Sensibilité aux conditions initiales et calculs flottants

Marc Lorenzi

22 octobre 2024

Résumé

Depuis bientôt 80 ans, les ordinateurs ont libéré les scientifiques de la tâche ingrate du calcul manuel. Pouvons nous faire aveuglément confiance aux résultats d'un calcul numérique effectué par une machine ? Nous allons voir dans cet article que c'est loin d'être le cas.

1 Un système dynamique

Considérons la fonction $f : [0,1] \longrightarrow \mathbb{R}$ définie pour tout $x \in [0,1]$ par

$$f(x) = 4x(1-x)$$

La courbe de f est un morceau de parabole. On a $f(0) = f(1) = 0$. La fonction f atteint son maximum en $1/2$ et ce maximum est égal à 1.

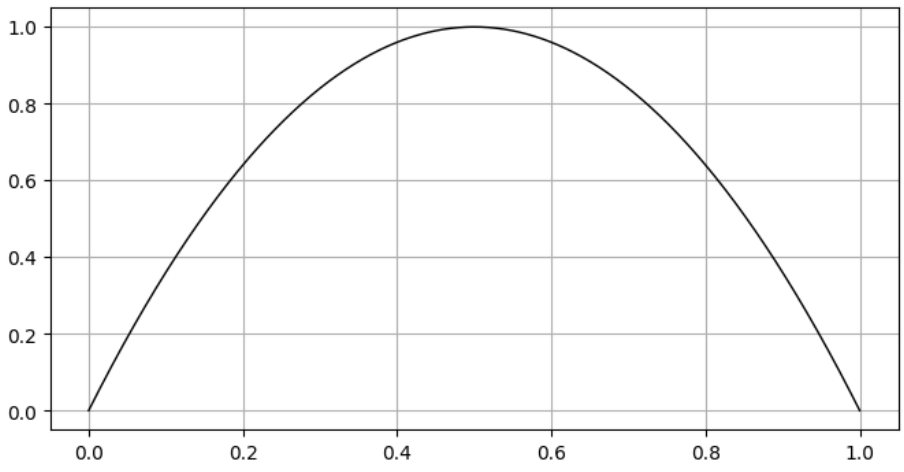

FIGURE 1 – La fonction f

Le *système dynamique* associé à f est la suite $(f^k)_{k\in\mathbb{N}}$ où $f^k = f \circ \ldots \circ f$ (k fois). Pour tout $x \in [0,1]$, *l'orbite* de x est la suite $(f^k(x))_{k\in\mathbb{N}}$. La question que nous nous posons est assez simple : que vaut $f^{100}(0.1)$? Le lecteur peut se demander

en quoi cette question nécessite un article de 6 pages. La question semble être à la limite du stupide. Une simple boucle `for` en Python permet d'obtenir cette valeur, non ?

2 Sensibilité aux conditions initiales

Faisons une expérience. Calculons $f^k(x)$ pour $k = 0, 1, 2, \ldots$ et deux valeurs de $x \in [0, 1]$ « proches », $x = 0.1$ et $x' = 0.1 + 10^{-10}$. Voici les valeurs que nous obtenons avec Python.

k	$f^k(x)$	$f^k(x')$
1	0.36000000000000004	0.36000000032
5	0.5854205387341974	0.5854205439891247
6	0.970813326249438	0.970813322658408
7	0.11333924730376121	0.11333926082939909
8	0.4019738492975123	0.40197389113617815
9	0.9615634951138128	0.9615635279240726
10	0.1478365599132853	0.1478364387611353
20	0.8200138733909665	0.8201481330880324
30	0.3203424751858141	0.4951901852120015
40	0.09790487641603256	0.9553388731535147
50	0.5600367632223772	0.91665363828596
100	0.37244749676375793	0.0008221389553296586

FIGURE 2 – $f^k(x)$ et $f^k(x')$ selon Python

Sur les dernières lignes du tableau, les valeurs de $f^k(x)$ et $f^k(x')$ sont complètement différentes. Voici $|f^k(x) - f^k(x')|$ en fonction de k. Le tracé est en coordonnées semi-logarithmiques.

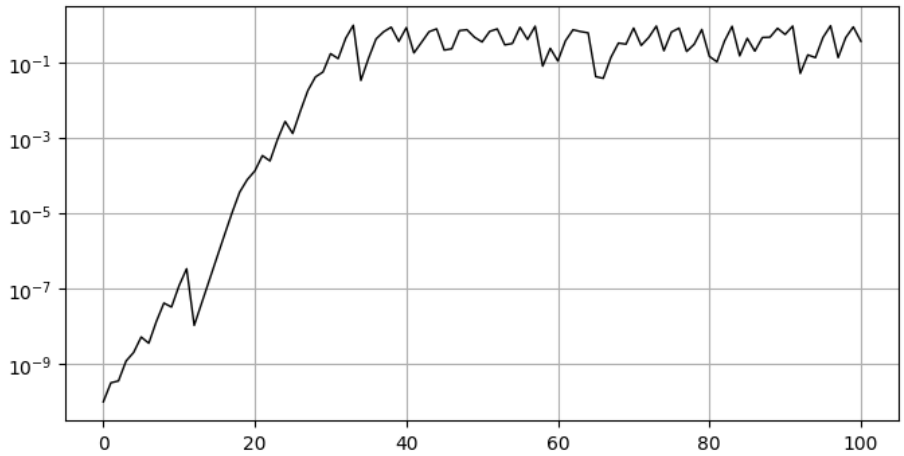

FIGURE 3 – $|f^k(x) - f^k(x')|$ en fonction de k

Au premier abord, cela n'a rien de surprenant. Après tout, même si x et x' sont « proches », n'est-il pas normal que pour de grandes valeurs de k, les réels $f^k(x)$ et $f^k(x')$ deviennent « éloignés » ? Ce comportement du système dynamique associé à f est une manifestation de ce que l'on appelle la *sensibilité aux conditions initiales* (s.c.i.). Plus précisément, on a le résultat suivant.

Proposition 1. [Sensibilité aux conditions initiales]

Il existe $\delta > 0$ tel que pour tout $a \in [0,1]$ et tout $\varepsilon > 0$, il existe $x \in [a-\varepsilon, a+\varepsilon]$ et $n \in \mathbb{N}$ tel que

$$|f^n(x) - f^n(a)| \geq \delta$$

Pour montrer ce résultat, remarquons tout d'abord qu'il n'est pas difficile de calculer $f^n(x)$ en fonction de n.

Lemme 2. *Soit $x \in [0,1]$. Soit $t \in [0, \pi/2]$ tel que $x = \sin^2 t$. On a pour tout $n \in \mathbb{N}$,*

$$f^n(x) = \sin^2(2^n t)$$

Démonstration. La fonction $t \longmapsto \sin^2 t$ est une bijection de $[0, \pi/2]$ sur $[0,1]$. Il existe donc un unique tel réel t. Montrons le résultat par récurrence sur n. Tout d'abord,

$$f^0(x) = x = \sin^2 t = \sin^2(2^0 t)$$

Soit $n \in \mathbb{N}$. Supposons que $f^n(x) = \sin^2(2^n t)$. on a alors

$$\begin{aligned} f^{n+1}(x) &= f(f^n(x)) \\ &= 4f^n(x)(1 - f^n(x)) \\ &= 4\sin^2(2^n t)\cos^2(2^n t) \\ &= \sin^2(2^{n+1} t) \end{aligned}$$

□

Venons-en à la preuve de la s.c.i.

Démonstration. Nous allons voir que $\delta = 1/2$ convient.

Soient $a \in [0,1]$ et $\varepsilon > 0$. Posons $a = \sin^2 \alpha$ où $\alpha \in [0, \pi/2]$. Par la continuité de la fonction $t \longmapsto \sin^2 t$, il existe $\varepsilon' > 0$ tel que pour tout $t \in [\alpha - \varepsilon', \alpha + \varepsilon']$, $\sin^2 t \in [a - \varepsilon, a + \varepsilon]$. Soit $n \in \mathbb{N}$ tel que l'intervalle $I_n = [2^n(\alpha - \varepsilon'), 2^n(\alpha + \varepsilon')]$ soit de longueur supérieure à π. La fonction $\sin^2$ prend toutes les valeurs de l'intervalle $[0,1]$ sur I_n, il existe donc $u \in I_n$ tel que

$$|\sin^2 u - \sin^2(2^n \alpha)| \geq \frac{1}{2}$$

Soit $t = u/2^n$, de sorte que $t \in [\alpha - \varepsilon', \alpha + \varepsilon']$. Soit $x = \sin^2 t$. On a alors

$x \in [a-\varepsilon, a+\varepsilon]$ et

$$\begin{aligned} |f^n(x) - f^n(a)| &= |\sin^2(2^n t) - \sin^2(2^n \alpha)| \\ &= |\sin^2(u) - \sin^2(2^n \alpha)| \\ &\geq \frac{1}{2} \end{aligned}$$

□

3 Pertinence des calculs avec les flottants

Nous allons maintenant réfléchir un peu aux *conséquences* de la s.c.i. Dans *tout* voisinage $[a-\varepsilon, a+\varepsilon]$ de a il y a des réels x tels que $f^n(x)$ soit, pour un certain n, éloigné de $f^n(a)$ de plus de $1/2$. Par exemple, pour $\varepsilon = 10^{-17}$. Sachant que Python calcule sur les flottants avec une précision d'environ 10^{-16}, nous commençons à nous poser des questions. Et si la valeur renvoyée par Python pour $f^{100}(0.1)$ n'était pas correcte ? Je ne parle pas ici de quelques chiffres inexacts mais d'une valeur totalement erronée.

Nous verrons que c'est le cas :

Python renvoie une valeur absolument aberrante.

Histoire d'insister nous allons d'abord constater qu'il y a pire ...

Considérons la fonction $g : [0,1] \longrightarrow [0,1]$ définie par

$$g(x) = 4x - 4x^2$$

Euh, $g = f$ me direz vous ? Oui, mathématiquement parlant. Mais si nous calculons $f(x)$ et $g(x)$ pour un *flottant* x, nous risquons fort de ne pas obtenir *exactement* la même valeur. Refaisons l'expérience en calculant $f^k(0.1)$ et $g^k(0.1)$ pour $k = 0, 1, 2, \ldots$.

k	$f^k(0.1)$	$g^k(0.1)$
1	0.36000000000000004	0.36
5	0.5854205387341974	0.5854205387341977
6	0.970813326249438	0.9708133262494376
7	0.11333924730376121	0.11333924730376266
8	0.4019738492975123	0.4019738492975167
9	0.9615634951138128	0.9615634951138161
10	0.1478365599132853	0.14783655991327294
20	0.8200138733909665	0.8200138734062712
30	0.3203424751858141	0.32034249422053085
40	0.09790487641603256	0.09791729107076982
50	0.5600367632223772	0.5387553642873352
100	0.37244749676375793	0.0970613634295674

FIGURE 4 – $f^k(0.1)$ et $g^k(0.1)$ selon Python

Une minuscule différence de 4×10^{-17} à la première itération fournit des valeurs calculées de $f^{100}(0.1)$ et $g^{100}(0.1)$ totalement différentes.

Voici $|f^k(0.1) - g^k(0.1)|$ (ou plutôt les valeurs *calculées* par Python) en fonction de k.

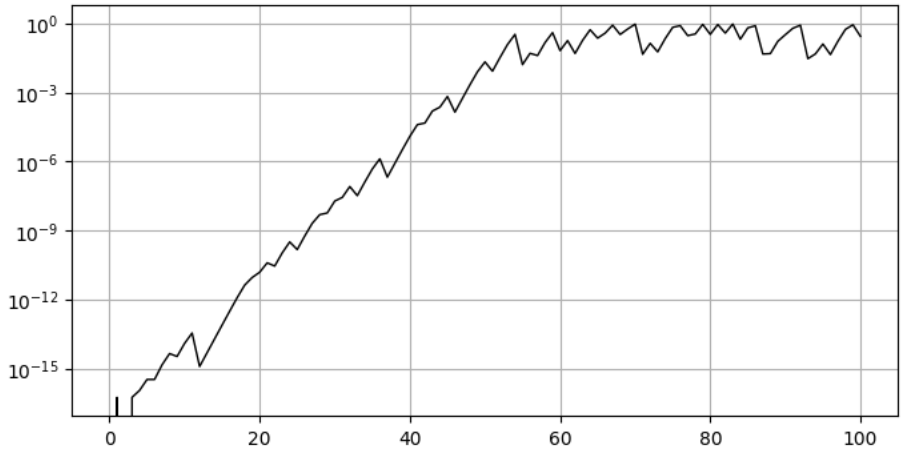

FIGURE 5 – $|f^k(0.1) - g^k(0.1)|$ en fonction de k

Aux alentours de $k = 60$, les valeurs obtenues avec f et celles obtenues avec g n'ont plus rien à voir entre elles. Mais alors, que vaut $f^{100}(0.1)$ (je parle du *réel* 0.1 et pas du flottant) ? Et donc $g^{100}(0.1)$ qui lui est égal ? Vaut-il mieux utiliser f ou g pour calculer ce nombre ? En fait, la conclusion est terrible : ni l'une ni l'autre de ces fonctions ! En faisant les calculs avec une précision bien plus grande que la précision standard des flottants, on obtient

$$f^{100}(0.1) \simeq 0.9301089741865547$$

Cette valeur n'a rien à voir avec le flottant 0.37244749676375793 que nous avions obtenu plus haut en itérant f. Ni d'ailleurs avec le flottant 0.0970613634295674 obtenu en itérant g. Nous sommes donc forcés d'en conclure que les nombres du tableau de la figure 2 sont faux et que la courbe de la figure 3 est fausse.

Les « bonnes » valeurs de $f^k(0.1)$ sont données dans le tableau ci-dessous.

Après toutes les mises en garde soulevées dans cet article, comment puis-je être certain que ces valeurs sont les bonnes ? Eh bien je ne le suis pas. Cela dit, j'ai fait les calculs avec une précision de 10^{-1000} puis je les ai refaits avec une précision de 10^{-10000} et j'ai trouvé les mêmes valeurs. Alors, non, je ne suis pas certain, mais j'ai une relative confiance dans ces valeurs. Le graphique qui suit représente donc sans doute les 100 premiers termes de la « vraie » orbite de 0.1.

k	$f^k(0.1)$
1	0.36
5	0.585420538734198
6	0.9708133262494375
7	0.11333924730376273
8	0.401973849297517
9	0.9615634951138163
10	0.14783655991327213
20	0.8200138734059479
30	0.3203424938183772
40	0.09791702877442154
50	0.5392058548918653
100	0.9301089741865547

FIGURE 6 – Valeurs améliorées de $f^k(0.1)$

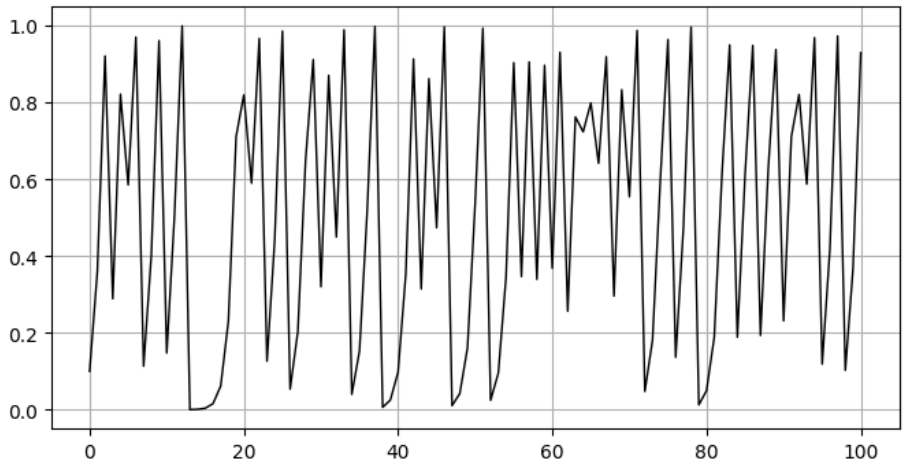

FIGURE 7 – L'orbite de 0.1 ?

Pour conclure, il n'est pas possible, sur le long terme, d'effectuer des calculs numériques fiables sur des systèmes dynamiques qui présentent une sensibilité aux conditions initiales. Ces systèmes ne sont pas exceptionnels, loin de là. Ils seraient même plutôt la généralité. Alors soyons toujours méfiants face à des résultats obtenus par un calcul approché. Même si de merveilleux algorithmes de calcul numérique nous promettent d'obtenir la solution à des problèmes que nous nous posons, la s.c.i. peut faire que ces algorithmes nous fournissent des résultats aberrants. Sommes nous prêts à faire confiance à l'algorithme du pivot de Gauss ? À la résolution approchée d'une équation différentielle obtenue par l'algorithme de Runge-Kutta ? À une intégrale calculée par la méthode de Gauss ? À la solution d'une équation algébrique renvoyée par l'algorithme de Newton-Raphson ? L'auteur de cet article espère que, au minimum, le lecteur aura dans les résultats obtenus par ces algorithmes « une confiance relative et raisonnée » et pas « une confiance aveugle ».