Fractales

Une fractale est un objet mathématique qui présente une structure similaire à toutes les échelles. De nombreux objets naturels très irréguliers (tracé de côtes, formes de végétaux, motifs de pelages d'animaux, montagnes, nuages...) peuvent être modélisés par des fractales approximatives (voir exemple ci-dessous).

Figure 1. Chou romanesco

Remarque. Dans tout le TP, on tracera les segments avec la méthode `plot` du module `matplotlib.pyplot`. Par exemple, pour tracer un segment d'épaisseur 3 et de couleur rouge entre les points $A(x_A, y_A)$ et $B(x_B, y_B)$ on pourra écrire :

```
fig, ax = plt.subplots()
ax.plot([xA, xB], [yA, yB], color='red', lw=3)
```

1 Systèmes de fonction itérées

Ces fractales sont définies par une règle de remplacement géométrique appliquée récursivement. Mais notons qu'on peut aussi *ne pas tracer* la partie à enlever pour dessiner la fractale, ce qui peut s'avérer judicieux si l'on a pas de fonction « d'effacement » à sa disposition.

1.1 Triangle de Sierpinski

Procédure de construction du triangle de Sierpinski :

- Tracer un triangle (équilatéral par exemple) ayant une base parallèle à l'axe des abscisses.
- Calculer les milieux de chaque côté du triangle. Ces trois nouveaux points définissent quatre nouveaux petits triangles.

- Ne rien faire avec le petit triangle central. Recommencer à la deuxième étape avec chacun des trois autres petits triangles.

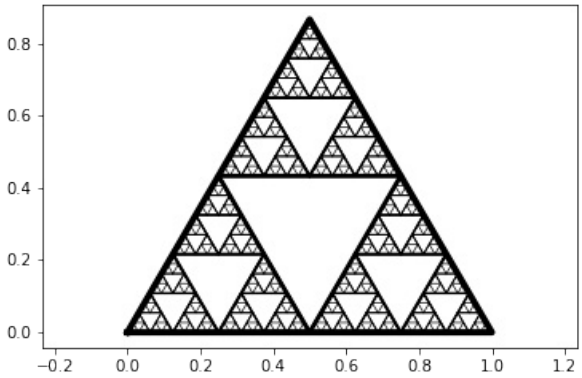

Figure 2. Triangle de Sierpinski et coquillage *Conus Textile*

Question 1. *Définir un triangle (= un triplet de points, chaque point est un doublet de* `float`*) équilatéral. S'aider par exemple du cercle trigonométrique...*

Question 2. *Utiliser une seule instruction* `plot` *pour tracer ce triangle.*

Question 3. *Écrire une fonction* `m(A, B)` *qui prend en argument deux points* `A` *et* `B` *(deux tuples à deux éléments) et qui retourne le point milieu du segment* [AB].

Question 4. *Écrire une fonction récursive* `triangle(ax, T, n)` *qui prend un argument une zone de tracé* `ax`*, un triangle* `T` *(un tuple de trois points) et un nombre d'itérations* `n` *et qui trace récursivement un triangle de Sierpinski de profondeur* `n` *dans le triangle* `T`*. Raffiner ensuite la fonction en ajoutant un argument* `lw` *(linewidth = épaisseur de ligne) qui décroît à chaque appel récursif (choisir par exemple un facteur multiplicatif de* `0.7`*) pour obtenir le résultat de la figure 2.*

1.2 Flocon de Koch

Procédure de construction :

1. On divise un segment de droite horizontal en trois segments de longueurs égales.
2. On construit un triangle équilatéral ayant pour base le segment médian de la première étape.
3. On supprime le segment de droite qui était la base du triangle de la deuxième étape.

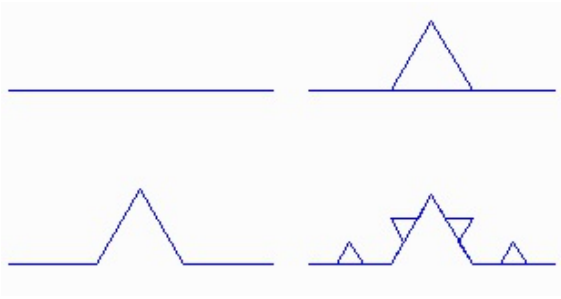

Figure 3. Les quatre premières étapes de la construction du flocon.

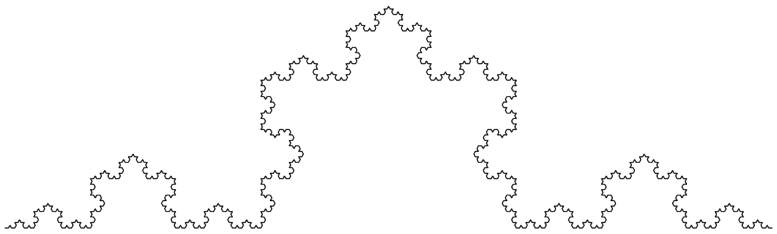

Figure 4. Flocon de Koch pour 5 itérations

Dans cette partie on va utiliser les facilités du module `numpy`. Un vecteur du plan sera modélisé par un tableau (`np.array`) à deux éléments (les coordonnées du vecteur). La multiplication d'un vecteur `v` par une constante `k` s'écrira alors simplement `k*v` (ou `v*k`), l'addition et la soustraction de deux vecteur `v1` et `v2` s'écriront respectivement `v1 + v2` et `v1 - v2` dans la mesure où toutes les opérations arithmétiques sur les tableaux `numpy` sont définies terme à terme.

La rotation d'un vecteur `v` d'un angle `alpha` pourra s'écrire comme le produit de la matrice de rotation :

$$R_\alpha = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

par le vecteur `v` (on obtient alors le vecteur `v2`) :

```
v2 = np.array([[cos(alpha), -sin(alpha)], [sin(alpha), cos(alpha)]]) @ v
```

Question 5. *Écrire une fonction* `rot(v, alpha)` *qui retourne le vecteur résultant de la rotation du vecteur* `v` *d'un angle* `alpha`.

Question 6. *Partant d'un segment* [AB]*, la construction du flocon nécessite de définir 3 nouveaux points :*

- *les points P et R qui découpent le segment en trois parties égales (*$3\mathrm{AP} = 3\mathrm{PR} = 3\mathrm{RB} = \mathrm{AB}$*)*
- *le point Q qui forme un triangle équilatéral* PQR *avec les points P et Q*

Exprimer P, Q et R en fonction des opérations vectorielles de base (multiplication par une constante, addition, soustraction et rotation) et des points A et B (on pourra écrire $M = \overrightarrow{\mathrm{OM}}$ *pour simplifier un peu les notations).*

Question 7. *Écrire une fonction récursive* `koch(A, B, n)` *qui prend un argument les extrémités* `A` *et* `B` *d'un segment et un nombre d'itérations* `n` *et qui retourne* ***la liste des points qui correspondent à la construction du flocon de Koch*** *avec* `n` *itérations.*

Question 8. *Écrire une fonction* ***récursive*** `zippe(li)` *qui prend un argument une liste de points* (x, y) *et qui retourne deux listes : la première qui contient les abscisses des points et la seconde les ordonnées.*

Exemple. `zippe([(1, 2), (3, 4), (5, 6)])` *doit retourner* `[1, 3, 5], [2, 4, 6]`.

Question 9. *Utiliser les deux fonctions précédentes pour tracer un flocon de Koch pour* $n = 5$.

2 Fractales aléatoires

Elles sont très utilisées pour modéliser des objets du monde réel (textures, terrains...). Elles sont obtenues par des procédures (= des fonctions qui ne retournent rien) récursives dans lesquels on fait varier certains paramètres aléatoirement.

2.1 Terrain fractal 2D

On peut générer des terrains 2D aléatoires d'aspect réaliste par la méthode suivante :

- on tire au sort (par exemple sur [0, 1] avec la fonction `random` du module `random`) les ordonnées des deux points extrêmes A et B (on choisit les abscisses comme on veut).
- à chaque itération, on calcule les coordonnées du point M milieur du segment [AB] et on ajoute à son ordonnée un facteur aléatoire dont l'amplitude diminue à chaque appel récursif.
- on recommence avec les segments AM et MB.

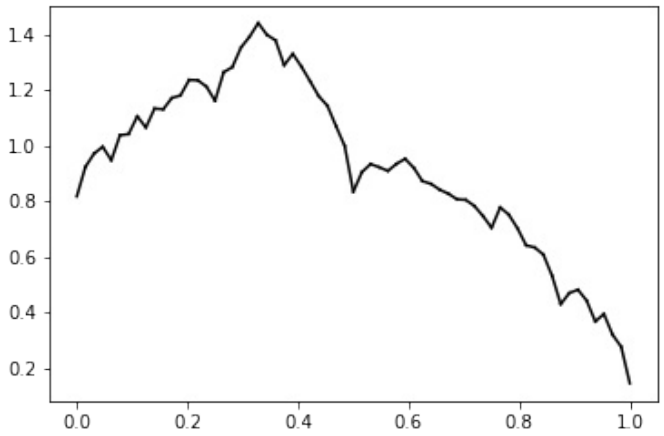

Figure 5. Terrain fractal 2D

Question 10. *Écrire une fonction* `terrain(ax, A, B, k, n)` *qui trace dans une zone de tracé* `ax` *un terrain fractal 2D.*

2.2 Arbres fractals

On se propose tout d'abord de construire un arbre fractal sans introduire de facteurs aléatoires : chaque branche donne naissance à deux branches qui font un angle α constant avec leur branche-mère.

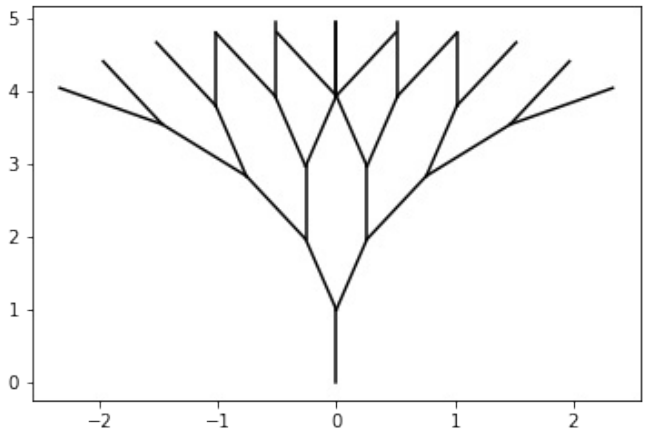

Figure 6. Un premier arbre fractal

Question 11. *Partant d'un segment $[M_0M_1]$ (branche mère), exprimer les points M_2 et M_3 (extrémités des branches filles M_1M_2 et M_1M_3) en fonction des points M_0 et M_1, des angles α_1 et α_2 que font les branches filles avec leur branche mère.et des opérations vectorielles de base définies précédemment.*

Question 12. *Écrire une fonction récursive* `tree(ax, M0, M1, alpha1, alpha2, n)` *qui trace dans une zone de tracé* `ax` *une arbre fractal à* `n` *itérations. Le tronc est défini par le segment M_0M_1, les arguments* `alpha1` *et* `alpha2` *correspondent aux angles constants entre branches filles et branche mère.*

L'arbre obtenu n'est pas encore très réaliste ! On peut alors commencer par faire diminuer la taille et l'épaisseur des branches lors de chaque appel récursif :

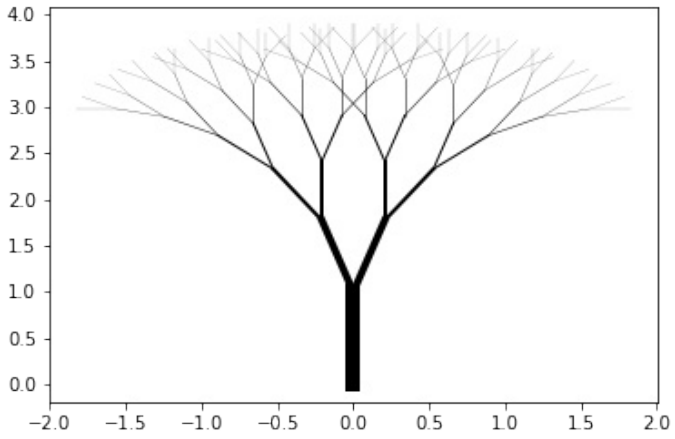

Figure 7. Avec diminution de la longueur et de l 'épaisseur des branches

Puis ensuite introduire des facteurs aléatoires :

- la longueur d'une branche fille est en partie aléatoire

- l'angle que fait une branche fille avec sa mère est en partie aléatoire
- il y a une certaine probabilité de voir apparaître trois branches filles au lieu de deux à chaque itération

Remarque. Pour introduire des facteurs aléatoires, on pourra utiliser les fonctions `random` et `gauss` du module `random`.

Question 13. *Modifier la fonction précédente pour implémenter ces améliorations et obtenir un arbre comme celui-ci :*

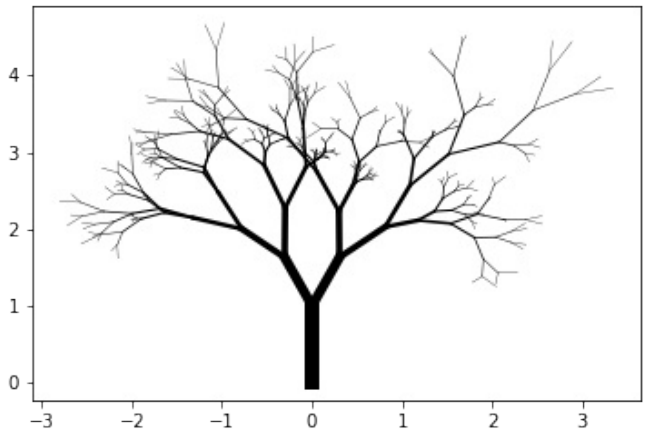

Figure 8. Arbre fractal obtenu avec l'introduction de facteurs aléatoires