Opérations élémentaires sur courbes splines cubiques en deux dimensions

Alain Batailly

Note technique

Opérations élémentaires sur courbes splines en 2D

Alain Batailly

Université McGill

Laboratoire de dynamique des structures et vibrations

février 2012

Introduction

Cette note technique a pour objectif de fournir une description détaillée des opérations élémentaires concernant les courbes B-splines cubiques en 2D. Cette note contient notamment tous les codes Matlab nécessaires à la réalisation des exemples présentés. Ces codes sont généraux et permettent de calculer une B-spline cubique pour tout ensemble de points **P** du plan.

Création d'une courbe B-spline cubique

Présentation du problème

Soit un ensemble de n points $\mathbf{P}_\mathrm{i}$, $i = 1 \ldots n$ du plan dont les coordonnées sont regroupées dans une matrice **P** de dimension $(n \times 2)$ telle que :

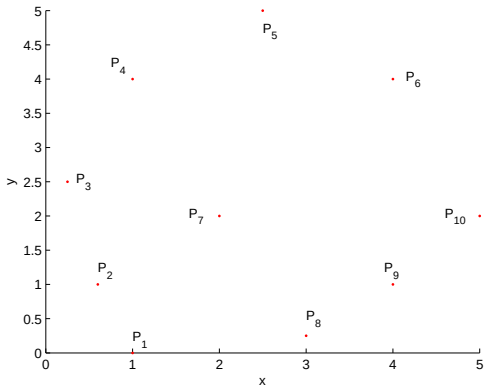
- la ligne i de la première colonne contient l'abscisse $x(\mathrm{P_i})$;
- la ligne i de la deuxième colonne contient l'ordonnée $y(\mathrm{P_i})$.

Nous nous intéressons à la création de la B-spline cubique passant par les points $\mathrm{P_i}$, $i = 1 \ldots n$ en respectant leur ordre dans la matrice **P** : $\mathrm{P_1}$ puis $\mathrm{P_2}$ ... jusqu'à $\mathrm{P_n}$. Par exemple, pour le nuage de points

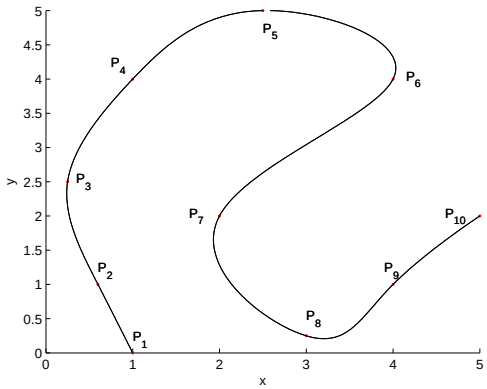
$$\mathbf{P} = \begin{bmatrix} 1 & 0 \\ 0.6 & 1 \\ 0.25 & 2.5 \\ 1 & 4 \\ 2.5 & 5 \\ 4 & 4 \\ 2 & 2 \\ 3 & 0.25 \\ 4 & 1 \\ 5 & 2 \end{bmatrix} \tag{1}$$

représenté sur la figure 1(a) on cherche à obtenir la B-spline cubique noire représentée sur la figure 1(b). Dans la suite de cette note, on utilise la notation suivante

$$\mathbf{P}_\mathrm{i} = [x(\mathrm{P_i})\ y(\mathrm{P_i})] \tag{2}$$



(a) Nuage de points **P**



(b) B-spline cubique passant par **P**

Rappels mathématiques

Une fonction spline polynômiale est une courbe paramétrique $\mathbf{c}(t)$ obtenue en multipliant une base de fonctions splines $\mathbf{B}_\mathrm{i}$ à un ensemble de points de contrôle $\mathbf{Q}_\mathrm{i}$, $i = 0 \dots n-1$:

$$\mathbf{c}(t) = \sum_{i=0}^{n-1} \mathbf{Q}_\mathrm{i} \mathbf{B}_\mathrm{i}(t) \tag{3}$$

Le degré des fonctions contenues dans la matrice $\mathbf{B}$ est ici fixé à 3, d'où le terme de B-spline *cubique*, ce qui permet d'obtenir la continuité de la normale sur toute la spline. Ceci est très utile lorsque la spline est utilisée dans le contexte d'un algorithme de contact par exemple.

La B-spline cubique est composée de $n-1$ segments $[\mathbf{P}_\mathrm{i}\mathbf{P}_{\mathrm{i}+1}]$, $i = 1 \dots n-1$. Le paramétrage intrinsèque t de ces segments est défini de façon à ce que $\mathbf{P}_\mathrm{i} = \mathbf{c}_\mathrm{i}(t=0)$ et $\mathbf{P}_{\mathrm{i}+1} = \mathbf{c}_\mathrm{i}(t=1)$, on parle alors d'une *paramétrisation uniforme*. Pour chaque segment i de la courbe B-spline cubique, on peut écrire

$$\mathbf{c}_\mathrm{i}(t) = \frac{1}{6} \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{\mathrm{i}\text{-}1} \\ \mathbf{Q}_\mathrm{i} \\ \mathbf{Q}_{\mathrm{i}+1} \\ \mathbf{Q}_{\mathrm{i}+2} \end{bmatrix} \tag{4}$$

Dans la suite de cette note, nous utiliserons la notation

$$\mathbf{A} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{5}$$

Par ailleurs, les dérivées première et seconde de $\mathbf{c}_\mathrm{i}(t)$ sont

$$\begin{aligned} \dot{\mathbf{c}}_\mathrm{i}(t) &= \frac{1}{6} \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -3 & 9 & -9 & 3 \\ 6 & -12 & 6 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{\mathrm{i}\text{-}1} \\ \mathbf{Q}_\mathrm{i} \\ \mathbf{Q}_{\mathrm{i}+1} \\ \mathbf{Q}_{\mathrm{i}+2} \end{bmatrix} \\ \ddot{\mathbf{c}}_\mathrm{i}(t) &= \frac{1}{6} \begin{bmatrix} t & 1 \end{bmatrix} \begin{bmatrix} -6 & 18 & -18 & 6 \\ 6 & -12 & 6 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{\mathrm{i}\text{-}1} \\ \mathbf{Q}_\mathrm{i} \\ \mathbf{Q}_{\mathrm{i}+1} \\ \mathbf{Q}_{\mathrm{i}+2} \end{bmatrix} \end{aligned} \tag{6}$$

et nous utiliserons les notations suivantes

$$\dot{\mathbf{A}} = \begin{bmatrix} -3 & 9 & -9 & 3 \\ 6 & -12 & 6 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \quad \text{et} \quad \ddot{\mathbf{A}} = \begin{bmatrix} -6 & 18 & -18 & 6 \\ 6 & -12 & 6 & 0 \end{bmatrix} \tag{7}$$

Pour la construction de la spline représentée sur la figure 1(b) – c'est à dire la connaissance explicite de tout point $\mathbf{c}_\mathrm{i}(t)$, $i = 1 \dots n-1$ – il ne reste plus qu'à déterminer la matrice des points de contrôle $\mathbf{Q}$ à partir de la matrice des points $\mathbf{P}$.

Points de contrôle

Dans le cas d'une paramétrisation uniforme de la B-spline cubique $\mathbf{P}_\mathrm{i} = \mathbf{c}_\mathrm{i}(t=0)$, ainsi en utilisant l'équation (4), on obtient :

$$\begin{aligned} \mathbf{P}_\mathrm{i} &= \mathbf{c}_\mathrm{i}(0) = \frac{1}{6}(\mathbf{Q}_{\mathrm{i}\text{-}1} + 4\mathbf{Q}_\mathrm{i} + \mathbf{Q}_{\mathrm{i}+1}) \\ \mathbf{P}_{\mathrm{i}+1} &= \mathbf{c}_\mathrm{i}(1) = \mathbf{c}_{\mathrm{i}+1}(0) = \frac{1}{6}(\mathbf{Q}_\mathrm{i} + 4\mathbf{Q}_{\mathrm{i}+1} + \mathbf{Q}_{\mathrm{i}+2}) \end{aligned} \tag{8}$$

Ce qui peut s'écrire sous la forme matricielle suivante

$$\begin{pmatrix} \mathbf{P}_{\text{CLg}} \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \\ \vdots \\ \\ \mathbf{P}_{\text{n-1}} \\ \mathbf{P}_{\text{n}} \\ \mathbf{P}_{\text{CLd}} \end{pmatrix} = \frac{1}{6} \begin{bmatrix} \text{Condition Limite } \textit{gauche} \\ 1 \;\; 4 \;\; 1 \;\; 0 \quad \dots \quad\quad 0 \\ 0 \;\; 1 \;\; 4 \;\; 1 \;\; 0 \;\; \dots \quad\quad 0 \\ \\ \vdots \quad\quad\quad\quad\quad\quad\quad \vdots \\ \\ 0 \;\; \dots \;\; 0 \;\; 1 \;\; 4 \;\; 1 \quad\quad 0 \\ 0 \;\; \dots \quad\quad 0 \;\; 1 \;\; 4 \quad 1 \\ \text{Condition Limite } \textit{droite} \end{bmatrix} \begin{pmatrix} \mathbf{Q}_0 \\ \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \\ \vdots \\ \\ \mathbf{Q}_{\text{n-1}} \\ \mathbf{Q}_{\text{n}} \\ \mathbf{Q}_{\text{n+1}} \end{pmatrix} \tag{9}$$

dans laquelle apparaîssent deux points supplémentaires dans la matrice $\mathbf{P}$: $\mathbf{P}_{\text{CLg}}$ et $\mathbf{P}_{\text{CLd}}$ qui, comme nous le verrons dans la section suivante, permettent le paramétrage des conditions limites *à gauche* et *à droite*, c'est à dire à chaque extrémité de la B-spline cubique.

L'équation (15) s'écrit plus simplement

$$\mathbf{P} = \frac{1}{6}\mathbf{\Phi}\mathbf{Q} \tag{10}$$

et les points de contrôle $\mathbf{Q}$ sont simplement obtenus en calculant

$$\mathbf{Q} = 6\mathbf{\Phi}^{-1}\mathbf{P} \tag{11}$$

Conditions limites

Afin de comprendre l'importance des conditions limites, considérons la matrice $\mathbf{P}$ suivante :

$$\mathbf{P} = \begin{bmatrix} 3 & 0.25 \\ 0.6 & 1 \\ 0.25 & 2.5 \\ 1 & 4 \\ 2.5 & 5 \\ 4 & 4 \\ 5 & 2 \\ 4 & 1 \\ 3 & 0.25 \end{bmatrix} \tag{12}$$

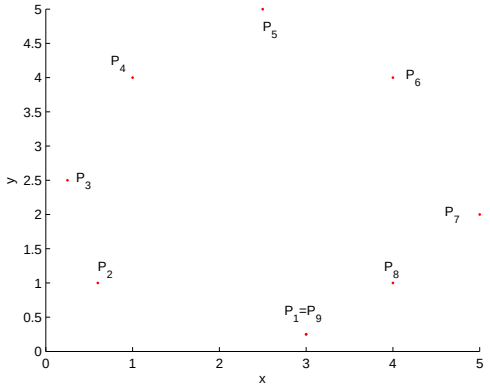
Le tracé de ces points sur la figure 1(c) met en évidence que le premier et le dernier point sont confondus. On peut alors envisager une infinité de conditions de raccordement (B-splines cubiques noires tracées sur la figure 1(d)) et notamment une condition de continuité de la normale à la spline en passant du point P_9 au point P_1 (B-spline cubique rouge sur la figure 1(d)).

Le choix des conditions limites se fait par l'intermédiaire des première et dernière lignes de l'équation matricielle (15). Par exemple, si on désire imposer une condition de continuité telle que celle qui permet de tracer la B-spline cubique rouge, c'est à dire si on désire que

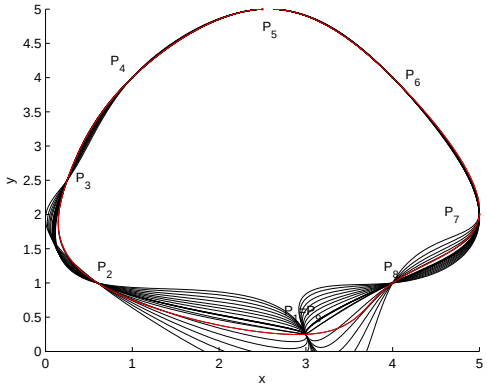
$$\begin{aligned} \dot{\mathbf{c}}_0(t=0) &= \dot{\mathbf{c}}_{\text{n-1}}(t=1) \\ \ddot{\mathbf{c}}_0(t=0) &= \ddot{\mathbf{c}}_{\text{n-1}}(t=1) \end{aligned} \tag{13}$$

En utilisant l'équation (4), on peut facilement montrer que la condition (13) est équivalente à

$$\begin{aligned} -\mathbf{Q}_0 + \mathbf{Q}_2 &= -\mathbf{Q}_{\text{n-1}} + \mathbf{Q}_{\text{n}} \\ \mathbf{Q}_0 - 2\mathbf{Q}_1 + \mathbf{Q}_2 &= \mathbf{Q}_{\text{n-1}} - 2\mathbf{Q}_{\text{n}} + \mathbf{Q}_{\text{n+1}} \end{aligned} \tag{14}$$



(c) Nuage de points **P**



(d) B-splines cubiques passant par **P**

La relation entre points de données **P** et points de contrôle **Q** s'écrit alors

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \vdots \\ \mathbf{P}_{n-1} \\ \mathbf{P}_n \\ \mathbf{0} \end{pmatrix} = \frac{1}{6} \begin{bmatrix} -1 & 0 & 1 & 0 & \dots & 1 & 0 & -1 \\ 1 & 4 & 1 & 0 & & \dots & & 0 \\ 0 & 1 & 4 & 1 & 0 & \dots & & 0 \\ \vdots & & & & & & & \vdots \\ 0 & \dots & 0 & 1 & 4 & 1 & & 0 \\ 0 & \dots & & & 0 & 1 & 4 & 1 \\ 1 & -2 & 1 & 0 & \dots & -1 & 2 & -1 \end{bmatrix} \begin{pmatrix} \mathbf{Q}_0 \\ \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \vdots \\ \mathbf{Q}_{n-1} \\ \mathbf{Q}_n \\ \mathbf{Q}_{n+1} \end{pmatrix} \tag{15}$$

On peut bien évidemment envisager tout autre type de condition limite (bords libres, angle droit...) suivant l'application visée.

Code Matlab

Le code donné dans cette section permet de tracer la B-spline cubique avec condition de continuité associée à la matrice des points de données **P** et reprend les notations précédentes.

```
clear;clc;close all;
%
% Matrice des points de données
P=[ 3 0.25;
    0.6 1;
    0.25 2.5;
    1 4;
    2.5 5;
    4 4;
    5 2;
    4 1;
    3 0.25];
% Nombre de points
n=length(P);
%
% Création de la matrice de passage Phi
```

```
Phi=zeros(n+2,n+2);
k=1;
for i=2:n+1
    Phi(i,k)=1;
    Phi(i,k+1)=4;
    Phi(i,k+2)=1;
    k=k+1;
end
%
% extension du vecteur de points (pour conditions limites)
P_=zeros(n+2,2);
P_(2:n+1,:)=P;
%
% Conditions limites de continuité (par exemple)
Phi(1,1)=-1;Phi(1,3)=1;Phi(1,n)=1;Phi(1,n+2)=-1;
Phi(n+2,n)=-1;Phi(n+2,n+1)=2;Phi(n+2,n+2)=-1;
Phi(n+2,1)=1;Phi(n+2,2)=-2;Phi(n+2,3)=1;
%
% Matrice des points de contrôle
Q=zeros(n+2,2);
Q=6*inv(Phi)*P_;
%
% Tracé des points et des points de contrôle
figure(1);hold on;
plot(P(:,1),P(:,2),'.r');
xlabel('x');ylabel('y');
%
% Paramètre intrinsèque t
A=[-1 3 -3 1;
    3 -6 3 0;
    -3 0 3 0;
    1  4 1 0];
%
k=1;
% obtention de c(t) tout au long de la B-spline cubique
%
for i=1:n-1 % itération sur tous les segments
    %
    for t=[0:0.01:1] % échantillonage choisi: 101 points
        %
        x=1/6*[t^3 t^2 t 1]*A*[Q(i,1);Q(i+1,1);Q(i+2,1);Q(i+3,1)];
        y=1/6*[t^3 t^2 t 1]*A*[Q(i,2);Q(i+1,2);Q(i+2,2);Q(i+3,2)];
        X(:,k)=x;
        Y(:,k)=y;
        k=k+1;
    end
end
%
% tracé
figure(1);hold on;
plot(X,Y,'-k');
```

Projection sur B-spline cubique

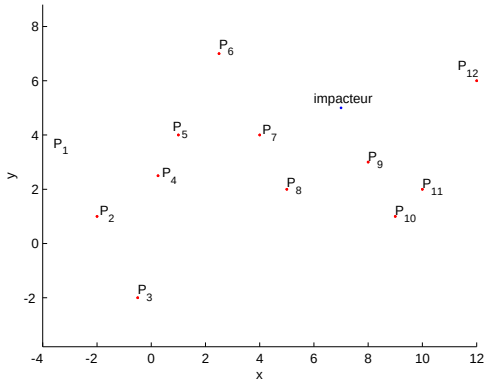
Outre l'obtention des coordonnées de tout point $\mathbf{c}_i(t)$ d'une spline bicubique, un problème fréquemment rencontré dans la littérature concerne la projection d'un point sur une spline. Bien que très similaires, nous distinguerons deux types de projection dans cette note :

- la projection orthogonale, qui est en fait, pour un point **M** quelconque du plan, la recherche de la distance minimale entre ce point et la spline S. On recherche le point **N** appartenant à la B-spline cubique tel que $\|\mathrm{MN}\| = \min_{\mathbf{c}_i(t)\in \mathrm{S}} \|\mathrm{M}\mathbf{c}_i(t)\|$
- la projection dans une direction quelconque : pour un point **M** du plan et un coefficient directeur a donné, on recherche le lieu de l'intersection entre la B-spline cubique et la droite de coefficient directeur a passant par **M**.

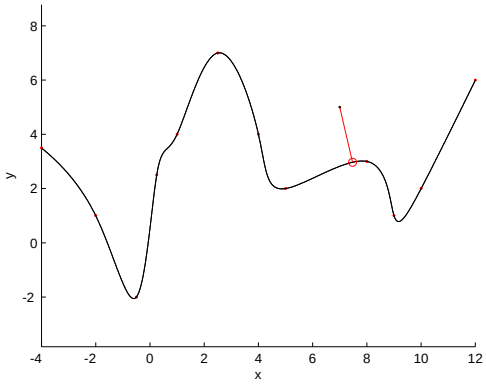
Des codes Matlab accompagnent chaque section.

Projection orthogonale

Considérons un ensemble **P** de points du plan tel que celui représenté sur la figure 1(e). La procédure décrite dans cette section permet tout d'abord d'obtenir la B-spline cubique tracée sur la figure 1(f) en utilisant la méthode présentée dans les sections précédentes simplement sous la forme d'une fonction Matlab. Une fois la B-spline cubique obtenue, un algorithme de type Newton-Raphson est utilisé pour



(e) Nuage de points **P**



(f) B-splines cubiques passant par **P**

déterminer, sur celle-ci, le point le plus proche d'un point *impacteur* du plan.

Le fichier principal fait appel à deux fonctions différentes et il y a donc trois fichiers Matlab fournis :

1. Le fichier principal

```
clear;
close all;
clc;
%
P=[ -4 3.5;
    -2 1;
    -0.5 -2;
    0.25 2.5;
    1 4;
    2.5 7;
    4 4;
    5 2;
    8 3;
```

```
    9 1;
    10 2;
    12 6];
%
figure(1);
axis equal
% calcul de la spline associée à la matrice P
spline=FCT_spline(P(:,1)',P(:,2)');
% coordonnées de l'impacteur
impx=7;
impy=5;
% détermination du point le plus proche de l'impacteur sur la spline
[x,y,dist,t,elem,coeff_dir]=FCT_projection_spline(spline,impx,impy);
% le point le plus proche a pour coordonnées "x" et "y", il se situe à
% la distance "dist" de l'impacteur, et appartient à  l'élément numéro
% "elem".  Le coefficient directeur de  la normale à la spline en  ce
% point est "coeff_dir". "t" est le paramètre intrinsèque de ce point.
```

2. La fonction permettant de créer la spline

```
function spline=FCT_spline(cox,coy);
%
% saisie des points pas lesquels la spline doit passer
P=[cox' coy'];
spline.coord=P;
%
% nombre de points
n=length(P);
%
% initialisation de la matrice de passage points/points de contrôle
spline.Phi=zeros(n+2,n+2);
% corps de la matrice
k=1;
for i=2:n+1
    spline.Phi(i,k)=1;
    spline.Phi(i,k+1)=4;
    spline.Phi(i,k+2)=1;
    k=k+1;
end
% extension du vecteur de points
P_=zeros(n+2,2);
P_(2:n+1,:)=P;
%
% conditions limites de bord libre
spline.Phi(2,1)=0;    spline.Phi(2,2)=5;
spline.Phi(1,1)=1;    spline.Phi(1,2)=-1;
%
spline.Phi(n+1,n+1)=5;    spline.Phi(n+1,n+2)=0;
spline.Phi(n+2,n+1)=-1;    spline.Phi(n+2,n+2)=1;
%
```

```
% matrice des points de contrôle
spline.Q=zeros(n+2,2);
spline.Q=6*inv(spline.Phi)*P_;
%
% tracé des points et des points de données
figure(2);
hold on;
plot(P(:,1),P(:,2),'.r');
xlabel('x');ylabel('y');
% tracé des points de contrôle
% hold on;
% plot(spline.Q(:,1),spline.Q(:,2),'.b');
%
figure(1);
hold on;
plot(P(:,1),P(:,2),'.r');
xlabel('x');ylabel('y');
%
% paramètre intrinsèque t
spline.A=[-1  3 -3 1;
           3 -6  3 0;
          -3  0  3 0;
           1  4  1 0];
%
% tracé de la spline
k=1;
for i=1:n-1 % itération sur les éléments
    for t=[0:0.01:1] % échantillonage choisi
        x=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(i,1);spline.Q(i+1,1);...
                                      spline.Q(i+2,1);spline.Q(i+3,1)];
        y=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(i,2);spline.Q(i+1,2);...
                                      spline.Q(i+2,2);spline.Q(i+3,2)];
        X(:,k)=x;
        Y(:,k)=y;
        k=k+1;
    end
end
figure(1);
hold on;
plot(X,Y,'-k');
```

3. La fonction permettant de déterminer le point le plus proche de l'*impacteur* sur la spline

```
function [x,y,dist,t,elem,coeff_dir]=FCT_projection_spline(spline,impx,impy);
%
FCT_spline(spline.coord(:,1)',spline.coord(:,2)');
hold on;plot(impx,impy,'.k');
%
P=spline.coord;
n=length(P);
```

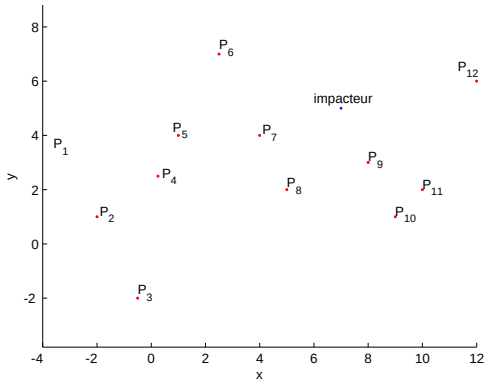
```
%
% point impacteur
Imp=[impx impy];
%
% dérivée de la matrice A pour le calcul de c'(t)
dA=[-3  9 -9 3;
     6 -12 6 0;
    -3  0  3 0];
% dérivée seconde de la matrice A pour le calcul de c''(t)
ddA=[-6  18 -18 6;
      6 -12  6  0];
%
% détection du point de donnée le plus proche de l'impacteur
DSPL=zeros(n,1);
for i=1:n
    DSPL(i,1)=sqrt((impy-P(i,2))^2+(impx-P(i,1))^2);
end
el_impact=find(DSPL==min(DSPL));
% on suppose qu'il ne peut y avoir qu'un seul point dans DSPL
el_impact=el_impact(1,1)
% correction éventuelle selon l'élément le plus proche
if el_impact==1
    elem=1;
else
    elem=el_impact-1;
end
%
t=-1;
elem=elem-1;
while ((t<0)&&(elem<(el_impact+2)))||((t>1)&&(elem<(el_impact+2)))
    elem=elem+1;
    k=1;
    t0=0;
    % valeur initiale du paramètre intrinsèque
    t=0.5;
    % coordonnées de l'impacteur
    xx=Imp(1,1);
    yy=Imp(1,2);
    % paramètre de convergence
    kmax=50;
    % test de convergence
    conv_stop=0;
    % boucle avec critère d'arrêt sur la convergence de t
    while ((abs(t-t0)>1e-12)&&(k<kmax))
        %
        % calcul des coordonnées du point de paramètre intrinsèque "t" sur
        % l'élément "elem"
        X1=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(elem,1);spline.Q(elem+1,1)...
            ;spline.Q(elem+2,1);spline.Q(elem+3,1)];
        X2=1/6*[t^2 t 1]*dA*[spline.Q(elem,1);spline.Q(elem+1,1);...
            spline.Q(elem+2,1);spline.Q(elem+3,1)];
```

```
            X3=1/6*[t 1]*ddA*[spline.Q(elem,1);spline.Q(elem+1,1);...
                spline.Q(elem+2,1);spline.Q(elem+3,1)];
            Y1=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(elem,2);spline.Q(elem+1,2)...
                ;spline.Q(elem+2,2);spline.Q(elem+3,2)];
            Y2=1/6*[t^2 t 1]*dA*[spline.Q(elem,2);spline.Q(elem+1,2);...
                spline.Q(elem+2,2);spline.Q(elem+3,2)];
            Y3=1/6*[t 1]*ddA*[spline.Q(elem,2);spline.Q(elem+1,2);...
                spline.Q(elem+2,2);spline.Q(elem+3,2)];
            %
            t0=t;
            % procédure de Newton-Raphson
            num=(X1-xx)*X2+(Y1-yy)*Y2;
            den=X2^2+Y2^2+(X1-xx)*X3+(Y1-yy)*Y3;
            t=t-(num)/(den);
            %
            % itération
            k=k+1;
            % test d'arrêt
            if k==kmax
                'pb de convergence'
                conv_stop=1
            end
        end
        % permet de s'assurer qu'en cas de non convergence, le paramètre
        % intrinsèque n'est pas considéré comme valide
        if (conv_stop==1)
            t=-1;
        end
    end
    %
    % coordonnées du point le plus proche détecté
    xi=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(elem,1);spline.Q(elem+1,1);...
        spline.Q(elem+2,1);spline.Q(elem+3,1)];
    yi=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(elem,2);spline.Q(elem+1,2);...
        spline.Q(elem+2,2);spline.Q(elem+3,2)];
    deriv=(1/6*[t^2 t 1]*dA*[spline.Q(elem,2);spline.Q(elem+1,2);...
        spline.Q(elem+2,2);spline.Q(elem+3,2)])/(1/6*[t^2 t 1]*dA...
        *[spline.Q(elem,1);spline.Q(elem+1,1);spline.Q(elem+2,1);...
        spline.Q(elem+3,1)]);
    coeff_dir=-1/deriv;
    x=xi;
    y=yi;
    % distance
    dist=sqrt((Imp(1,1)-xi)^2+(Imp(1,2)-yi)^2);
    %
    % tracé
    hold on;plot(xi,yi,'or')
    ord_org=yi-coeff_dir*xi;
    plot([impx xi],[impy yi],'-r')
    axis equal
```

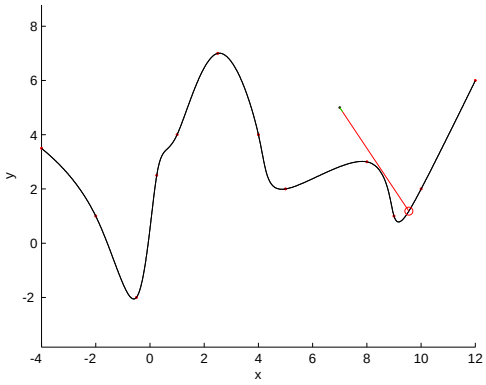
Projection dans une direction quelconque

Le problème n'est plus ici de déterminer le point le plus proche d'un *impacteur* du plan mais simplement de déterminer le lieu de l'intersection entre la B-spline cubique et la droite de pente a passant par un point *impacteur*.

Considérons le même nuage de points **P** que dans la section précédente (figure 1(g)) puis, par exemple, on s'intéresse à l'intersection entre la B-spline cubique et la droite de pente $a = -1,5$ passant par le point *impacteur* comme le montre la figure 1(h).



(g) Nuage de points **P**



(h) B-splines cubiques passant par **P**

De même que dans la section précédente, trois codes Matlab sont fournis :

1. Le fichier principal

```
clear;
close all;
clc;
%
P=[ -4 3.5;
    -2 1;
    -0.5 -2;
    0.25 2.5;
    1 4;
    2.5 7;
    4 4;
    5 2;
    8 3;
    9 1;
    10 2;
    12 6];
%
figure(1);
axis equal
% calcul de la spline associée à la matrice P
spline=FCT_spline(P(:,1)',P(:,2)');
% coordonnées de l'impacteur
impx=7;
impy=5;
% détermination de l'intersection
```

```
[x,y,dist,t,elem,coeff_dir]=FCT_projection_dirvar_spline(spline,...
impx,impy,-1.5);
% l'intersection a pour coordonnées "x" et "y", elle se situe à
% la distance "dist" de l'impacteur, et appartient à  l'élément numéro
% "elem".  Le coefficient directeur de  la droite passant par l'impacteur
% et l'intersection calculée est "coeff_dir". "t" est le paramètre
% intrinsèque de l'intersection.
```

2. La fonction permettant de créer la spline

```
function spline=FCT_spline(cox,coy);
%
% saisie des points pas lesquels la spline doit passer
P=[cox' coy'];
spline.coord=P;
%
% nombre de points
n=length(P);
%
% initialisation de la matrice de passage points/points de contrôle
spline.Phi=zeros(n+2,n+2);
% corps de la matrice
k=1;
for i=2:n+1
    spline.Phi(i,k)=1;
    spline.Phi(i,k+1)=4;
    spline.Phi(i,k+2)=1;
    k=k+1;
end
% extension du vecteur de points
P_=zeros(n+2,2);
P_(2:n+1,:)=P;
%
% conditions limites de bord libre
spline.Phi(2,1)=0;    spline.Phi(2,2)=5;
spline.Phi(1,1)=1;    spline.Phi(1,2)=-1;
%
spline.Phi(n+1,n+1)=5;    spline.Phi(n+1,n+2)=0;
spline.Phi(n+2,n+1)=-1;    spline.Phi(n+2,n+2)=1;
%
% matrice des points de contrôle
spline.Q=zeros(n+2,2);
spline.Q=6*inv(spline.Phi)*P_;
%
% tracé des points et des points de données
figure(2);
hold on;
plot(P(:,1),P(:,2),'.r');
xlabel('x');ylabel('y');
% tracé des points de contrôle
```

```
% hold on;
% plot(spline.Q(:,1),spline.Q(:,2),'.b');
%
figure(1);
hold on;
plot(P(:,1),P(:,2),'.r');
xlabel('x');ylabel('y');
%
% paramètre intrinsèque t
spline.A=[-1  3 -3 1;
           3 -6  3 0;
          -3  0  3 0;
           1  4  1 0];
%
% tracé de la spline
k=1;
for i=1:n-1 % itération sur les éléments
    for t=[0:0.01:1] % échantillonage choisi
        x=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(i,1);spline.Q(i+1,1);...
                                      spline.Q(i+2,1);spline.Q(i+3,1)];
        y=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(i,2);spline.Q(i+1,2);...
                                      spline.Q(i+2,2);spline.Q(i+3,2)];
        X(:,k)=x;
        Y(:,k)=y;
        k=k+1;
    end
end
figure(1);
hold on;
plot(X,Y,'-k');
```

3. La fonction permettant de déterminer l'intersection entre la droite de pente $a = -1,5$ passant par l'impacteur et la B-spline cubique

```
function [xi,yi,dist,t,elem,coeff_dir]=FCT_projection_dirvar_spline...
    (spline,xx,yy,pente);
%
FCT_spline(spline.coord(:,1)',spline.coord(:,2)');
hold on;plot(xx,yy,'.k');
%
P=spline.coord;
n=length(P);
%
% "dérivée" de la matrice A pour le calcul de c'(t)
dA=[-3 9 -9 3;
    6 -12 6 0;
    -3 0 3 0];
% "dérivée" seconde de la matrice A pour le calcul de c''(t)
ddA=[-6 18 -18 6;
    6 -12 6 0];
```

```
%
% HYPOTHESE => UNE SEULE INTERSECTION SPLINE / DROITE
position=spline.coord(:,2)-(pente*spline.coord(:,1)+(yy-pente*xx));
val_p=find(position>=0);
val_n=find(position<=0);
if (length(val_p)==0)     % tous les noeuds de la spline sont
                          % "au-dessous de la droite"
    elem=1;
elseif (length(val_n)==0) % tous les noeuds de la spline sont
                          % "au-dessus de la droite"
    elem=length(position)-1;
else                      % la droite coupe la spline
    elem=1;
    while ((position(elem,1)*position(elem+1,1))>0)
        elem=elem+1;
    end
end
%
k=1;
t0=0;
% valeur initiale du paramètre intrinsèque pour lancer les itérations
t=0.5;
% boucle avec critère d'arrêt sur la convergence de t
while (abs(t-t0)>1e-12)&&(k<150)
    %
    % coordonnées du point de paramètre intrinsèque "t" sur l'élément
    % "elem"
    xi=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(elem,1);spline.Q(elem+1,1);...
        spline.Q(elem+2,1);spline.Q(elem+3,1)];
    X2=1/6*[t^2 t 1]*dA*[spline.Q(elem,1);spline.Q(elem+1,1);...
spline.Q(elem+2,1);spline.Q(elem+3,1)];
    yi=1/6*[t^3 t^2 t 1]*spline.A*[spline.Q(elem,2);spline.Q(elem+1,2);...
        spline.Q(elem+2,2);spline.Q(elem+3,2)];
    Y2=1/6*[t^2 t 1]*dA*[spline.Q(elem,2);spline.Q(elem+1,2);...
spline.Q(elem+2,2);spline.Q(elem+3,2)];
    %
    % procédure de Newton Raphson
    num=yi-(pente*xi+(yy-pente*xx));
    den=Y2-pente*X2;
    t0=t;
    t=t-num/den;
    % itération
    k=k+1;
    %
end
%
coeff_dir=(yi-yy)/(xi-xx);
hold on;plot(xi,yi,'or')
ord_org=yi-coeff_dir*xi;
plot([xx xi],[yy yi],'-r')
plot([xx xx+0.1],[yy pente*0.1+yy],'-g')
```

```
    axis equal
    dist=sqrt((xx-xi)^2+(yy-yi)^2);
```