naadsmtools

A set of scripts and code to assist with the analysis of simulation results from the North American Animal Disease Simulation Model. Repository is online at https://github.com/afidd/naadsmtools.

NAADSM

The North American Animal Disease Simulation Model simulates spread of diseases among agricultural units on a landscape. It has two parts, a backend called NAADSM/SC (where SC stands for supercomputer) that is run from the command line, and a frontend, which is a Windows graphical user interface (GUI). The tools developed here have been targeted to work with NAADSM release 3.2.19, although we suspect that these should also work with more recent NAADSM 4.x releases.

These tools interpret NAADSM all-units-states output

We are specifically interested in output data from NAADSM that indicates the state of every unit on each day. From the NAADSM Windows GUI, this can be produced by selecting "Scenario Parameters -> Output Options", and then selecting the checkbox under "Daily unit states: Write a plain text file containing daily states for all units". Alternatively, NAADSM/SC can be configured to produce different types of output data, by specifying the relevant tags in a NAADSM parameter scenario XML file. The following entry in a scenario file will print the disease state of every unit (farm) for every day in a simulation:

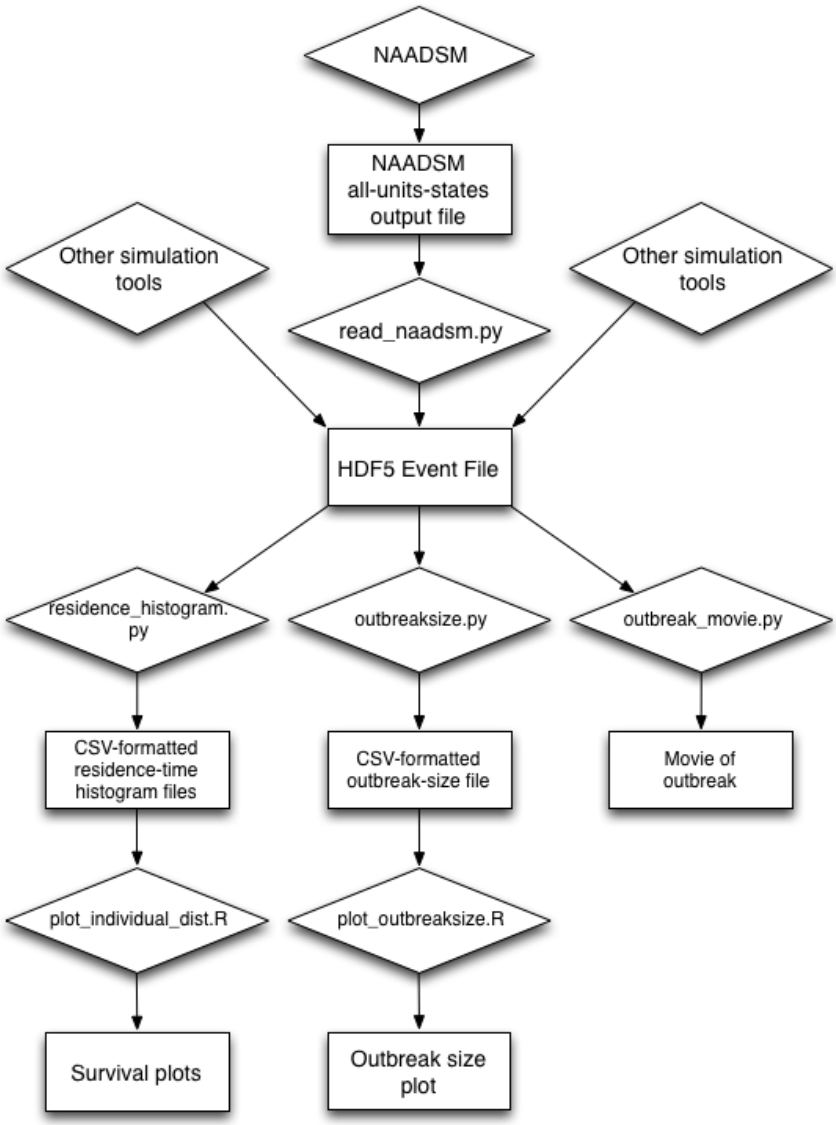
```
<output>
  <variable-name>all-units-states</variable-name>
  <frequency>daily</frequency>
</output>
```

It is these particular output data that the naadsmtools scripts are intended to analyze. In some of what follows, we refer to this all-units-states output file as a "NAADSM Trace". (Note: the formats of the daily trace data produced by NAADSM/SC and the NAADSM GUI are slightly different, although they contain the same basic information.) The first step is to process the NAADSM Trace data to identify the daily changes of state, and to record those events in a file formatted with Hierarchical Data Format, (HDF5). Subsequent scripts read that event data to create survival analysis plots and movies which augment analysis done within the NAADSM graphical user interface. These scripts also serve as examples, in Python and R, for how to read the HDF5 files with event data. The HDF5-formatted event file serves to as an intermediate data format that we can write results to from other simulation programs which specifically simulate individual transitions among states at arbitrary times (as opposed to updates of state at a set of discrete times).
By defining a common data format for two different modeling paradigms, we can build downstream tools capable of interpreting either set of results.

The tools consist of a mix of Python and R scripts for data processing, analysis and visualization. Information on Python and R installations, and associated packages, are described below in the Appendix.

List of tools

This diagram demonstrates how the naadsmtools interrelate and how data and analyses proceed through the pipeline.



Makefile

The individual tools described below are bundled up in a Makefile that can be run to process a NAADSM all-states-units output file and produce a number of analyses and plots. This requires having the "make" utility installed (which is standard on unix-like operating systems, but not on Windows). "make" is typically used for compiling source code; in this case, it is used to coordinate data processing by defining the dependencies of different files on one another. The Makefile could be replaced by a suitable shell script that carries out the same basic set of commands.

Using the Makefile requires a NAADSM Trace file as input, and optionally can be configured with an ID to label the various output and a name for the HDF5-encoded event data file. A call to "make" might look like this:

make NAADSMTRACE="my_naadsm_trace.out" ID="MyScenario" NAADSMDATA="my_naadsm_data.h5"

As described in the subsections below, this will produce the following outputs:

- my_naadsm_data.h5
- susceptible_MyScenario.csv
- susceptible_MyScenario.pdf
- latent_MyScenario.csv
- latent_MyScenario.pdf
- clinical_MyScenario.csv

- clinical_MyScenario.pdf
- outbreak_hist_MyScenario.csv
- outbreak_hist_MyScenario.pdf

Shell script: pipeline.sh

As an alternative to using the Makefile (without benefitting from make's dependency analysis), a shell script "pipeline.sh" is also provided showing the basic workflow. As with the Makefile, the variables NAADSMTRACE, ID and NAADSMDATA must be provided, although with a slightly different command-line format:

NAADSMTRACE=my_naadsm_trace.out ID=MyScenario NAADSMDATA=my_naadsm_data.h5 ./pipeline.sh

This will produce the same set of output files listed above for the Makefile.

convert_naadsm_xml.py

NAADSM/SC uses as input XML files specifying model parameters and unit properties, rather than the composite scenario files used by the NAADSM GUI. The NAADSM GUI can export the units and parameters XML files associated with a NAADSM scenario. Under some circumstances, however, the resulting XML files will be encoded in UTF-16, while the NAADSM/SC program expects UTF-8 encoded XML files. The convert_naadsm_xml.py script converts xml files to a UTF-8 encoding suitable for NAADSM/SC. (This script is only required if you find that NAADSM has exported in UTF-16.)

Usage:

python convert_naadsm_xml.py -i input_filename -o output_filename

converts a UTF-16 encoded naadsm XML file (input_filename) to an equivalent UTF-8 encoded file (output_filename)

read_naadsm.py

read_naadsm.py reads the all-states-units output of a NAADSM/SC run, and produces an HDF5-encoded events file for further processing, as described above.

Usage:

python read_naadsm.py --input naadsm_outputfile --output hdf5_events_file

reads the all-states-units data in the naadsm_outputfile and writes the corresponding hdf5_events_file.

residence_histogram.py

From an HDF5-encoded event file, residence_histogram.py computes histograms for the number of days spent within each of the susceptible, latent and clinical stages. Separate csv-formatted files are produced for each stage. These files are produced in order to carry out survival analysis [O. Aalen, O. Borgan, and H. Gjessing. Survival and event history analysis: a process point of view. Springer Science & Business Media, 2008][https://cran.r-project.org/web/packages/survival/index.html].

Usage:

python residence_histogram.py --input naadsm_event_data_file --id ID

produces the files susceptible_ID.csv, latent_ID.csv, and clinical_ID.csv, where "ID" is replaced by the label specified on the command line. (If no ID is specified, these are simply written as susceptible.csv, latent.csv and clinical.csv, respectively.)

plot_individual_dist.R

Given the csv-formatted susceptible/latent/clinical histograms generated by residence_histogram.py, plot_individual_dist.R computes and plots the survival fraction associated with each of these states.

Usage:

R --no-save --args "susceptible_ID" < plot_individual_dist.R
R --no-save --args "latent_ID" < plot_individual_dist.R
R --no-save --args "clinical_ID" < plot_individual_dist.R

where ID is replaced by the label in the associated filenames (e.g., MyScenario).

plot_outbreaksize.R

plot_outbreaksize.R plots the outbreak size distribution, given the csv-formatted outbreak size data provided by outbreak.py.

Usage:

R --no-save --args "outbreak_hist_ID" < plot_outbreaksize.R

where ID is replaced by the label in the associated filenames (e.g., MyScenario).

outbreak_movie.py

outbreak_movie.py creates an animation of the outbreak stored in an HDF5-encoded event file. It requires that the ffmpeg encoder be installed separately on the system, which will produce a movie in .mp4 format.

Usage:

python outbreak_movie.py -i input_file -u unit_xml_file -o output_file -I initial_sites

where:

- input_file is an HDF5-encoded event file (e.g., produced by read_naadsm.py)
- unit_xml_file is a NAADSM unit XML file (specifying herd/unit locations)
- output_file is the desired name of the resulting .mp4 file
- initial_sites is a comma-separate list of unit IDs indicating which premises were initially infected

Appendix

Python might already be installed on your system (it is sometimes used for some systems administration tasks), but we recommend installing a separate version with additional functionality included. The free Anaconda Python distribution (https://www.continuum.io/content/anaconda-subscriptions) is one such solution that we can recommend. Anaconda Python with its own python package manager named "conda". Some packages are installed by default, but others can be optionally added with conda. Execute the following commands to install the pyproj and docopt packages used by some of the naadsmtools:

```
conda install pyproj
conda install docopt
```

R might already be installed on your system too. If it is not, it can be downloaded from CRAN (https://cran.rstudio.com);

any additional packages needed for use with naadsmtools can be installed by running "make rpackages" (a defined target in the naadmtools Makefile) or an equivalent command-line installation.