12. CA 伝染病モデル

2018/1/15

1　はじめに

2 次元のセルオートマトンの応用として、伝染病モデルを扱います。

空間を 2 次元正方格子で表し、各セルには、一つの個体を置きます。ただし、空のセルも可能とします。システムの端は周期境界条件に従うとします。2 次元の周期境界ですから、左右に閉じていると同時に上下にも閉じています。ドーナツのような形ですが、数学ではトーラス (torus) と呼びます。

個体の状態は $s \in \{\mathrm{S}, \mathrm{I}, \mathrm{R}\}$ のいずれかであるとします (表 1)。状態 S の個体は、隣接するセルの一つでも状態 I の個体が居る場合に、確率 β で病気に罹り、状態 I になります。周囲の状態 I の個体数によって感染確率が変わらないことに注意します。また、状態 I の個体は、周囲のセルの状態にかかわらず、確率 γ で病気が治り、状態 R となります。状態 R となった個体は、免疫をもった状態ですから、それ以上、状態は変化しません。個体の状態は同期的に更新するものとします。

表 1　伝染病モデルにおける状態

S	健康な状態。隣接する状態 I の個体から病気をうつされる。
I	病気の状態。隣接する状態 S の個体に病気をうつす。
R	治癒した状態。免疫を得て、病気に罹らない。

システムのサイズを $n \times n$ とし、各時刻での状態 S、I、R の個体数を $S(t)$、$I(t)$、$R(t)$ とし、総個数 $N = S(t) + I(t) + R(t)$ は不変とします。時刻 $t = 0$ ででたらめに選んだセルに $I(0)$ 個の病気の個体と $S(0)$ 個の健康な個体をばらまき、シミュレーションを開始します。

シミュレーションの準備として、プロジェクト Epidemic を作成し、以下の URL から Java ソースファイルを取得し、プロジェクト Epidemic のソースファイルの下に置きます。また、ライブラリ MyLib も設定しましょう。

http://http://aoba.cc.saga-u.ac.jp/lecture/ModelingAndSimulation/javasrc/Epidemic/src.zip

2 個体のクラス Individual

個体の状態を表すクラス Individual を定義します。個体の状態は、このクラスの中の列挙型として定義します (ソースファイル **??**)。列挙型とすることで、変数が他の値をとることを防ぐことができます。また、static public として定義することで、クラスのインスタンスを定義しなくても、クラスの外から状態の種類として使用することが可能となります。コンストラクタでは、その個体の初期状態を与えます (表 2)。

```
1    //個体の状態
2    static public enum State {
3        S, I, R;
4    }
5    private State state;//現在の状態
6    private State nextState;//次の状態
```

表 2 Individual クラスのコンストラクタ

コンストラクタと説明
Individual(State state) 初期の状態 state を与えて初期化する。

表 3 Individual クラスのメソッド

修飾子と型	メソッドと説明
State	getNextState(List<Individual> neighbours, double beta, double gamma, double r) 隣接個体のリスト neighbours、感染確率 beta、治癒確率 gamma、及び $[0, 1)$ の乱数 r を与えて、次の状態 nextState を求める。状態は更新しない。
State	getState() 現在の状態を取得する。
State	update() 次の状態 nextState を state に上書きすることで、状態更新を行う。

各個体の次の時刻における状態を求めるためには、周囲の個体の状態が必要です。周囲の個体のリスト neighbours を与えることで、次の時刻における状態を求めるメソッドが getNextState() です (表 3)。このメソッドでは、内部で乱数を生成して用いるという方法ではなく、外から変数として乱数の値 r を与える形をとっています。こうすることで、このメソッドの応答を一意にすることが可能となり、単体テストが容易に実行できます。非決定的な動作の部分は、このメソッドを呼ぶ際に与えられる乱数の値 r だけになります。なお、このメソッドは、次の時刻における状態を求めるだけであり、その個体の状態を更新していないことに注意します。求めた次の時刻における状態へと、個体の状態の更新を行うのは update() メソッドです。

課題 1　メソッド getNextState() が、講義で説明した個体の状態変化の規則となるように、完成させなさい。

3　伝染病モデルのクラス EpidemicDynamics

クラス EpidemicDynamics は、CA 伝染病モデルを表すクラスです。システムの大きさ、個体数、初期の感染個体数、感染確率、治癒確率を与えて、初期化します (表 4)。

表 4　EpidemicDynamics クラスのコンストラクタ

コンストラクタと説明
`EpidemicDynamics(int n, int nPop, int initI, double beta, double gamma)` システムサイズ n (セル数は n×n)、個体数 nPop、初期の状態 I の個体数 initI、感染確率 beta、治癒確率 gamma を与えて初期化する。

3.1　2 次元格子の表現

このモデルでは、空間を二次元の正方格子で表します。二次元の座標を (x, y) で表すことにします。ここで、$0 \leq x, y < n$ とします。クラス EpidemicDynamics では、個体の配置は Individual クラスの一次元配列 individuals で表現します。ここでは、二次元の情報を一次元で表す方法を再確認しましょう。

一辺の長さ n の二次元格子は、n^2 個の格子で構成されます。各格子に番号 $0 \leq r < n^2$

を付けて、座標 (x, y) と対応付けます。最も簡単な方法の一つは

$$r = y \times n + x \tag{3.1}$$

と対応つける方法です。これは、x 座標を n 進数の一桁目、y を二桁目として、r を表したものと同等です。図 1 は $n = 8$ の例です。

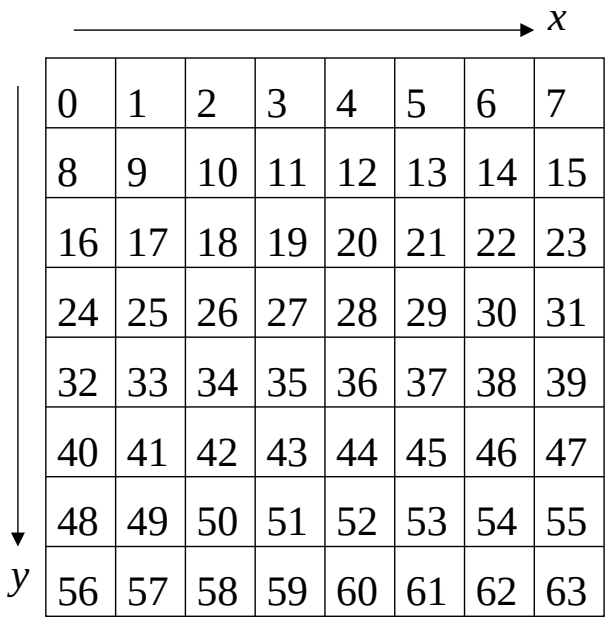

図 1　$n = 8$ の場合の二次元格子に番号を付ける例

課題 2　$n = 8$ の場合について、$(x, y) = (2, 4)$ 及び $(5, 0)$ が正しい位置にあることを確認しなさい。

3.2　初期配置の生成

コンストラクタを呼んだあと、メソッド `initialize()` を呼ぶことで、個体の初期配置を行います。その中で用いるランダムな初期配置を行うために、メソッド `initialize()` では、以下のような手順を行っています。

まず、セルの総数が n^2 個であることに注意し、0 から $n^2 - 1$ の整数をでたらめに並べ替えた配列 `randomIntegers` を得ます。この配列の先頭から整数を取り出すと、0 から $n^2 - 1$ の整数がでたらめな順番で一度だけ現れることに注意します。次に、メソッド

表 5　EpidemicDynamics クラスのメソッド

修飾子と型	メソッドと説明
MacroState	countMacroState() 各状態の個体数の組 (MacroState クラス) として返す。
static Individual.State[]	createRandomState(int randomIntegers[], int initI, int nPop, int nSite) 0 から nSite-1 個の整数をでたらめに並べた配列 randomIntegers を与え、個体数 nPop、初期の状態 I の個体数 initI をでたらめに並べた状態の配列を返す。
double	getBeta() beta の値を返す。
double	getGamma() Gamma の値を返す。
int	getInitI() 初期の状態 I の個体数を返す。
int	getN() セル数を返す。
int	getnPop() 個体数を返す。
List<MacroState>	getSequence() 個体数の組 (MacroState クラス) の変化の列を返す。
Individual.State	getState(int x, int y) 指定した位置の固体の状態を返す。
void	initialize() 個体の状態を初期化する。
int	update() 状態を更新する。状態 I の個体数を返す。

setRandomState() の中から、メソッド createRandomState() を呼びます。この時、この配列 randomIntegers を使って、状態のランダムな列を生成し、セルに対応した状態を持つ個体を生成します。

課題 3　メソッド createRandomState() では、以下のようにランダムな状態の列を生成します。

1. 大きさ nSite(n^2) の状態の配列を作る。
2. 配列 randomIntegers の先頭から initI 個の値で指定される要素には、状態 I を

置く。

3. 配列 `randomIntegers` のその後ろの `nPop` − `initI` 個の値、つまりインデクスが `initI` から nPop − 1 の値で指定される要素には、状態 S を置く。

相当部分を記述し、完成させなさい。

3.3 状態更新

系全体の状態更新を行うのが `update()` メソッドです。各位置 `(x,y)` のセルに対して、隣接するセルの個体一覧 `neighbours` を得て、個体の次の状態を `Individual` クラスの `getNextState()` メソッドを用いて得ます。全てのセルの個体に対して、次の時刻の状態が得られたら、クラス `Individual` のメソッド `update()` で、状態を更新します。

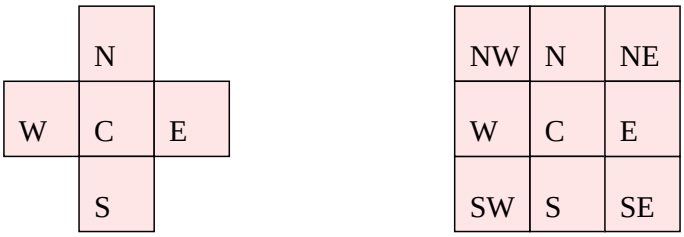

図 2　Neumann 近傍と Moore 近傍

ここで問題となるのは、各位置 `(x,y)` のセルに対して、隣接するセルの個体一覧 `neighbours` を得るところです。二次元正方格子において隣接格子の選び方には、Neumann 近傍と Moore 近傍の二つの選択肢があります (図 2)。ここでは Neumann 近傍を選択しましょう。

さらに、周期境界であることにも注意が必要です。y 軸方向では、$y = 0$ の格子点の上には $y = n - 1$ の格子点、$y = n - 1$ の下には $y = 0$ の格子点があります。x 軸方向についても同様です。このような操作を条件分岐等によって境界部分だけに行う方法ではなく、境界でない部分にも一貫した方法で行うようにしましょう。y 軸方向であれば、以下のように行うことができます。y の上の座標を y_{north}、下の座標を y_{south} とします。すると以下のように表すことができます。

$$y_{\mathrm{north}} = (y - 1 + n) \bmod n,\ y_{\mathrm{south}} = (y + 1) \bmod n \tag{3.2}$$

同様に x 方向に対して、以下のように表すことができます。

$$x_{\mathrm{west}} = (x - 1 + n) \bmod n,\ x_{\mathrm{east}} = (x + 1) \bmod n \tag{3.3}$$

課題 4　周期境界において、`Individual` クラスの一次元配列 `individuals` のインデクスとの対応を考えなさい。ある座標 (x, y) に対する上 r_{north}、下 r_{south}、左 r_{west}、及び右 r_{east} のインデクスを表現しなさい。

4　単体テスト

今回も単体テストを実行し、作成したメソッドが正しく動作することを確かめましょう。今回は、クラス `Individual` 中のメソッドの単体テストを実行します。

今回は、`getNextState()` だけをテストしますから、以下の URL からダウンロードしたものと置き換えてください。

`http://http://aoba.cc.saga-u.ac.jp/lecture/ModelingAndSimulation/javasrc/Epidemic/test.zip`

メソッド `getNextState()` をテストするのが `testGetNextState1()` ほか 4 つです。パラメタ β と γ は共通に設定します。各テストでは、周囲の個体の状態を `List<Individual> neighbours` で与えます。乱数で与える `r` を適切に与えて、メソッド `getNextState()` を実行します。結果 `result` と、予想される結果 `expResult` を `assertEquals()` で比較します。4 つのテストは、中心にある個体の状態が周囲の配置、`r` によって正しく変更されることを確認します。

$$\begin{array}{ccc} \mathrm{S} & \to & \mathrm{S} \\ \mathrm{S} & \to & \mathrm{I} \\ \mathrm{I} & \to & \mathrm{I} \\ \mathrm{I} & \to & \mathrm{R} \end{array}$$

課題 5　配布ファイルでは、乱数に対応する変数 `r` は全てゼロに、様相される結果もすべて R となっている。テストクラス `IndividualTest` に含まれる 4 つのテストの内容を正しく記述しなさい。

課題 6　テストクラス `IndividualTest` を実行すると、テスト専用のウィンドウが開き、テストが実行され、テスト結果専用のウィンドウに出力されます。これらのテストは成功しましたか？失敗した場合には、`getNextState()` を訂正しましょう。

5　シミュレーション

プロセッシングを使ってシミュレーションを実施しましょう。以下のファイルをダウンロードして、jar ファイルなどを適切に配置してください。

http://http://aoba.cc.saga-u.ac.jp/lecture/ModelingAndSimulation/javasrc/Epidemic/epidemic.pde

実行すると、指定したパラメタに対応したシミュレーションを実行します。各点は青が状態 S、赤が状態 I、水色が状態 R を表します。マウスボタンを押すと、実行が終了し、状態 I と状態 S の個体数の時間変化がデスクトップのファイル out.txt へ保存されます。

population を 0.7 とし

$$I_{\text{init}} = 0.1, \quad \beta = 0.1, \quad \gamma = 0.2$$

の時の動作を観測しましょう。状態 I と状態 S の個体数の時間変化を gnuplot を使って、時間変化を見ましょう。

課題 7　population を 0.7 とし

$$I_{\text{init}} = 0.1, \quad \beta = 0.1, \quad \gamma = 0.1$$

及び

$$I_{\text{init}} = 0.1, \quad \beta = 0.1, \quad \gamma = 0.3$$

の時の動作を観測しましょう。状態 I の個体数の時間変化を、gnuplot を使って観察しましょう。

ソースコード 5.1　状態 I の個体数の時間変化を示す gnuplot スクリプトの例

```
1 set terminal png enhanced 28
2 set xlabel "{/Italic␣t}"
3 set title "Epidemic␣Model"
4 set yrange [0:0.3]
5 set xrange [0:150]
6 set xtic 50
7 set output "epidemic-I.png"
8 set ytic 0.05
9 plot "Out.txt" using 1:2 with line lw 5 title "I"
```