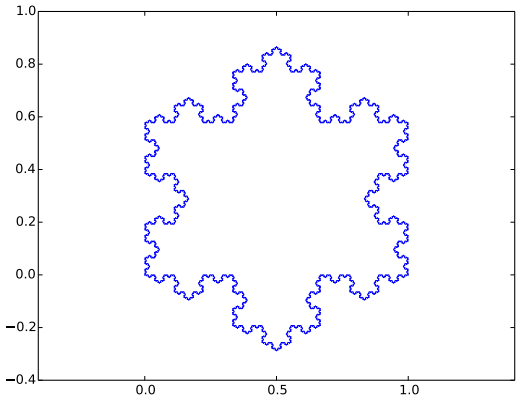
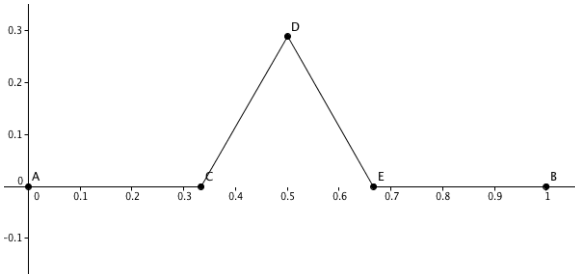
T.P. 9 : nombres complexes et une fractale

1 Introduction

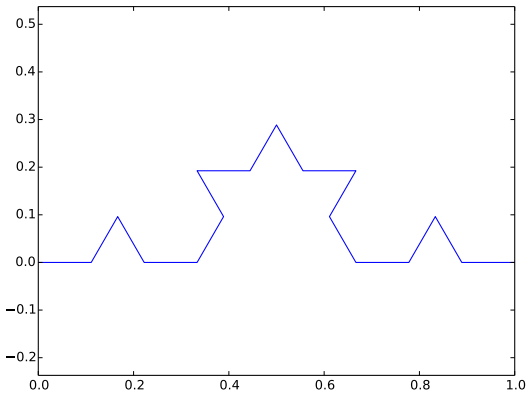
A la fin de ce TP, si tout va bien, vous saurez obtenir des dessins de ce genre avec PYTHON :



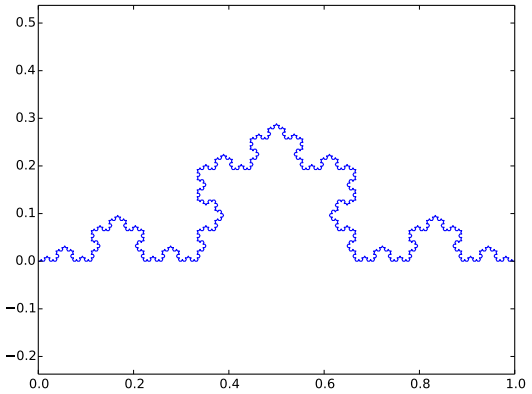
L'idée de base de la construction d'une courbe de Von Koch, sur le dessin suivant à partir du segment $[A, B] \subset \mathbb{C}$ avec A d'affixe 0 et B d'affixe 1, est de transformer ce segment en la ligne brisée suivante où C est d'affixe 1/3, E d'affixe 2/3 et CDE est équilatéral :



Ensuite, on itère cette opération sur chacun des quatre segments de la figure précédente, ce qui donne :



On peut alors recommencer autant de fois qu'on veut, par exemple après cinq itérations :



2 Informations sur les manipulations de complexes en PYTHON :

Il y a au moins deux façons de rentrer un nombre complexe en PYTHON :

```
>>>z=1+2j
>>>z=complex(1,2)
```

En pratique l'entrée sous forme `a+bj` n'est pas très commode (syntaxe pénible : coller la partie imaginaire à j ...), et on recommande plutôt la commande `complex`.

Pour un nombre complexe `z`, avec `z.real` et `z.imag` (sans parenthèses) on aura resp. la partie réelle et la partie imaginaire de `z`.

Conventions pour `numpy` et `matplotlib.pyplot` : dans tout ce qui suit, on convient qu'on a fait

```
import numpy as np
import matplotlib.pyplot as plt
```

On peut aussi remplacer ces deux importations par une seule, celle de `pylab` :

```
import pylab as pl
```

On rappelle que pour tracer un segment $[A, B]$ si $A = (1, 0)$ et $B = (2, 3)$ avec la commande `plt.plot` on doit créer des listes (ou un tableau `numpy`) `X=[1,2]` et `Y=[0,3]` puis les passer en argument à `plot` : `plt.plot(X,Y)`.

Mais si `A=complex(1,0)` et `B=complex(2,3)`, et `L=[A,B]` (liste de complexes) on peut aussi définir directement `X=np.real(L)` et `Y=np.imag(L)` qui fabriquent les tableaux des parties réelles et imaginaires respectivement (dans l'esprit des commandes `numpy` qui s'appliquent entrée par entrée à un tableau) et ensuite faire `plt.plot(X,Y)`.

Pour obtenir l'affichage dans un repère orthonormé, utilisez `plt.axis('equal')`.

3 Travail à faire pour obtenir les courbes de Von Koch

a) Ecrire une fonction `TS` (pour Transformation Segment), qui prend en argument deux complexes A et B, et renvoie la *liste* formée des complexes A,C,D,E de la construction de la courbe de Von Koch donnée au § 1.

N.B. 1 : Les coordonnées complexes de C, D, E sont *relatives* à celles de A et B.

N.B. 2 : on ne met pas B dans la liste, on va voir pourquoi après : pour coller les listes les unes à la suite des autres.

b) **Fonction donnant une transformation globale**
Ecrire une fonction `TG` (pour Transformation Globale) qui prend en argument une liste `L` de nombres complexes et qui retourne une liste qu'on appellera `LT` dans la fonction, qui est la liste obtenue à partir de `L` en intercalant entre deux nombres successifs A,B de la liste, les nombres C,D,E de la construction de Von Koch.

c) **Fonction d'affichage avec choix du nombre d'itérations**
Ecrire une fonction `VonKoch1` qui prend comme argument obligatoire un entier `n` et comme argument facultatif une liste `L` de nombres complexes, avec la valeur par défaut `L=[0,1]` et qui affiche la courbe de Von Koch obtenue en `n` itérations à partir du segment `L` grâce à la fonction `TG` précédente.

d) **Passage de la courbe de Von Koch au triangle de Von Koch**
En appliquant la fonction `VonKoch1` précédente aux trois côtés d'un triangle équilatéral, obtenir le flocon de neige de l'introduction.

4 Comme introduction à l'option info. : récursion

4.1 Définition et exemple

Définition Une *fonction récursive* est une fonction `f` telle que le code de définition de `f` contienne un (ou plusieurs) appel(s) à `f`. Ces appels sont appelés *appels récursifs*.

Comme on va le voir dans les exemples ci-dessous, cette méthode de définition de fonction est très proche de la notion de *définition par récurrence* en maths.

Exemple : En maths, on peut définir la factorielle d'un entier $n \in \mathbb{N}$ par $\begin{cases} 0! = 1, \\ \forall\, n \in \mathbb{N}^*,\ n! = n \times (n-1)! \end{cases}$

Une déf. récursive de la factorielle en Python, calquée sur la déf. mathématique précédente, est la suivante :

```
def fact(n):
    if n==0 :
        return 1
    else :
        return n*fact(n-1)
```

Remarque : On utilise la même syntaxe de déf. de fonctions : il n'y pas de déclaration particulière pour les fonctions récursives (ce sera différent en Caml).

4.2 Le von Koch en récursif

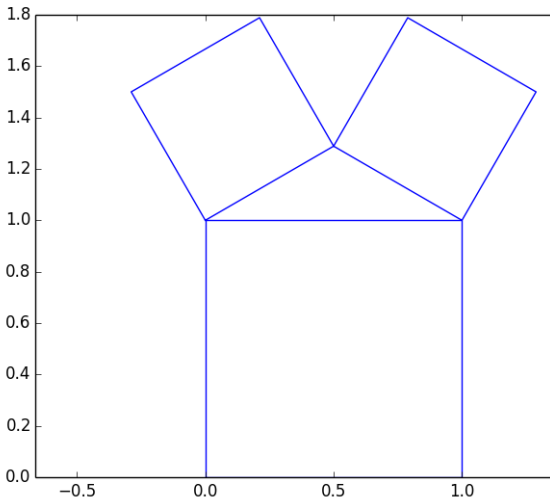
On peut alors écrire le programme donnant le flocon de Von Koch en récursif comme suit (code à compléter) ;

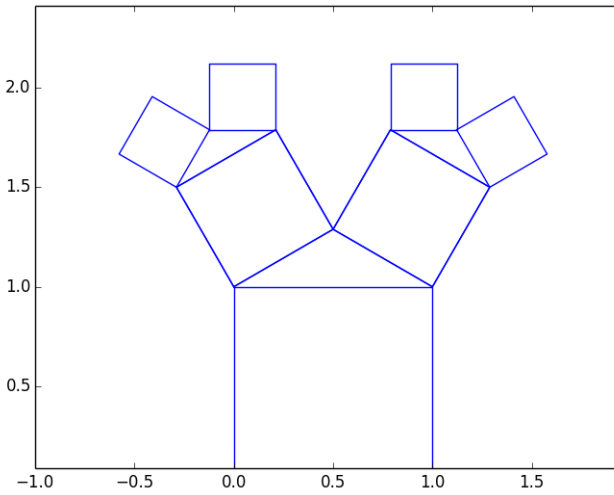
```
def KochRec(n,A=complex(0,0),B=complex(1,0)):
    alpha=np.pi/3
    R=np.exp(complex(0,alpha))
    if n==0:
        X=np.real([A,B])
        Y=np.imag([A,B])
        pl.plot(X,Y)
        pl.axis('equal')
        pl.show()
    else :
        Ecart=(B-A)/3
        C=A+Ecart
        E=B-Ecart
```

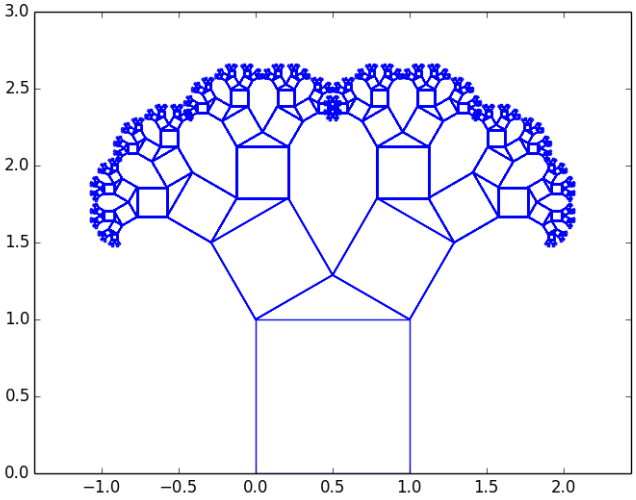
```
D=C+R*Ecart
# Puis plusieurs appels de KochRec(n-1, ..) à compléter.
```

4.3 Le chou fleur

En adaptant ce qui précède réaliser successivement les figures suivantes :







5 Tracés de lignes polygonales à partir des angles successifs

On veut écrire une fonction `trace` qui prend en argument une liste (ou tableau) de nombres $[\mathtt{a}_0, \ldots, \mathtt{a}_{n-1}]$ et trace la courbe polygonale $(M_0, \ldots, M_n)$ telle que $M_{-1} = (0,0)$, $M_0 = (1,0)$ et

- pour tout $k \in [\![0, n-1]\!]$ la distance entre M_k et M_{k+1} vaut 1,
- pour tout $k \in [\![0, n-1]\!]$ l'angle $(\overrightarrow{M_{k-1}M_k}, \overrightarrow{M_kM_{k+1}})$ est congru à a_k modulo 2π.

a) On note z_k les affixes des points M_k. Justifier qu'on peut écrire $z_{k+1} = z_k + e^{i\theta_k}$ et exprimer θ_k à l'aide des a_i.

b) En déduire un code de la fonction `trace` demandée.

c) A l'aide de la fonction `trace` précédente, comment faire tracer un polygone régulier à n côtés ?

Pour n suffisamment grand, que voit-on à l'écran ?

6 Application à la courbe du dragon

La courbe que nous allons construire, appelée *courbe du dragon* est une ligne polygonale $(M_0, \ldots, M_n)$ du même type qu'au paragraphe précédent, où l'on tourne à chaque fois d'un angle a_k constant en valeur absolue, mais dont le signe change, suivant une règle que l'on va préciser.

Dans ce qui suit, on prendra $a_k = \varepsilon_k \pi/2$.

Reste à définir la liste $T = [\varepsilon_0, \ldots, \varepsilon_n]$.

On construit `T` par récurrence.

• L'initialisation est le tableau $\mathtt{T}_0 = [1]$.

• Pour passer de l'étape k à l'étape $k+1$, on prend le tableau $\mathtt{T}_k$ obtenu à l'étape k qui est disons de taille n_k et on fabrique le tableau $\mathtt{T}_{k+1}$ de taille $2n_k + 1$ défini comme suit :

$\mathtt{T}_{k+1}$ commence par un 1, ensuite on met la première entrée de T_k, suivie d'un -1, puis la seconde entrée de T_k suivie d'un $+1$, puis la troisième entrée de T_k suivie d'un -1 etc, jusqu'à la dernière entrée de T_k suivie soit d'un 1 soit d'un -1 suivant la parité de n_k.

Exemple : on met en gras les entrées du tableau précédent ; T_1 =`[1,1,-1]` puis T_2 = `[1, 1, -1,1, 1,-1, -1]`.

a) Ecrire une fonction `dragon` qui reçoit un entier `n` et qui renvoie la liste T_n où toutes les entrées sont multipliées par $\pi/2$.

Remarque : on pourra utiliser une fonction auxiliaire `iter` pour passer de T_k à T_{k+1}.

b) A l'aide de la fonction `dragon` et de la fonction `trace` obtenir la courbe du dragon.

Exemple : avec `trace(dragon(15))` on obtient (en enlevant les axes avec `plt.axis("off")`)