

## 1 Présentation

## 2 Tracé naïf d'une courbe de Bézier

**Question 1** Écrire cette fonction en utilisant un algorithme récursif `factRec(n)`. Vous prendrez soin de documenter votre fonction.

## ■ Python

```
def factRec(n):
    """
    Calcul de  $n! = 1 \times 2 \times \dots \times (n-1) \times n$ 
    Par convention,  $0! = 1$ 
    n doit être un int
    """
    if n==0:
        return (1)
    else:
        return (n*factRec(n-1))
```

**Question 2** Écrire cette fonction en utilisant un algorithme itératif `factIt(n)`. Vous prendrez soin de documenter votre fonction.

## ■ Python

```
def factIt(n):
    p=1
    for i in range(1,n+1):
        p=p*i
    return (p)
```

**Question 3** En utilisant la fonction `calculPointCourbe(poles,t)` (donnée en annexe), réaliser le programme permettant de tracer une courbe sur 100 points. On rappelle que pour utiliser la fonction `plot` il est nécessaire de réaliser la liste des abscisses, qu'on pourra nommer `les_x`, et la liste des ordonnées, qu'on pourra nommer `les_y`. On fera l'hypothèse que la liste de pôles a déjà été renseignée dans la variable `poles`.

## Correction

## ■ Python

```
poles = [[0,0],[0,20],[40,20],[40,0]]
les_u = np.linspace(0,1,100)

les_x_bern = []
les_y_bern = []
for t in les_u:
    pt = calculPointCourbe(poles,t)
    les_x_bern.append(pt[0])
    les_y_bern.append(pt[1])

plt.plot(les_x_bern,les_y_bern,"b.")
plt.show()
```

**Question 4** On fait l'hypothèse que la complexité algorithmique de la fonction `pow`, appelée dans la fonction `fonctionBernstein`, est linéaire. Donner la complexité algorithmique temporelle de la fonction `fonctionBernstein`.

**Correction** Si fonction Bernstein a une complexité  $C(n)$ , alors  $C(n+1)$  vaut : pour obtenir le  $n+1$  élément, nous avons une affectation, une somme, 3 produits de termes obtenus par 3 appels de fonctions elles-mêmes de complexité linéaire  $C(n+1)=C(n)+3*(n+1)+5$ . La complexité algorithmique de la fonction de Bernstein est en  $\mathcal{O}(n^2)$ .

### 3 Utilisation de l'algorithme de De Casteljau

**Question 5** On donne la fonction `deCasteljau` permettant de calculer l'abscisse (ou l'ordonnée) d'un point d'une courbe. Déterminer ce que retourne l'appel suivant (en justifiant et détaillant votre démarche) : `deCasteljau([0,0,40,40],0,3,0.5)`.

**Correction**

$$\begin{aligned} \text{deCasteljau}(P,0,3,0.5) &= \text{deCasteljau}(P,0,2,0.5)*(1-0.5) + \text{deCasteljau}(P,1,2,0.5)*0.5 \\ &= (\text{deCasteljau}(P,0,1,0.5)*(1-0.5) + \text{deCasteljau}(P,1,1,0.5)*0.5)*(1-0.5) \\ &\quad + (\text{deCasteljau}(P,1,1,0.5)*(1-0.5) + \text{deCasteljau}(P,2,1,0.5)*0.5)*0.5 \\ &= ((\text{deCasteljau}(P,0,0,0.5)*(1-0.5) + \text{deCasteljau}(P,1,0,0.5)*0.5)*(1-0.5) \\ &\quad + (\text{deCasteljau}(P,1,0,0.5)*(1-0.5) + \text{deCasteljau}(P,2,0,0.5)*0.5)*0.5)*(1-0.5) \\ &\quad + ((\text{deCasteljau}(P,1,0,0.5)*(1-0.5) + \text{deCasteljau}(P,2,0,0.5)*0.5)*(1-0.5) \\ &\quad + (\text{deCasteljau}(P,2,0,0.5)*(1-0.5) + \text{deCasteljau}(P,3,0,0.5)*0.5)*0.5)*0.5 \\ &= 20 \end{aligned}$$

**Question 6** Évaluer la complexité algorithmique de l'algorithme de De Casteljau en fonction du nombre de pôles.

**Correction** Si la complexité de De Casteljau est  $C(n)$  pour  $n$  pôles alors  $C(n+1) = C(n)*2$  on double les appels à la fonction ainsi la complexité est exponentielle en  $\mathcal{O}(2^n)$ .

**Question 7** En identifiant un variant de boucle, montrer que l'algorithme se termine.

**Correction** Le variant de boucle est  $j$ , entier positif qui décroît strictement à chaque appel récursif jusqu'à 0. A  $j=0$ , l'algorithme se termine.

### 4 Utilisation de l'algorithme de Horner

**Question 8** Écrire un algorithme récursif, permettant de calculer un point de la courbe par la méthode de Horner. La fonction `horner` prendra comme argument  $L$  la liste des  $a_i$  ( $[a_n, a_{n-1}, \dots, a_1, a_0]$ ) et le paramètre  $t$ .

■ Python

```
def horner(L,t):
    if len(L)==0:
        return 0
    else :
        return horner(L[0:len(L)-1],t)*t+L[len(L)-1]
```

**Question 9** Quel est l'avantage d'évaluer un polynôme en un point par la méthode de Horner plutôt que par une méthode naïve ?

**Correction** La complexité de la méthode de Horner est linéaire : si la complexité de Horner au rang  $n$  est  $C(n)$  alors  $C(n+1)=C(n)+2$  soit  $\mathcal{O}(n)$  alors que la complexité de la méthode Naïve est quadratique :  $C(n+1)=C(n)+3*n+5$  soit  $\mathcal{O}(n^2)$

## 5 Bilan

**Question 10** Sachant que les polynômes de Bézier utilisés sont la plupart du temps de degré 3 (4 pôles), parmi les méthodes proposées (méthode naïve, de de Casteljau ou de Horner), laquelle préconiseriez vous évaluer les points d'une courbe de Bézier ?

**Correction** Pour 4 pôles, la comparaison des temps de résolution des trois méthodes donne : pour Horner (une multiplication, une addition, un test à chaque boucle) est de  $3 \cdot 4 = 12$  sachant qu'il faut aussi créer la liste des coefficients à partir du polynôme de base de Bernstein, pour la méthode naïve (une affectation, une boucle de taille  $n$ , une affectation, une somme et 3 produits) est  $5 \cdot n \cdot n + 1$  soit 81, enfin pour De Casteljau (un test, 2 produits, une somme par appel) est  $4 \cdot 2^4 = 64$ . L'algorithme de De Casteljau reste plus simple à programmer.