複雑系プログラミング特論
エージェントベースモデル

概要

前回は，セルオートマトンの挙動を相互情報量を用いて定量化し，それとλパラメータとの関係について可視化することを行いました． 先に述べたように，セルオートマトンは，各セルを一つの主体と見なしたり，ある種の物理的空間と見なしたりすることで，様々な要素間相互作用のモデルとして活用す ることができます．しかし．例えば相互に影響し合う要素が限られていたり，その構造が規則的で静的であるなど，実世界における主体間の相互作用を 一般的に記述するには制約が大きい場合があります．

エージェントベースモデリング（Agent-Based Model, ABM）とは，対象とする系の挙動を，系を構成する主体群とそれらの間の相互作用のルールから記述するモデル化手法です．ルールを手続き的に記述するため，多様な 個体間の相互作用構造や相互作用の仕方を自由に記述できます．

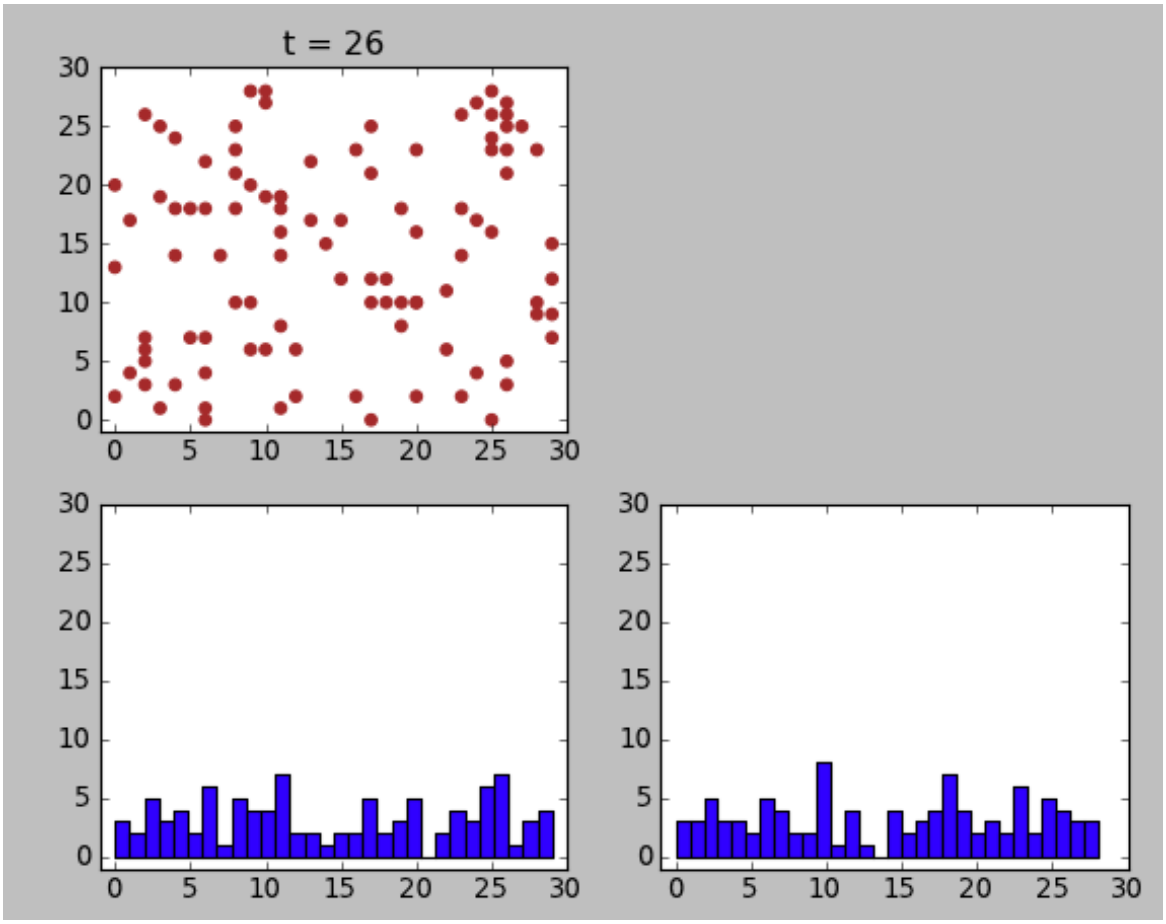
この授業の初回で紹介したイベント会場での混雑情報提供シミュレーションはこの手法を用いたもののひとつです．

これから数回は，このエージェントベースモデリングを用いて，簡単な社会的相互作用を扱ったモデルを構築します．特に，今回はpythonの持つオブジェクト指向 的な側面を，エージェントモデルの記述に活用してみます．

お決まりな雛形モデル

では，pythonでABMをつくってみましょう．スタートアップのために次のようなごくごく簡単なモデルを考えます．

W×Wの二次元トーラス格子状空間を考えます．
そこに，N体のエージェントを配置します．
各ステップにおいて，各エージェントはランダムな順番で，今いる場所を含む自分の周囲9近傍のいずれかに移動します．



これまでと比べるとやや長いサンプルプログラムを用意しました.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import random as rnd

# set parameters
N= 100
T= 100
W= 30
SEED=101
agents= []

## define functions
def clip(x):
    if x<0:
        return(x+W)
    elif x>=W:
        return(x-W)
    else:
        return(x)


## define classes
class Agent(object):

    def __init__(self):
        self.x= rnd.randint(0, W-1)
        self.y= rnd.randint(0, W-1)
```

```
29
30      def randomwalk(self):
31          self.x+= rnd.randint(-1, 1)
32          self.y+= rnd.randint(-1, 1)
33          self.x= clip(self.x)
34          self.y= clip(self.y)
35
36
37  # initialize variables
38  rnd.seed(SEED)
39  agents=[Agent() for i in range(N)]
40  fig = plt.figure()
41
42  # main loop (call back function for animation)
43  def main_loop(t):
44      step()
45      update(t)
46
47  # events in a step
48  def step():
49      rnd.shuffle(agents)
50      for a in agents:
51          a.randomwalk()
52
53  # update function for graph
54  def update(t):
55      fig.clear()
56      ax1= fig.add_subplot(2, 2, 1)
57      x= [a.x for a in agents]
58      y= [a.y for a in agents]
59      ax1.scatter(x, y, color='brown')
60      ax1.axis([-1, W, -1, W])
61      ax1.set_title('t = ' + str(t))
62
63      ax2= fig.add_subplot(2, 2, 3)
64      ax2.hist(x, W)
65      ax2.axis([-1, W, -1, W])
66
67      ax3= fig.add_subplot(2, 2, 4)
68      ax3.hist(y, W)
69      ax3.axis([-1, W, -1, W])
70
71
72  ani = animation.FuncAnimation(fig, main_loop, np.arange(0, T), interval
73  plt.show()
```

pythonでクラス定義

いくつか説明が必要な事項がありますが，一番大きな点はクラスを定義し利用していることです．
クラスとは，我々がこれまで使ってきた様々なオブジェクトの設計図のことです．設計図には，それぞれのクラスのオブジェクトがどんな特性を持っているか（変数），それを使ってどんな仕事ができるか（関 数）が定義されています．

この，

”オブジェクトの特性を表す設計図をつくり，オブジェクト間のやりとりでプログラムを記述する”

という方法は，これから我々が行う，

”複数の要素の特徴とそれらの間のルールを定義して動かす”

というエージェントベースモデルの手法と，とてもよくあいます．

今回は，エージェントを表すシンプルなAgentクラスを作成しています．Javaの経験がある人はクラスを自分で定義し使ったことがあると思いますが，定義の仕 方が 異なる面があります．

まず，上のクラス定義の部分を抜き出します．

```
## define classes
class Agent(object):

    def __init__(self):
        self.x= rnd.randint(0, W-1)
        self.y= rnd.randint(0, W-1)

    def randomwalk(self):
        self.x+= rnd.randint(-1, 1)
        self.y+= rnd.randint(-1, 1)
        self.x= clip(self.x)
        self.y= clip(self.y)
```

1行目は"Agent"という名前のクラス定義の宣言です．続く括弧内は定義するクラスが継承するクラス名を書きます．今回は，objectという全クラスの大元 になるクラスを指定します．

続くインデント領域で，４つの関数（Javaで言うメソッド．クラスのオブジェクトが実行できる関数）を定義しています．最初はやや変な名前 で，”__init__”という名前のメソッドです．これはJavaで言うコ ンストラクタ（クラス名で始まる初期化用メソッド）に相当するもので，Agentオブジェクトが生成されたときに最初に呼び出され，オブジェクトの初期化に用いられます．

ここで注意すべきことは，メソッドの第一引数です．メソッドを定義する際には必ず，第一引数を変数で指定することになっていて，この変数がメソッド内でオブジェク ト自身を指すものになります．Javaで言え ば"this"に相当すると言ってわかる人はその通りです．

__init__関数では，このselfを使って，オブジェクトの持つ変数xとyの値を定義しています．ここで，いきなり，

self.x=...

などと書くことで，”Agentクラスのオブジェクトにはxという変数（インスタンス変数，Javaで言うフィールド）がある”と宣言し，値を代入しています．

その次のrandomwalk関数では，それぞれのx, yに-1, 0, +1の乱数を足し，事前に定義されているclip関数でトーラス平面の端の処理を行って移動を表現しています．この場合も，第一引数をself

とし，関数の中では，オブ ジェクト自身のxはそれを明示するために，self.xと明記します．

こうして定義したAgentクラスのオブジェクトを生成し，実際に利用しているのが，以下の箇所です．

```
agents=[Agent() for i in range(N)]
```

```
# events in a step
def step():
    rnd.shuffle(agents)
    for a in agents:
        a.randomwalk()
```

一番最初で，N個のAgentクラスのオブジェクトを生成し並べたリストagentsを作成しています．このとき，注意すべきは，Agent()のよう に，引数をとっていないことです．この引数は__init__関数に対応し，本来__init__関数には第一引数としてselfが入っているはずなのですが，そ こは無視して引数無しで利用します．

同様に次の各ステップでのイベントを記述したstep関数においても，a.randomwalk()のように第一引数無しで呼び出しています．やや違和感がありますが，慣れましょう．なお，関数に第2以 降の引数がある場合は，それを指定します．

アニメーション

ちょっとややこしいのが，ところどころに散らばっているアニメーションの処理に関する記述です．
matplotlibにはanimationというモジュールがあって，今回はこれを使います．

ごく簡単に言えば，まず，main_loopと名付けた関数を定義して，その中で，モデルとして1ステップで行うべき命令を書いたstep名付けた関数と，その結果をもとにグラフを書き直すupdateという関数を実行しています．

pythonはmain_loop関数を各コマ毎に実行し，一定数繰り返すと停止します．

ani = animation.FuncAnimation(fig, main_loop, np.arange(0, T), interval=25, repeat=False)

には，描画がfigで示されたキャンバスで行われること，main_loopが繰り返し実行されること，0からT-1ステップまで実行されること，コマを進める時間のインターバルが25であることが示されています．

条件分岐の並列表記

最後になりましたが，冒頭のclip関数は，x座標，y座標がWで定められた範囲を超えるとその反対側から出てくるトーラスの条件を満たすための関数です．

```
## define functions
def clip(x):
    if x<0:
        return(x+W)
    elif x>=W:
        return(x-W)
    else:
        return(x)
```

ここで，elifはいわゆる"else if"のことで，これを使うと，不要なインデントをせずに並列に条件分岐が書けます．

これから，このプログラムを改変してエージェントベースモデルをつくってみます．

シェリングの分居モデル

と聞いて「あぁあれか」と思う人はそんなにはいないかと思います．でもトマス・シェリングは，2005年のノーベル経済学賞の受賞者で，いわゆるゲームの理論を用 いて社会における対立や協調の理解に貢献した有名人です．
実は，彼は社会科学においてエージェントベースモデルを初めて（たぶん）用いた人でもあります．1960年代に彼がペニーとダイム，チェス盤とサイコロを使って手 作業で行った，差別意識に関するシミュレーションが，”シェリングの分居モデル”と呼ばれるものです．
今回は，この最初のエージェントベース社会モデルを，前回のランダムウォークモデルを使ってつくってみます．

彼は，人種差別など，実社会において様々な場面やレベルで生じるすみ分け，つまり，同じ種類のもの同士が集まる傾向がどのように生じるかを理解するために，次のよ うな状況を考えました．
なお，プログラムの簡単な実装のために，エージェントの動き方は適宜改変していますが，それでも系の本質的な振る舞いはオリジナルとほぼ同様と考えて下さい．

W×Wの二次元トーラス格子状空間を考えます．
そこに，N体のエージェントをランダムな場所に配置します．ただし，エージェントには2種類あり，初期状態で各エージェントの種類は半数（N/2）ずつとします． また，初期状態において，エージェントは同じ場所に2体以上存在しない（重ならない）ものとします．

Tステップにわたって，各エージェントはランダムな順番で，次の行動を起こします．
（満足度sの計算）
　自分の周囲8近傍に他に誰もいないなら：
　　満足度sは0
　それ以外：
　　自分の周囲8近傍に存在する他のエージェントのうち，自分と同じ種類のエージェントの割合を満足度sとする．

（満足度に基づく移動）
　満足度sがパラメータ*TH*（ただし0より大1以下）より小さいなら：
　　現在地から9近傍のいずれかひとつをランダムに選び移動する．これを空き地に至るまで繰り返す．
　それ以外：
　　そのまま動かない

満足度sは高いほど近所に仲間が多いことをあらわします．パラメータPは各エージェントが等しく持つ満足度に対する許容閾値です．つまり，現在のsがTHより小さいと現状に不満で隣のどこかに引っ越し，そうでなければ現状をよしとして留まるということです．

シェリングは，このルールに従ってエージェントをTステップ動かしたときの各ステップの満足度の平均の推移を調べました．つまり，パラメータTHで決まる個々のエージェントの局所的な意志決定が社会全体の状態にどのような影響を与えるかを調べたということです．

練習１

以下のガイドラインに沿って，ランダムウォークモデルを拡張してシェリングモデルを作成してみましょう．

１：追加のパラメータの定義
全てのエージェントが参照する閾値THを変数としてプログラムの冒頭で定義します．また，プログラム全体でよく使う変数agentsを”agents= []”として，前もって定義しておき，クラス定義中で使えるようにします（多分必要．実際の中身はあとから初期化）．

２：クラス定義の拡張
初期化関数__init__関数を改変して，次のことを行います．

各エージェントの現在の満足度を保存する変数sをAgentクラスの定義に追加します．初期値を0とします．
各エージェントの種類を保存する変数pをAgentクラスの定義に追加します．このとき，初期化の際に用いる引数をもう一つ増やし，これをspとして，その値をp に割り当てることにします．

次に，以下の4つの関数をAgentクラスの定義に追加します．

isOverlapped(self)
自分自身のいる場所と同じ場所にいる他のエージェントが一体でもいればTrue，いなければFalseを返す関数．

findNewSpace(self)
周囲9近傍からランダムに選んだ場所に移動することを空き地に至るまで繰り返す関数．
randomWalk関数とisOverlapped関数をうまく用いる．

updateSatisfaction(self)

近傍の個体数と，そのうち同じ種の個体数を数えて現在の満足度を計算しエージェントの変数sに代入する関数．ただし近傍の個体数が0の場合はsは0とする．

seek(self)
満足度を更新し，それがTHより小さい場合には9近傍のうちランダムな場所に移動することを空き地に至るまで繰り返す．

３：エージェント集団の初期化とメインループの改変
エージェント群を入れるリストの変数agentsを初期化する際，２で定義したエージェントの種類の指定をAgentオブジェクトの生成に用いて，リストにおいて 偶数番目のエージェントのpを0，奇数番目を1として，0と1を半数ずつ定義しましょう．

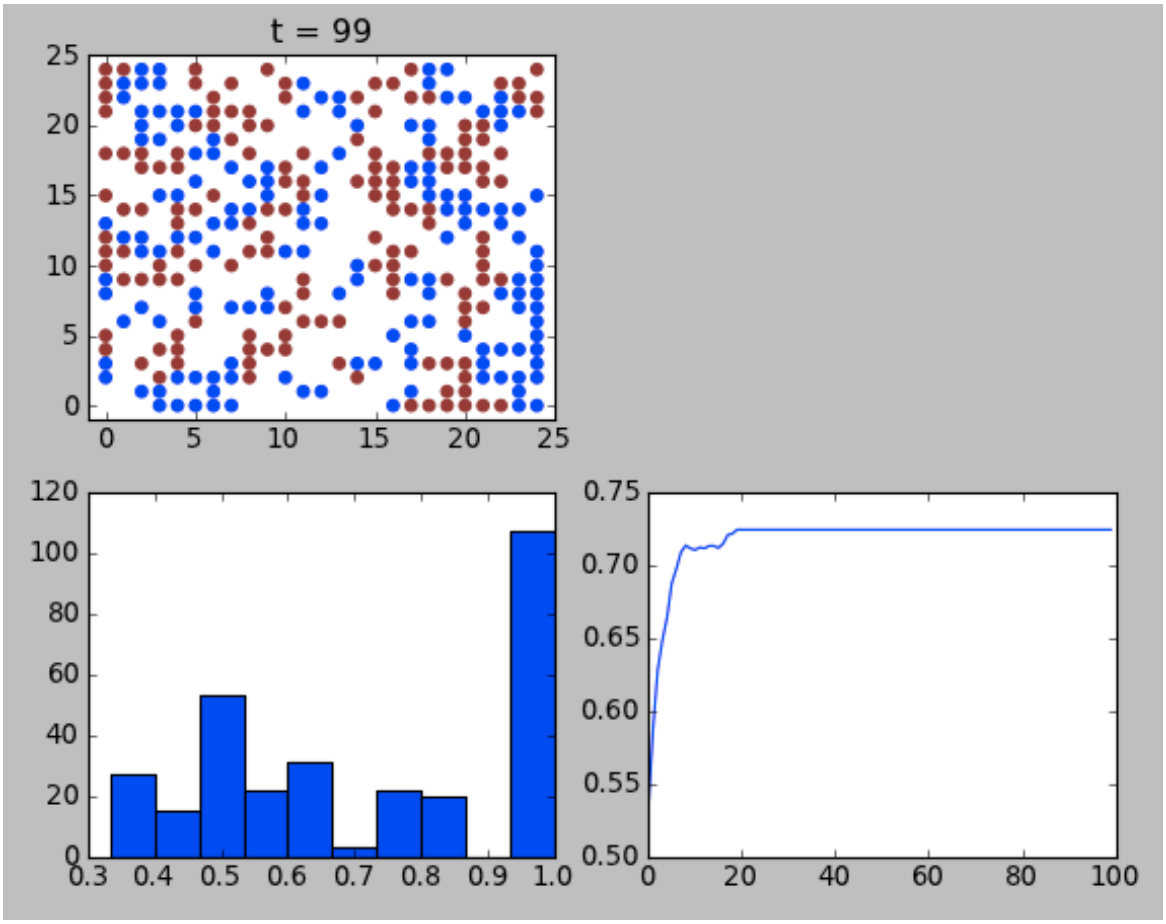
また，初期状態においてエージェント同士が重ならないように，各エージェントに対してfindNewSpace関数を一度ずつ実行させましょう．

以上のパラメータ，クラス定義を用いて，分居モデルを実行するメインループを作成しましょう．

４：グラフ表示の改変
エージェントの分布図，各ステップでの満足度の平均の推移，各ステップでの各エージェントの満足度のヒストグラムを表示するようにします．
満足度の平均については，これまでのステップまでの値を保存するリストを用意して，それに毎ステップ平均値を追加してグラフ化します．

作業２）横軸にパラメータTH，縦軸にその設定で実験した場合の最終ステップでの満足度の平均をとって，パラメータが系に与える影響のグラフを書きましょう．
シェリングは，実験結果から，”社会における各個人の排他的な意識がそれほど強くなくても，社会全体として強いすみ分けが生じうる”ことを示しました．
描いたグラフのどこがこれを示唆しているか，考えましょう．

（３の実行例）



TH=0.3の場合の系の様子

SIRモデル

SIRモデルとは，感染症が集団内で伝播して行く様をシミュレートしたものです．
この授業では次の様にエージェントベースモデルで記述します．

エージェントはS，I，Rの3種類の状態を持つ．

S：病原菌に感染していない．
I：病原菌に感染して現在発症している．
R：感染から回復．免疫があるので再度感染しない．

各ステップで，各エージェントはランダムな順番で選ばれ，次の動作を順に行う．

移動）現在地から9近傍のいずれかひとつをランダムに選び移動する．これを空き地に至るまで繰り返す．

感染）もし自分の状態がIである場合，そのエージェントは周囲8近傍に存在する未感染のエージェント（S）それぞれについて，確率PIで感染させ，状態Iにする．

回復）もし自分の状態がIである場合，そのエージェントは確率PRで回復し，状態をRにする．

練習２

１）これまでのエージェントベースモデルを改変して，SIRモデルを構築しなさい.

２）一体のエージェントが発症し（状態I），その他は未感染である状態（状態S）を初期状態としたとき，１）のモデルの典型的な挙動を示すグラフ（各状態にある エージェント数の推移）を描きなさい.

３）１）のモデルにひとつ新たな要素を加え，その影響を調整するパラメータを設定しなさい. そのパラメータ設定を変えて実験したときの系の挙動の違いを端的に表す グラフを作成し，その理由について検討しなさい.

例：

・感染後発症するまでの潜伏期間を導入する（SEIRモデル）.
・発症中のエージェントは体調が悪いのでフィールド中を動きにくくなる.
・発症中に一定確率で死亡する.
・獲得した免疫が一定確率で無くなってしまう.

なお，SIRモデルの拡張についてはインターネット上に多くの情報があるので，それらを適宜参照してアイデアを得て，構築してよい.
（その場合は特に参考にした情報の出典を付記すること．）