TP1 : algorithme de Casteljau (premiers exemples)

Dans les différents TP, vous allez découvrir différentes méthodes pour définir et tracer des courbes de Bézier. Ce premier TP est consacré à l'*algorithme de Casteljau*, qui est certainement la méthode la plus intuitive et visuelle. Cet algorithme est également connu pour sa grande stabilité numérique (c'est-à-dire qu'il est peu sensible aux erreurs d'arrondi inhérentes au calcul sur ordinateur).
L'algorithme de Casteljau est basé sur un calcul par récurrence : une courbe de Bézier d'ordre $N+1$ peut toujours se calculer comme la moyenne pondérée de 2 courbes de Bézier d'ordre N. Pour introduire le principe, ce TP se concentre sur les cas les plus simples qui sont les courbes d'ordre $N=1$ (courbe affine) $N=2$ (courbe quadratique) et $N=3$ (courbe cubique).

Représentation des points avec Numpy. Les courbes de Bézier manipulent des *points du plan*, à 2 coordonnées. Dans ce TP, un point sera représenté par un *numpy.array* à 2 éléments. Par exemple, le point $A(-1,3)$ sera représenté par

```
A = np.array( [-1,3] )
```

L'intérêt de cette représentation avec Numpy est qu'il est très simple de calculer des moyennes pondérées entre points. Par exemple, si *A* et *B* sont des points représentés avec Numpy, alors leur milieu peut se calculer simplement par

```
M = 0.5*A + 0.5*B
```

alors que, si *A* et *B* étaient de simples *listes* Python, la commande pour calculer *M* serait moins élégante. *(Tiens d'ailleurs, quelle commande faudrait-il taper ?)*

Exercice 1 (Courbe de Bézier d'ordre 1). Étant donnés 2 points de contrôle A et B, la courbe de Bézier correspondante est

$$F(t) = (1-t)A + tB$$

1. Écrivez une procédure

   ```
   def bezier1(t,A,B):
   ```

   qui prend en entrée 2 points A et B (au format Numpy expliqué ci-dessus), un nombre réel $t \in [0,1]$, et renvoie le point correspondant de la courbe de Bézier $F(t)$.
2. Vérification : définissez 2 points A et B, et visualisez la courbe de Bézier correspondante, en représentant (dans la même figure) les points de parmètre $t = 0,\ 0.1,\ 0.2, \ldots, 1$.

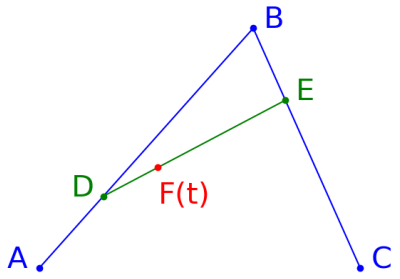
Exercice 2 (Courbe de Bézier d'ordre 2). 3 points de contrôle A, B et C définissent une courbe de Bézier d'ordre 2 par la formule

$$F(t) = (1-t)\big[(1-t)A + tB\big] + t\big[(1-t)B + tC\big]$$

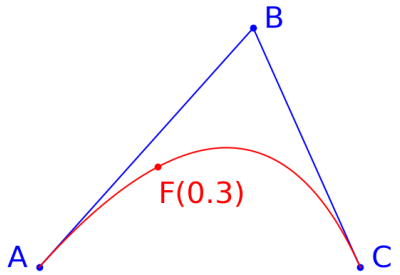
L'algorithme de Casteljau consiste à d'abord construire les 2 points intermédiaires :

$$D = (1-t)A + tB$$
$$E = (1-t)B + tC$$

puis d'en déduire le point $F(t)$, comme dans la figure ci-dessous :



Pour dessiner la courbe de Bézier dans sa totalité, on répète cette procédure de calcul pour chaque valeur de t entre 0 et 1 et on affiche tous les points $F(t)$ ainsi calculés.
Voici la courbe de Bézier résultante, pour le même exemple que précédemment. Le point $F(0.3)$, dont la construction était illustrée à la figure précédente, est mis en évidence.



Questions.

1. Écrivez une procédure

```
def bezier2(t,A,B,C):
    D = TODO() # premier point intermédiaire
    E = TODO() # deuxième point intermédiaire
    return TODO() # le point final F(t)
```

   qui implémente le calcul du point $F(t)$.

2. Vérification : choisissez 3 points A, B et C, une valeur de t entre 0 et 1, et reproduisez l'équivalent de la première figure de cet exercice. C'est-à-dire que vous devez représenter :

   (a) Le *polygone de contrôle* (ABC) (formé des 2 segments de droite $[AB]$ et $[BC]$).

(b) Les points intermédiaires D et E, ainsi que le segment $[DE]$.

(c) Le point $F(t)$ renvoyé par la fonction `bezier2(t,A,B,C)`

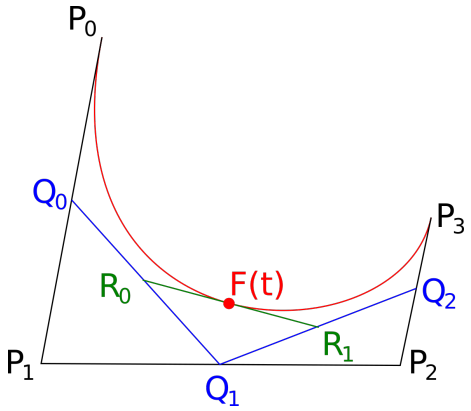
3. Dans une AUTRE figure, reproduisez l'équivalent de la seconde figure de cet exercice, c'est-à-dire, la totalité de la courbe $F(t)$ lorsque le paramètre t varie de 0 à 1. En détaillant :

(a) Établissez une discrétisation assez fine de l'intervalle $[0, 1]$ à l'aide d'un `linspace`.

(b) Pour chaque t dans votre discrétisation, calculez et sauvegardez le point $F(t)$.

(c) Tracez la courbe formée de tous les points $F(t)$ ainsi calculés. Rappel : `plt.plot(X,Y)`, où `X` sont toutes les abscisses des points, et `Y` toutes leurs ordonnées.

Exercice 3 (Courbe de Bézier d'ordre 3)**.** Nous généralisons maintenant la construction pour afficher une courbe de Bézier d'ordre 3, c'est-à-dire avec 4 points de contrôle. Afin de nous rapprocher des notations standard, ces 4 points de contrôle seront maintenant notés P_0, P_1, P_2, P_3, comme dans la figure ci-dessous (adaptée de Wikipedia) :



La courbe de Bézier d'ordre 3, associée aux points P_0, P_1, P_2, P_3, peut être définie par une moyenne pondérée de 2 courbes de Bézier d'ordre 2, par la formule suivante :

$$F_{P_0,P_1,P_2,P_3}(t) = (1-t)F_{P_0,P_1,P_2}(t) + tF_{P_1,P_2,P_3}(t)$$

1. (Question papier) Compréhension de la formule.

(a) Expliquez en quoi cette définition possède la même structure que la définition dans le cas $N = 2$, traité à l'exercice précédent.

(b) Dans la figure précédente, trouvez les associations entre chaque point, et le calcul qu'il représente :

- Q_0
- Q_1
- Q_2
- R_0
- R_1
- $F(t)$

- $F_{P_0,P_1,P_2,P_3}(t)$
- $F_{P_0,P_1,P_2}(t)$
- $F_{P_1,P_2,P_3}(t)$
- $F_{P_0,P_1}(t)$
- $F_{P_1,P_2}(t)$
- $F_{P_2,P_3}(t)$

2. Écrivez une procédure

```
def bezier3(t,P0,P1,P2,P3):
    return TODO() # le point final F(t)
```

qui implémente le calcul du point $F_{P_0,P_1,P_2,P_3}(t)$. Votre procédure pourra faire appel à d'autres procédures définies précédemment.

3. Vérification : choisissez 4 points P_0, P_1, P_2 et P_3, et représentez la totalité de la courbe $F_{P_0,P_1,P_2,P_3}(t)$ lorsque le paramètre t varie de 0 à 1.

Exercice 4 (Une animation). Pour conclure, réalisons une animation « en mode Wikipedia » qui illustre la construction progressive d'une courbe de Bézier d'ordre 3. Dans cette animation,

- On construit successivement 20 points de la courbe, associés aux valeurs du paramètre $t = 0.05$, $t = 0.1$, $t = 0.15$, ..., $t = 1$.
- Pour chaque point $F(t)$ construit, on illustre également la totalité des points de calcul intermédiaires utilisés par l'algorithme de Casteljau, comme dans la figure ci-dessus. La figure reste affichée pendant 0.2 secondes, puis on passe à la construction du point $F(t)$ suivant.
- L'ensemble des points $F(t)$ déjà construits restent affichés dans la figure, afin de voir la courbe entière se construire progressivement. Par contre, les points intermédiaires s'actualisent à chaque étape.

Pour cela, complétez le code ci-dessous :

```
plt.close('all')
plt.figure()
# Choix de 4 points de controle au hasard (P[0] est le premier point, P[1] le second, etc.)
P = np.random.rand(4,2)
# Liste qui stockera les points F(t) déjà calculés (afin de les réafficher à chaque itération)
allFt_x = []
allFt_y = []
for t in TODO(): # 20 valeurs équiréparties entre 0 et 1
    # Nettoie la figure
    plt.gcf().clear()
    # Calcul du nouveau point F(t)
    Ft = TODO()
    # Calcul et affichage de tous les polygones de contrôle intermédiaires
    TODO()
    # Stocke les coordonnées du nouveau point F(t)
    allFt_x.append(Ft[0])
    allFt_y.append(Ft[1])
    # Affiche l'ensemble des points construits jusqu'ici
    plt.plot(allFt_x,allFt_y,color='red',marker='o')
    # Pause (définit la vitesse de l'animation)
    plt.pause(.2)
```