Modelling Infectious Diseases with Kendrick

<Put the authors here>

Version of 2015-08-03

Contents

Chapter 1

Introduction

Understanding how infectious diseases propagate is a key challenge for the 21st century. Mathematical modelling is a powerful method for studying complex systems that is commonly used in many scientific disciplines. It is widely used to carry out researches on modelling infectious diseases in order to study the mechanisms of transmission, explore characteristics of epidemics, predict the future course of an outbreak and evaluate strategies to find a best control-program. The first mathematical model of epidemiology was proposed by Daniel Bernoulli in 1766 to defend the practice of inoculation against smallpox. The major contribution to modern mathematical epidemiology was carried out by Kermack and McKendrick who had formulated a compartmental model based on relatively simple assumptions on the rates of flow between different classes categorised by epidemiological status.

Kendrick is a domain-specific modelling language that provide tools in order to design, explore and visualize your epidemics models. Kendrick is an embedded DSL and use the Pharo programming language as its host language. This book shows how to visualize the spatio-temporal evolution of epidemiological models using Roassal.

Some examples from this book are coming from the book of M. Keeling & P. Rohani "Modeling Infectious Diseases in Humans and Animals". There is a website with on-line material for the book, where you can find the programs and the background of each program in C++, FORTRAN and Matlab.

In order to use Kendrick, you need first to install the last version of Pharo in your computer.

1.1 How to install Pharo

Pharo is available as a free download from http://pharo.org/download. Click the button for your operating system to download the appropriate `.zip` file. For example, the full Pharo 4.0 distribution for Mac OS X will be available at http://files.pharo.org/platform/Pharo4.0-mac.zip.

Once that file is unzipped, it will contain everything you need to run Pharo (this includes the VM, the image, and the sources, as explained below).

1.2 Pharo components

Like many Smalltalk-derived systems, Pharo currently consists of three main components. Although you do not need to deal with them directly for the purposes of this book, it is important to understand the roles that they play.

1. The **image** is a current snapshot of a running Pharo system, frozen in time. It consists of two files: an *.image* file, which contains the state of all of the objects in the system (including classes and methods, since they are objects too), and a *.changes* file, which contains the log of all of the changes to the source code of the system. For Pharo 4.0, these files are named `Pharo4.0.image` and `Pharo4.0.changes`. These files are portable across operating systems, and can be copied and run on any appropriate virtual machine.

2. The **virtual machine** (VM) is the only part of the system that is different for each operating system. Pre-compiled virtual machines are available for all major computing environments. (For example, on Windows, the VM file is named `Pharo.exe`).

3. The **sources** file contains the source code for all of the parts of Pharo that don't change very frequently. For Pharo 4.0, this file is named `PharoV40.sources`.

As you work in Pharo, the *.image* and *.changes* files are modified (so you need to make sure that they are writable). Always keep these two files together. Never edit them directly with a text editor, as Pharo uses them to store the objects you work with and to log the changes you make to the source code. It is a good idea to keep a backup copy of the downloaded *.image* and *.changes* files so you can always start from a fresh image and reload your code.

The *.sources* file and the VM can be read-only, and can be shared between different users. All of these files can be placed in the same directory, but it is also possible to put the Virtual Machine and sources file in separate directory where everyone has access to them. Do whatever works best for your style of working and your operating system.

1.3 Launching Pharo

To start Pharo, double click on the Pharo executable (or, for more advanced users, drag and drop the `.image` file onto the VM, or use the command line).

On **Mac OS X**, double click the `Pharo4.0.app` bundle in the unzipped download.

On **Linux**, double click (or invoke from the command line) the `pharo` executable bash script from the unzipped Pharo folder.

On **Windows**, enter the unzipped Pharo folder and double click `Pharo.exe`.

In general, Pharo tries to "do the right thing". If you double click on the VM, it looks for an image file in the default location. If you double click on an `.image` file, it tries to find the nearest VM to launch it with.

Once you have multiple VMs (or multiple images) installed on your machine, the operating system may no longer be able to guess the right one. In this case, it is safer to specify exactly which ones you meant to launch, either by dragging and dropping the image file onto the VM, or specifying the image on the command line (see the next section).

Launching Pharo via the command line

The general pattern for launching Pharo from a terminal is:

```
<Pharo executable> <path to Pharo image>
```

Linux command line

For Linux, assuming that you're in the unzipped `pharo4.0` folder:

```
./pharo shared/Pharo4.0.image
```

Mac OS X command line

For Mac OS X, assuming that you're in the directory with the unzipped `Pharo4.0.app` bundle:

```
Pharo4.0.app/Contents/MacOS/Pharo Pharo4.0.app/Contents/Resources/Pharo4.0.
    image
```

Incidentally, to drag-and-drop images on Mac OS in Finder, you need to right-click on `Pharo4.0.app` and select 'Show Package Contents'.

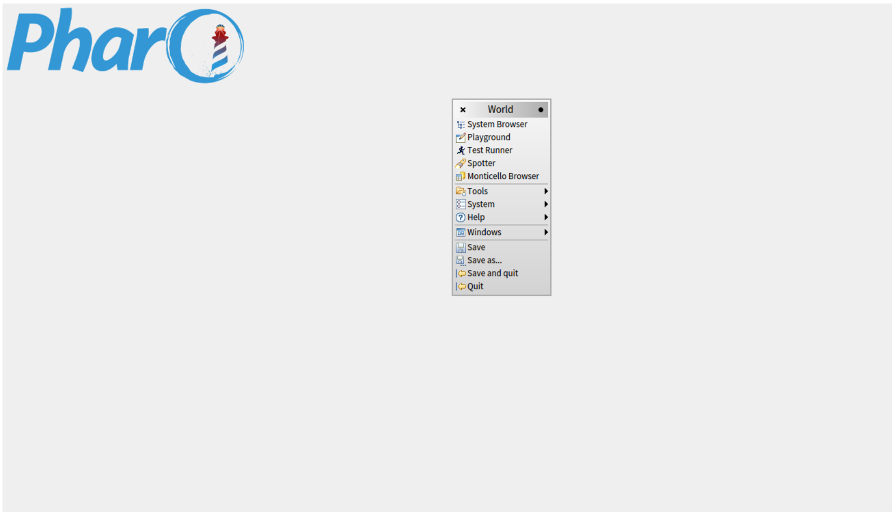


Figure 1.1: Pharo 4.0 window with World Menu activated.

Windows command line

For Windows, assuming that you're in the unzipped `Pharo4.0` folder:

```
Pharo.exe Pharo4.0.image
```

1.4 The World Menu

Once Pharo is running, you should see a single large window, possibly containing some open workspace windows (see Figure 1.1). You might notice a menu bar, but Pharo mainly makes use of context-dependent pop-up menus.

Clicking anywhere on the background of the Pharo window will display the World Menu. World Menu contains many of the Pharo tools, utilities and settings.

You will see a list of several core tools in Pharo, including the class browser and the workspace.

1.5 Install Kendrick

From the the World Menu, select the "Tools" and after that the Configuration Browser. Find in the Configuration Browser, the Kendrick configuration and

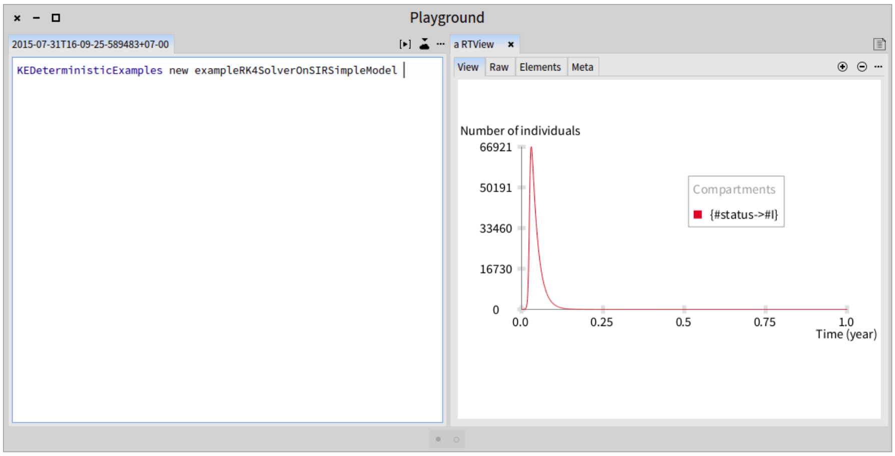

```
KEDeterministicExamples new exampleRK4SolverOnSIRSimpleModel
```

Figure 1.2: Run the script in the Playground.

select it. Click on "Install Stable version". You need to be connected to the Internet, in order to download the Kendrick package.

1.6 Finding out the Examples package integrated in Kendrick

From the World menu, select the "System Browser". Type "Kendrick" in the searching textbox to find the Kendrick packages then expanding "Kendrick". All the integrated examples of Kendrick could be found in the "Examples" package. To run an example, from the World menu, select Workspace. In the Playground form, type a script to create an example and run it. For example:

```
KEDeterministicExamples new exampleRK4SolverOnSIRSimpleModel
```

KEDeterministicExamples: class name.

exampleRK4SolverOnSIRSimpleModel: method name.

Then click on the play button to run the script. The result will be represented in the next view (Figure 1.2)

1.7 Launching models with simulation tools of Kendrick

An epidemiological model can be interpreted to run a deterministic, stochastic or agent-based simulation. The simulations are written once but for all the Kendrick models.

Once the model is specified, create new instance of KESimulator and determine the parameters of simulation such as: algorithm, starting time, ending time and the step. For example:

```
simulator := KESimulator new: #RungeKutta from: 0.0 to: 1.0 step: 0.001.
simulator executeOn: model.
```

For this moment, the platform supported the algorithms: ODE Solvers (RungeKutta, Euler, etc.), Gillespie's direct, Explicit Tau Leap and Individual-based simulation. The scripts following allows to create these simulations:

```
simulator := KESimulator new: #Gillespie from: 0.0 to: 1.0 step: 0.001.
simulator executeOn: model.
```

```
simulator := KESimulator new: #TauLeap from: 0.0 to: 1.0 step: 0.001.
simulator executeOn: model.
```

```
simulator := KESimulator new: #IBM from: 0.0 to: 1.0 step: 0.001.
simulator executeOn: model.
```

In Kendrick platform, the simulation results are stored as time series.

```
simulator allTimeSeries
```

This script will return an array of all time series. To obtain the time series of a compartment, just indicate the name of this compartment as parameter of the function "timeSeriesAt".

```
simulator timeSeriesAt: '{#status: #I}'
```

These scripts below allow to apply an additional function on the time series.

```
(simulator timeSeriesAt: '{#status: #I}') sqrt
```

```
(simulator timeSeriesAt: '{#status: #I}') log
```

The time series then are passed as data of the diagram builder to view the simulation results:

```
diag := (KEDiagramBuilder new) data: simulator allTimeSeries.
diag open
```

Other configurations of diagram:

```
diag xLabel: 'Time (years)'.
diag yLabel: 'Infectious'.
diag legendTitle: 'Vaccination rate'
```

By default, xLabel: Time (days), yLabel: Number of individuals, legendTitle: Compartments.

Chapter 2

Introduction to Simple Epidemic Model

The targeted model of the **Kendrick** language is compartmental model such as the SIR, SEIR model ... in which the individuals are first considered as *Susceptible* to pathogen (status S), then can be infected, assumed *Infectious* (status I) that can spread the infection and *Recovery* (status R) who are immunised and cannot become infected again. The transition of status between compartments is represented mathematically as derivatives of compartment size with respect to time.

At the moment, **Kendrick** supports for the mathematical models of epidemiology based on ordinary differential equations (**ODEs**). The system of ODEs followed represents the SIR classic model of epidemiology:

These models are specified using the Kendrick language and modeled using the simulation module integrated into the platform. The simulator takes the Kendrick model (the epidemiological model written in Kendrick language) and performs a simulation algorithm and give out the result show-

$$\begin{cases} \frac{dS}{dt} &= -\beta S I \\ \frac{dI}{dt} &= \beta S I - \gamma I \\ \frac{dR}{dt} &= \gamma I \end{cases}$$

Figure 2.1: Mathematical description of SIR model using ODEs.

ing the spatial and temporal evolution dynamics of each compartment. This visualization is done by using Roassal.

The simulation module supports three modeling formalisms: deterministic, stochastic and individual-based (also called agent-based). The modelers can switch between the simulation modes by indicating the algorithm used. At the moment, we use the RK4 method for deterministically resolving ODEs. The stochastic simulation converts the ODEs of the model to events and using Gillespie's algorithms to generate stochastic model. The individual-based simulator allows to reach the model at more detailed level.

2.1 Simple SIR (without births and deaths)

Program 2.1 is a simple SIR model. These are the equations and the code of the model:

Equations

$$\frac{dS}{dt} = -\beta * S * I \tag{2.1}$$

$$\frac{dI}{dt} = \beta * S * I - \gamma * I \tag{2.2}$$

$$\frac{dR}{dt} = \gamma * I \tag{2.3}$$

Pharo code

Here, we just demonstrate the code written in Smalltalk language to resolve the system of equations and view the results.

```
|solver system dt beta gamma values stepper diag colors maxTime st legend|
dt := 0.001.
beta := 0.0052.
gamma := 52.
maxTime := 1.
system := ExplicitSystem block: [ :x :t| |c|
    c := Array new: 3.
    c at: 1 put: (beta negated) * (x at: 1) * (x at: 2).
    c at: 2 put: (beta * (x at: 1) * (x at: 2)) − (gamma * (x at: 2)).
    c at: 3 put: gamma * (x at: 2).
    c
    ].
stepper := RungeKuttaStepper onSystem: system.
solver := (ExplicitSolver new) stepper: stepper; system: system; dt: dt.
```

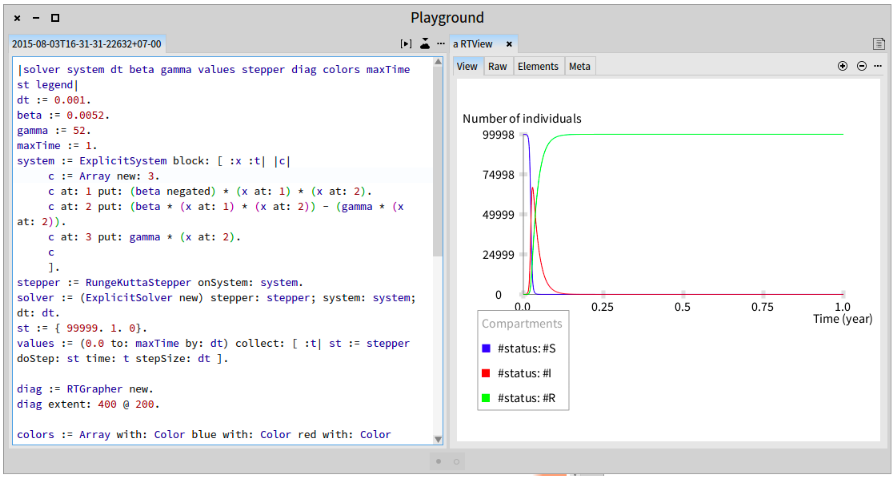
```
st := { 99999. 1. 0}.
values := (0.0 to: maxTime by: dt) collect: [ :t| st := stepper doStep: st time: t
      stepSize: dt ].

diag := RTGrapher new.
diag extent: 400 @ 200.

colors := Array with: Color blue with: Color red with: Color green.
1 to: 3 do: [ :i|
   |ds|
   ds := RTDataSet new.
   ds points: (1.0 to: ((maxTime/dt)+1) by: 1).
   ds y: [ :x| (values at: x) at: i ].
   ds x: [ :t| (t − 1)∗dt].
   ds noDot.
   ds connectColor: (colors at: i).
   diag add: ds ].
diag axisX title: 'Time (year)'.
diag axisY title: 'Number of individuals'.
diag axisY noDecimal.
legend := RTLegendBuilder new.
legend view: diag view.
legend addText: 'Compartments'.
legend addColor: (colors at: 1) text: '#status: #S'.
legend addColor: (colors at: 2) text: '#status: #I'.
legend addColor: (colors at: 3) text: '#status: #R'.
legend build.
diag build.
diag view @ RTZoomableView.
^ diag view
```

Executing this script we obtain the results of the system of equations as in Figure 2.2

Kendrick code

We use now the Kendrick DSL to express the SIR model. We start to create an instance of KEModel and then enumerate the compartment names with their initial value. In this model, we have 3 compartments S, I and R. The population has one attribute *status*. There is at least one infected in order to start the process. Two transitions are added to the model, one from S to I and another one from I to R. The parameters of the model $\beta = 0.0052$ and $\gamma = 52$.

```
| model |
   model := KEModel new.
 model population attributes: '{ #status: [#S, #I, #R] }'.
```

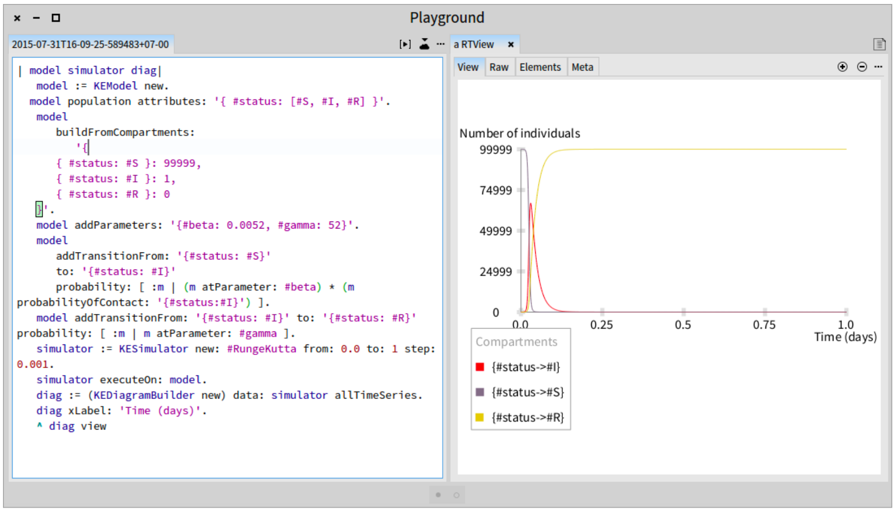


```
|solver system dt beta gamma values stepper diag colors maxTime
st legend|
dt := 0.001.
beta := 0.0052.
gamma := 52.
maxTime := 1.
system := ExplicitSystem block: [ :x :t| |c|
     c := Array new: 3.
     c at: 1 put: (beta negated) * (x at: 1) * (x at: 2).
     c at: 2 put: (beta * (x at: 1) * (x at: 2)) - (gamma * (x
at: 2)).
     c at: 3 put: gamma * (x at: 2).
     c
     ].
stepper := RungeKuttaStepper onSystem: system.
solver := (ExplicitSolver new) stepper: stepper; system: system;
dt: dt.
st := { 99999. 1. 0}.
values := (0.0 to: maxTime by: dt) collect: [ :t| st := stepper
doStep: st time: t stepSize: dt ].

diag := RTGrapher new.
diag extent: 400 @ 200.

colors := Array with: Color blue with: Color red with: Color
```

Figure 2.2: Deterministic dynamics of the SIR model using Smalltalk.

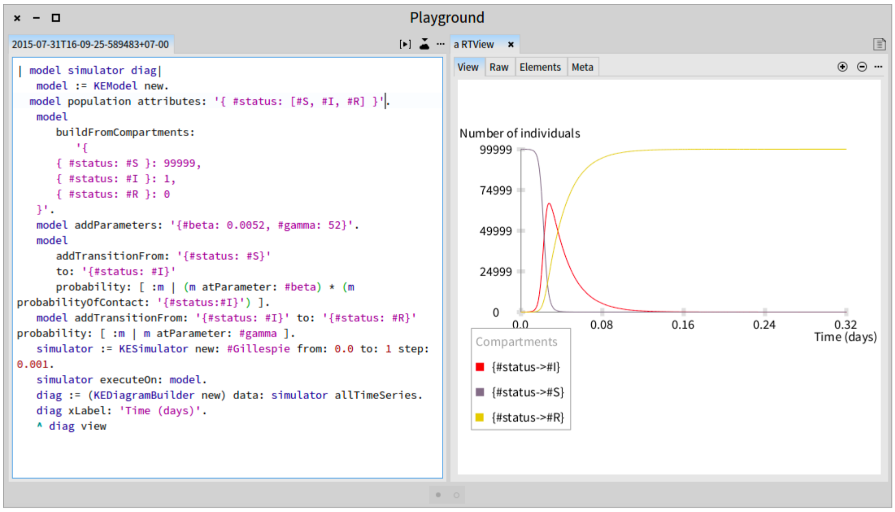
```
model
    buildFromCompartments:
        '{
    { #status: #S }: 99999,
    { #status: #I }: 1,
    { #status: #R }: 0
}'.
model addParameters: '{#beta: 0.0052, #gamma: 52}'.
model
    addTransitionFrom: '{#status: #S}'
    to: '{#status: #I}'
    probability: [ :m | (m atParameter: #beta) * (m probabilityOfContact: '{#status:#I
    }') ].
model addTransitionFrom: '{#status: #I}' to: '{#status: #R}' probability: [ :m | m
    atParameter: #gamma ].
```

Paste this script in the Workspace tool, define the simulations on this model, we obtain the results as in Figure 2.3. We can find that, the two scripts returns identical results. As a DSL for epidemiology, using Kendrick is more simple. Change the algorithm of simulation from *RungeKutta* to *Gillespie* and *IBM* to investigate two other formalisms (stochastic and agent-based). The results of these simulation are shown in Figure 2.4 and Figure 2.5.



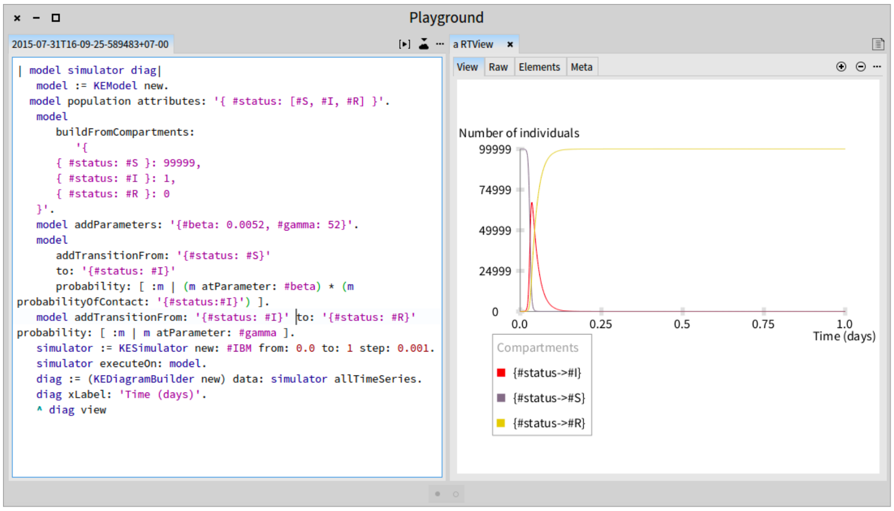
```
| model simulator diag|
  model := KEModel new.
 model population attributes: '{ #status: [#S, #I, #R] }'.
  model
     buildFromCompartments:
        '{
     { #status: #S }: 99999,
     { #status: #I }: 1,
     { #status: #R }: 0
  }'.
  model addParameters: '{#beta: 0.0052, #gamma: 52}'.
  model
     addTransitionFrom: '{#status: #S}'
     to: '{#status: #I}'
     probability: [ :m | (m atParameter: #beta) * (m
probabilityOfContact: '{#status:#I}') ].
  model addTransitionFrom: '{#status: #I}' to: '{#status: #R}'
probability: [ :m | m atParameter: #gamma ].
  simulator := KESimulator new: #RungeKutta from: 0.0 to: 1 step:
0.001.
  simulator executeOn: model.
  diag := (KEDiagramBuilder new) data: simulator allTimeSeries.
  diag xLabel: 'Time (days)'.
  ^ diag view
```

Figure 2.3: Deterministic dynamics of the SIR model without demography.



```
| model simulator diag|
  model := KEModel new.
 model population attributes: '{ #status: [#S, #I, #R] }'.
  model
     buildFromCompartments:
        '{
     { #status: #S }: 99999,
     { #status: #I }: 1,
     { #status: #R }: 0
  }'.
  model addParameters: '{#beta: 0.0052, #gamma: 52}'.
  model
     addTransitionFrom: '{#status: #S}'
     to: '{#status: #I}'
     probability: [ :m | (m atParameter: #beta) * (m
probabilityOfContact: '{#status:#I}') ].
  model addTransitionFrom: '{#status: #I}' to: '{#status: #R}'
probability: [ :m | m atParameter: #gamma ].
  simulator := KESimulator new: #Gillespie from: 0.0 to: 1 step:
0.001.
  simulator executeOn: model.
  diag := (KEDiagramBuilder new) data: simulator allTimeSeries.
  diag xLabel: 'Time (days)'.
  ^ diag view
```

Figure 2.4: Using Gillespie's direct algorithm to study the dynamics of model.



```
| model simulator diag|
  model := KEModel new.
 model population attributes: '{ #status: [#S, #I, #R] }'.
  model
     buildFromCompartments:
        '{
     { #status: #S }: 99999,
     { #status: #I }: 1,
     { #status: #R }: 0
  }'.
  model addParameters: '{#beta: 0.0052, #gamma: 52}'.
  model
     addTransitionFrom: '{#status: #S}'
     to: '{#status: #I}'
     probability: [ :m | (m atParameter: #beta) * (m
probabilityOfContact: '{#status:#I}') ].
  model addTransitionFrom: '{#status: #I}' to: '{#status: #R}'
probability: [ :m | m atParameter: #gamma ].
  simulator := KESimulator new: #IBM from: 0.0 to: 1 step: 0.001.
  simulator executeOn: model.
  diag := (KEDiagramBuilder new) data: simulator allTimeSeries.
  diag xLabel: 'Time (days)'.
  ^ diag view
```

Figure 2.5: Agent-based approach to investigate the model at individual level.

2.2 SIR model with births and deaths

Equations

$$\frac{dS}{dt} = \mu - \beta * S * I - \mu * S \tag{2.4}$$

$$\frac{dI}{dt} = \beta * S * I - \gamma * I - \mu * I \tag{2.5}$$

$$\frac{dR}{dt} = \gamma * I - \mu * R \tag{2.6}$$

Pharo code

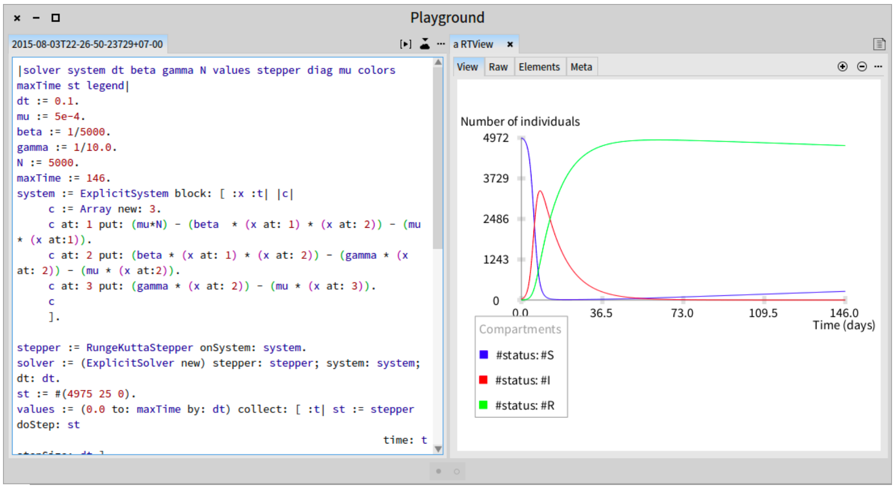
```
|solver system dt beta gamma N values stepper diag mu colors maxTime st legend|
dt := 0.1.
mu := 5e−4.
beta := 1/5000.
gamma := 1/10.0.
N := 5000.
maxTime := 146.
system := ExplicitSystem block: [ :x :t| |c|
    c := Array new: 3.
    c at: 1 put: (mu*N) − (beta  * (x at: 1) * (x at: 2)) − (mu * (x at:1)).
    c at: 2 put: (beta * (x at: 1) * (x at: 2)) − (gamma * (x at: 2)) − (mu * (x at:2)).
    c at: 3 put: (gamma * (x at: 2)) − (mu * (x at: 3)).
    c
    ].

stepper := RungeKuttaStepper onSystem: system.
solver := (ExplicitSolver new) stepper: stepper; system: system; dt: dt.
st := #(4975 25 0).
values := (0.0 to: maxTime by: dt) collect: [ :t| st := stepper doStep: st
                                                time: t stepSize: dt ].
diag := RTGrapher new.
diag extent: 400 @ 200.

colors := Array with: Color blue with: Color red with: Color green.
1 to: 3 do: [ :i|
   |ds|
   ds := RTDataSet new.
   ds points: (1.0 to: ((maxTime/dt)+1) by: 1).
   ds y: [ :x| (values at: x) at: i ].
   ds x: [ :t| (t − 1)*dt].
   ds noDot.
   ds connectColor: (colors at: i).
   diag add: ds ].
diag axisX title: 'Time (days)'.
diag axisY title: 'Number of individuals'.
diag axisY noDecimal.
legend := RTLegendBuilder new.
legend view: diag view.
legend addText: 'Compartments'.
legend addColor: (colors at: 1) text: '#status: #S'.
legend addColor: (colors at: 2) text: '#status: #I'.
legend addColor: (colors at: 3) text: '#status: #R'.
legend build.
diag build.
diag view @ RTZoomableView.
^ diag view
```

Running this script will give the results as can be seen in Figure 2.6

Kendrick code

Comparing to the previous model, in this model, it should add four other transitions. The first one represents the births of susceptible. The three others represent the deaths of each compartment. We use the ODE syntax to specify this model.

```
| model |
   model := KEModel new.
```



```
|solver system dt beta gamma N values stepper diag mu colors
maxTime st legend|
dt := 0.1.
mu := 5e-4.
beta := 1/5000.
gamma := 1/10.0.
N := 5000.
maxTime := 146.
system := ExplicitSystem block: [ :x :t| |c|
     c := Array new: 3.
     c at: 1 put: (mu*N) - (beta  * (x at: 1) * (x at: 2)) - (mu
* (x at:1)).
     c at: 2 put: (beta * (x at: 1) * (x at: 2)) - (gamma * (x
at: 2)) - (mu * (x at:2)).
     c at: 3 put: (gamma * (x at: 2)) - (mu * (x at: 3)).
     c
     ].

stepper := RungeKuttaStepper onSystem: system.
solver := (ExplicitSolver new) stepper: stepper; system: system;
dt: dt.
st := #(4975 25 0).
values := (0.0 to: maxTime by: dt) collect: [ :t| st := stepper
doStep: st
                                                         time: t
```

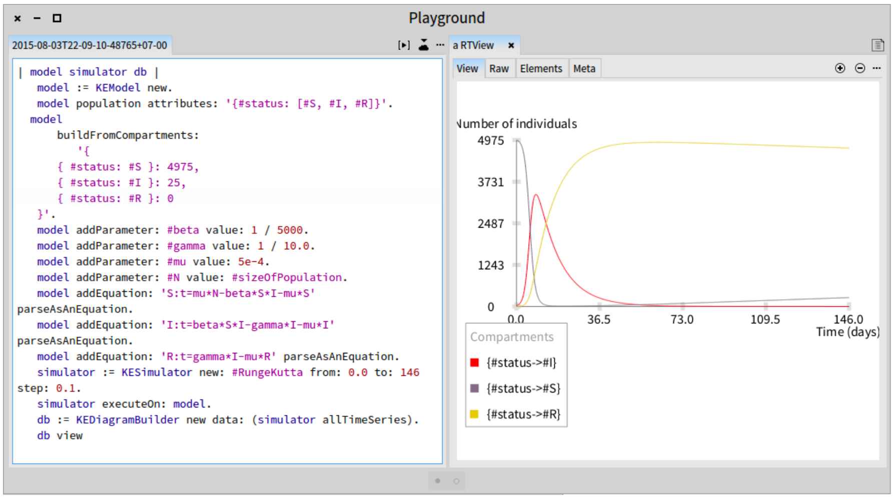
Figure 2.6: Resolving the system of equations of the SIR model with demography.

```
  model population attributes: '{#status: [#S, #I, #R]}'.
 model
       buildFromCompartments:
           '{
       { #status: #S }: 4975,
       { #status: #I }: 25,
       { #status: #R }: 0
  }'.
  model addParameter: #beta value: 1 / 5000.
  model addParameter: #gamma value: 1 / 10.0.
  model addParameter: #mu value: 5e−4.
  model addParameter: #N value: #sizeOfPopulation.
  model addEquation: 'S:t=mu*N−beta*S*I−mu*S' parseAsAnEquation.
  model addEquation: 'I:t=beta*S*I−gamma*I−mu*I' parseAsAnEquation.
  model addEquation: 'R:t=gamma*I−mu*R' parseAsAnEquation.
```

Using deterministic simulation on this model we obtain the result as can be seen in Figure 2.7

2.3 SEIR model with births and deaths

We introduce here a SEIR model. The E status means that a susceptible becomes infected but not yet infectious.



```
| model simulator db |
   model := KEModel new.
   model population attributes: '{#status: [#S, #I, #R]}'.
  model
      buildFromCompartments:
         '{
      { #status: #S }: 4975,
      { #status: #I }: 25,
      { #status: #R }: 0
   }'.
   model addParameter: #beta value: 1 / 5000.
   model addParameter: #gamma value: 1 / 10.0.
   model addParameter: #mu value: 5e-4.
   model addParameter: #N value: #sizeOfPopulation.
   model addEquation: 'S:t=mu*N-beta*S*I-mu*S'
parseAsAnEquation.
   model addEquation: 'I:t=beta*S*I-gamma*I-mu*I'
parseAsAnEquation.
   model addEquation: 'R:t=gamma*I-mu*R' parseAsAnEquation.
   simulator := KESimulator new: #RungeKutta from: 0.0 to: 146
step: 0.1.
   simulator executeOn: model.
   db := KEDiagramBuilder new data: (simulator allTimeSeries).
   db view
```

Figure 2.7: Deterministic dynamics of the SIR model with demography using Kendrick language.

Equations

$$\frac{dS}{dt} = \mu * N - \beta * S * I - \mu * S \tag{2.7}$$

$$\frac{dE}{dt} = \beta * S * I - \sigma * E - \mu * E \tag{2.8}$$

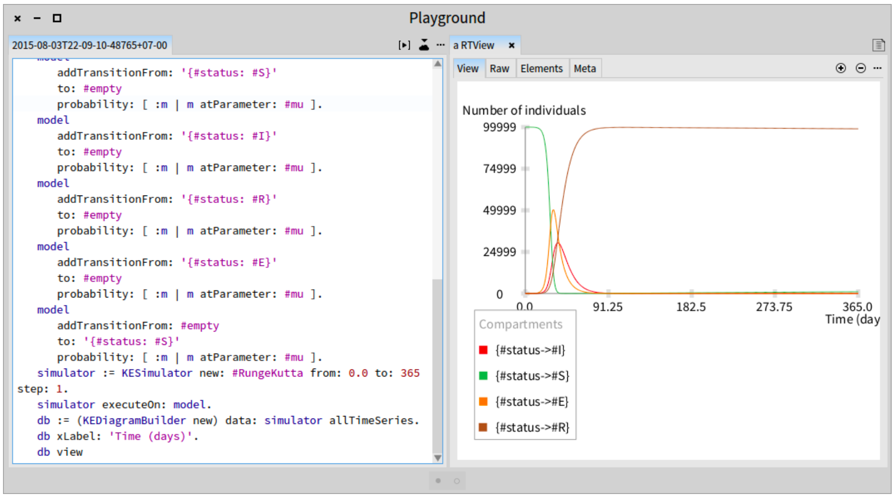
$$\frac{dI}{dt} = \sigma * E - \gamma * I - \mu * I \tag{2.9}$$

$$\frac{dR}{dt} = \gamma * I - \mu * R \tag{2.10}$$

Kendrick code

Here, we use the parameters of measles model. The time unit is day.

```
| model |
  model := KEModel new.
  model population attributes: '{#status: [#S, #E, #I, #R]}'.
  model
     buildFromCompartments:
        '{
     {#status: #S}: 99999,
     {#status: #I}: 1,
     {#status: #E}: 0,
     {#status: #R}: 0
  }'.
  model addParameters: '{
     #beta: 0.0000214,
     #gamma: 0.143,
     #mu: 0.0000351,
     #sigma: 0.125,
     #N: #sizeOfPopulation}'.
  model
     addTransitionFrom: '{#status: #S}'
     to: '{#status: #E}'
     probability: [ :m | (m atParameter: #beta) * (m probabilityOfContact: '{#status:#I
     }') ].
  model
     addTransitionFrom: '{#status: #E}'
     to: '{#status: #I}'
     probability: [ :m | m atParameter: #sigma ].
  model
     addTransitionFrom: '{#status: #I}'
     to: '{#status: #R}'
     probability: [ :m | m atParameter: #gamma ].
  model
     addTransitionFrom: '{#status: #S}'
     to: #empty
     probability: [ :m | m atParameter: #mu ].
  model
     addTransitionFrom: '{#status: #I}'
     to: #empty
     probability: [ :m | m atParameter: #mu ].
  model
     addTransitionFrom: '{#status: #R}'
     to: #empty
     probability: [ :m | m atParameter: #mu ].
  model
     addTransitionFrom: '{#status: #E}'
     to: #empty
     probability: [ :m | m atParameter: #mu ].
  model
     addTransitionFrom: #empty
     to: '{#status: #S}'
     probability: [ :m | m atParameter: #mu ].
```



```
        addTransitionFrom: '{#status: #S}'
        to: #empty
        probability: [ :m | m atParameter: #mu ].
    model
        addTransitionFrom: '{#status: #I}'
        to: #empty
        probability: [ :m | m atParameter: #mu ].
    model
        addTransitionFrom: '{#status: #R}'
        to: #empty
        probability: [ :m | m atParameter: #mu ].
    model
        addTransitionFrom: '{#status: #E}'
        to: #empty
        probability: [ :m | m atParameter: #mu ].
    model
        addTransitionFrom: #empty
        to: '{#status: #S}'
        probability: [ :m | m atParameter: #mu ].
    simulator := KESimulator new: #RungeKutta from: 0.0 to: 365
step: 1.
    simulator executeOn: model.
    db := (KEDiagramBuilder new) data: simulator allTimeSeries.
    db xLabel: 'Time (days)'.
    db view
```

Figure 2.8: Deterministic dynamics of the measles model using Kendrick language.

2.4 SEIR model with vaccination at births

Equations

$$\frac{dS}{dt} = \mu * N * (1 - p) - \beta * S * I - \mu * S \tag{2.11}$$

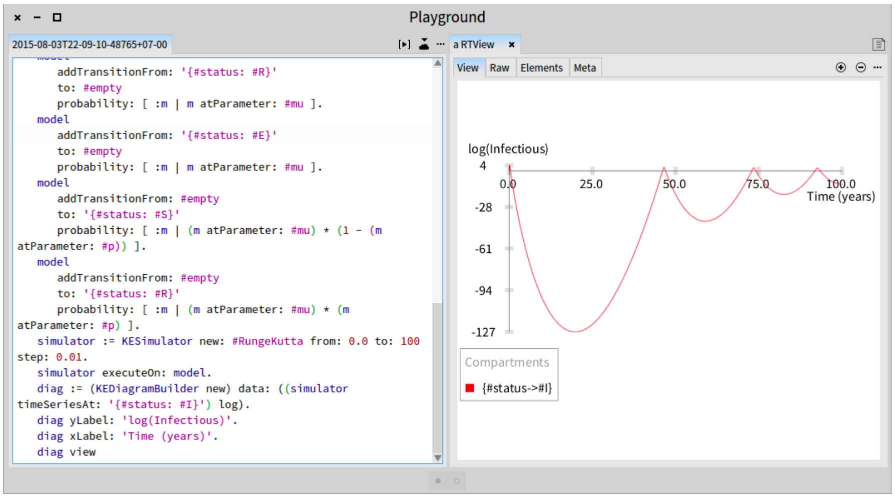
$$\frac{dE}{dt} = \beta * S * I - \sigma * E - \mu * E \tag{2.12}$$

$$\frac{dI}{dt} = \sigma * E - \gamma * I - \mu * I \tag{2.13}$$

$$\frac{dR}{dt} = \mu * N * P + \gamma * I - \mu * R \tag{2.14}$$

Kendrick code

```
| model |
   model := KEModel new.
   model population attributes: '{#status: [#S, #E, #I, #R]}'.
   model
      buildFromCompartments:
         '{
      {#status: #S}: 99999,
      {#status: #I}: 1,
      {#status: #E}: 0,
      {#status: #R}: 0
   }'.
   model addParameters: '{
      #beta: 0.00782,
      #gamma: 52.14,
      #mu: 0.0128,
      #sigma: 45.625,
      #N: #sizeOfPopulation,
      #p: 0.7}'.
   model
      addTransitionFrom: '{#status: #S}'
      to: '{#status: #E}'
      probability: [ :m | (m atParameter: #beta) * (m probabilityOfContact: '{#status:#I
      }') ].
   model
      addTransitionFrom: '{#status: #E}'
      to: '{#status: #I}'
      probability: [ :m | m atParameter: #sigma ].
   model
      addTransitionFrom: '{#status: #I}'
      to: '{#status: #R}'
      probability: [ :m | m atParameter: #gamma ].
   model
      addTransitionFrom: '{#status: #S}'
      to: #empty
      probability: [ :m | m atParameter: #mu ].
   model
      addTransitionFrom: '{#status: #I}'
      to: #empty
      probability: [ :m | m atParameter: #mu ].
   model
      addTransitionFrom: '{#status: #R}'
      to: #empty
      probability: [ :m | m atParameter: #mu ].
   model
      addTransitionFrom: '{#status: #E}'
      to: #empty
      probability: [ :m | m atParameter: #mu ].
   model
      addTransitionFrom: #empty
      to: '{#status: #S}'
      probability: [ :m | (m atParameter: #mu) * (1 − (m atParameter: #p)) ].
   model
      addTransitionFrom: #empty
      to: '{#status: #R}'
      probability: [ :m | (m atParameter: #mu) * (m atParameter: #p) ].
```



```
      addTransitionFrom: '{#status: #R}'
      to: #empty
      probability: [ :m | m atParameter: #mu ].
   model
      addTransitionFrom: '{#status: #E}'
      to: #empty
      probability: [ :m | m atParameter: #mu ].
   model
      addTransitionFrom: #empty
      to: '{#status: #S}'
      probability: [ :m | (m atParameter: #mu) * (1 - (m
atParameter: #p)) ].
   model
      addTransitionFrom: #empty
      to: '{#status: #R}'
      probability: [ :m | (m atParameter: #mu) * (m
atParameter: #p) ].
   simulator := KESimulator new: #RungeKutta from: 0.0 to: 100
step: 0.01.
   simulator executeOn: model.
   diag := (KEDiagramBuilder new) data: ((simulator
timeSeriesAt: '{#status: #I}') log).
   diag yLabel: 'log(Infectious)'.
   diag xLabel: 'Time (years)'.
   diag view
```

Figure 2.9: Deterministic dynamics of measles model with vaccination at birth (p=0.7).

Here we take the vaccination rate $p = 0.7$. In order to study the vaccination effect, we change the value of p and compare the two case as in Figure 2.10. After running a simulation on the first model (with p = 0.7), we reset the model with the command:

```
model resetCompartments
```

The script to show this comparison can be found in the Examples package of Kendrick. The parameters of Kendrick model is not only a constant but also a temporal function as in this model. The Y-axis is loged for readibilty.

2.5 SEIR model with seasonal forcing

Equations

$$\frac{dS}{dt} = \mu * N - \beta_0 * (1 + \beta_{amp} * cos(t)) * S * I - \mu * S \tag{2.15}$$

$$\frac{dE}{dt} = \beta_0 * (1 + \beta_{amp} * cos(t)) * S * I - \sigma * E - \mu * E \tag{2.16}$$

$$\frac{dI}{dt} = \sigma * E - \gamma * I - \mu * I \tag{2.17}$$

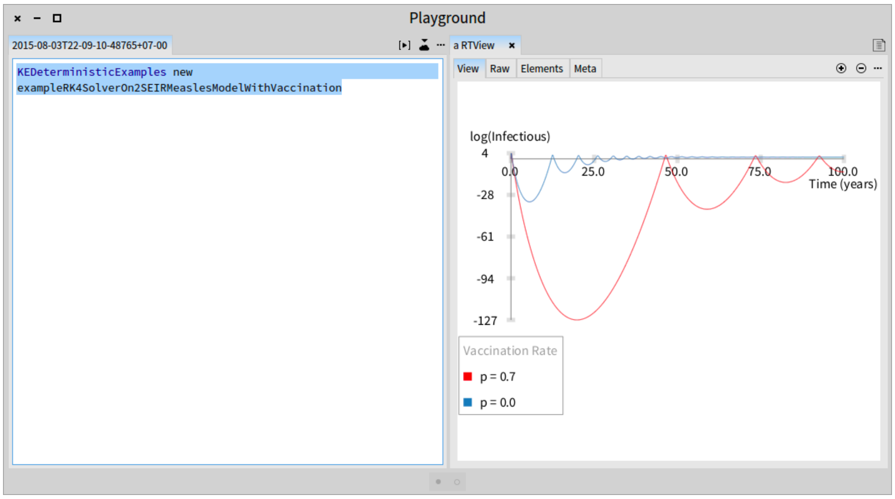
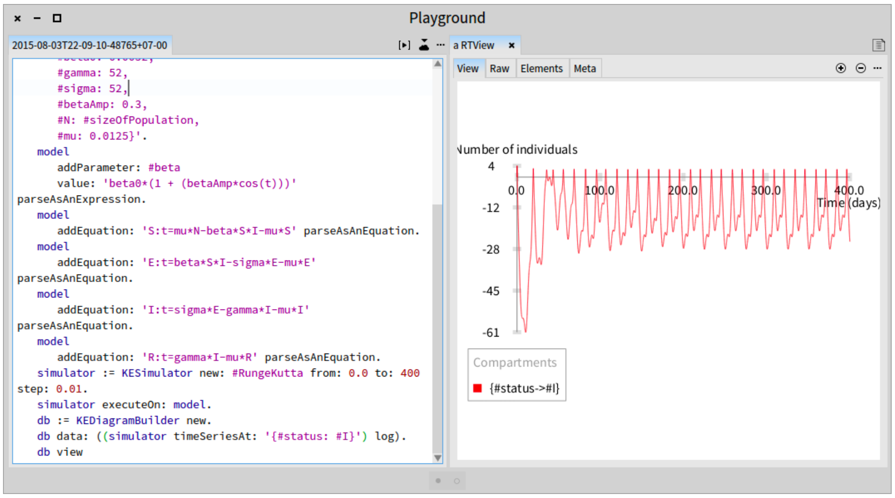

```
KEDeterministicExamples new
exampleRK4SolverOn2SEIRMeaslesModelWithVaccination
```

Figure 2.10: Comparison two models with different values of p.

$$\frac{dR}{dt} = \gamma * I - \mu * R \tag{2.18}$$

Kendrick code

```
| model |
model := KEModel new.
model population attributes: '{ #status: [#S, #E, #I, #R] }'.
model
    buildFromCompartments:
        '{
    { #status: #S }: 99999,
    { #status: #E }: 0,
    { #status: #I }: 1,
    { #status: #R }: 0
}'.
model addParameters: '{
    #beta0: 0.0052,
    #gamma: 52,
    #sigma: 52,
    #betaAmp: 0.3,
    #N: #sizeOfPopulation,
    #mu: 0.0125}'.
model
    addParameter: #beta
    value: 'beta0*(1 + (betaAmp*cos(t)))' parseAsAnExpression.
```



```
      #gamma: 52,
      #sigma: 52,
      #betaAmp: 0.3,
      #N: #sizeOfPopulation,
      #mu: 0.0125}'.
   model
      addParameter: #beta
      value: 'beta0*(1 + (betaAmp*cos(t)))'
parseAsAnExpression.
   model
      addEquation: 'S:t=mu*N-beta*S*I-mu*S' parseAsAnEquation.
   model
      addEquation: 'E:t=beta*S*I-sigma*E-mu*E'
parseAsAnEquation.
   model
      addEquation: 'I:t=sigma*E-gamma*I-mu*I'
parseAsAnEquation.
   model
      addEquation: 'R:t=gamma*I-mu*R' parseAsAnEquation.
   simulator := KESimulator new: #RungeKutta from: 0.0 to: 400
step: 0.01.
   simulator executeOn: model.
   db := KEDiagramBuilder new.
   db data: ((simulator timeSeriesAt: '{#status: #I}') log).
   db view
```

Figure 2.11: Deterministic dynamics of SEIR model with seasonal forcing.

```
model
   addEquation: 'S:t=mu*N−beta*S*I−mu*S' parseAsAnEquation.
model
   addEquation: 'E:t=beta*S*I−sigma*E−mu*E' parseAsAnEquation.
model
   addEquation: 'I:t=sigma*E−gamma*I−mu*I' parseAsAnEquation.
model
   addEquation: 'R:t=gamma*I−mu*R' parseAsAnEquation.
```

Chapter 3

Heterogeneous Host Models

3.1 SIS model with multiple risk groups

We consider here the the SIS model in which the population is structured into multiple risk groups (host-heterogeneous model) labelled 1, 2, ... The group labelled 1 is the lowest risk. The group labelled 5 is the highest risk.

Equations

$$\frac{dS_i}{dt} = \gamma_i I_i - \sum_j \beta_{ij} S_i I_j \tag{3.1}$$

$$\frac{dI_i}{dt} = \sum_j \beta_{ij} S_i I_j - \gamma_i I_i \tag{3.2}$$

Kendrick model

```
| model graph |
   model := KEModel new.
   model population: KEMetaPopulation new.
   model population attributes: {#riskGroup−>(1 to: 5). #status−>#(S I)}.
   model
      buildFromAttributes: #(#status #riskGroup)
      compartments: {
        (#(#S 1) −> 6000). (#(#I 1) −> 0).
      (#(#S 2) −> 31000). (#(#I 2) −> 0).
      (#(#S 3) −> 52000). (#(#I 3) −> 0).
      (#(#S 4) −> 8000). (#(#I 4) −> 0).
      (#(#S 5) −> 2999). (#(#I 5) −> 1).
```

```
    }.
model addParameter: #beta value: 16e−9.
model addParameter: #gamma value: 0.2.

graph := KEContactNetwork
                newOn: model population
                atAttribute: #riskGroup.
graph edges: { 2−>2. 2−>3. 2−>4. 2−>5. 3−>3. 3−>4. 3−>5. 4−>4. 4−>5. 5−>5
    };
        strengthOfConnections: #(9 30 180 300 100 600 1000 3600 6000 10000).

model addEquation: 'S:t=gamma*I−beta*S*I' parseAsAnEquation.
model addEquation: 'I:t=beta*S*I−gamma*I' parseAsAnEquation.
```

Chapter 4

Multi-Pathogen/Multi-Host Models

4.1 SIR model with three species of hosts

In the standard models of epidemiology, the population is compartmentalized by only clinic status. As such, the population has only one degree of subdivision. In a context of multi-host (multi-species) model, the host population has two degrees of subdivision due to the attribute species of each individual.

Equations

$$\frac{dS}{dt} = \mu_i * N_i - \sum_j \beta_{ij} * S_i * I_j - \mu_i * S_i \tag{4.1}$$

$$\frac{dI}{dt} = \sum_j \beta_{ij} * S_i * I_j - \gamma_i * I_i - \mu_i * I_i \tag{4.2}$$

$$\frac{dR}{dt} = \gamma_i * I_i - \mu_i * R_i \tag{4.3}$$

Configurations of Kendrick model

In epidemiology, it is important to distinguish between two basic assumptions in terms of the underlying structure of contacts within the population. Either the model is assumed to be mass action or pseudo mass action. The first kind reflects the situation where the number of contacts is independent of the population size. So that the force of infection $\lambda = \beta I/N$. In some circumstances, the transmission rate β is rescaled by N. The second one assumes that as the population size increases, so does the contact rate. As such the force of infection $\lambda = \beta I$.

At the moment, Kendrick model includes three parameters of configuration: sizeOfPopulation, rescale, mass_action. By default:

```
#sizeOfPopulation->#population
#rescale->true
#mass_action->true
```

In the context of the multi-species model, it is important to config the size of population for each species. As such:

```
model configurations: {#sizeOfPopulation->#(#species)}
```

Kendrick model

In this model, we define the parameter μ for three scopes corresponding to each species. In order to represent the interaction between three species, we define a contact network. Due to this network, the force of infection will be modified as:

$$\lambda = \beta * \sum_j \rho_{ij} * I_j \tag{4.4}$$

where ρ_{ij} denotes the strength of connection between species i and j.

```
| model graph |
   model := KEModel new.
   model
      population:
         (KEMetaPopulation new
            attributes:
               {(#status -> #(#S #I #R)).
               (#species -> #(#mosquito #reservoir1 #reservoir2))}).
   model
      buildFromAttributes: #(#status #species)
      compartments:
         {(#(#S #mosquito) -> 9800).
         (#(#I #mosquito) -> 200).
         (#(#R #mosquito) -> 0).
         (#(#S #reservoir1) -> 1000).
         (#(#I #reservoir1) -> 0).
         (#(#R #reservoir1) -> 0).
         (#(#S #reservoir2) -> 2000).
```

```
        (#(#I #reservoir2) −> 0).
        (#(#R #reservoir2) −> 0)}.
model addParameter: #mu
      inScopes: {
          #species−>#mosquito.
          #species−>#reservoir1.
          #species−>#reservoir2}
      values: #(12.17 0.05 0.05).
model addParameter: #gamma value: 52.
model addParameter: #beta value: 1.
model addParameter: #N value: #sizeOfPopulation.
model configurations: { #sizeOfPopulation−>#(#species) }.

graph := KEContactNetwork
        newOn: model population
        atAttribute: #species.
graph edges: { #mosquito−>#reservoir1. #mosquito−>#reservoir2 };
        strengthOfAllConnections: 0.02.
model
    addTransitionFrom: '{#status: #S}'
    to: '{#status: #I}'
    probability: [ :m | (m atParameter: #beta) * (m probabilityOfContact: '{#status:
    #I}') ].
model
    addTransitionFrom: '{#status: #I}'
    to: '{#status: #R}'
    probability: [ :m | m atParameter: #gamma ].
model
    addTransitionFrom: '{#status: #S}'
    to: #empty
    probability: [ :m | m atParameter: #mu ].
model
    addTransitionFrom: '{#status: #I}'
    to: #empty
    probability: [ :m | m atParameter: #mu ].
model
    addTransitionFrom: '{#status: #R}'
    to: #empty
    probability: [ :m | m atParameter: #mu ].
model
    addTransitionFrom: #empty
    to: '{#status: #S}'
    probability: [ :m | m atParameter: #mu ].
```

Chapter 5

Spatial models

5.1 SEIR model with spatial dynamics

We investigate the impact of spatial effects. Considering a spatial model organised by n patches arranged in a ring. The individuals can move between two neighbouring patches. In each patch, we have a sub-population with four compartments S, E, I and R. To specify this model, we use the Migration Network built in Kendrick. Due to this network, the model will have migration transitions from one compartment to other.

Equations

$$\frac{dS_i}{dt} = \mu_i N_i - \beta_i S_i I_i - \mu_i S_i - \sum_j \rho_{ij} S_i + \sum_j \rho_{ji} S_j \tag{5.1}$$

$$\frac{dE_i}{dt} = \beta_i S_i I_i - \mu E_i - \varepsilon E_i - \sum_j \rho_{ij} E_i + \sum_j \rho_{ji} E_j \tag{5.2}$$

$$\frac{dI_i}{dt} = \varepsilon E_i - \mu I_i - \gamma_i I_i - \sum_j \rho_{ij} I_i + \sum_j \rho_{ji} I_j \tag{5.3}$$

$$\frac{dR_i}{dt} = \gamma_i I_i - \mu R_i - \sum_j \rho_{ij} R_i + \sum_j \rho_{ji} R_j \tag{5.4}$$

Kendrick model

```
| model graph |
model := KEModel new.
model population: KEMetaPopulation new.
model population attributes: {
   #patch->((1 to: 5)).
   #status->#(S E I R)}.
model
   buildFromAttributes: #(#status #patch)
   compartments: {
        (#(#S 1) -> 900). (#(#E 1) -> 0). (#(#I 1) -> 100). (#(#R 1) -> 0).
      (#(#S 2) -> 1000). (#(#E 2) -> 0). (#(#I 2) -> 0). (#(#R 2) -> 0).
      (#(#S 3) -> 1000). (#(#E 3) -> 0). (#(#I 3) -> 0). (#(#R 3) -> 0).
      (#(#S 4) -> 1000). (#(#E 4) -> 0). (#(#I 4) -> 0). (#(#R 4) -> 0).
      (#(#S 5) -> 1000). (#(#E 5) -> 0). (#(#I 5) -> 0). (#(#R 5) -> 0).
   }.
model
   addParameter: #beta
   inScopes: {
      (#patch->1).
      (#patch->2).
      (#patch->3).
      (#patch->4).
      (#patch->5)
   }
   values: #(0.75 0.5 0.5 0.5 0.5).
model addParameter: '{
   #v: 0.00274,
   #d: 0.0000365,
   #epsilon: 0.5,
   #gamma: 0.25,
   #N: #sizeOfPopulation}'.
model configurations: {
      #sizeOfPopulation->#(#patch).
      #rescale->false }.
graph := KEMigrationNetwork
            newOn: model population
            atAttribute: #patch
            topology: (KERandomSmallWorldNetwork new K: 2; beta: 0).
graph strengthOfAllConnections: 0.03.
graph addMigrationTransitionsTo: model.

model addEquation: 'S:t=d*N-d*S-beta*S*I+v*R' parseAsAnEquation.
model addEquation: 'E:t=beta*S*I-d*E-epsilon*E' parseAsAnEquation.
model addEquation: 'I:t=epsilon*E-d*I-gamma*I' parseAsAnEquation.
model addEquation: 'R:t=gamma*I-d*R-v*R' parseAsAnEquation.
```