UNIVERSITÉ GRENOBLE ALPES

RAPPORT DE PROJET
SYSTEMES DYNAMIQUE ET CHAOS

APPLICATION DE GAUSS ET FRACTIONS CONTINUES

Léo REYNAUD
Julien VIVIER

Professeur
Frédéric FAURE

September 8, 2023

Contents

A Introduction

Dans ce rapport, nous allons vous présenter le résultat de notre étude de l'application de Gauss et des fractions continues. Nous allons dans un premier temps essayer de déterminer les points fixes de la dynamique et leur stabilité. Ensuite, nous allons montrer que l'application de Gauss à un comportement chaotique. Nous ferons un lien entre cette application et les fractions continues. Enfin, nous étudierons le flot de l'application de Gauss sur la surface modulaire.

B application de Gauss

L'application de Gauss est l'application G(x) décrite par l'équation suivante.

$$G : \begin{cases}]0,1[\to]0,1[\\ x \to \{\frac{1}{x}\} \end{cases} \tag{1}$$

B.1 Étude des points fixes

Dans un premier temps, nous avons codé une fonction application de Gauss cette fonction va nous permettre d'effectuer un grand nombre d'itérations. Afin d'augmenter la précision de notre programme nous avons utilisé le module Decimal() de python qui nous permet de définir des variables de type "float" avec un nombre de décimales arbitrairement grand.

Avec cette fonction, nous avons pu itérer l'application de Gauss à partir d'un très grand nombre de points de départ et voilà le résultat que nous obtenons.

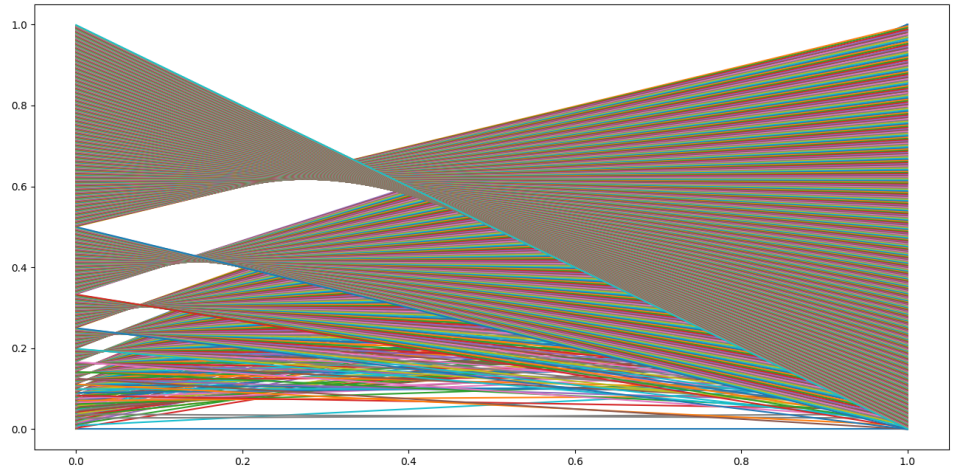

Figure 1: Balayage de l'application de Gauss

Ce graphique relie un nombre dans l'intervalle]0,1[(à gauche du graphique) à ça transformé par l'application de Gauss (à droite du graphique). Ainsi, toutes les lignes horizontales sont des points fixes et il semble en exister une infinité. Essayons de démontrer cela plus rigoureusement.

Nous cherchons des nombres x^* tel que : $G(x^*) = x^*$

Cela peut s'écrire :

$$x^* = \left\{\frac{1}{x^*}\right\} \qquad \Rightarrow \qquad x^* = \frac{1}{x^*} - n, \qquad n \in \mathbb{N}^+$$

Comme $x^* \in]0, 1[$, il n'y a qu'un seul n qui peut vérifier cette équation pour chaque x^*. Chaque point fixe est donc associé à un n différent.

On peut facilement résoudre cette équation en la transformant en un polynôme du second degré.

$$x^* = \frac{1}{x^*} - n \qquad \Rightarrow \qquad (x^*)^2 + nx^* - 1 = 0$$

On en conclu que tous les points fixes peuvent s'écrire sous la forme suivante :

$$\boxed{x^* = \frac{\sqrt{n^2+4} - n}{2}}$$

Comme $\sqrt{n^2+4} > n$, $\forall n \in \mathbb{N}^+$. Il existe un point fixe pour tous les n. Vu qu'il existe une infinité dénombrable de n, il existe une infinité dénombrable de points fixes pour l'application de Gauss.

On donne ci-dessous les trois premiers points fixes :

$$\begin{aligned}
n = 1 \qquad &\Rightarrow \qquad x^* = \frac{\sqrt{5}-1}{2} \approx 0.6180339887 \\
n = 2 \qquad &\Rightarrow \qquad x^* = \sqrt{2} - 1 \approx 0.4142135624 \\
n = 3 \qquad &\Rightarrow \qquad x^* = \frac{\sqrt{13}-3}{2} \approx 0.3027756377
\end{aligned}$$

Nous allons maintenant créer un programme qui va rechercher ces points fixes et en donner une valeur approchée. Pour cela, nous avons codé un programme qui balaye l'intervalle de 0 à 1 et qui compare la valeur x à la valeur de $G(x)$. Comme en pratique tester formellement tous les nombres réelles est impossible, nous enregistrons la valeur des interfaces entres les zones ou $x < G(x)$ et $x > G(x)$. Ces valeurs sont des valeurs approchées des points fixes. Nous pouvons ensuite balayer une intervalle beaucoup plus petite autour de ces valeurs approchées avec un pas beaucoup plus fin. Ainsi, nous pouvons avoir une précision arbitrairement grande sur les points fixes de l'application de Gauss. Ci-dessous, un exemple des résultats obtenus avec sept itérations de notre algorithme avec en rouge la partie qui diffère de la valeur analytique.($\approx$ 26 chiffres significatifs).

$$\begin{aligned}
n = 1 \qquad &\Rightarrow \qquad 0.6180339887498948482045868378165 \\
n = 2 \qquad &\Rightarrow \qquad 0.4142135623730950488016887\color{red}{3159158} \\
n = 3 \qquad &\Rightarrow \qquad 0.3027756377319946465596106\color{red}{3404992}
\end{aligned}$$

Nous pouvons faire une démonstration rigoureuse de l'instabilité des point fixes de l'application de Gauss en prenant la dérivé de cette application. Comme il s'agit d'une application à temps discret, si sa dérivé est inférieur à 1 alors les points fixes sont instables.

$$G'(x) = \left\{\frac{1}{x}\right\}' = \left(\frac{1}{x} - n\right)' = -\frac{1}{x^2} < 1 \qquad \text{avec : } n \in \mathbb{N}^+ \text{ une constante.}$$

Cette équation est vraie sur tout l'intervalle $]0, 1[$ donc tous les points fixes sont instables.

B.2 sensibilité aux conditions initiales

Nous pouvons étudier expérimentalement la sensibilité aux conditions initiales de l'application de Gauss. Pour cela, nous pouvons prendre plusieurs nombres très proches et voir si après quelques itération ils se retrouvent très éloignés.

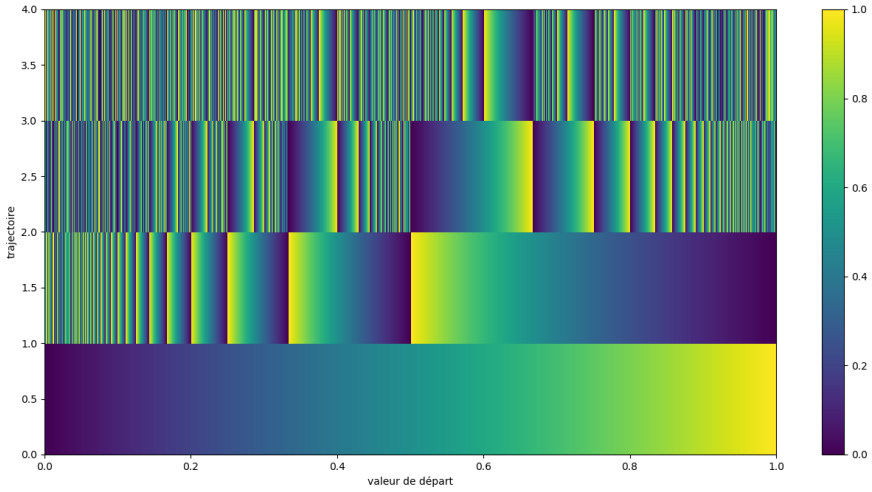

Figure 2: Évaluation de la sensibilité aux conditions initiales

Il est possible de prouver rigoureusement que cette application est sensible aux conditions initiales en calculant l'exposant de Lyapunov λ de l'application.

$$\lambda = \frac{1}{n}\sum_{i=0}^{n-1} ln\,|G'(x_i)|$$

Or, on a :

$$|G'(x_i)| = \left|\left\{\frac{1}{x}\right\}'\right| = \frac{1}{x^2} > 1 \qquad \Rightarrow \qquad \ln|G'(x_i)| > 0,\ \forall x_i \qquad \Rightarrow \qquad \lambda > 0$$

L'exposant de Lyapunov λ est supérieur à 0. Cela nous garantie que l'application est sensible aux conditions initiales.

Nous avons donc prouvé théoriquement que l'application de Gauss est sensible aux conditions initiales, le programme informatique lui permet de voir clairement que la dynamique de l'application est apériodique et mélangeante.

Une fonction mélangeante est une fonction qui applique sur un élément de volume une distorsion substantielle au cours des itérations. Ici (Figure 1) nous voyons que l'application marche d'une manière telle qu'avec une itération, l'espace de départ (]0;1[) est divisé en un nombre de sections équivalentes au nombre de points fixes. Chaque section est ensuite retournée puis étirée dans l'espace d'arrivée (]0;1[). L'application est donc hautement surjective, en effet à chaque élément de l'espace d'arrivée correspond un nombre d'éléments de l'espace de départ qui est égale au nombre de point fixe de notre système, donc une infinité (même si dans notre exemple en raison du pas discontinue et fini de notre programme le nombre de point fixe n'est pas infini). On comprend que cela créer un mélange efficace qui au bout de quelque itération rend notre système ergodique.

De plus, il est clair que l'application est apériodique. Cela en plus de se voir graphiquement a été prouvé théoriquement en montrant que notre système ne comporte aucun point fixe stable.

C Fraction continue

Une fraction continue est une expression de la forme :

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \ddots}}}}$$

Chaque nombre peut s'écrire sous forme de fraction continue, qui sera fini si le nombre est rationnel, infini si le nombre est irrationnel.

Une façon abrégé d'écrire une fraction continue est de garder seulement la suite des nombres entiers de la manière suivante

$$[a_0; a_1; a_2; a_3; a_4 \cdots]$$

Pour écrire un nombre sous forme de fraction continue, il nous faut agir de la façon suivante:

Soit $b \in \mathbb{R}$,

$$\begin{aligned} b &= [b] + \{b\}, \quad a_0 = [b] \\ c &= \frac{1}{\{b\}}, \quad c = [c] + \{c\}, \quad a_1 = [c] \\ d &= \frac{1}{\{c\}}, \quad d = [d] + \{d\}, \quad a_2 = [d]... \end{aligned}$$

Prenons par exemple $\pi = 3.14159265358979... = 3 + 0.14159265358979...$
On à donc

$$
\begin{aligned}
&3 + 0.14159265358979... \quad a_0 = 3 \\
&\frac{1}{0.14159265358979...} = 7 + 0.062513306... \quad a_1 = 7 \\
&\frac{1}{0.062513306...} = 15 + 0.99659439... \quad a_2 = 15 \\
&\frac{1}{0.99659439...} = 1 + 0.00341727... \quad a_3 = 1 \\
&\frac{1}{0.00341727...} = 292 + 0.6312524... \quad a_4 = 292
\end{aligned}
$$

Ce qui nous donne $\pi \approx [3; 7; 15; 1; 292] = 3.14159265301190$
Nous voyons que cette approximation de π nous donne une valeur exacte jusqu'à 10 décimal avec une suite de 5 entiers. Ce qui en fait une écriture compacte qui possède plusieurs propriétés intéressantes comme l'invariance par changement de base.

De plus, on peut facilement voir le lien entre l'écriture en fraction continue et l'application de Gauss, en effet pour écrire un nombre sous forme de fraction continue, il suffit de stocker la partie entière à chaque itération de l'application. Nous avons donc écrit un code python à partir de l'application de Gauss nous permettant d'écrire un nombre réel quelconque sous forme de fraction continue. Nous arrivons donc à écrire un code nous donnant les 38 premiers nombres entier. Après cela notre code manque de précision et nous donne des entiers ne correspondant pas aux vraies valeurs.

En ce qui concerne les points fixes de l'application de Gauss, nous avons vu précédemment qu'ils correspondent aux nombres restés invariant sous l'effet de G, c'est-à-dire que $\forall x \in]0, 1[, \quad G(x) = x$. Cela signifie que le n'ieme point fixe s'écrira sous forme $[a_0; a_1; a_2...]$ avec $a_0 = a_1 = a_2...$ identique et égale à n.

Nous avons surtout testé notre programme sur ces nombre, nommée irrationnel quadratique, car leur écriture en fraction continue est extrêmement simplifié et facile à analyser.

Par exemple, prenons la partie décimale du nombre d'or et itérons 40 fois l'application de Gauss dessus.

$$\{\phi\} = \frac{1 + \sqrt{5}}{2} - 1$$

Notre programme nous donne le résultat suivant:

$$\phi \approx [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3]$$

Nous voyons donc qu'au bout de la 38ème itération notre programme diverge. Cela est dû au fait que nous ayons choisi de prendre en compte seulement 17 décimales. Nous pourrions donc avoir une précision bien plus grande en prenant en compte un plus grand nombre de décimal.

D Flot géodésique sur la surface modulaire

Nous allons maintenant essayer de simuler le flot géodésique de l'application de Gauss sur la surface modulaire. Premièrement, une géodésique entre deux nombres réels sur la surface modulaire est un demi-cercle qui relie ces deux réelles. A partir de la, il est très simple de simuler le flot de l'application pour n'importe quel point de départ.

Voilà les résultats que l'on obtient pour 100 itérations.

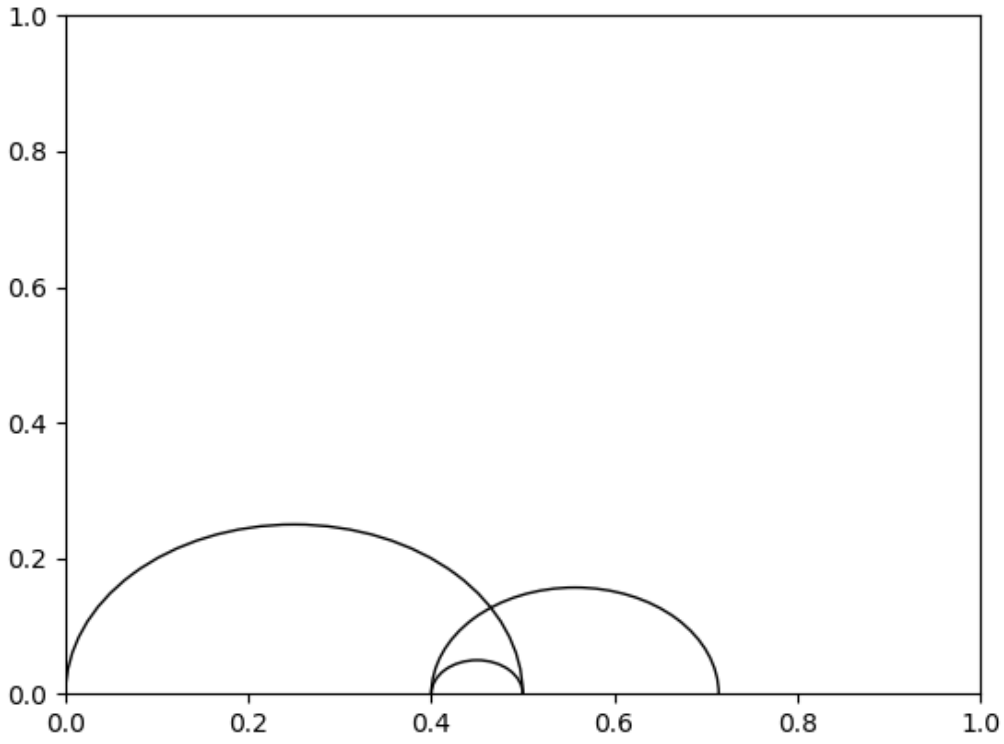

Figure 3: Flot géodésique partant de 5/7

On voit effectivement que lorsque le flot part de 5/7 qui est un nombre rationnel, le flot s'arrête très vite.

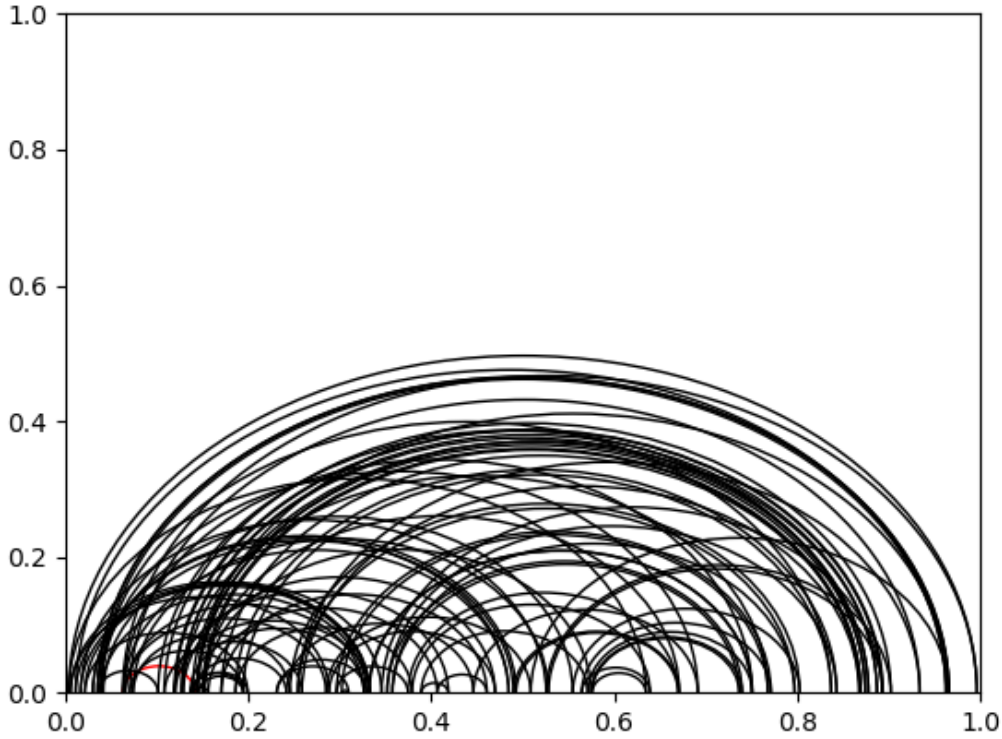

Figure 4: Flot géodésique, valeur initiale : $\pi - 3$

Pour un point de départ en $\pi - 3$. Le flot semble ergodique. Pour le vérifier, augmentons le nombre d'itérations. Ci-dessous, le résultat pour 1000 itérations.

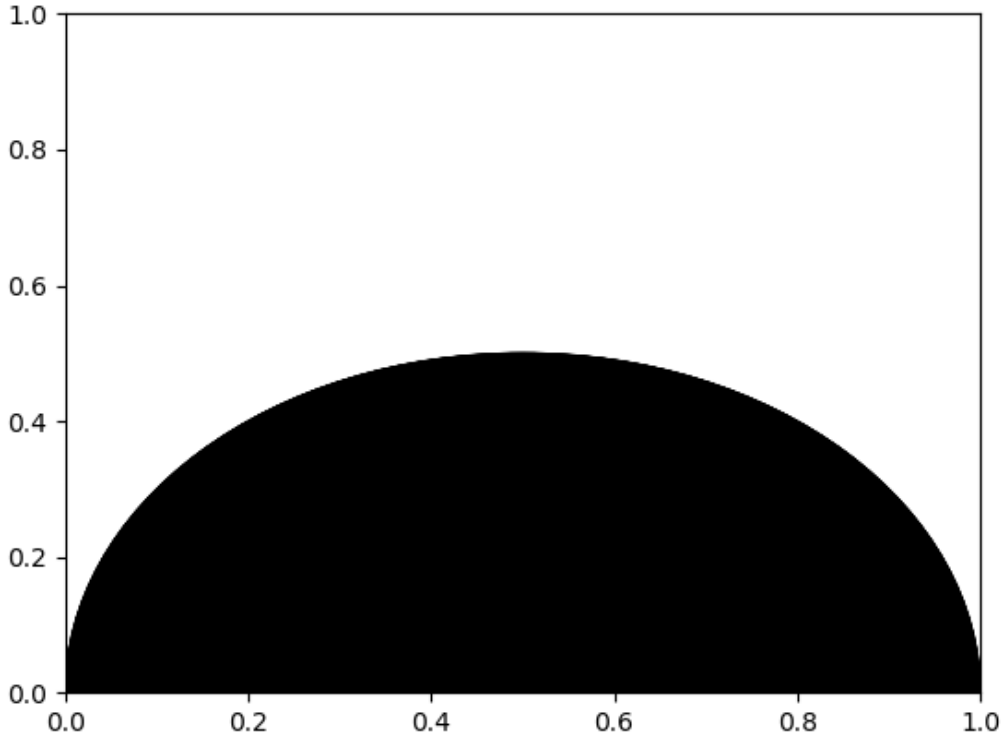

Figure 5: Flot géodésique pour $\pi - 3$

On voit que le flot semble effectivement être ergodique.

E Conclusion

Notre étude nous a permis d'observer et de démontrer les propriétés générales de l'application de Gauss. Nous avons pu montrer que l'application était chaotique et ergodique. Nous avons pu constater des propriétés avec certains nombres, notamment les nombres irrationnels quadratiques tels que ϕ ou $\sqrt{2}$. Le fait d'utiliser des simulations numériques nous a permis de visualiser le comportement de l'application et d'en avoir une compréhension plus concrète.
De plus cela nous à amené à étudier les fractions continues qui eux même nous ont ouvert au pan des mathématiques qu'est la géométrie hyperbolique. Grâce à cela nous avons pus avoir une première approche du demi-plan de Poincaré avec la représentation du flot géodésique sur cet espace.
Nous avons aussi commencé plusieurs autres cheminement qui n'aurons finalement pas abouti pour différente raison. Nous avons aussi commencé plusieurs autres cheminements qui n'aurons finalement pas abouti pour différente raison. Notamment la recherche de l'invariance de la mesure par l'application de Gauss car nous n'arrivions pas à avoir un programme assez précis, où sur la séquence de Faray que nous avons pus programmer mais dont nous n'avons pas réellement su nous servir.

F Annexe

F.1 Programme application de Gauss

```
import matplotlib.pyplot as plt
import numpy as np
from decimal import *


trajectoire=4
nbPoint=10000
pas=0.0001
debutPlage=0 #Decimal(np.sqrt(Decimal(2)))
getcontext().prec = 62        #nombre de d cimales
x=[]
x.append(debutPlage)
y=np.zeros((trajectoire,nbPoint))



def G(x):              #application de Gauss
    if x==0:
        return 0
    else:
        y=Decimal(1)/Decimal(x)
        z=Decimal(y)-int(y)
        return Decimal(z)

for i in range(nbPoint):       #On balaye la plage de nombre
    x=[]
    x.append(debutPlage+Decimal(pas*i))
    for j in range(trajectoire):              #on parcourt la trajectoire
        x.append(Decimal(G(x[j])))
        y[j][i]=x[j]

    #plt.plot(range(trajectoire+1),x)   #affichage une coube par point de d part(figure



im=plt.imshow(y, cmap='viridis',          #affichage figure 2
              extent=[0, 1, 0, trajectoire],
              interpolation='nearest', origin='lower', aspect='auto')
plt.colorbar(im, orientation='vertical')
plt.ylabel('trajectoire')
plt.xlabel('valeur de d part')
```

F.2 Programme recherche points fixes

```
import matplotlib.pyplot as plt
import numpy as np
from decimal import *


trajectoire=2
nbPoint=10000
pas=0.0001
debutPlage=0 #Decimal(np.sqrt(Decimal(2)))
getcontext().prec = 32        #nombre de decimales
x=[]
x.append(debutPlage)
y=np.zeros((trajectoire,nbPoint))
frac=[]


def affinage(invariant,pas,nbPoint,ordre):        #fonction qui recherche les points fixes
    pas2=(Decimal(pas*2))**Decimal(ordre)
    print(pas2)
    result=[]
    for i in invariant:
        a=True
        valeur=Decimal(Decimal(i)-pas2*nbPoint/2)
        for j in range(nbPoint):
            valeur=valeur+pas2
            if a==True:
                if valeur<G(valeur):
                    a=True
                else:
                    a=False
                    if valeur>Decimal(0.1):
                        result.append(valeur)
            else:
                if valeur>G(valeur):
                    a=False
                else:
                    a=True


    return result

def G(x):             #application de Gauss
    if x==0:
        return 0
    else:
        y=Decimal(1)/Decimal(x)
        z=Decimal(y)-int(y)
```

```
        frac.append(int(y))
        return Decimal(z)

print('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')
invariant=affinage([0.5],pas,nbPoint,1)
print(invariant)

for i in range(7):
    print('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')
    invariant=affinage(invariant,pas,nbPoint,i+2)
    print(invariant)
```

F.3 Programme fraction continu

```
import matplotlib.pyplot as plt
import numpy as np
from decimal import *


trajectoire=62
nbPoint=1
pas=0.001
debutPlage=(Decimal(1)+(np.sqrt(Decimal(5))))/Decimal(2)-Decimal(1)
getcontext().prec = 25        #nombre de d cimales
x=[]
x.append(debutPlage)
y=np.zeros((trajectoire,nbPoint))
frac=[]


def G(x):              #application de Gauss
    if x==0:
        return 0
    else:
        y=Decimal(1)/Decimal(x)
        z=Decimal(y)-int(y)
        frac.append(int(y))
        return Decimal(z)

for i in range(nbPoint):      #On balaye la plage de nombre
    x=[]
    x.append(Decimal(debutPlage)+Decimal(pas*i))
    for j in range(trajectoire):              #on parcourt la trajectoire
        x.append(Decimal(G(x[j])))
        y[j][i]=x[j]

    plt.plot(range(trajectoire+1),x)          #affichage (une coube par point de d part)
    print(frac)

#plt.pcolor(y)                    #affichage (Surface 2D)
```

F.4 Programme flot géodésique

```
import matplotlib.pyplot as plt
import numpy as np
from decimal import *

getcontext().prec = 50      #nombre de decimales

origine=Decimal(np.pi)-Decimal(3) #(np.sqrt(Decimal(5))-Decimal(1))/Decimal(2)
#np.sqrt(Decimal(2))
nbPoint=100



def G2(x):              #application de Gauss
    if x==0:
        return 0
    else:
        y=Decimal(1)/Decimal(x)
        z=Decimal(y)-int(y)
        return Decimal(z)


def geodesic(Ddepart,Darrive,i):
    depart=float(Ddepart)
    arrive=float(Darrive)
    centre=(depart+arrive)/2
    rayon=abs(depart-arrive)/2
    if i>0:
        color='k'
    else:
        color='r'
    cercle=plt.Circle((centre,0),rayon,fill=False, edgecolor=color)
    ax.add_artist(cercle)
    plt.pause(0.2)


fig, ax = plt.subplots()

depart=origine
print(depart)

for i in range(nbPoint):      #On parcours le flot
    arrive=G2(depart)
    print(arrive)
    geodesic(depart,arrive,i)
    depart=arrive
```