HAL open science

Agent-based Model Exploration

Arnaud Banos, Philippe Caillou, Benoit Gaudou, Marilleau Nicolas

▶ **To cite this version:**

Arnaud Banos, Philippe Caillou, Benoit Gaudou, Marilleau Nicolas. Agent-based Model Exploration. Arnaud Banos; Christophe Lang; Nicolas Marilleau. Agent-based Spatial Simulation with NetLogo - Volume 1 Introduction and Bases, 1, ISTE Press - Elsevier, pp.125-181, 2015, Volume 1: Introduction and Bases, 978-1-78548-055-3. 10.1016/B978-1-78548-055-3.50004-6 . hal-02402063

HAL Id: hal-02402063

https://hal.science/hal-02402063v1

Submitted on 5 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

4

Agent-based Model Exploration

4.1. Introduction

4.1.1. *Introductory example*

The previous chapters have allowed us to introduce the basics of agent-based model creation with NetLogo. This has resulted in a model such as the one used in our running example. Once the model has been built, the aim is to manipulate it in such a way that new knowledge about the modeled phenomenon can be created. For example, we could look to study the rate of infection resulting from certain parameter values. The use and study of a model is nonetheless as complex as its creation. As such, using our model, we could launch the simulation with standard initial parameters (say 300 humans, 500 mosquitoes, a contamination distance of 5 and a work–home distance of 500), and we would obtain the graph in Figure 4.1(a), indicating no infection beyond the source mosquito, which would lead us to conclude that these parameters lead to no infections. However, upon relaunching the simulation with the exact same values, we might obtain the graph in Figure 4.1(b), with the infection present in almost 100% of individuals after 1,000 iterations.

Such results, typical of stochastic models, invite us to proceed to a more detailed analysis of the situation:

Chapter written by Arnaud BANOS, Philippe CAILLOU, Benoît GAUDOU and Nicolas MARILLEAU.

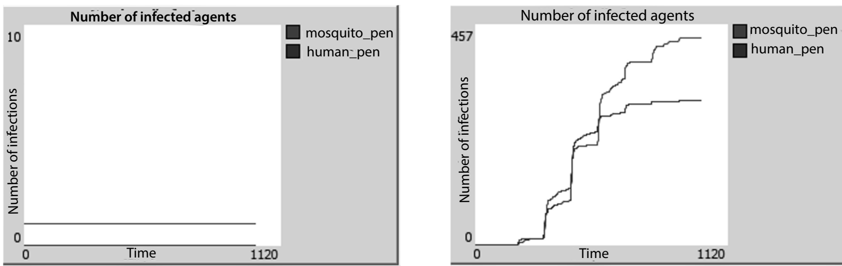

Figure 4.1. *Evolution of the number of humans and mosquitoes infected after 1,000 iterations, during two separate simulations based on the same initial conditions (identical initialization parameters). For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

– Where does the infection come from during the simulation? It would be useful to better visualize the simulation by perhaps representing the distribution of infection dates (Figure 4.2 (a), see section 4.2.2) and the infection sites compared to the movements of the source mosquito (Figure 4.5, see section 4.2.3.1)

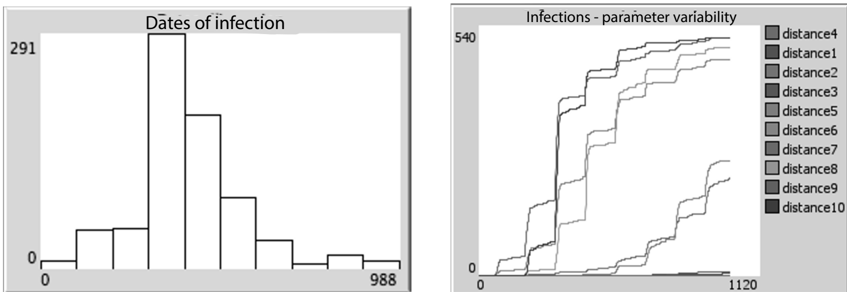

Figure 4.2. *Distribution of the agent infection dates during the course of a simulation (on the left) and the change in infection count across several simulations with different values for one of the model's variables (on the right). For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

– How can several simulations be displayed in order for a comparison? Given the different results obtained across several simulations, it is interesting to display a graph that compares different simulations (for example, the infection count across several simulations, Figure 4.2 (b), see section 4.2.6)

– How can the impact of different parameters on the final result be analyzed? We have defined four main parameters, but which has the greatest effect, and which of these have an impact upon the final infection count? Ideally, a large number of simulations would be carried out with different values for the parameters (for example, to analyze the impact of the contamination distance, Figure 4.2 (b), see section 4.3.2 for the definition of exploration and section 4.3.3.1 for a graph). We would also like to be able to say whether or not the impact is statistically significant (see section 4.3.4).

4.1.2. *Objectives*

More generally speaking, creating a simulation model is only the first step to receive useful results. Any data received from running a simulation once will not necessarily be the same if you launch the same simulation again: random phenomena or even simply the order in which certain agents act may have important consequences leading to widely varying results, even based on the same initial conditions. In our running example, the first infected mosquito may remain in a corner without infecting anyone, as his movements are random. Equally, if a human is infected near the start of the simulation, the infection might spread very quickly, as this agent will rapidly transmit it to his/her neighbors as he/she moves around. This instability phenomenon is even more amplified when the parameters are modified. The inherent unpredictability in complex systems renders a systematic exploration of a model very important, which is facilitated by certain functionalities offered by NetLogo, most particularly graphs and the BehaviorSpace.

Exploring a simulation consists of studying the behavior of the model during and after its execution, and in particular, observing variables defined as objectives (such as the number of infected individuals). Three main approaches are detailed as following:

– How does the model behave during a simulation for a given set of parameters? This first exploration step, which will be covered in the following section, is primarily based on numerical indicators and graphs, updated dynamically throughout the simulation.

– How do the different parameters influence the model’s behavior? Here, a possibility is to carry out several simulations with different parameter values, whether similar or far apart, thus to study the sensitivity of the model to different parameter changes. By doing this, we carry out an exploration of the parameter space. NetLogo’s BehaviorSpace allows for this type of exploration and will be covered in a second section. Analyzing data resulting from the implemented experimental design, with the goal of obtaining results from the sensitivity analysis, mainly requires the use of external tools such as R or Excel. This option will be covered at the end of the second section.

– How can we arrive at one of the simulation’s specific objectives? The goal here is to test virtual scenarios, searching for the “best” solution. For example: whom to vaccinate and at which point in time so that an epidemic spreads as slowly as possible? This is a case of objective optimization. This type of approach, typical of what is known as an “inverse problem”, can be carried out in NetLogo with the help of the Behavior Search extension (http://behaviorsearch.org/documentation/tutorial.html), or in a more sophisticated manner with the OpenMole platform (http://www.openmole.org/), which allows for distribution of simulations on distributed computing environments. This type of approach will not be studied in this chapter and will instead be covered in its own chapter in Volume 2 [BAN 15].

4.2. Exploring a simulation

4.2.1. *Objectives*

NetLogo’s interface is one of its main assets. Simple in use, it offers numerous possibilities for the exploration of models and in particular, the exploration of dynamic graphs which allow the user to follow the behavior of chosen variables during the simulation. The basics of graph creation in NetLogo were presented in Chapter 3. We will now concentrate on the use of these graphs for studying models.

Understanding what is happening during a single simulation requires an initial effort to obtain pertinent data. Knowing which data would

be interesting to extract out of a simulation depends on the simulation itself. For example, the interface of a social network simulation will be different from that of a stadium evacuation simulation. We will now present several commonalities and examples which demonstrate the different possible situations.

During a simulation, two different information types that can be studied may be distinguished as follows:

– variables belonging to agents, which may be represented as distributions or scatter plots;

– aggregated variables (such as the average evacuation speed of the number of infected individuals). These values will have a unique value at each tick of time, and are usually represented as time series (evolution of the number of infected agents) or coupled in an xy-plot (the number of humans infected over the number of mosquitoes).

We will now define three graph types which facilitate the analysis of these data categories:

– The study of the distribution of a variable within an agent population with the help of a histogram;

– Following an agents trajectory with an xy-plot;

– The use of a same graph across different simulations to carry out an initial sensibility study with identical or different parameters.

We will also modify the observers main window so as to produce maps, display infection links and follow a specific agent.

4.2.2. *Using a histogram to display distributions*

Objective: displaying the distribution of infection dates (Figure 4.3)

Method: using a histogram whose range of axes is automatically defined.

Histograms are particularly useful for studying the distribution of values against a continuous variable. For example, we will display the distribution of the infection date of each agent, which is stored in the

date variable of each infection link (see running example). NetLogo can automatically display this with a predetermined discretization.

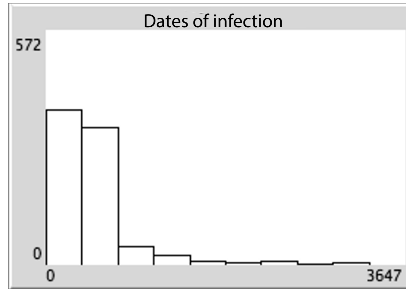

Figure 4.3. *Distribution of the number of infected individuals (along the y-axis) relative to the infection date (along the x-axis)*

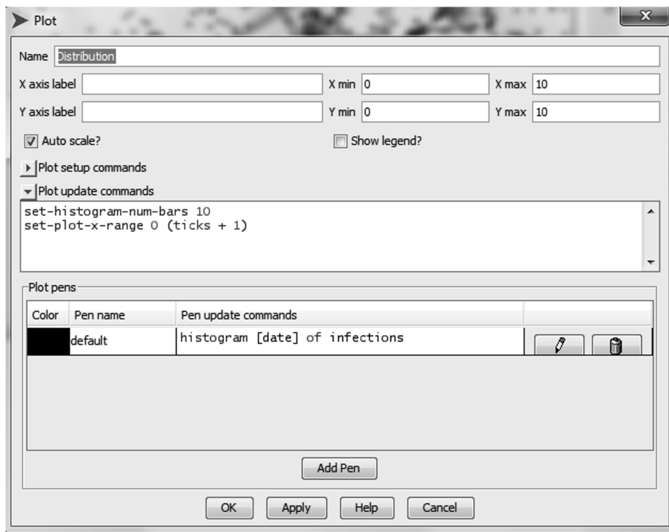

Figure 4.4. *Settings window for the graph displaying the distribution of infections in the form of a histogram*

The commands used to create a histogram are as follows:

– histogram allows for the displayed variable to be specified (date of each agent link infection with [date] of infections). This

command allows for the creation of a histogram from any numerical list (non-number variables are ignored);

– `set-histogram-num-bars` allows for the number of classes of the histogram to be specified. A default value (10 in this case) must be given, but a slider named after this value could also be created, which would allow for the dynamic modification of the number of classes of the histogram.

4.2.3. *Using an xy-plot*

Objective: displaying the locations of all infection events with a color dependent on the date and trajectory of the first infected mosquitoes (Figure 4.5).

Method: creating an xy-plot, defining a color taken from a color pallet based on a variable, adding pens and naming them automatically.

XY-plots allow for points to be traced and if desired, joined, by setting two coordinates (as opposed to series where the *x*-axis always displays time).

They can be used in many different ways during a simulation:

– to represent the change of a variable relative to another so as to display the evolution of the number of infected humans as compared to the number of infected mosquitoes, for example:

- a particular situation consists of displaying *x* and *y* positional coordinates along the *x*- and *y*-axes, respectively. In this case, the change of position and therefore the trajectory of one or several agents is displayed. However, it is also possible to display the change of average position, such as to show where the current center of infection is situated;

– finally, it is also possible to display the location of particular events, such as infection sites.

4.2.3.1. *Displaying a group of points (individuals/elements)*

A certain number of primitives are used to define the tracing of xy-plots. The `plotxy` primitive allows for a point to be added to the

current graph by specifying the x and y coordinates. For example, in order to display the location of each infection recorded in the infection links, the following command can be used to update the graph in Figure 4.5:

```
ask infections
[
  set-plot-pen-color scale-color red date 0 ticks
  plotxy lieuInfectionX lieuInfectionY
]
```

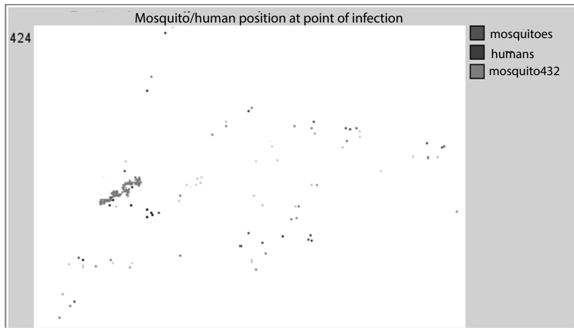

Figure 4.5. *Display of infection sites and of the trajectory of the first infected mosquitoes. The red/blue color of the infection sites is darker for earlier infections. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

The `scale-color` command allows for a color gradient to be easily defined by using a variable to define each level. Here, the `date` variable (infection date, attribute of each `infection` link) is used to display the location with a color (`red`) that is darker for earlier infections. The last two variables allow for the minimum (`0`) and maximum (`ticks`) limits of the scale to be defined.

4.2.3.2. *Displaying one or several trajectories with the help of dynamic pens*

As well as infection sites, we might wish to add the trajectories of the initially infected mosquito or mosquitoes. If we want a separate color for each originally infected mosquito, NetLogo allows for new pens to be

dynamically created. In this situation, we can create a new pen for each originally infected turtle (with isInfectionExternal being true).

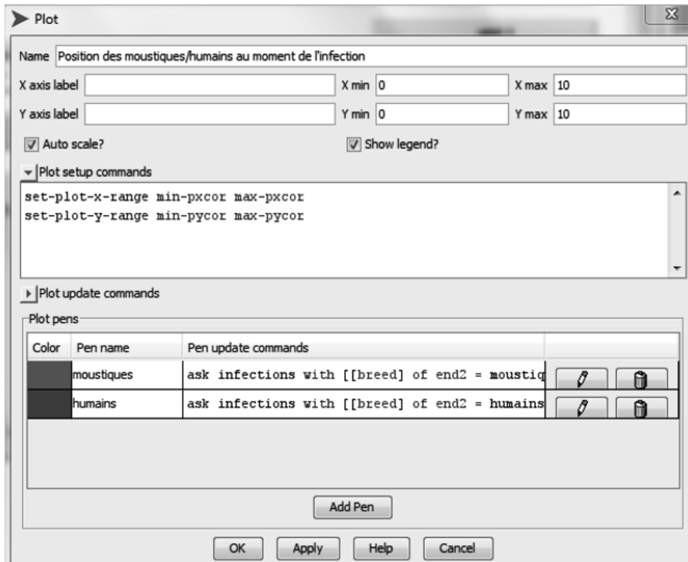

Figure 4.6. *Defining an xy-plot, with a pen for human infections and another for mosquito infections*

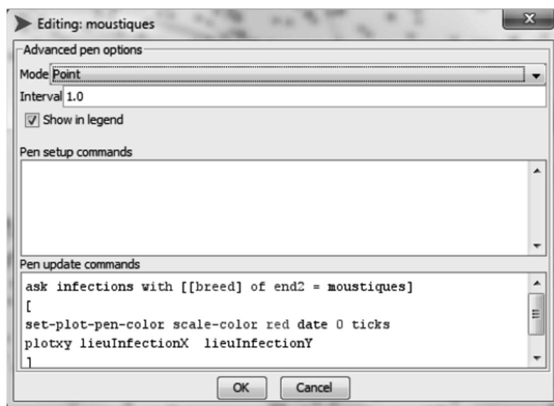

Figure 4.7. *Defining the xy-plots mosquito pen: points are placed at the infection site with a color that is lighter the closer the infection date (`date`) is to the current time of the simulation (`ticks`)*

In the *update* field of the graph, enter:

```
ask turtles with [estInfectionExterieur]
[
  create-temporary-plot-pen word breed who
  set-plot-pen-color green
  plotxy pxcor pycor
]
```

`create-temporary-plot-pen` creates the new pen for each identified agent. The only variable that this command takes is that of the pen name (which appears in the key). To obtain the name of the agent, its (`breed`) and identifier (`who`) are concatenated with the help of the `word` command. This gives us a legible (species) and unique (identifier) name for each trajectory.

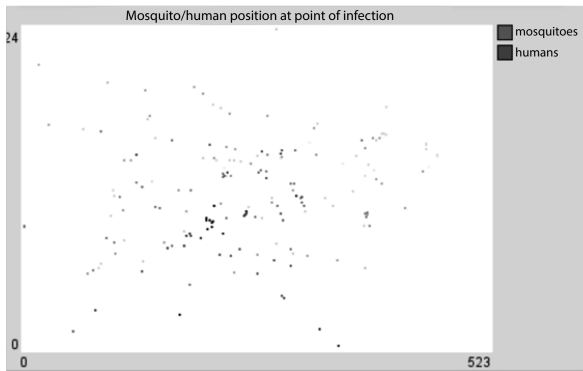

Figure 4.8. *Displaying infection sites with a color gradient dependent on the infection date. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

As this is a new pen, we cannot write this command in the pen update field (*Pen update command*) but instead it must be written in the graph update field (*Plot update command*). Another solution would be to place this command directly within the model's code, while specifying which graph is used with a similar command to the following:

```
set-current-plot "Position of mosquitoes and humans at
moment of infection"
```

4.2.3.3. *Following the infection source mosquito with watch-me*

Given the large number of displayed agents, it may be useful to focus on a single agent. The `watch-me` command allows for an agent to be focused upon (the initially infected agent, for example). A simple implementation of this is to create a *switch* named `Follow_Source-Agent?`, and then to call a simple procedure that uses this Boolean value within the `Go` procedure.

```
to follow-source-agent
 if FollowSource-Agent? and source-agent-follow? = 0
 [
  ask turtles with [isInfectionExternal]
  [
    watch-me
  ] set source-agent-follow? 1
 ]
 if not FollowSource-Agent? and source-agent-follow? = 1
   [
   reset-perspective
   set source-agent-follow? 0
   ]
end
```

The highly permissive character of the NetLogo language should be noted here, as it allows for a high-level primitive (`reset-perspective`) to be called by an agent. This permissivity offers great flexibility but is not always free from ambiguities and requires a certain formality. Equally, calling the `follow-source-agent` procedure from within the `Go` procedure, while practical, means that it is called at each tick. Resorting to a global variable (`source-agent-follow?`) allows for its use to be limited by introducing an intermediate test: the agent is only focused upon if the switch is activated and if the agent in question is not already being followed. We will see an alternative approach later, which separates the call function from the central `Go` procedure.

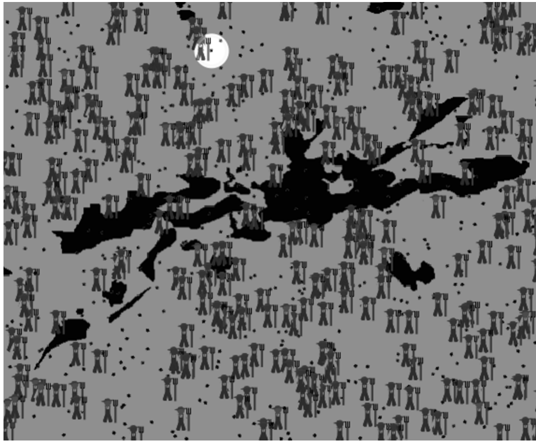

Figure 4.9. *Result of using the watch-me command which allows for an agent to be followed. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

4.2.4. *Mapping with the help of patches*

Objective: cartographic display of spatial variables (Figure 4.10).

Method: constructing a smoothed thematic map with the help of the `diffuse` *and* `scale-color` primitives.

Mapping spatial variables is particularly useful in the case of spatial models such as the one developed here. NetLogo does not have any predefined tools in this field, but certain included primitives offer remarkable possibilities, notably when the base patch entities are exploited. Nonetheless, there remains a delicate issue with the interaction with the user. The user must be able to create their maps at any moment, without interrupting the simulation, and without excessive computation time. This is all the more true with a larger number of patches: in our example, there are 222,700, a value which can be obtained by entering `show count patches` in the observer field.

Unlike the approach detailed in the previous example, the principal idea here is to only activate this cartographic option at the user's request, at the press of a button (`Map`) and with the help of a

scrollable menu (`Chooser`) which facilitates them to choose the variable to be mapped : `"Number of infected humans"` or `"Number of infected mosquitoes"`

Nevertheless, two important bridges must be crossed before arriving at proper cartographic representation. The first step consists of creating the spatial variable at the correct locations. While attempting to exploit the possibilities offered by the patches, it is during this stage that the new `nb-infections-humans` and `nb-infections-mosquitoes` must be stored:

```
patches-own[
locationHome?
nb-infections-humans
nb-infections-mosquitoes
]
```

The second step consists of updating these two variables throughout the simulation. In order to accomplish this, the principal process that influences the calculation of these variables is used, which is the `Sting` procedure in our case. At the moment when an interaction between a mosquito and a human takes place, the virus can be effectively transmitted from the mosquito to the human, or the other way round. In both cases, a record of this “transaction” is kept by incrementally increasing the variable of the corresponding patch (`nb-infections-humans` in the first case and `nb-infections-mosquitoes` in the second):

```
ask patch-here [
set nb-infections-humans
nb-infections-humans + 1
]
```

Once this variable is updated, it becomes possible to map it with the use of two of NetLogo’s primitives: `diffuse` and `scale-color`. The former (`diffuse`) allows for the smoothing of the variable in question by taking the value of each of a patch’s neighboring patches. The

amount of smoothing depends not only on the numerical variable (here, `0.5`), but also on the number of iterations of the procedure (`repeat 20`). The `scale-color` procedure, already discussed earlier, allows for the simple creation of a color gradient once the lower (`min-VC`) and higher (`max-VC`) limits have been defined. For increased legibility, it is possible to turn off the graphical display of the agents present (humans and mosquitoes but also links):

```
to map
 if MappedVariable = "Number of infected humans"
 [
  ask turtles [ht]
  ask links [hide-link]
  ask patches [set pcolor black]
  repeat 20 [diffuse nb-infections-humans 0.5]
  let min-VC min [nb-infections-humans] of patches
  let max-VC max [nb-infections-humans] of patches
  ask patches with [nb-infections-humans > 0]
  [set pcolor scale-color red nb-infections-humains
   min-VC max-VC]
 ]
 if MappedVariable = "Number of infected mosquitoes"
   [
    ask turtles [ht]
    ask links [hide-link]
    ask patches [set pcolor black]
    repeat 20 [diffuse nb-infections-mosquitoes 0.5]
    let min-VC min [nb-infections-mosquitoes] of patches
    let max-VC max [nb-infections-mosquitoes] of patches
    ask patches with [nb-infections-mosquitoes > 0]
    [set pcolor scale-color green nb-infections-
    mosquitoes min-VC max-VC]
   ]
   if MappedVariable = "Land use"
   [
    ask turtles [st]
    ask links [show-link]
```

```
    ask patches [set pcolor black]
    ask patches with [not locationHome?] [set pcolor
       white]
    ask patches with [locationHome?] [set pcolor gray]
  ]
end
```

The maps obtained (Figure 4.10) allow for the spatial distribution of the mosquito–human and human–mosquito transmissions to be visualized.

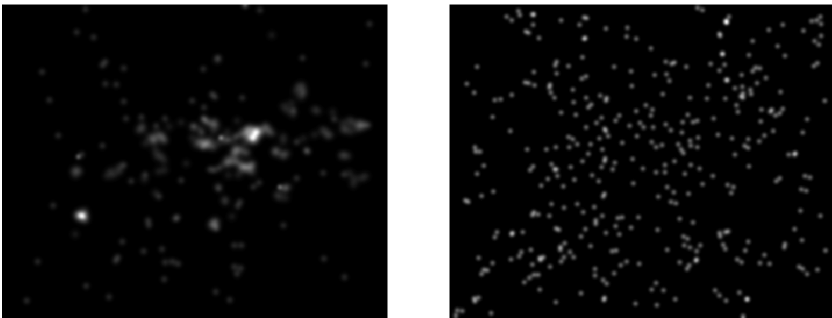

Figure 4.10. *Spatial distribution of the mosquito–human and human–mosquito transmissions. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

4.2.5. *Display of the mosquito/human interaction network*

The model is based on dynamic interactions between humans and mosquitoes. Due to this, it is useful to visualize the underlying interaction network. In order not to penalize the execution of the model by displaying an ever-increasing number of links, we may use a `Show_Links?` switch, which will activate and deactivate the display.

The `show-hide_links` procedure, called from within the Go function, allows for the display of the dynamically created interaction network. The `scale-color` procedure is once again very useful for differentiating objects. In this case, it takes the infection date as a variable: the later the infection, the lighter the link color.

```
to show-hide_links
ifelse Show_Links? and any? links
[
 let min-date min [date] of
 links let max-date max [date] of links
 ask links
 [
   set hidden? false
   if min-date != max-date [set color scale-color green
   date min-date max-date]
   ]]
[ask links [set hidden? true]]
end
```

The link primitives (*links*) allow for the simple manipulation of the graph. As such, the `my-in-links` and `my-out-links` functions called by the `human` agents would allow for the subgraphs of the mosquito–human infections and the human–mosquito infections to be displayed separately and respectively. A dynamic coupling with the GraphStream software (http://graphstream-project.org/) would allow for real-time calculations of the graph indicators, describing the structures displayed within NetLogo more precisely.

4.2.6. *Use of graphs across several simulations*

Objective: following the infection count across several simulations (Figure 4.15).

Method : creating a graph that does not reset in between simulations.

Graphs allow us to follow the state of a simulation at a particular instant (distributions) or its evolution across time (series). It may also be interesting to follow a variable across several simulations, or to compare its evolution between several simulations.

The definition of “persistent” graphs differs from that of standard graphs in the following two ways:

– the simulation must not clear the graph when it relaunches;

– the definition of a temporary graph must take into account that the pens must pass through the origin again, without drawing a line between the last point and the origin.

To accomplish the first objective, the commonly used `clear-all` function must not be used, as it automatically calls the `clear-plots` function, which clears all graphs.

A function alternative to `setup` must be created which includes all the elements of the basic function except for the function that clears all graphs:

```
to clean
  clear-ticks
  clear-turtles
  clear-patches
  clear-drawing
  clear-globals
  load-map
  init-mosquito
  init-human
  create-epidemic
  reset-ticks
end
```

We can link this function to an alternative button to the standard `Setup` (Figure 4.11).

Once this has been carried out, certain graph types can already be persistently used, such as the xy-plot displaying the infection age and the trajectory of the original host mosquitoes defined in the previous section (Figure 4.12). In this case, we can visualize the trajectories of the mosquitoes as well as the infection locations and dates across three simulations.

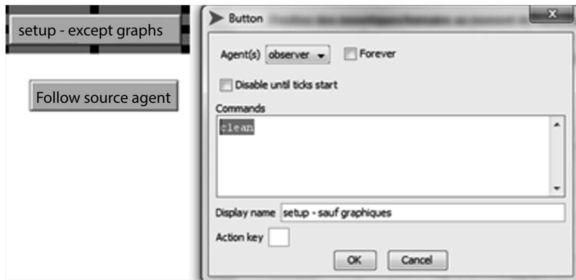

Figure 4.11. *Defining a button which calls the clean function, as opposed to the setup function, which allows for the simulation to be reset without clearing the graphs*

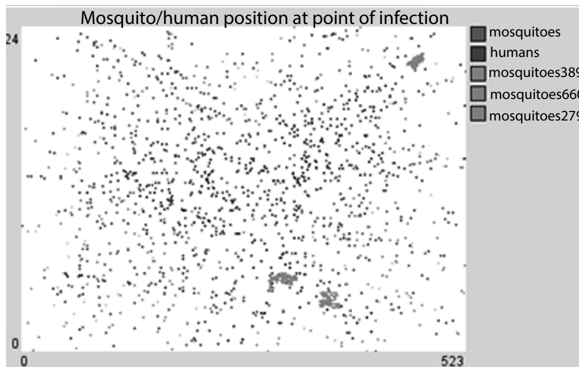

Figure 4.12. *xy-plot of the infection locations and mosquito trajectories across three successive simulations, which notably allows for the random locations of origin of the source mosquitoes. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

Numerous graphs are not, however, able to be used across different simulations. For example, a series representing the number of infected mosquitoes and humans leads to the following result (Figure 4.13). In effect, in a basic series graph, the *x*-axis is incrementally increased at each period. Also, when passing to the next simulation, the pen remains active and NetLogo therefore links the last value of the previous simulation to the first value of the new simulation.

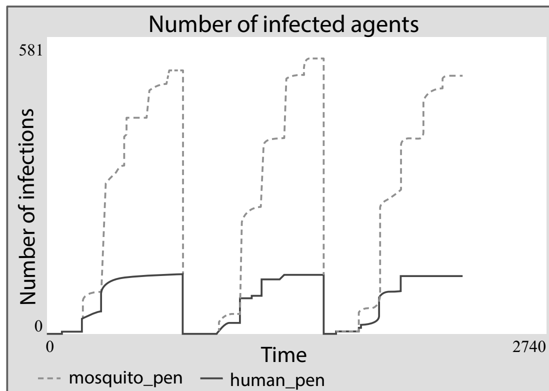

Figure 4.13. *Series graph used across three simulations: the simulation data is combined without returning to the origin for each new simulation*

One solution for defining a graph in series consists of:

– using an xy-plot with the *x*-axis manually defined as representing time (`ticks`);

– deactivating the pen (`plot-pen-up`) while moving to a new simulation, that is to say when `ticks` is equal to `0`.

The pen update command hence becomes (Figure 4.14):

```
if ticks = 0
[
      plot-pen-up
]
plotxy ticks count infections
plot-pen-down
```

This type of graph allows us to easily compare the same variable across several simulations (in our case, with the same parameters so as to carry out an initial stability test of the variable with said parameters) (Figure 4.15).

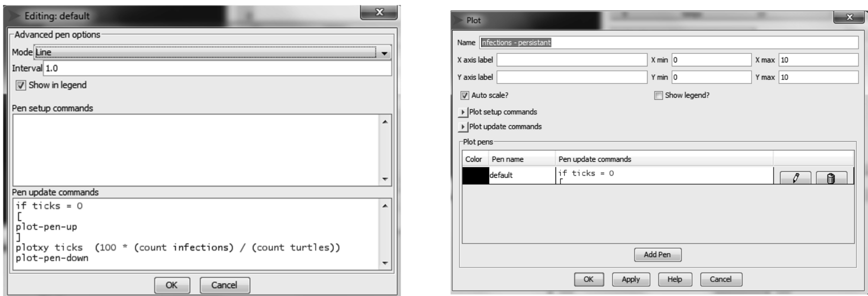

Figure 4.14. *Defining a new graph representing the infection count across several simulations, and defining the pen for this persistent series graph*

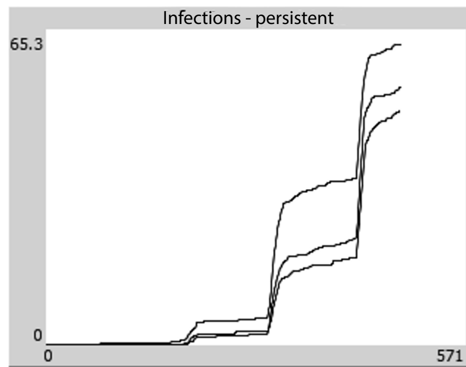

Figure 4.15. *Following the infection count across three successive simulations with identical parameters*

4.3. Exploring several simulations

4.3.1. *Introduction*

The persistent graphs studied in the previous section allow for the user to form an initial idea as to the behavior of the model across several simulations. Nonetheless, this does not allow for a more systematic and detailed study of the variable stability or the models parameter space. For this, NetLogo's BehaviorSpace allows for a large number of simulations to be launched by specifying the parameter values and the desired number of simulations (replications) to be launched for each combination of values.

This tool is particularly useful for:

– studying the stability of the results obtained with the current parameter values. This type of analysis may be carried out by analyzing the standard deviation of the results obtained across several replications;

– studying the impact of each variation of a parameter's value around the current solution (local sensitivity) so as to identify the most influential parameters;

– exploring the spread of possible results for each of the acceptable parameter values (exploring the parameter space). This gives an impression of the results attainable by the simulation based on the main possible configurations.

The BehaviorSpace allows for simple experiment designs to be defined (complete designs), to run these (this can be optionally done in parallel, on several cores), and to export the data resulting from these experiments in CSV files. The goal is usually not to analyze these within NetLogo, as its processing features are rather limited.

4.3.2. *Exploring the parameter space: the BehaviorSpace, step by step*

The BehaviorSpace tool is found in the *Tools* menu. It allows for different experiment plans to be defined (experiments, see Figure 4.16). Each experiment plan defines the values taken by the parameters, the number of times that the simulation is replicated for each combination of parameters, the exit variables and the stop conditions, etc.

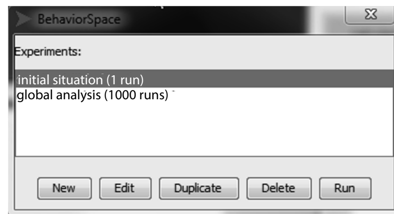

Figure 4.16. *The BehaviorSpace's startup window, giving access to the list of already defined experiment plans, allowing for their modification (*Edit*), their duplication (*Duplicate*), their deletion (*Delete*), their execution (*Run*) or for the creation of another (*New*)*

By default, the experiment plan is defined as the current situation: a single execution with the parameters fixed to their current values and with no stop conditions (see Figure 4.17).

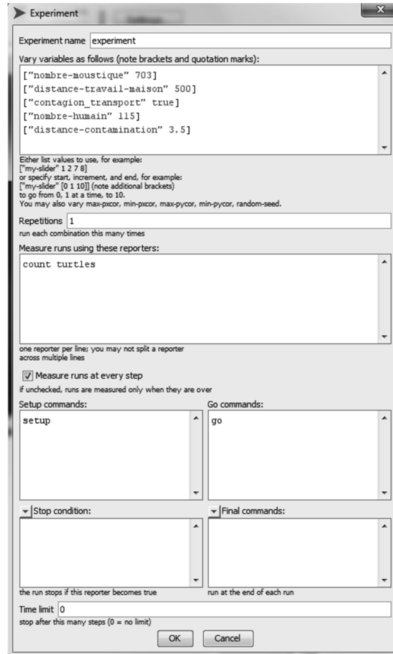

Figure 4.17. *Experiment plan with the current parameter values and a single execution by default*

We will define an experiment plan whose goal is to analyze what influences the number of humans infected after 1,000 ticks of time (Figure 4.18):

– *observed variables*: number of humans infected at the end of the execution, proportion of infected humans, number of mosquitoes and proportion of infected mosquitoes;

– *variable parameters*: number of initial humans and mosquitoes, contamination distance and home–work distance.

Figure 4.18. *Defining an experiment plan to globally explore the parameter space*

We begin by defining a new experiment plan (*New*). The different options to be defined are:

– *Experiment name*: the name of the experiment which will allow for easy identification and which will determine the default name of savefiles. Example: global exploration.

– *Vary variables as follows*: determines the values which will be taken by the parameters. Each line corresponds to a variable. After having specified the name of the variable in between quotation marks, we can specify the values taken by the parameters in two different ways:

- by directly stating which values will be taken by the variable, for e.g. `[‘‘number-mosquito’’ 300 500 1000]` which indicates that the possible values for `number-mosquito` are 300, 500 and 1,000, or `[‘‘contagion_transport’’ true false]` to indicate that `contagion_transport` can take the values `true` and `false`;

- by specifying an initial value, an increment and a final value for the variable, in the form `[‘‘name-variable [initial increment final]]`. For example, `[‘‘number-human’’ [100 100 300]]` indicates that `number-human` will take the values 100, 200 and

300 (which is, in this case, equivalent to writing `[‘‘number-human’’ 100 200 300]`).

– *Repetitions*: the number of times (replications) that each configuration will be executed. For identical parameter values, the observed variable may have different values (due to the stochasticity present within the model) and the simulation may be executed several times for each combination of chosen values so as to obtain more precise results.

– *Measure runs using these reporters*: the observed values which will be exported into a savefile. Each line defines an exported value. It may be a global variable (of the type `date-first-infection`) or more commonly a sum: e.g. `count humans with [isInfected?]` which returns the number of infected humans.

– *Measure runs at every step*: if this box is ticked, the previously defined observed values will be recorded at each simulation step. Otherwise, only the final value will be recorded.

– *Setup commands*: the commands to be executed at the start of each simulation. Usually, this will be the `setup` command, but other commands may also be included, if desired. A command different to `setup` may be used if we specifically require a certain graph or global variable to be conserved throughout the exploration (see following section on advanced analysis).

– *Go commands*: the command to be executed at each simulation step, usually `go`.

– *Stop conditions*: a stop condition if we desire for the simulation to end before the specified time limit (or if no limit is given). For example, if we wanted only to analyze the date of the first human infection, the stop condition could be defined as: `any? humans with [isInfected?]` and to export the infection date by adding `ticks` in the following variables.

– *Final commands*: potential commands to be executed when the stop condition or the time limit are reached. It is possible to export plots or to save the state of the world as new files, for example.

– *Time limit*: the maximum number of steps that the simulation may reach.

The BehaviorSpace will carry out all the possible combinations between the parameters defined in the list. The number of defined simulations can as such increase exponentially: four parameters with five values each already represent 5 x 5 x 5 x 5 = 625 possible combinations. With 10 parameters and 10 values per parameter, we reach 1010 combinations, which means that 10 million simulations must be carried out for a complete study.

The order in which the parameters are listed determines the order in which the simulations will be executed. Let us take the example of two parameters with two and three values, respectively, and a single execution per configuration:

```
[‘‘contagion_transport’’ true false]

[‘‘number-human’’ 100 200 300]
```

The six simulations will be successively executed as follows: (100; true), (200; true), (300; true), (100; false), (200; false), (300; false).

Once the experiment plan has been defined, we can execute it by selecting it in the experiment list and clicking on *Run*. In this case, we can choose the execution options (Figure 4.19):

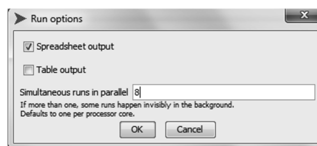

Figure 4.19. *Choice of options for launching an experiment plan within the BehaviorSpace*

Spreadsheet output and *Table output* allow for the file export format to be chosen:

– *Spreadsheet* will generate a spreadsheet with a single line per simulation step (so a single line if only the final value is exported), and one column per variable – simulation couple. If the plan includes 6 simulation runs and 4 observed variables, the spreadsheet will be

composed of $6 \times 4 = 24$ columns (as well as the column with headings). An example is given in Figure 4.21. This layout is adapted when all the intermediate steps are recorded (*Measure runs at every step* ticked within the experiment plan options window).

– *Table* will generate a file with one line per simulation–iteration couple (so as many lines as there are simulations if only the final value is recorded, and $n*m$ lines if n simulations are executed with m recorded iterations). Each column will correspond to a variable. One example is given in Figure 4.20. This layout is made for the analysis of data from a large number of simulations.

landuseV4cp6 exploration globale-table.csv - OpenOffice Calc

	A	B	C	D	E	F	G	H	I	J
1	BehaviorSpace results (NetLogo 5.1.0)									
2	landuseV4cp6.nlogo									
3	exploration globale									
4	08/08/2014 16:28:27:175 +0200									
5	min-pxcor	max-pxcor	min-pycor	max-pycor						
6	0	523	0	424						
7	[run number]	nombre-moustique	distance-travail-maison	contagion_transport	nombre-humain	distance-contamination	[step]	count humains with [estInfecte?]	count moustiques with [estInfecte?]	(count humains with [estInfecte?]) / nombr
8	4	100	100	true	100	3	1000	0	0	0
9	2	100	100	true	100	1	1000	0	0	0
10	3	100	100	true	100	1	1000	0	1	0
11	5	100	100	true	100	3	1000	0	1	0
12	6	100	100	true	100	3	1000	1	1	0.01
13	1	100	100	true	100	1	1000	5	1	0.05
14	8	100	100	true	100	10	1000	0	1	0
15	7	100	100	true	100	10	1000	19	10	0.19
16	9	100	100	true	100	10	1000	33	25	0.33
17	10	100	100	true	200	1	1000	0	1	0
18	11	100	100	true	200	1	1000	0	1	0
19	12	100	100	true	200	1	1000	1	1	0.0050
20	13	100	100	true	200	3	1000	0	0	0
21	14	100	100	true	200	3	1000	0	1	0
22	15	100	100	true	200	3	1000	0	1	0
23	16	100	100	true	200	10	1000	0	1	0
24	17	100	100	true	200	10	1000	1	2	0.0050

Figure 4.20. *Example of a CSV table (imported into OpenOffice) obtained from the BehaviorSpace with the Table output box ticked*

landuseV4cp6 exploration globale-spreadsheet3.csv - OpenOffice Calc

	A	B	C	D	E	F	G	H
1	BehaviorSpace results (NetLogo 5.1.0)							
2	landuseV4cp6.nlogo							
3	exploration globale							
4	08/08/2014 16:17:38:669 +0200							
5	min-pxcor	max-pxcor	min-pycor	max-pycor				
6	0	523	0	424				
7	[run number]	1	1	1	1	2	2	2
8	nombre-moustique	100				100		
9	distance-travail-maison	100				100		
10	contagion_transport	true				true		
11	nombre-humain	100				100		
12	distance-contamination	1				1		
13	[steps]	1000	1000	1000	1000	1000	1000	1000
14								
15	[initial & final values]	count humains with [estInfecte?]	count moustiques with [estInfecte?]	(count humains with	(count moustiques with	count humains with [estInfecte?]	count moustiques with [estInfecte?]	(count humains with [estInfecte?]) / nom
16		0	1	0	0.01	1	1	0.01
17								
18								
19								

Figure 4.21. *Example of a CSV table (imported into OpenOffice) obtained from the BehaviorSpace with the Spreadsheet output box ticked*

Simultaneous runs in parallel: number of simulations which will be run at the same time. NetLogo is able to run several simulations at once to speed up the analysis, as long as the computer's processor contains several cores (choosing a value higher than the number of cores would

slow down the execution instead of speeding it up). The default value is equal to the number of cores. This option is tempting but has several consequences:

– Using all of a processor's cores makes any other use of the computer during the execution of the experiment plan very difficult as all of the processing power is being used by NetLogo. At least one or two cores should be left free so that continued use of the computer is possible.

– As the number of parallel simulations increases, so does the memory used by NetLogo.

– Only one simulation can access the graphical display, which means that if several simulations are running alongside each other, they will not be able to be observed or displayed in graphs (see the use of graphs with the BehaviorSpace in the following section).

– If any global variable is conserved in between simulations, their value will depend on which core they are being run on (see the following section).

– The order of the lines of exported data will be randomly arranged if simulations are running in parallel, especially if variables are being exported at each iteration.

Once the options and filenames have been chosen, the simulations will be successively executed (and occasionally in parallel depending on the chosen options). In order to speed up the processing, it is possible to turn off the main view updates *update view update* and/or the graph and monitor updates *(update plots and monitors*). If the user has chosen to record each iteration's variables, a graph displaying these variables within the current simulation will automatically be generated (see Figure 4.22).

4.3.3. *Analyzing data within NetLogo (advanced use of BehaviorSpace)*

Basic use of the BehaviorSpace allows for data tables to be easily obtained, which can then be analyzed with external tools (Excel, R, etc.). Nonetheless, it is possible to obtain an initial display of data within

the simulation, of which we will present certain uses for carrying out a data analysis of the results within NetLogo.

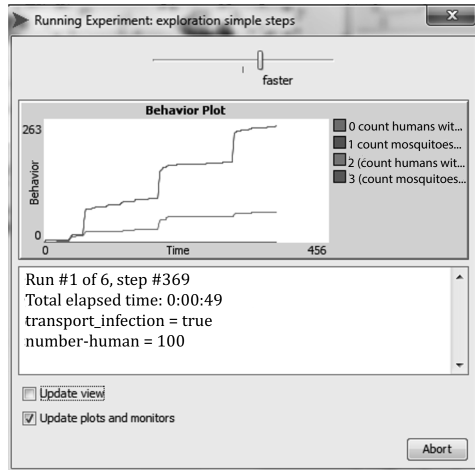

Figure 4.22. *Monitor window for the BehaviorSpace execution, with the number of completed and total simulations, as well as information about the current simulation and interface updating options. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

4.3.3.1. *Use of graphs with BehaviorSpace*

4.3.3.1.1. Constraints specific to BehaviorSpace

In the previous section, we studied the use of a graph across several simulations. It is possible to use the same principle and to improve it with the BehaviorSpace. As such, existing persistent graphs will work with the BehaviorSpace, and will allow for series graphs or xy-plots to be viewed across several simulations. The use of persistent graphs with the BehaviorSpace does, however, have two constraints:

– Just as a new initialization button calling the `clean` function was needed, the standard setup function (which usually calls the `clear-all` and therefore `clear-all-plots` functions) must be replaced by a new function such as `clean` so as to initialize simulations within the experiment plan.

– Only one simulation can access the interface at any given point in time; so if several simulations are being run in parallel (on several cores), only one of them will be displayed upon the graphs.

4.3.3.1.2. Transferring BehaviorSpace parameters onto a graph

In order to visualize the impact of a parameter on a variable, the BehaviorSpace allows for a large number of simulations to be run while varying one or several parameters, and for the results of each simulation to be displayed on persistent graphs. In order to obtain a more legible and complete result, it is possible to add the value of each pen's parameter as a tag, which gives a graph similar to the following one.

Even so, this adds additional constraints relative to the simple persistent graph previously defined:

– It is unfortunately impossible to ask the BehaviorSpace or the user what the name of the variable currently being analyzed is, therefore the graph's update function must be modified.

– Although a pen is defined by its name, this will depend on its value in this case, which means that the pen must be dynamically created and this can only occur at the beginning. The graph will therefore initially not have any pens, and the pen will be added when the graph next updates.

– The type of pen must also be defined, with the use of the `set-plot-pen-mode` command, which allows for the type of graph to be created by the pen (it corresponds to a choice taken from the scrollable *mode* menu in the advanced pen configuration window):

1) `set-plot-pen-mode 0` will cause the pen to trace lines (series graphs or connected xy-plots), see Figure 4.23;

2) `set-plot-pen-mode 1` will create a histogram;

3) `set-plot-pen-mode 2` will draw points (such as for xy-plots, see following example).

– The name of the pen to be created or chosen may be the name of the parameter followed by its chosen value, created

by using the concatenation function: `word : word "distance" distance-contamination`.

– Verifying the existence of a pen (once it has been created) can be done with `plot-pen-exists?`.

– A pen's color can be chosen from a color pallet by using `scale-color`. In order to better differentiate pens and if one is certain that the number of pencils will remain low, NetLogo's included color list (`base-colors`), which offers 14 basic colors, may be used. To receive a reference and to choose the color, the BehaviorSpace run number may be used with the `behaviorspace-run-number` variable.

The `update` function of the graph may be written as follows:

```
ifelse plot-pen-exists? word "distance"
  distance-contamination
[
  set-current-plot-pen word "distance"
    distance-contamination
]
[
ifelse (behaviorspace-run-number < 14)
[
  create-temporary-plot-pen word "distance"
    distance-contamination
  set-plot-pen-color item behaviorspace-run-number
    base-colors
]
[
  set-plot-pen-color wrap-color behaviorspace-run-number
]
]
set-plot-pen-mode 0
if ticks = 0
[
plot-pen-up ] plotxy ticks (100 * (count infections) /
  (count turtles))
plot-pen-down
```

This function allows for the display of the number of infections against the value of the `distance-contamination` parameter (Figure 4.23).

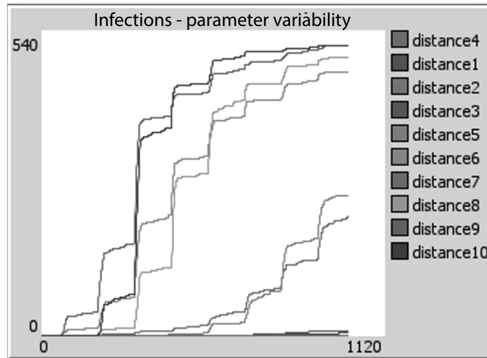

Figure 4.23. *Infection count obtained across several infections by using a pen with the value of its* `distance-contamination` *parameter for each simulation. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

4.3.3.2. *Analyzing data with BehaviorSpace*

In order to delve further within data analysis, data manipulation and aggregation across several simulations is necessary. NetLogo is not suitable for this type of analysis, and it would perhaps be advisable to use an external software such as R (see the next section) or Excel. It is nonetheless possible to carry out certain analyses directly within NetLogo so as to illustrate their potentials and limitations.

The aim is to carry out:

– a representation of the average infection count obtained across the previous simulations, which approximately equates to the standard deviation (Figure 4.24);

– a sensitivity analysis to view the part of the variance represented by each variable. This analysis allows us to see which variables have the largest impact on the final result, the infection count (Figure 4.27).

In order to achieve this aim, the following must be done:

– creating global variables which will record the values of the variable to be analyzed within each simulation;

– updating these variables with their new values;

– making sure these values are not cleared when the simulations are reset.

4.3.3.2.1. Graph of the average and standard deviation

This type of graph allows for the evolution of the average and the stability of a result, whether for fixed parameters or not, to be seen in a more synthetic manner than a graph superimposing all of the simulations.

For example, Figure 4.24 is obtained after 81 simulations following an exploration of the parameter space (4 parameters with 3 values each). The graph is illegible and gives little useful information (other than the fact that the result is unpredictable…). Representing the average and standard deviation allows for the data to be synthesized: the average is clearly increasing by jumps per day (transports). The standard deviation, and thus the instability, becomes particularly high from the fourth day onward, but remains stable at a very high level afterward.

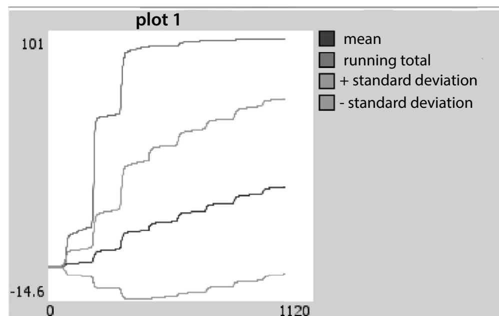

Figure 4.24. *On the left, the infection counts obtained over 81 simulations with different parameters. On the right, a representation of the average (in purple) plus or minus the standard deviation (in blue) of the previous simulations, as well as of the current one (in red). For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

Results such as those shown in Figure 4.25, obtained after 10 simulations with identical parameters, demonstrate great stability after the fourth day (as every entity is infected), while the first three days remain highly unpredictable (with a large standard deviation).

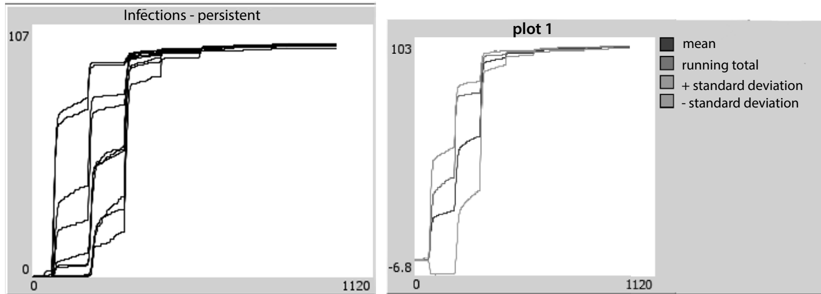

Figure 4.25. *Results of 10 simulations with identical parameters and a high transmission distance. For a color version of the figure, see www.iste.co.uk/banos/netlogo.zip*

4.3.3.2.2. Creating global variables to record the values of the variable to be analyzed within each simulation

In the section at the beginning of the NetLogo code, where species are declared, the global variable which will stock the variable values must be added:

```
globals [list-variable]
```

Then, the following initialization is added to the `setup` function (and not to the `clean` function, as this initialization must only be carried out once):

```
set list-variable [[]]
```

4.3.3.2.3. Updating these variables with the new values

The values can be added to the list both within the NetLogo code and within the `update` function of the graph. The advantage of using the `update` function is that it will not be called during usage of the BehaviorSpace where plots are not being updated, which will reduce

the memory use and calculation time. The plots `update` function is therefore:

```
while [(length list-variable) < (ticks + 1)]
 [
  set list-variable lput [] list-variable
 ]
 let current-list item ticks list-variable
 set current-list lput
    (100 * (count infections) / (count turtles))
      current-list
 set list-variable replace-item ticks list-variable
   current-list
```

`list-variable` is a list of lists containing the value taken by the variable for each tick of the previous simulations. If 10 simulations with a length of 50 ticks are carried out, then `list-variable` will contain 50 lists of 10 elements.

The first section of the code serves to add an empty list if the current tick has not been reached during previous simulations.

The second section finds the list corresponding to the current tick, adds the target variable (the infection count) to it and replaces the old list by the updated `current-list` version.

4.3.3.2.4. Defining graph and pen updates

Once the value has been saved within `list-variable`, all that is needed is to define the pens which will display the average plus and minus the standard deviation, and potentially the current simulation (Figure 4.26).

In order to calculate the average and standard deviation, NetLogo has the `mean` and `standard-deviation` functions allowing for these values to be obtained from a list.

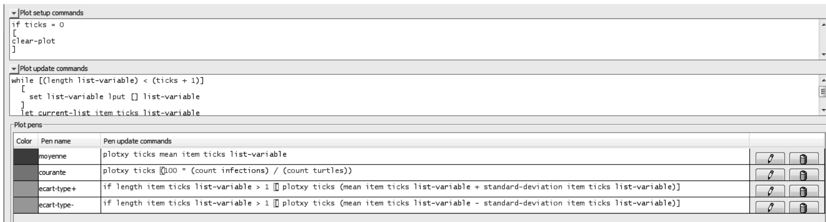

Figure 4.26. *Defining the average and standard deviation pens of the graph*

The `mean` pen will therefore be defined by:

```
plotxy ticks mean item ticks list-variable
```

For standard deviation, it is necessary to verify that the number of values is greater than or equal to 2 in order to avoid the `standard-deviation` function returning an error message:

```
if length item ticks list-variable > 1
[
      plotxy ticks (mean item ticks list-variable +
            standard-deviation item ticks list-variable)
]
```

4.3.3.2.5. Making sure these values are not cleared when the simulations are reset

One final important element to take into account when global variables are being used to store values across several simulations: these global variables must not be cleared when the simulation is reset. This problem is identical to that with graph reinitialization when defining persistent graphs.

The `clear-all` function that is usually used calls the `clear-globals` function, which clears all global variables. We have seen in the previous section that a new initialization function `clean` has to be defined, which does not call `clear-all`, but instead calls all of its elements barring `clear-plots`. Two solutions are possible for conserving global variables:

– not calling `clear-globals`, but then care must be taken to clear the global variables which are not to be kept;

– using `let` to keep the desired global variables: since local variables are not cleared by `clear-globals`, we can make the `clean` function keep the values of `list-variable`, for example:

```
let list-variable-temp list-variable
clear-globals
let list-variable list-variable-temp
```

4.3.3.3. *Analyzing variance: presentation*

Going even deeper within data analysis, we can wish to understand where the variability of received results arises from. For example, let us return to the 81 executions needed for a complete exploration of the parameter space for a model with three values for four parameters, which led to the infection count graph in Figure 4.24. These graphs show that results can vary considerably depending on the parameter values, and it is thus important to correctly calibrate these parameters if we want to obtain realistic results. However, searching for the correct parameter values is complicated and costly (when it is even possible), and it might be interesting to know which parameters have the greatest impact upon the result. If certain parameters have no impact, it is hardly important to precisely define them, and more attention should instead be put toward refining those with a greater importance.

To determine the weight of each parameter on the result (and thus to perform a sensibility analysis based on the variations of a parameter), a possible method is a variance analysis [FAI 13, SAL 09].

We will now break up the variance of a variable x, based on three parameters i, j and k, with n observations of x, written as x_{ijk} (to simplify the notation, the parameter *k* indicates the *k*th simulation for each of the *i* and *j* parameters). This analysis can be generalized to any number of parameters. The variance (square of the standard deviation

calculated above) summarizes the deviations from the mean of the variable:

$$V(x) = \frac{1}{(n-1)} \sum_{ijk} (x_{ijk} - \overline{x_{...}})^2$$

where $\overline{x_{...}}$ and $\overline{x}$ both represent the observed global mean of x: $\overline{x_{...}} = \overline{x} = \frac{1}{n} \sum_{ijk} x_{ijk}$.

As $n - 1$ is constant for all variables, we can consider only the sum of squares (SS) for charts/analysis:

$$SS_{total} = \sum_{ijk} (x_{ijk} - \overline{x_{...}})^2$$

These deviations correspond to the instability and unpredictability of the variable under study. It is possible to break down these deviations. For example, for the first parameter, two extreme cases would be:

– if this parameter is responsible for all the changes in the x variable, this would indicate that if the value of this parameter is fixed, so would be x. For a value of i, x will always be identical and therefore equal to the means of x for this value of i: $x_{ijk} = \overline{x_{i..}}$, with $\overline{x_{i..}} = \frac{1}{n_j n_k} \sum_{jk} x_{ijk}$. Here, $SS_{total} = \sum_{ijk} (\overline{x_{i..}} - \overline{x_{...}})^2 = n_j.n_k \sum_i (\overline{x_{i..}} - \overline{x_{...}})^2$, which means that the deviations correspond to the deviations between the partial means of the parameter and the global variable. The variability within classes is as such nil for the parameter, and the variability between classes is at maximum;

– if no parameters in a sufficiently large sample size have any impact on the final result, the mean of each parameter value will be equal to that of the global variable: $\overline{x_{i..}} = \overline{x_{...}}$. This will also be true for each parameter couple: $\overline{x_{ij.}} = \overline{x_{...}}$ In this case, the previously calculated value, which is the variability between classes, will be nil. Any and all variability will depend on the remainders: any variations of x which are not contained in the means: $SS_{total} = SS_{residual} = \sum_{ijk} (x_{ijk} - \overline{x_{ij.}})^2$. The residual SSs will measure what results from inherent randomness and from non-measured parameters. The higher this value, the less predictable the model is, if only the tested parameter values are known.

The general case is situated in between these two extremes.

When only one parameter, i, exists (and there remain n_k observations for each value of i), the variability (SS_{total}) can be broken down between the variability between classes ($SS_{variable}$) and the residuals (variability within classes, $SS_{residual}$)

$$SS_{total} = SS_{variable} + SS_{residual}$$

$$\sum_{ik} (x_{ik} - \overline{x})^2 = n_k \sum_{i} (\overline{x_{i.}} - \overline{x})^2 + \sum_{ik} (x_{ik} - \overline{x_{i.}})^2$$

For example, if the number of humans takes the values 100, 200 and 300, the variability from the number of humans (nh) will be:

$$SS_{nh} = n_k(\overline{x_{nh=100}} - \overline{x})^2 + n_k(\overline{x_{nh=200}} - \overline{x})^2 + n_k(\overline{x_{nh=300}} - \overline{x})^2$$

For a larger number of parameters, the effect of interactions between parameters must be added: the interaction SSs measure the impact of independent parameters. For example, this may be useful if both the number of humans and mosquitoes must be high for the infection to be significant (as opposed to only one of the values being large).

$$SS_{total} = SS_{variable} + SS_{interaction} + SS_{residual}$$

The variable SSs (the variability between intermediate and global means described previously) are easily calculated and will be analyzed within NetLogo. For the interaction SSs, see section 4.3.4.

4.3.3.4. *Graph of the variance analysis*

The graph obtained (Figure 4.27) represents the evolution of the percentage of total variance attributed to each variable (which is identical to the part of the SS_{total}).

To carry out the variance analysis, the variable values must be stored as with the previous graph, and the values of each parameter must be known for each simulation that is run (so as to be able to calculate the partial means depending on the parameter values).

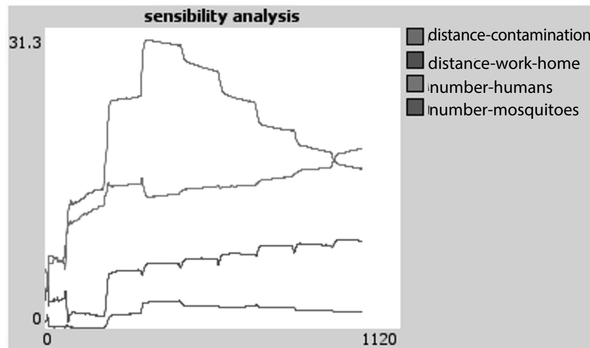

Figure 4.27. *Breaking down the infection count variance from the exploration of a 4 parameter space*

The process this graph follows can be summarized in several steps:

– creation of global variables for storing parameter names and values;

– storing of global variables in between simulations;

– recording of the parameter values at each period;

– calculation of parameter variance;

– display of these values on the corresponding pen.

4.3.3.4.1. In the NetLogo code

Creation of global variable for storing parameter names and values

As with the previous graph, we will use global variables to store the names of the studied parameters as well as their lists of values. We will also continue to use the `list-variable` variable which we defined previously so as to keep the chosen variable's list of values (here, the infection count).

```
globals [list-variable list-parameter-values
  list-parameters]
```

These variables are initialized within the setup as follows:

```
 set list-variable [[]]
```

```
set list-parameter-values [[]]
set list-parameters ["distance-contamination"
"distance-work-home"]
```

Storing of global variables in between simulations

As with the previous graph, we use the `clean` function (which does not call `clear-global`) to store the variable values in between simulations.

4.3.3.4.2. In the BehaviorSpace experiment setup options

Defining the list of parameters to be followed within the BehaviorSpace

Due to `list-parameters` being defined as a global variable, it is possible to choose and analyze different parameters in each experiment (the same method can be used for the variable to be analyzed, which here is still the global infection count).

The `list-parameters` variable is therefore defined within the *setup* of the experiment plan (Figure 4.28).

```
set list-parameters
 ["distance-contamination"
 "distance-work-home"
 "number-human"
 "number-mosquito"]
clean
```

4.3.3.4.3. In the graph setup options

Recording the parameter value at each tick

The graph's `setup` function will be used to record the parameter values for the current simulation. The code will be split into two parts.

First, the function will check that the variable containing the parameter values (`list-parameter-values`) contains at least as many lists as there are values (a list per parameter is required). If this is not the case, empty lists are added:

```
while [(length list-parameter-values) < (length
   list-parameters)]
 [
    set list-parameter-values lput [] list-parameter-
      values
 ]
```

Figure 4.28. *Defining an experiment plan allowing for a sensitivity analysis of the infection count*

Then, the function will go through the parameter list defined in `parameters-list`. We are now interested in the *i*th parameter corresponding to the *i*th list of `list-parameter-values`.

We cannot add a new value corresponding to the parameter value of the current simulation to this list without running into one of

NetLogo's limitations: we have the variable name (it is the *i*th element of `list-parameters` copied into `current-parameter`), but there is no simple way to obtain its value from its name.

We will therefore use NetLogo's `Run` function, which allows for an instruction to be directly passed from a text to a command line. It is impossible to create a local variable with this command, as it would be instantly deleted; instead, this command will be directly executed so as to update `list-parameter-values`, by adding the new value:

```
Run (word "set list-parameter-values replace-item i
list-parameter-values lput " current-parameter "
  current-list")
```

If `current-parameter` is equal to ‘‘`distance-contamination`’’, the command given to NetLogo will be:

```
set list-parameter-values replace-item i list-parameter-
  values
lput distance-contamination current-list
```

The command places in *i*th position in `list-parameter-values` the `current-list` list to which `distance-contamination` is added.

```
if ticks = 0
[
clear-plot
while [(length list-parameter-values) < (length
  list-parameters)]
[
 set list-parameter-values lput [] list-parameter-values
]
let i 0
while [i < length list-parameters]
[
  let current-parameter item i list-parameters
  let current-list item i list-parameter-values
  Run (word "set list-parameter-values replace-item i
```

```
    list-parameter-values lput" current-parameter "
      current-list")
  set i i + 1 ] ]
```

Calculating parameter variance and display of values

The graph's `update` function will carry out the calculations and display them for each of the parameters:

```
let i 0
let current-list item ticks list-variable
if length current-list > 1
[
let total-pop length current-list
let global-mean mean current-list
let total-variance variance current-list
if total-variance > 0
[
while [i < length list-parameters]
[
  let parameter-values item i list-parameter-values
```

The aim is to calculate, for each parameter, the parts of $SS_{variable}$: $\sum n_{i.}(\overline{x_{i.}} - \overline{x})^2/SS_{total}$

With $n_{i.}$, the number of simulations carried out with the value of the *i* parameter is $\overline{x_{i.}} = \sum_{jk} x_{ijk}/n_{i.}$

and $SS_{total} = (n-1) * V(x)$

Independent from the current parameter are the following:

– `Current-list` corresponds to the list of variable values for the current tick during previous simulations (the x_{ijk});

– `Total-pop` corresponds to the total number of simulations that have been carried out (n);

– `Global-mean` corresponds to the global mean ($\overline{x}$);

– `Total-variance` corresponds to the global variance ($V(x)$).

Therefore, we need to calculate $\overline{x_{i.}}$, as well as $n_{i.}$, the population size and the mean of the x_{ijk} – all simulations using the same parameter value.

During the loop, we are interested in the *i*th parameter, and `parameter-values` will correspond to the values of this parameter during previous simulations:

```
if (length current-list) < (length parameter-values)
[
set parameter-values sublist parameter-values
   (length parameter-values - length current-list)
       (length parameter-values)
]
```

If the list of values is smaller than the list of parameter values, the last parameter values of the list are used.

```
let unique-values []
let unique-means []
let unique-size []
```

To explain the importance of these three variables, let us say that we have carried out six simulations, with the two parameters `number-human` and `number-mosquito` set to the following values:

– `Number-human [100 200 100 200 100 200];`

– `Number-mosquito [100 100 500 500 1000 1000].`

Let us also state that the infection counts for the current tick (the x_{ijk} values contained in `current-list`) are the following: `[2 2 2 10 10 10].`

For the first parameter (i=0), `parameter-values` will therefore be `[100 200 100 200 100 200].`

We will attempt to find the means of x_{ijk} for the two values taken by parameter-values, 100 and 200. In order to do this, we use:

– `Unique-values` to store the different values taken by the parameter. At the end, `unique-values = [100 200];`

– `Unique-means` to store the sum of the x_{ijk} corresponding to each unique value (which we will divide by the number of elements to find the mean). At the end, `unique-means=[2+2+10 1+10+10]=[14 21];`

– `Unique-size` to store the number of simulations corresponding to each unique value. At the end, `unique-size=[3 3].`

```
(foreach current-list parameter-values
```

For each `current-list` / `Parameter-values` couple, `?1` will be the current x_{ijk}, with `?2` being the current parameter value

```
[
let index position ?2 unique-values
```

`index` is the position of `?2` in the already identified parameter values.

If the value does not yet exist, it will be added:

```
ifelse index = false
[
set unique-size lput 1 unique-size
set unique-values lput ?2 unique-values
set unique-means lput ?1 unique-means
]
```

Otherwise, the mean and size lists are updated by the current value:

```
[
set unique-means replace-item index unique-means
(?1 + item index unique-means)
set unique-size replace-item index unique-size
(1 + item index unique-size)
]
])
```

Then, the SS is calculated. In order to do this, the variance is updated for each mean – size couple:

```
let var-part 0
  (foreach unique-means unique-size
  [
  set var-part (var-part + ?2 * (?1 / ?2 -
     global-mean) ^ 2)
])
```

Finally, we divide by the population (so as to obtain the parameter variance) and by the total variance (to obtain the proportion of the total variance):

```
set var-part var-part / (total-pop - 1) * 100 /
  total-variance
```

The result can then be displayed, by creating the appropriate pen if it does not yet exist:

```
ifelse plot-pen-exists? item i list-parameters
[
set-current-plot-pen item i list-parameters
]
[
create-temporary-plot-pen item i list-parameters
set-plot-pen-color item i base-colors
]
plotxy ticks var-part
set i i + 1
```

The graph's complete `update` function is thus:

```
let i 0
let current-list item ticks list-variable
if length current-list > 1
[
let total-pop length current-list
```

```
let global-mean mean current-list
let total-variance variance current-list
if total-variance > 0
[
while [i < length list-parameters]
[
 let parameter-values item i list-parameter-values
 if (length current-list) < (length parameter-values)
 [
 set parameter-values sublist parameter-values
    (length parameter-values - length current-list)
       (length parameter-values)
 ]

 let unique-values []
 let unique-means []
 let unique-size []
 (foreach current-list parameter-values
 [
 let index position ?2 unique-values
 ifelse index = false
 [
 set unique-size lput 1 unique-size
 set unique-values lput ?2 unique-values
 set unique-means lput ?1 unique-means
 ]
 [
 set unique-means replace-item index unique-means
 (?1 + item index unique-means)
 set unique-size replace-item index unique-size
 (1 + item index unique-size)
 ]
])
let var-part 0
  (foreach unique-means unique-size
  [
  set var-part (var-part + ?2 * (?1 / ?2 -
```

```
    global-mean) 2)
])

set var-part var-part / (total-pop - 1) * 100 /
total-variance ifelse plot-pen-exists? item i
list-parameters [ set-current-plot-pen item
i list-parameters ] [ create-temporary-plot-pen
item i list-parameters set-plot-pen-color
item i base-colors ] plotxy ticks var-part set
i i + 1 ] ] ]
```

The graph obtained (Figure 4.27) allows for the part of the variance that each variable is causing to be clearly seen. The contamination distance appears to be the variable with the greatest effect during the first few days. Slowly, however, its importance diminishes and the total human population becomes more important. The work–home distance and the number of mosquitoes seem to have a much lesser impact.

The use of this graph/analysis has several limitations which must be noted:

– These results do not take into account the interactions between variables. A more complete variance analysis would allow for a larger part of the total variance to be explained by analyzing the impact of couple variables (the impact of a simultaneous increase of two factors).

– The calculations undertaken here do not allow us to say whether the results obtained are of any significance. Hypothesis tests must instead be added, similar to those we will use with the R software in the following section.

– From a technical point of view, this graph and the previous graph require a large number of calculations and a great deal of memory space to store the value history. They therefore slow down the exploration of the mode incredibly (and do not allow for running parallel simulations). As such, they should only be used as a first approach before carrying a more detailed (and faster) analysis with external tools.

4.3.4. *Data analysis beyond NetLogo: the example of R*

The files obtained from the BehaviorSpace may be analyzed with external tools. For example, the R freeware (http://www.r-project.org/) can be used to carry out simple or complex statistical analyses.

Based on the file from the previous analysis (three different values for four parameters, with one execution per combination), obtained in the form of a table, we will carry out a variance analysis in order to test whether the results are significant or not (see section 4.3.3.3 for a description of a variance analysis). The type of results obtained at the end of the procedure is as follows:

```
Df Sum Sq Mean Sq F value   Pr(>F)
distance.contamination  1 1.389 1.3893 15.407
       0.000212 ***
distance.travail.maison 1 0.703 0.7029  7.795
      0.006873 **
number.mosquito         1 0.095 0.0950  1.053 0.308539
number.human            1 1.712 1.7124 18.989
      4.78e-05 ***
```

4.3.4.1. *Preliminary stage: modifying the CSV file*

The file obtained from NetLogo is a CSV file (see Figure 4.20) which needs to be modified so that it may be imported by R.

By opening the file with a text editor (such as NotePad++ on Windows), we will modify the file in two ways:

1) *Removing the first lines which are not needed for the analysis*

The first 6 lines are deleted (all those before the variable list):

Deleted lines:

```
"BehaviorSpace results (NetLogo 5.1.0)"
"landuseV5a.nlogo"
"stability analysis"
```

```
"08/11/2014 18:00:06:646 +0200"
"min-pxcor","max-pxcor","min-pycor","max-pycor"
"0","523","0","424"
```

Lines kept:

```
"[run number]","distance-contamination",
   "distance-work-home",
"number-mosquito","number-human","[step]",
"count humans with [isInfected?]",
"count mosquitoes with [isInfected?]",
"(count humans with [isInfected?]) / number-human",
"(count mosquitoes with [isInfected?]) /
   number-mosquito"
"1","3","100","300","100","1000","6","8","0.06",
   "0.026...67"
"2","3","100","300","200","1000","30","24","0.15","0.08"
```

2) *Removing the inverted commas*

NetLogo adds inverted commas (") all over the place, including around numerical values. Therefore, a Search/Replace operation must be carried out to replace all occurrences of " by a blank space so as to remove them.

Once these modifications have been made, the file can be used with R.

4.3.4.2. *Analysis with R*

Once the current directory has been defined, the file can be imported into R:

```
> dataexf=read.table("landuse.csv",header=T,sep=",
   ",dec=".")
```

We can now check that the values and columns correspond correctly by displaying a data synthesis:

```
> summary(dataexf)
X.run.number. distance.contamination distance.work.home
number.mosquito number.human X.step.
Min.   : 1    Min.   : 3    Min.   : 100.0
Min.   : 300  Min.   :100   Min.   : 1000
1st Qu.: 21  1st Qu.: 3    1st Qu.: 100.0
1st Qu.: 300  1st Qu.:100   1st Qu.:1000
Median : 41  Median : 5     Median : 500.0
Median : 500  Median :200     Median : 1000
Mean   : 41   Mean   : 6    Mean   : 533.3
Mean   : 600  Mean   :200   Mean   :1000
3rd Qu.: 61  3rd Qu.:10    3rd Qu.:1000.0
3rd Qu.: 1000 3rd Qu.: 300  3rd Qu.:1000
Max.   : 81   Max.   : 10   Max.   :1000.0
Max.   : 1000 Max.   : 300  Max.   :1000

count.humans.with..isInfected..
count.mosquitoes.with..isInfected..
X.count.humans.with..isInfected.....number.human
 Min.   :  0.00                Min.   :  1.0
0   :25
 1st Qu.:  0.00                1st Qu.:  1.0
1   :14
 Median : 52.00                Median : 64.0
0.01   : 2
 Mean   : 98.96                Mean   :198.2
0.06   : 2
 3rd Qu.:199.00                3rd Qu.:290.0
0.78   : 2
 Max.   :300.00                Max.   :973.0
0.98   : 2

(Other):34
X.count.mosquitos..with..isInfected......
        number.mosquitos
 0.0010               : 9
 0.0033333333333333335: 9
```

```
0.0020                 : 8
0.128                  : 2
0.58                   : 2
0.918                  : 2
(Other)                : 49
```

If we want to carry out operations between columns (in a faster manner than with NetLogo), or if we have forgotten to calculate the target variable (like in this situation, where we only have the total of mosquitoes and the total of infected humans), we can use R's functionalities to perform calculations on matrices. Here, we will replace column 6 (which contained the final tick, 1000) by the infection count and then rename the column:

```
> dataexf[,6]<-(dataexf[,7]+dataexf[,8])/(dataexf[,4]+
  dataexf[,5])
> colnames(dataexf)[6]<-"infections"
```

We check that the values are coherent:

```
> dataexf[,6]
 [1] 0.0350000000 0.1080000000 0.0016666667 0.0016666667
0.1157142857 0.1650000000 0.0018181818 0.1725000000
0.1246153846 0.0025000000
[11] 0.1980000000 0.1350000000 0.0066666667 0.4414285714
0.2375000000 0.0009090909 0.0008333333 0.8446153846
0.0100000000  0.0020000000
[21] 0.3466666667 0.0016666667 0.1442857143 0.0012500000
0.0009090909 0.6875000000 0.0492307692 0.0850000000
0.0240000000  0.0016666667
[31] 0.0016666667 0.4128571429 0.5462500000 0.0254545455
0.3675000000 0.1138461538 0.5375000000 0.3240000000
0.7816666667 0.1216666667
[41] 0.8357142857 0.0012500000 0.0009090909 0.7766666667
0.2900000000 0.5825000000 0.6960000000 0.9266666667
0.6466666667 0.8628571429
[51] 0.0012500000 0.0009090909 0.7558333333 0.9538461538
0.0025000000 0.0080000000 0.0016666667 0.3333333333
0.3385714286 0.7462500000
```

```
[61] 0.0009090909 0.6466666667 0.7430769231 0.0025000000
0.0020000000 0.9833333333 0.0016666667 0.9500000000
0.9837500000 0.8545454545
[71] 0.9316666667 0.9792307692 0.0025000000 0.9760000000
0.9650000000 0.0016666667 0.9414285714 0.9787500000
0.0009090909 0.0008333333
[81] 0.9769230769
> summary(dataexf[,6])
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0008333 0.0020000 0.1443000 0.3440000 0.7431000 0.9838000
>
```

We can then launch the variance analysis in a single line:

```
> aovexf<-aov(infections~distance.contamination
*distance.work.home*number.mosquito
*number.human,data=dataexf)
```

Then we display a synthesis of results:

```
> summary(aovex)
                  Df Sum Sq Mean Sq F value Pr(>F)
distance.contamination 1 1.389 1.3893 15.407 0.000212 ***
distance.work.home 1 0.703 0.7029 7.795 0.006873 **
number.mosquito 1 0.095 0.0950 1.053 0.308539
number.human 1 1.712 1.7124 18.989 4.78e-05 ***
distance.contamination:distance.work.home
1 0.012 0.0118 0.130 0.719290
distance.contamination:number.mosquito
1 0.043 0.0431 0.478 0.491639
distance.work.home:number.mosquito 1
0.176 0.1765 1.957 0.166583 d
istance.contamination:number.human 1
0.653 0.6531 7.242 0.009044 **
distance.work.home:number.human
1 0.104 0.1040 1.154 0.286717
number.mosquito:number.human
1 0.066 0.0660 0.732 0.395395
distance.contamination:distance.work.home:number.mosquito
1 0.305 0.3053 3.386 0.070328
```

```
distance.contamination:distance.work.home:number.human
1 0.144 0.1440 1.597 0.210906
distance.contamination:number.mosquito:number.human
1 0.051 0.0512 0.568 0.453742
distance.work.home:number.mosquito:number.human
1 0.000 0.0001 0.001 0.973626
distance.contamination:distance.work.home:number.mosquito
  :number.human
1 0.022 0.0221 0.245 0.622349
Residuals           65 5.861 0.0902 ---
Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 1
```

The third column (`Sum Sq`) corresponds to the sums of the squares of the deviations (*SS*), such as those which we calculated with NetLogo in the previous part (the 4 first lines).

We can also obtain each variable relative part by dividing each SS by the total of the deviations (this also gives us the part of the variance):

```
> round(summary(aovex)[[1]][2]/
        sum(summary(aovex)[[1]][2])*100,2)
              zSum Sq
distance.contamination 12.25
distance.work.home 6.20
number.mosquito 0.84
number.human 15.10
distance.contamination:distance.work.home 0.10
distance.contamination:number.mosquito 0.38
distance.work.home:number.mosquito 1.56
distance.contamination:number.human 5.76
distance.work.home:number.human 0.92
number.mosquito:number.human 0.58
distance.contamination:distance.work.home
  :number.mosquito 2.69
distance.contamination:distance.work.home
  :number.human 1.27
distance.contamination:number.mosquito
  :number.human 0.45
```

```
distance.work.home:number.mosquito:number.human 0.00
distance.contamination:distance.work.home
  :number.mosquito:number.human 0.19
Residuals
```

We can confirm the results obtained from NetLogo in the previous section: the parameters with the greatest impacts are clearly number-human (15% of the total variance) and distance-contamination (12% of the total variance).

The complete analysis done with R offers us numerous additional information relative to our NetLogo graph:

– We also have the weight of paired factors now: the combined impact of `distance-contamination` and `number-human` also appears to be rather important (5.7% of the variance).

– The residual part (`Residuals`) is particularly interesting: it corresponds to the variance which is not explained by the deviations between partial means and the global mean, that is to say, all the deviations obtained with identical parameters. This is the variance which is due to other parameters or random phenomena present in the model (in our case, mosquito movement and, more importantly, the location of the original infected mosquito). We can see that our four parameters only justify 48% of the total variance. The model is therefore very unstable even with fixed parameters.

– Another piece of useful information received in the R analysis is the statistical test carried out for each parameter ((`F value`, `Pr(>F)`). The columns correspond to a Fisher's test which tests the following hypothesis: "the parameter has no impact upon the variable" (if we assume a linear impact). The indicated probability is the probability that this statement is true. There is therefore 0.02% probability that, with the obtained results, the `distance-contamination` parameter has no impact on the `count-infection` variable. The stars (`*** ** *`) synthesize this value. This value gives the analysis a statistical justification (the impact of `distance-contamination` is statistically significant within our model). Nonetheless, care must be taken so as

not to misinterpret this value: a parameter without a star (*) (where Pr>10%) does not mean that the parameter has no impact, but instead that no conclusion can be made using the obtained results.

4.4. Conclusion

The exploration of NetLogo models is greatly facilitated, both by the graphical, dynamic and reactive interface and by the BehaviorSpace, an integrated tool which allows for experiment plans to be carried out simply and intuitively. Coupled with NetLogo graphs, or with an external data analysis software such as R, this tool offers robust and statistically founded analysis perspectives for any NetLogo model.

Several important and more advanced aspects of model exploration with NetLogo have nonetheless not been covered here and will be specifically focused upon in Volume 2 [BAN 15].

First, these need to be done with automatic calibration of models, as well as optimization, which requires the maximization/minimization of an objective function, which might allow for a configuration that minimizes the final infection count to be found. This process requires a large number of calculations and is thus not included within the BehaviorSpace, but can nonetheless be used in NetLogo with the help of the BehaviorSearch extension (http://behaviorsearch.org/documentation/tutorial.html), or in a more sophisticated manner with the OpenMole platform ([REU 13] http://www.openmole.org/), which allows for the distribution of simulations on distributed computing environments.

Second, this has to be done with the direct integration of R's or GraphStream's analysis functions into the NetLogo code, via the use of existing plug-ins, so as to directly obtain the statistical indicators (R) or graphs (R, GraphStream) necessary for the evaluation of the model within NetLogo.

Finally, this has to be done with the use of more advanced statistical tools such as clustering algorithms, with or without interaction with the NetLogo model, allowing for the analysis of homogeneous groups of individuals or of parameter groups which produce homogeneous results from an exploration of their parameter space.