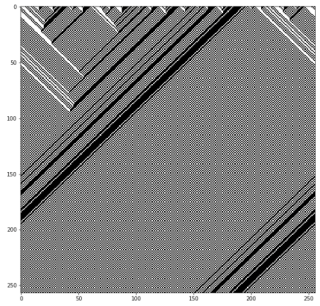
TP automates cellulaires



Hervé Wozniak - Université de Montpellier

19 septembre 2018
v 0.2

Table des matières

Objectifs

Créer et explorer des automates cellulaires en Python

Introduction

Les automates cellulaires (AC) sont des systèmes qui évoluent dans l'espace et dans le temps, avec des règles d'évolution les plus simples possibles. Si leur comportement microscopique est le plus souvent simpliste (qu'il soit déterministe ou probabiliste), le comportement macroscopique présente des caractéristiques qui les apparentent à des systèmes physiques. Ce comportement peut être complexe de façon non-triviale, comme dans le cas de systèmes thermodynamiques simples (par exemple des spins +1 ou -1) qui, au niveau macroscopique, révèlent des propriétés mesurables qui ne sont pas la simple somme des propriétés microscopiques (ferromagnétisme, température de Curie...).

Les AC sont donc utilisés comme alternative aux modèles physiques, pour représenter essentiellement des systèmes dynamiques. A titre d'illustration, on peut mentionner plusieurs problèmes physiques qui ont été étudiés sous l'angle des AC :

- avalanches, transport de la neige par le vent, déplacement des dunes de sable ;
- propagation des ondes ;
- diffusion, percolation ;
- ferromagnetisme ;
- gaz sur réseau (hydrodynamique) ;
- etc.

On trouve également des applications en biologie, chimie, sociologie, environnement, etc.

Certains AC ont des règles moins simples afin de reproduire un comportement physique donné. Mais comme le calcul d'un AC est nettement plus rapide que l'intégration d'équations aux dérivées partielles, il est possible d'explorer plus finement l'espace des paramètres et des conditions qui mènent à certains comportements.

Parmi les AC, ceux qui sont analogues à des systèmes dynamiques déterministes ont été longuement étudiés. En effet, ils présentent la même sensibilité aux conditions initiales et pour certains paramètres, ils peuvent devenir chaotiques, voire produire des structures fractales. On peut montrer que certains AC présentent des attracteurs étranges, structures découvertes dans le modèle de Lorenz (1963) de la convection atmosphérique.

Notions de base sur les automates cellulaires

I

Dans le cas à une unique dimension (1D), un AC est composé d'un réseau (grille) de cellules équidistantes, numérotées avec l'indice i, dont l'état est symbolisé par s_i (unique propriété) qui vaut 0 ou 1.

Cet état s_i évolue en fonction du temps t (discret) selon une règle qui fait intervenir l'état des cellules voisines. Pour un AC d'ordre k, l'état s_i à l'instant $t+1$ dépend de l'état des $k-1$ cellules les plus proches, ainsi que son propre état, à l'instant précédent t. En général k est impair pour respecter la symétrie du voisinage. Les AC les plus simples fixent k=3 (donc une cellule à gauche et une à droite).

Un AC n'est complet que lorsqu'on donne également les *conditions initiales* (valeurs des états de toutes les cellules à l'instant $t=0$) ainsi que les *conditions aux limites* (généralement périodiques).

Plusieurs généralisations sont possibles :

- l'état s_i peut prendre plus de 2 valeurs ;
- la règle de changement d'état peut ne pas être unique (notamment aux limites) ;
- la règle qui permet de changer l'état d'une cellule en fonction du voisinage peut être déterministe ou stochastique (aléatoire) ;
- le changement d'état peut être asynchrone (interaction à vitesse finie) ;
- la portée de l'interaction (k) peut certes augmenter mais on peut également considérer un voisinage non continu (on saute une cellule sur deux par exemple...).

Dans le cas de l'AC de Langton, il y a 8 états et 29 règles ; cet AC reproduit le fonctionnement auto-réplicant d'une cellule vivante. Et lorsqu'on lui ajoute une règle dite de "stérilisation", on fabrique des coraux par fossilisation...

Dans ce TP nous ne verrons que le cas le plus simple (voisinage continu, règle unique et déterministe, état binaire, changements synchrones).

1. Notation de Wolfram

Pour un AC à 1D d'ordre $k=3$, l'état $s_i(t+1)$ dépend de l'état du triplet de cellules : $s_{i-1}(t)$, $s_i(t)$ et $s_{i+1}(t)$. Autrement écrit : $s_i(t+1) = \Phi(s_{i-1}(t), s_i(t), s_{i+1}(t))$.

Ce triplet de cellules peut prendre 8 valeurs auxquelles on associe $\alpha_k = 0$ ou 1 selon la liste suivante :

$$\underbrace{111}_{\alpha_7}\ \underbrace{110}_{\alpha_6}\ \underbrace{101}_{\alpha_5}\ \underbrace{100}_{\alpha_4}\ \underbrace{011}_{\alpha_3}\ \underbrace{010}_{\alpha_2}\ \underbrace{001}_{\alpha_1}\ \underbrace{000}_{\alpha_0}.$$

Ainsi, le nombre binaire formé des valeurs des α_k (le bit de poids faible α_0 étant à droite) représente la règle de changement des états en fonction du voisinage. Ce nombre binaire de 8 bits $\alpha_7\alpha_6\alpha_5\alpha_4\alpha_3\alpha_2\alpha_1\alpha_0$ représente un entier entre 0 et 255, appelé la *règle* $\mathcal{R}$ de l'AC.

Ces 256 règles ont été étudiées par Wolfram (qui a introduit cette notation) en 1983 et bien d'autres depuis.

Exemple

Étudier un AC de règle 110 (donc 01101110 en binaire) signifie que la règle de changement d'état est la suivante :

$$\underbrace{111}_{0}\ \underbrace{110}_{1}\ \underbrace{101}_{1}\ \underbrace{100}_{0}\ \underbrace{011}_{1}\ \underbrace{010}_{1}\ \underbrace{001}_{1}\ \underbrace{000}_{0}$$

La règle distingue souvent le côté gauche $i-1$ du droit $i+1$. Mais de nombreuses règles sont identiques du fait de symétries.

Un AC s'étudie en faisant varier le temps. Un système qui change d'état en fonction du temps est un système dynamique. Lorsque l'espace et le temps sont continus, la physique statistique se charge d'en prédire le comportement (équation de Liouville, etc.).

Pour un AC l'espace et le temps sont discrets. C'est donc un système dynamique discret.

Une propriété importante des AC est que les changements d'état sont synchronisés (tous les états changent en même temps).

Fondamental

Du point de vue programmation, un AC est un objet très simple.

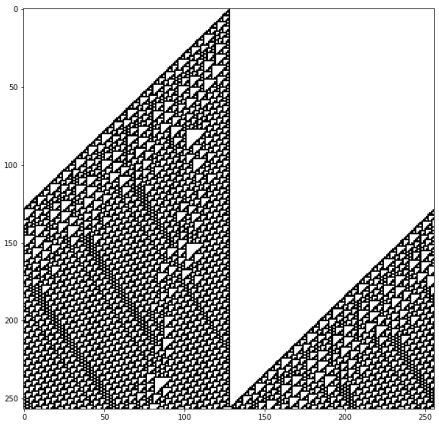
Le temps est discret, donc il s'agit seulement d'itérer sur une variable de temps. Pas d'équations différentielles.

L'espace est discret, donc nul besoin d'interpoler. Pas d'équations aux dérivées partielles.

Les états sont 0 ou 1, donc l'arithmétique est booléenne ; il n'y a *aucune perte d'information* ou de *précision* lors des calculs.

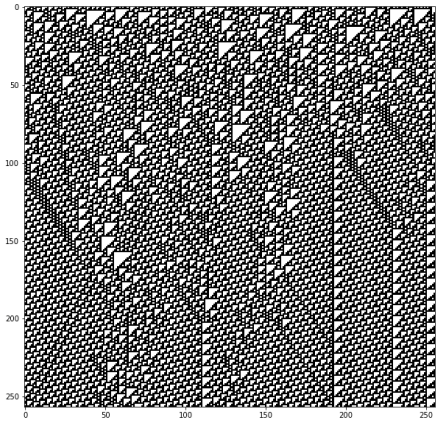
Exemple d'un AC de règle 110

Cet exemple illustre l'importance des conditions initiales.



ca(rule=110, largeur=256, niter=256, order=3, init='centre')

ca(rule=110, largeur=256, niter=256, order=3, init='random')



2. Exercice n°1

Objectif

Convertir une règle $\mathcal{R}$ numérotée selon la convention de Wolfram en une liste python (`list`) faite de 0 et de 1 contenant la représentation binaire sur 8 bits, le bit de poids faible étant à gauche afin qu'il corresponde à l'élément [0] de la `list`.

En entrée, la fonction prend le nom de la règle et l'ordre de l'AC (k=3 dans le cas Wolfram, mais la fonction est facilement généralisable pour traiter toutes valeurs de k).

```
>>> rule_to_binlist(126,order=3)
[0, 1, 1, 1, 1, 1, 1, 0]
>>> rule_to_binlist(110,order=3)
[0, 1, 1, 1, 0, 1, 1, 0]
```

3. Exercice n°2

Objectif

Il s'agit d'écrire une fonction qui, pour un k-uplet d'états de cellules en entrée, renvoie la conversion en valeur entière (`int`).

Autrement dit, pour un automate d'ordre 3, le triplet sous forme de `list` [0,1,1] renvoie la valeur `int` 3 qui sera interprétée comme étant α_3 :

```
>>> state([0,1,1])
3
>>> state([1,0,1])
5
```

4. Exercice n°3

Objectif

On considère une liste d'états de longueur N qui constitue la *condition initiale* de l'AC.

Une fonction (appelée *fenêtre glissante*) est appliquée à cette liste. On l'appelle *fenêtre* car elle retourne une extraction de k valeurs de la liste d'origine centrée sur une cellule donnée. On l'appelle *glissante* car à chaque appel on décale l'extraction d'une cellule vers la droite (indice de `list` croissant).

⚠ *Attention*

A l'itération 0, cette fonction retourne une liste de k valeurs dont la cellule 0 est au centre. Cela implique que $\frac{k-1}{2}$ valeurs se trouvent à gauche de 0.

Pour réaliser cette condition, on applique des *conditions aux limites périodiques*. On va donc copier les valeurs qui se trouvent tout à droite de la liste.

A l'itération i, elle retourne une liste de k valeurs centrée sur la cellule$i$.

A l'approche des dernières itérations, la liste est complétée avec les valeurs à gauche de la liste.

Complément : Python avancé

Au lieu d'appeler N fois une fonction qui ne renvoie qu'une seule valeur (fonction se terminant par `return`), on peut faire un usage avantageux d'une fonction génératrice se terminant par l'instruction `yield`, qui, combinée à une boucle dans la fonction, permet de ne l'appeler qu'une seule fois.

S'exercer sur l'exemple ci-dessous avant de l'utiliser pour l'AC.

```
def test_yield(niter):
    for i in range(niter):
        yield i

for j in test_yield(10):
    print(j)
```

```
7
```

Appliqué à la fonction demandée, cela peut donner le comportement suivant, où l'on constate que la liste renvoyée contient toutes les fenêtres de triplets possibles :

```
>>> print(list(moving_window([1,2,3,4,5,6], wsize=3)))
[[6, 1, 2], [1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], [5, 6, 1]]
```

5. Exercice n°4

Objectif

L'AC étant défini comme l'application d'une règle sur un k-uplet, il s'agit maintenant de créer une fonction qui réalise les changements d'état.

Pour un k-uplet en entrée, la fonction retourne le k-uplet modifié par la règle. Puisque les cellules d'un AC changent d'état de façon synchrone, on peut profiter de cette fonction pour parcourir toutes les cellules. Ainsi, on applique la fonction sur la liste complète des cellules. En sortie on récupère la liste mise à jour avec les nouveaux états.

On tirera partie des exercices précédents :

- l'exercice 2 permet de trouver la valeur de α qui viendra remplacer l'ancien état ;
- l'exercice 3 déplace une fenêtre de largeur k le long de la liste des cellules.

```
>>> print(apply_rule([0,0,1,0,0],rule_to_binlist(rule_name=110,order=3),order=3))
[0, 1, 1, 0, 0]
```

6. Exercice n°5

Objectif

Toutes les fonctions de base ont été construites. Il faut maintenant prévoir l'avancement dans le temps qui se réalise sous forme d'itération synchronisée pour toutes les cellules.

On écrit donc une fonction qui, à chaque appel, avance d'un pas de temps l'AC.

Autrement dit, à un instant $t+1$, la fonction applique la règle $\mathcal{R}$ sur l'état des cellules pris à l'instant t.

⚠ *Attention*

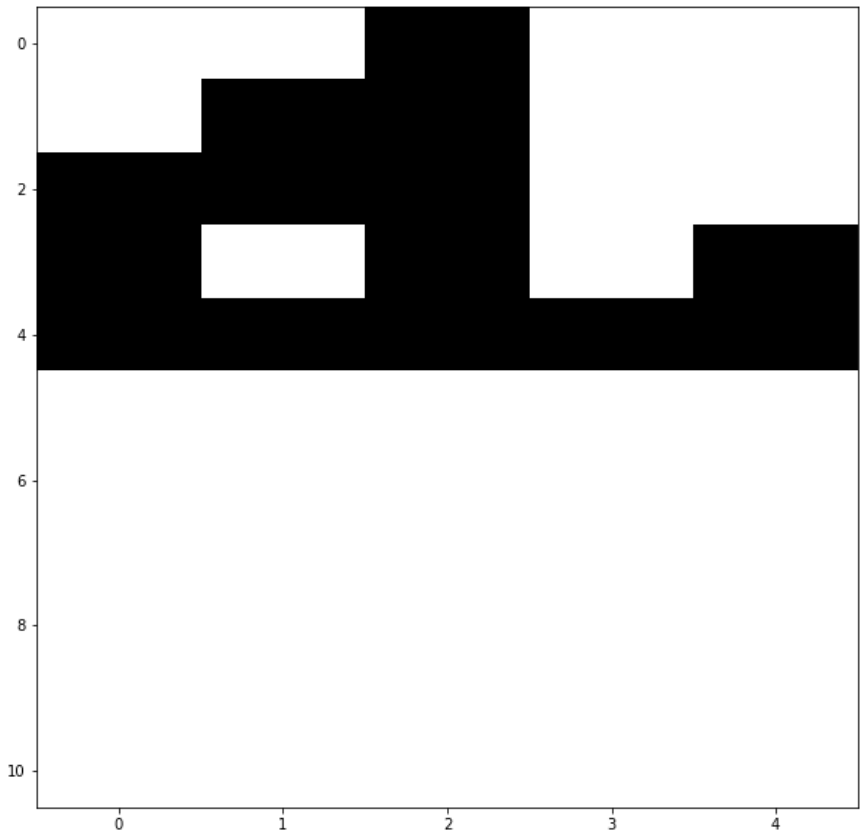
On se déplaçant de gauche à droite, il faut s'assurer que l'état de la (ou des) cellule de gauche soit pris à l'instant t et non t+1. A priori ce problème doit avoir été résolu à l'exercice n°4.

Un exemple de résultat attendu ;

```
>>> output=ca_iterate(init_cond=[0,0,1,0,0],rule=110,order=3,niter=10)
>>> print(list(output))
[[0, 0, 1, 0, 0], [0, 1, 1, 0, 0], [1, 1, 1, 0, 0], [1, 0, 1, 0, 1], [1, 1, 1, 1,
 1], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0
, 0, 0], [0, 0, 0, 0, 0]]
```

La fonction d'itération peut renvoyer une liste de liste (ou un objet *generator* produit par l'instruction `yield`) qui se trace alors comme suit :

```
>>> import numpy as np
>>> tab=np.array(list(output))
>>> import matplotlib.pyplot as plt
>>> plt.figure(figsize=(10,10))
>>> plt.imshow(tab,origin="upper",cmap="Greys",aspect='auto')
```



Conclusion

Nous pourrons voir d'autres types d'AC dans la suite des TP (AC probabilistes, 2D, totalisateurs, Q2R, etc.). Pour certaines règles, le résultat d'un AC dépend fortement des conditions initiales. A vous de jouer.

Les règles 'cosmétiques' :

- 150, 195, ... : triangles de Sierpinski
- 184 : couverture du TP
- 90 : converge vers un état stable à longueur/2
- 110 : des structures apparaissent en fonction des conditions initiales
- 17, 19, 23, etc... : avec des conditions initiales aléatoires mais à comparer avec un 1 au centre