# User's Guide for the BNT Soft Discretization Package (Version 1.0)

by Imme Ebert-Uphoff*

Adjunct Associate Professor
School of Mechanical Engineering
Georgia Institute of Technology
ebert@me.gatech.edu or ebert@stups.com

November 10, 2009

## 1 The BNT Soft Discretization Package

Many real world systems contain both discrete and continuous nodes. Bayesian networks can currently only model continuous nodes that follow a Gaussian distribution. Thus, if the continuous nodes of the system to be modeled do not follow a Gaussian distribution, discrete nodes must be used, raising the question of how to best discretize the continuous training data.

The BNT Soft Discretization Package serves as interface between discrete Bayesian networks and continuous data. Soft discretization uses soft boundaries, i.e. a value near a discretization interval can be modeled as falling into more than one interval, where weights describe the extend to which the value is associated with the different intervals. All the algorithms are described in great detail in [1]. The BNT Soft Discretization Package implements all of the algorithms from [1] for the Bayes Net Toolbox (BNT) for Matlab.

While [1] uses the example of precipitation down-scaling as example, we use a different example here for demonstration purposes. Namely the incinerator system defined by Lauritzen [2] is used here, because it is a mixed discrete-continuous system and thus better suited to demonstrate the use of the package for mixed discrete-continuous systems.

### 1.1 Mixed Discrete-Continuous Systems

The algorithms apply to systems where all nodes are continuous, as well as to systems that have both discrete and continuous nodes (see [1]). In the later case, which we call a mixed discrete-continuous system, the training data has continuous values for some nodes, discrete values for others. However, for each single node the training data has to be consistent - either all discrete or all continuous.

### 1.2 Algorithms Included in this Package

This package contains *all* algorithms from [1], namely

- Converting continuous node values to weights (soft evidence) using soft discretization;
- Training the CPTs of a discrete network from mixed discrete-continuous training data using soft discretization;

*Joint Appointment with the Robotics and Intelligent Machines Center, School of Interactive Computing, College of Computing, Atlanta, GA 30308.

- Performing soft inference in the new network, and converting the inference results to a single continuous (expected) value for each continuous node, if desired.

It is recommended that users first read [1] to familiarize themselves with the algorithms and terminology.

# 2 Availability, Updates and Permission to Use

The source code, this documentation and Technical Report [1] are available at www.DataOnStage.com. This may or may not be the newest version of the code. The newest version and any bug fixes are always available at www.DataOnStage.com.

Permission to use this package is granted according to the terms described in file license.txt in the source code directory. Please acknowledge the use of the *BNT Soft Discretization Package* and cite [1] in the corresponding publications.

Although not required, I would also appreciate it *very much* if you would send me an e-mail if you use this package. Knowing that this work is useful to other people is my main reward for providing these files! For any comments, questions or suggestions, please send e-mail to ebert@stups.com

# 3 Organization of this Document

The remainder of this document is organized as follows. Section 4 discusses the individual Matlab functions of the Soft Discretization Package. Section 5 shows how to use the functions using a sample model and discusses the results. Appendix A lists the sample code for this example.

# 4 Available Matlab Functions

Section 4.1 provides an overview of all the Matlab functions included in the Soft Discretization Package, while Section 4.2 focuses on the six User Interface Functions, which provide access to the complete functionality of the package.

## 4.1 Overview of All Functions

The files included in the Soft Discretization Package can be divided into the following three groups.

1. **User Interface Functions**
   The following six user interface functions provide the interface to all the functionalities of soft discretization (including inference).

   ```
   soft_discretization_weights.m
   soft_CPT_for_all_nodes.m
   check_sum.m
   soft_inference.m
   init_state_means.m
   create_discrete_bnet.m
   ```

2. **Auxiliary Functions**
   The following three auxiliary functions are used by the core files. You are not likely to call any of these directly.

   ```
   AUX_increment_vector_state.m
   AUX_lin_index.m
   my_norm_cdf.m
   ```

3. **Sample Use Functions**
   These following two files document how to use of soft discretization. You will not need them any more once you wrote your own application.

   ```
   sample_script.m
   mk_incinerator_bnet_modified.m
   ```

## 4.2 User Interface Functions

The interface to all the functionalities of soft discretization (including inference) is given by the six core functions. Each one of these six functions is discussed in detail below. Their use is demonstrated later in file sample_script.m, which is discussed in much detail in Section 5.

### 4.2.1 soft_discretization_weights

**Purpose:** This function converts evidence into soft evidence in the form of weights. See Sections 4.2.2 and 4.2.3 of [1] for the definition of the weights. This function is primarily used by functions "soft_CPT_for_all_nodes" and "soft_inference", but can also be used on its own.

**Format:**

```
function p_weight = soft_discretization_weights( bnet_soft, case_mixed_vec, ...
                  node_list, bounds, discretization_type, deviation, deviation_is_factor )

% CALCULATE WEIGHTS USING SOFT DISCRETIZATION
% As defined in Sections 4.2.2 and 4.2.3 of research report GT-ME-2009-002.
% Return single cell array for all nodes, where only entries for nodes in
% node_list are filled!
%
% INPUT:
%   bnet_soft      = discretized network, CPT values can still be random
%   case_mixed_vec = data vector for single case with possibly mixed discrete
%                    and continuous values
%   node_list      = list of nodes to be discretized, i.e. for which weights
%                    should be calculated.
%      Note: We often include ALREADY discrete nodes in this list, to get
%            weights for all nodes of interest.
%      Sample use:
%           node_list = evidence_nodes for inference,
%           node_list = CPT_nodes for learning CPT parameters
%
%   bounds         = list of discretization boundaries for continuous nodes
%    discretization_type:
%      'is_discrete'   % already discretized (do NOT discretize)
%      'hard'          % use hard discretization
%      'soft'          % use soft discretization
%    deviation = value
%    deviation_is_factor:  true/false
%       true:   sigma = deviation * x_hat  (use deviation value as percentage factor)
%       false:  sigma = deviation          (use deviation value as absolute value)
%
% OUTPUT:
%    p_weight  = cell array of the form p_weight{node}(state)
%                which contains weights for each node included in list of node_list
```

**Comments:** Use input parameter node_list to define for which nodes you want to calculate weights. node_list may contain nodes that are already discrete. If using for inference, simply use list of all evidence nodes as node_list, regardless of whether they are continuous or discrete.

#### 4.2.2 soft_CPT_for_all_nodes

**Purpose:** Train the CPTs for a discrete network from mixed discrete-continuous data using soft discretization according to Section 4 of [1].

**Format:**

```
function bnet  = soft_CPT_for_all_nodes( bnet, training_cases, bounds, ...
     discretization_type, deviation, deviation_is_factor, show_CPT, show_JPT )

% Learn all CPTs of the network using SOFT or HARD DISCRETIZATION as desired
%
% INPUT
%  bnet = BN for which to calculate CPTs
%         --> CPTs of that bnet will be OVERWRITTEN in this routine
%  training_cases = cell containing training cases
%                   (each case can contain discrete and continuous values)
%  discretization_type = contains type for each node
%      'is_discrete'   % already discretized (do NOT discretize)
%      'hard'          % use hard discretization
%      'soft'          % use soft discretization
%  deviation = array with one value for each node
%  deviation_is_factor:  array with one true/false value for each node
%       true:   sigma = deviation * x_hat  (use deviation value as percentage factor)
%       false:  sigma = deviation          (use deviation value as absolute value)
%  show_CPT = whether or not to display the CONDITIONAL probabilities calculated
%  show_JPT = whether or not to display the JOINT       probabilities calculated
%
% OUTPUT
%  bnet = original bnet, but now with CPTs assigned, and META data
%         appended (bounds, deviation, etc.)
```

**Comments:** The CPTs of bnet are overwritten by this function.

#### 4.2.3 check_sum

**Purpose:** This function calculates the check_sum entries for all CPT tables as defined in Section 7.4 of [1]. If all check_sum entries are one, that indicates that all CPTs are well defined. check_sum entries with values smaller than one indicate ill-defined CPTs. (Values larger than one can never occur.) See Section 7.4 of [1] for details.

**Format:**

```
function [one_entries, not_one_entries]  = check_sum(bnet, epsilon, ...
                                                     short_print_out)

% Calculate check_sum function for all CPT tables of bnet according to
% See Section 7.4 of research report GT-ME-2009-002 for definition of
% check_sum.
% In a nutshell, the check_sum entry is defined, for any parent state
% combination, as the sum of the CPTs over all child states
% given that particular parent state combination:
%    check_sum (parent state combi) = sum_j  P(y_j | parent state combi).
%
% Interpretation:
% If all check_sum entries are one, that indicates that all CPT entries
% are well defined.
% If many are NOT one, that indicates that many CPT entries are not well
% defined (BNT sets undefined CPT entries to default value 0).
%
```

```
% INPUT:
% bnet:              BN to be checked
% epsilon:           threshold value (0.0001 seems to be a good value) to
%                    distinguish between 1 and 'not 1'.
% short_print_out:   if true:  only print number of entries that are/are not 1.
%                    if false: also print all check_sum entries.
%
% OUTPUT:
% one_entries:       Number of check_sum entries that are one
% not_one_entries:   Number of check_sum entries that are not one
```

#### 4.2.4 soft_inference

**Purpose:** This function converts a data vector into soft evidence and returns an engine initialized with the soft evidence, as described in Section 6.1 of [1]. The input data_vec consists of the continuous/discrete values of all evidence nodes. Soft inference typically has two "soft" components: (1) The evidence is converted to soft evidence before it is entered into the network and (2) the CPTs of the network used for inference were usually obtained through soft discretization.

**Format:**

```
function  engine = soft_inference( bnet_soft, data_vec, evidence_nodes, ...
   bounds, discretization_type, deviation, deviation_is_factor)

% Initialize inference engine using SOFT EVIDENCE --
% as explained in Section 6.1 of research report GT-ME-2009-002.
%
% This routine first takes the evidence contained in data_vec and converts
% it into soft evidence (i.e. weights).  Only the values corresponding to nodes of
% evidence_nodes have to be filled in vector data_vec, since only those are used.
%
% The weights are entered into a new inference engine as soft evidence.
% The initialized engine is then returned as output.
%
% INPUT:
% bnet_soft:       bnet created by soft evidence
% data_vec:        data vector that contains evidence values for all
%                  evidence_nodes
% evidence_nodes:  list of nodees for which evidence is to be entered
% bounds, discretization_type:
%                  same as in function soft_discretization_weights
% deviation, deviation_is_factor:
%                  same as in function soft_discretization_weights
%
% OUTPUT:
% engine:          new engine initialized with the soft evidence
```

#### 4.2.5 init_state_means

**Purpose:** This function calculates the mean for each state of each node based on provided training data, as described in Section 6.2 of [1]. Our recommendation is to use the same training data here as is used for the training of the network (i.e. when calling function soft_CPT_for_all_nodes).

This function is **not** required to use soft inference. It is only required if we want to convert the output of the inference into a single continuous value for each node. The function needs to be called only once for the training data, then state_mean is available for all future calculations.

**Format:**

```
function state_mean  = init_state_means( mixed_samples, cnodes, bounds)
```

```
% Initialize state means for all continuous nodes
% following Section 6.2 of research report GT-ME-2009-002.
% Using the training cases, find the mean value for each state of each node.
% This mean value is used later on for inference to calculate an
% expected value, i.e. a continuous output value.
%
% INPUT:
% mixed_samples:  cell array of sample cases that may contain both continuous
%                 and discrete values
% cnodes:         list of all continuous nodes in the original network
% bounds:         cell array of discretization boundaries for the continuous
%                 nodes
%
% OUTPUT:
% state_mean:     cell array containing the mean values for each state of
%                 each node
% state_mean{node}(state) then provides the mean value for a specific node and
%                         state.
```

#### 4.2.6 create_discrete_bnet

**Purpose:** Take a network containing one or more continuous nodes and create a new network from it that has only discrete nodes. This function was not discussed in [1] and was only created because our example starts out with a mixed discrete-continuous *network* from which mixed discrete-continuous *training data* is created. This function provides a convenient way to define the discretized network directly from the mixed discrete-continuous network.

In most application one starts out with mixed discrete-continuous *data*, rather than a network, and in those cases function create_discrete_bnet cannot be used. Instead one would define the graph of the discrete network by hand and then use vector boundaries to define the number of states for each node. Thus you are unlikely to use this file in most applications, but it is a good sample for creating your own customized function.

**Format:**

```
function bnet_discrete = create_discrete_bnet( bnet_mixed, boundaries )

% CREATE DISCRETIZED NETWORK (WITH RANDOM CPT) FROM MIXED NETWORK
%
% INPUT
% bnet_mixed:      original BN with at least one continuous nodes,
%                  but may also include discrete nodes
% boundaries:      array containing discretization boundaries for all
%                  continouous nodes
%
% OUTPUT
% bnet_discrete:   corresponding bnet with only discrete nodes, CPTs are random.
```

**Comments:** The boundaries are needed as input only to determine the proper node sizes for the discrete network.

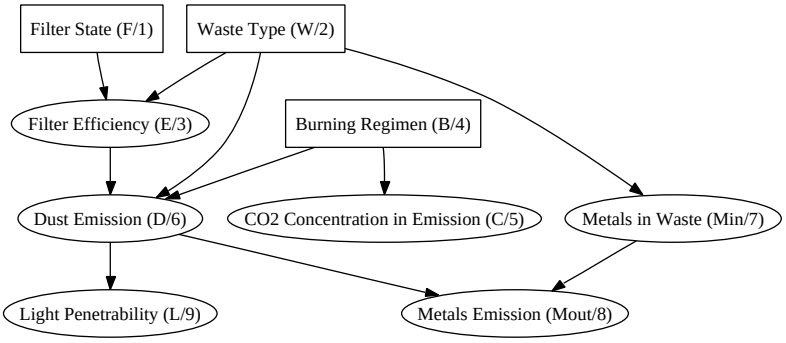


Figure 1: Original incinerator model with both discrete (rectangular) and continuous (oval) nodes

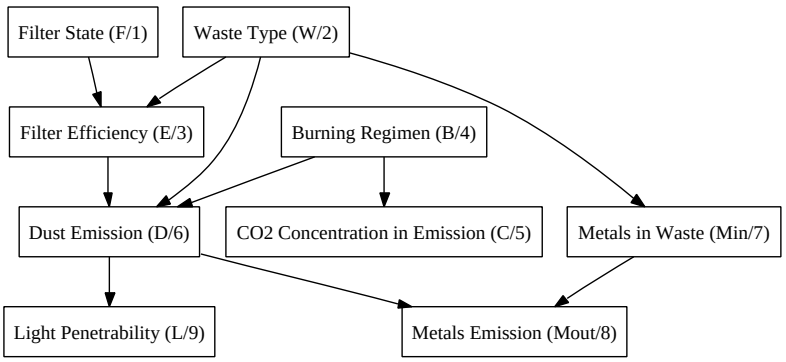


Figure 2: Discretized version of incinerator network

# 5 Sample Use

## 5.1 Incinerator Model

To demonstrate the use of the soft discretization files, let us look at the following example. The file

```
mk_incinerator_bnet_modified.m
```

initializes the Bayes Net for the incinerator network defined by Lauritzen [2]. This example demonstrates the use of the BNT Soft Discretization Package better than the example used in [1], because it is a mixed discrete-continuous system. Our version of the incinerator model contains two small modifications: it uses longer node names and all nodes are defined to be observed. (Those changes were made primarily to generate Figures 1 and 2 with the long node names.)

The graph of this model is shown in Figure 1, where rectangular nodes represent discrete nodes and oval nodes represent continuous nodes. The The letters listed in parentheses next to the node names denote short names for each node, also used in the original publication, [2], and the number in parentheses are the node numbers used in the BNT code.

This network is fictitious and was introduced by Lauritzen to demonstrate the use of continuous nodes with Gaussian distributions in Bayesian networks. There is no scientific reason to discretize this continuous model, since a Gaussian distribution seems to be a good representation of the system. However, discretizing this network is an excellent teaching tool to show how to use our soft discretization files, because the model provides a simple means to generate mixed sample vectors that can be used to train and test a discretized model. Figure 2 shows the corresponding model with only discrete nodes.

Note that in most applications one would not start out with a mixed Bayesian Network, but instead with data obtained from an application.

## 5.2 Using the Sample Script

The file sample_script can be executed simply by typing sample_script in the Matlab command window. Alternatively, one can open the file sample_script.m in a text editor and copy groups of lines directly into the Matlab command window, from beginning to end of the file. The sample code has the following main components:

PART I: CREATE TRAINING DATA, THEN INITIALIZE, TRAIN AND EVALUATE TWO DISCRETIZED NETWORKS (one with soft discretization, one with hard discretization)

1. **Create Training Data**
   Initialize mixed incinerator network, create 1000 samples.

2. **Define Parameters for Soft Discretization**
   Define the following discretization parameters for each node,

   (a) discretization type (soft/hard/is_discrete),

   (b) deviation (value) and deviation_is_factor (true/false) - for nodes with continuous training data,

   (c) discretization boundaries - for nodes with continuous training data.

3. **Create Discrete Network (with random CPTs): bnet_soft**
   This step is done here based on the mixed incinerator model. In applications where training data is given directly (no mixed BN model), one would instead define the graph structure directly by hand, but one could still use the cell array 'boundaries' to define the numbers of node states.

4. **Train bnet_soft Using Soft Discretization**
   Use function soft_CPT_for_all_nodes to calculate all CPTs for discrete model according to Section 4.3 of [1]. Display the resulting joint probabilities (Equation (16) of [1]) and conditional probabilities (last equation on p. 14 of [1]) for each node.

5. **Create and train another discrete network (bnet_hard) using Hard Discretization**
   This second network is created so that we can compare results for soft and hard discretization.

6. **Apply CHECK_SUM test**
   Apply the check_sum test of Section 7.4 of [1] to all CPT entries of bnet_hard and bnet_soft. There is one check_sum entry for each parent combination and it is

   $$\text{CHECK\_SUM}(\text{parent state combi}) = \sum_j P(y_j|\text{parent state combi}),$$

   i.e. for a given parent state combination it is the sum of the conditional probabilities over all child states. If all check_sum entries are one, then all CPTs are well defined. However, entries that are not one indicate undefined CPTs. Those usually indicate that a specific parent state combination did not occur in the training data, and thus all corresponding CPTs were set to 0 by BNT.

7. **Compare CPTs of bnet_hard and bnet_soft for Node 6**
   Node 6 is the node with the largest number of parents in this network (3 parents), thus it is most likely to have undefined CPT entries.

PART II: USE bnet_soft TO CALCULATE WEIGHTS AND DEMONSTRATE INFERENCE

1. **Calculate Soft Discretization Weights for a Sample**
   Create a new sample vector from original network and show how to calculate all weights for this sample.

2. **Demonstrate Inference**
   For the same sample, enter evidence for all nodes except 6,7,8 as soft evidence. Now we can infer probabilities for nodes 6,7,8. (See also Section 6.1 of [1].)

3. **Demonstrate Creating Continuous Output Estimates**
   Run init_state_means to get means of all states for all nodes based on training data, see Section 6.2 of [1]. Use the state means to generate continuous estimate for values of nodes 6, 7 and 8.

The sample code is fairly self-explanatory and is included in Appendix A for ease of reference.

## 5.3 Sample Results

Running the script sample_script.m yields slightly different results every time, because the training data and test data are generated by sampling the mixed incinerator network, which results in different data sets every time. Below is the sample output for a run.

```
>> sample_script

----------- CREATING INCINERATOR NETWORK ---------
Creating 1000 samples from original network

---------- CREATE DISCRETE BN (with random CPTs):   bn_soft
Creating BNet with only discrete nodes
Done
Initialize CPTs with random values
Done

------ TRAIN bnet_soft USING SOFT DISCRETIZATION -------
Calc CPT for child node 1 (0 parents -- 2 CPT entries)

Calc CPT for child node 2 (0 parents -- 2 CPT entries)

Calc CPT for child node 3 (2 parents -- 12 CPT entries)

Calc CPT for child node 4 (0 parents -- 2 CPT entries)

Calc CPT for child node 5 (1 parents -- 8 CPT entries)

Calc CPT for child node 6 (3 parents -- 48 CPT entries)

Calc CPT for child node 7 (1 parents -- 4 CPT entries)

Calc CPT for child node 8 (2 parents -- 32 CPT entries)

Calc CPT for child node 9 (1 parents -- 16 CPT entries)


---------- CREATE DISCRETE BN (with random CPTs):   bn_hard
Creating BNet with only discrete nodes
Done
Initialize CPTs with random values
Done

------ TRAIN bnet_hard USING HARD DISCRETIZATION -------
Calc CPT for child node 1 (0 parents -- 2 CPT entries)

Calc CPT for child node 2 (0 parents -- 2 CPT entries)

Calc CPT for child node 3 (2 parents -- 12 CPT entries)

Calc CPT for child node 4 (0 parents -- 2 CPT entries)

Calc CPT for child node 5 (1 parents -- 8 CPT entries)
```

```
Calc CPT for child node 6 (3 parents -- 48 CPT entries)

Calc CPT for child node 7 (1 parents -- 4 CPT entries)

Calc CPT for child node 8 (2 parents -- 32 CPT entries)

Calc CPT for child node 9 (1 parents -- 16 CPT entries)

----- Highlighting Differences between hard and soft CPTs -----
Hit ENTER to continue


CHECK SUM entries:
We WANT all entries to be one.  That means all CPTs are well defined.

Check Sum entries for CPTs of   bnet_hard:
# entries that ARE     1:  30   (85.714286 percent)
# entries that are NOT 1:  5   (14.285714 percent)

Check Sum entries for CPTs of   bnet_soft:
# entries that ARE     1:  35   (100.000000 percent)
# entries that are NOT 1:  0   (0.000000 percent)

Show CPT for Node 6 = node with the most parents in the network.
Hit ENTER to continue

Index set     CPT_hard        CPT_soft
(1,1,1,1)     1.000000        1.000000
(1,1,1,2)     0.000000        0.000000
(1,1,1,3)     0.000000        0.000000
(1,1,1,4)     0.000000        0.000000

(1,1,2,1)     0.153846        0.162710
(1,1,2,2)     0.846154        0.837290
(1,1,2,3)     0.000000        0.000000
(1,1,2,4)     0.000000        0.000000

(1,2,1,1)     0.000000        1.000000
(1,2,1,2)     0.000000        0.000000
(1,2,1,3)     0.000000        0.000000
(1,2,1,4)     0.000000        0.000000

(1,2,2,1)     0.000000        0.113121
(1,2,2,2)     0.000000        0.886879
(1,2,2,3)     0.000000        0.000000
(1,2,2,4)     0.000000        0.000000

(1,3,1,1)     0.000000        0.000000
(1,3,1,2)     0.000000        0.000000
(1,3,1,3)     1.000000        1.000000
(1,3,1,4)     0.000000        0.000000

(1,3,2,1)     0.000000        0.000000
(1,3,2,2)     0.000000        0.000000
(1,3,2,3)     0.500000        0.478950
(1,3,2,4)     0.500000        0.521050

(2,1,1,1)     0.000000        0.978099
```

```
(2,1,1,2)     0.000000        0.021901
(2,1,1,3)     0.000000        0.000000
(2,1,1,4)     0.000000        0.000000

(2,1,2,1)     0.000000        0.013650
(2,1,2,2)     0.000000        0.986350
(2,1,2,3)     0.000000        0.000000
(2,1,2,4)     0.000000        0.000000

(2,2,1,1)     0.979557        0.975611
(2,2,1,2)     0.020443        0.024389
(2,2,1,3)     0.000000        0.000000
(2,2,1,4)     0.000000        0.000000

(2,2,2,1)     0.009709        0.013447
(2,2,2,2)     0.990291        0.986553
(2,2,2,3)     0.000000        0.000000
(2,2,2,4)     0.000000        0.000000

(2,3,1,1)     0.000000        0.000000
(2,3,1,2)     0.000000        0.000000
(2,3,1,3)     1.000000        1.000000
(2,3,1,4)     0.000000        0.000000

(2,3,2,1)     0.000000        0.000000
(2,3,2,2)     0.000000        0.000000
(2,3,2,3)     1.000000        1.000000
(2,3,2,4)     0.000000        0.000000

Observations:
* Typically one can find several entries that are 0 for CPT_hard, but close to 1 for CPT_soft.
* Each vertical group of 4 entries above defines a check_sum entry and should be 1.
  As we already know, that is often not the case, especially for bnet_hard.

------ PART II - USING TRAINED NETWORK (bnet_soft) FOR INFERENCE ------

---------- DEMONSTRATING HOW TO CALCULATE WEIGHTS -----------
Hit ENTER to continue

Generate new sample from original network (mixed discrete-continuous):
1.000000 2.000000 -3.193085 1.000000 -1.942921 2.734501 -0.568029 2.133517 1.481560

Convert sample case to Weights
Node 1:
   Boundaries:  None   (node already discrete)
   Sample Value: 1.000000
   Corresponding Weights:  1.000  0.000
Node 2:
   Boundaries:  None   (node already discrete)
   Sample Value: 2.000000
   Corresponding Weights:  0.000  1.000
Node 3:
   Boundaries:  -Inf -3.550000 -1.850000 Inf
   Sample Value: -3.193085
   Corresponding Weights:  0.000  1.000  0.000
Node 4:
   Boundaries:  None   (node already discrete)
   Sample Value: 1.000000
   Corresponding Weights:  1.000  0.000
```

```
Node 5:
   Boundaries:  -Inf -2.500000 -1.500000 -0.500000 Inf
   Sample Value: -1.942921
   Corresponding Weights:  0.000  1.000  0.000  0.000
Node 6:
   Boundaries:  -Inf 3.200000 4.700000 7.000000 Inf
   Sample Value: 2.734501
   Corresponding Weights:  1.000  0.000  0.000  0.000
Node 7:
   Boundaries:  -Inf 0.000000 Inf
   Sample Value: -0.568029
   Corresponding Weights:  1.000  0.000
Node 8:
   Boundaries:  -Inf 2.800000 3.700000 4.700000 Inf
   Sample Value: 2.133517
   Corresponding Weights:  1.000  0.000  0.000  0.000
Node 9:
   Boundaries:  -Inf -0.200000 0.500000 1.800000 Inf
   Sample Value: 1.481560
   Corresponding Weights:  0.000  0.000  1.000  0.000

For this network and small deviation the weights often result as exactly 0 or 1.
However, if you run this a few times, you will see other cases, too.

----------- DEMONSTRATING INFERENCE -----------
Using the sample vector from above, estimate the values of nodes 6,7,8
based on the values of all other nodes.

Hit ENTER to continue

Calculate State Means ...
Done

Inference Results for Nodes 6,7 and 8:

Node: 6
Probabilities:    0.969558 0.030442 0.000000 0.000000
State means:      2.737457 3.703481 5.747650 7.368571
Estimate value:   2.766865
Actual   value:   2.734501

Node: 7
Probabilities:    1.000000 0.000000
State means:      -0.502786 0.499002
Estimate value:   -0.502786
Actual   value:   -0.568029

Node: 8
Probabilities:    0.970883 0.027450 0.001667 0.000000
State means:      2.302559 3.156884 4.184094 5.523484
Estimate value:   2.329146
Actual   value:   2.133517
>>
```

## 5.4 Discussion of Results

As mentioned above running the script sample_script.m yields slightly different results every time, because the training data and test data are generated by sampling the mixed incinerator network. However, the trends

always remain the same. Specific observations for the results generated by the script:

- **PART I – Check_sum entries**
  In most cases for the network obtained using *hard discretization* 30 (out of 35) check_sum entries are one, and the remaining 5 are not one, indicating several undefined CPTs. The main reason for this is Node 6, which is the node with the largest number of parents in the network, namely three parents. Node 6 has 64 CPT entries and given the small sample size (1,000), it is not surprising that not all parent state combinations occur in the training data. Interestingly, even if we use 100,000 samples (instead of 1,000) typically 4 of the 35 check_sum entries are still undefined, i.e. even for large sample size not all parent combinations are likely to occur. Thus, even for large sample size, hard discretization seems to result in many undefined CPTs for this example, making the use of soft discretization a necessity.

  In contrast, the 35 check_sum entries for *soft discretization* - even for a deviation of only $5\%$ and only 1000 samples - are *usually* all one. However, this is not always the case - occasionally one or more entries are not one for a deviation of $5\%$. Therefore it is important to always apply the check_sum test after training a network, and, if necessary, training the network again with an increased deviation value, until all CPT entries are well defined.

  One risk of using a Bayesian Network with undefined CPTs for inference is that it tends to provide erroneous results for rare parent state combinations. For example if a certain parent state combination does not occur very often, chances are that it does not occur in the training data. Using hard discretization the CPTs for that parent state combination are all set to zero. If this rare parent state combination happens to ever occur later as evidence and inference is performed, the network will return as default the *lowest state* of the child node as the *only possible state* for this parent state combination, without providing any warning that this estimate is based on undefined CPT values.

- **PART I – CPT comparison for Node 6**
  Node 6 has three parents and is the node with the largest number of parents in the network. This node is generally the most likely to have undefined CPTs using hard discretization and is the main culprit for check_sum entries that are not one for hard discretization.

  The output lists all 64 CPT values of Node 6 for both hard and soft discretization. Looking at the actual CPT values we note the following:

  - Typically we can find several entries that are 0 for CPT_hard, but close to 1 for CPT_soft.
  - Each vertical group of 4 entries in the results defines a check_sum entry and should be 1.; As we already know, that is often not the case, especially for bnet_hard.

- **PART II – Calculating Weights**
  For the given parameters the weights often result as exactly 0 or 1. The reason is that the boundaries for this network are spaced fairly wide and the deviation value is chosen small ($5\%$ of continuous value). Therefore a sample value must lie fairly close to one of the interval boundaries in order for the corresponding weights to differ from 0 or 1. However, if you execute this several times, you will usually see values other than 0 or 1, too.

- **PART II – Inference Results for Nodes 6, 7 and 8**
  The results first show the probabilities, which are obtained through *soft* inference. The soft inference contains two "soft" components: (1) The CPTs of the network were obtained by soft discretization and (2) the evidence was converted to soft evidence before it was entered into the network.

  The *estimate value* is obtained by multiplying the probabilities by the state means and adding the result, which yields an expected value. The estimate value thus provides a continuous output. In our simulations this continuous value always was fairly close to the actual value.

# 6 Contact

For any questions or suggestions, please send e-mail to ebert@stups.com.

# References

[1] I. Ebert-Uphoff. A probability-based approach to soft discretization for bayesian networks. Technical report, School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA (USA), Sept 2009. Report Number GT-ME-2009-002.

[2] Steffen L. Lauritzen. Propogation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87(420):1098–1108, Dec 1992.

# A sample_script.m

The sample code is fairly self-explanatory and is included below just for ease of reference.

```
% sample_script.m
%
% Copyright (c) 2009, Imme Ebert-Uphoff (ebert@stups.com).
% This file is part of the "BNT Soft Discretization Package".
% Available at DataOnStage.com.
% Permission to use: see license.txt in this directory.
%
% Sample Script to explain the use of the BNT Soft Discretization Package.
% Copy groups of lines into the command window of Matlab or run the whole
% script at once.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PART I: Create training data, then initialize, train and    %
%         evaluate the discretized network                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%  1) Create Training Data   %%%%%%%%%%%%%%%

% Create BN with both discrete and continuous nodes
fprintf('\n----------- CREATING INCINERATOR NETWORK ---------\n');
% THIS MODEL IS FROM THE BNT DISTRIBUTION,
% see BNT/examples/static/Models/mk_incinerator_bnet.m
% But with 2 changes:  all nodes are defined to be observed and
% long node names are used.  Everything else is the same!
bnet_mixed = mk_incinerator_bnet_modified();
%graph_plain_to_dot(bnet_mixed, 'incinerator_original.dot', ...
%      'use_node_names', true, 'shape1', 'box', 'shape2', 'ellipse');

% Create 1000 samples from original incinerator network
N = length(bnet_mixed.dag);
nsamples = 1000;
fprintf('Creating %d samples from original network\n', nsamples);
samples = cell(N, nsamples);
for i=1:nsamples
   samples(:,i) = sample_bnet(bnet_mixed);
end

%%%%%%%%%%  2) DEFINE PARAMETERS FOR SOFT DISCRETIZATION   %%%%%%%%%%

% Choose discretization type and discretization intervals for each node
% dnodes are already discrete
for (node=bnet_mixed.dnodes)   % for all discrete nodes
   discretization_type{node} = 'is_discrete';
   boundaries{node}=[];  % empty set, because node is already discrete
end

% Define discretization type for continuous nodes
% for continuous nodes we can choose from 'soft' or 'hard' discretization
for (node=bnet_mixed.cnodes)   % for all continuous nodes
   discretization_type{node} = 'soft';
   deviation(node) = 0.05;          % deviation is 5% of continuous value
   deviation_is_factor(node) = true;
   % Effective deviation is:  deviation(i) * continuous_value
end
```

```
% Choose discretization boundaries for continouous nodes
% Method used here: Generate a histogram for each node,
% then choose boundaries by hand.
% Example:
%    node = 6    % for a single node
%%%% generate vector of all sample values for current node
%    single_node_data = cell2num( samples(node,:) );
%    hist(single_node_data,100)  % generate histogram
%% Then select boundaries by hand (visual inspection) from histogram figure.

F = 1; W = 2; E = 3; B = 4; C = 5; D = 6; Min = 7; Mout = 8; L = 9;
boundaries{E} =    [-Inf, -3.55, -1.85, Inf];     % Node 3
boundaries{C} =    [-Inf, -2.5, -1.5, -0.5, Inf]; % Node 5
boundaries{D} =    [-Inf, 3.2, 4.7, 7, Inf];      % Node 6
boundaries{Min} =  [-Inf, 0, Inf];                % Node 7
boundaries{Mout} = [-Inf, 2.8, 3.7, 4.7, Inf];    % Node 8
boundaries{L} =    [-Inf, -0.2, 0.5, 1.8, Inf];   % Node 9


%%%%%%%%%  3) CREATE CORRESPONDING NETWORK WITH ONLY DISCRETE NODES %%%%%%%
fprintf('\n---------- CREATE DISCRETE BN (with random CPTs):   bn_soft\n');
bnet_soft = create_discrete_bnet(bnet_mixed, boundaries);
%graph_plain_to_dot(bnet_soft, 'incinerator_discrete.dot', ...
%        'use_node_names', true, 'shape1', 'box', 'shape2', 'ellipse');

%%%%%%%%%  4) TRAIN DISCRETE NETWORK USING SOFT DISCRETIZATION %%%%%%%%%%%%

fprintf('\n------ TRAIN bnet_soft USING SOFT DISCRETIZATION -------\n');
show_CPT = false;  % Should I print Conditional Probability Tables for all nodes?
show_JPT = false;  % Should I print Joint Probability Tables for all nodes?
[bnet_soft] = soft_CPT_for_all_nodes( bnet_soft, samples, boundaries, ...
  discretization_type, deviation, deviation_is_factor, show_CPT, show_JPT);

%%%%%%%%%  5) CREATE AND TRAIN ANOTHER DISCRETE NETWORK USING HARD DISCRETIZATION %%%

% For comparison - train another discrete network using HARD discretization
fprintf('\n---------- CREATE DISCRETE BN (with random CPTs):   bn_hard\n');
bnet_hard = create_discrete_bnet(bnet_mixed, boundaries);

fprintf('\n------ TRAIN bnet_hard USING HARD DISCRETIZATION -------\n');
for (node=bnet_mixed.dnodes)   % for all discrete nodes
   hard_discretization_type{node} = 'is_discrete';
end
for (node=bnet_mixed.cnodes)   % for all continuous nodes
   hard_discretization_type{node} = 'hard';    % request hard discretization
end
[bnet_hard] = soft_CPT_for_all_nodes( bnet_hard, samples, boundaries, ...
    hard_discretization_type, deviation, deviation_is_factor, show_CPT, show_JPT );

% FOR TESTING: COMPARE TO OLD CODE:
% CPT_for_all_nodes_old( bnet_hard, discretization_type, samples, boundaries, ...
% deviation, deviation_is_factor );

%%%%%%%%%  6) APPLY CHECK_SUM TEST TO BOTH DISCRETE NETWORKS %%%%%%%%%%%%%%%%%%

fprintf('----- Highlighting Differences between hard and soft CPTs -----\n');
dummy = input('Hit ENTER to continue\n');

%%% Check how many CPTs are well defined.
```

```
fprintf('\nCHECK SUM entries:\n');
fprintf('We WANT all entries to be one.  That means all CPTs are well defined.\n');
want_short_print_out = true;  % use 'false' to see the actual entries

fprintf('\nCheck Sum entries for CPTs of   bnet_hard:\n');
[ones, not_ones] = check_sum(bnet_hard, 0.00001, want_short_print_out);

fprintf('\nCheck Sum entries for CPTs of   bnet_soft:\n');
[ones, not_ones] = check_sum(bnet_soft, 0.00001, want_short_print_out);

%%%%%%%%%  7) HIGHLIGHT CPT DIFFERENCES BETWEEN DISCRETE NETWORKS %%%%%%%%%%
node = 6;
fprintf('\nShow CPT for Node %i = node with the most parents in the network.\n', node);
dummy = input('Hit ENTER to continue\n');

s_hard = struct(bnet_hard.CPD{node});  % violate privacy to get CPDs
s_soft = struct(bnet_soft.CPD{node});
t = size(s_hard.CPT);
fprintf('Index set     CPT_hard        CPT_soft\n');
for i1= 1 : t(1)  % child with 3 parents has 4-dimensional CPT table
   for i2= 1 : t(2)
      for i3 = 1 : t(3)
         for i4 = 1 : t(4)
            fprintf( '(%i,%i,%i,%i)     %f        %f\n', i1,i2,i3,i4, ...
               s_hard.CPT(i1,i2,i3,i4), s_soft.CPT(i1,i2,i3,i4) );
         end
         fprintf('\n');
      end
   end
end
fprintf('Observations:\n');
fprintf('* Typically one can find several entries that are 0 for CPT_hard,');
fprintf(' but close to 1 for CPT_soft.\n');
fprintf('* Each vertical group of 4 entries above defines a check_sum entry and should be 1.\n');
fprintf('  As we already know, that is often not the case, especially for bnet_hard.\n\n');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PART II: Use the network to calculate weights and           %
%          demonstrate inference                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('------ PART II - USING TRAINED NETWORK (bnet_soft) FOR INFERENCE ------\n');

%%%%%%%%%  1) Calculate Soft Discretization Weights for a Sample %%%%%%

fprintf('\n---------- DEMONSTRATING HOW TO CALCULATE WEIGHTS -----------\n');
dummy = input('Hit ENTER to continue\n');

% Create New Sample from original network (mixed discrete + continuous)
case_mixed_vec = cell2num( sample_bnet(bnet_mixed) );
fprintf('Generate new sample from original network (mixed discrete-continuous):\n');
for i=1:length(case_mixed_vec)
   fprintf('%f ', case_mixed_vec(i) );
end
fprintf('\n');

% Illustrate weight calculation
fprintf('\nConvert sample case to Weights\n');
```

```
N_nodes = length(bnet_mixed.dag);
node_list = [1:N_nodes];  % calculate weights for ALL nodes
p_weight = soft_discretization_weights( bnet_soft, case_mixed_vec, node_list, ...
            boundaries, discretization_type, deviation, deviation_is_factor );
for node = 1:N_nodes
   fprintf('Node %i:   \n   Boundaries: ', node);
   if ( length(boundaries{node})==0 )
      fprintf(' None   (node already discrete)');
   else
      for i=1:length(boundaries{node})
        fprintf(' %f', boundaries{node}(i) );
      end
   end
   fprintf('\n   Sample Value: %f ', case_mixed_vec(node));

   fprintf('\n   Corresponding Weights: ');
   for i=1:length(p_weight{node})
      fprintf(' %2.3f ',p_weight{node}(i));
   end
   fprintf('\n');
end
fprintf('\nFor this network and small deviation the weights often result as exactly 0 or 1.\n');
fprintf('However, if you run this a few times, you will see other cases, too.\n\n');

%%%%%%%%%  2) Demonstate Inference %%%%%%

fprintf('------------ DEMONSTRATING INFERENCE ------------\n');
fprintf('Using the sample vector from above, estimate the values of nodes 6,7,8\n');
fprintf('based on the values of all other nodes.\n\n');
dummy = input('Hit ENTER to continue\n');

%%% SET UP INFERENCE ENGINE WITH EVIDENCE
N_nodes = length(bnet_soft.dag);
all_nodes = [1:N_nodes];
test_nodes = [6,7,8];
evidence_nodes = [1,2,3,4,5,9];  % All but 6,7,8
% ENTER EVIDENCE FOR evidence_nodes
engine = soft_inference( bnet_soft, case_mixed_vec, evidence_nodes, ...
         boundaries, discretization_type, deviation, deviation_is_factor );
% engine is now ready for inference

%%%%%%%%%  3) Demonstate Creating Continuous Output Estimates %%%%%%

% for each originally continuous node, calculate the mean value of each state
% from the training data - this needs to be done only once for the training data
state_mean  = init_state_means( samples, bnet_mixed.cnodes, boundaries );

% EXTRACT INFERENCE RESULTS
fprintf('\nInference Results for Nodes 6,7 and 8:\n');
for node = test_nodes
   expected_value = [];
   marg = marginal_nodes(engine, node);  % Marginal probabilities for node
   fprintf('\nNode: %i\n', node);
   p = marg.T'; %'
   fprintf('Probabilities:    ');
   for i=1:length(p)
      fprintf('%f ', p(i));
   end
   fprintf('\n');
```

```
   % Calculate expected value
   fprintf('State means:      ');
   vec = state_mean{node};
   for i=1:length(vec)
      fprintf('%f ', vec(i) );
   end
   fprintf('\n');
   expected_value(node) = p * state_mean{node}';  %'

   fprintf('Estimate value:   %f\n', expected_value(node));
   fprintf('Actual   value:   %f\n', case_mixed_vec(node));
end
```