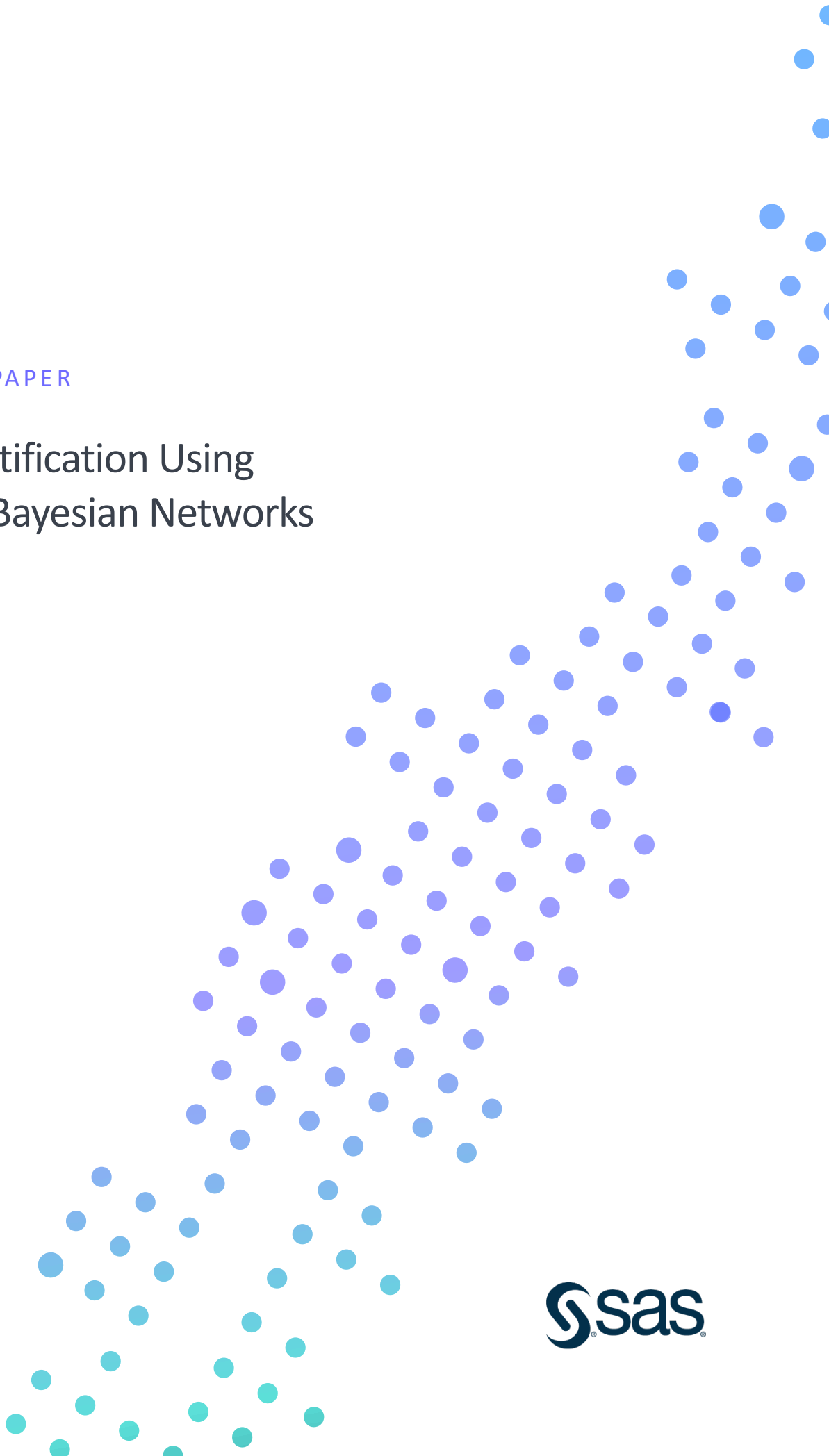
TECHNICAL PAPER

# Fault Identification Using Dynamic Bayesian Networks

Last update: July 2024



# Contents



## Relevant Products and Releases

- SAS® Viya® 4
  - PROC DYNBNET
- SAS® Event Stream Processing
  - Dynamic Bayesian network plug-in

# Introduction

## Fault Identification with IIoT

Malik et al. (2021) define the Industrial Internet of Things (IIoT) as "the concept of connecting any device with the internet." For example, industrial assets are often equipped with sensors that capture information about the asset. These sensor data can then be fed into analytical models to monitor and assess the health of the asset. More specifically, some analytical models can be used to identify potential fault scenarios for the asset.

## Fault Identification Using Dynamic Bayesian Networks

This paper introduces the new dynamic Bayesian network functionality available in SAS® Viya® 4, which enables you to identify fault states in assets by using data from sensors in those assets. In SAS Viya 4, the DYNBNET procedure (SAS Institute Inc. 2024a) implements dynamic Bayesian networks.

## Dynamic Bayesian Networks

Bayesian networks consist of nodes and edges. Nodes represent variables, and directed edges represent relationships among these variables. The variables are a mix of unobserved variables, which this paper refers to as "hidden variables," and observed variables. Dynamic Bayesian networks model the relationships among variables in a Bayesian network over time. Because of their temporal design, dynamic Bayesian networks can update the distributions of hidden variables as new values of observed variables become available over time. For industrial assets, if the relationships between the hidden fault states of the assets and the sensor data are known, then dynamic Bayesian networks can be used to update the distributions of the fault states as new values of sensor data become available over time.

## Audience for This Paper

The audience for this paper includes data scientists and engineers in the field of machine or system monitoring who understand the relationships between fault states of a system and related sensor measurements. The methodology outlined in this paper can detect fault states as well as identify which fault states the system is experiencing.

# Overview of Dynamic Bayesian Networks

The dynamic Bayesian network procedure (PROC DYNBNET) in SAS Viya 4 enables you to estimate variable information (means and standard deviations for hidden interval variables, and probability distributions for hidden nominal variables), given a model of the relationships between hidden and observed variables and the observations that provide sensor data values over time. PROC DYNBNET requires the following specifications:

1. *Variable names and their roles*. You must specify all variables in the model along with the role of each variable in the model. Variable roles include control, hidden, observed, and time index identifier (time ID).

    a. *Control variables*. A control variable is an observed nominal variable that is a parent of interval variables. An example of a control variable is the action taken by a climate-control system in a

building. If the system is programmed to maintain the temperature between 67 and 72 degrees Fahrenheit, then the system switches to cooling when the indoor temperature reads greater than 72 degrees and heating when the indoor temperature reads less than 67 degrees. If the temperature is between 67 and 72 degrees, the system remains idle. Therefore, the action that the system takes is managed as a control variable.

b. *Hidden and observed variables*. A hidden variable is an unobserved variable, in contrast to an observed variable. For example, with the climate-control system, perhaps the actual temperature inside the building is unknown; it is therefore a hidden variable. Further suppose that there is a temperature sensor in the thermostat for the climate-control system that measures the indoor temperature, and this sensor reading for the indoor temperature is an observed variable. If the sensor has an internal measurement error, its measurement can be different from the actual indoor temperature.

c. *Time index identifier (time ID)*. There should be exactly one variable whose role is time ID. This variable provides the time index for the observed variables.

2. *Levels of nominal variables*. You must specify the possible levels for all nominal variables. For a fault variable, which is usually a hidden nominal variable, a potential specification of the fault levels is "working" and "failing." Another option is "0" and "1," where "1" indicates that the fault is present and "0" indicates that the fault is absent. For this type of specification, the system might have several faults that can occur, and these faults could occur simultaneously. Consider a chemical process that requires several internal systems, including a separator and a stripper. There could be a fault in both the separator and the stripper at the same time. Imagine another fault specification scenario, where a single fault variable can take different states but only one fault state can occur at a time. For example, a pipe in an industrial plumbing system could have a sudden and complete block, or it could have a gradual block, but it is impossible for both types of blocks to occur at the same time. In this case, the levels of the fault variable for a block could be "none," "gradual," and "complete." (Note that this paper uses the terms "level" and "branch" interchangeably.)

3. *Parent-child relationships in the two-time-period Bayesian network*. You must specify all (known) relationships between variables in the model. These relationships can be specified either in the current (or same) time period or in previous (or across) time periods. Relationships that are defined by ordinary differential equations (ODEs) exist across time periods, whereas most other relationships in the network, including functional relationships, occur within the same time period. For example, a fault state might affect some hidden variable through an ODE, and there is a sensor that measures this hidden variable and produces a reading that has some internal measurement error. Therefore, the relationship between the hidden variable and the sensor (observed) variable exists within the same time period, but the relationship between the fault state and the hidden interval variable exists across time periods (because of the nature of the ODE).

4. *Initial beliefs of hidden variables*. The initial beliefs of the hidden variables provide a starting point for the dynamic Bayesian network model to estimate the updated beliefs of the hidden variables in subsequent time periods. For a belief in a fault variable, an initial specification could be a 95% chance that the system has no fault and a 5% chance that the system is experiencing a fault. For hidden interval variables, the initial beliefs are specified as means and standard deviations.

5. *FCMP procedure for the model*. The most important specifications for the dynamic Bayesian network are

the PROC FCMP (SAS Institute Inc. 2017) subroutines that define the behavior of the system of interest. This is the part of the model specification that must be defined by data scientists and engineers who understand the relationships between fault states and other variables, including sensor measurements and the hidden variables. A simple example is a system that contains a pipe, such as a plumbing system. If the pipe is blocked, then the conductance of the pipe is 0, rather than 1, which is the normal (or nonfault) state. Or if the pipe has a gradual block, the conductance is *changing over time* (that is, reducing at some rate from 1 to 0); this can be specified using a differential equation. Any differential equations present in the model are specified by using PROC FCMP. Relationships between hidden interval variables and observed interval variables—for example, the relationship between a hidden variable and a sensor with an internal measurement error that is measuring the value of this hidden variable—are also specified by using PROC FCMP.

6. *Observed data*. You must specify a table that contains Time (the variable that has the time ID role) and the values for observed variables, either as inline data or by reference to a SAS data table. The dynamic Bayesian network can then estimate the beliefs of hidden variables for as many observations (or rows) as the observed data table contains.

The output from PROC DYNBNET is a table that displays estimated means and standard deviations for hidden interval variables and estimated probability distributions for hidden nominal variables at each time when observed variable values are provided (see specification 6). If the main goal of modeling is to identify faults, then the probability distributions for each fault variable are of interest.

### Saving the State of a Model in PROC DYNBNET

Another feature of PROC DYNBNET is its ability to save the state of a model that you previously specified, including variables, network, and outgoing beliefs. The OUTPUTSTATE= option in the PROC DYNBNET statement enables you to save this information to an output data set. You can use this information in a subsequent PROC DYNBNET invocation by specifying the INPUTSTATE= option and new observed data. Therefore, specifications 1–5 are not needed, but specification 6 is still needed for observed data.

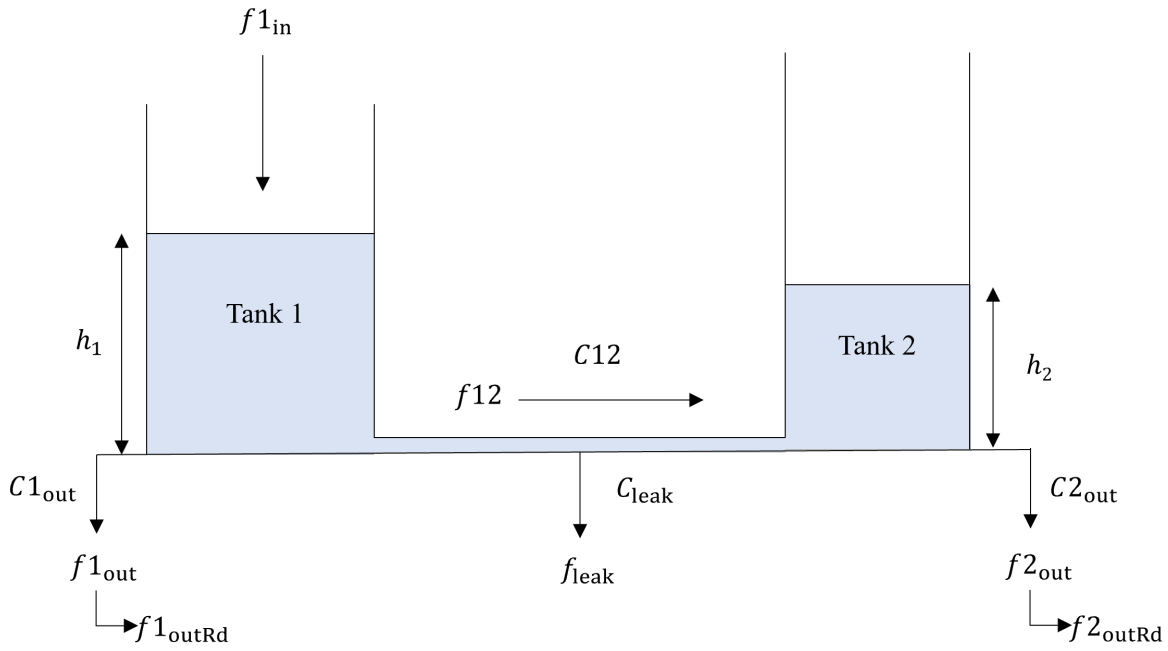
Another use of this saved model information is in a SAS Event Stream Processing model (SAS Institute Inc. 2024c). You can read a saved PROC DYNBNET model into an Event Stream Processing model by using a source for streaming data, and then you can pass that model through the new dynamic Bayesian network model plug-in (SAS Institute Inc. 2024d) available in SAS Event Stream Processing software. The two-tank example in the next section shows how to use a saved two-tank model in this software to score streaming data.

## Two-Tank Example

This example demonstrates how to use a dynamic Bayesian network model for a hypothetical, although somewhat realistic, example of a system of two tanks. This example is similar to the five-tank scenario from Lerner et al. (2000). For our example, there are two tanks connected by a pipe. There is liquid flowing into Tank 1 at a flow rate of $f1_{\text{in}}$ and flowing out of Tank 1 at a flow rate of $f1_{\text{out}}$, where $C1_{\text{out}}$ denotes the conductance (that is, the reciprocal of resistance) of the pipe coming out of Tank 1. As mentioned previously, there is a pipe that connects Tank 1 and Tank 2, and liquid is flowing through it from Tank 1 to Tank 2 at a flow rate of $f12$, where $C12$ denotes the conductance of this connecting pipe. Liquid is also flowing out of Tank 2 at a flow rate of $f2_{\text{out}}$, where $C2_{\text{out}}$ denotes the conductance of the pipe coming out of Tank 2. Let $A_1$ and $A_2$ denote the cross-sectional areas of Tank

1 and Tank 2, respectively, and let $h_1$ and $h_2$ denote the heights of the liquid in Tank 1 and Tank 2, respectively. A potential fault in this system is a leak in the pipe that goes from Tank 1 to Tank 2. When this leak fault is present, liquid leaks out of the pipe at a flow rate of $f_{\text{leak}}$, where $C_{\text{leak}}$ denotes the conductance of the leak in the pipe. Finally, $f1_{\text{outRd}}$ and $f2_{\text{outRd}}$ are the sensor measurements for $f1_{\text{out}}$ and $f2_{\text{out}}$, respectively, with measurement error. The diagram of this system is shown in Figure 1.

***Figure 1.*** *Two-Tank Model*




The flow variables are related to the conductance variables through the following mathematical relationships:

$$f1_{\text{out}} = C1_{\text{out}} \times h_1$$

$$f12 = C12 \times (h_1 - h_2)$$

$$f_{\text{leak}} = C_{\text{leak}} \times h_1$$

$$f2_{\text{out}} = C2_{\text{out}} \times h_2$$

The material balances in Tanks 1 and 2 are shown by the following two mathematical relationships, respectively, where $\frac{dh_1}{dt}$ and $\frac{dh_2}{dt}$ are the changes in the heights of the liquid in Tanks 1 and 2, respectively, by time:

$$A_1 \times \frac{dh_1}{dt} = f1_{\text{in}} - f12 - f_{\text{leak}} - f1_{\text{out}}$$

$$A_2 \times \frac{dh_2}{dt} = f12 - f2_{\text{out}}$$

Rearranging terms in the material balance equations produces the following ordinary differential equations (ODEs) for $\frac{dh_1}{dt}$ and $\frac{dh_2}{dt}$:

$$\frac{dh_1}{dt} = \frac{1}{A_1}[f1_{\text{in}} - (C12 + C_{\text{leak}} + C1_{\text{out}})h_1 + (C12)h_2] \quad (1)$$

$$\frac{dh_2}{dt} = \frac{1}{A_2}[(C12)h_1 - (C_{12} + C2_{\text{out}})h_2] \quad (2)$$

As mentioned previously, a potential fault in this system is a leak in the pipe that goes from Tank 1 to Tank 2. You can specify the change in the conductance of leak in the pipe ($C_{\text{leak}}$) by $leak\_rate$ as follows:

$$\frac{dC_{\text{leak}}}{dt} = leak\_rate \quad (3)$$

Equations (1), (2), and (3) are specified in PROC FCMP as ODEs that characterize the model. The hidden nominal variable for the fault state can take either of two values: None or Leak.

The sensor variables, $f1_{\text{outRd}}$ and $f2_{\text{outRd}}$, have the following models with respect to the known parameters $C1_{\text{out}}$ and $C2_{\text{out}}$ and the hidden variables $h_1$ and $h_2$:

$$f1_{\text{outRd}} = C1_{\text{out}} \times h_1 + \epsilon_1 \quad (4)$$

$$f2_{\text{outRd}} = C2_{\text{out}} \times h_2 + \epsilon_2 \quad (5)$$

For these models, $\epsilon_i \sim N(0, \sigma_i^2), i = 1, 2$, where $\sigma_i^2$ is the variance of the measurement error in the sensors for $f1_{\text{out}}$ and $f2_{\text{out}}$. Equations (4) and (5) are also specified in PROC FCMP for the model.

Given this information, you can now start to specify the information that PROC DYNBNET needs in order to estimate updated beliefs in the fault state by using observed sensor variable values.

Following the same outline as in the section Overview of Dynamic Bayesian Networks, you start by specifying the variable names and their roles. Time has the role of time ID; $f1_{\text{outRd}}$ and $f2_{\text{outRd}}$ are the only two observed variables; and the rest of the variables are hidden. This model does not include any control variables. The following code specifies the variableRoles table:

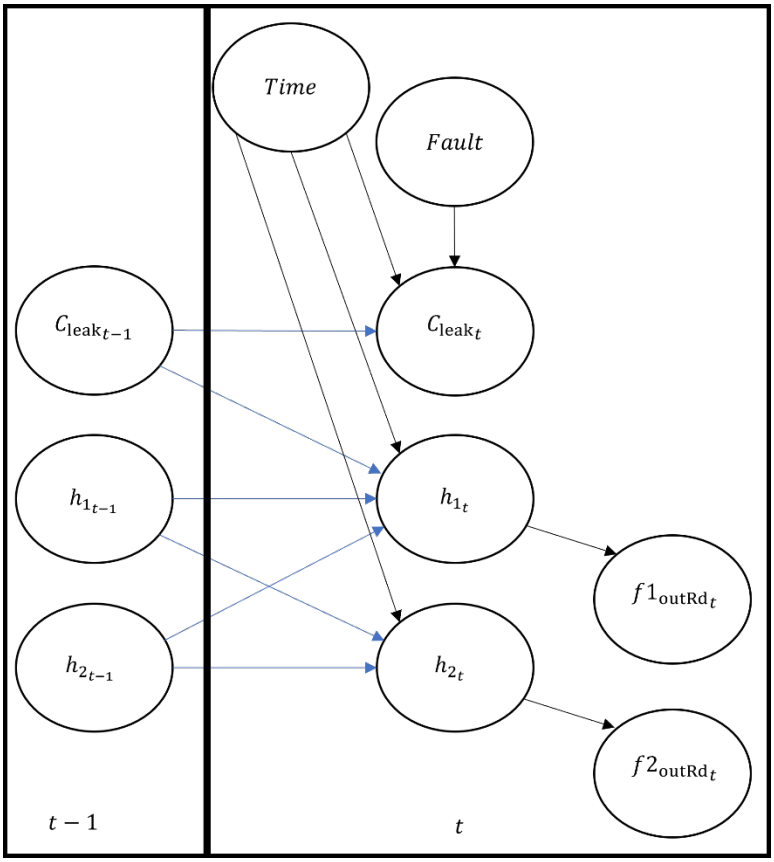
```
data casuser.variableRoles;
   length varname $20;
   input varname $ varrole $;
datalines;
Time         TIMEID
H1           HIDDEN
H2           HIDDEN
Cleak        HIDDEN
Fault        HIDDEN
F1out_rd     OBSERVED
F2out_rd     OBSERVED
;
```

The only nominal variable in the model is Fault, which can take either the value None or Leak. The following code specifies the variableLevels table:

```
data casuser.variableLevels;
   length varname $20;
   length varlevel $20;
   input varname $ varlevel $;
datalines;
Fault  None
Fault  Leak
;
```

You are now ready to specify the parent-child relationships (or "links") in the two-time-period network. The structure of the links table includes the parent, the child, and the stage. In this table, Stage is a numeric variable that can take either of two possible values, 1 or 2. A Stage value of 1 indicates that the parent-child relationship is in the current time period; a Stage value of 2 indicates that the parent variable in the previous time period is connected to the child variable in the current time period. Time is a parent of all variables that are defined by an ODE (that is, $h_1, h_2, C_{\text{leak}}$), and these parent-child relationships have a Stage value of 1. Moreover, all variables that are defined by an ODE are also parents of themselves across time periods (that is, they have a Stage value of 2). Digging deeper into the ODEs, $dh_1/dt$ includes $C_{\text{leak}}$ and $h_2$, so $C_{\text{leak}}$ and $h_2$ are parents of $h_1$ across time periods; and $dh_2/dt$ includes $h_1$, so $h_1$ is a parent of $h_2$ across time periods. For the functional relationships, $h_1$ is a parent of $f1_{\text{outRd}}$ in the current time period, and $h_2$ is a parent of $f2_{\text{outRd}}$ in the current time period. Finally, the variable Fault affects $leak\_rate$ and therefore $C_{\text{leak}}$, so Fault is a parent of $C_{\text{leak}}$ in the current time period. Figure 2 shows the two-time-period Bayesian network for the two-tank system. This diagram enables you to visualize all parent-child relationships and to determine the correct value of Stage for each link. Black links indicate a Stage value of 1; blue links indicate a Stage value of 2.

***Figure 2.*** *Two-Time-Period Bayesian Network for the Two-Tank Model*



The following code specifies the variableLinks table, which includes all parent-child relationships:

```
data casuser.variableLinks;
   length parent $20 child $20;
   input  parent $ child $ stage;
datalines;
Time            H1          1
Time            H2          1
Time            Cleak       1
H1              H1          2
H2              H2          2
Cleak           Cleak       2
Cleak           H1          2
H2              H1          2
H1              H2          2
H1              F1out_rd    1
H2              F2out_rd    1
Fault           Cleak       1
;
```

The next step is specifying initial beliefs of all hidden variables (that is, $h_1, h_2, C_{\text{leak}}$, and Fault). The structure of the initial beliefs table includes the time variable (Time), interval variable name (Intervalvar), mean (Mean), standard deviation (Std), nominal variable name (Nominalvar), and nominal variable level (Varlevel). Usually, Time is set to some initial value, such as 0 or 1. For this example, the initial Time value is 1. For $h_1$ and $h_2$, you can specify some initial mean height values along with a standard deviation. For this example, the initial belief in $h_1$ starts at 20, and the initial belief in $h_2$ starts at 2; both have standard deviations of 0.001. For the rows in the initial beliefs table for $h_1$ and $h_2$, the nominal variable name and nominal variable level are missing. This table has functionality for the case where a hidden interval variable might have different initial beliefs, depending on the level of a nominal variable. For $C_{\text{leak}}$, the initial mean is 0 (indicating that there is no leaking flow), with a standard deviation of 0.0001. For Fault, you can specify a 0.95 probability that the fault state is None and a 0.05 probability that the fault state is Leak. These values are specified in the mean column. Moreover, the interval variable name and standard deviation are missing for the initial belief for Fault. The following code specifies the initialBeliefs table:

```
data casuser.initialBeliefs;
   length intervalvar $20 nominalvar $20 varlevel $20;
   input Time intervalvar $ mean std nominalvar $ varlevel $;
datalines;
1       H1         20              0.001           .               .
1       H2          2              0.001           .               .
1       Cleak       0              0.0001          .               .
1       .           0.95            .              Fault           None
1       .           0.05            .              Fault           Leak
;
```

Next is the crucial part of defining the models (that is, ODEs and functional relationships) in the two-time-period Bayesian network. These are specified using PROC FCMP (SAS Institute Inc. 2017) according to the following guidelines:

- ODEs: The name of the subroutine must be "d_varname", where varname is the name of the child variable whose derivative with respect to time is defined by the ODE (for example, d_Cleak). Then, the first set of arguments for the subroutine includes the parents of the variable; for d_Cleak these are Time, $C_{\text{leak}}$, and Fault, where Fault is followed by a dollar sign ($) to indicate that it is a character variable. The last two arguments for the subroutine are the name of the subroutine, which returns the derivative of the variable with respect to time, given the values of the parent variables; and std, for standard deviation.

- Functional relationships: The name of the subroutine must be the name of the child variable in the relationship that this subroutine defines. The first set of arguments includes the parents of the variable. The last two arguments for the subroutine are mean and std, which return, respectively, the mean and standard deviation of the variable after applying the function, given the values of the parent variables.

The PROC FCMP subroutines for the two-tank system are specified as follows:

```
proc fcmp casoutlib = casuser.nonlinears.tank;

     /*****************************************************/
     /* subroutine to get derivative of variable Cleak    */
     /* over different branches of nominal variable;      */
     /* std is the noise in the ODE model of Cleak        */
     /*****************************************************/
     subroutine d_Cleak(Time, Cleak, Fault $, d_Cleak, std);
         outargs d_Cleak, std;
         if Fault = 'Leak' then do;
             d_Cleak = 1 / 120; * leak_rate;
             std = 0.0001;
         end;
         else do;
             d_Cleak = 0;
             std = 0.0001;
         end;
     endsub;

     /***********************************************************/
     /* subroutine to get derivative of variable H1             */
     /* as a function of variables Time, Cleak, H1, and H2;     */
     /* std is the noise in the ODE model of H1                 */
     /***********************************************************/
     subroutine d_H1(Time, Cleak, H1, H2, d_H1, std);
         outargs d_H1, std;
         A1 = 200;
         f1in = 1;
         C1out = 0.5;
         C12 = 1;
         d_H1 = (1/A1)*f1in - (C12+Cleak+C1out)/A1 * H1
                 + (C12)/A1 * H2;
         std = 0.001;
     endsub;

     /*********************************************************/
     /* subroutine to get derivative of variable H2           */
     /* as a function of variables Time, H1, and H2;          */
     /* std is the noise in the ODE model of H2               */
     /*********************************************************/
     subroutine d_H2(Time, H1, H2, d_H2, std);
         outargs d_H2, std;
         A2 = 100;
         C2out = 1;
         C12 = 1;
         d_H2 = C12/A2 * H1
                 - ((C12+C2out)/A2) * H2;
         std = 0.001;
     endsub;
```

```
    /*********************************************************/
    /* subroutine to get value of variable F1out_rd          */
    /* as a function of variable H1;                         */
    /* std is the noise in this function                     */
    /*********************************************************/
    subroutine F1out_rd(H1, mean, std);
        outargs mean, std;
        C1out = 0.5;
        mean = H1 * C1out;
        std = 0.0001;
    endsub;

    /*********************************************************/
    /* subroutine to get value of variable F2out_rd          */
    /* as a function of variable H2;                         */
    /* std is the noise in this function                     */
    /*********************************************************/
    subroutine F2out_rd(H2, mean, std);
        outargs mean, std;
        C2out = 1;
        mean = H2 * C2out;
        std = 0.0001;
    endsub;
quit;

proc fcmp setcascmplib="casuser.nonlinears" getcascmplib; run;
```

Finally, we specify the observed data for the variables $f1_{\text{outRd}}$ and $f2_{\text{outRd}}$. The observed data and code for the various scenarios are available on GitHub.

Now we have all the necessary information to call PROC DYNBNET as follows:

```
proc dynbnet data = casuser.tankData
          odeapprox = NONE
          outdetails = casuser.tankOutd;
    varroles data = casuser.variableRoles;
    varlevels data = casuser.variableLevels;
    links data = casuser.variableLinks;
    initbeliefs data = casuser.initialBeliefs;
    output out = casuser.tankOut;
run;
```

For this PROC DYNBNET call, the ODEAPPROX= option is set to NONE so that no approximation is used to solve the ODEs that the mean and gradient calculations use. The default setting of this option is GRADIENT, which approximately solves the ODEs that are used in the gradient calculations but does not use any approximation in solving the ODEs when calculating the means of hidden variables. The standard output from PROC DYNBNET comes from the OUTPUT statement, so the output is displayed in the data table tankOut. This output includes the outgoing beliefs of the hidden variables (both interval and nominal). The beliefs of interval variables differ by levels of nominal variables, so this table provides the interval variable beliefs for the most probable levels of the nominal variables. Another optional output is specified by the OUTDETAILS= option in the PROC DYNBNET statement; this output is displayed in the data table tankOutd. The difference between the standard output and the OUTDETAILS= option output is that the OUTDETAILS= option output includes beliefs for the hidden interval variables for all levels of the hidden nominal variables.

Data were simulated for the two-tank model in the IML procedure (SAS Institute Inc. 2018), based on the PROC FCMP subroutines. The simulations were for 100 time periods in three different scenarios: (1) no leak throughout

the simulation, (2) leak starting at the beginning of the simulation and continuing, and (3) leak starting after 60 time periods and continuing. In the sections that follow, PROC DYNBNET is applied to the data sets for these three simulation scenarios to evaluate whether it can accurately detect the presence of the leak fault.

## No Leak

We start by applying the PROC DYNBNET code for this two-tank example to simulated data without fault (that is, with no leak). For this example, Tables 1 and 2 show the differences between the output in the tankOut table (Table 1) and the tankOutd table (Table 2) for the first few observations in these two tables. The tankOutd table includes the beliefs for the hidden interval variables (that is, means and standard deviations) for each level of the hidden nominal variable (Fault), whereas the tankOut table includes beliefs for only the most probable branch of Fault. In Table 2, the first two rows provide the estimated probabilities for both branches of Fault for Time = 2. These rows indicate that None is the most probable branch of Fault (with a probability of 0.9550 for None versus 0.0450 for Leak), so the beliefs in hidden interval variables that Table 1 displays are for the None branch.

***Table 1.*** *Output in tankOut Table*

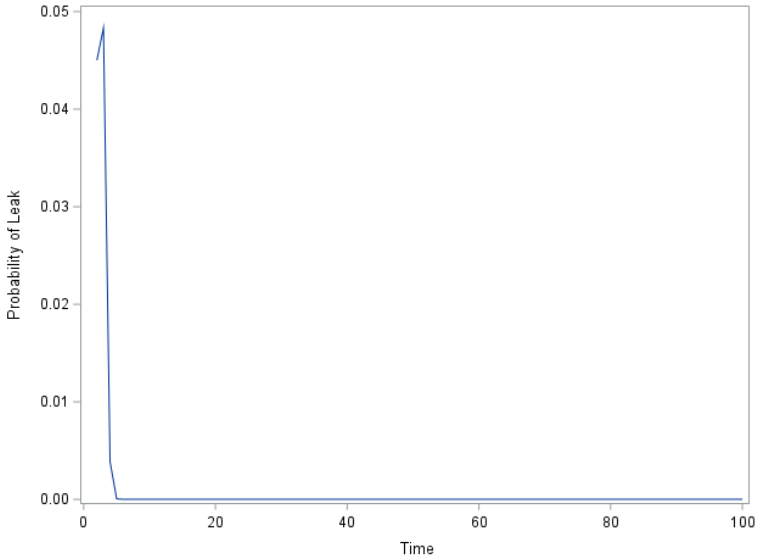
| Time | Cleak_MEAN | Cleak_SDEV | H1_MEAN | H1_SDEV | H2_MEAN | H2_SDEV | Fault_Leak | Fault_None |
|---|---|---|---|---|---|---|---|---|
| 2 | –0.000000164 | 0.000141420 | 19.8662 | 0.000198015 | 2.15762 | 0.000099746 | 0.045012 | 0.95499 |
| 3 | 0.000001091 | 0.000173193 | 19.7332 | 0.000196257 | 2.31113 | 0.000099508 | 0.048327 | 0.95167 |
| 4 | 0.000000311 | 0.000199970 | 19.6029 | 0.000196255 | 2.46004 | 0.000099508 | 0.003862 | 0.99614 |
| 5 | 0.000000033 | 0.000223548 | 19.4741 | 0.000196255 | 2.60477 | 0.000099508 | 0.000061 | 0.99994 |

***Table 2.*** *Output in tankOutd Table*

| Time | intervalvar | nominalvar | varlevel | mean | std |
|---|---|---|---|---|---|
| 2 | Fault | Fault | Leak | 0.0450 | . |
| 2 | Fault | Fault | None | 0.9550 | . |
| 2 | Cleak | Fault | Leak | 0.0083 | 0.000141420 |
| 2 | Cleak | Fault | None | –0.0000 | 0.000141420 |
| 2 | H1 | Fault | Leak | 19.8662 | 0.000198015 |
| 2 | H1 | Fault | None | 19.8662 | 0.000198015 |
| 2 | H2 | Fault | Leak | 2.1576 | 0.000099746 |
| 2 | H2 | Fault | None | 2.1576 | 0.000099746 |

The outgoing beliefs for Fault are shown in Figure 3. Although there is a slight chance of a leak in the first few time periods, the probability of a leak drops to about 0.0001 within five time periods, and it remains at this level for the rest of the provided time periods.

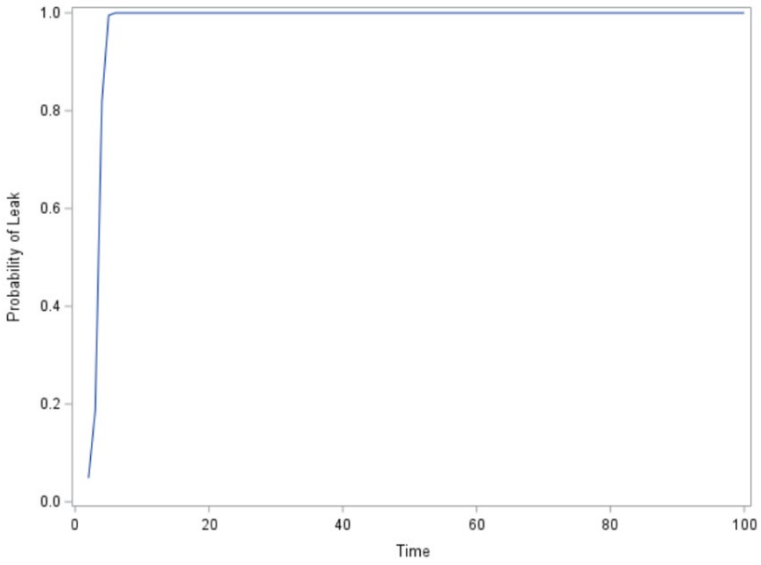
***Figure 3.*** *Probability of Leak Fault for Simulated Data without Fault*




## Leak

Next, we apply the PROC DYNBNET code for this two-tank example to simulated data for a fault (that is, a leak). The outgoing beliefs for the Fault variable are shown in Figure 4. Within one or two time periods, PROC DYNBNET is able to identify the presence of a leak, and the probability of the leak remains high for the rest of the provided time periods.

***Figure 4.*** *Probability of Leak Fault for Simulated Data with Leak Fault*



## No Leak, Then Leak

Finally, we apply the PROC DYNBNET code for this two-tank example to simulated data that do not have a fault until Time = 60, when a leak fault occurs. The outgoing beliefs for the Fault variable are shown in Figure 5. Shortly after the leak fault starts at Time = 60, PROC DYNBNET detects the presence of the leak as the probability of the leak fault increases from 0 to 1 around Time = 65.

***Figure 5.*** *Probability of Leak Fault for Simulated Data with Leak Fault Starting at Time = 60*

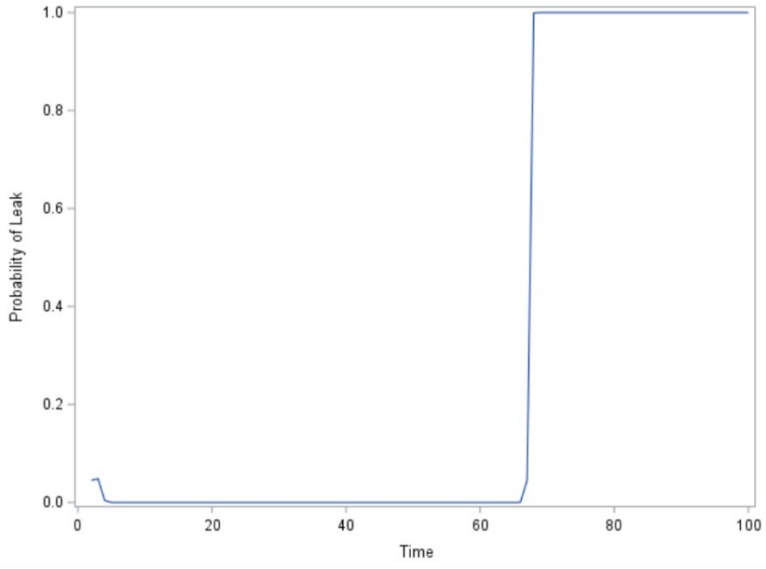
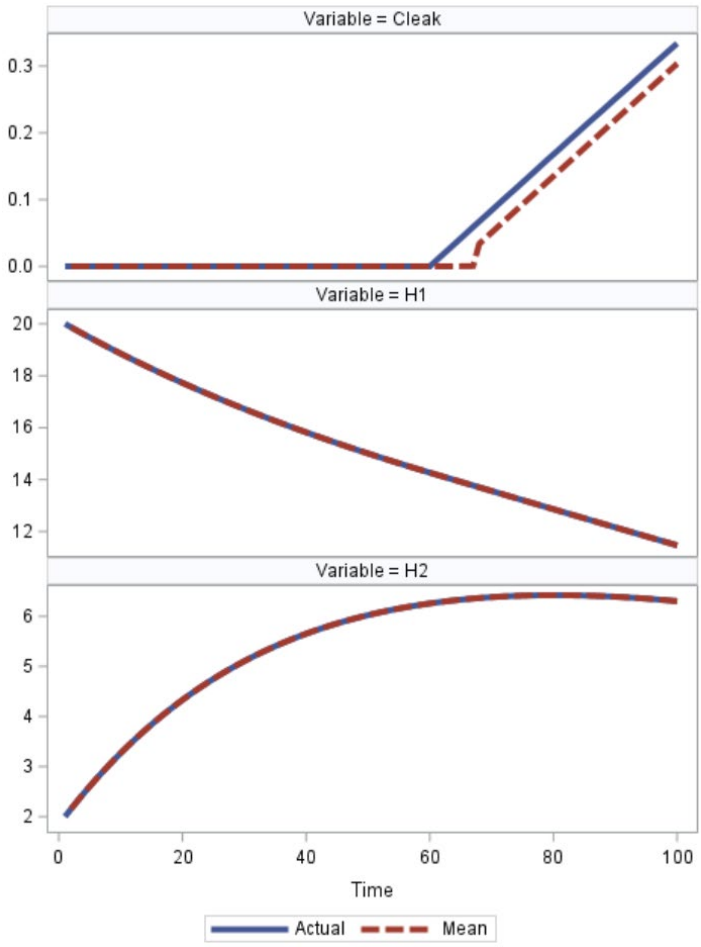



Figure 6 shows the outgoing mean beliefs (in red) for the three hidden interval variables ($C_{\text{leak}}, h_1, h_2$) for the most probable branch of the Fault variable, as well as the "true" values (in blue) based on the simulated data. The results in this figure demonstrate that PROC DYNBNET estimates the means of $h_1$ and $h_2$ with high accuracy. Moreover, the results show how the procedure identifies the presence of the leak fault, which then results in the mean of $C_{\text{leak}}$ slowly increasing.

***Figure 6.*** *Beliefs in Hidden Interval Variables*




## SAS Event Stream Processing

Figure 7 displays a diagram of an example SAS Event Stream Processing model for applying the dynamic Bayesian network plug-in (SAS Institute Inc. 2024d). The first node, Input, includes specification of a source of streaming observed data. The DynamicBNET node includes specification of a Python file that contains the variable relationships that are specified in Python functions rather than PROC FCMP subroutines, along with a saved model state for a dynamic Bayesian network model. The following PROC DYNBNET code demonstrates how to save the model by using the OUTPUTSTATE= option. Then the PROC ASTORE code saves the state to a file on disk.

```
proc dynbnet data = casuser.tankData
           odeapprox = NONE
           outdetails = casuser.tankOutd
        outputstate = casuser.tankState;
     varroles data = casuser.variableRoles;
     varlevels data = casuser.variableLevels;
     links data = casuser.variableLinks;
     initbeliefs data = casuser.initialBeliefs;
     output out = casuser.tankOut;
run;


proc astore;
     download rstore=casuser.tankState store='\Documents\tankState';
run;
```

The DynamicBNET node applies the dynamic Bayesian network plug-in to calculate updated beliefs in the hidden variables, given observed variable values from the streaming data. The Output node collects the results, which can then be displayed using SAS Event Stream Processing Streamviewer (SAS Institute Inc. 2023).

***Figure 7.*** *SAS Event Stream Processing Model Diagram with Dynamic Bayesian Network Plug-In*

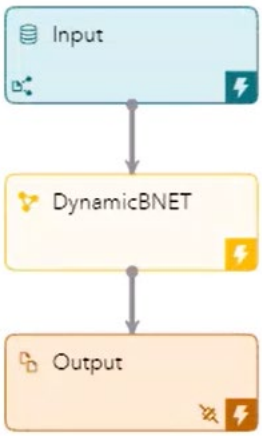
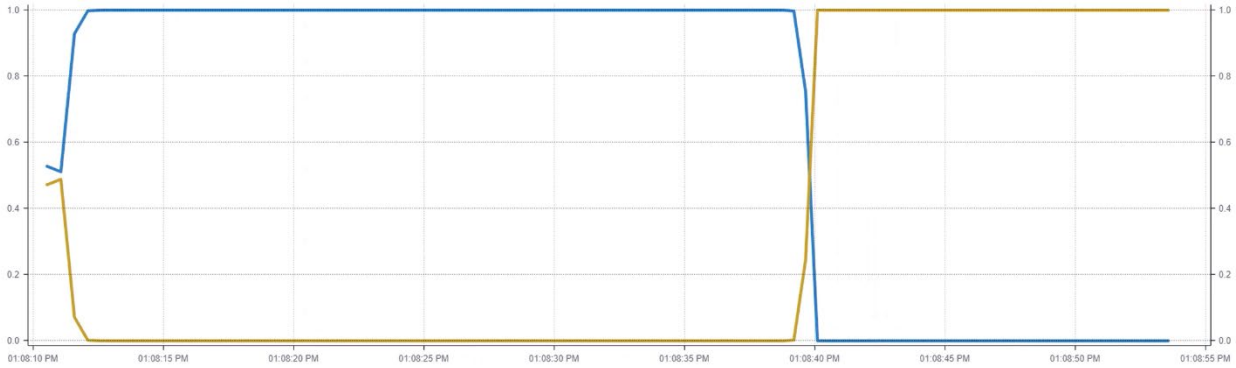



Figure 8 displays the results of applying this SAS Event Stream Processing model to streaming data, where the X axis represents time and the Y axis represents probability. For this particular stream of data, the system started with no leak, and eventually a leak appeared. Therefore, the probability of a leak (shown by the yellow curve) is close to 0 for the first part of the data stream, and then it increases to nearly 1.0 in the second part of the data stream, after the leak appears. The blue curve displays the probability of no leak (or Fault = None). So it is clear that the dynamic Bayesian network plug-in was able to correctly identify the presence of the leak fault in the streaming data.

***Figure 8.*** *Streamviewer Results of Applying Dynamic Bayesian Network Plug-In to Streaming Data*




# Tennessee Eastman Process Example

This example demonstrates the use of a dynamic Bayesian network on Tennessee Eastman process data. The Tennessee Eastman (TE) process is a realistic model of a typical chemical industry process. This process produces

two products from four reactants and has five major unit operators: reactor, product condenser, vapor-liquid separator, recycle compressor, and product stripper. The TE process is widely used to study process control and fault detection. MATLAB simulation code from Ricker (2002) was translated into PROC IML code and was then used to generate the TE process data that are used in the subsequent illustration of PROC DYNBNET. Data were generated for the normal operations of the process (that is, no faults) and for five fault conditions.

Each observation consists of the variable Time, which has the role of time ID and measures time in hours. There are 22 sensor variable measurements, which were taken every second.

Following the same outline as in Overview of Dynamic Bayesian Networks, we start by specifying the variable names and their roles. Time has the role of time ID; Sensor[1]–Sensor[22] are the 22 observed sensor variables; and the rest of the variables, including YY[1]–YY[50] and Fault[1]–Fault[5], are hidden. This model does not include any control variables. The following code specifies the variableRoles table:

```
data casuser.variableRoles;
   length varname $20;
   input varname $ varrole $;
datalines;
Time                TIMEID
YY[1..50]           HIDDEN
Sensor[1..22]       OBSERVED
Fault[1..5]         HIDDEN
;
```

The next step is to specify the levels of the nominal variables. The nominal variables in the model are the five fault variables, which can each take either of two values: Working or Fail. The following code specifies the variableLevels table:

```
data casuser.variableLevels;
   length varname $20;
   length varlevel $20;
   input varname $ varlevel $;
datalines;
Fault[1..5]         Working
Fault[1..5]         Fail
;
```

We are now ready to specify the parent-child relationships in the two-time-period network. As stated in the previous example, Time is a parent of all variables that are defined by an ODE (that is, YY[1]–YY[50]) in the current time period (that is, Stage = 1). Moreover, all variables that are defined by an ODE are also parents of themselves across time periods (that is, Stage = 2). The five fault variables are also parents of YY[1]–YY[50] across time periods. These three previous statements define the first three observations in the variableLinks table. The rest of the table defines parents of the 22 sensor variables. These relationships are evident in the FCMP subroutines for the 22 sensor variables that will be described subsequently. The following code specifies the variableLinks table, which includes all parent-child relationships:

```
data casuser.variableLinks;
   length parent $20 child $20;
   input  parent $ child $ stage;
datalines;
Time                YY[1..50]               1
YY[1..50]           YY[1..50]               2
Fault[1..5]         YY[1..50]               2
YY[41]              Sensor[1]               1
Fault[4]            Sensor[1]               1
YY[39]              Sensor[2]               1
```

```
YY[40]              Sensor[3]              1
YY[42]              Sensor[4]              1
Fault[5]            Sensor[4]              1
YY[10..18]          Sensor[5]              1
YY[28..36]          Sensor[5]              1
YY[43]              Sensor[5]              1
YY[1..9]            Sensor[6]              1
YY[28..36]          Sensor[6]              1
YY[1..9]            Sensor[7]              1
YY[4..9]            Sensor[8]              1
YY[4..9]            Sensor[9]              1
YY[10..18]          Sensor[10]             1
YY[44]              Sensor[10]             1
YY[13..18]          Sensor[11]             1
YY[13..18]          Sensor[12]             1
YY[10..18]          Sensor[13]             1
YY[13..18]          Sensor[14]             1
YY[45]              Sensor[14]             1
YY[19..27]          Sensor[15]             1
YY[28..36]          Sensor[16]             1
YY[19..27]          Sensor[17]             1
YY[46]              Sensor[17]             1
YY[19..27]          Sensor[18]             1
YY[19..27]          Sensor[19]             1
YY[47]              Sensor[19]             1
YY[10..18]          Sensor[20]             1
YY[28..36]          Sensor[20]             1
YY[37]              Sensor[21]             1
YY[38]              Sensor[22]             1
;
```

The next step is to specify initial beliefs of all hidden variables (that is, YY[1]–YY[50] and Fault[1]–Fault[5]). For this example, the initial time is 1/3600 ≈ 0.000278. For the five fault variables, we specify a 0.95 probability that the fault state at the start is Working, and a 0.05 probability that the fault state is Fail. For the 50 interval variables (that is, YY[1]–YY[50]), the initial means were taken from the TE simulation code, and the initial standard deviations were defined as the initial means divided by 1,000. The following code specifies the initialBeliefs table:

```
data initialBeliefs;
   length intervalvar $20 nominalvar $20 varlevel $20;
   input Time intervalvar $ mean nominalvar $ varlevel $;
datalines;
0.000277777777778 Fault[1..5]      0.95             Fault[1..5]      Working
0.000277777777778 Fault[1..5]      0.05             Fault[1..5]      Fail
0.000277777777778 yy[1]           10.40491389       .                .
0.000277777777778 yy[2]            4.363996017      .                .
0.000277777777778 yy[3]            7.570059737      .                .
0.000277777777778 yy[4]            0.4230042431     .                .
0.000277777777778 yy[5]           24.15513437       .                .
0.000277777777778 yy[6]            2.942597645      .                .
0.000277777777778 yy[7]          154.3770655        .                .
0.000277777777778 yy[8]          159.186596         .                .
0.000277777777778 yy[9]            2.808522723      .                .
0.000277777777778 yy[10]          63.75581199       .                .
0.000277777777778 yy[11]          26.74026066       .                .
0.000277777777778 yy[12]          46.38532432       .                .
0.000277777777778 yy[13]           0.2464521543     .                .
0.000277777777778 yy[14]          15.20484404       .                .
0.000277777777778 yy[15]           1.852266172      .                .
0.000277777777778 yy[16]          52.44639459       .                .
0.000277777777778 yy[17]          41.20394008       .                .
```

```
0.000277777777778 yy[18]           0.569931776       .              .
0.000277777777778 yy[19]           0.4306056376      .              .
0.000277777777778 yy[20]           0.0079906200783   .              .
0.000277777777778 yy[21]           0.9056036089      .              .
0.000277777777778 yy[22]           0.016054258216    .              .
0.000277777777778 yy[23]           0.7509759687      .              .
0.000277777777778 yy[24]           0.088582855955    .              .
0.000277777777778 yy[25]          48.27726193        .              .
0.000277777777778 yy[26]          39.38459028        .              .
0.000277777777778 yy[27]           0.3755297257      .              .
0.000277777777778 yy[28]         107.7562698         .              .
0.000277777777778 yy[29]          29.77250546        .              .
0.000277777777778 yy[30]          88.32481135        .              .
0.000277777777778 yy[31]          23.03929507        .              .
0.000277777777778 yy[32]          62.85848794        .              .
0.000277777777778 yy[33]           5.546318688       .              .
0.000277777777778 yy[34]          11.92244772        .              .
0.000277777777778 yy[35]           5.555448243       .              .
0.000277777777778 yy[36]           0.9218489762      .              .
0.000277777777778 yy[37]          94.59927549        .              .
0.000277777777778 yy[38]          77.29698353        .              .
0.000277777777778 yy[39]          63.05263039        .              .
0.000277777777778 yy[40]          53.97970677        .              .
0.000277777777778 yy[41]          24.64355755        .              .
0.000277777777778 yy[42]          61.30192144        .              .
0.000277777777778 yy[43]          22.21              .              .
0.000277777777778 yy[44]          40.06374673        .              .
0.000277777777778 yy[45]          38.1003437         .              .
0.000277777777778 yy[46]          46.53415582        .              .
0.000277777777778 yy[47]          47.44573456        .              .
0.000277777777778 yy[48]          41.10581288        .              .
0.000277777777778 yy[49]          18.11349055        .              .
0.000277777777778 yy[50]          50.                .              .
;
data casuser.initialBeliefs ;
    set initialBeliefs;
    std = mean / 1000.0;
run;
```

Next is the crucial part of defining the models (that is, ODEs and functional relationships) in the two-time-period Bayesian network by using PROC FCMP. The code for the FCMP subroutines in this example is rather long, so it is included on GitHub instead of here. The FCMP subroutines contain the following three parts:

1. *Differential equations for YY[1]–YY[50]*. The differential equations for YY[1]–YY[50] are defined for all the YY variables as a group. Time, current YY values, and fault settings are needed in order to compute the derivatives for the YY variables.

2. *Functional relationships for the 22 sensor variables*. The functional relationships for the 22 sensor variables are defined in one function; "index" is used as a parameter to this function to identify which sensor value is of interest. Current YY values and fault settings are also needed in order to compute the mean of the sensor variable of interest. As an example, the following code snippet illustrates the computation of the mean and standard deviation for Sensor[1]:

```
if index = 1 then do;
  idv[6] = 0.5;
  if (Fault[4] = 'Working') then
    idv[6] = 0;
  vpos[3] = YY[41];
```

```
        ftm3 = vpos[3] * (1 - idv[6]) * vrng[3] / 100;
        mean = ftm3 * .359 / 35.3145;
        sdev = xns[1];
      end;
```

Notice that the mean calculation for Sensor[1] relies on Fault[4] and YY[41]. Recall that in the variableLinks table, Fault[4] and YY[41] were identified as parents of Sensor[1].

3. *Helper subroutines that are used in the previous two subroutines*. The subroutines that are identified in parts (1) and (2) rely on several helper subroutines, such as "getxmws1" and "tesub8_". These subroutines are also defined using PROC FCMP.

Finally, we specify the observed data for the 22 observed sensor variables. The observed data and code for the various scenarios (normal and fault scenarios) are available on GitHub.

Now we have all the necessary information to call PROC DYNBNET as follows:

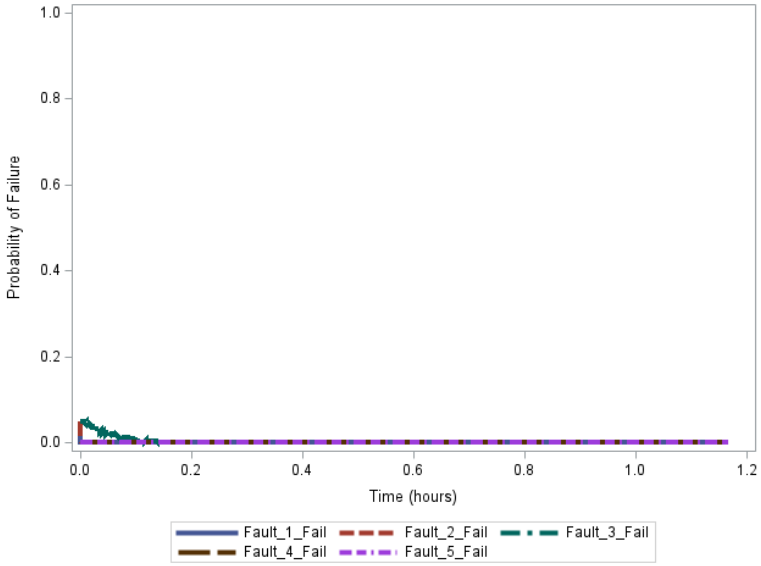
```
proc dynbnet data = casuser.chemData
           odeapprox = ALL;
     varroles data = casuser.variableRoles;
     varlevels data = casuser.variableLevels;
     links data = casuser.variableLinks;
     initbeliefs data = casuser.initialBeliefs;
     output out = casuser.chemOut;
run;
```

For this PROC DYNBNET call, the ODEAPPROX= option is set to ALL so that approximations are used to solve the ODEs that the mean and gradient calculations use.

## Normal

We start by applying the preceding PROC DYNBNET code to simulated data without fault (that is, normal). The outgoing beliefs for each of the five fault variables are shown in Figure 9. Although there is a slight chance of a fault for the first few time periods (probably because the initial belief for each fault variable is a 0.05 chance of fault), the probability for each of the fault variables quickly drops to about 0.0001, and this is maintained for the rest of the provided time periods.

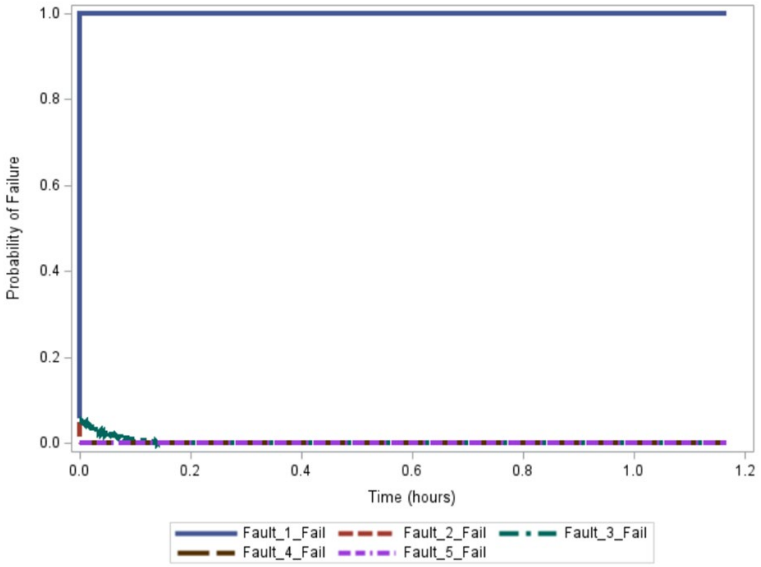
***Figure 9.*** *Probability of Fault for Simulated Data without Fault*




## Fault 1

Next, we apply the preceding PROC DYNBNET code to simulated data that contain Fault 1. The outgoing beliefs for each of the five fault variables are shown in Figure 10. Almost immediately, PROC DYNBNET identifies the presence of Fault 1 as opposed to the other four faults.

***Figure 10.*** *Probability of Fault for Simulated Data Containing Fault 1*




## Faults 2–5

Next, we apply the preceding PROC DYNBNET code to simulated data that contain each of the remaining four faults (Faults 2–5). Figure 11 shows the results for simulated data that contain Fault 2 (left plot) and Fault 3 (right plot). For Fault 2, PROC DYNBNET quickly identifies the presence of Fault 2, as opposed to the other four faults. For Fault 3, PROC DYNBNET takes a little longer to identify its presence, although it still does so within a quarter hour, and it correctly identifies Fault 3, as opposed to the other faults.

***Figure 11.*** *Probability of Fault for Simulated Data Containing Fault 2 (left) and Fault 3 (right)*

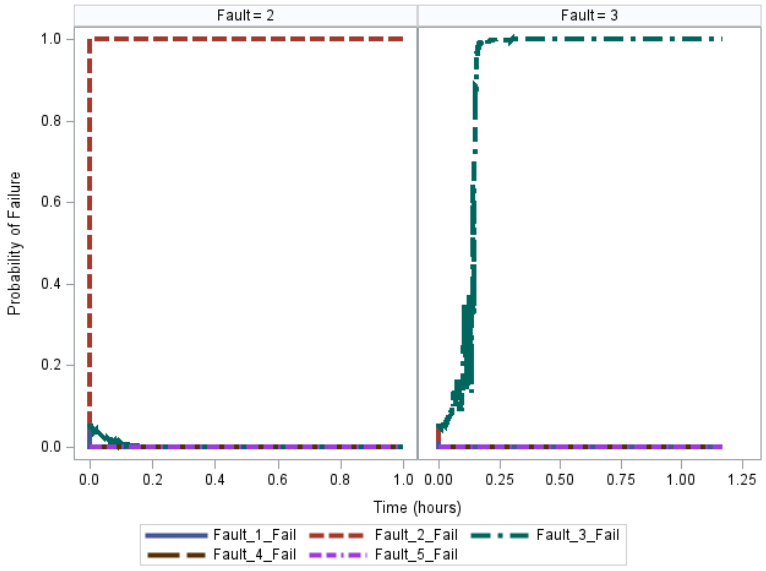
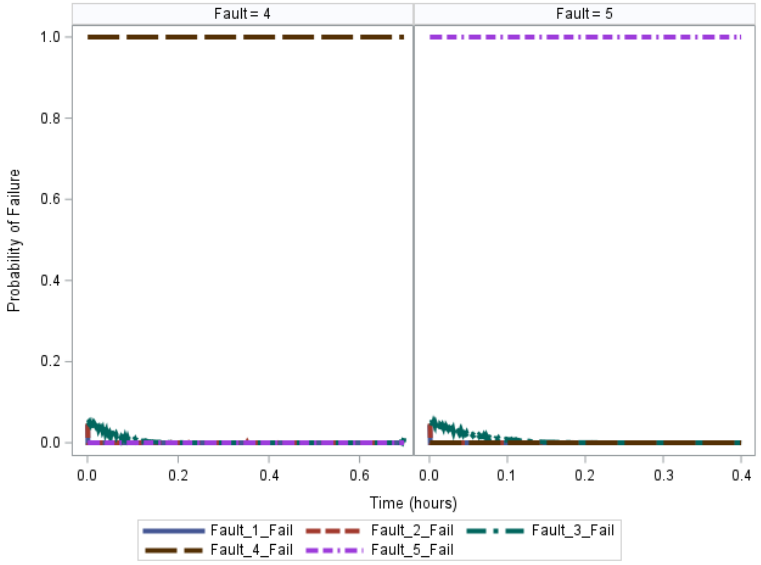



Figure 12 contains the results for simulated data that contain Fault 4 (left plot) and Fault 5 (right plot). As with the Fault 2 results in Figure 11, PROC DYNBNET identifies the presence of the correct fault immediately.

***Figure 12.*** *Probability of Fault for Simulated Data Containing Fault 4 (left) and Fault 5 (right)*




# Conclusion

In this paper, we have demonstrated how you can use PROC DYNBNET to identify faults in IIoT scenarios where there is a known model of the relationships among hidden and observed variables in an industrial system. The most crucial part of using PROC DYNBNET is specifying the known model by using PROC FCMP subroutines. For the two-tank example, we showed how PROC DYNBNET was able to detect the presence of a leak fault within just a few time periods of the introduction of the leak. We also illustrated the use of the new SAS Event Stream

Processing dynamic Bayesian network plug-in to update beliefs in hidden variables from streaming data after the state of a predefined model was saved in PROC DYNBNET. Finally, we used the Tennessee Eastman process example to demonstrate how PROC DYNBNET can also *identify* faults, meaning that it can detect a fault *and* categorize data that accompany the fault into one of several fault types.

# References


Lerner, U., Parr, R., Koller, D., & Biswas, G. (2000). "Bayesian Fault Detection and Diagnosis in Dynamic Systems." In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 531–537. New York: AAAI Press.

Malik, P. K., Sharma, R., Singh, R., Gehlot, A., Satapathy, S. C., Alnumay, W. S., Pelusi, D., Ghosh, U., & Nayak, J. (2021). "Industrial Internet of Things and Its Applications in Industry 4.0: State of the Art." *Computer Communications* 166 (2021): 125–139. DOI: 10.1016/j.comcom.2020.11.016.

Ricker, N. L. (2002). *Tennessee Eastman Challenge Archive, MATLAB 7.x Code.* Retrieved from University of Washington, Seattle, Department of Chemical Engineering. Available at http://depts.washington.edu/control/LARRY/TE/download.html.

SAS Institute Inc. (2017). *Base SAS 9.4 Procedures Guide.* 7th ed. Available at https://go.documentation.sas.com/api/collections/pgmsascdc/9.4_3.5/docsets/proc/content/proc.pdf?locale=en#nameddest=n0pio2crltpr35n1ny010zrfbvc9.

SAS Institute Inc. (2018). *SAS/IML 15.1 User's Guide*. Available at https://go.documentation.sas.com/api/collections/pgmsascdc/9.4_3.4/docsets/imlug/content/imlug.pdf?locale=en.

SAS Institute Inc. (2023). *Working with Charts and Tables*. Available at https://go.documentation.sas.com/api/collections/espcdc/v_048/docsets/espvisualize/content/espvisualize.pdf?locale=en#nameddest=n1afzvhtws0s23n19wy0elwhergw.

SAS Institute Inc. (2024a). *DYNBNET Procedure.* Available at https://go.documentation.sas.com/api/collections/pgmsascdc/v_048/docsets/casml/content/casml.pdf.

SAS Institute Inc. (2024b). *Fault Identification Using Dynamic Bayesian Networks*. Available at https://github.com/sassoftware/iotaa/tree/main/Fault%20Identification%20Using%20Dynamic%20Bayesian%20Networks.

SAS Institute Inc. (2024c). *SAS Event Stream Processing: Overview*. Available at https://go.documentation.sas.com/api/collections/espcdc/v_048/docsets/espov/content/espov.pdf?locale=en#nameddest=home.

SAS Institute Inc. (2024d). *Using Dynamic Bayesian Networks*. Available at https://go.documentation.sas.com/api/collections/espcdc/v_048/docsets/espan/content/espan.pdf?locale=en#nameddest=n1a24zmowg07opn1ul03ulh6g23c.

**Release Information** Content Version: 1.0 July 2024.

**Trademarks and Patents** SAS Institute Inc. SAS Campus Drive, Cary, North Carolina 27513

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. R indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

To contact your local SAS office, please visit: sas.com/offices

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © SAS Institute Inc. All rights reserved.