SMILE Wrappers

Programmer's Manual

Version 1.4.1.R2, Built on 9/12/2019
BayesFusion, LLC

This page is intentionally left blank.

This page is intentionally left blank.

Introduction

1 Introduction

Welcome to SMILE Wrapper Programmer's Manual, Version 1.4.1.R2, Built on 9/12/2019.

For the most recent version of this manual, please visit http://support.bayesfusion.com/docs/.

SMILE is a C++ software library for performing Bayesian inference. In order to ensure that its functionality can be easily integrated into software written in other languages, BayesFusion, LLC, provides wrapper libraries for Java, Python and .NET. The names of these products are:

- jSMILE (Java and environments which can instantiate and use the JVM, such as R)
- PySMILE (Python 2.7 and 3.x)
- SMILE.NET (.NET)
- rSMILE (R, in development now)

Each wrapper includes SMILE, so you do not need to download SMILE or know how to use C++ SMILE in order to use the wrappers. We strive to ensure feature symmetry between the wrappers - features available in one wrapper are generally available in other wrappers.

We have also developed SMILE.COM, a wrapper exposing SMILE functionality through Windows' COM (Common Object Model). The target audience for SMILE.COM are Microsoft Excel users, although the library will work with any environment extensible through COM. This manual does not contain documentation for SMILE.COM.

If you are new to SMILE and would like to begin with an informal, tutorial-like introduction, please start with the Java, Python or .NET section of *Platforms and Wrappers* [12] (depending on the programming language you're going to use), followed by *Hello SMILE Wrapper!* [16] section. If you are an advanced user, please browse through the *Table of Contents* or search for the topic of your interest.

This manual refers to a good number of concepts that are assumed to be known to the reader, such as probability, utility, decision theory and decision analysis, Bayesian networks, influence diagrams, etc. Should you want to learn more about these, please refer to GeNIe manual. SMILE is GeNIe's Application Programmer's Interface (API) and practically every elementary operation performed with GeNIe translates to calls to SMILE methods. Being familiar with GeNIe may prove extremely useful in learning SMILE. Understanding some of SMILE's functionality may be easier when performed interactively in GeNIe.

Licensing

2 Licensing

SMILE wrappers are commercial products that require a development license to use. There are two types of development licenses: Academic and Commercial. Academic license is free of charge for research and teaching use by those users affiliated with an academic institution. All other use requires a commercial license, available for purchase from BayesFusion, LLC.

Deployment of SMILE library or SMILE wrappers, i.e., embedding them into user programs, requires a deployment license. There are two types of deployment licenses: Server license, which allows a program linked with SMILE to be deployed on a computer server, and end-user program license, which allows distribution of user programs that include SMILE. Please contact BayesFusion, LLC, for details of the licenses and pricing.

The licensing system is implemented as a fragment of code which contains your license key and must be executed as first call into SMILE wrapper that your program uses. **This code is not included in the SMILE wrapper binary distribution**, it is personalized by BayesFusion, LLC, for you or your organization. The license keys are provided in a compressed .zip file, which contains Java, R, Python, C# and VB.NET code. 6-month academic and free 30-day evaluation licenses can be obtained directly at https://download.bayesfusion.com.

Platforms and Wrappers

3 Platforms and Wrappers

In addition to ensuring feature parity between wrappers, we designed them to use similiar names for the classes and methods. For example, all three wrappers contain the `Network` class. At the same time, wrappers honor the idioms of the specific languages they target:

- `Network` class in jSMILE has `addNode` method
- `Network` class in PySMILE has `add_node` method
- `Network` class in SMILE.NET has `AddNode` method

As you can see, the naming choices for the methods reflect the naming conventions of different programming languages.

The remainder of this chapter describes the platform-specific details related to the use of SMILE Wrappers.

3.1 Java and jSMILE

jSMILE is compatible with JDK 7 and newer.

jSMILE contains two parts: the jar file and the native library, the latter is specific to your operating system. The jar file, named `jsmile-x.y.z.jar` where x.y.z is the version number, can be used as any Java jar. You can add it to your local Maven repository, for example.

The native library is used by the code in the jar file through Java Native Interface (JNI). The name of the native library file is platform-dependent:

- `jsmile.dll` on Windows
- `jsmile.so` on Linux
- `jsmile.jnilib` on macOS

In the static initializer of the smile.Wrapper class, jSMILE will attempt to load its native library. If the `jsmile.native.library` system property is set, jSMILE calls `System.load` method, which requires a complete path name of the native library (no relative paths are allowed), using the value of the property as the path name. Otherwise, the `System.loadLibrary` method is used. In such case, the native library must be located in one of the directories specified by the

`java.library.path property`. Note that `java.library.path` cannot be set once JVM is running, but `jsmile.native.library` can be set with a call to `System.setProperty`.

Your license key should be pasted into the source code of your program.

The classes defined in jSMILE are located in `smile` and `smile.learning` packages. There are two types of classes:

- simple objects with public data members
- wrappers for C++ objects from SMILE library

The second group of classes, which includes `smile.Network`, is derived from `smile.Wrapper`. Each object instance of these classes has a related C++ object, which is deallocated by the Java object's finalizer or by a call to `Wrapper.Dispose` method. To ensure early deallocation of C++ object resources associated with your Java object, `Dispose` should be called as soon as the program doesn't need the object anymore. If `Dispose` is not called, the deallocation will happen in the finalizer during garbage collection.

For details on using jSMILE from R, see the Appendix 132.

3.2 Python and PySMILE

PySMILE binaries are available for Python 2.7 and Python 3.x. PySMILE consists of a single dynamically loaded library. The library name is platform-dependent:

- `pysmile.pyd` on Windows
- `pysmile.so` on Linux and macOS

To use PySMILE, you just need to import the `pysmile` module:

```
import pysmile
```

Your license key can be pasted directly into your Python code, or imported:

```
import pysmile_license
```

3.3 .NET and SMILE.NET

SMILE.NET, which is compatible with .NET framework 4.x, consists of a single mixed-mode assembly named `smilenet.dll`. To use the library, simply add `smilenet.dll` as a reference in your project.

We build SMILE.NET using C++/CLI compiler in Visual Studio 2013 to create a seamless package containing both native code (the C++ SMILE library) and .NET wrapper code. This compiler enforces the use of the dynamic C++ runtime library. This means that `smilenet.dll` has a runtime dependency on VS2013 C++ runtime. Your Windows system is likely to have this component already installed. However, if you are getting error messages referring to missing DLLs like `MSVCP120.DLL` or `MSVCR120.DLL`, you need to download and install the "Visual C++ Redistributable Package for Visual Studio 2013" from Microsoft's website.

Your license key should be pasted into the sources of your program. We provide the keys as C# and VB.NET code.

The classes defined in SMILE.NET are located in `Smile` and `Smile.Learning` namespaces. There are two types of classes:

- simple objects with public data members
- wrappers for C++ objects from SMILE library

The second group of classes, which includes `Smile.Network`, is derived from `Smile.WrappedObject` and also implements .NET's `IDisposable` interface. Each object instance of these classes has a related C++ object, which is deallocated by the .NET object's finalizer or its `IDisposable::Dispose` method. You can use C#'s `using` statement to ensure deterministic finalization, otherwise the deallocation will be performed in the finalizer during garbage collection.

```
using (Network net = new Network())
{
    // use the net object in this scope
}
```

Hello, SMILE Wrapper!

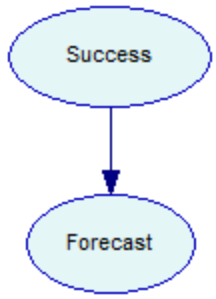
4 Hello, SMILE Wrapper!

In this section, we will show how SMILE can load and use a model created in GeNIe to perform useful work. We will use the model developed in the GeNIe on-line help (Section *Hello GeNIe!*). The model for this problem is available as one of the example networks (file `VentureBN.xdsl`). If you have GeNIe installed, you can copy the file into your working directory. The same file is also included in the zip file containing all source code for tutorials, available from http://support.bayesfusion.com/docs.

Alternatively, create a file named `VentureBN.xdsl` with any text editor by copying the content of the *VentureBN.xdsl* [16] section below.

4.1 Success/Forecast model

The model encodes information pertaining to a problem faced by a venture capitalist, who considers a risky investment in a startup company. A major source of uncertainty about her investment is the success of the company. She is aware of the fact that only around 20% of all start-up companies succeed. She can reduce this uncertainty somewhat by asking expert opinion. Her expert, however, is not perfect in his forecasts. Of all start-up companies that eventually succeed, he judges about 40% to be good prospects, 40% to be moderate prospects, and 20% to be poor prospects. Of all start-up companies that eventually fail, he judges about 10% to be good prospects, 30% to be moderate prospects, and 60% to be poor prospects.



Success	0.2
Failure	0.8

	Success	Failure
Good	0.4	0.1
Moderate	0.4	0.3
Poor	0.2	0.6

4.2 VentureBN.xdsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<smile version="1.0" id="VentureBN" numsamples="1000">
    <nodes>
        <cpt id="Success">
            <state id="Success" />
            <state id="Failure" />
            <probabilities>0.2 0.8</probabilities>
        </cpt>
        <cpt id="Forecast">
```

```
            <state id="Good" />
            <state id="Moderate" />
            <state id="Poor" />
            <parents>Success</parents>
            <probabilities>
                0.4 0.4 0.2 0.1 0.3 0.6
            </probabilities>
        </cpt>
    </nodes>
    <extensions>
        <genie version="1.0" app="GeNIe 2.1.1104.2"
            name="VentureBN"
            faultnameformat="nodestate">
            <node id="Success">
                <name>Success of the venture</name>
                <interior color="e5f6f7" />
                <outline color="0000bb" />
                <font color="000000" name="Arial" size="8" />
                <position>54 11 138 62</position>
            </node>
            <node id="Forecast">
                <name>Expert forecast</name>
                <interior color="e5f6f7" />
                <outline color="0000bb" />
                <font color="000000" name="Arial" size="8" />
                <position>63 105 130 155</position>
            </node>
        </genie>
    </extensions>
</smile>
```

4.3 The program

We will show how to load this model using SMILE, how to enter observations (evidence), how to perform inference, and how to retrieve the results of SMILE's calculations. Three complete source code versions (Java, Python, C#) are included below. Note that you'll need to use your SMILE license key. See the *Licensing* [10] section of this manual if you want to obtain your academic or trial license key. See *Platforms and Wrappers* [12] section for language-specific information on licensing.

Our network object - an instance of `Network` class is declared as local variable. We proceed to read the XDSL file from disk:

```
Java:   net.readFile("VentureBN.xdsl");
Python: net.read_file("VentureBN.xdsl")
C#:     net.ReadFile("VentureBN.xdsl");
```

We want to set the evidence on the *Forecast* node to *Moderate*. The wrappers can use node and outcome identifiers directly:

```
Java:   net.setEvidence("Forecast", "Moderate");
Python: net.set_evidence("Forecast", "Moderate")
C#:     net.SetEvidence("Forecast", "Moderate");
```

Now we can update the network:

```
Java:   net.updateBeliefs();
Python: net.update_beliefs()
C#:     net.UpdateBeliefs();
```

After network update we can retrieve the posterior probabilities of the *Success* node:

```
Java:   double[] beliefs = net.getNodeValue("Success");
Python: beliefs = net.get_node_value("Success")
C#:     double[] beliefs = net.GetNodeValue("Success");
```

To print the probabilities, we simply iterate over the elements of the returned numeric array. Each probability printed to the console is preceeded by the identifier of the node outcome. If you compile and run the program, the output should be:

```
Success=0.25
Failure=0.75
```

"Success" and "Failure" are outcome identifiers of the *Success* node, which in Java are returned by the `Network.getOutcomeId` method. At this point you should easily derive the equivalent method names for Python and C# from Java method name.

Note that in case of any of method calls failing, the exception would be thrown. SMILE wrappers do not use error codes on failure.

We will build upon the simple network described in this chapter in the *Tutorials*[68] section of this manual.

4.3.1 Hello.java

```
import smile.*;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Paste your license key below.");
        // new smile.License(...);

        Network net = new Network();
        net.readFile("VentureBN.xdsl");
        net.setEvidence("Forecast", "Moderate");
        net.updateBeliefs();
        double[] beliefs = net.getNodeValue("Success");
        for (int i = 0; i < beliefs.length; i ++) {
            System.out.println(
                net.getOutcomeId("Success", i) + " = " + beliefs[i]);
        }
    }
}
```

4.3.2 Hello.py

```
import pysmile
```

```
# pysmile_license is your license key
import pysmile_license

def hello_smile():
    net = pysmile.Network()
    net.read_file("VentureBN.xdsl");

    net.set_evidence("Forecast", "Moderate")
    net.update_beliefs()

    beliefs = net.get_node_value("Success")
    for i in range(0, len(beliefs)):
        print(net.get_outcome_id("Success", i) + "=" + str(beliefs[i]))

hello_smile()
```

4.3.3 Hello.cs

```
using System;
using Smile;

namespace SmileNetTutorial
{
    class Hello
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Paste your SMILE License Key here.");
            // new Smile.License(...);

            Network net = new Network();
            net.ReadFile("VentureBN.xdsl");
            net.SetEvidence("Forecast", "Moderate");
            net.UpdateBeliefs();
            double[] beliefs = net.GetNodeValue("Success");
            for (int i = 0; i < beliefs.Length; i++)
            {
                Console.WriteLine("{0} = {1}",
                    net.GetOutcomeId("Success", i), beliefs[i]);
            }
        }
    }
}
```

This page is intentionally left blank.

Using SMILE Wrappers

5 Using SMILE Wrappers

5.1 Error handling

SMILE Wrappers throw exceptions when the underlying C++ SMILE calls fail. The exception types are:

```
Java:   smile.SMILEException
Python: pysmile.SMILEExcetpion
C#:     Smile.SmileException
```

5.2 Networks, nodes and arcs

The most important class defined by SMILE Wrappers is `Network`. The objects of this class act as containers for nodes and are responsible for node creation and destruction. Nodes and arcs are always created and destroyed by invoking the methods of the `Network` class. The access to existing nodes and arcs is always performed through the `Network` object where they live.

There are no classes representing nodes and arcs. The `Network` object works as a facade for the complex underlying data structure.

5.2.1 Network

To create a network, simply use a default constructor:

Java:

```
import smile.*;
...
Network net = new Network();
```

Python:

```
import pysmile
...
net = pysmile.Network()
```

C#:

```
using Smile;
...
Network net = new Network();
```

See the *Platforms and Wrappers* [12] section for the platform-specific information about the lifetime of instances of `Network` class.

5.2.2 Nodes

Use the following methods to add and delete nodes:

Java:

```
Network.addNode
Network.deleteNode
```

Python:

```
Network.add_node
Network.delete_node
```

C#:

```
Network.AddNode
Network.DeleteNode
```

When creating the node you need to specify its type.

Within the network, the nodes are uniquely identified by their handle. The node handle is a non-negative integer, which is preserved when network is copied. In addition to handles, each node has an unique (in the context of its containing network), persistent, textual identifier. This identifier is specified as an argument to `Network.addNode` method at node creation (it may be changed later). The identifiers in SMILE start are case-sensitive, start with a letter and contain letters, digits and underscores. Node's identifier can be converted to node handle with a call to `Network.getNode` method.

All methods of the `Network` class dealing with nodes can use either integer node handles or string identifiers to specify the nodes. However, the versions using identifiers will be internally performing O(N) string lookup (where N is the number of nodes in the network), while the versions using handlers are only using O(1) validity check.

The values of the handles are not guaranteed to be consecutive or start from any particular value. To iterate over nodes in the network, use `Network.getFirstNode` and `getNextNode`:

Java:

```
for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
    // do something with net and node identified by h
}
```

Python:

```
h = net.get_first_node()
while (h >= 0):
```

```
    # do something with net and node identified by h
    h = net.get_next_node(h)
```

C#:

```
for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
{
    // do something with net and node identified by h
}
```

Note the loop exit condition - we proceed only if the returned handle is greater than or equal to zero.

Here are the methods to obtain an array of all node handles, an array of all node identifiers or a number of nodes in the network:

Java:

```
Network.getAllNodes
Network.getAllNodeIds
Network.getNodeCount
```

Python:

```
Network.get_all_nodes
Network.get_all_node_ids
Network.get_node_count
```

C#:

```
Network.GetAllNodes
Network.GetAllNodeIds
Network.NodeCount (read-only property)
```

Nodes may be marked as targets with `Network.setTarget` method. Target nodes are always guaranteed to be updated by the inference algorithm. Other nodes, i.e., nodes that are not designated as targets, may be updated or not, depending on the internals of the algorithm used, but are not guaranteed to be updated. Focusing inference on the target nodes can reduce time and memory required to complete the calculation. When no targets are specified, SMILE assumes that all nodes are of interest to the user.

5.2.3 Arcs

Here are the `Network` class methods to add and delete arcs between nodes:

Java:

```
Network.addArc
Network.deleteArc
```

Python:

```
Network.add_arc
```

```
Network.delete_arc
```

C#:

```
Network.AddArc
Network.DeleteArc
```

The graph defined by nodes and arcs in a `Network` class instance a directed acyclic graph (DAG) at all times. An attempt to add an arc which would create a cycle in the graph will cause an exception.

To inspect the graph structure, you can use the following methods:

Java:

```
Network.getParents
Network.getParentIds
Network.getChildren
Network.getChildIds
```

Python:

```
Network.get_parents
Network.get_parent_ids
Network.get_children
Network.get_child_ids
```

C#:

```
Network.GetParents
Network.GetParentIds
Network.GetChildren
Network.GetChildIds
```

The methods with the "Ids" suffix return the arrays of string identifiers of the related nodes, otherwise the arrays of integer node handles are used.

5.3 Anatomy of a node

The different aspects of a node available through `Network` class methods are:

- node definition
- node value
- node evidence
- attributes which do not affect inference, but determine node's location, color, name, etc.

The definition of the node specifies how it interacts with other nodes in the network. The node definition is written as part of the network when the network is saved to a file or serialized as a string variable. For general chance node the definition consists of conditional probability table (CPT) and list of state names.

The value of the node contains the values (typically, the marginal probability distribution or the expected utilities) calculated for the node by the inference algorithm. Unlike the definition, the value is not written as part of the network.

5.3.1 Node definition

The definition of the node specifies how it interacts with other nodes in the network. The node definition is written as part of the network when the network is saved to a file or serialized.

Use the following methods to read or write the numeric part of node definition:

Java:

```
Network.setNodeDefinition
Network.getNodeDefinition
```

Python:

```
Network.set_node_definition
Network.get_node_definition
```

C#:

```
Network.SetNodeDefinition
Network.GetNodeDefinition
```

The discrete nodes (including the general chance nodes) also have the list of outcomes as part of their definition. The outcomes are identified by string identifiers, which must be unique within the node and follow SMILE rules for identifier: are case-sensitive, start with a letter and contain letters, digits and underscores. Outcomes can be added, deleted or renamed.

5.3.2 Node value

The value of the node contains the values (typically, the marginal probability distribution or the expected utilities) calculated for the node by the inference algorithm. Unlike the definition, the value is not written as part of the network during I/O operations. In influence diagrams the node value is calculated for multiple sets of decisions. In such case, you also need to retrieve the indexing parents of node value to properly interpret the numbers. Note that value indexing parents are not the same as node parents. The value also stores its validity status; a flag set to true by the inference

algorithm after it succesfully completes the calculations for the given node. You should not call any value-reading methods unless the value is valid.

Here are the methods related to node values:

Java:

```
Network.isValueValid
Network.getNodeValue
Network.getValueIndexingParents
Network.getValueIndexingParentIds
```

Python:

```
Network.is_value_valid
Network.get_node_value
Network.get_value_indexing_parents
Network.get_value_indexing_parent_ids
```

C#:

```
Network.IsValueValid
Network.getNodeValue
Network.getValueIndexingParents
Network.getValueIndexingParentIds
```

5.3.3 Node evidence

The output from the inference algorithms depends on the node definitions and the evidence set on nodes in the network. The evidence for discrete nodes is specified by outcome index or outcome identifier. For continuous, the evidence is a number, and related methods have 'cont' in their names (for example, `Network.getContEvidence`). In dynamic Bayesian networks (DBNs), evidence is specified for a specified time slice. The evidence methods for DBNs have 'temporal' in their names (for example, `Network.getTemporalEvidence`).

Nodes can become propagated evidence (implied by other evidence set in the network).

Special type of evidence for discrete nodes is virtual evidence, which is a probability distribution over the outcomes of the node.

The methods related to evidence include:

Java:

```
Network.isEvidence
Network.isPropagatedEvidence
Network.isRealEvidence
Network.clearEvidence
Network.getEvidence
Network.getEvidenceId
```

```
Network.setEvidence
Network.getContEvidence
Network.setContEvidence
Network.isVirtualEvidence
Network.getVirtualEvidence
Network.setVirtualEvidence
Network.isTemporalEvidence
Network.hasTemporalEvidence
Network.clearTemporalEvidence
Network.getTemporalEvidence
Network.getTemporalEvidenceId
Network.getTemporalVirtualEvidence
Network.setTemporalEvidence
Network.setTemporalVirtualEvidence
```

Python:

```
Network.is_evidence
Network.is_propagated_evidence
Network.is_real_evidence
Network.clear_evidence
Network.get_evidence
Network.get_evidence_id
Network.set_evidence
Network.get_cont_evidence
Network.set_cont_evidence
Network.is_virtual_evidence
Network.get_virtual_evidence
Network.set_virtual_evidence
Network.is_temporal_evidence
Network.has_temporal_evidence
Network.clear_temporal_evidence
Network.get_temporal_evidence
Network.get_temporal_evidence_id
Network.get_temporal_virtual_evidence
Network.set_temporal_evidence
Network.set_temporal_virtual_evidence
```

C#:

```
Network.IsEvidence
Network.IsPropagatedEvidence
Network.IsRealEvidence
Network.ClearEvidence
Network.GetEvidence
Network.GetEvidenceId
Network.SetEvidence
Network.GetContEvidence
Network.SetContEvidence
Network.IsVirtualEvidence
Network.GetVirtualEvidence
Network.SetVirtualEvidence
Network.IsTemporalEvidence
Network.HasTemporalEvidence
Network.ClearTemporalEvidence
Network.GetTemporalEvidence
Network.GetTemporalEvidenceId
Network.GetTemporalVirtualEvidence
```

```
Network.SetTemporalEvidence
Network.SetTemporalVirtualEvidence
```

5.3.4 Other node attributes

Nodes have other attributes, which are not used by inference algorithms, but are required for graphical interpretation of the Bayesian network. Among these are: node name, description, position, color, etc. For brevity, the list below only includes the getter methods. The corresponding setters are also available.

Java:

```
Network.getNodeName
Network.getNodeDescription
Network.getNodePosition
Network.getNodeBgColor
Network.getNodeTextColor
Network.getNodeBorderColor
```

Python:

```
Network.get_node_name
Network.get_node_description
Network.get_node_position
Network.get_node_bgColor
Network.get_node_textColor
Network.get_node_borderColor
```

C#:

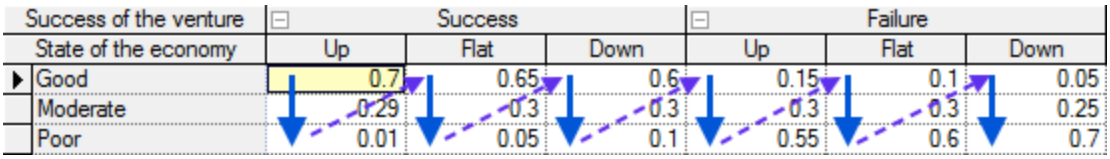
```
Network.GetNodeName
Network.GetNodeDescription
Network.GetNodePosition
Network.GetNodeBgColor
Network.GetNodeTextColor
Network.GetNodeBorderColor
```

5.4 Multidimensional arrays

Conditional probability table (CPT) describes the interaction between a node and its immediate predecessors. The number of dimensions and the total size of a conditional probability table are determined by the number of parents, the number of states of each of these parents, and the number of states of the child node. Essentially, there is a probability for every state of the child node for every combination of the states of the parents. Nodes that have no predecessors are specified by a prior probability distribution table, which specifies the prior probability of every state (outcome) of the node.

The conditional probability tables are stored as vectors of doubles that are a flattened version of multidimensional tables with as many dimensions as there are parents plus one for the node itself. The order of the coordinates reflects the order in which the arcs to the node were created. The most significant (leftmost) coordinate will represent the state of the first parent. The state of the node itself corresponds to the least significant (rightmost) coordinate.

The image below is an annotated screenshot of GeNIe's node properties window open for the *Forecast* node in the model created in Tutorial 1. *Forecast* has three outcomes and two parents: *Success of the venture* and *State of the economy*, with two and three outcomes, respectively. Therefore, the total size of the CPT is 2 x 3 x 3 = 18. The annotation arrows in the image (not part of the actual GeNIe window) show the ordering of the entries in the linear buffer used internally. The first (or rather, the zero-th, because all indexes in SMILE are zero-based) element, the one with the value of 0.7 and yellow background, represents *P(Forecast=Good | Success of the venture=Success & State of the economy=Up)*. It is followed by the probabilties for *Moderate* and *Poor* outcomes given the same parent configuration. The next parent configuration is *Success of the venture=Success* & *State of the economy=Flat*, and so on.



The node definition returned by the Network.getNodeDefinition is a single-dimensional array. If your program needs to convert between linear and multidimensional coordinates, see *Tutorial 3* [83]. The code in this tutorial includes the conversion function.

Note that multidimensional arrays are not used exclusively for CPTs. Other uses include expected utility tables and marginal probability distributions.

5.5 Input and Output

SMILE supports two types of network I/O: string-based and file-based.

The native format for SMILE networks is XDSL. The format is XML-based and the definition schema is available at BayesFusion documentation website (http://support.bayesfusion.com/docs/). When writing network in this format to file, the `.xdsl` extension should be used.

XDSL is the only format supported by string I/O methods. File-based methods can read and write other formats. Depending on the feature parity between SMILE and the 3rd party software using other file types, some of the information may be lost.

Java:

```
Network.readFile
Network.writeFile
```

```
Network.readString
Network.writeString
```

Python:

```
Network.read_file
Network.write_file
Network.read_string
Network.write_string
```

C#:

```
Network.ReadFile
Network.WriteFile
Network.ReadString
Network.WriteString
```

5.6 Inference

SMILE includes functions for several popular Bayesian network inference algorithms, including the clustering algorithm, and several approximate stochastic sampling algorithms. To run the inference , obtain the probability of evidence currently set in the network, or switch between various inference algorithm implementations, use the following methods:

Java:

```
Network.updateBeliefs
Network.probEvidence
Network.setBayesianAlgorithm
Network.getBayesianAlgorithm
Network.setInfluenceDiagramAlgorithm
Network.getInfluenceDiagramAlgorithm
```

Python:

```
Network.update_beliefs
Network.prob_evidence
Network.set_bayesian_algorithm
Network.get_bayesian_algorithm
Network.set_influence_diagram_algorithm
Network.get_influence_diagram_algorithm
```

C#:

```
Network.UpdateBeliefs
Network.ProbEvidence
Network.BayesianAlgorithm (read/write property)
Network.InfluenceDiagramAlgorithm (read/write property)
```

The default algorithm for discrete Bayesian Networks is clustering over network preprocessed with relevance. The output of this algorithm is exact (as opposed to various sampling algorithms also available in the library).

The sampling inference algorithms can be controlled by setting the number of generated samples with the `Network.setSampleCount` method. Obviously, the more samples are generated, the more time it takes to complete the inference.

`Network.updateBeliefs` throws an exception with error code -42 if the temporary data structures required to complete the inference take too much memory. In such case, or if the inference takes too long, consider taking advantage of SMILE's relevance reasoning layer. Relevance reasoning runs as a preprocessing step, which can lessen the complexity of later stages of inference algorithms. Relevance reasoning takes the target node set into account, therefore, to reduce the workload you should reduce the number of nodes set as targets if possible. Note that by default all nodes are targets (this is the case when no nodes were marked as such). If your network has 1,000 nodes and you only need the probabilities of 20 nodes, by all means call `Network.setTarget` on them.

If changing the model to use *Noisy-MAX*[35] nodes is possible, then it's definitely worth trying. The inference can be performed very efficiently on the networks with Noisy-MAX nodes when Noisy-MAX decomposition is enabled. To enable it, call `Network.setNoisyDecompEnabled`. If enabled, the Noisy-MAX decomposition runs in the relevance layer and reduces the complexity of the subsequent phases of the inference algorithm. To further control the decomposition you can call `Network.setNoisyDecompLimit`, which controls the maximal number of parents in the temporary structures managed by SMILE during inference.

5.7 User properties

To integrate data specific to your application with SMILE, you can use SMILE's user properties. The user properties are arrays of key/value pairs available at the network and node level. The user properties are stored in the XDSL files. Property name (key) is unique for the set of properties defined for given node or for a network. It also follows the convention of SMILE identifiers: is case-sensitive, starts with a letter and contains letters, digits and underscores. The property is represented by an `UserProperty` object, which has `name` and `value` fields (both are strings).

Java:

```
Network.getUserProperties
Network.setUserProperties
Network.getNodeUserProperties
Network.setNodeUserProperties
```

Python:

```
Network.get_user_properties
Network.set_user_properties
Network.get_node_user_properties
Network.set_node_user_properties
```

C#:

```
Network.GetUserProperties
Network.SetUserProperties
Network.GetNodeUserProperties
Network.SetNodeUserProperties
```

5.8 Submodels

For user interface purposes, like making complex network structure easier to understand and navigate, nodes can be placed in a submodel hierarchy. Each network contains at least one main submodel (which can't be deleted and is default submodel for new models to be placed in). Other submodels can be added and deleted. Submodels, except main submodel, have exactly one parent submodel, and can have multiple submodel children.

The assignment of nodes to submodels has no effect on inference.

Use the following methods to manage submodels:

Java:

```
Network.addSubmodel
Network.deleteSubmodel
Network.getSubmodelCount
Network.getSubmodel
Network.getMainSubmodel
Network.getMainSubmodelId
Network.getFirstSubmodel
Network.getNextSubmodel
Network.getSubmodelId
Network.setSubmodelId
Network.getSubmodelName
Network.setSubmodelName
Network.getSubmodelPosition
Network.setSubmodelPosition
Network.getSubmodelOfNode
Network.setSubmodelOfNode
Network.getSubmodelOfSubmodel
Network.setSubmodelOfSubmodel
```

Python:

```
Network.add_submodel
Network.delete_submodel
Network.get_submodel_count
Network.get_submodel
Network.get_main_submodel
Network.get_main_submodel_id
Network.get_first_submodel
Network.get_next_submodel
Network.get_submodel_id
Network.set_submodel_id
Network.get_submodel_name
Network.set_submodel_name
```

```
Network.get_submodel_Position
Network.set_submodel_Position
Network.get_submodel_of_node
Network.set_submodel_of_node
Network.get_submodel_of_submodel
Network.set_submodel_of_submodel
```

C#:

```
Network.AddSubmodel
Network.DeleteSubmodel
Network.GetSubmodelCount
Network.GetSubmodel
Network.GetMainSubmodel
Network.GetMainSubmodelId
Network.GetFirstSubmodel
Network.GetNextSubmodel
Network.GetSubmodelId
Network.SetSubmodelId
Network.GetSubmodelName
Network.SetSubmodelName
Network.GetSubmodelPosition
Network.SetSubmodelPosition
Network.GetSubmodelOfNode
Network.SetSubmodelOfNode
Network.GetSubmodelOfSubmodel
Network.SetSubmodelOfSubmodel
```

5.9 Canonical nodes

Canonical probabilistic nodes, such as Noisy-MAX/OR, Noisy-MIN/AND, and Noisy-Adder gates, implemented by SMILE, are convenient knowledge engineering tools widely used in practical applications. In case of a general CPT binary node with *n* binary parents, the user has to specify *2^n* parameters, a number that is exponential in the number of parents. This number can quickly become prohibitive: when the number of parents *n* is equal to 10, we need 1,024 parameters, when it is equal to 20, the number of parameters is equal to 1,048,576, with each additional parent doubling it. A Noisy-OR model allow for specifying this interaction with only *n+1* parameters, one for each parent plus one more number. This comes down to 11 and 21 for *n* equal to 10 and 20 respectively.

Canonical models are not only great tools for knowledge engineering - they also lead to significant reduction in computation through the independences that they model implicitly. Using canonical gates makes thus model construction easier but also leads to models that are easier to solve.

To create canonical nodes, pass `Network.NodeType.NOISY_MAX` or `Network.NodeType.NOISY_ADDER` to `Network.addNode`. Each node type has its own specific attributes accessible through `Network` class methods, but otherwise works in exactly the same way as a CPT node (has at least two outcomes, can be used as a parent or a child wherever the CPT node can, etc).

5.9.1 Noisy-MAX

Noisy-MAX is a generalization of the popular canonical gate Noisy-OR and is capable of modeling interactions among variables with multiple states. If all the nodes in question are binary, a Noisy-MAX node reduces to a Noisy-OR node. The Noisy-MAX, as implemented in SMILE, includes an equivalent of negation. By DeMorgan's laws, the OR function (or its generalization, the MAX function) along with a negation, is capable of expressing any logical relationship, including the AND (and its generalization, MIN). This means that SMILE's Noisy-MAX can be used to model the Noisy-AND/MIN functions, as well as other logical relationships.

SMILE's inference algorithm contains special code path for networks with Noisy-MAX nodes, which can speed up computations significantly. See the *Inference* [31] section of this manual for details.

To create a Noisy-MAX node, use the `NOISY_MAX` type with `Network.addNode`:

Java:

```
net.addNode(Network.NodeType.NOISY_MAX, "node1");
```

Python:

```
net.add_node(pysmile.NodeType.NOISY_MAX, "node1");
```

C#:

```
net.AddNode(Network.NodeType.NoisyMax, "node1");
```

Noisy-MAX node has discrete outcomes (just like CPT nodes). Noisy-MAX specific attributes are:

- an array of conditionally independent (CI) probabilities, which can be set and retrieved by `Network.setNodeDefinition` and `getNodeDefinition`.
- for each of node's parents, an array of parent outcome strengths. Parent outcome strengths enable control of the order of states of the parent nodes, as they enter their relation with the child. A Noisy-MAX CI table always follows the order of strengths. Use `Network.setNoisyParentStrengths` and `getNoisyParentStrengths` to modify or read the parent strengths.

As an example, consider a binary Noisy-MAX node with two parents, each with three outcomes. The following snippet modifies the probabilities and outcome strengths for the second parent (with zero-based index 1).

Java:

```
double[] ci = net.getNodeDefiniton(h);
int BASE = 2 * 3;
ci[BASE]     = 0.1;
ci[BASE + 1] = 0.9;
ci[BASE + 2] = 0.3;
ci[BASE + 3] = 0.7;
```

```
net.setNodeDefiniton(h, ci);
net.setNoisyParentStrengths(h, 1, new int[] { 2, 0, 1 });
```

Python:

```
ci = net.get_node_definition(h)
BASE = 2 * 3
ci[BASE]     = 0.1
ci[BASE + 1] = 0.9
ci[BASE + 2] = 0.3
ci[BASE + 3] = 0.7
net.set_node_definition(h, ci)
net.set_noisy_parent_strengths(h, 1, [2, 0, 1])
```

C#:

```
double[] ci = net.GetNodeDefinition(h);
int BASE = 2 * 3;
ci[BASE]     = 0.1;
ci[BASE + 1] = 0.9;
ci[BASE + 2] = 0.3;
ci[BASE + 3] = 0.7;
net.SetNoisyParentStrengths(h, 1, new int[] { 2, 0, 1 });
```

The value of `BASE` is calculated as a product of node outcome count and preceding parents' outcome counts (in this case, there is just one preceding parent with three outcomes). The probabilities for the parent are written into an array returned by `Network.getNodeDefiniton` and used in a `setNodeDefinition` call, because we just want to modify one parents' probabilities and leave other CI probabilities unchanged. The next step changes the order of parents' outcomes in relationship to the Noisy-MAX node with `Network.setParentOutcomeStrengths` call, the first column of probabilities for parent with index 1 (the one with 0.1 and 0.9) represents the probabilities for the parent outcome with index 2 (because 2 is the first element in the `strengths` array). Note that this does not modify the parent node in any way and the ordering is valid only in the context of this particular parent-child relationship. Assuming that we started with the default uniform probabilities in the table, our modifications yields the following Noisy-Max definition, as viewed in GeNIe:

Parent	p1			p2			LEAK
State	State0	State1	State2	State2	State0	State1	
State0	0.5	0.5	0	0.1	0.3	0	0.5
State1	0.5	0.5	1	0.9	0.7	1	0.5

The outcomes of both parent are *{State0, State1, State2}*. However, by using `setParentOutcomeStrenghts` for parent *p2*, its outcomes are seen by the Noisy-MAX child node as *{State2, State0, State1}*. Other Noisy-MAX nodes in the same network can set up their own parent outcome ordering if *p2* becomes their parent.

5.9.2 Noisy-Adder

Noisy-Adder nodes are not fully supported yet in the wrappers. Contact us if you need to use them; we can provide pre-release version of an SMILE wrapper compatible with your programming language and operating system.

5.10 Influence diagrams

Influence diagrams use two additional node types next to chance (CPT and canonical) and deterministic nodes:

- Decision nodes represent variables that are under control of the decision maker and model available decision alternatives, modeled explicitly as possible states of the decision node. They have no numerical parameters, only a discrete set of outcomes. Decision nodes can be children of decision and chance nodes. The node type identifier passed to `Network.addNode` is `Network.NodeType.DECISION`.

- Value nodes, i.e., a measure of desirability of the outcomes of the decision process. They are quantified by the utility of each of the possible combinations of outcomes of the parent nodes. Value nodes can be children of decision and chance nodes. Pass `Network.NodeType.UTILITY` to `Network.addNode` to create a value node.

- Multi-attribute utility (MAU) nodes, which combine value nodes to form a multi-attribute utility function. The function can be specified as a set of weights of a linear function (in such case, the node becomes an additive linear utility, ALU) or any expression that refers to identifiers of the value node parents. See the *Equations reference*[46] chapter for a list of available functions. MAU nodes can be children of decision, value, and other MAU nodes. If decision parents exist, the definition of the MAU node contains a separate set of weights or expressions for each combination of decision parents. Use `Network.NodeType.MAU` with `Network.addNode` to create a MAU node. By default, the MAU node is defined by weights. To use expressions, call `Network.setMauExpressions`.

As is the case with Bayesian networks, the values calculated by influence diagram inference algorithms are stored in node values and can be accessed through `Network.getNodeValue`. However, the interpretation of the numbers returned by `getNodeValue` is extended. The matrices are indexed by the set of nodes called indexing parents. These are are unobserved decision nodes that precede the current node or unobserved chance nodes that are predecessors of decision nodes and should have been observed before the decisions can be made. Call `Network.getIndexingParents` to retrieve indexing parent handles, or `Network.getIndexingParentIds` to get indexing parent identifiers. The set of outcomes of indexing parents is called a policy. After a successful inference in an influence diagram, node values are:

- for chance and deterministic nodes: posterior probabilities for each policy
- for decision nodes: expected utilities for all outcomes and for each policy
- for value and MAU nodes: expected utility for each policy

See *Tutorial 4* [91] for a simple influence diagram demo program.

5.11 Dynamic Bayesian networks

A Bayesian network is a snapshot of the system at a given time and is used to model systems that are in some kind of equilibrium state. Unfortunately, most systems in the world change over time and sometimes we are interested in how these systems evolve over time more than we are interested in their equilibrium states. Whenever the focus of our reasoning is change of a system over time, we need a tool that is capable of modeling dynamic systems.

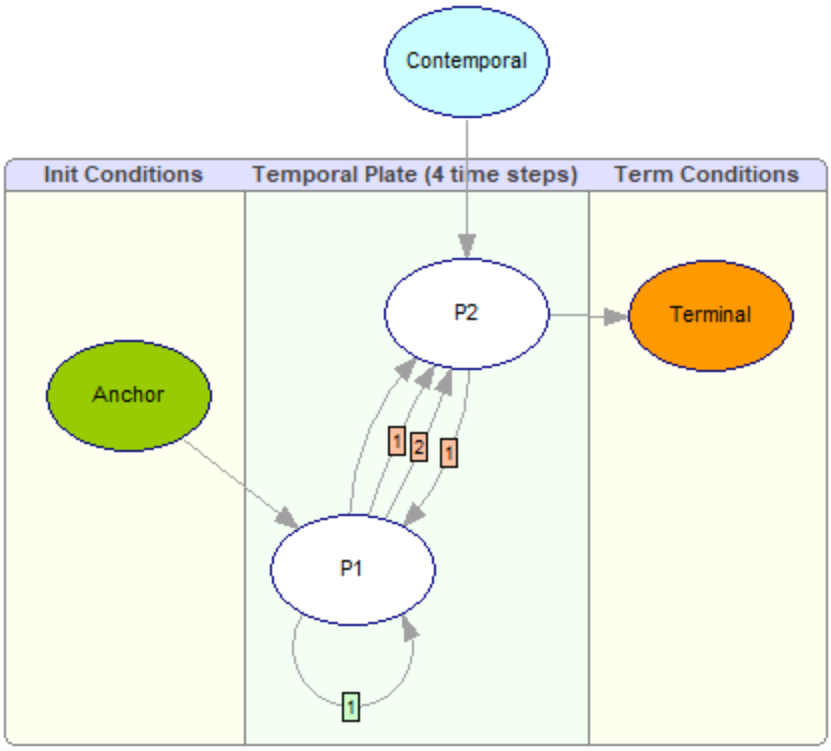
A dynamic Bayesian network (DBN) is a Bayesian network extended with additional mechanisms that are capable of modeling influences over time (Murphy, 2002). The temporal extension of BNs does not mean that the network structure or parameters changes dynamically, but that a dynamic system is modeled. In other words, the underlying process, modeled by a DBN, is stationary. A DBN is a model of a stochastic process.

The implementation of DBNs in SMILE allows for use both chance (CPT and canonical) and deterministic node types in dynamic models.

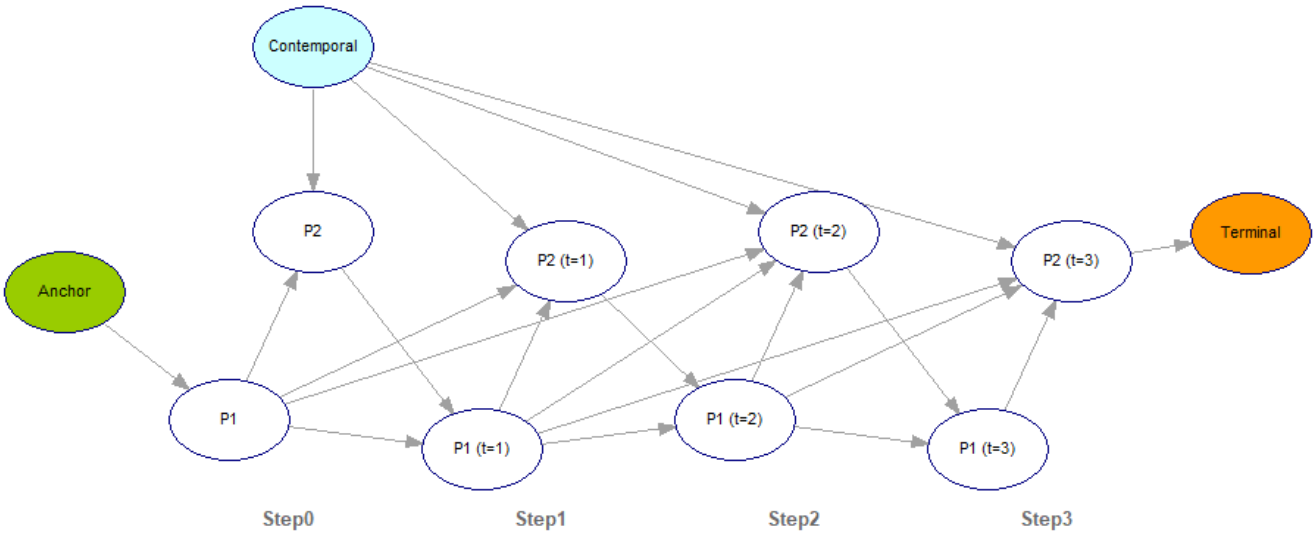
Tutorial 6 [103] contains a complete program demonstrating the use of DBNs.

5.11.1 Unrolling

SMILE's inference algorithm for dynamic Bayesian networks converts DBNs (unrolls them over time) to temporary static networks and then solves these networks. Results of this inference are copied back into the node values in the original DBNs. The structure of the following DBN (in GeNIe format) does not represent any real system, it is created just for demonstration purposes.



Different colors of the nodes represent their location relative to the abstract "temporal plate." By default, the nodes in the network are set to be `Network.NodeTemporalType.CONTEMPORAL` and is located outside of the plate. By using `Network.setNodeTemporalType` you can change their temporal type assignment. In the example model above, the plate nodes (`Network.NodeTemporalType.PLATE`) are white, the anchor node (`Network.NodeTemporalType.ANCHOR`) is green, the terminal node (`Network.NodeTemporalType.TERMINAL`) is orange and the blue node is contemporal. For simplicity, we use only single anchor, terminal, and contemporal node. Note the presence of multiple arcs between two plate nodes and the arc linking *P1* with itself. Some of the arcs between plate nodes have a tag with a number; these are *temporal arcs*, which are created by `Network.addTemporalArc`. The number on the tag is a *temporal order* of an arc. The arcs without the tag were added by `Network.addArc`. The result of unrolling the above DBN (this is performed automatically by the DBN inference algorithm; it is possible to create such network with `Network.unroll`) is the following:



To reduce the size of the unrolled network, we changed the number of time steps (slices) from the default value of 10 to 4 with a call to `Network.setSliceCount`. The colors of the original nodes were carried over to the unrolled model, which is extended by creating new copies of the plate nodes. The structure of this network shows how the temporal arcs are used to express the conditional dependency of plate nodes in time step *i* on the plate nodes in time step *j*, where *i* > *j*.

Consider the (temporal) arc between *P1* and *P2* with temporal order 2. It was copied twice to link *P1* with *P2 (t=2)*, and *P1 (t=1)* with *P2 (t=3)*. Note that there is no copy of this arc starting from *P1 (t=2)*, because its child would be in *t=2+2=4*, and (zero-based) time stops at 3 in our example. On the other hand, all three temporal arcs with temporal order 1 are copied three times. The relationship represented by an arc linking *P1* with itself in the DBN is clearly visible between *P1* and *P1 (t=1)*, *P1 (t=1)* and *P1 (t=2)* and *P1 (t=2)* and *P1 (t=3)*.

Note the difference between the arcs originating in *Contemporal* and *Anchor* nodes. The *Anchor* node has children only in the initial slice, while the *Contemporal* node is linked to all time slices. The *Terminal* node has parents only in the last slice.

To ensure that unrolled network remains a directed acyclic graph, the following arcs are forbidden in DBNs:

- from plate nodes to normal and anchor nodes
- from terminal nodes to non-terminal nodes

Unrolling is performed automatically during inference. For debug/explanatory purposes it is also possible to obtain an unrolled network by `Network.unroll`. The unrolled network created this way is an independent model, which means that changes made to the DBN after `unroll` call are not propagated into the unrolled network.

5.11.2 Temporal definitions

Consider node *P2* from the example in the previous section. It has four incoming arcs:

- normal arc from *Contemporal*
- normal arc from *P1*
- two temporal arcs from *P1* with temporal orders 1 and 2

However, in the unrolled network's slice for *t=0*, *P2* has only two incoming arcs. This is because there are no slices representing timepoints before *t=0*. *P2* in the slice for *t=1* has three incoming arcs, because it's now possible to link *P1* at *t=0* with *P2* at *t=1*. Finally, starting with slice for *t=2*, *P2* has four incoming arcs. The structure of the unrolled network requires *P2* to have separate CPTs for *t=0*, *t=1* and *t>=2*. Generally speaking, if a plate node has an incoming arc of order *x*, it will require *x+1* separate definitions. To get and set the temporal definitions, use `Network.getNodeTemporalDefinition` and `setNodeTemporalDefinition`.

All parents for the temporal definition with a specified temporal order can be retrieved by calling `Network.getUnrolledParents`. Note that `Network.getTemporalParents` called for the same temporal order returns only a subset of parents indexing this temporal definition. This is caused by unrolling: in the unrolled network node *P2* has four incoming arcs in slices for *t>=2*, but only two of these are actually arcs with temporal order 2.

5.11.3 Temporal evidence

To specify evidence for the plate nodes in the DBN, use `Network.setTemporalEvidence` and `setTemporalVirtualEvidence`. The code snippet below sets the temporal evidence in slices 5 and 7. Node represented by `evidenceNodeHandle` is assumed to have two outcomes.

Java:

```
net.setTemporalEvidence(evidenceNodeHandle, 5, 1);
net.setTemporalVirtualEvidence(evidenceNodeHandle, 7,
    new double[] { 0.4, 0.6 });
```

Python:

```
net.set_temporal_evidence(evidenceNodeHandle, 5, 1);
net.set_temporal_virtual_evidence(evidenceNodeHandle, 7, [0.4,0.6])
```

C#:

```
net.SetTemporalEvidence(evidenceNodeHandle, 5, 1);
```

```
net.SetTemporalVirtualEvidence(evidenceNodeHandle, 7,
    new double[] { 0.4, 0.6 });
```

To retrieve the temporal evidence, use `Network.getTemporalEvidence` and `Network.getTemporalVirtualEvidence`.

Other useful methods are `Network.hasTemporalEvidence` and `isTemporalEvidence`, which check whether a node has any temporal evidence or temporal evidence in specified temporal order, respectively.

5.12 Continuous models

Graphical models, such as Bayesian networks, are not necessarily consisting of only discrete variables. They are, in fact, close relatives of systems of simultaneous structural equations. SMILE allows for constructing models consisting of equation nodes that are alternative, graphical representations of systems of systems of simultaneous structural equations.

Tutorial 7 [113] contains a complete program demonstrating the use of continuous models.

5.12.1 Equation-based nodes

To create an equation node, use `Network.NodeType.EQUATION` node type when creating a node with `Network.addNode`. The node equation will be initialized to *id=0*, where *id* is the node identifier passed to `addNode`. The code snippet below changes the default equation of the freshly created node (*x=0*) to an equation representing the standard Gaussian distribution: *x=Normal(0, 1)*.

Java:

```
int hx = net.addNode(Network.NodeType.EQUATION, "x");
net.setNodeEquation(hx, "x=Normal(0,1)");
```

Python:

```
hx = net.add_node(pysmile.NodeType.EQUATION, "x")
net.set_node_equation(hx, "x=Normal(0,1)")
```

C#:

```
int hx = net.AddNode(Network.NodeType.Equation, "x");
net.SetNodeEquation(hx, "x=Normal(0,1)");
```

SMILE defines many functions for use in node equations. The complete list of functions is available in the *Equations reference* [46] section. Among these functions, there are random generators, which

generate a single sample based on the passed parameters. In the example above, the `Normal` is the name of SMILE's random generator function.

Node equations can reference other nodes by using their identifiers. Continuing with our code snippet:

Java:

```
int hy = net.addNode(Network.NodeType.EQUATION, "y");
net.setNodeEquation(hx, "y=2*x");
```

Python:

```
hy = net.add_node(pysmile.NodeType.EQUATION, "y")
net.set_node_equation(hy, "y=2*x")
```

C#:

```
int hy = net.AddNode(Network.NodeType.Equation, "y");
net.SetNodeEquation(hx, "y=2*x");
```

Second node is added and its equation is set to *y=2*x*, where *x* is a reference to previously defined node. SMILE **adds an arc** between x and y automatically and it is not necessary to call `Network.addArc` before setting the equation referencing other nodes. Calling `addArc` will change the child node equation by adding a term representing the parent node as a last term on the right hand side of the child equation. Assuming network with equation nodes a, b, c and d and d's equation set to *d=Normal(a, 1)+Normal(b, 2)*, calling `AddArc` to with node handles of c and d would rewrite d's equation to *d=Normal(a, 1)+Normal(b, 2)+c*.

If an arc is removed, either by calling `Network.deleteArc` or `Network.deleteNode` on one of the parents, the node equation will be rewritten as sum of the remaining parents. This ensures that equations and arcs are always in sync. If node b would be removed with `deleteNode`, or an arc from b to d would be removed by `deleteArc`, d's equation would become *d=a+c*.

Node identifier changes are propagated into the equations. If the identifier of the first node from the code snippet above was changed from *x* to *x_prime*, the equation of node *y* would change to *y=2*x_prime*.

5.12.2 Continuous inference

To run the inference in continuous model, use `Network.updateBeliefs`; the same method which is used in discrete networks.

To set evidence in an equation node, use `Network.setContEvidence`. In the snippet below we are assuming that `evidenceNodeHandle` is the handle of the equation node.

Java:

```
net.setContEvidence(evidenceNodeHandle, 1.5);
```

Python:

```
net.set_cont_evidence(evidenceNodeHandle, 1.5)
```

C#:

```
net.SetContEvidence(evidenceNodeHandle, 1.5);
```

To retrieve evidence from continuous node, use `Network.getContEvidence`.

The inference in continuous networks is based on stochastic sampling when there is no evidence in the network, or the evidence is specified only for nodes without parents. Otherwise, the inference is performed on a temporary discrete network, derived from the original continuous model. The definitions for the temporary discrete nodes are derived from the discretization intervals defined in each continuous node. In addition to `Network.setSampleCount`, stochastic sampling can be controlled at the network level by setting the discretization sample count with `Network.setDiscretizationSampleCount`. Large number of samples provides better approximation of the solution to the set of equations, which continuous network represents, but requires more time to complete.

Both types of inference algorithm use the lower and upper bounds defined for each equation node. To set the bounds, use `Network.setNodeEquationBounds` method. Stochastic sampling can reject a sample when its value falls outside of the bounds defined for the node. You can control this behavior by `Network.setOutlierRejectionEnabled`. By default, outlier rejection is disabled.

To set node's discretization intervals, use `Network.setNodeEquationDiscretization`. The method accepts an array of `DiscretizationInterval` objects intervals as its argument. Each interval is defined by its optional identifier (not used for inference) and an upper bound. The lower bound of the interval with index j is defined by upper interval of the interval with index j-1. The lower bound of the first interval is defined by the lower bound defined for the node with a `Network.setNodeEquationBounds` call.

Discretization intervals are used to obtain CPTs for the temporary discrete network. The CPTs are calculated by drawing a number of samples specified at the network level for each CPT column. The size of the discretized CPT is a product of the number of node intervals and parent node intervals. Note that this may lead to excessive memory use when the node has many parents.

The results of the inference in a continuous model can be either samples or discretized beliefs, depending on the inference algorithm:

- sampling algorithm outputs samples for each node. `Network.getNodeValue` will return an array of samples and `Network.getNodeSampleStats` returns an array with simple statistics for the sample set (mean, standard deviation, minimum and maximum. `Network.isValueDiscretized` returns false.

- discretizing algorithm outputs probability distribution over discretization intervals. `Network.getNodeValue` returns this distribution. `Network.isValueDiscretized` returns true.

5.12.3 Temporal beliefs

For plate nodes in the DBN, the inference algorithm calculates the temporal beliefs, which are marginal posterior probability distributions as a function of time. Temporal beliefs are returned by `Network.getNodeValue`.

The dependency of the beliefs on time makes the temporal beliefs array larger. If a node has *X* outcomes and the slice count was set to *Y*, the matrix will have *X * Y* elements. The elements representing a single time slice are adjacent. Therefore, elements with indices *[0..X-1]* in the matrix are the beliefs for *t=0*, elements *[X..2*X-1]* are the beliefs for *t=1* and so on.

Only the plate nodes have the temporal beliefs. Other temporal node types have normal beliefs (the number of elements in the belief array is equal to their outcome count).

5.13 Hybrid models

Hybrid models are networks with both discrete and continuous nodes. The arcs in a hybrid network can link all combinations of node types, so it is possible to add an arc from a continuous to a discrete node, and from a discrete to a continuous node. Iinference in hybrid models follows the rules defined for continuous nodes (sampling when there is no evidence in nodes with parents, discretization otherwise). Basically, hybrid models can be treated as continuous models with discrete nodes representing the specialized function, namely conditional probability table specified by an array of numbers.

Adding arcs from discrete to continuous nodes is performed by including discrete node identifier in the continuous node equation (as is the case for the continuous to continuous arcs). The following discussion assumes that reader is familiar with the functions from which the node equations are built, described in detail in the *Equations reference* [46] section of the reference part of this manual.

For example, assuming that a node *c* is continuous and a node *d* is discrete, the equation for *c* may look like this: *c=Normal(If(d=1, 1, -1), 5)*. When a sample for node *c* is evaluated, one of its inputs will be the value of its parent node *d*. Discrete node values are numbers drawn from the interval *[0 .. N-1]*, where *N* is the number of discrete node outcomes. Therefore, the example equation for *c* above says that c should be drawn from a normal distribution with standard deviation equal to 5, but with mean depending on the parent node *d*. If *d* is in its (zero-based) state with index 1, the mean will be 1 and -1 otherwise.

To improve the readability of the equation, the outcomes of the parent discrete nodes can also be represented as text literals. If *d* has three outcomes *High*, *Medium* and *Low*, then the equation from the preceding example could be rewritten as *c=Normal(If(d="Medium"), 1, -1), 5)*.

In addition to the function *If* or its counterpart, the ternary operator *?:*, the common functions to use with discrete nodes are *Switch* and *Choose*. For example, the equation for node *c* with three possible means of its normal distributions can look like this: *c=Normal(Switch(d, "High", 3.2, "Medium", 2.5, "Low", 1.4), 5)*. An alternative notation would be *c=Normal(Choose(d, 3.2, 2.5, 1.4), 5)*.

Important: SMILE will not modify the text literals representing the outcomes of discrete parent nodes if the outcome identifiers change. If the text literal cannot be associated with any parent node outcome, it's value is evaluated as -1 (minus one).

Generally speaking, discrete nodes can appear anywhere in the equation where numbers can be used: *c=log(1+d)* or *c=2^d*. This kind of equation does not use the text literals representing the discrete node outcomes (because there is no comparison involved in evaluation the equation).

To add an arc from a continuous to a discrete node, use `Network.addArc` (the method used in discrete models). In such case, the discretization intervals of the parent node are considered to be the equivalent of the outcomes of discrete parent.

Tutorial 8[122] contains a complete program demonstrating the use of hybrid models.

5.14 Equations reference

The subsections of this chapter describe the notation used when specifying the definition for equation nodes and expression-based MAU nodes.

5.14.1 Operators

Arithmetic operators

+	addition, e.g., if x=3 and y=2, x+y produces 5
-	subtraction and unary minus, e.g., if x=3 and y=2, x-y produces 1 and -x produces -3
^	exponentiation (a^b means a^b), e.g., if x=3 and y=2, x^y produces 9
*	multiplication, e.g., if x=3 and y=2, x*y produces 6
/	division, e.g., if x=3 and y=2, x/y produces 1.5

Comparison operators

> greater than, e.g., if x=3 and y=2, x>y produces 1

< smaller than, e.g., if x=3 and y=2, x<y produces 0

>= greater or equal than, e.g., if x=3 and y=2, x>=y produces 1

<= smaller or equal than, e.g., if x=3 and y=2, x<=y produces 0

<> not equal, e.g., if x=3 and y=2, x<>y produces 1

= equal, e.g., if x=3 and y=2, x=y produces 0

Conditional selection operator

? : ternary conditional operator like in C, C++ or Java programming languages.

This operator is essentially a shortcut to the `If` function. For example, `a=b?5:3` is equivalent to `If(a=b, 5, 3)`.

Order of calculation, operator precedence, and parentheses

Expressions are evaluated from left to right, according to the precedence order specified below (1 denotes the highest precedence).

Precedence order	Operator
1	**-** (unary minus)
2	**^** (exponentiation)
3	***** and **/** (multiplicative operators)
4	**+** and **-** (additive operators)
5	**>**, **<**, **>=**, **<=**, **=** (comparison operators)
6	**?:** (conditional selection)

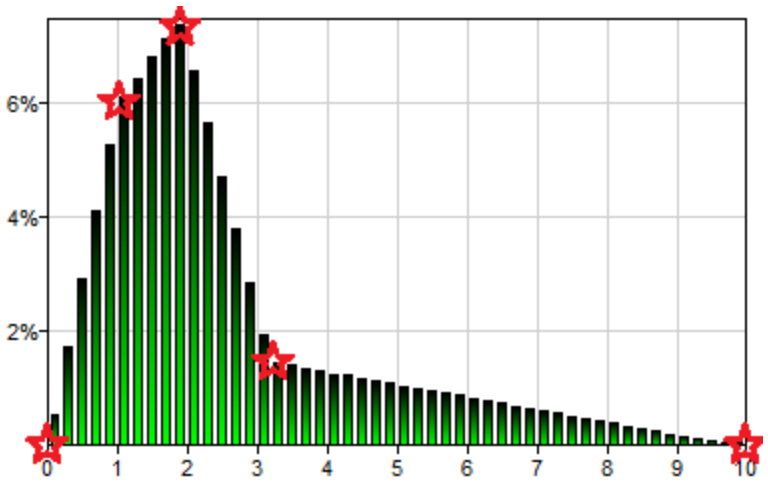
To change the order of calculation, enclose in parentheses those parts of the formula that should be calculated first. This can be done recursively, i.e., parentheses can be nested indefinitely. For example, if x=3 and y=2, 2*y+x/3-y+1 produces 4, 2*(y+x)/(3-y)+1 produces 11, and 2*((y+x)/(3-y)+1) produces 12.

5.14.2 Random Number Generators

Random generators **each generate a single sample** from the distributions defined below. In most equations, they can be imagined as random noise that distort the equation. Because the fundamental algorithm for inference in continuous and hybrid models is stochastic simulation, it is possible to visualize what probability distributions these single errors result in for each of the variables in the model. Probability distributions are not allowed in expression-based MAU nodes.

CustomPDF(x1,x2,...y1,y2,...)

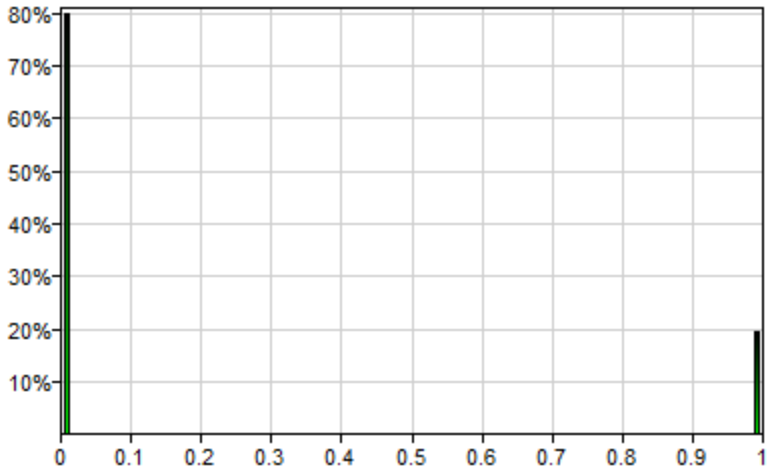
The `CustomPDF` distribution allows for specifying a non-parametric continuous probability distribution by means of a series of points on its probability density (PDF) function. Pairs (x_i,y_i) are coordinates of such points. The total number of parameters of `CustomPDF` function should thus to be even. Please note that x coordinates should be listed in increasing order. The PDF function specified does not need to be normalized, i.e., the area under the curve does not need to add up to 1.0. For example, `CustomPDF(0,1.02,1.9,3.2,10,0,4,5,1,0)` generates a single sample from the following distribution:



Stars on the plot mark the points defined by the `CustomPDF` arguments, i.e., (0, 0), (1.02, 4), (1.9, 5), (3.2, 1), and (10, 0).

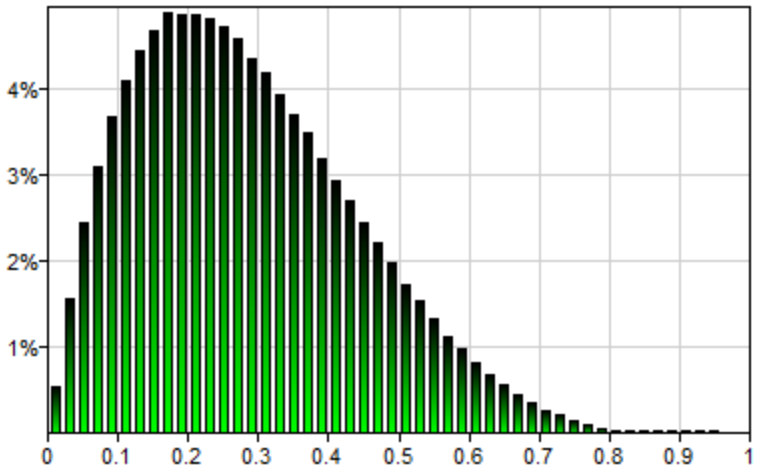
Bernoulli(p)

Bernoulli is a discrete distribution that generates 0 with probability 1-*p* and 1 with probability *p*. `Bernoulli(0.2)` will generate a single sample (0 or 1) from the following distribution, i.e., 1 with probability 0.2 and 0 with probability 0.8:



Beta(a,b)

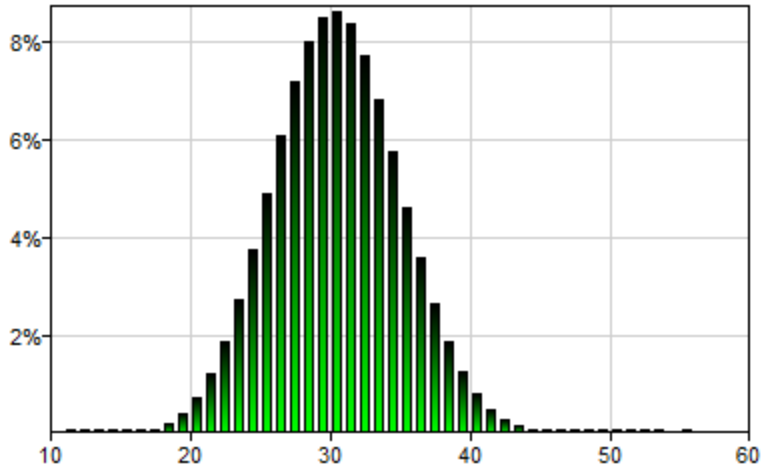
The Beta distribution is a family of continuous probability distributions defined on the interval [0, 1] and parametrized by two positive shape parameters, *a* and *b* (typically denoted by α and β), that control the shape of the distribution. `Beta(2,5)` will generate a single sample from the following distribution:



Binomial(n,p)

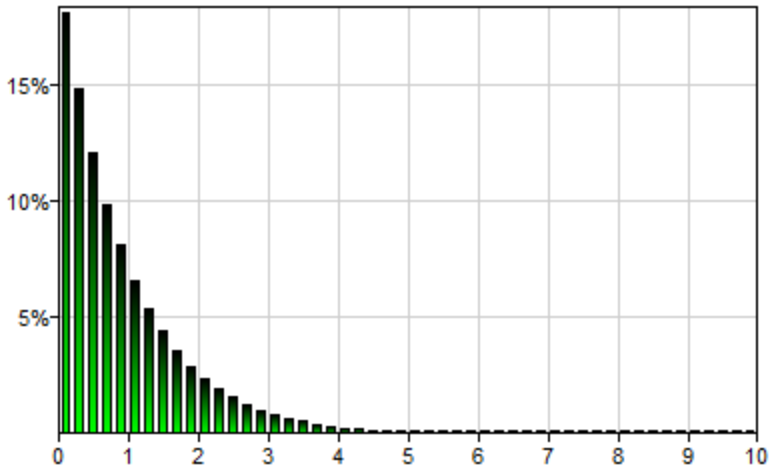
Binomial is a discrete probability distribution over the number of successes in a sequence of *n* independent trials, each of which yields a success with probability *p*. It will generate a single sample, which will be an integer number between 0 and *n*. A success/failure experiment is also

called a Bernoulli trial. Hence, `Binomial(1,p)` is equivalent to `Bernoulli(p)`. `Binomial(100,0.3)` will generate a single sample from the following distribution:



Exponential(lambda)

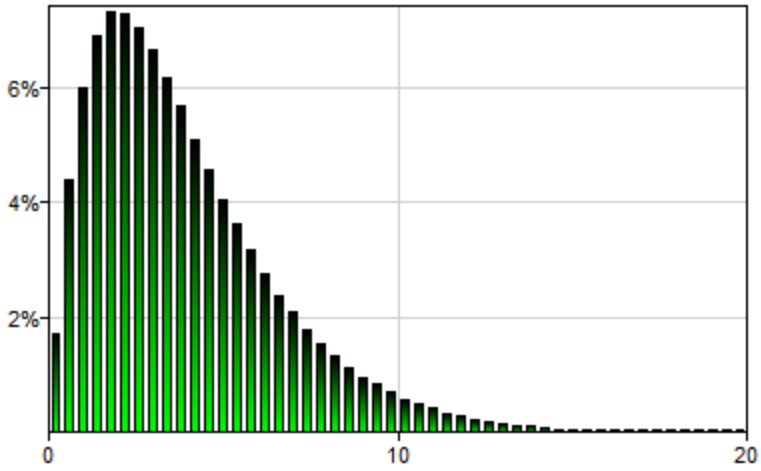
The exponential distribution is a continuous probability distribution that describes the time between events in a Poisson process, i.e., a process in which events occur continuously and independently at a constant average rate. Its only real-valued, positive parameter *lambda* (typically denoted by λ) determines the shape of the distribution. It is a special case of the Gamma distribution. `Exponential(lambda)` generates a single sample from the domain (0,∞). `Exponential(1)` will generate a single sample from the following distribution:



Gamma(shape,scale)

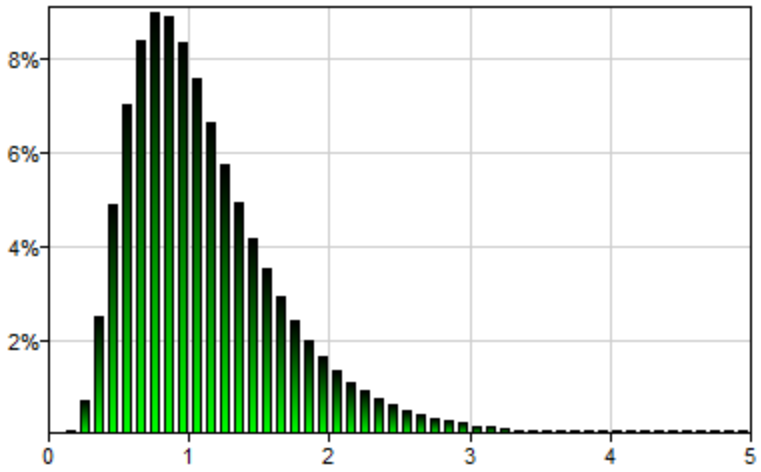
The Gamma distribution is a two-parameter family of continuous probability distributions. There are different parametrizations of the Gamma distribution in common use. SMILE parametrization follows one of the most popular parametrizations, with *shape* (often denoted by *k*) and *scale* (often

denoted by θ) parameters, both positive real numbers. `Gamma(2.0,2.0)` will generate a single sample from the following distribution:



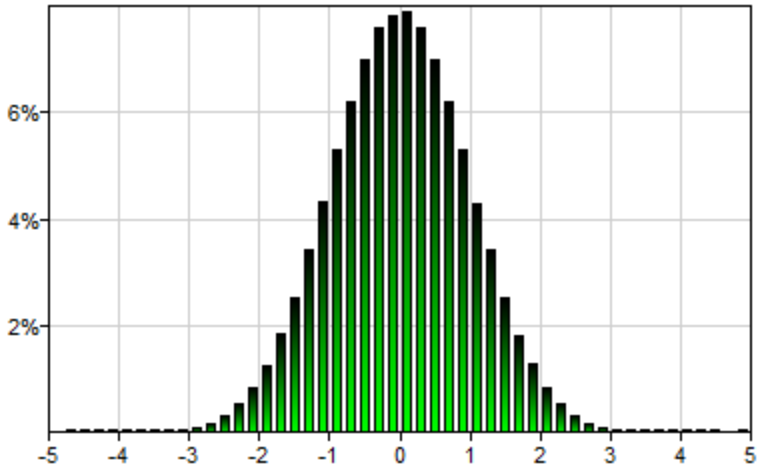
Lognormal(mu,sigma)

The lognormal distribution is a continuous probability distribution of a random variable, whose logarithm is normally distributed. Thus, if a random variable X is lognormally distributed, then a variable Y=Ln(X) has a normal distribution. Conversely, if Y has a normal distribution, then $X=e^Y$ has a lognormal distribution. A random variable which is lognormally distributed takes only positive values. `Lognormal(0,0.5)` will generate a single sample from the following distribution:



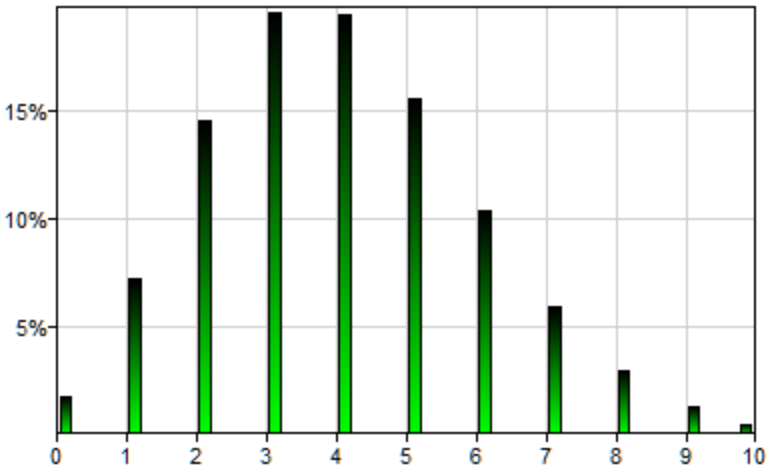
Normal(mu,sigma)

Normal (also known as Gaussian) distribution is the most commonly occurring continuous probability distribution. It is symmetric and defined over the real domain. Its two parameters, *mu* (mean, μ) and *sigma* (standard deviation, σ), control the position of its mode and its spread respectively. `Normal(0,1)` will generate a single sample from the following distribution:



Poisson(lambda)

Poisson distribution is a discrete probability distribution typically used to express the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant rate and independently of the time since the last event. Its only parameter, lambda, is the expected number of occurrences (which does not need to be integer). Poisson(4) will generate a single sample from the following distribution:

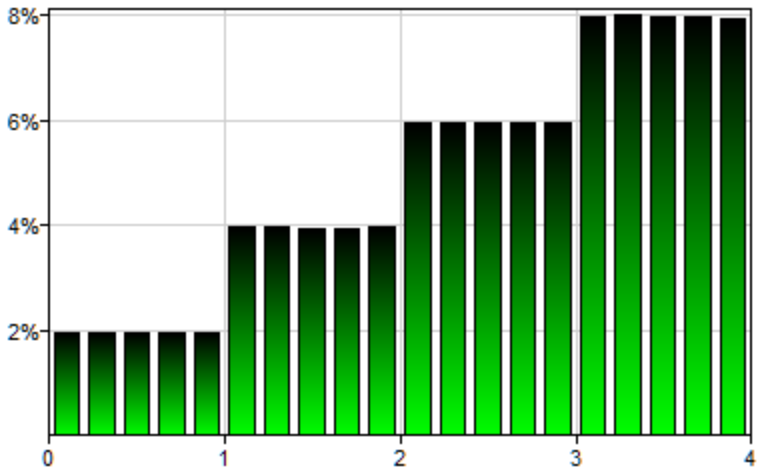


Steps(x1,x2,...y1,y2,...)

The Steps() distribution allows for specifying a non-parametric continuous probability distribution by means of a series of steps on its probability density (PDF) function. It is similar to the CustomPDF() function, although it does not specify the inflection points but rather intervals and the height of a step-wise probability distribution in each of the intervals. Because the number of interval borders is always one more than the number of intervals between them, the total number of parameters of Steps() function should be odd. Please note that x coordinates should be listed in

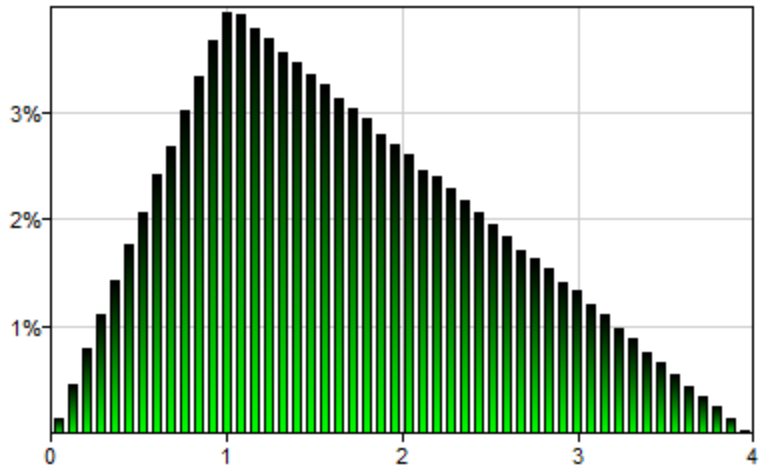
increasing order. The PDF function specified does not need to be normalized, i.e., the area under the curve does not need to add up to 1.0.

Example: Steps(0,1,2,3,4,1,2,3,4) generates a single sample from the following distribution::



Triangular(min,mod,max)

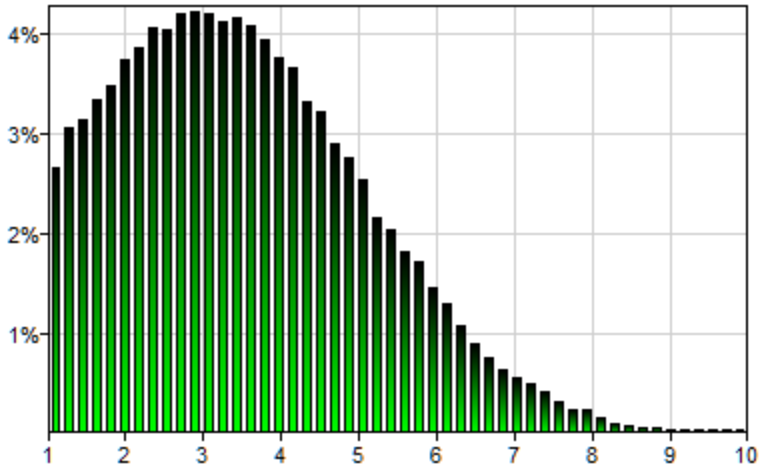
Triangular distribution is a continuous probability distribution with lower limit *min*, upper limit *max* and mode *mod*, where $min \leq mod \leq max$. `Triangular(0,1,3)` will generate a single sample from the following distribution:



TruncNormal(mu,sigma,min)

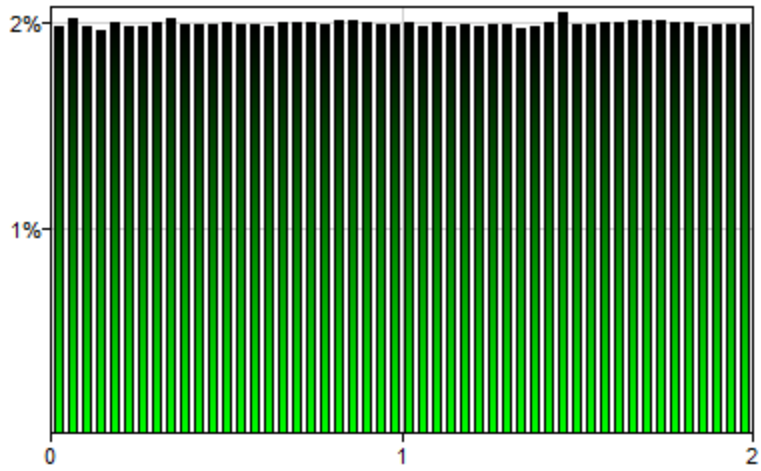
Truncated Normal distribution is essentially a Normal distribution that is truncated at the value *min* (*min*<=*mu*). This distribution is especially useful in situation when we want to limit physically impossible values in the model. The constraint *min*<=*mu* is not restrictive in practice and allows to control efficiency of sample generation, which is performed by rejecting samples smaller than min

from a *Normal*(*mu*,*sigma*) distribution. `TruncNormal(3,2,1)` will generate a single sample from the following distribution:



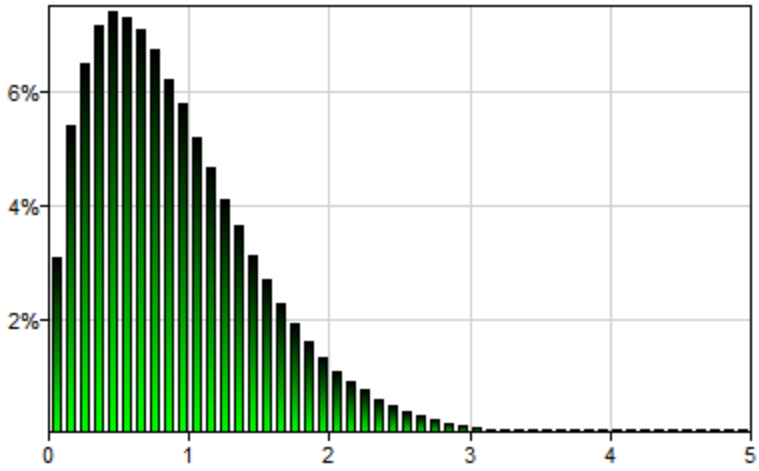
Uniform(a,b)

The continuous uniform distribution, also known as the rectangular distribution, is a family of probability distributions under which any two intervals of the same length are equally probable. It is defined two parameters, *a* and *b*, which are the minimum and the maximum values of the random variable. `Uniform(0,2)` will generate a single sample from the following distribution:



Weibull(lambda,k)

Weibull distribution is a continuous probability distribution named after a Swedish mathematician Waloddi Weibull, used in modeling such phenomena as particle size. It is characterized by two positive real parameters: the scale parameter *lambda* (λ) and the shape parameter *k*. `Weibull(1,1.5)` will generate a single sample from the following distribution:



5.14.3 Arithmetic Functions

Abs(x)

Returns the absolute value of a number, e.g., `Abs(5.3)=Abs(-5.3)=5.3`

Erf(x)

Returns error function ((also called the Gauss error function, inverse of the Normal PDF), e.g., Erf(1.0)=0.842701

Exp(x)

Returns *e* (Euler's number) raised to the power of *x*, e.g., `Exp(2.2)=`$e^{2.2}$`=9.02501`

Gammaln(x)

Returns the natural logarithm of the Gamma function ($\Gamma(x)$), e.g., `Gammaln(2.2)=0.0969475`

GCD(n,k)

Returns the greatest common integer divisor of its two integer arguments *n* and *k*, e.g., `GCD(15,25)=5`. When the arguments are not integers, their fractional part is ignored.

LCM(n,k)

Returns the least common integer multiple of its two integer arguments *n* and *k*, e.g., `LCM(15,25)=75`.When the arguments are not integers, their fractional part is ignored.

Ln(x)

Returns the natural logarithm of *x*, which has to be non-negative, e.g., `Ln(10)=2.30259`

Log(x,b)

Returns the base *b* logarithm of *x*, which has to be non-negative, e.g., `Log(10,2)=3.32193`

Log10(x)

Returns decimal logarithm of *x*, which has to be non-negative, e.g., `Log10(100)=2`

Pow10(x)

Returns 10 raised to the power of *x*, e.g., `Pow10(2)=10`^{2}`=100`

Round(x)

Returns the integer that is nearest to x, e.g., `Round(2.2)=2, Round(3.5)=4`

Sign(x)

Returns 1 if *x*>0, 0 when *x*=0, and -1 if *x*<0, e.g., `Sign(2.2)=1, Sign(0)=0, Sign(-3.5)=-1`

Sqrt(x)

Returns the square root of *x*, which has to be non-negative, e.g., `Sqrt(2)=1.41421`

SqrtPi(x)

Returns square root of π multiplied by *x*, which has to be non-negative, e.g. `SqrtPi(2)=Sqrt(Pi()*2)=2.50663.` This function is provided for the sake of compatibility with Microsoft Excel.

Sum(x1,x2,...)

Returns the sum of its arguments, e.g., `Sum(2.2, 3.5, 1.3)=7.0.` `Sum()` requires at least two arguments.

SumSq(x1,x2,...)

Returns the sum of squares of its arguments, e.g., `SumSq(2.2, 3.5, 1.3)=18.78. SumSq()` requires at least two arguments.

Trim(x,lo,hi)

Trims the value of the argument *x* to a value in the interval <*lo*, *hi*>. If *x*≤*lo*, the function returns *lo*, if *x*≥*hi*, the function returns *hi*, if *lo*<*x*<*hi*, the function returns *x*. The function is a shortcut to two nested conditional functions `If()` and is equivalent to `If(x<lo,lo,If(x>hi,hi,x))`. For example, `Trim(-0.5,0,1)=0, Trim(0.5,0,1)=0.5, Trim(1.5,0,1)=1`

Truncate(x)

Returns the integer part of *x*, e.g., `Truncate(2.2)=2`

5.14.4 Combinatoric Functions

Combin(n,k)

Returns the number of combinations of distinct *k* elements from among *n* elements, e.g., `Combin(10,2)=45`

Fact(n)

Returns the factorial of *n*, e.g., `Fact(5)=5!=120. Fact` of a negative number returns 0.

FactDouble(n)

Returns the product of all even (when *n* is even) or all odd (when *n* is odd) numbers between 1 and *n*, e.g., `FactDouble(5)=15, FactDouble(6)=48. FactDouble` of a negative number returns 0.

Multinomial(n1,n2,...)

Factorial of sum of arguments, divided by the factorials of all arguments, e.g., `Multinomial(2,5,3)=Fact(2+5+3)/(Fact(2)*Fact(5)*Fact(3)=10!/(2!*5!*3!)=2520.` All arguments of `Multinomial` have to be positive.

5.14.5 Trigonometric Functions

Acos(x)

Returns arccosine (*arcus cosinus*) of *x*, e.g., `Acos(-1)=3.14159`

Asin(x)

Returns arcsine (*arcus sinus*) of *x*, e.g., `Asin(1)=1.5708`

Atan(x)

Returns arctangent (*arcus tangens*) of *x*, e.g., `Atan(1)=0.785398`

Atan2(y,x)

Returns arctangent (*arcus tangens*) from *x* and *y* coordinates, e.g., `Atan2(1,1)=0.785398`

Cos(x)

Returns cosine (*cosinus*) of *x*, e.g., `Cos(1)=0.540302`

Pi()

Returns constant π, e.g., `Pi()=3.14159`

Sin(x)

Returns sine (*sinus*) of *x*, e.g., `Sin(1)=0.841471`

Tan(x)

Returns tangent (*tangens*) of *x*, e.g., `Tan(1)=1.55741`

5.14.6 Hyperbolic Functions

Cosh(x)

Returns the hyperbolic cosine of *x*, e.g., `Cosh(1)=1.54308`

Sinh(x)

Returns the hyperbolic sine of *x*, e.g., `Sinh(1)=1.1752`

Tanh(x)

Returns the hyperbolic tangent of *x*, e.g., `Tanh(1)=0.761594`

5.14.7 Logical/Conditional functions

And(b1,b2,...)

Returns the logical conjunction of the arguments, which are all interpreted as Boolean expressions. If any of the expressions evaluates to a zero, `And` returns 0. For example, `And(1=1,2,3)=1`, `And(1=2,2,3)=0`

Choose(index,v0,v1,...,vn)

Returns v_i if *index* is equal to *i*. When *index* evaluates to a value smaller than 0 or larger than *n*-1, the function returns 0. Examples:

```
Choose(0,1,2,3,4,5)=1
Choose(4,1,2,3,4,5)=5
Choose(7,1,2,3,4,5)=0
```

If(cond,tval,fval)

If *cond* evaluates to non-zero return *tval*, *fval* otherwise, e.g., `If(1=2,3,4)=4, If(1,5,10)=5`

Max(x1,x2,...)

Returns the largest of the arguments x_i. Examples:

```
Max(1,2,3,4,5)=5
Max(-3,2,0,2,1)=2
```

Min(x1,x2,...)

Returns the smallest of the arguments x_i. Examples:

```
Min(1,2,3,4,5)=1
Min(-3,2,0,2,1)=-3
```

Or(b1,b2,...)

Returns the logical disjunction of the arguments, which are all interpreted as Boolean expressions. If any of the expressions evaluates to a non-zero value, `Or` returns 1. For example, `Or(1=1,0,0)=1`, `Or(1=0,0,0)=0`

Switch(x,a1,b1,a2,b2,...,[def])

If *x*=*a1*, return *b1*, if *x*=*a2*, return *b2*, when *x* is not equal to any of *a*s, return *def* (default value), which is an optional argument. When *x* is not equal to any of *a*s and no *def* is defined, return 0. Examples:

```
Switch(3,1,111,2,222,3,333,4,444,5,555,999)=333
Switch(8,1,111,2,222,3,333,4,444,5,555,999)=999
Switch(8,1,111,2,222,3,333,4,444,5,555)=0
```

Xor(b1,b2,...)

Returns logical exclusive OR of all arguments, which are all interpreted as Boolean expressions. `Xor` returns a `1` if the number of logical expressions that evaluate to non-zero is odd and a `0` otherwise.

Examples:

```
Xor(0,1,2,-3,0)=1
Xor(1,1,0,2,2)=0
Xor(-5,1,1,-2,2)=1
```

5.14.8 Custom Functions

It is possible to extend the available function set by calling `Network.setExtFunctions`. The functions are defined at the network level and stored within XDSL file. `setExtFunctions` requires an array of strings with the textual definition of functions. Each function definition has the following form:

```
function-name([parameter1[,parameter2...]]) = expression
```

For example, a function adding two numbers can be defined by the following string:

```
add(a,b)=a+b
```

Recursion is not allowed, but it is possible to use the preceding custom functions in the function expression, for example:

```
dbl(x)=x+x
quad(x)=2*dbl(x)
```

Reverse ordering of `dbl` and `quad` in the vector passed to `setExtFunctions` will result in a parser error for `quad`, because the `dbl` function is not defined when `quad` is parsed.

Each parameter specified on the LHS of the equal sign must be used in the function body on the RHS. The following is an incorrect custom function definition, because parameter `c` is not used:

```
myFunc(a,b,c)=sin(a)+cos(b)
```

Custom function can be defined as a constant. In such case the function has no parameters, but its definition and function calls in node equations still require parenthesis:

```
gEarth()=9.80665
```

Random number generators can be used in function definitions:

```
StdNormal()=Normal(0,1)
TwoStep(lo,hi)=Steps(lo,(lo+hi)/2,hi,1,2)
```

To retrieve the custom functions defined for the network, call `Network.getExtFunctions`. By default, there are no custom functions. Each call to `SetExtFunctions` sets up a complete set of custom functions. To add new functions to the existing set, call `getExtFunctions` first, then add new function definition to the output vector before calling `setExtFunctions`.

5.15 Learning

Learning in SMILE can perform two tasks:

- structure learning: create a new network from a dataset
- parameter learning: refine parameters (CPTs) in an existing network

SMILE also supports network validation, which is frequently used after learning to evaluate the results.

5.15.1 Learning network structure

The following classes can be used to learn `Network` from `DataSet`:

- `BayesianSearch`: Bayesian Search, a hill climbing procedure guided by scoring heuristic with random restarts
- `NaiveBayes`: Naive Bayes
- `TAN`: Tree Augmented Naive Bayes, semi-naive method based on the Bayesian Search approach
- `ABN`: Augmented Naive Bayes, another semi-naive method based on the Bayesian Search approach

In the simplest scenario, just create the object representing learning algorithm and call its `learn` method:

Java:

```
DataSet ds = new DataSet();
ds.readFile("mydatafile.txt");
BayesianSearch baySearch = new BayesianSearch();
Network net = baySearch.learn(ds);
```

Python:

```
ds = pysmile.learning.DataSet()
ds.read_file("mydatafile.txt")
baySearch = pysmile.learning.BayesianSearch()
net = baySearch.learn(ds)
```

C#:

```
DataSet ds = new DataSet();
ds.ReadFile("mydatafile.txt");
BayesianSearch baySearch = new BayesianSearch();
Network net = baySearch.Learn(ds);
```

Note that every variable in the dataset takes part in the learning process. After learning the structure, each of the algorithms listed above performs parameter learning with EM, so the output network has nodes with parameters based on the data.

The code example above used the default settings for Bayesian Search. To tweak the learning process, you can change some control options in the learning object before calling its `learn` method. The example below sets the number of iterations and maximum number of parents:

Java:

```
BayesianSearch baySearch = new BayesianSearch();
baySearch.setIterationCount(10);
baySearch.setMaxParents(4);
Network net = baySearch.learn(ds);
```

Python:

```
baySearch = pysmile.learning.BayesianSearch()
baySearch.set_iteration_count(10)
baySearch.set_max_parents(4)
net = baySearch.learn(ds)
```

C#:

```
BayesianSearch baySearch = new BayesianSearch();
baySearch.IterationCount = 10;
baySearch.MaxParents = 4;
Network net = baySearch.Learn(ds);
```

For `NaiveBayes` and its semi-naive derivatives (`TAN` and `ABN`) it is required to specify a class variable identifier (a string value) with a call to `setClassVariableId` before invoking `learn`. This identifier has to match one of the columns in the `DataSet` objects passed to `learn`.

SMILE also contains the `PC` class, which implements the PC structure learning algorithm (algorithm name is an acronym derived from its inventors' names). This algorithm uses `DataSet` as data source, but instead of `Network` learns the `Pattern` object, which is a graph with directed and undirected edges, which is not guaranteed to be acyclic.

BayesianSearch and PC algorithms can use background knowledge, provided by the caller. The background knowledge influences the learned structure by:

- forcing arcs between specified variables
- forbidding arcs between specified variables
- ordering specified variables by temporal tiers: in the resulting structure, there will be no arcs from nodes in higher tiers to nodes in lower tiers

To specify the background knowledge, use BayesianSearch.setBkKnowledge or PC.setBkKnowledge methods.

5.15.2 Learning network parameters

To learn the parameters in the existing Network object, you can use the EM algorithm implemented in EM class. As with structure learning, the data comes in DataSet object. However, the network and the data must be matched to ensure that learning algorithm knows the relationship between the dataset variables and network nodes. If the variables and nodes have identical identifiers, you can use the DataSet.matchNetwork method:

Java:

```
DataSet ds = new DataSet();
Network net = new Network();
// load network and data here
DataMatch[] matching = ds.matchNetwork(net);
em = new EM();
em.learn(ds, net, matching);
```

Python:

```
ds = pysmile.learning.DataSet()
net = pysmile.Network()
# load network and data here
matching = ds.match_network(net)
em = pysmile.learning.EM()
em.learn(ds, net, matching)
```

C#:

```
DataSet ds = new DataSet();
Network net = new Network();
// load network and data here
DataMatch[] matching = ds.MatchNetwork(net);
em = new EM();
em.Learn(ds, net, matching);
```

If your network and data cannot be automatically matched with `DataSet.matchNetwork`, you can build the array of `DataMatch` objects in your own code. `DataMatch` has `node` and `column` fields representing node handle and variable index, respectively. For each node/variable pair you need one element of the array.

5.15.3 Validation

To evaluate the predictive quality of your network you can use the `Validator` class.

The `Validator` constructor requires references to `DataSet` and `Network` objects to be specified. To properly match the network and data the constructor also requires the array of `DataMatch` objects (as did `EM.learn` method).

After the validator object is constructed, you need to specify which nodes in the network are considered class nodes by calling `Validator.addClassNode` method. Validation requires at least one class node.

For each record in the dataset during the validation, the variables matched to non-class nodes are used to set the evidence. The posterior probabilities are then calculated and for each class node the outcome with the highest probability is selected as a predicted outcome. The prediction is compared with an outcome in the dataset variable associated with the class node. The number of matches and calculated posteriors are used to obtain the accuracy, confusion matrix, ROC and calibration curves.

Validation can be either performed without parameter learning using `Validator.test` method, or with parameter learning using `Validator::kFold` and `leaveOneOut` methods. K-fold crossvalidation divides the dataset into K parts of equal size, trains the network on K-1 parts, and tests it on the last, Kth part. The process is repeated K times, with a different part of the data being selected for testing. Leave-one-out is an extreme case of K-fold, in which K is equal to the number of records in the data set.

The example below performs K-fold crossvalidation with 5 folds using one class node. The accuracy is obtained for the outcome with the index zero (that is, the first outcome of the node).

Java:

```
DataSet ds = new DataSet();
Network net = new Network();
// load network and data here
DataMatch[] matching = ds.matchNetwork();
Validator validator = new Validator(ds, net, matching);
int classNodehandle = net.getNode("someNodeId");
validator.addClassNode(classNodeHandle);
EM em = new EM();
```

```
// optionally tweak EM options here
validator.kFold(em, 5);
double acc = validator.getAccuracy(classNodeHandle, 0);
```

Python:

```
ds = pysmile.learning.DataSet()
net = pysmile.Network()
# load network and data here
matching = ds.match_network(net)
validator = pysmile.learning.Validator(ds, net, matching)
classNodehandle = net.getNode("someNodeId")
validator.addClassNode(classNodeHandle)
em = pysmile.learning.EM()
# optionally tweak EM options here
validator.k_fold(em, 5)
acc = validator.get_accuracy(classNodeHandle, 0)
```

C#:

```
DataSet ds = new DataSet();
Network net = new Network();
// load network and data here
DataMatch[] matching = ds.MatchNetwork();
Validator validator = new Validator(ds, net, matching);
int classNodehandle = net.GetNode("someNodeId");
validator.AddClassNode(classNodeHandle);
EM em = new EM();
// optionally tweak EM options here
validator.KFold(em, 5);
double acc = validator.GetAccuracy(classNodeHandle, 0);
```

This page is intentionally left blank.

Tutorials

6 Tutorials

The complete, functionally equivalent source code for all the tutorials in Java, Python and C# is provided later in this chapter. The multi-line code snippets in the tutorial descriptions preceeding the source code are also given in Java, Python and C#. The inline examples in text paragraphs are using Java syntax (like `Network.updateBeliefs`) for brevity.

You can also download tutorial sources from our documentation site at https://support.bayesfusion.com/docs

6.1 Tutorial 1: Creating a Bayesian Network

Consider a slight twist on the problem described in the *Hello, SMILE Wrapper!* [16] section of this manual.

The twist will include adding an additional variable *State of the economy* (with the identifier *Economy*) with three outcomes (*Up*, *Flat*, and *Down*) modeling the developments in the economy. These developments are relevant to the decision, as they are impacting expert predictions. When the economy is heading *Up*, our expert makes more optimistic predictions, when it is heading *Down*, the expert makes more pessimistic predictions for the same venture. This is reflected by a directed arc from the node *State of the economy* to the node *Expert forecast*. *State of the economy* also impacts the probability of venture being successful, which we model by adding another arc.

We will show how to create this model and how to save it to disk. In subsequent tutorials, we will show how to enter observations (evidence), how to perform inference, and how to retrieve the calculation results.

We start by declaring our network variable. The three nodes in the network are subsequently created by calling a helper method `createCptNode`.

Java:

```
Network net = new Network();
```

```
int e = createCptNode(net,
    "Economy", "State of the economy",
    new String[] {"Up","Flat","Down"},
    160, 40);
int s = createCptNode(net,
    "Success", "Success of the venture",
    new String[] {"Success","Failure"},
    60, 40);
int f = createCptNode(net,
    "Forecast", "Expert forecast",
    new String[] {"Good","Moderate","Poor"},
    110, 140);
```

Python:

```
net = pysmile.Network()
e = self.create_cpt_node(net,
    "Economy", "State of the economy",
    ["Up","Flat","Down"],
    160, 40)
s = self.create_cpt_node(net,
    "Success", "Success of the venture",
    ["Success","Failure"],
    60, 40)
f = self.create_cpt_node(net,
    "Forecast", "Expert forecast",
    ["Good","Moderate","Poor"],
    110, 140)
```

C#:

```
Network net = new Network();
int e = CreateCptNode(net,
    "Economy", "State of the economy",
    new String[] { "Up", "Flat", "Down" },
    160, 40);
int s = CreateCptNode(net,
    "Success", "Success of the venture",
    new String[] { "Success", "Failure" },
    60, 40);
int f = CreateCptNode(net,
    "Forecast", "Expert forecast",
    new String[] { "Good", "Moderate", "Poor" },
    110, 140);
```

Before connecting the nodes with arcs, let's have a look at the createCptNode method.

Java:

```
private static int createCptNode(
        Network net, String id, String name,
        String[] outcomes, int xPos, int yPos) {
    int handle = net.addNode(Network.NodeType.CPT, id);
    net.setNodeName(handle, name);
    net.setNodePosition(handle, xPos, yPos, 85, 55);
    int initialOutcomeCount = net.getOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i ++) {
        net.setOutcomeId(handle, i, outcomes[i]);
```

```
    }
    for (int i = initialOutcomeCount; i < outcomes.length; i ++) {
        net.addOutcome(handle, outcomes[i]);
    }
    return handle;
}
```

Python:

```
def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.CPT, id)
    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)
    initial_outcome_count = net.get_outcome_count(handle)
    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])
    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])
    return handle
```

C#:

```
private static int CreateCptNode(
    Network net, String id, String name,
    String[] outcomes, int xPos, int yPos)
{
    int handle = net.AddNode(Network.NodeType.Cpt, id);
    net.SetNodeName(handle, name);
    net.SetNodePosition(handle, xPos, yPos, 85, 55);
    int initialOutcomeCount = net.GetOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i++)
    {
        net.SetOutcomeId(handle, i, outcomes[i]);
    }
    for (int i = initialOutcomeCount; i < outcomes.Length; i++)
    {
        net.AddOutcome(handle, outcomes[i]);
    }
    return handle;
}
```

The function creates a CPT node with specified identifier, name, outcomes and position on the screen. CPT nodes are created with two outcomes named *State0* and *State1*. To change the number of node outcomes and rename them, we will use two loops - the first renames the default outcomes and the second adds new outcomes.

We can now add arcs linking the nodes. Note that we can use either node handles or node identifiers. In this simple program, we use both forms just to showcase this feature.

Java:

```
net.addArc(e, s);
net.addArc(s, f);
net.addArc("Economy", "Forecast");
```

Python:

```
net.add_arc(e, s)
net.add_arc(s, f)
net.add_arc("Economy", "Forecast")
```

C#:

```
net.AddArc(e, s);
net.AddArc(s, f);
net.AddArc("Economy", "Forecast");
```

Next step is to initialize the conditional probability tables of the nodes. See the *Multidimensional arrays*[29] section in this manual for the description of the CPT memory layout. For each of three nodes we call the `Network.setNodeDefinition` method. Here's how we set the probabilities of the *Success* node - note that the comments for each probability show the combination of outcomes it is defined for.

Java:

```
double[] successDef = new double[] {
    0.3, // P(Success=S|Economy=U)
    0.7, // P(Success=F|Economy=U)
    0.2, // P(Success=S|Economy=F)
    0.8, // P(Success=F|Economy=F)
    0.1, // P(Success=S|Economy=D)
    0.9  // P(Success=F|Economy=D)
};
net.setNodeDefinition(s, successDef);
```

Python:

```
successDef = [
    0.3, # P(Success=S|Economy=U)
    0.7, # P(Success=F|Economy=U)
    0.2, # P(Success=S|Economy=F)
    0.8, # P(Success=F|Economy=F)
    0.1, # P(Success=S|Economy=D)
    0.9  # P(Success=F|Economy=D)
]
net.set_node_definition(s, successDef)
```

C#:

```
double[] successDef = new double[]
{
    0.3, // P(Success=S|Economy=U)
    0.7, // P(Success=F|Economy=U)
    0.2, // P(Success=S|Economy=F)
    0.8, // P(Success=F|Economy=F)
    0.1, // P(Success=S|Economy=D)
    0.9  // P(Success=F|Economy=D)
};
net.SetNodeDefinition(s, successDef);
```

With CPTs initialized our network is complete. We write its contents to the tutorial1.xdsl file. Tutorial 2 will load this file and perform the inference. The split between tutorials is artificial, your program can use networks right after its creation without the need to write/read from the file system.

Java:

```
net.writeFile("tutorial1.xdsl");
```

Python:

```
net.write_file("tutorial1.xdsl");
```

C#:

```
net.WriteFile("tutorial1.xdsl");
```

6.1.1 Tutorial1.java

```
package tutorials;

import smile.*;

// Tutorial1 creates a simple network with three nodes,
// then writes its content as XDSL file to disk.

public class Tutorial1 {
    public static void run() {
        System.out.println("Starting Tutorial1...");
        Network net = new Network();

        int e = createCptNode(net,
            "Economy", "State of the economy",
            new String[] {"Up","Flat","Down"},
            160, 40);

        int s = createCptNode(net,
            "Success", "Success of the venture",
            new String[] {"Success","Failure"},
            60, 40);

        int f = createCptNode(net,
            "Forecast", "Expert forecast",
            new String[] {"Good","Moderate","Poor"},
            110, 140);

        net.addArc(e, s);
        net.addArc(s, f);

        // we can also use node identifiers when creating arcs
        net.addArc("Economy", "Forecast");

        double[] economyDef = {
            0.2, // P(Economy=U)
            0.7, // P(Economy=F)
            0.1  // P(Economy=D)
        };
        net.setNodeDefinition(e, economyDef);

        double[] successDef = new double[] {
            0.3, // P(Success=S|Economy=U)
            0.7, // P(Success=F|Economy=U)
            0.2, // P(Success=S|Economy=F)
```

```
            0.8, // P(Success=F|Economy=F)
            0.1, // P(Success=S|Economy=D)
            0.9  // P(Success=F|Economy=D)
        };
        net.setNodeDefinition(s, successDef);

        double[] forecastDef = new double[] {
            0.70, // P(Forecast=G|Success=S,Economy=U)
            0.29, // P(Forecast=M|Success=S,Economy=U)
            0.01, // P(Forecast=P|Success=S,Economy=U)

            0.65, // P(Forecast=G|Success=S,Economy=F)
            0.30, // P(Forecast=M|Success=S,Economy=F)
            0.05, // P(Forecast=P|Success=S,Economy=F)

            0.60, // P(Forecast=G|Success=S,Economy=D)
            0.30, // P(Forecast=M|Success=S,Economy=D)
            0.10, // P(Forecast=P|Success=S,Economy=D)

            0.15, // P(Forecast=G|Success=F,Economy=U)
            0.30, // P(Forecast=M|Success=F,Economy=U)
            0.55, // P(Forecast=P|Success=F,Economy=U)

            0.10, // P(Forecast=G|Success=F,Economy=F)
            0.30, // P(Forecast=M|Success=F,Economy=F)
            0.60, // P(Forecast=P|Success=F,Economy=F)

            0.05, // P(Forecast=G|Success=F,Economy=D)
            0.25, // P(Forecast=G|Success=F,Economy=D)
            0.70  // P(Forecast=G|Success=F,Economy=D)
        };
        net.setNodeDefinition(f, forecastDef);

        net.writeFile("tutorial1.xdsl");

        System.out.println(
            "Tutorial1 complete: Network written to tutorial1.xdsl");
    }


    private static int createCptNode(
            Network net, String id, String name,
            String[] outcomes, int xPos, int yPos) {
        int handle = net.addNode(Network.NodeType.CPT, id);

        net.setNodeName(handle, name);
        net.setNodePosition(handle, xPos, yPos, 85, 55);

        int initialOutcomeCount = net.getOutcomeCount(handle);
        for (int i = 0; i < initialOutcomeCount; i ++) {
            net.setOutcomeId(handle, i, outcomes[i]);
        }

        for (int i = initialOutcomeCount; i < outcomes.length; i ++) {
            net.addOutcome(handle, outcomes[i]);
        }
```

```
        return handle;
    }
}
```

6.1.2 Tutorial1.py

```
import pysmile

# Tutorial1 creates a simple network with three nodes,
# then writes its content as XDSL file to disk.

class Tutorial1:
    def __init__(self):
        net = pysmile.Network()

        e = self.create_cpt_node(net,
            "Economy", "State of the economy",
            ["Up","Flat","Down"],
            160, 40)

        s = self.create_cpt_node(net,
            "Success", "Success of the venture",
            ["Success","Failure"],
            60, 40)

        f = self.create_cpt_node(net,
            "Forecast", "Expert forecast",
            ["Good","Moderate","Poor"],
            110, 140)

        net.add_arc(e, s)
        net.add_arc(s, f)

        # we can also use node identifiers when creating arcs
        net.add_arc("Economy", "Forecast");

        economyDef = [
            0.2, # P(Economy=U)
            0.7, # P(Economy=F)
            0.1  # P(Economy=D)
        ]
        net.set_node_definition(e, economyDef)

        successDef = [
            0.3, # P(Success=S|Economy=U)
            0.7, # P(Success=F|Economy=U)
            0.2, # P(Success=S|Economy=F)
            0.8, # P(Success=F|Economy=F)
            0.1, # P(Success=S|Economy=D)
            0.9  # P(Success=F|Economy=D)
        ]
        net.set_node_definition(s, successDef)

        forecastDef = [
            0.70, # P(Forecast=G|Success=S,Economy=U)
            0.29, # P(Forecast=M|Success=S,Economy=U)
```

```
            0.01, # P(Forecast=P|Success=S,Economy=U)

            0.65, # P(Forecast=G|Success=S,Economy=F)
            0.30, # P(Forecast=M|Success=S,Economy=F)
            0.05, # P(Forecast=P|Success=S,Economy=F)

            0.60, # P(Forecast=G|Success=S,Economy=D)
            0.30, # P(Forecast=M|Success=S,Economy=D)
            0.10, # P(Forecast=P|Success=S,Economy=D)

            0.15,  # P(Forecast=G|Success=F,Economy=U)
            0.30, # P(Forecast=M|Success=F,Economy=U)
            0.55, # P(Forecast=P|Success=F,Economy=U)

            0.10, # P(Forecast=G|Success=F,Economy=F)
            0.30, # P(Forecast=M|Success=F,Economy=F)
            0.60, # P(Forecast=P|Success=F,Economy=F)

            0.05, # P(Forecast=G|Success=F,Economy=D)
            0.25, # P(Forecast=G|Success=F,Economy=D)
            0.70  # P(Forecast=G|Success=F,Economy=D)
        ]
        net.set_node_definition(f, forecastDef)

        net.write_file("tutorial1.xdsl")

        print("Tutorial1 complete: Network written to tutorial1.xdsl")


    def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
        handle = net.add_node(pysmile.NodeType.CPT, id)

        net.set_node_name(handle, name)
        net.set_node_position(handle, x_pos, y_pos, 85, 55)

        initial_outcome_count = net.get_outcome_count(handle)

        for i in range(0, initial_outcome_count):
            net.set_outcome_id(handle, i, outcomes[i])

        for i in range(initial_outcome_count, len(outcomes)):
            net.add_outcome(handle, outcomes[i])

        return handle
```

6.1.3 Tutorial1.cs

```
using System;
using Smile;

// Tutorial1 creates a simple network with three nodes,
// then writes its content as XDSL file to disk.

namespace SmileNetTutorial
{
    class Tutorial1
```

```
{
    public static void Run()
    {
        Console.WriteLine("Starting Tutorial1...");
        Network net = new Network();

        int e = CreateCptNode(net,
            "Economy", "State of the economy",
            new String[] { "Up", "Flat", "Down" },
            160, 40);

        int s = CreateCptNode(net,
            "Success", "Success of the venture",
            new String[] { "Success", "Failure" },
            60, 40);

        int f = CreateCptNode(net,
            "Forecast", "Expert forecast",
            new String[] { "Good", "Moderate", "Poor" },
            110, 140);

        net.AddArc(e, s);
        net.AddArc(s, f);

        // we can also use node identifiers when creating arcs
        net.AddArc("Economy", "Forecast");

        double[] economyDef =
        {
            0.2, // P(Economy=U)
            0.7, // P(Economy=F)
            0.1  // P(Economy=D)
        };
        net.SetNodeDefinition(e, economyDef);

        double[] successDef = new double[]
        {
            0.3, // P(Success=S|Economy=U)
            0.7, // P(Success=F|Economy=U)
            0.2, // P(Success=S|Economy=F)
            0.8, // P(Success=F|Economy=F)
            0.1, // P(Success=S|Economy=D)
            0.9  // P(Success=F|Economy=D)
        };
        net.SetNodeDefinition(s, successDef);

        double[] forecastDef = new double[]
        {
            0.70, // P(Forecast=G|Success=S,Economy=U)
            0.29, // P(Forecast=M|Success=S,Economy=U)
            0.01, // P(Forecast=P|Success=S,Economy=U)

            0.65, // P(Forecast=G|Success=S,Economy=F)
            0.30, // P(Forecast=M|Success=S,Economy=F)
            0.05, // P(Forecast=P|Success=S,Economy=F)

            0.60, // P(Forecast=G|Success=S,Economy=D)
```

```
                0.30, // P(Forecast=M|Success=S,Economy=D)
                0.10, // P(Forecast=P|Success=S,Economy=D)

                0.15,  // P(Forecast=G|Success=F,Economy=U)
                0.30, // P(Forecast=M|Success=F,Economy=U)
                0.55, // P(Forecast=P|Success=F,Economy=U)

                0.10, // P(Forecast=G|Success=F,Economy=F)
                0.30, // P(Forecast=M|Success=F,Economy=F)
                0.60, // P(Forecast=P|Success=F,Economy=F)

                0.05, // P(Forecast=G|Success=F,Economy=D)
                0.25, // P(Forecast=G|Success=F,Economy=D)
                0.70  // P(Forecast=G|Success=F,Economy=D)
            };
            net.SetNodeDefinition(f, forecastDef);

            net.WriteFile("tutorial1.xdsl");

            Console.WriteLine(
                "Tutorial1 complete: Network written to tutorial1.xdsl");
        }


        private static int CreateCptNode(
            Network net, String id, String name,
            String[] outcomes, int xPos, int yPos)
        {
            int handle = net.AddNode(Network.NodeType.Cpt, id);

            net.SetNodeName(handle, name);
            net.SetNodePosition(handle, xPos, yPos, 85, 55);

            int initialOutcomeCount = net.GetOutcomeCount(handle);
            for (int i = 0; i < initialOutcomeCount; i++)
            {
                net.SetOutcomeId(handle, i, outcomes[i]);
            }

            for (int i = initialOutcomeCount; i < outcomes.Length; i++)
            {
                net.AddOutcome(handle, outcomes[i]);
            }

            return handle;
        }
    }

}
```

6.2 Tutorial 2: Inference with a Bayesian Network

Tutorial 2 begins with the model we have previously created. We will perform multiple calls to Bayesian inference algorithm through the `Network.updateBeliefs`, starting with network

without any evidence. After each `updateBeliefs` call the posterior probabilities of nodes will be displayed.

The model is loaded with the `Network.readFile`.

Java:

```
Java:
net.readFile("tutorial1.xdsl");
```

Python:

```
net.read_file("tutorial1.xdsl");
```

C#:

```
net.ReadFile("tutorial1.xdsl");
```

We update the probabilities and proceed to display them using the helper method `printAllPosteriors` defined in this tutorial.

Java:

```
printf("Posteriors with no evidence set:\n");
net.updateBeliefs();
printAllPosteriors(net);
```

Python:

```
print("Posteriors with no evidence set:")
net.update_beliefs()
self.print_all_posteriors(net)
```

C#:

```
Console.WriteLine("Posteriors with no evidence set:");
net.UpdateBeliefs();
PrintAllPosteriors(net);
```

`printAllPosteriors` displays posterior probabilities calculated by `updateBeliefs` for each node. To iterate over the nodes, `Network.getFirstNode` and `getNextNode` are used. In the body of the loop we call another helper function, `printPosteriors`.

Java:

```
for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
    printPosteriors(net, h);
}
```

Python:

```
handle = net.get_first_node()
while (handle >= 0):
    self.print_posteriors(net, handle)
    handle = net.get_next_node(handle)
```

C#:

```
for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
{
    PrintPosteriors(net, h);
}
```

`printPosteriors` checks if node has evidence set by calling `Network.isEvidence`; if this is the case the name of the evidence state is displayed. Please note that for demonstration purposes this tutorial calls `Network.getEvidence` to obtain an integer representing the index of the evidence state, then converts the integer index to outcome id with `Network.getOutcomeId`. The `Network.getEvidenceId` method combines these two actions into one call.

If node is does not have any evidence set, `printPosteriors` iterates over all states and displays the posterior probability of each.

Back in the main function of the tutorial, we repeatedly call `changeEvidenceAndUpdate` helper function to change evidence, update network and display posteriors. Note that we need to call different methods of the `Network` class to set evidence to the specified outcome (`setEvidence`) and remove the evidence (`clearEvidence`).

6.2.1 Tutorial2.java

```
package tutorials;

import smile.*;

// Tutorial2 loads the XDSL file created by Tutorial1,
// then performs the series of inference calls,
// changing evidence each time.

public class Tutorial2 {
    public static void run() {
        System.out.println("Starting Tutorial2...");
        Network net = new Network();

        // load the network created by Tutorial1
        net.readFile("tutorial1.xdsl");

        System.out.println("Posteriors with no evidence set:");
        net.updateBeliefs();
        printAllPosteriors(net);

        System.out.println("Setting Forecast=Good.");
        changeEvidenceAndUpdate(net, "Forecast", "Good");

        System.out.println("Adding Economy=Up.");
        changeEvidenceAndUpdate(net, "Economy", "Up");

        System.out.println("Changing Forecast to Poor, keeping Economy=Up.");
        changeEvidenceAndUpdate(net, "Forecast", "Poor");

        System.out.println(
```

```
            "Removing evidence from Economy, keeping Forecast=Poor.");
        changeEvidenceAndUpdate(net, "Economy", null);

        System.out.println("Tutorial2 complete.");
    }


    private static void printPosteriors(Network net, int nodeHandle) {
        String nodeId = net.getNodeId(nodeHandle);
        if (net.isEvidence(nodeHandle)) {
            System.out.printf("%s has evidence set (%s)\n",
                nodeId,
                net.getOutcomeId(nodeHandle, net.getEvidence(nodeHandle)));
        } else {
            double[] posteriors = net.getNodeValue(nodeHandle);
            for (int i = 0; i < posteriors.length; i ++) {
                System.out.printf("P(%s=%s)=%f\n",
                    nodeId,
                    net.getOutcomeId(nodeHandle, i),
                    posteriors[i]);
            }
        }
    }


    private static void printAllPosteriors(Network net) {
        for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
            printPosteriors(net, h);
        }
        System.out.println();
    }


    private static void changeEvidenceAndUpdate(
            Network net, String nodeId, String outcomeId) {
        if (outcomeId != null) {
            net.setEvidence(nodeId, outcomeId);
        } else {
            net.clearEvidence(nodeId);
        }

        net.updateBeliefs();
        printAllPosteriors(net);
    }

}
```

6.2.2 Tutorial2.py

```
import pysmile

# Tutorial2 loads the XDSL file created by Tutorial1,
# then performs the series of inference calls,
# changing evidence each time.

class Tutorial2:
```

```
def __init__(self):
    print("Starting Tutorial2...")
    net = pysmile.Network()

    # load the network created by Tutorial1
    net.read_file("tutorial1.xdsl")

    print("Posteriors with no evidence set:")
    net.update_beliefs()
    self.print_all_posteriors(net)

    print("Setting Forecast=Good.")
    self.change_evidence_and_update(net, "Forecast", "Good")

    print("Adding Economy=Up.")
    self.change_evidence_and_update(net, "Economy", "Up")

    print("Changing Forecast to Poor, keeping Economy=Up.")
    self.change_evidence_and_update(net, "Forecast", "Poor")

    print("Removing evidence from Economy, keeping Forecast=Poor.")
    self.change_evidence_and_update(net, "Economy", None)

    print("Tutorial2 complete.")


def print_posteriors(self, net, node_handle):
    node_id = net.get_node_id(node_handle)
    if net.is_evidence(node_handle):
        print(node_id + " has evidence set (" +
              net.get_outcome_id(node_handle,
                                 net.get_evidence(node_handle)) + ")")
    else :
        posteriors = net.get_node_value(node_handle)
        for i in range(0, len(posteriors)):
            print("P(" + node_id + "=" +
                  net.get_outcome_id(node_handle, i) +
                  ")=" + str(posteriors[i]))

def print_all_posteriors(self, net):
    for handle in net.get_all_nodes():
        self.print_posteriors(net, handle)


def change_evidence_and_update(self, net, node_id, outcome_id):
    if outcome_id is not None:
        net.set_evidence(node_id, outcome_id)
    else:
        net.clear_evidence(node_id)

    net.update_beliefs()
    self.print_all_posteriors(net)
    print("")
```

6.2.3 Tutorial2.cs

```
using System;
using Smile;

// Tutorial2 loads the XDSL file created by Tutorial1,
// then performs the series of inference calls,
// changing evidence each time.

namespace SmileNetTutorial
{
    class Tutorial2
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial2...");
            Network net = new Network();

            // load the network created by Tutorial1
            net.ReadFile("tutorial1.xdsl");

            Console.WriteLine("Posteriors with no evidence set:");
            net.UpdateBeliefs();
            PrintAllPosteriors(net);

            Console.WriteLine("Setting Forecast=Good.");
            ChangeEvidenceAndUpdate(net, "Forecast", "Good");

            Console.WriteLine("Adding Economy=Up.");
            ChangeEvidenceAndUpdate(net, "Economy", "Up");

            Console.WriteLine("Changing Forecast to Poor, keeping Economy=Up.");
            ChangeEvidenceAndUpdate(net, "Forecast", "Poor");

            Console.WriteLine(
                "Removing evidence from Economy, keeping Forecast=Poor.");
            ChangeEvidenceAndUpdate(net, "Economy", null);

            Console.WriteLine("Tutorial2 complete.");
        }


        private static void PrintPosteriors(Network net, int nodeHandle)
        {
            String nodeId = net.GetNodeId(nodeHandle);
            if (net.IsEvidence(nodeHandle))
            {
                Console.WriteLine("{0} has evidence set ({1})",
                    nodeId,
                    net.GetOutcomeId(nodeHandle, net.GetEvidence(nodeHandle)));
            }
            else
            {
                double[] posteriors = net.GetNodeValue(nodeHandle);
                for (int i = 0; i < posteriors.Length; i++)
                {
                    Console.WriteLine("P({0}={1})={2}",
```

```
                    nodeId,
                    net.GetOutcomeId(nodeHandle, i),
                    posteriors[i]);
            }
        }
    }


    private static void PrintAllPosteriors(Network net)
    {
        for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
        {
            PrintPosteriors(net, h);
        }
        Console.WriteLine();
    }


    private static void ChangeEvidenceAndUpdate(
        Network net, String nodeId, String outcomeId)
    {
        if (outcomeId != null)
        {
            net.SetEvidence(nodeId, outcomeId);
        }
        else
        {
            net.ClearEvidence(nodeId);
        }

        net.UpdateBeliefs();
        PrintAllPosteriors(net);
    }

}
}
```

6.3 Tutorial 3: Exploring the contents of a model

This tutorial will not perform any calculations. Instead, we will display the information about the network structure (nodes and arcs) and parameters (in this case, conditional probability tables).

We load the network created by `Tutorial1` and for each node invoke the locally defined helper method `printNodeInfo`. This is where real work is done. The first displayed node attributes are identifier and name.

Java:

```
System.out.printf("Node id/name: %s/%s\n",
    net.getNodeId(nodeHandle),
    net.getNodeName(nodeHandle));
```

Python:

```
print("Node id/name: " + net.get_node_id(node_handle) + "/" +
    net.get_node_name(node_handle))
```

C#:

```
Console.WriteLine("Node id/name: {0}/{1}",
    net.GetNodeId(nodeHandle),
    net.GetNodeName(nodeHandle));
```

The identfiers of node's outcomes are next.

Java:

```
System.out.print("  Outcomes:");
for (String outcomeId: net.getOutcomeIds(nodeHandle)) {
    System.out.print(" " + outcomeId);
}
```

Python:

```
print("  Outcomes: " + " ".join(net.get_outcome_ids(node_handle)))
```

C#:

```
foreach (String outcomeId in net.GetOutcomeIds(nodeHandle))
{
    Console.Write(" " + outcomeId);
}
```

The parents of the node follow.

Java:

```
String[] parentIds = net.getParentIds(nodeHandle);
if (parentIds.length > 0) {
    System.out.print("  Parents:");
    for (String parentId: parentIds) {
        System.out.print(" " + parentId);
    }
    System.out.println();
}
```

Python:

```
parent_ids = net.get_parent_ids(node_handle)
if len(parent_ids) > 0:
    print("  Parents: " + " ".join(parent_ids))
```

C#:

```
String[] parentIds = net.GetParentIds(nodeHandle);
if (parentIds.Length > 0)
{
    Console.Write("  Parents:");
    foreach (String parentId in parentIds)
    {
        Console.Write(" " + parentId);
    }
    Console.WriteLine();
```

```
}
```

We proceed to display information about node's children. The code fragment is virtually identical to the iteration over parents. Note that while tutorial uses `Network.getParentIds` and `getChildIds` to obtain identifiers of related nodes, we could alternatively use the `Network.getParents` and `getChildren`, which return the node handles (integer numbers) instead of identifiers (strings).

Finally, the node probabilities are displayed by the `printCptMatrix` helper. We retrieve the single-dimensional array containing the probabilities with `Network.getNodeDefinition` and iterate over its elements, translating each index into a multi-dimensional coordinates of the elements within node's CPT. This conversion is performed in the helper method `indexToCoords`, which requires the information about the outcome counts of the node and its parents in its `dimSizes` input parameter. The first part of printCptMatrix obtains the CPT and initializes dimSizes

Java:

```
double[] cpt = net.getNodeDefinition(nodeHandle);
int[] parents = net.getParents(nodeHandle);
int dimCount = 1 + parents.length;
int[] dimSizes = new int[dimCount];
for (int i = 0; i < dimCount - 1; i ++) {
    dimSizes[i] = net.getOutcomeCount(parents[i]);
}
dimSizes[dimSizes.length - 1] = net.getOutcomeCount(nodeHandle);
```

Python:

```
cpt = net.get_node_definition(node_handle)
parents = net.get_parents(node_handle)
dim_count = 1 + len(parents)
dim_sizes = [0] * dim_count
for i in range(0, dim_count - 1):
    dim_sizes[i] = net.get_outcome_count(parents[i])
dim_sizes[len(dim_sizes) - 1] = net.get_outcome_count(node_handle)
```

C#:

```
double[] cpt = net.GetNodeDefinition(nodeHandle);
int[] parents = net.GetParents(nodeHandle);
int dimCount = 1 + parents.Length;
int[] dimSizes = new int[dimCount];
for (int i = 0; i < dimCount - 1; i++)
{
    dimSizes[i] = net.GetOutcomeCount(parents[i]);
}
dimSizes[dimSizes.Length - 1] = net.GetOutcomeCount(nodeHandle);
```

The main loop in the printCptMatrix uses performs the iteration over the elements of the CPT. The `coords` integer array is filled by indexToCoords and used to print the textual information about the outcome of the node and parents for each element. Note that the node outcome is the rightmost entry in the `coords`. The parents' outcome indexes start from the left at index 0 in the `coords`. Part of the `Tutorial3` output for the node *Forecast* is show below. All lines starting with "P"(were printed by `printCptMatrix`.

```
Node: Expert forecast
```

```
    Outcomes: Good Moderate Poor
    Parents: Success Economy
    Definition type: CPT
      P(Good | Success=Success,Economy=Up)=0.7
      P(Moderate | Success=Success,Economy=Up)=0.29
      P(Poor | Success=Success,Economy=Up)=0.01
      P(Good | Success=Success,Economy=Flat)=0.65
      P(Moderate | Success=Success,Economy=Flat)=0.3
      P(Poor | Success=Success,Economy=Flat)=0.05
      P(Good | Success=Success,Economy=Down)=0.6
      P(Moderate | Success=Success,Economy=Down)=0.3
      P(Poor | Success=Success,Economy=Down)=0.1
      P(Good | Success=Failure,Economy=Up)=0.15
      P(Moderate | Success=Failure,Economy=Up)=0.3
      P(Poor | Success=Failure,Economy=Up)=0.55
      P(Good | Success=Failure,Economy=Flat)=0.1
      P(Moderate | Success=Failure,Economy=Flat)=0.3
      P(Poor | Success=Failure,Economy=Flat)=0.6
      P(Good | Success=Failure,Economy=Down)=0.05
      P(Moderate | Success=Failure,Economy=Down)=0.25
      P(Poor | Success=Failure,Economy=Down)=0.7
```

6.3.1 Tutorial3.java

```
package tutorials;

import smile.*;

// Tutorial3 loads the XDSL file and prints the information
// about the structure (nodes and arcs) and the parameters
// (conditional probabilities of the nodes) of the network.

public class Tutorial3 {
    public static void run() {
        System.out.println("Starting Tutorial3...");
        Network net = new Network();

        // load the network created by Tutorial1
        net.readFile("tutorial1.xdsl");

        for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
            printNodeInfo(net, h);
        }

        System.out.println("Tutorial3 complete.");
    }


    private static void printNodeInfo(Network net, int nodeHandle) {
        System.out.printf("Node id/name: %s/%s\n",
            net.getNodeId(nodeHandle),
            net.getNodeName(nodeHandle));

        System.out.print("  Outcomes:");
        for (String outcomeId: net.getOutcomeIds(nodeHandle)) {
```

```
            System.out.print(" " + outcomeId);
        }
        System.out.println();

        String[] parentIds = net.getParentIds(nodeHandle);
        if (parentIds.length > 0) {
            System.out.print("  Parents:");
            for (String parentId: parentIds) {
                System.out.print(" " + parentId);
            }
            System.out.println();
        }

        String[] childIds = net.getChildIds(nodeHandle);
        if (childIds.length > 0) {
            System.out.print("  Children:");
            for (String childId: childIds) {
                System.out.print(" " + childId);
            }
            System.out.println();
        }

        printCptMatrix(net, nodeHandle);
    }


    private static void printCptMatrix(Network net, int nodeHandle) {
        double[] cpt = net.getNodeDefinition(nodeHandle);
        int[] parents = net.getParents(nodeHandle);
        int dimCount = 1 + parents.length;

        int[] dimSizes = new int[dimCount];
        for (int i = 0; i < dimCount - 1; i ++) {
            dimSizes[i] = net.getOutcomeCount(parents[i]);
        }
        dimSizes[dimSizes.length - 1] = net.getOutcomeCount(nodeHandle);

        int[] coords = new int[dimCount];
        for (int elemIdx = 0; elemIdx < cpt.length; elemIdx ++) {
            indexToCoords(elemIdx, dimSizes, coords);

            String outcome = net.getOutcomeId(nodeHandle, coords[dimCount - 1]);
            System.out.printf("    P(%s", outcome);

            if (dimCount > 1) {
                System.out.print(" | ");
                for (int parentIdx = 0; parentIdx < parents.length; parentIdx++)
                {
                    if (parentIdx > 0) System.out.print(",");
                    int parentHandle = parents[parentIdx];
                    System.out.printf("%s=%s",
                        net.getNodeId(parentHandle),
                        net.getOutcomeId(parentHandle, coords[parentIdx]));
                }
            }

            double prob = cpt[elemIdx];
```

```
            System.out.printf(")=%f\n", prob);
        }
    }


    private static void indexToCoords(int index, int[] dimSizes, int[] coords) {
        int prod = 1;
        for (int i = dimSizes.length - 1; i >= 0; i --) {
            coords[i] = (index / prod) % dimSizes[i];
            prod *= dimSizes[i];
        }
    }

}
```

6.3.2 Tutorial3.py

```
import pysmile

# Tutorial3 loads the XDSL file and prints the information
# about the structure (nodes and arcs) and the parameters
# (conditional probabilities of the nodes) of the network.

class Tutorial3:
    def __init__(self):
        print("Starting Tutorial3...")
        net = pysmile.Network()

        # load the network created by Tutorial1
        net.read_file("tutorial1.xdsl")
        for h in net.get_all_nodes():
            self.print_node_info(net, h)
        print("Tutorial3 complete.")


    def print_node_info(self, net, node_handle):
        print("Node id/name: " + net.get_node_id(node_handle) + "/" +
              net.get_node_name(node_handle))
        print("  Outcomes: " + " ".join(net.get_outcome_ids(node_handle)))

        parent_ids = net.get_parent_ids(node_handle)
        if len(parent_ids) > 0:
            print("  Parents: " + " ".join(parent_ids))
        child_ids = net.get_child_ids(node_handle)
        if len(child_ids) > 0:
            print("  Children: " + " ".join(child_ids))

        self.print_cpt_matrix(net, node_handle)


    def print_cpt_matrix(self, net, node_handle):
        cpt = net.get_node_definition(node_handle)
        parents = net.get_parents(node_handle)
        dim_count = 1 + len(parents)

        dim_sizes = [0] * dim_count
        for i in range(0, dim_count - 1):
```

```
            dim_sizes[i] = net.get_outcome_count(parents[i])
        dim_sizes[len(dim_sizes) - 1] = net.get_outcome_count(node_handle)

        coords = [0] * dim_count
        for elem_idx in range(0, len(cpt)):
            self.index_to_coords(elem_idx, dim_sizes, coords)

            outcome = net.get_outcome_id(node_handle, coords[dim_count - 1])
            out_str = "    P(" + outcome

            if dim_count > 1:
                out_str += " | "
                for parent_idx in range(0, len(parents)):
                    if parent_idx > 0:
                        out_str += ","
                    parent_handle = parents[parent_idx]
                    out_str += net.get_node_id(parent_handle) + "=" + \
                    net.get_outcome_id(parent_handle, coords[parent_idx])

            prob = cpt[elem_idx]
            out_str += ")=" + str(prob)
            print(out_str)


    def index_to_coords(self, index, dim_sizes, coords):
        prod = 1
        for i in range(len(dim_sizes) - 1, -1, -1):
            coords[i] = int(index / prod) % dim_sizes[i]
            prod *= dim_sizes[i]
```

6.3.3 Tutorial3.cs

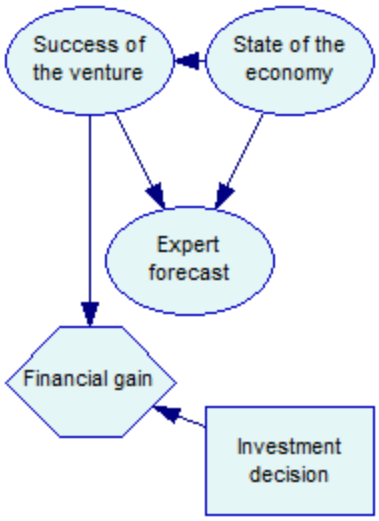
```
using System;
using Smile;

// Tutorial3 loads the XDSL file and prints the information
// about the structure (nodes and arcs) and the parameters
// (conditional probabilities of the nodes) of the network.

namespace SmileNetTutorial
{
    class Tutorial3
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial3...");
            Network net = new Network();

            // load the network created by Tutorial1
            net.ReadFile("tutorial1.xdsl");

            for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
            {
                PrintNodeInfo(net, h);
            }
```

```
    Console.WriteLine("Tutorial3 complete.");
}


private static void PrintNodeInfo(Network net, int nodeHandle)
{
    Console.WriteLine("Node id/name: {0}/{1}",
        net.GetNodeId(nodeHandle),
        net.GetNodeName(nodeHandle));

    Console.Write("  Outcomes:");
    foreach (String outcomeId in net.GetOutcomeIds(nodeHandle))
    {
        Console.Write(" " + outcomeId);
    }
    Console.WriteLine();

    String[] parentIds = net.GetParentIds(nodeHandle);
    if (parentIds.Length > 0)
    {
        Console.Write("  Parents:");
        foreach (String parentId in parentIds)
        {
            Console.Write(" " + parentId);
        }
        Console.WriteLine();
    }

    String[] childIds = net.GetChildIds(nodeHandle);
    if (childIds.Length > 0)
    {
        Console.Write("  Children:");
        foreach (String childId in childIds)
        {
            Console.Write(" " + childId);
        }
        Console.WriteLine();
    }

    PrintCptMatrix(net, nodeHandle);
}


private static void PrintCptMatrix(Network net, int nodeHandle)
{
    double[] cpt = net.GetNodeDefinition(nodeHandle);
    int[] parents = net.GetParents(nodeHandle);
    int dimCount = 1 + parents.Length;

    int[] dimSizes = new int[dimCount];
    for (int i = 0; i < dimCount - 1; i++)
    {
        dimSizes[i] = net.GetOutcomeCount(parents[i]);
    }
    dimSizes[dimSizes.Length - 1] = net.GetOutcomeCount(nodeHandle);

    int[] coords = new int[dimCount];
```

```
            for (int elemIdx = 0; elemIdx < cpt.Length; elemIdx++)
            {
                IndexToCoords(elemIdx, dimSizes, coords);

                String outcome =
                    net.GetOutcomeId(nodeHandle, coords[dimCount - 1]);
                Console.Write("    P({0}", outcome);

                if (dimCount > 1)
                {
                    Console.Write(" | ");
                    for (int pIdx = 0; pIdx < parents.Length; pIdx++)
                    {
                        if (pIdx > 0) Console.Write(",");
                        int parentHandle = parents[pIdx];
                        Console.Write("{0}={1}",
                            net.GetNodeId(parentHandle),
                            net.GetOutcomeId(parentHandle, coords[pIdx]));
                    }
                }

                double prob = cpt[elemIdx];
                Console.WriteLine(")={0}", prob);
            }
        }

        private static void IndexToCoords(
            int index, int[] dimSizes, int[] coords)
        {
            int prod = 1;
            for (int i = dimSizes.Length - 1; i >= 0; i--)
            {
                coords[i] = (index / prod) % dimSizes[i];
                prod *= dimSizes[i];
            }
        }

    }
}
```

6.4 Tutorial 4: Creating the Influence Diagram

We will further expand the model created in *Tutorial 1* [68] and turn it into an influence diagram. To this effect, we will add a decision node *Investment decision* and a utility node *Financial gain*. The decision will have two possible states: *Invest* and *DoNotInvest*, which will be the two decision options under consideration. Which option is chosen will impact the financial gain and this will be reflected by a directed arc from *Investment decision* to *Financial gain*. Whether the venture succeeds of fails will also impact the financial gain and this will be also reflected by a directed arc from *Success of the venture* to *Financial gain*.



We will show how to create this model. In the subsequent tutorial, we will show how to enter observations (evidence), how to perform inference, and how to retrieve the utilities calculated for the *Financial gain* node.

The programs starts by reading the file, just like *Tutorial 2* [77] and *Tutorial 3* [83]. We convert the identfier of the node *Success* to node handle, which we will use later to create an arc between *Success* and *Gain*. Two new nodes will be created by calling a `createNode` helper function, which is slightly modified version of the `createCptNode` from *Tutorial 1* [68]. The difference is that we now want to create different types of nodes. Therefore, `createNode` has one additional input parameter, an integer which specifies the node type. Another difference is that `createNode` needs to be able to add utility nodes, which do not have outcomes. The function checks for the value of its outcomes input parameter and if it is `null`, the outcome initialization is skipped.

createNode is called to add two new nodes:

- a decision node *Investment* , with a `Network.NodeType.DECISION` type identifier
- an utility node Gain, with a `Network.NodeType.UTILITY` type identifier

With new nodes created we connect them to the rest of the model with `Network.addArc`. The only thing left is the initialization of the parameters for the *Gain* node (*Invest* is a decision node and decision nodes have no numeric parameters). *Gain* has two parents with two outcomes each and size of its definition is 2 x 2 x 1 = 4. The program therefore specifies four numbers for the utilities.

Java:

```
double[] gainDefinition = new double[] {
    10000, // Utility(Invest=I, Success=S)
    -5000, // Utility(Invest=I, Success=F)
    500,   // Utility(Invest=D, Success=S)
    500    // Utility(Invest=D, Success=F)
};
```

```
net.setNodeDefinition(g, gainDefinition);
```

Python:

```
gain_definition = [
    10000, # Utility(Invest=I, Success=S)
    -5000, # Utility(Invest=I, Success=F)
    500,   # Utility(Invest=D, Success=S)
    500    # Utility(Invest=D, Success=F)
]
net.set_node_definition(g, gain_definition)
```

C#:

```
double[] gainDefinition = new double[]
{
    10000, // Utility(Invest=I, Success=S)
    -5000, // Utility(Invest=I, Success=F)
    500,   // Utility(Invest=D, Success=S)
    500    // Utility(Invest=D, Success=F)
};
net.SetNodeDefinition(g, gainDefinition);
```

The influence diagram is now complete, so we can write its contents to the file and exit the function. *Tutorial 5*[97] will load the file and perform the inference.

6.4.1 Tutorial4.java

```
package tutorials;

import smile.*;

// Tutorial4 loads the XDSL file file created by Tutorial1
// and adds decision and utility nodes, which transforms
// a Bayesian Network (BN) into an Influence Diagram (ID).

public class Tutorial4 {
    public static void run() {
        System.out.println("Starting Tutorial4...");
        Network net = new Network();

        net.readFile("tutorial1.xdsl");

        int s = net.getNode("Success");

        int i = createNode(net, Network.NodeType.DECISION,
                "Invest", "Investment decision",
                new String[]{ "Invest", "DoNotInvest" }, 160, 240);

        int g = createNode(net, Network.NodeType.UTILITY,
                "Gain", "Financial gain", null, 60, 200);

        net.addArc(i, g);
        net.addArc(s, g);

        double[] gainDefinition = new double[] {
            10000, // Utility(Invest=I, Success=S)
            -5000, // Utility(Invest=I, Success=F)
```

```
            500,   // Utility(Invest=D, Success=S)
            500    // Utility(Invest=D, Success=F)
        };
        net.setNodeDefinition(g, gainDefinition);

        net.writeFile("tutorial4.xdsl");

        System.out.println(
            "Tutorial4 complete: Influence diagram written to tutorial4.xdsl.");
    }


    private static int createNode(
            Network net, int nodeType, String id, String name,
            String[] outcomes, int xPos, int yPos) {
        int handle = net.addNode(nodeType, id);

        net.setNodeName(handle, name);
        net.setNodePosition(handle, xPos, yPos, 85, 55);

        if (outcomes != null) {
            int initialOutcomeCount = net.getOutcomeCount(handle);
            for (int i = 0; i < initialOutcomeCount; i ++) {
                net.setOutcomeId(handle, i, outcomes[i]);
            }

            for (int i = initialOutcomeCount; i < outcomes.length; i ++) {
                net.addOutcome(handle, outcomes[i]);
            }
        }

        return handle;
    }

}
```

6.4.2 Tutorial4.py

```
import pysmile

# Tutorial4 loads the XDSL file file created by Tutorial1
# and adds decision and utility nodes, which transforms
# a Bayesian Network (BN) into an Influence Diagram (ID).

class Tutorial4:
    def __init__(self):
        print("Starting Tutorial4...")
        net = pysmile.Network()

        net.read_file("tutorial1.xdsl")

        s = net.get_node("Success")
        i = self.create_node(net, pysmile.NodeType.DECISION,
            "Invest", "Investment decision",
            ["Invest", "DoNotInvest"], 160, 240)
```

```
        g = self.create_node(net, pysmile.NodeType.UTILITY,
            "Gain", "Financial gain", None, 60, 200)

        net.add_arc(i, g)
        net.add_arc(s, g)

        gain_definition = [
            10000, # Utility(Invest=I, Success=S)
            -5000, # Utility(Invest=I, Success=F)
            500,   # Utility(Invest=D, Success=S)
            500    # Utility(Invest=D, Success=F)
        ]
        net.set_node_definition(g, gain_definition)

        net.write_file("tutorial4.xdsl")

        print("Tutorial4 complete:" +
            " Influence diagram written to tutorial4.xdsl.")


    def create_node(self,
        net, node_type, id, name,
        outcomes, xPos, yPos):
        handle = net.add_node(node_type, id)

        net.set_node_name(handle, name)
        net.set_node_position(handle, xPos, yPos, 85, 55)

        if outcomes is not None:
            initial_outcome_count = net.get_outcome_count(handle)
            for i in range(0, initial_outcome_count):
                net.set_outcome_id(handle, i, outcomes[i])

            for i in range(initial_outcome_count, len(outcomes)):
                net.add_outcome(handle, outcomes[i])

        return handle
```

6.4.3 Tutorial4.cs

```
using System;
using Smile;

// Tutorial4 loads the XDSL file file created by Tutorial1
// and adds decision and utility nodes, which transforms
// a Bayesian Network (BN) into an Influence Diagram (ID).

namespace SmileNetTutorial
{
    class Tutorial4
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial4...");
            Network net = new Network();
```

```
            net.ReadFile("tutorial1.xdsl");

            int s = net.GetNode("Success");

            int i = CreateNode(net, Network.NodeType.List,
                    "Invest", "Investment decision",
                    new String[] { "Invest", "DoNotInvest" }, 160, 240);

            int g = CreateNode(net, Network.NodeType.Table,
                    "Gain", "Financial gain", null, 60, 200);

            net.AddArc(i, g);
            net.AddArc(s, g);

            double[] gainDefinition = new double[]
            {
                10000, // Utility(Invest=I, Success=S)
                -5000, // Utility(Invest=I, Success=F)
                500,   // Utility(Invest=D, Success=S)
                500    // Utility(Invest=D, Success=F)
            };
            net.SetNodeDefinition(g, gainDefinition);

            net.WriteFile("tutorial4.xdsl");

            Console.WriteLine(
                "Tutorial4 complete: ID written to tutorial4.xdsl.");
        }

        private static int CreateNode(
            Network net, Network.NodeType nodeType, String id, String name,
            String[] outcomes, int xPos, int yPos)
        {
            int handle = net.AddNode(nodeType, id);

            net.SetNodeName(handle, name);
            net.SetNodePosition(handle, xPos, yPos, 85, 55);

            if (outcomes != null)
            {
                int initialOutcomeCount = net.GetOutcomeCount(handle);
                for (int i = 0; i < initialOutcomeCount; i++)
                {
                    net.SetOutcomeId(handle, i, outcomes[i]);
                }

                for (int i = initialOutcomeCount; i < outcomes.Length; i++)
                {
                    net.AddOutcome(handle, outcomes[i]);
                }
            }

            return handle;
        }

    }
}
```

6.5 Tutorial 5: Inference in an Influence Diagram

This tutorial loads the influence diagram that we have created in *Tutorial 4* [91]. We will perform multiple inference calls and display calculated utilities.

With model loaded, we calculate the probabilities and utilities by calling `Network.updateBeliefs`. A helper function, `printFinancialGain`, is called to print out the utilities stored in the *Gain* node. `printFinancialGain` uses `Network.getNodeValue` to obtain the utilities. To properly interpret the utilities, we also need to call `Network.getValueIndexingParents`, which is an array containing the handles of decision nodes not set to evidence.

Java:

```
double[] expectedUtility = net.getNodeValue("Gain");
int[] utilParents = net.getValueIndexingParents("Gain");
printGainMatrix(net, expectedUtility, utilParents);
```

Python:

```
expected_utility = net.get_node_value("Gain")
util_parents = net.get_value_indexing_parents("Gain")
self.print_gain_matrix(net, expected_utility, util_parents)
```

C#:

```
double[] expectedUtility = net.GetNodeValue("Gain");
int[] utilParents = net.GetValueIndexingParents("Gain");
printGainMatrix(net, expectedUtility, utilParents);
```

`printGainMatrix` is a modification of `printCptMatrix` from *Tutorial 3* [83]. It iterates over the elements of an array and converts the linear index into multi-dimensional coordinates.

When we call `printGainMatrix` for the first time, the network has no evidence set. The utilities calculated without evidence suggest that we should not invest - here's what is printed on the screen:

```
Financial gain:
   Utility(Invest=Invest)=-1850
   Utility(Invest=DoNotInvest)=500
```

Next, we model the analyst's forecast to be good by calling local helper function `changeEvidenceAndUpdate`, which is based on the function with the same name from *Tutorial 2* [77].

Java:

```
changeEvidenceAndUpdate(net, "Forecast", "Good");
```

Python:

```
self.change_evidence_and_update(net, "Forecast", "Good")
```

C#:

```
ChangeEvidenceAndUpdate(net, "Forecast", "Good");
```

The expected gain changes, now the optimal decision is to invest:

```
Financial gain:
   Utility(Invest=Invest)=4455.78
   Utility(Invest=DoNotInvest)=500
```

Now we observe the state of the economy and conclude that it is growing - the program performs another `changeEvidenceAndUpdate` call . Growing economy makes our chances even better:

```
Financial gain:
   Utility(Invest=Invest)=5000
   Utility(Invest=DoNotInvest)=500
```

This concludes Tutorial 5.

6.5.1 Tutorial5.java

```
package tutorials;

import smile.*;

// Tutorial5 loads the XDSL file created by Tutorial4,
// then performs the series of inference calls,
// changing evidence each time.

public class Tutorial5 {
    public static void run() {
        System.out.println("Starting Tutorial5...");
        Network net = new Network();

        net.readFile("tutorial4.xdsl");

        System.out.println("No evidence set.");
        net.updateBeliefs();
        printFinancialGain(net);

        System.out.println("Setting Forecast=Good.");
        changeEvidenceAndUpdate(net, "Forecast", "Good");

        System.out.println("Adding Economy=Up");
        changeEvidenceAndUpdate(net, "Economy", "Up");

        System.out.println("Tutorial5 complete.");
    }


    static void changeEvidenceAndUpdate(
            Network net, String nodeId, String outcomeId) {
        if (outcomeId != null) {
            net.setEvidence(nodeId, outcomeId);
        } else {
            net.clearEvidence(nodeId);
        }

        net.updateBeliefs();
        printFinancialGain(net);
```

```
    }


    static void printFinancialGain(Network net) {
        double[] expectedUtility = net.getNodeValue("Gain");
        int[] utilParents = net.getValueIndexingParents("Gain");
        printGainMatrix(net, expectedUtility, utilParents);
    }


    static void printGainMatrix(Network net, double[] mtx, int[] parents) {
        int dimCount = 1 + parents.length;

        int[] dimSizes = new int[dimCount];
        for (int i = 0; i < dimCount - 1; i ++) {
            dimSizes[i] = net.getOutcomeCount(parents[i]);
        }
        dimSizes[dimSizes.length - 1] = 1;

        int[] coords = new int[dimCount];
        for (int elemIdx = 0; elemIdx < mtx.length; elemIdx ++) {
            indexToCoords(elemIdx, dimSizes, coords);

            System.out.print("    Utility(");

            if (dimCount > 1)
            {
                for (int parentIdx = 0; parentIdx < parents.length; parentIdx++)
                {
                    if (parentIdx > 0) System.out.print(",");
                    int parentHandle = parents[parentIdx];
                    System.out.printf("%s=%s",
                        net.getNodeId(parentHandle),
                        net.getOutcomeId(parentHandle, coords[parentIdx]));
                }
            }

            System.out.printf(")=%f\n", mtx[elemIdx]);
        }
        System.out.println();
    }


    static void indexToCoords(int index, int[] dimSizes, int[] coords) {
        int prod = 1;
        for (int i = dimSizes.length - 1; i >= 0; i --) {
            coords[i] = (index / prod) % dimSizes[i];
            prod *= dimSizes[i];
        }
    }

}
```

6.5.2 Tutorial5.py

```
import pysmile

# Tutorial5 loads the XDSL file created by Tutorial4,
# then performs the series of inference calls,
# changing evidence each time.

class Tutorial5:
    def __init__(self):
        print("Starting Tutorial5...")
        net = pysmile.Network()

        net.read_file("tutorial4.xdsl")

        print("No evidence set.")
        net.update_beliefs()
        self.print_financial_gain(net)

        print("Setting Forecast=Good.")
        self.change_evidence_and_update(net, "Forecast", "Good")

        print("Adding Economy=Up")
        self.change_evidence_and_update(net, "Economy", "Up")

        print("Tutorial5 complete.")


    def change_evidence_and_update(self, net, node_id, outcome_id):
        if outcome_id is not None:
            net.set_evidence(node_id, outcome_id)
        else:
            net.clear_evidence(node_id)

        net.update_beliefs()
        self.print_financial_gain(net)


    def print_financial_gain(self, net):
        expected_utility = net.get_node_value("Gain")
        util_parents = net.get_value_indexing_parents("Gain")
        self.print_gain_matrix(net, expected_utility, util_parents)


    def print_gain_matrix(self, net, mtx, parents):
        dim_count = 1 + len(parents)

        dim_sizes = [0] * dim_count
        for i in range(0, dim_count - 1):
            dim_sizes[i] = net.get_outcome_count(parents[i])
        dim_sizes[len(dim_sizes) - 1] = 1
        coords = [0] * dim_count
        for elem_idx in range(0, len(mtx)):
            self.index_to_coords(elem_idx, dim_sizes, coords)
            str_to_print = "    Utility("
            if dim_count > 1:
                for parent_idx in range(0, len(parents)):
```

```
                    if parent_idx > 0:
                        str_to_print += ","
                    parent_handle = parents[parent_idx]
                    str_to_print += net.get_node_id(parent_handle) + \
                    "=" + net.get_outcome_id(parent_handle,
                                             coords[parent_idx])
            str_to_print += ")=" + str(mtx[elem_idx])
            print(str_to_print)
        print("")


    def index_to_coords(self, index, dim_sizes, coords):
        prod = 1
        for i in range(len(dim_sizes) - 1, -1, -1):
            coords[i] = int(index / prod) % dim_sizes[i]
            prod *= dim_sizes[i]
```

6.5.3 Tutorial5.cs

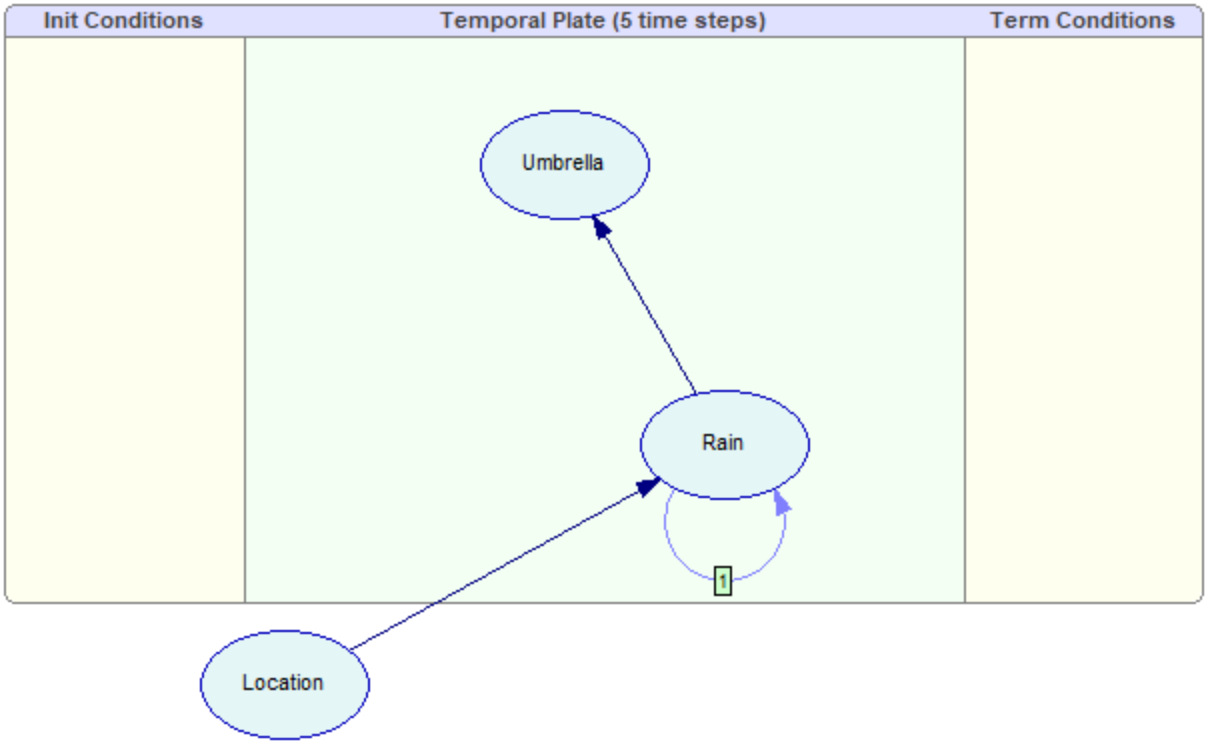
```
using System;
using Smile;

// Tutorial5 loads the XDSL file created by Tutorial4,
// then performs the series of inference calls,
// changing evidence each time.

namespace SmileNetTutorial
{
    class Tutorial5
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial5...");
            Network net = new Network();

            net.ReadFile("tutorial4.xdsl");

            Console.WriteLine("No evidence set.");
            net.UpdateBeliefs();
            PrintFinancialGain(net);

            Console.WriteLine("Setting Forecast=Good.");
            ChangeEvidenceAndUpdate(net, "Forecast", "Good");

            Console.WriteLine("Adding Economy=Up");
            ChangeEvidenceAndUpdate(net, "Economy", "Up");

            Console.WriteLine("Tutorial5 complete.");
        }


        static void ChangeEvidenceAndUpdate(
            Network net, String nodeId, String outcomeId)
        {
            if (outcomeId != null)
            {
```

```
            net.SetEvidence(nodeId, outcomeId);
        }
        else
        {
            net.ClearEvidence(nodeId);
        }

        net.UpdateBeliefs();
        PrintFinancialGain(net);
    }


    static void PrintFinancialGain(Network net)
    {
        double[] expectedUtility = net.GetNodeValue("Gain");
        int[] utilParents = net.GetValueIndexingParents("Gain");
        printGainMatrix(net, expectedUtility, utilParents);
    }


    static void printGainMatrix(Network net, double[] mtx, int[] parents)
    {
        int dimCount = 1 + parents.Length;

        int[] dimSizes = new int[dimCount];
        for (int i = 0; i < dimCount - 1; i++)
        {
            dimSizes[i] = net.GetOutcomeCount(parents[i]);
        }
        dimSizes[dimSizes.Length - 1] = 1;

        int[] coords = new int[dimCount];
        for (int elemIdx = 0; elemIdx < mtx.Length; elemIdx++)
        {
            IndexToCoords(elemIdx, dimSizes, coords);

            Console.Write("    Utility(");

            if (dimCount > 1)
            {
                for (int pIdx = 0; pIdx < parents.Length; pIdx++)
                {
                    if (pIdx > 0) Console.Write(",");
                    int parentHandle = parents[pIdx];
                    Console.Write("{0}={1}",
                        net.GetNodeId(parentHandle),
                        net.GetOutcomeId(parentHandle, coords[pIdx]));
                }
            }

            Console.Write(")={0}\n", mtx[elemIdx]);
        }
        Console.WriteLine();
    }


    static void IndexToCoords(int index, int[] dimSizes, int[] coords)
```

```
        {
            int prod = 1;
            for (int i = dimSizes.Length - 1; i >= 0; i--)
            {
                coords[i] = (index / prod) % dimSizes[i];
                prod *= dimSizes[i];
            }
        }

    }
}
```

6.6 Tutorial 6: Dynamic model

Consider the following example, inspired by (Russell & Norvig, 1995), in which a security guard at some secret underground installation works on a shift of seven days and wants to know whether it is raining on the day of her return to the outside world. Her only access to the outside world occurs each morning when she sees the director coming in, with or without an umbrella. Furthermore, she knows that the government has two secret underground installations: one in Pittsburgh and one in the Sahara, but she does not know which one she is guarding. For each day *t*, the set of evidence contains a single variable $Umbrella_t$ (observation of an umbrella carried by the director) and the set of unobservable variables contains $Rain_t$ (a propositional variable with two states *true* and *false*, denoting whether it is raining) and *Location* (with two possible states: *Pittsburgh* and *Sahara*). The prior probability of rain depends on the geographical location and on whether it rained on the previous day. For simplicity, we do not use the initial and terminal condition nodes in this tutorial.



The model contains three discrete chance nodes (CPT), created with a call to `createCptNode` helper function, just like in nodes in *Tutorial 1* [68]. The *Rain* and *Umbrella* nodes are marked as belonging to the temporal plate by calling `Network.SetNodeTemporalType`:

Java:

```
net.setNodeTemporalType(rain, Network.NodeTemporalType.PLATE);
net.setNodeTemporalType(umb, Network.NodeTemporalType.PLATE);
```

Python:

```
net.set_node_temporal_type(rain, pysmile.NodeTemporalType.PLATE)
net.set_node_temporal_type(umb, pysmile.NodeTemporalType.PLATE)
```

C#:

```
net.SetNodeTemporalType(rain, Network.NodeTemporalType.Plate);
net.SetNodeTemporalType(umb, Network.NodeTemporalType.Plate);
```

Two of the three arcs in the model are created by `Network.addArc`. The temporal arc expressing the dependency of $Rain_t$ on $Rain_{t-1}$ is created by the `Network.addTemporalArc` method. Please note the temporal order of the arc passed as the 3rd parameter:

Java:

```
net.addArc(loc, rain);
net.addTemporalArc(rain, rain, 1);
net.addArc(rain, umb);
```

Python:

```
net.add_arc(loc, rain)
net.add_temporal_arc(rain, rain, 1)
net.add_arc(rain, umb)
```

C#:

```
net.AddArc(loc, rain);
net.AddTemporalArc(rain, rain, 1);
net.AddArc(rain, umb);
```

CPTs for the nodes are initialized with `Network.setNodeDefinition`, as in *Tutorial 1* [68]. However, the node Rain requires **two** CPTs, because it has an incoming temporal arc of order 1. The additional CPT is set by the call to Network.setNodeTemporalDefinition:

Java:

```
double[] rainDefTemporal = new double[] {
    0.7,   // P(Rain=true |Location=Pittsburgh,Rain[t-1]=true)
    0.3,   // P(Rain=false|Location=Pittsburgh,Rain[t-1]=true)
    0.3,   // P(Rain=true |Location=Pittsburgh,Rain[t-1]=false)
    0.7,   // P(Rain=false|Location=Pittsburgh,Rain[t-1]=false)
    0.001, // P(Rain=true |Location=Sahara,Rain[t-1]=true)
    0.999, // P(Rain=false|Location=Sahara,Rain[t-1]=true)
    0.01,  // P(Rain=true |Location=Sahara,Rain[t-1]=false)
    0.99  // P(Rain=false|Location=Sahara,Rain[t-1]=false)
};
net.setNodeTemporalDefinition(rain, 1, rainDefTemporal);
```

Python:

```
rain_def_temporal = [
    # probabilities go here
]
net.set_node_temporal_definition(rain, 1, rain_def_temporal)
```

C#:

```
double[] rainDefTemporal = new double[]
{
    // probabilities go here
};
net.SetNodeTemporalDefinition(rain, 1, rainDefTemporal);
```

Finally, we adjust the number of slices created during the network unrolling with a call to `Network.setSliceCount`. The network is complete. The program proceeds to the inference, first without any evidence in the network, then with two observations of the *Umbrella*, set in the time steps *t=1* and *t=3*:

Java:

```
net.setTemporalEvidence(umb, 1, 0);
net.setTemporalEvidence(umb, 3, 1);
```

Python:

```
net.set_temporal_evidence(umb, 1, 0)
net.set_temporal_evidence(umb, 3, 1
```

C#:

```
net.SetTemporalEvidence(umb, 1, 0);
net.SetTemporalEvidence(umb, 3, 1);
```

In dynamic Bayesian networks, the plate nodes have their beliefs calculated for each time slice. The number of elements in the node value matrix for these nodes is a product of outcome count and slice count. The helper function `updateAndShowTemporalResults` iterates over this matrix using two nested loops in order to print the results. Since the probabilities for the same slice are adjacent, the inner loop uses the product of the outcome count and slice index from the outer loop as a base index.

Java:

```
double[] v = net.getNodeValue(h);
for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++) {
    System.out.printf("\tt=%d:", sliceIdx);
    for (int i = 0; i < outcomeCount; i++) {
           System.out.printf(" %f", v[sliceIdx * outcomeCount + i]);
    }
    System.out.println();
}
```

Python:

```
v = net.get_node_value(h)
for slice_idx in range(0, slice_count):
    s = "\tt=" + str(slice_idx) + ":"
    for i in range(0, outcome_count):
        s = s + " " + str(v[slice_idx * outcome_count + i])
    print(s)
```

C#:

```
double[] v = net.GetNodeValue(h);
for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++)
{
    Console.Write("\tt={0}:", sliceIdx);
    for (int i = 0; i < outcomeCount; i++)
    {
           Console.Write(" {0}", v[sliceIdx * outcomeCount + i]);
    }
    Console.WriteLine();
}
```

6.6.1 Tutorial6.java

```
package tutorials;

import smile.*;

// Tutorial6 creates a dynamic Bayesian network (DBN),
```

```
// performs the inference, then saves the model to disk.

public class Tutorial6 {
    public static void run() {
        System.out.println("Starting Tutorial6...");
        Network net = new Network();

        int loc = createCptNode(
              net, "Location", "Location",
              new String[] { "Pittsburgh", "Sahara" },
              160, 360);

        int rain = createCptNode(
              net, "Rain", "Rain",
              new String[] { "true", "false" },
              380, 240);

        int umb = createCptNode(
              net, "Umbrella", "Umbrella",
              new String[] { "true", "false" },
              300, 100);

        net.setNodeTemporalType(rain, Network.NodeTemporalType.PLATE);
        net.setNodeTemporalType(umb, Network.NodeTemporalType.PLATE);

        net.addArc(loc, rain);
        net.addTemporalArc(rain, rain, 1);
        net.addArc(rain, umb);

        double[] rainDef = new double[] {
            0.7,  // P(Rain=true |Location=Pittsburgh)
            0.3,  // P(Rain=false|Location=Pittsburgh)
            0.01, // P(Rain=true |Location=Sahara)
            0.99  // P(Rain=false|Location=Sahara)
        };
        net.setNodeDefinition(rain, rainDef);

        double[] rainDefTemporal = new double[] {
            0.7,   // P(Rain=true |Location=Pittsburgh,Rain[t-1]=true)
            0.3,   // P(Rain=false|Location=Pittsburgh,Rain[t-1]=true)
            0.3,   // P(Rain=true |Location=Pittsburgh,Rain[t-1]=false)
            0.7,   // P(Rain=false|Location=Pittsburgh,Rain[t-1]=false)
            0.001, // P(Rain=true |Location=Sahara,Rain[t-1]=true)
            0.999, // P(Rain=false|Location=Sahara,Rain[t-1]=true)
            0.01,  // P(Rain=true |Location=Sahara,Rain[t-1]=false)
            0.99  // P(Rain=false|Location=Sahara,Rain[t-1]=false)
        };
        net.setNodeTemporalDefinition(rain, 1, rainDefTemporal);

        double[] umbDef = new double[] {
              0.9, // P(Umbrella=true |Rain=true)
            0.1, // P(Umbrella=false|Rain=true)
            0.2, // P(Umbrella=true |Rain=false)
            0.8  // P(Umbrella=false|Rain=false)
        };
        net.setNodeDefinition(umb, umbDef);
```

```
    net.setSliceCount(5);

    System.out.println("Performing update without evidence.");
    updateAndShowTemporalResults(net);

    System.out.println(
        "Setting Umbrella[t=1] to true and Umbrella[t=3] to false.");
    net.setTemporalEvidence(umb, 1, 0);
    net.setTemporalEvidence(umb, 3, 1);
    updateAndShowTemporalResults(net);

    net.writeFile("tutorial6.xdsl");
    System.out.println(
        "Tutorial6 complete: Network written to tutorial6.xdsl");
}


private static void updateAndShowTemporalResults(Network net) {
   net.updateBeliefs();
       int sliceCount = net.getSliceCount();
       for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h)) {
          if (net.getNodeTemporalType(h) == Network.NodeTemporalType.PLATE) {
                int outcomeCount = net.getOutcomeCount(h);
              System.out.printf(
               "Temporal beliefs for %s:\n", net.getNodeId(h));
              double[] v = net.getNodeValue(h);
                       for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++) {
                              System.out.printf("\tt=%d:", sliceIdx);
                              for (int i = 0; i < outcomeCount; i++) {
                                  System.out.printf(" %f", v[sliceIdx*outcomeCount+i]);
                              }
                              System.out.println();
                       }
          }
       }
       System.out.println();
}


private static int createCptNode(
        Network net, String id, String name,
        String[] outcomes, int xPos, int yPos) {
    int handle = net.addNode(Network.NodeType.CPT, id);

    net.setNodeName(handle, name);
    net.setNodePosition(handle, xPos, yPos, 85, 55);

    int initialOutcomeCount = net.getOutcomeCount(handle);
    for (int i = 0; i < initialOutcomeCount; i ++) {
        net.setOutcomeId(handle, i, outcomes[i]);
    }

    for (int i = initialOutcomeCount; i < outcomes.length; i ++) {
        net.addOutcome(handle, outcomes[i]);
    }

    return handle;
```

```
    }

}
```

6.6.2 Tutorial6.py

```
import pysmile

# Tutorial6 creates a dynamic Bayesian network (DBN),
# performs the inference, then saves the model to disk.

class Tutorial6:
    def __init__(self):
        print("Starting Tutorial6...")
        net = pysmile.Network()
        loc = self.create_cpt_node(net,
              "Location", "Location",
              ["Pittsburgh", "Sahara"],
              160, 360)
        rain = self.create_cpt_node(net,
               "Rain", "Rain",
               ["true", "false"],
               380, 240)
        umb = self.create_cpt_node(net,
              "Umbrella", "Umbrella",
              ["true", "false"],
              300, 100)

        net.set_node_temporal_type(rain, pysmile.NodeTemporalType.PLATE)
        net.set_node_temporal_type(umb, pysmile.NodeTemporalType.PLATE)

        net.add_arc(loc, rain)
        net.add_temporal_arc(rain, rain, 1)
        net.add_arc(rain, umb)

        rain_def = [
            0.7,  # P(Rain=true |Location=Pittsburgh)
            0.3,  # P(Rain=false|Location=Pittsburgh)
            0.01, # P(Rain=true |Location=Sahara)
            0.99  # P(Rain=false|Location=Sahara)
        ]

        net.set_node_definition(rain, rain_def)

        rain_def_temporal = [
            0.7,   # P(Rain=true |Location=Pittsburgh,Rain[t-1]=true)
            0.3,   # P(Rain=false|Location=Pittsburgh,Rain[t-1]=true)
            0.3,   # P(Rain=true |Location=Pittsburgh,Rain[t-1]=false)
            0.7,   # P(Rain=false|Location=Pittsburgh,Rain[t-1]=false)
            0.001, # P(Rain=true |Location=Sahara,Rain[t-1]=true)
            0.999, # P(Rain=false|Location=Sahara,Rain[t-1]=true)
            0.01,  # P(Rain=true |Location=Sahara,Rain[t-1]=false)
            0.99   # P(Rain=false|Location=Sahara,Rain[t-1]=false)
        ]
        net.set_node_temporal_definition(rain, 1, rain_def_temporal)
```

```
    umb_def = [
        0.9, # P(Umbrella=true |Rain=true)
        0.1, # P(Umbrella=false|Rain=true)
        0.2, # P(Umbrella=true |Rain=false)
        0.8  # P(Umbrella=false|Rain=false)
    ]
    net.set_node_definition(umb, umb_def)

    net.set_slice_count(5)

    print("Performing update without evidence.")
    self.update_and_show_temporal_results(net)

    print("Setting Umbrella[t=1], to true and Umbrella[t=3] to false.")
    net.set_temporal_evidence(umb, 1, 0)
    net.set_temporal_evidence(umb, 3, 1)
    self.update_and_show_temporal_results(net)

    net.write_file("tutorial6.xdsl")
    print("Tutorial6 complete: Network written to tutorial6.xsdl")


def update_and_show_temporal_results(self, net):
    net.update_beliefs()
    slice_count = net.get_slice_count()
    for h in net.get_all_nodes():
        if net.get_node_temporal_type(h) == pysmile.NodeTemporalType.PLATE:
            outcome_count = net.get_outcome_count(h)
            print("Temporal beliefs for " + net.get_node_id(h) + ":")
            v = net.get_node_value(h)
            for slice_idx in range(0, slice_count):
                s = "\tt=" + str(slice_idx) + ":"
                for i in range(0, outcome_count):
                    s = s + " " + str(v[slice_idx * outcome_count + i])
                print(s)
    print("")


def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.CPT, id)

    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)

    initial_outcome_count = net.get_outcome_count(handle)

    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])

    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])

    return handle
```

6.6.3 Tutorial6.cs

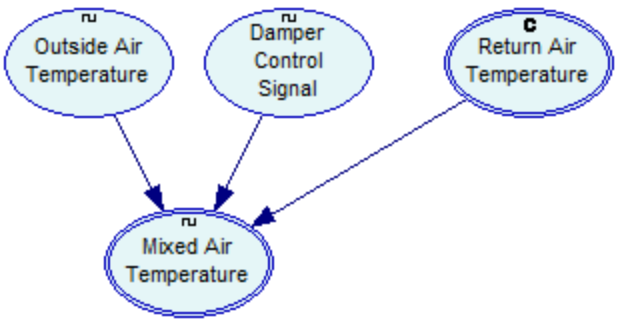
```
using System;
using Smile;

// Tutorial6 creates a dynamic Bayesian network (DBN),
// performs the inference, then saves the model to disk.

namespace SmileNetTutorial
{
    class Tutorial6
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial6...");
            Network net = new Network();

            int loc = CreateCptNode(
                net, "Location", "Location",
                new String[] { "Pittsburgh", "Sahara" },
                160, 360);

            int rain = CreateCptNode(
                net, "Rain", "Rain",
                new String[] { "true", "false" },
                380, 240);

            int umb = CreateCptNode(
                net, "Umbrella", "Umbrella",
                new String[] { "true", "false" },
                300, 100);

            net.SetNodeTemporalType(rain, Network.NodeTemporalType.Plate);
            net.SetNodeTemporalType(umb, Network.NodeTemporalType.Plate);

            net.AddArc(loc, rain);
            net.AddTemporalArc(rain, rain, 1);
            net.AddArc(rain, umb);

            double[] rainDef = new double[]
            {
                0.7,  // P(Rain=true |Location=Pittsburgh)
                0.3,  // P(Rain=false|Location=Pittsburgh)
                0.01, // P(Rain=true |Location=Sahara)
                0.99  // P(Rain=false|Location=Sahara)
            };
            net.SetNodeDefinition(rain, rainDef);

            double[] rainDefTemporal = new double[]
            {
                0.7,   // P(Rain=true |Location=Pittsburgh,Rain[t-1]=true)
                0.3,   // P(Rain=false|Location=Pittsburgh,Rain[t-1]=true)
                0.3,   // P(Rain=true |Location=Pittsburgh,Rain[t-1]=false)
                0.7,   // P(Rain=false|Location=Pittsburgh,Rain[t-1]=false)
                0.001, // P(Rain=true |Location=Sahara,Rain[t-1]=true)
                0.999, // P(Rain=false|Location=Sahara,Rain[t-1]=true)
                0.01,  // P(Rain=true |Location=Sahara,Rain[t-1]=false)
```

```
            0.99  // P(Rain=false|Location=Sahara,Rain[t-1]=false)
        };
        net.SetNodeTemporalDefinition(rain, 1, rainDefTemporal);

        double[] umbDef = new double[]
        {
            0.9, // P(Umbrella=true |Rain=true)
            0.1, // P(Umbrella=false|Rain=true)
            0.2, // P(Umbrella=true |Rain=false)
            0.8  // P(Umbrella=false|Rain=false)
        };
        net.SetNodeDefinition(umb, umbDef);

        net.SetSliceCount(5);

        Console.WriteLine("Performing update without evidence.");
        UpdateAndShowTemporalResults(net);

        Console.WriteLine(
            "Setting Umbrella[t=1] to true and Umbrella[t=3] to false.");
        net.SetTemporalEvidence(umb, 1, 0);
        net.SetTemporalEvidence(umb, 3, 1);
        UpdateAndShowTemporalResults(net);

        net.WriteFile("tutorial6.xdsl");
        Console.WriteLine(
            "Tutorial6 complete: Network written to tutorial6.xdsl");
    }


    private static void UpdateAndShowTemporalResults(Network net)
    {
        net.UpdateBeliefs();
        int sliceCount = net.GetSliceCount();
        for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
        {
            if (net.GetNodeTemporalType(h) ==
                Network.NodeTemporalType.Plate)
            {
                int outcomeCount = net.GetOutcomeCount(h);
                Console.WriteLine(
                    "Temporal beliefs for {0}:", net.GetNodeId(h));
                double[] v = net.GetNodeValue(h);
                for (int sliceIdx = 0; sliceIdx < sliceCount; sliceIdx++)
                {
                    Console.Write("\tt={0}:", sliceIdx);
                    for (int i = 0; i < outcomeCount; i++)
                    {
                        Console.Write(
                            " {0}", v[sliceIdx * outcomeCount + i]);
                    }
                    Console.WriteLine();
                }
            }
        }
        Console.WriteLine();
    }
```

```
        private static int CreateCptNode(
            Network net, String id, String name,
            String[] outcomes, int xPos, int yPos)
        {
            int handle = net.AddNode(Network.NodeType.Cpt, id);

            net.SetNodeName(handle, name);
            net.SetNodePosition(handle, xPos, yPos, 85, 55);

            int initialOutcomeCount = net.GetOutcomeCount(handle);
            for (int i = 0; i < initialOutcomeCount; i++)
            {
                net.SetOutcomeId(handle, i, outcomes[i]);
            }

            for (int i = initialOutcomeCount; i < outcomes.Length; i++)
            {
                net.AddOutcome(handle, outcomes[i]);
            }

            return handle;
        }
    }

}
```

6.7 Tutorial 7: Continuous model

The continuous Bayesian network used in this tutorial focuses on a fragment of a forced air heating and cooling system. In order to improve the system's efficiency, return air is mixed with the air that is drawn from outside. Temperature of the outside air depends on the weather and is specified by means of a Normal distribution. Return air temperature is constant and depends on the thermostat setting. Damper control signal determines the composition of the mixture.



Temperature of the mixture is calculated according to the following equation: *tma=toa*u_d+(tra-tra*u_d)*, where *tma* is mixed air temperature, *toa* is outside air temperature, *u_d* is the damper signal, and *tra* is return air temperature.

The nodes in this model are created by the helper function `createEquationNode`. It is a modified version of the `createCptNode` function used in the previous tutorials. The bold text marks the difference between two functions.

```
static int createEquationNode(
        Network net, String id, String name,
        String equation, double loBound, double hiBound,
        int xPos, int yPos) {
    int handle = net.addNode(Network.NodeType.EQUATION, id);
    net.setNodeName(handle, name);
    net.setNodeEquation(handle, equation);
    net.setNodeEquationBounds(handle, loBound, hiBound);

    net.setNodePosition(handle, xPos, yPos, 85, 55);

    return handle;
}
```

Instead of discrete outcomes, we specify the node equation and the bounds for the node value. Another helper function, `setUniformIntervals`, is used to define the discretization intervals. They will be used by the inference algorithm when the evidence is set for *Mixed Air Temperature* node (which has parents). The uniform intervals are chosen for simplicity here; in general case the choice of interval edges should be done based on the actual expected distribution of the node value (for example, in case of the Normal distribution, we might create narrow discretization intervals close to the mean.)

Note that while the model has three arcs, there are no calls to `Network.addArc` in this tutorial. The arcs are created implicitly by `Network.setNodeEquation` method (called by `createEquationNode` function).

The network is complete now and we can proceed to inference. Three inference calls are made, one without evidence and two with continuous evidence specified by calling `Network.setContEvidence`. Setting the *Outside Air Temperature* to 28.5 degrees (`toa` is the name of int variable holding the handle of the *Outside Air Temperature* node):

Java:

```
net.setContEvidence(toa, 28.5);
```

Python:

```
net.set_cont_evidence(toa, 28.5)
```

C#:

```
net.SetContEvidence(toa, 28.5);
```

The program uses `updateAndShowStats` helper function for inference. The helper calls `Network.updateBeliefs` and iterates over the nodes in the network and calls another helper, `showStats`, for each node. `showStats` first checks if the node has evidence set. If it does, the evidence value is printed, and the function returns. If node has no evidence, we need to check if its value comes from the sampling or discretized inference.

If discretization was used, `Network.isValueDiscretized` returns true. In such case, the array returned from `Network.getNodeValue` contains the probability distribution over node discretization intervals. To display human-readable information about the intervals and probabilities, `showStats` calls `Network.getNodeEquationDiscretization` and `getNodeEquationBounds`. Note that the program setup these node attributes in `createEquationNode`.

If node value was sampled, `Network.isValueDiscretized` returns false. `Network.getNodeValue` returns the array of sampled node values. After sampling, it's also possible to obtain sample statistics (mean, standard deviation, min and max values) with a call to `Network.getNodeSampleStats`.

At the end of the tutorial, the model is saved to disk. *Tutorial 8*[122] will expand it into a hybrid network by adding CPT nodes.

6.7.1 Tutorial7.java

```
package tutorials;

import smile.*;

// Tutorial7 creates a network with three equation-based nodes
// performs the inference, then saves the model to disk.

public class Tutorial7 {
    public static void run() {
        System.out.println("Starting Tutorial7...");
        Network net = new Network();

        net.setOutlierRejectionEnabled(true);

        createEquationNode(net,
            "tra", "Return Air Temperature",
            "tra=24", 23.9, 24.1,
            280, 100);

        createEquationNode(net,
            "u_d", "Damper Control Signal",
            "u_d = Bernoulli(0.539)*0.8 + 0.2", 0, 1,
            160, 100);

        int toa = createEquationNode(net,
            "toa", "Outside Air Temperature",
            "toa=Normal(11,15)", -10, 40,
            60, 100);

        // tra, toa and u_d are referenced in equation
        // arcs are created automatically
        int tma = createEquationNode(net,
            "tma", "Mixed Air Temperature",
            "tma=toa*u_d+(tra-tra*u_d)", 10, 30,
```

```
        110, 200);

    setUniformIntervals(net, toa, 5);
    setUniformIntervals(net, tma, 4);

    System.out.println("Results with no evidence:");
    updateAndShowStats(net);

    net.setContEvidence(toa, 28.5);
    System.out.println(
        "Results with outside air temperature set to 28.5:");
    updateAndShowStats(net);

    net.clearEvidence(toa);
    System.out.println(
        "Results with mixed air temperature set to 21:");
    net.setContEvidence(tma, 21.0);
    updateAndShowStats(net);

    net.writeFile("tutorial7.xdsl");
    System.out.println(
        "Tutorial7 complete: Network written to tutorial7.xdsl");
}


static int createEquationNode(
       Network net, String id, String name,
       String equation, double loBound, double hiBound,
       int xPos, int yPos) {
          int handle = net.addNode(Network.NodeType.EQUATION, id);
   net.setNodeName(handle, name);
   net.setNodeEquation(handle, equation);
   net.setNodeEquationBounds(handle, loBound, hiBound);

   net.setNodePosition(handle, xPos, yPos, 85, 55);

   return handle;
}


static void showStats(Network net, int nodeHandle) {
          String nodeId = net.getNodeId(nodeHandle);

          if (net.isEvidence(nodeHandle)) {
                 double v = net.getContEvidence(nodeHandle);
                 System.out.printf("%s has evidence set (%g)\n", nodeId, v);
                 return;
          }

          if (net.isValueDiscretized(nodeHandle)) {
                 System.out.printf("%s is discretized.\n", nodeId);
                 DiscretizationInterval[] iv =
            net.getNodeEquationDiscretization(nodeHandle);
                 double[] bounds = net.getNodeEquationBounds(nodeHandle);
                 double[] discBeliefs = net.getNodeValue(nodeHandle);
                 double lo = bounds[0];
              for (int i = 0; i < discBeliefs.length; i++) {
```

```
                    double hi = iv[i].boundary;
                    System.out.printf(
                  "\tP(%s in %g..%g)=%g\n", nodeId, lo, hi, discBeliefs[i]);
                    lo = hi;
                }
            } else {
                   double[] stats = net.getNodeSampleStats(nodeHandle);
             System.out.printf("%s: mean=%g stddev=%g min=%g max=%g\n",
                 nodeId, stats[0], stats[1], stats[2], stats[3]);
            }
    }


    static void setUniformIntervals(Network net, int nodeHandle, int count) {
       double[] bounds = net.getNodeEquationBounds(nodeHandle);
       double lo = bounds[0];
       double hi = bounds[1];

       DiscretizationInterval[] iv = new DiscretizationInterval[count];
        for (int i = 0; i < count; i++) {
            iv[i] = new DiscretizationInterval(
                null, lo + (i + 1) * (hi - lo) / count);
        }

        net.setNodeEquationDiscretization(nodeHandle, iv);
    }


    static void updateAndShowStats(Network net) {
        net.updateBeliefs();
        for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h))
        {
            showStats(net, h);
        }
        System.out.println();
    }

}
```

6.7.2 Tutorial7.py

```
import pysmile

# Tutorial7 creates a network with three equation-based nodes
# performs the inference, then saves the model to disk.

class Tutorial7:
    def __init__(self):
        print("Starting Tutorial7...")
        net = pysmile.Network()

        net.set_outlier_rejection_enabled(True)

        self.create_equation_node(net,
                                  "tra", "Return Air Temperature",
                                  "tra=24", 23.9, 24.1,
```

```
                              280, 100)

    self.create_equation_node(net,
                              "u_d", "Damper Control Signal",
                              "u_d=Bernoulli(0.539)*0.8+0.2", 0, 1,
                              160, 100)

    toa = self.create_equation_node(net,
                               "toa", "Outside Air Temperature",
                               "toa=Normal(11,15)", -10, 40,
                               60, 100)

    # tra, toa and u_d are referenced in equation
    # arcs are created automatically
    tma = self.create_equation_node(net,
                               "tma", "Mixed Air Temperature",
                               "tma=toa*u_d+(tra-tra*u_d)", 10, 30,
                               110, 200)

    self.set_uniform_intervals(net, toa, 5)
    self.set_uniform_intervals(net, tma, 4)

    print("Results with no evidence:")
    self.update_and_show_stats(net)

    net.set_cont_evidence(toa, 28.5)
    print("Results with outside air temperature set to 28.5:")
    self.update_and_show_stats(net)

    net.clear_evidence(toa)
    print("Results with mixed air temperature set to 21:")
    net.set_cont_evidence(tma, 21.0)
    self.update_and_show_stats(net)

    net.write_file("tutorial7.xdsl")
    print("Tutorial7 complete: Network written to tutorial7.xdsl")


def create_equation_node(self, net, id, name, equation, lo_bound,
                         hi_bound, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.EQUATION, id)
    net.set_node_name(handle, name)
    net.set_node_equation(handle, equation)
    net.set_node_equation_bounds(handle, lo_bound, hi_bound)

    net.set_node_position(handle, x_pos, y_pos, 85, 55)

    return handle


def show_stats(self, net, node_handle):
    node_id = net.get_node_id(node_handle)

    if net.is_evidence(node_handle):
        v = net.get_cont_evidence(node_handle)
        print(node_id + " has evidence set " + str(v))
        return
```

```
        if net.is_value_discretized(node_handle):
            print(node_id + " is discretized.")
            iv = net.get_node_equation_discretization(node_handle)
            bounds = net.get_node_equation_bounds(node_handle)
            disc_beliefs = net.get_node_value(node_handle)
            lo = bounds[0]
            for i in range(0, len(disc_beliefs)):
                hi = iv[i].boundary
                print("\tP(" + node_id + " in " + str(lo) + ".." + str(hi)
                      + ")=" + str(disc_beliefs[i]))
                lo = hi
        else:
            stats = net.get_node_sample_stats(node_handle)
            print(node_id + ": mean=" + str(stats[0]) + " stddev="
                  + str(stats[1]) + " min=" + str(stats[2]) + " max="
                  + str(stats[3]))


    def set_uniform_intervals(self, net, node_handle, count):
        bounds = net.get_node_equation_bounds(node_handle)
        lo = bounds[0]
        hi = bounds[1]

        iv = [None] * count
        for i in range(0, count):
            iv[i] = pysmile.DiscretizationInterval("",lo + (i + 1) * (hi - lo)
                                                   / count)

        net.set_node_equation_discretization(node_handle, iv)


    def update_and_show_stats(self, net):
        net.update_beliefs()
        for h in net.get_all_nodes():
            self.show_stats(net, h)

        print("")
```

6.7.3 Tutorial7.cs

```
using System;
using Smile;

// Tutorial7 creates a network with three equation-based nodes
// performs the inference, then saves the model to disk.

namespace SmileNetTutorial
{
    class Tutorial7
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial7...");
            Network net = new Network();
```

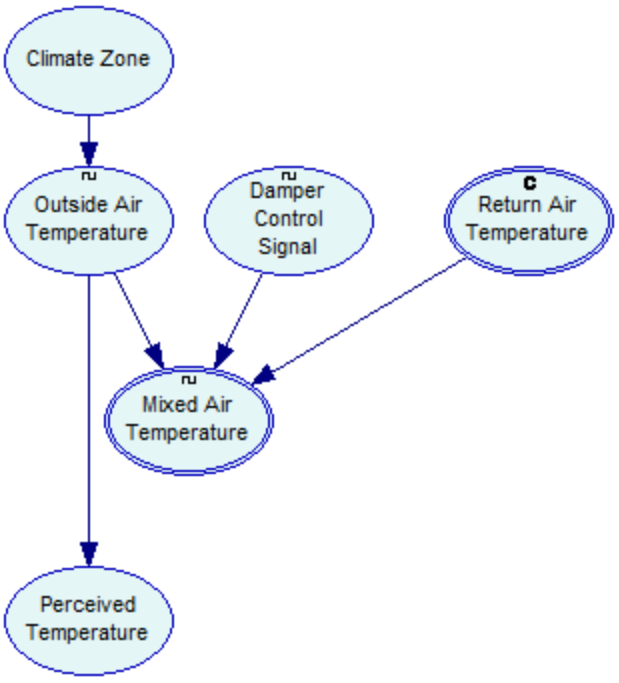
```
    net.OutlierRejectionEnabled = true;

    CreateEquationNode(net,
        "tra", "Return Air Temperature",
        "tra=24", 23.9, 24.1,
        280, 100);

    CreateEquationNode(net,
        "u_d", "Damper Control Signal",
        "u_d = Bernoulli(0.539)*0.8 + 0.2", 0, 1,
        160, 100);

    int toa = CreateEquationNode(net,
        "toa", "Outside Air Temperature",
        "toa=Normal(11,15)", -10, 40,
        60, 100);

    // tra, toa and u_d are referenced in equation
    // arcs are created automatically
    int tma = CreateEquationNode(net,
        "tma", "Mixed Air Temperature",
        "tma=toa*u_d+(tra-tra*u_d)", 10, 30,
        110, 200);

    SetUniformIntervals(net, toa, 5);
    SetUniformIntervals(net, tma, 4);

    Console.WriteLine("Results with no evidence:");
    UpdateAndShowStats(net);

    net.SetContEvidence(toa, 28.5);
    Console.WriteLine(
        "Results with outside air temperature set to 28.5:");
    UpdateAndShowStats(net);

    net.ClearEvidence(toa);
    Console.WriteLine(
        "Results with mixed air temperature set to 21:");
    net.SetContEvidence(tma, 21.0);
    UpdateAndShowStats(net);

    net.WriteFile("tutorial7.xdsl");
    Console.WriteLine(
        "Tutorial7 complete: Network written to tutorial7.xdsl");
}


private static int CreateEquationNode(
    Network net, String id, String name,
    String equation, double loBound, double hiBound,
    int xPos, int yPos)
{
    int handle = net.AddNode(Network.NodeType.Equation, id);
    net.SetNodeName(handle, name);
    net.SetNodePosition(handle, xPos, yPos, 85, 55);
    net.SetNodeEquation(handle, equation);
    net.SetNodeEquationBounds(handle, loBound, hiBound);
```

```
    return handle;
}


private static void ShowStats(Network net, int nodeHandle)
{
    String nodeId = net.GetNodeId(nodeHandle);

    if (net.IsEvidence(nodeHandle))
    {
        double v = net.GetContEvidence(nodeHandle);
        Console.WriteLine("{0} has evidence set ({1})", nodeId, v);
        return;
    }

    if (net.IsValueDiscretized(nodeHandle))
    {
        Console.WriteLine("{0} is discretized.", nodeId);
        DiscretizationInterval[] iv =
            net.GetNodeEquationDiscretization(nodeHandle);
        double[] bounds = net.GetNodeEquationBounds(nodeHandle);
        double[] discBeliefs = net.GetNodeValue(nodeHandle);
        double lo = bounds[0];
        for (int i = 0; i < discBeliefs.Length; i++)
        {
            double hi = iv[i].boundary;
            Console.WriteLine("\tP({0} in {1}..{2})={3}",
                nodeId, lo, hi, discBeliefs[i]);
            lo = hi;
        }
    }
    else
    {
        double[] stats = net.GetNodeSampleStats(nodeHandle);
        Console.WriteLine("{0}: mean={1} stddev={2} min={3} max={4}",
            nodeId, stats[0], stats[1], stats[2], stats[3]);
    }
}


private static void SetUniformIntervals(
    Network net, int nodeHandle, int count)
{
    double[] bounds = net.GetNodeEquationBounds(nodeHandle);
    double lo = bounds[0];
    double hi = bounds[1];

    DiscretizationInterval[] iv = new DiscretizationInterval[count];
    for (int i = 0; i < count; i++)
    {
        iv[i] = new DiscretizationInterval(
            null, lo + (i + 1) * (hi - lo) / count);
    }

    net.SetNodeEquationDiscretization(nodeHandle, iv);
}
```

```
        private static void UpdateAndShowStats(Network net)
        {
            net.UpdateBeliefs();
            for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
            {
                ShowStats(net, h);
            }
            Console.WriteLine();
        }

    }
}
```

6.8 Tutorial 8: Hybrid model

We extend the model described in *Tutorial 7*[113] by adding two discrete nodes: *Climate Zone* and *Perceived Temperature*. *Climate Zone* refines the probability distribution of the outside air temperature and *Perceived Temperature* is an additional input originating from a subjective perception of the temperature, useful in case of a failure in the outside temperature sensor.



After loading the network created in the previous tutorial, the program adds two discrete nodes using the `CreateCptNode` helper function first seen in *Tutorial 1*[68]. The arc from *Climate Zone* (node identifier is *zone*) to *Outside Air Temperature* (node identifier is *toa*) is created by changing the equation of the latter:

Java:

```
net.setNodeEquation(toaHandle, "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");
```

Python:

```
net.set_node_equation("toa", 'toa=If(zone="Desert",Normal(22,5),Normal(11,10))')
```

C#:

```
net.SetNodeEquation(toaHandle, "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");
```

In the Java and C# code, we need to escape the double quote characters in order to use the text literal representing the *Desert* outcome of the parent. Python allows for single quotes to be used as string literal delimiter.

The new equation switches between two normal distribution based on the outcome of the parent. The equation could be also written in the following ways, all being functionally identical (assuming that *Climate Zone* has two outcomes, *Temperate* and *Desert*):

```
toa=zone="Desert" ? Normal(22,5) : Normal(11,10)
toa=zone="Temperate" ? Normal(11,10) : Normal(22,5)
toa=Switch(zone, "Desert",Normal(22,5),"Temperate",Normal(11,10))
toa=Choose(zone,Normal(11,10),Normal(22,5))
```

To create the arc from *Outside Air Temperature* to *Perceived Temperature* the program calls `Network.addArc`. The model loaded from disk had 5 discretization intervals already defined for *Outside Air Temperature*. With 5 possible discretized outcomes of its single parent and its own 3 outcomes, the CPT for *Perceived Temperature* has 3*5=15 entries. It is initialized by a call to Network.setNodeDefinition with an appropriately sized array as input. Note that for simplicity we do not change the default uniform distribution for the binary *Climate Zone* node.

The program performs inference for each of the *Climate Zone* outcomes set as evidence. The output displayed for the *Outside Air Temperature* node (*toa*) shows changing mean and standard deviation. Finally, the network is saved to disk. This concludes the tutorial.

6.8.1 Tutorial8.java

```
package tutorials;

import smile.*;

// Tutorial8 loads continuous model from the XDSL file written by Tutorial7,
// then adds discrete nodes to create a hybrid model. Inference is performed
// and model is saved to disk.

public class Tutorial8 {
    public static void run() {
        System.out.println("Starting Tutorial8...");
        Network net = new Network();

        net.readFile("tutorial7.xdsl");

        createCptNode(
              net, "zone", "Climate Zone",
```

```
            new String[] { "Temperate", "Desert" },
            60, 20);

        int toaHandle = net.getNode("toa");
        net.setNodeEquation(toaHandle,
            "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");

        int perceivedHandle = createCptNode(
            net, "perceived", "Perceived Temperature",
            new String[] { "Hot", "Warm", "Cold" },
            60, 300);
        net.addArc(toaHandle, perceivedHandle);

        double[] perceivedProbs = new double[] {
            0,    // P(perceived=Hot |toa in -10..0)
            0.02, // P(perceived=Warm|toa in -10..0)
            0.98, // P(perceived=Cold|toa in -10..0)
            0.05, // P(perceived=Hot |toa in 0..10)
            0.15, // P(perceived=Warm|toa in 0..10)
            0.80, // P(perceived=Cold|toa in 0..10)
            0.10, // P(perceived=Hot |toa in 10..20)
            0.80, // P(perceived=Warm|toa in 10..20)
            0.10, // P(perceived=Cold|toa in 10..20)
            0.80, // P(perceived=Hot |toa in 20..30)
            0.15, // P(perceived=Warm|toa in 20..30)
            0.05, // P(perceived=Cold|toa in 20..30)
            0.98, // P(perceived=Hot |toa in 30..40)
            0.02, // P(perceived=Warm|toa in 30..40)
            0     // P(perceived=Cold|toa in 30..40)
        };
        net.setNodeDefinition(perceivedHandle, perceivedProbs);

        net.setEvidence("zone", "Temperate");
        System.out.println("Results in temperate zone:");
        updateAndShowStats(net);

        net.setEvidence("zone", "Desert");
        System.out.println("Results in desert zone:\n");
        updateAndShowStats(net);

        net.writeFile("tutorial8.xdsl");
        System.out.println(
            "Tutorial8 complete: Network written to tutorial8.xdsl");
    }


    static void showStats(Network net, int nodeHandle) {
              String nodeId = net.getNodeId(nodeHandle);

              if (net.isEvidence(nodeHandle)) {
                     double v = net.getContEvidence(nodeHandle);
                     System.out.printf("%s has evidence set (%g)\n", nodeId, v);
                     return;
              }

              if (net.isValueDiscretized(nodeHandle)) {
                     System.out.printf("%s is discretized.\n", nodeId);
```

```
                    DiscretizationInterval[] iv =
                net.getNodeEquationDiscretization(nodeHandle);
                    double[] bounds = net.getNodeEquationBounds(nodeHandle);
                    double[] discBeliefs = net.getNodeValue(nodeHandle);
                    double lo = bounds[0];
                  for (int i = 0; i < discBeliefs.length; i++) {
                      double hi = iv[i].boundary;
                      System.out.printf(
                     "\tP(%s in %g..%g)=%g\n", nodeId, lo, hi, discBeliefs[i]);
                      lo = hi;
                  }
              } else {
                    double[] stats = net.getNodeSampleStats(nodeHandle);
               System.out.printf("%s: mean=%g stddev=%g min=%g max=%g\n",
                   nodeId, stats[0], stats[1], stats[2], stats[3]);
              }
    }


    static void updateAndShowStats(Network net) {
        net.updateBeliefs();
        for (int h = net.getFirstNode(); h >= 0; h = net.getNextNode(h))
        {
            if (net.getNodeType(h) == Network.NodeType.EQUATION) {
              showStats(net, h);
            }
        }
        System.out.println();
    }


    private static int createCptNode(
            Network net, String id, String name,
            String[] outcomes, int xPos, int yPos) {
        int handle = net.addNode(Network.NodeType.CPT, id);

        net.setNodeName(handle, name);
        net.setNodePosition(handle, xPos, yPos, 85, 55);

        int initialOutcomeCount = net.getOutcomeCount(handle);
        for (int i = 0; i < initialOutcomeCount; i ++) {
            net.setOutcomeId(handle, i, outcomes[i]);
        }

        for (int i = initialOutcomeCount; i < outcomes.length; i ++) {
            net.addOutcome(handle, outcomes[i]);
        }

        return handle;
    }

}
```

6.8.2 Tutorial8.py

```
import pysmile

# Tutorial8 loads continuous model from the XDSL file written by Tutorial7,
# then adds discrete nodes to create a hybrid model. Inference is performed
# and model is saved to disk.

class Tutorial8:
    def __init__(self):
        print("Starting Tutorial8...")
        net = pysmile.Network()

        net.read_file("tutorial7.xdsl")

        self.create_cpt_node(net,
                        "zone", "Climate Zone",
                        ["Temperate", "Desert"],
                        60, 20)
        toa_handle = net.get_node("toa")
        net.set_node_equation("toa",
                        'toa=If(zone="Desert",Normal(22,5),Normal(11,10))')

        perceived_handle = self.create_cpt_node(net,
                                        "perceived", "Perceived Temperature",
                                        ["Hot", "Warm", "Cold"],
                                        60, 300)

        net.add_arc(toa_handle, perceived_handle)

        perceived_probs = [
            0,    # P(perceived=Hot |toa in -10..0)
            0.02, # P(perceived=Warm|toa in -10..0)
            0.98, # P(perceived=Cold|toa in -10..0)
            0.05, # P(perceived=Hot |toa in 0..10)
            0.15, # P(perceived=Warm|toa in 0..10)
            0.80, # P(perceived=Cold|toa in 0..10)
            0.10, # P(perceived=Hot |toa in 10..20)
            0.80, # P(perceived=Warm|toa in 10..20)
            0.10, # P(perceived=Cold|toa in 10..20)
            0.80, # P(perceived=Hot |toa in 20..30)
            0.15, # P(perceived=Warm|toa in 20..30)
            0.05, # P(perceived=Cold|toa in 20..30)
            0.98, # P(perceived=Hot |toa in 30..40)
            0.02, # P(perceived=Warm|toa in 30..40)
            0     # P(perceived=Cold|toa in 30..40)
        ]

        net.set_node_definition(perceived_handle, perceived_probs)

        net.set_evidence("zone", "Temperate")
        print("Results in temperate zone:")
        self.update_and_show_stats(net)

        net.set_evidence("zone", "Desert")
        print("Results in desert zone:")
        self.update_and_show_stats(net)
```

```
    net.write_file("tutorial8.xdsl")
    print("Tutorial8 complete: Network written to tutorial8.xdsl")


def show_stats(self, net, node_handle):
    node_id = net.get_node_id(node_handle)

    if net.is_evidence(node_handle):
        v = net.get_cont_evidence(node_handle)
        print(node_id + " has evidence set " + str(v))
        return

    if net.is_value_discretized(node_handle):
        print(node_id + " is discretized.")
        iv = net.get_node_equation_discretization(node_handle)
        bounds = net.get_node_equation_bounds(node_handle)
        disc_beliefs = net.get_node_value(node_handle)
        lo = bounds[0]
        for i in range(0, len(disc_beliefs)):
            hi = iv[i].boundary
            print("\tP(" + node_id + " in " + str(lo) + ".." + str(hi)
                  + ")=" + str(disc_beliefs[i]))
            lo = hi
    else:
        stats = net.get_node_sample_stats(node_handle)
        print(node_id + ": mean=" + str(stats[0]) + " stddev="
              + str(stats[1]) + " min=" + str(stats[2]) + " max="
              + str(stats[3]))


def update_and_show_stats(self, net):
    net.update_beliefs()
    for h in net.get_all_nodes():
        if net.get_node_type(h) == pysmile.NodeType.EQUATION:
            self.show_stats(net, h)

    print("")


def create_cpt_node(self, net, id, name, outcomes, x_pos, y_pos):
    handle = net.add_node(pysmile.NodeType.CPT, id)

    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)

    initial_outcome_count = net.get_outcome_count(handle)

    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])

    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])

    return handle
```

6.8.3 Tutorial8.cs

```
using System;
using Smile;

// Tutorial8 loads continuous model from the XDSL file written by Tutorial7,
// then adds discrete nodes to create a hybrid model. Inference is performed
// and model is saved to disk.

namespace SmileNetTutorial
{
    class Tutorial8
    {
        public static void Run()
        {
            Console.WriteLine("Starting Tutorial8...");
            Network net = new Network();

            net.ReadFile("tutorial7.xdsl");

            CreateCptNode(
                net, "zone", "Climate Zone",
                new String[] { "Temperate", "Desert" },
                60, 20);

            int toaHandle = net.GetNode("toa");
            net.SetNodeEquation(toaHandle,
                "toa=If(zone=\"Desert\",Normal(22,5),Normal(11,10))");

            int perceivedHandle = CreateCptNode(
                net, "perceived", "Perceived Temperature",
                new String[] { "Hot", "Warm", "Cold" },
                60, 300);
            net.AddArc(toaHandle, perceivedHandle);

            double[] perceivedProbs = new double[] {
                0,    // P(perceived=Hot |toa in -10..0)
                0.02, // P(perceived=Warm|toa in -10..0)
                0.98, // P(perceived=Cold|toa in -10..0)
                0.05, // P(perceived=Hot |toa in 0..10)
                0.15, // P(perceived=Warm|toa in 0..10)
                0.80, // P(perceived=Cold|toa in 0..10)
                0.10, // P(perceived=Hot |toa in 10..20)
                0.80, // P(perceived=Warm|toa in 10..20)
                0.10, // P(perceived=Cold|toa in 10..20)
                0.80, // P(perceived=Hot |toa in 20..30)
                0.15, // P(perceived=Warm|toa in 20..30)
                0.05, // P(perceived=Cold|toa in 20..30)
                0.98, // P(perceived=Hot |toa in 30..40)
                0.02, // P(perceived=Warm|toa in 30..40)
                0     // P(perceived=Cold|toa in 30..40)
            };
            net.SetNodeDefinition(perceivedHandle, perceivedProbs);

            net.SetEvidence("zone", "Temperate");
            Console.WriteLine("Results in temperate zone:");
            UpdateAndShowStats(net);
```

```
    net.SetEvidence("zone", "Desert");
    Console.WriteLine("Results in desert zone:\n");
    UpdateAndShowStats(net);

    net.WriteFile("tutorial8.xdsl");
    Console.WriteLine(
        "Tutorial8 complete: Network written to tutorial8.xdsl");
}


private static int CreateEquationNode(
    Network net, String id, String name,
    String equation, double loBound, double hiBound,
    int xPos, int yPos)
{
    int handle = net.AddNode(Network.NodeType.Equation, id);
    net.SetNodeName(handle, name);
    net.SetNodePosition(handle, xPos, yPos, 85, 55);
    net.SetNodeEquation(handle, equation);
    net.SetNodeEquationBounds(handle, loBound, hiBound);
    return handle;
}


private static void ShowStats(Network net, int nodeHandle)
{
    String nodeId = net.GetNodeId(nodeHandle);

    if (net.IsEvidence(nodeHandle))
    {
        double v = net.GetContEvidence(nodeHandle);
        Console.WriteLine("{0} has evidence set ({1})", nodeId, v);
        return;
    }

    if (net.IsValueDiscretized(nodeHandle))
    {
        Console.WriteLine("{0} is discretized.", nodeId);
        DiscretizationInterval[] iv =
            net.GetNodeEquationDiscretization(nodeHandle);
        double[] bounds = net.GetNodeEquationBounds(nodeHandle);
        double[] discBeliefs = net.GetNodeValue(nodeHandle);
        double lo = bounds[0];
        for (int i = 0; i < discBeliefs.Length; i++)
        {
            double hi = iv[i].boundary;
            Console.WriteLine("\tP({0} in {1}..{2})={3}",
                nodeId, lo, hi, discBeliefs[i]);
            lo = hi;
        }
    }
    else
    {
        double[] stats = net.GetNodeSampleStats(nodeHandle);
        Console.WriteLine("{0}: mean={1} stddev={2} min={3} max={4}",
            nodeId, stats[0], stats[1], stats[2], stats[3]);
```

```
            }
        }


        private static void UpdateAndShowStats(Network net)
        {
            net.UpdateBeliefs();
            for (int h = net.GetFirstNode(); h >= 0; h = net.GetNextNode(h))
            {
                if (net.GetNodeType(h) == Network.NodeType.Equation)
                {
                    ShowStats(net, h);
                }
            }
            Console.WriteLine();
        }


        private static int CreateCptNode(
            Network net, String id, String name,
            String[] outcomes, int xPos, int yPos)
        {
            int handle = net.AddNode(Network.NodeType.Cpt, id);

            net.SetNodeName(handle, name);
            net.SetNodePosition(handle, xPos, yPos, 85, 55);

            int initialOutcomeCount = net.GetOutcomeCount(handle);
            for (int i = 0; i < initialOutcomeCount; i++)
            {
                net.SetOutcomeId(handle, i, outcomes[i]);
            }

            for (int i = initialOutcomeCount; i < outcomes.Length; i++)
            {
                net.AddOutcome(handle, outcomes[i]);
            }

            return handle;
        }

    }
}
```

Appendix: R, rJava and jSMILE

7 Appendix: R, rJava and jSMILE

NOTE: we're currently working on a native R wrapper for SMILE. Contact us if you want to use a pre-release version.

R programmers can use SMILE's functionality through the rJava package (more information available at *https://CRAN.R-project.org/package=rJava*). rJava provides low-level interface to Java Virtual Machine, effectively making possible calls from R to jSMILE.

To ensure that JVM can find jSMILE's jar file and native library, use `.jinit` parameters:

```
.jinit("<jSMILE jar path>", "-Djsmile.native.library=<native library path>")
```

Alternatively, set the locations later:

```
.jaddClassPath("<jSMILE jar path>")
J("java.lang.System")$setProperty("jsmile.native.library","<native library path>")
```

For more information regarding the native library part of jSMILE, see Java and jSMILE [12].

Here's the R equivalent of the "Hello, SMILE" program, which loads the model from the VentureBN.xsdl file, sets the evidence and displays the calculated posterior probabilties. The code assumes that JVM was started with `.jinit` and SMILE license was initialized with `.jnew("smile.License", ...)` call.

```
net <- .jnew("smile.Network")
.jcall(net,, "readFile", "VentureBN.xdsl")
.jcall(net,, "setEvidence", "Forecast", "Moderate")
.jcall(net,, "updateBeliefs")
beliefs <- .jcall(net, "[D", "getNodeValue", "Success")
for (i in 1 : length(beliefs)) {
    outcomeId = .jcall(net, "S", "getOutcomeId", "Success", as.integer(i-1))
    print(paste(outcomeId, "=", beliefs[i]))
}
```

By default, rJava marshals numeric parameters as doubles to the JVM side. If the jSMILE method to be called requires integer parameters, the `as.integer` function must be called to coerce the value, as in the `getOutcomeId` call inside the for loop. Without as.integer, the code would cause the following rJava error:

```
method getOutcomeId with signature (Ljava/lang/String;D)Ljava/lang/String; not found
```

Since 2nd input parameter of Network.getOutcomeId is an integer, the proper singature is `(Ljava/lang/String;I)Ljava/lang/String;`, the difference is just one letter (I for integer instead of D for double). Using `as.integer` ensures that rJava can correctly infer the signature.

Note how calls to `smile.Network` method returning `void` do not require the declaration of the return type (the default value of the second `.jcall` parameter is used). On the other hand, to retrieve the posteriors with `smile.getNodeValue`, its return type, which is `double[]` in Java, must be passed to `.jcall`. rJava uses Java Native Interface (JNI) type signatures and `double[]` becomes `"[D"`. For convenience, rJava treats `"S"` as signature for Java's strings (the full signature

is `"Ljava/lang/String;")` - the `smile.Network.getOutcomeId` invocation inside the for loop takes advantage of the shorthand notation. The complete list of type signatures is included in the JNI documentation at *https://docs.oracle.com/javase/9/docs/specs/jni/types.html#type-signatures*

This page is intentionally left blank.

Acknowledgments

8 Acknowledgments

SMILE internally uses portions of the following two software libraries: micro-ECC and Expat. PySMILE uses pybind11. These three libraries require an acknowledgment that we are reproducing below.

micro-ECC

Expat

and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

pybind

Copyright (c) 2016 Wenzel Jakob <wenzel.jakob@epfl.ch>, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVE CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.