Revue de la Filière Mathématiques
RMS

770. Soit $P_n : t \mapsto \sum_{k=0}^{n} \frac{t^k}{k!}$.

a) **Tracer le graphe de** P_n **pour** n **allant de 2 à 7 sur l'intervalle qui vous paraît le plus judicieux. Que dire des racines de** P_n **?**

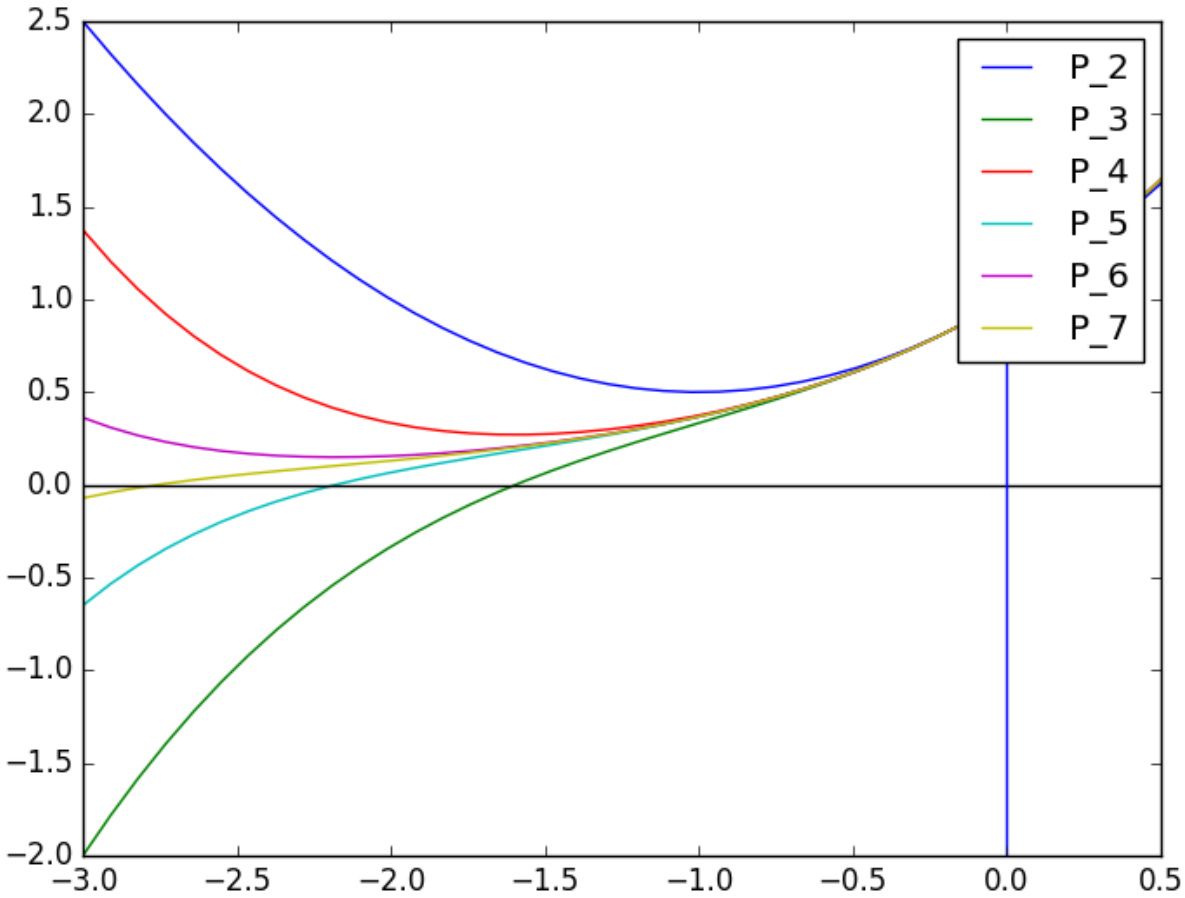
b) **Déterminer des valeurs approchées des racines complexes de** P_n **pour** n **allant de 2 à 7 en utilisant la méthode de Newton.**

c) **Représenter ces racines sur le plan complexe et commenter.**

d) **Montrer que** P_n **est scindé à racines simples sur C.**

Solution de Patrice Lassère et Georges Marty

a)



On peut conjecturer que P_n admet une unique racine réelle lorsque n est impair, et qu'il n'en a aucune lorsque n est pair.

b) La méthode de Newton est utilisée pour une fonction dérivable par rapport à la variable réelle ou complexe. Elle permet de construire une suite de valeurs convergeant rapidement, dans certaines conditions, vers un zéro de P_n, à partir d'une valeur approchée initialement choisie. La relation de récurrence est donnée par la formule

$$z_{p+1} = z_p - \frac{P_n(z_p)}{P_n'(z_p)}.$$

Dans notre étude, on a $P_n' = P_{n-1}$. Voici les valeurs approchées des racines des P_n qui ont été obtenues par cette méthode. La recherche s'est effectuée de manière autonome. Le programme a lui-même initialisé les recherches par des nombres aléatoires. Le tout n'a pris que quelques secondes.

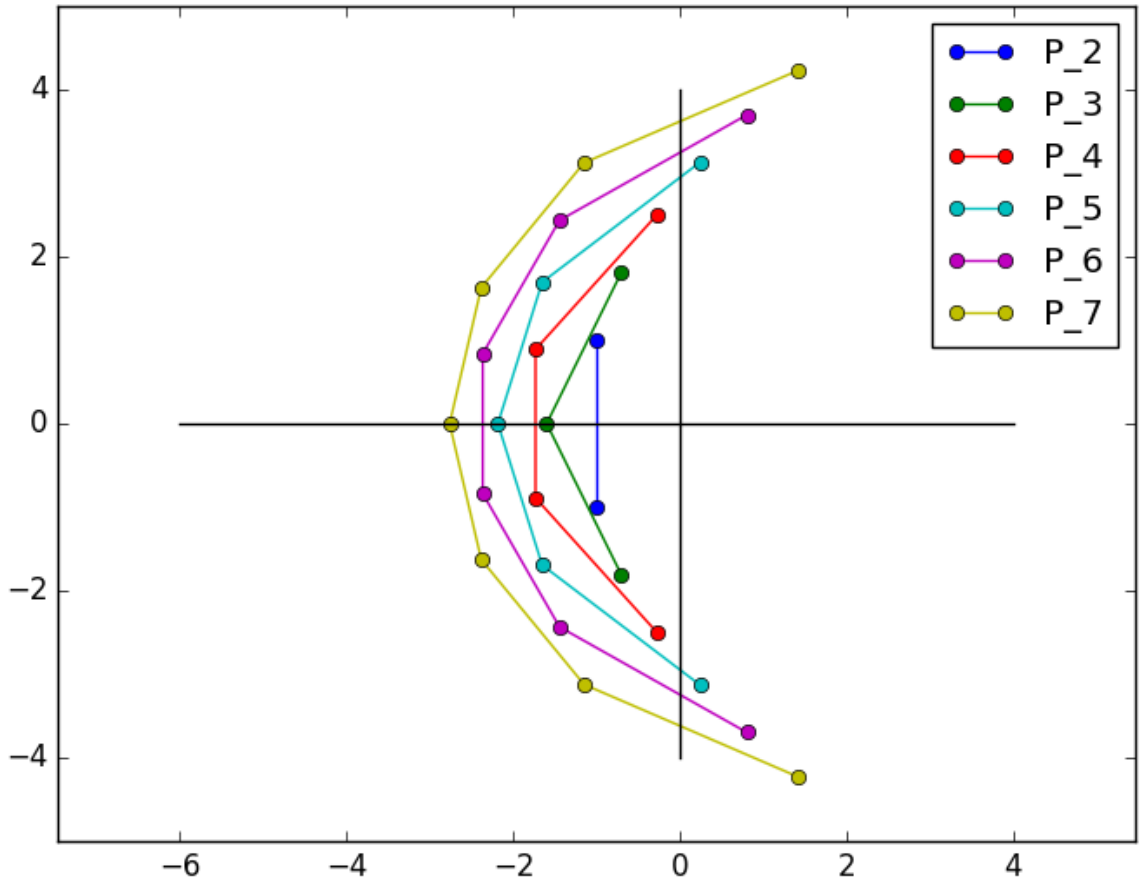
Racines de P_2 : -1.0 + 1.0 i ; -1.0 + -1.0 i ;

Racines de P_3 : -0.701 + 1.807 i ; -0.702 + -1.807 i ; -1.596 + 0.0 i ;

Racines de P_4 : -0.270 + 2.504 i ; -0.271 + -2.505 i ; -1.730 + 0.889 i ; -1.729 + -0.889 i ;

(etc. ; le lecteur retrouvera facilement ces valeurs à l'aide du programme ci-dessous)

c) Lorsque l'on dessine ces racines dans le plan complexe, on obtient les points ci après qui semblent s'organiser sur des arcs d'ellipse ou de parabole.



On constate l'existence d'un vaste domaine libre de toute racine. Il s'agit approximativement d'un quart de plan situé dans la zone de partie réelle positive, et vérifiant |Im(*z*)| < |Re(*z*)|. À regarder plus en détail, il apparaît que ces courbes sont placées les unes à côté des autres à distance régulière, comme s'il s'agissait d'une progression arithmétique. Cela laisse à penser qu'on aurait intérêt à orienter les recherches non pas sur les racines de P_n

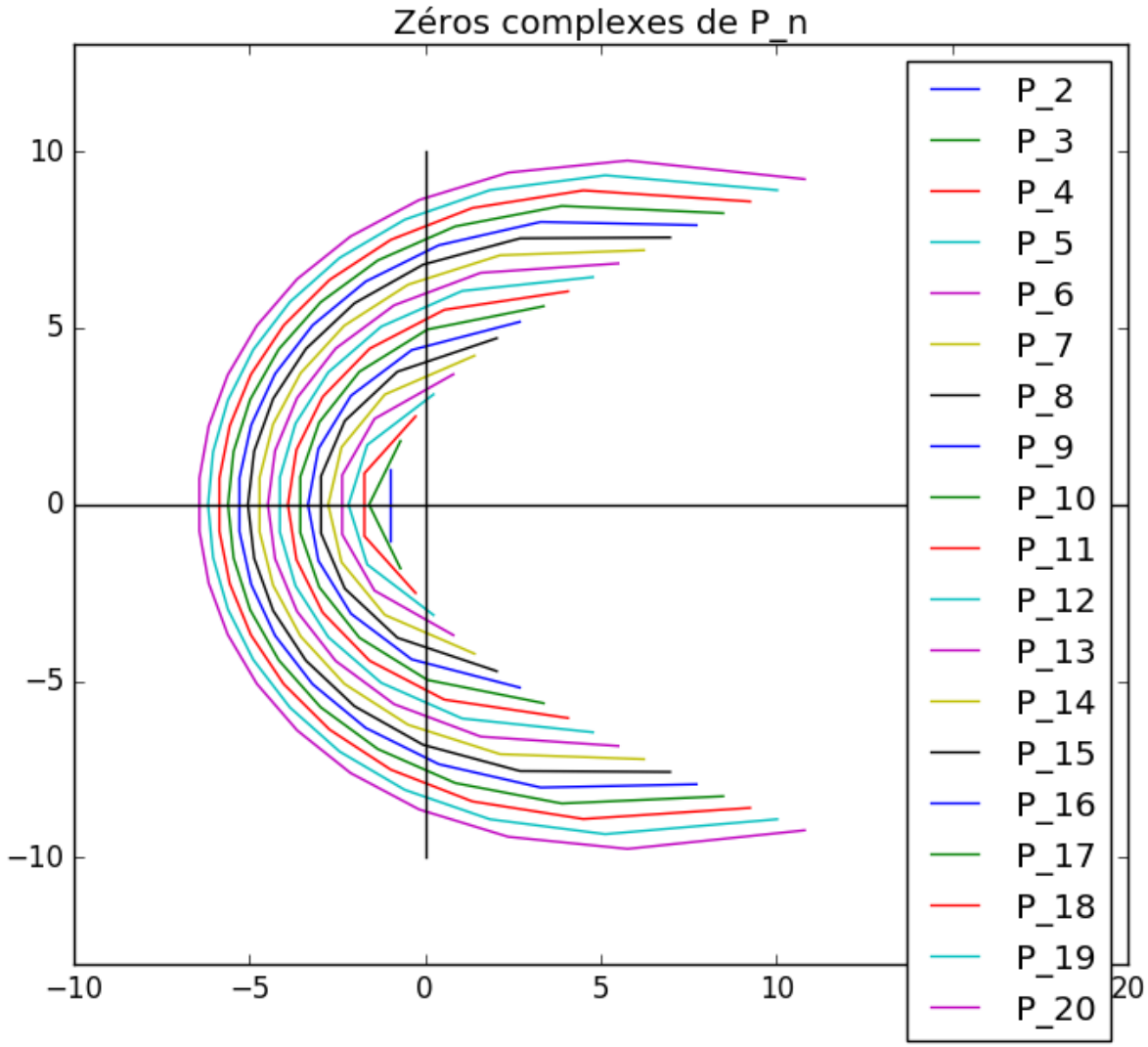
elles-mêmes, mais sur leurs quotients par n. Ainsi ce mouvement d'expansion serait mieux contrôlé et d'autres propriétés pourraient apparaître. On pose donc $S_n(z) = P_n(nz)$. Les racines de S_n sont donc les quotients des racines de P_n par n car :

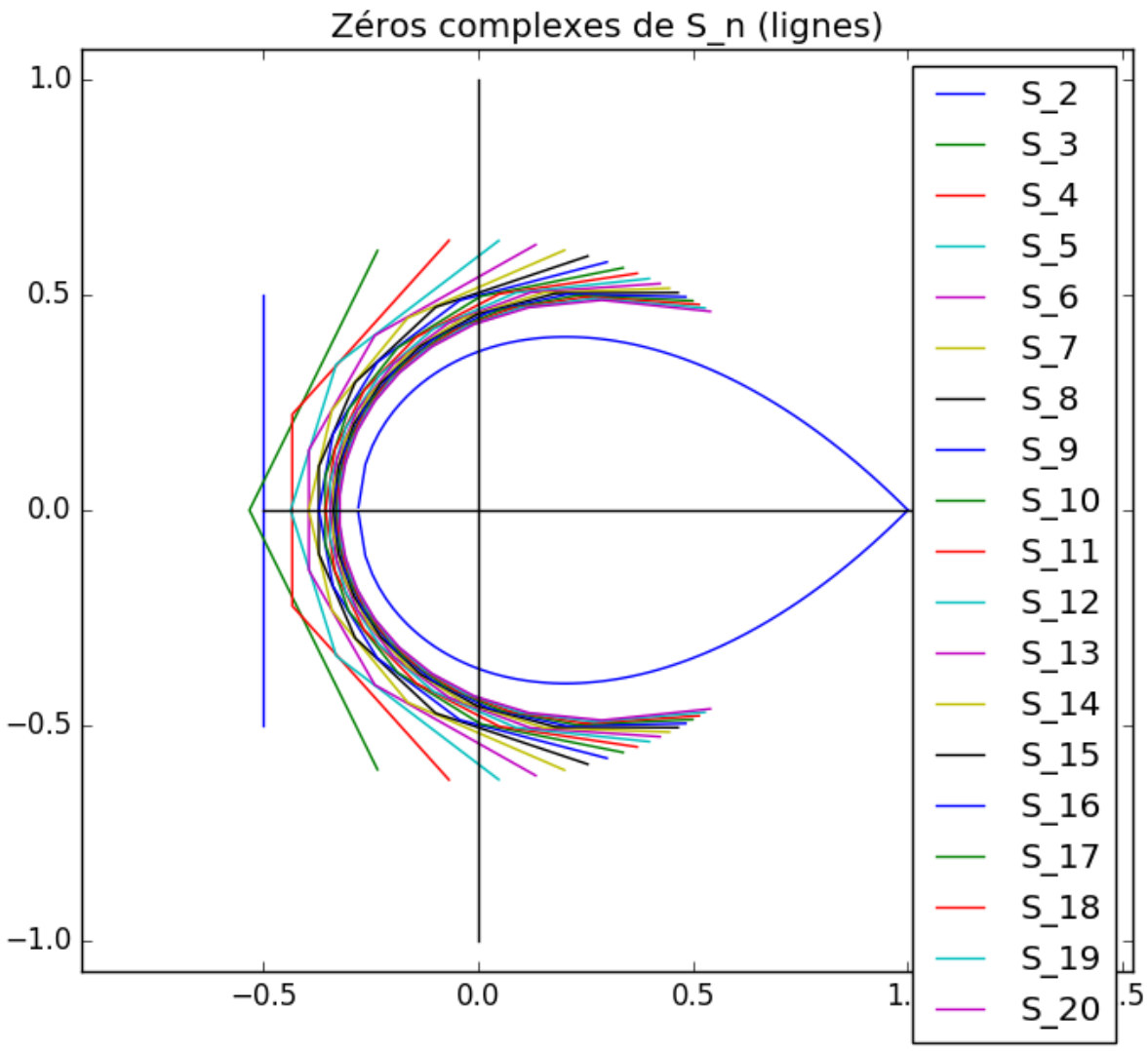
$$P_n(z) = 0 \iff P_n\left(n\frac{z}{n}\right) = 0 \iff S_n\left(\frac{z}{n}\right) = 0.$$

Il s'imposait donc une étude complémentaire, mais cette fois en poussant jusqu'à n = 20. Le programme Python le supporte aisément et les conjectures seront plus aisées.

Le premier schéma, ci-dessous, des racines de P_n nous permet de percevoir leur organisation pour une même valeur de n. Le second schéma nous montre une autre organisation, globale cette fois, toutes valeurs de n réunies. On distingue bien les allures paraboliques des lignes obtenues.

Le schéma des racines de S_n est évidemment plus condensé. L'hypothèse de la progression arithmétique régulière a été payante. On constate que lorsque n augmente, la position des racines de S_n tend à s'organiser le long d'une courbe de forme semblable à une goutte d'eau.
Cette courbe a été rajoutée pour que l'on puisse mieux percevoir ce phénomène. Pour aller plus loin, on pourra consulter les travaux de G. Szego et de J. Dieudonné qui ont étudié cette distribution asymptotique des racines complexes.





d) P_n est nécessairement scindé sur C d'après le théorème de d'Alembert. Montrons qu'il n'admet que des racines simples. Si a est une racine multiple de P_n, alors $P_n(a) = 0$ et $P_n'(a) = P_{n-1}(a) = 0$, d'où $\frac{a^n}{n!} = 0$ et $a = 0$. C'est absurde, car 0 racine de P_n.

Programme Python complet ayant permis cette expérimentation

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

def fcomp(n,nbc):#fonction complexe P_n
    X=nbc
    S=complex(1,0)
    terme=complex(1,0)
    for i in range(n+1):
        if i==0:continue
        terme=X/complex(i,0)*terme
        S=S+terme
    return S

def newton(n,nbc):#méthode éponyme pour les racines des fonctions holomorphes
    X=nbc
    varModCarre=1
    while varModCarre>1E-20:
```

```
        XX=X-fcomp(n,X)/fcomp(n-1,X)
        D=X-XX
        varModCarre=abs(D)**2
        X=XX
    return X

def f(n,x):#fonction réelle P_n
    s=1
    terme=1
    for i in range(n+1):
        if i==0:continue
        terme*=(x/i)
        s+=terme
    return s

def courbes():#représentations réelles de P_n
    x=np.linspace(-3,0.5,40)
    for i in range(2,8):
        plt.plot(x,f(i,x),label="P_"+str(i))
    plt.plot(x,x*0)
    a=np.array([0,0])
    b=np.array([-2,2])
    plt.plot(a,b)
    plt.legend()
    plt.show()

def dessineRacinesPn():#racines de P_n
    #affichage en ligne brisée
    for famille in ListeRacines:
        x=[c.real for c in famille[1:]]
        y=[c.imag for c in famille[1:]]
        plt.plot(x,y,"-",label="P_"+str(famille[0]))
    plt.title("Zéros complexes de P_n")
    plt.legend()

    x=np.array([-10,20])
    y=np.array([0,0])
    plt.plot(x,y,"k-")

    x=np.array([0,0])
    y=np.array([-10,10])
    plt.plot(x,y,"k-")

    plt.axis("equal")
    plt.show()

    #affichage en points libres
    for famille in ListeRacines:
        x=[c.real for c in famille[1:]]
        y=[c.imag for c in famille[1:]]
        plt.plot(x,y,"o",label="P_"+str(famille[0]))
    plt.title("Zéros complexes de P_n")
    plt.legend()

    x=np.array([-10,20])
    y=np.array([0,0])
    plt.plot(x,y,"k-")

    x=np.array([0,0])
    y=np.array([-10,10])
    plt.plot(x,y,"k-")

    plt.axis("equal")
    plt.show()


def dessineRacinesSn():#racines de P_n divisées par n
    #affichage en ligne brisée
    x=np.linspace(-0.2784,1,80)
    y=[math.sqrt(math.exp(2*a-2)-a*a) for a in x]
    z=[-math.sqrt(math.exp(2*a-2)-a*a) for a in x]
    plt.plot(x,y,"b",x,z,"b")

    for famille in ListeRacines:
        x=[c.real/famille[0] for c in famille[1:]]
```

```
            y=[c.imag/famille[0] for c in famille[1:]]
            plt.plot(x,y,"-",label="S_"+str(famille[0]))
        plt.title("Zéros complexes de S_n (lignes)")
        plt.legend()

        x=np.array([-0.5,1.2])
        y=np.array([0,0])
        plt.plot(x,y,"k-")

        x=np.array([0,0])
        y=np.array([-1,1])
        plt.plot(x,y,"k-")

        plt.axis("equal")
        plt.show()

        #affichage en points libres
        x=np.linspace(-0.2784,1,80)
        y=[math.sqrt(math.exp(2*a-2)-a*a) for a in x]
        z=[-math.sqrt(math.exp(2*a-2)-a*a) for a in x]
        plt.plot(x,y,"b",x,z,"b")

        for famille in ListeRacines:
            x=[c.real/famille[0] for c in famille[1:]]
            y=[c.imag/famille[0] for c in famille[1:]]
            plt.plot(x,y,"o",label="S_"+str(famille[0]))
        plt.title("Zéros complexes de S_n (points)")
        plt.legend()

        x=np.array([-0.5,1.2])
        y=np.array([0,0])
        plt.plot(x,y,"k-")

        x=np.array([0,0])
        y=np.array([-1,1])
        plt.plot(x,y,"k-")

        plt.axis("equal")
        plt.show()

    def organiseListeRacines():#tri des racines pour formation en ligne brisée
        for liste in ListeRacines:
            n=len(liste)
            for n1 in range(1,n-1):
                for n2 in range(n1+1,n):
                    if liste[n1].imag>=0:
                        if liste[n2].imag>=0:
                            if liste[n1].real<liste[n2].real:
                                permute(liste,n1,n2)
                    elif liste[n2].imag>=0:
                        permute(liste,n1,n2)
                    elif liste[n1].real>liste[n2].real:
                        permute(liste,n1,n2)


    def permute(liste,n1,n2):#échange de deux racines
        liste[n1],liste[n2]=liste[n2],liste[n1]


    def afficheRacinesTexte(n):#affichage console des racines de P_n
        for liste in ListeRacines:
            if liste[0]==n:break
        if liste[0]!=n:
            print("non calculées")
            return
        for c in liste[1:]:
            print(str(c.real)+" + "+str(c.imag)+" i"+" ; ",end='')
        print()

    def racines(n):#lance la recherche et enregistre les zéros complexes de P_n
        liste=[n]
        ListeRacines.append(liste)
        print("\nRecherche des racines de P_"+str(n)+" : ",end='')
        while len(liste)<=n:
            chercheRacine(n,liste)
```

```
def cut(n, z):
    return complex(math.floor(10**n*z.real+0.5)/(10**n), math.floor(10**n*z.imag+0.5)/(10**n))

def chercheRacine(n,liste):#détermine n zéros complexes de P_n
    x=40*random.random()-12
    if x>0:y=x/3+(10-x/3)*random.random()-0.1
    else: y=10*random.random()-0.1
    z=complex(x,y)
    Z=newton(n,z)
    Z=cut(7, Z)
    completeListe(Z,liste)
    Zc=Z.conjugate()
    completeListe(Zc,liste)


def completeListe(Z,liste):#complète la liste des racines si elle ne s'y trouve pas déjà
    absente=True
    for nb in range(1,len(liste)):
        if abs(Z.real-liste[nb].real) + abs(Z.imag-liste[nb].imag)==0:absente=False
    if absente:
        print(" "+str(len(liste)),end='')
        liste.append(Z)
```

On obtiendra les différentes figures en exécutant :

```
plt.clf()
courbes()

ListeRacines=[]
for i in range(2,21):
    racines(i)
    print()
    afficheRacinesTexte(i)

organiseListeRacines()

plt.clf()
dessineRacinesPn()

plt.clf()
dessineRacinesSn()
```

[Liste des corrigés]

[<]