複雑系プログラミング特論
Celluar Automataとλパラメーター

概要

これまで，単純だけれども条件によって複雑な挙動を示す個体群動態についてプログラムをつくり，matplotlibを使った可視化方法についてお話ししてき ました．

　今回は，複数の構成要素のシンプルな相互作用から創発する系全体の複雑な挙動を理解するのによく用いられるセルオートマトンを取り上げます．そのなかでも，一番 シンプルなElementry Cellular Automataと呼ばれる，高々256種類のルール（実際には同等なルールを除くともっと少ない）の中に，様々な挙動が生じることと，ＣＡの定性的な挙動を予測するのに提案されたλパラメータについて紹介し，λパラメータと系の挙動の関係について論ずるためのプログラミング手法を学びます．

（公開に際して一部図や内容を変更・省略しています．）

セルオートマトン（Cellular Automata, CA）

セルオートマトンとは，一般に，離散値で表された状態を持つセルと呼ばれる状態遷移機械を，一次元や2次元空間などに規則正しく並べ，次の離散時間における各セル の状態が自分自身と近傍のセルの状態に依存して決まるようにした系のことです．

もともとは，数学者ウラムとノイマン型の現在主流の計算機をつくった（また，ゲーム理論の創始者でもある）フォン・ノイマンが，自己複製（自分で自分のコピーをつ くる）する機械をつくることができるかについて考えているとき，その土台としてつくり出された概念です．

彼は2次元のセルオートマトン上で自己複製する機械の設計を検討し，このような機械は，対応する設計図があればそれを読んで何でもつくることができる機械，設計図のコピー機，それらを一定の手順で操作する機械（とそれら全体の設計図）から構成されればよいことを示しました．

その後，1970年代のコンウェイが見つけたライフゲーム（Game of Life）のブーム等を経て，1980年代には，Mathematicaの開発者でも有名なウルフラムが，1980年代に最もシンプルな一次元 CA（Elementary Cellular Automata）について調べ，秩序とカオスや複雑系に関わる重要な知見が得られました（後述）．

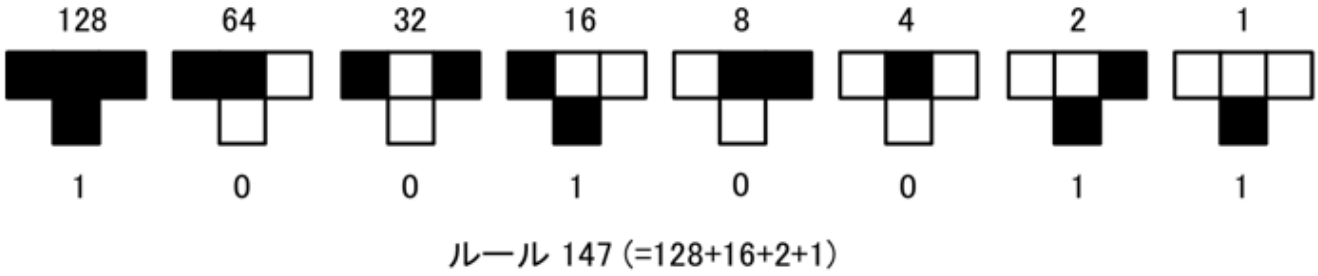
現在では，その基本的な数理的性質などの解析に限らず，交通流や生態などの自律分散系の挙動を抽象化するためのモデルとしても幅広く活用されています．

Elementary Cellular Automata (ECA)

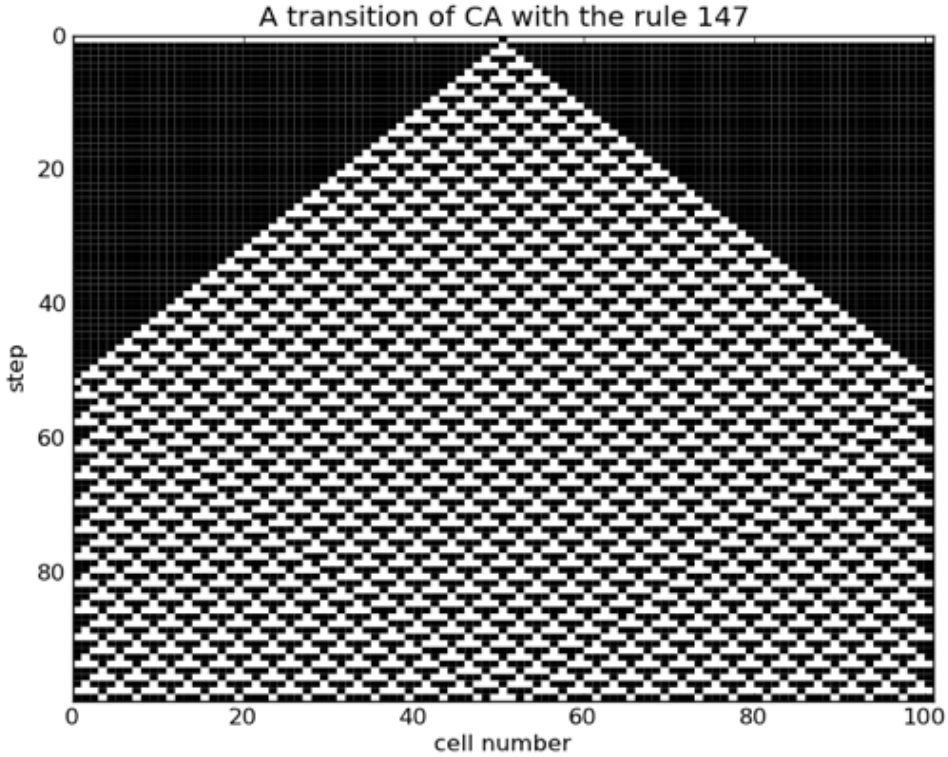
ウルフラムが解析したElementary Cellular Automata(ECA)とは，2状態3近傍の一次元セルオートマトンのことです．これは，一直線に並べられた各セルが，自分自身の状態0か1とその両隣の状態からなる 3ビットの近傍の状態から，次のステップにおいて自身の状態0であるか1であるかを決めるルールに従って動くという，極めて簡単なものです．なお，今回は端と端が つながったリ

ング状のセルオートマトンを考えます.

3ビットで表される近傍の状態は2^3=8通りあります. そのそれぞれに対して, 0か1を割り当てれると一つの完全な遷移ルールができるので, 2^8=256通り の遷移ルールがあり得ます.
ウルフラムは, 8つの各状態からの遷移先の値を並べた8ビットの値を10進数で読み替えることでルールにコード番号をつけました. 以下はルール147と, それを 使って100ステップＣＡを遷移させた例です.
※黒いセルは1, 白いセルは0に対応します.



ルール 147 (=128+16+2+1)



一点だけが1という状態でこのルールを元に遷移を始めると, この場合, 小さな凸型のパターンが繰り返される周期4のサイクルに収束することがわかります.

ECAのpythonでの実装

今回は, 上のようなグラフを表示するプログラムを以下のように用意しました.

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  import random as rnd
4
5  def ca_1d(l, t, rule, cell_i):
```

```
    cell= cell_i
    data= [cell]
    for i in range(t):
        cell_next= [0 for i in range(l)]
        for j in range(l):
            neighboringstate= cell[(j-1+l)%l]*4+cell[j]*2+cell[(j+1)%l]
            cell_next[j]= rule[neighboringstate]
        cell= cell_next
        data.append(cell)
    return(data)

L=101
T=100
SEED=100
rnd.seed(SEED)

RNO= 90
RULE= [(RNO>>i)&1 for i in range(8)]

#[0, 0, ..., 0, 1, 0, ..., 0, 0]
cell_init= [0 for i in range(L)]
cell_init[L//2]= 1

#random
#cell_init= [rnd.randint(0, 1) for i in range(L)]


dataXY= ca_1d(L, T, RULE, cell_init)

fig= plt.figure(figsize=(5, 6))
ax= fig.add_subplot(1,1,1)
ax.pcolor(np.array(dataXY), vmin = 0, vmax = 1,  cmap= plt.cm.binary)
ax.set_xlim(0, L)
ax.set_ylim(T-1, 0)
ax.set_xlabel("cell number")
ax.set_ylabel("step")
ax.set_title("rule" + str(RNO))
plt.show()
```

上半分が実際にCAの計算をする関数ca_1d，真ん中が初期化と関数の実行，最後がmatplotlibを使ったグラフ表示に対応します．
前後しますが，関数の前に初期設定等について説明すると，まず，セルの数を表す変数L，CAをまわす回数を表すT，ルールの通し番号を表すRNOが定義されていま す．
さらに，定義したRNOを使って，その通し番号を2進数表現にしたときの小さい桁から値(0か1）を順に並べたリストをつくり，変数RULEに割り当てています． 今回は，iビットシフトの演算子>>と，論林和演算子&，リストの内包表現を組み合わせて表現しています．
最後に，cell_initにセルの初期状態を表すリストを代入しています．

関数ca_1dは，セルの数l，CA実行回数t，ルールのリストrule，セルの初期状態を入れたリストcell_iを受け取ります．この関数は，実験結 果，つまり，i番目のセルの時刻tの状態がdata[i][j]に入っている2次元配列型のリスト（リストのリスト）dataを作成して，それを返り値として返し ます．そのために，関数の中で次の事を行っています．

- 現在のセルの状態を表すリストcellを定義し，cell_iで初期化
- 実験結果を入れるdataに現在のcell（リスト）が一番目の要素になっているリストを割り当

て．
- t回分以下を繰り返す
    - 次のセルの状態を入れる長さlで0からなるリストcell_nextを作成
        - l回分以下を繰り返す
            - l番目のセルの次の状態を計算するとして，l-1番目，l番目，l+1番目のセルの状態から，対応する近傍状態の番号を算出
            - それをルールのリストの添え字として使って次の状態を算出し，cell_next[]に代入
        - cell_nextをcellに割り当て
        - cellをdataに追加する
- dataを返す

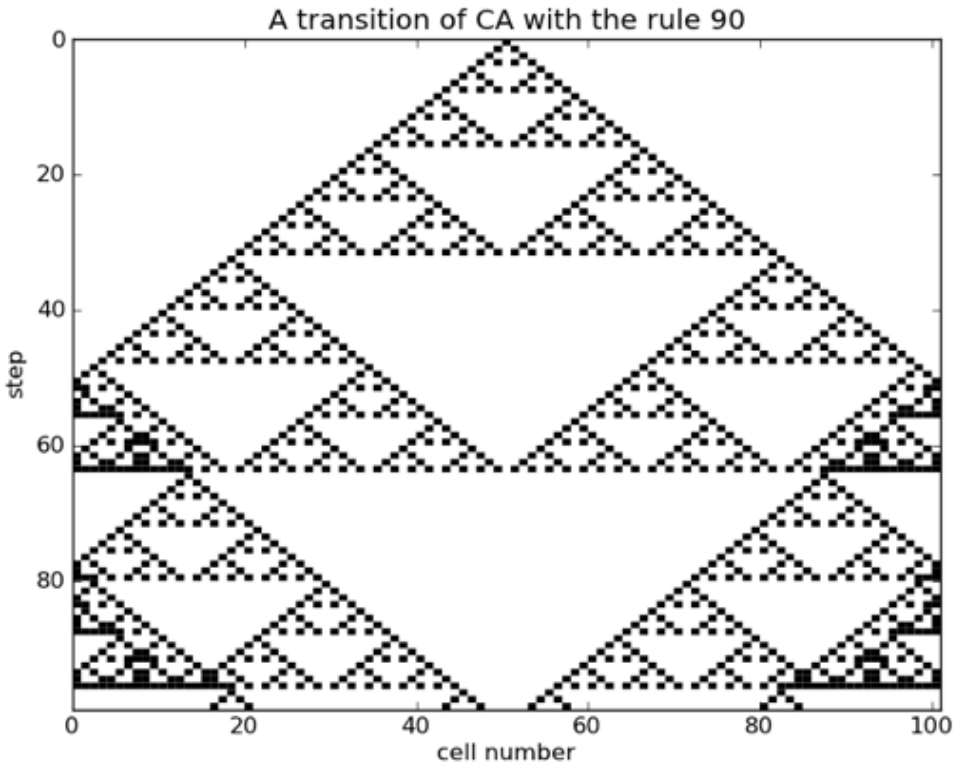
残りはmatplotlibのplt.pcolor関数を使ったグラフ表示です．ca_1d関数から結果を受け取ったdataXYは，行が時刻，列が各セ ルに対応しているので，そのままpcolor関数に入れてちょうど良い訳です．ただ2つだけポイントがあります．一つは，cmapという色の割り当てを決める関数 で二値を表すplt.cm.binaryを指定することです．もう一つは，y軸の範囲を決めるとき，T-1から0として，y軸の向きを逆さにする と，上から下へ向かう時間変化が表せるということです．

ECAの様々な挙動

ECAには，よく知られた典型的な挙動を示すルールがいくつかあります．

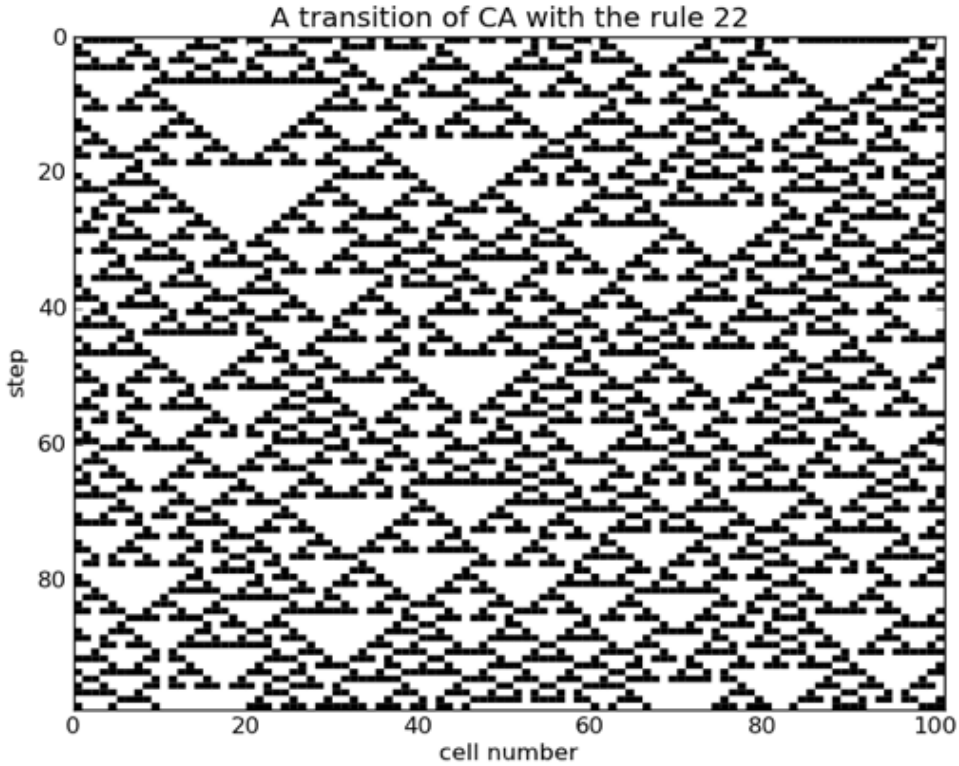
ルール90：シェルピンスキーのガスケット

フラクタル構造を持つ図形として有名．



ルール22：貝殻模様

巻き貝や二枚貝の貝殻のパターンに似ている.

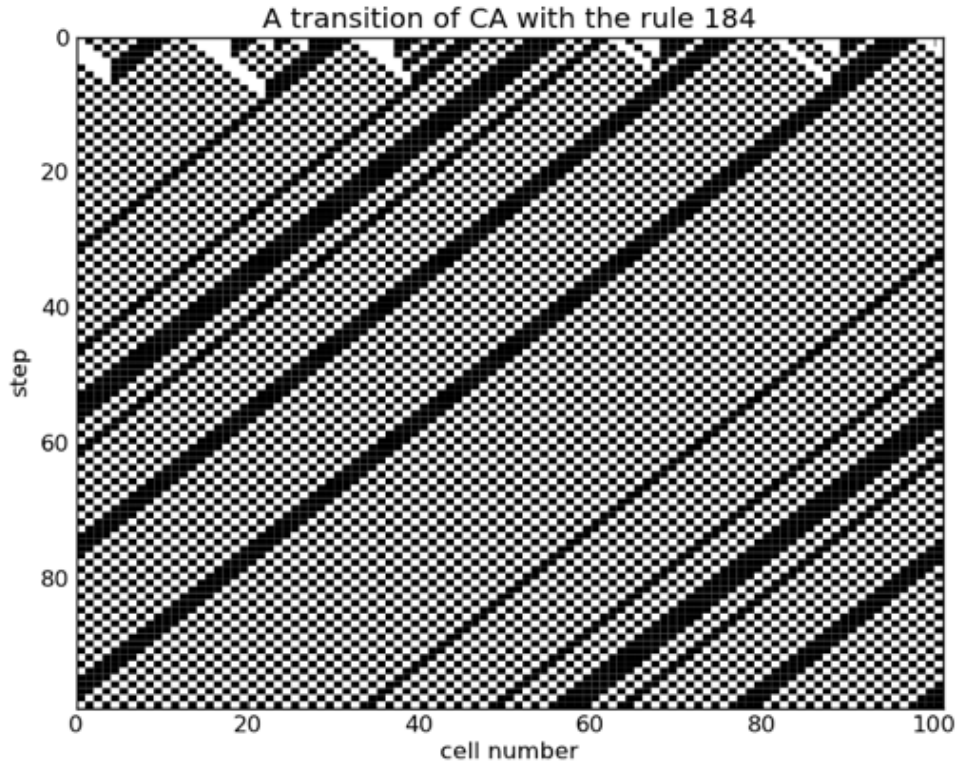


こんなの

wikipediaより
（http://ja.wikipedia.org/wiki/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:Textile_cone.JPG）

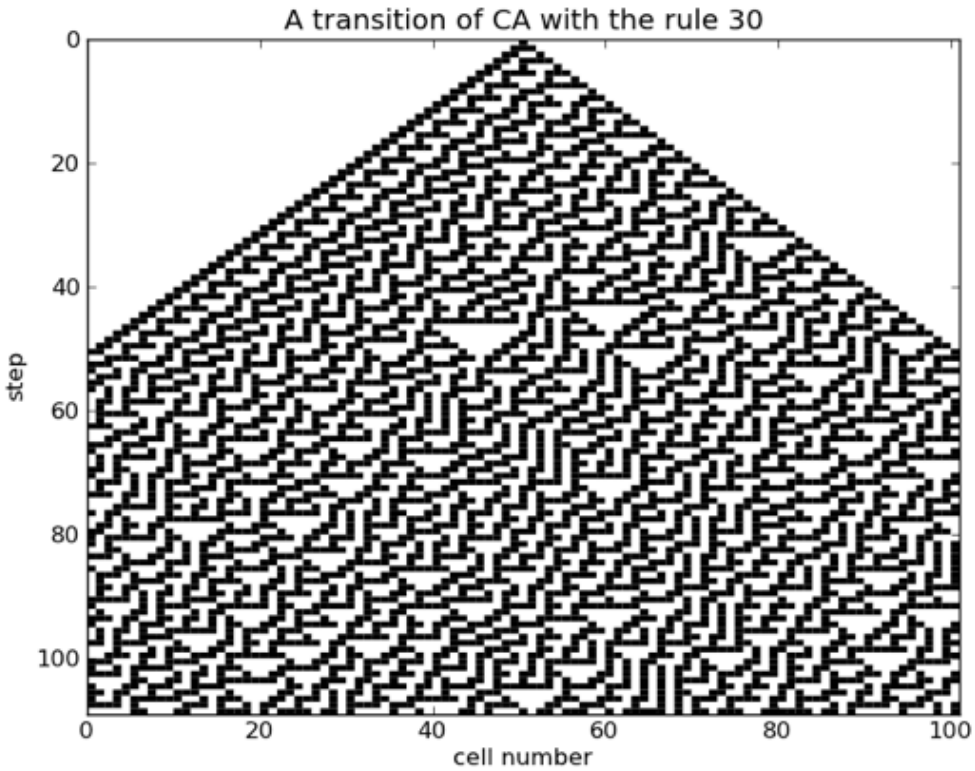
ルール184：交通流

黒い車が左から右へ. 一つ前があいていたら一つ進む. あいていなかったらその場に留まる.
渋滞（黒のかたまり）が進行方向とは逆向きに流れているのがわかる.



ルール30：カオス（的）

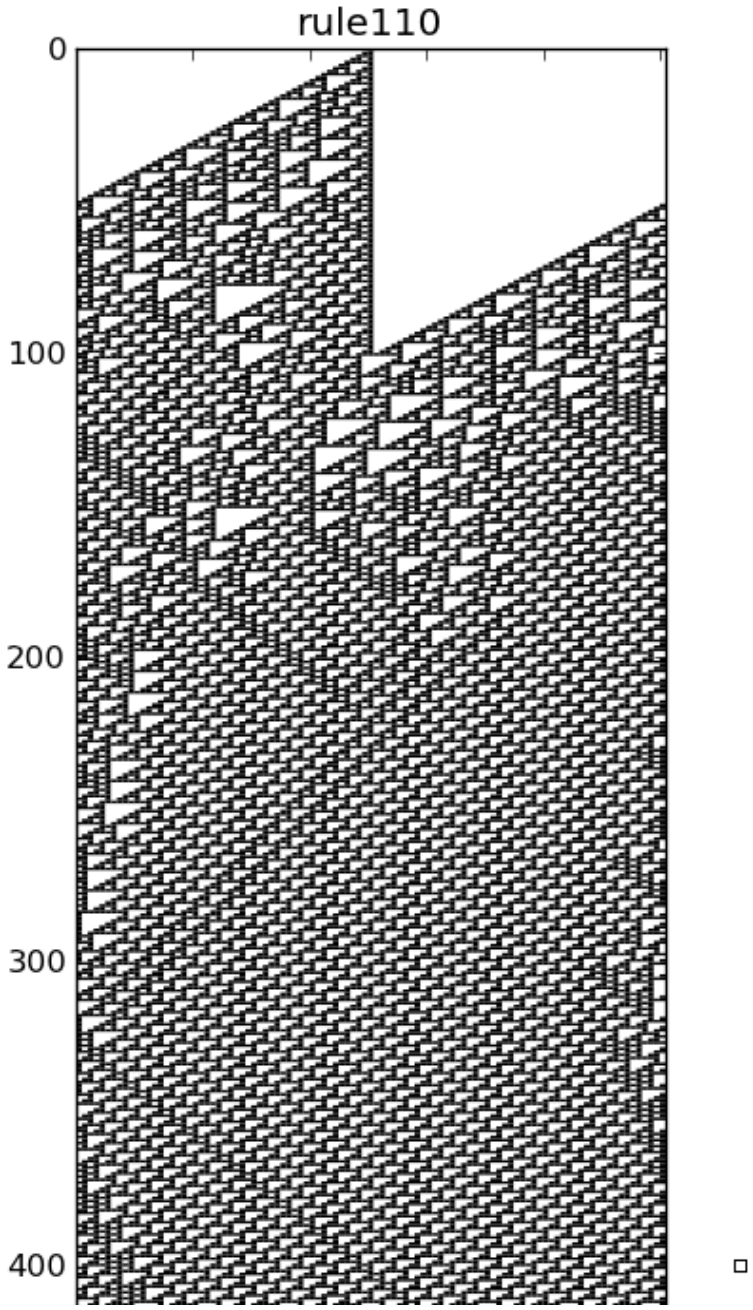
一点だけが黒の状態からはじめても，しばらくすると乱雑な状態へ.

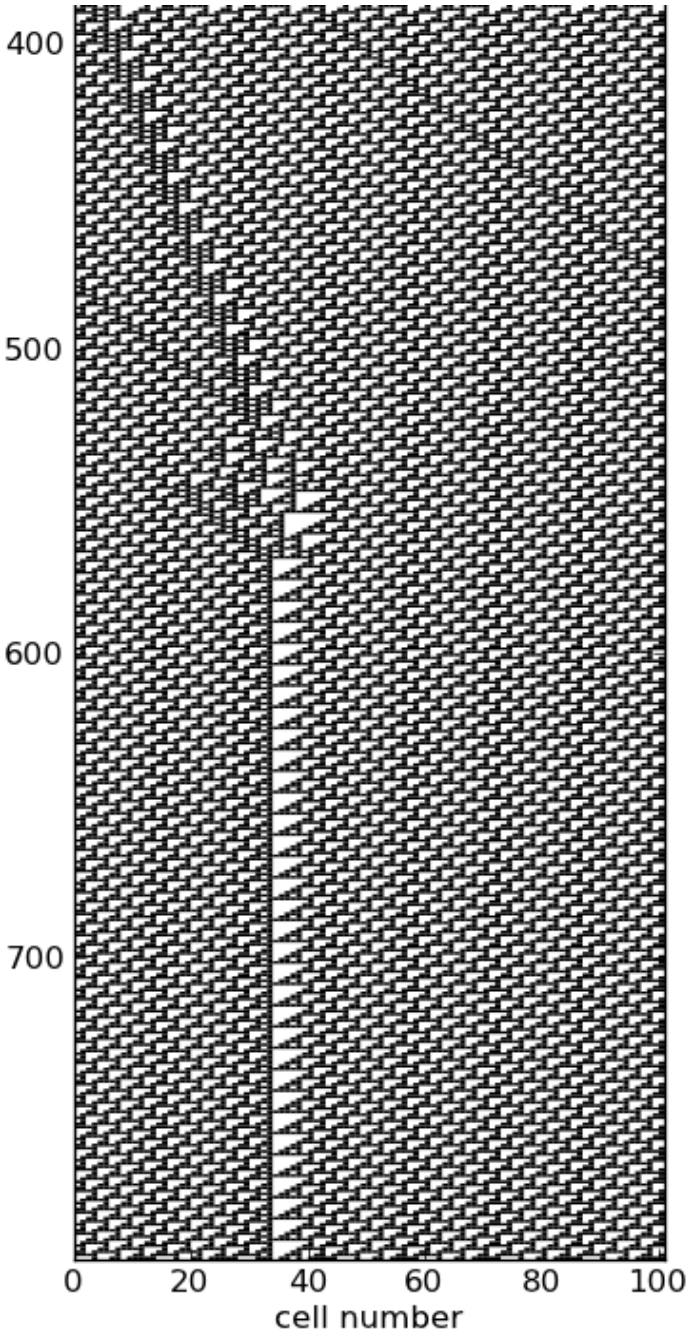


ルール110：計算万能

このルールはいわゆる計算万能性を持ったものとして知られています．つまり，効率は別にして，原理的には，対応する初期値をこのCAに与えて動かせば，どんなプロ グラムの計算もできるということです．小さな構造が徐々に伝播していって，それらがぶつかったりして新たな構造が

出てくるのが特徴的です.





4つのクラス

ウルフラムは上のような様々なECAの挙動を以下のように大きく4つに分類したことでよく知られています.

- クラス１：系全体の変化が完全に止まり，収束.
- クラス２：系の挙動が周期的なパターンを繰り返す.
- クラス３：セル全体が乱雑な挙動を示し続ける．カオス.
- クラス４；規則的，不規則的なパターンが共存し，全体として時間的・空間的複雑.

ラングトンのλと複雑さの指標

ところで，多くの場合，あるルールがどんな振る舞いを生じさせるか，つまり，上のクラスのどれに当てはまるかは，ルール自体を見ただけでは見当がつきませ ん．そこで，人工生命研究の創

始者ラングトンは，セルオートマトンの挙動をルールで分類するための指標として，λパラメータと呼ばれる値を定義しました．
K状態N近傍のセルオートマトンを考えたとき，λは以下のように定義されます．

$$\lambda = \frac{K^N - n}{K^N}$$

但し，nは，ルールにおいて，任意に決めた一種類の状態（よく死状態と呼ばれる）に遷移する近傍の状態の数（上のようなルール表に現れる死状態の数のこと）．

ラングトンは，論文（Langton, C. G.: Computation at the edge of chaos: Phase transitions and emergent computation, Physica D, 42(1-3): 12-37 (1990)）中で，λパラメータとCAの挙動に関する解析を総合して次の模式図を描き，ウルフラムのクラス分けとの対応関係について論じました．

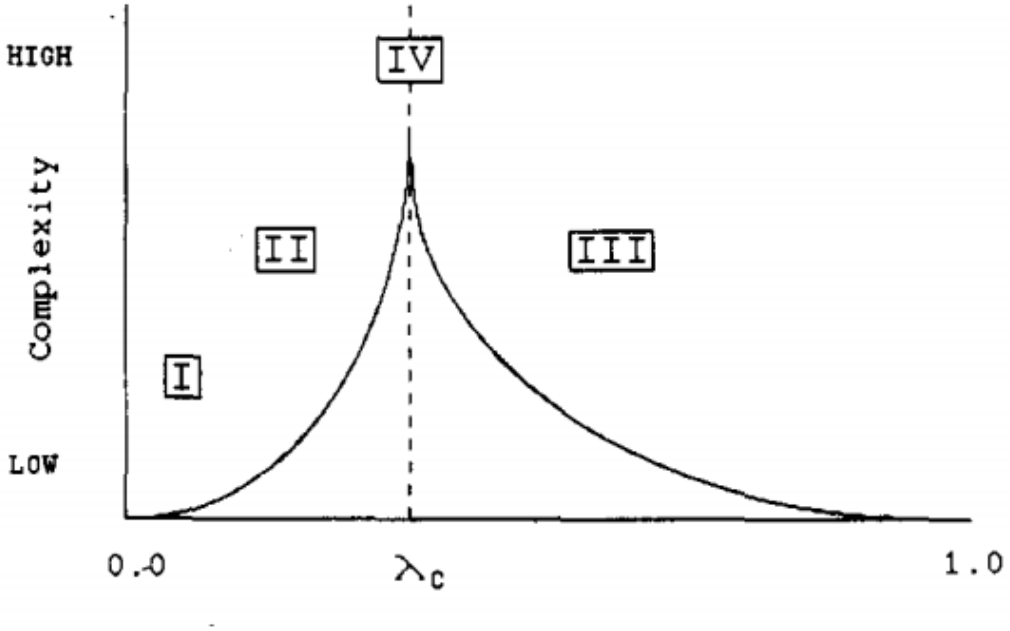


Fig. 16. Location of the Wolfram classes in λ space.

（Langton, C. G.: Computation at the edge of chaos: Phase transitions and emergent computation, Physica D, 42(1-3): 12-37 (1990)より）

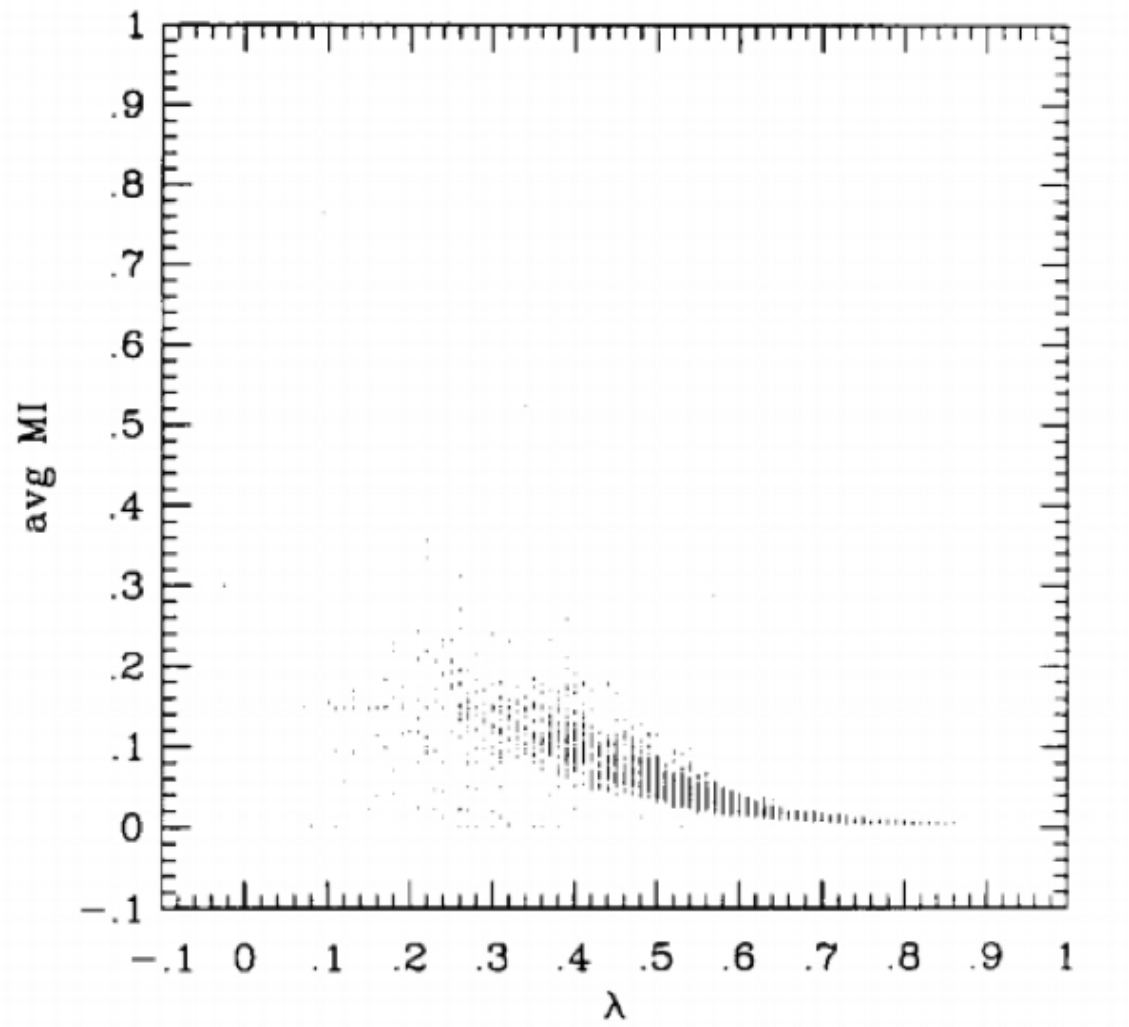
クラス１はλが小さい値のところに存在し，その隣にクラス2が存在し，ピークに近づく．一方クラス3はピークの反対側にあり，クラス4はピークの頂点，クラス２と ４の間の小さい区間に存在すると言い，ここを”カオスの縁”と呼びました．

最初のロジスティック成長式の場合は，内的自然増加率rが系の挙動を決めていましたが，今回は，このλが先に挙げたような複雑なＣＡの挙動を把握する指標になるという主張です．

複雑さの軸って何？

ところで，ラングトンの図にあった縦軸の”複雑さ”って，何でしょうか．ラングトンは論文中でい

くつかの解析を行っていますが，上の図に直接関係する実験データが，以下の図です．



（Langton, C. G.: Computation at the edge of chaos: Phase transitions and emergent computation, Physica D, 42(1-3): 12-37 (1990)より）

これは，横軸をあるCAのλパラメータ，縦軸を，そのルールを使ってCAを動かしたときの，相互情報量（Mutual Information, MI）にして，両者の関係を表した図です．

やや見にくいですが，λ最小から徐々にMIが増加し，２から３付近でピークを迎え，その後は徐々に減少しており，元の模式図とよく似ています．

相互情報量

皆さんは情報理論についてはおそらくどこかで一度は話を聞いたことがあるかと思います．なので，やや釈迦に説法かもしれませんが，復習のために関連事項を少し紹介 します．

情報量

確率P(E)で起きる事象Eが生じたことを知ったときに受け取る情報量I(E)は以下で定義されます．

$$I(E) = -\log P(E)$$

コインを一回投げて表が出る事象Oの確率P(O)は1/2です．なので，その情報量はI(O)=-log1/2=1ビット(logの底が2の場合）となります．

サイコロを振って出た目を知ると，その情報量は2.585...です．これは，コインを2.5回投げたときと同じくらいの情報量を得たことになることを示していま す．

平均情報量，エントロピー

ある有限の集合Uの値をとる確率変数Xがあって，それが確率分布P(x)に従うとき，確率変数Xのエントロピーー H(X)は次で定義されます．ただし，

$$H(X) = -\sum_{x \in U} P(X = x) \log P(X = x)$$

ただし，P(A)=0のとき，P(A)logP(A)=0と定義．

この値は，もしXが一つの値しかとらないときは最小になり，Xが全ての値をまんべんなくとるときに最大になる．
いつも”晴れ”としか言わない天気予報士の予報を聞いても何の情報量もないけど，いろんな予報をする人から"晴れ”と聞けば，それは情報量が高い（あたっているか どうかは別にして）．

結合エントロピー

xとyがそれぞれ確率変数X, Yに従う場合，その組み合わせ(x, y)も確率変数と見なすことができます．この確率変数(X, Y)のエントロピーは結合エントロピーと呼ばれ，以下で定義されます．

$$H(X,Y) = -\sum_{x,y} P(X = x, Y = y) \log P(X = x, Y = y)$$

相互情報量

ここで，XとYが互いに独立でない場合，H(X, Y)とH(X)+H(Y)は一致せず，後者の方が大きくなります．両者の差が相互情報量であり，以下で定義されます．

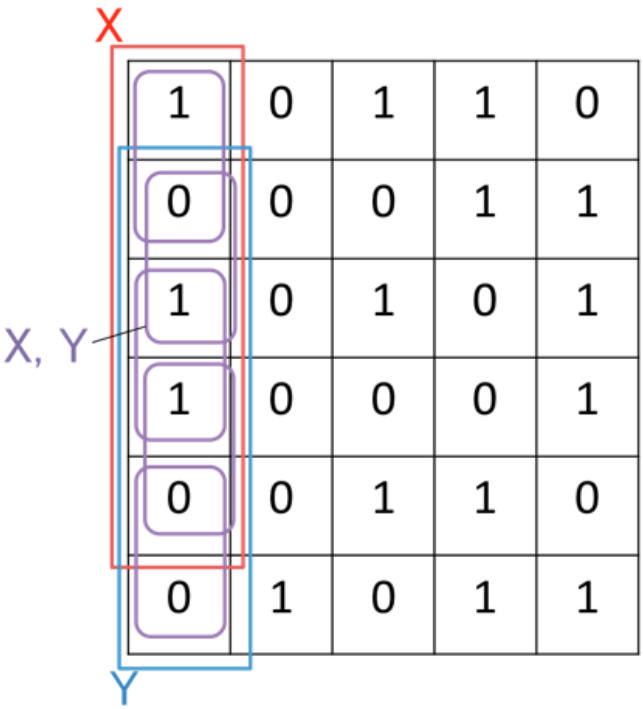
$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

相互情報量は常に非負の値をとります．

I(X; Y)= H(X)-H(HIY)
（H(HIY)は条件付エントロピー）
などと書き方はいくつかありますが，今回は上の定義を計算で使うことにします．

ラングトンの測った相互情報量

ラングトンは，あるセルの時刻tの状態と，そのセルの時刻t+1の状態との相互情報量を測って上の図を計算しました．つまり，あるセルの状態を見て，次の時刻のセ ルの状態がどれだけわかるかという意味での情報量を測った訳です．



$H(X)=-2/5\log(2/5)-3/5\log(3/5)=0.9709\cdots$
$H(Y)=-3/5\log(3/5)-2/5\log(2/5)=0.9709\cdots$
$H(X, Y)=-1/5\log(1/5)-1/5\log(1/5)-2/5\log(2/5)-1/5\log(1/5)=1.9219\cdots$
$I(X;Y)=H(X)+H(Y)-H(X, Y)= 0.0199\cdots$

ただし上のlogの底は2.

pythonで相互情報量を計算する

上のプログラムでca_1d関数を用いて作成した2次元配列データから，上のラングトンが用いた方法で各セルの相互情報量を測り，その全セルでの平均を CA全体の相互情報量と見なして返す関数をつくってみましょう．ただし，logの底は2とし，np.log(x, 2)関数を使うことにしましょう．

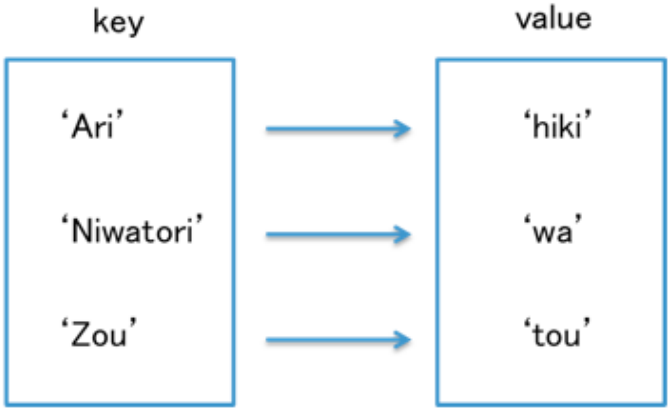
今回は，次の順序で行うことを考えましょう．

1. 一次元データの系列をリストで受け取り，そのエントロピーを返す関数calcEntropy(data)を作成
2. 同じ長さの一次元データの時系列をリストで2種類受け取り，その結合エントロピーを返す関数calcJointEntropy(x, y)を作成
3. １と２を使って，同じ長さのデータの時系列をリストで2種類受け取り，その相互情報量を返す関数calcMI(x, y)を作成
4. CAの実験データを受け取り，3を使って各セルの相互情報量を算出し，その値を並べたリストを返す関数calcCAMIList(data)を作成
5. CAの実験データを受け取り，４の関数を使ってすべてのセルの相互情報量の平均を返す関数calcCAMI(data)を作成

辞書

ここでは，pythonのリストに続くもう一つの重要なデータ表現である，”辞書”を使ってつくってみましょう．pythonの辞書とは，いわゆるハッシュテーブ ルのことで，ある文字列や数値に対してオブジェクトを割り当てる対応表を記述するものです．
例えば，次の文は次のような辞書dicをつくります．

```
>>>dic= {'Ari':'hiki', 'Niwatori':'wa', 'Zou':'tou'}
```



ここで，参照元はキー（鍵），参照先は値と一般に呼ばれます．
キー'Ari'に対応する値を見るには，

```
>>>dic['Ari']
'hiki'
```

と，かぎ括弧の中に直接キーを書きます．
同様な表記であたりの割り当てもできます．

```
>>>dic['Usagi']= dic['Niwatori']
```

あるキーが辞書に含まれるかどうかを調べるには，inを使います．

```
>>>if 'Usagi' in dic:
   print "Usagi ha " + dic['Usagi']
```

ある辞書に含まれるキーの一覧をリストで取得するには，その辞書の関数keys()を使います．同様に，値の一覧はvalues()，キーと値の組み合わせの一覧 はitems()で取得可能．

```
>>> dic.keys()
['Ari', 'Usagi', 'Zou', 'Niwatori']

>>> dic.values()
['hiki', 'wa', 'tou', 'wa']

>>> dic.items()
[('Ari', 'hiki'), ('Usagi', 'wa'), ('Zou', 'tou'), ('Niwatori', 'wa')]
```

例えば，上の内容のdicを使って

```
Ari ha hiki to kazoemasu.
Zou ha tou to kazoemasu.
Niwatori ha wa to kazoemasu.
```

を表示するには次の様に書けば良いわけです.

```
<br>
for i in dic.keys():
    print str(i)+ " ha " +dic[i] +str(" to kazoemasu.")
```

calcEntropy(data)

この辞書を使って記述したcalcEntropy(data)の例を以下に示します.

```
import numpy as np

def calcEntropy(data):
    dic= {}
    for d in data:
        if d in dic:
            dic[d]= dic[d]+1
        else:
            dic[d]= 1
    probdist= np.array(dic.values())/(float)(len(data))
    return(np.sum([-p * np.log2(p) for p in probdist]))
```

If文がようやく登場しました（python文法ページ参照）.
ところで，このifは，単に条件付で代入をしているだけなので，三項演算子をつかうと,

dic[d]= dic[d]+1 if d in dic else 1

と，簡単に書けて便利です．三項演算子とは，条件付で変数に値を割り当てる場合に用いる簡便な記述法で，pythonの場合,

割り当て変数 = 条件を満たした時の代入値 if 条件 else 条件を満たさない場合の代入値

と書きます.

もうちょっと詳しく：
例えば，[0, 1, 0, 1, 1]というリストの平均情報量を出すことを考える.
forループでは，0の出現回数と1の出現回数を辞書dicに保存している．この場合，{0:2, 1:3}.
probdistの式の中のdic.values()はその中の値の部分だけのリスト，つまり[2, 3]を表している.
これをnumpyの配列形式に変換したのが，np.array(dic.values()).これに対しては各要素への演算が簡単にできて，np.array(dic.values())/(float)(len(data))とすると，各要素をリストdataの大きさで割り算で きる.
つまり，この場合は各要素を5で割って，確率分布のリスト[2/5, 3/5]を出している.
これに対して，[-p * np.log2(p) ....]でリストの各要素の確率に関して情報量を計算した値のリストをつくっている.
最後に，np.sum(...)でリストの要素の総和をとって，平均情報量を最終的に出している.

練習 1

１）残りの関数をつくりましょう．

ヒント１）結合エントロピーを計算する際は，事象XとYの組み合わせを，タプルと呼ばれるオブジェクトの順序つき組み合わせを用いて表現すると都合がよいです （pythonの文法のページにに補足あり）．例 えば，事象(X=1，Y=0)を表すなら，"1"と"0"の組み合わせなので，pythonではタプルは括弧とカンマで以下の様に書きます．

(1, 0)

これをキーにしてエントロピーの計算を行います．

他にも，単純な方法として，XとYの値を文字列にして結合し，キーにすることが考えられます．例えば，X=1，Y=0なら，"1:0"とするとか．

ヒント２）CAの2次元データが入っている変数をdata(時刻iのj番目のセルの値がdata[i][j]) としたとき，セルiのみの時系列データを data_iとしてとるには，

data_i= [data[i][j] for i in range(len(data))]

とします．

２）ECAを表示するプログラムと１でつくった関数を合わせて，タイトルにCAの相互情報量を表示するように変更したグラフを作成しましょう．

３）２のグラフに加え，横軸にセル番号，縦軸に各セルの相互情報量を表したグラフを同時にもう一つ出力するように，２のプログラムを改変（追加）しましょう．

４）確認のため，ルール９０を使って真ん中にのみ状態１の条件で実験したときの３の結果を出してみましょう．他のルールや初期状態を使うとどうなるでしょうか．

練習２：N状態K近傍のCAへの拡張

次に，これまで用いてきた２状態３近傍のCAのソースを，N状態K近傍のCAに改変してみましょう．

練習３：λパラメータと相互情報量の関係を表した図を描く

いよいよ，最後に，λパラメータと相互情報量の関係を表した図を描きます．これは，最初のトピックで描いた分岐図，つまり，横軸をパラメータ，縦軸をそのパラメータで実験したときの系の変数の行き先にした図を書くのとほぼ同様です．

この場合は，横軸にルールを生成するのに使用したλパラメータ，縦軸にそのルールで実験したときのCAの相互情報量としてプロットした図をかけば良い訳です．

ただし，ルールの表現は，ECAのように一意の番号を割り当てたものを直接指定するのではなく，λの期待値を指定することにしましょう．つまり，ルールを記述する長さN^Kの列をつくるとき，各値を確率1-λで死状態=0，確率λで死状態以外の状態からランダムに選ぶということです．

具体的な作業としては，個体群動態で分岐図を書いたときのように，λの値を少しずつ変えてルールを生成し実験したときの相互情報量を計算し，横軸を生成に使ったλ，縦軸をそのときの相互情報量としてプロットします．