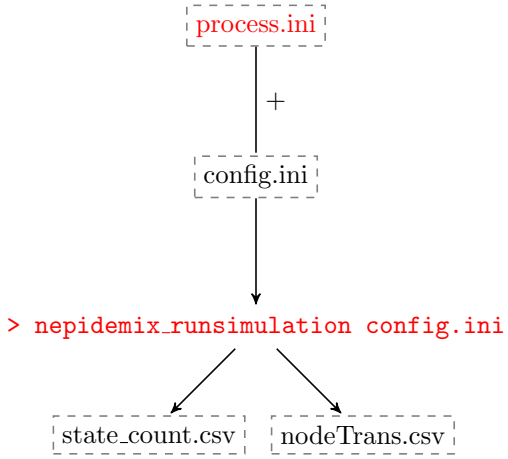
Modelling Workshop - Disease and Testing NepidemiX

March 30, 2014



1 Logging in and preparing the computer

1.1 Create a working directory

Create a new folder on the nepidemix drive. Name it after yourself. You will be working in this directory.

1.2 Copy tutorial data files to your working directory

Before we can start the tutorial you will need to copy some simulation data to your working folder. This can be found in the folder `tutorial_files`.

Select the files `uniform_simulation.ini` and `motivated_simulation.ini` as well as the folder `network` and copy them all to your working folder.

The `.ini` files are the NepidemiX simulation configuration files (you will have a closer look at them later). The `network` directory contains the seed network for the simulations.

You can skip copying the other two directories (`scripts` contains a script we will be using later, but it is already in the path; `solution` actually contains finished and documented versions of the process files you will be building. Good to know, go there for a peek if things seem hopeless - beats banging your head to the keyboard).

1.3 About the software we'll be using

NepidemiX is still in an infant stage as software goes; to keep its complexity at a minimum it is currently a command line only software, configured using text files.

Therefore you will need to use a text editor to write the process files, and the terminal to run the software. The resulting data files can be loaded into a spreadsheet application such as Excel.

1.3.1 The terminal

Don't worry if you are not familiar with running programs from the terminal. These instructions point out what you have to do, explaining each step. You will only have to do three things 1) Change directory to your working directory; 2) execute the nepidemix command; 3) execute a script that extracts incidence data.

Let's do step 1) already now! The terminal that popped up when you launched the tutorial has its current directory set to the nepidemix drive. We will need to change it to your tutorial working directory (the one you created above).

The command used to change directory is `cd`.

So, type `cd` followed by a space, and the name of your working directory. To make things easier, a nice trick is to simply drag and drop your working folder onto the terminal (after you wrote `cd` and space). This will fill in the full path. Then press enter/return, and you should have changed directory.

To check this you can type `ls` followed by enter/return. This is the *list* command and will give you a list of files and folders. You should see the names of the files you copied earlier.

If this is not the case, or the cd command gave you any error message, let us know, and we will come to help you out!

2 Writing the first testing process: uniform testing

Now you are ready to write your first process, and we will start off with the uniform testing. You will write the process description in your editor so you should start it up.

In essence the process description will encode the state space and the transitions laid out in Figure 1. Together with the NepidemiX simulation configuration file it makes up the input we will be using.

The simulation configuration file is provided (you copied it earlier) and you should be able to leave it as it is. However if you to have a better understanding of its role it is quite good to have a look inside it. Which is what we will do next.

2.1 Inspect `uniform_simulation.ini` using an editor

This is one of the files you copied to your working directory. Go ahead and open it up.

Don't change any values, just have a look in it. It looks very similar to when you were shown in the lecture, but with comments explaining every option.

There are section headings written in square brackets: `[Simulation]`, `[NetworkParameters]`, `[ProcessParameters]`, and `[Output]`. In each of the sections there are option names and values on the form **option** = **value**.

Starting with the `[Simulation]` section at the top of the file, most of the settings there are related to how to initialize the network (`network_func`, `node_init`, `edge_init`) and in which mode NepidemiX should run (`process_class`).

The settings `iterations` and `dt` are of interest however. The first one tells how many times NepidemiX should go over all nodes and perform the process rules. The second states the unit time difference between each iteration. (That is this particular simulation will run for 2000×0.01 time units [years].)

Next, `[NetworkParameters]` section only contains the file name to the prepared network. In another type of simulation we may have generated a network at the beginning, and this section would have had many more options related to that process. But not now.

Move on and have a closer look at the values in the `[ProcessParameters]` section (reprinted here):

```
1 [ProcessParameters]
2 # Path and file name to process specification.
3 # If you have named your file something else than
4 # random_process.ini, change it here.
```

```
5  file = uniform_process.ini

7  # These are the process parameters as named in
8  # the tutorial description, and used in the
9  # process description file.
10 # If you have used another name change the
11 # corresponding parameter name here.
12 beta = 6.0
13 lambda_A = 0.64
14 lambda_C = 0.016
15 tau_r = 0.1
16 delta_A = 0.018
17 delta_C = 0.1
18 delta_T = delta_A
```

This is where your process related settings are stored. The values of any parameters used in your process description must be defined here. Thus, if you for some reason use other names for your parameters later than the ones used here you will need to change the names in this section.

Moreover, the option `file = uniform_process.ini` gives the expected name of your process file. If you name it anything else than `uniform_process.ini` you must change this value.

However, if you follow the instructions and recommended naming below you should not have to change anything.

Finally, the `[Output]` section lists many options related to how data should be saved. Types, and how often sampling should be made. One setting worth attention here is `base_name`. Its value (`uniform_testing`) will be prefixed to all the data files generated by the simulation. If in addition `unique` is also set to yes a time stamp will be added as well (as it is now, each new simulation will overwrite the previous one).

That concludes your brief tour of the configuration file, let us know if you have any questions.

Now, close down the file without saving any changes you may have made while poking around.

2.2 Create a new file called `uniform_process.ini`

In your editor select new file and make sure it is saved in your working directory.

You are now ready to write the process file. It will have three sections: `[NodeAttributes]`, `[NodeRules]`, and `[MeanFieldStates]`. The first specifies the state space, the second state transitions, and the third, while not strictly needed for our purposes is a format requirement.

2.2.1 Write the `[NodeAttributes]` section

Start by putting in the header `[NodeAttributes]` in top of your new file. Note that NepidemiX input files are case sensitive.

Each line in this section specifies a node attribute and its possible values. As you remember from the problem description we have three node states: *disease*, *treatment*, and *motivation*[1]. Each with its own set of possible values.

With the possible values of *disease* being S, A, and C this attribute is declared as `disease = S, A, C` in the file. Go ahead and write that row now.

For *treatment* the possible symbols are U or T yielding `treatment = U,T`, and for *motivation* N, or Y, which of course is declared using `motivation = N, Y`.

Your `[NodeAttributes]` section should now look like:

```
1 [NodeAttributes]

3 disease = S, A, C

5 treatment = U, T

7 motivation = N, Y
```

This in effect defines a three dimensional grid with twelve points, each denoting a unique state. In the next section we will show how to write the transition rules between those states, or between sets of states.

Note by the way that attribute values does not have to be single letters, any word will do. In this tutorial you should use the specified ones however.

2.2.2 Write the `[NodeRules]` section

Now it is time to specify the rules governing how nodes in a specific state (or set of states) move into another state. That is to specify the actual process.

The heading for this section is `[NodeRules]` so please go ahead and put that on a new line in the file.

As you will remember from the lecture there are two processes: a disease process, a treatment process. We also have to take death (resetting to a ground state) into account.

All transitions are mapped out in Figure 1.

[1] You may now be wondering why *motivation* is part of the attributes for nodes in the uniform testing model, where it does not play any role. The answer is that as we use the same prepared network (in order to be able to compare the methods) motivation will be present there and so needs to be declared and used here. However, as you will see the value of this attribute is never used in this testing model and not active.

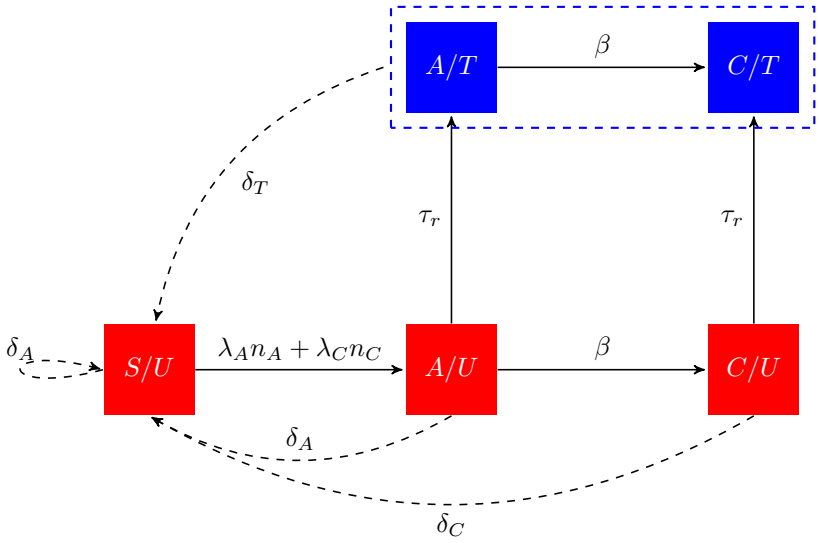


Figure 1: Uniform testing model transitions. Disease attribute values: S, A, C; Testing attribute values: U, T.

We will jump in at the deep end and write the rule of infection. This is the edge in Fig. 1 between S/U and A/U, having the transition probability $\lambda_A n_A + \lambda_C n_C$.

Encoded as a process rule in NepidemiX it looks like this

```
1 {disease : S} -> {disease : A} = lambda_A * NN({disease : A,
      treatment : U}) + lambda_C * NN({disease : C, treatment :
      U})
```

Let's break it down; you can go ahead and put it in the file while we do so:

The first part, {disease : S} declares the rule to be valid for all nodes with the *disease* attribute set to *S*, irrespectively of what their other attributes are set to. Following it is the symbol -> ('minus' 'greater than'), an arrow pointing to the attribute updates that should be made through this rule. In this case it is {disease : A} meaning that *disease* is changed to *A* through this rule (again, all other attributes are left untouched).

Next is an equal sign, =, and *on the right hand side is an expression for the probability in unit time of the rule to be carried out.*

In our case this is lambda_A * NN({disease : A, treatment :U})+ lambda_C * NN({ disease : C, treatment : U}) and corresponds directly to the expression $\lambda_A n_A + \lambda_C n_C$ that we saw during the lecture. Here `NN` is a special function giving the number of neighbours with a certain set of attributes set. Thus NN({disease : A, treatment :U}) will give the number of neighbours with *disease* set to *A* **and** *treatment* set to *U*.

That is it. Please double check that you have a line in your file corresponding to the above (and only one line, NepidemiX does not allow line breaks in rules, even though they are printed so here to fit on a page).

That was actually the hard part. Once you have grasped the format of this rule, the remaining will be easy, as they are close variations on a theme.

The full `[NodeRules]` section encoding all relevant rules looks like this:

```
1  [NodeRules]

3  {disease : S} -> {disease : A} =  lambda_A * NN({disease : A,
       treatment : U}) + lambda_C * NN({disease : C, treatment :
       U})

5  {disease : (A,C), treatment : U} -> {treatment : T} = tau_r

7  {disease : A} -> {disease : C} = beta

9  {treatment : T} -> {disease : S, treatment : U} = delta_T

11 {disease : C, treatment : U} -> {disease : S, treatment : U} =
       delta_C

13 {disease : (S,A), treatment : U} -> {disease : S, treatment :
       U} = delta_A
```

It is uncommented here (for brevity), but feel free to add comments to your own file to remind yourself of what the rules are for. A comment line starts with the symbol `#`.

As you can see there are five additional rules, and we will deal with them in order written. The first one, on line 3 is the infection rule described above. Then follows the testing rule (line 5). It selects untreated nodes in either the acute or chronic disease stage and moves them from untreated to treated. Uniform testing means there is a constant probability of getting tested, `tau_r`. We make no distinction between those in the acute and chronic stages with respect to testing, so this rule can be combined for both corresponding edges in Fig. 1. It is done by {disease : (A,C), treatment : U} selecting nodes with disease attribute set to either *A* **or** *C* **and** *treatment* attribute set to *U*. Note also that the state update for this rule, {treatment : T}, only updates treatment but leaves the *disease* attribute unmodified.

Line 7 then encodes the disease progression from acute to chronic stage. This is again with constant probability in unit time, `beta`. This rule is possibly the simplest kind; valid for nodes with *disease* set to *A*, and changing that same attribute to *C*. Such a rule matches a set of states, as it is valid no matter what the other two attributes are set to.

The remaining three rules are death rules. Which, in this case of a static network

simply means that the node state is reset to a healthy, untreated ground state. That is *disease* is set to S, and *treatment* to U.

The first of these rules, on line 9, is for treated individuals. Figure 1 states that treated nodes have a death rate of δ_T no matter what disease state they are in. This is also what the rule on line 9 encodes. It matches all treated nodes no matter what values their other attributes happens to have, and changes all attributes to the ground state with probability rate `delta_T` (in unit time).

Next up are the untreated and chronically infected that are put in the ground state with probability `delta_C`.

Finally, the last rule captures those either susceptible, or acutely infected and untreated, putting them back in the ground state with probability `delta_A`. This last rule does in fact encode a couple of subtle issues that you should note. First, it encodes an assumption that susceptible and acutely infected have the same probability to die. This is not unreasonable, but is through this rule encoded. There are other ways of doing this, in fact people on treatment have the same probability to die in unit time as susceptible and acutely infected. This is because `delta_T = delta_A` in the simulation configuration file. (Look back in the [ProcessParameters] section in `uniform_simulation.ini` to verify.) However, the assumptions are very differently encoded. In the second case it is a simple matter of setting two parameters equal. In the former the behaviour of the process is encoded to be the same for both. There is no actual reason for making the distinction here, other than to point it out to you.

Secondly, you may notice that the state *disease* $= S$ and *treatment* $= U$ is both source and target for some nodes selected by this rule. That is: it is a self map. There is no real reason for including the selection of this subset in the rule, as they will be reset to the same state. This is true, and in fact the simpler rule {disease : A, treatment : U} −> {disease : S, treatment : U} = delta_A would have resulted in the same behaviour for this simulation.

However, in general when there are further attributes spanning the state space (such as *motivation* in the next testing model) involved it is good to have a catch-all case such as this. The reason being that some attributes may not have any influence on the event of death, but should still be reset, and it is otherwise easy to create forgotten 'pockets' of these in state space.

Now, after this lengthy explanation you will have done the bulk of the work. If you have the above lines filled out in your file, and understand what they represent the rest of the tutorial, will be a breeze.

2.3 Write the [MeanFieldStates] section

The final section in any process definition is called `[MeanFieldStates]`. Here all tracked states are declared. This currently serves two purposes in NepidemiX: First, it allows the fraction of the total population in the declared states to be

used in probability rules (similar to the function `NN` that gives the nearest neighbours) which is something we do not do in this tutorial. Secondly, all tracked state will be saved in the output files defined in the simulation configuration. For this tutorial we will simply tell NepidemiX to track the total number of nodes in all states, as well as the number of nodes in all other possible (fully specified) state-sets. This is done by simply declaring {} which will match every possible state.

So, make sure your section says:

```
1 [MeanFieldStates]
2 {}
```

That is everything. To recapitulate, here is the whole process description in one go:

```
1  [NodeAttributes]
2  disease = S, A, C
3  treatment = U, T
4  motivation = N,Y

6  [NodeRules]
7  {disease : S} -> {disease : A} =  lambda_A * NN({disease : A,
       treatment : U}) + lambda_C * NN({disease : C, treatment :
       U})

9  {disease : (A,C), treatment : U} -> {treatment : T} = tau_r

11 {disease : A} -> {disease : C} = beta

13 {treatment : T} -> {disease : S, treatment : U} = delta_T
14 {disease : C, treatment : U} -> {disease : S, treatment : U} =
       delta_C
15 {disease : (S,A), treatment : U} -> {disease : S, treatment :
       U} = delta_A

17 [MeanFieldStates]
18 {}
```

2.4 Double check that you have saved the file

Do it! (The name should be `uniform_process.ini`, and the location your working directory.)

You are now ready to run your first simulation!

3 Run the uniform testing simulation

In your working directory, there should now be a file called `uniform_simulation.ini` that you copied previously, and another one called `uniform_process.ini` that you just created. (There are other files as well, but let's ignore them for now.) You are ready to execute the simulation with NepidemiX.

To do so go to the terminal window and type the following

```
1 nepidemix_runsimulation uniform_simulation.ini
```

Then hit enter/return. If you have typed everything correctly nepidemix will run (again, a handy tip: pressing the tab key when typing a command or file name in the terminal will auto-complete it as far as possible), if not the terminal or nepidemix will complain. If so try to find the error, or give a shout and we will come running to help you.

Once the program executes NepidemiX will take the simulation file and create (in this case load) a network structure, set up output structures, and load values for the parameters specified. It will also load the file `uniform_process.ini` and create a set of rules from it. (How does it know what process file to load, that is specified in the simulation file, remember the `file` setting in the `[ProcessParameters]` section we saw earlier?)

Then NepidemiX starts the simulation, which basically means that it goes through the specified number of iterations, each time changing node states according to the process rules you just specified.

If there is an error in the process description NepidemiX will complain. In some cases it does so politely, in other cases it shows its Beta status and simply dumps a lot of unfriendly information. In either case, you are back at the command prompt, and we are here to help. Wave your hands.

Most likely however your file will work, and all NepidemiX will be chugging along happily. You know this because it will print a whole lot of (still not very helpful) information. The printouts won't have much meaning for us now, but the last thing NepidemiX prints before starting to run the process on the network is a line ending in `Process will leave topology constant?: True`. If you see this, you will know that NepidemiX has reached as far as launching some kind of simulation. You will also not be shown a new command prompt again (until the simulation is finished of course). One last thing: keep an eye on NepidemiX the first 20 seconds or so, if the simulation finishes that quickly there's something wrong that was not detected by the parser[2]. If this happens while you do the tutorial it most likely means that your file name does not match the one given in the `file` option in the `uniform_simulation.ini` file. Double check that you use the file name suggested above, or change the value of the option to match whatever file name you are using.

[2]Known issue, beta status, we do apologize.

NepidemiX is now running 2000 iterations on a network of 10000 nodes and save a bit of data[3]. This will take ten minutes or so. (More if you keep facetweeting so hands off that browser!)

Now is an Excellent time to go for a walk or have that coffee you've been dreaming of the last hour. See you in a while!

4 Output and Analysis

Welcome back!

Before we go on to writing and running the motivated testing simulation we'll look at the data, and how to load it into Excel.

4.1 Look in your working directory

Hopefully everything worked out and your simulation is finished. You should now have two new files in your working directory: `uniform_testing_state_count.csv` and `uniform_testing_nodeTrans.csv`. These are two files created by NepidemiX. Both are in comma separated value (csv) format. This is a simple text format that is easily imported into other programs such as Excel.

We will not be using either of the files listed directly (but rather run a second script extracting incidence information); for the interested, an overview of the files follows. If you'd rather get on with the tutorial you can easily skip to the next section.

The file `uniform_testing_state_count.csv` contains the number of nodes in each state (combinations of attribute values) for every time step. This data can for instance be used to plot prevalence over time.

We will not be doing that in this tutorial as the number of iterations are too few to give a reliable indication in prevalence. However if you like you can start up Excel and take a look at the file. The first row contains header information. This is a quite technical format, but you will see that the names and letters we use as attribute values are present, indicating which attribute values make up the state (or set of states) counted in each column.

The file `uniform_testing_nodeTrans.csv` is somewhat more complicated. It contains a list of the number of nodes changing from one state (rows) to another (columns) in a given time step (first column). Imported into Excel this yields a quite large matrix.

[3]What determines these values? Well, the 2000 iterations is controlled by the `iterations` setting in the files `uniform_simulation.ini`, the network size is given by the network, and as we have prepped it for this tutorial, you will have to trust us there.

Selecting the right combination of rows and columns in this matrix would allow you to compute the incidence, however this is a quite laborious procedure, so you will simply run a script that destiles that information into a new csv file instead (then you will work on that information).

4.2 Run the command `extract_incidence.py`

In your terminal window, type and run (press enter/return) the command

```
1  extract_incidence.py
```

This is a small script we have provided for this workshop. It will go through all files ending in `nodeTrans.csv` and for every time step in the simulation extract the number of nodes that has changed disease state from susceptible (S) to acute (A). That is the incidence per time step (in this case 3.65 days as our `dt` is set to 0.01 years).

Once you have run it you will see a new file in your working directory called `uniform_testing_incidence.csv`. This file contains a very simple table; the first column is a point in time (in years), and the second column is the number of new cases of HIV happening during the corresponding simulation step.

4.3 Import `uniform_testing_incidence.csv` in Excel

You should now import (or load) the csv file into Excel. If you have the program open already, you can use the menu option *Data - Get External Data - Import Text File*.

Navigate to your working directory and select `uniform_testing_incidence.csv`.

In the dialog that pops up: Make sure Delimited is selected and click 'Next', then choose 'Comma' as delimiter and click Finish.

4.4 Plot the data

Select the columns and use a scatter plot to view the data. You can do this by moving to 'Charts' and selecting Scatter plot - Smooth Lined Scatter.

This will show a very noisy dataset. This is the incidence per iteration for the simulation you just ran.

4.5 Investigate the data using a moving average

We'll be using a moving average to look at the data trend. This computes a trend curve by averaging the data points in a window around every point.

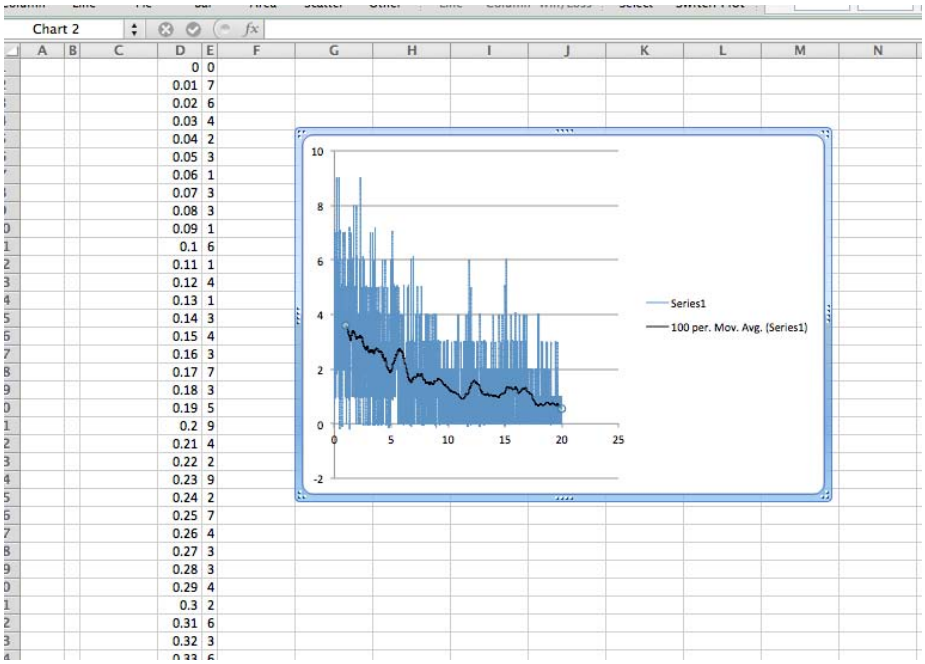


Figure 2: Excel plot of uniform testing incidence data.

The quickest way of adding a trend line to your plot is by right-clicking on the series plot and selecting 'Add Trendline'. You can try out linear and polynomial regression if you like. In the end choose a moving average and a period of at least 100.

The result should look similar - but not exactly like (stochastic, remember?) - to the screen shot shown in Figure 2.

We will now move on to implementing a motivated testing process, running it, and then you will plot that one as well.

5 Writing the second testing process: motivated testing

Now it is time to implement the motivated testing process. The whole thing is depicted in Figure 3. As you can see it is in essence only two different versions of the uniform testing that we just implemented, plus the motivation process causing the boost in testing rate.

As you already have the experience from writing one process we will get right on it here.

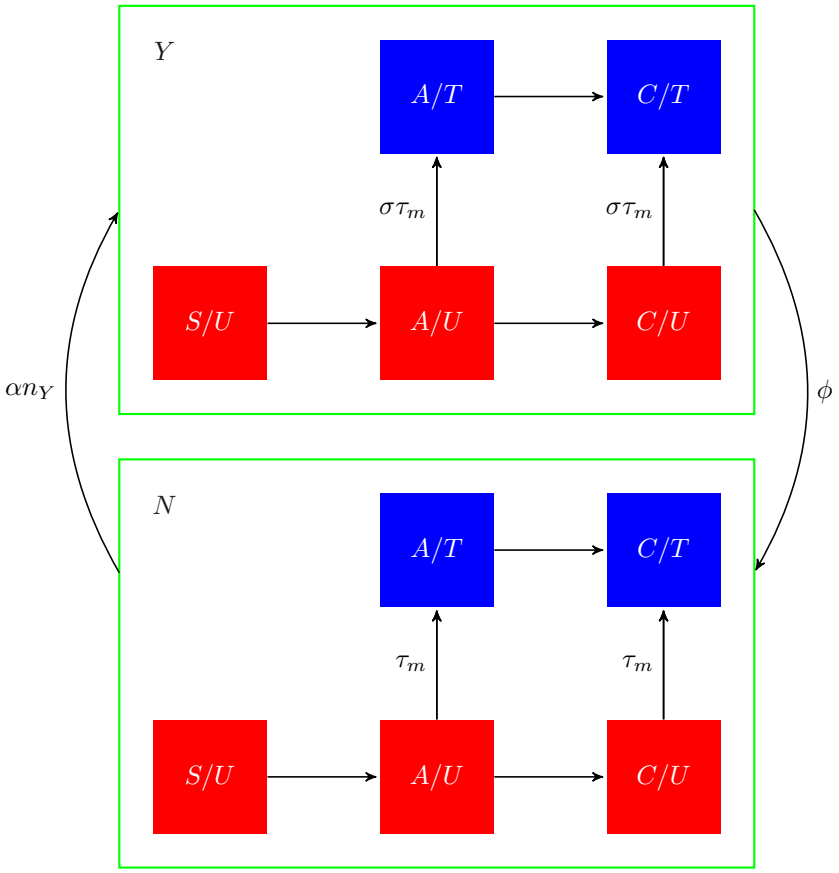


Figure 3: Motivated testing model. Transition probabilities same as those used in Fig. 1 left out for clarity, as are death transitions. Disease attribute values: S, A, C; Testing attribute values: U, T; Motivation attribute values: N,Y.

5.1 Open a new file called `motivated_process.ini`

Do it in yoru editor, and *do not* overwrite or reuse the uniform testing process. That way lies madness, or at least mild frustration.

5.2 Write the motivated testing process

Below is the full motivated testing process in one go. Read it through compare to Figure 3. After this we'll go through some of the rules. You can type it in as you go, or wait until the end of the section if you prefer.

```
1  [NodeAttributes]
2  disease = S, A, C
3  treatment = U, T
4  motivation = N, Y

6  [NodeRules]
7  {disease : S} -> {disease : A} =  lambda_A * NN({disease : A,
       treatment : U}) + lambda_C * NN({disease : C, treatment :
       U})

9  {disease : (A,C), treatment : U, motivation : N} -> {treatment
       : T} = tau_m
10 {disease : (A,C), treatment : U, motivation : Y} -> {treatment
       : T} = tau_m * sigma

12 {motivation : N} -> {motivation : Y} = alpha * NN({motivation
       : Y})
13 {motivation :Y} -> {motivation : N} = phi

15 {disease : A} -> {disease : C} = beta

17 {treatment : T} -> {disease : S, treatment : U, motivation : N
       } = delta_T
18 {disease : C, treatment : U} -> {disease : S, treatment : U,
       motivation : N} = delta_C
19 {disease : (S,A), treatment : U} -> {disease : S, treatment :
       U, motivation : N} = delta_A

21 [MeanFieldStates]
22 {}
```

Now then, looking at the process description you will probably recognize most of it: The `[NodeAttributes]` and `[MeanFieldStates]` sections are identical to the ones used in the uniform testing process, and so is the infection rule (line 7) and the disease progression (line 15).

You will probably also recognize the rules for 'death' (lines 17–19) where a node is recycled by having its state set to the 'ground state'. The difference here is that the *motivation* attribute must be part of the ground state as well (N) as the attribute values has an impact on process behaviour this time. It reflects that motivation is lost when a node 'dies', and so a new node must start unmotivated.

Note also that motivation does not influence the probability for a node to die.[4]

As for new rules, note that the testing has changed. We now have two testing rules (lines 9-10). On the selection side of the rule (left of $->$) we now select also based on the *motivation* attribute, choosing the first probability (tau_m) if the node is unmotivated, and the second (tau_m $*$ sigma) if it is motivated.

There are also the motivation spread rules on lines 12-13. The first one selecting unmotivated nodes (*motivation* attribute set to N) and making them motivated (Y). This is a linear function alpha $*$ NN({motivation : Y}) where `NN` as before gives the number of neighbouring nodes with the submitted attribute combinations. Thus the probability of a node becoming motivated increases with the value of the parameter `alpha` in unit time for each currently motivated neighbour (nagging).

The second motivation rule (line 13) changes back to unmotivated, and is a constant probability rate in unit time. It simply means that nodes stop being motivated after some time (but may become motivated again due to their neighbours of course).

That is it for the motivation process. If you haven't done so already. Type it in. Save it as `motivated_process.ini`.

5.3 Run the motivation simulation

Now it is time to run the motivated simulation. If you like, you can open up the configuration file (`motivated_simulation.ini`) for a look. It will look very much alike to the one for the uniform simulation, but the `[ProcessParameters]` section will look different, also defining the new parameters `alpha`, `phi`, `tau_m`, and `sigma`. Note also that the name of the process file has changed to `motivated_process.ini`.

```
1 [ProcessParameters]
2 file = motivated_process.ini
```

[4] In light of this it also makes sense why the last of the death rules (line 19) includes susceptibles (S). If you remember we mentioned this in the context of the uniform testing process. There it would not have mattered, however here it does: If susceptibles are allowed to live on because we assume that they were already in the ground disease and testing stages, it would have meant that some nodes could be spreading the social motivation longer than a life time. This would have been quite unlikely, given the parameter values used. Yet it would have given the motivation spread a small advantage in the simulation.

```
beta = 6
lambda_A = 0.64
lambda_C = 0.016
tau_m = 0.02
sigma = 25.0
alpha = 0.1
phi = 0.6
delta_A = 0.018
delta_C = 0.1
delta_T = delta_A
```

Another change, not shown in the above listing is that the `base_name` setting in the `[Output]` section now is `motivated_testing`. This will cause the files generated during this simulation to be begin with this name.

In any case, there is nothing that needs to be changed, and you can go ahead to run the simulation using the command

```
nepidemix_runsimulation motivated_simulation.ini
```

As in the uniform testing case the program `nepidemix_runsimulation` loads the configuration file, the network, and your newly written process and runs the simulation. The same things apply as with the last simulation you ran, so please review that section if you need to.

5.4 Run `extract_incidence.py` again

After the simulation is finished you will find a new set of files with the results from the simulation: `motivated_testing_nodeTrans.csv` and `motivated_testing_state_count.csv`.

Same as with the uniform testing simulation you now need to create incidence data by running the command

```
extract_incidence.py
```

in your terminal window.

After which you will also have the file `motivated_testing_incidence.csv`, as before it contains the new infections for each time step in the simulation.

6 Compare incidence from the uniform and motivated simulations

Now it is time to compare the incidence data from the uniform and the motivated testing strategies. To do so, import both data series in Excel and plot them

together (the same way you did in sections 4.3 – 4.5).

Looking at the moving average, are you confident to say anything about the different testing approaches? The data is of course stochastic, but perhaps you think you can see a trend there.

To put the numbers into context: remember that the y-axis will be the number of new cases in a 3.65 day period, and the x-axis is time. Using a moving average the smoothed curve is the number of daily new cases as averaged over some time period (a year if using a window size of 100).

You should also try some of the other trend lines built into Excel if you are curious.

Finally, in many studies incidence data is shown as a year by year total, and not a curve. If you enjoy playing around in Excel you may want to produce one of these diagrams[5].

7 Averaging data

Now, to prove this trend, we should do repeated simulations (or you could convince yourself by running around and checking the results from everyone else in the lab).

However, we will try to be a little bit more scientifically minded and actually create a data average. You won't have time to run multiple simulations during your tutorial, and so we will instead collect simulation data from the whole group to create an average.

To do that however we will need to collect it from each and every one of you. So, at this point call on one of the instructors to come over and copy your data files.

Afterwards at the end of the day we will show and discuss the averaged data.

8 Now what?

You are finished. Well done! Go have a coffee or tea.

If you still want to play around with the model, changing parameters or even the process, go ahead. You can also work through a more basic NepidemiX tutorial, showing some other uses (including a less powerful, but simpler way of writing state transition rules, and how to write processes in python). You can find it here: `http://nepidemix.irmacs.sfu.ca/tutorial.html` . That

[5]However, for the type of data that we have here ('true' incidence in regular fine-grained intervals) it is of course better to compute a trend using a moving average.

tutorial also covers what you need to set up NepidemiX on your own computer if you are interested in learning more.

For any comments or questions you are welcome to contact Lukas Ahrenberg at `ahrenberg@irmacs.sfu.ca`.

See you at the discussion in a little while!