DEVOIR MAISON D'INFORMATIQUE 1
Ensemble de Mandelbrot

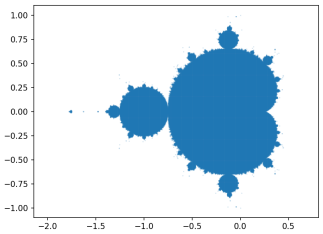
Ce devoir maison est facultatif et non noté, et doit être pris comme une opportunité pour s'entrainer. Vous pouvez déposer votre travail (réalisé sur machine) sur cahier de prepa avant le 5 janvier. N'hésitez pas à me poser vos questions par mail (pompigne@crans.org) en cas de blocage.

1 Tracé de l'ensemble de Mandelbrot

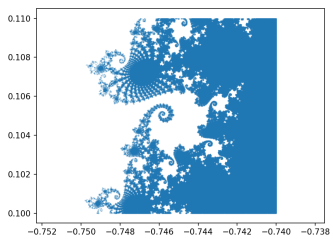
Notre objectif est de tracer le célèbre ensemble fractal de Mandelbrot. Celui-ci est défini comme l'ensemble des points z du plan complexe tels que la suite (u_n) définie par $u_0 = z,\ u_{n+1} = u_n^2 + z$ a son module borné par 2.

Pour savoir si un point donné est dans l'ensemble, nous allons donc calculer les termes de cette suite et regarder si leur module dépasse 2. Bien entendu, on ne peut pas ainsi vérifier l'infinité des termes de la suite, nous allons donc nous limiter aux N premiers termes de la suite, avec N suffisamment grand pour avoir une bonne approximation de l'ensemble.

1. Importer les bibliothèques `numpy` sous le nom `np` et `matplotlib.pyplot` sous le nom `plt`.
2. Définir une variable `N` de valeur 200.
3. Écrire une fonction `appartient` prenant en argument un nombre complexe z, renvoyant `True` si les N premiers termes de la suite (u_n) correspondante sont tous de module inférieur ou égal à 2, et `False` sinon.
   Indication : Un nombre complexe se se note en Python sous la forme `a + bj`, où `a` et `b` sont des flottants, et se manipule exactement comme un nombre flottant en Python. On peut obtenir son module en utilisant la fonction `abs`.
   Vous pouvez tester votre fonction `appartient` sur les complexes suivants :
   - 0 ; 1j ; -1+0.25j doivent renvoyer `True`
   - 1 ; -1 + 1j ; -0.75+0.2j doivent renvoyer `False`
4. Nous allons à présent tracer l'ensemble de Mandelbrot entre les abscisses $a < b$, et les ordonnées $c < d$. Pour cela, on divise chacun de ces intervalles en 1000 avec la fonction `linspace` de `numpy`, puis pour chaque point (x, y) du rectangle ainsi formé, on considère le complexe $x + y * 1j$ et on l'ajoute aux points à tracer si appartient renvoie `True` dessus.
   Écrire une fonction `afficher_mandelbrot` prenant en argument les quatre bornes `a, b, c, d` et réalisant ce tracé. On utilisera une instruction de la forme `plt.plot(X, Y, ".", ms = 0.2)` pour tracer les points sous la forme de ronds fins, qu'on précédera par `plt.axis('equal')` pour avoir la même échelle en abscisses et ordonnées. On n'oubliera pas de terminer la fonction par un appel `plt.show()`.
   `afficher_mandelbrot(-2,1,-1,1)` doit donner la figure suivante (au bout d'un certain temps) :



   et `afficher_mandelbrot(-0.75,-0.74,0.1,0.11)` doit donner la figure suivante :



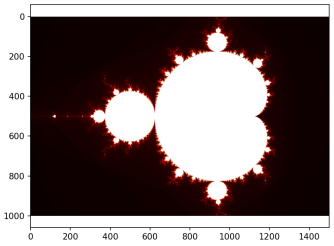
2 Tracé en couleur

L'affichage sans couleur de l'ensemble de Mandelbrot présente déjà une structure extrêmement riche, mais on peut souhaiter le détailler encore en colorant les points en dehors de l'ensemble selon la vitesse à laquelle leur suite (u_n) associée dépasse 2 en module.
Pour cela, on définit le **rang** d'un complexe z comme le premier entier $k < N$ tel que $|u_k| > 2$, ou N si un tel k n'existe pas. Il ne reste plus qu'à afficher chaque point avec une couleur donnée par son rang.

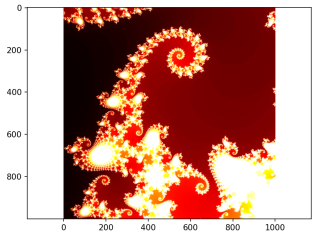
1. Écrire une fonction `rang` prenant en argument un nombre complexe et renvoyant son rang.
   Cette fonction doit en particulier passer les tests suivants :
   - 3 est de rang 0
   - 2 est de rang 1
   - 1 est de rang 2
   - $-0.75 + 0.2j$ est de rang 15
   - $-1 + 0.25j$ est de rang N
2. Pour obtenir un rendu plus propre et détaillé, nous allons afficher une image plutôt que des points. Cette image sera représentée par un tableau numpy de h lignes et l colonnes. Le **pixel** `img[i][j]` de l'image `img` est la case située à la ligne `i` et la colonne `j` dans l'image. Attention, les lignes seront affichées de haut en bas, et les colonnes de gauche à droite, de sorte que :
   - `img[0][0]` est le pixel en haut à gauche de l'image, qui doit donc correspondre au point de coordonnées (a, d) ;
   - `img[h-1][0]` doit correspondre au point (a, c) ;
   - `img[0][l-1]` doit correspondre au point (b, d) ;
   - `img[h-1][l-1]` doit correspondre au point (b, c).

   Déterminer par des formules affines l'abscisse et l'ordonnée du point correspondant au pixel sur la ligne `i` et la colonne `j`.
3. Écrire une fonction `afficher_mandelbrot_couleur` prenant les mêmes arguments que `afficher_mandelbrot`, et affichant l'image où chaque pixel a la valeur du rang du point correspondant. Pour cela :
   - on prendra `h = 1000` et `l = int(h*(b-a)/(d-c))` ;
   - on créera un tableau numpy `img` de h lignes et l colonnes contenant initialement des zéros avec `img = np.zeros((h,l))` ;
   - on remplira chaque case de `img` avec le rang du point correspondant ;
   - on affichera `img` avec `plt.imshow(img, cmap = "hot")` suivi de `plt.show()`.

   `afficher_mandelbrot_couleur(-2,1,-1,1)` doit afficher la figure suivante :



   et `afficher_mandelbrot_couleur(-0.748,-0.744,0.102,0.106)` doit afficher :



4. Afficher votre partie préférée de l'ensemble de Mandelbrot.