Using Bayesian Networks to Describe Hydrologic Processes

Espen Willads Torgersen

Supervisors:
Ole-Christoffer Granmo
Bernt Viggo Matheussen

This master's thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

University of Agder
Faculty of Engineering and Science
Department of ICT

2 June 2014

Preface

This master thesis is submitted in partial fulfillment of the requirements for the degree Master of Science in Information and Communication Technology at the University of Agder, Faculty of Engineering and Science. I would like to thank my supervisors Professor Ole-Christoffer Granmo of the University of Agder and Bernt Viggo Matheussen, Head of Development at Agder Energi. Their input, advice and guidance has been invaluable for getting this thesis completed.

Espen Willads Torgersen

Contents

1 Introduction

Humans have always striven to map and make sense of the world around them, trying to figure out how things work and relationships between happenings. To help with this we create models of the systems or part of the systems we are examining.

Hydrologic models are simplified representations of complex physical systems, and carry with them some amount of uncertainty. Uncertainties come from model structure, both because of the simplifications that they necessarily implement and that the processes might not be fully understood. There are also uncertainties tied to the inputs used with the model, how accurate are the measurements or weather forecasts? And there are uncertainties with the parameters that represent aspects that vary between the different areas the model is used on, some of which may be difficult or impossible to measure and must be inferred from observations.

1.1 Background

As I already mentioned, hydrologic models carry with them some amount of uncertainty connected to the models structure, input and parameters. Quantifying these uncertainties is important both in research settings and real world applications such as flood forecasting or planning of hydroelectric power production. But as standard practice today, if uncertainties are considered at all, current methods in hydrologic forecasting typically focus on one or two error sources [1] and so an approach that considers a wider perspective is needed.

Model structure uncertainties come from inaccurate representation of the physical processes involved. These inaccuracies can be caused by the simplifications that are necessary to implement to make representation of complex physical processes computationally feasible, but also an incomplete knowledge about the physics that govern the processes [2]. Parameter calibration is usually done through some form of trial and error, adjusting parameters until model output is acceptably close to observed data. Closeness of fit is decided by an objective function, and which one you choose can influence the results. Different objective functions can emphasize different aspects such as high values vs low values, or other systematic/behavioral errors [3].

Bayesian networks have been used for a long time and for a wide variety of purposes, including medical diagnosis and language understanding. While probabilistic methods have been used by the hydrologic community for years, the use of Bayesian networks are largely unexplored in this setting

1.2 Thesis definition

The goal for this Masters thesis is to explore the use of dynamic Bayesian networks for describing hydrologic processes. The main intent is to try and provide better descriptions of the uncertainties that are tied to dealing with such complex and partially unknown processes, while also trying to reduce these uncertainties. For this purpose I have translated part of a well known and widely used deterministic model, the snow module of the HBV model, into a dynamic Bayesian network:

1.3 Related work

Bayesian networks has been used for a long time and for a wide variety of purposes, including medical diagnosis [4] and language understanding [5]. The term was introduced by Judea Pearl in 1985 [6], and is summarized in his 1988 book [7].

Although Bayesian networks have not seen widespread use for hydrologic modeling, several articles describe probabilistic approaches to uncertainty estimation using deterministic models. Vrugt et al presents a Markov chain Monte Carlo sampler designed to estimate the posterior probability density function of model parameters. Bayesian model averaging is a common method used for error estimation, and several articles describe how this can be approached [8][9]. However these methods typically focus on one or two sources of error, neglecting the others.

1.4 Report outline

The next chapter will provide the theory that is the foundation for this report. I will describe Bayesian probability and Gibbs sampling, and provide a brief overview of the HBV model. The chapter also includes in introduction to the tools I have used.

In chapter 3 I present the model I propose for representing the HBV models snow module in a DBN. I explain the implementation and the testing to validate it. Chapter 4 describes how I calibrated the model using JAGS, the results of the process is presented and discussed, then a conclusion follows in chapter 5.

2 Theoretical Background

2.1 Bayesian probability

Before I get on to Bayesian networks, I will first give a short introduction to Bayesian probability. Bayesian probability, named after 18th century English statistician Thomas Bayes, provides a framework for probabilistic inference and reasoning. It is an interpretation of the concept of probability where the probabilities are representations of a state of belief that have values even before hypothesis testing, as opposed to the frequentist view. Equation (2.1) shows the well known Bayes theorem, which is derived from the basic axioms of probability.

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

It is stating the mathematical relationship between two probabilities and their conditional probabilities. In the context of Bayesian inference it describes how we can update the probability of a hypothesis based on observations or evidence as seen in equation 2.2

$$P(h|o)=\frac{P(o|h)P(h)}{P(o)} \tag{2.2}$$

- P(h|o) is the posterior probability, the probability of the hypothesis conditional on the observations.
- P(o|h) is the likelihood, the probability of the observations given the hypothesis.
- P(h) is the prior probability, the probability of the hypothesis before the observations are considered.
- P(o) is the prior probability of the observations.

Summing this, equation 2.2 states that by accounting for observations, the prior probability of the hypothesis changes to the posterior probability.

Bayesian networks

To introduce Bayesian networks I will quote Charniak [10]:

> *“The best way to understand Bayesian networks is to imagine trying to model a situation in which causality plays a role but where our understanding of what is actually going on is incomplete, so we need to describe things probabilistically”*.

Bayesian networks, sometimes also called belief nets or causal networks, are graphical models describing the probabilistic relationship between variables as a directed acyclic graph. The nodes represent objects or events and are usually called variables or states, and edges describe causal relationships with direction from the cause variable to the effect variable.

To complete a model we also need to specify a Conditional Probability Distribution (CPD) for each node. For discrete nodes this takes the form of a Conditional Probability Table (CPT), where the probability for the node is given for every combination of values of its parents.

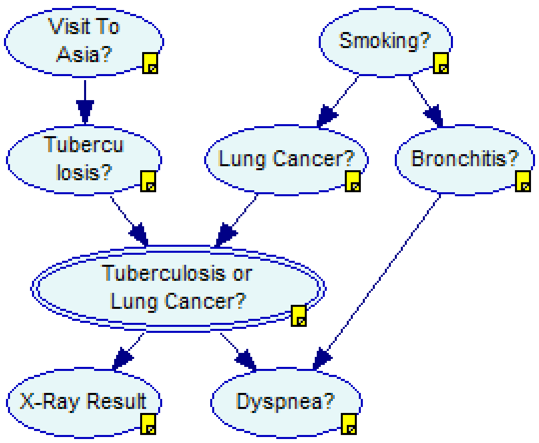


Figure 2.1: Example of Bayesian network for medical diagnosis.

Table 2.1: CPT for node "Dyspnea?"

Tuberculosis or Lung Cancer?	Nothing		CancerORTuberculosis	
Bronchitis?	Absent	Present	Absent	Present
Absent	0.9	0.2	0.3	0.1
Present	0.1	0.8	0.7	0.9

An example from medical diagnosis [11] can be seen in Figure 2.1. Table 2.1 shows the CPT for node "Dyspnea?". The node has two possible values, "Absent" and "Present". Both of its two parents also have two possible values, "Bronchitis" has "Absent" or "Present", and "Tuberculosis or Lung Cancer?" has "Nothing" or "CancerORTuberculosis". This gives a table with 2*2*2=8 entries as stated earlier, one for every outcome given every combination of values from the parents.

A potential problem with CPTs can be their size. Since the size of the CPT for a node depends on it own number of states as well as the number of states of all its parents, it can quickly become too large to handle, and so care must be taken to keep the number of nodes that each node depends on at a minimum.

Despite the name, Bayesian networks don't necessarily imply a commitment to Bayesian probability, frequentist methods are common to estimate the CPDs. The name instead comes from their use of Bayes' rule for statistical inference. We introduce evidence into the network, setting some variables into a fixed state and then compute the probabilities of interest conditioned on the evidence, and here Bayes theorem is central as shown earlier.

Dynamic Bayesian networks

Another name that can be a bit confusing is that of the Dynamic Bayesian Network (DBN). It does not mean that the network or the parameters changes dynamically, but that it is a dynamic system that is modeled. Or in other words, it is a Bayesian network with an added time dimension, and consists of time-steps that each has its own variables. It can be visualized as a normal Bayesian network copied one or several times and put next to each other, but with one or more variables being dependent on a variable from the previous time step. An illustration of this can be seen in Figure 2.2.

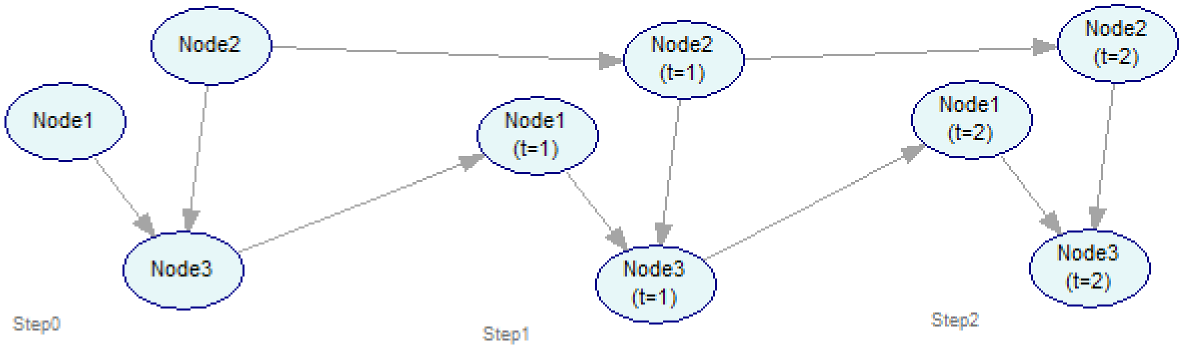


Figure 2.2: Sample DBN

There are different way to perform inference in DBNs. Three main types are:

- Filtering: The current belief state is computed based on all evidence from the past.
- Prediction: A future belief state is computed based on all evidence from the past.
- Smoothing: A past belief state is computed based on all evidence up to the present.

Different types of DBNs require different types of calculations/estimations. There are several different inference algorithms, some exact and some use approximation.

2.2 Gibbs sampling

Gibbs sampling is a Markov chain Monte Carlo (MCMC) method that has a wide range of applications within Bayesian statistics. Monte Carlo is a name for methods that use random sampling to get numerical results, e.g. an unknown probability distribution. They were first developed to help calculate complex integrals. A Markov chain is a system that transitions stochastically from one state to another and is memoryless; the transition probabilities are only dependent on the current state. MCMC methods are based on sampling from a probability distribution by constructing a Markov chain that has the desired distribution as its stationary distribution.

The Gibbs algorithm was first used in the context of image processing [12] and has been shown to be well suited for inference in probabilistic models [13]. The way it works is that each random variable is iteratively resampled from its conditional distribution given the remaining variables which are assigned fixed values. Lets consider a bivariate random variable (x, y) with conditional probabilities p(x|y) and p(y|x). The sampler starts with an initial value x_0 for x and generating a random variable y_0 from p(y|x_0). Then at each step new values for x_i are generated from p(x|y_{i-1}) and for y_i from p(y|x_i). This principle is the same for multivariates.

The idea behind the Gibbs sampler is that (a subset of) the samples approximate the joint distribution of all the variables. The marginal distribution for a subset or a single variable can also be approximated by examining the subset of samples for that subset or single variable. It is common to have a burn-in period, meaning that a number of samples at the beginning are discarded to be sure that the initial values doesn't have any effect. *Thinning* is also sometimes practiced, since there are some amount of correlation between samples only every *n*th sample is used.

There are several different implementations of Gibbs sampling, among them BUGS (Bayesian inference Using Gibbs Sampling), and JAGS (Just Another Gibbs Sampler) which I will look further into later in this chapter.

2.3 The HBV model

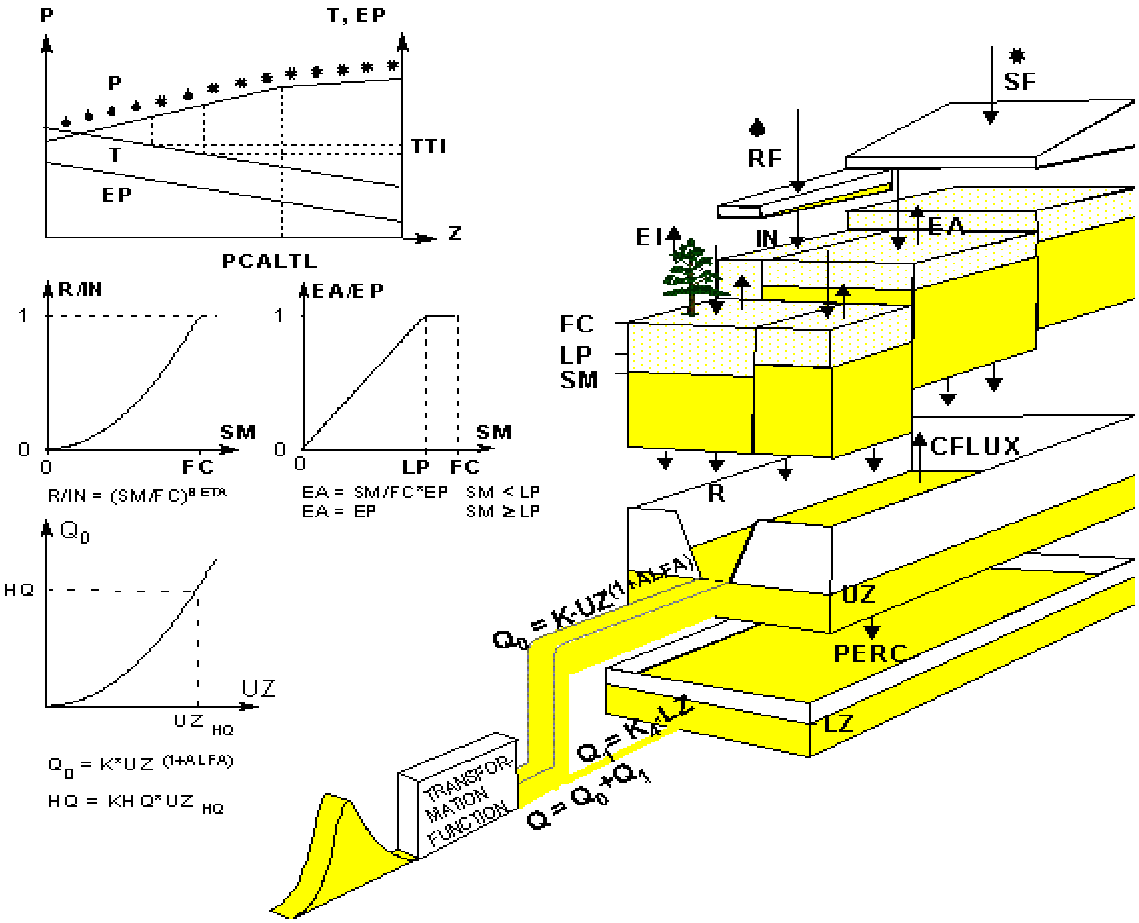
The HBV model [14] is a mathematical rainfall-runoff model developed at the Swedish Meteorological and Hydrological Institute, by Dr. Sten Bergstrøm at Hydrologiska Byråns avdeling for Vattenbalans (HBV). It was developed in the 1970s, and has since then been revised several times. Some of the main features and characteristics of the HBV-model are as follows:

- The model uses data for precipitation, air temperature and potential evapotranspiration to simulate the runoff process.
- It is a conceptual model which consists of numerical description of hydrologic processes in the modeled catchment area, meaning it is based on an understanding of the physical structure and processes.
- It is a deterministic model, meaning that if two identical set of input are run through the model with identical starting conditions and parameter setups, the outputs will always be identical.
- The model has a fixed structure, but is equipped with a set of parameters that are used to tune the model to a specific catchment area. This means that the model has to be calibrated before it can be of practical use.

The parameters of the HBV model can be grouped into two categories, free and confined parameters. Confined parameters are the parameters that can be determined by using maps, field surveys or other sources. Free parameters on the other hand are the the parameters that can't be decided directly. They need to be set through the process called model calibration. In this context it calibration means *to determine the set of free parameters in the model that gives the best possible correspondence between observed and simulated runoff for a catchment* [15].

Model structure

The model consist of several modules or subroutines: Meteorological interpolation, snow accumulation and melt, evapotranspiration estimation, soil moisture accounting, runoff generation, and routing between subbasins and in lakes. Figure 2.3 [16] displays the schematic structure of the HBV-96 [17] model with only the most important characteristics showing.



P = Precipitation
T = Temperature
SF = Snow
RF = Rain
Z = Elevation
PCALTL = Threshold for altitude correction
TTI = Threshold temperature interval
IN = Infiltration
EP = Potential evapotranspiration
EA = Actual evapotranspiration
EI = Evaporation from interception
SM = Soil moisture storage
FC = Maximum soil moisture storage
LP = Limit for potential evapotranspiration

BETA = Soil parameter
R = Recharge
CFLUX = Capillary transport
UZ = Storage in upper response box
LZ = Storage in lower response box
PERC = Percolation
K, K_4 = Recession parameters
ALFA = Recession parameter
Q_0, Q_1 = Runoff components
HQ = High flowparameter
KHQ = Recession at HQ
HQ_{UZ} = UZ level at HQ

Figure 2.3: Structure of HBV-96

Snow routine

The structure of the snow routine can be seen in Figure 2.4. Air temperature is computed based on observed temperature and adjusted according to the air temperature lapse rate. Likewise, amount of precipitation is based on observed precipitation and the precipitation lapse rate. Based on air temperature, precipitation type (snow or rain) and snowmelt or refreezing is computed. These use threshold values specified within the free parameters. Snow melt is based on a degree-day approach (discussed further below) using air temperature, and the snow pack has a water holding capacity which delays runoff. The most important results from the routine are the following variables:

- Snow storage in mm of water equivalent
- Free water contents in the snow in mm
- Snow melt/refreeze in mm/timestep

If the water content in the snow exceeds a threshold value given as a fraction of the dry snow, excess water is released from the snow pack.

The HBV model uses a degree-day approach for snow melt. Formally the degree day factor indicates the amount of dry snow converted to liquid water when the temperature is one degree above freezing for one time step. This approach is widely used since air temperatures are generally easily available, whereas physically based models have much higher demands when it comes to data. Air temperature has a high correlation to radiation, wind and humidity, which are key factors involved in heat transfer to the snow. The degree day approach has been shown to be just as good as radiation based approaches under certain conditions [18].

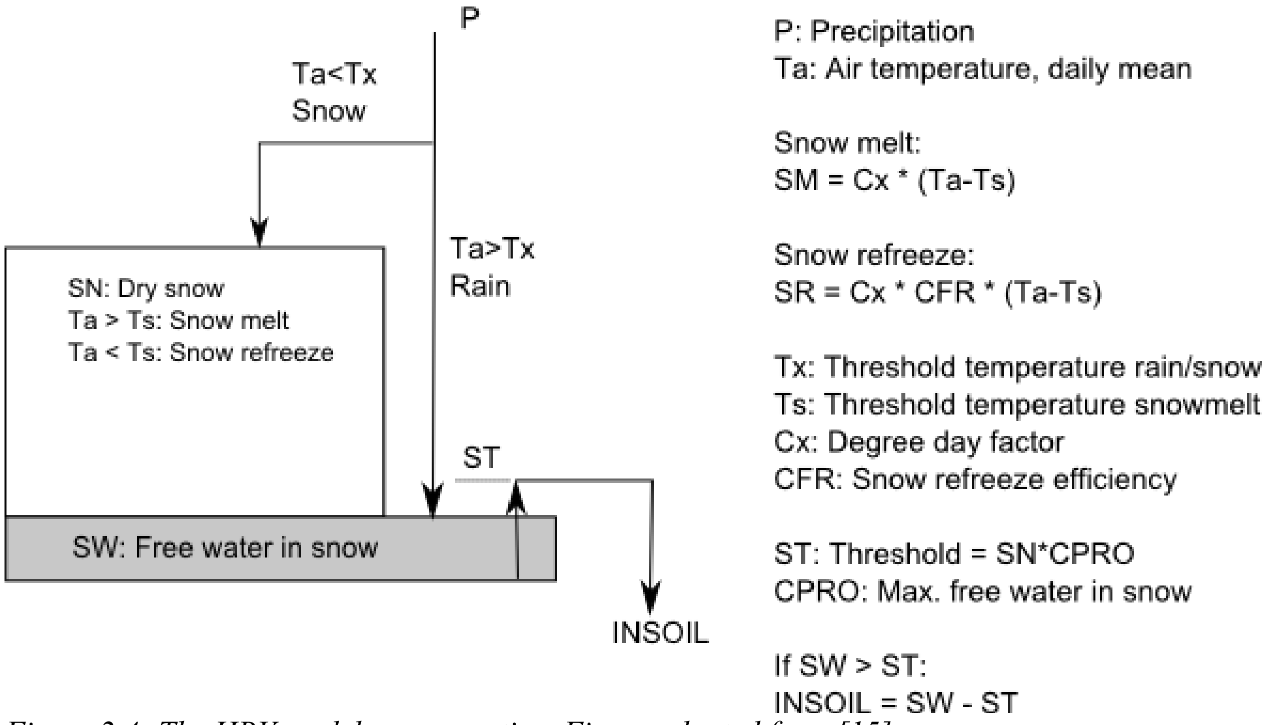


Figure 2.4: The HBV models snow routine. Figure adapted from [15]

2.4 Test data

The Natural Resources Conservation Service of the United States Department of Agriculture operates an extensive network of automated sites that collect snow pack and related climate data called

SNOTEL (short for Snow Telemetry). The programs main goal is to conduct snow surveys and develop accurate and reliable water supply forecasts, this is critical since the majority of the water supply in the Western United States come from melted snow. The network now consists of more than 850 stations in the Western United States and Alaska, and each site has at a minimum sensors that measure air temperature, precipitation, snow water content and snow depth. All data are made public almost in real time through an Internet delivery system.

2.5 Tools

SMILE/GeNIe

SMILE (Structural Modeling, Inference, and Learning Engine) is a powerful and platform independent library of C++ classes that implements graphical probabilistic decision-theoretic methods, such as Bayesian networks and influence diagrams, that is well suited to be included in intelligent systems . It is developed at the Decision Systems Laboratory, University of Pittsburgh, and enables the user to create and edit, save and load models and use them for probabilistic reasoning and decision making under uncertainty.

GeNIe (Graphical Network Interface) is a Windows graphical user interface to SMILE. It was developed with an emphasis on accessibility and user friendliness using a graphical click-and-drop approach while still providing a robust and versatile development environment.

SMILE and GeNIe have been used have been used for teaching statistics and decision-theoretic methods at several universities, and some applications around the world include battle damage assessment (Rockwell International and U.S. Air Force Rome Laboratory), group decision support models for regional conflict detection (Decision Support Department, U.S. Naval War College), intelligent tutoring systems (Learning and Development Research Center, University of Pittsburgh), medical therapy planning (National university of Singapore), medical diagnosis (Medical Informatics Training Program, University of Pittsburgh; Technical University of Bialystok, Poland) [19].

JAGS

JAGS (Just Another Gibbs Sampler) is as mentioned earlier a program for analysis of Bayesian hierarchical models using MCMC simulation. It is closely related to BUGS (Bayesian inference Using Gibbs Sampling), using a dialect of the language, but is written in C++ to have platform independence, while the original BUGS family was written in Component Pascal which is only available on Windows. It has no graphical user interface (GUI) of its own for model building and sample post-processing so a separate program is needed for this. However JAGS is designed to work closely with the R language and environment for statistical computation and graphics, the package *rjags* provides and interface to JAGS in R, and the package *coda* is useful to analyze the output. I will discuss running a model in JAGS a bit further in section 3.2 where I talk about the implementation of my model.

3 A Bayesian model for snow accumulation and melt

3.1 Model design

To test if modeling hydrologic processes in a DBN is viable I created one based on the HBV models snow module. The design is shown in Figure 3.1 with explanations in Table 3.1. The parameters from the HBV model are transformed into stochastic variables taking on some probability distribution. The rest of the nodes are then just deterministic manipulation of numbers.

I have chosen to work with continuous nodes rather than discrete. Continuous nodes loses some precision due to the reliance on sampling, but discretization also brings some approximation and the finer resolution you want the larger the CPTs will be, quickly growing larger than what is manageable.

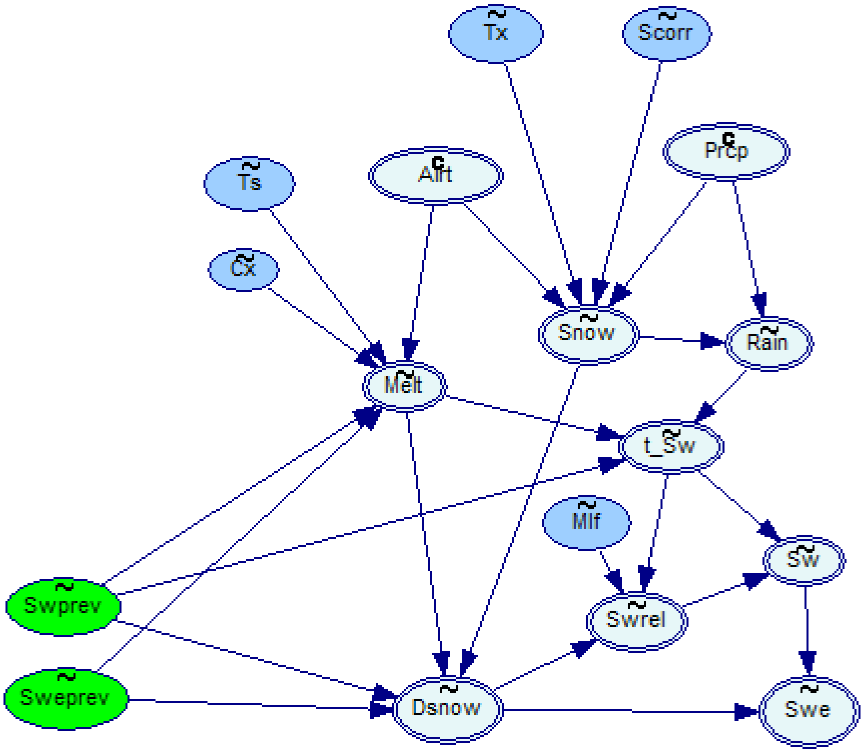


Figure 3.1: The HBV models snow module as a Bayesian network. Green nodes signify variables from the previous time step, darker blues are the parameters.

At each time step two variables from the previous step (or initialization for the first step) are used. Snow water equivalent (SWE) is the estimated snow storage, given in mm of water equivalent, and is the most important and interesting resulting output from the model. Also carried between steps is the amount of liquid water in the snow (SW), and subtracting the liquid water from the SWE total gives the amount of dry snow. Water released from the snow pack occurs if the amount of liquid water exceeds a certain fraction of the dry snow. The model takes measurements for air temperature and precipitation as input. Precipitation is computed as either snow or rain, depending on whether the temperature is above or below the threshold value. Melt has its own threshold, when temperatures are above it snow melt occurs, providing there is snow to melt. Snowfall is added to the amount of dry snow, rain to the free water content, and melt is subtracted from the dry snow and added to SW. I have not implemented snow refreezing in this test model to keep it simpler, and a precipitation correction factor for rain is also omitted since that has very little impact on snow melt and accumulation.

Table 3.1: Explanation of nodes from DBN.

Name	Meaning	Equation
Tx	Threshold temperature rain/snow in °C. Temperatures above gives rain, below snow.	
Ts	Threshold temperature for snowmelt in °C.	
Scorr	Correction factor for snowfall.	
Cx	Degree-day factor.	
Mlf	Max liquid fraction – The maximum of free water contained in the snow given as fraction of dry snow.	
Airt	Measured air temperature – daily mean.	Measured values.
Prcp	Measured precipitation – day.	Measured values.
Melt	Snow melt.	*Min*(*If* (Airt>Ts) *then* Cx*(Airt-Ts) *else* 0 , Swe(t-1) - Sw(t-1))
Snow		*If* (Airt < Tx) *then* Prcp*Scorr *else* 0
Rain		*If* (Snow == 0) *then* Prcp *else* 0
Swe	Snow Water Equivalent – Amount of snow stored given as mm of water equivalent.	Dsnow + Sw

Sw	Free (liquid) water in snow.	t_Sw - Swrel
t_sw	Intermediary calculation for Sw.	Sw(t-1) + Melt + Rain
Dsnow	Dry snow.	Swe(t-1) - Sw(t-1) + Snow - Melt
Swrel	Amount of released water from snow.	*If* (t_Sw > Dsnow * Mlf) *then* t_Sw - (Dsnow * Mlf) *else* 0

3.2 Implementation

I tried several different approaches to implementing the model. Though GeNIe is a powerful tool for working with discrete-valued DBNs it doesn't support the temporal dimension when working with continuous nodes, taking that off the plate. Instead I created the DBN with C++ code using the SMILE library, and using a GeNIe model as template. It is a cumbersome approach since the equations has to be hard-coded into the program, and this makes making even small changes a tedious task. In addition to that, the SMILE classes that work with continuous nodes are not documented at all at the moment, complicating thing further. On the back of these things I did not use the SMILE version for further testing, but the code is included in Appendix C.

Instead I created the model using JAGS. JAGS is as discussed earlier designed for inference on Bayesian models using MCMC simulation. Running a model in JAGS refers to generating samples from the posterior distribution of the model parameters. It is a five step process:

- Defining the model: This is done in a text file, using a dialect of the BUGS language. I have included the model definition in Appendix B. This part also includes definition of the data and that is done in a separate text file.
- Compilation: A graph representing the model is created in computer memory.
- Initialization: Setting initial values for the models parameters, choosing a Random Number Generator (RNG) and choosing a Sampler. All of these can be done automatically.
- Adaptation and burn-in: As mentioned earlier a burn-in period is often used. In addition the model may run in adaptive mode for a period of the burn-in, allowing the Samplers to modify their behavior to increase efficiency.
- Monitoring: A monitor in JAGS is an object that records sampled values, by default only the current value of each node is kept in the model, so a monitor is needed for variables of interest.

I also recreated the model with Python code, to be used for reference in the model verification. This code is included in Appendix A.

3.3 Verification

To create confidence that the model is sound and that the implementations behave as expected I tested it on real data from the Mount Hood SNOTEL site. Mount Hood lies in Northern Oregon, and the station is located at an elevation of 5370 feet. From the NRCS website I got data for daily mean air temperature, daily accumulated precipitation and daily readings of snow water equivalent for several years.

Figure 3.2: Mount Hood SNOTEL site, photo from http://www.wcc.nrcs.usda.gov/

Using both the JAGS implementation and the Python implementation for reference, I set the parameters to be drawn from uniform distributions with pretty wide but plausible limits and used the same distributions on both implementations. Then I drew 5000 samples for each of them and ran the models on a time series with two years worth of temperature and precipitation data. The expectation was that the results should be the same, and that a plot of the output variable (SWE) should have some resemblance to a plot of the observations.

Table 3.2: Parameter ranges used for model verification.

Parameter	**Range**
Tx	-1.0 – 2.0
Ts	-1.0 – 2.0
Scorr	1.0 – 1.5
Cx	3.0 – 6.0
Mlf	0.1 – 0.6

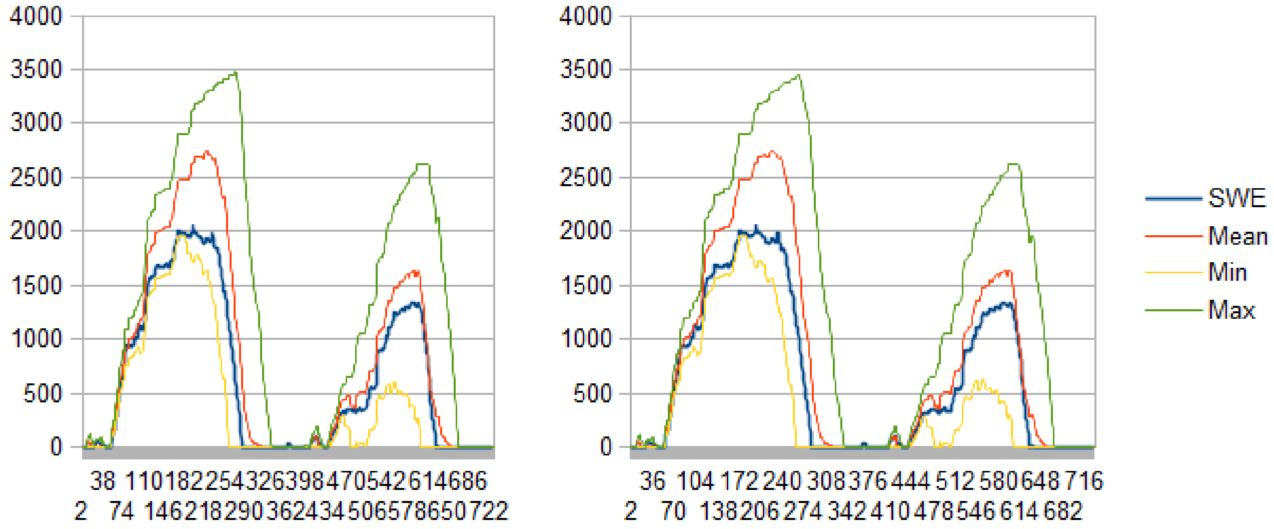


Figure 3.3: Model verification results: Plots of SWE mean, minimum and maximum from model output together with the actual observations.

The results of this test can be seen in Figure 3.3, with the Python implementation to the left and the JAGS implementation to the right. The blue lines are the observations, while the red yellow and green are mean, minimum and maximum from the model output. As expected both are nearly identical, and the model outputs also follow the general shape of the observations. That the spread is so big is not surprising, since the ranges for the parameters were set pretty wide. These results helps create confidence that the implementations work as intended, and that the model is sound.

4 Calibration using JAGS

To demonstrate calibration of the model I have chosen to use JAGS. This was done by adding the observations into to model implementation and saying that it should have a normal distribution with the calculated SWE value as mean and some precision. Experimenting with this precision showed that it had to be set very small (meaning standard error is big), else the parameters would not converge. They can't explore the parameter space properly, and different chains will get stuck with varying very narrow distributions. As convergence diagnostics is a large field of its own, I will not go further into it here. I had to find a value so that the parameter samples converged properly, while the actual observations are still covered by the model SWE distributions. For the test I present next I used the value 10e^{-6}.

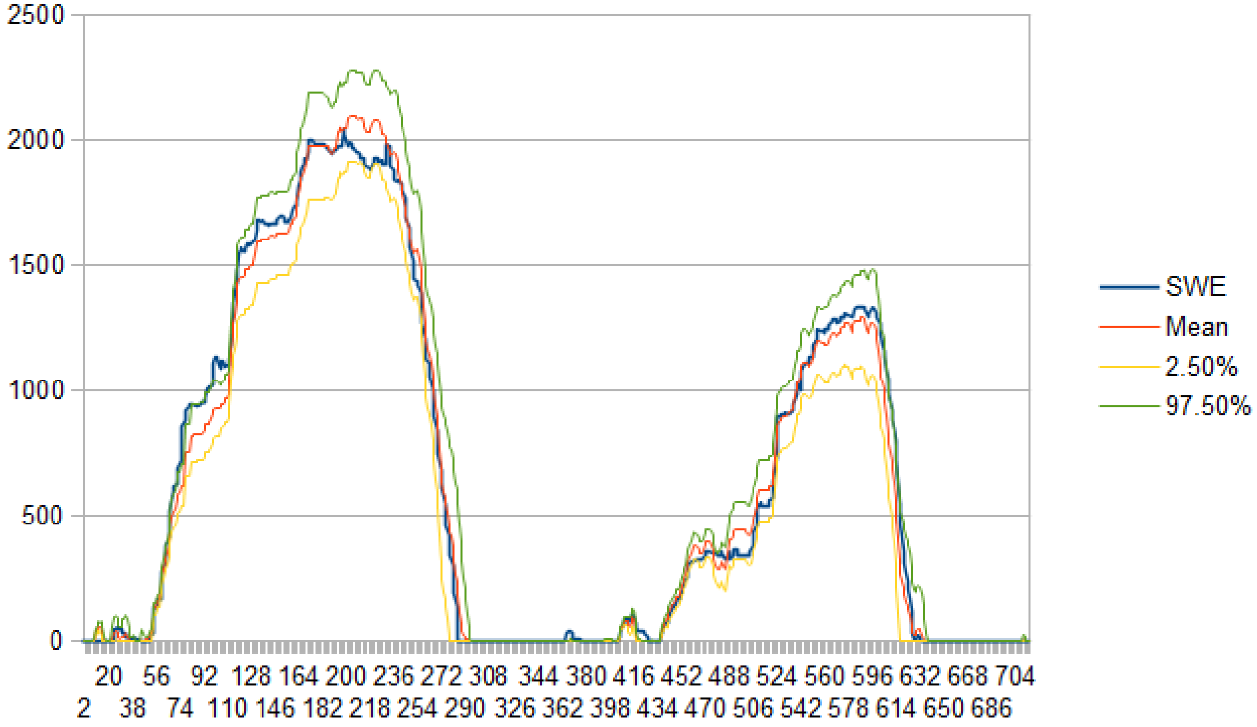


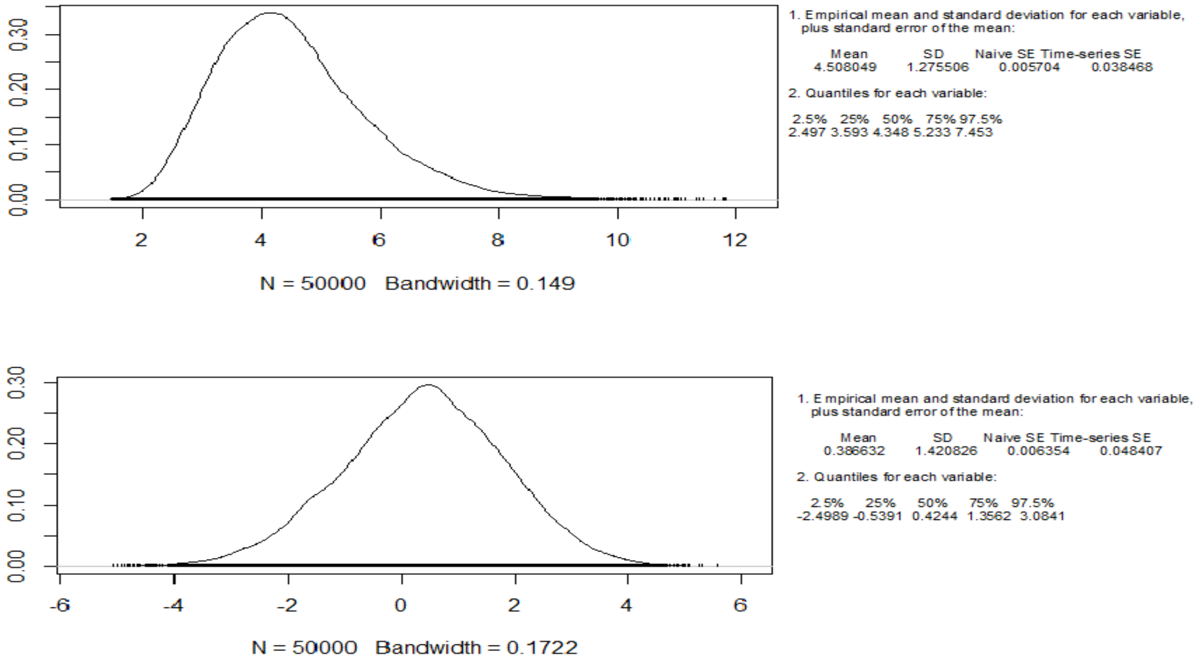
Figure 4.1: Plot of SWE samples from model: mean (red), and 0.025 and 0.975 percentiles (yellow and green). Observations are also plotted (blue).

Figure 4.1 shows a plot of SWE ranges from a model run using a burn-in of 15000 and then running 50000 samples, observations are also included in the graph. For the SWE plot I have used a 95% interval here, cutting away the highest and lowest samples (0.025 and 0.975 percentiles). Comparing this with Figure 3.3 we can see significant improvement as is expected. The spread is a lot narrower while still covering the observations for the most part. As an estimation of performance we can calculate the accuracy of the model output by counting how often the observations are within the bounds of the chosen interval, in this case it is 664/720=0.922 – an accuracy of 92.2%. Model accuracy for various intervals are listed in Table 4.1.

Table 4.1: Model accuracy using different amounts of the samples, every time cutting an equal percentile from top and bottom.

Samples	**Accuracy**
100.00%	697/720 = 0.968
95.00%	664/720 = 0.922
90.00%	643/720 = 0.893
75.00%	614/720 = 0.853
50.00%	555/720 = 0.771

Figure 4.2 shows what the distributions for the variables Cx and Ts looks like, while Figure 4.3 is plots of the distribution for SWE on some arbitrarily chosen days. As we can see they take on a distribution that looks Gaussian.



```
1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

      Mean          SD     Naive SE Time-series SE
   4.508049    1.275506    0.005704       0.038468

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
2.497 3.593 4.348 5.233 7.453
```

```
1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

      Mean          SD     Naive SE Time-series SE
   0.386632    1.420826    0.006354       0.048407

2. Quantiles for each variable:

   2.5%     25%     50%     75%   97.5%
-2.4989 -0.5391  0.4244  1.3562  3.0841
```

Figure 4.2: Distribution for variables Cx (top) and Ts (bottom)

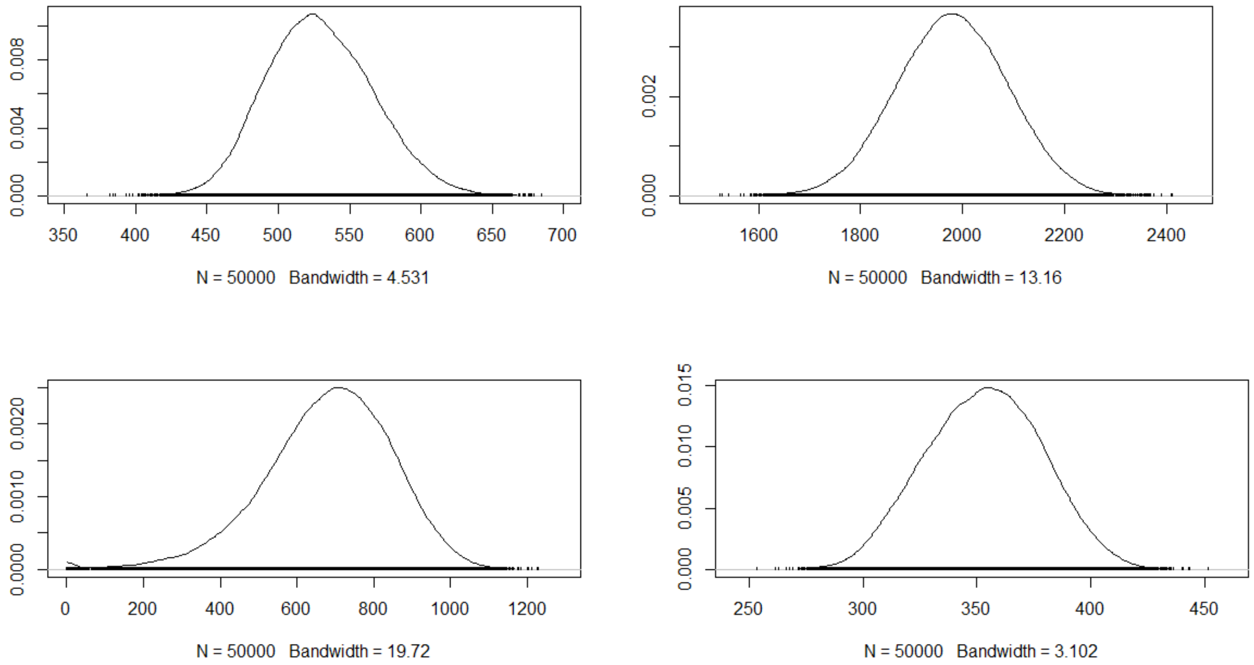


Figure 4.3: SWE distribution for days 70 (top left), 180 (top right), 270 (bottom left) and 460 (bottom right)

I have also tested on data from another site to ensure that the results are replicable. Figure 4.4 shows plots of results from running the model on data from the Mount Jackson SNOTEL site. Here I've also included a plot of a model run where I set the precision for the observations higher (10e^{-5}) resulting in a smaller spread of the samples. The accuracy of the models performance in the two runs is summarized in Table 4.2.

Table 4.2: Model accuracy on Mount Jackson data runs

Samples	**Accuracy 10e^{-5}**	**Accuracy 10e^{-6}**
100.00%	0.996	1.000
95.00%	0.945	0.995
90.00%	0.924	0.987
75.00%	0.795	0.962
50.00%	0.603	0.887

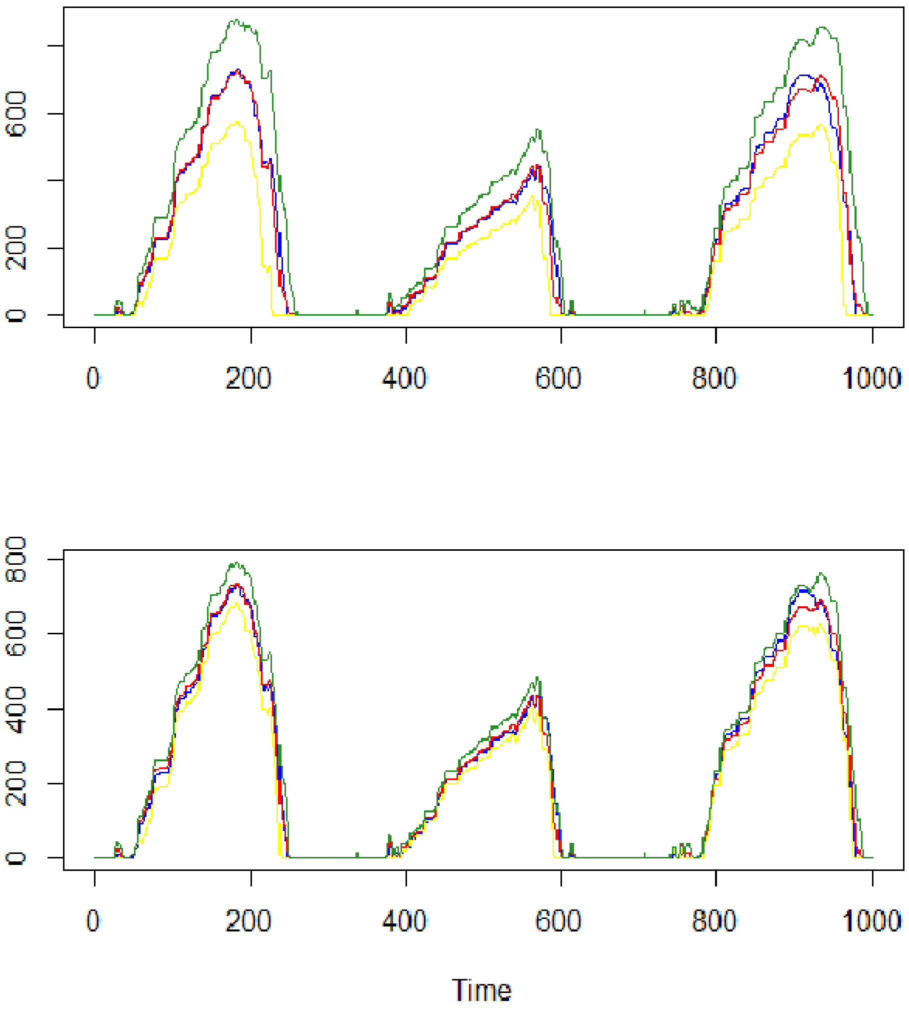


Figure 4.4: Plot of SWE samples mean (red) and 0.025 and 0.975 percentiles (yellow and green), and including observations (blue) for model run on data from Mount Jackson SNOTEL site, using precision 10e^{-6} (top) and 10e^{-5} (bottom) in the model for observations.

The mean of the samples are similar in both cases, and lie close to the observations for the most part, but the difference in spread in these results highlight an important point: We need to balance the numbers so that the observations are within a desired range of the samples an acceptable amount of the time. Looking back to Figure 3.3, it doesn't give information that can be used for much other than saying “there probably is snow”, estimating that there is between 1500 and 3000 mm of snow is not very meaningful. However the data from the model can be used to say with reasonable certainty an interval the value exist within, and even give its probability distribution (as seen in Figure 4.3).

5 Conclusion

When working with hydrologic models, both in research settings and for real world applications, it is important to be able to quantify the uncertainties that inevitably are present. These uncertainties have several origins; The model structure itself, since the complex workings of nature is hard to understand fully and completely describe numerically. There are uncertainties tied to the parameters of the models, and also to the observations that are used to feed it. Bayesian networks are popular in many areas, but have not been explored widely in hydrology, though other Bayesian methods have been gaining popularity.

To try and create a model that explicitly states uncertainties in its output I have translated the snow module of a popular deterministic model, the HBV model, into a Dynamic Bayesian Network (DBN). The model takes observations for precipitation and daily mean air temperatures and use it to calculate snow accumulation and melt. The main output from the model is the amount of snow stored, given in water equivalent (shortened SWE). After gaining confidence that the model design was sound, I attempted calibrating its parameters through Gibbs sampling. The results were that the model produced samples for SWE that when compared to the real observations, showed that the mean followed them pretty well, and distribution covered them for the most part.

With this I have shown that modeling hydrologic processes in a DBN could be viable, and this could provide a new way to explicitly include estimations of uncertainties in hydrologic models.

References

[1] D. R. Bourdin, S. W. Fleming and R. B. Stull: "Streamflow modelling: A primer on applications, approaches and challenges", Atmosphere-Ocean, 50, 2012

[2] P. K. Kitandis, R. L. Bras: "Real-time forecasting with a conceptual hydrologic model: 1. Analysis of uncertainty",Water Resources Research, pp.1025-1033, 1980

[3] P. Krause, D. P. Boyle and F. Bäse: "Comparison of different efficiency criteria for hydrological model assessment",Advances in Geosciences, 5, 2005

[4] D. Heckerman: "Probabilistic Similarity Networks", Technical Report, STAN-CS-1316, Dept. of Computer Science and Medicine, Stanford Univ., 1990

[5] R. Goldman: "A Probabilistic Approach to Language Understanding", Technical Report, CS-90-34, Dept. of Computer Science, Brown Univ., 1990

[6] J. Pearl: "Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning", Proc. of the 7th Conf. of the Cognitive Science Society, University of California, Irvine, CA., pp. 329-334, 1985

[7] J. Pearl: “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference”, Morgan Kaufmann, 1988

[8] Q. Duan, N. K. Ajami, X. Gao, S. Sorooshian: "Multi-model ensemble hydrologic prediction using Bayesian model averaging", pp.1371-1386, May 2007

[9] A. Raftery, F. Balabdaoui, T. Gneiting, M. Polakowski: "Using Bayesian Model Averaging to Calibrate Forecast Ensembles", Technical Report no 440, Dept. of Statistics, Univ. of Washington, 2003

[10] E. Charniak: "Bayesian networks without tears.",, 1991

[11] S. L. Lauritzen, D. J. Spiegelhalter: "Local computations with probabilities on graphical structures and their application to expert systems.", Journal of the Royal Statistical Society. Series B (Methodological), 1988

[12] S. Geman, D. Geman: "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 609-628, 1984

[13] A. E. Gelfand, A. F. M. Smith: "Sampling-Based Approaches to Calculating Marginal Densities", J. Am. Stat. Asso., 85, 389-409

[14] S. Bergstrøm: "Development and application of a conceptual runoff model for Scandinavian catchments.", Sveriges Meteorologiska och Hydrologiska Institut, 1976

[15] Å. Killingtveit, N. R. Sælthun: “Hydrology”, Norwegian Institute of Technology, 1995

[16] Sveriges Meteorologiska och Hydrologiska Institutt, “HBV-96”, [online] May 2014, http://www.smhi.se/sgn0106/if/hydrologi/images/HBV_fig1.gif

[17] G. Lindström, B. Johansson, M. Persson, M. Gardelin, and S. Bergstrøm: "Development and test of the distributed HBV-96 model.", J. Hydrol. 201, pp. 272-288, 1997

[18] Rango A., Martinec J. : "Revisiting the Degree-Day Method for Snowmelt Computations",Journal of the American Water Resources Association, 31, pp. 657-669, June 1995

[19] M. J. Druzdzel: "SMILE: Structural modeling, inference, and learning engine and GeNIe: A development environment for graphical decision-theoretic models", Proceedings of the 16th national conference on artifical intelligence (AAAI-99), pp. 342-343, July 1999

Appendixes

A Python code

```python
import random
import numpy as np

class Parameters(object):
    def __init__(self, **kwds):
        self.__dict__ = kwds


def makepara():
    d=dict(
        cx=random.uniform(3.0,6.0),
        scorr=random.uniform(1.1,1.5),
        meltth=random.uniform(-1.0,2.0),
        prcpth=random.uniform(-1.0,2.0),
        mlf=random.uniform(0.1,0.6)
    )
    return Parameters(**d)


def readdata(infile):
    data = []
    with open(infile,'r') as f:
        for line in f:
            data.append(line.split())
    return data

def runmodel(data,p):
    v=[]
    i=0

    swprev=0
    sweprev=0

    for line in data:
        prcp=float(line[0])
        airt=float(line[1])

        pmelt = p.cx*(airt-p.meltth) if airt>p.meltth else 0
        snow = 0 if airt>p.prcpth else prcp*p.scorr
        rain = 0 if snow > 0 else prcp
        melt = min(pmelt,sweprev-swprev) if pmelt>0 else 0

        tsw = swprev+melt+rain
        dsnow = sweprev-swprev+snow-melt
        swrel = tsw - (p.mlf*dsnow) if tsw>p.mlf*dsnow else 0
        sw = tsw-swrel

        swe = dsnow + sw
        v.append(swe)

        swprev=sw
        sweprev=swe

        i+=1
    return v

def main():
    infile='mthoodshort.dat'
    data = readdata(infile)
    datalen = len(data)
    n=5000
    p=[]
    v=[]
    for x in range(0,n):
        p.append(makepara())
        v.append(runmodel(data,p[x]))

    lists=[]
    for x in range(0,datalen):
        l=[]
        for y in v:
            l.append(y[x])
        lists.append(l)

    with open('dout.dat', 'w') as f:
        for x in lists:
            f.write(str(np.mean(x)) + ' ' + str(min(x)) + ' ' + str(max(x)) + '\n')



if __name__ == '__main__':
    main()
```

B JAGS model definition

```
model {
        for (i in 1:N) {
                snow[i] <- (airt[i] < prcpth) * prcp[i] * scorr
                rain[i] <- (1-(airt[i] < prcpth)) * prcp[i]

                swrel[i] <- (t_sw[i]>(dsnow[i]*mlf)) * (t_sw[i]-(dsnow[i]*mlf))

                sw[i] <- t_sw[i] - swrel[i]
                swe[i] <- dsnow[i] + sw[i]
                sweobs[i] ~ dnorm(swe[i], 0.000001)
        }


        melt[1] <- min((airt[1] > meltth) * cx * (airt[1]-meltth),sweinit-swinit)
        dsnow[1] <- sweinit - swinit + snow[1] - melt[1]
        t_sw[1] <- swinit + rain[1] + melt[1]

        for (i in 2:N) {
                melt[i] <- min((airt[i] > meltth) * cx * (airt[i]-meltth),swe[i-1]-sw[i-1])
                dsnow[i] <- swe[i-1] - sw[i-1] + snow[i] - melt[i]
                t_sw[i] <- sw[i-1] + rain[i] + melt[i]
        }


        scorr ~ dnorm(1.2, 1.0)
        cx ~ dnorm(4.0, 0.1)
        meltth ~ dnorm(0.0, 0.2)
        prcpth ~ dnorm(1.0, 0.2)
        mlf ~ dnorm(0.2, 3.0) I(0.0,)

        swinit <- 0
        sweinit <- 0
}
```

C C++ code

```
#include <iostream>
#include "smile/smile.h"
#include <typeinfo>
#include <sstream>
#include <fstream>
#include <random>
#include <stdio.h>
#include <time.h>
using namespace std;

vector<pair<double,double>> fileread(string datafile) {
        string line;
        ifstream infile(datafile);
        vector<pair<double,double>> data;
        if (infile.is_open())
        {
                double prcp,temp;
                while (getline(infile,line))
                {
                        istringstream iss(line);
                        string sub;
                        iss >> sub;
                        prcp = atof(sub.c_str());
                        iss >> sub;
                        temp = atof(sub.c_str());
                        data.push_back(make_pair(prcp,temp));
                }
                infile.close();
        }
        else cout << "Fail" << endl;

        return data;
}

void filewrite(int n){
        ofstream outfile;
        outfile.open("datatest.dat");
        tr1::mt19937 mt; // Mersenne Twister generator
        tr1::uniform_int<int> prcp(-40, 50);
        tr1::uniform_int<int> temp(-10, 10);
        for (int i = 1 ; i <= n; i++){
                int p=max(prcp(mt),0);
                int t=temp(mt);
                outfile << p << " " << t << endl;
        }
}



void writeResults (DSL_network& net, int n, string of) {
        cout << "Update beliefs.." << endl;
        clock_t starttime = clock();
        net.UpdateBeliefs();
        clock_t endtime = clock();
        cout << "Done in " << (double) (endtime - starttime) / CLOCKS_PER_SEC << endl;
        string nodename;
        int nodeid;
        DSL_valEqEvaluation *val;

        ofstream outfile;
        outfile.open(of.c_str());
        cout << "Write to file.." << endl;
        for (int i = 0 ; i < n ; i++) {
                if (i%100==0) cout << i << endl;
                nodename = "Swe" + to_string(i);
                nodeid = net.FindNode(nodename.c_str());
                val = dynamic_cast<DSL_valEqEvaluation *>(net.GetNode(nodeid)->Value());
                double mean,stddev,vmin,vmax;
                val->GetStats(mean,stddev,vmin,vmax);
                outfile << mean << " " << vmin << " " << vmax << endl;
        }

}


void makeNet(DSL_network& net, vector<pair<double,double>> indata){
        cout << "Make net" << endl;
        clock_t starttime = clock();
        //node identifiers
        int n=indata.size();
        int nid;

        DSL_equation *nodedef;
        DSL_node *node;
        DSL_node *firstswenode;

        firstswenode = net.GetNode(net.FindNode("Swe0"));

        int itype = DSL_EQUATION;
        string prcp_id,airt_id,pmelt_id,melt_id,snow_id,dsnow_id,swe_id,rain_id,t_sw_id,sw_id,swrel_id;
        string sweprev_id, swprev_id;

        //equations
        string prcp_eq="%s=%f"; //Prcp=val
        string airt_eq="%s=%f";
        string rain_eq="%s=If(%s>0,0,%s*Rcorr)";//Rain0=If(Snow0>0,0,Prcp0*Rcorr)
        string melt_eq="%s=If(%s>0,Min(%s, %s - %s),0)"; //Melt0=If(Pmelt0>0,Min(Pmelt0,Sweinit-Swinit),0)
        string pmelt_eq="%s=If(%s%>Meltth, Cx*(%s-Meltth), 0)";//Pmelt0=If(Airt0>Meltth,Cx*(Airt0-Meltth),0)
        string snow_eq="%s=If(%s>Prcpth,0,%s*Scorr)";//Snow0=If(Airt0>Prcpth,0,Prcp0*Scorr)
        string dsnow_eq="%s=%s-%s+%s-%s";//Dsnow0=Sweinit-Swinit+Snow0-Melt0
        string swe_eq="%s=%s+%s"; //Swe0=Dsnow0+Sw0
        string t_sw_eq="%s=%s+%s+%s";//t_Sw0=Swinit+Melt0+Rain0
        string swrel_eq="%s=If(%s>%s*0.4,%s-0.4*%s,0)";//Swrel0=If(t_Sw0>Dsnow0*0.4,t_Sw0-0.4*Dsnow0,0)
        string sw_eq="%s=%s-%s";//Sw0=t_Sw0-Swrel0
        char eqbuffer[500];

        //step 1
        nid = net.FindNode("Prcp0");
        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
        sprintf(eqbuffer, prcp_eq.c_str(), "Prcp0", indata[0].first);
        nodedef->SetEquation(eqbuffer);

        nid = net.FindNode("Airt0");
        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
        sprintf(eqbuffer, airt_eq.c_str(), "Airt0", indata[0].second);
        nodedef->SetEquation(eqbuffer);

        int netsize=100;


        for (int i = 1 ; i < n ; i++){
                if (i%100==0) cout << i << endl;
                prcp_id = "Prcp" + to_string(i);
                airt_id = "Airt" + to_string(i);
                rain_id = "Rain" + to_string(i);
                melt_id = "Melt" + to_string(i);
                pmelt_id = "Pmelt" + to_string(i);
                snow_id = "Snow" + to_string(i);
                swe_id = "Swe" + to_string(i);
                t_sw_id = "t_Sw" + to_string(i);
                dsnow_id = "Dsnow" + to_string(i);
                sw_id = "Sw" + to_string(i);
                swrel_id = "Swrel" + to_string(i);

                sweprev_id = "Swe" + to_string(i-1);
                swprev_id = "Sw" + to_string(i-1);


                // prcp
                nid = net.AddNode(itype, prcp_id.c_str());
                nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                sprintf(eqbuffer, prcp_eq.c_str(), prcp_id.c_str(), indata[i].first);
                nodedef->SetEquation(eqbuffer);



                // airt
                nid = net.AddNode(itype, airt_id.c_str());
                nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                sprintf(eqbuffer, airt_eq.c_str(), airt_id.c_str(), indata[i].second);
                nodedef->SetEquation(eqbuffer);

                // snow
                nid = net.AddNode(itype, snow_id.c_str());
                nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                sprintf(eqbuffer, snow_eq.c_str(), snow_id.c_str(), airt_id.c_str(), prcp_id.c_str());
//Snow0=If(Airt0>Uniform(-1,2),0,Prcp0*Uniform(1.15,1.5))
                nodedef->SetEquation(eqbuffer);


                // rain
                nid = net.AddNode(itype, rain_id.c_str());
                nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                sprintf(eqbuffer, rain_eq.c_str(), rain_id.c_str(), snow_id.c_str(), prcp_id.c_str());
```

```
//Rain0=If(Snow0>0,0,Prcp0*Uniform(1,1.2))
                        nodedef->SetEquation(eqbuffer);

                        //pmelt
                        nid = net.AddNode(itype, pmelt_id.c_str());
                        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                        //Pmelt0=If(Airt0>Meltth,Cx*(Airt0-Meltth),0)
                        sprintf(eqbuffer, pmelt_eq.c_str(), pmelt_id.c_str(), airt_id.c_str(),airt_id.c_str());
                        nodedef->SetEquation(eqbuffer);

                        // melt
                        nid = net.AddNode(itype, melt_id.c_str());
                        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                        //Melt0=If(Pmelt0>0,Min(Pmelt0,Sweinit-Swinit),0)
                        sprintf(eqbuffer, melt_eq.c_str(), melt_id.c_str(), pmelt_id.c_str(),pmelt_id.c_str(),sweprev_id.c_str(),
swprev_id.c_str());
                        nodedef->SetEquation(eqbuffer);

                        // t_sw
                        nid = net.AddNode(itype, t_sw_id.c_str());
                        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                        sprintf(eqbuffer, t_sw_eq.c_str(), t_sw_id.c_str(), swprev_id.c_str(), melt_id.c_str(), rain_id.c_str());
                        // t_Sw0=Swinit+Melt0+Rain0
                        nodedef->SetEquation(eqbuffer);


                        // dsnow
                        nid = net.AddNode(itype, dsnow_id.c_str());
                        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                        // Dsnow0=Sweinit-Swinit+Snow0-Melt0
                        sprintf(eqbuffer, dsnow_eq.c_str(), dsnow_id.c_str(), sweprev_id.c_str(), swprev_id.c_str(),
snow_id.c_str(), melt_id.c_str());
                        nodedef->SetEquation(eqbuffer);

                        //swrel
                        nid = net.AddNode(itype, swrel_id.c_str());
                        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                        //Swrel0=If(t_Sw0>Dsnow0*0.4,t_Sw0-0.4*Dsnow0,0)
                        sprintf(eqbuffer, swrel_eq.c_str(), swrel_id.c_str(), t_sw_id.c_str(), dsnow_id.c_str(), t_sw_id.c_str(),
dsnow_id.c_str());
                        nodedef->SetEquation(eqbuffer);

                        //sw Sw0=t_Sw0-Swrel0
                        nid = net.AddNode(itype, sw_id.c_str());
                        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                        sprintf(eqbuffer, sw_eq.c_str(), sw_id.c_str(), t_sw_id.c_str(), swrel_id.c_str());
                        nodedef->SetEquation(eqbuffer);



                        // swe Swe0=Dsnow0+Sw0
                        nid = net.AddNode(itype, swe_id.c_str());
                        nodedef = dynamic_cast<DSL_equation *>(net.GetNode(nid)->Definition());
                        sprintf(eqbuffer, swe_eq.c_str(), swe_id.c_str(), dsnow_id.c_str(), sw_id.c_str());
                        nodedef->SetEquation(eqbuffer);
```

```
        node = net.GetNode(nid);
        node->Info().Screen().position = firstswenode->Info().Screen().position;
        node->Info().Screen().position.center_X += 100*i;
    }
    clock_t endtime = clock();
    cout << "Done in " << (double) (endtime - starttime) / CLOCKS_PER_SEC << endl;
}

int main()
{
    string datafile = "mthoodshort.dat";
    string proto = "../dbn/eq6.xdsl";
    string outnet = "../dbn/a6.xdsl";
    string outfile = "out.dat";

    vector<pair<double,double>> indata;
    indata = fileread(datafile);
    int datasize = indata.size();
    cout << "Datas: " << datasize << endl;

    DSL_network net;

    net.ReadFile(proto.c_str());

    makeNet(net, indata);
    net.WriteFile(outnet.c_str());
    writeResults(net, datasize, outfile);

    return 0;
}
```