AGENARISK 10

Developer User Manual

www.AgenaRisk.com

Copyright Notice

1 GETTING STARTED 6

1.1 The files and information you need before installing the API 6

1.2 Activating AgenaRisk 10 Developer 6

1.3 The basic prerequisites for every API program you create 7

1.4 Creating and running your first simple API program 8

1.5 Checking the API Version number 9

1.6 What you need to know before doing any serious API programming 9

2 CREATING, SAVING AND LOADING A SIMPLE MODEL 11

2.1 Preliminaries 11

2.2 Creating a model with some nodes 11

2.3 Saving a model 12

2.4 Calculating a model 12

2.5 Loading a model 12

2.6 Adding the necessary exception handling 12

3 ADDING AND CHANGING NODE STATES 13

4 WORKING WITH NPTS 15

5 GETTING NODE MARGINAL PROBABILITY VALUES AND STATISTICS 17

6 ENTERING EVIDENCE AND WORKING WITH RESULTS 19

7 WORKING WITH EXPRESSIONS AND SIMULATION NODES 21

7.1 Defining expressions on nodes without parents 21

7.2 Defining expressions on nodes with parents 24

7.3 Defining partitioned expressions 26

8 WORKING WITH MULTIPLE RISK OBJECTS 31

9 CREATING A WEB APPLICATION 33

10 LEARNING PROBABILITY TABLES FROM DATA 39

10.1 Getting node probability tables 39

10.2 Generating example data file as a template...39

10.3 Learning from data alone...41

10.4 Learning from data with incorporation of expert judgement 43

10.5 Learning from data with nodes' individual settings for incorporation of expert judgement 44

11 APPENDIX: SUMMARY OF API CLASSES AND METHODS 48

11.1 Model Class...48

11.2 ExtendedBN Class..48

11.3 Node Class...49

11.4 ContinuousEN Class 49

11.5 State Class...50

11.6 Scenario Class...50

11.7 DataPoint Class 50

11.8 AbsoluteDataPoint Class 50

11.9 IntervalDataPoint Class 50

11.10 DataSet Class...50

11.11 MarginalDataStore Class 51

11.12 MarginalDataItemList Class...51

11.13 MarginalDataItem Class 51

11.14 Function Classes...51

11.15 MetaData Class 52

11.16 MetaDataItem Class 52

11.17 SensitivityAnalyser Class 52

11.18 ClassQueryModel Class...53

11.19 EMCal Class..53

11.20 Data Class 53

11.21 SampleDataGenerator Class 53

1 Getting Started

1.1 *The files and information you need before installing the API*

1. Before attempting to use the AgenaRisk 10 Developer, aka Applications Programming Interface (API) it is assumed that you have some familiarity with AgenaRisk 10 Desktop, which is a visual design environment for creating Bayesian Networks.
2. It is assumed that you have Java JDK 1.8 installed on your machine (AgenaRisk Desktop is distributed with a Java JRE 1.8 which is insufficient for Java application development).
3. You should have received an email message with an AgenaRisk 10 Developer license activation key, which is a string like:

   AB1CD-2EFG3-H4567-89012-I3J45
4. You should have downloaded a zip file like:

   AgenaRiskSDK_5181.zip

   The email message with the activation key provides the link for this download.

   Check https://resources.agenarisk.com/download/sdk/ for the latest revision.
5. If any of the above are missing contact support@agenarisk.com

1.2 *Activating AgenaRisk 10 Developer*

Make sure that the working directory of your project contains the following files/folders (included with the example SDK project):

- lexactivator\
- lexfloatclient\
- AgenaRisk.dat
- aid
- guid

When deploying to a headless server, you will need to download dependencies for the enterprise from https://resources.agenarisk.com/download/sdk/enterprise.zip Then extract files AgenaRisk.dat, aid, guid from the archive to replace the files in the working directory above.

You will need to use the License class provided:

```
import uk.co.agena.minerva.util.License;
public class DeveloperKeyActivate {
        public static void main(String[] args) {
                License.keyActivate("AB1CD-2EFG3-H4567-89012-I3J45");
        }
}
```

To deactivate the license:

```
License.keyRelease();
```

To check out a floating license:

`License.floatingLicenseLease("localhost", "0000");`

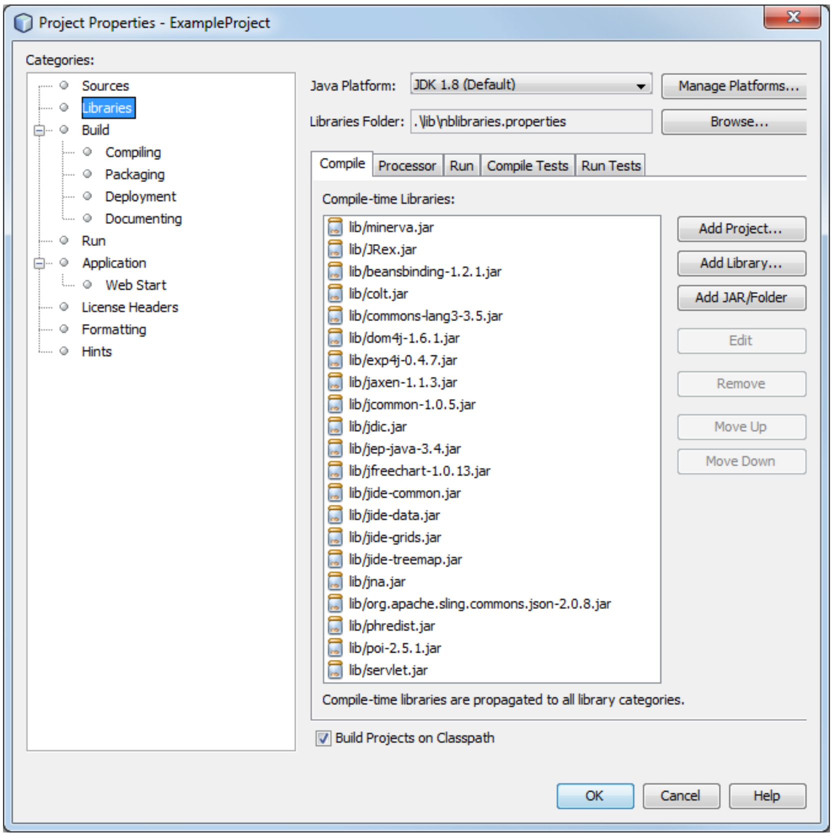
Where “localhost” and “0000” are the address and port respectively of your licensing server provided by Agena staff.

To release the floating license:

`License.floatingLicenseRelease();`

1.3 The basic prerequisites for every API program you create

1. ***Create a new Project:*** Create a new Java project in your favourite Java Editor or IDE, such as Netbeans, with an associated folder, which we will assume is called testAPI.
2. ***Add the jar files as libraries:*** In the Project properties you will need to add to the libraries all of the AgenaRiskAPI jar files that were unzipped so that the libraries looks as shown in Figure 1 in NetBeans[1]:



Java Platform: JDK 1.8 (Default)

Libraries Folder: .\lib\nblibraries.properties

lib/minerva.jar
lib/JRex.jar
lib/beansbinding-1.2.1.jar
lib/colt.jar
lib/commons-lang3-3.5.jar
lib/dom4j-1.6.1.jar
lib/exp4j-0.4.7.jar
lib/jaxen-1.1.3.jar
lib/jcommon-1.0.5.jar
lib/jdic.jar
lib/jep-java-3.4.jar
lib/jfreechart-1.0.13.jar
lib/jide-common.jar
lib/jide-data.jar
lib/jide-grids.jar
lib/jide-treemap.jar
lib/jna.jar
lib/org.apache.sling.commons.json-2.0.8.jar
lib/phredist.jar
lib/poi-2.5.1.jar
lib/servlet.jar

Compile-time libraries are propagated to all library categories.

☑ Build Projects on Classpath

Figure 1 Library setup

3. ***Import relevant packages:*** There are dozens of packages and hundreds of class files in the API. The complete set of information about all these classes and packages is contained in the javadoc supplied with the API. Fortunately, for most API applications you will only need a small subset of the packages and classes available. In particular, you should include the following import statements in your class file:

---

[1] In the screenshot here the AgenaAPI files were unzipped to the location E:\MyProgs rather than C:

```
import uk.co.agena.minerva.model.*;
import uk.co.agena.minerva.model.extendedbn.*;
import uk.co.agena.minerva.model.scenario.Scenario;
import uk.co.agena.minerva.util.model.DataPoint;
import uk.co.agena.minerva.util.model.DataSet;
```

Since you will also normally be manipulating lists of objects you should also import the following java.util package classes:

```
import java.util.ArrayList;
import java.util.List;
```

1.4 Creating and running your first simple API program

1. Finally, you will want to test that the setup has been successful.
2. Assuming you have followed the steps in Section 1.3, create a class called FirstExample and copy into it the following code[2] (that is in the file C:AgenaAPI/FirstExample.java):

```
public class FirstExample {

   public static void main(String[] args) {

             try {
                Model myModel = Model.createEmptyModel();
                // First get the single BN
                ExtendedBN ebn = myModel.getExtendedBNAtIndex(0);
                // Add a new node of type Boolean to this BN;
                // its identifier is “b” and its name is “B”
                System.out.println("AgenaAPI Test started");
                BooleanEN booleanNode = ebn.addBooleanNode("b", "B");
                myModel.calculate();
                myModel.save("test.cmp");
                System.out.println("AgenaAPI Test finished");

            }
            catch (Exception e){};
  }

}
```

3. Compile and run the project.
4. If successful you will see something like the following output:

```
run:
GUI not initiated
Headless mode: false
AgenaRisk 10 Revision 5660
AgenaAPI Test started
AgenaAPI Test finished
BUILD SUCCESSFUL (total time: 1 second)
```

---

[2] The main part of the API user manual explains what this code means, but for the time being you should simply be concerned about getting it to compile and run.

5. Next check the project folder. It should contain a file called *test.cmp* which is an AgenaRisk model file that your project just created.
6. Open the file *test.cmp* with AgenaRisk desktop.
7. You should find a model with a node named "B".
8. This confirms that you have successfully created your first AgenaRsk API program. Congratulations.

1.5 Checking the API Version number

When discussing API issues with Agena support you may be asked to supply the API version number. There are two ways to find this:

When you run any program written with the API using an IDE like NetBeans the first several lines in the output window should include something like:

```
AgenaRisk 10 Revision 5660
```

This line is the API version number.

There is a method *getApplicationVersionAndRevisionNumber()* in the Model class that enables you to output the version number. So include the following lines of code to output the version number

```
Model myModel = Model.createEmptyModel();
String versionNumber = myModel.getApplicationVersionAndRevisionNumber();
System.out.println(versionNumber);
```

This will produce an output like:

```
AgenaRisk 10 Revision 5660
```

1.6 What you need to know before doing any serious API programming

Before attempting any API programming with AgenaRisk it is imperative that you are familiar with AgenaRisk and its terminology and are a competent Java programmer. In particular, here are the things you need to be aware of in what follows:

- A model consists of one or more ‘risk objects’. Risk objects are what are also commonly referred to as Bayesian networks (BNs). Hence, a model contains one or more BNs.
- In the API a risk object is formally known as an ExtendedBN (so ‘risk object’ and ‘BN’ are just informal names for an ExtendedBN).
- The files that you open and save in AgenaRisk (.cmp and .ast files) are therefore models ***not*** BNs (although it is common to create and save a model that consists of just a single BN).
- A node in a BN is referred to in the API as an ExtendedNode. The ExtendedNode class in the API has subclasses (BooleanEN, ContinuousEN,

RankedEN etc.) corresponding to each of the types of node you can define in AgenaRisk.

- Entering evidence (observations) on nodes is all done by the use of *scenarios*. A scenario is a set of observations on nodes in the model. By default a model has a single scenario.
- Whereas AgenaRisk desktop allows for a range of settings related to a model's visual appearance (such as node colour, shape, size, location) the API does not currently support such settings. Hence after saving a model in the API using the save() method (see Section 2) any such settings previously defined for the model in the desktop version will be lost. Later, when such a saved model is reopened in AgenaRisk it will be displayed with the default visual appearance settings.

2 Creating, saving and loading a simple model

2.1 Preliminaries

In this, and all following sections, it is assumed that you have created a Java project called testAPI (e.g. in NetBeans) as described in Section 1.3, and a Java class. All of the examples we present are methods in a class called *ExamplesFromManual* that you will find in the folder *api example models*. To run any of these methods you will need to add a *main* method[3], so that the starting code you need looks like this:

```
package testAPI;

import java.util.ArrayList;
import java.util.List;
import uk.co.agena.minerva.model.Model;
import uk.co.agena.minerva.model.extendedbn.*;

public class ExamplesFromManual{

  public static void main(String args[]) {

       ExamplesFromManual ex = new ExamplesFromManual();
       ex.method1();
       ex.method2();
       ex.method3();
           etc..
  }
}
```

The method in this section is called *createSaveLoadSimpleModel()*

2.2 Creating a model with some nodes

To create a new model add:

```
Model myModel = Model.createEmptyModel();
```

This creates a model with a single empty ExtendedBN.

To add a (Boolean) node to this ExtendedBN use the following code:

```
// First get the single BN
ExtendedBN ebn = myModel.getExtendedBNAtIndex(0);

// Add a new node of type Boolean to this BN;
// its identifier is “b” and its name is “B”
BooleanEN booleanNode = ebn.addBooleanNode("b", "B");
```

The default states for a Boolean node, **True** and **False** are automatically added.

Now add another node (a Labelled node this time):

```
LabelledEN labelledNode = ebn.addLabelledNode("l", "L");
```

---

[3] Such a main method is provided in the file *ExamplesFromManual.java*

A set of default states for a Labelled node (**State_1**, **State_2** and **State_3**) is automatically added.

Next we make booleanNode a child of labelledNode:

```
labelledNode.addChild(booleanNode);
```

The following code would have produced an identical result:

```
booleanNode.addParent(labelledNode);
```

2.3 Saving a model

Next save the model:

```
//saves the model to a file called “test.cmp”
//in the current directory
myModel.save("test.cmp");
```

2.4 Calculating a model

Calculating a model is the equivalent of pressing the ‘Run’ button in AgenaRisk desktop. Once a model has been created to ‘calculate’ the model:

```
myModel.calculate();
```

2.5 Loading a model

To load an existing AgenaRisk model, or to reload one previously saved in the API:

```
Model m = Model.load("test.cmp");
```

2.6 Adding the necessary exception handling

The final piece of code you need to add is a try-catch exception (If you attempt to compile and run the code above you will get an unreported exception message because the Model class needs to check that appropriate files are available). To ensure appropriate error handling you simply need to enclose the above code inside a try-catch exception:

```
try {
        Model myModel = Model.createEmptyModel();
        // First get the single BN
        ExtendedBN ebn = myModel.getExtendedBNAtIndex(0);
        etc......
}
catch (Exception e){};
```

After you run the code you should check that there is indeed a file called *test.cmp* and that when you open it in AgenaRisk it has two nodes labeled *B* and *L*.

As mentioned at the beginning of the section, the full code for the method *createSaveLoadSimpleModel()* can be found in the file *ExamplesFromManual.java*

3 Adding and changing node states

This section explains how to edit the set of states for nodes of any type. The method in this section is called *editStates()*

In this example you will need to add the following import statements:

```
import uk.co.agena.minerva.util.model.DataPoint;
import uk.co.agena.minerva.util.model.DataSet;
```

Create a model *m* and get the single extendedBN *ebn* as explained in Section 2

Now add new nodes of each of the different types:

```
LabelledEN len = ebn.addLabelledNode("l", "L");
BooleanEN ben = ebn.addBooleanNode("b", "B");
ContinuousIntervalEN cien = ebn.addContinuousIntervalNode("ci", "CI");
IntegerIntervalEN iien = ebn.addIntegerIntervalNode("ii", "II");
DiscreteRealEN dren = ebn.addDiscreteRealNode("dr", "DR");
RankedEN ren = ebn.addRankedNode("r", "R");
```

Each different node type comes with its own set of default states, which is a list. If you just want to remove a state you can simply use the standard java List remove method. For example:

```
List states = len.getExtendedStates();
states.remove(0);
```

This will remove the first state (named “State_1”), i.e. the state at index 0 in the List. However, if you want to add or change states it is much better to use the AgenaRisk DataSet Class to define the revised set of states you want.

Thus, to redefine the set of for the labeled node *len* we first create a DataSet object:

```
DataSet lds = new DataSet();
```

Next, add the set of state names to lds:

```
lds.addLabelledDataPoint("Red");
lds.addLabelledDataPoint("Amber");
lds.addLabelledDataPoint("Green");
```

Now completely redefine the set of states of len:

```
len.createExtendedStates(lds);
```

Similarly, for the Boolean node:

```
DataSet bds = new DataSet();
bds.addLabelledDataPoint("Yes");
bds.addLabelledDataPoint("No");
ben.createExtendedStates(bds);
```

For the Continuous Interval Node:

```
DataSet cids = new DataSet();
cids.addIntervalDataPoint(0.0, 100.0);
cids.addIntervalDataPoint(100.0, 200.0);
cids.addIntervalDataPoint(200.0, 300.0);
cien.createExtendedStates(cids);
```

For the Integer Interval Node:

```
DataSet iids = new DataSet();
iids.addIntervalDataPoint(10, 20);
iids.addIntervalDataPoint(20, 30);
iids.addIntervalDataPoint(30, 40);
iien.createExtendedStates(iids);
```

For the Discrete Real Node:

```
DataSet drds = new DataSet();
drds.addAbsoluteDataPoint(1.0);
drds.addAbsoluteDataPoint(2.0);
drds.addAbsoluteDataPoint(3.0);
dren.createExtendedStates(drds);
```

For the ranked node:

```
DataSet rds = new DataSet();
rds.addLabelledDataPoint("Bad");
rds.addLabelledDataPoint("OK");
rds.addLabelledDataPoint("Good");
ren.createExtendedStates(rds);
```

Save the model (we will use the saved model for the next example):

```
m.save("test.cmp");
```

Finally, remember to enclose the above code inside a try-catch exception as explained in Section 2.

After running the code you should inspect the *test.cmp* file in AgenaRisk and check that the node states are as you defined them in this method. It is also important to note that, currently, the API does not support synchronization with the Risk Table view. Hence, you will see that the risk table entries still contain the default states for the nodes. However, in the Risk Table view you can manually synchronise node states with risk table entries.

The full code listing for the above appears in the method *editStates().*

4 Working with NPTs

In this section we show how to redefine NPTs. The method in this section is called *nptManipulation().* The example uses the file *test.cmp* saved in the previous section.

First load the saved file and get the single BN from it:

```
// Load the model that was created and saved
// in the method editStates
Model m = Model.load("test.cmp");

// Get the single BN
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Next we get three of the nodes in the model, namely the nodes labelled *B*, *L*, and *R*. Note the use of casting as the ebn.getExtendedNodewithName() method returns the ExtendedNode superclass:

```
BooleanEN ben = (BooleanEN) ebn.getExtendedNodeWithName("B");
LabelledEN len = (LabelledEN) ebn.getExtendedNodeWithName("L");
RankedEN ren = (RankedEN) ebn.getExtendedNodeWithName("R");
```

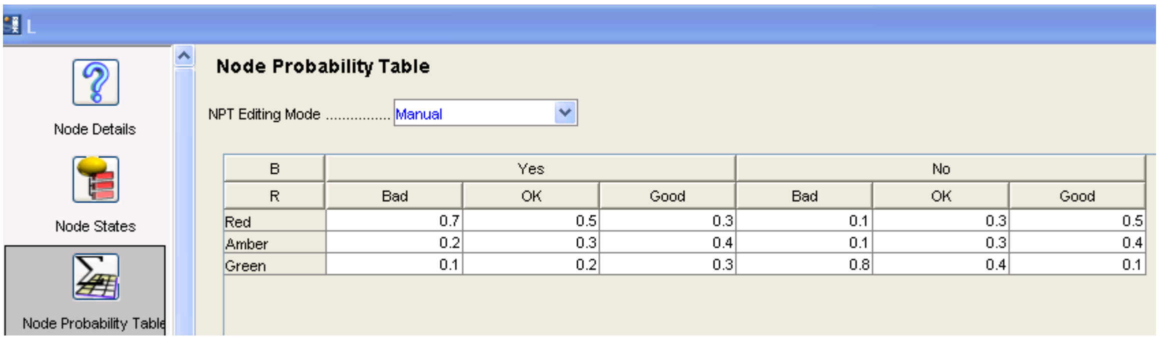
Make nodes *B* (ben) and *R* (ren) parents of *L* (len):

```
len.addParent(ben);
len.addParent(ren);
```

We now redefine the NPTs of nodes *B* and *R* to overwrite the default NPTs. These NPTs are easy because the nodes have no parents. *B* has two states so we need to define an array of length 2, while *R* has three states so we need to define an array of length 3:

```
ben.setNPT(new double[] { 0.9, 0.1 });
ren.setNPT(new double[] { 0.2, 0.3, 0.5 });
```

It is trickier to redefine the NPT for the node *L* because it has two parents (*B* and *R*). Before showing you the code you need note that Figure 2 shows in AgenaRisk the actual NPT that we are going to create. The important thing to note is that the NPT is an array of 6 columns where each column is an array of length 3. The order of the entries is based on the exact order of the states for each node.



Node Probability Table

NPT Editing Mode .... Manual

B	Yes			No		
R	Bad	OK	Good	Bad	OK	Good
Red	0.7	0.5	0.3	0.1	0.3	0.5
Amber	0.2	0.3	0.4	0.1	0.3	0.4
Green	0.1	0.2	0.3	0.8	0.4	0.1

Figure 2 The NPT for the node L (len)

To generate this NPT we must first get the list of parents of *L*:

```
List lenParents = ebn.getParentNodes(len);
```

Now we define the NPT of L:

```
len.setNPT(new double[][] {
          {0.7, 0.2, 0.1},
          {0.5, 0.3, 0.2},
          {0.3, 0.4, 0.3},
          {0.1, 0.1, 0.8},
          {0.3, 0.3, 0.4},
          {0.5, 0.4, 0.1} },
        lenParents);
```

Save the model:

```
m.save("test.cmp");
```

Finally, remember to enclose the above code inside a try-catch exception.

After running this code you can open the model test.cmp in AgenaRisk and check that the NPTs for the nodes *B*, *R* and *L* are as defined above.

The full code listing for the above appears in the *nptManipulation()* method.

5 Getting node marginal probability values and statistics

In this section we show how to extract marginal probability values from a node as well as statistics like the mean and variance (the latter are only really relevant for numerical node types). We will create a parameterized method *printMarginals* and illustrate it using a method called *workingWithEvidence()* (which uses the file *test.cmp* saved in the previous section).

First you will need to add the following import statement to your project

```
import uk.co.agena.minerva.model.MarginalDataItemList;
```

The general purpose method *printMarginals* has three parameters: Model, ExtendedBN and ExtendedNode, thus:

```
private void printMarginals(Model m, ExtendedBN ebn, ExtendedNode enode) {}
```

To get the Marginals for the node ebn we first need the following code:

```
MarginalDataItemList mdil =
        m.getMarginalDataStore().getMarginalDataItemListForNode(ebn, enode);
```

The resulting list mdil assumes there can be multiple scenarios. To get the marginals for the first scenario (for the saved file *test.cmp* there is only one default scenario) we get the first MarginalDataItem in the list:

```
MarginalDataItem mdi = mdil.getMarginalDataItemAtIndex(0);
System.out.println(enode.getName().getShortDescription());
List marginals = mdi.getDataset().getDataPoints();
for (int i = 0; i < marginals.size(); i++) {
  DataPoint marginal = (DataPoint) marginals.get(i);
  System.out.println(marginal.getLabel() + " = " + marginal.getValue());
}
```

To print the mean of the distribution

```
System.out.println("Mean = " + mdi.getMeanValue());
```

To print the variance of the distribution

```
System.out.println("Variance = " + mdi.getVarianceValue());
```

Now we can test the method out by loading the model *test.cmp* and running the method first with the labelled node L and then with the continuous node CI:

```
public void workingWithEvidence() {
        try {
          Model m = Model.load("test.cmp");
          ExtendedBN ebn = m.getExtendedBNAtIndex(0);

          //Calculate the entire model:
          m.calculate();
```

```
             //get the node L
            LabelledEN len = (LabelledEN) ebn.getExtendedNodeWithName("L");

            //print the marginals and statistics for L
            printMarginals(m, ebn, len);

            //get the node CI

            ContinuousIntervalEN cien =
                (ContinuousIntervalEN)ebn.getExtendedNodeWithName("CI");

            //print the marginals and statistics for CI
            printMarginals(m, ebn, cien);

            }
            catch (Exception e){};
    } //workingWithEvidence
```

The output you should get is:

```
L
Red = 0.4320000112056732
Amber = 0.328000009059906
Green = 0.23999997973442078
Mean = 0.0
Variance = 0.0
CI
0.0 - 100.0 = 0.3333333432674408
100.0 - 200.0 = 0.3333333432674408
200.0 - 300.0 = 0.3333333432674408
Mean = 150.00000447034836
Variance = 6666.666865348836
```

The full code listing for the above appears in the *printMarginals* method and the *workingWithEvidence* method.

6 Entering evidence and working with results

In this section we show how to enter evidence into a model. The method in this section is called *enteringEvidence().* The example uses the file *test.cmp* saved in the previous section.

First load the saved file, get the single BN from it, and calculate the model:

```
Model m = Model.load("test.cmp");
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
m.calculate();
```

Next we get the node *L* whose NPT we created in the nptManipulation() method:

```
LabelledEN len = (LabelledEN) ebn.getExtendedNodeWithName("L");
```

We are going to enter evidence on one of the parents of the node *L*. So, first we need to get one such parent. In this case, we are going to use the Ranked node parent *R*:

```
RankedEN ren = (RankedEN) ebn.getExtendedNodeWithName("R");
```

All evidence entry is done via “scenarios” - there is an enterEvidence() method associated with the ExtendedNode class but you should ignore this. So we need to get the single (default) scenario, which currently has no evidence:

```
Scenario s = m.getScenarioAtIndex(0);
```

Now we enter hard evidence on node *R* (ren):

```
s.addHardEvidenceObservation(ebn.getId(), ren.getId(),
    ren.getExtendedStateAtIndex(2).getId());
```

Note that you have to pass *three* parameters to this method. The first parameter is simply the numerical ID of the ExtendedBN. The second parameter is the ID of the node we wish to enter hard evidence on. The third parameter is the ID of the state of the node we are entering as an observation. In the example here we are making the third state (i.e. index 2 with the value **Good**) an observation.

Now calculate again and print the marginals:

```
m.calculate();
printMarginals(m, ebn, len);
```

When you run this you should see a revised set of marginals printed:

```
Red = 0.32000000265240663
Amber = 0.3999999949336054
Green = 0.28000000241398804
```

Entering evidence in numeric nodes (either Continuous Interval or Integer Interval nodes) uses a different method because you are not simply nominating a particular state to be true. Instead, you are entering a numeric value. To see how to do this, we are going to get the Continuous Interval node *CI*:

```
ContinuousIntervalEN cien =
(ContinuousIntervalEN)ebn.getExtendedNodeWithName("CI");
```

Now enter a “real number” observation, 48.0, using the same scenario s:

```
s.addRealObservation(ebn.getId(),cien.getId(),48.0 );
```

Note: For an Integer Interval node you would use the method *addIntegerObservation().*

Calculate the model again

```
m.calculate();
```

and print the marginals for the node *CI* (cien):

```
printMarginals(m, ebn, cien);
```

When you run this you should see the following output:

```
Mean = 50.0
Variance = 0.0
0.0 - 100.0 = 1.0
100.0 - 200.0 = 0.0
200.0 - 300.0 = 0.0
```

It is worth noting that, because this Continuous Interval node has not been set as a ‘simulation node’, it still has a discrete set of states and so the mean is not equal to the observation you entered (45). This is because entering 45 simply tells the model that the observation is ‘in the state range 0-100’. In the next section you will see how to create simulation nodes (or make numeric nodes simulation nodes) so that we do not get these sorts of approximation errors.

Save the model:

```
m.save("test.cmp");
```

Finally, remember to enclose the above code inside a try-catch exception.

The full code listing for the above appears in the method enteringEvidence().

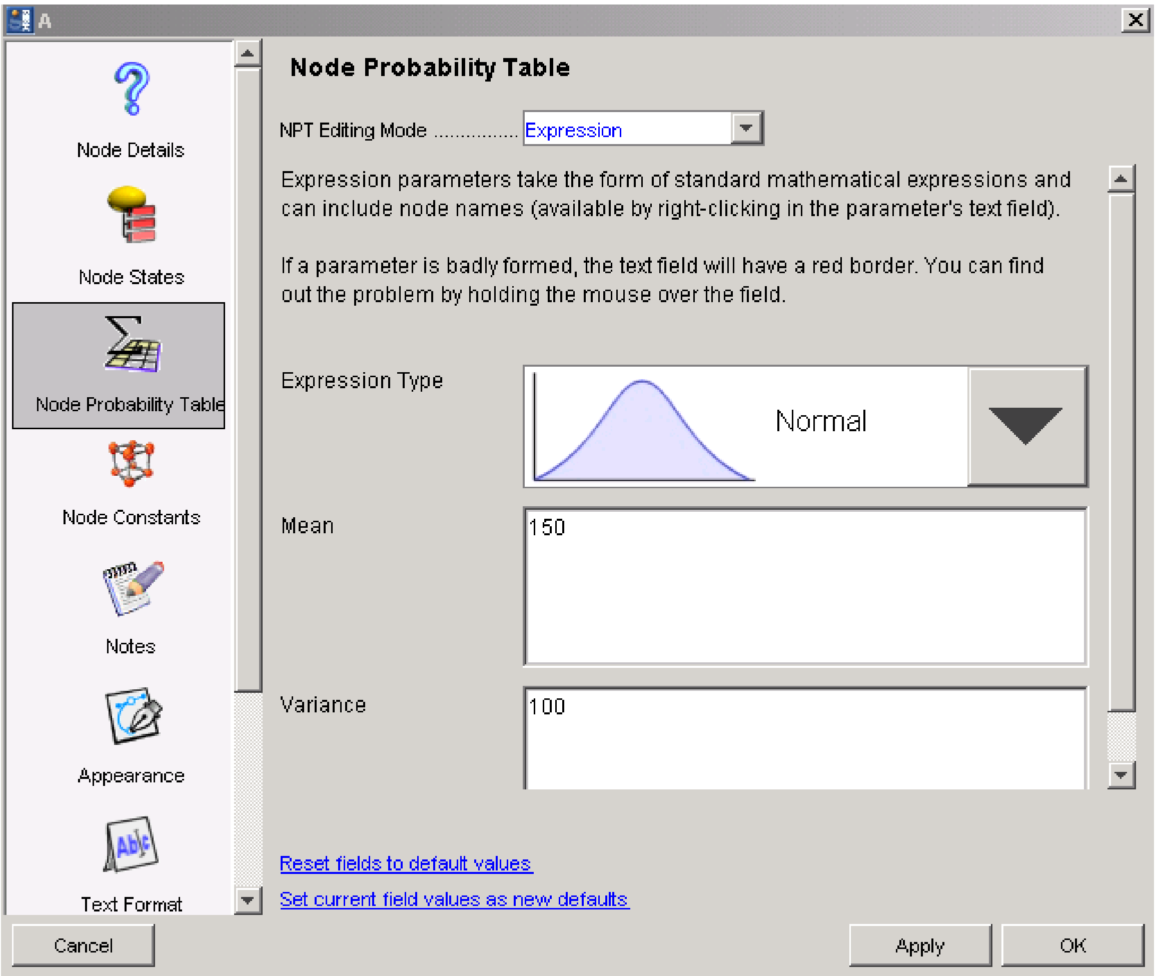
7 Working with expressions and simulation nodes

A particular strength of AgenaRisk is that you can define the NPT of a node as an expression rather than as a large complex table. There is a library of mathematical and statistical functions you can use (depending on what type the node is); the AgenaRisk manual lists the relevant functions. If the node has parent nodes then you can use these nodes as parameters of expressions.

In this section we show how to define expressions on: a) nodes without parents (Section 7.1); and b) nodes with parents (Section 7.2). We also show (Section 7.3) how to define a partitioned expression.

7.1 Defining expressions on nodes without parents

An example of an NPT defined as an expression is shown in Figure 3.



Node Probability Table

NPT Editing Mode .... Expression

Expression parameters take the form of standard mathematical expressions and can include node names (available by right-clicking in the parameter's text field).

If a parameter is badly formed, the text field will have a red border. You can find out the problem by holding the mouse over the field.

Expression Type: Normal

Mean: 150

Variance: 100

Reset fields to default values

Set current field values as new defaults

Figure 3 Defining a node NPT as an expression

Here the node is a Continuous Interval node and the expression is a Normal distribution with mean (150 in this case) and variance (100 in this case) parameters. We will next show how to create exactly this node and NPT.

The method in this section is called *testSimpleExpression().*

First you will need to add the following import statement to your project (you will need this for all the examples in this Section).

```
import uk.co.agena.minerva.util.nptgenerator.*;
```

Create a new model and get the single BN:

```
Model m = Model.createEmptyModel();
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Next we want to add a continuous node to this BN. As explained in detail in Chapter 6 of the AgenaRisk User Manual, continuous nodes can be set as simulation nodes (recommended in most situations). If you know that you want your continuous node to be a simulation node then you use the addSimulationNode method[4] as follows:

```
ContinuousIntervalEN cien = ebn.addSimulationNode("s", "S");
```

This creates a simulation node with the same default settings as in AgenaRisk desktop (i.e. the same states [–inf to -1], [-1 to 1], [1 to inf]) and with its NPT equal to a Normal Distribution with mean 0 and variance 1000000) but with id “s”, name “S”.

It is however, instructive to show how to create a (non-simulation) continuous node since this also explains how to define the states and NPT manually. To add a new Continuous node to the BN:

```
ContinuousIntervalEN a = ebn.addContinuousIntervalNode("a", "A");
```

Redefine the states of node *A* as just a single state range from 0 to infinity:

```
DataSet cids = new DataSet();
cids.addIntervalDataPoint(0.0, Double.POSITIVE_INFINITY);
a.createExtendedStates(cids);
```

If you want to manually define the node as a simulation node you simply use:

```
a.setSimulationNode(true);
```

However, because we have constructed this continuous node manually we still need to define an expression for its NPT. To define a Normal distribution as the expression for the NPT of this node:

```
List parameters = new ArrayList();
parameters.add("150"); // mean
parameters.add("100"); // variance
ExtendedNodeFunction enf = new
ExtendedNodeFunction(Normal.displayName, parameters);
a.setExpression(enf);
```

Now regenerate the NPT for this node:

```
ebn.regenerateNPT(a);
```

---

[4] This was first introduced to the API in April 2014

We can choose the level of accuracy we want when we have simulation nodes by setting the Simulation Convergence (the default in both the API and Desktop version if you do not set it explicitly is now 0.001):

```
m.setSimulationEntropyConvergenceTolerance(0.0001);
```

We can also adjust the accuracy by changing the number of iterations (the default is 25 if you do not set it explicitly) although this is less important that the simulation convergence:

```
m.setSimulationNoOfIterations(30);
```

Calculate the entire model:

```
m.calculate();
```

Use the printMarginals() method defined above to print out the marginals:

```
printMarginals(m,ebn,a);
```

Save the model so you can inspect it in AgenaRisk:

```
m.save("test2.cmp");
```

Finally, remember to enclose the above code inside a try-catch exception.

When you run the method you should get the following output:

```
A
55.0 - 100.0 = 2.8665158424701076E-7
100.0 - 114.0625 = 1.6268965555354953E-4
114.0625 - 121.09375 = 0.0017594066448509693
121.09375 - 124.60938 = 0.003635115223005414
124.60938 - 128.125 = 0.008795523084700108
128.125 - 131.64062 = 0.018830427899956703
131.64062 - 133.39844 = 0.015258061699569225
133.39844 - 135.15625 = 0.020413219928741455
135.15625 - 136.91406 = 0.0264812670648098
136.91406 - 138.67188 = 0.033310502767562866
138.67188 - 140.42969 = 0.04062924161553383
140.42969 - 142.1875 = 0.048051994293928146
142.1875 - 143.94531 = 0.05510605499148369
143.94531 - 145.70312 = 0.06127769872546196
145.70312 - 149.21875 = 0.13515281677246094
149.21875 - 152.73438 = 0.13887722790241241
152.73438 - 156.25 = 0.12627293169498444
156.25 - 158.00781 = 0.05435638129711151
158.00781 - 159.76562 = 0.04723625257611275
159.76562 - 161.52344 = 0.039802972227334976
161.52344 - 163.28125 = 0.03252151608467102
163.28125 - 165.03906 = 0.02576565369963646
165.03906 - 166.79688 = 0.019793687388300896
166.79688 - 170.3125 = 0.025394245982170105
170.3125 - 173.82812 = 0.012524348683655262
173.82812 - 177.34375 = 0.005465530324727297
177.34375 - 184.375 = 0.0028313857037574053
184.375 - 212.5 = 2.9355514561757445E-4
212.5 - 325.0 = 2.0522628041419466E-10
Mean = 150.0080160145819
Variance = 101.10184464125241
```

The full code listing for the above is provided in the method *testSimpleExpression()* .

7.2 Defining expressions on nodes with parents

In this example we are going to create a three node BN with three Continuous Interval nodes, *A*, *B* and *C*, where *A* and *B* are parents of *C*. We will declare Uniform distributions on *A* and *B* and we will make *C* a simple arithmetic function of *A* and *B*.

First create a new model and get the single BN:

```
Model m = Model.createEmptyModel();
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Create three Continuous Interval nodes, *A* and *B*, and *C*:

```
ContinuousIntervalEN a = ebn.addContinuousIntervalNode("a", "A");
ContinuousIntervalEN b = ebn.addContinuousIntervalNode("b", "B");
ContinuousIntervalEN c = ebn.addContinuousIntervalNode("c", "C");
```

Redefine the states of each node as just a single state range from 0 to infinity

```
DataSet cids = new DataSet();
cids.addIntervalDataPoint(0.0, Double.POSITIVE_INFINITY);
a.createExtendedStates(cids);
b.createExtendedStates(cids);
c.createExtendedStates(cids);
```

Make *C* the child of both *A* and *B*:

```
a.addChild(c);
b.addChild(c);
```

Make all nodes simulation nodes:

```
a.setSimulationNode(true);
b.setSimulationNode(true);
c.setSimulationNode(true);
```

Define the NPTs of *A* and *B* to be Uniform:

```
List uniformParameters = new ArrayList();
uniformParameters.add("0.0"); //lower bound
uniformParameters.add("10000000000.0"); //upper bound
ExtendedNodeFunction uniform =
new ExtendedNodeFunction(Uniform.displayName, uniformParameters);
a.setExpression(uniform);
b.setExpression(uniform);
```

Define the NPT of C as an arithmetic expression of the parents (we will use a simple sum):

```
List parameters = new ArrayList();
```

```
parameters.add("a + b");
ExtendedNodeFunction enf =
        new ExtendedNodeFunction(Arithmetic.displayName, parameters);
c.setExpression(enf);
```

Now regenerate the NPTs:

```
ebn.regenerateNPT(a);
ebn.regenerateNPT(b);
ebn.regenerateNPT(c);
```

Calculate the entire model and print the marginals:

```
m.calculate();
printMarginals(m, ebn, c);
```

Now we are going to enter observations for A and B. First we get the single Scenario:

```
Scenario s = m.getScenarioAtIndex(0);
s.addRealObservation(ebn.getId(), a.getId(), 33.0);
s.addRealObservation(ebn.getId(), b.getId(), 44.0);
```

Calculate again:

```
m.calculate();
printMarginals(m, ebn, c);
```

Save the model:

```
m.save("test3.cmp");
```

Finally, remember to enclose the above code inside a try-catch exception.

When you run the method you should get the following output (note as the method contains two instances of the printMarginals() method for node C there are two sets of data here for C representing the marginals before and after entering the evidence):

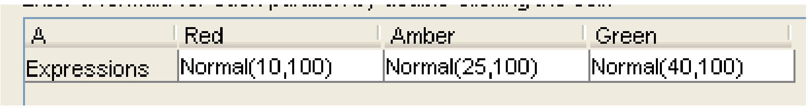
```
C
100.0 - 1.0E7 = 4.999976113140292E-7
1.0E7 - 1.0E8 = 5.270000474411063E-5
1.0E8 - 1.0E9 = 0.004941313993185759
1.0E9 - 1.5625E9 = 0.007281494792550802
1.5625E9 - 2.125E9 = 0.010454691015183926
2.125E9 - 3.25E9 = 0.030262276530265808
3.25E9 - 3.8125E9 = 0.019819753244519234
3.8125E9 - 4.375E9 = 0.022984370589256287
4.375E9 - 5.5E9 = 0.05576619505882263
5.5E9 - 6.0625E9 = 0.032445307821035385
6.0625E9 - 6.625E9 = 0.03559710085391998
6.625E9 - 7.1875E9 = 0.03877400606870651
7.1875E9 - 7.75E9 = 0.042113292962312 7
7.75E9 - 8.3125E9 = 0.04524944722652435
8.3125E9 - 8.875E9 = 0.04817969352006912
8.875E9 - 1.0E10 = 0.10633198916912079
1.0E10 - 1.125E10 = 0.1166599169373512 3
1.125E10 - 1.1875E10 = 0.05249999091029167
1.1875E10 - 1.25E10 = 0.04916015639901161
1.25E10 - 1.3125E10 = 0.04505859687924385
1.3125E10 - 1.375E10 = 0.040957026183605194
1.375E10 - 1.4375E10 = 0.03720703348517418
1.4375E10 - 1.5E10 = 0.033076174557209015
```

```
1.5E10 - 1.5625E10 = 0.02920898236334324
1.5625E10 - 1.625E10 = 0.025341803207993507
1.625E10 - 1.6875E10 = 0.021591799333691597
1.6875E10 - 1.75E10 = 0.0175781287252903
1.75E10 - 1.8125E10 = 0.01374023500829935
1.8125E10 - 1.875E10 = 0.009638672694563866
1.875E10 - 2.0E10 = 0.008027344010770321
Mean = 1.0000016546718266E10
Variance = 1.6748964277031E19
C
76.225586 - 76.269531 = 0.02566964365541935
76.269531 - 76.357422 = 0.057071834802627563
76.357422 - 76.445312 = 0.057071834802627563
76.445312 - 76.796875 = 0.22828733921051025
76.796875 - 77.5 = 0.4565746784210205
77.5 - 77.675781 = 0.11414366960525513
77.675781 - 77.719727 = 0.028535917401313782
77.719727 - 77.741699 = 0.014267958700656891
77.741699 - 77.763672 = 0.014267958700656891
77.763672 - 77.774658 = 0.004109172150492668
77.774658 - 77.785645 = 0.0
Mean = 76.99995349023425
Variance = 0.17619509565760666
```

The full code listing for the above appears in the method *testExpressionWithParent().*

7.3 Defining partitioned expressions

In this example we are going to create a BN with two nodes A (a labeled node) and B (a continuous node that is a child of A). We are going to make the NPT of B the partitioned expression shown in Figure 4.



A	Red	Amber	Green
Expressions	Normal(10,100)	Normal(25,100)	Normal(40,100)

Figure 4 The partitioned expression you will create as the NPT for node B

To do this we need to make node A be what is called a *model* node for B (this means that the NPT for B is a partitioned expression conditioned on A).

The method in this example is called *testPartitionedExpression().*

First, as usual create a new model and get the single BN:

```
Model m = Model.createEmptyModel();
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Create a Labelled node, *A*, and set up new states for it:

```
LabelledEN a = ebn.addLabelledNode("a", "A");
DataSet lds = new DataSet();
lds.addLabelledDataPoint("Red");
lds.addLabelledDataPoint("Amber");
lds.addLabelledDataPoint("Green");
```

```
a.createExtendedStates(lds);
```

Create a Continuous Interval node, *B*:

```
ContinuousIntervalEN b = ebn.addContinuousIntervalNode("b", "B");
```

Make it a child of *A*:

```
a.addChild(b);
```

Make it a simulation node:

```
b.setSimulationNode(true);
```

Set the new state range to be 0 to infinity:

```
DataSet cids = new DataSet();
cids.addIntervalDataPoint(0.0, Double.POSITIVE_INFINITY);
b.createExtendedStates(cids);
```

Make *A* a model node of *B*:

```
List modelNodes = new ArrayList();
b.setPartitionedExpressionModelNodes(modelNodes);
modelNodes.add(a);
```

Next we create a partitioned expression for each state of *A*. First we need a List to store the expressions we will create:

```
List expressions = new ArrayList();
b.setPartitionedExpressions(expressions);
```

Create the expression Normal(10, 100) for the first state (**Red**):

```
List redParameters = new ArrayList();
redParameters.add("10");
redParameters.add("100");
expressions.add(new ExtendedNodeFunction(Normal.displayName,
                                                        redParameters));
```

Create the expression Normal(25, 100) for second state (**Amber**):

```
List amberParameters = new ArrayList();
amberParameters.add("25");
amberParameters.add("100");
expressions.add(new ExtendedNodeFunction(Normal.displayName,
                                                            amberParameters));
```

Create the expression Normal(40, 100) for third state (**Green**):

```
List greenParameters = new ArrayList();
greenParameters.add("40");
greenParameters.add("100");
expressions.add(new ExtendedNodeFunction(Normal.displayName,
                                                            greenParameters));
```

Now regenerate the NPT for *B*:

```
ebn.regenerateNPT(b);
```

Calculate the entire model and print the marginals:

```
m.calculate();
printMarginals(m, ebn, b);
```

Next we will add observations, so we need to get the single scenario:

```
Scenario s = m.getScenarioAtIndex(0);
```

Enter observation **Red**, calculate and view the marginals:

```
s.addHardEvidenceObservation(ebn.getId(),
                    a.getId(), a.getExtendedStateAtIndex(0).getId());
m.calculate();
printMarginals(m, ebn, b);
```

Enter observation **Amber**, calculate and view the marginals:

```
s.addHardEvidenceObservation(ebn.getId(),
               a.getId(), a.getExtendedStateAtIndex(1).getId());
m.calculate();
printMarginals(m, ebn, b);
```

Enter observation **Green**, calculate and view the marginals:

```
s.addHardEvidenceObservation(ebn.getId(),
        a.getId(), a.getExtendedStateAtIndex(2).getId());
m.calculate();
printMarginals(m, ebn, b);
```

Save the model (so you can inspect it in AgenaRisk):

```
m.save("test4.cmp");
```

Finally, remember to enclose the above code inside a try-catch exception.

When you run the method you should get the following output (note the method contains four instances of the printMarginals() method for node B there are four sets of data here for B):

```
B
0.0 - 1.0 = 0.010737423785030842
1.0 - 3.25 = 0.028303800150752068
3.25 - 5.5 = 0.03398272022604942
5.5 - 7.75 = 0.03924122452735901
7.75 - 10.0 = 0.04373741149902344
10.0 - 12.8125 = 0.059497151523828506
12.8125 - 15.625 = 0.06298930943012238
15.625 - 21.25 = 0.12994219362735748
21.25 - 26.875 = 0.12889264523983002
26.875 - 32.5 = 0.12401936203241348
32.5 - 35.3125 = 0.05888725444674492
35.3125 - 38.125 = 0.05557863414287567
38.125 - 40.9375 = 0.050962239503860474
40.9375 - 42.34375 = 0.02330564521253109
42.34375 - 43.75 = 0.021634183824062347
```

```
43.75 - 46.5625 = 0.03774848207831383
46.5625 - 47.96875 = 0.015961604192852974
47.96875 - 49.375 = 0.013996724970638752
49.375 - 52.1875 = 0.022320454940199852
52.1875 - 55.0 = 0.015537639148533344
55.0 - 57.8125 = 0.010071229189634323
57.8125 - 60.625 = 0.006064079236239195
60.625 - 66.25 = 0.005138171371072531
66.25 - 77.5 = 0.0014209174551069736
77.5 - 100.0 = 2.949854388134554E-5
100.0 - 212.5 = 3.2888369805306183E-10
Mean = 26.0338134406882
Variance = 209.07894645516527
B
0.0 - 1.0 = 0.030195554718375206
1.0 - 2.125 = 0.037362199276685715
2.125 - 3.25 = 0.04081949591636658
3.25 - 4.375 = 0.04403642937541008
4.375 - 5.5 = 0.04691004380583763
5.5 - 6.625 = 0.04934336990118027
6.625 - 7.75 = 0.051250848919153214
7.75 - 10.0 = 0.10579535365104675
10.0 - 12.8125 = 0.13162341713905334
12.8125 - 14.21875 = 0.06263924390077591
14.21875 - 15.625 = 0.05903695151209831
15.625 - 17.03125 = 0.05455406382679939
17.03125 - 18.4375 = 0.04942607134580612
18.4375 - 19.84375 = 0.04390468820929527
19.84375 - 21.25 = 0.03823767229914665
21.25 - 22.65625 = 0.03265110030770302
22.65625 - 24.0625 = 0.02733568474650383
24.0625 - 25.46875 = 0.02243819274008274
25.46875 - 26.875 = 0.018058080226182938
26.875 - 28.28125 = 0.014248888939619064
28.28125 - 29.6875 = 0.011023416183888912
29.6875 - 31.09375 = 0.008361364714801311
31.09375 - 32.5 = 0.006218188442289829
32.5 - 35.3125 = 0.0077752103097736835
35.3125 - 38.125 = 0.0038330769166350365
38.125 - 43.75 = 0.0024827192537486553
43.75 - 55.0 = 4.3463846668601036E-4
55.0 - 100.0 = 4.038383849547245E-6
Mean = 12.887065302041265
Variance = 63.16632073329458
B
0.0 - 1.0 = 0.002000291831791401
1.0 - 5.5 = 0.01749918796122074
5.5 - 7.75 = 0.01677987352013588
7.75 - 10.0 = 0.024696823209524155
10.0 - 12.8125 = 0.04494141787290573
12.8125 - 14.21875 = 0.02920076809823513
14.21875 - 15.625 = 0.03397268429398537
15.625 - 17.03125 = 0.038751740008592606
17.03125 - 18.4375 = 0.043338946998119354
18.4375 - 21.25 = 0.09861093014478683
21.25 - 26.875 = 0.22191345691680908
26.875 - 29.6875 = 0.10667254030704498
29.6875 - 32.5 = 0.09357790648937225
32.5 - 33.90625 = 0.04031248763203621
33.90625 - 35.3125 = 0.0355742946267128
35.3125 - 36.71875 = 0.03077930584549904
36.71875 - 38.125 = 0.02611001394689083
38.125 - 39.53125 = 0.0217160675674769
39.53125 - 40.9375 = 0.017708469182252884
40.9375 - 43.75 = 0.02525649219751358
43.75 - 46.5625 = 0.014957166276872158
46.5625 - 49.375 = 0.00818831380456686
```

```
49.375 - 55.0 = 0.006082479376345873
55.0 - 100.0 = 0.0013583328109234571
100.0 - 156.25 = 3.2174214464539275E-14
Mean = 25.19825286214885
Variance = 98.03294762143545
B
0.0 - 1.0 = 1.642562165216077E-5
1.0 - 10.0 = 0.001301842974498868
10.0 - 15.625 = 0.006044900510460138
15.625 - 18.4375 = 0.008137725293636322
18.4375 - 21.25 = 0.014864757657051086
21.25 - 24.0625 = 0.025100452825427055
24.0625 - 26.875 = 0.03918096423149109
26.875 - 29.6875 = 0.056537847965955734
29.6875 - 31.09375 = 0.03535450994968414
31.09375 - 32.5 = 0.04006342962384224
32.5 - 35.3125 = 0.09299976378679276
35.3125 - 38.125 = 0.10601349920034409
38.125 - 43.75 = 0.22054244577884674
43.75 - 46.5625 = 0.09800169616937637
46.5625 - 49.375 = 0.08158351480960846
49.375 - 50.78125 = 0.033762793987989426
50.78125 - 52.1875 = 0.029020359739661217
52.1875 - 55.0 = 0.04466376081109047
55.0 - 57.8125 = 0.0293723214417696
57.8125 - 60.625 = 0.01785629615187645
60.625 - 63.4375 = 0.010034915059804916
63.4375 - 66.25 = 0.005213198252022266
66.25 - 77.5 = 0.0042441654950380325
77.5 - 100.0 = 8.841909584589303E-5
100.0 - 156.25 = 9.866188976914714E-10
Mean = 40.01151810194175
Variance = 101.12114523585728
```

The full code listing for the above appears in the method *testPartitionedExpression().*

8 Working with multiple Risk Objects

In AgenaRisk you can create multiple BNs (risk objects) in the same model and link them. When you link BNs you can also link ‘output’ nodes to ‘input’ nodes providing they have the same set of states. When you subsequently run the model the marginals from the output node replace the marginals of the input node it is joined to (this is also known as ‘message passing’). In this example we will create a model with two simple BNs linked by an output and input node.

The name of the method in this example is *testBNOs().*

In this example you need to add the following import statement

```
import uk.co.agena.minerva.util.model.NameDescription;
```

As usual, first create an empty model and get the existing BN:

```
Model m = Model.createEmptyModel();
ExtendedBN one = m.getExtendedBNAtIndex(0);
```

However, now we want to rename the existing BN:

```
NameDescription name = new NameDescription("One", "First BN");
        one.setName(name);
```

Also, this time we want to create a new BN called “Two”:

```
ExtendedBN two = m.addExtendedBN("Two", "Second BN");
```

For simplicity we are going to populate both BNs with similar nodes, so we define a method to do this, which we call twice. The method is as follows:

```
private void populateSimpleExtendedBN(ExtendedBN ebn) throws Exception {

        // Create two nodes, A and B, and a child C
        ContinuousIntervalEN a = ebn.addContinuousIntervalNode("a", "A");
        ContinuousIntervalEN b = ebn.addContinuousIntervalNode("b", "B");
        ContinuousIntervalEN c = ebn.addContinuousIntervalNode("c", "C");
        a.addChild(c);
        b.addChild(c);

        List parameters = new ArrayList();
        parameters.add("a + b");
        ExtendedNodeFunction enf = new
                    ExtendedNodeFunction(Arithmetic.displayName, parameters);
        c.setExpression(enf);


        //For the node c redefine its states (default states are just 0-10,10-20)
        DataSet cids = new DataSet();
        cids.addIntervalDataPoint(0.0, 10.0);
        cids.addIntervalDataPoint(10.0, 20.0);
        cids.addIntervalDataPoint(20.0, 30.0);
        cids.addIntervalDataPoint(30.0, 40.0);
        c.createExtendedStates(cids);
        // Now regenerate the NPT for C
```

```
		ebn.regenerateNPT(a);
		ebn.regenerateNPT(b);
		ebn.regenerateNPT(c);
	}
```

Using this method we can now populate both BNs:

```
populateSimpleExtendedBN(one);
populateSimpleExtendedBN(two);
```

Now get the node *A* in BN “One” and set it as an output node:

```
ExtendedNode source = one.getExtendedNodeWithUniqueIdentifier("a");
source.setConnectableOutputNode(true);
```

Get the node *A* in BN “Two” and set it as an input node:

```
ExtendedNode target = two.getExtendedNodeWithUniqueIdentifier("a");
target.setConnectableInputNode(true);
```

Link the two nodes (and, thus, the two BNs)

```
m.link(source, target);
```

Add an observation to C in BN "one":

```
Scenario s = m.getScenarioAtIndex(0);
s.addRealObservation(one.getId(),
one.getExtendedNodeWithUniqueIdentifier("c").getId(), 30.0);
```

Calculate and print the marginals of *A* in both BNs:

```
m.calculate();
printMarginals(m, one,
one.getExtendedNodeWithUniqueIdentifier("a"));
printMarginals(m, two,
two.getExtendedNodeWithUniqueIdentifier("a"));
```

Save the model so that you can subsequently open it in AgenaRisk (making sure to view it in Risk Explorer view):

```
m.save("test5.cmp");
```

Finally, remember to enclose the above code inside a try-catch exception.

When you run the method you should get the following output:

```
A
0.0 - 10.0 = 0.3333333432674408
10.0 - 20.0 = 0.6666666865348816
Mean = 11.666667014360428
Variance = 22.222222884496173
A
0.0 - 10.0 = 0.3333333432674408
10.0 - 20.0 = 0.6666666865348816
Mean = 11.666667014360428
Variance = 22.222222884496173
```

The full code listing for the above method appears in *testBNOs*().

9 Creating a web application

This section explains how to set up a basic infrastructure to be able to use a web front-end interface to run AgenaRisk models with the AgenaRisk API. In this example we assume a very common server set-up (also known as LAMP for Linux Apache MySQL PHP):

- Server operating system (OS) is Linux
- Web server running in the OS is Apache with PHP installed
- Database running in the OS is MySQL

The primary components to running API over the Web are summarised in Figure 5.

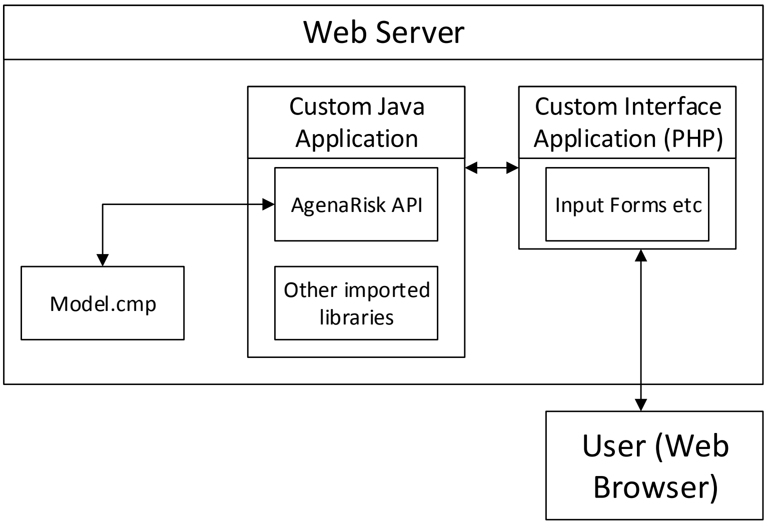


Figure 5 The Basic Set-Up

In this set-up the following sequence happens:

1. A visitor sends a request to the server (either directly, or via an input form provided by the server).
2. PHP processes this request, extracting information such as observations, model name, etc.
3. PHP launches the custom Java application via console (shell) and passes extracted data as parameters to it.
4. The custom Java application uses the AgenaRisk API to load the model, input observations sent from PHP and calculate. Then it prints desired results back to the console.
5. PHP reads the output from the console, processes it and prints them to the page that the user sees.

Important points to note regarding server configuration, implementation etc.:

- Java 8 must be installed on the OS.
- PHP must have permission to execute *shell_exec* command.
- Any location used for disk I/O must have read/write permissions for the web server user which executes PHP scripts. This username may vary.
- The “home” location considered by Java in this set-up will be the location of the PHP script which was executed.
- You must place license files into the “home” location.

If you need to run PHP scripts in different locations, then you will need to duplicate license files to all these locations.

- The OS user that is used to execute PHP scripts, and subsequently execute the custom Java application, must be configured to have read/write operations.
- Some models need a long time to calculate, and the PHP's request will time out, so special arrangements must be made, such as:
    - Increase PHP time-out period
    - Send results via email
    - Poll for results periodically
- Exception handling must be done in a way that does not corrupt console output. One option is to capture all console output to a buffer variable prior to the point when calculation results must be printed.

Detailed example for Linux step-by-step:

1. Install Linux from http://www.ubuntu.com/download/server
2. During installation, choose to install LAMP
    - The directory for the website (PHP scripts) will be: /var/www/html/
    - Note that /var/www/html/ is owned by user root
    - The user and group of the Apache server will be: www-data
      You can check this in Apache environmental variables by running:

```
sudo nano /etc/apache2/envvars
```

3. Install Java8:

```
sudo apt-get install oracle-java8-installer
```

4. Check your server's IP address by performing command:

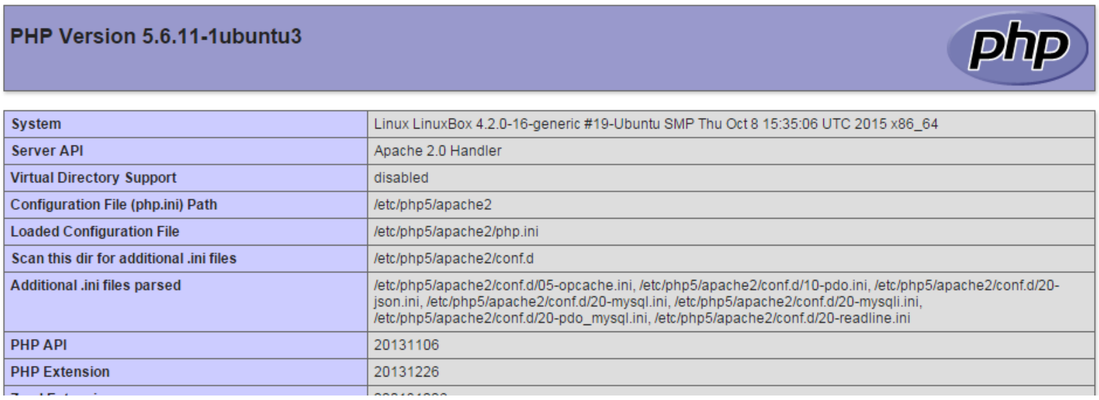
```
ifconfig
```

   Look for something like inet addr:192.168.60.3 in the output. "192.168.60.3" is the IP address in our example. Your server may have a different address.

5. Remove the default /var/www/html/index.html file and replace it with index.php file with contents:

```
<?php
        phpinfo();
```

6. Then open your browser and go to address: http://192.168.60.3/
7. You will see a page that looks similar to Figure 6:



PHP Version 5.6.11-1ubuntu3

System	Linux LinuxBox 4.2.0-16-generic #19-Ubuntu SMP Thu Oct 8 15:35:06 UTC 2015 x86_64
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-json.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-readline.ini
PHP API	20131106
PHP Extension	20131226

Figure 6 Example PHPINFO Page

8. In this example we use a tutorial model Asia.ast, supplied with AgenaRisk installation. Note that you can add a selection of models to your input form.

9. Create an input form `/var/www/html/form.html` with the code:

```
<html>
	<head><title>Basic API Input Form</title></head>
	<body>
		<form action="process_form.php">
			<p>
				<span>Visit to Asia?</span>
				<select name="A">
					<option value="">Please select observation</option>
					<option value="yes">yes</option>
					<option value="no">no</option>
				</select>
			</p>
			<p>
				<span>Smoker?</span>
				<select name="S">
					<option value="">Please select observation</option>
					<option value="yes">yes</option>
					<option value="no">no</option>
				</select>
			</p>
			<p>
				<span>Positive X-ray?</span>
				<select name="X">
					<option value="">Please select observation</option>
					<option value="yes">yes</option>
					<option value="no">no</option>
				</select>
			</p>
			<p>
				<span>Dyspnoea?</span>
				<select name="D">
					<option value="">Please select observation</option>
					<option value="yes">yes</option>
					<option value="no">no</option>
				</select>
			</p>
			<p><input type="submit" value="Submit"/></p>
		</form>
	</body>
</html>
```

   Note that copying from a PDF may result in broken quotation marks, so make sure to use simple straight quotes.

10. Create a Java application project in the normal way you would.
    Download and import library Commons JSON from
    https://sling.apache.org/downloads.cgi

Use the following code in the executable class of your application:

```
public static void main(String[] args) throws Exception {
	// Capture all output
	PrintStream output_console = System.out;
	ByteArrayOutputStream baos = new ByteArrayOutputStream();
	PrintStream output_capture = new PrintStream(baos);
	System.setOut(output_capture);
	System.setErr(output_capture);
	// Assume that first argument contains observations in encoded JSON format
	String encoded_json = args[0];
	String decoded_json = java.net.URLDecoder.decode(encoded_json, "UTF-8");
	JSONObject json = new JSONObject(decoded_json);
	// Load the model file
	Model model = Model.load("Asia.ast");
	// Get Risk object and scenario
	ExtendedBN ebn = model.getExtendedBNAtIndex(0);
	Scenario scenario = model.getScenarioAtIndex(0);
	String scenario_name = scenario.getName().getShortDescription();
	// Go over observations and put them in
	Iterator<String> variables = json.keys();
	while(variables.hasNext()){
		String var = variables.next();
		String val = json.getString(var);
		// Get node by var name
		ExtendedNode en = ebn.getExtendedNodeWithUniqueIdentifier(var);
		// Get state by var value
		ExtendedState state = en.getExtendedStateWithShortDesc(val);
		// Set observation
		scenario.addHardEvidenceObservation(ebn.getId(), en.getId(), state.getId());
	}
	// Calcualte the model and output desired variables in JSON format
	model.calculate();
	// Create output JSON
	JSONObject json_out = new JSONObject();
	json_out.put("nodes", new JSONObject());
	// Get marginals
	MarginalDataStore mds = model.getMarginalDataStore();
	// Get marginals for "yes" of variables tuberculosis (T), cancer (L) and bronchitis (B)
	for(String var: new String[]{"T","L","B"}){
		ExtendedNode en = ebn.getExtendedNodeWithUniqueIdentifier(var);
		MarginalDataItemList mdil = mds.getMarginalDataItemListForNode(ebn, en);
		List<MarginalDataItem> mdis = mdil.getMarginalDataItems();
		// Find marginals for scenario
		MarginalDataItem mdi = mdil.getMarginalDataItemAtIndex(0);
		for(MarginalDataItem mdi_temp: mdis){
			if (mdi_temp.getScenarioName().equalsIgnoreCase(scenario_name)){
				mdi = mdi_temp;
				break;
			}
		}
		// Find value for "yes"
		for(DataPoint dp: (List<DataPoint>)mdi.getDataset().getDataPoints()){
			if (dp.getLabel().equalsIgnoreCase("yes")){
				((JSONObject)json_out.get("nodes")).put(var, dp.getValue());
			}
		}
	}
	// Add captured console output to JSON for debugging
	json_out.put("output", baos.toString());
	// Stop capturing output
	System.out.flush();
	System.setErr(output_console);
	System.setOut(output_console);
	// Print results
	System.out.println(json_out.toString());
}
```

Again, make sure any copied quotes are simple straight quotes.

You may need to include the following imports:

```
        import java.io.ByteArrayOutputStream;
        import java.io.PrintStream;
        import java.util.Iterator;
        import java.util.List;
        import org.apache.sling.commons.json.JSONObject;
        import uk.co.agena.minerva.model.MarginalDataItem;
        import uk.co.agena.minerva.model.MarginalDataItemList;
        import uk.co.agena.minerva.model.MarginalDataStore;
        import uk.co.agena.minerva.model.Model;
        import uk.co.agena.minerva.model.extendedbn.ExtendedBN;
        import uk.co.agena.minerva.model.extendedbn.ExtendedNode;
        import uk.co.agena.minerva.model.extendedbn.ExtendedState;
        import uk.co.agena.minerva.model.scenario.Scenario;
        import uk.co.agena.minerva.util.model.DataPoint;
```

Make sure that this project is compiled with options:

```
        -Xlint:unchecked -Xlint:deprecation
```

Also make sure to set the main class of the project to the class that contains the code given above.

11. Create a PHP form processor `process_form.php` page with the code:

```
<?php
        $variables = array();
        # Copy non-blank variables from the form
        foreach($_POST as $var => $val){
                if (trim($val) !== ''){
                        $variables[$var] = $val;
                }
        }
        # Encode observations as JSON
        $json = json_encode($variables);
        # Escape JSON for passing it as console argument
        $command_line_formatted = urlencode($json);
        $console_command1 = 'cd '.getcwd().'/';
        $console_command2 = 'java -jar driver.jar "'.$command_line_formatted.'" 2>&1';
        $console_command = "$console_command1 && $console_command2";
        $output = shell_exec($console_command);
        $json_out = json_decode($output, true);
?>
<html>
        <head><title>Basic API Output</title></head>
        <body>
                <?php
                        foreach($json_out['nodes'] as $node => $value){
                                $node = htmlspecialchars($node, ENT_QUOTES, 'UTF-8');
                                $value = htmlspecialchars($value, ENT_QUOTES, 'UTF-8');
                                echo "<p>$node: $value</p>";
                        }
                ?>
                <p>Captured console output:</p>
                <p>
                        <textarea style="width:80%; height:300px;"><?php
                            echo htmlspecialchars($json_out['output'], ENT_QUOTES, 'UTF-8');
                        ?></textarea>
                </p>
        </body>
</html>
```

12. Place files into the Web server directory similarly to the following:

```
        groot@LinuxBox:/var/www/html$ ls -la /var/www/html/
        total 372
        drwxr-xr-x 3 root root   4096 Dec  3 14:08 .
        drwxr-xr-x 3 root root   4096 Dec  2 20:04 ..
        -rwxr-xr-x 1 root root 341793 Dec  3 14:03 Asia.ast
        -rwxr-xr-x 1 root root   6836 Dec  3 14:03 driver.jar
        -rwxr-xr-x 1 root root   1072 Dec  3 14:03 form.html
        -rw-r--r-- 1 root root     17 Dec  2 20:04 index.php
        drwxr-xr-x 2 root root   4096 Dec  3 13:40 lib
        -rwxr-xr-x 1 root root   1112 Dec  3 14:03 process_form.php
        -rwxr-xr-x 1 root root      0 Dec  3 14:03 ycodtcdu.alf
        -rwxr-xr-x 1 root root   1080 Dec  3 14:03 ycodtcdu.ull
```

where driver.jar is the custom Java application packaged into a JAR file.

and

```
        groot@LinuxBox:/var/www/html$ ls -la /var/www/html/lib
        total 33536
        drwxr-xr-x 2 root root    4096 Dec  3 13:40 .
        drwxr-xr-x 3 root root    4096 Dec  3 14:08 ..
        -rwxr-xr-x 1 root root  841794 Dec  3 13:40 colt.jar
        -rwxr-xr-x 1 root root  610259 Dec  3 13:40 commons-collections4-4.0.jar
        -rwxr-xr-x 1 root root 2169984 Dec  3 13:40 commons-collections4-4.0-javadoc.jar
        -rwxr-xr-x 1 root root  185140 Dec  3 13:40 commons-io-2.4.jar
        -rwxr-xr-x 1 root root  724124 Dec  3 13:40 commons-io-2.4-javadoc.jar
        -rwxr-xr-x 1 root root  315805 Dec  3 13:40 commons-lang3-3.1.jar
        -rwxr-xr-x 1 root root 1765707 Dec  3 13:40 commons-lang3-3.1-javadoc.jar
        -rwxr-xr-x 1 root root 2035066 Dec  3 13:40 commons-math3-3.4.1.jar
        -rwxr-xr-x 1 root root 6372470 Dec  3 13:40 commons-math3-3.4.1-javadoc.jar
        -rwxr-xr-x 1 root root  313898 Dec  3 13:40 dom4j-1.6.1.jar
        -rwxr-xr-x 1 root root   32058 Dec  3 13:40 foxtrot.jar
        -rwxr-xr-x 1 root root  236060 Dec  3 13:40 idb.jar
        -rwxr-xr-x 1 root root  223395 Dec  3 13:40 jaxen-1.1.3.jar
        -rwxr-xr-x 1 root root  307510 Dec  3 13:40 jcommon-1.0.5.jar
        -rwxr-xr-x 1 root root   54091 Dec  3 13:40 jdic.jar
        -rwxr-xr-x 1 root root   31649 Dec  3 13:40 jdic_stub.jar
        -rwxr-xr-x 1 root root  216332 Dec  3 13:40 jep3.jar
        -rwxr-xr-x 1 root root 1425747 Dec  3 13:40 jfreechart-1.0.13.jar
        -rwxr-xr-x 1 root root 1395616 Dec  3 13:40 jfreechartc-1.0.5.jar
        -rwxr-xr-x 1 root root 1457794 Dec  3 13:40 jide-common.jar
        -rwxr-xr-x 1 root root  140845 Dec  3 13:40 jide-data.jar
        -rwxr-xr-x 1 root root 1900911 Dec  3 13:40 jide-grids.jar
        -rwxr-xr-x 1 root root  286348 Dec  3 13:40 jide-treemap.jar
        -rwxr-xr-x 1 root root  473185 Dec  3 13:40 JRex.jar
        -rwxr-xr-x 1 root root  118932 Dec  3 13:40 junit-3.8.2.jar
        -rwxr-xr-x 1 root root  196787 Dec  3 13:40 junit-4.5.jar
        -rwxr-xr-x 1 root root 9325301 Dec  3 13:40 minerva.jar
        -rwxr-xr-x  1  root  root     61836  Dec   3  13:40  org.apache.sling.commons.json-
2.0.8.jar
        -rwxr-xr-x 1 root root  170030 Dec  3 13:40 phredist.jar
        -rwxr-xr-x 1 root root  802216 Dec  3 13:40 poi-2.5.1.jar
        -rwxr-xr-x 1 root root   80054 Dec  3 13:40 servlet.jar
```

13. Try to run this simple framework
    a. Open http://192.168.60.3/form.html
    b. Select some observations and click “Submit”
    c. You should see calculated output for nodes “T”, “L” and “B”.
14. However, you will also notice that in the output there are errors:

```
java.io.FileNotFoundException: random.seed (Permission denied)
java.io.FileNotFoundException: /var/www/AgenaRisk/Original_0.cor (No such file
or directory)
```

    This is an issue with permissions in the OS. You must do the following:

```
        sudo mkdir /var/www/AgenaRisk
        sudo chown -R www-data:www-data /var/www/
```

15. Try running the whole thing again. There should be no errors.
16. If you still notice errors related to uk.co.agena.minerva.util.io.FileHandler, it may mean that your temporary folder contains files that belong to a different OS user. You can find out the location of the temporary folder by running a command:

```
        java -XshowSettings:properties -version
```

    Then look for a property java.io.tmpdir = /tmp where /tmp is the temporary folder. The most common problem may be caused by XML files that were created by AgenaRisk when ran by another user (e.g. a local OS user other than www-data). They look like this:

```
        -rw-rw-r--  1 groot    groot    59958 Dec  3 13:54 Asia_model.xml
        -rw-rw-r--  1 groot    groot       51 Dec  3 13:54 Asia_report.xml
        -rw-rw-r--  1 groot    groot       51 Dec  3 13:54 Asia_taxonomy.xml
```

    Note that they belong to user groot, and not to www-data. The fix is to remove these files manually if they are causing the problem.
17. You can download an example archive without licence files from Agena website. Note that libraries and AgenaRisk API will most likely be outdated.
    Files to go into /var/www/html/:
    http://agena.co.uk/lholders/web_deployment/web_html_files.zip
    NetBeans example project for custom Java application:
    http://agena.co.uk/lholders/web_deployment/Simple_AgenaRisk_Driver_NBProject.zip

10 Learning probability tables from data

This section explains how to learn probability tables from data. Example files, i.e., models and datasets, used in this tutorial are located in “Examples\Advanced\Table Learning” folder. We are going to use the *Asia* model.

The learning process is performed using an Expectation-Maximisation (EM) algorithm. Note that currently AgenaRisk supports table learning for Boolean and Labelled nodes (learning for ranked nodes and continuous/simulation nodes is planned for a future release).

The first section provides a helper method that will be used in several further sections. The second section explains how to prepare data. The following three sections demonstrate how to perform table learning with different settings.

10.1 Getting node probability tables

To simplify providing output for learning probability tables from data we prepare a helper method that will print out probability tables for a relevant BN. This method is called *printNPTs()*. It expects a parameter – an ExtendedBN for which the probability tables should be printed.

First, we need to import additional classes:

```
import java.util.List;
import java.util.Arrays;
```

The method itself has been defined as follows:

```
private void printNPTs(ExtendedBN ebn) {
        System.out.println("=====================");
        ((List<ExtendedNode>) ebn.getExtendedNodes())
                .stream()
                .forEach(e -> {
                        System.out.println(e.getConnNodeId() + " " + e
                              + " " + e.getConfidence());
                        try {
                              System.out.println(Arrays.deepToString(e.getNPT()));
                        } catch (ExtendedBNException ex) {
                }
        });
}
```

This method prepares a list of nodes in the provided ExtendedBN object. Then, for each node in the list it first prints basic information on a node, i.e., a node’s unique identifier, node’s name, and node’s confidence (a knowledge to data ratio – a degree at which probability tables should be learnt from knowledge rather than from data – see examples in Section 0).
Finally, a probability table is printed for each node. Note the exception handling – the method *getNPT()* may throw an ExtendedBNException.

10.2 Generating example data file as a template

In this example we show how to generate an example data file. Such a data file is used when learning probability tables from data. Generating such a data file is useful to understand how the data should be formatted.

The method in this section is called *generateExampleDataFile().*

First, we need to import additional relevant classes:

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import uk.co.agena.minerva.util.io.CSVWriter;
import uk.co.agena.minerva.util.model.SampleDataGenerator;
```

The process of generating an example data file requires providing a BN for which the data should be generated. Thus, we start with loading an example *Asia.ast* model that is provided with the AgenaRisk distribution:

```
//Load example model
String examplesDirectory = System.getProperty("user.home")
        + System.getProperty("file.separator")
        + "AgenaRisk" + System.getProperty("file.separator")
        + "Examples" + System.getProperty("file.separator")
        + "Advanced" + System.getProperty("file.separator")
        + "Table Learning" + System.getProperty("file.separator");
Model m = Model.load(examplesDirectory + "Asia.ast");
```

If you would like to generate such an example data file for another model, you need to load the model like this:

```
Model m = Model.load(pathToYourModel);
```

Then, choose a single ExtendedBN:

```
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Next, provide the number of samples, i.e. rows of data, that should be generated and the name of the output file for the dataset.

```
int numberOfDataSamplesToGenerate = 10;
File dataFile = new File("data_example.csv");
```

Generating the data can be executed as follows:

```
//Initialize generator
SampleDataGenerator generator = new SampleDataGenerator();
//Generate data
List sampleData = generator.generateDataForEBN(ebn,
        numberOfDataSamplesToGenerate, true);
```

Generating the data requires three parameters: an ExtendedBN object for which the data should be generated, number of samples to generate, and a Boolean flag indicating if the generated dataset should contain a first row with node unique identifiers. This last parameter should always be kept *true* if generated data will later be used to learn probability tables.

When the data have been generated, it is time to save them in a file:

```
try (CSVWriter writer =
        new CSVWriter(new BufferedWriter(new FileWriter(dataFile)), ',', '\0')) {
            writer.writeAll(sampleData);
} catch (IOException ex) {
        //custom exception handling for problems with saving data file
}
```

Note that this example uses a try-with-resources structure that will close resources for file access upon either successful or unsuccessful save operation. It uses a comma ‘,’ to separate values and a null character ‘\0’ for quoting values.

Since the operation of saving data to a file may throw an *IOException*, you may provide a custom exception handling.

The complete example can be found in the *generateExampleDataFile()* method.

Once such an example data file is generated you may open it in a tool supporting .csv files to see how the data should be formatted. Your real data may contain missing values which you may encode as special strings, such as “NA” which is the default and recommended encoding used in AgenaRisk.

10.3 Learning from data alone

In this example we show how to learn probability tables from data alone, i.e. during the learning process existing probability tables will not be taken into consideration and will be completely replaced. The method in this section is called *learnTablesPurelyFromData().*

First, we need to import additional relevant classes:

```
import uk.co.agena.minerva.model.MessagePassingLinkException;
import uk.co.agena.minerva.model.PropagationException;
import uk.co.agena.minerva.model.PropagationTerminatedException;
import uk.co.agena.minerva.model.corebn.CoreBNException;
import uk.co.agena.minerva.model.corebn.CoreBNInconsistentEvidenceException;
import uk.co.agena.minerva.util.EM.Data;
import uk.co.agena.minerva.util.EM.EMCal;
import uk.co.agena.minerva.util.io.FileHandlingException;
```

Now load the model file and get the single BN from it:

```
//Load example model
String examplesDirectory = System.getProperty("user.home")
        + System.getProperty("file.separator")
        + "AgenaRisk" + System.getProperty("file.separator")
        + "Examples" + System.getProperty("file.separator")
        + "Advanced" + System.getProperty("file.separator")
        + "Table Learning" + System.getProperty("file.separator");
Model m = Model.load(examplesDirectory + "Asia.ast");

// Get the single BN
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Next, we print the NPTs for the provided ExtendedBN object before executing the table learning process so that we can later see the differences in these NPTs.

```
printNPTs(ebn);
```

Then, we load the data by reading in the data from a .csv file and storing it in an object of class Data:

```
//Prepare data path
String dataFileName = examplesDirectory
        + "Asia_example_dataset_with_missing_values.csv";

//Read in data
Data data = new Data(dataFileName, "NA");
```

```
//If the values in the data file are separated not by a comma,
//you can provide custom separator
data = new Data(dataFileName, "NA", ",");
```

If you prefer to have some technical information provided during table learning to be logged into a log file you can set this up:

```
m.setEMLogging(true);
```

Then, set up the object of EMCal class that is used for table learning:

```
Model.EM_ON = true;    //important!
EMCal emcal = new EMCal(m, ebn, data, "NA");
```

Creating an object of EMCal class requires four parameters: an object of Model class, an object of ExtendedBN class, an object of Data class, and a String that represents how the missing values are represented in the dataset (in our example as “NA” strings).

You may also explicitly provide advanced settings for the calculation process. However, this is optional, and if you do not provide them the following default values will be used:

```
emcal.setMaxIterations(25);      //default is 20
emcal.threshold = 0.01;          //default is 0.01
```

Then, start the table learning process by calling:

```
emcal.calculateEM();
```

When the calculation is completed, you may check the learnt probability tables by calling:

```
printNPTs(ebn);
```

Finally, if you want to save the model call:

```
m.save("model_learnt_purely_from_data.cmp");
```

Since certain statements may throw an exception you may surround the above block of code with try-catch. Specifically, during creation of EMCal object, the data from provided dataset are checked if they are consistent with the model’s nodes and their possible states.

```
try {

    //the above block of code

} catch (FileHandlingException | IOException | CoreBNException
            | CoreBNInconsistentEvidenceException | PropagationException
            | MessagePassingLinkException | PropagationTerminatedException ex) {
   //custom exception handling
} catch (InconsistentDataVsModelStatesException ex) {
   //custom exception handling
} catch (ExtendedBNException ex) {
   //custom exception handling
}
```

The complete example can be found in *learnTablesPurelyFromData()* method.

10.4 Learning from data with incorporation of expert judgement

In this example we show how to learn probability tables from data but also with incorporation of expert judgement. During the learning process existing probability tables (treated as expert knowledge) will partially be taken into consideration. The method in this section is called *learnTablesWithExpertJudgement().*

First we need to import additional relevant classes:

```
import uk.co.agena.minerva.model.MessagePassingLinkException;
import uk.co.agena.minerva.model.PropagationException;
import uk.co.agena.minerva.model.PropagationTerminatedException;
import uk.co.agena.minerva.model.corebn.CoreBNException;
import uk.co.agena.minerva.model.corebn.CoreBNInconsistentEvidenceException;
import uk.co.agena.minerva.util.EM.Data;
import uk.co.agena.minerva.util.EM.EMCal;
import uk.co.agena.minerva.util.io.FileHandlingException;
```

Now load the model file and get the single BN from it:

```
//Load example model
String examplesDirectory = System.getProperty("user.home")
        + System.getProperty("file.separator")
        + "AgenaRisk" + System.getProperty("file.separator")
        + "Examples" + System.getProperty("file.separator")
        + "Advanced" + System.getProperty("file.separator")
        + "Table Learning" + System.getProperty("file.separator");
Model m = Model.load(examplesDirectory + "Asia.ast");

// Get the single BN
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Next, we print the NPTs for the provided ExtendedBN object before executing the table learning process so that we can later see the differences in these NPTs.

```
printNPTs(ebn);
```

Then, we load the data by reading in the data from a .csv file and storing it in object of class Data:

```
//Prepare data path
String dataFileName = examplesDirectory
        + "Asia_example_dataset_with_missing_values.csv";

//Read in data
Data data = new Data(dataFileName, "NA");

//If the values in the data file are separated not by a comma,
//you can provide custom separator
data = new Data(dataFileName, "NA", ",");
```

If you prefer to have some technical information provided during table learning to be logged into a log file you can set this up:

```
m.setEMLogging(true);
```

As a main difference in comparison with the previous example, here we set a ratio of knowledge to data at which these two sources should be mixed. For example, a value ‘0’ means that only the data will be used; a value ‘0.5’ means that the data and knowledge are equally important; a value of ‘0.75’ means that knowledge is three times more important than data (the confidence in data is 1 – 0.75 = 0.25); and a value of ‘1’ means that only knowledge is important – in this case the probability tables will be kept unchanged. In this example we assume a value ‘0.5’:

```
ebn.setConfidence(0.5);
```

Then, set up the object of EMCal class that is used for table learning:

```
Model.EM_ON = true;    //important!
EMCal emcal = new EMCal(m, ebn, data, "NA");
```

Creating an object of EMCal class requires four parameters: an object of Model class, an object of ExtendedBN class, an object of Data class, and a String that represents how the missing values are represented in the dataset (in our example as “NA” strings).

You may also explicitly provide advanced settings for the calculation process. However, this is optional, and if you do not provide them default values will be used:

```
emcal.setMaxIterations(25);       //default is 20
emcal.threshold = 0.01;           //default is 0.01
```

Then, start table learning process by calling:

```
emcal.calculateEM();
```

When the calculation is completed, you may check the learnt probability tables by calling:

```
printNPTs(ebn);
```

Finally, if you want to save the model call:

```
m.save("model_learnt_from_data_and_expert_judgement.cmp");
```

As in the previous example, certain statements may throw an exception. Thus, you may surround the above block of code with try-catch:

```
try {

    //the above block of code

} catch (FileHandlingException | IOException | CoreBNException
            | CoreBNInconsistentEvidenceException | PropagationException
            | MessagePassingLinkException | PropagationTerminatedException ex) {
    //custom exception handling
} catch (InconsistentDataVsModelStatesException ex) {
    //custom exception handling
} catch (ExtendedBNException ex) {
    //custom exception handling
}
```

The complete example can be found in *learnTablesWithExpertJudgement()* method.

10.5 Learning from data with nodes’ individual settings for incorporation of expert judgement

Section 10.4 showed how to learn probability tables from data and expert judgement. But it was assumed there that the balance between knowledge and data applied to the whole BN. In this example we show how it is possible also to provide settings for individual nodes, using a new method called:

learnTablesWithExpertJudgementForIndividualNodes().

First we need to import additional relevant classes:

```
import java.util.List;
import java.util.ArrayList;
import uk.co.agena.minerva.model.MessagePassingLinkException;
import uk.co.agena.minerva.model.PropagationException;
import uk.co.agena.minerva.model.PropagationTerminatedException;
import uk.co.agena.minerva.model.corebn.CoreBNException;
import uk.co.agena.minerva.model.corebn.CoreBNInconsistentEvidenceException;
import uk.co.agena.minerva.util.EM.Data;
import uk.co.agena.minerva.util.EM.EMCal;
import uk.co.agena.minerva.util.io.FileHandlingException;
```

Now load the model file and get the single BN from it:

```
//Load example model
String examplesDirectory = System.getProperty("user.home")
        + System.getProperty("file.separator")
        + "AgenaRisk" + System.getProperty("file.separator")
        + "Examples" + System.getProperty("file.separator")
        + "Advanced" + System.getProperty("file.separator")
        + "Table Learning" + System.getProperty("file.separator");
Model m = Model.load(examplesDirectory + "Asia.ast");

// Get the single BN
ExtendedBN ebn = m.getExtendedBNAtIndex(0);
```

Next, we print the NPTs for the ExtendedBN object before executing the table learning process so that we can later see the differences in these NPTs.

```
printNPTs(ebn);
```

Then, we load the data by reading in the data from a .csv file and storing it in object of class Data:

```
//Prepare data path
String dataFileName = examplesDirectory
        + "Asia_example_dataset_with_missing_values.csv";

//Read in data
Data data = new Data(dataFileName, "NA");

//If the values in the data file are separated not by a comma,
//you can provide custom separator
data = new Data(dataFileName, "NA", ",");
```

If you prefer to have some technical information provided during table learning to be logged into a log file you can set this up:

```
m.setEMLogging(true);
```

Similarly to the previous example we set up a ratio of knowledge to data:

```
ebn.setConfidence(0.5);
```

In addition to the previous example, here we provide values of knowledge to data ratio individually for some nodes. This ratio may take any values between 0 and 1 inclusive. In addition, a value -1 (default for each node) may be assigned to say that particular node will use the global setting for ExtendedBN provided in the above statement.

```
//only from data
```

```
ebn.getExtendedNodeWithUniqueIdentifier("D").setConfidence(0);

//knowledge is 3x more important than data
ebn.getExtendedNodeWithUniqueIdentifier("B").setConfidence(0.75);

//100% knowledge, do not learn
ebn.getExtendedNodeWithUniqueIdentifier("L").setConfidence(1);

//use the default confidence for the whole ExtendedBN but make it a fixed node
//this is another way of blocking NPT to learn from data
ebn.getExtendedNodeWithUniqueIdentifier("T").setConfidence(-1);
List<String> fixedNodes = new ArrayList();
fixedNodes.add("T");
```

Trying to provide a value outside the possible range of values will not change the previous setting:

```
//try to assign a value form outside possible range
ebn.getExtendedNodeWithUniqueIdentifier("A").setConfidence(1.3);
```

To verify what values of this confidence have been assigned to individual nodes you may run this:

```
//check confidence for individual nodes
((List<ExtendedNode>) ebn.getExtendedNodes())
        .stream()
        .forEach(e -> System.out.println(e.getConnNodeId() + " " + e + " "
                + e.getConfidence()));
```

This would provide the following output:

```
B Has bronchitis 0.75
D Dyspnoea? 0.0
A Visit to Asia? -1.0
S Smoker? -1.0
T Has tuberculosis -1.0
L Has lung cancer 1.0
E Tuberculosis or cancer -1.0
X Positive X-ray? -1.0
```

Then, set up the object of EMCal class that is used for table learning:

```
Model.EM_ON = true;    //important!
EMCal emcal = new EMCal(m, ebn, data, "NA");
```

Creating an object of EMCal class requires four parameters: an object of Model class, an object of ExtendedBN class, an object of Data class, and a String that represents how the missing values are represented in the dataset (in our example as “NA” strings).

You may also explicitly provide advanced settings for the calculation process. However, this is optional, and if you do not provide them default values will be used:

```
emcal.setMaxIterations(25);     //default is 20
emcal.threshold = 0.01;         //default is 0.01
```

Then, start table learning process by calling:

```
emcal.calculateEM();
```

When the calculation is completed, you may check the learnt probability tables by calling:

```
printNPTs(ebn);
```

Finally, if you want to save the model call:

```
        m.save("model_learnt_from_data_and_expert_judgement_individual_nodes.cmp");
```

As in previous example, certain statements may throw an exception. Thus, you may surround the above block of code with try-catch:

```
    try {

        //the above block of code

    } catch (FileHandlingException | IOException | CoreBNException
                | CoreBNInconsistentEvidenceException | PropagationException
                | MessagePassingLinkException | PropagationTerminatedException ex) {
       //custom exception handling
    } catch (InconsistentDataVsModelStatesException ex) {
       //custom exception handling
    } catch (ExtendedBNException ex) {
       //custom exception handling
    }
```

The complete example can be found in

learnTablesWithExpertJudgementForIndividualNodes() method.

11 Appendix: Summary of API classes and methods

This section lists the API classes and methods that you will find most useful. Full javadoc for these is available in the javadoc directory of your API distribution.

11.1 Model Class

createEmptyModel()

save(String filename)

load(String filename)

addExtendedBN(String name, String description):

deleteExtendedBN(ExtendedBN ebn)

deleteExtendedBNs(List ebns)

getExtendedBNAtIndex(int index)

getExtendedBNWithName(String name)

addScenario(String name)

deleteScenario(Scenario scenario)

deleteScenarios(List scenarios)

getApplicationVersionAndRevisionNumber()

getScenarioAtIndex(int index)

getScenarioWithName(String name)

clearAllObservations()

clearAllObservations(ExtendedBN ebn)

calculate()

calculate(List scenarios)

calculate(List scenarios, List exbns)

getSimulationNumberOfIteratons()

setSimulationNumberOfIteratons(int numberOfIts)

setSimulationEntropyConvergenceTolerance(double sim);

getMarginalDataStore()

link(ExtendedNode source, ExtendedNode target)

exportToCSV(String path, NamedFamilyMember familyMember, Scenario scenario)

importFromCSV(String path, NamedFamilyMember familyMember, Scenario scenario)

11.2 ExtendedBN Class

getId()

addLabelledNode(String id, String name)

addBooleanNode(String id, String name)

addContinuousIntervalNode(String id, String name) addIntegerIntervalNode(String id, String name)

addRankedNode(String id, String name)
addSimulationNode(String id, String name)
addDiscreteRealNode(String id, String name)
getExtendedNodeWithName(String name)
getExtendedNodeWithUniqueIdentifier(String name)
deleteExtendedNode(ExtendedNode enode)
regenerateNPT(ExtendedNode enode)

11.3 Node Class

getId()
createExtendedStates(DataSet ds)
addChild(Node x)
addParent(Node x)
getExtendedStates()
getExtendedStateAtIndex(int index)
getExtendedStateWithName(String name)
setNPT(double[] npt)
setNPT(double[][] values, List orderedParents)
getExpression()
setExpression(ExtendedNodeFunction expression)
getPartitionedExpressionModelNodes()
setPartitionedExpressionModelNodes(List modelNodes)
getPartitionedExpressions()
setPartitionedExpressions(List expressions)
getNPT()
isConnectableInputNode()
setConnectableInputNode(boolean connectableInputNode)
isConnectableOutputNode()
setConnectableOutputNode(boolean connectableOutputNode)

11.4 ContinuousEN Class

isSimulationNode()
setSimulationNode(boolean simulationNode)
setUniform()

11.5 State Class

getId()

createContinuousIntervalState(double lower, double upper)

createIntegerIntervalState(int lower, int upper)

createLabelledState(String name, String description)

createDiscreteRealState(double numericalValue)

11.6 Scenario Class

clearAllObservations()

clearObservationsForNode(int extendedBNId, int extendedNodeId)

addHardEvidenceObservation(int extendedBNId, int extendedNodeId, int extendedStateId)

addIntegerObservation(int extendedBNId, int extendedNodeId, int value)

addRealObservation(int extendedBNId, int extendedNodeId, double value)

addSoftEvidenceObservation(int extendedBNId, int extendedNodeId, int[] stateIds, double[] probabilities)

getObservations()

11.7 DataPoint Class

getLabel()

getValue()

11.8 AbsoluteDataPoint Class

getAbsoluteValue()

11.9 IntervalDataPoint Class

getIntervalLowerBound()

getIntervalUpperBound()

getMidPoint()

11.10 DataSet Class

addIntervalDataPoint(double lower, double upper)

addAbsoluteDataPoint(double value)

addLabelledDataPoint(String label)

getDataPoints()

11.11 MarginalDataStore Class

getMarginalDataItemListForNode(ExtendedBN exBN, ExtendedNode exNode)

11.12 MarginalDataItemList Class

getMarginalDataItemAtIndex(int index)

11.13 MarginalDataItem Class

getDataset()

getMeanValue()

getMedianValue()

getStandardDeviationValue()

getVarianceValue()

getLowerPercentile()

getPercentileValue()

getUpperPercentile()

11.14 Function Classes

There are different classes corresponding to each of the following functions (which are documented in the AgenaRisk manual). In each case there is only one publically accessible field call displayName:

Arithmetic

Beta

Binomial

ChiSquared

Comparative

Exponential

ExpressionParser

ExtremeValue

Function

Gamma

Geometric

Hypergeometric
Logistic
LogNormal
NegativeBinomial
Normal
Poisson

Student
TNormal
Triangle
Uniform
Weibull

11.15 MetaData Class

addMetaDataLink(MetaDataLink mdl)
addAggregateLink (MetaDataAggregationInfo mdai)
createNewMetaDataItem(NameDescription name, MetaDataType type)
addMetaDataItem(MetaDataItem mdi)

11.16 MetaDataItem Class

addExtendedBN(ExtendedBN exBN, Model m, Boolean generateQuestionnaire)
addQuestionnaire(Questionnaire q)
addScenario(Scenario s)
setName(NameDescription name)
setType(MetaDataType type)
setChildren(List children)
setAttributes(Map attributes)

11.17SensitivityAnalyser Class

setTarget(NodeBNPair target)
setSources(List sources)
addScenario(Scenario scenario)
exportTo(String htmlPath, String path)
analyse()

11.18 ClassQueryModel Class

loadUsingModel (Model model, String filename)

runInserts()

11.19 EMCal Class

calculateEM()

getLengthOfProgressableTask()

getMaxIterations()

isProgressableTaskDone()

resetProgressableTask()

setFixedNodes(List<String> lst)

setMaxIterations(int maxIterations)

terminateProgressableTask()

updateCurrentProgress()

11.20 Data Class

Data(String filename, String missingType, String valueDelimeter)

Data(String filename, String missingType)

11.21 SampleDataGenerator Class

generateDataForEBN(ExtendedBN ebn, int numberOfSamples, boolean includeVariableNames)