Laurea Magistrale in Data Science, Calcolo Scientifico & Intelligenza Artificiale

# Numerical approximation for data modeling

Università degli Studi di Firenze
a.a. 2024/2025

Carlotta Giannelli

# Curve parametriche

1. Considerare la rappresentazione trigonometrica e razionale di una (semi)-circonferenza centrata nell'origine e raggio $R$,

$$\mathrm{X}(t) = R \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \; \theta \in [-\frac{\pi}{2}, +\frac{\pi}{2}], \quad \mathrm{X}(t) = R \begin{pmatrix} \frac{1-t^2}{1+t^2} \\ \frac{2t}{1+t^2} \end{pmatrix} \; t \in [-1, 1]$$

e vedere la distribuzione dei punti sulla circonferenza al variare di intervalli parametrici uniformi nei due casi.

2. Scrivere la function `k = curvature(dX,d2X)` che, date la derivata prima e seconda di una curva, calcoli la curvatura (con segno nel caso piano):

$$\kappa(t) := \frac{\left|\dot{\mathrm{X}}(t) \wedge \ddot{\mathrm{X}}(t)\right|}{\left|\dot{\mathrm{X}}(t)\right|^3}$$

al variare di $t$ nell'intervallo parametrico.

3. Verificare che nel caso della circonferenza la curvatura è costante (e uguale a $1/R$) per ogni valore del parametro di definizione.

# Curve parametriche nello spazio

1. Scrivere la function `tau = torsion(dX,d2X,d3X)` che, date la derivata prima, seconda e terza di una curva nello spazio, calcoli la torsione:

$$\tau(t) = \frac{\det(\dot{X}(t)\ \ddot{X}(t)\ \dddot{X}(t))}{\left|\dot{X}(t) \wedge \ddot{X}(t)\right|^2}$$

Considerare l'elica circolare retta

$$X(\theta) = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ a\,\theta \end{pmatrix}, \qquad \theta \in [0, T]$$

e verificare che in questo caso curvatura e torsione sono costanti e uguali a

$$\kappa(\theta) = \frac{1}{1+a^2}, \qquad \tau(\theta) = \frac{a}{1+a^2}.$$

# Polinomi di Bernstein e curve di Bézier

1. Scrivere la function `B = bernstein(i,n,t)` che calcola l'$i$-esimo polinomio di Bernstein di grado $n$:

   $$B_i^n(t) := \left(\begin{array}{c} n \\ i \end{array}\right) t^i(1-t)^{n-i}, \qquad \left(\begin{array}{c} n \\ i \end{array}\right) = \frac{n!}{i!(n-i)!}, \qquad i = 0, \ldots, n$$

   Fare il plot dei polinomi di base al variare di $n$ fra 1 e 10 e verificare la proprietá di partizione dell'unitá.

2. Utilizzando la funzione `ginput` per definire una serie di punti di controllo,
   - visualizzate la corrispondente curva di Bézier piana insieme al grafico della sua curvatura (dopo aver calcolato i punti sulla curva e sulle sue derivate prima e seconda rispetto ad una tabulazione fissata);
   - verificate il valore della curvatura negli estremi dell'intervallo parametrico.

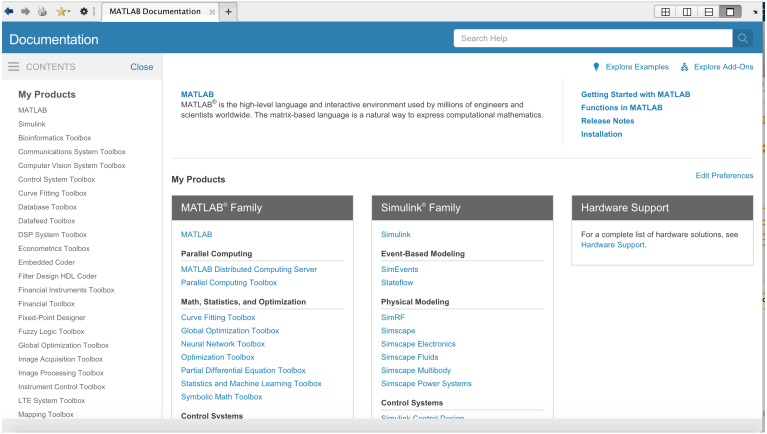
3. Verificare cosa succede scambiando due punti di controllo o modificandone uno, o invertendo l'ordine di tutti i punti di controllo.

4. Verificare la proprietà di linear precision.

5. Definire una curva di Bézier nello spazio e visualizzarla insieme al grafico della sua curvatura e torsione (dopo aver calcolato i punti sulla curva e sulle sue derivate prima, seconda e terza rispetto ad una tabulazione fissata).

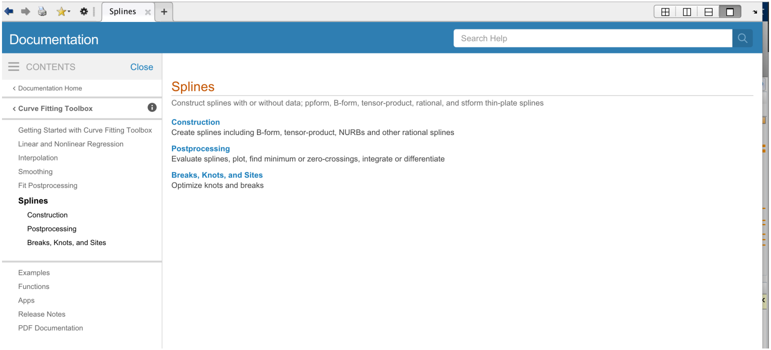
# B-splines nel MATLAB toolbox

## Documentation » Curve fitting toolbox » Splines



# B-splines nel MATLAB toolbox

Documentation » Curve fitting toolbox » Splines



# Construction

Documentation » Curve fitting toolbox » Splines » Construction

- **Functions**
  - `bspline` ⇒ provare `bspligui`

# Construction

Documentation » Curve fitting toolbox » Splines » Construction

- **Functions**
  - `bspline` ⇒ provare `bspligui`
  - B-spline di grado 1 e 2 su nodi uniformi
    1) Confrontare i comandi
    `bspline([0 1 2])` e `bspline([0 1 2 3])`
    con

$$N_{0,2}(t) = \begin{cases} t & \text{se } t \in [0,1) \\ 2-t & \text{se } t \in [1,2] \end{cases} \qquad N_{0,3}(t) = \begin{cases} t^2 & \text{se } t \in [0,1) \\ \frac{t(2-t)+(3-t)(t-1)}{2} & \text{se } t \in [1,2) \\ \frac{(3-t)^2}{2} & \text{se } t \in [2,3] \end{cases}$$

# Construction

Documentation » Curve fitting toolbox » Splines » Construction

- **Functions**
  - `bspline` ⇒ provare `bspligui`
  - B-spline di grado 1 e 2 su nodi uniformi

    1) Confrontare i comandi

    `bspline([0 1 2])` e `bspline([0 1 2 3])`

    con

$$N_{0,2}(t) = \begin{cases} t & \text{se } t \in [0,1) \\ 2-t & \text{se } t \in [1,2] \end{cases} \qquad N_{0,3}(t) = \begin{cases} t^2 & \text{se } t \in [0,1) \\ \frac{t(2-t)+(3-t)(t-1)}{2} & \text{se } t \in [1,2) \\ \frac{(3-t)^2}{2} & \text{se } t \in [2,3] \end{cases}$$

2) Definire una B-spline su un vettore dei nodi locale generico

# Construction

Documentation » Curve fitting toolbox » Splines » Construction

- **Functions**
  - `bspline` $\Rightarrow$ provare `bspligui`
  - B-spline di grado 1 e 2 su nodi uniformi
    1) Confrontare i comandi
    `bspline([0 1 2])` e `bspline([0 1 2 3])`
    con

$$N_{0,2}(t) = \begin{cases} t & \text{se } t \in [0,1) \\ 2-t & \text{se } t \in [1,2] \end{cases} \qquad N_{0,3}(t) = \begin{cases} t^2 & \text{se } t \in [0,1) \\ \frac{t(2-t)+(3-t)(t-1)}{2} & \text{se } t \in [1,2) \\ \frac{(3-t)^2}{2} & \text{se } t \in [2,3] \end{cases}$$

2) Definire una B-spline su un vettore dei nodi locale generico
3) Utilizzando il comando `spcol`, scrivere una function che dati il vettore dei nodi e il grado da considerare, disegni il grafico della base delle B-spline.

# B-spline di grado *d*

```
function getBsplineBasis(t,d)
  x = [t(1):.1:t(end)];
  c = spcol(t,d+1,x);
  [l,m] = size(c);
  figure (1); hold on;
  for tt=t, plot([tt tt],[0 1],'-'), end
  plot(x,c,'linew',2); axis image; box;
  figure(2); hold on;
  c = c+ones(l,1)*[0:m-1];
  for tt=t,plot([tt tt],[0 m],'-'), end
  plot(x,c,'linew',2), axis image; box;
end
```

# B-spline

4) Utilizzando i comandi `spmak` e `fnplt` visualizzare i polinomi di Bernstein di grado 1, 2, 3 ($k+1$ zeri e uni nel vettore dei nodi)

# B-spline

4) Utilizzando i comandi `spmak` e `fnplt` visualizzare i polinomi di Bernstein di grado 1, 2, 3 ($k+1$ zeri e uni nel vettore dei nodi)

5) Scrivere una function che visulizza i polinomi di Bernstein di grado $d$ utilizzando `spcol`.

# B-spline

4) Utilizzando i comandi `spmak` e `fnplt` visualizzare i polinomi di Bernstein di grado 1, 2, 3 ($k+1$ zeri e uni nel vettore dei nodi)

5) Scrivere una function che visulizza i polinomi di Bernstein di grado $d$ utilizzando `spcol`.

```
function getBerpol(d)
  knots = [zeros(1,d+1) ones(1,d+1)];
  x = [0:.01:1];
  c = spcol(knots,d+1,x);
  plot(x,c,'linew',2)
  title(['Polinomi di Bernstein di grado ', num2str(d)]);
  axis equal; axis([0 1 0 1]);
  end
```

6) Verifica della proprietà di partizione dell'unità

# B-spline

4) Utilizzando i comandi `spmak` e `fnplt` visualizzare i polinomi di Bernstein di grado 1, 2, 3 ($k+1$ zeri e uni nel vettore dei nodi)

5) Scrivere una function che visulizza i polinomi di Bernstein di grado $d$ utilizzando `spcol`.

```
function getBerpol(d)
  knots = [zeros(1,d+1) ones(1,d+1)];
  x = [0:.01:1];
  c = spcol(knots,d+1,x);
  plot(x,c,'linew',2)
  title(['Polinomi di Bernstein di grado ', num2str(d)]);
  axis equal; axis([0 1 0 1]);
  end
```

6) Verifica della proprietà di partizione dell'unità

# B-spline

7) Interpolazione funzionale

- Considerare il vettore dei breakpoints $[0, 0.25, 0.5, 0.75, 1]$.
- Utilizzare il comando `augknt` per definire il vettore esteso dei nodi `t` (con nodi ausiliari ripetuti) al variare di $k = 2, 3, 4$.
- Utilizzare `aveknt` per definire il vettore $\mathrm{x} = [x_0, \dots, x_n]$ delle ascisse di interpolazione (ascisse di Greville)

$$x_i = \frac{t_{i+1} + \ldots + t_{k-1}}{k-1} = \frac{\sum_{j=1}^{k-1} t_{i+j}}{k-1}, \qquad i = 0, \dots, n,$$

  definite come valor medio di $k-1$ nodi consecutivi ($n+1 =$`length(t)`$- k$).
- Associare un valore $y_i$ ad ogni ascissa di interpolazione (es. con la funzione `rand(length(x),1)`) per definire il vettore di valori da interpolare $\mathrm{y} = [y_0, \dots, y_n]^T$.
- Calcolare la funzione B-spline $s(t)$ interpolante i valori $y_i$ nelle ascisse $x_i$:

$$s(t) = \sum_{j=0}^{n} d_j N_{j,k}(t) \quad \text{t.c.} \quad s(x_i) = \sum_{j=0}^{n} d_j N_{j,k}(x_i) = y_i, \quad i = 0, \dots, n,$$

  risolvendo il sistema $C\mathrm{d} = \mathrm{y}$, dove $C$ è la matrice di collocazione delle B-spline (`spcol`), per calcolare il vettore $\mathrm{d} = [d_0, \dots, d_n]^T$.
- Confrontare il risultato con la spline calcolata da `spapi`.

# B-spline

7) Interpolazione funzionale

- Considerare il vettore dei breakpoints $[0, 0.25, 0.5, 0.75, 1]$.
- Utilizzare il comando `augknt` per definire il vettore esteso dei nodi `t` (con nodi ausiliari ripetuti) al variare di $k = 2, 3, 4$.
- Utilizzare `aveknt` per definire il vettore $\mathrm{x} = [x_0, \dots, x_n]$ delle ascisse di interpolazione (ascisse di Greville)

$$x_i = \frac{t_{i+1} + \ldots + t_{k-1}}{k-1} = \frac{\sum_{j=1}^{k-1} t_{i+j}}{k-1}, \qquad i = 0, \dots, n,$$

  definite come valor medio di $k-1$ nodi consecutivi ($n+1 =$`length(t)`$-k$).
- Associare un valore $y_i$ ad ogni ascissa di interpolazione (es. con la funzione `rand(length(x),1)`) per definire il vettore di valori da interpolare $\mathrm{y} = [y_0, \dots, y_n]^T$.
- Calcolare la funzione B-spline $s(t)$ interpolante i valori $y_i$ nelle ascisse $x_i$:

$$s(t) = \sum_{j=0}^{n} d_j N_{j,k}(t) \quad \text{t.c.} \quad s(x_i) = \sum_{j=0}^{n} d_j N_{j,k}(x_i) = y_i, \quad i = 0, \dots, n,$$

  risolvendo il sistema $C\mathrm{d} = \mathrm{y}$, dove $C$ è la matrice di collocazione delle B-spline (`spcol`), per calcolare il vettore $\mathrm{d} = [d_0, \dots, d_n]^T$.
- Confrontare il risultato con la spline calcolata da `spapi`.

8) Curve B-spline

# Spline Function Naming Conventions

Most of the spline commands in this toolbox have names that follow one of the following patterns:

- `cs...` commands construct cubic splines (in ppform)
- `sp...` commands construct splines in B-form
- `fn...` commands operate on spline functions
- `..2...` commands convert something
- `..api` commands construct an approximation by interpolation
- `..aps` commands construct an approximation by smoothing
- `..ap2` commands construct a least-squares approximation
- `...knt` commands construct (part of) a particular knot sequence
- `...dem` commands are examples.

# Parametrizzazioni

Siano

- $\mathbf{t} := \{t_0, \ldots, t_{n+k}\}$: vettore esteso dei nodi associato a $[a, b] = [0, 1]$;
- $\{\mathbf{p}_i\}_{i=0,\ldots,n}$: punti di interpolazione.

# Parametrizzazioni

Siano

- $\mathrm{t} := \{t_0, \dots, t_{n+k}\}$: vettore esteso dei nodi associato a $[a, b] = [0, 1]$;
- $\{\mathrm{p}_i\}_{i=0,\dots,n}$: punti di interpolazione.

I valori del parametro $\{x_i\}_{i=0,\dots,n}$ in cui interpolare i punti $\mathrm{p}_i$ possono essere scelti tramite:

- ascisse di Greville:

$$x_i = \frac{t_{i+1} + \dots + t_{k-1}}{k-1} = \frac{\sum_{j=1}^{k-1} t_{i+j}}{k-1}, \qquad i = 0, \dots, n,$$

- parametrizzazione uniforme:

$$x_i = i/n$$

- parametrizzazione chordlength: posto $\Delta \mathrm{p}_i = ||\mathrm{p}_{i+1} - \mathrm{p}_i||$, $i = 0, \dots, n-1$,

$$x_0 = 0, \quad x_{i+1} = x_i + \frac{\Delta \mathrm{p}_i}{\Delta \mathrm{p}}, \quad \text{con} \quad \Delta \mathrm{p} = \sum_{i=0}^{n-1} \Delta \mathrm{p}_i$$

- parametrizzazione centripeta: posto $\Delta \mathrm{p}_i = ||\mathrm{p}_{i+1} - \mathrm{p}_i||^{1/2}$, $i = 0, \dots, n-1$,

$$x_0 = 0, \quad x_{i+1} = x_i + \frac{\Delta \mathrm{p}_i}{\Delta \mathrm{p}}, \quad \text{con} \quad \Delta \mathrm{p} = \sum_{i=0}^{n-1} \Delta \mathrm{p}_i.$$