Boyer Benjamin
Couton Charlotte
Rozanes Camille

Groupe 1AG10
ENTPE 2007/2008.

Projet de Calcul Scientifique

Fractales de Lyapunov :

Diagrammes de bifurcation, Suite logistique et Chaos.

Dans le cadre de notre projet de calcul scientifique, nous allons chercher à construire des objets fractals. Dans cette démarche, nous allons appliquer la méthode de Lyapunov. Dans un premier temps, nous étudierons et calculerons les exposants de Lyapunov sur le support de la suite logistique. Ensuite, nous mettrons en relation ces calculs avec la création de matrices d'exposants pour aboutir à la création des fractales de Lyapunov.

1. Suite logistique et exposant de Lyapunov

Nous nous intéressons à l'étude de la suite logistique : $P_{i+1} = kP_i(1 - P_i)$

On utilise une quantité $\lambda(P_0)$ pour mesurer le degré de stabilité d'un système, suite à la variation infinitésimale des conditions initiales. On peut calculer cette dernière avec la formule suivante : $\lambda(P_0) = \lim_{n \to +\infty} \frac{1}{n} \sum_{i=1}^{n} \ln \left| \frac{df_k}{dx}(P_{i-1}) \right|$

Question 1 : Programmation et optimisation d'une fonction vectorielle de l'exposant de Lyapunov.

Proposition 1 :

```
17/01/03 00:00      C:\Mes Fichiers\work\projet CS\lyapounov.m                1 of 1

function [lambda]=lyapunov(P0,K,N)
% P0 est le vecteur des P0
% K est le vecteur séquence des ki
% N est l'ordre considéré pour approximer la limite

l=length(K);

% on cherche dans un premier temps les Pi

P=P0;
for i=1:N
    P=[P;K(ones(1,l)).*P(i,:).*(1-P(i,:))];
end

% on peut ensuite calculer les fk'(Pi-1)
F=[];
for i=1:N
F=[F;K(ones(1,l)).*(1-2.*P(i+1,:))];

end
% on exclue le terme de la dérivée calculée en P0, en effet si P0=1/2 on a
% directement -inf, ce qui n'est pas cohérent avec la figure donnée.


%sum(F,1) est la somme par colonne

lambda=(1/N).*(sum.(log(abs(F))),1));
```

Afin de vérifier la cohérence des résultats de la fonction, on trace la courbe des exposants de Lyapunov en fonction de k (Voir le script Untitled3 en annexe). Ainsi on exclut le terme en P_0, on a une aberration avec $P_0 = ½$.

La vectorisation est prise en compte dans cette fonction, elle renvoie un vecteur des λ si on lui fournit un vecteur des P_0.

On peut voir certaines limites à cette fonction notamment du fait qu'il y ait deux boucles for. On peut ensuite envisager une autre fonction *Lyapunov2,* qui tire partie des propriétés du Ln.

Proposition 2 :

```
function [lambda]=lyapunov2(P0,K,N)
% P0 est le vecteur des Po
% K est le vecteur séquence des ki
% N est l'ordre considéré pour approximer la limite

% on peut penser que la fonction lyapunov précédente est plus lente à cause
% de la double boucle for, on en construit alors une unique. On utilise la
% fait que SIGMA(ln xi)=ln(PI(xi))

l=length(K);

P=P0;
exposant=ones(1,length(P0));

for i=1:N
    % on calcule au fur et à mesure les Pi

    ind=K(rem(i-1,l)+1);

    P=ind.*P.*(1-P);


    exposant=exposant.*(1-2.*P).*ind;

    if exposant==0
        break
    end

end

% on le produit des fk'(Pi)

lambda=(1/N).*(log(abs(exposant)));
```

On peut comparer le temps d'exécution des deux fonctions grâce à « tic toc ».

On compare leur utilisation dans le script *Untitled3* :
30,43s pour *Lyapunov*, 9,42s pour *Lyapunov2*.

On peut mettre une condition de rupture de boucle, en effet si exposant=0, il sera toujours égal à 0 pour les termes suivants. En revanche si Matlab, le multiplie par un terme trop grand, il conduira à une opération infini x 0 → Nan.

Dans la démarche d'optimisation, nous avons essayer de sommer à chaque exécution le Ln.

Nous avons ainsi la Proposition 3 :

17/04/08 08:12 C:\MATLAB7\work\projet CS\lyapunov3.m

```
function [lambda]=lyapunov3(P0,K,N)
% P0 est le vecteur des Po
% K est le vecteur séquence des ki
% N est l'ordre considéré pour approximer la limite

% on peut penser que la fonction lyapunov précédente est plus lente à cause
% de la double boucle for, on en construit alors une unique. On utilise la
% fait que SIGMA(ln xi)=ln(PI(xi))

l=length(K);

P=P0;
exposant=ones(1,length(P0));

for i=1:N
    % on calcule au fur et à mesure les Pi

    ind=K(rem(i-1,l)+1);

    P=ind.*P.*(1-P);


    exposant=exposant+log(abs((1-2.*P).*ind));

    if abs(exposant)==Inf
        break
    end

end

% on le produit des fk'(Pi)

lambda=(1/N).*exposant;
```

On obtient ici 10,6 s dans l'utilisation de *Untitled3* et on évite des erreurs de calculs par rapport à *Lyapunov2*.

En utilisant un script pour faire varier k de 0 à 4, on peut tracer l'exposant de Lyapunov en fonction de k. On retrouve la figure proposée dans l'énoncé.

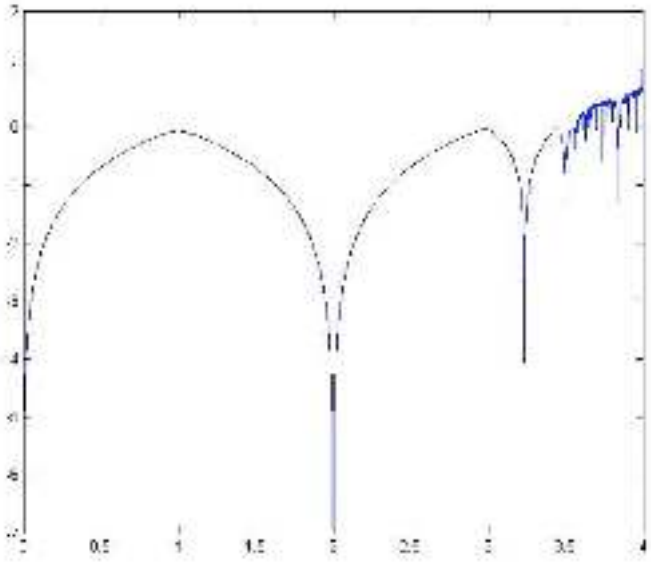

Fig1. Exposants de Lyapunov

Remarque : Dans chacune des fonctions Lyapunov proposée, nous avons pris en compte les exigences de la séquence des ki (racine) abordée en seconde partie.

2. Matrices d'exposants et construction de fractales.

On va maintenant utiliser la fonction *Lyapunov3* afin de calculer les λ pour des racines différentes. On choisit k1 et k2 entre deux valeurs et on construit une racine avec une séquence cyclique de ki. On fournit ainsi un vecteur K à la fonction *Lyapunov3*.

On construit ainsi une grille des λ pour les k1, k2 et on obtient une matrice T. Nous avons mis en place plusieurs fonctions tableaux avec différents paramètres pour k1,k2 et la séquence.

Nous pouvons donner quelques exemples :

NB. M est la taille de la matrice T construite, N est le nombre pris pour approcher la limite dans le calcul de λ.

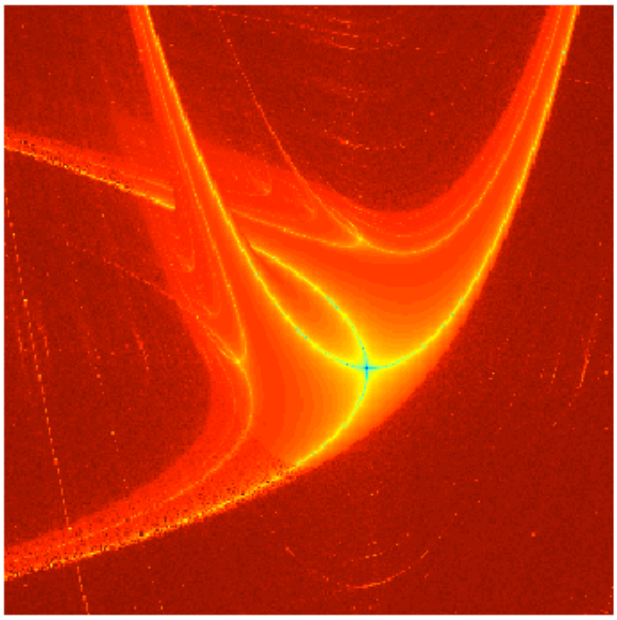

Fig2. Racine '21' avec ki variant entre 3,867 et 3,808 N=400 , M=300.

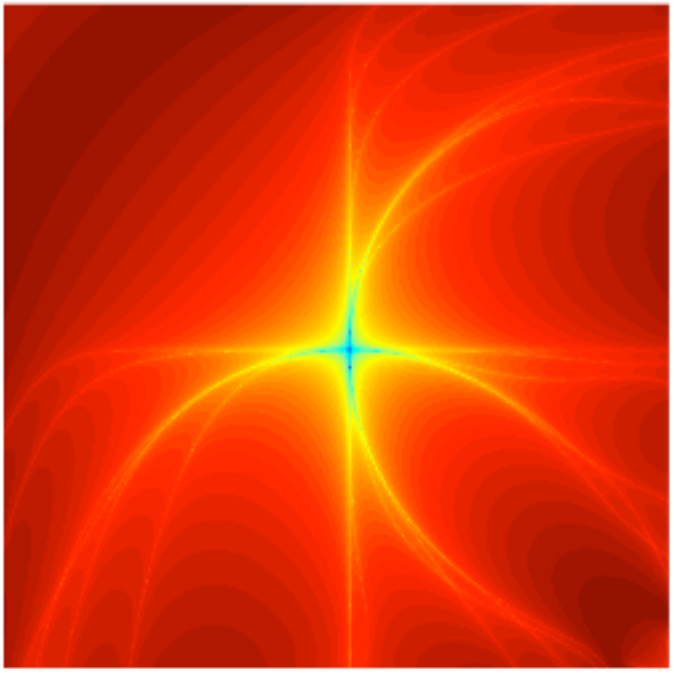

Fig3. Racine '222 121 211 121' avec k entre 0.7 et 3.4 M=1700 et N=500

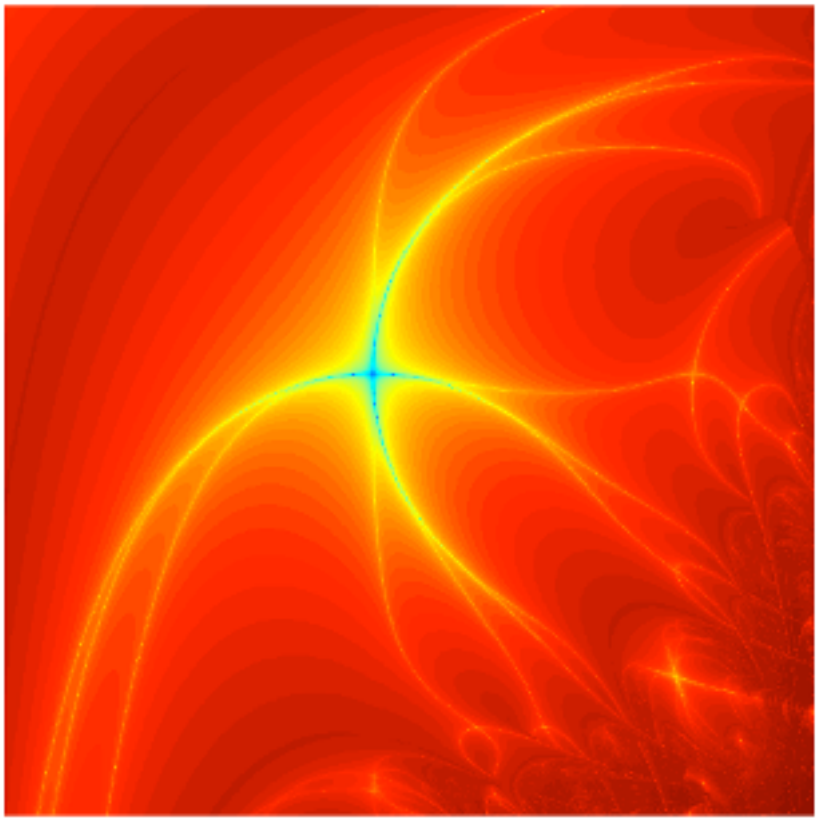

Fig4. Racine '2212121' avec k entre 0.5 et 3.8 M=800 N=400

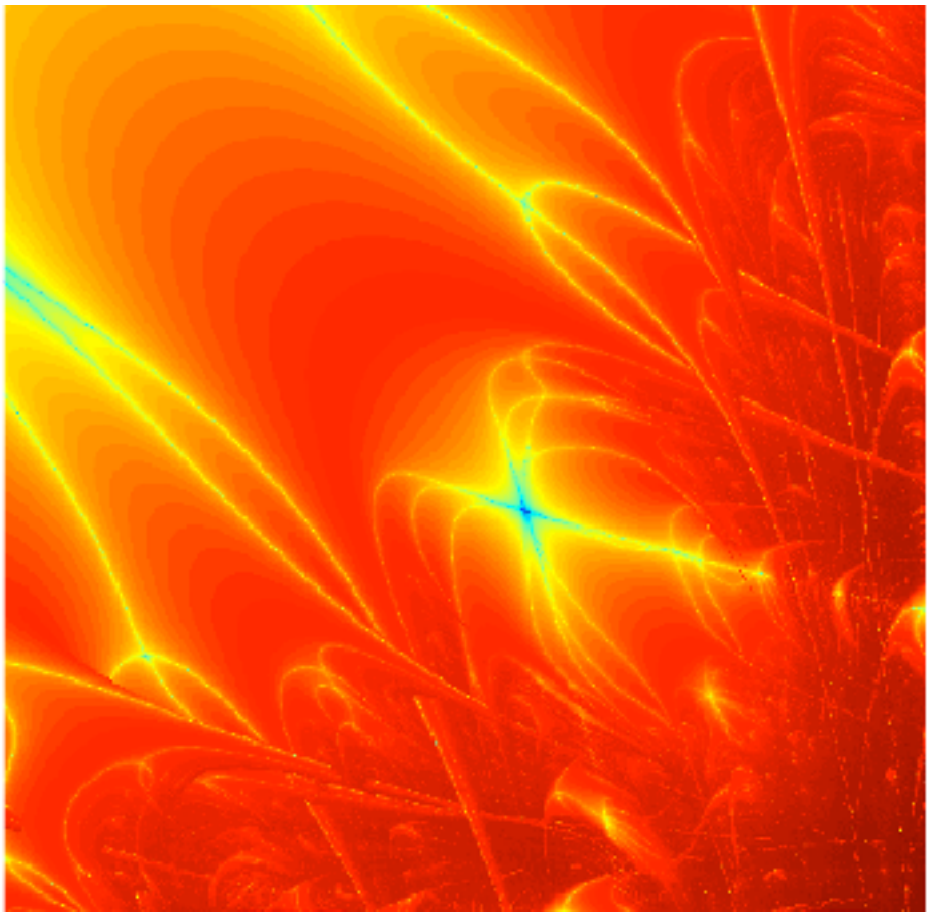

Fig5. Racine' 2212121' avec k entre 2.5 et 3.8 M=800 N=400.

NB. On peut voir l'influence du choix de l'intervalle des ki.

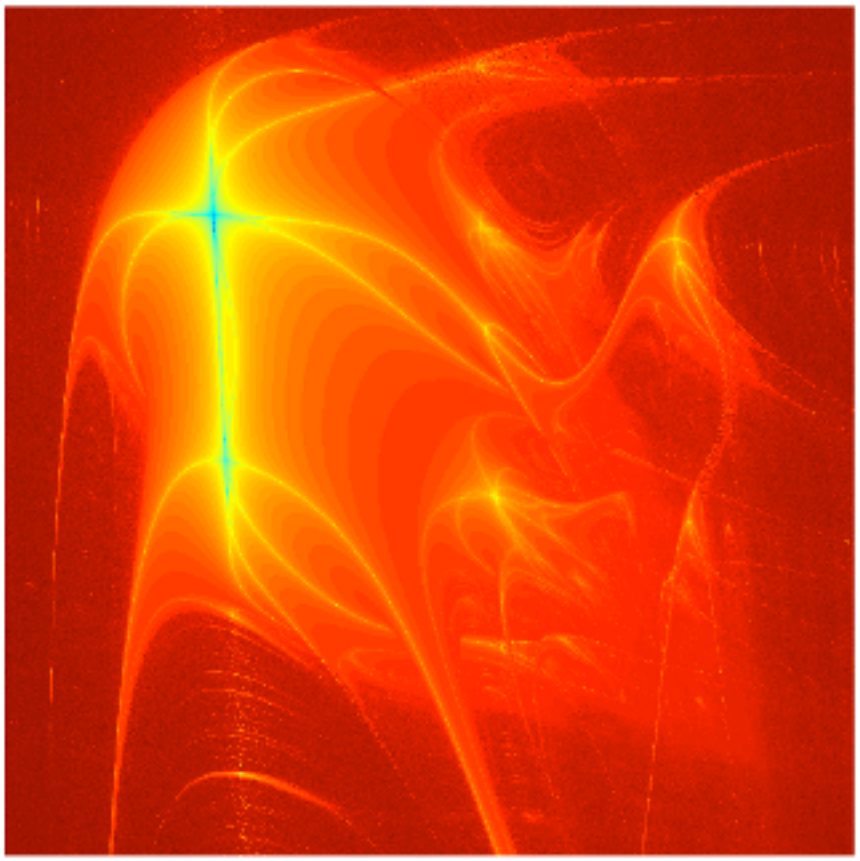

Fig6. Racine ’221 21’ M=1000 N=600 k entre 3,882 et 3,3607

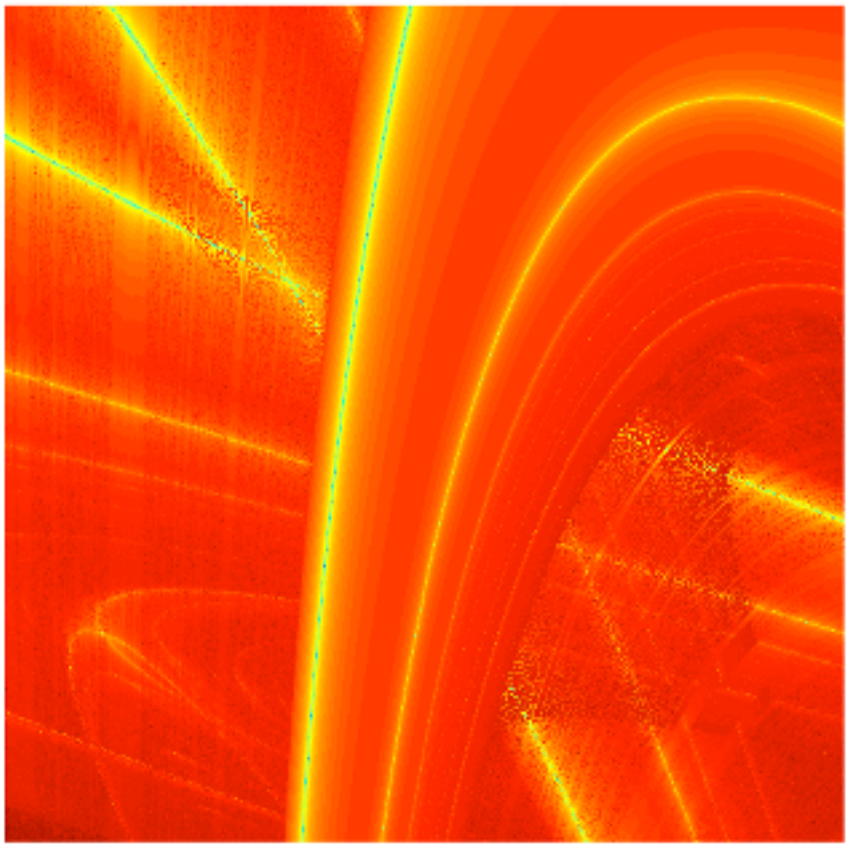

Fig7. Racine’221 21 21’ avec k1 entre 3.8308 et 3.8471 ; k2 entre 3.5842 et 3.6050 M=1000 N=500

```
% ce script doit nous permettre de tracer l'exposant de lyapunov en
% fonction de k, on choisit P0=0.5

% il suffit d'appeler la fonction vectorisée lyapunov
tic
P0=0.5;
L=[];
N=100;
k=[];
for K=0:(0.0001):4
    k=[k,K];
    L=[L,lyapunov3(P0,K,N)];
end
axis auto
plot(k,L)
toc
```

```
function [T,temps]=tableau(N,P0,M)
% cette fonction renvoit les lambda par couple (k1,k2) de la séquence K

% on fixe K=2211

tic

T=zeros(M,M);
U=linspace(1,3,M);

for k1=1:M
    for k2=1:M

        K=[U(k2),U(k2),U(k1),U(k1)];
        T(k1,k2)=lyapunov3(P0,K,N);

    end
end

save T;

temps=toc;
```

```
function [T,temps]=tableau2(N,P0,M)
% cette fonction renvoit les lambda par couple (k1,k2) de la séquence K

% on fixe K=222 222 222 211 122 121

tic

T=zeros(M,M);
U=linspace(0.1,4.2,M);


for k1=1:M
    for k2=1:M

        K=[U(k2),U(k2),U(k2),U(k2),U(k2),U(k2),U(k2),U(k2),U(k2),U(k1),U(k1),U ↙
(k1),U(k1),U(k2),U(k2),U(k1),U(k2),U(k1)];
        T(k1,k2)=lyapunov3(P0,K,N);

    end
end

save T;

temps=toc;
```

```
function [T,temps]=tableau3(N,P0,M)
% cette fonction renvoit les lambda par couple (k1,k2) de la séquence K

% on fixe K=2212121

tic

T=zeros(M,M);
U=linspace(0.2,4.2,M);

for k1=1:M
    for k2=1:M

        K=[U(k2),U(k2),U(k1),U(k2),U(k1),U(k2),U(k1)];
        T(k1,k2)=lyapunov3(P0,K,N);

    end
end

save T;

temps=toc;
```

```
function Construction(T)

imagesc(T);
colormap(jet);

hold off;
axis equal;
axis off;
```