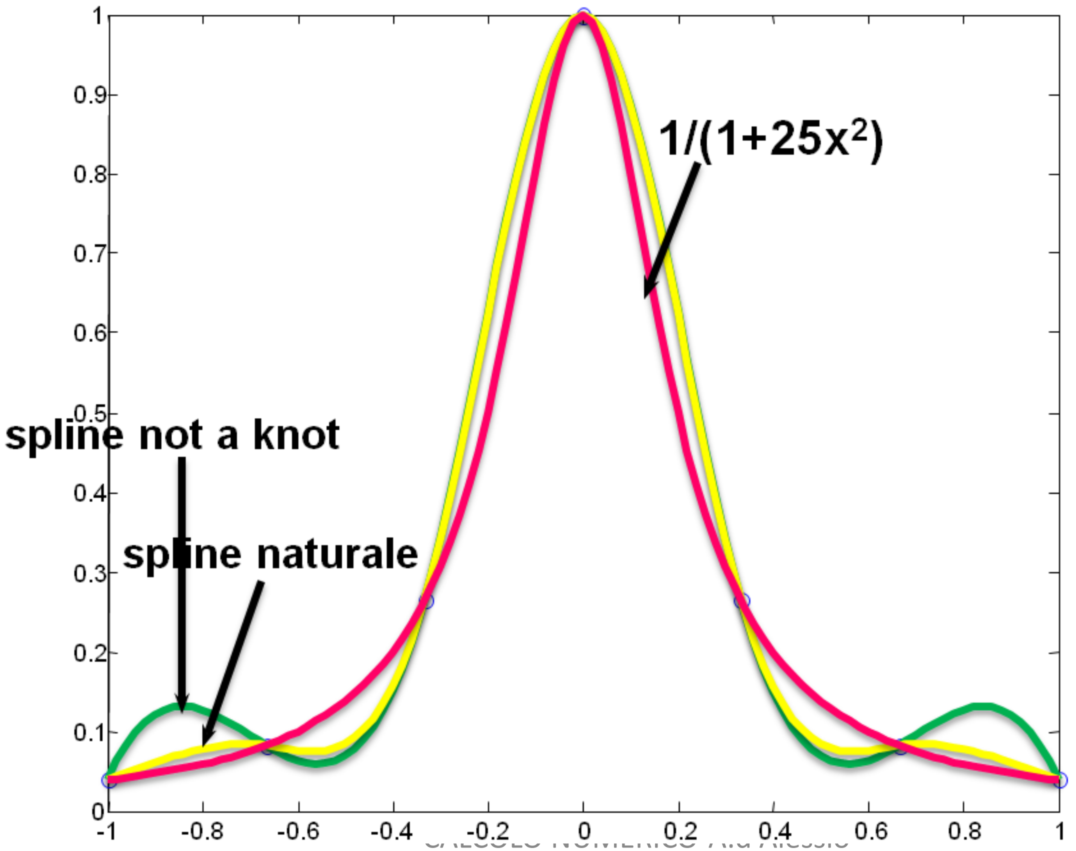
```
xx = linspace(-1,6,101);
pp=csape(x,y,'second’);
yy=ppval(pp,xx);
plot(xx,yy);
```



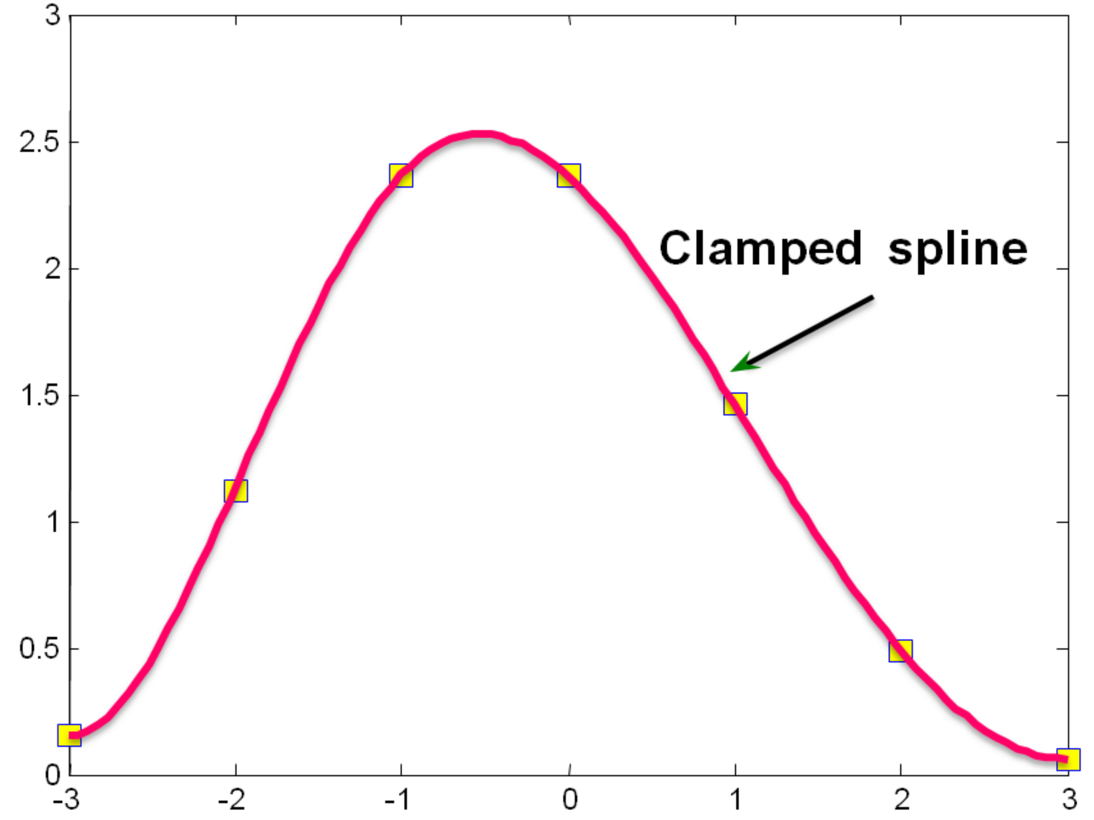
```
x = [-1 1 2 3 5 6];y = [-7 7 -4 -1 35 30];
xx = linspace(-1,6,101);
cs = spline(x,y,xx);
pp=csape(x,y,'second');yy=ppval(pp,xx);
plot(x,y,'o',xx,cs,'r',xx,yy,'g');
```



```
x = linspace(-1,1,7);y = 1./(1+25.*x.^2);
xx = linspace(-1,1,101);
cs = spline(x,y,xx);
%funzione di Runge
yf = 1./(1+25.*xx.^2);
pp=csape(x,y,'second');yy=ppval(pp,xx);
plot(x,y,'o',xx,cs,'r',xx,yy,'g',xx,yf,'b');
```



```
>> x=-3:3;y=[0.15 1.12 2.36 2.36 1.46 .49 .06];
>> pp=csape(x,[0 y 0],'clamped');
>> xx=linspace(-3,3);
>> yy=ppval(pp,xx);
>> plot(x,y,'s',xx,yy)
```

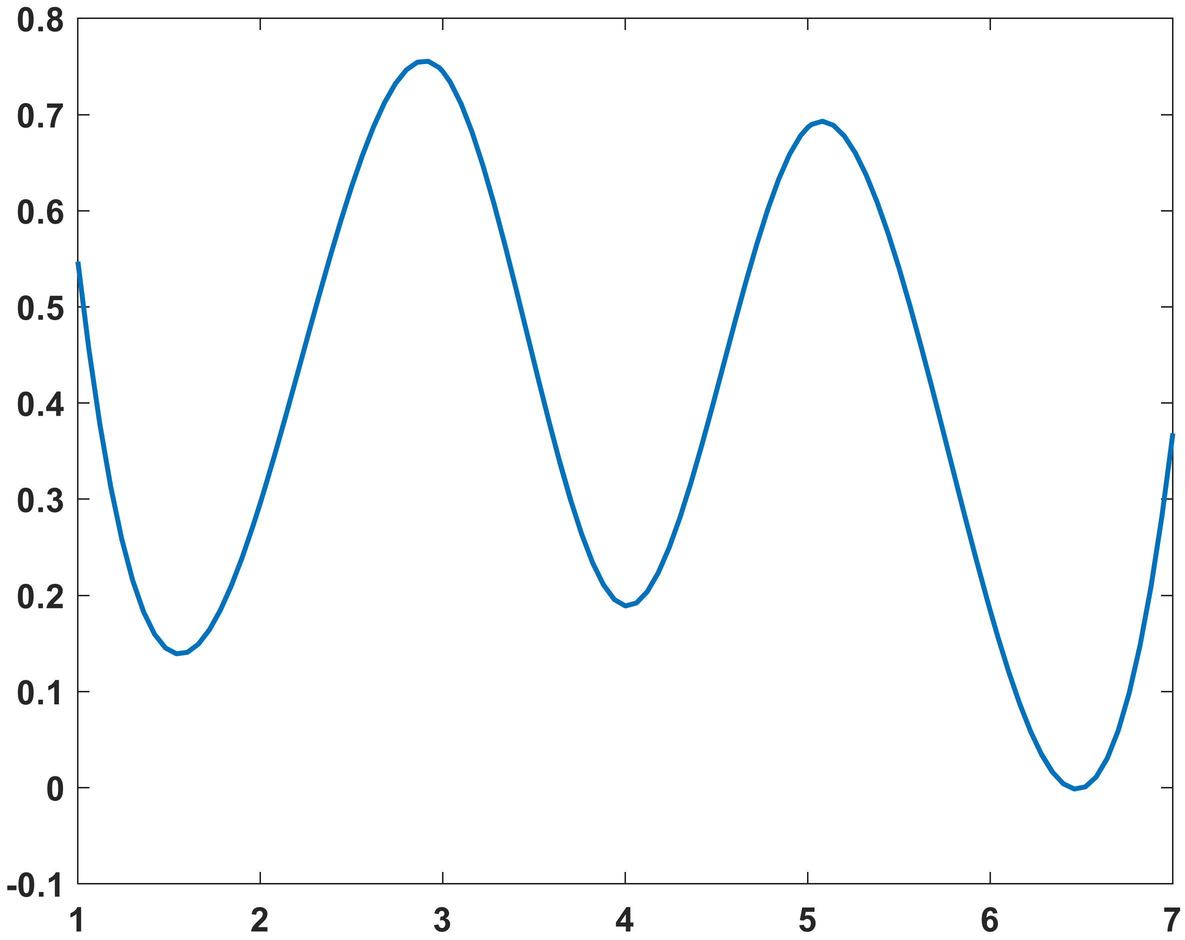


fnplt(s)

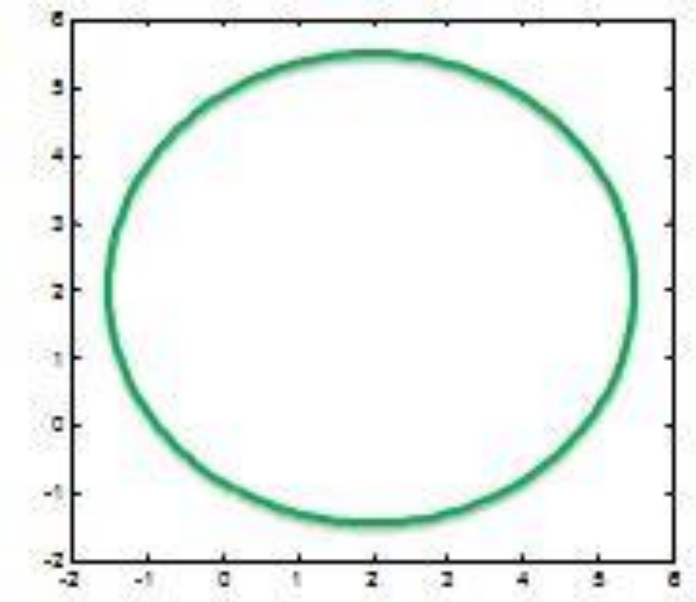
s= pp-form della spline (not-a-knot,naturale o clamped)

Fa il grafico della funzione definita come struttura

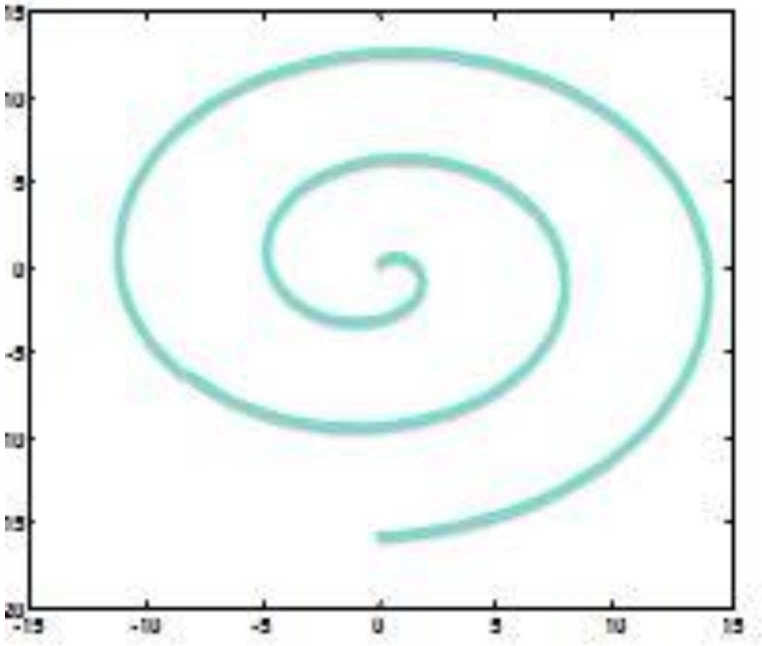
```
>>x=1:7;
>>y=rand(1,7);
>>pp=spline(x,y)
>> fnplt(pp)
```



```
%cerchio di centro(2,2),raggio 3.5
>> t=linspace(0,2*pi,200);
>> x=2+3.5*cos(t);
>> y=2+3.5*sin(t);
>> plot(x,y),axis square
```



```
%spirale
t=linspace(0,5*pi,200);
x =t.*sin(t);
y =t.*cos(t);
plot(x,y)
```



cscvn

"Natural" or periodic interpolating cubic spline curve

coll

Syntax

```
curve = cscvn(points)
```

Description

`curve = cscvn(points)` returns a parametric variational, or *natural*, cubic spline curve (in ppform) passing through the given sequence points (:*j*), *j* = 1:end.

Impossibilità di rappresentare tali curve con un'unica funzione interpolante

$\Downarrow$

spline interpolante parametrica

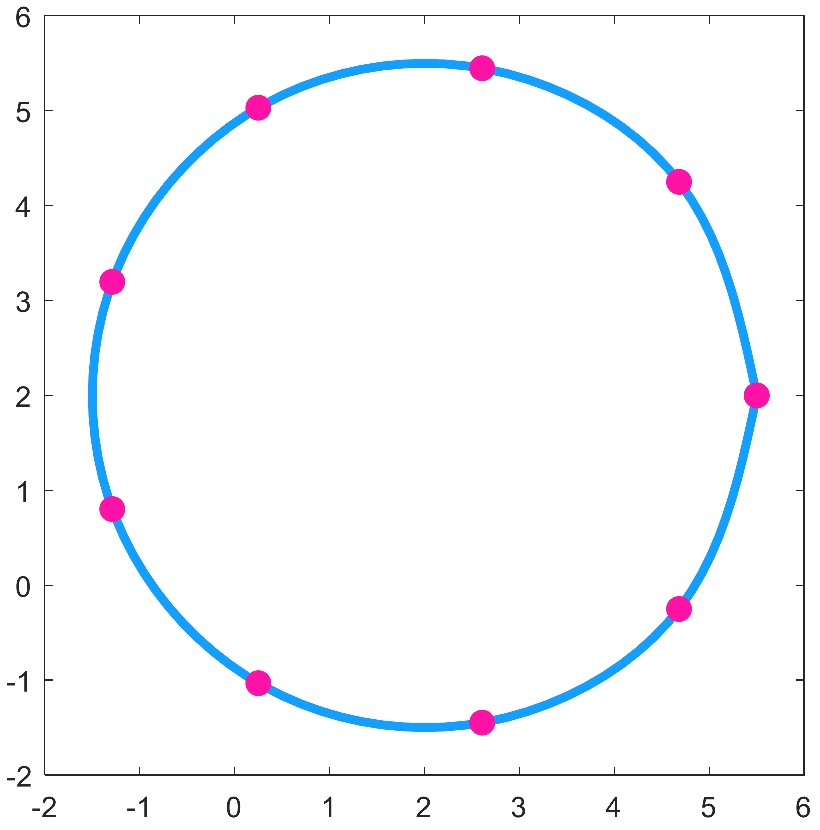
curva = cscvn(points)

- ✓ **points** matrice 2xn(o 3xn) contenente i punti di interpolazione (x_i, y_i) (o (x_i, y_i, z_i)), $i=1,..,n$
- ✓ **curva** pp-form della spline parametrica interpolante

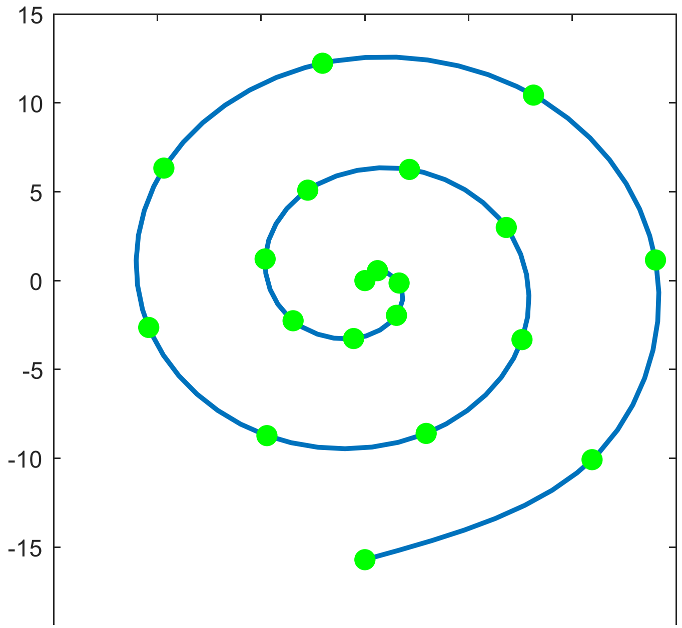
fnplt(curva)

fa il grafico della spline

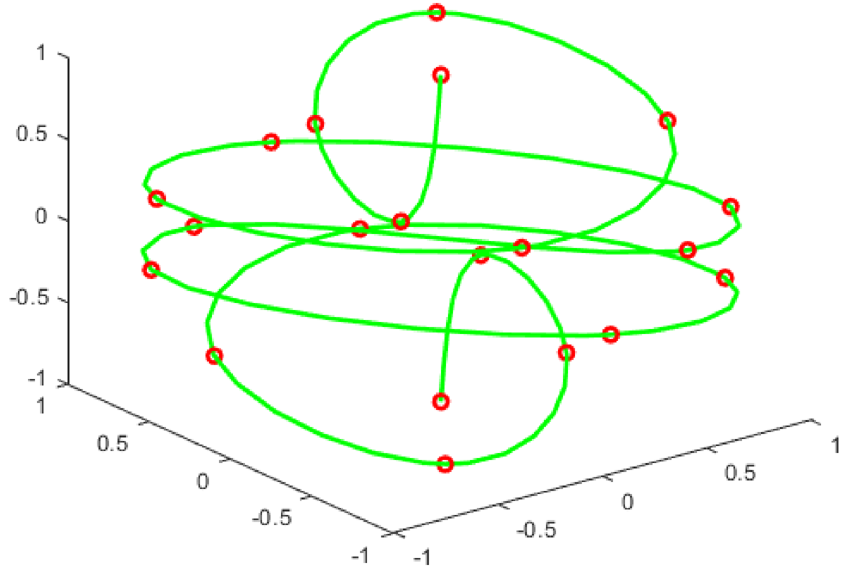
```
%interpolazione cerchio
 t=linspace(0,2*pi,10);
x=2+3.5*cos(t);y=2+3.5*sin(t);
points=[x;y];
pp=cscvn(points);
fnplt(pp);
hold on
plot(points(1,:),points(2,:),'o')
axis square
```



```
%interpolazione spirale
t=linspace(0,5*pi,20);
x =t.*sin(t);y =t.*cos(t);
points=[x;y];
 fnplt(cscvn(points)); hold on,
plot(points(1,:),points(2,:),'o')
 axis square
```



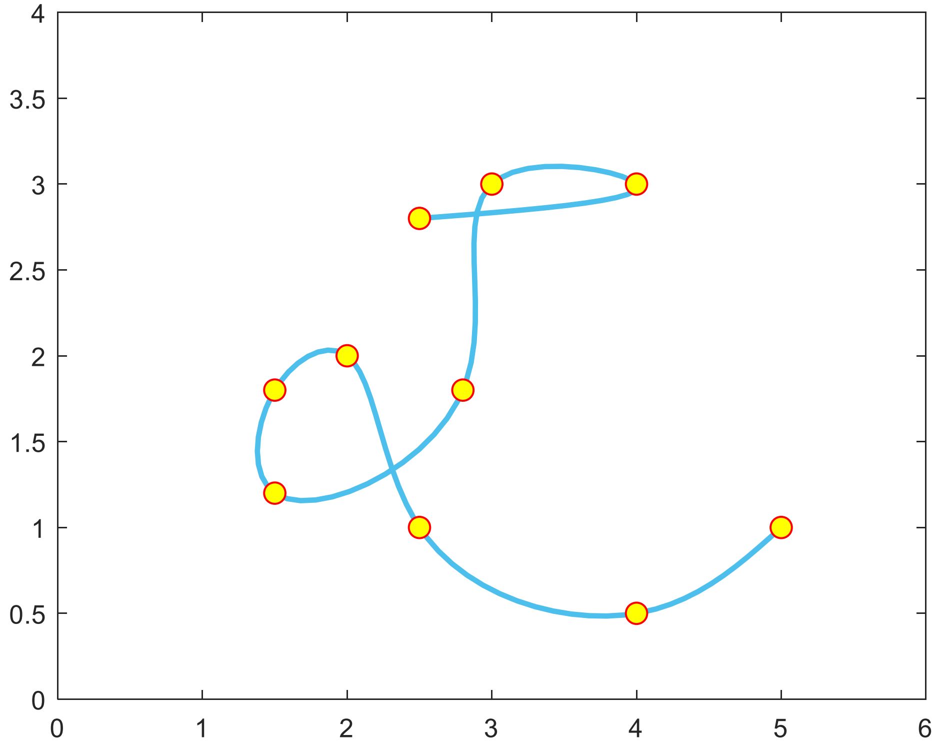

```
%una linea nello spazio
t=linspace(0,2*pi,20)
x=sin(4*t);y=sin(5*t);
z = linspace(-1,1,length(t));
xyz = [x;y;z];
plot3(xyz(1,:),xyz(2,:),xyz(3,:),'ro');
hold on
fnplt(cscvn(xyz),'g’)
```



Se bisogna rappresentare curve così complesse non riconducibili a funzioni matematiche note?

si memorizzano i punti **(x_i ,y_i), i=1,..,n** nell'ordine in cui si susseguono nella curva e si interpola

```
%la lettera L
x=[5 4 2.5 2 1.5 1.5 2.8 3 4 2.5];
y=[1 0.5 1 2 1.8 1.2 1.8 3 3 2.8];
points=[x;y];
fnplt(cscvn(points)); hold on
plot(points(1,:),points(2,:),'o')
axis([0 6 0 4])
```



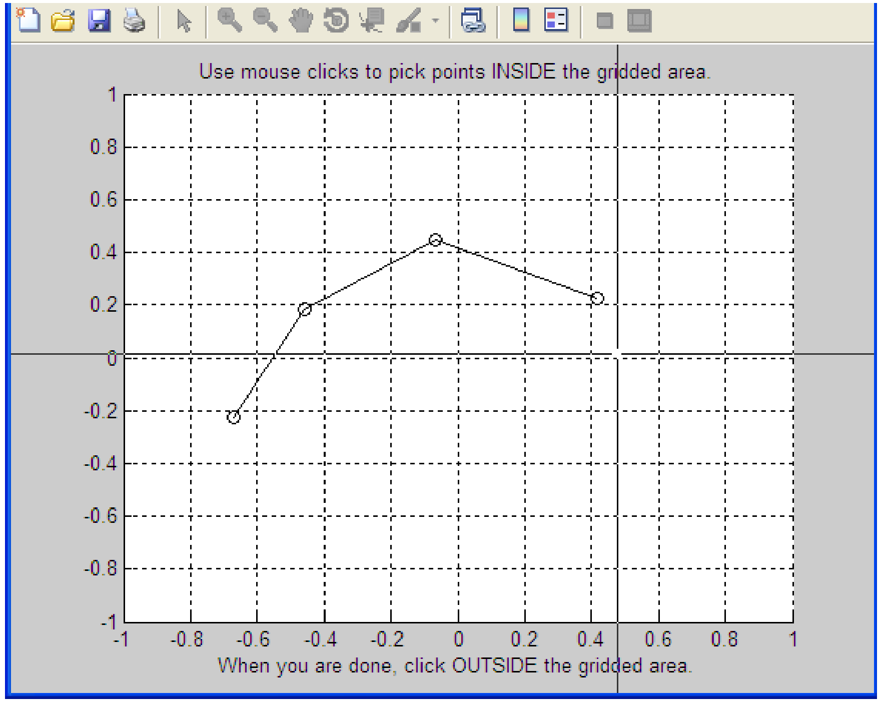
input interattivo di nodi con il mouse e costruzione spline

[xy,s] = getcurve

- ✓xy matrice 2xn di coordinate di n punti selezionati con il mouse
- ✓s struttura contenente la spline costruita tramite xy

>>getcurve

Per terminare clicca fuori dalla griglia



Curve di Bezier

bernsteinMatrix

R20

Bernstein matrix

collapse all i

Syntax

```
B = bernsteinMatrix(n,t)
```

Description

ex

`B = bernsteinMatrix(n,t)`, where `t` is a vector, returns the `length(t)`-by-`(n+1)` Bernstein matrix `B`, such that `B(i,k+1)= nchoosek(n,k)*t(i)^k*(1-t(i))^(n-k)`. Here, the index `i` runs from 1 to `length(t)`, and the index `k` runs from `0` to `n`.

The Bernstein matrix is also called the Bezier matrix.

Use Bernstein matrices to construct Bezier curves:

```
bezierCurve = bernsteinMatrix(n, t)*P
```

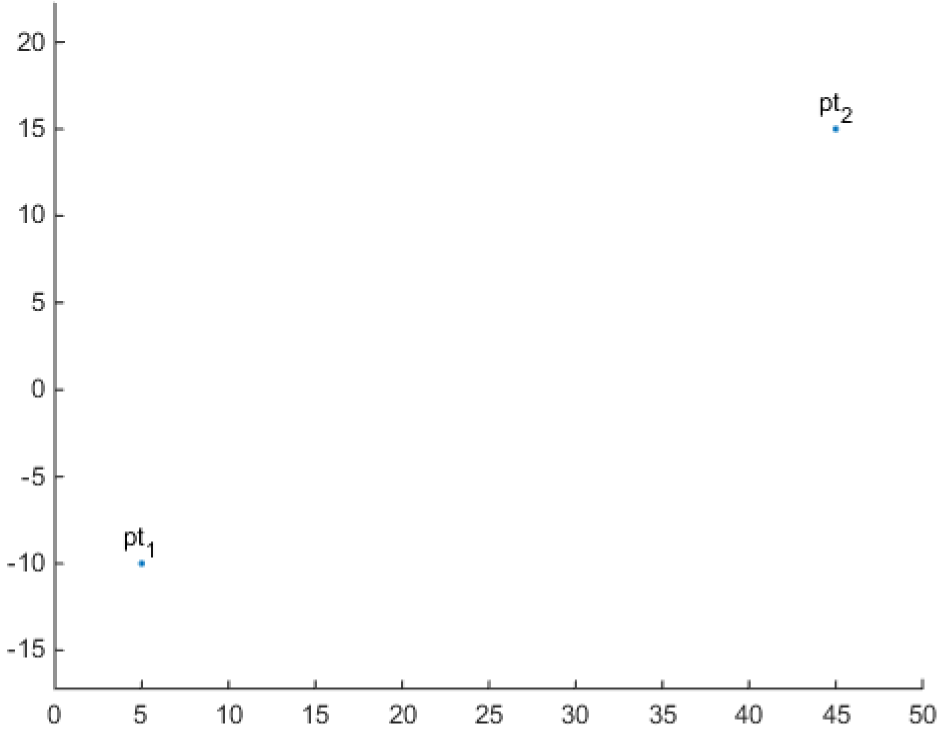
Here, the `n+1` rows of the matrix `P` specify the control points of the Bezier curve. For example, to construct the second-order 3-D Bezier curve, specify the control points as:

```
P = [p0x, p0y, p0z;  p1x, p1y, p1z;  p2x, p2y, p2z]
```

Curve di Bezier

```
pt1 = [ 5;-10];
pt2 = [45; 15];

placelabel(pt1,'pt_1');
 placelabel(pt2,'pt_2');
 xlim([0 50])
axis equal
```



Curve di Bezier 2D

Punti di controllo

```
pt1 = [ 5 -10];
pt2 = [45 15];
```

Matrice costruita a partire dai punti di controllo

```
P = [5 -10; 45 15];
```

Calcolo della matrice di Bernstein del 1 ordine

```
>>syms t
>>B = bernsteinMatrix(1, t)

B =

[ 1 - t, t]
```

Costruzione curva di Bezier

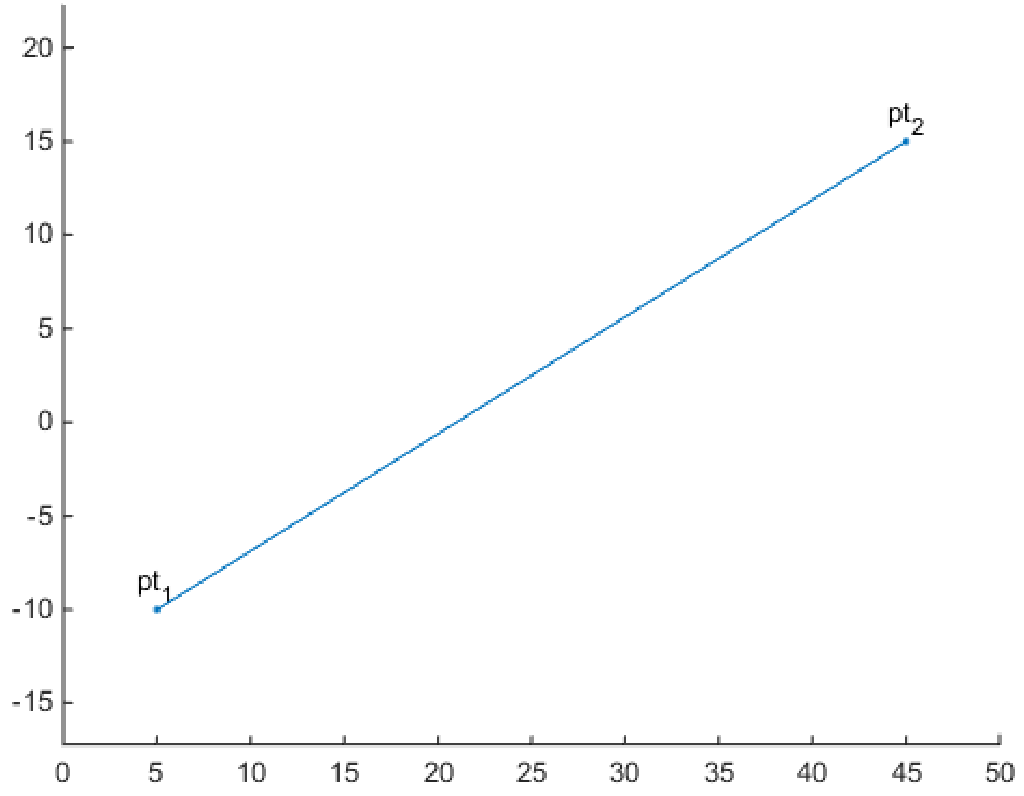
```
>> bezierCurve = simplify(B*P)

bezierCurve =

[ 40*t + 5, 25*t - 10]
```

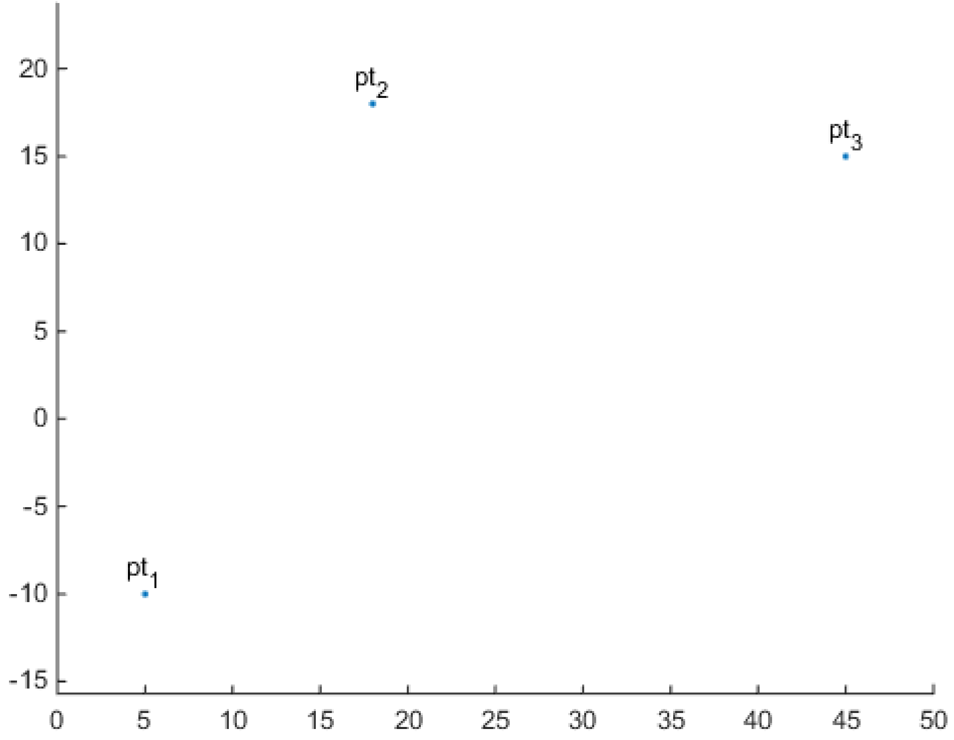
Curve di Bezier 2D

```
fplot(bezierCurve(1), bezierCurve(2), [0, 1])
hold on
 scatter(P(:,1), P(:,2),'filled’)
```



Curve di Bezier 2D

```
pt1 = [ 5;-10];
pt2 = [18; 18];
 pt3 = [45; 15];
placelabel(pt1,'pt_1');
placelabel(pt2,'pt_2');
placelabel(pt3,'pt_3');
xlim([0 50])
axis equal
```



Curve di Bezier 2D

Punti di controllo

```
pt1 = [ 5 -10];
pt2 = [18 18];
pt3 = [45 15];
```

Matrice costruita a partire dai punti di controllo

```
P = [5 -10; 18 18; 45 45];
```

Calcolo della matrice di Bernstein del 2 ordine

```
syms t
B = bernsteinMatrix(2, t)

B =

[ (t - 1)^2, -2*t*(t - 1), t^2]
```

Costruzione curva di Bezier

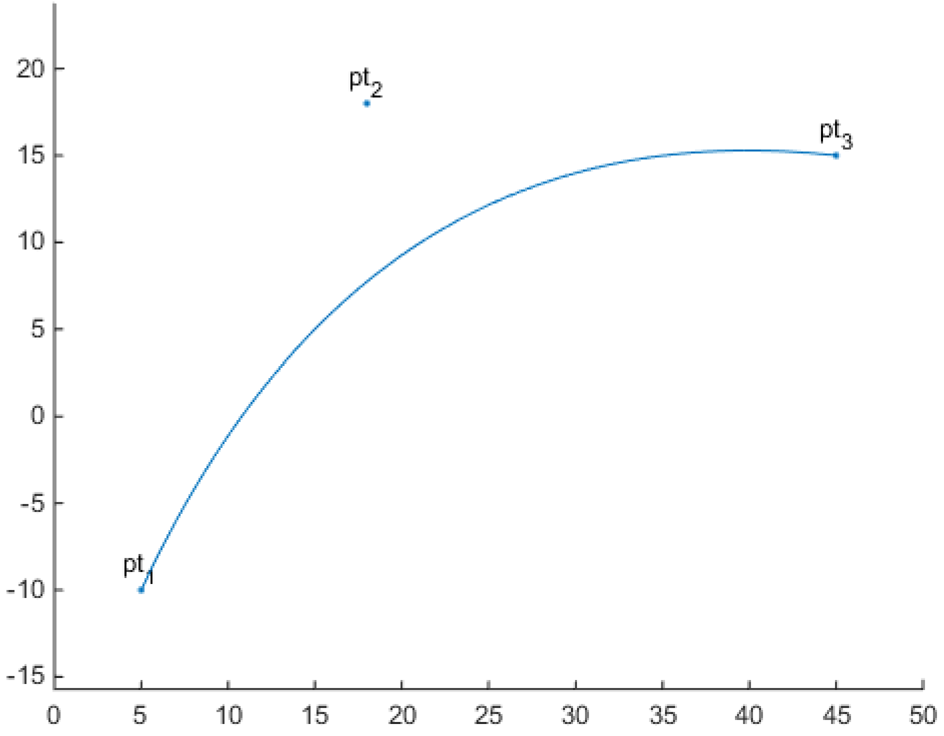
```
>> bezierCurve = simplify(B*P)

bezierCurve =

[ 14*t^2 + 26*t + 5, - t^2 + 56*t - 10]
```

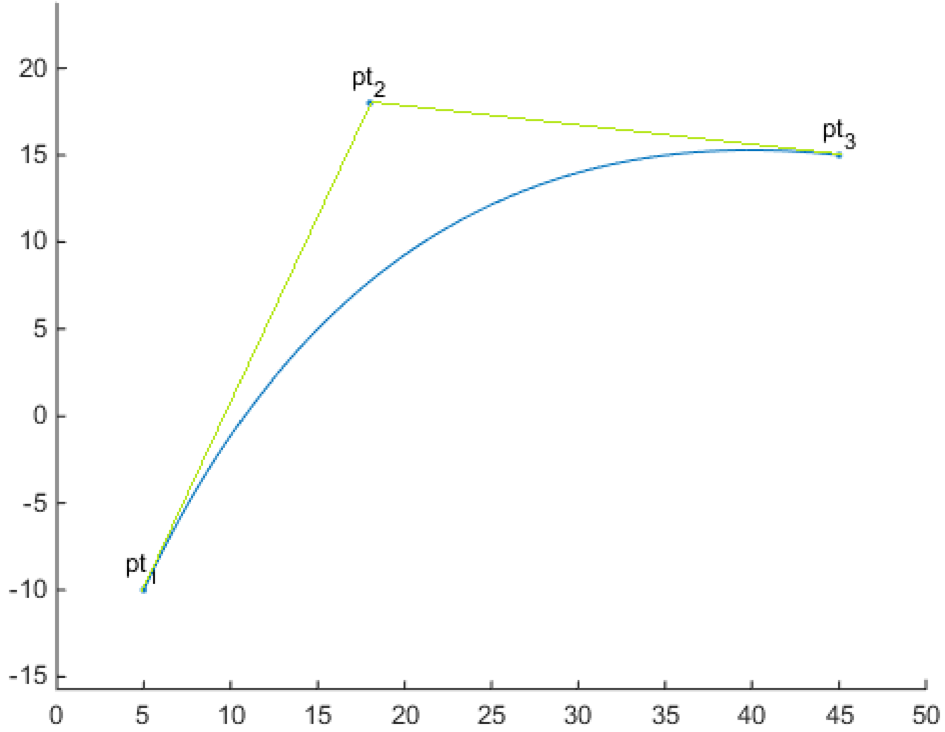
Curve di Bezier 2D

```
fplot(bezierCurve(1), bezierCurve(2), [0, 1])
hold on
scatter(P(:,1), P(:,2),'filled’)
```



Curve di Bezier 2D

```
fplot(bezierCurve(1), bezierCurve(2), [0, 1])
hold on
scatter(P(:,1), P(:,2),'filled’)
```



Curve di Bezier 2D

Punti di controllo

p0 = [0 1], p1 = [4 3], p2 = [6 2], p3 = [3 0], p4 = [2 4]

Matrice costruita a partire dai punti di controllo

```
P = [0 1; 4 3; 6 2; 3 0; 2 4];
```

Calcolo della matrice di Bernstein del quarto ordine

```
>>syms t
>>B = bernsteinMatrix(4, t)
```

```
B = [ (t - 1)^4, -4*t*(t - 1)^3, 6*t^2*(t - 1)^2, -4*t^3*(t - 1), t^4]
```

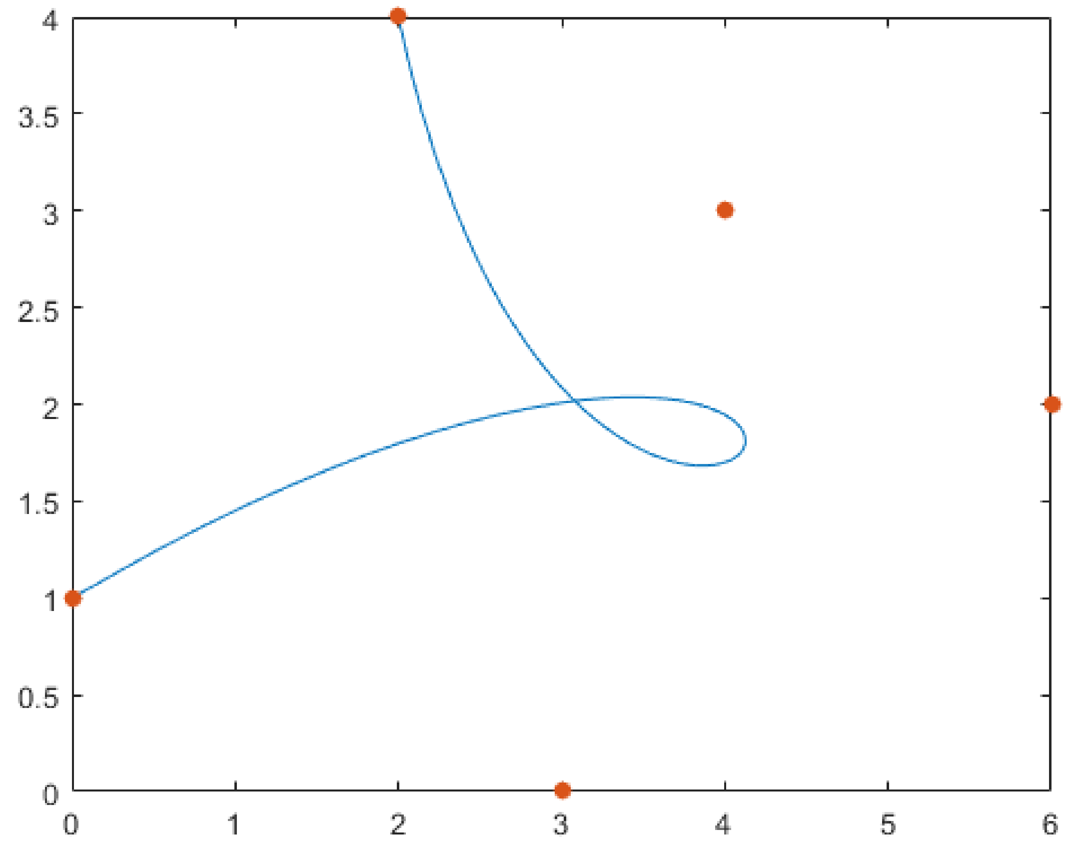
Costruzione curva di Bezier

```
bezierCurve = simplify(B*P)
```

```
bezierCurve = [ -2*t*(- 5*t^3 + 6*t^2 + 6*t - 8), 5*t^4 + 8*t^3 - 18*t^2 + 8*t + 1]
```

Curve di Bezier 2D

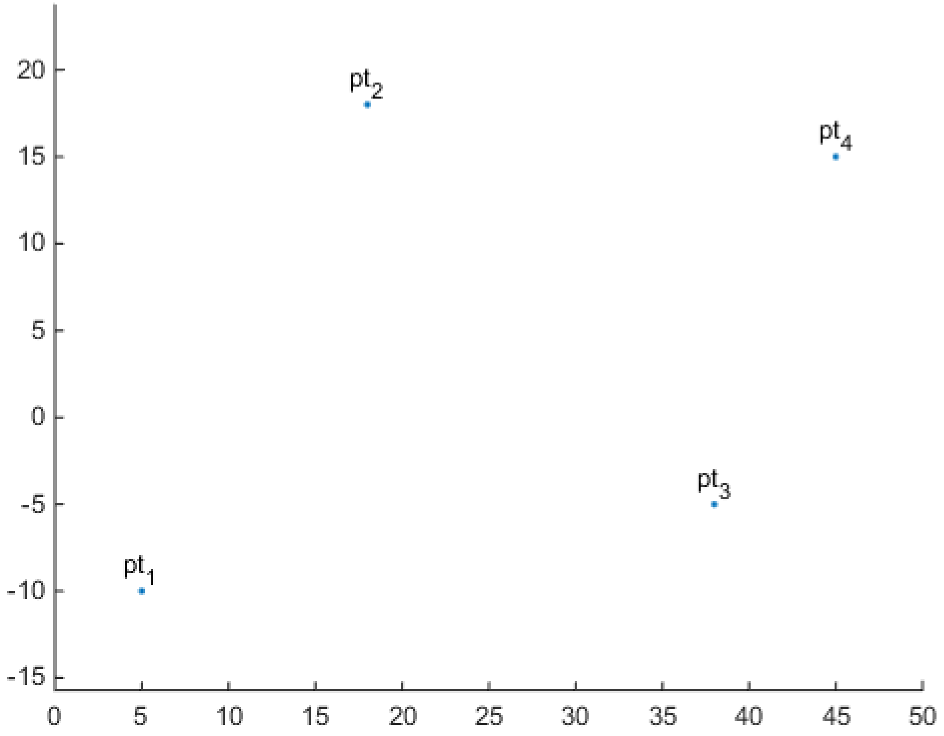
```
fplot(bezierCurve(1), bezierCurve(2), [0, 1])
hold on
 scatter(P(:,1), P(:,2),'filled')
```



p0 = [0 1],
p1 = [4 3],
p2 = [6 2],
p3 = [3 0],
p4 = [2 4]

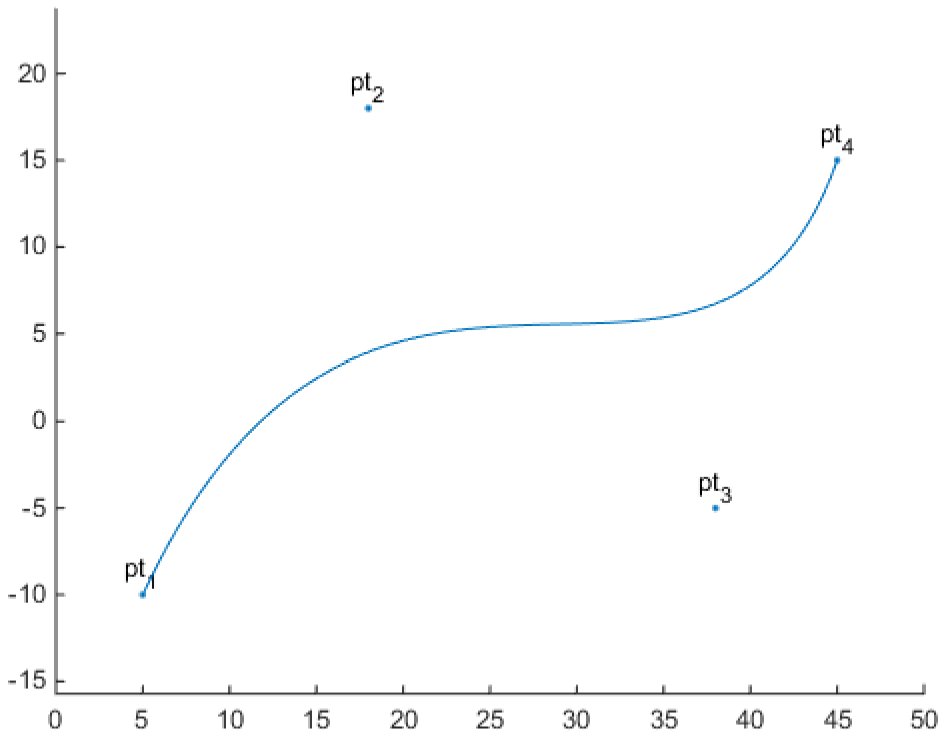
Curve di Bezier 2D

4 punti



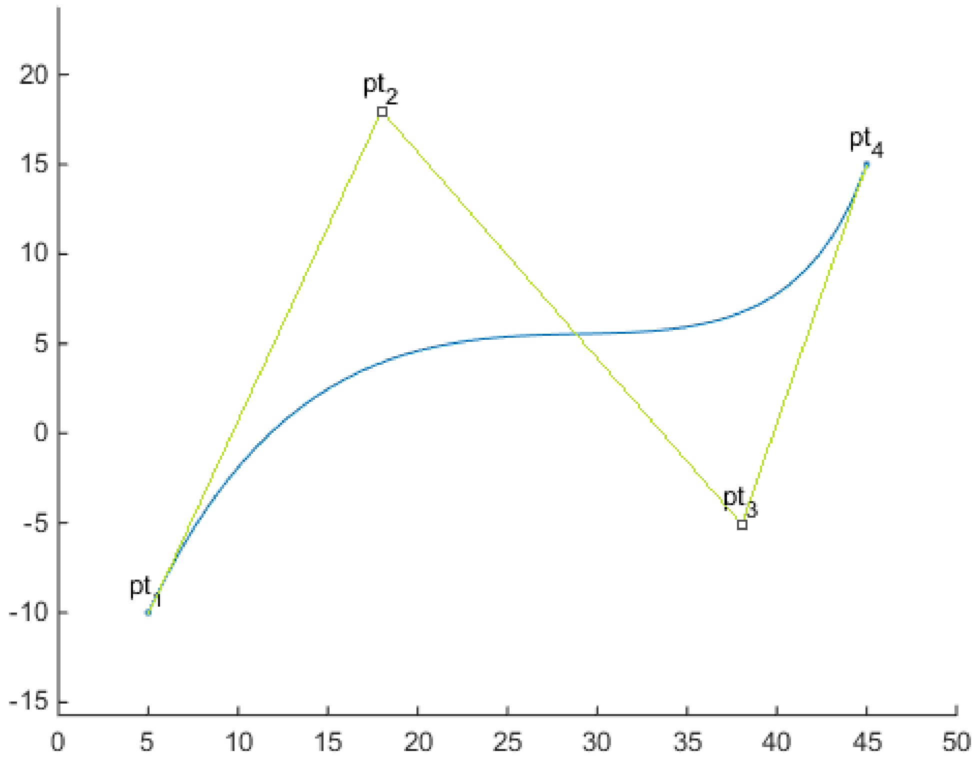
Curve di Bezier 2D

4 punti



Curve di Bezier 2D

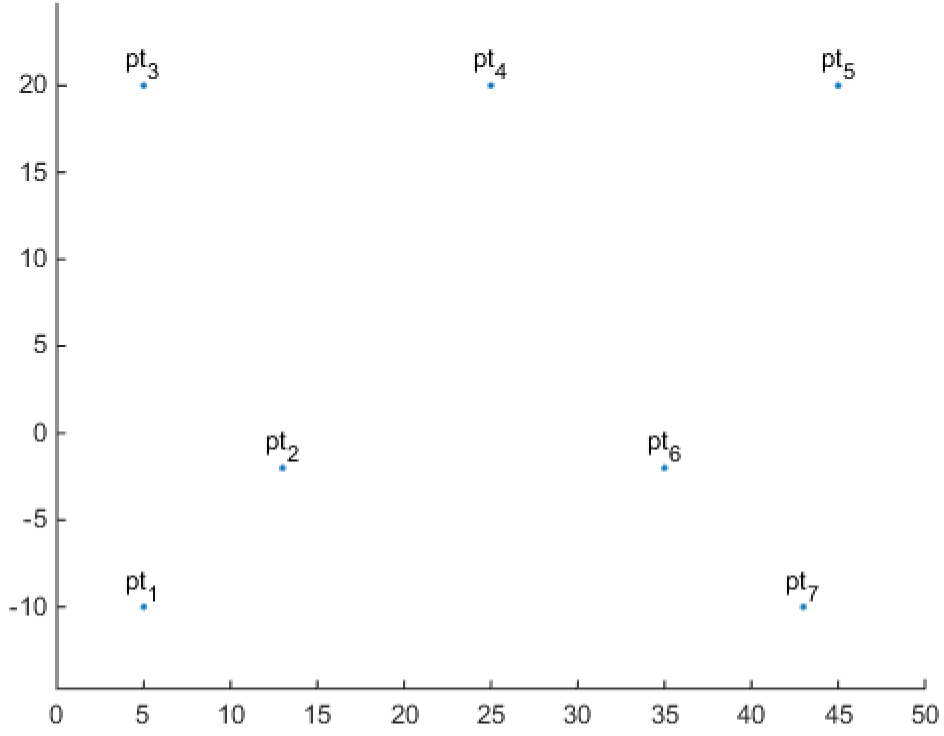
4 punti



Curve di Bezier 2D

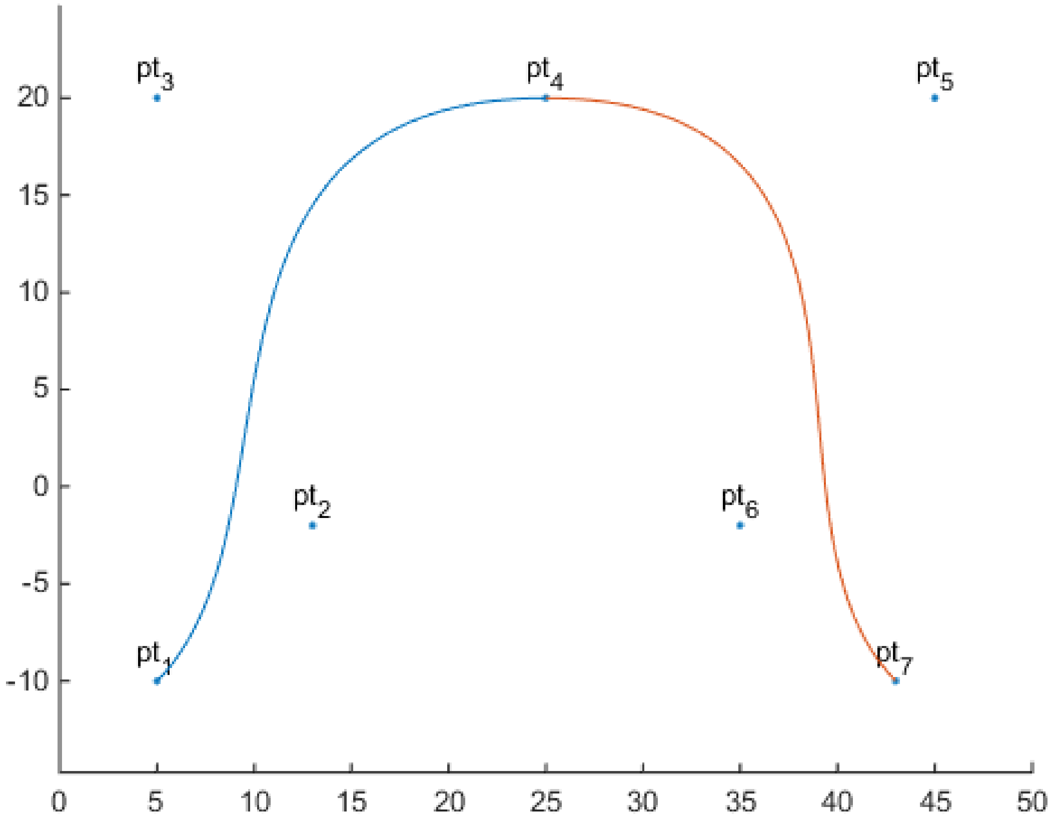
Connettere 2 curve di Bezier

```
xlim([0 50])
 axis equal
 pt1 = [ 5;-10];
pt2 = [13; -2];
 pt3 = [ 5; 20];
 pt4 = [25; 20];
placelabel(pt1,'pt_1’);
 placelabel(pt2,'pt_2’);
 placelabel(pt3,'pt_3’);
placelabel(pt4,'pt_4’);
pt5 = [45; 20];
 pt6 = [35; -2];
pt7 = [43;-10];
placelabel(pt5,'pt_5’);
 placelabel(pt6,'pt_6’);
placelabel(pt7,'pt_7');
```



Curve di Bezier 2D

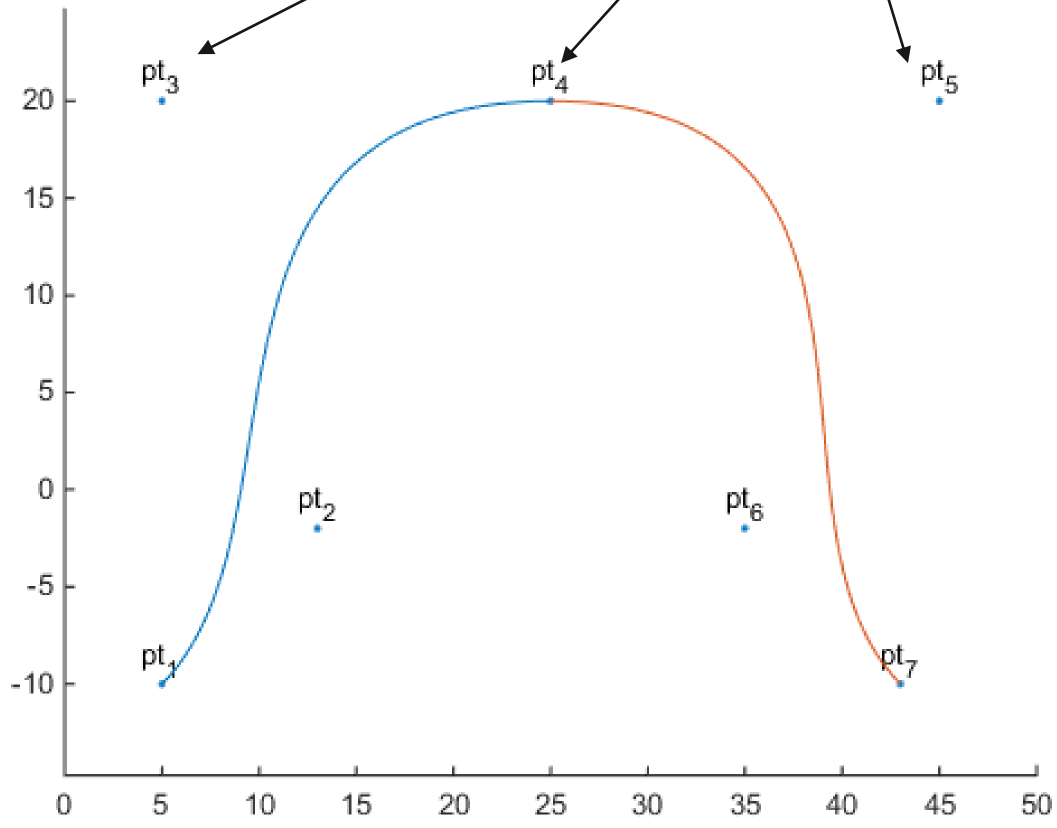
Connettere 2 curve di Bezier



Curve di Bezier 2D

Connettere 2 curve di Bezier

Gli ultimi 2 punti della prima curva allineati ai primi 2 della Seconda curva



Curve di Bezier 3D

Matrice costruita a partire dai punti di controllo

```
P = [0 0 0; 2 2 2; 2 -1 1; 6 1 3];
```

Calcolo della matrice di Bernstein del terzo ordine

```
>>syms t
>>B = bernsteinMatrix(3,t)

B = [ -(t - 1)^3, 3*t*(t - 1)^2, -3*t^2*(t - 1), t^3]
```

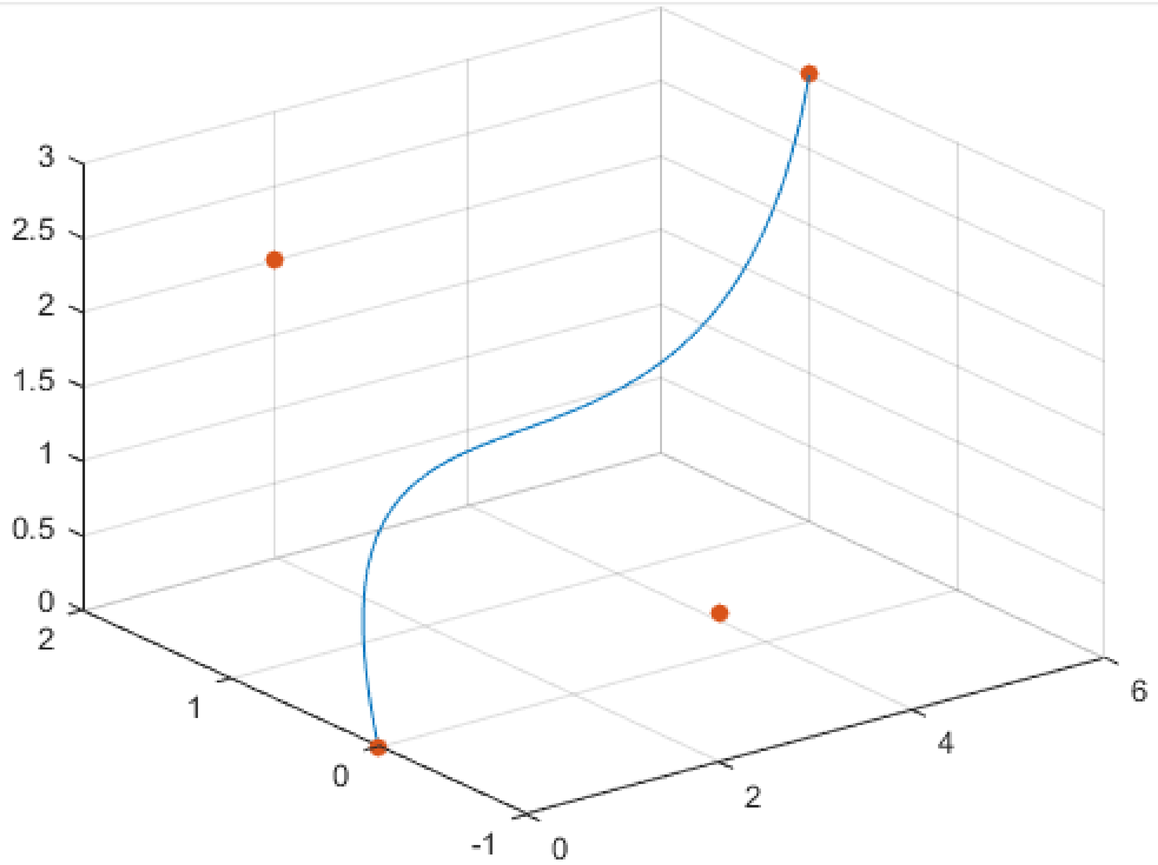
Costrure la curva di bezier

```
>>bezierCurve = simplify(B*P)
bezierCurve = [ 6*t*(t^2 - t + 1), t*(10*t^2 - 15*t + 6), 3*t*(2*t^2 - 3*t + 2)]
```

Curve di Bezier 3D

Plot curva

```
>>fplot3(bezierCurve(1), bezierCurve(2),
bezierCurve(3), [0, 1])
 >>hold on
>>catter3(P(:,1), P(:,2), P(:,3),'filled')
```



funzione Matlab **interp2**

ZI = interp2(X,Y,Z,XI,YI,method)

X,Y,Z :valori noti

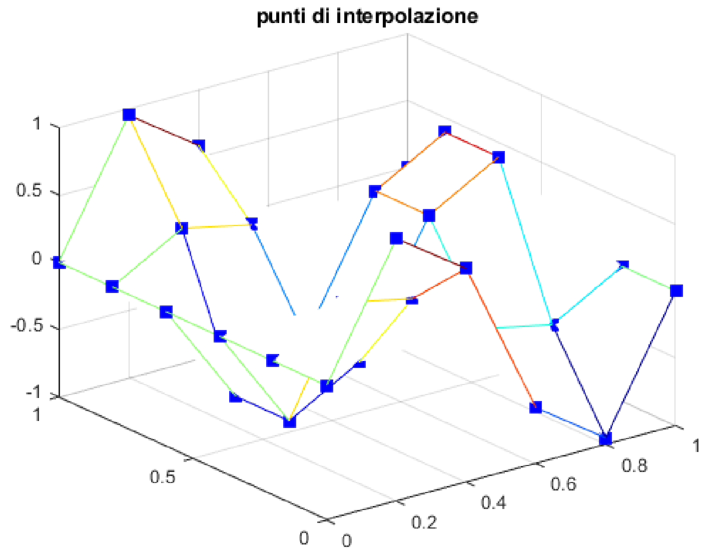
XI,YI : griglia in cui valutare l'interpolante

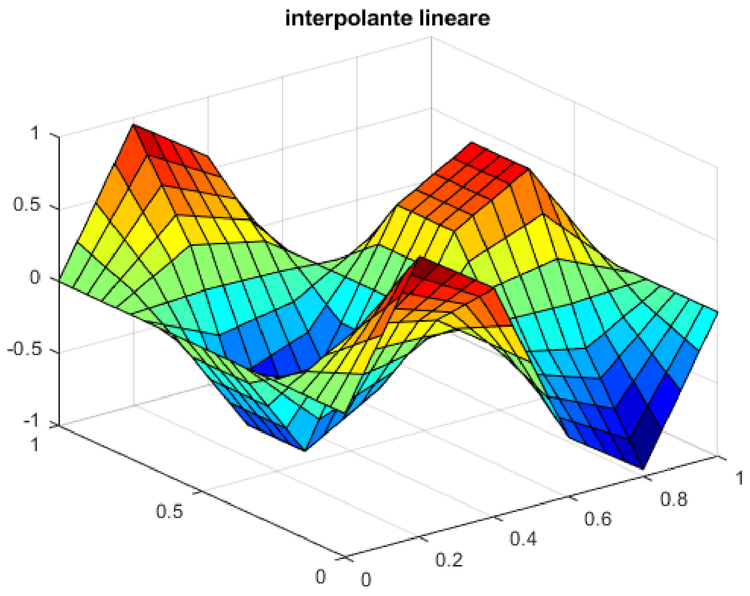
ZI : valori dell'interpolante nei punti XI,YI

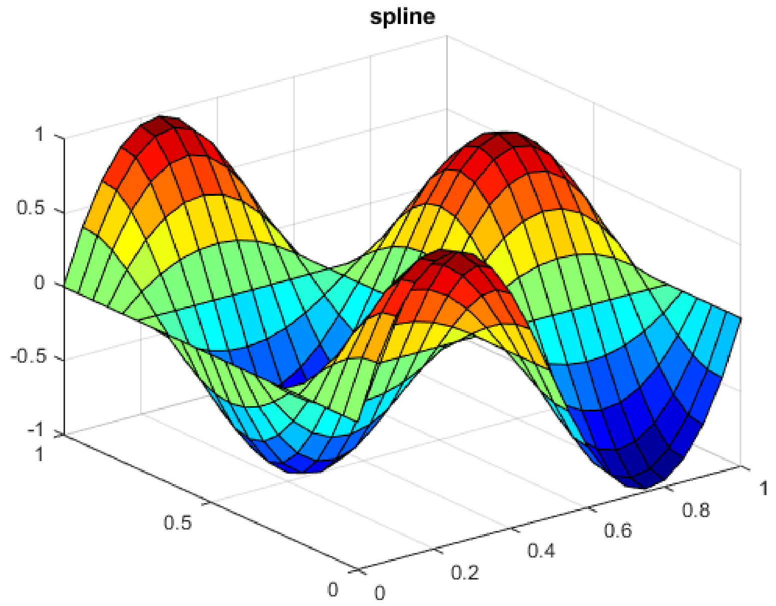
method : 'linear' bilineare

' spline' spline bilineare cubica

```
% interp. Bidimensionale
%griglia
 [x,y]=meshgrid(0:0.2:1,0:0.2:1);
 z=sin(2*pi*x).*cos(2*pi*y);
%griglia in cui valutare l'interpolante
 xi=0:0.05:1;yi=0:0.05:1;
 [xf,yf]=meshgrid(xi,yi);
%plot dei punti dati e quelli ottenuti con l'int.
figure(1)
mesh(x,y,z), title('punti di interpolazione');
figure(2)
 zi=interp2(x,y,z,xf,yf,'linear');
surf(xf,yf,zi),title('interpolante lineare');
figure(3)
zi=interp2(x,y,z,xf,yf,'spline');
surf(xf,yf,zi),title('spline');
```


punti di interpolazione
1
0.5
0
-0.5
-1
1
0.5
0
0
0.2
0.4
0.6
0.8
1


interpolante lineare
1
0.5
0
-0.5
-1
1
0.5
0
0
0.2
0.4
0.6
0.8
1


spline
1
0.5
0
-0.5
-1
1
0.5
0
0
0.2
0.4
0.6
0.8
1