# **MathLab Tool ANMGLib 4.1**

## **Funzioni per il Grafico**

### **Impostazioni di base**

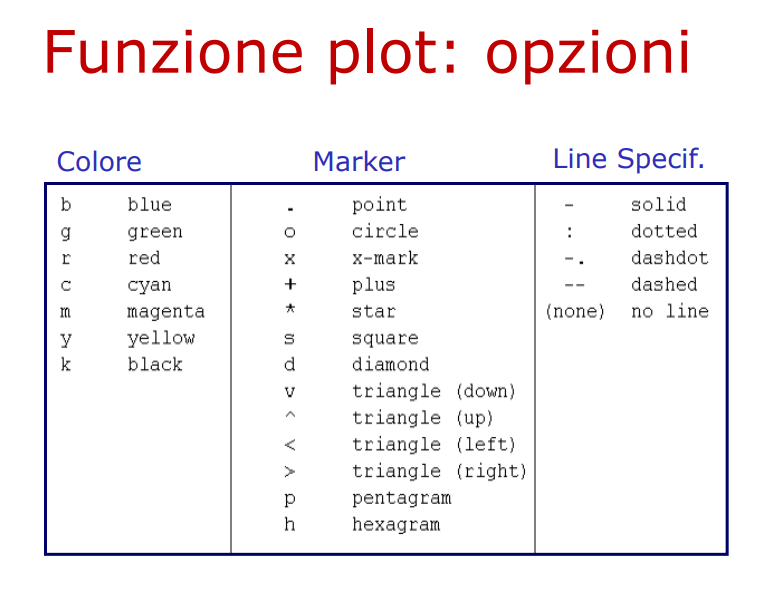
* **Sfondo colorato:** Ax = gca;
* Ax.Color = 'g';
* **Alternativa per settare il colore dello sfondo:** set(gca, 'color', [1.0, 1.0, 0.8]);
* **Apertura della figura (da usare con curve di Bézier o interpolazione, evitare figure(1), potrebbe non funzionare):** open\_figure(1);

### **Comandi per la visualizzazione**

* **Mantenere il disegno attivo:** hold on;
* **Scala degli assi uniforme:** axis equal;
* **Titolo del grafico:** title('Esempio di Testo');
* **Attivare la griglia:** grid on;

### **Disegno degli assi cartesiani**

* **Funzione:** axis\_plot(lax);
* **Descrizione:** Disegna il sistema di assi cartesiani 2D o 3D. Se chiamata all'inizio, imposta la visualizzazione in 3D.
* **Parametro:**
  + lax: Lunghezza degli assi (valore predefinito: 1 se omesso).
* **Argomenti opzionali (varargin):** Opzioni di disegno da specificare nell'ordine seguente:
  + **LineSpecification**: Stile della linea, simbolo del marker e colore (in un'unica stringa).
  + **LineWidth**: Larghezza della linea.
  + **MarkerEdgeColor**: Colore esterno del marker.
  + **MarkerFaceColor**: Colore interno del marker.
  + **MarkerSize**: Dimensione del marker.



### **Struttura: bezier**

**Descrizione:** Definizione di una curva di Bézier.

* **bezier.deg**: Grado della curva.
* **bezier.cp**: Lista dei punti di controllo (dimensione (bezier.deg+1)x2).
* **bezier.ab**: Intervallo di definizione. (mettere [0 1])

## **Esplorazione del Codice: Funzioni per Curve e Superfici**

Questa sezione riassume una serie di funzioni MATLAB per la creazione, manipolazione e visualizzazione di curve e superfici, con un focus su spline e NURBS. Le funzioni sono organizzate per tipologia e funzionalità.

### 

### 

### **1. Funzioni Base**

* **basis\_bernst2d\_plot(gu,gv,a,b,c,d)**: Visualizza le funzioni base di Bernstein bivariate su un dominio rettangolare [a,b]x[c,d], con gradi gu e gv.
* **basis\_nurbs1d\_plot(g,t,w)**: Visualizza le funzioni base B-spline razionali univariate di grado g, definite dai nodi t e dai pesi w.
* **basis\_spline1d\_plot(g,t,flag)**: Valuta e visualizza le funzioni base B-spline univariate di grado g definite dai nodi t.  
  Parametro flag controlla la visualizzazione di derivate prime e seconde.
* **bernst(g,x):** Calcola i valori delle funzioni base di Bernstein di grado **g** nei punti **x** nell'intervallo **[0,1]**.
* **bernst\_valder(g,x)**: Calcola i valori e le derivate prime delle funzioni base di Bernstein di grado **g** nei punti **x** nell'intervallo **[0,1].**
* **gcw\_bspl(g,t,w,x)**: Valuta le funzioni base B-spline razionali (non nulle) di grado **g** definite dai nodi **t** e dai pesi **w** nei punti **x**.
* **gcw\_bspl\_2v(gu,gv,u,v,W,x,y):** Valuta le funzioni base B-spline razionali bivariate di grado **gu** in u e **gv** in **v**, definite dai nodi **u** e **v** e dalla **matrice dei pesi W**, su una griglia di punti **(x,y).**
* **gc\_bpspl(g,t,x)**: Calcola i valori e le derivate prime delle funzioni base B-spline (non nulle) di grado **g** definite dai nodi **t** nei punti **x**.
* **gc\_bpsspl(g,t,x):** Calcola i valori, le derivate prime e seconde delle funzioni base B-spline (non nulle) di grado **g** definite dai nodi **t** nei punti **x**.
* **gc\_bspl(g,t,x):** Valuta le funzioni base B-spline (non nulle) di grado g definite dai nodi t nei punti **x**.
* **gc\_bspl\_valder(g,t,x,od):** Valuta i valori e le derivate fino all'ordine **od** delle funzioni base B-spline (non nulle) di grado **g** definite dai nodi **t** nei punti **x**.

### **2. Funzioni di Utilità**

* **gc\_findinppab(ppab,x)**: Determina l’indice dell’intervallo nodale contenente i punti x in un vettore di nodi ppab.
* **gc\_findint(g,t,x):** Determina l'indice dell'intervallo nodale contenente i punti **x** in un vettore di nodi **t** per una spline di grado **g.**
* **gc\_max\_dist(pin, p1, pmed, p2, pfin):** Calcola la massima distanza dei punti **p1**, **pmed** e **p2** dalla retta passante per **pin** e **pfin**.
* **gc\_mesh(g,t,n):** Genera una mesh opportuna dell'intervallo di definizione di una spline di grado **g** con nodi **t**, suddividendo ogni intervallo in **n** sottointervalli.
* **greville(g,t):** Calcola le ascisse di Greville per uno spazio spline definito dal grado **g** e dalla partizione nodale **t**.
* **partition\_equi(x)**: Genera una partizione nodale equidistante a partire da un vettore di punti **x**.
* **partition\_nak(x):** Genera una partizione nodale per l'interpolazione con spline cubiche e condizione not-a-knot a partire da un vettore di punti **x**.
* **partition\_nat(x):** Genera una partizione nodale per l'interpolazione con spline cubiche e condizione naturale a partire da un vettore di punti **x**.
* **partition\_per(x):** Genera una partizione nodale per l'interpolazione con spline cubiche e condizione periodica a partire da un vettore di punti **x**.
* **scalar\_derivatives(qx,qy):** Stima le derivate discrete a partire da una lista di punti **(qx,qy)**.

### **3. Funzioni di Valutazione e Suddivisione**

* deboor\_val(spline, x): Calcola il valore di una curva spline nD usando l'algoritmo di de Boor nei punti x.
* deboor\_valder(spline, kk, x): Calcola il valore e le derivate fino all'ordine kk di una curva spline nD definita dalla struttura spline nei punti x utilizzando l'algoritmo di de Boor.
* decast\_subdiv(bezier, x): Suddivide una curva di Bézier nD definita dalla struttura bezier nel punto x in due curve di Bézier (bez\_sx e bez\_dx).
* decast\_val(bezier, t): Calcola il valore di una curva di Bézier nD definita dalla struttura bezier nei punti t utilizzando l'algoritmo di de Casteljau.
* decast\_valder(bezier, k, t): Calcola il valore e le derivate fino all'ordine k di una curva di Bézier nD definita dalla struttura bezier nei punti t utilizzando l'algoritmo di de Casteljau.
* lane\_riesenfeld(fun, TOL): Determina gli zeri di una funzione polinomiale data nella base di Bernstein utilizzando il metodo di Lane-Riesenfeld, con una tolleranza TOL.
* mdppdecast\_val(ppbez, t): Calcola il valore di una curva di Bézier a tratti multi-grado nD definita dalla struttura ppbez nei punti t.
* nurbs\_val(nurbs, x): Calcola il valore di una curva NURBS nD definita dalla struttura nurbs nei punti x utilizzando l'algoritmo di de Boor.
* nurbs\_valder(nurbs, kk, x): Calcola il valore e le derivate fino all'ordine kk di una curva NURBS nD definita dalla struttura nurbs nei punti x utilizzando l'algoritmo di de Boor.
* ppbezier\_valder(ppP, k, t): Calcola il valore e le derivate fino all'ordine k di una curva di Bézier a tratti nD definita dalla struttura ppP nei punti t.
* ppdecast\_val(ppbez, t): Calcola il valore di una curva di Bézier a tratti nD definita dalla struttura ppbez nei punti t.

### **4. Funzioni per Curve 2D**

* curv2\_bezier\_load(filename): Carica una curva di Bézier 2D da un file .db.
* curv2\_bezier\_plot(bezier, np, varargin): Disegna una curva di Bézier 2D definita dalla struttura bezier, valutandola in np punti.
* curv2\_bezier\_tan\_plot(bezier, np, varargin): Disegna il vettore tangente di una curva di Bézier 2D definita dalla struttura bezier, valutandola in np punti.
* curv2\_closest\_point(x, y, x0, y0): Determina il punto più vicino a (x0, y0) in una lista di punti (x, y).
* curv2\_closest\_points(x, y, x0, y0): Determina i tre punti più vicini a (x0, y0) in una lista di punti (x, y).
* curv2\_kur\_plot(cs2name, a, b, np, varargin): Calcola e disegna la curvatura di una curva 2D definita nel file cs2name, valutandola in np punti nell'intervallo [a, b].
* curv2\_mdppbezier\_load(filename): Carica una curva di Bézier a tratti multi-grado 2D da un file .db.
* curv2\_nurbs\_load(filename): Carica una curva NURBS 2D da un file .db.
* curv2\_nurbs\_plot(nurbs, np, varargin): Disegna una curva NURBS 2D definita dalla struttura nurbs, valutandola in np punti.
* curv2\_nurbs\_tan\_plot(nurbs, np, varargin): Disegna il vettore tangente di una curva NURBS 2D definita dalla struttura nurbs, valutandola in np punti.
* curv2\_spline\_length(spline): Calcola la lunghezza di una curva spline 2D definita dalla struttura spline.
* curv2\_spline\_load(filename): Carica una curva spline 2D da un file .db.
* curv2\_spline\_tan\_plot(spline, np, varargin): Disegna il vettore tangente di una curva spline 2D definita dalla struttura spline, valutandola in np punti.
* curv2\_spline\_vel\_plot(spline, np, varargin): Disegna la funzione velocità di una curva spline 2D definita dalla struttura spline, valutandola in np punti per tratto.
* curv2\_trans\_plot(curvname, a, b, np, M, varargin): Trasforma una curva 2D definita nel file curvname mediante la matrice di trasformazione M, valutandola in np punti nell'intervallo [a, b].
* **function [x,y]=curv2\_bezier\_plot(curvname,a,b,np,varargin):**Disegna una curva 2D curvname --> nome del file con l'espressione parametrica (function [x,y]=nameFct(t))della curva a,b --> intervallo di definizione np --> numero di punti di valutazione
* **function val=curv2\_ppbezier\_area(ppP):**Calcola l'area della curva 2D di Bezier a tratti degree ppP. ppP --> struttura della curva 2D di Bezier a tratti:ppP.deg --> grado della curva bppP.cp --> lista dei punti di controllo ppP.ab --> partizione nodale di [a b] val <-- valore dell'area racchiusa dalla curva; se la curva e' aperta, viene considerata la parte di piano limitata dalla curva e dall'origine
* **function bezQ = curv2\_ppbezier\_de(bezP,de)** Calcola la curva 2D di Bezier a tratti degree elevata di de della curva di Bezier a tratti bezP:bezP --> struttura della curva 2D di Bezier a tratti: bezP.deg --> grado della curva bezP.cp --> lista dei punti di controllo bezP.ab --> partizione nodale di [a b] bezQ <-- struttura della curva 2D di Bezier a tratti :bezQ.deg --> grado della curva bezQ.cp --> lista dei punti di controllo bezQ.ab --> partizione nodale di [a b] utilizza la function gc\_pol\_de2d
* **function bezQ = curv2\_bezier\_de(bezP,de)** Calcola la curva 2D di Bezier degree elevata di de della curva bezP bezP --> struttura della curva 2D di Bezier : bezP.deg --> grado della curva bezP.cp --> lista dei punti di controllo bezP.ab --> intervallo di definizione [a b] de --> intero positivo che indica di quanto elevare il grado della curva in input bezQ <-- struttura della curva 2D di Bezier : bezQ.deg --> grado della curva bezQ.cp --> lista dei punti di controllo bezQ.ab --> intervallo di definizione [a b] utilizza la function gc\_pol\_de2d

### **5. Funzoni per Curve 3D**

* curv3\_bezier\_load(filename): Carica una curva di Bézier 3D da un file .db.
* curv3\_closest\_point(x, y, z, x0, y0, z0): Determina il punto più vicino a (x0, y0, z0) in una lista di punti (x, y, z).
* curv3\_closest\_points(x, y, z, x0, y0, z0): Determina i tre punti più vicini a (x0, y0, z0) in una lista di punti (x, y, z).
* curv3\_kur\_plot(ct3name, a, b, np, varargin): Calcola e disegna la curvatura di una curva 3D definita nel file ct3name, valutandola in np punti nell'intervallo [a, b].
* curv3\_mdppbezier\_load(filename): Carica una curva di Bézier a tratti multi-grado 3D da un file .db.
* curv3\_nurbs\_load(filename): Carica una curva NURBS 3D da un file .db.
* curv3\_nurbs\_plot(nurbs, np, varargin): Disegna una curva NURBS 3D definita dalla struttura nurbs, valutandola in np punti.
* curv3\_nurbs\_tan\_plot(nurbs, np, varargin): Disegna il vettore tangente di una curva NURBS 3D definita dalla struttura nurbs, valutandola in np punti.
* curv3\_nurbs\_tor\_plot(nurbs, np, varargin): Calcola e disegna la torsione di una curva NURBS 3D definita dalla struttura nurbs, valutandola in np punti per tratto.
* curv3\_splineCC2\_nak\_interp(Q, param): Calcola la spline cubica di interpolazione not-a-knot 3D per una lista di punti Q utilizzando la parametrizzazione param.
* curv3\_splineCC2\_nat\_interp(Q, param): Calcola la spline cubica di interpolazione naturale 3D per una lista di punti Q utilizzando la parametrizzazione param.
* curv3\_splineCC2\_per\_interp(Q, param): Calcola la spline cubica di interpolazione periodica 3D per una lista di punti Q utilizzando la parametrizzazione param.
* curv3\_spline\_plot(spline, np, varargin): Disegna una curva spline 3D definita dalla struttura spline, valutandola in np punti per tratto.
* curv3\_spline\_subdiv(spline, x): Suddivide una curva spline 3D definita dalla struttura spline nel punto x in due curve spline, spline\_sx e spline\_dx.
* curv3\_spline\_tor\_plot(spline, np, varargin): Calcola e disegna la torsione di una curva spline 3D definita dalla struttura spline, valutandola in np punti per tratto.
* curv3\_spline\_vel\_plot(spline, np, varargin): Disegna la funzione velocità di una curva spline 3D definita dalla struttura spline, valutandola in np punti per tratto.
* curv3\_tan\_plot(ct3name, a, b, np, varargin): Disegna il vettore tangente di una curva 3D definita nel file ct3name, valutandola in np punti nell'intervallo [a, b].
* curv3\_tor\_plot(ct3name, a, b, np, varargin): Calcola e disegna la torsione di una curva 3D definita nel file ct3name, valutandola in np punti nell'intervallo [a, b].
* curv3\_vel\_val(cp3name, x): Calcola il valore della velocità di una curva 3D definita nel file cp3name nel punto x.

### **6. Funzioni per Superfici**

* **circle3d(O, r, n)**:  
  Genera i punti di un cerchio 3D di centro OOO, raggio rrr e normale nnn.
* **solid\_cylinder\_trans\_plot(A, B, r, nu, nv, M, type, varargin)**:  
  Disegna un cilindro solido definito dall'asse di estremi AAA e BBB, dal raggio rrr, applicando la trasformazione MMM e utilizzando una griglia di valutazione nu×nvnu \times nvnu×nv.
* **solid\_plot(solidname, nu, nv, type, varargin)**:  
  Disegna un solido elementare (cubo, cilindro o cono) specificato da solidname, utilizzando una griglia di valutazione nu×nvnu \times nvnu×nv e il tipo di disegno type.
* **surf\_bezier\_load(filename)**:  
  Carica una superficie di Bézier 3D da un file .db.
* **surf\_bezier\_plot(surfbezier, nu, nv, type, varargin)**:  
  Disegna una superficie di Bézier 3D definita dalla struttura surfbezier, utilizzando una griglia di valutazione nu×nvnu \times nvnu×nv e il tipo di disegno type.
* **surf\_nurbs\_iso\_curv(surfnurbs, fuv, uv)**:  
  Estrae una curva isoparametrica (isocurva) da una superficie NURBS 3D definita dalla struttura surfnurbs nella direzione fuv (1 per u, 2 per v) al valore del parametro uv.
* **surf\_nurbs\_load(filename)**:  
  Carica una superficie NURBS 3D da un file .db.
* **surf\_nurbs\_plot(surfnurbs, ni, nj, type, varargin)**:  
  Disegna una superficie NURBS 3D definita dalla struttura surfnurbs, utilizzando una griglia di valutazione ni×njni \times njni×nj e il tipo di disegno type.
* **surf\_nurbs\_save(filename, surfnurbs)**:  
  Salva una superficie NURBS 3D definita dalla struttura surfnurbs in un file .db.
* **surf\_nurbs\_swung(nurbsA, nurbsB, scal)**:  
  Crea una superficie NURBS 3D ruotando una curva NURBS nurbsA attorno a un asse definito da una seconda curva NURBS nurbsB, con un fattore di scala scal.
* **surf\_spline\_iso\_curv(surfspline, fuv, uv)**:  
  Estrae una curva isoparametrica (isocurva) da una superficie spline 3D definita dalla struttura surfspline nella direzione fuv (1 per u, 2 per v) al valore del parametro uv.
* **surf\_spline\_iso\_curv\_plot(surfspline, fuv, uv, varargin)**:  
  Disegna una curva isoparametrica (isocurva) estratta da una superficie spline 3D definita dalla struttura surfspline nella direzione fuv (1 per u, 2 per v) al valore del parametro uv.
* **surf\_spline\_kurHK\_plot(surfspline, ni, nj, cflag)**:  
  Disegna una superficie spline 3D definita dalla struttura surfspline, calcolando e visualizzando la curvatura media o gaussiana in base al valore di cflag, utilizzando una griglia di valutazione ni×njni \times njni×nj.
* **surf\_spline\_load(filename)**:  
  Carica una superficie spline 3D da un file .db.
* **surf\_spline\_plot(surfspline, ni, nj, type, varargin)**:  
  Disegna una superficie spline 3D definita dalla struttura surfspline utilizzando una griglia di valutazione ni×njni \times njni×nj e il tipo di disegno type.

### **Funzioni di Trasformazione Geometrica**

* **gc\_translate(vect, dist)**:  
  Trasla un vettore vect di una distanza dist nella direzione di vect.

### **Algoritmi e Funzionalità Aggiuntive**

* **INTERX(L1, L2)**:  
  Calcola i punti di intersezione tra due curve 2D L1 e L2 definite da matrici di coordinate.
* **offset(x, y, offset, haxes, intersectremove)**:  
  Calcola l'offset di una curva 2D definita da x e y di una distanza offset.
* **point\_plot(p,varargin):** Disegna punti p, liste di punti o griglie di punti 2D e 3D
* **function vect2\_plot(p,v,varargin):** Disegna il vettore 2D definito dal punto 2D p e dal vettore v
* **function [IP1P2,t1,t2]=curv2\_intersect(P1,P2):**Determina i punti di intersezione delle curve P1 e P2 e i loro parametri.Questa function prende in input due curve in forma parametrica di tipo Bézier, ppBézier, mdppBézier, spline e nurbs. Se viene chiamata con un solo argomento (una sola curva), vengono calcolate le auto intersezioni e i loro parametri.

P1, P2 --> strutture delle curve in input dei tipi dopra elencati

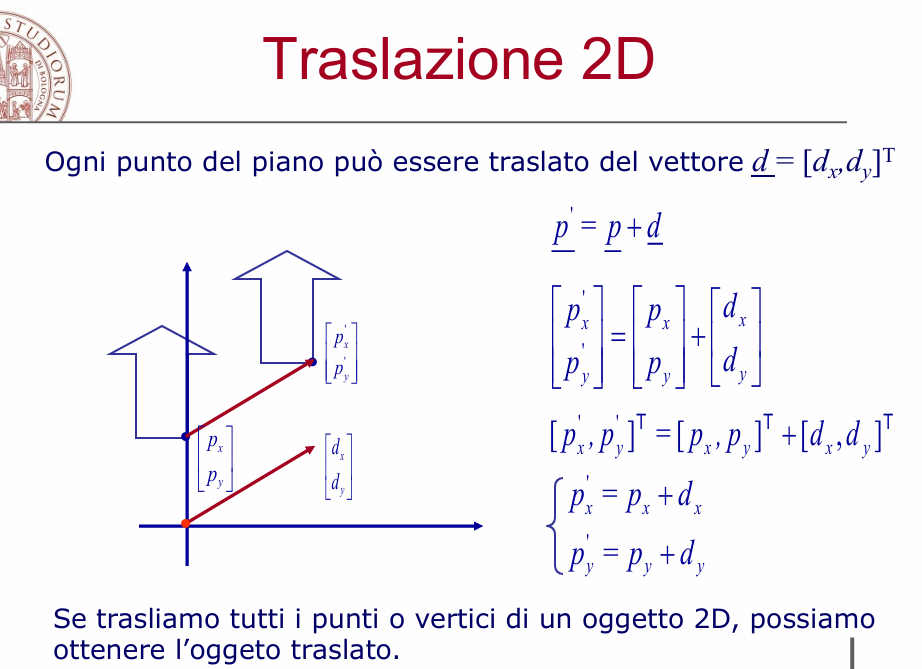
IP1P2 <-- struttura 2xn contenente le n intersezioni trovate

t1, t2 <-- array contenenti i parametri dei punti di intersezione rispettivamente per la prima e seconda curva

# **Traslazione, Scala e Rotazione**

**Traslazione**

- Data una matrice di punti [x,y] (2\*n) la sommo a un vettore di traslazione.



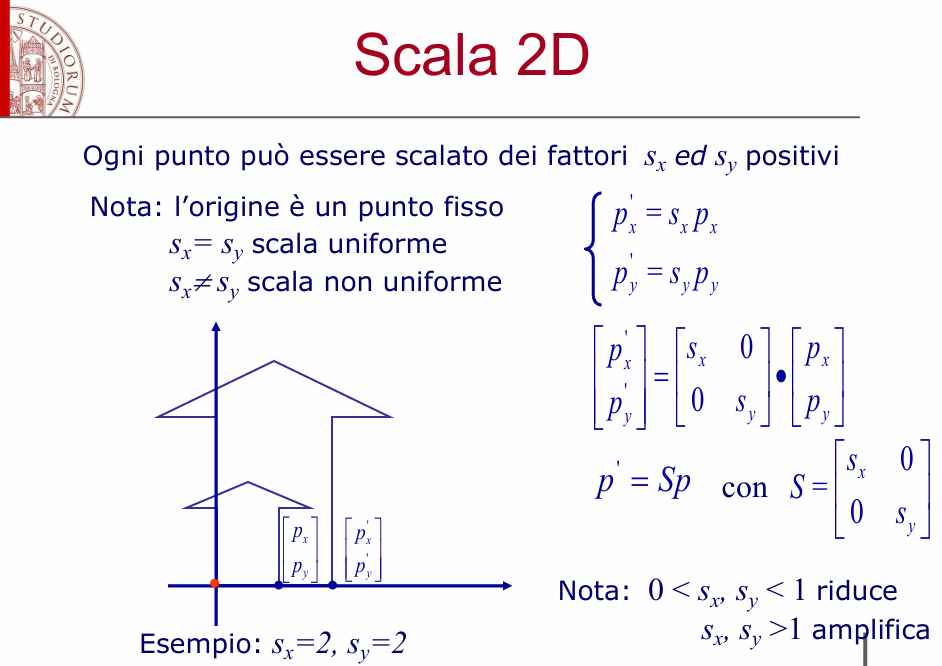
esempio con curva di Bez

-bezT.cp=bez.cp+[5,8];

**Scala**

-Data una metrice di punti [x,y] (2\*n) la moltiplico per un vettore [n1,n2];

-Ricordarsi di mettere il punto nel prodotto (.\*)

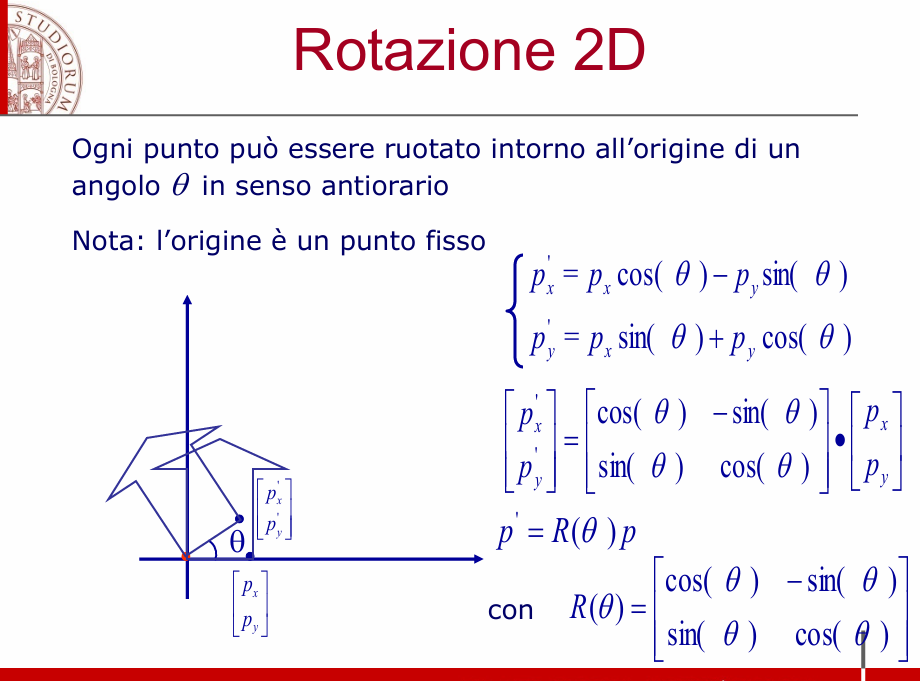


esempio di curva con curva di Bez

-bezS.cp = bez.cp.\*[2,2]

**Ruotata**

-Data una matrice di punti [x,y] (2\*n) la moltiplico per la matrice [cos(q),sin(q);-sin(q),cos(q)] dove q è l’angolo di rotazione.



Esempi con le curve di Bez

-bezR.cp = bez.cp\*[cos(q),sin(q);-sin(q),cos(q)] ;

Se voglio traslare il punto fisso di rotazione aggiungo il vettore di traslazione:

-bezR.cp = bez.cp\*[cos(q),sin(q);-sin(q),cos(q)]-[9,5] ;

**Note tratti curvati:**

Contesto: Sto facendo un esercizio dove devo fare un cuore, partendo dai punti di interpolazione, noto che se metto tanti punti il cuore diventa più curvato.

Forse perchè in questo caso è una curva con delle rientranze, tipo S. Concludo che:

Se si vuole fa una curva tipo S è meglio fare una curva a tratti, se si vuole fare una cosa più specifica mettere più punti (complesso perchè non so disegnare coi punti), in generale appena si fa un arco consiglio per andare avanti fare un altra curva, per evitare curvature impreviste.Di conseguenza utilizzre join.

**Note rotazione attorno a punto diverso da origine:**

Guarda fiore.m oppure ssquare\_rot.m della simulazione 3.

Noto che il metodo di rotazione utilizzato in questo file è molto meglio di quello che utilizzavo in precedenza.

Spiegazione del nuovo metodo:

%trovo il centro della crf

%potevo trovare il baricentro della crf ma essendo che ci sono pochi punti

%il baricentro viene con impreciso sia nei casi in cui considero i punti

%.cp che quelli della crf

%B=mean(bPP.cp(1:4,:));

B=[1 0];

%definisce matrice di traslazione

T=get\_mat\_trasl(-B);

Tinv=get\_mat\_trasl(B);

%definisce angolo alfa di rotazione

alfa=pi/4

%definisce matrice di rotazione usando la get\_mat2\_rot

R=get\_mat2\_rot(alfa)

%definisce matrice composta di rotazione rispetto al baricentro

M=Tinv\*R\*T

%rotazione attorno a un punto tipo mandala applicato alle curve di bez

ppbFoR=ppbFo;

for i=0:pi/3.5:2\*pi

R=get\_mat2\_rot(i);

M=Tinv\*R\*T;

ppbFoR.cp=point\_trans(ppbFo.cp,M);

curv2\_ppbezier\_plot(ppbFoR,60,'k-');

end

Se si volesse cambiare la distanza dal punto devo traslare la matrice originale (ppFo.cp) di conseguenza se volessi allontanare i miei punti di [9 0] farò:

ppbFoR.cp=point\_trans(ppbFo.cp+[9 0],M);

**Usare Control-Maiusc-D per mettere la finestra nel Worksapce**

**Meto di per calcolare le lunghezza di curva di Bez anche a tratti(ppP)**

**%disegno della curva a tratti**

**curv2\_ppbezier\_plot(ppP,50,'b-',2);**

**nt=length(ppP.ab)-1;**

**[ncp,~] = size(ppP.cp);**

**val = 0;**

**%estrae le singole curve di Bezier e calcola la loro lunghezza**

**bez.deg=ppP.deg;**

**for i=1:nt**

**vcol=col(mod(i,7)+1);**

**i1=(i-1)\*ppP.deg+1;**

**i2=i1+ppP.deg;**

**bez.cp=ppP.cp(i1:i2,:);**

**bez.ab=[ppP.ab(i),ppP.ab(i+1)];**

**curv2\_ppbezier\_plot(bez,50,’r’,2);**

**val =val+integral(@(x)norm\_c1\_val(bez,x),bez.ab(1),bez.ab(2));**

**end**

**fprintf('Lunghezza della curva: %e\n',val);**

**function val = norm\_c1\_val(bezier,t)**

**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%**

**%Calcola il valore della funzione ||C'(t)|| con C(t) curva di Bezier 2D**

**%bezier --> struttura di una curva di Bezier:**

**% bezier.deg --> grado della curva**

**% bezier.cp --> lista dei punti di controllo (bezier.deg+1)x2**

**% bezier.ab --> intervallo di definizione**

**%t --> valore/i parametrici in cui valutare**

**%val <-- valore/i della funzione ||C'(t)|| in corrispondenza di t**

**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%**

**Pt=decast\_valder(bezier,1,t);**

**%norm sta per norma**

**for i=1:length(t)**

**val(i)=norm([Pt(2,i,1),Pt(2,i,2)],2);**

**end**

**end**

**Calcolo dell’area di una curva di bez a tratti ppbBez**

**utilizzare la funzione curv2\_ppbezier\_area(ppP)**

**Aggiungere**

**val = curv2\_ppbezier\_area(ppP);**

**if val<0**

**val = -val;**

**end**

**fprintf('Area della curva: %e\n',val);**

**Direzioni delle curve di bez con la funzione curv2\_bezier\_tan\_plot**

**%uso la funzione della libreria curv2\_bezier\_tan\_plot**

**Px = curv2\_bezier\_tan\_plot(bezP,2,'r-',9)**

**% Px=decast\_valder(bezP,1,0)**

**vect2\_plot([Px(1,1,1),Px(1,1,2)],[Px(2,1,1),Px(2,1,2)],col(1),1,col(1),col(1),1,0.025);**

**val = curv2\_ppbezier\_area(ppP) %se è positiv a curva orientata in senso antiorario**

**vect2\_plot([Px(1,2,1),Px(1,2,2)],[Px(2,2,1),Px(2,2,2)],col(1),1,col(1),col(1),1,0.025);**

**Simmetria ali farfalla**

[ppbQ,T,R]=align\_curve (ppbezP) ;

%Cambia il segno delle coordinate y dei puntii di controllo della curva

%ppbQ. Questo crea una riflessione (simmetrice) della curva ppbQ rispetto

%all'asse x

ppbQ.cp(: ,2)=-ppbQ.cp(: ,2) ;

%Si calcola Minv, l'inversa della combinazione R\*T, che serve a riportare

%la curva ppbQ nella sua posizione originale, ma nella versione riflessa

Minv=inv(R\*T) ;

%funzione che applica questa trasformazione inversa a ppbQ, così che la

%curva rispecchiata mantenga la sua forma simmetrica, ma si riallinei alla

%posizione originale della curva ppbP

ppbQ.cp=point\_trans(ppbQ.cp ,Minv) ;

curv2\_ppbezier\_plot(ppbQ,np,'m-');

%disegna la curva riflessa con la linea verde

point\_plot(ppbQ.cp(1 ,:) ,'mo-')

function [ppbezQ,T,R]=align\_curve (ppbezP)

%numero totale di punti di controllo della curva

ncp=length(ppbezP.cp(: ,1) ) ;

ppbezQ=ppbezP;

%1 Traslazione

%T è una matrice di traslazione che sposta il primo punto della curva

%ppbezP, all'origine del sistema di coordinate.In pratic a, viene

%calcolato il vettore di spostamento che porta il primo punto (x1,y1)

%della curva a (0,0)

T=get\_mat\_trasl(-ppbezP.cp(1 ,:) ) ;

%alfa è l'angolo tra la linea che collega il primo punto e l'ultimo punto

%della curva e l'asse x. atan2 calcola l'angolo (in radianti) a partire

%dalle differenze y e x tra il punto finale e il punto iniziale. L'angolo

%è negativo per ruotare in senso orario, portando così il segmento

%iniziale-finale a essere orizzontale.

alfa = -atan2(ppbezP.cp(ncp ,2) - ppbezP.cp(1 ,2) , ppbezP.cp(ncp ,1) - ppbezP.cp(1 ,1)) ;

%R è una matrice di rotazione che ruota i punti della curva di alfa

%radianti, allineando il segmento tra primo e ultimo con l'asse x.

%Definisce la matrice 3x3 di rotazione 2D in base all'angolo alfa in input

R=get\_mat2\_rot(alfa ) ;

%La matrice M combina la traslazione T e la rotazione R. Viene poi

%applicata alla curva ppbezP per trasformare tutti i suoi punti di

%controllo, ottenendo ppbezQ, una curva allineata orizzontalmente e pronta

%per l'operazione di simmetria

M=R\*T;

%point\_trans calcola le lista di punti 2D dopo averli trasformati con la

%matrice M ( matrice di trasformazione)

ppbezQ.cp=point\_trans(ppbezP.cp ,M) ;

end

**Trovare il massimo delle y dei una curva di bez a tratti/ bez**

**a=0;**

**b=1;**

**npt=150;**

**t=linspace(a,b,npt);**

**Pxy=ppbezier\_val(bezCuo,t);**

**point\_plot(Pxy,'m',9);**

**Massimay = max(Pxy(:,2));**

**Minimay = min(Pxy(:,2));**

**Calcolo dell’errore circonferenza**

%calcolo errore

%valutazione in punti equispaziati per test sull'errore

npt=150;

t=linspace(0,2\*pi,npt);

Pxy=ppbezier\_val(ppbezCrf,t);

%si fa solo per crf, per casi normali utilizzare quella non der

[x,y]=cp2\_circle(t);

Qxy=[x',y'];

%calcola la distanza euclidea fra i punti della curva test

%e della curva di Bézier a tratti interpolante e considera

%la distanza massima

MaxErr=max(vecnorm((Pxy-Qxy)'));

fprintf('MaxErr3: %e\n',MaxErr);

**Traslazione dal centro (utile per mandala in caso che il primo punto non tocchi l’origine ) es. mandala cuori ( caso in cui la fila di cuori non tocca l’origine)**

%traslazione dal centro

C = ppG.cp(1,:);

T = get\_mat\_trasl(-C);

T1 = get\_mat\_trasl([0 1.05]);

M = T\*T1;

ppG.cp = point\_trans(ppG.cp,M);

curv2\_ppbezier\_plot(ppG,60,'r');

**Intersezione**

[IP1P2,t1,t2]=curv2\_intersect(ppbezCrf,bezSeg2);

[sx11,dx11]=ppbezier\_subdiv(ppbezCrf,t1(1));

[sx12,dx12]=ppbezier\_subdiv(sx11,t1(2));

% curv2\_ppbezier\_plot(sx11,60,'g',2);

% curv2\_ppbezier\_plot(dx11,60,'b',2);

[sx21,dx21]=ppbezier\_subdiv(bezSeg2,t2(1));

[sx22,dx22]=ppbezier\_subdiv(sx21,t2(2));

% curv2\_ppbezier\_plot(sx21,60,'g',2);

% curv2\_ppbezier\_plot(dx21,60,'b',2);

Osservazione su get\_mat\_trasl(bez.cp,d) trasla il Baricentro di bez.cp di un vettore d, attenzione lo somma non lo mette in quel punto. Se voglio il baricentro in [2 2] e sono in [1 0] devo mettere d=[1 2];