TP1 CAO Compte rendu

Constanza Corentin

Novembre 2021

1 Introduction

Le But de ce TP était de manipuler tout d'abord des polynomes de Bernstein pour constuire des courbes de Bézier puis de construire et manipuler des NURBS. Ce TP à été réaliser sous Matlab.

2 Échauffement : Courbes de Bézier

Nous devons écrire un programme où l'on saisis à la souris N points, et qui trace le polygone contenant ces points et la courbe de bezier associée à ces points de contrôle. Il s'écrit comme ceci :

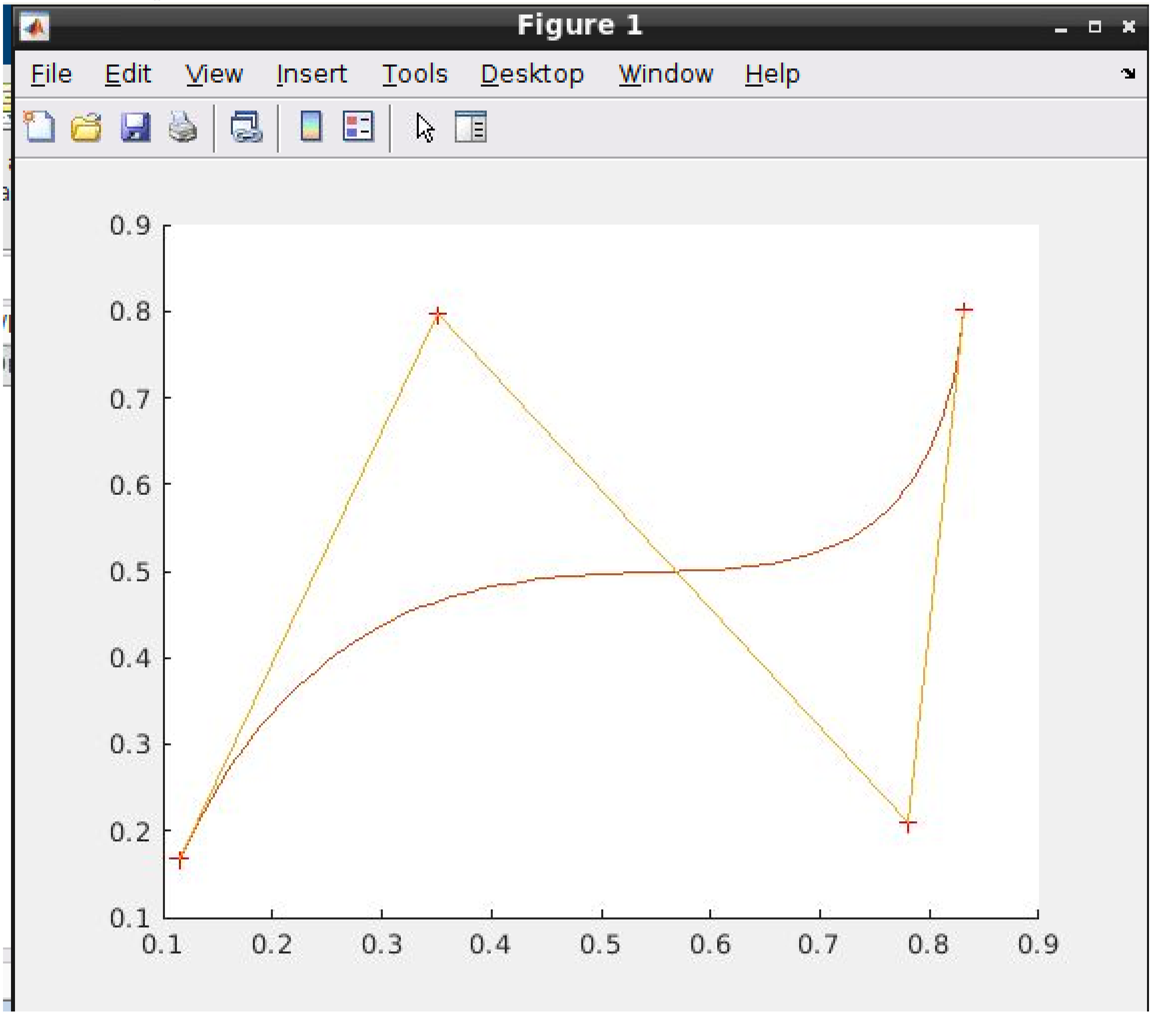
```
clear all;
close all;

figure %initialisation de la figure
hold on

[Px,Py]=ginput();
P=[Px,Py];
plot(Px,Py,'r+')

%discretisation
s = 1/100;
u = [0:s:1];
%calcul
C = CourbeB(P,u);
Ctr = C';

X␣=␣Ctr(1,:);
Y␣=␣Ctr(2,:);
plot(X,Y)
plot(P(:,1),P(:,2));
```

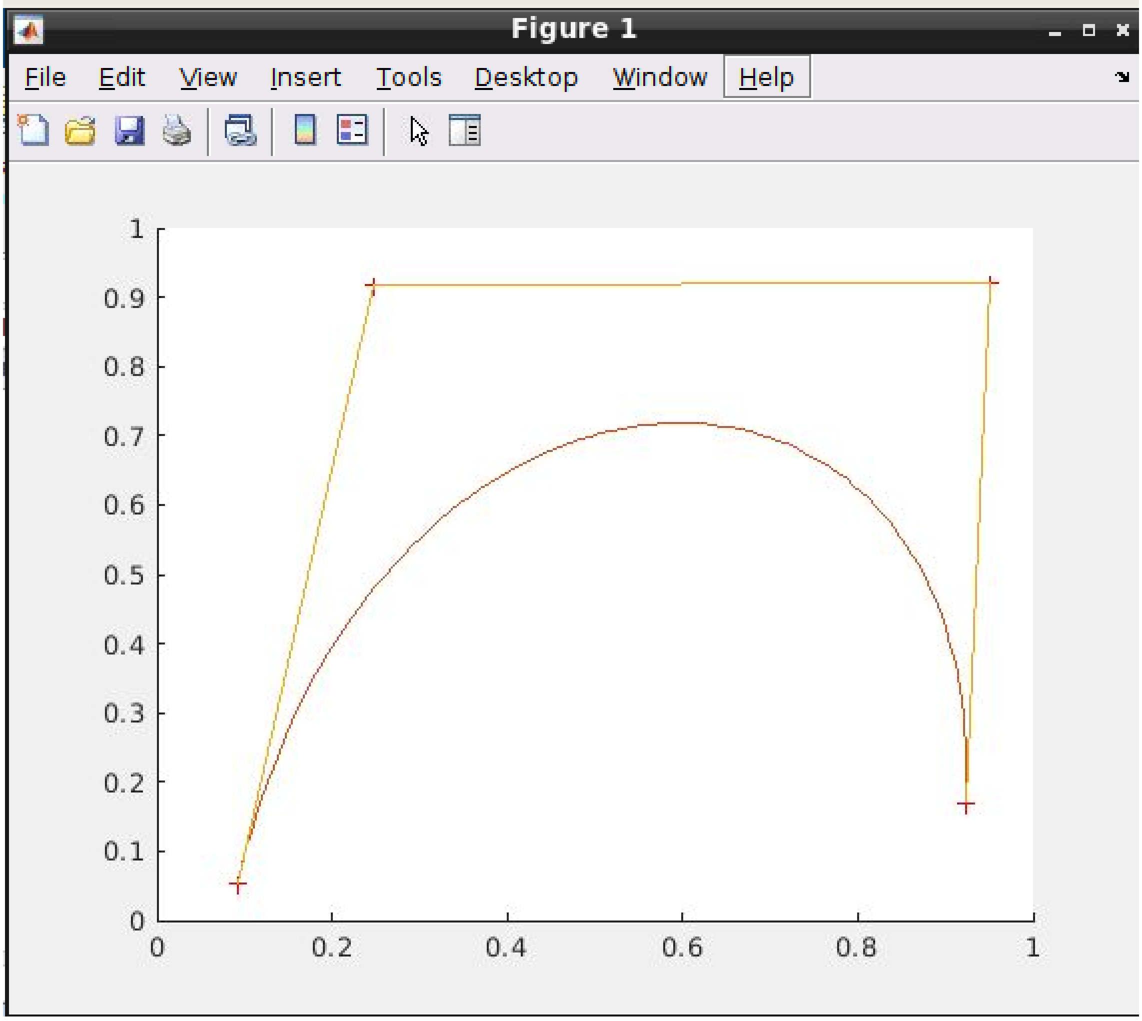
Les deux fonctions utilisées s'écrivent :

```
function [B] = Bernsteinscal(u,n)
B=zeros (n+1,1);
for k=0:n
    B(n+1)=u^k;
    for i = n:-1:k+1
        B(i)=B(i)*u + B(i+1)*(1-u);
    end
end
end

function [C]=CourbeB(P,u)

np = length(P);
nu = length(u);
C  = zeros(nu,2);

for i = 1:nu
```

```
    Bn = Bernsteinscal(u(i),np-1);
    for j = 1:np
        C(i,1) = C(i,1) + Bn(j)*P(j,1);
        C(i,2) = C(i,2) + Bn(j)*P(j,2);
    end
end
end
```

Voici deux exemple :





3 B-spline et NURBS

3.1 Fonction FindSpan

L'objectif de cette fonction est de savoir dans quel interval de la forme $[U(k), U(k+1)]$ se trouve u lorsque l'on construit notre courbe paramétré $C(u)$ (et que donc u varie sur chaques intervals $[U(k), U(k+1)]$). Il s'agit en réaliter d'une dichotomie. Voici le code utiliser :

```
function k = FindSand(p,u,U)
n=length(U)- p - 1; % n=m-p-1

if u==U(n)
  k=n;
  return
else
  inf = p+1;
  sup = n+1;
  x=floor((sup+inf)/2);
  while (u<U(x)) || (u>U(x+1))
        if (u<U(x))
        sup=x;
    else
        inf=x;
    end
    x=floor((sup+inf)/2);
end
k=x;
```

```
end
```

Éffectuons quelques test pour vérifier si notre fonction marche bien. Nous utiliserons le script suivant et ferons simplement varier la valeur de u :

```
clc
U=[0,0,0,0,0.25,0.5,0.75,1,1,1,1]
P=[[0,0]; [1,0] ;[1,1] ;[2,1]; [2.5,0]; [1.5,-0.5]; [0,-1/3]]
p=3;
u=0.6;
k = FindSand(p,u,U)
```

Voici les sorties des différents tests :
Pour $u = 0,6$, résultat attendu : $k = 6$

```
k =

     6
```

Pour $u = 0$, résultat attendu : $k = 4$

```
k =

     4
```

Pour $u = 80.0$, résultat attendu : $k = 7$

```
k =

     7
```

Cas limite : Pour $u = 1$, résultat attendu : $k = 7$ car les interval sont définie comme ci $u_i <= u < u_{i+1}$ Et donc par convention la borne superieur du dernier u_{i+1} est incluse. C'est à ca que sert le premier test if u==U(n).

```
k =

     7
```

Tout les tests sont concluent, notre fonction est donc valide.

3.2 Fonction Vunique

La fonction Vunique retourne l'ensemble V des noeuds distincts dans le vecteur des noeuds U et leur nombre *munique* ainsi que leur multiplicité S. Elle s'écrit comme ceci :

```
function [V,munique,S] = Vunique(U)

V(1)=U(1)
munique=1
S=[1]
for i= 2:length(U)
    if U(i)==V(munique)
        S(munique)=S(munique)+1;
    else
        munique=munique + 1
        V(munique)=U(i)
        S=[S,1]
    end

end
munique=munique-1;
end
```

Testons cette fonction avec le script suivant qui reprend les données de l'énnoncé :

```
clc
U=[0,0,0,0,0.25,0.5,0.75,1,1,1,1];
P=[[0,0]; [1,0] ;[1,1] ;[2,1]; [2.5,0]; [1.5,-0.5]; [0,-1/3]];
```

```
p=3;
u=0.75;
[V,munique,S] = Vunique(U)
```

Nous obtenons :

```
V =

         0    0.2500    0.5000    0.7500    1.0000


munique =

     4


S =

     4     1     1     1     4
```

Ce qui est bien le résultat attendu pour $U = [0, 0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1, 1]$. Notre fonction est donc valide.

3.3 Fonction BasisFuns

La fonction BasisFuns qui retourne dans un tableau N l' ´evaluation des p+1 fonctions de base $N_{kp,p}(u), ..., N_{k,p}$ non nulles pour une valeur de $u \in [uk, uk + 1]$.
Notre fonction s'écrit comme ceci :

```
function N = BasisFuns(u,p,U)
k=FindSand(p,u,U);
termeg = zeros(p+1,1);
termedr = zeros(p+1,1);
N(1)=1;
for j=1:p
    termeg(j+1) = u - U(k+1-j);
    termedr(j+1) = U(k+j) - u;
    s=0;
    for r=0:j-1
        temp = N(r+1)/(termedr(r+2) + termeg(j-r+1));
        N(r+1) = s + termedr(r+2)*temp;
        s = termeg(j-r+1)*temp;

    end
N(j+1) = s;
end
end
```

Testons notre fonction avec le script suivant :

```
clc
U=[0,0,0,0,0.25,0.5,0.75,1,1,1,1]
P=[[0,0]; [1,0] ;[1,1] ;[2,1]; [2.5,0]; [1.5,-0.5]; [0,-1/3]]
p=3;
m=length(U)-1;
n=m-p;
u=1;

N = BasisFuns(u,p,U)
```

On obtient la sortie suivante pour $u = 1$:

```
N =

     0     0     0     1
```

pour $u = 0$:

```
N =

      1       0       0       0
```

pour $u = 0.25$

```
N =

     0.2500     0.5833     0.1667          0
```

Nous avons des résultat cohérent (somme des $N(i)$ qui vaut 1 et symétrie). Notre fonction est donc valide.

3.4 Fonction CurvePoint

La fonction CurvePoint pour un $u \in [U_k, U_{k+1}]$ retourne dans un tableau C les composantes de $C(u)$. Pour cela on utilisera les $p+1$ fonctions de base $N_{kp,p}(u), ..., N_{k,p}$ non nulles pour un $u \in [U_k, U_{k+1}]$ qui permettent de calculer le point C(u) de la courbe.
Notre fonction s'écrit comme ceci :

```
function C = CurvePoint(U, P, p, u, N)
k=FindSand(p,u,U);
C = zeros(1, 2);

for i = k-p:k
    C(1,1) = C(1,1) + N(i-k+p+1)*P(i,1);
    C(1,2) = C(1,2) + N(i-k+p+1)*P(i,2);
end
end
```

Testons notre fonction avec le script suivant :

```
clc
U=[0,0,0,0,0.25,0.5,0.75,1,1,1,1]
P=[[0,0]; [1,0] ;[1,1] ;[2,1]; [2.5,0]; [1.5,-0.5]; [0,-1/3]]
p=3;
m=length(U)-1;
n=m-p;
u=1;
N = BasisFuns(u,p,U);
C = CurvePoint(U, P, p, u, N)
```

Nous obtnons la sortie suivante :

```
C =

         0   -0.3333
```

Nous avons bien un couple (x, y) (i.e un point) en sortie. Notre fonction est donc valide.

3.5 Fonction CurveNURBS

La fonction CurveNURBS calcule les composante de $C(u)$ sur $q = 30$ points régulièrement espacés de $[Vk, Vk+1]$ et les retourne dans un tableau pt.
Notre fonction s'écrit comme ceci :

```
function pt = CurveNURBS (n, p, U, P, V, munique)
  for k = 1:munique
    step=(V(k+1)-V(k))/29;
    Discretisation=V(k):step:V(k+1)
    for i=1:length(Discretisation)
      N=BasisFuns(Discretisation(i),p,U);
      C=CurvePoint(U,P,p,Discretisation(i),N);
      pt(i+30*k,1)=C(1);                                 %on utilise
         i+30*k afin pouvoir tout mettre dans un m me vecteur
      pt(i+30*k,2)=C(2);                                 %car chaque
         interval est discretis  en 30 point. Les 30 premiers point sont
         donc pour le 1er interval,
```

```
    end                                                              %de 31 a 60
        pour le deuxi me interval, etc..
  end

end
```

Testons notre fonction avec le script suivant :

```
clc
U=[0,0,0,0,0.25,0.5,0.75,1,1,1,1]
P=[[0,0]; [1,0] ;[1,1] ;[2,1]; [2.5,0]; [1.5,-0.5]; [0,-1/3]]
p=3;
m=length(U)-1;
n=m-p;
[V,munique,S]=Vunique(U);

pt=CurveNURBS(n,p,U,P,V,munique)
```

On obtient la sortie :

```
pt =

         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
    0.0999    0.0018
    0.1930    0.0069
    0.2795    0.0152
    0.3598    0.0266
    0.4340    0.0407
    0.5026    0.0576
    0.5658    0.0768
    0.6238    0.0984
    0.6770    0.1221
    0.7256    0.1476
```

0.7700	0.1749
0.8104	0.2037
0.8471	0.2339
0.8804	0.2652
0.9106	0.2975
0.9379	0.3306
0.9627	0.3644
0.9853	0.3985
1.0059	0.4330
1.0248	0.4674
1.0423	0.5018
1.0587	0.5358
1.0743	0.5694
1.0893	0.6022
1.1042	0.6343
1.1190	0.6652
1.1342	0.6950
1.1500	0.7233
1.1667	0.7500
1.1667	0.7500
1.1845	0.7750
1.2034	0.7982
1.2235	0.8197
1.2445	0.8394
1.2665	0.8574
1.2893	0.8738
1.3130	0.8885
1.3374	0.9016
1.3625	0.9130
1.3883	0.9229
1.4146	0.9311
1.4415	0.9378
1.4688	0.9430
1.4964	0.9467
1.5245	0.9488
1.5527	0.9495
1.5812	0.9487
1.6099	0.9465
1.6386	0.9429
1.6673	0.9379
1.6960	0.9315
1.7246	0.9237
1.7530	0.9146
1.7812	0.9042
1.8091	0.8926
1.8367	0.8796
1.8638	0.8654
1.8905	0.8500
1.9167	0.8333
1.9167	0.8333
1.9422	0.8155
1.9671	0.7965
1.9913	0.7765
2.0147	0.7554
2.0373	0.7333
2.0589	0.7103
2.0796	0.6864
2.0993	0.6617
2.1179	0.6362
2.1353	0.6100
2.1515	0.5831

```
    2.1665    0.5556
    2.1801    0.5275
    2.1923    0.4989
    2.2031    0.4698
    2.2124    0.4403
    2.2201    0.4104
    2.2261    0.3802
    2.2304    0.3497
    2.2330    0.3190
    2.2337    0.2882
    2.2326    0.2572
    2.2295    0.2262
    2.2244    0.1952
    2.2173    0.1642
    2.2080    0.1333
    2.1965    0.1025
    2.1827    0.0720
    2.1667    0.0417
    2.1667    0.0417
    2.1482    0.0117
    2.1272   -0.0180
    2.1035   -0.0471
    2.0769   -0.0757
    2.0473   -0.1036
    2.0145   -0.1307
    1.9783   -0.1570
    1.9386   -0.1823
    1.8953   -0.2066
    1.8480   -0.2298
    1.7968   -0.2518
    1.7413   -0.2724
    1.6815   -0.2917
    1.6172   -0.3094
    1.5483   -0.3256
    1.4744   -0.3401
    1.3956   -0.3528
    1.3117   -0.3636
    1.2223   -0.3725
    1.1275   -0.3793
    1.0271   -0.3840
    0.9208   -0.3865
    0.8085   -0.3866
    0.6901   -0.3843
    0.5654   -0.3795
    0.4342   -0.3721
    0.2963   -0.3620
    0.1516   -0.3491
         0   -0.3333
```

Nous avons une liste de point (ce qui est le type de sortie attendue) que nous allons afficher grâce à la fonction suivante.

3.6 Fonction DrawNURBS

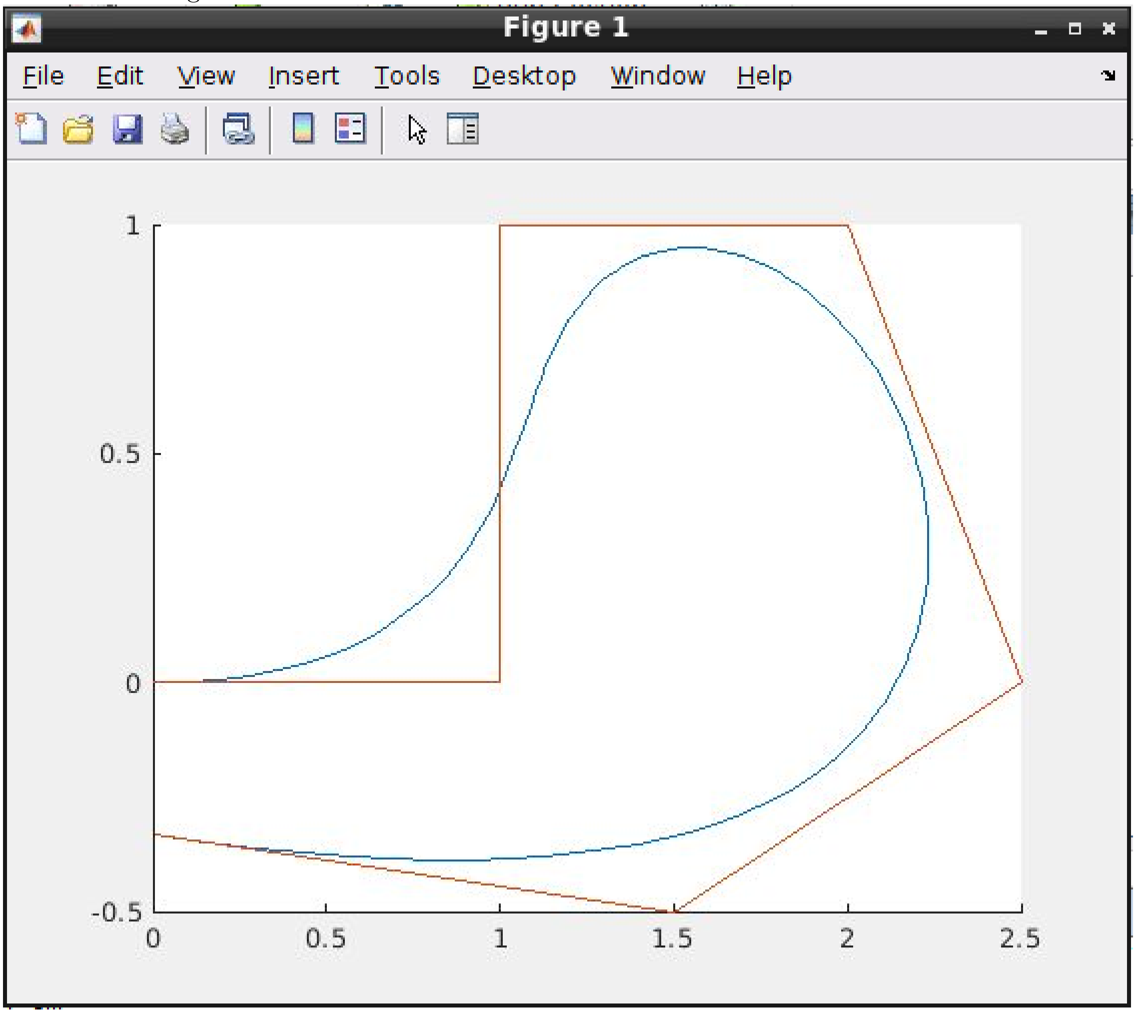
La fonction drawNURBS pour un tableau de points contenus dans un tableau pt trace la courbe correspondante ainsi que le polygone des points de contrôle. Notre fonction s'écrit comme ceci :

```
function[]=DrawNURBS(n,p,pt,P)
hold on
plot(pt(:,1),pt(:,2));            %trace la courbe
plot(P(:,1),P(:,2));              %trace le polygone des points de controle
end
```

Testons notre fonction et notre programme au complet avec le script suivant :

```
clc
U=[0,0,0,0,0.25,0.5,0.75,1,1,1,1]
P=[[0,0]; [1,0] ;[1,1] ;[2,1]; [2.5,0]; [1.5,-0.5]; [0,-1/3]]
p=3;
m=length(U)-1;
n=m-p;
[V,munique,S]=Vunique(U);
pt=CurveNURBS(n,p,U,P,V,munique);
DrawNURBS(n,p,pt,P);
```

Nous obtenons la figure suivante :



C'est bien la courbe que nous voulions obtenir. Nos differents programmes sont donc fonctionnels.