EpiGrass User Guide
Release 1.4.1

Cláudia Torres Codeço Flávio Codeço Coelho

Acknowledgements

EpiGrass is funded by CNPq. Participants of this project are: Maria Goreti Freitas Rosas (Fiocruz - IOC); Alexios Zavras (Univ. of Athens); Pantelis Tsouris (Univ. of Athens); Oswaldo Gonçalves Cruz (Fiocruz - PROCC); Flavio Codeço Coelho (Fiocruz - PROCC); Cláudia Torres Codeço (Fiocruz-PROCC).

Preface

THE emergence and re-emergence of infectious diseases and the evolution of resistant pathogens pose a great challenge for Public Health worldwide. Brazil, in particular, is facing increasing problems with the expansion, to urban centers, of diseases previously rural (leishmaniosis), increasing mortality caused by previously benign diseases (dengue), potencial emergence of new diseases, and reduction of efficacy of traditional treatment protocols (tuberculosis, malaria, leprosy).

To deal with this situation, it is necessary to think strategically. How can we use our limited resources the best way possible? In other words, how to define public health policies for the use of chemicals and vaccines that optimize their impact in the short and long term? To develop such strategies, it is necessary to consider, and integrate, information from different sources, including biological information regarding host-parasite interaction, parasite response to chemicals or other control element, identification of risk situations for the population under study, identification of alternative control strategies, quantification of available resources, etc. The integration of all these data is often done in an incomplete way, resulting in sub-optimal decision making.

In EpiGrass, the georefered space is the background for simulating intervention scenarios. Computer simulations are more and more used for evaluations of risk and formulations of disease control strategies (Casman et al., 2000). These models are useful for determining, among other things, the expected number of cases in an epidemic (and the required medical cost), compare alternative control strategies (mass vaccination x localized vaccination, for example), for example. The integration of mathematical models to georefered data is strategic as it increases the applicability of mathematical models.

EpiGrass is a simulator which implements transmission models in a network where disease transmission occurs in the nodes (that may represent cities, neighborhoods or households) and spatial spread occurs via edges (that represent transportation routes or other elements of contact networks). Examples of application of this approach can be found in (Barrett et al., 2005;

Meyers et al., 2005).

EpiGrass is not a part of the GRASS GIS system (`http://grass.itc.it`), though it can read GRASS's ascii vector format to import maps which will serve as background for its graphical display of simulations. EpiGrass also makes use of the R statistical package. Statistical analysis in EpiGrass is done in R. The 'R data analysis programming language and environment' is an open source system for statistical computing and graphics (`http://www.r-project.org/`). R consists of a base package and a set of modules for data handling and storage, calculations on arrays, several methods for data analysis, graphical facilities. It also has a well-developed programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

EpiGrass is an opensource software. The financial support for EpiGrass comes from the Brazilian Research Council (CNPq), as part of a nation-wide initiative by the federal government `http://www.iti.br/`, to develop and use free/opensource software as the standard computational platform throughout the country.

EpiGrass is the product of a colaboration between researchers from many disciplines: Flávio Coelho, Cláudia Codeço and Oswaldo Gonçalves Cruz are epidemiological modellers at the Oswaldo Cruz Foundation (Rio de Janeiro - Brazil); Maria Goreti Rosa Freitas is an entomologist and epidemiologist working on spatial processes associated to Dengue Fever, at Oswaldo Cruz Foundation; Alexios Zavras is an expert in software development and an advocate of the Free Source Movement in Greece; Pantelis Tsouris is an engineer specialized in hardware development. We hope that, with this first release, this small community will grow and prosper.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Overview of the EpiGrass

EpiGrass is a platform for network epidemiological simulation and analysis. It enables researchers to perform comprehensive spatio-temporal simulations incorporating epidemiological data and models for disease transmission and control in order to create sophisticated scenario analyses.

1.1 Components

The EpiGrass system is composed of 4 components, The EpiGrass simulator, the EpiGrass database, EpiGrass visualization module, and the report generator. They are all accessed via EpiGrass' graphical user interface (GUI).

1.2 Modeling Approach

The geographical networks over which epidemiological processes take place can be very straightforwardly represented in a object-oriented framework. In such a framework, the nodes and edges of the geographical networks are objects with their own attributes and methods.

Once the archetypal node and edge objects are defined with appropriate attributes and methods, then a code representation of the real system can be constructed, where cities (or other geographical localities) and transportation routes are instances of the node and edge objects, respectively. The whole network is also an object with a whole collection of attributes and methods.

This framework leads to a compact and hierarchical computational model consisting of a network object containing a variable number of node and edge objects. This framework also do not pose limitations to encapsulation, potentially allowing for networks within networks if desirable (not yet implemented).

For the end user this framework is transparent since it mimics the natural structure of the real system. Even after the model is converted into a code object, all of its component objects remain accessible internally, facilitating the exchange of information between all levels of the model.

1.3 Geographical Network Models

EpiGrass' geo-referenced models are built from two basic sources of data: Two comma-separated-values files containing information about nodes and edges, which are provided by the user. This files represent the network of interest.

If the user has a vectorial representation (map) for the geographical area where the network is located, this file can be passed to visualization module to be used as a backdrop to the network dynamics animation.

1.3.1 Defining Nodes

A graph has nodes and edges. Nodes can be cities or other localities depending on the model being constructed. The definition of nodes require the setting of several attributes as listed below. besides, these nodes will have many more attributes defined at run-time which will depend on other aspects of the model. These will be discussed later.

The data required, at build time, to create nodes come from a CSV (comma-separated-values) ASCII-text file, with the following attributes, (in this order. See demos):

Latitude, Longitude This attribute will be used to geo reference the node. The coordinate system must match those used in the cartographic base imported from GRASS. Coordinates can be coded in either decimal or sexagesimal format.

Name Used for identification purposes only. It can be a city name, for instance.

Population Since the simulation models will all be of a populational nature. Population size must be specified at build time.

Geocode A Unique Geocode (a integer number) is required. It will be used as a label/index to facilitate reference to specific nodes.

These files can be easily put together in a spreadsheet and then saved in CSV format.

1.3.2 Defining Edges

In EpiGrass' graphs, edges represent transportation routes. Similarly to nodes, edges are defined at build-time with a minimum set of attributes which will be extended at run-time. EpiGrass also expects to get these attributes from a CSV file (See demos for format details):

Source The name of the source node. The edges are bi-directional, but the nodes are labeled source and destination for reference purposes.

Destination The name of the destination node.

Forward migration Migration rate from source to destination, in number of persons per unit of time.

Backward migration Migration rate from destination to source, in number of persons per unit of time.

Length distance in kilometers (or another unit) from source to destination via the particular route (not straight line distance).

Source's geocode This is the unique numerical identifier used in the sites file.

Destination's geocode This is the unique numerical identifier used in the sites file.

1.3.3 Defining models

The word model in EpiGrass can mean two distinct objects: The network model and the node's epidemic model.

Node objects, in an EpiGrass model, contain well-mixed population dynamic models within them. These models determine the dynamics of epidemics within the particular environments of each node. EpiGrass comes with a few standard epidemiological models to choose from[1] when setting up your network. Currently, Epigrass comes with a small set of predefined model types but users can easily define their own. The building of custom models is discussed in detail in chapter 4.

Network models are specified in a ASCII-text script file (see appendix A). EpiGrass comes with a few demo network models for the users to play with until they are confident enough to build their own. Even then, it is advisable to use the demo scripts provided as templates to minimize syntax errors.

[1]See chapter 2

The script on the appendix A specifies a network model with an stochastic SEIR (see chapter 2) epidemic model in its nodes. The user should study this model and play with its parameters to understand the features of EpiGrass. A step-by-step tutorial on how to edit the model script can be found on chapter 4.

1.4 The Simulation

A simulation run in EpiGrass consists of a series of tasks performed at each time step[2].

Calculate migration For all edges in the network, the number of persons traveling each way is determined for the current time-step.

Run epidemic models For each node in the network the epidemic demographics are updated based on the local number of infected and susceptible individuals which have been updated by the transportation system.

All aspects of the simulation such as number of passengers traveling on each edge, number of infected/susceptible on each node and etc., are recorded in a step-by-step basis. This complete record allows for the model to be analyzed after the simulation has been completed without having to recalculate it.

1.4.1 Output

The output of a simulation in EpiGrass consists of several csv formatted files with some global results from the simulation, a (optional) written PDF report, and two database tables containing the simulated time series of many variables for each node and edge.

Graphical display

After the simulation is completed, user-selected epidemiological variables can be animated in a 3-dimensional rendering over the map of the region containing the network (if one is available).

[2]The number of time steps is defined in the model script

Report Generation

The report contains a detailed analysis of the network model and the simulations ran with it. The report generates a LaTeXsource file and compiles it to a PDF document for visualization.

Three types of report are currently available:

Report = 1 Returns a set of descriptors of the network, described in chapter 5

Report = 2 Returns a set of basic epidemiological measures and plots of the time series.

Report = 3 Report 1 + Report 2

Report Generation is an optional[3], though recommended, step done at the end of the simulation. For the report, descriptive statistics are generated for the network. These have to do with network topology and properties. Additional sections can be added to the report with basic statistical analyses of the output of pre-selected nodes[4].

Database output

Time series from simulations, are stored in a MySQL database named *epigrass*. The results of each individual simulation is stored in two tables named after the model's script name, the date and time the simulation has been run. For instance, suppose you run a simulation of a model stored in a file named `script.epg`, then at the end of the simulation, two new tables in the epigrass database will be created with the following name: `script_Wed_Jan_26_154411_2005` and `script_Wed_Jan_26_154411_2005e`. The first table contains the simulated time series associated with the nodes. The second table (ending with an ***e***) contains time-series associated with the connections of the networks. Results of multiple runs of the same model are stored independently.

[3]CAUTION! For large networks, the calculation of the statistics for the report may take a very long time!

[4]Listed in the siteRep variable at the script

Chapter 2

Epidemic Models

2.1 Introduction

Epidemiological models are implemented in the EpiGrass environment as a tool to predict, understand and develop strategies to control the spread of infectious diseases at different time/spatial scales. In this context, there are two direct potential applications. One is to model the spread of new diseases through an entirely susceptible population (ecological invasion). The velocity of spread of new diseases in a network of susceptible populations depends on their spatial distribution, size, susceptibility and patterns of contact. In a spatial scale, climate and environment may also impact the dynamics of geographical spread as it introduces temporal and spatial heterogeneity. Understanding and predicting the direction and velocity of an invasion wave is key for emergency preparedness.

Besides invasion, network epidemiological models can also be used to understand patterns of geographical spread of endemic diseases. Many infectious diseases can only be maintained in a endemic state in cities with population size above a threshold, or under appropriate environmental conditions (climate, availability of a reservoir, vectors, etc). The variables and the magnitudes associated with endemicity threshold depends on the natural history of the disease (Keeling and Grenfell, 1997). Theses magnitudes may vary from place to place as it depends on the contact structure of the individuals. Predicting which cities are sources for the endemicity and understanding the path of recurrent traveling waves may help us to design optimal surveillance and control strategies. For measles, for example, models suggest that geographical spread tend to follow a hierarchical pattern, starting in big cities (core) and then spreading to smaller neighborhood cities (satellites) (Grenfell et al., 2001).

EpiGrass implements a series of epidemiological models which, integrated with a set of analytical and visualization tools, may give us clues about the overall pattern of diseases spread in the geographical space. Besides, it can be used for scenario analysis to compare alternative intervention scenarios.

2.2 Disease models

Models of disease dynamics are quite diverse, ranging from caricatures to very detailed simulations. Traditional models of spread of diseases are based on the mean field assumption, i.e., that individuals interact randomly at a certain rate. Important references to the subject are (Diekmann and Heesterbeek, 2000; Daley et al., 2001; Isham and Medley, 1996; Anderson et al., 1992). These models are expressed mathematically as difference equations (discrete time) or differential equations (continuous time). In the simplest form, these models do not take into consideration either individual heterogeneity or the local nature of transmission events. Increased realism is achieved by structuring the population according to age, risk behavior, sex, susceptibility, or other category associated with different risk of getting or transmitting the disease. Within each sub-population, however, the assumption of well mixing must hold. When other species are involved in the transmission process (non-human hosts and vectors), these are also considered as compartments that may be sub-divided as well according to covariates associated with the risk of acquiring or transmitting the disease.

In this context, epidemiological models take the form of multi-compartmental models where each compartment is a well-mixed homogeneous population. The model describes the transition of the individuals in this population through a sequence of disease-related stages. These stages could be *Susceptible*, *Infected*, *Recovered*, for example. And the transitions could be

$$Susceptible \longrightarrow Infected$$

$$Infected \longrightarrow Recovered$$

If only these transitions are allowed, then individuals in the recovered class never become susceptible again (lifelong immunity). If, on the other hand, immunity is only temporary (as in pertussis), then another transition should be included:

$$Susceptible \longrightarrow Infected$$

$$Infected \longrightarrow Recovered$$

$$Recovered \longrightarrow Susceptible$$

One way to visualize these models is using state-flow diagrams, where boxes represent states (compartments) and arrows indicate the transitions. It is the state identity together with the transitions allowed that define the type of model in use.

2.2.1 Dispersal of infected individuals

A key feature of EpiGrass is to allow the simulation of geographical spread of infection from one locality to others. Dispersal of infection is modeled as in a 'forest fire' model (Grenfell et al., 2001). An infected individual, traveling to another city, acts as a spark that may trigger an epidemic in the new locality. This approach is based on the assumption that individuals commute between localities and contribute temporarily to the number of infected in the new locality, but not to its demography. Further details about this approach can be found in Grenfell et al (2001).

In all models presented below, new infections in locality i arise from the contact between Susceptibles in i and Infectious individuals. Infectious individuals are of two types: those residents in i (I_t) and those visiting i (θ). Mathematically, this is written as:

$$L_{t+1} = \beta S_t \frac{(I_t + \theta)^\alpha}{N_t + n_t}$$

where L_{t+1} is the number of new cases, β is the contact rate between Susceptibles and Infectious individuals, S_t is the number of susceptibles, I_t is the number of infectious individuals resident in the locality, θ is the number of infectious individuals visiting the locality, N_t is the population residing in the locality and n_t is the total number of individuals visiting the locality. α is a mixing parameter. $\alpha = 1$ corresponds to homogeneous mixing (Finkenstadt and Grenfell, 2000).

2.2.2 Typology of infectious diseases and corresponding models

Here we present a brief description of the typology of infectious diseases models based on the main route of transmission, and type of immunity resulting from infection. These models correspond to the types of models built into EpiGrass.

$$\begin{tikzcd}
\text{SIR} & \boxed{S} \arrow[r, "\lambda"] & \boxed{I} \arrow[r, "r"] & \boxed{R} & \\
\text{SIS} & \boxed{S} \arrow[r, "\lambda"] & \boxed{I} \arrow[l, bend left, "r"] & & \\
\text{SEIR} & \boxed{S} \arrow[r, "\lambda"] & \boxed{E} \arrow[r, "e"] & \boxed{I} \arrow[r, "r"] & \boxed{R} \\
\text{SEIS} & \boxed{S} \arrow[r, "\lambda"] & \boxed{E} \arrow[r, "e"] & \boxed{I} \arrow[ll, bend left, "r"] &
\end{tikzcd}$$

Figure 2.1: SIR-like models

SIR-like models

The natural history of many directly transmitted infectious diseases can be appropriately described by a SIR-like model. *SIR* stands for Susceptible (S), Infected (I) and Recovered (R). Archetypal *SIRs* are measles or chicken-pox, i.e., diseases that confer lifelong immunity (but see (Glass and Grenfell, 2004)). An individual starts his life in the S state and may progress to the I state. The rate of progression of individuals from S to I is called the incidence rate or force of infection (λ) which is a function of contact rate, probability of transmission per contact and density of infectious individuals. Individuals stay in the infectious period for a certain time and then move to the recovered state where they become immune to new infections. Generally, the removal rate from the infectious class is the inverse of the infectious period (i.e., it is assumed that the duration of infection is exponentially distributed).

Variations of this model allow cases where infected individuals do not acquire immunity after infection, thus returning to the susceptible pool (*SIS* model). Another variation is the inclusion of a latent stage to hold individuals that are infected but not infectious to others yet (incubation period). These are the *SEIR* (with immunity) and *SEIS* (no immunity) models.

Next, we describe in more detail each one of these models in their deterministic and stochastic versions, as used by EpiGrass.

SIR models Examples of diseases represented by SIR models are measles, chickenpox. Some diseases that do not confer lifelong immunity may be represented by this model if only short term dynamics is of interest. In the scale of a year, influenza and pertussis, for example, could be described using SIR. The SIR model is implemented in EpiGrass as a system of four difference equations. Besides the three equations describing the dynamics of S, I and R, a fourth equation explicitly defines the number of new cases per time step, $L(t)$ (i.e., the incidence). In general, this quantity is embedded in the I equation (prevalence), but it is important to keep track of the incidence if one wishes to compare prediction with notification data.

$$\begin{aligned}
L_{t+1} &= \beta S_t \frac{(I_t+\theta)^\alpha}{N_t+n_t} \\
I_{t+1} &= L_{t+1} + (1-r)I_t \\
S_{t+1} &= S_t + B - L_{t+1} \\
R_{t+1} &= N_t - (S_{t+1} + I_{t+1})
\end{aligned} \tag{2.1}$$

This model can be easily extended to include diseases without recovery, for example AIDS, the so called SI models. Basically, the recovery rate is set to 0.

SIS models In the SIS model, individuals do not acquire immunity after the infection. They return directly to the susceptible class.

The only difference between SIS and SIR models is the absence of R and the flow of recovered individuals to the susceptible stage:

$$\begin{aligned}
L_{t+1} &= \beta S_t \frac{(I_t+\theta)^\alpha}{N_t+n_t} \\
I_{t+1} &= L_{t+1} + (1-r)I_t \\
S_{t+1} &= S_t + B - L_{t+1} + rI_{t+1}
\end{aligned} \tag{2.2}$$

Table 2.1: Symbols used in the models and their meaning

Symbol	**Meaning**.
L_t	number of newly infected individuals at time t
E	number of exposed but not infectious individuals at time t
I	number of infectious individuals at time t
R	number of recovered individuals at time t
β	contact rate (t^{-1})
θ	number of infectious visitors
α	mixing parameter ($\alpha = 1$ means homogeneous mixing)
n	number of visitors
N	population ($S + E + I + R$)
B	susceptible pool replenishment
r	fraction of I recovering from infection per unit of time ($[0, 1]$)
e	fraction of E becoming infectious per unit of time ($[0, 1]$)
δ	probability of acquiring immunity ($[0, 1]$)
w	probability of losing immunity($[0, 1]$)
p	probability of recovered individual acquiring infection, given exposure ($[0, 1]$)

SEIR models These models have an extra compartment for those individuals who have acquired the infection but are still not infectious to others. This is the latent period and it is often parameterized as the inverse of the incubation period. Note, however, that for many diseases, initiation of infectiousness does not necessarily coincides with symptoms. In principle, any disease described by the SIR model can also be described by the SEIR model. The decision regarding the use of one or another depends on the magnitude of the latent period in relation to the time frame of other events in the simulation. The model has the form:

$$\begin{aligned}
L_{t+1} &= \beta S_t \frac{(I_t + \theta)^\alpha}{N_t + n_t} \\
E_{t+1} &= (1 - e)E_t + L_{t+1} \\
I_{t+1} &= eE_t + (1 - r)I_t \\
S_{t+1} &= S_t + B - L_{t+1} \\
R_{t+1} &= N_t - (S_{t+1} + I_{t+1} + E_{t+1}) \qquad (2.3)
\end{aligned}$$

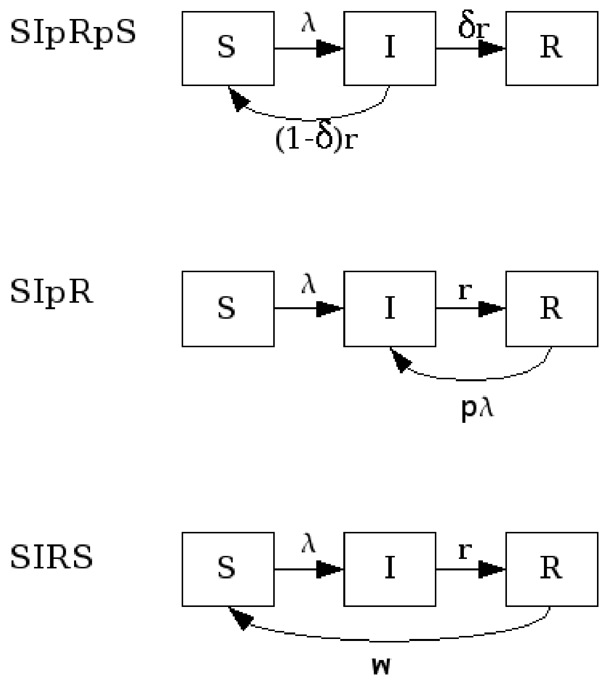


Figure 2.2: SIpR-like models.

SEIS models These are SIS models with the inclusion of the latent stage.

$$\begin{aligned}
L_{t+1} &= \beta S_t \frac{(I_t+\theta)^\alpha}{N_t+n_t} \\
E_{t+1} &= (1-e)E_t + L_{t+1} \\
I_{t+1} &= eE_t + (1-r)I_t \\
S_{t+1} &= S_t + B - L_{t+1} + rI_t
\end{aligned} \tag{2.4}$$

SIpR-like models

These are *SIR* models with immunity intermediary between full (*SIR*) and null (*SIS*). Some possibilities arise here: 1) Infection confers full immunity to a fraction of the individuals and the remaining ones return to the susceptible class again, after infection. (*SIpRpS*); 2) Infection provides only partial immunity and recovered individuals are partially susceptible to new infection (*SIpR*); 3) Immunity is full right after infection but wanes with time (*SIRS*). Each model is presented below. Figure 2.2 illustrates them diagrammatically.

Related models, that included the latent state E are: *SEIpRpS*, *SEIpR*, *SEIRS*.

SIpRpS model This model assumes that a fraction δ of infectious individuals acquire full immunity while the remaining $(1-\delta)$ returns to the susceptible stage. The model is:

$$\begin{aligned} L_{t+1} &= \beta S_t \frac{(I_t+\theta)^\alpha}{N_t+n_t} \\ I_{t+1} &= L_{t+1} + (1-r)I_t \\ S_{t+1} &= S_t + B - L_{t+1} + (1-\delta) r I_t \\ R_{t+1} &= N_t - (S_{t+1} + I_{t+1}) \end{aligned} \tag{2.5}$$

SIpR model This model assumes that immunity is only partial and recovered individuals may acquire infection again (at a lower rate $p\lambda$, where $0 \le p \le 1$). Two equations calculate the number of new infections. L_S calculates the number of susceptibles that become infected at $t+1$. L_R calculates the number of recovered that become infected at $t+1$. The latter are less susceptible to the disease when compared to susceptibles. The model is:

$$\begin{aligned} L_{S,t+1} &= \beta S_t \frac{(I_t+\theta)^\alpha}{N_t+n_t} \\ L_{R,t+1} &= p\beta R_t \frac{(I_t+\theta)^\alpha}{N_t+n_t} \\ I_{t+1} &= L_{S,t+1} + L_{R,t+1} + (1-r)I_t \\ S_{t+1} &= S_t + B - L_{S,t+1} \\ R_{t+1} &= N_t - (S_{t+1} + I_{t+1}) \end{aligned} \tag{2.6}$$

SIRS model Here, the immunity acquired by infection wanes with time. Pertussis is an example of this dynamic.

$$\begin{aligned} L_{S,t+1} &= \beta S_t \frac{(I_t+\theta)^\alpha}{N_t+n_t} \\ I_{t+1} &= L_{S,t+1} + L_{R,t+1} + (1-r)I_t \\ S_{t+1} &= S_t + B - L_{S,t+1} + wR_t \\ R_{t+1} &= N_t - (S_{t+1} + I_{t+1}) \end{aligned} \tag{2.7}$$

SnInRn-like models

These are models with more than one compartment for susceptibles, infected and recovered stages. They are used when infection involves more than one distinct populations. Vector borne diseases are classical examples where a SIR model is used to describe infection in humans and another SIR-like model is used to describe infection in the vector (and/or the reservoir(s)). Dengue fever and yellow fever are examples. Sexually transmitted diseases may also be modeled with SnInRn models if male and female populations are distinguished. These models can be define by the user as a custom model.

2.2.3 Stochastic models

All models described so far are deterministic. EpiGrass allows simulation of stochastic processes. This is done by assuming that L_{t+1} is a random variable with expected value given by the expressions found in the deterministic models. The user may choose the probability distribution for L_{t+1} between Poisson or Negative Binomial to draw realizations of L_{t+1}:

$$l_{t+1} \sim Poisson(L_{t+1})$$

or

$$l_{t+1} \sim NegBin(I_t, \frac{I_t}{I_t + L_{t+1}})$$

The Poisson distribution assumes independent events while the negative Binomial assume clustering of transmission events.

2.3 Network transportation models

The transmission models describe the dynamics of infection in a well-mixed population. EpiGrass allows the user to model the movement of infectious individuals between well-mixed populations, thus simulating the spread of disease through space. EpiGrass represents geographical space as a network where cities or localities are nodes and transportation routes are edges. The term network refers to the framework of routes within a system of locations, identified as nodes or sites. An edge is a single link between two sites (a road, a railroad, an air route or a river/sea corridor).

Transportation networks, like many networks, are generally embodied as set of locations and a set of links representing connections between those locations. The arrangement and connectivity of a network is known as its *topology*. Major types of topology are illustrated in figure 4.4.2. Velocity and direction of disease spread depend on the topology and weight of the edges of the transport network and there are many properties of networks that may useful when analyzing the spread of diseases. EpiGrass calculates a set of these properties as described in chapter 5.

In a transportation network, each edge (or link) is characterized by a variable $flow$ which states the number of passengers that travel through that link per unit time. EpiGrass uses this information to calculate the number of passengers arriving at each city, per time step. For example, consider node $N1$ in figure 2.3. At each time step, it receives 10 passengers from $N2$, 5 from $N5$, 1 from $N4$. Now suppose that, at this time step, 10% of the population within each site is infectious (I state), according to the epidemic model. Thus, a total of $10\% \times 10 + 10\% \times 5 + 10\% \times 1 = 1.6$ infectious individuals are visiting site $N1$. In the epidemic model embedded in $N1$, EpiGrass sets $n = 16$ and $\theta = 1.6$. This calculation of θ is based on a deterministic argument. The other possibility, allowed by Epigrass, is to define θ as a random variable, that follows a binomial distribution with parameters n,p, where n is as given in the deterministic version and p is the proportion of infectious individuals in the source population. From version 1.4 on, EpiGrass can also take into account the specific delays of each connection. When the average speed of the transportation system is set to a value greater than zero, epigrass calculates the time required to complete each trip and attributes this delay (in units of time) to the parameter δ. So, the number of infectious passengers arriving at any given city, at time t corresponds to the number of infectious passengers that left the city of origin at time $t - \delta$.

Deterministic:

$$\Theta_t = n \times \frac{I_{t-\delta}}{N}$$

Stochastic:

$$\Theta_t \sim Binomial(N, \frac{I_{t-\delta}}{N})$$

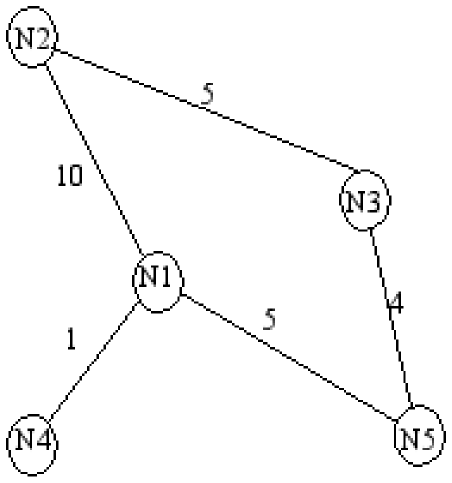


Figure 2.3: A simple transportation network

Chapter 3

Building and Installing

This chapter will walk through all aspects of EpiGrass installation. From obtaining, building and installing the prerequisites to the installation of EpiGrass itself.

Most of the steps will be quite simple and similar since they will make use of standard tools for package installation on two very popular GNU/Linux distributions: Debian (and derivatives), and Gentoo. If you use a different distribution, you should check its documentation for package installation instructions.

If, on your distribution, a package is not available for the required version, you can try to obtain an updated version of the package at the web-sites provided. On the rare cases where pre-built packages are not available, instructions on how to build the software from source should also be available from its web-site.

3.1 Required Packages

Python

Web-site: `http://www.python.org`

Version required: ≥ 2.3

Python is a simple but powerful object-orientated language. Its simplicity makes it easy to learn, but its power means that large and complex applications can be created. Its interpreted nature means that Python programmers are every productive because there is no edit/compile/link/run development cycle.

Python is probably installed automatically by your GNU/Linux distribution (it is on Gentoo). If not, it is best to use your distribution's standard tools for package installation. On Debian for example:

Listing 3.1: Installation of Python in a Debian-based Gnu/Linux distribution.

```
# apt-get install python
# apt-get install python2.3-dev
```

On Debian it is also necessary to install the development package of python whose name may vary depending on the version of python you have installed.

Numeric Python

Web-site: `http://www.numpy.org`

Version required: $\geq$ 23.0

Numeric Python is a module for fast numeric computations in Python. On-Debiann there are several packages, one for each version of python supported. make sure you install all python related packages for the correct version number.

Example installations:

Listing 3.2: Installing Numeric python on Gentoo GNU/Linux

```
# emerge numeric
```

Listing 3.3: Installing Numeric python on Debian GNU/Linux

```
# apt-get install python2.3-numeric
# apt-get install python2.3-numeric-ext
```

Matplotlib

Web-site: `http://matplotlib.sourceforge.net`

Version required: $\geq$ 0.80.0

Matplotlib is a Module that provides plotting capabilities to Python.

Listing 3.4: Installing Matplotlib on Gentoo GNU/Linux

```
# emerge matplotlib
```

Before using `apt-get` to install matplotlib, add these lines to your `/etc/apt/sources.list`:

Listing 3.5: Adding specific sources to apt-get.

```
deb http://anakonda.altervista.org/debian packages/
deb-src http://anakonda.altervista.org/debian sources/
```

Listing 3.6: Installing Matplotlib on Debian GNU/Linux

```
# apt-get update
# apt-get install python-matplotlib python-matplotlib-
  doc
```

PyQt

Web-site: `http://www.riverbankcomputing.co.uk/pyqt/index.php`

Version required: ≥ 3.13

PyQt is a set of Python bindings for the Qt toolkit. PyQt combines all the advantages of Qt and Python. A programmer has all the power of Qt, but is able to exploit it with the simplicity of Python.

PyQt depends on the Qt libraries to run. This dependency will be taken care by the package installation tools of most distributions, which will automatically install the required version of Qt.

Example installations:

Listing 3.7: Installing PyQt python on Gentoo GNU/Linux

```
# emerge pyqt
```

Listing 3.8: Installing PyQt python on Debian GNU/Linux

```
# apt-get install python2.3-qt3
```

MySQL

Web-site: `http://www.mysql.com`

Version required: ≥ 4.0

MySQL is a fast, multi-threaded, multi-user SQL database server. If you have a MySQL server available in your LAN, you may skip this step after making sure you have permission to access and use it to store your data.

Example installations:

Listing 3.9: Installing MySQL on Gentoo GNU/Linux

```
# emerge mysql
```

Listing 3.10: Installing MySQL on Debian GNU/Linux

```
# apt-get install mysql-server
# apt-get install mysql-client
```

Post-install configuration: MySQL requires a few extra configuration steps that must be completed after the installation described above. These steps must be performed by the root user. If you already have a MySQL server installed you will need to know the root password to configure EpiGrass later.

Listing 3.11: Post-install configuration of mysql on Gentoo

```
# /etc/init.d/mysql start
# mysql_install_db
# mysqladmin -u root password new-password
# rc-update add mysql default
```

In the mysqladmin line, replace new-password with a password of your own.

Listing 3.12: Post-install configuration of mysql on Debian

```
# mysql_install_db
# safe_mysqld &
# /etc/init.d/mysql start
# mysqladmin -u root password new-password
```

MySQL-python

Web-site: `http://sourceforge.net/projects/mysql-python/`

Version required: $\geq$ 1.2.1

This package is a MySQL module for Python.

Example installations:

Listing 3.13: Installing MySQL-python on Gentoo GNU/Linux

```
# emerge mysql-python
```

Listing 3.14: Installing MySQL-python on Debian GNU/Linux

```
# apt-get install python2.3-mysqldb
```

R

Web-site: `http://www.r-project.org`

Version required: ≥ 2.0

Example installations:

Listing 3.15: Installing R on Gentoo GNU/Linux

```
# emerge R
```

Listing 3.16: Installing R on Debian GNU/Linux

```
# apt-get install r-base
```

Post-install configuration: You have to install a few packages from within R afterward.

Listing 3.17: Installing aditional packages from within R

```
> install.packages('RMySQL')
> install.packages('DBI')
> install.packages('lattice')
```

RPy

Web-site: `http://rpy.sourceforge.net/`

Version required: $\geq 0.4.6$

RPy is a very simple, yet robust, Python interface to the R Programming Language. It can manage all kinds of R objects and can execute arbitrary R functions (including the graphic functions). Example installations:

Listing 3.18: Installing RPy on Gentoo GNU/Linux

```
# emerge rpy
```

If the `rpy` package on Gentoo is masked[1], Build and install it from source as described below.

Listing 3.19: Installing RPy on Debian GNU/Linux

```
# apt-get install python2.3-rpy
```

[1]Meaning that it can't be installed normally.

Depending on the Version of R installed, The installation of Rpy may not be successful. To test `rpy`, open a Python shell and type:

Listing 3.20: Testing the installation of RPy

```
>>> from rpy import *
>>> r.plot([1,2,3])
```

You should see a plot of three points generated by `R`. If instead, you get some error messages,you must install `RPy` from source. Download the RPy source tarball, unpack it, `cd` to the directory to which you unpacked it and type:

Listing 3.21: Installing RPy from source on GNU/Linux

```
# python setup.py install
```

RPy depends on R having been compiled with the option `--enable-R-shlib`. This is the default on Gentoo. If this installation fails on your system, you may have to get the latest version from rpy from its website and install from source by following these steps:

1. First of all, you **must** check that you have built R with the configure option '–enable-R-shlib', in order to make R as a shared library. If not, the following steps should be enough:

   Listing 3.22: Building R from source

   ```
    <go to the R source directory>
   # make distclean
   # ./configure --enable-R-shlib
   # make
   # sudo make install
   ```

2. Then, configure the path to the R library. For this, make a link to RHOME/bin/libR.so in /usr/local/lib or /usr/lib, then run `ldconfig`, (substitute RHOME with the path where R is installed, usually usrlocallibR):

   ```
   # sudo ln -s /usr/local/lib/R/lib/libR.so /usr/lib/libR.so
   ```

3. Ensure that you have the necessary header files for the version of R you are compiling against. You can check the version of R by running:

   ```
   # R --version
   R 2.0.1 (2004-11-15).
   Copyright (C) 2004 R Development Core Team
   ```

```

R is free software and comes with
ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under
the terms of the GNU General Public License.
For more information about these matters,
see http://www.gnu.org/copyleft/gpl.html.
```

There should be a subdirectory of the Rpy package with the name `R-<version>`.For the example above, `R-2.0.1`

If the correct version directory does not exist, you may have to go back to a version of `R` supported by rpy.

4. Now, just type:

```
# python setup.py install
```

and that's all!

GRASS GIS

Web-site: `http://grass.itc.it/`

Version required: $\geq$ 5.0.3

Listing 3.23: Installing GRASS on Gentoo GNU/Linux

```
# emerge grass
```

Listing 3.24: Installing GRASS on Debian GNU/Linux

```
# apt-get install grass
# apt-get install grass-doc
```

LaTeX

Web-site: `http://www.tug.org/teTeX/`

Version required: $\geq$ 2.0

EpiGrass uses PDFLaTeX to generate a report with a summary analysis of your network and simulation model. Thus, it is necessary to have the Tetex package installed.

Listing 3.25: Installing LaTeXand PDFLaTeXon Gentoo GNU/Linux

```
# emerge tetex
```

Listing 3.26: Installing LaTeXand PDFLaTeXon Debian GNU/Linux

```
# apt-get install tetex-base
```

Installing VPython

Web-site: `http://www.vpython.org`

Version required: $\geq$ 3.1.1

VPython is a library for the visualization of 3D objects in Python. Its required for the visualization module of EpiGrass.

Listing 3.27: Installing Vpython pre-requisites on Gentoo GNU/Linux

```
# emerge boost
```

Listing 3.28: Installing Vpython pre-requisites on Debian GNU/Linux

```
# apt-get install libboost-dev
# apt-get install libboost-python1.32.0
# apt-get install libboost-python-dev
# apt-get install gtkglarea5-dev
```

On Both distributions, after installing the boost libraries, you need to build Vpython from sources. Please refer to VPython's web site for instructions. After your are done installing it, You can test it from a python shell (see listing 3.29)

Listing 3.29: Testing Vpython

```
>>> import visual
>>> sphere()
```

Installing EpiGrass

If you got through all the steps above, it will be an easy task to install EpiGrass:

Listing 3.30: Intalling EpiGrass

```
# python setup.py install
```

We have written an ebuild for installingEpiGrasss on Gentoo. If it is unmasked at the time you decide to installEpiGrasss, you don't need to worry about the dependencies above and only need to type the following command:

Listing 3.31: Installing EpiGrass on Gentoo GNU/Linux

```
# emerge epigrass
```

Chapter 4

Using Epigrass

To simulate an epidemic process in EpiGrass, the user needs to have in hand three files: Two files containing the site and edge data and a third file which is a script that defines what it is to be done. In this chapter, we go through each one of them in detail. At the end part of this chapter, there is a step-by step guide the Graphical User Interface (GUI).

After a fresh installation of EpiGrass, a few example scripts and data sets are installed to the directory `/usr/share/epigrass/demos`. In order to be able to play with these demos, copy the contents of that directory to a directory on which you have write permission(e.g., any you create inside your home directory), open a shell window, go to that directory and invoke EpiGrass from there. The reason for that is that the reports of the simulations are written to the same directory epigrass was invoked from. It is also a good idea to keep the various simulation projects you develop inside their own separate directory so that the results from different simulations don't get mixed up.

4.1 Data

4.1.1 Site data file

See below an example of the content of a site file for a network of 10 cities. Each line corresponds to a site (except the first line which is the title). For each site, it is required that you declare the following variables, in this order: its *spatial location* in the form of a pair of coordinates ([X,Y]); a site *name* to be used in the output; the site's population; the site geocode (an arbitrary unique number which is used internally by EpiGrass).

Listing 4.1: Defining sites.

```
X,Y,City,Pop,Geocode
1,4,N1,1000000,1
2,4,N2,100000,2
3,4,N3,1000,3
4,4,N4,1000,4
5,4,N5,1000,5
1,3,N6,100000,6
2,3,N7,1000,7
3,3,N8,100000,8
4,3,N9,100000,9
5,3,N10,1000,10
1,2,N11,1000,11
```

In this example, the first site is located at $[X, Y] = [1, 4]$, it is named N1, its population is 1000000 and its geocode is 1. This is the minimum configuration of a site data file and it must contains this information in exactly this order.

In some situations, the user may want to add other attributes to the sites (different transmission parameters, or vaccine coverage or initial conditions for simulations). This information is provided by adding new columns to the minimum file. For example, if one wishes to add information on the vaccine coverage in cities N1 to N10 (*vac*) as well as information about average temperature (which hypothetically affects the transmission of the disease), the file becomes:

Listing 4.2: Adding extra variables to the sites file.

```
X,Y,City,Pop,Geocode,Vac,Temp
1,4,N1,1000000,1,0.9,32
2,4,N2,100000,2,0.88,29
3,4,N3,1000,3,0.7,25
4,4,N4,1000,4,0.2,34
5,4,N5,1000,5,0,26
1,3,N6,100000,6,0,27
2,3,N7,1000,7,0,31
3,3,N8,100000,8,0,30
4,3,N9,100000,9,0,24
5,3,N10,1000,10,0,31
```

During the simulation, each site object receives these informations and store them in appropriate variables that can be used later during model specification. Population is stored in the variable N; while the extra columns (those beyond the geocode) are stored in a tuple named *values*. For example, for the city $N1$, we have $N = 1000000$ and $values = [0.9, 32]$. During model specification, we may use N to indicate the population size, use $values[0]$ to indicate the level of vaccination of that city and $values[1]$ to indicate the temperature.

It is up to the user, to know the meaning of the elements in the tuple *values* (Note that the first element of the tuple has index 0,the second one has index 1 and so on).

When using real data, one may wish to use actual geocodes and coordinates. For example, for a network of Brazilian cities, one may build the following file:

Listing 4.3: Sites defined from real localities and their data.

```
latitude ,longitude ,local ,pop,geocode
-16:19:41,-48:57:10,ANAPOLIS,280164,520110805
-10:54:32,-37:04:03,ARACAJU,461534,280030805
-21:12:27,-50:26:24,ARACATUBA,164449,350280405
-18:38:44,-48:11:36,ARAGUARI,92748,310350405
-21:13:17,-43:46:12,BARBACENA,103669,310560805
-22:32:53,-44:10:30,BARRA_MANSA,165134,330040705
-20:33:11,-48:34:11,BARRETOS,98860,350550005
-26:54:55,-49:04:15,BLUMENAU,241943,420240405
-22:57:09,-46:32:30,B.PAULISTA,111091,350760505
```

In this example, the coordinates are the actual geographical latitude and longitude coordinates. This information is important when using EpiGrass integrated with Grass GIS. The geocode is also the official geocode of these localities. Despite the cumbersome size of the number, it may be worth using it because demographic official databases are often linked by this number. The site coordinates may be in sexagesimal or decimal format.

4.1.2 Edge data file

The edge data file contains all the direct links between sites. Each line in the file (except the first) corresponds to an edge. For each edge (or link) one must specify (in this order): the *names of the sites* connected by that edge; the *number of individuals traveling from source to destination*; the *number of individuals traveling from destination to source* per time step; the *distance or length* of the edge. At last, the file must contain, in the fifth and sixth columns, the *geocodes of the source and destination sites.* This is very important as the graph is built internally connecting sites through edges and this is done based on geocode info.

IMPORTANT: It is required that the order of columns is kept the same

See below the list of the 8 edges connecting the sites $N1$ to $N10$. Let's look the first one, as an example. It links $N1$ to $N2$. Through this link passes 11 individuals backwards and forwards per time step (a day, for example). This edge has length 1. The last two columns show the geocode of $N1$ (geocode 1) and the geocode of $N2$ (geocode 2).

Listing 4.4: Defining edges.

```
Source,Dest,flowSD,flowDS,Distance,geoSource,geoDest
N1,N2,11,11,1,1,2
N2,N4,0.02,0.02,1,3,4
N3,N8,1.01,1.01,1,3,8
N4,N9,1.01,1.01,1,4,9
N5,N10,0.02,0.02,1,5,10
N6,N5,1.01,1.01,1,7,8
N7,N10,1.01,1.01,1,7,8
N9,N10,1.01,1.01,1,9,10
```

Note that it doesn't matter which site is considered a Source and which one is considered a Destination, i.e., if there is a link between A and B, one may either named A as source and B as destination, or the other way around.

If the edge represents a road or a river, one may use the actual metric distance as length. If the edge links arbitrary localities, one may opt to use Euclidean distance, calculated from the x and y coordinates.

4.2 Specifying a Simulation: the *.epg* script

Once the user has specified the two data files, the next step is to define the details of th simulation to be executed. This is done in the `.epg` script file. The `.epg` script is a text file and can be edited with any text editor (not a word processor!). This script must be prepared with care.

The best way to write down your own `.epg` is to edit an already existing .epg file. So, open **EpiGrass**, choose an `.epg` file and click on the **Edit** button. Your favorite editor will open and you can start editing. Don't forget to save it as a new file in your working directory. Of course, there is an infinite number of possibilities regarding the elaboration of the script. It all depends on the goals of the user.

For the beginner, we suggest him/her to take a look at the .epg files in the demo directory. They are all commented and may help the user in getting used with Epigrass language and capabilities.

Some hints to be successful when editing your `.epg`:

- All comments in the script are preceded by the symbol #. These comments may be edited by the user as he/she wishes and new lines may be added at will. Don't forget, however, to place the symbol # in every line corresponding to a comment.

- The script is divided into a few parts. These parts have capital letter titles within brackets. Don't touch them!

- Don't remove any line that is *not* a comment. See below how to appropriately edit these command lines.

Let's take a look now at each part of a script (this is the script.epg demo file):

4.2.1 PART 1: THE WORLD

The first section of the script is titled: THE WORLD. An example of its content is shown:

Listing 4.5: declaring sites and edges files on "THE WORLD" section of the .epg file.

```
#==========================================================#
[THE WORLD]
#==========================================================#

sites = sitios2.csv
edges = edgesout.csv
```

where

sites: this is the name of the .CSV file containing the list of sites and their attributes.

edges: this is the name of the .CSV file containing the list of edges and their attributes.

Table 4.1: Type of models implemented in EpiGrass

Model	**Deterministic**	**Stochastic**.
Susceptible-Infected-Recovered	`SIR`	`SIR_s`
Susceptible-Exposed-Infected-Recovered	`SEIR`	`SEIR_s`
Susceptible-Infected-Susceptible	`SIS`	`SIS_s`
Susceptible-Exposed-Infected-Susceptible	`SEIS`	`SEIS_s`
SIR with fraction with full immunity	`SIpRpS`	`SIpRpS_s`
SEIR with fraction with full immunity	`SEIpRpS`	`SEIpRpS_s`
SIR with partial immunity for all	`SIpR`	`SIpR_s`
SEIR with partial immunity for all	`SEIpR`	`SEIpR_s`
SIR with immunity wane	`SIRS`	`SIRS_s`
Example Influenza model	Not available	`Influenza`
User defined model	`Custom`	`Custom`

4.2.2 PART 2: EPIDEMIOLOGICAL MODEL

This is the main part of the script. It defines the epidemiological model to be run. The script reads:

Listing 4.6: Choosing model type.

```
#===============================================================#
[EPIDEMIOLOGICAL MODEL]
#===============================================================#
#model types available: SIS, SIS_s ,SIR, SIR_s, SEIS, SEIS_s,
    SEIR, SEIR_s,
# SIpRpS, SIpRpS_s,SIpR,SIpR_s, Influenza, Custom (see
    documentation for description)
modtype = SIR
```

Here, the type of epidemiological model is defined, in this case is a deterministic SIR model. EpiGrass has some built-in models:

A description of these models can be found in section 2. The stochastic models use Poisson distribution as default for the the number of new cases (L_{t+1}).

The keyword *Custom* means that the user will provide a custom epidemiological model. A detailed discussion on how to create user-defined models can be found below (section 4.3).

Following the script, we find:

Listing 4.7: Defining model parameters

```
#===================================================================#
[MODEL PARAMETERS]
#===================================================================#
```

```

# They can be specified as constants or as functions of global
    or
# site-specific variables. these site-specific variables, are
    provided
# in the sites file. All the numbers given after the geocode (4
    th column)
# are collected into the values tuple.
# Examples:
# beta = 0.001
# beta=values[0] #assigns the first element of values to beta
# beta=0.001*values[1]

beta = 0.4  #transmission coefficient (contact rate *
    transmissibility)
alpha = 1  # clumping parameter
e = 1   # inverse of incubation period
r = 0.1   # inverse of infectious period
delta = 1  # probability of acquiring full immunity [0,1]
B = 0         # Birth rate
w = 0         # probability of immunity waning [0,1]
p = 0         #
```

These are the model parameters, as described in table 2.1. Not all parameters are necessary for all models. For example, e is only required for SEIR-like models. Don't remove the line, however, because that will cause a syntax error. We recommend that, if the parameter is not necessary, just add a comment after it as a reminder that it is not being used by the model. If you are running a custom model, You may add new parameters as needed.

In some cases, one may wish to assign site-specific parameters. For example, transmission rate may be different between localities that are very distant and are exposed to different climate. In this case site specific variables can be added as new columns to the site file. All columns after the geocode are packed into a tuple named *values* and can be referenced as shown in listing 4.7. I.e., the first element of the tuple is values[0], the second element is values[1], the third element is values[2] and so on.

In the next part of the script, the initial conditions are defined. Here, the number of individuals in each epidemiological state, at the start of the simulation, is specified.The script reads:

Listing 4.8: Setting initial conditions.

```
[INITIAL CONDITIONS]

# Here, the number of individuals in each epidemiological
# state (SEI) is specified. They can be specified in absolute
# or relative numbers.
# N is the population size of each site.
# The rule defined here will be applied equally to all sites.
# For site-specific definitions, use EVENTS (below)
# Examples:
#         S,E,I = 0.8*N, 10, 0.5*N
#         S,E,I = 0.5*N, 0.01*N, 0.05*N
#         S,E,I = N-1, 1, 0
S = N
E = 0
I = 0
```

Here, N is the total population in a site (as specified in the sites file). In this example, we set all localities to the same initial conditions (all individuals susceptible) and use an event (see below) to introduce an infectious individual in a locality. The number of recovered individuals is implicit, as $R = N - (S + E + I)$

Another possibility is to define initial conditions that are different for each site. For this, the data must be available as extra columns in the site datafile and these columns are referenced to using the *values* tuple explained above.

If you are running a custom model, You may add new variables as needed.

The next step is to define events that will occur during the simulation. These events may be epidemiological (arrival of an infected individual, for example) or a public health action (vaccination campaign, for example):

Listing 4.9: Defining epidemic events

```
#==================================================================#
[EPIDEMIC EVENTS]
#==================================================================#
#  Specify isolated events.
#  Localities where the events are to take place should be
   Identified by the geocode, which
# comes after population size on the sites data file.
# All coverages must be a number between 0 and 1.
# Seed : [('locality1's geocode', variable, n),('locality2's
   geocode',variable, n)].
# N infected cases will be added to locality at time 0.
# Vaccinate: [('locality1's geocode', [t1,t2], [cov1,cov2]),('
   locality2's geocode', [t1,t2], [cov1,cov2])]
# Multiple vaccination campaigns with specific coverages can be
   specified as lists (see manual)
# Quarantine: [(locality1's geocode,time,coverage), (locality2'
   s geocode,time,coverage)]
# Keywords may be used instead of the geocode in the vaccinate
   variable:
# 'all': Apply the same values to all sites
```

```
seed = [(355030800,'I',1)] #Sao Paulo
Vaccinate = []
Quarantine = []
```

The events currently implemented are:

seed One infected individual is introduced into a site(s) at the beginning of the simulation. The notation for a single event is:

$$seed = [(geocode, variable, n)]$$

For example, $seed = [(2, I, 1)]$ programs the arrival of one infectious (I) individual at site geocode 2, at time 1. For several events, the notation is:

$$seed = [(geocode1, variable, n), (geocode2, variable, n), (geocode3, variable, n)]$$

There is no limit to the number of events that can be included in this list.

Vaccinate Implements a campaign that vaccinates a fraction of the population in a site, at pre-defined times. For a single vaccination event, the notation is:

$$[(geocode1, [time], [coverage])]$$

where the first element is the geocode of the city, the second element is a list of the times when the campaign is carried on, and the third element is a list with the coverages for each vaccination campaign, respectively. For example, the event $[(2, [10], [0.7])]$ means that city 2, at time 10, has 70% of its population vaccinated. Mathematically, it means (in the model), the removal of individuals from a susceptible to a recovered state (built-in models). In custom models (user defined), the user defines in the model what the effects of the vaccination should be. Multiple, independent vaccination campaigns may be specified. For example, we may have simultaneous vaccinations at two sites:

$$Vaccinate = [(1, [31], [0.9]), (2, [31], [0.9])]$$

or subsequent campaigns in one site (at different times and with different coverages), plus a single campaign on another:

$$Vaccinate = [(1, [31, 61], [0.2, 0.3]), (2, [61], [0.3])]$$

If we want to vaccinate all cities at the same time we can replace the geocode number by the keyword "all":

$$Vaccinate = [('all', [31], [0.2])]$$

Quarantine Prevent individuals from leaving a site, starting at t with a certain efficacy. The notation is:

$$Quarantine = [(geocode1, t, efficacy), (geocode2, t, efficacy)]$$

As an example, a quarantine in locality 2, that starts at day 10 is noted: $Quarantine = [(2, 10, 0.7)]$

4.2.3 PART 3: TRANSPORTATION MODEL

Here, there are three options regarding the movement of infected individuals from site to site (through the edges). The parameter `doTransp` controls wether or not the transportation will be calculated. On most models it will be set to 1.

The parameter `stochastic` refers to how the number of people traveling is calculated. If *stochastic* $= 0$, the process is simulated deterministically, that is, the number of infected passengers commuting through an edge is given by the number of passengers (as given in the edges file) times the proportion of infected individuals in the source population. If *stochastic* $= 1$, the number of infectious passengers is drawn from a binomial distribution with parameters $p = \frac{I}{N}$, where I is the number of infectious individuals in the population of size N, and n is the total number of passengers as above.

The parameter `speed` correspond to the speed of the transportation system in unit of distance per units of time. The user must be careful to make the units in this file agree with the one on the edges file.

The transportation speed, when set to a value greater than zero, will cause a delay in the transportation of passengers across edges, proportional to the edge length (as explained in 2.3).

Listing 4.10: Transportation parameters

```
#==========================================================#
[TRANSPORTATION MODEL]
#==========================================================#
# If doTransp = 1 the transportation dynamics will be
# included. Use 0 here only for debugging purposes.
doTransp = 1

# Mechanism can be stochastic (1) or deterministic(0).
stochastic = 1

#Average speed of transportation system in km per time step.
    Enter 0 for instantaneous travel.
#Distance unit must be the same specified in edges files
speed =1440  # km/day -- equivalent to 60 km/h
```

That ends the definition of the model.

4.2.4 SIMULATION AND OUTPUT

Now it is time to define some final operational variables for the simulation:

Listing 4.11: Simulation and output section

```
#==========================================================#
[SIMULATION AND OUTPUT]
#==========================================================#

# Number of steps
steps = 200

# Output dir. Must be a full path. If empty the output will be
    generated on the
# outdata-<date> sub-directory of same path as the model script.
outdir =


# Database Output
# MySQLout can be 0 (no database output) or 1
MySQLout = 1


# Report Generation
# The variable report can take the following values:
# 0 - No report is generated.
# 1 - A network analysis report is generated in PDF Format.
# 2 - An epidemiological report is generated in PDF Format.
# 3 - A full report is generated in PDF Format.
# siteRep is a list with site geocodes. For each site in this
    list , a detailed report is appended to the main report.
report = 0
siteRep = [230440005,355030800]

#Replicate runs
#If replicas is set to an integer(n) larger than zero, the model
     will be run n times and the results will be con-
#solidated before storing.
# if RandSeed is set to 1 the seed will be randomized on each
    replicate
#Replicate mode automatically turn off report and batch options.
Replicas = 0
RandSeed = 0


#Batch Run
```

```
# list other scripts to be run in after this one. don't forget
    the extension .epg
# model scripts must be in the same directory as this file or
    provide full path.
# Example: Batch = ['model2.epg','model3.epg','/home/jose/
    model4.epg']
Batch = []
```

where

step Number of time steps for the simulation.

outdir Directory for data output. In this directory, several datasets useful to posterior analyses, are saved. Currently, the adjacency matrix, the shortest path matrix and the shortest path distance matrix are being saved. In addition, tables with network-wide epidemiological statistics and site statistics are also put in this directory. If not specifically named, `outdir` defaults to `outdata-<date/time>`.

MySQLout Use MySQLout = 1 if simulated time series are to be stored in the MySQL database. Time series of all of the models' variables, for every site, are stored in a MySQL database named *epigrass*. The results of each individual simulation is stored in a different table named after the model's script name, the date and time the simulation has been run. For instance, suppose you run a simulation of a model stored in a file named `script.epg`, then at the end of the simulation, a new table in the epigrass database will be created with the following name: `script_Wed_Jan_26_154411_2005`. Thus, the results of multiple runs from the same model get stored independently. Time series for traffic though the edges are also saved on a table with the same name but with an `_e` appended at the end.

report Three types of report are currently available: *Report* = 1 returns a set of descriptors of the network, described in 5; *Report* = 2 returns a set of basic epidemiological measures and plots of the time series; *Report* = 3 is *Report*1 + *Report*2. Report Generation is optional, though recommended, step done at the end of the simulation. For the report, descriptive statistics are generated for the network. These have to do with network topology and properties. Additional sections can be added to the report with basic statistical analyses of the output of pre-selected nodes, listed in the *siteRep* variable at the script.

siteRep=[ ] List of nodes for which network and epidemiological measures are to be calculated and included in the report.

Replicas The model can be run in replicate mode. This mode turns off the report and batch options. A separate database table is generated for each of the replicate runs.

RandSeed If this parameter is set to one, a random seeding site is selected for each run.

Batch=[ ] script files included in this list are executed after the currently file is finished.

4.3 User defined models

Starting with release 1.4.0, Epigrass allows the user to define his/her own epidemiological models, instead of using the built-in models. Naturally these models have to follow some convention regarding input and output data, but apart from this, the user is free to do anything. To use custom models, besides writing the model the user must select the type `custom` in the the model type parameter in the `.epg` file.

4.3.1 Starting from the example template

In the demo directory of your Epigrass installation, you will find a file named `CustomModel.py` (listing 4.12. This file contains a replica of the built-in model SIR. We will analyze this file to understand how it is written, an how it can be modified. Custom models are written in the Python language, which is a very easy to grasp language with a very clean syntax.

Epigrass expects to find a file named `CustomModel.py` in the directory from where it was started (where all the other required model definition files also reside). Custom models must always be on a file named `CustomModel.py` and contain at least a function named Model. Everything in this file is case sensitive, including the file name, so be careful.

Listing 4.12: Template custom model

```python
# This is a custom model to used in place of Epigrass' built-in models. Custom
# models must always be on a file named CustomModel.py and contain at least
# a function named Model. Both the File name and the function Names are case-sensitive,
# so be careful. Please refer to the manual for intructions on how to write your
# own custom models.

def Model(self,vars,par,theta=0, npass=0):
        """
        Calculates the model SIR, and return its values.
```

```
10         - inits = (E,I,S)
11         - par = (Beta, alpha, E,r,delta,B, w, p) see docs.
12         - theta = infectious individuals from neighbor sites
13         """
14         #Initializing
15         if self.parentSite.parentGraph.simstep == 1: #get initial values
16             E,I,S = (self.bi['e'],self.bi['i'],self.bi['s'])
17         else:
18             E,I,S = vars
19         N = self.parentSite.totpop
20         beta,alpha,e,r,delta,B,w,p = par
21         #Vacination event (optional)
22         if self.parentSite.vaccineNow:
23             S -= self.parentSite.vaccov*S
24 
25         Lpos = beta*S*((I+theta)/(N+npass))**alpha #Number of new cases
26         self.parentSite.totalcases += Lpos #update number of cases
27         # Model
28         Ipos = (1-r)*I + Lpos
29         Spos = S + B - Lpos
30         Rpos = N-(Spos+Ipos)
31         # Updating stats
32         self.parentSite.incidence.append(Lpos)
33         # Raises site infected flag and adds parent site to the epidemic
               history list.
34         if not self.parentSite.infected:
35             if Lpos > 0:
36                 #if not self.parentSite.infected:
37                 self.parentSite.infected = self.parentSite.parentGraph.
                       simstep
38                 self.parentSite.parentGraph.epipath.append((self.parentSite.
                       parentGraph.simstep, self.parentSite, self.parentSite.
                       infector))
39         #Migrating infecctious
40         self.parentSite.migInf.append(Ipos)
41 
42         return [0,Ipos,Spos]
```

The first thing we need to do before we start editing the template file, is to copy it to a work directory to which you have write permission. This directory should naturally be the one you intend to contain all the files concerning your custom simulation model: sites and edges files, `.epg` files, etc.

In the beginning of the file, preceded by the symbol #, is a comment. This comment can be completely removed or modified to include whatever information the user wants to add.

At line 7, we have the beginning of the obligatory Model Function. Between the parentheses are the input arguments to the function. These cannot be changed since they correspond to the arguments passed by Epigrass to all model functions.

From line 8 to 13, we have the function's docstring. This is optional and corresponds to documentation specific to the `Model` function.

Lines 14-20, cannot be removed but must be adapted to reflect the variables of the model. These lines set the values of the state variables of the model, which receive their initial values (specified in the `.epg` file and stored in the self.bi dictionary) at the the first time step of the simulation and the values of the last simulation step (`vars`) subsequently. The general structure of the `if/else` clause cannot be changed, only variables can be added or removed as needed.

Now come the optional lines regarding vaccination. If you plan to simulate vaccination scenarios using your models you must include these lines to determine what will be the effects of the vaccination.

The following lines up to line 30 are the equations of the model itself. These should be replaced by the user's formulation. Care should be taken that variables `Lpos`(number of new cases) and `Ipos`(number of infectious individuals) are later stored in the `incidence` and `migInf` lists, respectively. If these variables names are changed, References to them in the following lines should be updated as well.

The rest of the function should not be altered except for the possibly necessary replacement of `Lpos` and `Ipos` by new names and, on the last line, `return [...]` where the list returned must contain the variables of the model on the exact same order in which they are received in lines 16 and 18.

4.4 Using Epigrass for specific tasks

Here we describe some things you could do with epigrass and some specific hints:

4.4.1 Describing a network

A user wants to obtain the topological properties of a network. Reasons for this could be: 1) learn how to interpret these measures, 2) describe a air transportation or a road transportation network. To do that, you need:

Set steps = 1 . If no model of disease is needed, then most of the .epg script can be ignored. Don't remove anything, however, from the script. Note that Epigrass requires a model in order to work properly, even if the user does not want it. One solution to reduce the run time, in this case, is to set **steps** to 1 (steps = number of steps in the simulation).

Set MySQLout = 0 . Network measures are not sent to database.

Set report = 1 . Report 1 calculates network measures and save them in a .pdf file.

Specify siteRep . If siteRep = [], only global network measures are included in the report. If site-specific measures are needed, include their geocodes in the list siteRep. For example, to calculate site stats for all nodes, mesh1.epg has:

```
report = 1
siteRep = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
```

Script mesh1.epg is configured this way. Run it and take a look at the report.

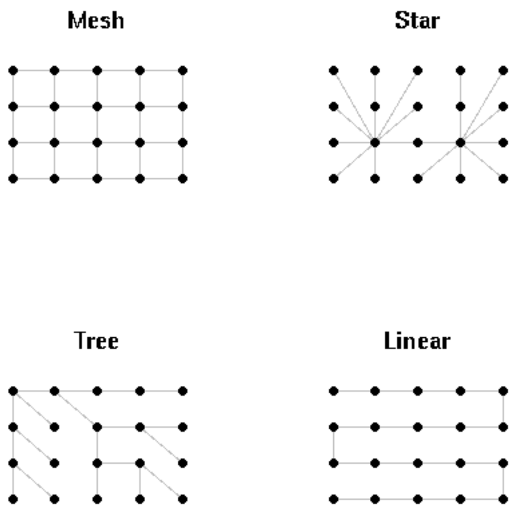
4.4.2 Comparing networks

A user wants to compare the properties of a set of networks. Reasons for this could be: 1) learn how to interpret these measures, 2) describe/compare a air transportation to a road transportation network, 3) analyse how network topology changes by adding/removing specific nodes or edges.

If there are four graphs, then four .epg files must be created. They all must set *report* = 1 and *siteRep* to the desired specification (as above). Each file must be executed and each one will provide a report.

To speed thinks a little bit, the script allows the user to choose one of the files as a master file. In the option *BatchRun*, one may list the other scripts to be run after this one. They all must be in the same directory as the master file (or you may provide full path).

For example, suppose you want to compare the topologies of the four networks displayed in 4.4.2.



We created four files (mesh2.epg, star2.epg, lin2.epg, tree2.epg) and used one of them as master (mesh2.epg). The four files are exactly the same, except for the name of the edge file and the **Batch** specification. I.e., in mesh2.epg we specify:

```
Batch = ['star2.epg','lin2.epg','tree2.epg']
```

Now, mesh2.epg is run. One report will be delivered for each script (In future version of Epigrass, a more integrated result is planned). From the reports, we get network measures for the four graphs. These network measures are explained in chapter 5).

4.4.3 Simulate disease spread from a single site

The user specifies a network (let's say, a tree network) and wishes to simulate disease spread in this network. The graph is disease-free at time 0. At time 1, an infected person arrives at site $N1$. No control measures are introduced. The model chosen is SipRpS. The script file tree3.epg was built following these guidelines:

Initial conditions All individuals are initially susceptible, i.e., $S = N$.

Epidemic events An infected individual arrives at time 1 in N1.

steps=200 This may be increased or reduced, depending on the parameters.

Report = 2 . Report 2 returns only the epidemiological results.

Specify siteRep . If site-specific measures are needed, include their geocodes in the list siteRep. For example, to calculate site stats for three nodes, tree3.epg has:

```
report = 2
siteRep = [1,12,14]
```

Run this script and take a look at the report. A suggestion: change the script and seed the disease in a more central node. See how this affect the velocity of disease propagation.

4.4.4 Simulating vaccination campaigns

Vaccination may be simulated in different ways, using the section Epidemic Events in the script. These are some examples:

A single, local campaign . In site N1, exactly at time 10, with coverage 0.5

```
Vaccinate = [(1,10,0.5)]
```

Simultaneous campaigns in three sites . In sites N1, N4 and N6, exactly at time 10, with coverage 0.5

```
Vaccinate = [(1,10,0.5),(4,10,0.5),(6,10,0.5)]
```

Campaign with a time span . In site N1, a campaign that occurs from day 10 to day 15, with daily coverage of 0.1.

```
Vaccinate = [(1,10,0.1),(1,11,0.1),(1,12,0.1),(1,13,0.1),
(1,14,0.1),(1,15,0.1)]
```

4.4.5 Simulating quarantines

Quarantines are simulated similarly to vaccinations, but once they are initiated, they last until the end of the simulation:

Quarantine in a single place . In site N1, starting at time 10, with coverage 0.2

```
Quarantine = [(1,10,0.2)]
```

Quarantine in two places . In sites N1 and N3, at times 10 and 12, respectively, with coverage 0.5

```
Quarantine = [(1,10,0.5),(3,12,0.5)]
```

Quarantine with a time span . In site N1, a quarantine that starts at day 10 and ends at time 30, with daily coverage of 0.75.

```
Quarantine = [(1,10,0.75),(1,30,0)]
```

4.4.6 Comparing strategies

One goal of modeling diseases is to compare alternative control measures in terms of number of cases prevented. A set of scripts may be prepared to compare six alternative strategies for controlling the spread of an epidemic in the star graph, that was initiated at time 4 in site N1. For example:

Strategy vac1 Vaccinate site $N1$, at time 7, with coverage 0.8 .

Strategy vac2 Vaccinate sites $N1$, $N12$ and $N14$, at time 7, with coverage 0.8. $N12$ and $N14$ are central nodes of the star network and are natural candidates for vaccination.

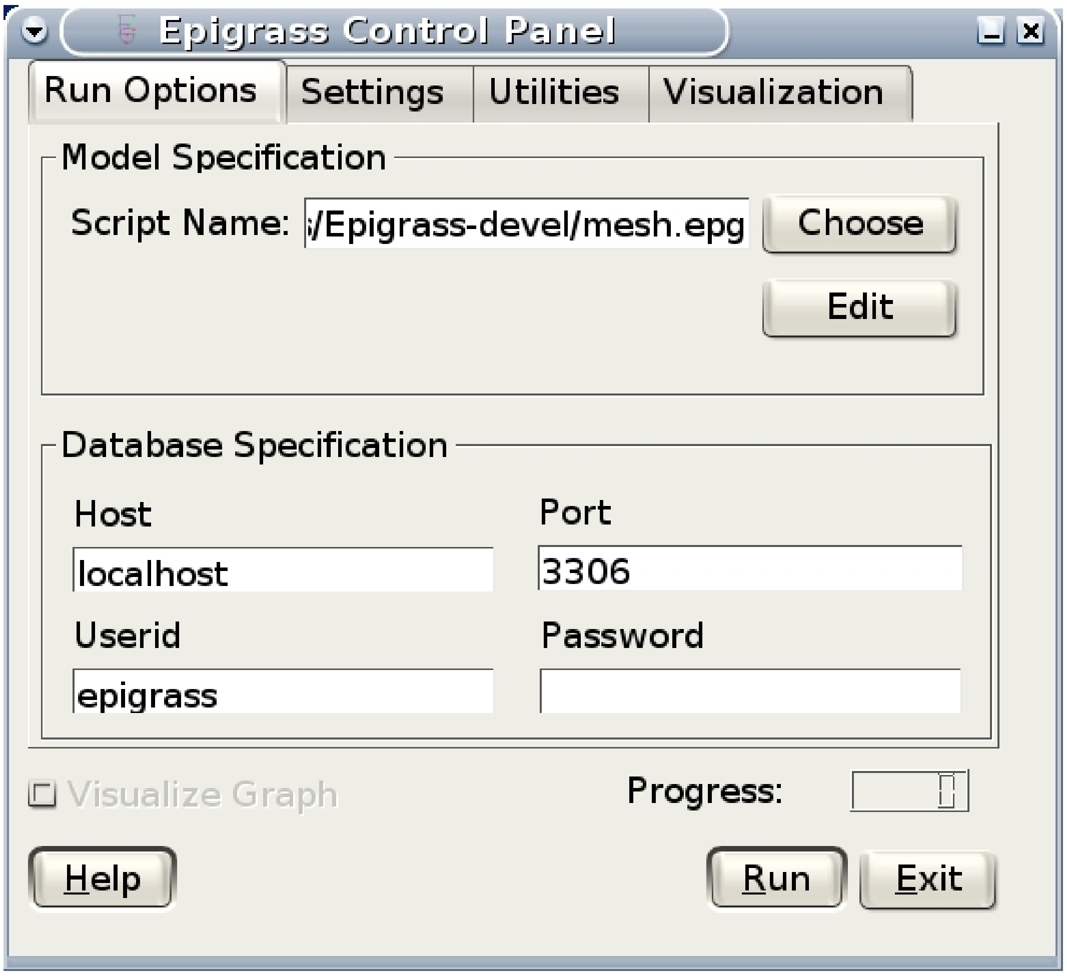
Strategy mixed Strategy vac1 + quarantine in sites $N12$ and $N14$, coverage of 0.7.

Strategy quar Quarantine in sites $N1$, $N12$ and $N14$, coverage of 0.7.

4.5 The Graphical User Interface(GUI)

Epigrass comes with a simple but effective GUI(figure 4.1), that allows the user to control some aspects of the run-time behavior of the system. The GUI can be invoked by typing `epigrass.py` in prompt of a console. EpiGrass should be started from the same directory where his/her model definition is located (.csv and .epg files).

All the information that is entered via the GUI gets stored in a hidden file called `.epigrassrc` stored in the home folder of the user. Every time the GUI is invoked, the data stored in the `.epigrassrc` file is used to fill the forms in the GUI. The GUI is designed as a tabbed notebook with four tabs (Run Options, Settings, Utilities, and Visualization).



Script Name: /Epigrass-devel/mesh.epg

Host: localhost

Port: 3306

Userid: epigrass

Password:

Figure 4.1: First tab of the EpiGrass GUI.

At the bottom of the GUI there are three buttons `Help`, `Start` and `Exit`. Their functions will be explained below. Immediately above the `Run` and `Exit` buttons, there is a small numeric display that will display the simulation progress after it has been started.

4.5.1 Run Options

The first tab of the GUI(figure 4.1), contains a number of variables that, with the exception of the model script filename, should remain the same for most simulations you are going to run.

On the top of the first tab is a text box to enter the file name of the model script (`something.epg`). By clicking on the `Choose` button at the right of this box, you get a file selection dialog to select your script file. If you need you can click on the `Edit` button below to edit the script file with your favorite text editor.

Below, you can enter details about the MySQL database that will store the output of your simulations. Here you can enter the server IP, port, user and password. On the first time you run the GUI these input boxes will be filled with the default values for these variables (server on localhost, port 3306, user epigrass and password epigrass)

4.5.2 Settings

On the settings page(figure 4.2), you can enter personal details such as user name (To be used in the simulation report), preferred text editor and preferred PDF reader. The preferred text editor will be used to open your script from the GUI, when you click on the edit button in the first tab. The PDF reader specified, will be used to open the report file, when requested (Utilities tab) and the user manual, when the user clicks on the help button on the bottom-left corner of the GUI.

On this tab, the language of the GUI can also be selected from a list of available translations. The effects of language changes will only take place when the next time the GUI is started.

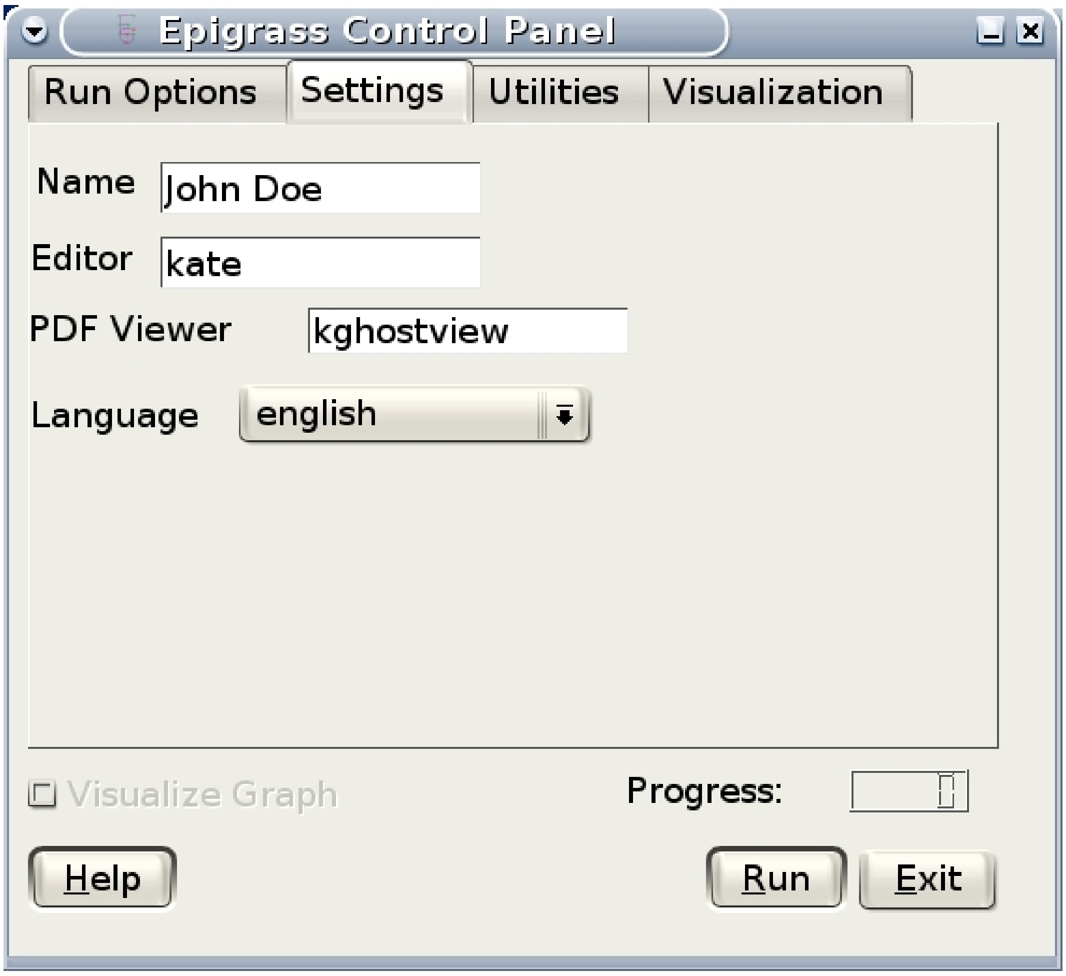
4.5.3 Utilities

In the Utilities tab(figure 4.3), you can get feed back from the simulator. Especially during long simulation runs, it is good to know how it is progressing. During the simulation, text messages regarding the status of the simulation are written to the text box on the left.

On the right, there is a button for backing up the data base and another for opening the report generated by the last simulation. Since report PDFs ar stored in folder directly below the ones on which the simulation is started, older reports should still be accessible and can be opened directly by selecting the desired report using the operating system's file manager.

4.5.4 Visualization

The fourth tab of the GUI is the visualization Tab. This tab was designed for playing animations of any simulation data that is stored in the database. Pressing the `Scan DB` button, causes the available tables in the epigrass database to be listed in the `Simulations stored` combo box. The user can then select one of these simulations to visualize.



Name: John Doe
Editor: kate
PDF Viewer: kghostview
Language: english

Figure 4.2: Second tab of the GUI.

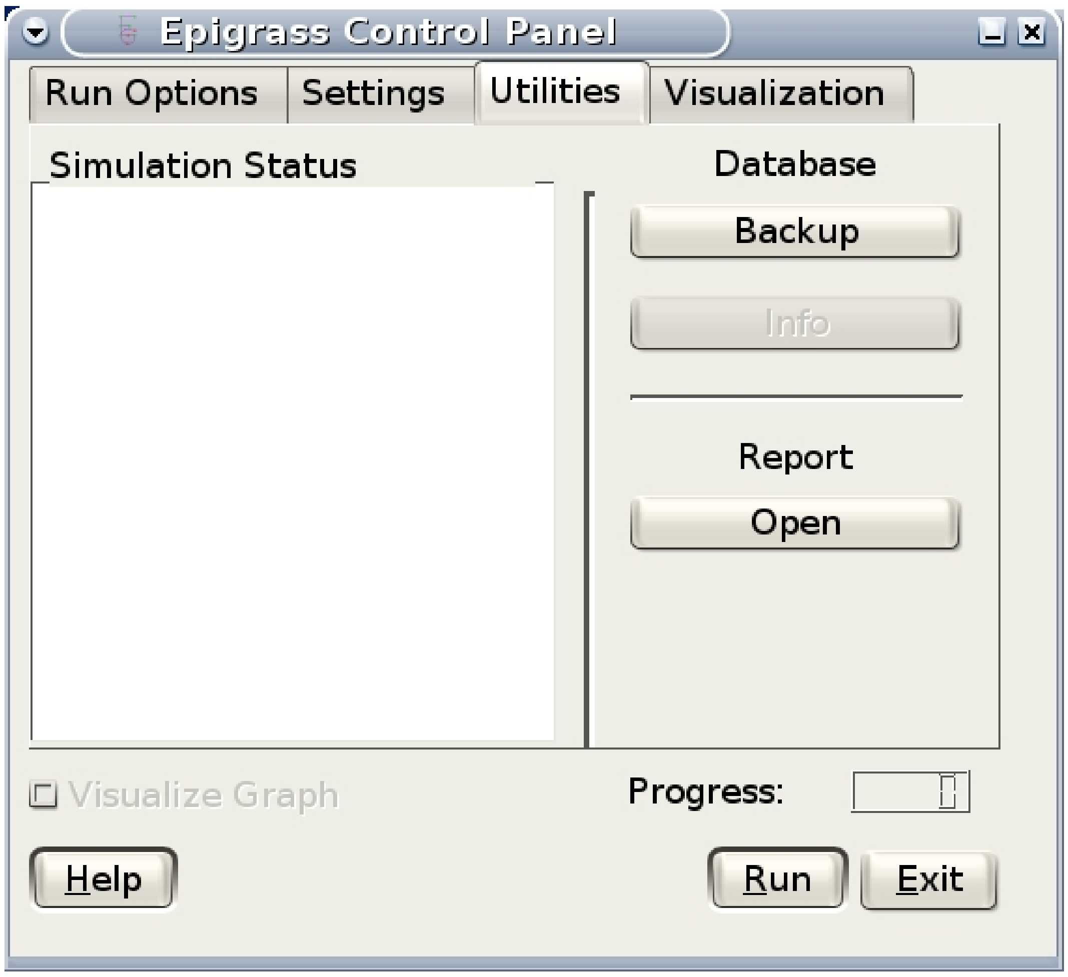


Figure 4.3: Third tab of the GUI

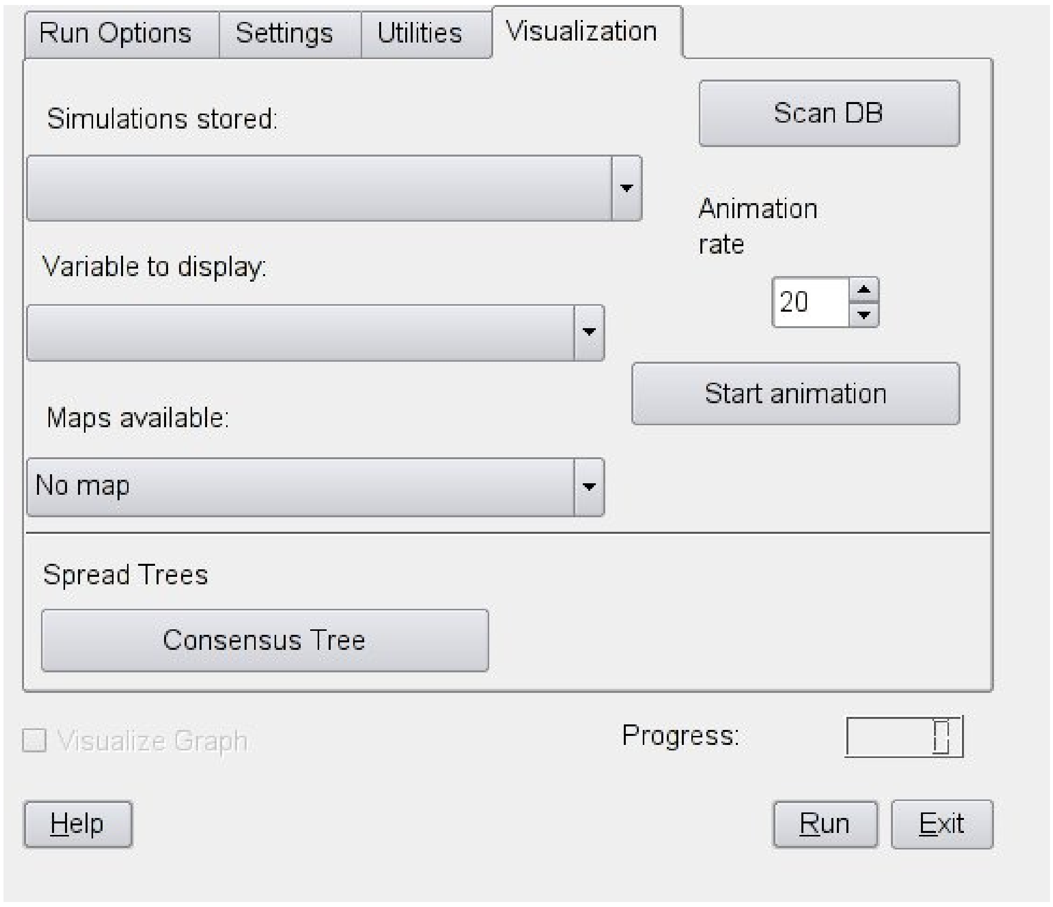


Figure 4.4: Fourth tab of the GUI

Once the `Start animation` button is pressed, a graphical display window pops up, and the simulation is replayed at a speed given by the frame rate set by the user or the maximum speed of the computer (whichever is smaller). In the animation, the nodes of the network are represented by boxes whose volume is given by the number of infected at each node. The node colors are as following: Green for uninfected nodes, and red to blue for infected nodes. bright red for for first infected node with the nodes becoming infected later assuming a color with progressively more blue.

Maps can also be selected from the `Maps available combo box` to be used as background for the network display. The maps must be in the GRASS ascii vectorial format and have coordinates compatible with those given to the nodes of the simulation.

4.5.5 Operation

After all the information has been entered and checked on the GUI, you can press the `Run` button to start the simulation or the `Exit` button. When the `Run` button is pressed, the `.epigrassrc` file is updated with all the information entered in the GUI. If the `Exit` button is pressed, all information entered since the last time the `Run` button was pressed is lost.

Chapter 5

Interpreting the output

The outputs of EpiGrass simulations were designed to be as flexible as possible. Beside the automated report generation which serves as an overview of the model and its results, raw data is also made available for ready utilization from other softwares such as R and Grass.

5.1 Visualization

On the graphical user interface, There is a visualization tab. On this tab the user can choose the simulation dataset, a variable and a map to compose the visualization. When the visualization is started, a display will pop up, and plot the network on the map (if a map is available). As the simulation progresses, the nodes change color from green to red as the localities become infected. Localities that get infected later are assigned a different shades of red which will tend to become blue as the time progresses. This way the sequence of infection remains visible throughout the simulation by means of this color scale.

The visualization module is to be invoked independently of the simulation to review the dynamics of simulation data stored on the database. This way, any previous simulation can be reviewed at any time. It is also recommended that simulataneous visualization be turned off to speed up calculations.

Currently, the visualization module displays only the temporal dynamics of infection with the number of infected individuals in each infected locality is represented by the size of the node object in the network plot.

5.2 Database

The simulated time series are stored in a MySQL table in the database epigrass: E, I, S, incidence (L), together with time, geocode, coordinates and site name. The table is named after the filename of the script and date and time of the simulation.

Epigrass provides an R script for importing the data into R for further analysis and display [1].

5.3 Epidemiological descriptors

At the end of the simulation, Epigrass calculates a set of epidemiological stats. These stats are presented to the user in two ways: as a .csv table that can be imported by R or any other statistical package; and as a written (PDF format) report. Stats include descriptors of the epidemic dynamics at the whole graph level and also node-specific stats:

Graph-level stats:

Epidemic pop size Total number of cases in the whole graph during the simulation.

Epidemic size Total number of cities that had authoctonous transmission during the simulation.

Mean epidemic speed Average number of new localities infected per time step.

Epidemic Duration Time between the first and last case.

Median survival Time to reach 50% of the cities.

Vaccinated Total number of vaccinated individuals.

Quarantined Total number of quarantined individuals.

At the site scale, the report returns for each site i:

Incidence Accumulated number of new cases per time step

Local epidemic size Total number of cases that occurred in the site during the simulation.

Infectious arriving Number of infected individuals arriving per time step

[1] see appendix C

5.4 Network Descriptive Statistics

EpiGrass automatically calculates and displays descriptive statistics about the network structure in Reports 1 and 3.

Basic Numbers

Order (Number of Nodes): The Number of localities the network is composed of.

Size (Number of Edges): The number of transportation routes connecting any pair of nodes.

Eulerian: Can be Yes or No, depending on if the network graph is eulerian or not. An Eulerian graph contains a circuit (Eulerian circuit) which includes all nodes an edges of the graph.

Traversable: Can be Yes or No, depending on if the network graph contains an Eulerian trail, i.e. a trail tht includes all the nodes an edges of the graph.

Hamiltonian Can be Yes or Possibly, depending on if there is a circuit that contains all nodes of the graph (Hamiltonian circuit). This is not the result of an exhautive search for Hamiltonian circuits on the graph (Since this would take a very long time for large graphs). This answer reflects the following theorem:

> If the order O of a graph G is at least 3, and its nodes orders are at least $O/2$ for every node in G, then the graph contains a hamiltonian circuit.

Shortest-path Distribution

In a network there are frequently more than one path from locality A to locality B. Of these possible routes, the shortest path is the most important when dealing with epidemic processes over networks.

A network distance matrix can be calculated whose elements represent the number of edges separating any pair of nodes via the shortest path between them. From this matrix, a histogram of the shortest path lengths can be generated which gives us an idea of give us an idea of how fast an epidemic would spread in our network, if distance was the only factor.

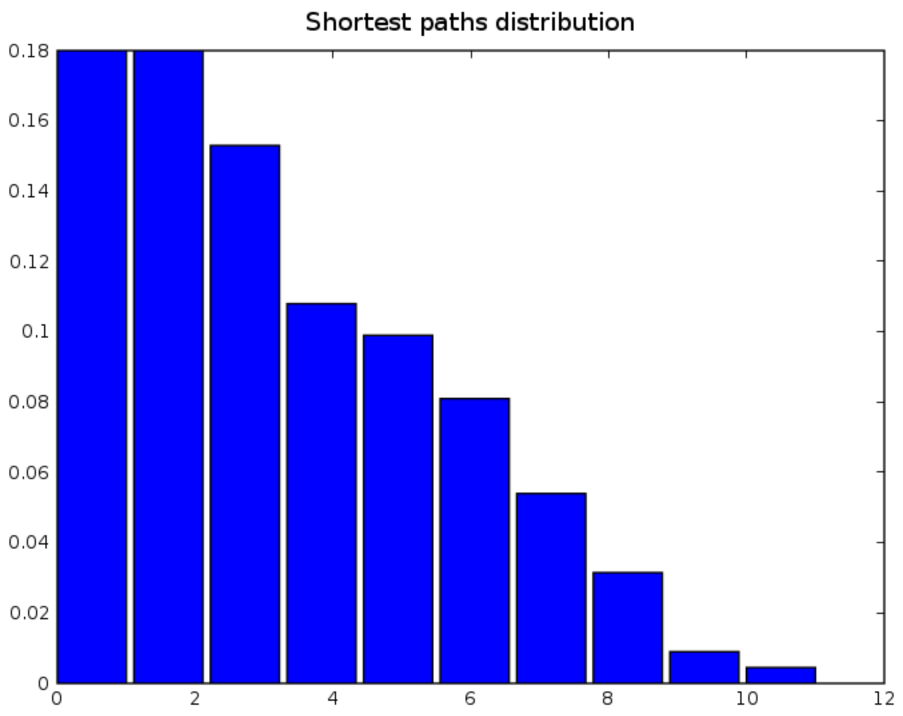


Figure 5.1: Shortest path distribution for the mesh graph (see demo).

Adjacency Matrix

The most basic measure of accessibility involves network connectivity where a network is represented as a connectivity matrix(figure 5.2), which expresses the connectivity of each node with its adjacent nodes.

The number of columns and rows in this matrix is equal to the number of nodes in the network and a value of 1 is given to each cell representing a directly connected pair and a value of 0 to each cell representing an unconnected pair. The summation of this matrix, along its rows or collumns, provides a very basic measure of node accessibility, also known as the degree of a node.

indices = r”””

Number of Cycles

A cycle is a circular path, meaning that it ends where it started, and does not repeat an edge. The index presented here is the maximum number of

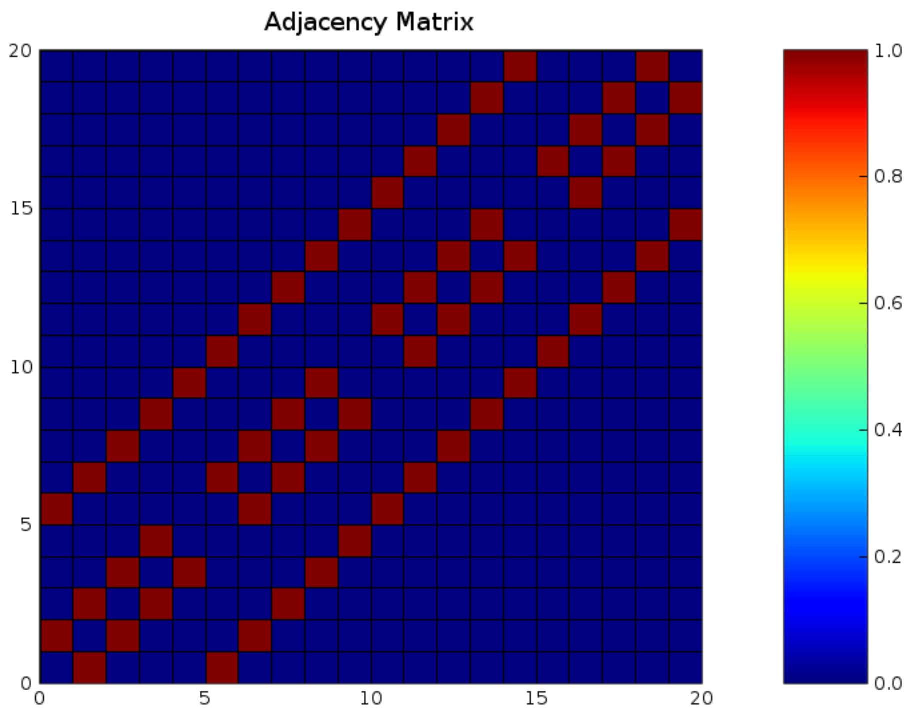


Figure 5.2: Adjacency matrix of a simple mesh network (see demo)

independent cycles in a network.

This number (u) is estimated by knowing the number of nodes (v), links (e) and of sub-graphs (p);

Trees and simple networks will have a value of 0 since they have no cycles. The more complex a network is, the higher the value of u, so it can be used as an indicator of the level of development of a transport system.

$$u = e - v + p$$

Wiener Distance (D_W)

The Wiener distance is the sum of all the shortest distances in the network.

$$D_W = \sum_{i=1}^{v} \sum_{j=1}^{i} D_{ij}$$

where D is the Shortest distance matrix.

Mean Distance($\bar{D}$)

The mean distance of a network is the mean of the set of shortest paths, excluding the 0-length paths.

$$\bar{D} = \frac{1}{v} \sum_{i=1}^{v} \sum_{j=1}^{i} D_{ij} \quad \forall\ D_{ij} \neq 0$$

Network Diameter

The diameter of a network is the longest element of the shortest paths set.

Length of the Network (L)

The length of a network is the sum in metric units (e.g., km) of all the edges in the network.

Weight of the Network(W)

The weight of a network is the weight of all nodes in the graph (W), which is the summation of each node's order (o) multiplied by 2 for all orders above 1.

$$W = 2 \sum_{i} o_i \ \ \forall o > 1$$

Iota Index(ι)

The Iota index measures the ratio between the network and its weighed vertices. It considers the structure, the length and the function of a network and it is mainly used when data about traffic is not available.

It divides the length of a network (L) by its weight (W). The lower its value, the more efficient the network is. This measure is based on the fact that an intersection (represented as a node) of a high order is able to handle large amounts of traffic.

$$\iota = \frac{L}{W} =$$

Pi Index(Π)

The Pi index represents the relationship between the total length of the network L and the distance along the diameter D.

It is labeled as Pi because of its similarity with the trigonometric Π (3.14), which is expressing the ratio between the circumference and the diameter of a circle.

A high index shows a developed network. It is a measure of distance per units of diameter and an indicator of the shape of a network.

$$\Pi = \frac{L}{D}$$

Beta (β) Index

The Beta index measures the level of connectivity in a network and is expressed by the relationship between the number of edges (e) over the number of nodes (v).

Trees and simple networks have Beta value of less than one. A connected network with one cycle has a value of 1. More complex networks have a value greater than 1. In a network with a fixed number of nodes, the higher the number of links, the higher the number of paths possible in the network. Complex networks have a high value of Beta.

$$\beta = \frac{e}{v}$$

5.5 Site Oriented Statistics

Centrality: Also known as closeness. A measure of global centrality, is the inverse of the sum of the shortest paths to all other nodes in the graph.

Degree: The order (degree) of a node is the number of its attached links and is a simple, but effective measure of nodal importance.

The higher its value, the more a node is important in a graph as many links converge to it. Hub nodes have a high order, while terminal points have an order that can be as low as 1.

A perfect hub would have its order equal to the summation of all the orders of the other nodes in the graph and a perfect spoke would have an order of 1.

Theta Index: Measures the function of a node, that is the average amount of traffic per intersection. The higher theta is, the greater the load of the network.

Betweeness: Is the number of times any node figures in the the shortest path between any other pair of nodes.

Appendices

Appendix A

Example of Model Definition Script

```
##################################################################
#
#  EPIGRASS -Model Definition
#  This script describes model and parameters specified
#  by the user.
#  It can be edited by the user directly, by means of a text
     editor.
#  WARNING: No variables may be removed, even if not used by the
      chosen model .
#  Any comments added by the user must be preceeded  by the
     symbol #
#
##################################################################
##################################################################

#================================================================#
[THE WORLD]
#================================================================#
# Here you add information about the files that described the
    world the model is enclosed
# "sites" and "edges" are the files that describe the topology
    of the network (see Documentation)

sites = sitios2.csv
edges = edgesout.csv

#================================================================#
[EPIDEMIOLOGICAL MODEL]
#================================================================#
#model types available: SIS, SIS_s ,SIR, SIR_s, SEIS, SEIS_s,
    SEIR, SEIR_s,
# SIpRpS, SIpRpS_s,SIpR,SIpR_s,Influenza or Custom (see
    documentation for description).
modtype = SIR

#==================================================================#
[MODEL PARAMETERS]
#==================================================================#

# They can be specified as constants or as functions of global
    or
#   site-specific variables. these site-specific variables, are
     provided
# in the sites file. All the numbers given after the geocode (4
    th column)
# are collected into the values tuple.
#   Examples:
#   beta = 0.001
#   beta=values[0] #assigns the first element of values to beta
#   beta=0.001*values[1]

beta = 0.6   #transmission coefficient Lipsitch 2003 for R0 = 3
alpha = 1   # clumping parameter
e = 1          # inverse of incubation period
r = 0.1        # inverse of infectious period
delta = 1      # probability of acquiring full immunity [0,1]
B = 0          # Birth rate
w = 0          # probability of immunity waning [0,1]
p = 0          # Probability of a recovered become infected per
     time step [0,1]


#==================================================================#
[INITIAL CONDITIONS]
#==================================================================#
#   Here, the number of individuals in each epidemiological
#   state (SEI) is specified. They can be specified in absolute
#   or relative numbers.
#   N is the population size of each site.
#   The rule defined here will be applied equally to all sites.
#   For site-specific definitions, use EVENTS (below)
#   Examples:
#   S,E,I = 0.8*N, 10, 0.5*E
#   S,E,I = 0.5*N, 0.01*N, 0.05*N
#   S,E,I = N-1, 1, 0
S = N
E = 0
I = 0

#==================================================================#
```

```
[EPIDEMIC EVENTS]
#===========================================================================#
#   Specify isolated events.
#   Localities where the events are to take place should be
    Identified by the geocode, which
#  comes after population size on the sites data file.
#  All coverages must be a number between 0 and 1.
#  Seed : [('locality1's geocode', variable, n),('locality2's
    geocode',variable, n)].
# N infected cases will be added to locality at time 0.
#  Vaccinate: [('locality1's geocode', time, coverage),('
    locality2's geocode', time, coverage)]
#  Quarantine: [(locality1's geocode,time,coverage), (locality2'
    s geocode,time,coverage)]
#  Keywords may be used instead of the geocode in the vacinate
    variable:
#  all: Apply the same values to all sites
seed = [(355030800,'I',1)] #Sao Paulo
Vaccinate = []
Quarantine = []

# The following events have not yet been implemented
#Screening = (locality, time, coverage) #screening for sick
    people on aiports bus stations
#Vector_control = (locality, time, coverage)
#Prophylaxis = (locality, time, target, coverage, eficacy)

# Intervention acts reducing progression between epidemiological
     states
# target is the name of the modified parameter
# One may specify as many treatments as necessary
#
#Intervention = (locality, time, target, coverage, eficacy)
#Intervention = [('all', 0, 'pp1', 0.3, 0.5),('all', 0, 'pp2',
    0.3, 0.5)]



#============================================================================#
[TRANSPORTATION MODEL]
#============================================================================#
# If doTransp = 1 the transportation dinamics will be included.
    Use 0 here only for debugging purposes.
doTransp = 1

# Mechanism can be stochatic (1) or deterministic(0).
stochastic = 1

```

```
#Average speed of transportation system in km per time step.
    Enter 0 for instantaneous travel.
#Distance unit must be the same specified in edges files
speed =0 #1440  km/day -- equivalent to 60 km/h

#===========================================================================#
[SIMULATION AND OUTPUT]
#===========================================================================#

# Number of steps
steps = 100

# Output dir. Must be a full path. If empty the output will be
    generated on the
# outdata-<date> sub-directory of same path as the model script.
outdir =


# Database Output
# MySQLout can be 0 (no database output) or 1
MySQLout = 1


# Report Generation
# The variable report can take the following values:
# 0 - No report is generated.
# 1 - A network analysis report is generated in PDF Format.
# 2 - An epidemiological report is generated in PDF Format.
# 3 - A full report is generated in PDF Format.
# siteRep is a list with site geocodes. For each site in this
    list, a detailed report is apended to the main report.
report = 0
siteRep = [230440005,355030800]

#Replicate runs
#If replicas is set to an integer(n) larger than zero, the model
     will be run n times and the results will be con-
#solidated before storing.
# if RandSeed is set to 1 the seed will be randomized on each
    replicate
#Replicate mode automatically turn off report and batch options.
Replicas = 0
RandSeed = 0
#Batch Run
#  list other scripts to be run in after this one. don't forget
    the extension .epg
#  model scripts must be in the same directory as this file or
    provide full path.
```

```
#   Example: Batch = ['model2.epg','model3.epg','/home/jose/
    model4.epg']
Batch = []

######################################################################
######################################################################
```

Appendix B

Useful GRASS Commands

In this appendix, a few GRASS commands will be introduced based on their usefulness for the average EpiGrass user.

All the tasks presented in this chapter will be done using the GRASS command line interface. However most of them can also be done through the GRASS GUI. We leave to the user to find out how to perform these tasks using the GUI.

B.1 Exporting a GRASS Vector Layer

Like in any GIS, in GRASS maps are stored as vectorial objects. however these object are stored in binary formats that can't be read by EpiGrass. To be able to use maps from a given GRASS Location dataset in EpiGrass, we must export the vector layers to an ASCII file. Fortunately, the GRASS GIS provides us with the means to do that.

To be able to follow this example, you must have GRASS installed and a Location dataset available on your disk. Once you start GRASS you will be asked to select the location dataset to open. Once you open the desired dataset you will be able to explore your map collection and choose which layers to export.

Listing B.1: Listing vector layers.

```
GRASS 6.0.0~/> g.list vect
```

Listing B.1, shows how to obtain a list of the available vector layers. Once you identify a layer that looks like the on you want to use as a background for your EpiGrass simulations take note of its name. You may want to visualize it before exporting.

Listing B.2: Visualizing a vector layer

```
GRASS 6.0.0~/> dmon x0
GRASS 6.0.0~/> d.vect Layer_name
```

The commands on listing B.2 will plot the chosen vector layer. the first command will open a display window, and the second will plot the layer we chose.

Once we are certain that is the map we want, we can proceed to exporting it.

Listing B.3: Exporting a vector layer to an ascii file

```
GRASS 6.0.0~/> v.out.ascii input=Layer_name output=
    filename.map format=standard
```

It is important that we add the extension `.map` to the file we generate, so that EpiGrass visualization module will recognize it.

The map file generated here must be put in the working directory of your EpiGrass model to be used.

Appendix C

Using R to Analyze EpiGrass' Data

Depending on the size of your network or the number of scenarios being simulated on a given network within EpiGrass, A large amount of data is generated as output. This data is stored, by default on a MySQL database named epigrass.

On this appendix, we illustrate how to access this data on the MySQL server and do some simple analysis using the statistical package R. The R statistical package (`http://www.r-project.org`) is an incredibly resourceful environment for data management and analysis.

C.1 Accessing The MySQL server

Before we can access the database and retrieve our data there are some preliminary steps that must be performed. First and foremost, we need to learn how to start the R system. Just open a console window and type `R` as shown on listing C.1.

Listing C.1: Starting R

```
$ R
R : Copyright 2004, The R Foundation for Statistical Computing
Version 2.0.1  (2004-11-15), ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
```

```
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

>
```

Very well, Now you are running R. R has a modular architecture that allows the user to specify the tools he/she desires to use on a given session. In order to access the MySQL database server and retrieve our data, we will need some special tools from R's vast collection of modules (also called packages). Listing C.2, shows how to load the packages needed.

Listing C.2: Loading required packages.

```
> library(DBI)
> library(RMySQL)
```

To connect to a database server we need to tell R the type of server we will be connecting to, and open a connection to it (listing C.3).

Listing C.3: Specifying the type of server and opening a connection.

```
> drv <- dbDriver("MySQL")
> con <- dbConnect(drv, username='epigrass',password='epigrass',
    host='localhost',dbname='epigrass')
```

Once we have a connection to the database it works as a two-way communications pipeline between R and the MySQL server. Through this connection we can send SQL commands to the Server and retrieve the results of these commands.

Fortunately, the RMySQL package has many common SQL statements packed into easy to remember commands. Listing C.4 shows us how to find out the tables available at the epigrass database. This is a very useful command for us because the results of each simulation completed in EpiGrass is stored in a separate table whose title is contains a reference to when it was ran. So, after running a few simulations with the demo model mesh, we will end up with a list of tables similar to the one shown on listing C.4.

Listing C.4: Listing existing tables.

```
> tables<-dbListTables(con)
> tables
 [1] "mesh_Mon_Feb_14_110816_2005"    "mesh_Mon_Feb_14_111026_2005
    "
 [3] "mesh_Tue_Feb_15_144139_2005"    "mesh_Tue_Feb_15_171743_2005
    "
```

```
5 [5] "mesh_Tue_Feb_15_172037_2005"     "mesh_Tue_Feb_15_172134_2005
    "
```

After we identify which set of data(table) we want to work with, we can read it into a data frame, a very versatile data structure of R.

```
1 > results<- dbReadTable(con, tables[2])
2 > names (results)
3 [1] "geocode"    "time"       "name"       "lat"        "longit"
    "E"
4 [7] "I"          "S"          "incidence"
```

On line 1 of listing C.1, we read the second table of our database into a data frame object called `results`. The `names` command, lists the names of the variables contained in that data frame.

C.2 Visualizing the Data

Once we have have the data inside R there countless ways in which we can manipulate and visualize it. For the first plot we will need to load another package called `lattice`.

```
1 > library(lattice)
2 > xyplot(I+E+S~time|name, type="l", data=results)
```

Bibliography

Anderson, R., May, R., and Anderson, B. (1992). *Infectious Diseases of Humans: Dynamics and Control*. Oxford University Press.

Barrett, C., Eubank, S., and Smith, J. (2005). If smallpox strikes portland... *Scientific American*, www.sciam.com:41–49.

Bivand, R. (2000). Usando a linguagem de análise de dados estatísticos r em arquivos de banco de dados do grass 5.0 sig. *Computers & Geosciences*, 26:1043–1052.

Casman, E., Fischhoff, B., Palmgren, C., Small, M., and Wu, F. (2000). An integrated risk model of a drinking-water-borne cryptosporidiosis outbreak. *Risk Anal*, 20(4):495–511.

Daley, D., Gani, J., and Cannings, C. (2001). *Epidemic Modelling : An Introduction*. Cambridge University Press.

Diekmann, O. and Heesterbeek, J. (2000). *Mathematical Epidemiology of Infectious Diseases : Model Building, Analysis and Interpretation*. John Wiley & Sons.

Glass, K. and Grenfell, B. (2004). Waning immunity and subclinical measles infections in England. *Vaccine*, 22(29-30):4110–6.

Grenfell, B., Bjørnstad, O., and Kappey, J. (2001). Travelling waves and spatial hierarchies in measles epidemics. *Nature*, 414(6865):716–23.

Isham, V. and Medley, G., editors (1996). *Models for Infectious Human Diseases : Their Structure and Relation to Data*. Cambridge University Press.

Keeling, M. and Grenfell, B. (1997). Disease extinction and community size: modeling the persistence of measles. *Science*, 275(5296):65–7.

Meyers, L. A., Pourbohloul, B., Newman, M. E. J., Skowronski, D. M., and Brunham, R. C. (2005). Network theory and SARS: predicting outbreak diversity. *J Theor Biol*, 232(1):71–81.

Zaric, G. S. and Brandeau, M. L. (2002). Dynamic resource allocation for epidemic control in multiple populations. *IMA J Math Appl Med Biol*, 19(4):235–55.

Index