ALARM LECTURE AND PRACTICE

System Analysis Course
Prepared by Dr. Oleksii Ignatenko

OBJECTIVES

Learn about specific spatial models

Try in practice established models and evaluate their properties

Use Netlogo to perform simulation and build multi-agent systems from scratch

Create a report with measured metrics and findings

WHAT TO DO

1. Follow this document. Read the model description and create an appropriate Netlogo model.
2. Read chapters in books with additional information
3. Do exercises and fill your report document with the images and plots required.

SPATIAL MODELS. AGGREGATION

The most salient feature of many spatial patterns is summarized in Waldo Tobler's 'first law of geography: everything is related to everything else, but near things are more related than distant things. Tobler's law arises in the context of simulating urban growth, from the necessity of simplifying the system to be modeled from one where 'everything is related to everything else, a situation that presents obvious difficulties when it comes to developing a simple model.

Most of the models in this lecture adopt the cellular grid structure that we have already encountered, which rests on ideas about how to represent time and space. Broadly, time is represented by cells determining, and iteratively updating to their next state, while space is represented as a grid of square cells. In many cases, the grid ‘wraps’ around from left to right and from top to bottom, so that cells in the extreme left-hand column have as neighbors those in the extreme right-hand column as well as their immediate neighbors. This simulates an infinite space and avoids model artifacts due to edge effects, and is called toroidal wrapping. Another important decision concerns the neighborhood structure used, that is, what is adjacent or connected to what? The most common neighborhood definitions are either the four orthogonal immediate neighbors (the von Neumann neighborhood) or the eight adjacent neighbors (orthogonal and diagonal, the Moore neighborhood). Also, we can define our unusual neighborhoods depending on variants.

At any given time, each cell has some state, generally, a number, often restricted to integer values when cell states are discrete. Change occurs iteratively, by updating cell states with transition rules governing how the current state and the state of neighbors in the grid lead to new cell states. The timing of state changes may occur either synchronously, when all cells determine their next state and are updated simultaneously, or asynchronously, when one cell at a time is updated. In the asynchronous case, it is useful to distinguish between ticks of the model clock and generations. During each tick, one cell may change state. A generation is a number of ticks equal to the number of grid cells so that during a generation, every cell may have an opportunity to update its state. However, due to the (usually) random selection of cells for the update, it is unlikely that every cell will change state during a generation. Under the synchronous update, ticks and generations can be considered equivalent.

1. AVERAGING MODEL

Create a new Netlogo file.

In the interface section:

1. Define setup and go buttons.
2. Make worlds size 50x50 patches (in total, you should have 2500 patches).

In the programming section, create empty procedures

to setup

```
end

to go

end
```

Exercise 1.

Create a new attribute **c** for each patch using

```
patches-own [ c ]
```

In the setup procedure, initialize every patch with c, and draw from a uniform distribution from [0, 1] interval.

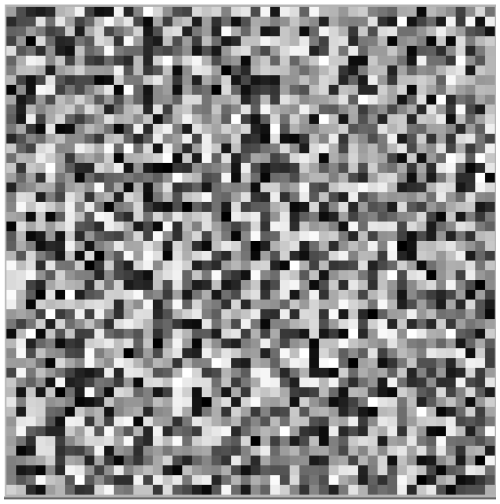
Implement rescale procedure

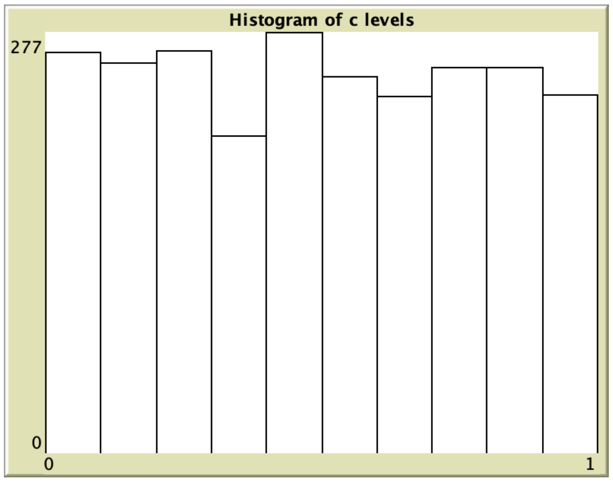
```
to rescale-c-values
  let max-c max [c] of patches
  let min-c min [c] of patches
  let range-c max-c - min-c
  ask patches [
    set c (c - min-c) / range-c
  ]
end
```

Read about scale-color primitive in Netlogo Dictionary. Create a procedure

```
to colour-patches
  ask patches [
    set pcolor scale-color black z 1 0
  ]
end
```

Create a histogram of c levels of patches. Read about histogram creation in the Netlogo dictionary. Try to recreate the following image.





Now we need to implement the go procedure. The most obvious way to make places similar to their neighbors is to repeatedly apply a local averaging procedure so that each location's current value moves towards the mean value of its neighbors. Local values that are high relative to neighboring values are dragged down by this averaging process, while values low relative to neighboring values are pulled up.

First, create a slider with a **w** variable. This will be the weight of averaging. Suppose we defined a neighborhood as a set of patches around.

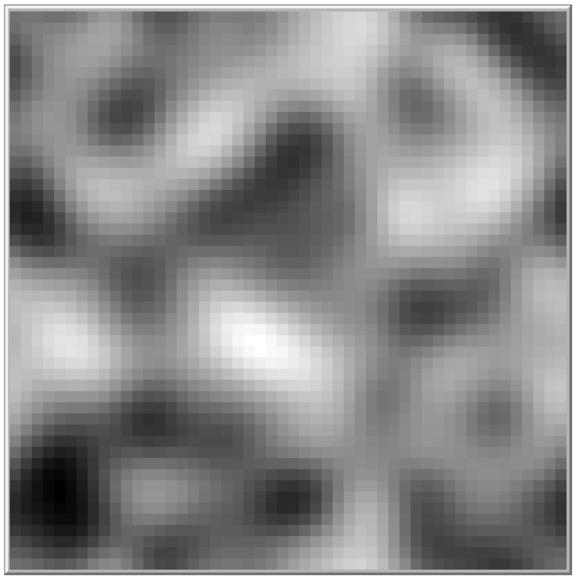
Let us implement Moore neighborhood first. The first approach is Netlogo-based and can not be generalized. Read about **diffuse** in the Netlogo dictionary. Use go as follows:

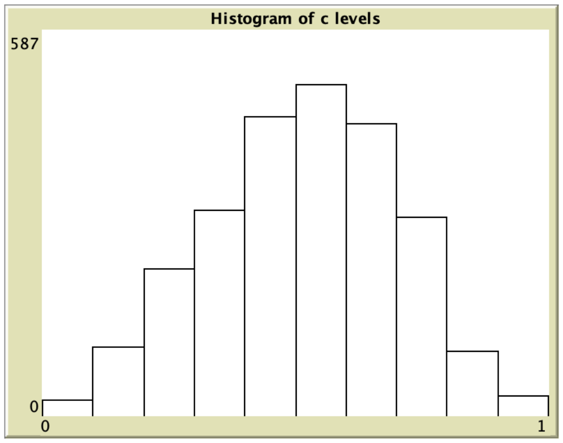
```
to go
  diffuse c weight-w
  rescale-c-values
  colour-patches
  tick
end
```

This is a fast implementation of the following formula:

$$c_i(t+1) = (1-w)c_i(t) + \frac{w}{N}\sum_{j\in N_i} c_j(t)$$

After 100 ticks, you should have the following image





How can you explain the distribution of c levels? How it will evolve with time? Can you estimate the final distribution? Write your answers in the report.

Now perform several simulations. Do you have another answer now?

Depending on your variant, implement custom neighborhood patches that take part in the change of the c level of the current patch.

Var 0.

X		X
	C	
X		X

Var 1.

		X
X	C	
	X	X

Var 2.

	C	X
	X	X

Var 3.

X	X	X
X	C	
X		X

Implement your variant and answer the following questions.

1. What is stable distribution after a long time? Estimate its type.
2. What is the influence of parameter w?
3. If we remove rescaling, how will it affect the model behavior?

2. AVERAGING MODEL WITH NOISE

It is straightforward to include a random element in the local averaging process, but its effects can be difficult to detect against the global smoothing dynamic. A model that simplifies the idea dramatically so that this aspect becomes clearer is presented by Vicsek and Szalay (1987). Their model is again grid-based but uses a von Neumann neighborhood. The initial state of the model assigns c = 0 to every grid cell.

$$c_i(t+1) = (1-w)c_i(t) + \frac{w}{N}\sum_{j \in N_i} c_j(t)$$

$$+\epsilon_i(t)$$

Where the last random part is noise, distributed as -1 or 1 with equal probabilities. We also introduce noise level.

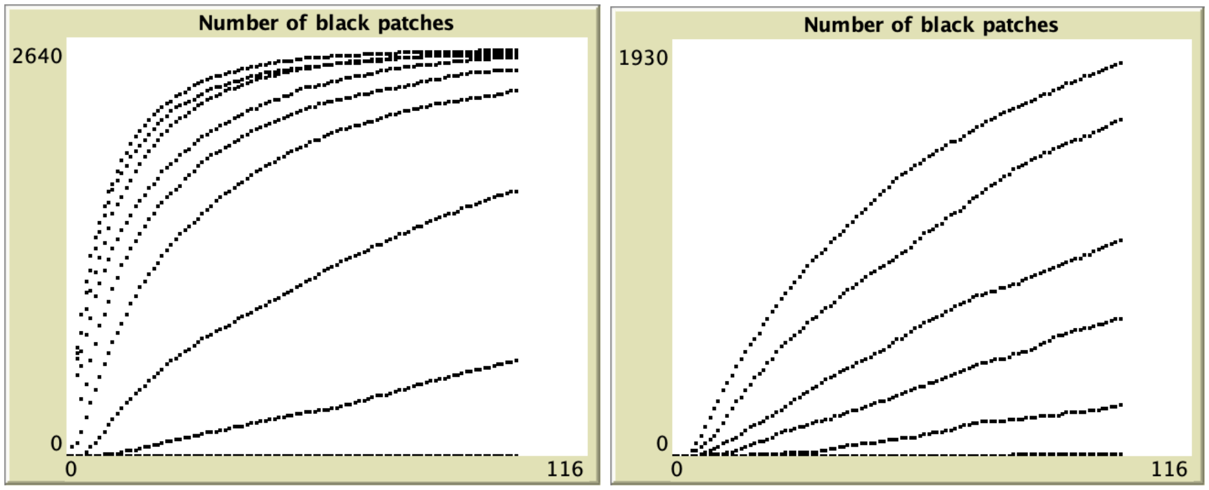
Exercise 2.

Take previous Netlogo model and save it as a new file. Remove scaling process and introduce new slider - noise. This value will be multiplied on epsilon, which is 1 or -1 with probability 0.5. Now in setup initialize all patches with white color and initial c = 0. Implement in the go procedure formula of averaging with noise for von Neumann neighborhood. Subsequently, where we might expect the averaging component of Equation to produce the convergence we see in simple local averaging, the relative weight of the averaging and random elements is such that the system remains in a constant state of flux. Vicsek and Szalay (1987) add an additional dynamic (or rule), such that if at any time the **c** value in a grid cell exceeds a threshold **t_c** then that cell is permanently switched 'on'.

Next you need to implement threshold lock of patches. You can use following procedure.

```
to colour-patches

 ask patches with [pcolor != black][

  ifelse c > threshold [set pcolor black][

   set pcolor scale-color black c threshold 0

  ]]

end
```

Now you need to create plot with ticks on x axis (in this case we simulate model for 100 ticks every time) and number of black patches on y axis for different noise level values.



Plots for threshold 1 (left) and 3 (right)

Implement you neighborhood from variant and recreate plots for different threshold levels (0.5 1 1.5 2 2.5 3).

3. A MORE GENERAL FRAMEWORK: INTERACTING PARTICLE SYSTEMS

Cellular automata are deterministic systems. Given its current state, and the current state of its neighbours, the next state of each cell is exactly specified by the transition rules. It is not a huge departure from the deterministic case to make state transitions probabilistic. This more general framework is provided by interacting particle systems, which are a special case of spatial stochastic processes and constitute an important branch of mathematical statistics.

An interacting particle system (IPS) consists of a lattice of locations– generally, for the cases we are interested in, a two-dimensional grid, with cells indexed by integer values. Any particular site at time t, at location (x, y) has an integer-valued state c, which we denote c_t(x,y) or simply c_t, where the meaning is clear. As in a cellular automaton, a transition rule governs how the state of each site changes, dependent on its current state and on the state of neighbouring sites N_x,y, with a variety of neighbourhood definitions possible. Neighbourhoods at longer ranges are also possible.

State updating in interacting particle systems may proceed either synchronously or asynchronously. While synchronous updating is closer to the cellular automata case that we have just considered, we focus on the asyn- chronous update case because most of the literature presents models in this way. Mathematical results are more easily proven for asynchronous updating because the system state changes one site at a time, compared to the synchronous case when many sites change state simultaneously. In practical terms, this means that each model time step involves randomly selecting a single site in the grid and updating it in accordance with the model transition rule.

The key difference between interacting particle systems and cellular automata is that state transitions are probabilistic. The most basic interacting particle system is the contact process, where each site is in some integer state c. Most simply, $c \in \{0, 1\}$ and the state indicates presence or absence of a 'particle', indicating some entity or condition. The differences only become significant when precise mathematical analysis is required, so we focus on the two-dimensional version with an asynchronous update rule as follows:

(i) If the site is occupied, then with some probability δ the particle dies.

(ii) If the site is vacant, then a new particle is born with a probability given by the proportion of neighbouring sites that are currently occupied.

Practically, the second rule is easily implemented by assigning to the vacant site the state of a randomly selected neighbouring site. The basic contact process in two dimensions is the application of this transition rule and von Neumann neighbourhoods are used. This set-up produces surprisingly interesting behaviour over a range of values of δ. In particular, there is a critical value δc of the death-rate above which the process always dies out, but below which the process will settle to an equilibrium density of occupied sites, with the density continuing to change over time. At $\delta = \delta c$ the process will eventually die out, but it survives for a very long time.

Lets try to implement this model.

First we define global variable N - number of active patches.

In setup we create percent of

```
  set N count patches
 ask patches [
  ifelse random 100 < initial-pop
  [ set pcolor black ]
```

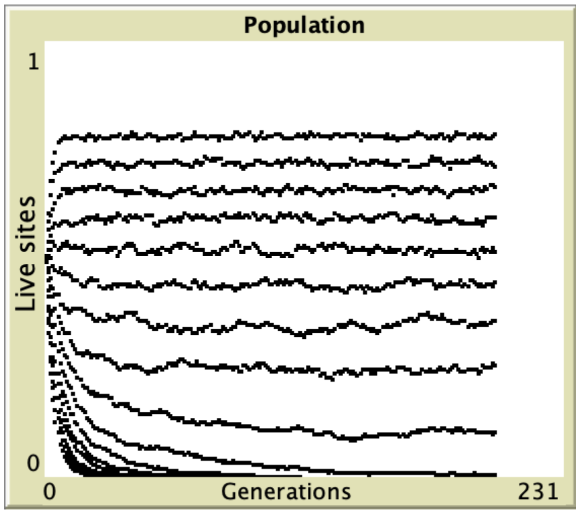
```
    [ set pcolor white ]
  ]
```

where **initial-pop** is slider in the interface tab.

In go procedure we need to create

```
to go
  if not any? patches with [pcolor = black] [ stop ]
  repeat N [
    ask one-of patches [
      ifelse pcolor = black
      [ death-event ]
      [ birth-event ]
    ] ]
  tick
end
```

Create **death-event -** when patch dies with probability δ and **birth-event -** when patch becomes alive (patch copy state from one of his immediate neighbors).



Percent of alive states depending on death probability

Exercise 3.

1. Perform survival analysis for delta depending to you variant of neighborhood. Create plot with justification.

Laboratory work 2.

1. Implement models from exercise 1-3. Answer all questions and do plots. Create report with answers and explanations. Your Netlogo files are also part of laboratory work.
2. Read chapter 3 from the book.

Sources

1. Spatial Simulation. Exploring Pattern and Process David O’Sullivan and George L. W. Perry University of Auckland – Te Whare Wānanga o Tāmaki Makaurau, New Zealand