Informatique en CPGE (2018-2019)
TP 9 : méthodes de dichotomie et de Newton

Exercice 1

1. Programmer l'algorithme de dichotomie pour résoudre l'équation $\cos x - x = 0$ sur l'intervalle $[0; 1]$ à 10^{-2} près, puis à 10^{-4} près et enfin à 10^{-8} près. Pour cela, écrire une fonction `zero_dic` qui prend en paramètre une fonction f, trois réels a, b, eps et qui renvoie une solution approchée à eps près, et le nombre d'itérations effectuées. Ecrire la définition de la fonction `f` qui renvoie $\cos x - x$ et exécuter la fonction `zero_dic`.

2. Il a été vu en cours que la boucle est exécutée k fois si et seulement si

$$\ln\left(\frac{b-a}{\epsilon}\right) \leq k \ln 2 < \ln 2 + \ln\left(\frac{b-a}{\epsilon}\right)$$

   Si $b - a = 1$ et $\epsilon = 10^{-p}$, montrer qu'on obtient

$$p \ln 10 / \ln 2 \leq k < 1 + p \ln 10 / \ln 2$$

   k est donc égal à la partie entière de $p \ln 10 / \ln 2$ augmentée de 1.

3. Compléter le programme afin d'afficher la solution à 10^{-p} près pour p entier variant de 1 à 10, ainsi que, pour chaque valeur de p, le nombre d'itérations effectuées et la valeur de int$(p \ln 10 / \ln 2) + 1$.

4. Pour comparer avec la solution obtenue par la fonction `bisect` du module `scipy.optimize`, on écrira le code suivant :

```
from scipy.optimize import bisect
print("sol avec scipy:", bisect(f, a, b))
```

Exercice 2

On reprend l'exercice 1 mais cette fois en utilisant la méthode de Newton.

1. Ecrire la fonction `f(x)` qui renvoie $\cos x - x$, puis écrire la fonction `df(x)` qui renvoie $f'(x)$.

2. Ecrire ensuite une fonction `newton(f,x,df,eps,N=100)` qui renvoie une valeur approchée de la solution et le nombre d'itérations effectuées. Le paramètre $N = 100$ permet d'arrêter le passage dans la boucle lorsque la suite des valeurs de x ne converge pas vers la solution.

3. Tester le programme avec différentes valeurs initiales de x pour constater la convergence ou la non convergence de la méthode, par exemple $x = 0$, $x = -3$, $x = -5$, $x = 3$, $x = 10$.

4. On choisit une valeur initiale $x = 0$ et on souhaite vérifier que, en raison de la convergence quadratique, il suffira d'environ $d = \ln n / \ln 2$ itérations pour parvenir à n décimales exactes. Pour cela, comme dans l'exercice 1, modifier le programme pour afficher la solution à 10^{-n} près pour n entier variant de 1 à 25, ainsi que, pour chaque valeur de n, le nombre d'itérations effectuées et la valeur de int$(\log_2 n)+1$.

5. Pour comparer avec la solution obtenue par la fonction `newton` du module `scipy.optimize`, on écrira le code suivant :

```
from scipy.optimize import newton
print("sol avec scipy:", newton(f, x))
```

Exercice 3

Nous admettons ici que la méthode de Newton peut s'utiliser dans un cadre plus général. Par exemple pour rechercher les solutions de l'équation $z^3 = 1$ dans l'ensemble des nombres complexes.

On considère les fonctions f et f' définies dans le plan complexe respectivement par $f(z) = z^3 - 1$ et $f'(z) = 3z^2$.

1. Comme dans le programme de l'exercice 2, on écrit les différentes fonctions mais en utilisant z à la place de x, soit `f(z)`, `df(z)` et `newton(f,z,df,eps,N=100)`. La fonction `newton` renvoie juste la valeur de `z`.

2. Afin d'éviter le risque de diviser par zéro ou par un nombre très petit, ajouter dans la boucle `while` un test qui permet d'interrompre le programme si par exemple $|x| < 10^{-6}$ et renvoie alors la valeur de `z`.

3. On connaît les trois solutions qui sont 1, $-\frac{1}{2}+\mathrm{i}\frac{\sqrt{3}}{2}$ et $-\frac{1}{2}-\mathrm{i}\frac{\sqrt{3}}{2}$; affecter ces trois valeurs respectivement à trois variables nommées `sol1`, `sol2` et `sol3`. Pour cela, on utilise par exemple l'instruction `sol2=complex(-0.5,0.5*3**(0.5))`.

4. On souhaite tester la convergence de la méthode pour différentes valeurs initiales de z mais aussi savoir vers laquelle des trois solutions cette convergence a lieu. On va écrire les valeurs des parties réelles et imaginaires x et y de chaque z initial dans des fichiers différents suivant que la convergence a lieu vers `sol1`, `sol2` ou `sol3`, ou s'il n'y a pas de convergence. On doit donc ouvrir quatre fichiers en écriture :

```
fic0 = open('sol0.dat', 'w') # pas de convergence
fic1 = open('sol1.dat', 'w') # converge vers sol1
fic2 = open('sol2.dat', 'w') # converge vers sol2
fic3 = open('sol3.dat', 'w') # converge vers sol3
```

5. A l'aide de deux boucles `while` imbriquées, on fait varier x et y dans l'intervalle $[-1,5; 1,5]$ avec un pas de 0,004. Pour chaque couple $(x; y)$, on affecte à z le complexe $x+\mathrm{i}y$ (rappel de l'instruction : `z=complex(x,y))` et à une variable `sol` la valeur renvoyée par la fonction `newton` avec eps$=10^{-2}$. Ensuite il faut écrire dans le bon fichier les valeurs de x et y, par exemple :

```
if abs(sol-sol1) < 10**(-2):
    fic1.write(str(x) + '\t' + str(y) + '\n')
```

Ne pas oublier de fermer les quatre fichiers.

6. Finalement, on ouvre le logiciel **Gnuplot** et on écrit les commandes :

gnuplot > set xrange [-2 :2]

gnuplot > plot 'adresse du fichier \sol1.dat' w d title "", 'adresse du fichier \sol2.dat' w d title "", 'adresse du fichier \sol3.dat' w d title ""

(w d = with dots, pour placer les points, et title "" pour ne pas avoir de titre)

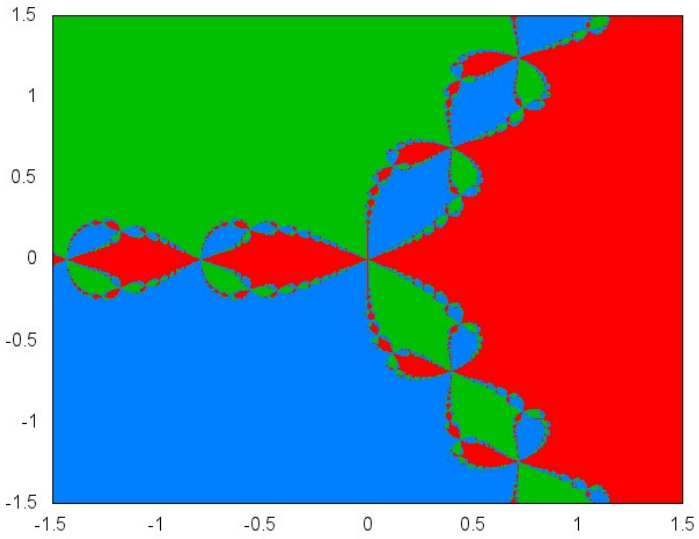
Pour sauvegarder l'image obtenue, on écrit au préalable dans **Gnuplot** les commandes :

gnuplot > set terminal jpeg

gnuplot > set output 'fractale_Newton.jpg'

gnuplot > plot ...

Et on obtient la figure suivante :



Exercice 4
Utilisation de **numpy** et **matplotlib**.
On commence par importer les modules :

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

1. On résout comme dans l'exercice précédent l'équation $z^3 - 1 = 0$ dans le plan complexe.
   Ecrire les fonction `f(z)`, `df(z)` et `newton(f,z,df,eps,N=100)`. La fonction `newton` renvoie les valeurs de z et du nombre d'itérations.
   Compléter le programme avec le code suivant :

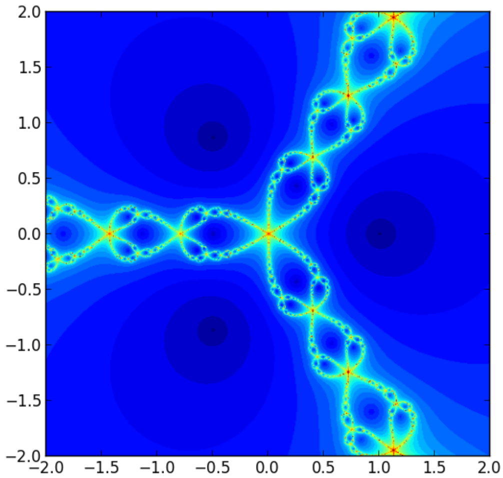
```
newton_vect = np.vectorize(newton)
K = 500
xliste = np.linspace(-2,2,K)
yliste = np.linspace(-2,2,K)
x, y = np.meshgrid(xliste,yliste)
```

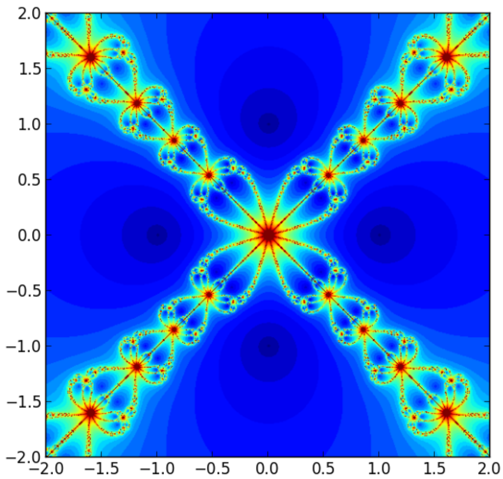
   Puis le code suivant :

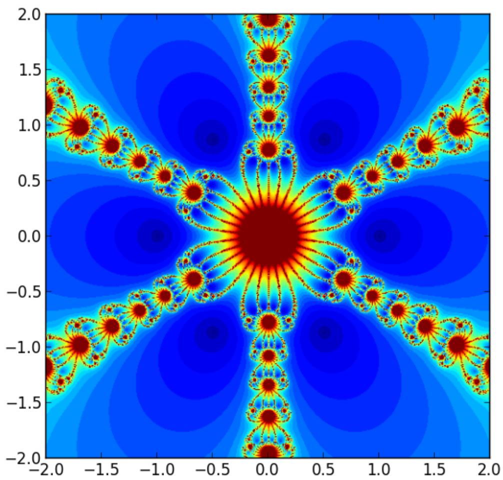
```
tab = newton_vect(f, x+y*1j, df, 1e-3)[1]
plt.clf
plt.imshow(tab, extent=[-2,2,-2,2])
plt.show()
```

   Tester le programme.
2. Modifier les fonctions `f(z)` et `df(z)` pour résoudre l'équation $z^4 - 1 = 0$.
3. Modifier les fonctions `f(z)` et `df(z)` pour résoudre l'équation $z^6 - 1 = 0$.

On obtient les figures suivantes :


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0