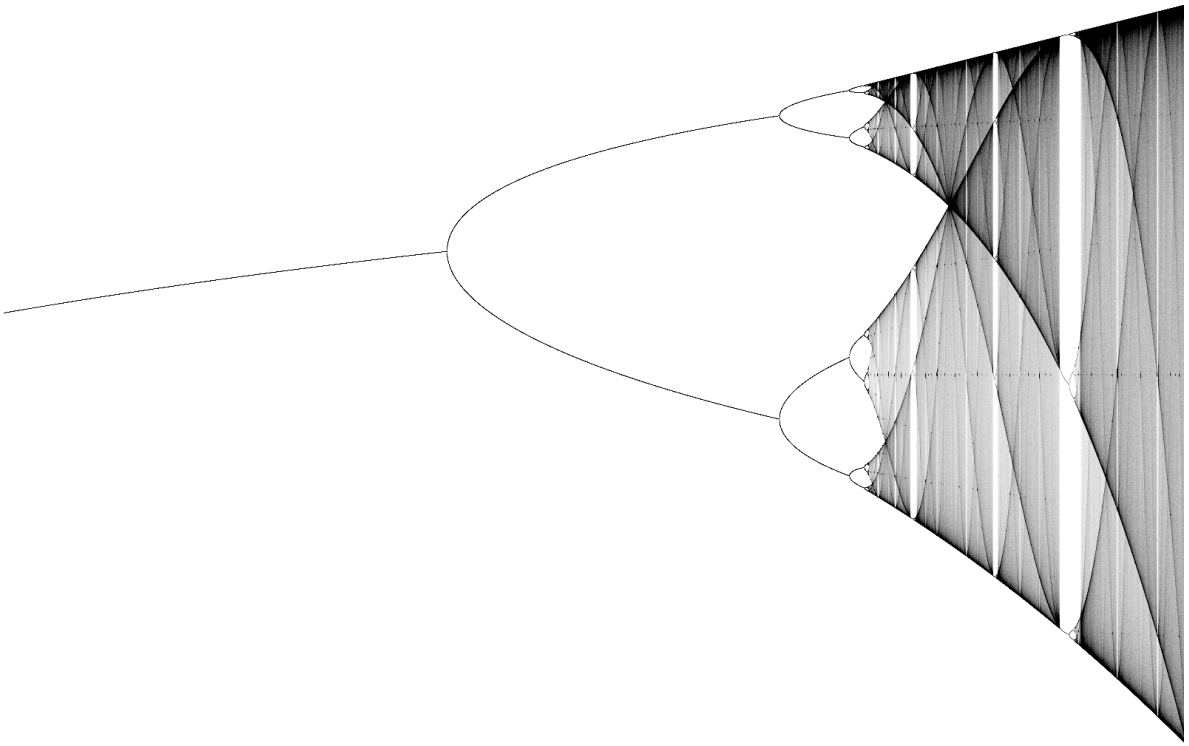
TP1 Python : Suites
L'escalier, l'escargot et le chaos

1 Présentation

Ce TP est consacré à l'étude du comportement des *suites logistiques*, définies par

$$\begin{cases} u_0 \in]0,1[\\ \forall\, n \in \mathbb{N},\ u_{n+1} = a\, u_n (1-u_n) \end{cases}$$

avec $a \in [0,4]$ (si a n'est pas dans $[0,4]$, l'intervalle $[0,1]$ n'est pas stable par l'application $f_a : x \mapsto ax(1-x)$ et le comportement de la suite est différent – et plus ennuyeux).
Nous allons voir que le comportement asymptotique de la suite ne dépend pas de u_0 (à l'exception de quelques valeurs qui rendent triviale la suite) mais dépend par contre beaucoup de a. Le but final du TP est le tracé d'un *diagramme de bifurcation*, que voici en avant-première (mais pour l'instant vous ne savez pas à quoi ça correspond) :



2 Outils élémentaires de calcul

Pour calculer efficacement sur ces suites, on définit les fonctionnalités élémentaires :

Exercice 1. Réécrire les imports nécessaires, sous leurs alias usuels :

- package `numpy` de calcul numérique ;
- package `matplotlib.pyplot` d'affichage graphique.

Exercice 2. Définir (à l'aide de `def`) la fonction $f : (a,x) \mapsto ax(1-x)$.

Exercice 3. Coder une fonction Python `liste_termes(a,u0,N)` qui prend pour arguments un réel u_0, un entier N et un réel a, et renvoie la liste $[u_0, u_1, \dots, u_N]$ des N+1 premiers termes de la suite logistique.
On fera intervenir la fonction f dans ce code.

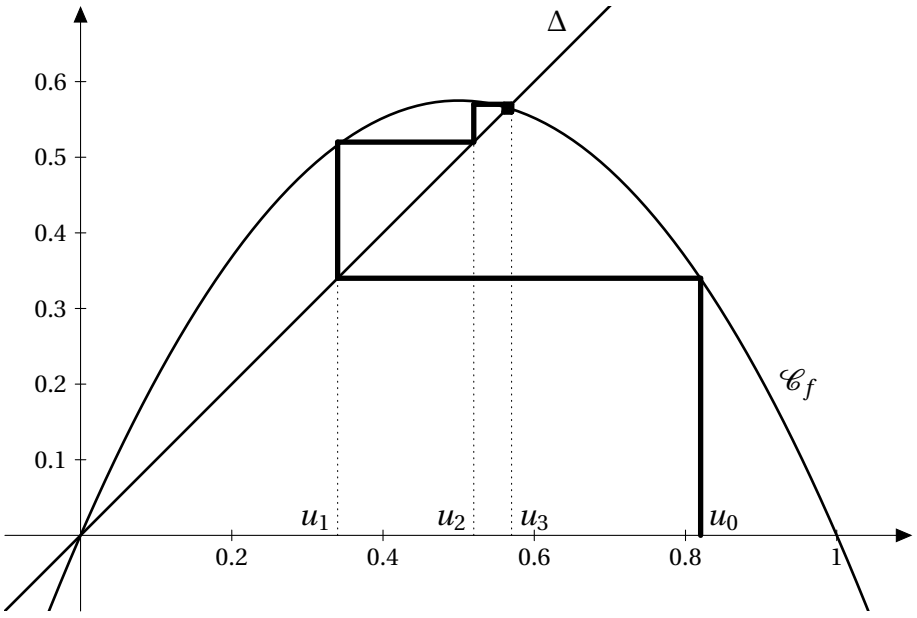
Exercice 4. On rappelle que la commande `np.arange(N+1)` renvoie le `np.array [0,1,2,...,N]`.
À l'aide de la commande `plt.scatter` et de la fonction de l'exercice précédent, représenter graphiquement la suite logistique pour diverses valeur de $a \in [0,4]$. Quels comportements observe-t-on ?

3 L'escalier et l'escargot

Dans cette section on introduit (ou on rappelle) une méthode de représentation graphique d'une suite récurrente qui permet une visualisation intéressante.
Le principe est le suivant : on trace sur un même schéma la courbe $\mathscr{C}_f$ de la fonction f telle que $u_{n+1} = f(u_n)$, et la « bissectrice » Δ d'équation $y = x$.
On se place sur l'axe des abscisses, à l'abscisse u_0 ; puis on se déplace verticalement jusqu'à arriver sur $\mathscr{C}_f$. Le point de $\mathscr{C}_f$ d'abscisse u_0 a pour coordonnées $(u_0, f(u_0))$, c'est-à-dire (u_0, u_1). En se déplaçant horizontalement jusqu'à la droite $y = x$ on sera donc au point de coordonnées (u_1, u_1) ; puis en se déplaçant verticalement jusqu'à la droite on sera en $(u_1, f(u_1)) = (u_1, u_2)$; puis en (u_2, u_2) ; etc...
On trace donc une ligne brisée reliant les points :

$$(u_0,0) - (u_0,u_1) - (u_1,u_1) - (u_1,u_2) - (u_2,u_2) - \cdots - (u_n,u_n) - (u_n,u_{n+1}) - (u_{n+1},u_{n+1}) - \dots$$

On obtient quelque chose de ce type, pour $a = 2,3$ et $u = 0,82$:



Cette représentation permet d'illustrer les propriétés de (non-)convergence de la suite (u_n) et de la comprendre intuitivement dans pas mal de cas. Dans la figure ci-dessus, (u_n) tend vers une valeur proche de 0,58 : c'est l'intersection de la courbe et de la droite... donc un point fixe !

Pour définir cette fonction on commence par tracer la droite Δ et la courbe de f_a sur $[0,1]$ par les commandes usuelles :

```
# définition de la liste des abscisses
X = np.linspace(...,...,...)
plt.plot(...,...) # tracé de la bissectrice
# courbe de f
Y = f(a,X) # opérations composante par composante sur les np.array
plt.plot(...,...)
```

On rappelle que la commande `plt.plot(X,Y)` prend en argument deux `np.array` X = $[x_1,\dots,x_n]$ et Y = $[y_1,\dots,y_n]$ et trace la ligne brisée reliant, dans cet ordre, les points $(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$.

En particulier on trace le segment entre le point (a,b) et le point (c,d) par

```
plt.plot([a,c],[b,d])
```

Le code suivant trace donc le premier segment de la ligne décrite plus haut :

```
# on suppose qu'une valeur a été affectée à a
u = u0
v = ... # v vaut maintenant u1
plt.plot(...,...)
```

Exercice 5. À partir de cette « initialisation », la ligne brisée se construit par une itération : à chaque étape on rajoute les segments (u_n,u_{n+1}) — (u_{n+1},u_{n+1}) et (u_{n+1},u_{n+1}) — (u_{n+1},u_{n+2}).
Le code suivant reprend le précédent et trace la ligne brisée jusqu'à un rang n :

```
### premier segment ###
# on suppose qu'une valeur a été affectée à a
u = u0
v = ... # v vaut maintenant u1
plt.plot(...,...)

### segments suivants ###
for k in range(n): # avant la nème itération u vaut u_(n-1) et v vaut u_n
    plt.plot([...,...,...],[...,...,...])
    # et on "passe au rang suivant"
    u = v
    v = f(a,v)
plt.grid() # affichage d'une grille pour la lisibilité
plt.show() # affichage
```

Finalement, en regroupant les résultats précédents, on se retrouve avec la fonction suivante, qui prend en arguments a, u_0 et N et trace le diagramme en escalier, ou en escargot, pour les N premiers termes de la suite logistique de paramètres a et u_0 :

```
def trace(a,u0,N):
    plt.figure(dpi=600)  # pour une figure à résolution élevée

    ### tracé de la bissectrice
    X = np.linspace(...,...,...)
    plt.plot(...,...,color='black',linewidth=0.7)
    ### tracé de la courbe
    Y = f(a,X)
    plt.plot(...,...,color='black',linewidth=0.7)


    ### tracé de la ligne brisée
    ## premier segment
    u = u0
    v = ... # v vaut u_1
    plt.plot(...,...,color='blue',linewidth=1.5)

    ## segments suivants
    for k in range(N): # avant la nème itération u vaut u_(n-1) et v vaut u_n
        plt.plot([...,...,...],[...,...,...],color='blue',linewidth=1.5)
        # et on "passe au rang suivant"
        u=v
        v=f(a,v)

    ### ajustement de la fenêtre graphique
    # la fonction f atteignant son max en 1/2,
    # on coupe la figure un peu plus haut
    plt.axis([0,1,0,f(a,0.5)+0.1])

    plt.grid() # affichage d'une grille pour la lisibilité
    plt.show() # affichage
```

(NB : au passage, on a mis quelques ajustements graphiques pour avoir des jolis dessins).

Tester cette fonction pour différentes valeurs de $a \in [0,4]$ (et de u_0 si vous voulez, mais vous verrez que le comportement de la suite ne dépend pas de u_0).
En particulier regarder : $a \in [0,1]$, $a \in [1,3]$, $a \geqslant 3$ proche de 3, $a \leqslant 4$ proche de 4. Décrire les comportements de la suite associés.

4 Quelques comportements particuliers de la suite logistique

Reprendre les fonctions de la section précédente et regarder le comportement de (u_n) pour les valeurs de a suivantes :

- $a = 0.6$
- $a = 1.7$
- $a = 3.2$
- $a = 3.5$
- $a = 3.7$
- $a = 3.83$
- $a = 4$

On voit que lorsque'il n'y a pas convergence, on peut observer divers comportements : oscillation entre 2 valeurs, 3 valeurs, 4 valeurs, ou quelque chose de plus compliqué... le diagramme de bifurcation va permettre de visualiser tous ces comportements sur une seule figure.

5 Diagramme de bifurcation

Pour tracer ce diagramme on fixe une valeur de $u_0 \in]0,1[$ une bonne fois pour toutes. Pour chaque valeur de a, on considère les points (a,u_n) où n est assez grand.
Si (u_n) converge une limite ℓ, tous les points de ce type seront pratiquement tous confondus en (a,ℓ) : on ne verra qu'un point !
Si (u_n) oscille entre 2 « valeurs limites » ℓ_1 et ℓ_2 on verra en fait 2 points : (a,ℓ_1) et (a,ℓ_2). Et si (u_n) ne se « stabilise » pas, on verra beaucoup de points.

Exercice 6. Pour tracer ce diagramme, nous avons besoin d'une fonction qui renvoie une liste de termes de la suite (u_n) de la forme

$$[u_m, u_{m+1}, \dots, u_{m+\mathrm{N}}]$$

où m sera pris grand (pour ne voir que le comportement limite de la suite). Coder cette fonction, avec l'en-tête `def liste_termes_asympt(a,u0,m,N)` qui prendra comme arguments a, u_0, m, N.
On remarquera qu'il suffit d'appeler la fonction liste_termes avec une condition initiale qui n'est plus u_0 !

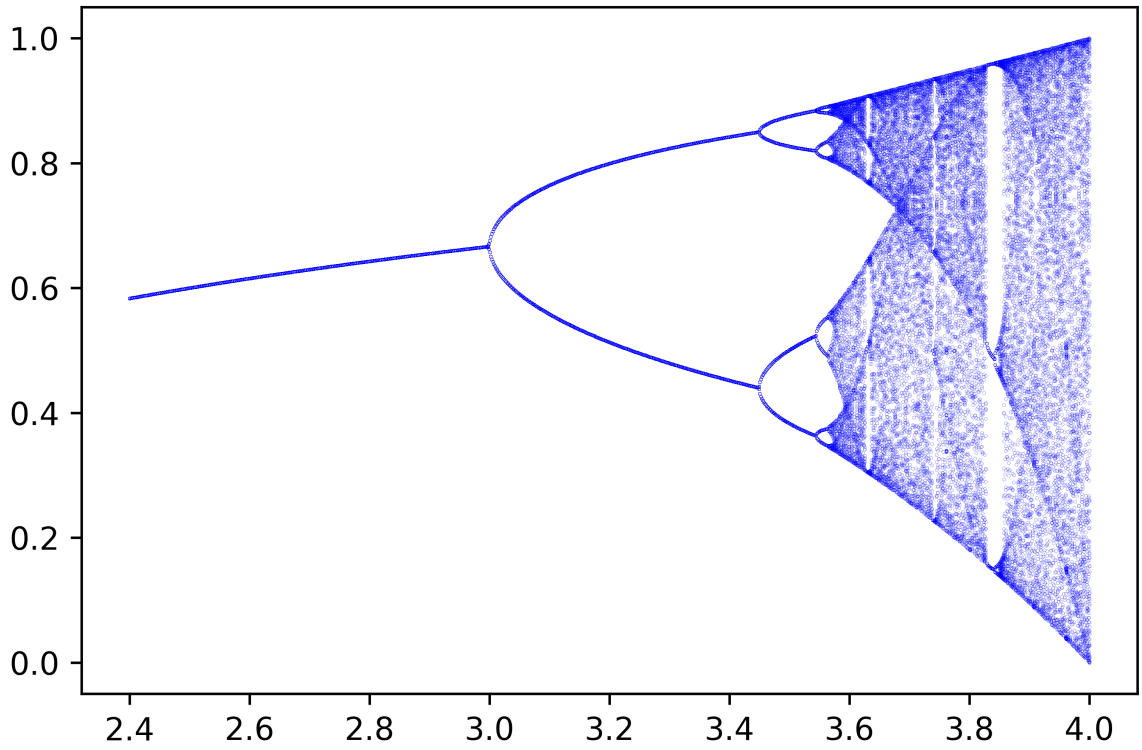
Exercice 7. On peut maintenant tracer le diagramme de bifurcation : on se donne une certaine liste de valeurs de a ; et, pour les éléments de cette liste, on affiche (avec `plt.scatter`) les points $(a,u_m),(a,u_{m+1}),\dots,(a,u_{m+\mathrm{N}})$ où m et N ont été introduits dans l'exercice précédent.
Il faut donc spécifier la liste des a choisie (on rappelle qu'on se limite à $a \in [0,4]$), par une valeur minimale, une valeur maximale, et un nombre de points intermédiaires (se souvenir de la bonne commande `numpy` !!)
On aura aussi besoin de fixer les entiers m et N introduits dans l'exercice précédent.
Enfin, même si le diagramme de bifurcation est insensible à cette quantité, on doit préciser un premier terme u_0 pour nos suites.

Compléter le code suivant :

```
def bifurc(a_min,a_max,nb_a,m,N,u0):
    plt.figure(dpi=600)
    for a in ............ :
        plt.scatter( ..... , liste_termes_asympt( ... , ... , ... , ... ),
                    s=0.01,color='blue',marker='.')
    plt.show()
```

et admirer le résultat sur la page suivante, obtenu avec la commande

```
bifurc(2.4,4,1000,100,100,0.3)
```



Sur ce dernier graphe, on voit alors, par exemple :

- À l'abscisse $a = 2,6$ il n'y a qu'un seul point : la suite (u_n) converge (vers 0,6 environ) ;
- à l'abscisse $a = 3,2$ il y a deux points : la suite (u_n), après un régime transitoire, oscille entre 2 valeurs limites (0,5 et 0,8 environ) .
- ...

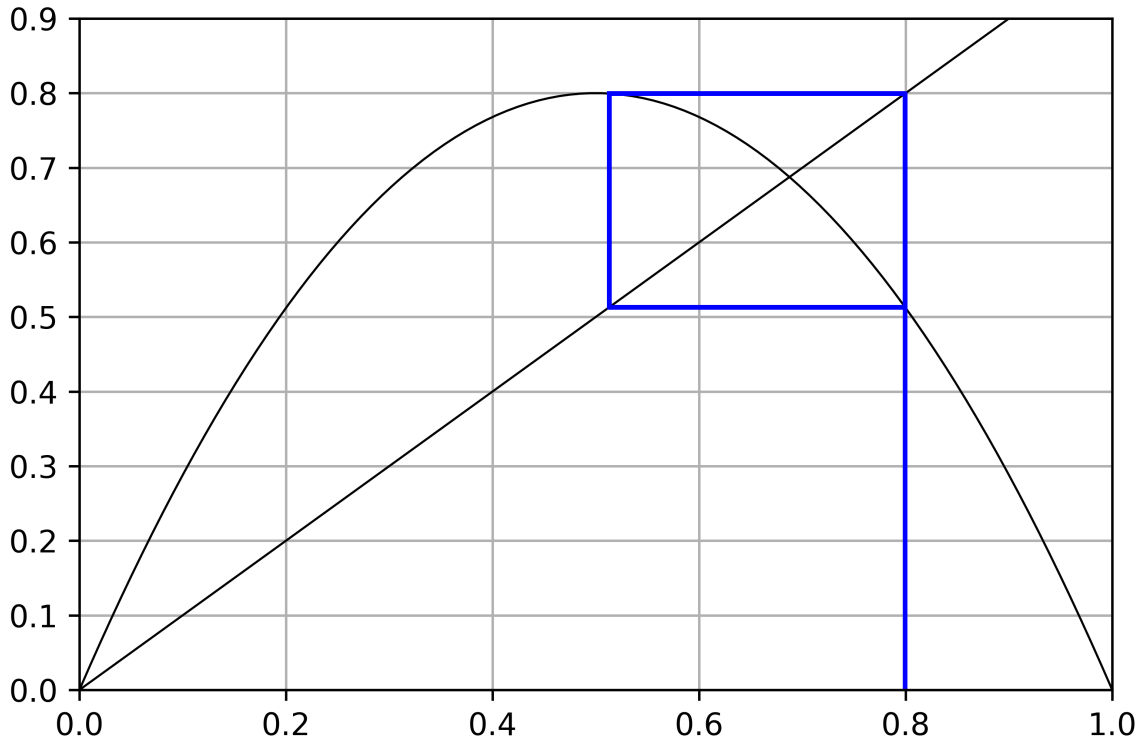
Dans les détails de ce graphe se cachent ensuite plein de phénomènes étranges... à explorer !

6 Bonus : visualisation des « cycles limite »

Dès que la suite ne converge pas, le diagramme en escalier / escargot n'est plus très lisible : on a du mal à séparer le comportement à n grand des premiers termes. Une solution assez simple pour régler ce problème consiste à omettre les premiers termes, et à commencer le tracé des segments à u_m (m assez grand) au lieu de u_0.

Exercice 8. À l'aide d'une observation proche de celle qui nous a permis d'écrire la fonction `liste_termes_asympt`, définir une fonction `trace_asympt(a,u0,m,N)` qui trace `N` étapes du diagramme en escalier / escargot en « démarrant » à u_m.

Voici une figure obtenue avec $a = 3,2$. Comment l'interpréter ?



NB : le premier segment qui part de l'axe des abscisses n'a plus vraiment de sens ici ; mais pour l'enlever il faut réécrire la fonction `trace`. Si vous vous ennuyez....

Regarder ensuite les comportements accidentels : le cycle à 5 points aux alentours de $a \simeq 3,7$; celui à 3 points de $\simeq 3,8$; des exemples de cycles à 2 points, 4 points, 8 points, 16 points (?) obtenus suite aux bifurcations successives observées en $a = 3$, $a \simeq 3,45$, $a \simeq 3,53$,... et le comportement qui reste obstinément chaotique lorsqu'on se rapproche de $a = 4$.