DUMIRE Thibault
MOREL Rémi
Date du TP (22/11/2021)

Rapport de TP1 – Visualisation de fractales

I. Introduction

L'objectif de ce TP est de visualiser les fractales de Mandelbrot. Pour cela, nous avons défini une classe nombre complexe pour utiliser la suite définissant une fractale appartenant à l'ensemble des complexes. Nous avons également ajouté des grilles complexes nous permettant de visualiser les fractales à l'aide de la librairie matplotlib.

II. Préliminaires : les nombres complexes

1. Question (1).

Définition du constructeur de la classe NombreComplexe :

```
def __init__(self, real, imag):
    self.real = real
    self.imag = imag
```

2. Question (2).

Ajout de la méthode module :

```
def module(self):
    return sqrt((self.real)**2+(self.imag)**2)
```

3. Question (3).

Ajout de la méthode permettant l'affichage d'un nombre complexe :

```
def __str__(self):
    ch=''
    if (self.real!=0):
        ch+=str(self.real)
    if (self.imag!=0):
        if (self.imag>0):
            ch+=' + '+str(self.imag)+'i'
        else:
            ch+=' - '+str(abs(self.imag))+'i'
    return ch
```

4. Question (4).

Ajout des méthodes addition, soustraction et multiplication surchargeant respectivement + , - et * :

```
def __add__(self,nc):
    return NombreComplexe(self.real+nc.real,self.imag+nc.imag)

def __sub__(self,nc):
    return NombreComplexe(self.real-nc.real,self.imag-nc.imag)

def __mul__(self,nc):
    return NombreComplexe(self.real*nc.real-self.imag*nc.imag,self.real*nc.imag+self.imag*nc.real)
```

5. Question (5).

Ajout de la méthode permettant de faire la puissance n-ième d'un nombre complexe :

```
def __pow__(self,n):
    res=self
    for i in range(1,n,1):
        res=res*self
    if (n==0):
        res=NombreComplexe(1,0)
    return res
```

6. Tests

Voici le résultat des tests pour l'ensemble de la partie 2 :

```
.............
----------------------------------------------------------------------
Ran 13 tests in 0.005s

OK
An exception has occurred, use %tb to see the full traceback.

SystemExit: False
```

III. Le plan complexe comme une image

1. Question (1).

Les valeurs sont : $pas_x = 3/(n\_x - 1)$, $pas_y = -2/(n\_y - 1)$ et Zd = 2 - i

2. Question (2).

Ajout de la fonction nombre_complexe :

```
Zd=NombreComplexe(2,-1)
def nombre_complexe(k,l,n_y,n_x):
    return NombreComplexe(l*(3/(n_x - 1))-Zd.real,k*(-2/(n_y - 1))-Zd.imag)
```

3. Question (3).

Ajout de la fonction grille_complexe :

```
def grille_complexe(n_y,n_x):
    grille=[]
    for i in range(0,n_y,1):
        l=[]
        for j in range(0,n_x,1):
            l+=[nombre_complexe(i,j,n_y,n_x)]
        grille+=[l]
    return grille
```

4. Tests

Voici les résultats des tests pour l'ensemble de la partie 3 :

```
....................
----------------------------------------------------------------------
Ran 20 tests in 6.116s

OK
An exception has occurred, use %tb to see the full traceback.

SystemExit: False
```

IV. Visualisation à l'aide de la librairie matplotlib

1. Question (1).

Cette ligne import une librairie graphique, et lui donne un alias (plt)

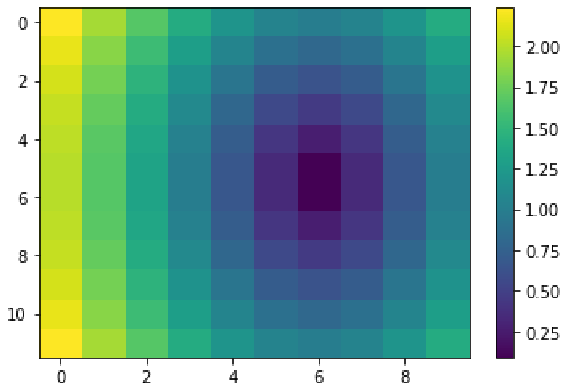
2. Question (2).

Création de tableau_module :

```
n_y = 12 ; n_x = 10
grille = grille_complexe(n_y,n_x)
tableau_module = copy.deepcopy(grille)
for i in range(0,len(tableau_module),1):
    for j in range(0,len(tableau_module[0]),1):
        tableau_module[i][j]=grille[i][j].module()
```

3. Question (3).

Voici l'affichage :



V. Algorithme de calcul de la fractale

1. Question (1).

Ajout de la fonction est_divergente :

```
def est_divergente(c,N):
    z=NombreComplexe(0,0)
    cpt=0
    while(cpt<N and z.module()<=2):
        z=z**2+c
        cpt+=1
    return (cpt<N)
```

2. Question (2).

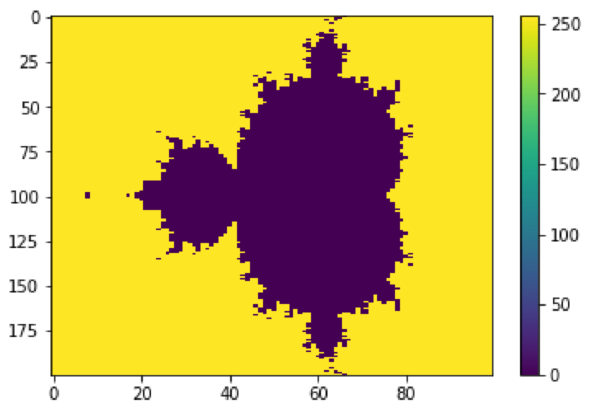
Ajout de la fonction image_mandelbrot :

```
def image_mandelbrot(n_y,n_x,N):
    grille = grille_complexe(n_y,n_x)
    for i in range(0,len(grille),1):
        for j in range(0,len(grille[0]),1):
            if (est_divergente(grille[i][j],N)==True):
                grille[i][j]=255
            else:
                grille[i][j]=0
    return grille
```

3. Question (3).

Création de l'image puis affichage :

```
image = image_mandelbrot(200,100,20)
plt.figure()
plt.imshow(image, aspect='auto')
plt.colorbar()
plt.show()
```



4. Question (4).

n_y et n_x font varier la résolution de l'image, N influe sur la précision de la divergence.

5. Question (5).

Voici une ébauche à la question :

```
def est_divergente_couleur(c):
    z=NombreComplexe(0,0)
    cpt=0
    while(z.module()<=2):
        z=z**2+c
        cpt+=1
    return [z.module()>2,cpt]

def image_mandelbrot_couleur(n_y,n_x):
    grille = grille_complexe(n_y,n_x)
    for i in range(0,len(grille),1):
        for j in range(0,len(grille[0]),1):
            tab=est_divergente_couleur(grille[i][j])
            if (tab[0]==True):
                grille[i][j]=255
            else:
                grille[i][j]=tab[1]
    return grille

image_couleur = image_mandelbrot_couleur(200,100)
plt.figure()
plt.imshow(image_couleur, aspect='auto')
plt.colorbar()
plt.show()
```

6. Tests

Nous n'avons pas eu le temps d'exécuter le fichier test_mandelbrot