Projet Calcul Scientifique

Racines d'un polynôme

24 Juin 2019

CAMPBELL Dorian
DERLANGE Benjamin
RODRIGUEZ Manuela

Groupe BS4
Intervenant : GOURDON Emmanuel

Sujet : Recherche des racines du polynôme $P(z) = z^8 - 12z^4 - 64$, c'est-à-dire des z^* qui vérifient $P(z^*) = 0$.

Questions

1) Trouver analytiquement les racines du polynôme P

Afin de trouver les racines de manière analytique du polynôme, on pose $x = z^4$ et on obtient :

$$P(x) = x^2 - 12x - 64$$

$$\Delta = 12^2 + 4 \times 64 = 400$$

On a donc :

$$x_1 = \frac{12 - \sqrt{\Delta}}{2} = -4 \quad ; \quad x_2 = \frac{12 + \sqrt{\Delta}}{2} = 16$$

Et :

$$z_1{}^4 = -4 \quad ; \quad z_2{}^4 = 16$$

Dans le plan complexe, on sait que pour tout $z^4 = y$, $\forall y \in \mathbb{R}$, alors il suffit de trouver z tel que $z^4 = y$ et $-z, \bar{z}, -\bar{z}$ seront aussi solution de l'équation.

$$z_1 = 1 - i \quad ; \quad -z_1 = -1 + i \quad ; \quad \overline{z_1} = 1 + i \quad ; \quad -\overline{z_1} = -1 - i$$

Et

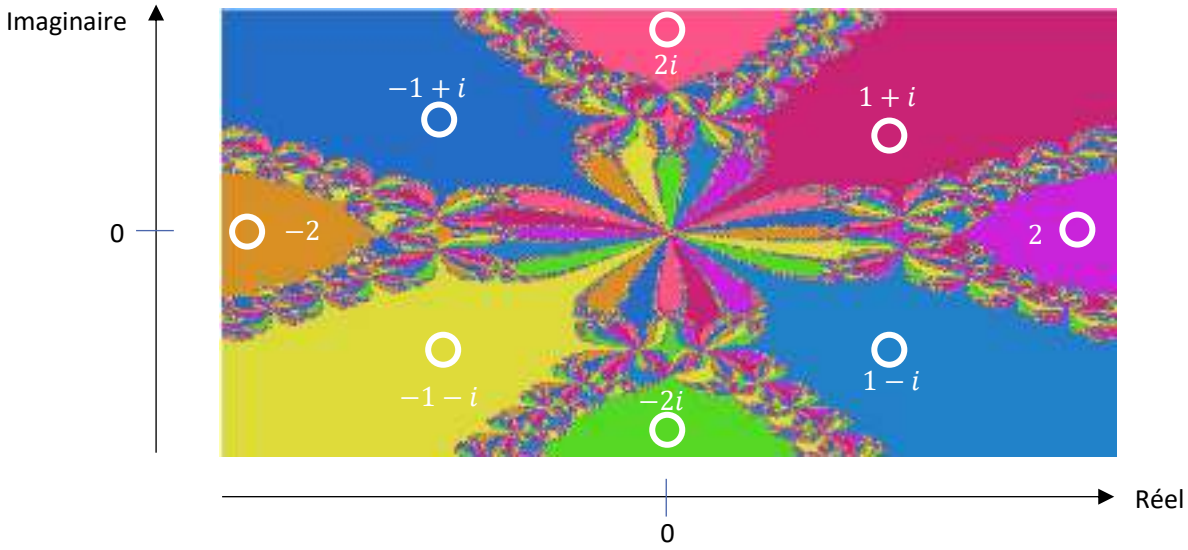
$$z_2 = 2 \quad ; \quad -z_2 = -2 \quad ; \quad \overline{z_2} = 2i \quad ; \quad -\overline{z_2} = -2i$$

$$z^* \in \{1 - i\,;\, -1 + i\,;\, 1 + i\,;\, -1 - i\,;\, 2\,;\, -2\,;\, 2i\,;\, -2i\}$$

2) Trouver numériquement les racines du polynôme P

Pour trouver les racines du polynôme $P(z)$ de manière itérative, on utilise l'algorithme de Newton avec un tolérance $\varepsilon = 1.10^{-4}$ et un pas h=1.10^{-4} pour assurer une convergence plus rapide de l'algorithme vers la solution.

Voir annexe 1



En prenant des conditions initiales proche des huit racines (cercle blanc) on se rend compte que les valeurs renvoyées sont les racines elles-mêmes car les zones proches des racines sont des zones stables.

Voir graphique ci-dessus fractale $[-2.5,2.5] \times [-2.5,2.5]$

Voir annexe 3 pour construire la fractale

3) Considération d'un grand nombre de conditions initiales dans un carré du plan complexe

Dans un carré de $[-1,1] \times [-1,1]$ du plan complexe, on se rend compte que les solutions sont identiques dans certaines zones avec des conditions initiales différentes. En revanche, on remarque que certaines zones renvoient des valeurs différentes (*Voir tableau ci-dessous*).

Voir annexe 2 pour construire le tableau

Ceci peut s'expliquer car dans le plan complexe, de nombreux points sont associés à chaque racine du polynôme. Cela introduit le concept de bassin d'attraction. Ainsi, le bassin d'attraction de chaque solution de $f(z) = 0$ est l'ensemble des valeurs de z_0 pour lesquelles la suite converge vers cette solution.

condition initiale	solution
(-0.98+0.52j)	(-1+1j)
(-0.98+0.54j)	(-1+1j)
(-0.98+0.56j)	(-1+1j)
(-0.98+0.580000000000	(-1+1j)
(-0.98+0.600000000000	(-1+1j)
(-0.98+0.620000000000	(-1+1j)
(-0.98+0.640000000000	(-1+1j)
(-0.98+0.660000000000	(-1+1j)
(-0.98+0.679999999999	(-1+1j)
(-0.98+0.7j)	(-1+1j)
(-0.98+0.72j)	(-1+1j)
(-0.98+0.74j)	(-1+1j)
(-0.98+0.76j)	(-1+1j)
(-0.98+0.78j)	(-1+1j)
(-0.98+0.8j)	(-1+1j)
(-0.98+0.820000000000	(-1+1j)
(-0.98+0.840000000000	(-1+1j)
(-0.98+0.860000000000	(-1+1j)
(-0.98+0.880000000000	(-1+1j)
(-0.98+0.900000000000	(-1+1j)
(-0.98+0.919999999999	(-1+1j)
(-0.98+0.94j)	(-1+1j)
(-0.98+0.96j)	(-1+1j)
(-0.98+0.98j)	(-1+1j)

Zone de solution homogène

condition initiale	solution
(0.94 0.54j)	(1 1j)
(-0.94-0.52j)	-2j
(-0.94-0.5j)	(-2+0j)
(-0.94-0.48j)	(-2+0j)
(-0.94-0.459999999999	(-1+1j)
(0.94 0.439999999999	2j
(-0.94-0.420000000000	(-1+1j)
(-0.94-0.4j)	(2+0j)
(-0.94-0.38j)	(-0-2j)
(-0.94-0.36j)	(-1-1j)
(0.94 0.339999999999	(2-0j)
(-0.94-0.319999999999	(-1-1j)
(-0.94-0.299999999999	(-0-2j)
(-0.94-0.28j)	-2j
(-0.94-0.26j)	(1-1j)
(0.94 0.24j)	(1-1j)

Zone de solution chaotique

4) Tracer les bassins d'attraction des différentes solutions

Afin de représenter graphiquement les bassins d'attraction des différentes solutions du polynôme P, on a attribué une couleur à chaque condition initiale prise selon la solution qu'on trouvait avec cette condition initiale. Ainsi, on a établi les couleurs suivantes :

- Bleu clair : si $z = 1 - i$
- Bleu : si $z = -1 + i$
- Magenta : si $z = 1 + i$
- Jaune : si $z = -1 - i$
- Violet : si $z = 2$
- Orange : si $z = -2$
- Lime : si $z = -2i$
- Rose : si $z = 2i$

$$fractale\ [-1,1] \times [-1,1]$$

FIGURE 1. FRACTALE POUR LES CONDITIONS INITIALES DANS [-1,1]X[-1,1]

$$fractale\ [-2.5,2.5] \times [-2.5,2.5]$$

FIGURE 2. FRACTALE POUR LES CONDITIONS INITIALES DANS [-2.5,2.5]X[-2.5,2.5]

$$fractale\ [-6,6] \times [-6,6]$$

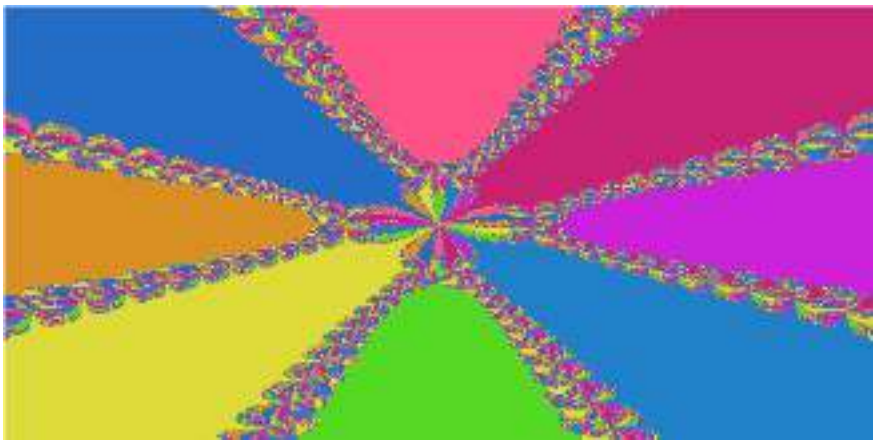


FIGURE 3. FRACTALE POUR LES CONDITIONS INITIALES DANS [-6,6]X[-6,6]

Voir annexe 3

5) Retracer les bassins d'attraction en mettant des nuances pour chaque couleur

Afin de représenter la vitesse de convergence de l'algorithme selon la condition initiale prise, on a implémenté des nuances de chaque couleur. Dans cet ordre d'idées, plus la couleur est foncée plus le nombre d'itérations nécessaires pour arriver à une solution a été grand. Ainsi, on observe plus précisément qu'avec certaines conditions initiales la convergence est beaucoup plus rapide qu'avec d'autres.

$$fractale\ [-1,1] \times [-1,1]\ avec\ ombre$$

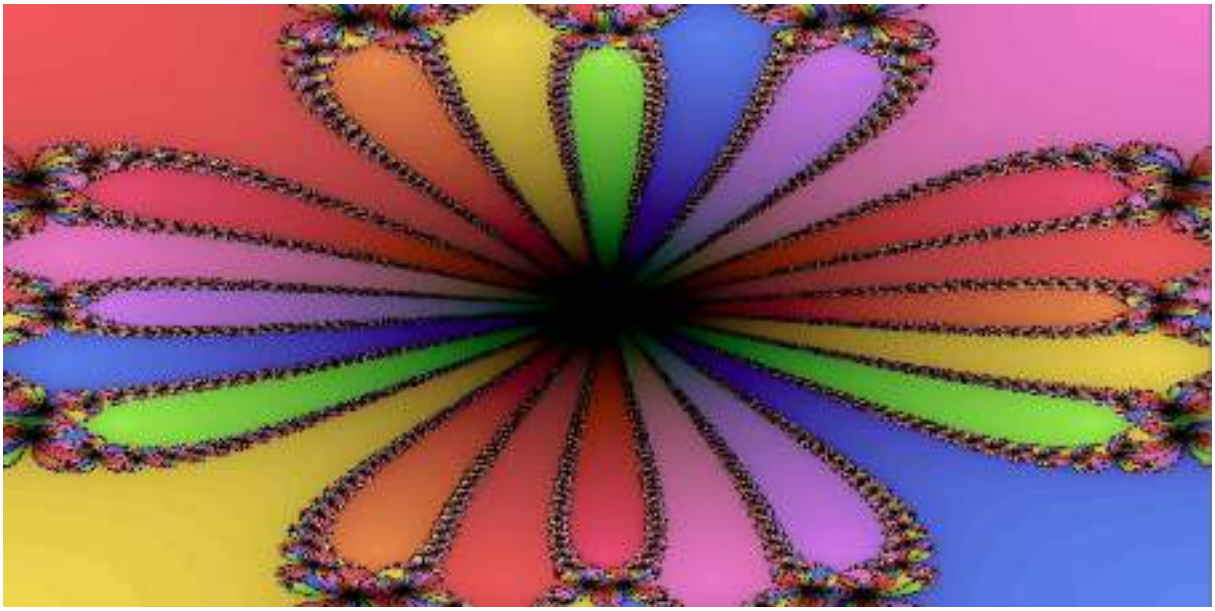


FIGURE 4. FRACTALE AVEC NUANCES POUR LES CONDITIONS INITIALES DANS [-1,1]X[-1,1]

$$fractale\ [-2.5,2.5] \times [-2.5,2.5]\ avec\ ombre$$

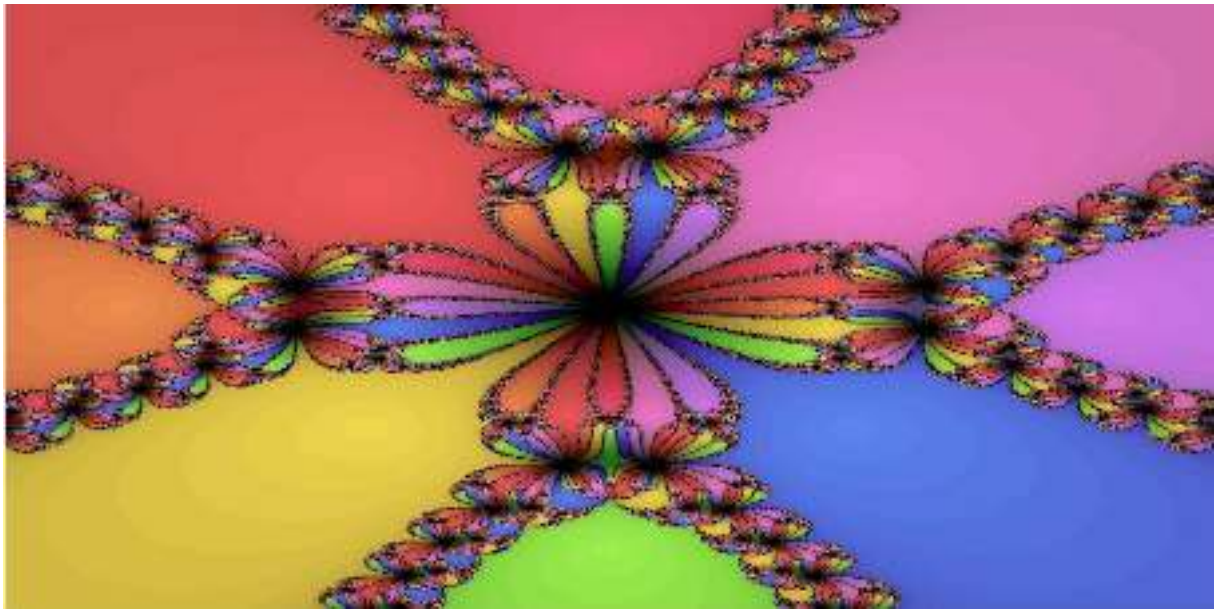


FIGURE 5. FRACTALE AVEC NUANCES POUR LES CONDITIONS INITIALES DANS [-2.5,2.5]X[-2.5,2.5]

$fractale\ [-6{,}6] \times [-6{,}6]\ avec\ ombre$

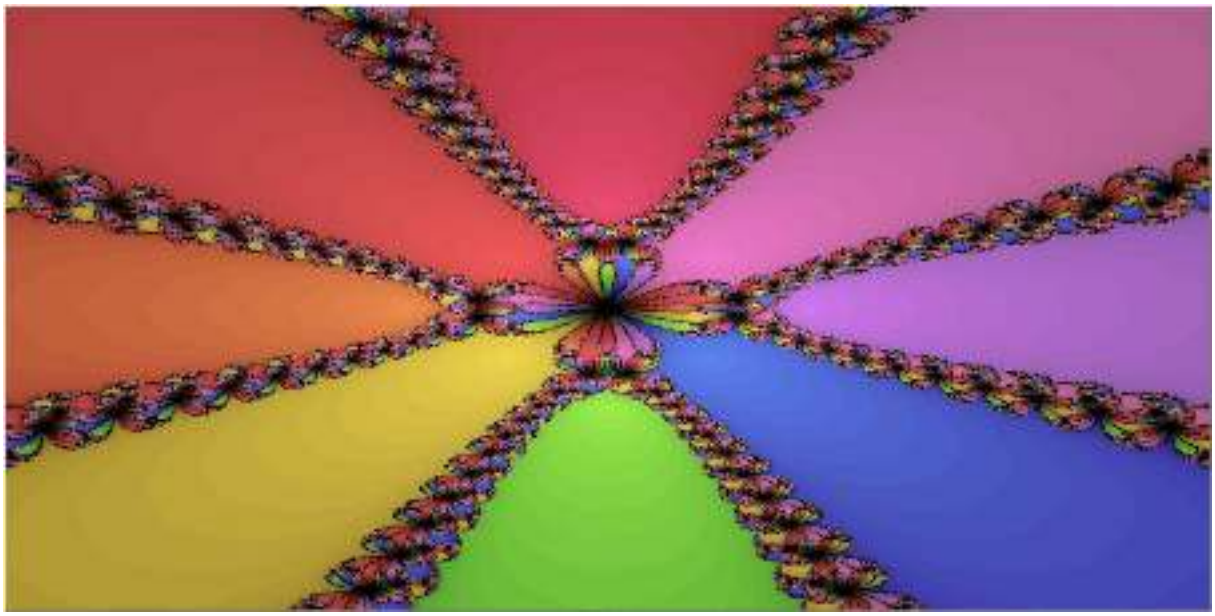


FIGURE 6. FRACTALE AVEC NUANCES POUR LES CONDITIONS INITIALES DANS [-6,6]X[-6,6]

Voir annexe 4

Pour conclure, trouver de manière algorithmique les racines d'un polynôme n'est pas quelque chose de simple. En effet, les fractales ci-dessus montrent que même pour un polynôme qui peut se résoudre simplement de manière analytique, les solutions doivent être approchées et on doit balayer un grand nombre de conditions initiales pour être sûr d'obtenir toutes les racines. Cela a pour effet négatif d'augmenter les temps de calcul et donc l'efficacité de l'algorithme.

Annexes

Tout le code est réalisé sous python 3.8.0

Les bibliothèques utilisées sont :

```
import matplotlib.pyplot as plt
import pixel

#la bibliothèque pixel se télécharge sur le site :
http://www.grappa.univ-lille3.fr/~coulom/Python/_downloads/pixel.py
```

Annexe 1 :

```
def Newton(f,x0,epsilon=1e-4,h=1e-4):
    i=0                              # Initialisation
    x = x0                           # Initialisation
    while abs(f(x)) > epsilon:       # Tant que |f(x)| > epsilon choisi
        i=i+1                        # calcul du nombre d'itération
        derivee = (f(x+h) - f(x))/ h # dérivée en x_k par taux d'accroissement
        x = x - f(x)/derivee         # nouvelle valeur x_{k+1} de x
    return x,i
```

Annexe 2 :

```
for i in range(1,100):                    # boucle de 100 elements
    for j in range(1,100):                # boucle de 100 éléments
        z0=complex(-1+2/100*i,-1+2/100*j) # on créer un nombre complexe
                                          correspondant aux conditions initiales
        if z0 != 0:                       # si z0 est différent de 0
            z,ite=Newton(P,z0)            # on récupère la solution et le
                                          nombre d'itération
            ite_liste=ite_liste+[ite]
            racine=racine+[complex(round(z.real, 1), round(z.imag, 1))]
            z0_liste=z0_liste+[z0]
```

Annexe 3 :

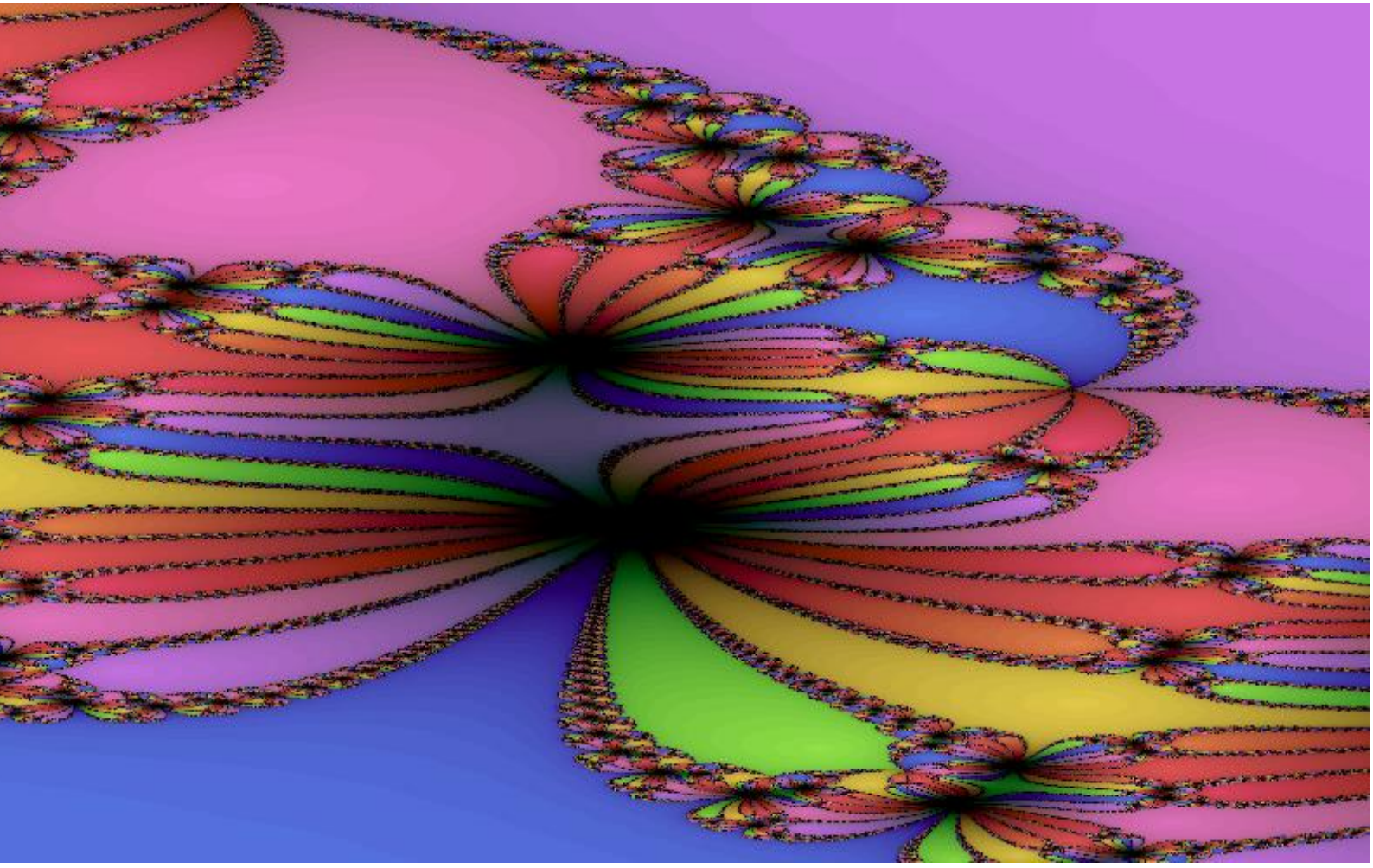
```
for i in range(1,400):
    for j in range(1,200):
        z0=complex(-1+2/400*i,-1+2/200*j)
        if z0 != 0:
            z,ite=Newton(P,z0)
            real=real+[z0.real]
            imag=imag+[z0.imag]
            if round(z.real, 1)==1:
                if round(z.imag, 1)==1:
                    pixel.marquer(i, j, 1)
                    pixel.couleur((98)/255 , (147)/255 , (255)/255)  #bleu
                    pixel.afficher(0)
                elif round(z.imag, 1)==-1:
                    pixel.marquer(i, j, 1)
                    pixel.couleur((255)/255 , (125)/255 , (219)/255) #rose
                    pixel.afficher(0)
            elif round(z.real, 1)==-1:
                if round(z.imag, 1)==1:
                    pixel.marquer(i, j, 1)
                    pixel.couleur((253)/255 , (229)/255 , (83)/255)  #jaune
                    pixel.afficher(0)
                elif round(z.imag, 1)==-1:
```

```
                pixel.marquer(i, j, 1)
                pixel.couleur((255)/255 , (95)/255 , (95)/255)    #mauve
                pixel.afficher(0)
            elif round(z.real, 1)==2:
                if round(z.imag, 1)==0:
                    pixel.marquer(i, j, 250)
                    pixel.couleur((225)/255 , (125)/255 , (252)/255) #violet
                    pixel.afficher(0)
            elif round(z.real, 1)==-2:
                if round(z.imag, 1)==0:
                    pixel.marquer(i, j, 25)
                    pixel.couleur((253)/255 , (156)/255 , (83)/255)  #orange
                    pixel.afficher(0)
            elif round(z.real, 1)==0:
                if round(z.imag, 1)==2:
                    pixel.marquer(i, j, 75)
                    pixel.couleur((174)/255 , (253)/255 , (83)/255)  #lime
                    pixel.afficher(0)
                elif round(z.imag, 1)==-2:
                    pixel.marquer(i, j, 125)
                    pixel.couleur((253)/255 , (83)/255 , (137)/255)  #magenta
                    pixel.afficher(0)
```

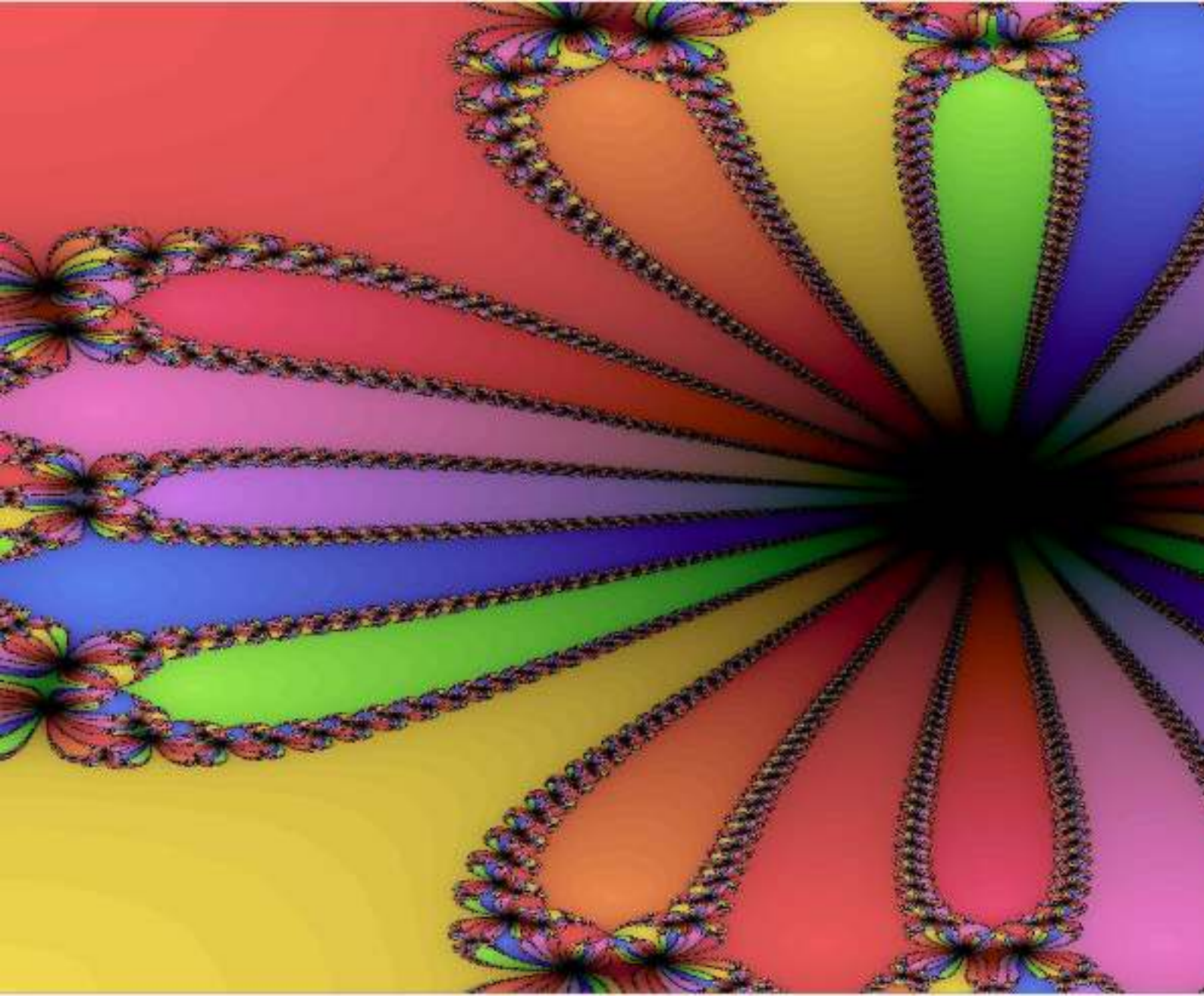
Annexe 4 :

```
for i in range(1,1600):
    for j in range(1,800):
        z0=complex(-1+2/1600*i,-1+2/800*j)
        if z0 != 0:
            z,ite=Newton(P,z0)
            real=real+[z0.real]
            imag=imag+[z0.imag]
            if round(z.real, 1)==1:
                if round(z.imag, 1)==1:
                    pixel.marquer(i, j, 1)
                    pixel.couleur((98-ite*2)/255 , (147-ite*5)/255 , (255-
                    ite*5)/255) #bleu
                    pixel.afficher(0)
                elif round(z.imag, 1)==-1:
                    pixel.marquer(i, j, 1)
                    pixel.couleur((255-ite*5)/255 , (125-ite*2)/255 , (219-
                    ite*5)/255) #rose
                    pixel.afficher(0)
            elif round(z.real, 1)==-1:
                if round(z.imag, 1)==1:
                    pixel.marquer(i, j, 1)
                    pixel.couleur((253-ite*5)/255 , (229-ite*5)/255 , (83-
                    ite*2)/255) # jaune
                    pixel.afficher(0)
                elif round(z.imag, 1)==-1:
                    pixel.marquer(i, j, 1)
                    pixel.couleur((255-ite*5)/255 , (95-ite*2)/255 , (95-
                    ite*2)/255) #mauve
                    pixel.afficher(0)
            elif round(z.real, 1)==2:
                if round(z.imag, 1)==0:
                    pixel.marquer(i, j, 250)
                    pixel.couleur((225-ite*5)/255 , (125-ite*2)/255 , (252-
                    ite*5)/255) #violet
                    pixel.afficher(0)
```

```
elif round(z.real, 1)==-2:
    if round(z.imag, 1)==0:
        pixel.marquer(i, j, 25)
        pixel.couleur((253-ite*5)/255 , (156-ite*5)/255 , (83-
        ite*2)/255) # orange
        pixel.afficher(0)
elif round(z.real, 1)==0:
    if round(z.imag, 1)==2:
        pixel.marquer(i, j, 75)
        pixel.couleur((174-ite*5)/255 , (253-ite*5)/255 , (83-
        ite*2)/255) # lime
        pixel.afficher(0)
    elif round(z.imag, 1)==-2:
        pixel.marquer(i, j, 125)
        pixel.couleur((253-ite*5)/255 , (83-ite*2)/255 , (137-
        ite*5)/255) # rose/rouge
        pixel.afficher(0)
```



$fractale\ [1.3,1.5] \times [0,0.8], haute\ résolution, python$



$fractale\ [-1{,}1] \times [-1{,}1]\ coupé, haute\ résolution, python$