COMP2004 Coursework: Uncertain Reasoning

Tristan Aubrey-Jones (taj105@ecs.soton.ac.uk)

April 2, 2007

1 Simulation of events

In order to generate the 100,000 sample events for the Bayesian network I have written a MatLab function *model*1 to model the network and generate a single event, and another function *samplemodel* to repeat this 100,000 times and store the results in a matrix. These functions can both be found in Appendix 1 of this document.

2 Exercises based on generated events

1. The answers below were calculated from the data set generated above using a function I have called *calculateProbs* [Appendix 2] which takes a matrix of events, a matrix of 'test' conditions, and a matrix of 'given' conditions. For example to calculate $\tilde{P}(b1, c0|a1)$ on the sample matrix $s1$ I would use:

   ```
   answer = calculateProb(s1, [2 1; 3 0], [1 1]);
   ```

   calculateProb works by iterating all of the events in a sample, and keeping a count N_{given} of the number of events where the prior (given) conditions hold, and a count N_{cond} of the number of events where both the prior and conditional conditions hold. The empirical estimate of the probability is then given by N_{cond}/N_{given}.

   (a) These are the empirical estimates of the marginal probabilities.

Variable	Value	Probability
$\tilde{P}(A)$	a1	0.20038
	a0	0.79962
$\tilde{P}(B)$	b1	0.08975
	b0	0.91025
$\tilde{P}(C)$	c1	0.05892
	c0	0.94108
$\tilde{P}(D)$	d1	0.08234
	d0	0.91766
$\tilde{P}(E)$	e1	0.08289
	e0	0.91711

(b) Here, as one would expect, $\tilde{P}(B, C|A)$ approximates closely to $\tilde{P}(B|A) \times \tilde{P}(C|A)$. If worked out systematically from the network itself these probabilities should be equal, as by product rule $P(B, C|A) \equiv P(B|A) \times P(C|A)$, and so we can infer that the miner difference in this case is due to the randomness of the sample, and would diminish if a larger sample was taken.

$$
\begin{aligned}
\tilde{P}(b1, c1|a1) = 0.02395449 &\simeq \tilde{P}(b1|a1) \times \tilde{P}(c1|a1) = 0.02407680 \\
\tilde{P}(b1, c1|a0) = 0.00230109 &\simeq \tilde{P}(b1|a0) \times \tilde{P}(c1|a0) = 0.00245257 \\
\tilde{P}(b1, c0|a1) = 0.22666933 &\simeq \tilde{P}(b1|a1) \times \tilde{P}(c0|a1) = 0.22654702 \\
\tilde{P}(b1, c0|a0) = 0.04713489 &\simeq \tilde{P}(b1|a0) \times \tilde{P}(c0|a0) = 0.04698341 \\
\tilde{P}(b0, c1|a1) = 0.07211299 &\simeq \tilde{P}(b0|a1) \times \tilde{P}(c1|a1) = 0.07199068 \\
\tilde{P}(b0, c1|a0) = 0.04730997 &\simeq \tilde{P}(b0|a0) \times \tilde{P}(c1|a0) = 0.04715849 \\
\tilde{P}(b0, c0|a1) = 0.67726320 &\simeq \tilde{P}(b0|a1) \times \tilde{P}(c0|a1) = 0.67738551 \\
\tilde{P}(b0, c0|a0) = 0.90325405 &\simeq \tilde{P}(b0|a0) \times \tilde{P}(c0|a0) = 0.90340552
\end{aligned}
$$

(c) Below is a comparison of the empirical probabilities given by $\tilde{P}(B, C|A, D)$ and $\tilde{P}(B|A) \times \tilde{P}(C|A)$. Unlike part (b), the probabilities here do not approximate to each other, as the information about the outcome D gives posterior, information about the likely outcomes of B and C.

$$
\begin{aligned}
&\tilde{P}(b1, c1|a1, d1) = 0.16253444 \quad \tilde{P}(b1|a1) \times \tilde{P}(c1|a1) = 0.02407680\\
&\tilde{P}(b1, c1|a1, d0) = 0.00705487 \quad ..\\
&\tilde{P}(b1, c1|a0, d1) = 0.02113606 \quad \tilde{P}(b1|a0) \times \tilde{P}(c1|a0) = 0.00245257\\
&\tilde{P}(b1, c1|a0, d0) = 0.00075772 \quad ..\\
&\tilde{P}(b1, c0|a1, d1) = 0.20752984 \quad \tilde{P}(b1|a1) \times \tilde{P}(c0|a1) = 0.22654702\\
&\tilde{P}(b1, c0|a1, d0) = 0.22900336 \quad ..\\
&\tilde{P}(b1, c0|a0, d1) = 0.06439894 \quad \tilde{P}(b1|a0) \times \tilde{P}(c0|a0) = 0.04698341\\
&\tilde{P}(b1, c0|a0, d0) = 0.04572024 \quad ..\\
&\tilde{P}(b0, c1|a1, d1) = 0.32552801 \quad \tilde{P}(b0|a1) \times \tilde{P}(c1|a1) = 0.07199068\\
&\tilde{P}(b0, c1|a1, d0) = 0.04120941 \quad ..\\
&\tilde{P}(b0, c1|a0, d1) = 0.30828930 \quad \tilde{P}(b0|a0) \times \tilde{P}(c1|a0) = 0.04715849\\
&\tilde{P}(b0, c1|a0, d0) = 0.02592482 \quad ..\\
&\tilde{P}(b0, c0|a1, d1) = 0.30440771 \quad \tilde{P}(b0|a1) \times \tilde{P}(c0|a1) = 0.67738551\\
&\tilde{P}(b0, c0|a1, d0) = 0.72273236 \quad ..\\
&\tilde{P}(b0, c0|a0, d1) = 0.60617569 \quad \tilde{P}(b0|a0) \times \tilde{P}(c0|a0) = 0.90340552\\
&\tilde{P}(b0, c0|a0, d0) = 0.92759722 \quad ..
\end{aligned}
$$

(d) Below is a comparison between $\tilde{P}(D|A, B, C)$ and $\tilde{P}(D|B, C)$. As one would expect the two approximate to each other, due to the markov property of D such that it is only conditional upon B and C. Thus when the outcomes of these are both known, the outcome of A gives no additional information.

$$
\begin{aligned}
&\tilde{P}(d1|b1, c1, a1) = 0.73750000 \quad \simeq \tilde{P}(d1|b1, c1) = 0.72590361\\
&\tilde{P}(d1|b1, c1, a0) = 0.69565217 \quad ..\\
&\tilde{P}(d1|b1, c0, a1) = 0.09951563 \quad \simeq \tilde{P}(d1|b1, c0) = 0.10131151\\
&\tilde{P}(d1|b1, c0, a0) = 0.10347572 \quad ..\\
&\tilde{P}(d1|b0, c1, a1) = 0.49065744 \quad \simeq \tilde{P}(d1|b0, c1) = 0.49273145\\
&\tilde{P}(d1|b0, c1, a0) = 0.49352366 \quad ..\\
&\tilde{P}(d1|b0, c0, a1) = 0.04885417 \quad \simeq \tilde{P}(d1|b0, c0) = 0.05051459\\
&\tilde{P}(d1|b0, c0, a0) = 0.05082657 \quad ..
\end{aligned}
$$

(e) Below is a comparison between $\tilde{P}(E|A, B, C, D)$ and $\tilde{P}(E|C)$. Here, in a similar way to above, we see that the two approximate. This is due to markov property of E such that it is only directly dependent on C.

$$
\begin{array}{ll}
\tilde{P}(e1|c1,a1,b1,d1) = 0.59039548 & \simeq \tilde{P}(e1|c1) = 0.60420910 \\
\tilde{P}(e1|c1,a1,b1,d0) = 0.61904762 & .. \\
\tilde{P}(e1|c1,a1,b0,d1) = 0.58956276 & .. \\
\tilde{P}(e1|c1,a1,b0,d0) = 0.61277174 & .. \\
\tilde{P}(e1|c1,a0,b1,d1) = 0.62500000 & .. \\
\tilde{P}(e1|c1,a0,b1,d0) = 0.57142857 & .. \\
\tilde{P}(e1|c1,a0,b0,d1) = 0.60685592 & .. \\
\tilde{P}(e1|c1,a0,b0,d0) = 0.60490605 & .. \\
\tilde{P}(e1|c0,a1,b1,d1) = 0.05973451 & \simeq \tilde{P}(e1|c0) = 0.05025078 \\
\tilde{P}(e1|c0,a1,b1,d0) = 0.05061125 & .. \\
\tilde{P}(e1|c0,a1,b0,d1) = 0.05580694 & .. \\
\tilde{P}(e1|c0,a1,b0,d0) = 0.05113108 & .. \\
\tilde{P}(e1|c0,a0,b1,d1) = 0.06153846 & .. \\
\tilde{P}(e1|c0,a0,b1,d0) = 0.04557561 & .. \\
\tilde{P}(e1|c0,a0,b0,d1) = 0.04848815 & .. \\
\tilde{P}(e1|c0,a0,b0,d0) = 0.05020786 & ..
\end{array}
$$

2. Below are the exact probabilities for 1, computed by directly manipulating the joint distribution. In order to compute these probabilities I have programmed a MATLAB function named *computeProbs* which can be found in Appendix 3. For example to calculate $P(B, C|A)$ I would invoke:

```
answer = computeProbs([2 3], [1]);
```

computeProbs takes 2 row vectors, of conditional variables and prior variables. It the proceeds through all combinations of these input variables using recursion, and for each combination calculates its probability by exploiting the identity $P(X|Y) = P(X, Y)/P(Y)$. It uses a function named *computeProb* to calculate the joint probability of both $P(X, Y)$ and $P(Y)$.

computeProb works by summing over the individual probabilities of all of the events in the joint distribution $P(A, B, C, D, E)$ where the constrained variables take the values specified. It does this by proceeding recursively over all of the variables, and if the current variable is in the constraints, it is fixed at that value, otherwise the sum of the probabilities is taken for when that variable is 1 or 0. It uses a function named $profOfEvent$ to calculate the probability of an individual event in the network.

probOfEvent works by using the product rule, and multiplying the appropriate probabilities from the variables' CDT's, for the event specified.

Thus all of the probabilities can be calculated by directly manipulating the joint distribution.

(a) These are the marginal probabilities computed directly from the joint distribution using the *computeProbs* function explained above.

i. $P(a1) = 0.2$
ii. $P(b1) = 0.09$
iii. $P(c1) = 0.06$
iv. $P(d1) = 0.0829$
v. $P(e1) = 0.083$

(b) Below the equivalence $P(B, C|A) \equiv P(B|A) \times P(C|A)$ is demonstrated.

$$\begin{aligned}
P(b1, c1|a1) = 0.025 &= P(b1|a1) \times P(c1|a1) = 0.025 \\
P(b1, c1|a0) = 0.0025 &= P(b1|a0) \times P(c1|a0) = 0.0025 \\
P(b1, c0|a1) = 0.225 &= P(b1|a1) \times P(c0|a1) = 0.225 \\
P(b1, c0|a0) = 0.0475 &= P(b1|a0) \times P(c0|a0) = 0.0475 \\
P(b0, c1|a1) = 0.075 &= P(b0|a1) \times P(c1|a1) = 0.075 \\
P(b0, c1|a0) = 0.0475 &= P(b0|a0) \times P(c1|a0) = 0.0475 \\
P(b0, c0|a1) = 0.675 &= P(b0|a1) \times P(c0|a1) = 0.675 \\
P(b0, c0|a0) = 0.9025 &= P(b0|a0) \times P(c0|a0) = 0.9025
\end{aligned}$$

(c) Below is a comparison between $P(B, C|A, D)$ and $P(B|A)$ and $P(C|A)$:

$$
\begin{array}{ll}
P(b1, c1|a1, d1) = 0.16666667 & P(b1|a1) \times P(c1|a1) = 0.025 \\
P(b1, c1|a1, d0) = 0.00704225 & \\
P(b1, c1|a0, d1) = 0.02483444 & P(b1|a0) \times P(c1|a0) = 0.0025 \\
P(b1, c1|a0, d0) = 0.00067604 & \\
P(b1, c0|a1, d1) = 0.20000000 & P(b1|a1) \times P(c0|a1) = 0.225 \\
P(b1, c0|a1, d0) = 0.22816901 & \\
P(b1, c0|a0, d1) = 0.06291391 & P(b1|a0) \times P(c0|a0) = 0.0475 \\
P(b1, c0|a0, d0) = 0.04624121 & \\
P(b0, c1|a1, d1) = 0.33333333 & P(b0|a1) \times P(c1|a1) = 0.075 \\
P(b0, c1|a1, d0) = 0.04225352 & \\
P(b0, c1|a0, d1) = 0.31456954 & P(b0|a0) \times P(c1|a0) = 0.0475 \\
P(b0, c1|a0, d0) = 0.02568956 & \\
P(b0, c0|a1, d1) = 0.30000000 & P(b0|a1) \times P(c0|a1) = 0.675 \\
P(b0, c0|a1, d0) = 0.72253521 & \\
P(b0, c0|a0, d1) = 0.59768212 & P(b0|a0) \times P(c0|a0) = 0.9025 \\
P(b0, c0|a0, d0) = 0.92739319 &
\end{array}
$$

(d) Below the equivalence $P(D|A, B, C) \equiv P(D|B, C)$ is demonstrated. This is due to the markov property of D such that it is only dependent directly on the outcomes of B and C. Thus the additional knowledge about the outcome of A has no effect.

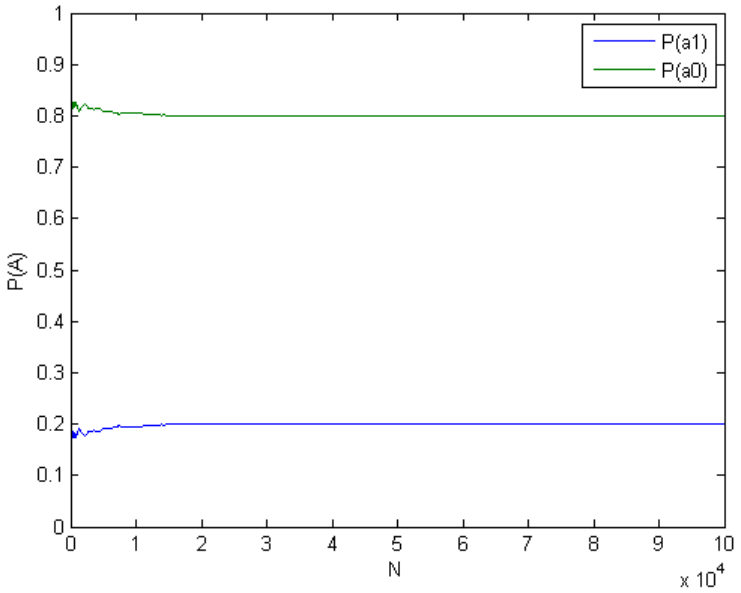
$$
\begin{aligned}
P(D|A, B, C) &= \frac{P(A, B, C, D)}{P(A, B, C)} \\
&= \frac{P(A)P(B|A)P(C|A)P(D|B, C)}{P(A)P(B|A)P(C|A)} \\
&= P(D|B, C)
\end{aligned}
$$

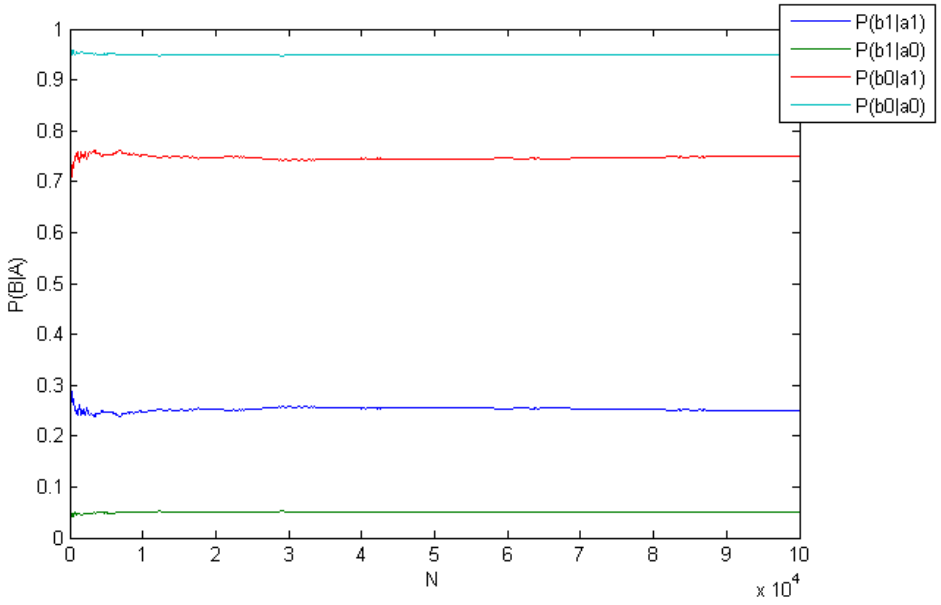
$$
\begin{array}{ll}
P(d1|b1, c1, a1) = 0.75000000 & = P(d1|b1, c1) = 0.75000000 \\
P(d1|b1, c1, a0) = 0.75000000 & .. \\
P(d1|b1, c0, a1) = 0.10000000 & = P(d1|b1, c0) = 0.10000000 \\
P(d1|b1, c0, a0) = 0.10000000 & .. \\
P(d1|b0, c1, a1) = 0.50000000 & = P(d1|b0, c1) = 0.50000000 \\
P(d1|b0, c1, a0) = 0.50000000 & .. \\
P(d1|b0, c0, a1) = 0.05000000 & = P(d1|b0, c0) = 0.05000000 \\
P(d1|b0, c0, a0) = 0.05000000 & ..
\end{array}
$$

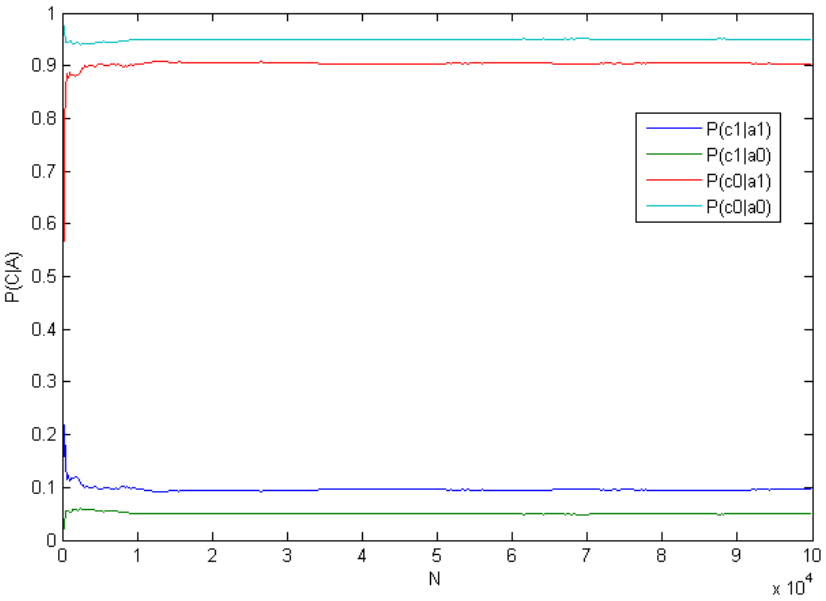
(e) Below the equivalence $P(E|A, B, C, D) \equiv P(E|C)$ is demonstrated. This is due to the markov property of E such that it is only dependent directly on the outcome of C.

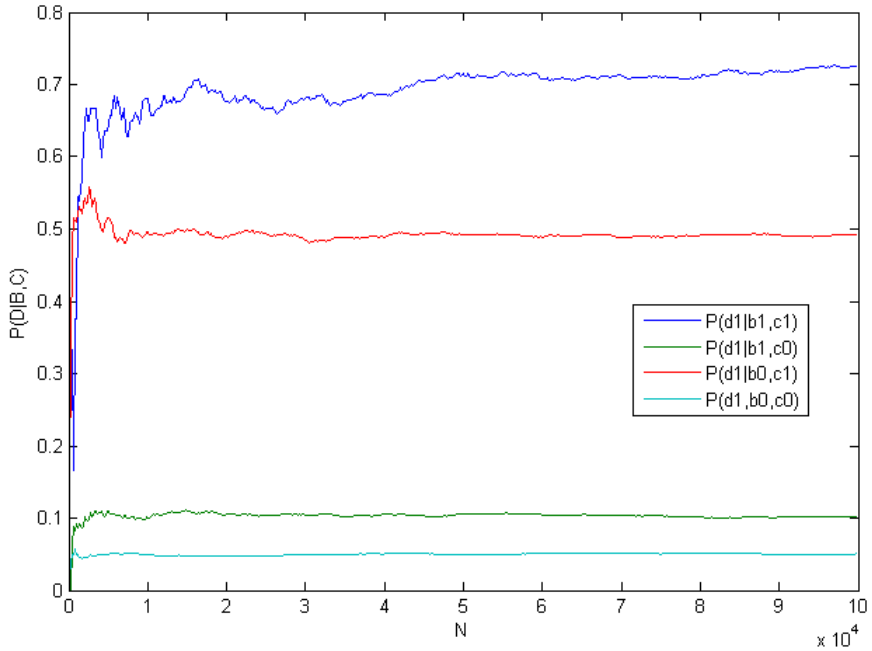
$$
\begin{aligned}
P(e1|c1,a1,b1,d1) &= 0.60000000 && = P(e1|c1) = 0.60000000 \\
P(e1|c1,a1,b1,d0) &= 0.60000000 && .. \\
P(e1|c1,a1,b0,d1) &= 0.60000000 && .. \\
P(e1|c1,a1,b0,d0) &= 0.60000000 && .. \\
P(e1|c1,a0,b1,d1) &= 0.60000000 && .. \\
P(e1|c1,a0,b1,d0) &= 0.60000000 && .. \\
P(e1|c1,a0,b0,d1) &= 0.60000000 && .. \\
P(e1|c1,a0,b0,d0) &= 0.60000000 && .. \\
P(e1|c0,a1,b1,d1) &= 0.05000000 && = P(e1|c0) = 0.05000000 \\
P(e1|c0,a1,b1,d0) &= 0.05000000 && .. \\
P(e1|c0,a1,b0,d1) &= 0.05000000 && .. \\
P(e1|c0,a1,b0,d0) &= 0.05000000 && .. \\
P(e1|c0,a0,b1,d1) &= 0.05000000 && .. \\
P(e1|c0,a0,b1,d0) &= 0.05000000 && .. \\
P(e1|c0,a0,b0,d1) &= 0.05000000 && .. \\
P(e1|c0,a0,b0,d0) &= 0.05000000 && ..
\end{aligned}
$$

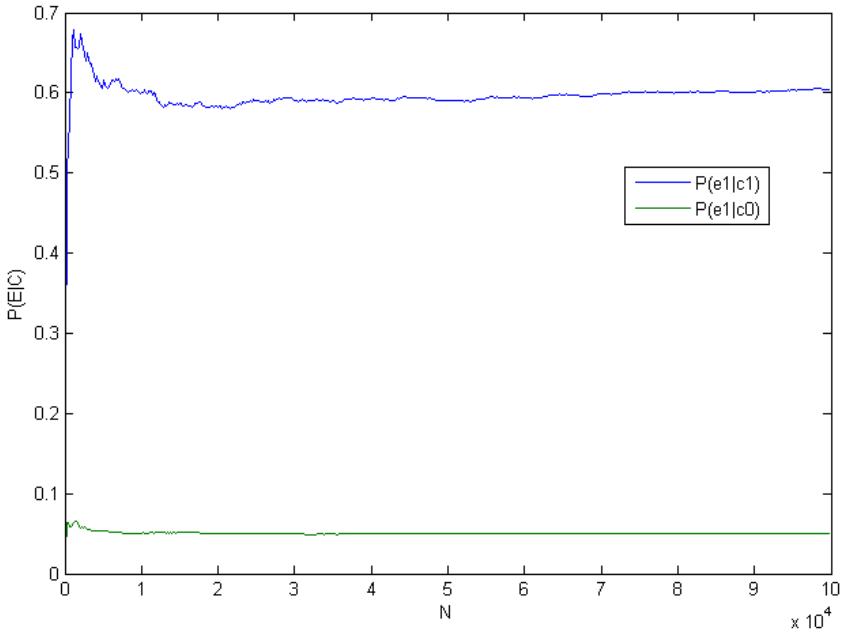
3. (a) Graphs showing comparisons between exact probabilities and the empirical estimates for increasing sample sizes are shown below. From them it is clear to see that with small sizes of N the estimates vary wildly, but as N increases the values settle close to the actual probabilities. These graphs were produced by the function *plotQuestion3A* which can be found in Appendix A.




P(b1|a1)
P(b1|a0)
P(b0|a1)
P(b0|a0)
P(B|A)
N
x 10^4

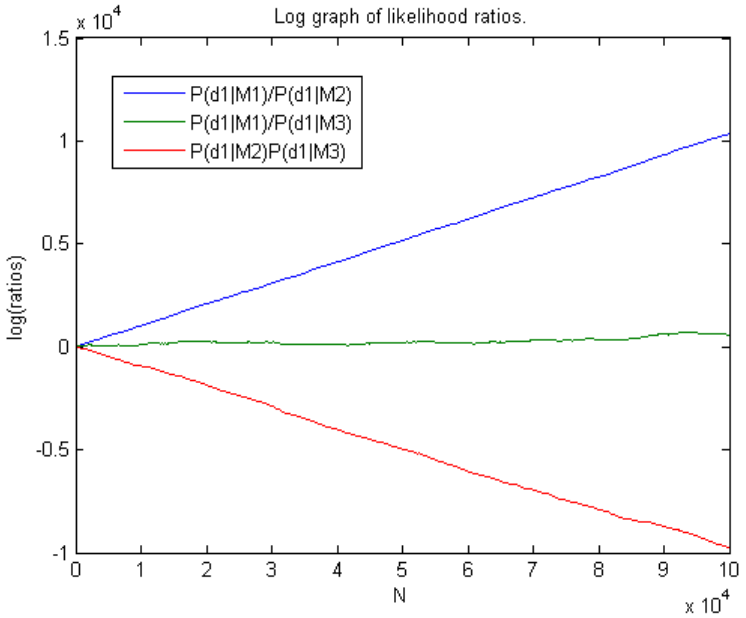

P(c1|a1)
P(c1|a0)
P(c0|a1)
P(c0|a0)
P(C|A)
N
x 10^4



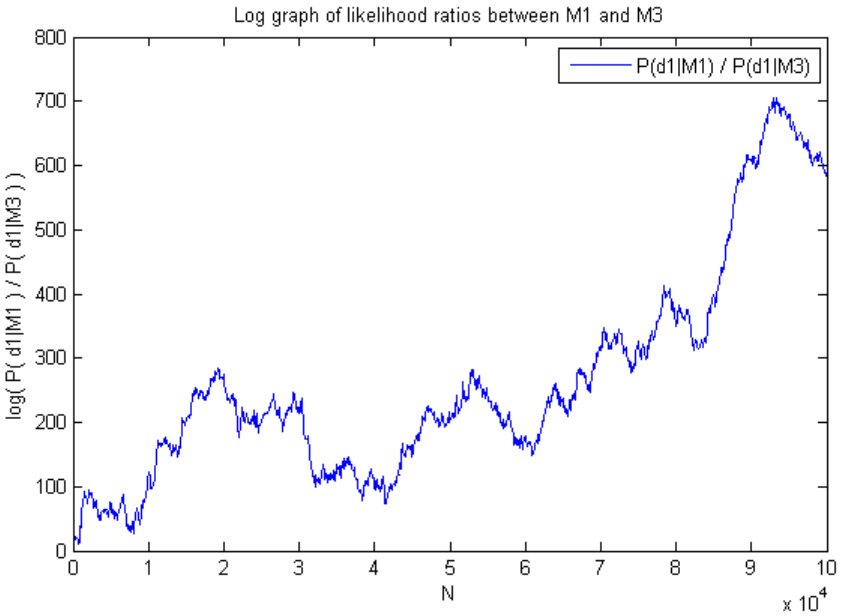


(b) The graph below shows the likelihood ratios [*computeLikelihoodRatios* Appendix 4] of the model for the data in $d_i \in S_N$. As the sample size increases the ratios increase (or decrease) exponentially and so I have computed the values (and plotted) them using natural

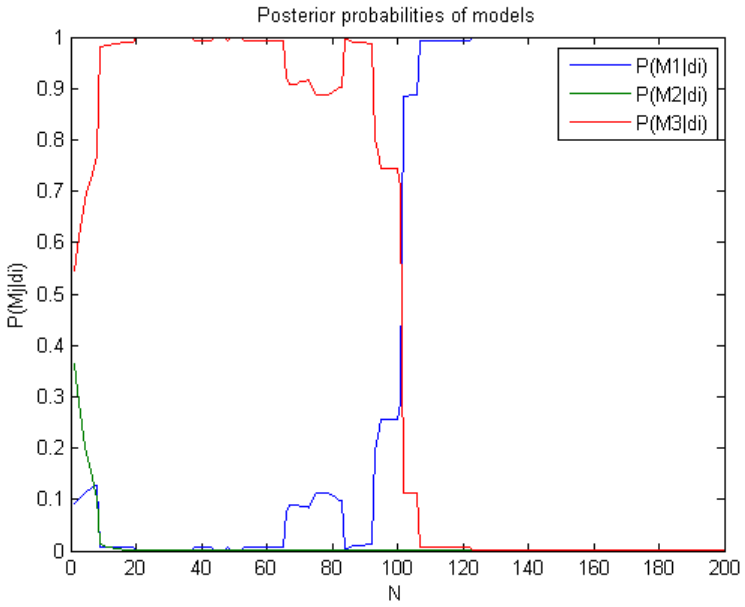
logarithms. From this it is clear to see that M_2 is much less likely than M_1 or M_3 and that M_1 and M_3 seem to have comparable likelihoods.



When the likelihood ratio between M_1 and M_3 is examined more closely (see graph below), it is still clear that M_1 has a greater likelihood, as one would expect. However, the graph above highlights that this is not not nearly as pronounced as in the case of M_2. This is due to the fact that M_3 only differs from M_1 in that C depends on both A and B, whereas in M_2 B is not conditional on A and D is not conditional on B.



(c) In order to calculate and compare the posterior probabilities on the three models I have written a function *computePosteriorProbs* [Appendix 4]. The sum over the hypotheses which makes up the denominator of the posterior probability calculation drops to 0 very quickly (giving divide by zero errors) as the size of N is increased. This is due to the fact that the probabilities settle to very close to certainty (1) and impossabilty (0), quickly growing beyond machine representation. I am therefore only showing the function in the range $N \in [0, 200]$ here.



Here above we see that immediately (around $N = 10$) $P(M_2|d_i)$ tends to impossible, whereas it is not until nearly $N = 110$ that $P(M_3|d_i)$ drops to the same. Interestingly for $N \in [1, 100]$ it seems that M_3 is much more likely than M_1, though this changes at $N = 100$ and from then on $P(M_1|d_i)$ tends to certainty.

4. (a) I worked out the following d-separation properties of nodes in Model 1, using the Bayes Ball algorithm.

$P(B, C\|A)$	$\{B\} \perp \{C\}\|\{A\}$
$P(B, C\|A, D)$	$\{B\}$ is not d-separate $\{C\}\|\{A, D\}$
$P(A, D\|B)$	$\{A\}$ is not d-separate $\{D\}\|\{B\}$
$P(A, D\|B, C)$	$\{A\} \perp \{D\}\|\{B, C\}$

(b) Below are the d-separation properties for models M_2 and M_3.

Sets	Model 1	Model 2	Model 3
$\{B\} \perp \{C\}\|\{A\}$	Yes	No	No
$\{B\} \perp \{C\}\|\{A, D\}$	No	No	No
$\{A\} \perp \{D\}\|\{B\}$	No	No	No
$\{A\} \perp \{D\}\|\{B, C\}$	Yes	Yes	Yes

$\{B\} \perp \{C\}\|\{A\}$ differs from Model 1 in models 2 and 3. This is showing that in M_1 B is independent of C, given A and thus $P(B, C|A) = P(B|A) \times P(C|A)$, whereas in M_2, M_3 C is dependent on B.

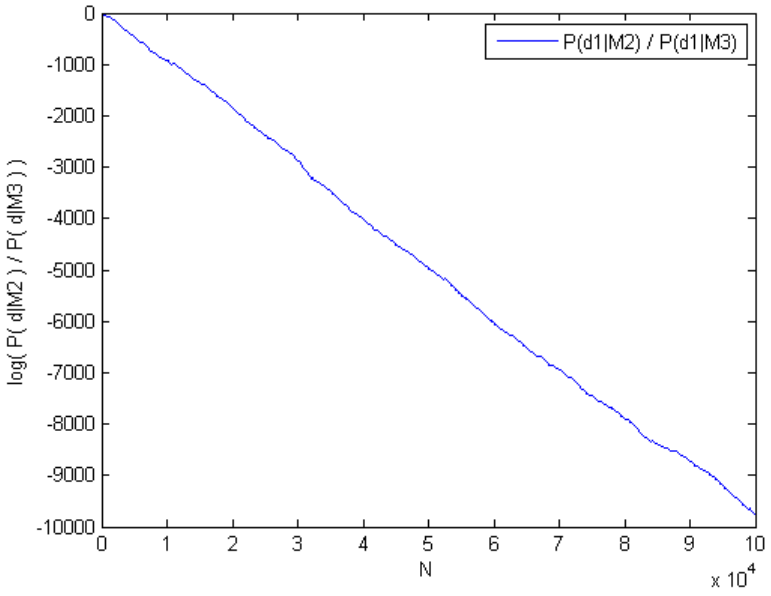
5. Below is the likelihood ratio between M_2 and M_3, in terms of the conditional probabilities in their respective CDTs.

$$\begin{aligned} d_i &= (a, b, c, d, e) \\ \frac{P(d_i|M_2)}{P(d_i|M_3)} &= \frac{P(a)P(b)P(c|a,b)P(d|c)P(e|c)}{P(a)P(b|a)P(c|a,b)P(d|b,c)P(e|c)} \\ &= \frac{P(b)P(d|c)}{P(b|a)P(d|b,c)} \end{aligned}$$

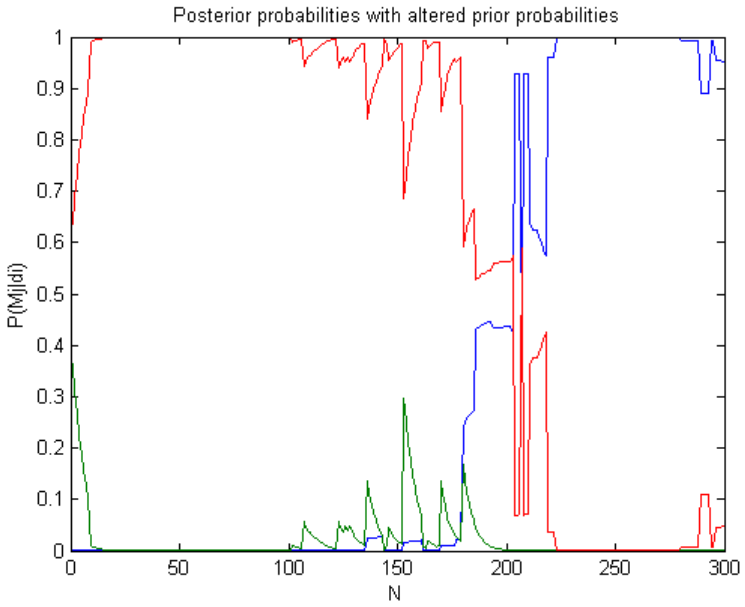
Here we see that the likelihood of M_2 is less than M_3 when:

$$P(b)P(d|c) < P(b|a)P(d|b,c) \tag{1}$$

This will occur in data where there is some correlation in the data between a and b, and between d and b. Both of these correlations are true of the sample data generated for M_1, and so this relationship can be illustrated with the following graph of the likelihood ratios from question 3b.



6. In order to make M_3 come out favorably for $N \in [1, 200]$ the prior probability of M_1 must be drastically decreased, and the prior probability $P(M_3)$ similarly increased. Below is a graph of the function for the range $N \in [1, 300]$ with prior probabilities in the ratio $5 : 10,000,000 : 13,999,999$.



Here M_3 now comes out favorably over M_1 between $N = [110, 200]$, but in order to ensure that the prior probabilities have had to be changed by near a factor of a million.

3 Appendix 1: Simulation source code

samplemodel **function** This function takes a function handle, pointing to a function which returns a random event for its Bayesian network, and the size of the sample to generate. It repeatedly invokes the model's function, generating row vector events, and placing them as the rows in the resultant matix.

```
function set = samplemodel(fmodel, n);
% SAMPLEMODEL samples the fmodel function handle n times
% and returns the resulting events in a matrix
% with 1 event per row.

%% sample
set = [];
for i = 1:n
    event = fmodel();
    set(i, 1:length(event)) = event;
end
```

*model*1 **function** This function models the 1st Bayesian network in the coursework, and returns a random event for this network as a row vector $(ABCDE)$. A handle of this function can be passed to *samplemodel* to generate a sample for it.

```
function event = model1();
% NETWORK1 is the first bayesian network
% shown in figure 1 of the coursework.

%% A
a = brand(0.2);

%% B
if (a == 1), b = brand(0.25);
else b = brand(0.05);
end

%% C
if (a == 1), c = brand(0.1);
else c = brand(0.05);
end
```

```
%% D
if (b == 1)
    if (c == 1), d = brand(0.75);
    else d = brand(0.1);
    end
else
    if (c == 1), d = brand(0.5);
    else d = brand(0.05);
    end
end

%% E
if (c == 1), e = brand(0.6);
else e = brand(0.05);
end

%% assign to result
event = [a b c d e];
```

brand **function** This function makes a random binary decision returning 1 for true, and 0 for false. It returns decsions in the distribution given by the probability of a true, specified by the parameter p.

```
function f = brand(p);
% BRAND makes a random decision
% It generates a random number and if that
% number is less than the probability p, it returns
% 1 otherwise it returns 0.

%% Return
if rand <= p, f = 1;
else f = 0;
end
```

4 Appendix 2: Empirical estimation source

calculateProbs **function** This function calculates the probabilities for $\tilde{P}(condVars|givenVars)$f the sample data generated in section 1. It uses recursion to go through all of the different combinations values for the variables specified, and uses *calculateProb* to calculate the probability of each.

```
function probs = calculateProbs(s1, condVars, givenVars,
                                 conds, given, printLatex)
% CALCULATEPROBS calculates the vector of probabilities
% for all the combinations of conditions and givens.

% start
if (nargin == 3)
    display('\begin{tabular}{c}');
    probs = calculateProbs(s1, condVars, givenVars, [], [], 1);
    display('\end{tabular}');
else
    % recursing through combinations
    if (nargin == 6)
        % check if all conditions are specified
        szConds = size(conds);
        szGiven = size(given);
        if szConds(1) == length(condVars)
            if szGiven(1) == length(givenVars)
                % base case
                p = calculateProb(s1, conds, given);
                probs = [p];
                % display in latex
                sconds = condsToStr(conds);
                sgiven = condsToStr(given);
                if (printLatex)
                    display(sprintf(
                    '  $\\tilde{P}(%s|%s) = %4.8f$\\\\', sconds, sgiven, p));
                end
            else
                % branch for current given
                i = szGiven(1) + 1;
                v = givenVars(i);
                % when var is 1
                given(i, 1:2) = [v 1];
```

```
                probs = calculateProbs(s1, condVars, givenVars,
                 conds, given, printLatex);
                % when var is 0
                given(i, 1:2) = [v 0];
                p2 = calculateProbs(s1, condVars, givenVars,
                 conds, given, printLatex);
                sz = size(probs);
                probs((sz(1)+1):(sz(1)*2), 1:sz(2)) = p2;
            end
        else
            % branch for current cond
            i = szConds(1) + 1;
            v = condVars(i);
            % when var is 1
            conds(i, 1:2) = [v 1];
            probs = calculateProbs(s1, condVars, givenVars,
             conds, given, printLatex);
            % when var is 0
            conds(i, 1:2) = [v 0];
            p2 = calculateProbs(s1, condVars, givenVars,
             conds, given, printLatex);
            sz = size(probs);
            probs((sz(1)+1):(sz(1)*2), 1:sz(2)) = p2;
        end
    else
        error('incorrect arguments.');
    end
```

calculateProb **function** This function is used to calculate the probability $\tilde{P}(X = x|Y = y)$ for the sample data. It iterates over the sample events keeping a count N_x of the number of events for where $X = x$ holds, and a count $N_{x,y}$ of the number of events where $Y = y$ and $X = x$ hold. The probability is then simply given by $N_{x,y}/N_x$ as $P(x, y) = P(x, y)/P(x)$.

```
function result = calculateProb(sample, conds, given);
% CALCULATEPROB calculates the probability of the
% conditions conds occuring for all cases where given occurs.
%
% The parameters 'conds' and 'given' are matrices where each row
% is a condition. Each condition is a two column row vector where
% the left value cond(1) is the column index of the variable in
```

```
% the event, and the right value cond(2) is the value that it should
% take for the event to be true.
%
% e.g. calculateProb([1 1; 1 0; 1 0], [2 1], []) = 1/3;
%      calculateProb([1 1; 1 0; 1 0], [2 1], [2 1]) = 1;
%      calculateProb([1 1; 1 0; 1 0], [2 1], [1 1]) = 1;

sz = size(sample);
cz = size(conds);
gz = size(given);
tot = 0.0;
count = 0;

% for all events in sample
for i = 1:sz(1)
    % check if all given conditions apply
    alltrue = 1;
    for ci = 1:gz(1)
        % check current condition
        c = given(ci,1:gz(2));
        if sample(i, c(1)) ~= c(2)
            alltrue = 0;
            break;
        end
    end

    % if given true
    if alltrue == 1
        % increment "out of" count
        count = count + 1;

        % check if all conditions apply
        alltrue = 1;
        for ci = 1:cz(1)
            % check current condition
            c = conds(ci, 1:cz(2));
            if sample(i, c(1)) ~= c(2)
                alltrue = 0;
                break;
            end
        end
```

```
        % if conditions hold add to
        if alltrue == 1
            tot = tot + 1;
        end
    end
end

% return probability of conds occuring given that
% given already has
result = tot / count;
```

5 Appendix 3: Analytical source code

computeProbs **function** This function computes the probabilities for the variables specified as $P(condVars|givenVars)$ by using *computeProb* to directly manipulate the joint distribution of model 1. It uses recursion to go through all the combinations of values for the variables specified.

```
function probs = computeProbs(condVars, givenVars, conds, given)
% computeProbs computes the vector of probabilities
% for all the combinations of conditions and givens
% by directly manipulating the joint distribution.

% start
if (nargin == 2)
    display('\begin{tabular}{c}');
    probs = computeProbs(condVars, givenVars, [], []);
    display('\end{tabular}');
else
    % recursing through combinations
    if (nargin == 4)
        % check if all conditions are specified
        szConds = size(conds);
        szGiven = size(given);
        if szConds(1) == length(condVars)
            if szGiven(1) == length(givenVars)
                % P(X|Y) = P(X,Y) / P(Y)
                if szGiven(1) > 0
                    cs = conds;
                    for i = 1:szGiven(1)
                        cs(i + szConds(1), 1:2) = given(i, 1:2);
                    end
                    pXY = computeProb(cs);
                    pY = computeProb(given);
                    p = pXY ./ pY;
                else
                    p = computeProb(conds);
                end
                probs = [p(1)];
                % display in latex
                sconds = condsToStr(conds);
                sgiven = condsToStr(given);
```

```
                display(sprintf(
                  '  $P(%s|%s) = %4.8f$\\\\', sconds, sgiven, probs));
            else
                % branch for current given
                i = szGiven(1) + 1;
                v = givenVars(i);
                % when var is 1
                given(i, 1:2) = [v 1];
                probs = computeProbs(condVars, givenVars, conds, given);
                % when var is 0
                given(i, 1:2) = [v 0];
                p2 = computeProbs(condVars, givenVars, conds, given);
                sz = size(probs);
                probs((sz(1)+1):(sz(1)*2), 1:sz(2)) = p2;
            end
        else
            % branch for current cond
            i = szConds(1) + 1;
            v = condVars(i);
            % when var is 1
            conds(i, 1:2) = [v 1];
            probs = computeProbs(condVars, givenVars, conds, given);
            % when var is 0
            conds(i, 1:2) = [v 0];
            p2 = computeProbs(condVars, givenVars, conds, given);
            sz = size(probs);
            probs((sz(1)+1):(sz(1)*2), 1:sz(2)) = p2;
        end
    else
        error('incorrect arguments.');
    end
end
```

computeProb **function** This function computes a probability by directly manipulating the joint distribution for model 1. It takes a matrix of random variables, and their corresponding values, and works out the probablity of the disjunction of those variables by summing over the probabilities of all the events where they hold. It uses $probOfEvent$ to calculate the probability of each given event.

```
function prob = computeProb(conds, currentVar)
```

```
% COMPUTEPROB computes the probability of all of
% the conditions occuring by directly manipulating
% the joint distribution. It works by summing over
% all values of unspecified variables.

if nargin == 1
    % start
    prob = computeProb(conds, 1);
else
    % if still constructing events
    sz = size(conds);
    if currentVar ~= 6
        % recursing through
        sumOver = 1;
        for i = 1:sz(1)
            if conds(i, 1) == currentVar
                sumOver = 0;
                break;
            end
        end
        if sumOver
            % summing over both values of this variable
            conds(sz(1)+1, 1:2) = [currentVar 1];
            prob = computeProb(conds, currentVar+1);
            conds(sz(1)+1, 1:2) = [currentVar 0];
            prob = prob + computeProb(conds, currentVar+1);
        else
            % this variable appears in the interesection
            prob = computeProb(conds, currentVar+1);
        end
    else
        % compute the probability of this event
        event = zeros(1,5);
        for i = 1:sz(1)
            event(1,conds(i,1)) = conds(i,2);
        end
        prob = probOfEvent(event);
    end
end
```

probOfEvent **function** This function calculates the probability of the event e occuring, given each of the 3 models. It works by multiplying the probabilities of the variables $P(A)P(B|A)P(C|A)P(D|B,C)P(E|C)$ for the values in the event.

```
function probs = probOfEvent(e)
% PROBOFEVENT calculates the probability of
% the event e occuring, given that a bayesian model
% produced it for the 3 models.

probs = [0.0; 0.0; 0.0];

% MODEL 1
% A,B,C
if e(1)
    p1 = 0.2;
    p1 = p1 * ifv(e(2),0.25,0.75);
    p1 = p1 * ifv(e(3),0.1,0.9);
else
    p1 = 0.8;
    p1 = p1 * ifv(e(2),0.05,0.95);
    p1 = p1 * ifv(e(3),0.05,0.95);
end
% D
if e(2)
    if e(3)
        p1 = p1 * ifv(e(4),0.75,0.25);
    else
        p1 = p1 * ifv(e(4),0.1,0.9);
    end
else
    if e(3)
        p1 = p1 * 0.5;
    else
        p1 = p1 * ifv(e(4),0.05,0.95);
    end
end
% E
if e(3)
    p1 = p1 * ifv(e(5),0.6,0.4);
else
```

```
    p1 = p1 * ifv(e(5),0.05,0.95);
end
probs(1,1) = p1;

% MODEL 2
% A,B
p2 = ifv(e(1),0.2,0.8);
p2 = p2 * ifv(e(2),0.2,0.8);
% C
if e(1)
    if e(2)
        p2 = p2 * ifv(e(3),0.25,0.75);
    else
        p2 = p2 * ifv(e(3),0.05,0.95);
    end
else
    if e(2)
        p2 = p2 * ifv(e(3),0.2,0.8);
    else
        p2 = p2 * ifv(e(3),0.05,0.95);
    end
end
% D,E
if e(3)
    p2 = p2 * ifv(e(4),0.6,0.4);
    p2 = p2 * ifv(e(5),0.6,0.4);
else
    p2 = p2 * ifv(e(4),0.1,0.9);
    p2 = p2 * ifv(e(5),0.05,0.95);
end
probs(2,1) = p2;

% MODEL 3
% A
p3 = ifv(e(1),0.2,0.8);
% B,C
if e(1)
    p3 = p3 * ifv(e(2),0.25,0.75);
    if e(2)
        p3 = p3 * ifv(e(3),0.25,0.75);
    else
```

```
        p3 = p3 * ifv(e(3),0.05,0.95);
    end
else
    p3 = p3 * ifv(e(2),0.05,0.95);
    if e(2)
        p3 = p3 * ifv(e(3),0.2,0.8);
    else
        p3 = p3 * ifv(e(3),0.05,0.95);
    end
end
% D,E
if e(3)
    p3 = p3 * ifv(e(5),0.6,0.4);
    if e(2)
        p3 = p3 * ifv(0.75,0.25);
    else
        p3 = p3 * 0.5;
    end
else
    p3 = p3 * ifv(e(5),0.05,0.95);
    if e(2)
        p3 = p3 * ifv(e(5),0.1,0.9);
    else
        p3 = p3 * ifv(e(5),0.05,0.95);
    end
end
probs(3,1) = p3;
```

ifv **function** This is a simple shorthand 'if' function for use in $probOfEvent$.

```
function v = ifv(t,wt,wf)
% IFV shorthand if statement
% when the condition t is true (or 1)
% returns the value wt, else it returns wf.

if t
    v = wt;
else
    v = wf;
end
```

6 Appendix 4: Likelihood calculation source

computePosteriorProbs **function** This function is used in question 3c to calculate the posterior probabilities on the three models $P(M_j|d_i)$ for $j \in \{1,2,3\}, d_i \in S_N$. It does this by proceeding through the data, summing the natural logarithms of the likelihood of each event. Then in order to calculate the posterior probabilities it computes

$$hyp = \sum_{j \in \{1,2,3\}} P(d_i|M_j)P(M_j) \tag{2}$$

and then for each $j \in 1,2,3$ computes $P(M_j|d_i) = P(d_i|M_j)P(M_j)/hyp$.

```
function results = computePosteriorProbs(sample, step, limit)

% size of sample
sz = size(sample);
if nargin == 2
    limit = sz(1);
end

% init results
results = [];

% prior probabilities of models
prior = [1/12; 5/12; 6/12];

% current log likelihoods
loglhood = [0; 0; 0];

% iterate through increasing sample sizes
for i = 1:limit
    % compute prob of current event
    p = probOfEvent(sample(i,1:sz(2)));

    % compute product of likelihoods so far
    loglhood = loglhood + log(p);

    % if should take a reading
    if mod(i,step) == 0
        % get current likelihoods
        lhoods = exp(loglhood);
```

```
        % sum over hypotheses
        hyp = sum(lhoods .* prior);

        % compute posteriors
        posts = (lhoods .* prior) ./ hyp;

        % add to results
        rsz = size(results);
        results(rsz(1)+1, 1:3) = posts';
    end
end
```

computeLikelihoodRatios **function** This function computes the likelihood ratios of the models by suming the natural logarithms of the probability of each event. It returns $\log(P(d_i|M_j)/P(d_i|M_k))$ because the function grows exponentially and quickly exceeds double floating point representation capacity.

```
function ratios = computeLikelihoodRatios(sample,step)
% COMPUTELIKELIHOODRATIOS computes the likelyhood ratios
% each of the models M in {M1,M2,M3} for subsets of the
% sample in steps of 'step' size and returns them as
% a matrix of rows where each row is like:
%
% row = [M1/M2 M1/M3 M2/M3];

ratios = [];
sz = size(sample);
t = [0; 0; 0];

for i = 1:sz(1)
    % compute P of current event occuring
    p = probOfEvent(sample(i,1:sz(2)));

    % multiply likelihoods by totals
    t = [t(1) + log(p(1));
         t(2) + log(p(2));
         t(3) + log(p(3))];

    % if step add to results
```

```
    if mod(i,step) == 0
        rsz = size(ratios);
        ratios(rsz(1)+1, 1:3) = [t(1) - t(2); t(1) - t(3); t(2) - t(3)];
    end
end
```