A CALIBRATION CHALLENGE

Rating: Intermediate
Prerequisites: "Building a Minimalist Network-Based Model Framework"
Estimated Time Required: 1 hour

1) Save "SIR Agent Based Calibration" to "Extended SIR Agent Based Calibration".
2) Try calibrating the existing model for the entire duration of the calibration 5 separate times. Comment on the precision of the calibration in matching the "true values" as documented in the "Calibration" experiment (ContactRate=1.5, InfectionProbability=0.4), and hypothesize a reason behind the observed calibration results. (Please note that you might find experimentation with the experiment created in Step 5 helpful in exploring hypotheses, so you may want to delay completion of this problem until after step 5).
3) Open Main class, and do the following
   a) Add a variable "nCumulativeInfections" to accumulate total infections. This will be sampled periodically.
   b) Add a variables "nCumulativeInfectionsPrevious". The purpose of this variable is to remember the previous cumulative count of infections (i.e. the count of cumulative infections observed one sample ago), but for the moment this is simply a variable – the mechanism will be added in the next several steps
   c) Add a dataset "IncidentCasesDS" to hold the values of each successive incident cases (i.e. this will be the number of *new* cases observed in successive intervals of 2 days in length). This should be set such that
      1. It does *not* update automatically
      2. It has sufficient capacity to hold the samples (as required by considering the sampling schedule below)
   d) Add an event called "eventSampleNewInfections". The purpose of this event is to sample (using the variables created above) and record the values of the incident cases of diabetes that occur over each successive 2 day interval. The properties of the event should be set such that
      1. The event fires roughly every other day (i.e. every 2 days). For the purposes of this exercise, we will consider the unit of time to be a month, so for simplicity, treat the time between events as being every 0.06 months (approximately 2 days). The first occurrence time for the event should be time 0.06 (*i.e.* the event should fire at time 0.06, 0.12, 0.18, etc.).
      2. This event should use "nCumulativeInfections" and "nCumulativeInfectionsPrevious" to compute the number of incident cases (*i.e.* the number of *new* infections) that have taken place over the past 2 day period (i.e. over the period since the last sample). The resulting values for the number of "incident" infections observed over the most recent 2 day period should be inserted into "IncidentCasesDS" using the method call *add*(double).
      3. "nCumulativeInfectionsPrevious" should also be updated accordingly. The logic for this should recognize that the value of "nCumulativeInfections" for a given sample point is to be used as the value of "nCumulativeInfectionsPrevious" in the previous timestep.
4) In the above, you have been working on the Main class. You should now add logic to the *Person* class to do bookkeeping for "nCumulativeInfections" (i.e. to actually

operationalize the fact that this counts the number of cumulative infections experienced.)

5) Add an experiment (a traditional simulation) "TestSimulation". Set the simulation to run to 60.0 months (i.e. 5 years – the same stop time as will be used in the Calibration).
   a) Verify that the components of the model added above work as intended.
   b) Export the datasetIncidentCases manually from the interface and paste it into Excel.
   c) In Excel, plot the exported count incident cases (added to Excel in the last step).
6) Now switch your attention to the "Calibration" experiment, which sets out the calibration framework involved. Create a copy of this experiment by performing a "Copy" on it, and then clicking on the project and selecting "Paste".
7) For new calibration experiment
   a) In the "General" tab of the "Properties" for the experiment
      a. Name the new experiment "CalibrationMystery".
      b. Indicate "Main" as the root object (note that when an experiment is copied, the original value for this is *not* automatically retained. As noted in lecture, we can set this field so as to allow for alternative model versions – represented as different Main classes – for different model experiments).
      c. Set the model duration to 60.0 months
      d. Set "AverageIllnessDuration" to vary from 3 to 30 months
      e. Set "ContactRate" to vary from 0.5 to 5 per month
      f. Set "InfectionProbability" to be fixed with a value of 0.8.
   b) There will need to be some adjustment to the experiment reflect the fact that we are going to be adjusting the average illness duration and the contact rate (rather than the contact rate and infection probability). This adjustment will alert you to ways that you can find out information during a simulation on the current values of parameters being "tuned" in the calibration.
      a. For text18, set the text (in the "General" tab of the properties window) to be "IllnessDuration".
      b. For both text19 text20 (which are associated with the reporting the current & best values, respectively, for the second parameter – which was infection probability but is being changed to be average illness duration), set the code to use the value of the OptQuest variable associated with the average illness duration "_oqvar_AverageIllnessDuration" rather than the OptQuest variable for the infection probability. (Please note that this code is located in the "Dynamic" tab of these Text boxes)
      c. For button1 (which saves away the current best values to the clipboard), the code must be modified to use the appropriate variables (and to substitute other values for variables that are not changing). Please use the following code, which obtains the value for the two parameters being modified – *AverageIllnessDuration* and *ContactRate*, and summarizes the values for the other parameters*)*:

```
String s = "";
s += "AverageIllnessDuration\t" + format(getBestParamValue(_oqvar_AverageIllnessDuration)) + "\n";
s += "ContactRate\t" + format(getBestParamValue(_oqvar_ContactRate)) + "\n";
s += "InfectionProbability\t" + format(0.8) + "\n";
```

Exercise Rating: Intermediate

```
s += "TotalPopulation\t" + format(10000) + "\n";

copyToClipboard( s );
```

c) Modify the calibration (in the Advanced tab of the in the CalibrationMystery optimization experiment) to print out the current parameter values that has just explored after each "iteration" (in other words, after it has performed all of the realizations for a certain specified pair of for each of *AverageIllnessDuration* and *ContactRate)*. To undertake this task, please draw on the following information:
   a. The OptQuest variable names shown above (*_oqvar_AverageIllnessDuration* for the Average illness duration and *_oqvar_ContactRate* for the contact rate)
   b. `getCurrentParamValue(OptQuestVariableName)` can be used to get the current value of an OptQuestVariable
   c. `System.out.println(String)` can be called to print a value out to the console
   d. This should be done for each iteration, that is
      i. This should be printed not for each realization alone, but instead after each "iteration" as a whole.
      ii. This should be done regardless as to whether this iteration has yielded the best value thus far. In other words, we want to print all value explored – not just those that have yielded the most competitive matches yet observed.
   e. AnyLogic uses the term "simulation" to denote a specific realization (i.e. a specific run of the model, with a unique random number seed).
   f. Strings can be concatenated in Java using the "+" operator
   g. The string "\t" denotes the tab character in Java
   h. Distinct rows of numeric values separated by tabs in the console can be easily copied and pasted into Excel
   i. Copy and paste the contents of the console into Excel, and plot those points using a Scatterplot with lines and markers.
   j. Comment on the characteristics of the plot and/or the output used to create the plot.

d) Add *to the CalibrationMystery experiment* a dataset "dsIncidentCasesCurrent" to hold the incident case counts resulting from a given realization (i.e. simulation). This should be set so as to not update automatically. It is important to note that while incident case counts are already stored in Main within the dataset IncidentCasesDS, we *do* need a separate dataset to hold this information in the *CalibrationMystery* experiment. This is because the contents of "Main" (including IncidentCasesDS) disappear immediately following a given a realization, but the datasets within the CalibrationMystery experiment can live as long as the experiment as a whole. We will need to have this data in a "safe" place (one that does not disappear when the realization is over) so that we can compare it to the historic data – and potentially copy the data points to a further "best results" dataset.

e) In a manner similar to what is already in place for "dsInfectiousCurrent", update the contents of "dsIncidentCasesCurrent" from the corresponding dataset in the

Main object after each simulation run (so that this data is preserved after the Main object disappears).

f) Within *CalibrationMystery* experiment is a table function called "InfectiousHistoric" that will hold the historic *prevalence* data (i.e. data regarding the extant number of infectious cases in successive 2 day intervals). This data is of unknown provenance, and you will be trying to find the best fit of the model to this data, adjusting the two parameter values *AverageIllnessDuration* and *ContactRate).*
This table function currently holds data. You will retain the table function but change its contents. Specifically, you will instead populate this table function with given data using a two-step process.
    a. Clear out the current contents of the table function. You can accomplish this by using the  button to delete each row in turn, until the table function data is empty.
    b. Paste in the data from the file "Mystery Infectious Prevalence.csv". Note that this is th

g) Create within *CalibrationMystery* experiment a table function called "IncidentCasesHistoric". This will hold the historic *incidence* data (i.e. data regarding the number of *new* infectious cases observed over the course of successive 2 day intervals).
    a. This should have the same basic settings (e.g. regarding interpolation) as "InfectiousHistoric".
    b. For this table function, please paste in the data from the file "Mystery Incident Cases.csv". Before you paste in from Excel, be aware that AnyLogic may treat "commas" in the spreadsheet in European fashion – as decimal points. To work around this, select the cells in Excel, and set the format not to show commas (for example, you can use "Increase Decimal" to eliminate the commas).

8) Add a dataset "dsIncidentCasesHistoric". This should be set so as to not update automatically.
    a. Give the dataset sufficient capacity to hold all of the data required from the historic datapoints
    b. Initialize the dataset at the same point as is used for the dataset dsInfectiousHistoric.

9) Add an additional term into the objective function (with equal weight to the current term) that takes the difference (as computed by the built-in difference function) between the incident cases as computed by a given realization of the model and the historic incident case count values.
    a. Each term of the objective function will involve a comparison between simulated output values on the one hand and the corresponding historic data on the other
    b. There are two difference terms
    c. Add the two "difference" terms to one another
    d. It bears noting that in a real model,

   i. we would probably weight these values differently (reflecting the importance to our model purpose, and the statistical reliability of the data).
   ii. We would probably normalize the values more carefully so that both are the same dimension. For example, we might make each value *dimensionless* by "normalizing the discrepancy" (dividing the discrepancy by) some reference level or by the historic data, model data or average thereof. (The way we have currently computed this is in fact dimensionally correct, because both are in fact measured in terms of people – even the incidence values are measured in terms of people rather than people per unit time. But we would normally recognize this issue more clearly.)

10) Calibrate the resulting model, running the calibration until it completes.
    a. Record the calibrated values of the parameters.
    b. To lessen the amount of work involved, only a single figure is currently shown. Test the accuracy of the results against the given data by using the "TestSimulation" experiment to run one realization for the calibration estimated parameter values, and comparing it graphically against the data given within Excel.
11) Given the appearance of the two data sets found in the previous step, comment on how the reliability of calibration might be improved?
12) In Main, add a transition from Recovered back to Susceptible, with a Rate that is very small (0.02).
13) Recalibrate the (modified) model using the "CalibrationMystery" experiment, running it until it completes. Please hand in comments on
    a. The consistency of the results obtained with earlier results
    b. The goodness of fit of the calibrated value to the sources of data
    c. Implications for calibration when building models of the external world.

Exercise Rating: Intermediate