

1.1

$$p(x, y|z) = \frac{p(x, y, z)}{p(z)} = \frac{p(y|x, z)p(x, z)}{p(z)} = p(y|x, z)p(x|z) \quad (\text{B.0.1})$$

$$p(x|y, z) = \frac{p(x, y, z)}{p(y, z)} = \frac{p(y|x, z)p(x, z)}{p(y, z)} = \frac{p(y|x, z)p(x|z)}{p(y|z)} \quad (\text{B.0.2})$$

1.2

$$p(a \cup b) = p(a) + p(b) - p(a, b) \leq 1 \rightarrow p(a, b) \geq p(a) + p(b) - 1 \quad (\text{B.0.3})$$

1.3

$$p(\text{box} = 1|\text{redball}) = \frac{p(\text{box} = 1, \text{redball})}{p(\text{redball})} \quad (\text{B.0.4})$$

$$= \frac{p(\text{redball}|\text{box} = 1)p(\text{box} = 1)}{p(\text{redball}|\text{box} = 1)p(\text{box} = 1) + p(\text{redball}|\text{box} = 2)p(\text{box} = 2)} \quad (\text{B.0.5})$$

$$= \frac{3/8}{3/8 + 2/7} = \frac{21}{37} \quad (\text{B.0.6})$$

1.4 We want

$$p(2 \text{ red in box} | 3 \text{ red drawn}) = \frac{p(3 \text{ red drawn} | 2 \text{ red in box})p(2 \text{ red in box})}{p(3 \text{ red drawn})}$$

A priori, there are 3 possibilities for what balls can be in the box

$$a : 2 \text{ white balls in box}, \quad p(a) = 1/4 \quad (\text{B.0.7})$$

$$b : 2 \text{ red balls in box}, \quad p(b) = 1/4 \quad (\text{B.0.8})$$

$$c : 1 \text{ red 1 white ball in box}, \quad p(c) = 1/2 \quad (\text{B.0.9})$$

In the use of Bayes' rule above we have

$$p(3 \text{ red drawn} | 2 \text{ red in box})p(2 \text{ red in box}) = 1 \times 1/4 \quad (\text{B.0.10})$$

Also, using the shorthand for the above events:

$$p(3 \text{ red drawn}) = p(3 \text{ red drawn} | a)p(a) + p(3 \text{ red drawn} | b)p(b) + p(3 \text{ red drawn} | c)p(c) \quad (\text{B.0.11})$$

$$= 0 + 1 \times 1/4 + 0 + 1/8 \times 1/2 \quad (\text{B.0.12})$$

Hence the result is 4/5.

1.5 One simple approach (which is not quite correct) is as follows. Let's call 'Suspect' the unfortunate individual that is hauled off the plane. Using 'terr' as shorthand for 'Suspect is a terrorist' and 'scan' for 'scanner goes off for Suspect'.

$$p(\text{terr} = \text{tr} | \text{scan} = \text{tr}) = \frac{p(\text{scan} = \text{tr} | \text{terr} = \text{tr})p(\text{terr} = \text{tr})}{p(\text{scan} = \text{tr} | \text{terr} = \text{tr})p(\text{terr} = \text{tr}) + p(\text{scan} = \text{tr} | \text{terr} = \text{fa})p(\text{terr} = \text{fa})} \quad (\text{B.0.13})$$

$$= \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.05 \times 0.99} = 0.161 \quad (\text{B.0.14})$$

However, this calculation ignores the information given in the question that the Suspect is the *first* person to test positive. To answer the question correctly¹, we assume some ordering of the passengers, 1, ..., 100. We also define the notation:

¹Thanks to Alex Botev for this form of the solution.

- s_i - scanner indicates person i is the terrorist
- t_i - person i is the terrorist
- h - a passenger is hauled off the plane
- c - identification of the terrorist is correct

Using '1' as shorthand to represent 'true' and '0' to represent 'false', the question then provides the following information:

$$\begin{aligned} p(s_i = 1|t_i = 1) &= \gamma_1 = 0.95 \\ p(s_i = 0|t_i = 0) &= \gamma_2 = 0.95 \\ p(t_i = 1) &= n^{-1} \end{aligned} \tag{B.0.15}$$

where $n = 100$ in the above.

The question asks us to compute the probability that the hauled off passenger is the terrorist. This is $p(c = 1|h = 1)$. This means that the scanner first went off for some individual i , namely $s_i = 1$. If the scanner goes off for person i , then it must not have gone off for all individuals $1, \dots, i-1$. We're not told which passenger seat the individual hauled off the plane was in, hence:

$$\begin{aligned} p(c = 1|h = 1) &= \sum_{i=1}^n p(t_i = 1, s_i = 1|h = 1) \\ &= \sum_{i=1}^n \frac{\underbrace{p(h = 1|t_i = 1, s_i = 1)}_{=1} p(t_i = 1, s_i = 1)}{p(h = 1)} \\ &= \sum_{i=1}^n \frac{p(t_i = 1, s_i = 1, s_{1\dots i-1} = 0)}{p(h = 1)} \\ &= \sum_{i=1}^n \frac{p(s_i = 1, s_{1\dots i-1} = 0|t_i = 1)p(t_i = 1)}{p(h = 1)} \\ &= \sum_{i=1}^n \frac{p(t_i = 1)p(s_i = 1|t_i = 1) \prod_{j=1}^{i-1} p(s_j = 0|t_j = 0)}{p(h = 1)} \\ &= \sum_{i=1}^n \frac{\gamma_1 \gamma_2^{i-1}}{np(h = 1)} = \frac{\gamma_1}{np(h = 1)} \sum_{i=1}^n \gamma_2^{i-1} \\ &= \frac{\gamma_1}{np(h = 1)} \sum_{i=0}^{n-1} \gamma_2^i = \frac{\gamma_1}{np(h = 1)} \frac{1 - \gamma_2^n}{1 - \gamma_2} \\ &= \frac{1}{p(h = 1)} \frac{\gamma_1}{n} \frac{1 - \gamma_2^n}{1 - \gamma_2} \end{aligned} \tag{B.0.16}$$

The last thing we need to calculate is $p(h = 1)$. This can be calculated as:

$$\begin{aligned} p(h = 1) &= 1 - p(h = 0) = 1 - \sum_{i=1}^n p(h = 0, t_i = 1) \\ &= 1 - \sum_{i=1}^n p(h = 0|t_i = 1)p(t_i = 1) \\ &= 1 - \frac{1}{n} \sum_{i=1}^n p(s_{1\dots n} = 0|t_i = 1) \\ &= 1 - \frac{1}{n} \sum_{i=1}^n (1 - \gamma_1) \gamma_2^{n-1} \\ &= 1 - (1 - \gamma_1) \gamma_2^{n-1} \end{aligned} \tag{B.0.17}$$

Finally this means that we get:

$$p(c = 1|h = 1) = \frac{\gamma_1 (1 - \gamma_2^n)}{n(1 - \gamma_2)(1 - (1 - \gamma_1) \gamma_2^{n-1})} \tag{B.0.18}$$

Plugging in $n = 100$, $\gamma_1 = 0.95$ and $\gamma_2 = 0.95$ we get $p(c = 1|h = 1) = 0.18893$. This value is strikingly low – even though the scanner is 95% accurate, there is less than 20% probability that the first person identified by the scanner is actually the terrorist.

1.6 $2 + 2 + 1 = 5$. This is compared to the 7 parameters in a general three binary-variable distribution.

1.7 See `demoClouseau2.m`.

1.8

$$a \cap (b \cup c) = (a \cap b) \cup (a \cap c) \tag{B.0.19}$$

so

$$p(a \cap (b \cup c)) = p(a \cap b) + p(a \cap c) - p(a \cap b \cap a \cap c) = p(a, b) + p(a, c) - p(a, b, c) \tag{B.0.20}$$

1.9

$$p(x|z) = \sum_y p(x, y|z) = \sum_y \frac{p(x, y, z)}{p(z)} = \sum_y \frac{p(x|y, z)p(y, z)}{p(z)} = \sum_y p(x|y, z)p(y|z) \quad (\text{B.0.21})$$

Similar argument for second assertion, beginning with:

$$p(x|z) = \sum_{y, w} p(x, y, w|z) \quad (\text{B.0.22})$$

1.10 The issue here is one of interpretation. Essentially, what the confidence argument is saying is that if we were able to repeat this experiment of visiting Berlin at some point during its wall's lifetime, and use the procedure above to predict the end time, then 95% of the time our predictions will be correct. However, this is an odd thing to want. Intuitively, we are really interested in

$$p(t_{\text{end}}|t_{\text{now}}) = \frac{p(t_{\text{now}}|t_{\text{end}})p(t_{\text{end}})}{p(t_{\text{now}})} \quad (\text{B.0.23})$$

To manipulate this quantity, we must use a prior $p(t_{\text{end}})$. Perhaps a less passionate example is that you are given a uniformly generated random positive value from the interval $[1, M]$ with unknown maximum M . Your task is to estimate M based on this single observed value. For example, you observe the value 80. You then say, OK with 95% confidence I think M is between 82 and 3200. I then tell you the true answer. We then repeat this experiment, giving each time a possible range for M . In the limit of many repetitions, the true M each time will indeed lie in the predicted interval in 95% of the cases. However, this is really more a statement about the procedure than an interesting statement about M for any particular experiment. We really want to know something about the Berlin wall, not about the delta-t method. To answer that, we must use a prior.

See www.cs.ucl.ac.uk/staff/D.Barber/publications/tipping-barber-future.pdf for a more detailed discussion.

1.11 See `softxor.m`.

1.12 The point here is that if we wish to define a distribution $p(\text{ham}|KJ)$ for use with BRMLTOOLBOX, we are not explicitly given $p(\text{ham} = \text{tr}|KJ = \text{fa}) = \gamma$. However, we can figure out this probability using

$$\underbrace{p(\text{ham} = \text{tr})}_{\alpha} = \underbrace{p(\text{ham} = \text{tr}|KJ = \text{tr})}_{0.9} \underbrace{p(KJ = \text{tr})}_{\beta} + \underbrace{p(\text{ham} = \text{tr}|KJ = \text{fa})}_{\gamma} \underbrace{p(KJ = \text{fa})}_{1-\beta} \quad (\text{B.0.24})$$

where $\beta = 1/100000$, $\alpha = 1/2$. Solving this gives

$$\gamma = \frac{\alpha - 0.9\beta}{1 - \beta} \quad (\text{B.0.25})$$

which can then be used to define the table entries required for BRMLTOOLBOX. See `hamburger.m`.

1.13 See `twoDiceExample.m`.

1.14 There will be $1000,000/100 = 10000$ people each week that choose 3, 5, 7, 9. If they win, they will each take home $\pounds 1000,000/10000 = \pounds 100$. The probability that 3, 5, 7, 9 (or any four numbers) arises each week is $4!/(9 \times 8 \times 7 \times 6)$. The expected winnings for someone choosing 3, 5, 7, 9 on any given week is therefore $\pounds 0.7937$. Given that it costs $\pounds 1$ to enter each week, they will lose around 21 pence a week. On the other hand for the least popular number 1, 2, 3, 4, if this number comes up, each player will win $\pounds 10,000$, so that the average profit per week is $\pounds 79.37 - 1 = \pounds 78.37$. The lottery is purely random, but there is 'skill' involved in maximising the amount of money one wins (contrary to some government definitions of a lottery for which skill should play no part in the 'success').

1.15 Probability of getting 0, 1, 2, 3, 4, 5 correct matches is $44/120, 45/120, 20/120, 10/120, 0, 1/120$. The expected number of correct matches is $(45 + 2 \times 20 + 3 \times 10 + 5)/120 = 1$. Probability of at least 1 correct is 1 minus the probability of no correct matches $1 - 44/120 = 76/120$. The clever way to do this is using 'rencontres numbers'. The brute force enumeration approach is in `psychometry.m`.

- 1.17
1. A nice way to show this is to consider using 'dividers' that separate the 7 friends into their pizza choices. For example `oo|ooo|o|o` would say that two have chosen pizza A, three pizza B, one pizza C and one pizza D. We are interested in the number of such partitions. There are then $7 + 3 = 10$ positions, each position containing either a friend o or a divider |. We can place the dividers in $10 \times 9 \times 8 = 720$ positions. Since the dividers have themselves $3 \times 2 = 6$ arrangements, the number of partitions is $720/6 = 120$.
 2. Note that this is not $1/120$ – this would be the probability of the chef uniformly choosing one of the 120 partitions. However, he chooses a pizza at random, so that some partitions are more probable than others. For example, it is less likely that all friends chose pizza A, than two friends choosing pizza A, two friends pizza B, two friends pizza C and one friend pizza D. One needs therefore to compute these probabilities, $p_i, i = 1, \dots, 120$. Since the chef and friends chose independently, the probability of agreement is given by $\sum_i p_i^2 \approx 0.0184$. See `pizza.m`.

1.18 Assuming that $p(\text{Alice}) = p(\text{Bella}) = 0.5$, $\text{dom}(f) = \{\text{Alice}, \text{Bella}\}$, and $p(\text{sushi}, \text{car}|f) = p(\text{sushi}|f)p(\text{car}|f)$, then

$$p(\text{Alice}|\text{dislike sushi, white car}) = \frac{p(\text{dislike sushi, white car}|\text{Alice})p(\text{Alice})}{p(\text{dislike sushi, white car}|\text{Alice})p(\text{Alice}) + p(\text{dislike sushi, white car}|\text{Bella})p(\text{Bella})} \quad (\text{B.0.26})$$

$$= \frac{0.5 \times 0.9 \times 0.5}{0.5 \times 0.9 \times 0.5 + 0.1 \times 0.5 \times 0.5} = 0.9 \quad (\text{B.0.27})$$

1.19 1. The transition matrix $p(w_{t+1}|w_t)$ for $w \in \{\text{rain}, \text{sun}\}$ is

$$\begin{pmatrix} p(\text{rain}|\text{rain}) & p(\text{rain}|\text{sun}) \\ p(\text{sun}|\text{rain}) & p(\text{sun}|\text{sun}) \end{pmatrix} = \begin{pmatrix} 0.7 & 0.6 \\ 0.3 & 0.4 \end{pmatrix}$$

We just use then Bayes' rule to compute the probability.

$$p(w_t = \text{rain} | w_{t+1} = \text{sun}) = \frac{p(w_{t+1} = \text{sun} | w_t = \text{rain})p(w_t = \text{rain})}{p(w_{t+1} = \text{sun} | w_t = \text{rain})p(w_t = \text{rain}) + p(w_{t+1} = \text{sun} | w_t = \text{sun})p(w_t = \text{sun})} = 3/7$$

where we used $p(\text{rain}) = p(\text{sun}) = 0.5$

2. In this case, the stationary distribution can be found either by explicitly solving for the stationary distribution, or just multiplying the transition matrix. The stationary distribution satisfies

$$\begin{pmatrix} p(\text{rain}|\text{rain}) & p(\text{rain}|\text{sun}) \\ p(\text{sun}|\text{rain}) & p(\text{sun}|\text{sun}) \end{pmatrix} \begin{pmatrix} p(\text{rain}) \\ p(\text{sun}) \end{pmatrix} = \begin{pmatrix} p(\text{rain}) \\ p(\text{sun}) \end{pmatrix}$$

Using $p(\text{sun}) = 1 - p(\text{rain})$ gives the equation

$$0.7p(\text{rain}) + 0.6(1 - p(\text{rain})) = p(\text{rain})$$

This gives $p(\text{rain}) = 2/3, p(\text{sun}) = 1/3$.

3. Using $p(\text{rain}) = 2/3, p(\text{sun}) = 1/3$ in the previous calculation, we arrive at $p(w_t = \text{rain} | w_{t+1} = \text{sun}) = 3/5$.

1.20 See `solutionBattleships.m`. Answer is (5, 1) or (10, 6) with value 0.2069.

Let \mathcal{D} be the collection of 'hit' and 'miss' information. If we denote the origin of ship 1 by s_1 and ship 2 by s_2 , then

$$p(s_1, s_2 | \mathcal{D}) = \frac{p(\mathcal{D} | s_1, s_2)p(s_1, s_2)}{p(\mathcal{D})}$$

If X is then the matrix of pixel occupancy,

$$p(X | \mathcal{D}) = \sum_{s_1, s_2} p(X, s_1, s_2 | \mathcal{D}) = \sum_{s_1, s_2} p(X | s_1, s_2)p(s_1, s_2 | \mathcal{D})$$

import brml.*

% miss information in question:

```
data = [
    0    0    0    0    0    0    0    0    0    -1
    0   -1    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0   -1    0    0
    0    0    0   -1    0    0    0    0    0    0
    0    0    0    0    0   -1    0    0    0    0
    0    0    0    0   -1    0    0    0    0    0
    0    0    0   -1    0    0   -1    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0   -1    0    0    0    0    0    0   -1    0
    0    0    0    0    0    0    0    0    0    0]
```

```
ship(1).x=[0 1 2 3 4]; ship(1).y=[0 0 0 0 0];
ship(2).x=[0 0 0 0 0]; ship(2).y=[0 1 2 3 4];
```

```
D=10; S=D*D; M=reshape(1:S,D,D);
```

```
count=0;
post=zeros(S^2,1);
poss=zeros(D,D,2); prob=zeros(D,D); postocc=zeros(D,D);
for s1=1:S
    mask1=false(D,D);
    [x1,y1]=find(M==s1);
    onboard1=true;
    for i=1:length(ship(1).x)
        if ~validgridposition(x1+ship(1).x(i),y1+ship(1).y(i),D,D)
            onboard1=false;
        else
            mask1(x1+ship(1).x(i),y1+ship(1).y(i))=true;
        end
    end
end
```

```
onboard2=true;
if onboard1
    for s2=1:S
        onboard2=true;
        mask2=false(D,D);
        [x2,y2]=find(M==s2);
```

```

count=count+1;

for i=1:length(ship(2).x)
    if ~validgridposition(x2+ship(2).x(i),y2+ship(2).y(i),D,D)
        onboard2=false;
    else
        mask2(x2+ship(2).x(i),y2+ship(2).y(i))=true;
    end
end

if onboard1 && onboard2
    if ~any( mask1(:) & mask2(:)) % no clashing ships
        mask=mask1+mask2;
        if ~( any(mask(:)==1 & data(:)==-1) || any(data(:)==1 & mask(:)==0)) % compatible with data
            post(count)=1;
            postocc=postocc+mask;
        end
    end
end
end
end
end

postocc=postocc/sum(post(:));
[val ind]=max(postocc(:))
[i,j]=find(postocc==val)

```

postocc=postocc/sum(post(:));
[val ind]=max(postocc(:))
[i,j]=find(postocc==val)

1.21 See `demoBattleshipsBig.m`. Answer is cell (4, 5) (or) (5, 4) with probability 0.5518.

2.1 Let n_{ij}^k be the number of paths from j to i in k steps. Then $n_{ij}^k = \sum_l A_{il} n_{lj}^{k-1}$ since the number of paths must be the number of paths that get one to an intermediate node l in $k-1$ steps, provided one can get from l to i in an additional step. In matrix notation this is $N^{(k)} = AN^{(k-1)} = A^2 N^{(k-2)} = A^k$.

2.2 From the previous exercise, we know that $[A^k]_{ij}$ is the number of paths from j to i in k steps. If a subset of variables is connected, there must be a path of some length, less than or equal to n (the number of nodes in the graph) between them. Defining $B_{ij} = 1$ if $\sum_k [A^k]_{ij} > 0$, and zero otherwise, B_{ij} is 1 if i and j are connected by some path. We then look at column i (or equivalently a row) of B , which describes the nodes that can be reached from node i , and put these in a connected component. Since connected components are disjoint, we can then remove these variables from consideration, and continue to another column to find another set of connected variables.

2.3 $1-2-3-1$ and $4-5$ is multiply connected and satisfies $E = V - 1$.

2.4 If the graph is connected and the number of edges is less than the number of nodes, it must be a tree. However, to deal with the general case in which it is unknown if the graph is connected we check using elimination. A tree/singly-connected graph must admit a recursive simplicial node elimination. That is at any stage in the elimination there must be a node with either zero or 1 neighbours in the remaining graph. See `istree.m`.

```

2.5 done=false;
a=parents(A,x); % start with the parents of x
while ~done
    aold=a;
    a=union(a,parents(A,a)); % include the parents of the current ancestors
    done isempty(setdiff(a,aold)); % done if this doesn't introduce any more nodes
end
a=setdiff(a,x);
See ancestors.m.

```

2.6 If we interpret i 'knowing' j as the same as j knowing i , we need to first symmetrize the matrix \mathbf{A} . Then if $[\mathbf{A}^k]_{ij}$ is non-zero then there is a length k path between i and j . However, in order to find the separation, we need to make sure that we only count the lowest length path between i and j . This means that the separation is k , provided that no shorter length path already exists. Doing so, we find that the number of unique pairs with separations from 1 to 7 are

1898, 19988, 55258, 46679, 13709, 1781, 96

with no longer separations. See `makeWikiGraph.m`.

Remarks

1. There is some ambiguity in the question and the answer will depend on whether you interpret that i 'knows' j , implies that j 'knows' i . (That is you make the matrix A symmetric by using $A = A + A'$)
2. A common error is to simply look for a length k path between two users i and j . This is not what is intended by 'separation'. What we want is the length of the *shortest* path that connects i and j .

2.7 There are many different ways to do this based on finding which cliques are wholly contained in another. The solution is

119, 447, 463, 474, 487, 703, 751, 755, 765, 831, 863, 886, 893, 954, 983, 1006, 1013

The main routine for finding the unique cliques is

```

uniqueflag=true(1,length(c1));
for i=1:length(c1)
    for j=find(uniqueflag)
        if i~=j
            if isempty(setdiff(c1{i},intersect(c1{i},c1{j})));
                uniqueflag(i)=false;
                break;
            end
        end
    end
end
end
end

```

This first assumes all cliques are unique and then looks at clique i . The code checks all other remaining unique cliques to see if clique i is a subset of one of them. If so, clique i is removed from the list of unique cliques, and the search continues. See `cliquesproblem.m`.

Remarks

Note that there are some cliques that are exactly identical. A common error is to remove both cliques when they are identical, leaving only 15 cliques, rather than the 17 unique cliques.

2.8 We construct a bipartite graph, placing half of the nodes in the upper part and half in the lower. We then connect the both parts together.

2.9 In this case every maximal clique sizes must be $N/3$ since if we were to include another node, this means that there must be one of the $N/3$ partitions that contains two nodes in the proposed clique – however this cannot be the case since the lack of connections within a partition prohibits such a clique formation. Each clique can then be formed by choosing one of the 3 nodes from each of the $N/3$ partitions, giving $3^{N/3}$ maximal cliques.

Remarks

This result is perhaps at first sight surprising and helps explain why finding maximal cliques or enumerating all the cliques in a graph is in general a hard problem.

2.10 The best way to think about this is to assume that each person enters the room (once) and stays for some time before leaving. We can then draw horizontal lines corresponding to the times that each person was in the room. Where the lines of two individuals overlap, they were both present in the room during that time period. The first thing to note is that for any 4 people that are telling the truth, if the graph corresponding to their connections has a 4 cycle, it must have a chord. To see this, consider that A was present with B , B was present with C , C was present with D and D was present with A . A entered the room at time a_1 and exited at a_2 , and similarly for the others. The above four statements say

$$\begin{aligned}
 a_2 &> b_1, & b_2 &> a_1 \\
 b_2 &> c_1, & c_2 &> b_1 \\
 c_2 &> d_1, & d_2 &> c_1 \\
 d_2 &> a_1, & a_2 &> d_1
 \end{aligned}$$

The graph corresponds to the 4 cycle $A - B - C - D - A$. If we assume that there is no chord between A and C , this means that $a_1 > c_2$ or $c_1 > a_2$. Under the assumption $a_1 > c_2$ then

$$d_2 > a_1 > c_2 > b_1, \quad b_2 > a_1 > c_2 > d_1$$

which says that B and D are connected. By symmetry, if we had assumed that $c_1 > a_2$, we would have arrived at the same conclusion. Similarly, if we had assumed no chord between B and D , then we would have inferred a chord between A and C . If we now take the set-up in the question, we find that only the statement of D introduces a cycle that does not have a chord – hence this is the incorrect statement.

3.1 The answer is 0.609660. See `partyanimal.m`.

3.2 (i) a and b are independent. (ii) a and b are dependent given c .

3.3 `tuberculosis \perp smoking | shortness of breath = fa; lung cancer \perp bronchitis | smoking = tr; visit to Asia \perp smoking | lung cancer = tr; visit to Asia \perp smoking | lung cancer, shortness of breath = fa.` One can verify this using `demoChestClinic; A=dag(pot); [condindep(A,xray,smoker,dys) condindep(A,lcancer,bronch,smoker) condindep(A,asia,smoker,lcancer) condindep(A,asia,smoker,[lcancer dys])]`

1. $tuberculosis \perp smoking | shortness of breath$. Paths connecting t to s are:

(i) $t \rightarrow e \leftarrow l \leftarrow s$. Path is not blocked since node e is a collider and its descendant, node d , is in the conditioned set. The non-collider nodes on this path are not being conditioned on.

(ii) $t \rightarrow e \rightarrow d \leftarrow b \leftarrow s$. Path is not blocked for the same reasons. So $tuberculosis \perp smoking | shortness of breath$ is false.

2. $lung cancer \perp bronchitis | smoking$. Paths connecting l to b are:

(i) $l \leftarrow s \rightarrow b$. Path is blocked. Node s is not a collider and is in the conditioning set.
(ii) $l \rightarrow e \rightarrow d \leftarrow b$. Path is blocked. Node d is a collider and is not in the conditioning set.
Every path from l to b is blocked given s so $lung cancer \perp bronchitis | smoking$ is true.

3. $visit to Asia \perp smoking | lung cancer$. Paths connecting a to s are:

(i) $a \rightarrow t \rightarrow e \leftarrow l \leftarrow s$. Path is blocked by l - node is not a collider and is in conditioning set.
(ii) $a \rightarrow t \rightarrow e \rightarrow d \leftarrow b \leftarrow s$. Path is blocked by node d - collider and not in conditioning set.
 $visit to Asia \perp smoking | lung cancer$ true since all paths are blocked.

4. *visit to Asia* \perp *smoking* | *lung cancer*, *shortness of breath*. Paths connecting a to s are:

(i) $a \rightarrow t \rightarrow e \leftarrow l \leftarrow s$. Again, path is blocked by l - node is not a collider and is in conditioning set.

(ii) $a \rightarrow t \rightarrow e \rightarrow d \leftarrow b \leftarrow s$. Path is not blocked by d since it is collider and is in the conditioning set - all other nodes on this path are not colliders and are not in conditioning set.

visit to Asia \perp *smoking* | *lung cancer*, *shortness of breath* is false - since there exists unblocked paths from node a to node s .

3.4 The marginal $p(d)$ can be calculated as

$$\begin{aligned} p(d) &= \sum_{a,s,t,l,b,e,x} p(a, s, t, l, b, e, x, d) \\ &= \sum_{a,s,t,l,b,e,x} p(a)p(s)p(t|a)p(l|s)p(b|s)p(e|t,l)p(x|e)p(d|b,e) \\ &= \sum_{a,s,t,l,b,e} \left(p(a)p(s)p(t|a)p(l|s)p(b|s)p(e|t,l)p(d|b,e) \sum_x p(x|e) \right) \\ &= \sum_{s,t,l,b,e} \left(p(s)p(l|s)p(b|s)p(e|t,l)p(d|b,e) \sum_a p(t|a)p(a) \right), \end{aligned}$$

where we have noted that $\sum_x p(x|e) = 1$. The final term on the RHS is just the marginal $p(t)$, which is

$$\begin{aligned} p(t = \text{tr}) &= p(t = \text{tr} | a = \text{tr}) \times p(a = \text{tr}) + p(t = \text{tr} | a = \text{fa}) \times p(a = \text{fa}) \\ &= 0.05 \times 0.01 + 0.01 \times 0.99 = 0.01 \times 1.04 = 0.0104. \end{aligned}$$

Armed with this, we can further simplify the expression for $p(d)$ to

$$p(d) = \sum_{s,l,b,e} \left(p(s)p(l|s)p(b|s)p(d|b,e) \sum_t p(e|t,l)p(t) \right).$$

The last term on the RHS is now $p(e|l)$, which is

$$\begin{aligned} p(e = \text{tr} | l = \text{tr}) &= p(e = \text{tr} | l = \text{tr}, t = \text{tr}) \times p(t = \text{tr}) + p(e = \text{tr} | l = \text{tr}, t = \text{fa}) \times p(t = \text{fa}) \\ &= 1 \times 0.0104 + 1 \times 0.9896 = 1, \\ p(e = \text{tr} | l = \text{fa}) &= p(e = \text{tr} | l = \text{fa}, t = \text{tr}) \times p(t = \text{tr}) + p(e = \text{tr} | l = \text{fa}, t = \text{fa}) \times p(t = \text{fa}) \\ &= 1 \times 0.0104 + 0 \times 0.9896 = 0.0104. \end{aligned}$$

The marginal $p(d)$ is now

$$p(d) = \sum_{s,b,e} p(s)p(b|s)p(d|b,e) \sum_l p(e|l)p(l|s),$$

which we can further simplify by calculating $p(e|s)$, which is

$$\begin{aligned} p(e = \text{tr} | s = \text{tr}) &= p(e = \text{tr} | l = \text{tr}) \times p(l = \text{tr} | s = \text{tr}) + p(e = \text{tr} | l = \text{fa}) \times p(l = \text{fa} | s = \text{tr}) \\ &= 1 \times 0.1 + 0.0104 \times 0.9 = 0.10936, \\ p(e = \text{tr} | s = \text{fa}) &= p(e = \text{tr} | l = \text{tr}) \times p(l = \text{tr} | s = \text{fa}) + p(e = \text{tr} | l = \text{fa}) \times p(l = \text{fa} | s = \text{fa}) \\ &= 1 \times 0.01 + 0.0104 \times 0.99 = 0.020296. \end{aligned}$$

This now gives us

$$p(d) = \sum_{s,b} p(b|s)p(s) \sum_e p(d|b,e)p(e|s),$$

leading us to calculate $p(d|b,s)$, which is

$$\begin{aligned} p(d = \text{tr} | b = \text{tr}, s = \text{tr}) &= p(d = \text{tr} | b = \text{tr}, e = \text{tr}) \times p(e = \text{tr} | s = \text{tr}) + p(d = \text{tr} | b = \text{tr}, e = \text{fa}) \times p(e = \text{fa} | s = \text{tr}) \\ &= 0.9 \times 0.10936 + 0.8 \times 0.89064 = 0.8109360, \\ p(d = \text{tr} | b = \text{tr}, s = \text{fa}) &= p(d = \text{tr} | b = \text{tr}, e = \text{tr}) \times p(e = \text{tr} | s = \text{fa}) + p(d = \text{tr} | b = \text{tr}, e = \text{fa}) \times p(e = \text{fa} | s = \text{fa}) \\ &= 0.9 \times 0.020296 + 0.8 \times 0.979704 = 0.8020296, \\ p(d = \text{tr} | b = \text{fa}, s = \text{tr}) &= p(d = \text{tr} | b = \text{fa}, e = \text{tr}) \times p(e = \text{tr} | s = \text{tr}) + p(d = \text{tr} | b = \text{fa}, e = \text{fa}) \times p(e = \text{fa} | s = \text{tr}) \\ &= 0.7 \times 0.10936 + 0.1 \times 0.89064 = 0.1656160, \\ p(d = \text{tr} | b = \text{fa}, s = \text{fa}) &= p(d = \text{tr} | b = \text{fa}, e = \text{tr}) \times p(e = \text{tr} | s = \text{fa}) + p(d = \text{tr} | b = \text{fa}, e = \text{fa}) \times p(e = \text{fa} | s = \text{fa}) \\ &= 0.7 \times 0.020296 + 0.1 \times 0.979704 = 0.1121776 \end{aligned}$$

We now have

$$p(d) = \sum_s p(s) \sum_b p(d|b,s)p(b|s),$$

so we calculate $p(d|s)$, which is

$$\begin{aligned} p(d = \text{tr}|s = \text{tr}) &= p(d = \text{tr}|b = \text{tr}, s = \text{tr}) \times p(b = \text{tr}|s = \text{tr}) + p(d = \text{tr}|b = \text{fa}, s = \text{tr}) \times p(b = \text{fa}|s = \text{tr}) \\ &= 0.8109360 \times 0.6 + 0.1656160 \times 0.4 = 0.5528080, \\ p(d = \text{tr}|s = \text{fa}) &= p(d = \text{tr}|b = \text{tr}, s = \text{fa}) \times p(b = \text{tr}|s = \text{fa}) + p(d = \text{tr}|b = \text{fa}, s = \text{fa}) \times p(b = \text{fa}|s = \text{fa}) \\ &= 0.8020296 \times 0.3 + 0.1121776 \times 0.7 = 0.3191332. \end{aligned}$$

Now, we can calculate $p(d) = \sum_s p(d|s)p(s)$, which is

$$\begin{aligned} p(d = \text{tr}) &= p(d = \text{tr}|s = \text{tr}) \times p(s = \text{tr}) + p(d = \text{tr}|s = \text{fa}) \times p(s = \text{fa}) \\ &= 0.5528080 \times 0.5 + 0.3191332 \times 0.5 = 0.4359706 \end{aligned}$$

Thus we have $p(d = \text{tr}) = 0.4359706$, $p(d = \text{tr}|s = \text{tr}) = 0.5528080$ and $p(d = \text{tr}|s = \text{fa}) = 0.3191332$.

3.5 $p(d = \text{true}|b = \text{false})$ could be inferred by removing the b node from the graph and setting $b = \text{false}$ in node b 's children (*i.e.* node d). This result would differ from $p(d = \text{true}|b = \text{false})$ since this inference calculation would include the factor $p(b = \text{false}|s)$ whereas the causal calculation would not.

3.6 $p(\text{fuel} = \text{empty}|\text{start} = \text{false}) \approx 0.4537$. See `carstart.m`.

3.7 1. A and S are independent : $p(A, S) = p(A)p(S)$.

2. Include extra variable B : $p(C|A, S)p(A|B)p(S|B)P(B)$

3.8 All results are in `demoHolmes.m`. The first part is straightforward.

$$p(B = \text{tr}|W = \text{tr}) = \frac{\sum_{A, G} p(B = \text{tr}, A, G, W = \text{tr})}{\sum_{A, B, G} p(B, A, G, W = \text{tr})} \quad (\text{B.0.28})$$

$$\begin{aligned} &= \frac{p(B = \text{tr}) \sum_A p(A|B = \text{tr}) p(W = \text{tr}|A) \sum_G p(G|A)}{p(B = \text{tr}) \sum_A p(A|B = \text{tr}) p(W = \text{tr}|A) + p(B = \text{fa}) \sum_A p(A|B = \text{fa}) p(W = \text{tr}|A)} \quad (\text{B.0.29}) \\ &= \frac{0.01 \times (0.99 \times 0.9 + 0.01 \times 0.5)}{0.01 \times (0.99 \times 0.9 + 0.01 \times 0.5) + 0.99 \times (0.05 \times 0.9 + 0.95 \times 0.5)} \quad (\text{B.0.30}) \\ &= 0.0171 \quad (\text{B.0.31}) \end{aligned}$$

$$\begin{aligned} p(B = \text{tr}|W = \text{tr}, G = \text{fa}) &= \frac{\sum_A p(B = \text{tr}, A, G = \text{fa}, W = \text{tr})}{\sum_{A, B} p(B, A, G = \text{fa}, W = \text{tr})} \\ &= \frac{p(B = \text{tr}) \sum_A p(A|B = \text{tr}) p(W = \text{tr}|A) p(G = \text{fa}|A)}{p(B = \text{tr}) \sum_A p(A|B = \text{tr}) p(W = \text{tr}|A) p(G = \text{fa}|A) + p(B = \text{fa}) \sum_A p(A|B = \text{fa}) p(W = \text{tr}|A) p(G = \text{fa}|A)} \\ &= \frac{0.01 \times (0.99 \times 0.9 \times 0.3 + 0.01 \times 0.5 \times 0.3)}{0.01 \times (0.99 \times 0.9 \times 0.3 + 0.01 \times 0.5 \times 0.3) + 0.99 \times (0.95 \times 0.9 \times 0.3 + 0.95 \times 0.5 \times 0.8)} \\ &= 0.0069 \end{aligned}$$

The second part can be solved by computing $p(B|W)$;

$$p(B = \text{tr}|\tilde{W}) = \sum_W p(B = \text{tr}|W) p(W|\tilde{W}) \quad (\text{B.0.32})$$

So we need to calculate $p(B = \text{tr}|W = \text{fa})$

$$\begin{aligned} p(B = \text{tr}|W = \text{fa}) &= \frac{p(B = \text{tr}) \sum_A p(A|B = \text{tr}) p(W = \text{fa}|A)}{p(B = \text{tr}) \sum_A p(A|B = \text{tr}) p(W = \text{fa}|A) + p(B = \text{fa}) \sum_A p(A|B = \text{fa}) p(W = \text{fa}|A)} \\ &= \frac{0.01 \times (0.99 \times 0.1 + 0.01 \times 0.5)}{0.01 \times (0.99 \times 0.1 + 0.01 \times 0.5) + 0.99 \times (0.05 \times 0.1 + 0.95 \times 0.5)} \\ &= 0.0022 \end{aligned}$$

So,

$$p(B = \text{tr}|\tilde{W}) = 0.0171 \times 0.3 + 0.0022 \times 0.7 \quad (\text{B.0.33})$$

$$= 0.0068 \quad (\text{B.0.34})$$

Assuming \tilde{W} and \tilde{G} are independent then:

$$p(B = \text{tr}|\tilde{W}, \tilde{G}) = \sum_{W, G} p(B = \text{tr}|W, G) p(W|\tilde{W}) p(G|\tilde{G})$$

$$\begin{aligned}
p(B = tr|W = tr, G = tr) &= \frac{p(B = tr, W = tr, G = tr)}{p(W = tr, G = tr)} = \frac{\sum_A p(A, B = tr, W = tr, G = tr)}{\sum_{A,B} p(A, B, W = tr, G = tr)} = 0.0472 \\
p(B = tr|W = fa, G = fa) &= \frac{p(B = tr, W = fa, G = fa)}{p(W = fa, G = fa)} = \frac{\sum_A p(A, B = tr, W = fa, G = fa)}{\sum_{A,B} p(A, B, W = fa, G = fa)} = 0.00089 \\
p(B = tr|W = fa, G = tr) &= \frac{p(B = tr, W = fa, G = tr)}{p(W = fa, G = tr)} = \frac{\sum_A p(A, B = tr, W = fa, G = tr)}{\sum_{A,B} p(A, B, W = fa, G = tr)} = 0.0072
\end{aligned}$$

Hence

$$p(B = tr|\tilde{W}, \tilde{G}) = 0.0069 \times 0.3 \times 0.9 + 0.0472 \times 0.1 \times 0.1 + 0.0089 \times 0.7 \times 0.9 + 0.0072 \times 0.7 \times 0.1 = 0.0043$$

3.9 (1) $a \rightarrow (r, d); g \rightarrow (d, r); d \rightarrow r$. (2) This can be computed using standard inference. (3). In this case we need to remove the term $p(d|a, g)$ from the distribution and clamp d to *drug* and g to state *young*.

$$p(A, G, D, R) = p(R|A, D, G)p(D|A, G)p(A)p(G) \quad (\text{B.0.35})$$

$$p(R = tr|D = tr) = \frac{\sum_{A,G} p(A, G, D = tr, R = tr)}{\sum_{A,G,R} p(A, G, D = tr, R)} \quad (\text{B.0.36})$$

To calculate $p(R = tr|do(drug), A = young)$ we need to remove the D node from the graph and set its children's potentials to $Drug = true$ and then carry out standard inference. So,

$$p(R = tr|do(drug), A = young) = \frac{\sum_G p(R = tr|A = young, D = tr, G)p(A = young)p(G)}{\sum_{G,R} p(R|A = young, D = tr, G)p(A = young)p(G)} \quad (\text{B.0.37})$$

$$= \sum_G p(R = tr|A = young, D = tr, G)p(G) \quad (\text{B.0.38})$$

3.10 See `wetgrass.m`.

3.11 Need to include a link from B to E and replace $p(B)$ with the factor $p(B|E)$ in the distribution. $p(B|E)$ is suitably defined so that $p(B = tr|E = tr)$ is higher than $p(B = tr)$.

3.12 See `MarkovEquiv.m` and `immoralities.m`. Finding the skeleton is easy since we just need to remove the arrows by symmetrising the adjacency matrix. To find the immoralities, we can make a new adjacency matrix using

```

function IM=immoralities(A)
%IMMORALITIES find the immoralities of a DAG
% IM=immoralities(A)
% immoralities are returned in the matrix IM
% If there is a child C with unconnected parents A,B, the IM(A,B)=IM(B,A)=1
% and IM(A,C)=IM(B,C)=1;
% That is we link the unmarried parents and connect them to the child
N=size(A,1); IM=zeros(N,N);
A=A>0; % convert to logical (faster)
B=~(A|A'); % B is the matrix of non-connected nodes (either direction)
B=triu(B,1); % ignore lack of self-connections and only need to non-connectivity in one direction
for i=1:N
    pa=brml.parents(A,i); % get the parents of node i
    [ind1 ind2]=find(B(pa,pa)); % find those parents that are not connected
    for j=1:length(ind1)
        IM(pa(ind1(j)),pa(ind2(j)))=1; % connect parents
        IM(pa(ind2(j)),pa(ind1(j)))=1; % .. both directions
        IM(pa(ind1(j)),i)=1; % connect a parent to the child
        IM(pa(ind2(j)),i)=1; % and the other parent to child
    end
end
end

```

3.13 The two graphs are Markov equivalent. See `MarkovEquivExample.m`. The skeletons are clearly the same and both have the same parent-parent-child immoralities, namely $1 - 2 - 3$, $1 - 5 - 3$, $4 - 7 - 9$.

3.14 The only possibilities consistent with \mathcal{C} are

C_{13}	C_{23}	C_{21}	C_{12}	C_{32}	probability
0	0	0	1	0	$p(1-p)^4$
0	0	0	1	1	$p^2(1-p)^3$
0	0	1	1	0	$p^2(1-p)^3$
0	0	1	1	1	$p^3(1-p)^2$
1	0	0	1	0	$p^2(1-p)^3$
1	0	0	1	1	$p^3(1-p)^2$
1	0	0	0	1	$p^2(1-p)^3$

where $p = p(C_{ij} = 1)$, $i \neq j$. Then

$$\begin{aligned}
 p(C_{12} = 1|C) &= \frac{p(C_{12} = 1, C)}{p(C)} = \frac{p(1-p)^4 + 3p^2(1-p)^3 + 2p^3(1-p)^2}{p(1-p)^4 + 4p^2(1-p)^3 + 2p^3(1-p)^2} = \frac{110}{119} \approx 0.9244 \\
 p(C_{13} = 1|C) &= \frac{p(C_{13} = 1, C)}{p(C)} = \frac{19}{119} \approx 0.1597 \\
 p(C_{23} = 1|C) &= \frac{p(C_{23} = 1, C)}{p(C)} = 0 \\
 p(C_{32} = 1|C) &= \frac{p(C_{32} = 1, C)}{p(C)} = \frac{20}{119} \approx 0.1681 \\
 p(C_{21} = 1|C) &= \frac{p(C_{21} = 1, C)}{p(C)} = \frac{10}{119} \approx 0.0840 \\
 p(C_{31} = 1|C) &= \frac{p(C_{31} = 1, C)}{p(C)} = p(C_{31} = 1) = 0.1
 \end{aligned}$$

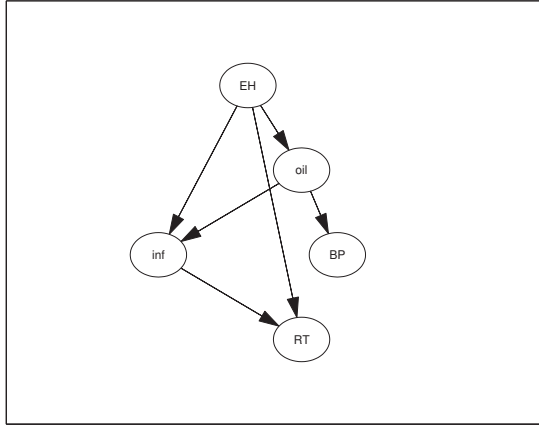
See also `ConnectedComputerProblem.m` which suggests how this may be extended to more general situations numerically.

3.15 See `EconHealth.m`.

```

inf =high 0.984746
inf =low 0.015254

```



3.16 See `uniquepots.m`. The basic idea is to form a tree of the form $A_{ji} = 1$ if the variables of potential i are a subset of the variables of potential j . The tree is formed such that each potential has at most a single parent. Given this tree we can identify the connected components which will form the new unique potentials. For each connected component we first instantiate each non-maximal clique with its potential. We then find the parent of each node (starting from the leaves) and multiply the parent potential with the node potential. We retain then only the potentials of the root leaves of the connected components.

```

function [newpot A] = uniquepots(inpots,varargin)
%UNIQUEPOTS Eliminate redundant potentials (those contained wholly within another) by multiplying redundant potentials
% [newpot A]= uniquepots(pot,<tables>)
% if tables=0 then just merge the variables
% The matrix has A(j,i)=1 if pot(i) is contained within pot(j)
import brml.*
inpots=str2cell(inpots); % conver to cell array if needed
tables=1; if nargin==2; tables=varargin{1}; end
% Find a tree of cliques by identifying a single parent clique j that contains clique i.
C=length(inpots); r=zeros(1,C);
for i=1:C; r(i)=isempty(inpots{i}.variables); end
pot=inpots(~r);C=length(pot);
A=sparse(C,C);
for i=1:C
    j=1;
    while j<=C
        A(j,i)=ischildpot(pot,i,j);
        if A(j,i);break;end
        j=j+1;
    end
end
newpot=pot;
% Now merge from bottom up:
[tree elimset sched]=istree(A); % returns an elimination schedule for all components
remove=zeros(1,C);
if tables
    for s=1:size(sched,1)
        ch=sched(s,1); pa=sched(s,2);
        if ch~=pa
            newpot{pa}=multpots(newpot([pa ch]));

```

```

        remove(ch)=1;
    end
end
else
    for s=1:size(sched,1)
        ch=sched(s,2); pa=sched(s,1);
        if ch~=pa
            newpot{pa}.variables=union(newpot{ch}.variables,newpot{pa}.variables);
            remove(ch)=1;
        end
    end
end
newpot=newpot(remove==0);

function m = ischildpot(pot,i,j)
if i==j; m=0; return; end
m=all(ismember(pot{i}.variables,pot{j}.variables));

```

3.17 See ABCproblem.m.

1. We need to show that $p(a, c) = p(a)p(c)$:

```

[a b c]=assign([1 2 3]);
pA=array(a,[3/5 2/5]);
pBgA=array([b a],[1/4 15/40; 1/12 1/8; 2/3 1/2]);
pCgB=array([c b],[1/3 1/2 15/40; 2/3 1/2 5/8]);
pABC=pA*pBgA*pCgB;

```

```

pAC=sum(pABC,b);
pApC = sum(pAC,a)*sum(pAC,c);

```

This gives

$p(a, c)$:

```

ans =

    0.2250    0.3750
    0.1500    0.2500

```

$p(a)p(c)$:

```

ans =

    0.2250    0.3750
    0.1500    0.2500

```

showing that the distributions are the same and hence that $a \perp\!\!\!\perp c$.

- 2.

$$p(a = i, c = k) = \sum_j p(a = i, b = j, c = k) = \frac{1}{Z} \sum_j \phi(a = i, b = j) \psi(b = j, c = k) = \frac{1}{Z} \sum_j M_{ij} N_{kj} = \frac{1}{Z} [\mathbf{M}\mathbf{N}^T]_{ik}$$

3. If $\mathbf{M}\mathbf{N}^T = \mathbf{m}_0 \mathbf{m}_0^T$ then

$$p(a = i, c = k) = \frac{1}{Z} [\mathbf{M}\mathbf{N}^T]_{ik} = \frac{1}{Z} [\mathbf{m}_0 \mathbf{m}_0^T]_{ik} = \frac{1}{Z} m_{0,i} m_{0,k}$$

Hence $p(a, c)$ factorises into a function of a times a function of c , implying therefore that a and c are independent.

- 4.

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{pmatrix} \begin{pmatrix} n_{11} & n_{21} \\ n_{12} & n_{22} \\ n_{13} & n_{23} \end{pmatrix} = \begin{pmatrix} m_{11}n_{11} + m_{12}n_{12} + m_{13}n_{13} & m_{11}n_{21} + m_{12}n_{22} + m_{13}n_{23} \\ m_{21}n_{11} + m_{22}n_{12} + m_{23}n_{13} & m_{21}n_{21} + m_{22}n_{22} + m_{23}n_{23} \end{pmatrix}$$

Since

$$\mathbf{m}_1 \mathbf{n}_1^T = \begin{pmatrix} m_{11}n_{11} & m_{11}n_{21} \\ m_{21}n_{11} & m_{21}n_{21} \end{pmatrix}$$

and similarly for the other cases $\mathbf{m}_2 \mathbf{n}_2^T$, $\mathbf{m}_3 \mathbf{n}_3^T$, the result follows.

- 5.

$$\begin{aligned} \mathbf{M}\mathbf{N}^T &= \mathbf{m}_1 \mathbf{n}_1^T + \lambda \mathbf{m}_1 \mathbf{n}_2^T + \gamma \mathbf{m}_3 (\mathbf{n}_1 + \lambda \mathbf{n}_2)^T \\ &= (\mathbf{m}_1 + \gamma \mathbf{m}_3) \mathbf{n}_1^T + \lambda (\mathbf{m}_3 + \gamma \mathbf{m}_1) \mathbf{n}_2^T = (\mathbf{m}_1 + \gamma \mathbf{m}_3) (\mathbf{n}_1 + \lambda \mathbf{n}_2)^T \end{aligned}$$

6. We now know how to construct the joint distribution $p(a, b, c)$ up to some unknown normalisation constant Z . We simply normalise the table to fill in this missing normalisation constant. The routine `condpot.m` also returns a correctly normalised distribution.

```
M(:,1)=rand(2,1);
lambda=rand;
M(:,2)=lambda*M(:,1);
M(:,3)=rand(2,1);
N(:,1)=rand(2,1);
N(:,2)=rand(2,1);

gamma=rand;
N(:,3)=gamma*(N(:,1)+lambda*N(:,2));
[a b c]=assign([1 2 3]);

potAB=array([a b],M);
potBC=array([b c],N');
potABC=condpot(potAB*potBC); % normalise

disp('p(a):'); table(condpot(sum(potABC,[b c])))
disp('p(b|a):'); table(condpot(potABC,b,a))
disp('p(c|b):'); table(condpot(potABC,c,b))

disp('check if a and c are independent:')
potAC=sumpot(potABC,b);
potApotC = sum(potAC,a)*sum(potAC,c);
disp('p(a,c):');table(potAC)
disp('p(a)p(c):');table(potApotC)
```

This produces for example

`p(a):`

```
ans =
    0.6876
    0.3124
```

`p(b|a):`

```
ans =
    0.6193    0.7498
    0.0933    0.1130
    0.2873    0.1372
```

`p(c|b):`

```
ans =
    0.1392    0.4014    0.1736
    0.8608    0.5986    0.8264
```

check if a and c are independent:

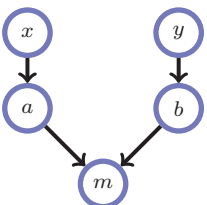
`p(a,c):`

```
ans =
    0.1194    0.5683
    0.0542    0.2582
```

`p(a)p(c):`

```
ans =
    0.1194    0.5683
    0.0542    0.2582
```

- 3.18 $A \leftarrow T \rightarrow B$. A and B are dependent since when we sum over T , we introduce a link between A and B .



- 3.19 Conditioned on m , x and y are dependent.

3.20 See `demoCars.m`. Intuitively, it's clear in this case that h and w are dependent since, if we know that the husband has a cheap car, the wife must also have a cheap car.

3.21 See `exerciseGameSkill.m`. A posteriori the player skills are dependent. The joint distribution over skill levels s_1, s_2, s_3, s_4 and game outcomes is

$$p(s_1)p(s_2)p(s_3)p(s_4) [p(a \text{ beats } b)]^2 [p(b \text{ beats } c)]^2 [p(a \text{ beats } c)]^2 p(c \text{ beats } a) [p(c \text{ beats } f)]^2$$

From this we can then calculate the posterior skill levels $p(s_{1:4} | \text{gameoutcomes})$, which is proportional to the above joint distribution.

Using this we can find the distribution of the skills of players A and D , $p(s_1, s_4 | \text{gameoutcomes})$. Then the probability that D beats A is the expectation of $1/(1 + \exp(s_1 - s_4))$ with respect to this distribution. This gives a value of 0.045.

The expected skill levels are then calculated using $s_i = \sum_{l=1}^{10} lp(s_i = l | \text{gameoutcomes})$. This gives expected skill levels 7.8671, 6.8493, 5.9844, 3.0456 for players a, b, c, d respectively.

```
import brml.*
[a b c d]=assign(1:4);
S=10; % number of skill levels
% players and their prior skills:
pot(a)=array(a,condp(ones(S,1)));
pot(b)=array(b,condp(ones(S,1)));
pot(c)=array(c,condp(ones(S,1)));
pot(d)=array(d,condp(ones(S,1)));

% game outcomes:
o(1)=skillpot(a,b,true,S);
o(2)=skillpot(a,b,true,S);
o(3)=skillpot(b,c,true,S);
o(4)=skillpot(b,c,true,S);
o(5)=skillpot(a,c,true,S);
o(6)=skillpot(a,c,true,S);
o(7)=skillpot(a,c,false,S);
o(8)=skillpot(d,c,false,S);
o(9)=skillpot(d,c,false,S);

p=multpots([pot o]); % joint distribution of skill levels and game outcomes

% marginal skills of each player
pa=condpot(p,a); meana=sum((pa.table).*(1:S)')
pb=condpot(p,b); meanb=sum((pb.table).*(1:S)')
pc=condpot(p,c); meanc=sum((pc.table).*(1:S)')
pd=condpot(p,d); meand=sum((pd.table).*(1:S)')

% prob that D beats A given evidence
margda=condpot(p,[d a]);
s=skillpot(d,a,true,S);
pDbeatsAgivenEvidence=sumpot(s*margda,[],0)

% prob that D beats A given evidence (and assuming a post indep skills)
margdmarga=condpot(p,d)*condpot(p,a);
pDbeatsAgivenEvidenceIndep=sumpot(s*margdmarga,[],0)

function p=skillpot(a,b,win,S)
% p_{i,j} = probability that a beats b given their respective skill levels s_i , s_j
import brml.*
for A=1:S
    for B=1:S
        table(A,B)=1./(1+exp(B-A));
        if ~win
            table(A,B)=1-table(A,B);
        end
    end
end
end
p=array([a,b],table);
```

3.22 This is quite similar to the previous question, with the advert interest levels playing a similar role as the skill level. When we display links to adverts A, B, C , the viewer must select one. He selects A with probability

$$p(\text{clicks on } A | \text{shown } A, B, C) = \frac{\exp(s_A - \max(s_B, s_C))}{\exp(s_A - \max(s_B, s_C)) + \exp(s_B - \max(s_C, s_A)) + \exp(s_C - \max(s_B, s_A))}$$

Since we assume time we show the three adverts, the results will be independent each time, we have

$$p(s_{1:10} | \text{observed clicks}) \propto \left(\prod_i p(s_i) \right) \prod_{j=1}^{20} p(\text{clicks on advert } A_j | \text{shown others})$$

where $p(s_i) = \text{const.}$. From this we can then calculate the marginal $p(s_i)$ and the expected interest levels:

3.2089, 2.6285, 1.6505, 3.2459, 3.2508, 4.3782, 2.5462, 4.3543, 3.2048, 3.9501

See `exerciseAdServe.m`.

```
import brml.*
S=5; % number of interest levels
% prior interest levels:
for i=1:10
    pot(i)=array(i,condp(ones(S,1)));
end
% ad outcomes:
o(1)=skillpotABC(1,2,3,S);
o(2)=skillpotABC(2,5,3,S);
o(3)=skillpotABC(4,7,10,S);
o(4)=skillpotABC(6,3,4,S);
o(5)=skillpotABC(6,8,5,S);
o(6)=skillpotABC(9,3,7,S);
o(7)=skillpotABC(10,2,4,S);
o(8)=skillpotABC(7,1,2,S);
o(9)=skillpotABC(8,6,4,S);
o(10)=skillpotABC(4,3,9,S);
o(11)=skillpotABC(5,4,1,S);
o(12)=skillpotABC(9,5,1,S);
o(13)=skillpotABC(6,7,8,S);
o(14)=skillpotABC(4,9,7,S);
o(15)=skillpotABC(10,8,6,S);
o(16)=skillpotABC(5,4,3,S);
o(17)=skillpotABC(8,7,2,S);
o(18)=skillpotABC(8,3,6,S);
o(19)=skillpotABC(6,3,2,S);
o(20)=skillpotABC(1,4,2,S);

p=multipots([pot o]); % joint distribution of interest levels and ad click information

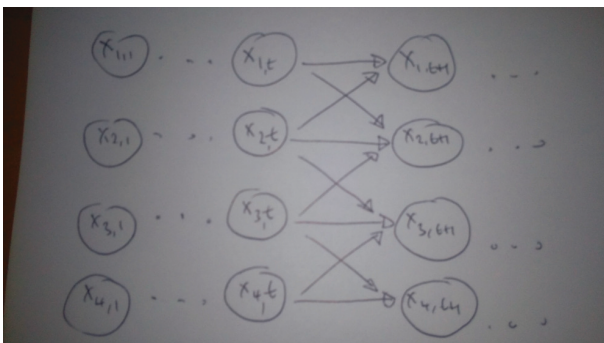
% marginal desirabilty of each ad
for i=1:10
    marg(i)=condpot(p,i); mn(i)=sum((marg(i).table).*(1:S)');
end
close all; bar(mn)

function p=skillpotABC(a,b,c,S)
import brml.*
for A=1:S
    for B=1:S
        for C=1:S
            table(A,B,C)=1./(1+exp(-A+max(B,C))*(exp(B-max(A,C))+exp(C-max(A,B))));
        end
    end
end
p=array([a,b,c],table);
```

3.23 1. Sequence probability = 5.35×10^{-6} .

2. Probability of random play = $1/3^{22} = 3.18 \times 10^{-11}$, so this gives a factor of 168151.
3. Probability that next move is rock is 0.25, paper 0.75 and scissors is zero.
4. Player 1 is most likely to play paper and we should therefore player 2 should play scissors.

3.25 1. Graph with nearest neighbour connections:



2. This is clear since for a chain we can simply reverse the arrows and obtain a Markov equivalent graph.

3. This can be done by considering two timesteps. Let's call the variables at timestep 1 $x = (x_1, x_2, x_3, x_4)$ and at timestep 2 $y = (y_1, y_2, y_3, y_4)$. Then the forward model describes

$$p(y_1|x_1, x_2)p(y_2|x_1, x_2, x_3)p(y_3|x_2, x_3, x_4)p(y_4|x_3, x_4)p(x_1)p(x_2)p(x_3)p(x_4)$$

We are now interested in $p(x|y)$. We can write this as

$$p(x_1|x_2, x_3, x_4, y_1, y_2, y_3, y_4)p(x_2|x_3, x_4, y_1, y_2, y_3, y_4)p(x_3|x_4, y_1, y_2, y_3, y_4)p(x_4|y_1, y_2, y_3, y_4)$$

Using the conditional independencies in the forward model, we can simplify this to

$$p(x_1|x_2, x_3, y_1, y_2)p(x_2|x_3, x_4, y_1, y_2, y_3)p(x_3|x_4, y_2, y_3, y_4)p(x_4|y_3, y_4)$$

If we count the total number of variables to the right of the conditioning bar, this is 15.

3.26 There are 45 possible combinations of modules. For each, we need to calculate the probability that the student will pass the msc. This requires introducing the constraint that all 8 modules must be passed, in addition to the project. I do this using julia code, as below. We then form the joint distribution of all modules and project and calculate the distribution $p(\text{pass}|\text{set of optionstaken})$. We can then search over the 45 possible option choices.

See the Julia code `msc.jl` which uses the BRML Julia toolbox. This gives solution

1. Graphical Models, Applied Machine Learning, Natural Language Modelling, Information Retrieval, Machine Vision, Advanced Topics, Affective Computing and Computational Modelling. Has probability of passing 0.636
2. Graphical Models, Research Methods, Applied Machine Learning, Natural Language Modelling, Machine Vision, Advanced Topics, Affective Computing and Computational Modelling. Has probability of passing 0.7698.
3. Graphical Models, Bioinformatics, Applied Machine Learning, Natural Language Modelling, Machine Vision, Advanced Topics, Affective Computing and Computational Modelling. Has probability of passing 0.728

```
include("passmodule.jl")
```

```
gm,rm,bi,aml,nlp,ir,mv,atml,ac,cm,proj,msc=1,2,3,4,5,6,7,8,9,10,11,12
smaths,sprog,sorg,sbio,swrit=13,14,15,16,17
```

```
prior=Array{Any,20}
prior[smaths]=PotArray(smaths,[0.95 0.05])
prior[sprog]=PotArray(sprog,[0.8 0.2])
prior[sorg]=PotArray(sorg,[0.7 0.3])
prior[sbio]=PotArray(sbio,[0.3 0.7])
prior[swrit]=PotArray(swrit,[0.95 0.05])
```

```
#There are 45 choices of 8 out of 10 modules:
choices=collect(combinations(1:10,8))
```

```
# Set up a potential pmsc that corresponds to the constraint that one must pass all eight modules and the project in order to
```

```
t=zeros(tuple{2*ones{Int,1,12}...})
for c=1:length(choices)
    par=2*ones{Int,1,10}
    for i=1:8
        par[choices[c][i]]=1
    end
    t[1,1,par...]=1
end
t[1,2,:]=0; # fail if you don't pass the project
t[2,:]=1-t[1,:]; # normalisation
pmsc=PotArray([msc proj gm rm bi aml nlp ir mv atml ac cm],t)
```

```
# now enumerate over all 45 possible choices, computing the marginal probability of passing, and conditional marginals as re
```

```
probpas=zeros(1,length(choices))
probpas1=zeros(1,length(choices))
probpas2=zeros(1,length(choices))
priors=prior[smaths]*prior[sprog]*prior[sorg]*prior[sbio]*prior[swrit]
for c=1:length(choices)
    take=Array{Bool,40}; fill!(take,false)
    take[proj]=true
    for i=1:8
        take[choices[c][i]]=true
    end

    joint=priors
    joint*=passmodule(gm,[smaths sprog],take[gm])
    joint*=passmodule(rm,[sprog sorg],take[rm])
    joint*=passmodule(bi,[sbio swrit],take[bi])
    joint*=passmodule(aml,[smaths sprog],take[aml])
```

```

joint*=passmodule(nlp,[smaths sprog],take[nlp])
joint*=passmodule(ir,sprog,take[ir])
joint*=passmodule(mv,smaths,take[mv])
joint*=passmodule(atml,smaths,take[atml])
joint*=passmodule(ac,swrit,take[ac])
joint*=passmodule(cm,smaths,take[cm])
joint*=passmodule(proj,[swrit sorg],take[proj])
joint*=pmc
probpas[c]=(condpot(joint,msc)).content[1] # first part of question
probpas1[c]=(condpot(setpot(joint,[smaths sorg],[1 1]),msc)).content[1] # 2nd part of question
probpas2[c]=(condpot(setpot(joint,[sbio swrit],[1 1]),msc)).content[1] # 3rd part of question
end

val,ind=findmax(probpas) # find which combination of modules is most likely to pass
choices[ind]

val,ind=findmax(probpas1) # find which combination of modules is most likely to pass, given student is skilled maths and organisation
choices[ind]

val,ind=findmax(probpas2) # find which combination of modules is most likely to pass, given student is skilled in biology and writing
choices[ind]

function passmodule(Module,Skills,takeModule)

v=hcat(Module,Skills)
t=zeros(tuple(2*ones(Int,1,length(v))...))
pass=1; fail=2; skill=1; noskill=2

if length(Skills)==1
    t[pass,skill]=0.995; t[fail,skill]=0.005
    t[pass,noskill]=0.1; t[fail,noskill]=0.9
    if ~takeModule
        t[pass,:]=0; t[fail,:]=1
    end
else
    t[pass,skill,skill]=0.99; t[fail,skill,skill]=0.01
    t[pass,skill,noskill]=0.75; t[fail,skill,noskill]=0.25
    t[pass,noskill,skill]=0.75; t[fail,noskill,skill]=0.25
    t[pass,noskill,noskill]=0.01; t[fail,noskill,noskill]=0.99

    if ~takeModule
        t[pass,:,:]=0; t[fail,:,:]=1
    end
end

p=PotArray(hcat(Module, Skills),t)
end

```

4.1 (1)

$$p(x_1|x_2, x_4) = \frac{\phi(x_1, x_2)\phi(x_4, x_1)}{\sum_{x_1} \phi(x_1, x_2)\phi(x_4, x_1)} \quad (\text{B.0.39})$$

$$p(x_2|x_1, x_3) = \frac{\phi(x_1, x_2)\phi(x_2, x_3)}{\sum_{x_2} \phi(x_1, x_2)\phi(x_2, x_3)} \quad (\text{B.0.40})$$

$$p(x_3|x_2, x_4) = \frac{\phi(x_2, x_3)\phi(x_3, x_4)}{\sum_{x_3} \phi(x_2, x_3)\phi(x_3, x_4)} \quad (\text{B.0.41})$$

$$p(x_4|x_1, x_3) = \frac{\phi(x_3, x_4)\phi(x_4, x_1)}{\sum_{x_4} \phi(x_3, x_4)\phi(x_4, x_1)} \quad (\text{B.0.42})$$

(2) It is not always possible to do this. One way to see this is to consider for example

$$p_1(x_1|x_2, x_4) \sum_{x_1, x_3} p(x_1, x_2, x_3, x_4) = p(x_1, x_2, x_3, x_4)$$

If the variables are all binary, since there are 4 states of x_2, x_4 , this gives a set of 4 equations that have to be satisfied by p . The same is true for each other conditional statement, giving a set of 16 linear equations that need to be satisfied. However, $p(x_1, x_2, x_3, x_4)$ only has 15 independent degrees of freedom (due to the normalisation requirement). Hence, in general, there are too many equations to satisfy.

4.2 Marginal is

$$\phi(a, b = 1)\phi(b = 1, c) + \phi(a, b = 2)\phi(b = 2, c)$$

If $\phi(b = 1, c) = 0$, then a and c are independent.

4.3 The only place \mathbf{W} enters is through the expression $\phi \equiv \mathbf{x}^\top \mathbf{W} \mathbf{x}$. Since this is a scalar, we can take the transpose, so that $\phi \equiv \mathbf{x}^\top \mathbf{W}^\top \mathbf{x}$, and $\phi = \mathbf{x}^\top \frac{1}{2} (\mathbf{W} + \mathbf{W}^\top) \mathbf{x}$. Hence any non-symmetric \mathbf{W} is essentially converted to a symmetric form, so that we may therefore assume this symmetry with loss of generality.

4.4 (1)

$$p(\mathbf{h}|\mathbf{v}) \propto e^{\mathbf{h}^\top (\mathbf{b} + \mathbf{W}^\top \mathbf{v})} = \prod_i e^{h_i (b_i + \sum_j W_{ji} v_j)} \quad (\text{B.0.43})$$

Using the fact that $h_i \in \{0, 1\}$, we obtain the required results. (2)

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}), \quad p(v_i|\mathbf{h}) = \sigma \left(a_i + \sum_j W_{ij} h_j \right) \quad (\text{B.0.44})$$

(3) $p(\mathbf{h})$ is not factorised since summing over \mathbf{v} introduces dependencies. (4) For a general \mathbf{W} , there is no known efficient way to compute Z . One way to argue this is that, after summing over \mathbf{v} , all the \mathbf{h} components become coupled into one clique. This clique has no graphical structure that can be exploited to easily reduce the computation. Hence the computation will scale with $O(V2^H)$ (or $O(H2^V)$ if $H > V$).

4.5 From the first constraint we have

$$p(x, y|z, u) = p(x|zu)p(y|zu) \quad (\text{B.0.45})$$

we can then write the joint as

$$p(x, y, z, u) = p(x|zu)p(y|zu)p(z, u) \quad (\text{B.0.46})$$

The second statement says $p(z, u) = p(z)p(u)$. Hence the most general form of the distribution is

$$p(x, y, z, u) = p(x|zu)p(y|zu)p(z)p(u) \quad (\text{B.0.47})$$

4.6 Using the obvious shorthand, we have

$$p_{12345} = \phi_{12}\phi_{23}\phi_{34}\phi_{45}\phi_{51} \quad (\text{B.0.48})$$

By summing over 3, 4, we obtain

$$p_{125} = \phi_{12}\phi_{51} \sum_{34} \phi_{23}\phi_{34}\phi_{45} \quad (\text{B.0.49})$$

Similarly, we obtain

$$p_{245} = \phi_{45} \sum_{13} \phi_{12}\phi_{23}\phi_{34}\phi_{51}, \quad p_{234} = \phi_{23}\phi_{34} \sum_{15} \phi_{12}\phi_{45}\phi_{51} \quad (\text{B.0.50})$$

Multiplying these together we obtain

$$p_{125}p_{245}p_{234} = \phi_{12}\phi_{51}\phi_{45}\phi_{23}\phi_{34} \left(\sum_{34} \phi_{23}\phi_{34}\phi_{45} \right) \left(\sum_{13} \phi_{12}\phi_{23}\phi_{34}\phi_{51} \right) \left(\sum_{15} \phi_{12}\phi_{45}\phi_{51} \right) \quad (\text{B.0.51})$$

Also

$$p_{25} = \sum_{134} \phi_{12}\phi_{23}\phi_{34}\phi_{45}\phi_{51}, \quad p_{24} = \sum_{135} \phi_{12}\phi_{23}\phi_{34}\phi_{45}\phi_{51} \quad (\text{B.0.52})$$

Hence

$$\frac{p_{125}p_{245}p_{234}}{p_{25}p_{24}} = p \frac{(\sum_{34} \phi_{23}\phi_{34}\phi_{45}) (\sum_{13} \phi_{12}\phi_{23}\phi_{34}\phi_{51}) (\sum_{15} \phi_{12}\phi_{45}\phi_{51})}{(\sum_{134} \phi_{12}\phi_{23}\phi_{34}\phi_{45}\phi_{51}) (\sum_{135} \phi_{12}\phi_{23}\phi_{34}\phi_{45}\phi_{51})} \quad (\text{B.0.53})$$

$$= p \frac{(\sum_{34} \phi_{23}\phi_{34}\phi_{45}) (\sum_{13} \phi_{12}\phi_{23}\phi_{34}\phi_{51}) (\sum_{15} \phi_{12}\phi_{45}\phi_{51})}{(\sum_1 \phi_{12}\phi_{51} \sum_{34} \phi_{23}\phi_{34}\phi_{45}) (\sum_3 \phi_{23}\phi_{34} \sum_{15} \phi_{12}\phi_{45}\phi_{51})} \quad (\text{B.0.54})$$

$$= p \frac{(\sum_{34} \phi_{23}\phi_{34}\phi_{45}) (\sum_1 \phi_{12}\phi_{51}) (\sum_3 \phi_{23}\phi_{34}) (\sum_{15} \phi_{12}\phi_{45}\phi_{51})}{(\sum_1 \phi_{12}\phi_{51} \sum_{34} \phi_{23}\phi_{34}\phi_{45}) (\sum_3 \phi_{23}\phi_{34} \sum_{15} \phi_{12}\phi_{45}\phi_{51})} \quad (\text{B.0.55})$$

$$= p \quad (\text{B.0.56})$$

The reason this works is more clearly seen when considering the junction tree representation. By triangulation, we can write the distribution using cliques on $(1, 2, 5)$, $(2, 4, 5)$ and $(2, 3, 4)$, with $(2, 4)$ and $(2, 5)$ being their separators.

4.7 (1) The MN is obtained by summing over h_1, h_2 , which fully couples all x_1, x_2, x_3 . (2) This is not a perfect map since in $p(x_1, x_2, x_3)$ we have $x_1 \perp\!\!\!\perp x_3 | \emptyset$, which is violated by the MN representation.

4.8 (1) Not possible in general since if we are given $p(x|y)$ we only need to specify $p(y)$. In specifying $p(y|x)$ we have too many degrees of freedom. However, if the data x, y that both labs used to compute their estimates was the same and generated from the same joint distribution $p(x, y)$, one would hope that these conditional marginals are reasonably consistent. (2) Writing

$$p(x) = \sum_y p_A(x|y)p(y) = \sum_y p_A(x|y) \sum_x p_B(y|x)p(x) = \sum_x \underbrace{\sum_y p_A(x|y)p_B(y|x)}_C p(x) \quad (\text{B.0.57})$$

we have an eigen-equation. The matrix C is by construction stochastic so is guaranteed to have an probability distribution with eigenvalue 1. The same is true for $p(y)$ with a stochastic matrix D . This defines $p(x)$ and $p(y)$.

4.9 (1) The fully factored distribution is consistent with all independence statements, so there is at least one trivial solution. (2) This is a difficult problem. One approach would be to assume a graphical model class, for example a Markov Network, and then search over the exponentially many graph structures, checking that each is consistent with all statements in both lists.

4.10 (1)

$$p(x|z) = \frac{\sum_{w,y} p(x, y, w, z)}{\sum_{x,w,y} p(x, y, w, z)} = \frac{\sum_{w,y} p(z|w)p(w|x)p(x)p(y)}{\sum_{x,w,y} p(z|w)p(w|x)p(x)p(y)} \quad (\text{B.0.58})$$

(2)

$$p(y|z) = \frac{\sum_{w,x} p(x, y, w, z)}{\sum_{x,w,y} p(x, y, w, z)} = \frac{\sum_{w,x} p(z|w)p(w|x)p(x)p(y)}{\sum_{x,w,y} p(z|w)p(w|x)p(x)p(y)} \quad (\text{B.0.59})$$

(3)

$$p(x, y|z) = \frac{\sum_w p(x, y, w, z)}{\sum_{x,w,y} p(x, y, w, z)} = \frac{\sum_w p(z|w)p(w|x)p(x)p(y)}{\sum_{x,w,y} p(z|w)p(w|x)p(x)p(y)} \quad (\text{B.0.60})$$

The numerator does not split into a product of a function of x times a function of y and hence in general this is not product of $p(x|z)p(y|z)$ and x and y are dependent conditioned on z .

4.11 (2) No, since summing over h couples all t_1, t_2, y_1, y_2 together. (3) One can show the independence either by explicit summation and then using the definition of independence. Alternatively a simple observation that there is a collider on the path is sufficient.

4.12 (1) An undirected 4 cycle. (2) Distribution cannot be written as a belief network since a BN must have a collider (two arrows pointing in). We can then marginalise over this collider node and the remaining three variables will have a missing edge. Viewed as a Markov network, the graph of these three variables has a missing edge. If we take the original Markov network over 4 variables, and marginalise over one variable, the remaining three become fully connected, with no missing edges. (3) a and c are dependent since they are connected by b .

4.13 Simply choose any node on the undirected graph as the root of the DAG and then orient edges consistently away from this root.

4.14 We first note that $\mathbb{I}[s_i = 0, s_j = 0] = (1 - s_i)(1 - s_j)$, $\mathbb{I}[s_i = 0, s_j = 1] = (1 - s_i)s_j$, $\mathbb{I}[s_i = 1, s_j = 0] = s_i(1 - s_j)$, $\mathbb{I}[s_i = 1, s_j = 1] = s_i s_j$. Consider a term

$$\begin{aligned} \phi_{ij}(s_i, s_j) &= e^{\log \phi_{ij}(s_i, s_j)} \\ &= e^{\mathbb{I}[s_i=0, s_j=0] \log \phi_{ij}(0,0) + \mathbb{I}[s_i=0, s_j=1] \log \phi_{ij}(0,1) + \mathbb{I}[s_i=1, s_j=0] \log \phi_{ij}(1,0) + \mathbb{I}[s_i=1, s_j=1] \log \phi_{ij}(1,1)} \end{aligned}$$

Since each $\mathbb{I}[\cdot]$ term is a quadratic form in s , the above can be written as a Boltzmann machine by collecting all the quadratic terms $s_i s_j$ and linear terms s_i with suitable coefficients.

4.15 The file `marginalconsistency.m` contains a routine for this based on a simple linear solver. One could also call the `linprog.m` routine, but the routine is potentially more instructive.

1.

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix}}_{\mathbf{M}} \underbrace{\begin{pmatrix} q_{12}(0,0) \\ q_{12}(0,1) \\ q_{12}(1,0) \\ q_{12}(1,1) \\ q_{13}(0,0) \\ q_{13}(0,1) \\ q_{13}(1,0) \\ q_{13}(1,1) \\ q_{23}(0,0) \\ q_{23}(0,1) \\ q_{23}(1,0) \\ q_{23}(1,1) \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\mathbf{c}}$$

2.

$$\underbrace{\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} p_{123}(0,0,0) \\ p_{123}(0,0,1) \\ p_{123}(0,1,0) \\ p_{123}(0,1,1) \\ p_{123}(1,0,0) \\ p_{123}(1,0,1) \\ p_{123}(1,1,0) \\ p_{123}(1,1,1) \end{pmatrix}}_{\mathbf{z}} = \mathbf{y}$$

5.1 By definition, a singly connected network must have an edge node that is connected to at most one neighbour. Let's call this x_e with neighbour x_n . Then there is a factor in the MN $\phi(x_e, x_n)$. We can then write

$$Z = \sum_{\mathcal{X}} \prod_{i \sim j} \phi(x_i, x_j) = \sum_{\mathcal{X} \setminus \mathcal{E}} \phi(\text{rest}) \underbrace{\sum_{x_e} \phi(x_e, x_n)}_{\psi(x_n)}$$

where $\phi(\text{rest})$ represents the product of the potentials on the remaining graph, excluding $\phi(x_e, x_n)$. Since $\psi(x_n)$ is a function only of x_n , we can merge $\psi(x_n)$ with another existing potential to form a new singly connected graph on $N - 1$ variables (assuming there are N variables in the distribution). That is, we identify a variable m that is a neighbour of n and replace $\phi(x_n, x_m) \rightarrow \phi(x_n, x_m)\psi(x_n)$. The remaining graph has $N - 1$ variables and is singly connected. We then recursively apply this rule, ending up with a pair of nodes at the end whose states we can simply sum over. This results in a time complexity that scales with N .

5.2 Naively, this would appear difficult since it is not singly connected so that when we sum over one of the variables, we introduce links to between the remaining variables. However, if we condition on one of the variables, say x_1 , the graph is singly connected. This means that we can carry out two efficient maximisations, one for each state of x_1 , and then choose that with the highest value. This is a form of loop-cut conditioning. More formally, we can write

$$\max_{x_1, \dots, x_{100}} p(x) = \max_{x_1} \underbrace{\max_{x_2, \dots, x_{100}} \phi(x_1, x_{100}) \prod_{i=1}^{99} \phi(x_i, x_{i+1})}_{f(x_1)}$$

For each state of x_1 , $f(x_1)$ can be computed efficiently using message passing since the graph (conditioned on a state of x_1) is singly connected. By enumerating over the states of x_1 , one may then find x_1^* . Given this one then sets x_1 to this optimal state, breaking the loop, and may carry out the maximisation in the resulting singly connected structure. Alternatively, if the two sets of messages (for binary variables) have been retained, one backtracks, knowing that x_1 is optimally in the state x_1^* .

5.3 1. 14, 33, 74, 77, 91, 97, 98, 99, 100. One may achieve this by taking $(\mathbf{M} + \mathbf{I})^{10}$, and listing which elements of this matrix have zero in their second column.

2. Yes. This can be verified by taking powers $(\mathbf{M} + \mathbf{I})^n$, noticing there is a zero component in this matrix until $n = 13$.

3. We first need to compute a transition matrix with elements

$$T_{ij} = \frac{M_{ij}}{\sum_i M_{ij}}$$

This is the stationary distribution $\mathbf{T}^\infty \mathbf{s}$ where \mathbf{s} is the zero vector except for $s_1 = 1$. One can compute \mathbf{M}^∞ using an eigendecomposition (or approximately by repeated multiplication):

$$\mathbf{T}^n = \mathbf{E} \boldsymbol{\lambda}^n \mathbf{E}^\top$$

For $n \rightarrow \infty$, only those eigenvalues $\lambda_{i,i}$ that are 1 will survive. For the rest λ_{jj}^n tends to 0 for $\lambda_{jj} < 1$. Then, taking the first element of the vector $\mathbf{T}^\infty \mathbf{s}$ gives the stationary probability for being in room 1, given that we started in room 1. This gives value 0.0080808080808. Note that there are in fact three eigenvectors with eigenvalue 1 in this case - that is there is no unique stationary distribution (there is no 'equilibrium' distribution. The stationary distribution we end up with depends on which room we begin in.

4.

$$\begin{aligned} p(\text{at least one in room 1}) &= 1 - p(\text{neither in room 1}) = 1 - p(\text{player 1 not in room 1})p(\text{player 2 not in room 1}) \\ &= 1 - (1 - p(\text{player 1 in room 1}))(1 - p(\text{player 2 in room 1})) = 1 - (1 - 0.0080808080808)^2 = 0.01609 \end{aligned}$$

5.4 1. This is given in fig(23.4).

2. See fig(23.5a).

3. Since $p(h_{1:T}|v_{1:T}) \propto p(h_{1:T}, v_{1:T})$, we may compute the ‘marginals’ $p(h_t, v_{1:T})$ in which all the observations are set. Once we’ve set $v_{1:T}$, then as a factor graph, we can consider

$$\psi_t(h_t) \equiv p(v_t|h_t), \quad \phi_t(h_t, h_{t-1}) \equiv p(h_t|h_{t-1})$$

and

$$p(h_{1:T}, v_{1:T}) = \prod_t \psi_t(h_t) \phi_t(h_t, h_{t-1})$$

We can pass a factor-to-variable message:

$$\mu_{\psi_t \rightarrow h_t}(h_t) = \psi_t(h_t), \quad \mu_{\phi_t \rightarrow h_t}(h_t) = \sum_{h_{t-1}} \phi_t(h_t, h_{t-1}) \mu_{h_{t-1} \rightarrow \phi_t}(h_{t-1})$$

4. We can pass variable-to-factor messages:

$$\mu_{h_{t-1} \rightarrow \phi_t}(h_{t-1}) = \mu_{\psi_{t-1} \rightarrow h_{t-1}}(h_{t-1}) \mu_{\phi_{t-1} \rightarrow h_{t-1}}(h_{t-1})$$

We can pass messages by starting from time 1, computing the ψ_1 to h_1 message, and then the h_1 to ϕ_1 factor message, then the ϕ_1 to h_2 message. We continue in this way until time T . Once we are at the end, we then pass the reverse messages:

We can pass a factor-to-variable message:

$$\mu_{\phi_t \rightarrow h_{t-1}}(h_{t-1}) = \sum_{h_t} \phi_t(h_t, h_{t-1}) \mu_{h_t \rightarrow \phi_t}(h_t)$$

We can pass variable-to-factor messages:

$$\mu_{h_t \rightarrow \phi_t}(h_t) = \mu_{\psi_t \rightarrow h_t}(h_t) \mu_{\phi_{t+1} \rightarrow h_t}(h_t)$$

Once we’ve passed these messages back from time T to 1, then the marginal is

$$p(h_t|v_{1:T}) \propto \mu_{\phi_t \rightarrow h_t}(h_t) \mu_{\psi_t \rightarrow h_t}(h_t) \mu_{\phi_{t+1} \rightarrow h_t}(h_t)$$

5. To compute $p(h_t, h_{t+1}, v_1, \dots, v_T)$ we need to sum over all the variables in the distribution before h_t and after h_{t+1} . This will leave us with

$$p(h_t, h_{t+1}|v_1, \dots, v_T) \propto \mu_{\phi_t \rightarrow h_t}(h_t) \mu_{\psi_t \rightarrow h_t}(h_t) \phi_{t+1}(h_t, h_{t+1}) \mu_{\phi_{t+2} \rightarrow h_{t+1}}(h_{t+1}) \mu_{\psi_{t+1} \rightarrow h_{t+1}}(h_{t+1})$$

5.5 Consider a simple linear chain x_1, \dots, x_n . In this case the summation over the variables x_{m+1}, \dots, x_n is straightforward and will not affect the structure of the chain. This intuition then suggests that as long as the connection between the removed variables and the remaining variables is limited in a similar manner, the marginal distribution still retains a tree structure.

5.6 This is essentially answered in the book. The basic idea is that the unit eigenvector of \mathbf{M} corresponds to the stationary distribution, and the i^{th} entry of this eigenvector is the corresponding probability of visiting page i by random surfing. There are some subtleties, however, since it might be that there is more than one unit eigenvalue and therefore no unique stationary eigenvector. Intuitively, this can happen when there are essentially isolated parts of the internet that cannot be reached from other islands – in this case it matters in which island one begins in as to what the stationary distribution will be. In order to prevent this, one can make an ‘ergodic’ Markov chain by adding a small amount ϵ to each entry of \mathbf{M} and renormalising. In this way, we will be able to visit any node from any other node, and avoid having isolated islands. In this case the stationary distribution will be independent of where we start. The relevance of page i for a search engine can then be taken to be the i^{th} entry of the corresponding unique unit-eigenvector.

- 5.7** 1. The distribution

$$\begin{aligned} p(h_1, \dots, h_T, x_1, \dots, x_T) &= \sum_{y_1, \dots, y_T} p(x_1, \dots, x_T, y_1, \dots, y_T, h_1, \dots, h_T) \\ &= \sum_{y_1, \dots, y_T} p(x_1|h_1)p(y_1|h_1)p(h_1) \prod_{t=2}^T p(h_t|h_{t-1})p(x_t|h_t)p(y_t|h_t) \\ &= \left(\prod_{t=1}^T \sum_{y_t} p(y_t|h_t) \right) p(x_1|h_1)p(h_1) \prod_{t=2}^T p(h_t|h_{t-1})p(x_t|h_t) \\ &= p(x_1|h_1)p(h_1) \prod_{t=2}^T p(h_t|h_{t-1})p(x_t|h_t) \end{aligned}$$

Hence $p(h_1, \dots, h_T, x_1, \dots, x_T)$ is a simple HMM. We can then compute the optimal state h_1^*, \dots, h_T^* by the max-product algorithm on the corresponding factor graph. Then

$$p(y_1, \dots, y_T, h_1, \dots, h_T) = \sum_{x_1, \dots, x_T} p(h_1, \dots, h_T, x_1, \dots, x_T)$$

By symmetry with the above argument, we obtain

$$p(y_1, \dots, y_T, h_1, \dots, h_T) = p(y_1|h_1)p(h_1) \prod_{t=2}^T p(h_t|h_{t-1})p(y_t|h_t)$$

which is again a HMM. When h_1, \dots, h_T is clamped into a known state h_1^*, \dots, h_T^* , then the above, as a function of y_1, \dots, y_T simply decomposes into independent terms, for which we can compute each $y_t^* = \arg \max_{y_t} p(y_t|h_t^*)$.

2. See `demoBanana.m`

The most likely y -sequence, based on the most likely h :

CTTGACTTGACTGACTGACTGACTGACTGATTGACTGAGACTGACTGTTTTTTCTGACTGACTGACTGACTGACTGACTGACTGACTGACTGACTGA

```
function demoBanana
import brml.*
load banana

T=100; H=5;
ht=1:T;
xt=T+1:2*T;
yt=2*T+1:3*T;

xx(find(x=='A'))=1;
xx(find(x=='C'))=2;
xx(find(x=='G'))=3;
xx(find(x=='T'))=4;
x=xx;

for t=1:T
    varinf(xt(t)).name=''; varinf(xt(t)).domain={'a','c','g','t'};
    varinf(yt(t)).name=''; varinf(yt(t)).domain={'a','c','g','t'};
    varinf(ht(t)).name=''; varinf(ht(t)).domain=1:H;
end

for t=1:T
    pot{xt(t)}=array([xt(t) ht(t)],pxgh);
    pot{yt(t)}=array([yt(t) ht(t)],pygh);
end
pot{ht(1)}=array(ht(1),ph1);

for t=2:T
    pot{ht(t)}=array([ht(t) ht(t-1)],phtghm);
end

for t=1:T
    newpot{t}=multpots([setpot(pot{xt(t)}),xt(t),x(t)] pot{ht(t)}]);
end

% This computes the most likely joint h-state first, and then the most
% likely y's from that h-state.
A = FactorGraph(newpot); % variable nodes are first in A
[hmax maxval mess]=maxprodFG(newpot,A);
for t=1:T
    y(t)=argmax(pygh(:,hmax(t)));
end
yy(find(y==1))='A';
yy(find(y==2))='C';
yy(find(y==3))='G';
yy(find(y==4))='T';

fprintf(1,'most likely y-sequence, based on the most likely h:\n%s\n\n',yy)
```

3. This is not generally tractable since this is a form of mixed inference. Intuitively, when we sum over h_1, \dots, h_T we couple together all the y_1, \dots, y_T into one large clique, for which there is no structure left for a message passing approach.
4. By treating y_1, \dots, y_T as parameters, effectively we wish to find the most likely parameters. The EM algorithm would then use

$$q(h_1, \dots, h_T) \propto p(x_1, \dots, x_T, y_1^{old}, \dots, y_T^{old}, h_1, \dots, h_T)$$

which is just a HMM. The M-step is then

$$y_1^{new}, \dots, y_T^{new} = \underset{y_1, \dots, y_T}{\operatorname{argmax}} \langle \log p(x_1, \dots, x_T, y_1, \dots, y_T, h_1, \dots, h_T) \rangle_{q(h_1, \dots, h_T)}$$

Since $p(x_1, \dots, x_T, y_1, \dots, y_T, h_1, \dots, h_T)$ is a Markov chain, taking the logarithm means that we will have a sum of transitions such as

$$\langle \log p(x_1, \dots, x_T, y_1, \dots, y_T, h_1, \dots, h_T) \rangle_{q(h_1, \dots, h_T)} = \sum_t \underbrace{\langle \log p(x_t | h_t) p(y_t | h_t) p(h_t | h_{t-1}) \rangle_{q(h_t, h_{t-1})}}_{\psi(y_t)}$$

This is simply a sum of terms(given x_1, \dots, x_T)

$$f(y_1, \dots, y_T) \equiv \psi(y_1) + \psi(y_2) + \dots \psi(y_T)$$

The maximum of this function is easy to compute:

$$y_t^* = \underset{y_t}{\operatorname{argmax}} \psi(y_t)$$

The EM algorithm will then converge to a (possibly) local optimum. One needs to run this several times to be reasonably confident that the most likely state is found - that with the corresponding highest likelihood under several different EM runs. See `demoBanana.m`.

```
EM for approximating the most likely state of p(y|x):
EM loop=1, log likelihood=-185.973942
most likely sequence:
ATTGACTTGACTGACTGACTGACTGACCTGATTTTTTGACTGAGACTGACTGTTTTTTTTTATGACTGACTGACTGACTGACTGACTGACTGACTGACTGA
EM loop=2, log likelihood=-184.913254
most likely sequence:
ATTGACTTGACTGACTGACTGACTGACCTGATTTTTTGACTGAGACTGACTGTTTTTTTTTGTGACTGACTGACTGACTGACTGACTGACTGACTGACTGA
EM loop=3, log likelihood=-184.479771
most likely sequence:
ATTGACTTGACTGACTGACTGACTGACCTGATTTTTTGACTGAGACTGACTGTTTTTTTTTGTGACTGACTGACTGACTGACTGACTGACTGACTGACTGA
EM loop=4, log likelihood=-184.479771
most likely sequence:
ATTGACTTGACTGACTGACTGACTGACCTGATTTTTTGACTGAGACTGACTGTTTTTTTTTGTGACTGACTGACTGACTGACTGACTGACTGACTGACTGA
EM loop=5, log likelihood=-184.479771
most likely sequence:
ATTGACTTGACTGACTGACTGACTGACCTGATTTTTTGACTGAGACTGACTGTTTTTTTTTGTGACTGACTGACTGACTGACTGACTGACTGACTGACTGA
```

See `demoBanana.m`:

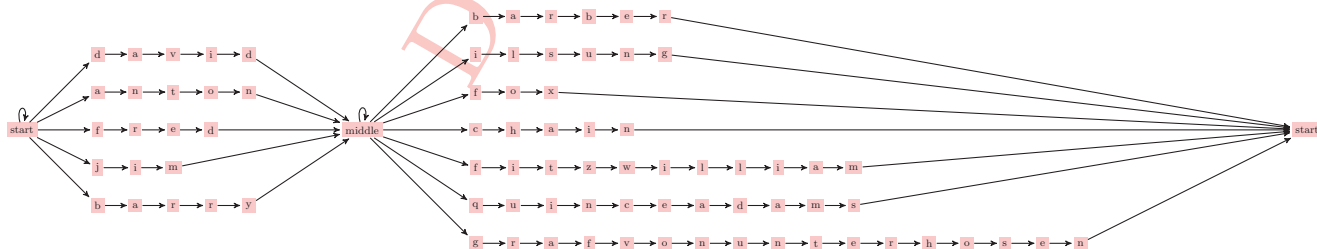
```
% The following approximates the the most likely joint y given x (after
% summing over all h) using an EM approximation approach
disp('EM for approximating the most likely state of p(y|x):')
for t=1:T
    potx{t} = setpot(pot{xt(t)},xt(t),x(t));
end
maxstateEM=y;

st=[];
for emloop=1:5
    for t=1:T
        newpot{t}=multpots([potx{t} setpot(pot{yt(t)},yt(t),maxstateEM(t)) pot{ht(t)}]);
    end
    [marg mess normconstpot]=sumprodFG(newpot,A);
    for t=1:T
        tmppot = multpots([logpot(pot{yt(t)}) marg{t}]);
        [dum maxstateEM(t)]=maxpot(sumpot(tmppot,ht(t)),yt(t));
    end

    yEM(find(maxstateEM==1))='A';
    yEM(find(maxstateEM==2))='C';
    yEM(find(maxstateEM==3))='G';
    yEM(find(maxstateEM==4))='T';

    fprintf(1,'EM loop=%d, log likelihood=%f\n',emloop,log(normconstpot.table));
    fprintf(1,'most likely sequence:\n%s\n',yEM)
end
```

5.8 This can be addressed by using a HMM in which we define a state for each character in all the firstnames and surnames, with an additional two states, giving the state-transition diagram as depicted.



The `start` and `middle` states self-transition with probability 0.8. The transitions into the the names are uniform and the transitions within and out of the names are deterministic. This defines a sparse 84×84 transition matrix A for the HMM. The 26×84 dimensional emission matrix has an entry 0.3 for emitting the correct letter, with a uniform probability on the other letters. The `start` and `middle` states emit a letter with uniform probability. This then defines the HMM, for which running Viterbi decoding on the $T = 10,000$ length sequence and computing the matrix m gives:

$$\begin{pmatrix} 6 & 3 & 7 & 12 & 19 & 10 & 11 \\ 8 & 7 & 7 & 6 & 16 & 10 & 13 \\ 8 & 7 & 8 & 13 & 11 & 7 & 19 \\ 14 & 12 & 12 & 16 & 21 & 12 & 18 \\ 11 & 6 & 10 & 12 & 5 & 10 & 13 \end{pmatrix}$$

See `solutionFirstnameSurname.m`.

5.9 The full generative model of this situation would be to have latent variable for each of the 500 people describing their x, y location on the grid. One of these 500 can be given the ‘fast’ dynamics, whilst the others can be described by ‘normal’ dynamics. These individuals move independently, giving rise to a set of 500 latent Markov chains. However, the observation process couples these chains together since if two individuals are occupying the same position, then we only observe that that position is occupied. If we were to then attempt to write this as a standard Hidden Markov Model, we would require $(50 \times 50)^{500}$ states, which is totally impractical. As an alternative, we may therefore attempt to model the position of the fast mover alone. If $h_t \in \{1, \dots, 2500\}$ represents the position on the $50 \times 50 = 2500$ state grid, then then we can define a 2500×2500 transition distribution which states that the fast mover can move 2 grid points at a time. If we know the location h_t of the fast mover, then we know that the pixel which the fast mover is at must be occupied. We want then the observations simply to set the set of possible positions that the fast mover could be in; without knowledge of the dynamics, the possible location at time t of the fast mover is any location which is currently occupied. Hence we can replace the standard emission term $p(v_t|h_t)$, with known v_t state by simply a time-dependent potential $\phi_t(h_t)$ which states which positions h_t are possible (using value 1) based on the observation at time t and impossible (value 0). This gives a form of Markov network

$$\prod_{t=2}^T \phi_t(h_t) p(h_t|h_{t-1})$$

We can then find the most likely $h_{1:T}$ by a modified form of Viterbi. This gives the following as the most likely locations of the fast mover:

32	35
34	37
32	39
30	37
28	39
30	41
32	43
30	45
28	47
26	45
24	47
22	49
24	47
22	45
24	43
22	41
24	39
22	37
24	39
26	37
28	39
26	37
28	35
26	33
24	35
22	37
24	35
22	37
24	39
26	41
28	39
26	41
28	43
26	41
28	39
30	37
32	39
34	41
32	43
34	45
32	43
30	41
32	43
34	41
36	39
34	37
36	39
34	41
32	39
30	41
32	43
34	41
36	43
34	41
36	43
38	41
40	39
42	37

```

40 39
38 37
40 35
42 33
40 35
38 33
36 35
34 33
36 31
38 33
36 35
38 37
36 35
38 37
40 35
38 33
36 31
38 33
40 31
42 29
40 31
42 29
40 31
42 33
44 31
46 29
48 27
46 25
48 27
46 25
44 27
46 25
48 27
46 25
48 27
46 25
44 23
46 25
44 23
46 21
44 19
46 21

close all; clear all; import brml.*

load drunkproblemX
load drunkproblemXtrue

Gx=50; Gy=50;

% map each x,y position to a unique state number
S=zeros(Gy, Gx);
i=0;
for x=1:Gx
    for y=1:Gy
        i=i+1;
        S(x,y)=i;
    end
end

% fast mover dynamics:
phghm=zeros(Gx*Gy);
for x=1:Gx
    for y=1:Gy
        xp=max(1,cap(x+2,Gx)); xm=max(1,cap(x-2,Gx));
        yp=max(1,cap(y+2,Gy)); ym=max(1,cap(y-2,Gy));
        phghm(S(xp,yp),S(x,y))=1;
        phghm(S(xp,ym),S(x,y))=1;
        phghm(S(xm,yp),S(x,y))=1;
        phghm(S(xm,ym),S(x,y))=1;
    end
end

phghm=condp(phghm+0.01);

T=100; P=500;

% Define the possible locations of the fast mover based on the observations alone:
phiht=zeros(Gx*Gy,T);
for t=1:T

```



```

i=0;
for xx=1:Gx
    for yy=1:Gy
        i=i+1;
        if X{t}(yy,xx)>0
            phiht(i,t)=1;
        end
    end
end
end

Nstates=Gx*Gy;
[maxstate logprob]=HMMviterbiGeneral(phghm,condp(ones(Nstates,1)),phiht);

for t=1:T
    Y=zeros(Gy,Gx);
    [i(t),j(t)]=find(S==maxstate(t));
    subplot(1,2,1); imagesc(Xtrue{t});
    Y(j(t),i(t))=1;
    subplot(1,2,2); imagesc(Y);
    drawnow; pause(0.1)
end
[i(:) j(:)]

function [maxstate logprob]=HMMviterbiGeneral(phghm,ph1,phiht)
%HMMVITERBI Viterbi most likely joint hidden state of a HMM for an
%inhomogenous emission distribution
%
% [maxstate logprob]=HMMviterbiGeneral(phghm,ph1,phiht)
%
% Inputs:
% v : visible (obervation) sequence being a vector v=[2 1 3 3 1 ...]
% phghm : homogeneous transition distribution phghm(i,j)=p(h(t)=i|h(t-1)=j)
% ph1 : initial distribution
% phiht : p(v,t)\propto phi(h,t) (for a fixed observed sequence, the
% emission distribution depends only on h and potentially t as well for an
% inhomogenous emission distribution)
%
% Outputs:
% maxstate : most likely joint hidden (latent) state sequence
% logprob : associated log probability of the most likely hidden sequence
% See also demoHMMinference.m
import brml.*
T=size(phiht,2); H=size(phghm,1);
mu(:,T)=ones(H,1);
for t=T:-1:2
    tmp = repmat(phiht(:,t).*mu(:,t),1,H).*phghm;
    mu(:,t-1)= condp(max(tmp)'); % normalise to avoid underflow
end
% backtrack
[val hs(1)]=max(ph1.*phiht(:,1).*mu(:,1));
for t=2:T
    tmp = phiht(:,t).*phghm(:,hs(t-1));
    [val hs(t)]=max(tmp.*mu(:,t));
end
maxstate=hs;
logprob=log(ph1(hs(1)))+log(phiht(hs(1),t));
for t=2:T
    logprob=logprob+log(phghm(hs(t),hs(t-1)))+log(phiht(hs(t),t));
end

```

5.10 The expected price gain is 1.432298 with standard deviation 8.540987. See `solutionBearBull.m`. This can be achieved by filtering. If $\alpha(h_t)$ represents $p(h_t|v_{1:t})$, then

$$\alpha(h_t) \propto \sum_{h_{t-1}} p(h_t|h_{t-1})p(v_t|v_{t-1}, h_t)\alpha(h_{t-1}) \quad (\text{B.0.61})$$

with initialisation $\alpha(h_1) = 0.5$. Prediction is then given by

$$p(v_{T+1}|v_{1:T}) \propto \sum_{h_{T+1}} p(h_{T+1}|h_T)\alpha(h_T)p(v_{T+1}|h_{T+1}) \quad (\text{B.0.62})$$

Note that one can also solve this question by defining a joint hidden variable $h_t, price_t$ and a deterministic emission that maps the latent price to itself. One can then define a time homogeneous HMM for this joint $H = 200$ state latent variable. This approach however is computationally very inefficient since it requires inference in a HMM with 200 latent states, compared to inference in a simple $H = 2$ singly-connected model in the direct approach outlined above.

Another approach would be to simply use for example the junction tree approach on suitably defined potentials.

```

function solutionBearBull
close all
import brml.*

[bear bull]=assign(1:2); % bear goes up, bull goes down

Tran(bear,bear)=0.8;
Tran(bull,bear)=0.2;
Tran(bull,bull)=0.7;
Tran(bear,bull)=0.3;

P=100; % number of price values
price=1:P; % price values
load BearBullproblem
T=length(p); % length of timeseries
subplot(2,1,1); plot(1:T,p,'-o'); title('price')

% dynamic programming for filtering:
[htm ht vtm vt]=assign(1:4);
tranpot=array([ht htm],Tran);
for h=1:2
    for ptm=price
        for pt=price
            if h==bull
                tmp(pt,ptm,h)=pbull(pt,ptm);
            else
                tmp(pt,ptm,h)=pbear(pt,ptm);
            end
        end
    end
end
empot=array([vt vtm ht],tmp);

f(:,1)=[0.5 0.5]';
filt=array(1,[0.5 0.5]);
for t=2:T
    filt=condpot(setpot(empot,[vtm vt],[p(t-1) p(t)])*tranpot*filt,ht); % filtered update
    f(:,t)=filt.table;
    filt.variables=1; % need to reset the filtered distribution variable number for the next timestep
end
subplot(2,1,2); plot(f(2,:),'-o'); title('p bull')

% prediction:
predh=condpot(filt*tranpot,ht); % latent state prediction
predv=condpot(setpot(empot,vtm,p(T))*predh,vt); % output state prediction

U=0; V=0;
for i=price
    U=U+predv.table(i)*(i-p(T)); % expected gain
    V=V+predv.table(i)*(i-p(T)).^2; % expected squared gain
end
fprintf(1,'Expected price gain = %f\n', U); % total expected gain
fprintf(1,'Expected price gain standard deviation = %f\n', sqrt(V-U^2));

```

5.11 We can write this as a deterministic Markov chain on the joint variables $\mathbf{x}_t, \mathbf{v}_t, \mathbf{a}_t$. However, we are only interested in the constraint \mathbf{x}_{102} being in the specified state. This means that we need to sum over all $\mathbf{x}_{2:101}, \mathbf{v}_{2:101}$ and finally to max over $\mathbf{a}_{1:100}$. There is no obvious efficient way to distribute this max-sum operation. Carrying out the sum is equivalent to eliminating the \mathbf{v} and \mathbf{x} variables to obtain (for a single dimension)

$$x_{T+2} = x_1 + T\delta v_1 + \delta^2 \sum_{t=1}^T (T-t+1)a_t$$

where $T = 100$. For the first dimension, we then need to find the 100 dimensional vector \mathbf{a} that satisfies the above equation for the given start position $x_1 = 0$ and end position $x_{T+2} = 4.71$. We can write this equation as

$$c = \sum_{t=1}^T b_t a_t, \quad a_t \in \{-1, 0, 1\}$$

where \mathbf{b} contains decreasing integers, decremented by 1.

Assuming there is a feasible solution means that the known constant c is an integer. Without loss of generality, we can assume this is positive. Since \mathbf{b} contains entries $1, 2, \dots, T$, we can obtain the value c by the simple process of including the largest elements of \mathbf{b} provided that the positive sum of these elements is below or equal to c . This greedy algorithm is efficient.

One repeats this for each of the three dimensions independently. My solution gives a total minimum fuel consumption of 22. See `BangBangSolution.m`.

To explain why this works, we first note that an optimal solution cannot contain any a_t set to -1 . The reason is that if we were to sum up all b_t for which $a_t = -1$, the total from the positive $a_t = +1$ contributions c' must be greater than c (since c is assumed positive). Obtaining c' by adding a positive subset of the b_t must require at least as many terms as obtaining c by adding up the b_t . Hence the optimal solution cannot contain any $a_t = -1$.

Now consider a problem of the form: find a subset (with the smallest number of elements) of $10, 9, 8, \dots, 1$, that adds up to 17. A solution is $10+7$. We can always get the optimal solution by starting from 10 and greedily including terms provided the sum does not exceed 17. To see this, let's assume that an optimal subset cannot contain 10. Then it cannot contain any subset that sums to 10 (since we could otherwise have replaced this subset by the single element 10). Now, the question is, if an optimal solution cannot have a subset that sums to 10, can it have a subset that sums to 9? Let's assume there is one. In our example, consider a set that adds to 17, namely 6, 3, 8. This has a subset 6, 3 that adds to 9. However, we can equivalently write this as $6+3+1$ and $8-1$. The first subset sums to 10 and the second to a value less than it was before. We can always arrange it so that the first subset sums now to 10 and the second to contain unique elements. But now we can replace $6+3+1$ by 10 and $8-1$ by 7 and we have a solution which contains the subset sum 10. By this wasn't allowed. Hence, not allowing an optimal solution to have a subset sum of 10 means that we do not allow an optimal solution to have a subset sum of 9. By induction, we therefore cannot allow any subset to sum to any value, meaning that no optimal solution exists. But this is a contradiction. Hence an optimal solution must contain 10.

```
clear all
T=100; D=3; del=0.1;
x=zeros(D,1); v=zeros(D,1); % initial position and velocity
xx = [4.71, -6.97, 8.59]'; % desired position

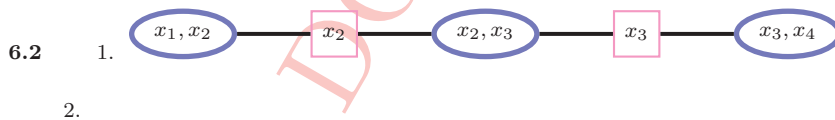
b=T:-1:1;
aopt=zeros(D,T);
for d=1:D
    c=(xx(d)-x(d)-T*del*v(d))/del^2;

    if c>0
        total=0;
        for t=1:T
            if abs(total+b(t)-c)<abs(total-c) && (total+b(t) < c)
                total=total+b(t);
                aopt(d,t)=1;
            end
        end
    end

    if c<0
        total=0;
        for t=1:T
            if abs(total-b(t)-c)<abs(total-c) && (total-b(t) > c)
                total=total-b(t);
                aopt(d,t)=-1;
            end
        end
    end
end
sum(abs(aopt(:))) % minimal total amount of fuel used
```

5.12 Consider a graph with a path $a \rightarrow b$ with weight $+1$ and an alternative path $a \rightarrow c \rightarrow d \rightarrow b$ with weights $-1, -1, 2$. The minimum weight path is $a \rightarrow c \rightarrow d \rightarrow b$ with weight 0. If we add $+1$ to each edge, then the path $a \rightarrow b$ has weight $1+1=2$ and the path $a \rightarrow c \rightarrow d \rightarrow b$ has weight 3, reversing the previous decision.

5.13 See `exerciseSimoHurrt.m`. The answer is -209.6499 .



$$p(x_1, x_2, x_3, x_4) = \frac{p(x_1, x_2)p(x_2, x_3)p(x_3, x_4)}{p(x_2)p(x_3)}$$

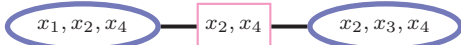
For interest:

$$\begin{aligned}\phi^*(x_2) &= \sum_{x_1} \phi(x_1, x_2) \\ \phi^*(x_2, x_3) &= \phi(x_2, x_3)\phi^*(x_2) = \phi(x_2, x_3) \sum_{x_1} \phi(x_1, x_2) \\ \phi^*(x_3) &= \sum_{x_2} \phi^*(x_2, x_3) = \sum_{x_2} \phi(x_2, x_3) \sum_{x_1} \phi(x_1, x_2) \\ \phi^*(x_3, x_4) &= \phi(x_3, x_4)\phi^*(x_3) = \phi(x_3, x_4) \sum_{x_2} \phi(x_2, x_3) \sum_{x_1} \phi(x_1, x_2) = \sum_{x_1, x_2} p(x_1, x_2, x_3, x_4) = p(x_3, x_4)\end{aligned}$$

Going back from right to left we have

$$\begin{aligned}
\phi^{**}(x_3) &= \sum_{x_4} \phi^*(x_3, x_4) = p(x_3) \\
\phi^{**}(x_2, x_3) &= \phi^*(x_2, x_3) \frac{\phi^{**}(x_3)}{\phi^*(x_3)} = \frac{\phi(x_2, x_3) \sum_{x_1} \phi(x_1, x_2) \sum_{x_4} \phi(x_3, x_4) \sum_{x_2} \phi(x_2, x_3) \sum_{x_1} \phi(x_1, x_2)}{\sum_{x_2} \phi(x_2, x_3) \sum_{x_1} \phi(x_1, x_2)} \\
&= \sum_{x_1, x_4} \phi(x_1, x_2) \phi(x_2, x_3) \phi(x_3, x_4) = \sum_{x_1, x_4} p(x_1, x_2, x_3, x_4) = p(x_2, x_3) \\
\phi^{**}(x_2) &= \sum_{x_3} \phi^{**}(x_2, x_3) = p(x_2) \\
\phi^{**}(x_1, x_2) &= \frac{\phi(x_1, x_2) \phi^{**}(x_2)}{\phi^*(x_2)} = \frac{\phi(x_1, x_2) \sum_{x_3} \sum_{x_1, x_4} \phi(x_1, x_2) \phi(x_2, x_3) \phi(x_3, x_4)}{\sum_{x_1} \phi(x_1, x_2)} \\
&= \sum_{x_3, x_4} \phi(x_1, x_2) \phi(x_2, x_3) \phi(x_3, x_4) = \sum_{x_3, x_4} p(x_1, x_2, x_3, x_4) = p(x_1, x_2)
\end{aligned}$$

6.3

1. 
2. Assign $\phi(x_1, x_2, x_4) = \phi(x_1, x_2)\phi(x_1, x_4)$, $\phi(x_2, x_3, x_4) = \phi(x_2, x_3)\phi(x_3, x_4)$. We'll go from left to right with initial separators set to unity:

$$\begin{aligned}
\phi^*(x_2, x_4) &= \sum_{x_1} \phi(x_1, x_2, x_4) = \sum_{x_1} \phi(x_1, x_2) \phi(x_1, x_4) \\
\phi^*(x_2, x_3, x_4) &= \phi(x_2, x_3, x_4) \phi^*(x_2, x_4) = \phi(x_2, x_3) \phi(x_3, x_4) \sum_{x_1} \phi(x_1, x_2) \phi(x_1, x_4) \\
&= \sum_{x_1} p(x_1, x_2, x_3, x_4) = p(x_2, x_3, x_4)
\end{aligned}$$

Going back from right to left we have

$$\begin{aligned}
\phi^{**}(x_2, x_4) &= \sum_{x_3} \phi^*(x_2, x_3, x_4) = \sum_{x_3} p(x_2, x_3, x_4) = p(x_2, x_4) \\
\phi^*(x_1, x_2, x_4) &= \frac{\phi(x_1, x_2, x_4) \phi^{**}(x_2, x_4)}{\phi^*(x_2, x_4)} \\
&= \frac{\phi(x_1, x_2, x_4) \sum_{x_3} \phi(x_2, x_3) \phi(x_3, x_4) \sum_{x_1} \phi(x_1, x_2) \phi(x_1, x_4)}{\sum_{x_1} \phi(x_1, x_2) \phi(x_1, x_4)} \\
&= \sum_{x_3} \phi(x_1, x_2) \phi(x_2, x_3) \phi(x_3, x_4) \phi(x_1, x_4) \\
&= \sum_{x_3} p(x_1, x_2, x_3, x_4) = p(x_1, x_2, x_3, x_4) = p(x_1, x_2, x_4)
\end{aligned}$$

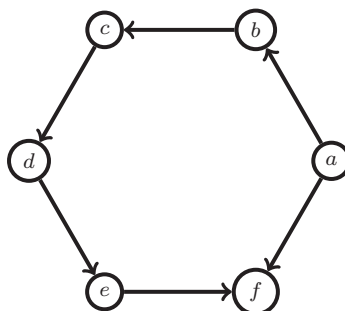
We can then find $p(x_1) = \sum_{x_2, x_4} \phi^{**}(x_1, x_2, x_4) = \sum_{x_2, x_4} p(x_1, x_2, x_4) = p(x_1)$.

6.4

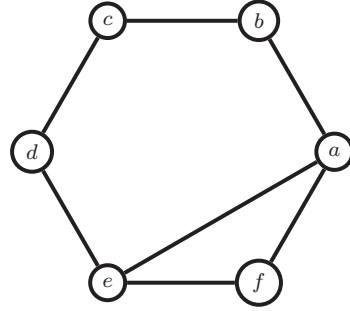
1. To be done.
2. To be done.
3. No unique solution. A triangulated graph contains cliques heb , hba , $hacg$, adg , $hcfg$, fgi . A clique of size 4 is the minimal largest clique available in this case.
4. To be done.
5. To be done.
6. To be done.

6.5

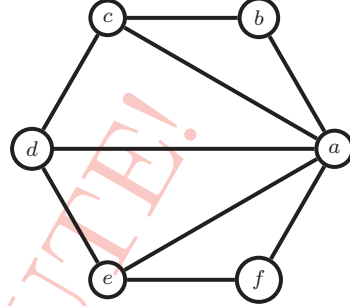
1. A directed hexagon with incoming links to f , with a as the root.



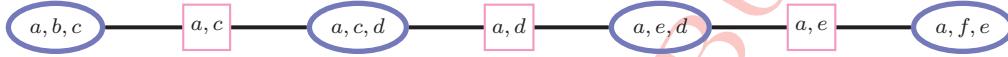
2. Extra link $a - e$.



3. Several equivalent answers. One is to radiate all links from a .



- 4.



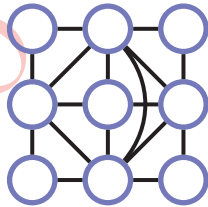
5. Several possible answers, but just distribute probability factors in one of the cliques in the junction tree, *e.g.*

$$\begin{aligned}\phi(a, b, c) &= p(a)p(b|a)p(c|a) \\ \phi(a, c, d) &= p(d|c) \\ \phi(a, e, d) &= p(e|d) \\ \phi(a, f, e) &= p(f|a, e)\end{aligned}$$

Set the separator potentials to 1.

6. A valid schedule is $abc \rightarrow acd \rightarrow aed \rightarrow afe \rightarrow aed \rightarrow acd \rightarrow abc$.
7. This is clear from the fact that this is a junction tree for the original distribution. One can also show this directly by repeated use of Bayes rule and the conditional independencies encoded in the distribution:

$$\begin{aligned}p(a|f)p(b|a)p(c|a)p(d|a)p(e|a)p(f) &= \frac{p(a,f)p(b,a,c)p(c,a,d)p(d,a,e)p(e,a,f)p(f)}{p(f)p(a,c)p(a,d)p(a,e)p(a,f)} \\ &= \frac{p(b,a,c)p(c,a,d)p(d,a,e)p(f|a,e)}{p(a,c)p(a,d)} \\ &= p(b,a,c)p(d|a,c)p(e|d,a)p(f|a,e) \\ &= p(a,b)p(c|b,a)p(d|c)p(e|d)p(f|a,e) \\ &= p(a)p(b|a)p(c|b)p(d|c)p(e|d)p(f|a,e)\end{aligned}$$



6.6 Different possible answers. One is

6.7 Complexity is $O(n2^{2n})$. For $n = 10$, $\log Z = 186.7916$. See `IsingZ.m`:

```
function IsingZ
import brml.*
N=10;

% form the potentials for each pair of neighbouring variables:
for x=1:2
    for y=1:2
        isingtable(x,y)=exp(1*(x==y));
    end
end
end
S = reshape(1:N*N,N,N);
c=0;
for s1=1:N*N
```

```

[i1 j1]=find(S==s1);
for s2=s1+1:N*N
    [i2 j2]=find(S==s2);
    if (j1==j2)&(abs(i1-i2)==1) | (i1==i2)&(abs(j1-j2)==1)
        c=c+1;
        phi{c}=array([s1 s2],isingtable);
        tab(i1,i2,j1,j2)=c;
        tab(i2,i1,j2,j1)=c;
    end
end
end

% form column potentials:
colphi{1}=const(1);
for i=1:N-1
    colphi{i}=const(1);
    for j=1:N-1
        colphi{i}=multpots([colphi{i} phi{tab(j,j,i,i+1)}}];
        colphi{i}=multpots([colphi{i} phi{tab(j,j+1,i,i)}}];
    end
    colphi{i}=multpots([colphi{i} phi{tab(N,N,i,i+1)}}];
end
for j=1:N-1
    colphi{j}=multpots([colphi{j} phi{tab(j,j+1,i+1,i+1)}}];
end

% do message passing in this new chain with the column potentials:
mmess=[];
sumover=setdiff(colphi{1}.variables,intersect(colphi{1}.variables,colphi{2}.variables));
mess=sumpot(colphi{1},sumover);
for i=2:N-2
    sumover=setdiff(colphi{i}.variables,intersect(colphi{i}.variables,colphi{i+1}.variables));
    mess=sumpot(multpots([mess colphi{i}]),sumover);
    mmess=[mmess max(mess.table(:))];
    mess.table=mess.table./mmess(end); % tables may overflow so remove constant factor
end
logZ = log(table(sumpot(multpots([mess colphi{N-1}]),[],0)));
logZ = logZ+ sum(log(mmess)) % add back any removed constant factor

```

6.8 This is given by the reabsorption procedure, as described in the text.

6.9 function diseaseNet

```

import brml.*
D=20;S=40; dv=1:D; sv=D+1:D+S;
load diseaseNet
pot=str2cell(setpotclass(pot,'array'));
drawNet(dag(pot),variable);

tstart=tic;
[jtpot jtsep infostruct]=jtree(pot); % setup the Junction Tree
jtpot=absorption(jtpot,jtsep,infostruct); % do full round of absorption
t=toc(tstart);
disp(['Junction Tree takes ',num2str(t),' seconds to form and perform absorption'])

for i=1:length(jtpot); label{i}=horzcat(variable(jtpot{i}.variables).name); sz(i)=length(jtpot{i}.variables); end
figure; draw_layout(infostruct.cliquetree,label);
drawnow
disp(['Maximum JT clique size is ',num2str(max(sz))])

for s=1:S
    jtpotnum = whichpot(jtpot,sv(s),1); % find a single JT potential that contains the symptom
    margpot=sumpot(jtpot{jtpotnum},sv(s),0);
    ps_jt(s)=margpot.table(1);
end

% More efficient summation based on needing only the parents of each symptom:
tstart=tic;
A = dag(pot);
for s=1:S
    margpot2 = sumpot(multpots([pot(sv(s)) pot(parents(A,sv(s)))],sv(s),0);
    ps_eff(s)=margpot2.table(1);
end
t=toc(tstart);
disp(['Efficient inference takes ',num2str(t),' seconds'])

disp('JT marginals and Efficient Inference marginals are the same:')
[ps_jt' ps_eff']

[jtpot jtsep]=jtassignpot(setpot(pot,[sv(1:5) sv(6:10)],[1 1 1 1 2 2 2 2 2]),infostruct);

```

```

jtpot=absorption(jtpot,jtsep,infostruct); % do full round of absorption
for d=1:D
    jtpotnum = whichpot(jtpot,d,1); % find a single JT potential that contains the disease
    margpot=condpot(sumpot(jtpot(jtpotnum),d,0));
    pd_jt(d)=margpot.table(1);
end
disp('The marginals p(disease=1|evidence):')

```

1. See diseaseNet.m.

```

p(s(1)=1): Junction tree 0.441834 : efficient inference: 0.441834
p(s(2)=1): Junction tree 0.456675 : efficient inference: 0.456675
p(s(3)=1): Junction tree 0.441405 : efficient inference: 0.441405
p(s(4)=1): Junction tree 0.491275 : efficient inference: 0.491275
p(s(5)=1): Junction tree 0.493892 : efficient inference: 0.493892
p(s(6)=1): Junction tree 0.657483 : efficient inference: 0.657483
p(s(7)=1): Junction tree 0.504563 : efficient inference: 0.504563
p(s(8)=1): Junction tree 0.268693 : efficient inference: 0.268693
p(s(9)=1): Junction tree 0.649077 : efficient inference: 0.649077
p(s(10)=1): Junction tree 0.49074 : efficient inference: 0.49074
p(s(11)=1): Junction tree 0.422552 : efficient inference: 0.422552
p(s(12)=1): Junction tree 0.429096 : efficient inference: 0.429096
p(s(13)=1): Junction tree 0.545021 : efficient inference: 0.545021
p(s(14)=1): Junction tree 0.632959 : efficient inference: 0.632959
p(s(15)=1): Junction tree 0.42954 : efficient inference: 0.42954
p(s(16)=1): Junction tree 0.458794 : efficient inference: 0.458794
p(s(17)=1): Junction tree 0.427559 : efficient inference: 0.427559
p(s(18)=1): Junction tree 0.404255 : efficient inference: 0.404255
p(s(19)=1): Junction tree 0.582093 : efficient inference: 0.582093
p(s(20)=1): Junction tree 0.589591 : efficient inference: 0.589591
p(s(21)=1): Junction tree 0.76127 : efficient inference: 0.76127
p(s(22)=1): Junction tree 0.695588 : efficient inference: 0.695588
p(s(23)=1): Junction tree 0.508702 : efficient inference: 0.508702
p(s(24)=1): Junction tree 0.419962 : efficient inference: 0.419962
p(s(25)=1): Junction tree 0.351942 : efficient inference: 0.351942
p(s(26)=1): Junction tree 0.389611 : efficient inference: 0.389611
p(s(27)=1): Junction tree 0.325973 : efficient inference: 0.325973
p(s(28)=1): Junction tree 0.469624 : efficient inference: 0.469624
p(s(29)=1): Junction tree 0.522868 : efficient inference: 0.522868
p(s(30)=1): Junction tree 0.717312 : efficient inference: 0.717312
p(s(31)=1): Junction tree 0.524199 : efficient inference: 0.524199
p(s(32)=1): Junction tree 0.353704 : efficient inference: 0.353704
p(s(33)=1): Junction tree 0.512679 : efficient inference: 0.512679
p(s(34)=1): Junction tree 0.529404 : efficient inference: 0.529404
p(s(35)=1): Junction tree 0.385751 : efficient inference: 0.385751
p(s(36)=1): Junction tree 0.489095 : efficient inference: 0.489095
p(s(37)=1): Junction tree 0.633595 : efficient inference: 0.633595
p(s(38)=1): Junction tree 0.589604 : efficient inference: 0.589604
p(s(39)=1): Junction tree 0.423164 : efficient inference: 0.423164
p(s(40)=1): Junction tree 0.528234 : efficient inference: 0.528234

```

2. The point is that in computing the marginals $p(s_i)$, we can do this efficiently using

$$p(s_i) = \sum_{\text{pa}(s_i)} p(s_i | \text{pa}(s_i)) p(\text{pa}(s_i))$$

where the parent diseases $\text{pa}(s_i)$ are independent. Since there are only 3 parental diseases, we can compute these marginals by $2^3 = 8$ summations. In contrast, the Junction Tree has cliques of size 12, meaning that the complexity of carrying out absorption for the JT is at least 2^{12} , which is much more expensive. The point therefore is that the JT doesn't know that there is structure that can be exploited in the tables, and is therefore an upper bound on the complexity of inference.

3. The marginals $p(\text{disease}=1|\text{evidence})$:

```

p(d(1)=1|s(1:10))=0.0297756
p(d(2)=1|s(1:10))=0.38176
p(d(3)=1|s(1:10))=0.954235
p(d(4)=1|s(1:10))=0.396644
p(d(5)=1|s(1:10))=0.496467
p(d(6)=1|s(1:10))=0.435154
p(d(7)=1|s(1:10))=0.187487
p(d(8)=1|s(1:10))=0.701183
p(d(9)=1|s(1:10))=0.0431266
p(d(10)=1|s(1:10))=0.610313
p(d(11)=1|s(1:10))=0.287322
p(d(12)=1|s(1:10))=0.489833
p(d(13)=1|s(1:10))=0.8996
p(d(14)=1|s(1:10))=0.619565
p(d(15)=1|s(1:10))=0.920476
p(d(16)=1|s(1:10))=0.706096

```

```

p(d(17)=1|s(1:10))=0.201247
p(d(18)=1|s(1:10))=0.908494
p(d(19)=1|s(1:10))=0.864967
p(d(20)=1|s(1:10))=0.883929

```

- 6.10** 1. The JT consists of a single clique on all the variables (due to marrying all the x_1, \dots, x_T). The complexity of inference from the JT would suggest that this is $O(2^T)$.
2. If we sum first over y , the remaining distribution on x_1, \dots, x_T is simply a linear chain, for which inference of $p(x_T)$ is $O(T)$ time.

6.11 See `ShaferShenoy.m` and `demoJTreeShaferShenoy.m`.

```

function [jtmargpot jtmess]=ShaferShenoy(jtpot,infostruct)
%SHAFERSHENYOY Perform full round of Shafer Shenoy messages on a Junction Tree
% [jtmargpot jtmess]=ShaferShenoy(jtpot,infostruct)
%
% Perform (sum) messages for a JT specified with potentials jtpot. The distribution marginals are contained in
% the returned jtmargpot.
%
% infostruct is a structure with information about the Junction Tree:
% infostruct.cliquetree : connectivity structure of Junction Tree cliques
% infostruct.EliminationSchedule is a list of clique to clique
% eliminations. The root clique is contained in the last row.
% infostruct.ForwardOnly is optional. If this is set to 1 only the schedule contained
% in infostruct.EliminationSchedule is carried out. Otherwise both a
% forward and backward schedule is carried out.
Atree=infostruct.cliquetree; % connectivity structure of JT cliques
import brml.*
schedule=infostruct.EliminationSchedule;
ForwardOnly=0;
if isfield(infostruct,'ForwardOnly')
    ForwardOnly=infostruct.ForwardOnly;
end
if ~ForwardOnly
    reverseschedule=flipud(fliplr(schedule));
    schedule=vertcat(schedule,reverseschedule); % full round over all separators in both directions
end
for count=1:length(schedule)
    [elim neighb]=assign(schedule(count,:));
    if elim~=neighb
        incoming=setdiff(neighb,neigh(ATree,elim));
        if isempty(incoming)
            jtmess{elim,neighb} = sumpot(jtpot{neighb}, jtpot{neighb}.variables,0);
        else
            jtmess{elim,neighb} = sumpot(multipots([jtmess(incoming) jtpot{neighb}]), jtpot{neighb}.variables,0);
        end
    end
end
for c=1:length(jtpot)
    jtmargpot{c}=multipots(jtmess(neigh(infostruct.cliquetree,c),c));
end

```

- 6.12** It's easiest to consider a specific example, with the generalisation being straightforward. Let's consider a JT (using the obvious shorthand)

$$p(1, 2, 3, 4) = \frac{\phi(1, 2)\phi(2, 3)\phi(3, 4)}{\psi(2)\psi(3)}$$

1. Clique $\phi(1, 2)$ is on the 'left' edge, so let's eliminate this clique (this corresponds to clique 1 in the elimination sequence). To do this we perform

$$\sum_1 p(1, 2, 3, 4) = \sum_1 \frac{\phi(1, 2)\phi(2, 3)\phi(3, 4)}{\psi(2)\psi(3)} = \underbrace{\phi(2, 3) \sum_1 \frac{\phi(1, 2)}{\psi(2)}}_{\phi'(2, 3)} \frac{\phi(3, 4)}{\psi(3)} = p(2, 3, 4)$$

This means that the effect of summing over the 'edge' variables (in this case variable 1) can be represented as simply updating the edge potential of the new reduced tree, here the potential $\phi(2, 3)$.

Without loss of generality, we can write the original distribution as

$$p(1, 2, 3, 4) = p(1|2, 3, 4)p(2, 3, 4)$$

Because of the JT structure, variable 1 is connected to graph only through clique 1, which means that we have a conditional independence

$$p(1, 2, 3, 4) = p(1|2)p(2, 3, 4) = \frac{\phi(1, 2)}{\sum_1 \phi(1, 2)} p(2, 3, 4)$$

What this says is that we can represent the original distribution in the same JT form, in which $\phi(2, 3)$ is updated to $\phi'(2, 3)$ and the separator $\psi(1)$ is updated to $\sum_1 \phi(1, 2)$, with the effect that the part of the JT on the variables 2, 3, 4 represents the marginal distribution $p(2, 3, 4)$. Note that this is exactly the absorption procedure, since by defining the new separator

$$\psi'(2) = \sum_1 \phi(1, 2)$$

we update the potential

$$\phi'(2, 3) = \phi(2, 3) \frac{\psi'(2)}{\psi(2)}$$

which is the definition of the absorption procedure. An important observation is that the factor

$$\frac{\phi(1, 2)}{\psi'(2)}$$

is the conditional distribution $p(1|2)$, so that the absorption process can be thought of as updating the distribution so that we have a JT on a reduced set of variables $p(2, 3, 4)$, with the connection to the original distribution represented by $p(1|2)p(2, 3, 4)$, where the factor $p(1|2)$ is given by $\frac{\phi(1, 2)}{\sum_1 \phi(1, 2)}$.

2. We can now perform the same elimination of the edge clique again, applying the same idea now to $p(2, 3, 4)$. Since $p(2, 3, 4)$ is also a JT, exactly the same holds, except on a graph with a smaller number of variables. As the updating process for ϕ simply eliminates an edge clique on the JT by summation, the final clique $\phi'(3, 4)$ must contain the marginal $p(3, 4)$.

This forward pass updated (from left to right) the clique $\phi(2, 3)$ to $\phi'(2, 3)$ and then $\phi(3, 4)$ to $\phi'(3, 4)$ (and also the separators). For the reverse pass we will then update (from right to left) these new cliques, so that we'll get $\phi''(2, 3)$ and then $\phi'(1, 2)$ (as well as updating the separators).

3. For the first update in the reverse pass, absorbing from $\phi'(3, 4)$ to update $\phi'(2, 3)$, the new potential $\phi''(2, 3)$ is given by

$$\phi''(2, 3) = \frac{\phi'(2, 3)}{\psi'(3)} \sum_4 \phi'(3, 4) = p(2|3)p(3) = p(2, 3)$$

That is, the new clique $\phi''(2, 3)$ now contains the marginal $p(2, 3)$.

Similarly,

$$\phi'(1, 2) = \frac{\phi(1, 2)}{\psi'(2)} \sum_3 \phi''(2, 3) = p(1|2)p(2) = p(1, 2)$$

so that the new potential $\phi'(1, 2)$ is the marginal $p(1, 2)$.

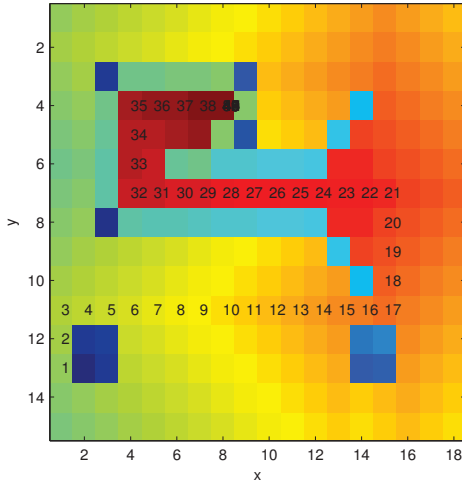
4. The above process shows that the absorption procedure has the effect that after a forward and then reverse pass, each clique will contain the marginal of the distribution over the variables in the clique. The consistency happens because the final distributions in each clique are the marginals of the same joint distribution. Hence, for example, $\sum_1 p(1, 2) = p(2) = \sum_3 p(2, 3)$.

7.1 The expected gain is

$$pS - (1 - p)S = (2p - 1)S$$

7.2 The probability that a salary s is larger than a random salary is given by the cumulative sum function since : $\int \mathbb{I}[s > s'] p(s') ds' = \int_{-\infty}^s p(s') ds'$. This gives rise to a 'sigmoidal' (s-shaped) curve. Repeating the experiment, provided we have a high salary, we would have an expected utility much lower than our salary and would therefore be not advised to take the bet. One can compute the salary at which one would be advised to take the bet and this will depend heavily on the shape of the salary profile. However, in general, the salary at which one would be advised to take the bet would be rather low typically. Intuitively, since there are only a few people with very high salaries, and human status intuition is typically about comparing ourselves to each others, we would only be advised to take the bet if we would expect to be better off than a randomly chosen person. If our existing salary is high, we are already better off than nearly everyone, so we wouldn't therefore be advised to take the bet. See also `demoMoneySalary.m`.

7.3 $\{test, drill\} < \{Seismic, Oil\}$ The difference is that this doesn't really make a great deal of sense from a real-world perspective! In figure 7.5, this corresponds to a scenario in which we first have to take a decision as to whether or not to undertake a seismic test. The outcome of the the test depends on the unknown amount of oil. Subsequently we decide to drill for oil (or not), knowing the result of the seismic test. However, in our new ID, with $\{test, drill\} < \{Seismic, Oil\}$, we lose any meaningful order about when to do the test, since we cannot distinguish between whether to do a seismic test first or drill for oil first.



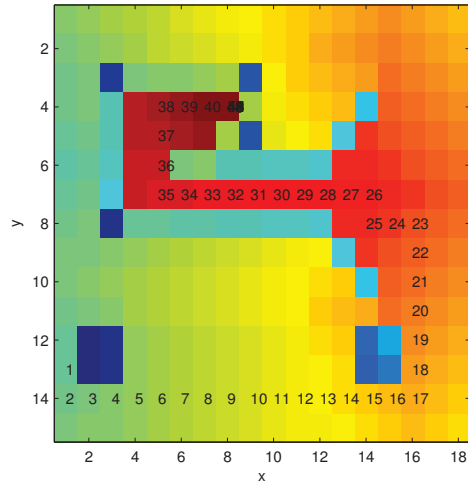
Optimal (x,y) position is:

```

1 13
1 12
1 11
2 11
3 11
4 11
5 11
6 11
7 11
8 11
9 11
10 11
11 11
12 11
13 11
14 11
15 11
15 10
15 9
15 8
15 7
14 7
13 7
12 7
11 7
10 7
9 7
8 7
7 7
6 7
5 7
4 7
4 6
4 5
4 4
5 4
6 4
7 4

```

See `demoMDPairplane.m`



Optimal (x,y) position is:

```

1 13
1 14
2 14
3 14
4 14
5 14
6 14
7 14
8 14
9 14
10 14
11 14
12 14
13 14
14 14
15 14
16 14
16 13
16 12
16 11
16 10
16 9
16 8
15 8
14 8
14 7
13 7
12 7
11 7
10 7
9 7
8 7
7 7
6 7
5 7
5 6
5 5
5 4
6 4
7 4
8 4

```

7.4

7.5 $U(d_1 = \text{no play}) = 0$. Utility of playing is the probability that we win p_1 times the amount we would win W_1 . There is a cost of C_1 to play.

- 7.6**
1. Replicate the same structure from the first stage to the second stage. Then include a link from x_1 to x_2 and from x_1 to d_2 .
 2. We need to compute the expected utility of the first decision not knowing whether or not we will win either of the first or both stages. If we win the first stage we assume that we will take the optimal decision d_2 to maximise the expected return. This gives (following the course notes):

$$U(d_1) = \sum_{w_1} \mathbb{I}[w_1 = \text{win}] p(w_1|d_1) \max_{d_2} \left(\sum_{w_2} p(w_2|w_1, d_2) u(w_2) - c(d_2) \right) + \sum_{w_1} p(w_1|d_1) u_1(w_1) - c(d_1) \quad (\text{B.0.63})$$

The term $\sum_{w_1} p(w_1|d_1) u_1(w_1) - c(d_1)$ is just the first stage which for $d_1 = \text{play}$ contributes $p_1 W_1 - C_1$. The second stage contribution is, assuming we win the first stage:

$$p_1 \max_{d_2} \left(\sum_{w_2} p(w_2|w_1, d_2) u(w_2) - c(d_2) \right) \quad (\text{B.0.64})$$

If $d_2 = \text{play}$ this is

$$p_1 (p_2 W_2 - C_2) \quad (\text{B.0.65})$$

else if $d_2 = \text{no play}$ this is 0. Hence provided $p_2 W_2 - C_2 \geq 0$ we have $p_1 (p_2 W_2 - C_2)$, otherwise 0. Using these two conditions and adding on the first stage expected utility, we arrive at the answer.

7.7 1.

$$U(\text{bet}) = p_w W + (1 - p_w)L, \quad U(\text{no bet}) = B \quad (\text{B.0.66})$$

2.

$$U_{\text{gain}}(\text{bet}) = p_w(W - B) - (1 - p_w)(B - L) = p_w W - (B - L + p_w L) = p_w W - B + L - p_w L \quad (\text{B.0.67})$$

$$U_{\text{gain}}(\text{nobet}) = 0 \quad (\text{B.0.68})$$

3.

$$U_{\text{gain}}(\text{bet}) - U_{\text{gain}}(\text{nobet}) = p_w W - B + L - p_w L \quad (\text{B.0.69})$$

$$U(\text{bet}) - U(\text{no bet}) = p_w W + (1 - p_w)L - B \quad (\text{B.0.70})$$

which gives the required result.

7.8 EM fails since the energy in the M-step is negative infinity until $\theta_{\text{new}} = \theta_{\text{old}}$ in which case the energy is zero.

7.10 See IDjensen.m.

7.11 In a POMDP, we need to sum first over all the unobserved $h_{1:T}$. This creates a single large clique that contains all the other variables. From this perspective, the complexity of exact inference in the POMDP is exponential in T .

7.12 $(abc)- > (bcde)- > (bdei) < -(degi) < -(dgh) < -(fdh)$

7.13 See exerciseInvest.m and optdec.m. See also demoInvest.m. The answer is that one should place 25% of one's wealth in asset A.

```
function exerciseInvest
import brml.*
pars.WealthValue=0:0.2:5; % these are the possible states that wealth can take

pars.epsilonAval=[0 0.01]; % these are the possible states that price A change by
pars.epsilonBval=[-0.12 0 0.15]; % these are the possible states that price A change by
% for example, if we use the state 2 of epsilonAval, the price for A at the next timestep will
% be (1+0.01) times the current value. Similarly, if we use state 1 for B,
% B's new price will be (1-0.12) times the current price.

pars.DecisionValue=[1:-0.25:0]; % these are the decision values (order this way so that for two deci

% transition matrices:
% asset A
pars.epsilonAtran=condp(ones(length(pars.epsilonAval)));

% asset B
for st=1:length(pars.epsilonBval)
    for stp=1:length(pars.epsilonBval)
        epsilonBtran(stp,st)=exp(- 10*(pars.epsilonBval(stp)-pars.epsilonBval(st)).^2);
    end
end
pars.epsilonBtran=condp(epsilonBtran);

epsilonA(1)=1; epsilonB(1)=1; w(1)=1; % initial states

T=40;
desired=1.5;
[dec val]=optdec(epsilonA(1),epsilonB(1),desired,T,w(1),pars);
fprintf('optimally place %f of wealth in asset A, giving expected wealth at time %d of %f\n',pars.Dec

function [d1 val]=optdec(epsilonA1,epsilonB1,desired,T,w1,pars)
import brml.*
epsilonAtran=pars.epsilonAtran;
```

```

epsilonBtran=pars.epsilonBtran;
epsilonAval=pars.epsilonAval;
epsilonBval=pars.epsilonBval;
DecisionValue=pars.DecisionValue;
WealthValue=pars.WealthValue;

w(1)=w1;

% dynamic programming:
[vdAt,vdAtm,vdBt,vdBtm,vwtm,vdtm]=assign(1:6);
Atran=array([vdAt,vdAtm],epsilonAtran);
Btran=array([vdBt,vdBtm],epsilonBtran);
pot=array([vdAt vdBt vwtm vdtm]);
for t=T:-1:3
    for wtm=1:length(WealthValue)
        for dtm=1:length(DecisionValue)
            for dAt=1:length(epsilonAval)
                for dBt=1:length(epsilonBval)
                    DecisionValuetm=DecisionValue(dtm);
                    wtval=WealthValue(wtm)*(DecisionValuetm*(1+epsilonAval(dAt))+(1-DecisionValuetm)*(1+epsilonBval(dBt)));
                    wt=val2state(wtval,WealthValue); % get the nearest state corresponding to this value
                    if t==T
                        tmp=ufun(WealthValue(wt),desired);
                    else
                        tmp=gamma(t).table(dAt,dBt,wt);
                    end
                    pot.table(dAt,dBt,wtm,dtm)=tmp;
                end
            end
        end
    end
    gamma(t-1)=maxpot(sumpot(multipots([Atran Btran pot])),[vdAt vdBt]),vdtm);
end

clear pot
pot=array([vdAt vdBt vdtm]);
wtm=val2state(w(1),WealthValue);
for dtm=1:length(DecisionValue)
    for dAt=1:length(epsilonAval)
        for dBt=1:length(epsilonBval)
            DecisionValuetm=DecisionValue(dtm);
            wtval=WealthValue(wtm)*(DecisionValuetm*(1+epsilonAval(dAt))+(1-DecisionValuetm)*(1+epsilonBval(dBt)));
            wt=val2state(wtval,WealthValue); % get the nearest state corresponding to this value
            pot.table(dAt,dBt,dtm)=gamma(2).table(dAt,dBt,wt);
        end
    end
end
U=sumpot(multipots([Atran Btran pot]),[vdAt vdBt]);
Ud1=setpot(U,[vdAtm vdBtm],[epsilonA1 epsilonB1]);
d1=argmax(Ud1.table); val=max(Ud1.table);

function u=ufun(w,desired)
u=real(w>desired);

function s=val2state(x,prices)
s=brml.argmax(abs(prices-x));

```

7.14. **Bayesian Approach** You should switch to the other cheque. Even though you have an equal chance of having the higher valued cheque, it is better (on average) to swap. To see this, consider the possible scenarios we could be in:

1.
 - Cheque A: 100
 - Cheque B: 50

2.
Cheque A: 50
Cheque B: 100

3.
Cheque A: 100
Cheque B: 200

4.
Cheque A: 200
Cheque B: 100

We can be in any of the 4 above scenarios with equal probability $1/4$. If we stay with £100, then clearly our expected value is £100. If we switch, our expected value is $50/4 + 50/4 + 200/4 + 200/4 = £125$. It's therefore better to switch, even if we don't look at the value of the cheque we first select.

Bayesian Approach The above reasoning seems correct, particularly if we think of, not a physical quantity of money, but an abstract quantity. For example, a computer has a 1 and 2 symbol. It will select at random (without replacement) one of the two symbols and place this in position A , placing the other non-selected symbol in position B . If position A contains 2, then the value in both A and B is divided by 2; otherwise there is no change. Tom is then given 'cheque' A and asked if he wishes to switch to 'cheque' B . In this setup, I think the above reasoning is fine.

However, the actual question involves a 'physical' amount of money. In this sense we need to think about a mechanism that could write cheques for certain quantities. Since in any physical system, there must be a largest possible amount, say M , then we can generate cheques by first sampling from a distribution $p(\text{cheque value} | \text{cheque} = \text{high amount})$. Given this, we generate the low value cheque by halving this amount. We then randomly give Tom one of these two cheques. Clearly, if $M = £100$, then Tom knows that he must have the high value cheque, and he should stick with this one. If we use H to denote the high value cheque and L the low value cheque, and V the observed value, then in general we wish to compute

$$p(C = H|V) = \frac{p(V|C = H)p(C = H)}{p(V|C = H)p(C = H) + p(V|C = L)p(C = L)}$$

Assuming that the cheques are randomly given then $p(C = H) = p(C = L) = 1/2$. Also

$$p(V|C = L) = \begin{cases} 2p(V|C = H) & V \leq M/2 \\ 0 & V > M/2 \end{cases}$$

If we assume for simplicity that we write the high value cheque as a natural number amount uniformly between 1 and M , then $p(V|C = H) = 1/M$ and

$$p(C = H|V) = \frac{1}{1 + 2\mathbb{I}[V \leq M/2]}$$

The expected utility of sticking with the £100 check is 100. The expected utility of switching to the other cheque is

$$200p(C = L|V) + 50p(C = H|V)$$

Since $p(C = L|V) = 1 - p(C = H|V)$, this gives the utility of switching as

$$200 - \frac{150}{1 + 2\mathbb{I}[V \leq M/2]}$$

In the non-Bayesian (non-physical) approach, we effectively assumed $M = \infty$, in which case, we recover the expected value of £125. More generally, we should switch if the observed value is less than half the assumed maximum value M .

For the second part, effectively we don't know the value of V , so in the above calculations, we simply replace $p(C = H|V)$ by $p(C = H) = 1/2$. In this case, we are back to same reasoning as in the non-Bayesian approach and we should switch since the expected utility of switching will be $5/4$ times the utility of sticking. Note that this doesn't mean that we should continually switch, exponentially increasing our expected utility. The expected utility of switching twice is the same as the expected utility of sticking.

7.15 1. One way to write this is to use a Decision Tree. For the DT, the branch $a \rightarrow b$ has expected cost

$$\begin{aligned} & p \min(C_{ab} + C_{bd}, C_{ab} + C_{bc} + C_{cd}) + (1 - p)(C_{ab} + C_{bc} + C_{cd}) \\ &= C_{ab} + p \min(C_{bd}, C_{bc} + C_{cd}) + (1 - p)(C_{bc} + C_{cd}) \\ &= C_{ab} + \min(pC_{bd} + (1 - p)(C_{bc} + C_{cd}), C_{bc} + C_{cd}) \end{aligned}$$

The branch $a \rightarrow d$ has expected cost C_{ad} . We take the $a \rightarrow b$ branch if it has lower expected cost than C_{ad} .

2. The 3 paths have the following expected costs:

$$\begin{aligned} a \rightarrow d &= C_{ad} \\ a \rightarrow b \rightarrow d &= C_{ab} + pC_{bd} \\ a \rightarrow b \rightarrow c \rightarrow d &= C_{ab} + C_{bc} + C_{cd} \end{aligned}$$

and we would therefore take the $a \rightarrow b$ path if

$$C_{ab} + \min(pC_{bd}, C_{bc} + C_{cd}) < C_{ad}$$

This will in general give a different decision to the one in part (1). The reason for this difference is that in this second calculation we are not considering that the information about whether the road $b \rightarrow d$ will be open will be revealed to us when we arrive at b . This is therefore an overoptimistic situation. If the cost C_{bd} is very low, we could end up taking the route $a \rightarrow b$ since we are 'hoping' (with probability p) that the route will be open. If it's not, we have to consider that we will have to take an alternative route.

7.16 1.

$$\begin{aligned}
\max_x f(x, y) &\geq f(x, y) \\
\sum_y \max_x f(x, y) &\geq \sum_y f(x, y) \\
\max_x \sum_y \max_x f(x, y) &\geq \max_x \sum_y f(x, y) \\
\sum_y \max_x f(x, y) &\geq \max_x \sum_y f(x, y)
\end{aligned}$$

2. Standard dynamic programming gives the Bellman updates:

$$\gamma_{t-1}(s_{t-1}) = \max_{a_{t-1}} \sum_{s_t} p(s_t | s_{t-1}, a_{t-1}) \gamma_t(s_t)$$

3. Let's look at P^* . This is

$$\begin{aligned}
P^*(s_1) &= \dots \sum_{s_{T-1}} \max_{a_{T-1}} p(s_{2:T-1}, s_T = 1 | a_{1:T-1}, s_1) \\
&\geq \dots \max_{a_{T-1}} \sum_{s_{T-1}} p(s_{2:T-1}, s_T = 1 | a_{1:T-1}, s_1) \\
&= \dots \sum_{s_{T-2}} \max_{a_{T-2}} \max_{a_{T-1}} p(s_{2:T-2}, s_T = 1 | a_{1:T-1}, s_1) \\
&\geq \dots \max_{a_{T-2}} \max_{a_{T-1}} p(s_{2:T-3}, s_T = 1 | a_{1:T-1}, s_1)
\end{aligned}$$

We can thus repeatedly switch the sum and max to give eventually the expression $P^{**}(s_1)$. Intuitively, P^* is higher since at each stage we have more information, namely the state s_t after the action a_{t-1} is taken, enabling us to take a more optimal decision.

7.17 This is a simple dynamic programming problem that follows largely the same structure as part 2 of the previous exercise, except that the utility is to be, not in state 1, but either one of two possible specified states. The answer is 0.7709 for up and down and 0.7690 for same. See HitTheTarget.m

8.1 The Professor is wrong (in principle). Consider a highly non-symmetric distribution with many (slightly) above average values and a few extremely low values. A simple demonstration is to assume that the average IQ is 100, and the minimum 0. If there are only two possible scores, the above average score, and the below average score, then one can show that 90 percent of people can indeed have an above average IQ if the above average IQ score is less than 111.111.

8.2 The distribution is symmetric around $(0, 0)$, so the mean is therefore 0. Furthermore, x and y are uncorrelated since the distribution is spherically symmetric (isotropic), so that $\langle xy \rangle = 0$. They are nevertheless dependent since

$$p(x|y) \propto (x^2 + y^2)^2 e^{-x^2}$$

so that the distribution of x depends on y^2 (and vice versa).

8.3 If we write the states as the real values 0, 1, then the question is equivalent to the distribution of the new variable

$$y = \sum_{i=1}^n x_i \tag{B.0.71}$$

Note that we can write $p(x_i) = \theta^{x_i} (1 - \theta)^{1-x_i}$. Formally,

$$\begin{aligned}
p(y = k) &= \sum_x p(y = k | x) p(x) = \sum_x \delta \left(k - \sum_{i=1}^n x_i \right) \prod_i p(x_i) \\
&= \sum_x \delta \left(k - \sum_{i=1}^n x_i \right) \prod_i \theta^{x_i} (1 - \theta)^{1-x_i} \\
&= \sum_x \delta \left(k - \sum_{j=1}^n x_j \right) \theta^{\sum_i x_i} (1 - \theta)^{n - \sum_i x_i} \\
&= \binom{n}{k} \theta^k (1 - \theta)^{n-k}
\end{aligned}$$

The factor $\binom{n}{k}$ arises since we are summing over all the configurations in which k of the n variables are in state 1.

8.4

$$I^2 = \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2 + y^2} dx dy \tag{B.0.72}$$

Using Polar coordinates, $r^2 = x^2 + y^2$, $\theta = \arccos(x/r)$, the Jacobian is $r dr d\theta$. Hence

$$I^2 = \int_{\theta=0}^{2\pi} \int_{r=0}^{\infty} e^{-\frac{1}{2}r^2} r dr d\theta = -2\pi e^{-\frac{1}{2}r^2} \Big|_0^{\infty} = 2\pi \tag{B.0.73}$$

For part (ii), just use the change of variables $z = \frac{x - \mu}{\sigma}$.

8.5 1.

$$\begin{aligned}
 \langle x \rangle_{\mathcal{N}(x|\mu, \sigma^2)} &= \frac{1}{\sqrt{2\pi\sigma^2}} \int x e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx \\
 &= \frac{1}{\sqrt{2\pi\sigma^2}} \int (x - \mu + \mu) e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx \\
 &= \frac{1}{\sqrt{2\pi\sigma^2}} \int (x - \mu) e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx + \mu \frac{1}{\sqrt{2\pi\sigma^2}} \int e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx \\
 &= \mu \frac{1}{\sqrt{2\pi\sigma^2}} \int e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx = \mu
 \end{aligned}$$

2.

$$\langle (x - \mu)^2 \rangle_{\mathcal{N}(x|\mu, \sigma^2)} = \frac{1}{\sqrt{2\pi\sigma^2}} \int (x - \mu)^2 e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx$$

Changing variables, to $z \equiv (x - \mu)/\sigma$, we have

$$\begin{aligned}
 \langle (x - \mu)^2 \rangle_{\mathcal{N}(x|\mu, \sigma^2)} &= \frac{\sigma^2}{\sqrt{2\pi\sigma^2}} \int z^2 e^{-\frac{1}{2}z^2} \sigma dz \\
 &= \sigma^2 \frac{1}{\sqrt{2\pi}} \int z^2 e^{-\frac{1}{2}z^2} dz
 \end{aligned}$$

The latter integral can be readily shown to be 1 by integration by parts.

8.6 Follows from

$$\Sigma = \langle \mathbf{x}\mathbf{x}^\top \rangle - \mu\mu^\top$$

8.7

$$\begin{aligned}
 \int_{\theta_j} \text{Dirichlet}(\theta|\mathbf{u}) &= \frac{1}{Z(\mathbf{u})} \int_{\theta_j} \prod_i \theta_i^{u_i-1} \\
 &= \frac{1}{Z(\mathbf{u})} \left[\prod_{i \neq j} \theta_i^{u_i-1} \right] \int_{\theta_j} \theta_j^{u_j-1} \\
 &= \frac{1}{Z(\mathbf{u})} Z(u_j) \prod_{i \neq j} \theta_i^{u_i-1} \\
 &= \text{Dirichlet}(\theta_{\setminus j} | \mathbf{u}_{\setminus j})
 \end{aligned}$$

8.8

$$\langle x^k \rangle = \frac{1}{B(\alpha, \beta)} \int x^k x^{\alpha-1} (1-x)^{\beta-1} = \frac{B(\alpha+k, \beta)}{B(\alpha, \beta)}$$

Since

$$\begin{aligned}
 B(\alpha+k, \beta) &= \frac{\Gamma(\alpha+k)\Gamma(\beta)}{\Gamma(\alpha+k+\beta)} = \frac{(\alpha+k)\Gamma(\alpha+k-1)\Gamma(\beta)}{(\alpha+\beta+k)\Gamma(\alpha+k-1+\beta)} = \frac{(\alpha+k)(\alpha+k-1)\dots(\alpha)\Gamma(\alpha)\Gamma(\beta)}{(\alpha+\beta+k)(\alpha+\beta+k-1)\dots(\alpha+\beta)\Gamma(\alpha+\beta)} \\
 \langle x^k \rangle &= \frac{(\alpha+k)(\alpha+k-1)\dots(\alpha)}{(\alpha+\beta+k)(\alpha+\beta+k-1)\dots(\alpha+\beta)}
 \end{aligned}$$

8.9

$$\frac{d}{dt} g(t) = \int p(x) \frac{d e^{tx}}{dt} = \int p(x) x e^{tx}$$

Similarly,

$$\frac{d^k}{dt^k} g(t) = \int p(x) x^k e^{tx}$$

Taking then the limit $t \rightarrow 0$, we obtain the required result.

8.10 Using Bayes' rule as

$$\begin{aligned}
 p(y) &= \int p(y|x) p(x) dx \\
 &= \int \delta(y - f(x)) p(x) dx \\
 &= \int \delta(y - f(x)) p(x) \frac{dx}{df} df \\
 &= \int \delta(y - f(x)) p(x) \frac{dx}{df} df \\
 &= p(x) \left(\frac{df}{dx} \right)^{-1}, \quad x = f^{-1}(y)
 \end{aligned}$$

where $f^{-1}(f(x)) = x$.

8.11 The Jacobian is $\det(\Sigma^{-\frac{1}{2}})$. Hence

$$I = \det(\Sigma^{\frac{1}{2}}) \int_{-\infty}^{\infty} e^{-\frac{1}{2}(z)^T z} dz \quad (\text{B.0.74})$$

Since $z^T z = \sum_i w_i^2$, the integral splits into a product of one-dimensional integrals

$$I = \det(\Sigma^{\frac{1}{2}}) \prod_i \int_{-\infty}^{\infty} e^{-\frac{1}{2}z_i^2} dz_i = \det((2\pi)^{\frac{1}{2}} \Sigma^{\frac{1}{2}}) = \sqrt{\det(2\pi\Sigma)}. \quad (\text{B.0.75})$$

8.13 The skewness is clearly zero since the Gaussian is symmetric. For the kurtosis, w.l.o.g we can consider a zero mean Gaussian, for which

$$\langle x^4 \rangle = \frac{1}{\sqrt{2\pi\sigma^2}} \int x^4 e^{-\frac{1}{2\sigma^2}x^2} dx = \sigma^4 \frac{1}{\sqrt{2\pi}} \int z^4 e^{-\frac{1}{2}z^2} dz$$

Using integration by parts, with parts xz^3 and $ze^{-\frac{1}{2}z^2} dz$, this becomes

$$\langle x^4 \rangle = \frac{\sigma^4}{\sqrt{2\pi}} \left(-z^2 e^{-\frac{1}{2}z^2} \Big|_{-\infty}^{\infty} + 3 \int z^2 e^{-\frac{1}{2}z^2} dz \right) = 3\sigma^4$$

Using the definition of kurtosis, we obtain the required result.

8.14 Let's first find the probability there is no event in the interval, which is $1 - \gamma$, where $\gamma = \theta\delta t$. Then the probability there is no event in all intervals is

$$(1 - \gamma)^{t/\delta t} = (1 - \theta\delta t)^{t/\delta t} \quad (\text{B.0.76})$$

In the limit $\delta t \rightarrow 0$, this becomes

$$e^{-\theta t} \quad (\text{B.0.77})$$

Hence the probability that there is at least one event is $1 - e^{-\theta t}$.

8.15 1. The joint distribution factorises since $e^{\sum_i \theta_i \phi_i(x_i)} = \prod_i e^{\theta_i \phi_i(x_i)}$, and therefore all x_i are independent.

2. Defining $Z_i = \int e^{\theta_i \phi_i(x_i)} dx_i$, then $Z = \prod_i Z_i$.

3. Using the Jacobian, the distribution of \mathbf{y} is given by

$$p(\mathbf{y}) = \det(\mathbf{M}) \prod_i e^{\theta_i \phi_i(\sum_j M_{ij} y_j)}$$

This is clearly non-factorised for a general \mathbf{M} . However, this is a tractable distribution since the correlations in \mathbf{y} simply appear because of a linear coordinate transformation.

8.16

$$m = \frac{\alpha}{\alpha + \beta} \rightarrow \alpha = \beta\gamma, \gamma = m/(1 - m) \quad (\text{B.0.78})$$

Substitute this in the equation for the variance and solve for β .

8.23 For $\gamma \rightarrow \infty$, we have $\tilde{a} = N, \tilde{b} = \sum_n x^n, \tilde{c} = \sum_n (x^n)^2$. As $\beta \rightarrow \infty$, $1/\tilde{\beta} = N\sigma^2/2$. As a function of λ the posterior is

$$-\lambda \left(\frac{1}{\tilde{a}} \left(\mu - \tilde{b}/\tilde{a} \right)^2 + 1/\tilde{\beta} \right) + (\alpha + N/2 - 1/2) \log \lambda \quad (\text{B.0.79})$$

Differentiating with respect to λ and equating to zero, we have that optimally $1\lambda = N\sigma^2/(2\alpha + N - 1)$. Setting $\alpha = 1/2$ gives then the standard result for the variance. The maximum likelihood mean can be obtained in a similar way by optimising with respect to μ , which gives $\mu = \tilde{b}/\tilde{a}$ which is $\sum_n x^n/N$ in the given limits.

8.24 In the expressed limits, $\tilde{b} = \sum_n x^n$, $\tilde{a} = N$, and $\tilde{c} = \sum_n (x^n)^2$. In this case

$$\tilde{c} - \frac{\tilde{b}^2}{\tilde{a}} = \sum_n (x^n)^2 - \frac{(\sum_n x^n)^2}{N} = N\sigma_{ML}^2 \quad (\text{B.0.80})$$

where $\sigma_{ML}^2 = \frac{1}{N} \sum_n (x^n - \bar{x})^2$, $\bar{x} = \frac{1}{N} \sum_n x^n$.

The optimal setting for μ is clearly given at $\mu_* = \tilde{b}/\tilde{a} = \bar{x}$. Using this setting for μ , the log posterior distribution, ignoring constants, is

$$\frac{1}{2} \log \lambda + (\alpha + N/2 - 1) \log \lambda - \lambda/\tilde{\beta} \quad (\text{B.0.81})$$

Differentiating and equating to zero we have the optimal solution given by

$$\frac{1}{\lambda_*} = \frac{1}{\tilde{\beta}} + \frac{1}{\alpha - 1 + \frac{N+1}{2}} \quad (\text{B.0.82})$$

In the limits given, $1/\tilde{\beta} = \frac{1}{2}N\sigma_{ML}^2$, hence

$$\sigma_*^2 = \frac{N}{2\alpha - 2 + N + 1} \sigma_{ML}^2 \quad (\text{B.0.83})$$

which for $\alpha = 1$ gives the ‘unbiased’ estimator of the variance².

8.25

$$p(\mu|\mathcal{X}) = \frac{1}{\Gamma(\alpha)\beta^\alpha} \sqrt{\frac{N'}{2\pi}} \int_{\lambda} \lambda^{\frac{1}{2}} e^{-\frac{\lambda N'}{2}(\mu - \hat{m})^2} \lambda^{\alpha-1} e^{-\lambda/\beta} \quad (\text{B.0.84})$$

where

$$\hat{m} \equiv \tilde{a}/\tilde{b}, \quad N' \equiv (1 + \tilde{a}) \quad (\text{B.0.85})$$

Collecting the polynomial terms in λ together, and collecting the exponential terms in λ together, this becomes proportional to a Gamma distribution. Using the fact that the normalisation constant of a Gamma distribution is $\Gamma(\alpha)\beta^\alpha$ we have

$$p(\mu|\mathcal{X}) = \frac{\Gamma(\alpha + \frac{1}{2})}{\Gamma(\alpha)\beta^\alpha} \sqrt{\frac{N'}{2\pi}} \left[\frac{1}{\beta} + \frac{N'}{2}(\mu - \hat{m})^2 \right]^{-\alpha - \frac{1}{2}} \quad (\text{B.0.86})$$

which is a Student’s t -distribution for suitably chosen parameters. This is easily seen by taking out a factor of $1/\beta$ to give

$$p(\mu|\mathcal{X}) = \frac{\Gamma(\alpha + \frac{1}{2})}{\Gamma(\alpha)} \sqrt{\frac{\beta N'}{2\pi}} \left[1 + \frac{\beta N'}{2}(\mu - \hat{m})^2 \right]^{-\alpha - \frac{1}{2}} \quad (\text{B.0.87})$$

which is a Student’s t -distribution, definition(8.26) with mean \hat{m} , degrees of freedom 2α and ‘scale’ $N'\beta\alpha$.

8.29 1.

$$p(\tau|\mathcal{X}) \propto p(\mathcal{X}|\tau)p(\tau) = p(\tau) \prod_n p(x^n|\tau) \propto \tau^{a-1} e^{-b\tau} \prod_n \left[\tau^{1/2} e^{-\frac{\tau}{2}(x^n - \mu)^2} \right]$$

Gathering terms in the exponent establishes the result.

2.

$$\int_{\tau} e^{-\frac{\tau}{2}(x - \mu)^2} \tau^{a'-1} e^{-b'\tau}$$

3. We can appeal directly to equation (8.3.28) to establish the result.

8.35 Ignoring the $1/2$ factors, the first two terms give

$$\left(\Sigma_1^{-1} \mu_1 + \Sigma_2^{-1} \mu_2 \right)^\top \left(\Sigma_1 (\Sigma_1 + \Sigma_2)^{-1} \Sigma_2 \right) \left(\Sigma_1^{-1} \mu_1 + \Sigma_2^{-1} \mu_2 \right) - \left(\mu_1^\top \Sigma_1^{-1} \mu_1 + \mu_2^\top \Sigma_2^{-1} \mu_2 \right)$$

which is

$$\left(\Sigma_1^{-1} \mu_1 + \Sigma_2^{-1} \mu_2 \right)^\top \left(\Sigma_1 (\Sigma_1 + \Sigma_2)^{-1} \Sigma_2 \right) \left(\Sigma_1^{-1} \mu_1 + \Sigma_2^{-1} \mu_2 \right) - \left(\mu_1^\top \Sigma_1^{-1} \mu_1 + \mu_2^\top \Sigma_2^{-1} \mu_2 \right)$$

Defining $\mathbf{S} = (\Sigma_1 + \Sigma_2)$,

$$\begin{aligned} & \left(\Sigma_1^{-1} \mu_1 + \Sigma_2^{-1} \mu_2 \right)^\top \left(\Sigma_2 \mathbf{S}^{-1} \mu_1 + \Sigma_1 \mathbf{S}^{-1} \mu_2 \right) - \left(\mu_1^\top \Sigma_1^{-1} \mu_1 + \mu_2^\top \Sigma_2^{-1} \mu_2 \right) \\ & \mu_1^\top \left(\Sigma_1^{-1} \Sigma_2 \mathbf{S}^{-1} \mu_1 + \mathbf{S}^{-1} \mu_2 \right) + \mu_2^\top \left(\Sigma_2^{-1} \Sigma_1 \mathbf{S}^{-1} \mu_2 + \mathbf{S}^{-1} \mu_1 \right) - \left(\mu_1^\top \Sigma_1^{-1} \mu_1 + \mu_2^\top \Sigma_2^{-1} \mu_2 \right) \\ & = \mu_1^\top \left(\Sigma_1^{-1} \Sigma_2 \mathbf{S}^{-1} - \Sigma_1^{-1} \right) \mu_1 + \mu_2^\top \left(\Sigma_2^{-1} \Sigma_1 \mathbf{S}^{-1} - \Sigma_2^{-1} \right) \mu_2 + 2\mu_1^\top \mathbf{S}^{-1} \mu_2 \\ & 2\mu_1^\top \mathbf{S}^{-1} \mu_2 - \mu_1^\top \mathbf{S}^{-1} \mu_1 - \mu_2^\top \mathbf{S}^{-1} \mu_2 \end{aligned} \quad (\text{B.0.88})$$

²For readers familiar with David MacKay’s Bayesian setting of the mean and variance of a Gaussian, here for conjugacy we constrain the prior mean $\sigma_0^2 = \gamma\sigma^2$, which leads to a slightly different treatment than presented in [197].

Note that the form equation (3) only holds provided that all quantities are well defined. This is not the case for example if we want to find the product of two Gaussians on different spaces

$$\mathcal{N}(x_1 | \mu_1, \sigma_1^2) \mathcal{N}(x_2 | \mu_2, \sigma_2^2)$$

Defining $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, we can then write the exponent for this as proportional to

$$(\mathbf{x} - \tilde{\boldsymbol{\mu}}_1)^\top \underbrace{\begin{pmatrix} \sigma_1^{-2} & 0 \\ 0 & 0 \end{pmatrix}}_{\tilde{\boldsymbol{\Sigma}}_1^{-1}} (\mathbf{x} - \tilde{\boldsymbol{\mu}}_1) + (\mathbf{x} - \tilde{\boldsymbol{\mu}}_2)^\top \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & \sigma_2^{-2} \end{pmatrix}}_{\tilde{\boldsymbol{\Sigma}}_2^{-1}} (\mathbf{x} - \tilde{\boldsymbol{\mu}}_2)$$

where

$$\tilde{\boldsymbol{\mu}}_1 = \begin{pmatrix} \mu_1 \\ 0 \end{pmatrix}, \quad \tilde{\boldsymbol{\mu}}_2 = \begin{pmatrix} 0 \\ \mu_2 \end{pmatrix}$$

However in this case the covariance in \mathbf{x} is not well defined. A general expression that works in this case is therefore to define the matrices

$$\tilde{\boldsymbol{\Sigma}}_1^{-1}, \tilde{\boldsymbol{\Sigma}}_2^{-1}$$

which may contain zeros on their diagonals, and have the $\boldsymbol{\Sigma}_i^{-1}$ embedded within them, as above. Then defining $\mathbf{A} = \tilde{\boldsymbol{\Sigma}}_1^{-1} + \tilde{\boldsymbol{\Sigma}}_2^{-1}$ and $\mathbf{b} = \tilde{\boldsymbol{\Sigma}}_1^{-1} \tilde{\boldsymbol{\mu}}_1 + \tilde{\boldsymbol{\Sigma}}_2^{-1} \tilde{\boldsymbol{\mu}}_2$, the product is a Gaussian with mean $\mathbf{A}^{-1} \mathbf{b}$ and covariance \mathbf{A}^{-1} with log prefactor

$$\frac{1}{2} \mathbf{b}^\top \mathbf{A}^{-1} \mathbf{b} - \frac{1}{2} (\boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}_1^{-1} \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^\top \boldsymbol{\Sigma}_2^{-1} \boldsymbol{\mu}_2) - \frac{1}{2} \log \det(2\pi \boldsymbol{\Sigma}_1) \det(2\pi \boldsymbol{\Sigma}_2) - \frac{1}{2} \log \det(\mathbf{A}/(2\pi))$$

8.41 Defining $\Delta x^n \equiv x^n - \langle x \rangle$, $\Delta y^n \equiv y^n - \langle y \rangle$, one can first show that

$$\rho_{x,z} = \frac{\sqrt{N}}{\sqrt{\sum_{n=1}^N (\Delta x^n + \Delta y^n)^2}} (\sigma_x + \rho_{x,y} \sigma_y) \quad (\text{B.0.89})$$

Then use the triangle inequality:

$$\frac{1}{\sqrt{N}} \sqrt{\sum_{n=1}^N (\Delta x^n + \Delta y^n)^2} \leq \sigma_x + \sigma_y \quad (\text{B.0.90})$$

which gives

$$\rho_{x,z} \geq \frac{\sigma_x + \sigma_y \rho_{x,y}}{\sigma_x + \sigma_y} \quad (\text{B.0.91})$$

Defining $0 \leq p \equiv \sigma_x / (\sigma_x + \sigma_y) \leq 1$ then

$$\rho_{x,z} \geq \rho_{x,y} + p(1 - \rho_{x,y}) \quad (\text{B.0.92})$$

Since $-1 \leq \rho_{x,y} \leq 1$, and $p > 0$ we must have $p(1 - \rho_{x,y}) \geq 0$, giving the desired result. The result is ‘geometrically obvious’ since if we take two vectors \mathbf{x} and \mathbf{y} then $\mathbf{z} = \mathbf{x} + \mathbf{y}$ must be closer in angle to \mathbf{x} than \mathbf{y} is to \mathbf{x} .

9.1 See `demoMLprinter.m`. For the final part, the symptom variables for which no evidence is known simply disappear under marginalisation (since they are nodes on the graph with no children), leaving a graph with the same structure but on a reduced set of variables. One can then set the evidential states of this new distribution and carry out max-absorption.

Part 1:

```
p(fuse):
fuse =0  4/5
fuse =1  1/5
```

```
-----
p(drum):
drum =0  11/15
drum =1  4/15
```

```
-----
p(toner):
toner =0  2/3
toner =1  1/3
```

```
-----
p(poorpaper):
poorpaper =0  7/15
poorpaper =1  8/15
```

```

-----
p(roller):
roller =0 4/5
roller =1 1/5

```

```

-----
p(burning| fuse):
burning =0 fuse =0 1
burning =1 fuse =0 0
burning =0 fuse =1 1/3
burning =1 fuse =1 2/3

```

```

-----
p(poorprint| drum toner poorpaper):
poorprint =0 drum =0 toner =0 poorpaper =0 1
poorprint =1 drum =0 toner =0 poorpaper =0 0
poorprint =0 drum =1 toner =0 poorpaper =0 0
poorprint =1 drum =1 toner =0 poorpaper =0 1
poorprint =0 drum =0 toner =1 poorpaper =0 0
poorprint =1 drum =0 toner =1 poorpaper =0 1
poorprint =0 drum =1 toner =1 poorpaper =0 0
poorprint =1 drum =1 toner =1 poorpaper =0 1
poorprint =0 drum =0 toner =0 poorpaper =1 4/5
poorprint =1 drum =0 toner =0 poorpaper =1 1/5
poorprint =0 drum =1 toner =0 poorpaper =1 0
poorprint =1 drum =1 toner =0 poorpaper =1 1
poorprint =0 drum =0 toner =1 poorpaper =1 0
poorprint =1 drum =0 toner =1 poorpaper =1 1
poorprint =0 drum =1 toner =1 poorpaper =1 0
poorprint =1 drum =1 toner =1 poorpaper =1 1

```

```

-----
p(wrinkled| fuse poorpaper):
wrinkled =0 fuse =0 poorpaper =0 4/5
wrinkled =1 fuse =0 poorpaper =0 1/5
wrinkled =0 fuse =1 poorpaper =0 1/2
wrinkled =1 fuse =1 poorpaper =0 1/2
wrinkled =0 fuse =0 poorpaper =1 5/7
wrinkled =1 fuse =0 poorpaper =1 2/7
wrinkled =0 fuse =1 poorpaper =1 0
wrinkled =1 fuse =1 poorpaper =1 1

```

```

-----
p(multiple| poorpaper roller):
multiple =0 poorpaper =0 roller =0 1
multiple =1 poorpaper =0 roller =0 0
multiple =0 poorpaper =1 roller =0 5/7
multiple =1 poorpaper =1 roller =0 2/7
multiple =0 poorpaper =0 roller =1 1/2
multiple =1 poorpaper =0 roller =1 1/2
multiple =0 poorpaper =1 roller =1 0
multiple =1 poorpaper =1 roller =1 1

```

```

-----
p(jam| fuse roller):
jam =0 fuse =0 roller =0 3/5
jam =1 fuse =0 roller =0 2/5
jam =0 fuse =1 roller =0 1/2
jam =1 fuse =1 roller =0 1/2
jam =0 fuse =0 roller =1 0
jam =1 fuse =0 roller =1 1
jam =0 fuse =1 roller =1 1
jam =1 fuse =1 roller =1 0

```

Part 2:

```

probability fuse assembly malfunctioned is:
fuse =0
fuse =1 1.000000e+000

```

Part 3 (Bayesian tables):

```

p(fuse):
fuse =0 13/17
fuse =1 4/17

```

```

-----
p(drum):
drum  =0  12/17
drum  =1  5/17

-----

p(toner):
toner  =0  11/17
toner  =1  6/17

-----

p(poorpaper):
poorpaper  =0  8/17
poorpaper  =1  9/17

-----

p(roller):
roller  =0  13/17
roller  =1  4/17

-----

p(burning| fuse):
burning  =0  fuse  =0  13/14
burning  =1  fuse  =0  1/14
burning  =0  fuse  =1  2/5
burning  =1  fuse  =1  3/5

-----

p(poorprint| drum toner poorpaper):
poorprint  =0  drum  =0  toner  =0  poorpaper  =0  4/5
poorprint  =1  drum  =0  toner  =0  poorpaper  =0  1/5
poorprint  =0  drum  =1  toner  =0  poorpaper  =0  1/3
poorprint  =1  drum  =1  toner  =0  poorpaper  =0  2/3
poorprint  =0  drum  =0  toner  =1  poorpaper  =0  1/4
poorprint  =1  drum  =0  toner  =1  poorpaper  =0  3/4
poorprint  =0  drum  =1  toner  =1  poorpaper  =0  1/3
poorprint  =1  drum  =1  toner  =1  poorpaper  =0  2/3
poorprint  =0  drum  =0  toner  =0  poorpaper  =1  5/7
poorprint  =1  drum  =0  toner  =0  poorpaper  =1  2/7
poorprint  =0  drum  =1  toner  =0  poorpaper  =1  1/3
poorprint  =1  drum  =1  toner  =0  poorpaper  =1  2/3
poorprint  =0  drum  =0  toner  =1  poorpaper  =1  1/3
poorprint  =1  drum  =0  toner  =1  poorpaper  =1  2/3
poorprint  =0  drum  =1  toner  =1  poorpaper  =1  1/3
poorprint  =1  drum  =1  toner  =1  poorpaper  =1  2/3

-----

p(wrinkled| fuse poorpaper):
wrinkled  =0  fuse  =0  poorpaper  =0  5/7
wrinkled  =1  fuse  =0  poorpaper  =0  2/7
wrinkled  =0  fuse  =1  poorpaper  =0  1/2
wrinkled  =1  fuse  =1  poorpaper  =0  1/2
wrinkled  =0  fuse  =0  poorpaper  =1  2/3
wrinkled  =1  fuse  =0  poorpaper  =1  1/3
wrinkled  =0  fuse  =1  poorpaper  =1  1/3
wrinkled  =1  fuse  =1  poorpaper  =1  2/3

-----

p(multiple| poorpaper roller):
multiple  =0  poorpaper  =0  roller  =0  6/7
multiple  =1  poorpaper  =0  roller  =0  1/7
multiple  =0  poorpaper  =1  roller  =0  2/3
multiple  =1  poorpaper  =1  roller  =0  1/3
multiple  =0  poorpaper  =0  roller  =1  1/2
multiple  =1  poorpaper  =0  roller  =1  1/2
multiple  =0  poorpaper  =1  roller  =1  1/3
multiple  =1  poorpaper  =1  roller  =1  2/3

-----

p(jam| fuse roller):
jam  =0  fuse  =0  roller  =0  7/12
jam  =1  fuse  =0  roller  =0  5/12
jam  =0  fuse  =1  roller  =0  1/2
jam  =1  fuse  =1  roller  =0  1/2
jam  =0  fuse  =0  roller  =1  1/4
jam  =1  fuse  =0  roller  =1  3/4
jam  =0  fuse  =1  roller  =1  2/3
jam  =1  fuse  =1  roller  =1  1/3

```

```
-----
probability fuse assembly malfunctioned is:
fuse  =0  3.813848e-001
fuse  =1  6.186152e-001
-----
```

Part 4 (using Bayesian tables):

```
optimal joint state of faults given evidence:
optimal states in 1/2 coding is:
fuse=2
drum=1
toner=1
poorpaper=1
roller=1
-----
```

Part 5 (using Bayesian tables):

```
optimal joint state of faults given only burning smell and paper jammed:
optimal states in 1/2 coding is:
fuse=2
drum=1
toner=1
poorpaper=2
roller=1
-----
```

9.4 For a discrete variable, the normalisation constraint is

$$1 = \sum_s p(x_i = s | \text{pa}(x_i) = \mathbf{t}^i) = \sum_s \theta_s^i(\mathbf{t}^i) \quad (\text{B.0.93})$$

The log likelihood as a function of the table parameters is then

$$\sum_{i=1}^K \sum_{n=1}^N \mathbb{I}[x_i^n = s] \mathbb{I}[\text{pa}(x_i^n) = \mathbf{t}^i] \log \theta_s^i(\mathbf{t}^i) + \sum_{i=1}^K \sum_{\mathbf{t}^i} \lambda^i(\mathbf{t}^i) \left(1 - \sum_s \theta_s^i(\mathbf{t}^i)\right) \quad (\text{B.0.94})$$

where $\lambda^i(\mathbf{t}^i)$ is a set of Lagrange multipliers. Differentiating the above with respect to $\theta_s^j(\mathbf{t}^j)$ and equating to zero, we have

$$\sum_{n=1}^N \mathbb{I}[x_j^n = s] \mathbb{I}[\text{pa}(x_j^n) = \mathbf{t}^j] \frac{1}{\theta_s^j(\mathbf{t}^j)} - \lambda^j(\mathbf{t}^j) = 0 \quad (\text{B.0.95})$$

Rearranging gives

$$\theta_s^j(\mathbf{t}^j) = \frac{1}{\lambda^j(\mathbf{t}^j)} \sum_{n=1}^N \mathbb{I}[x_j^n = s] \mathbb{I}[\text{pa}(x_j^n) = \mathbf{t}^j] \quad (\text{B.0.96})$$

From normalisation, we have

$$\theta_s^j(\mathbf{t}^j) = \frac{\sum_{n=1}^N \mathbb{I}[x_j^n = s] \mathbb{I}[\text{pa}(x_j^n) = \mathbf{t}^j]}{\sum_{n=1}^N \sum_s \mathbb{I}[x_j^n = s] \mathbb{I}[\text{pa}(x_j^n) = \mathbf{t}^j]} \quad (\text{B.0.97})$$

9.5 Following the standard maximum likelihood example, one can write

$$CL(\theta) \stackrel{N \rightarrow \infty}{\rightarrow} \langle \log p(y|x, \theta) \rangle_{p(x, y | \theta^0)} \quad (\text{B.0.98})$$

We assume for concreteness that the variables are continuous (the same holds for discrete variables on replacing integration with summation) and rewrite the above as

$$CL(\theta) = \int p(x|\theta^0) \int p(y|x, \theta^0) \log p(y|x, \theta) \quad (\text{B.0.99})$$

We can write this as

$$CL(\theta) = - \int p(x|\theta^0) \text{KL}(p(y|x, \theta^0) || p(y|x, \theta)) + \text{const}. \quad (\text{B.0.100})$$

When $\theta = \theta^0$ the KL term is zero, and $CL(\theta)$ is maximal. Hence $\theta = \theta^0$ corresponds to an optimum of the conditional likelihood. However, this does not mean that one can necessarily learn all parameters. For example if a distribution is parameterised as

$$p(x, y|\theta) = p(y|x, \theta_1)p(x|\theta_2) \quad (\text{B.0.101})$$

then the conditional likelihood of $p(y|x, \theta)$ is independent of θ_2 .

9.6 The mean and variance of a distribution are given by

$$m = \frac{\alpha}{\alpha + \beta}, \quad s = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

Simply plugging in the definitions and rearranging gives the result.

9.7 Follows from the definition and the Beta distribution and the digamma function. We can then use any gradient based technique.

9.9 Using $Z(u) = \frac{\prod_i \Gamma(u_i)}{\Gamma(\sum_i u_i)}$ and substituting in we obtain the following expression for the likelihood

$$\prod_{k,j} \frac{\prod_i \Gamma(u'_i(v_k; j))}{\Gamma(\sum_i u'_i(v_k; j))} \frac{\Gamma(\sum_i u_i(v_k; j))}{\prod_i \Gamma(u_i(v_k; j))} = \prod_k \prod_j \frac{\Gamma(\sum_i u_i(v_k; j))}{\Gamma(\sum_i u'_i(v_k; j))} \prod_i \left[\frac{\Gamma(u'_i(v_k; j))}{\Gamma(u_i(v_k; j))} \right] \quad (\text{B.0.102})$$

- 9.10
1. $(1 + 7 + 7 * 6/2) * (1 + 6 + 6 * 5/2) * (1 + 5 + 5 * 4/2) * (1 + 4 + 4 * 3/2) * (1 + 3 + 3 * 2/2) * 4 * 2 = 6288128$
 2. $(1 + 7 + 7 * 6/2) + (1 + 6 + 6 * 5/2) + (1 + 5 + 5 * 4/2) + (1 + 4 + 4 * 3/2) + (1 + 3 + 3 * 2/2) + 4 + 2 = 91$ seconds since we can use the decomposability to optimise each variable separately.
 3. One way to do this would be to repeat the above calculation for different ancestral orders (of which there are $8!$), giving $91 \times 8!$ seconds or about 42 days.

Another route is to realise that, for the node last in the ancestral ordering, its optimal parents don't depend on the order of preceding variables in the ancestral ordering. For example, if the last node in the ordering is number 8, then the optimal parents for node 8 don't depend on the ordering of the variables 1 through to 7. Since there are 8 possible nodes that could be last in the ancestral ordering, there are

$$8 \left(\binom{7}{2} + \binom{7}{1} + 1 \right)$$

calculations required to compute the optimal last node. Once this is determined, we can then continue to the node in the 7^{th} position in the ordering. We can then continue, knowing that one of the variables has now been eliminated, thus requiring

$$7 \left(\binom{6}{2} + \binom{6}{1} + 1 \right)$$

calculations. We add up these computations, giving a total number of computations

$$2 \left(\binom{2}{1} + 1 \right) + \sum_{n=3}^8 n \left(\binom{n-1}{2} + \binom{n-1}{1} + 1 \right) = 583$$

Assuming that each of these computations takes 1 second, we require 583 seconds to find the optimal member of \mathcal{N} .

9.12 Sketch: One continues as in the standard IS development, reaching equation (9.6.43). At this point we may use, for arbitrarily non-negative p_c , $\sum_c p_c = 1$,

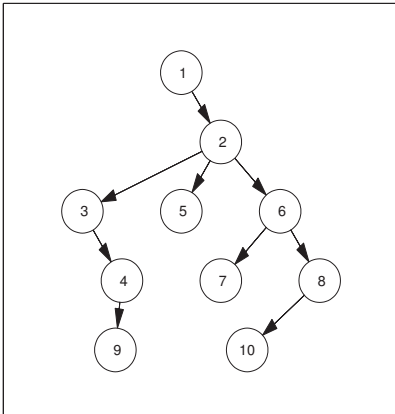
$$Z = \sum_{\mathcal{X}} e^{\sum_c \theta_c f_c(\mathcal{X}_c)} = \sum_{\mathcal{X}} e^{\langle \theta_c f_c(\mathcal{X}_c) \rangle_p}$$

Now $e^{\langle x \rangle} \leq \langle e^x \rangle$. Hence

$$Z \leq \sum_{\mathcal{X}} \sum_c p_c e^{\theta_c f_c(\mathcal{X}_c)/p_c} = \sum_c \gamma_c p_c \sum_{\mathcal{X}_c} e^{\theta_c f_c(\mathcal{X}_c)/p_c}$$

here γ_c takes care of enumerating over the states that are outside of \mathcal{X}_c . This then results in a parameter decoupling. However, we now have additional parameters to set, namely p_c . These can be optimised by including a Lagrange constraint $\sum_c p_c = 1$.

Chow Liu Net from data



9.13 See `ChowLiu.m` and run `drawNet(ChowLiu(X))`. See also `demoChowLiu.m` for a demo.

```

function A=ChowLiu(data)
%CHOWLIU Chow Liu algorithm
%A=ChowLiu(data)
%data is a data matrix. See demoChowLiu
import brml.*
data=squeezestates(data);
nstates=maxarray(data,2); nvars=size(data,1);
c=0; % compute all pairwise Mutual Informations
for i=1:nvars
    for j=i+1:nvars
        w(i,j)=MIemp(data(i,:),data(j,:),nstates(i),nstates(j));
        c=c+1;
        link(c,:)=[i j]; val(c)=w(i,j);
    end
end
[dum b]=sort(val,'descend'); edgelist=link(b(1:c),:); % sort the edges
A=spantree(edgelist); % find a spanning
A=triu(A); % orient away from node 1 to produce a DAG

```

9.14 Lets consider an ordering of the nodes $1, 2, 3, \dots, N$ with the last node N having parents from lower ordered nodes. For node N , we can either include or not an edge from each of lower ordered nodes. This gives 2^{N-1} possible parental graphs. We then move on to node $N-1$. Since this is a DAG we assume it can only have connections from nodes numbered from 1 to $N-2$, for which there are 2^{N-2} possible graphs, etc. This gives the lower bound result. The upper bound is obtained from the fact that there are $N!$ possible orderings of the nodes, each of which can have many parental graphs. This is an upper bound since we will overcount some graphs this way. For example, trivially, the graphs with no parental connections at all are clearly the same graph, whatever the ordering of the nodes.

10.1 Looking at the data, the estimates using maximum likelihood are

$$p(C = 1|Young) = 0.5, p(F = 1|Young) = 1, p(SP = 1|Young) = 1, p(B = 1|Young) = 0 \quad (\text{B.0.103})$$

and

$$p(C = 1|Old) = 0.75, p(F = 1|Old) = 0.25, p(SP = 1|Old) = 0.25, p(B = 1|Old) = 0.75 \quad (\text{B.0.104})$$

and $p(Young) = 2/6$ and $p(Old) = 4/6$. Plugging this into Bayes formula, we get

$$p(Young|C = 0, F = 1, SP = 1, B = 0) \propto 0.5 * 1 * 1 * 1/6 \quad (\text{B.0.105})$$

$$p(Old|C = 0, F = 1, SP = 1, B = 0) \propto 0.25 * 0.25 * 0.25 * 0.25 * 4/6 \quad (\text{B.0.106})$$

Using the fact that these probabilities sum to 1, this gives $p(Young|C = 0, F = 1, SP = 1, B = 0) = 64/65$

10.3 See exercisePoliticsSport.m.

```

probability x is about politics = 0.830599

P=[1 0 1 1 1 0 1 1; % Politics
   0 0 0 1 0 0 1 1;
   1 0 0 1 1 0 1 0;
   0 1 0 0 1 1 0 1;
   0 0 0 1 1 0 1 1;
   0 0 0 1 1 0 0 1]

xS=[1 1 0 0 0 0 0 0; % Sport
    0 0 1 0 0 0 0 0;
    1 1 0 1 0 0 0 0;
    1 1 0 1 0 0 0 1;
    1 1 0 1 1 0 0 0;
    0 0 0 1 0 1 0 0;
    1 1 1 1 1 0 1 0]

pP = size(xP,1)/(size(xP,1) + size(xS,1)); pS = 1-pP; % ML class priors pE = p(c=E), pS=p(c=S)

mP = mean(xP); % ML estimates of p(x=1|c=E)
mS = mean(xS); % ML estimates of p(x=1|c=S)

xtest=[1 0 0 1 1 1 1 0]; % test point

npP = pP*prod(mP.^xtest.*(1-mP).^(1-xtest)); % p(x,c=P)
npS = pS*prod(mS.^xtest.*(1-mS).^(1-xtest)); % p(x,c=S)

pxP = npP/(npP+npS);
fprintf(1,'probability x is about politics = %g\n',pxP);

```

10.4 The decision boundary is given by

$$p_1 \prod_i p(x_i|1) = p_0 \prod_i p(x_i|0)$$

$$p_1 \prod_i \theta_i^{1-x_i} (1-\theta_i)^{1-x_i} = p_0 \prod_i \theta_i^{0-x_i} (1-\theta_i)^{1-x_i}$$

Taking log, and defining $\alpha_i = \log \theta_i$, $\beta_i = \log(1-\theta_i)$, we have

$$\log p_1 + \sum_i x_i \alpha_i^1 + (1-x_i) \beta_i^1 = \log p_0 + \sum_i x_i \alpha_i^0 + (1-x_i) \beta_i^0$$

$$\sum_i x_i \underbrace{(\alpha_i^1 - \beta_i^1 - \alpha_i^0 + \beta_i^0)}_{w_i} + \underbrace{\log p_1 - \log p_0 + \sum_i (\beta_i^1 - \beta_i^0)}_b = 0$$

10.5 1. Standard Naive Bayes training as in the main text.

2.

$$p(c|x) = \frac{p(c) \prod_i p(x_i|c)}{\sum_c p(c) \prod_i p(x_i|c)}$$

3. The classification is certain that it is not spam. This is because the probability is zero, which annihilates all terms in the product in Naive Bayes. We can use a Bayesian method on the tables to help with this. Placing for example a uniform Dirichlet prior on the tables has the effect of simply adding 1 to all the data counts. This means that there would no longer be any zero counts in the data and the problem is thereby diminished. A spammer could attempt to fool a simple Naive Bayes classifier by for example including the spam text at the beginning of the email, and then appending a large amount of normal text at the end of the email. This would have the effect of biasing the statistics of the email towards a normal email, meaning that the Naive Bayes classifier would class this as a normal ‘ham’ email.

10.6 This just follows from defining

$$p(x, c) = \frac{1}{N} \sum_n \delta(x - x^n) \delta(c - c^n)$$

Then

$$\text{KL}(p|q) = \text{const.} - \langle \log q(x, c) \rangle_{p(x, c)} = \text{const.} - \frac{1}{N} \sum_n \log q(x^n, c^n)$$

Hence minimising $\text{KL}(p|q)$ corresponds to maximising $\sum_n \log q(x^n, c^n)$, which is the data log likelihood assuming the data is i.i.d..

10.7 Plugging in the structured form of $q(x, c)$ it is immediately clear that the optimal form is

$$q(x_i|x_{pa(i)}, c) = p(x_i|x_{pa(i)}, c) \tag{B.0.107}$$

Plugging this back in to the Kullback-Leibler divergence, we obtain that minimising the Kullback-Leibler divergence is

$$= - \sum_i \langle \log p(x_i|x_{pa(i)}, c) \rangle_{p(x_i, x_{pa(i)}, c)}$$

$$- \sum_i \left[\langle \log p(x_i, x_{pa(i)}, c) \rangle_{p(x_i, x_{pa(i)}, c)} - \langle \log p(x_{pa(i)}, c) \rangle_{p(x_{pa(i)}, c)} \right]$$

$$- \sum_i \left[\langle \log p(x_i, x_{pa(i)}, c) \rangle_{p(x_i, x_{pa(i)}, c)} - \langle \log p(x_{pa(i)}, c) \rangle_{p(x_{pa(i)}, c)} - \langle \log p(x_i|c) \rangle_{p(x_i, c)} + \langle \log p(x_i|c) \rangle_{p(x_i, c)} \right]$$

$$- \sum_i \left\langle \log \frac{p(x_i, x_{pa(i)}, c)}{p(x_{pa(i)}, c)p(x_i|c)} \right\rangle_{p(x_i, x_{pa(i)}, c)} + \text{const.}$$

where in the last line we used the fact that the final term $\langle \log p(x_i|c) \rangle_{p(x_i, c)}$ is independent of the order $pa(i)$, with which we wish to maximise the expression L over. Dividing the numerator and denominator by $p(c)$, we arrive at the result that the maximum likelihood optimal setting is given by computing the weights associated with conditional mutual information. To learn the optimal tree we then find a maximum weighted spanning tree using these weights, and then orient the edges outwards from a chosen root. Finally we add links from c to each variable x_i .

11.1 See `demoEMprinter.m`. Based on 50 EM iterations, the probability of a drum unit problem, given the evidence and the learned table, is 0.618 to the decimal places.

11.2

11.3 This does not correspond to maximum likelihood. The reason is the available information is not fully used. When we observe y , even if we don’t observe x , this gives some information about the state of x – indeed, this is what is used in the EM approach, since we will make use of the distribution $p(x|y^n)$. The suggested approach is not unreasonable, and essentially corresponds to throwing away the whole datapoint if there are missing entries. In general this would be very wasteful.

- 11.4 1. The energy is (for $q^n(h) \equiv p^{old}(h|\mathbf{v}^n)$):

$$\sum_n \langle \log p(\mathbf{v}^n|h)p(h) \rangle_{q^n(h)} = \sum_n \sum_{t=1}^{T-1} \langle \log p(v_{t+1}^n|v_t^n, h) \rangle_{q^n(h)} + \sum_n \langle \log p(v_1^n|h) \rangle_{q^n(h)} + T \sum_n \langle \log p(h) \rangle_{q^n(h)}$$

We need to optimise this *w.r.t.* the transitions $p(v_{t+1} = i|v_t = j, h) \equiv \theta_{ij}^h$, initial distribution $p(v_1|h)$ and prior $p(h)$. Using a Lagrange term, the contribution of the transition to the energy is

$$\mathcal{L} = \sum_n \sum_{t=1}^{T-1} \sum_{i,j} q^n(h) \mathbb{I}[v_t^n = j, v_{t+1}^n = i] \log \theta_{ij}^h + \sum_h \sum_j \lambda_j \sum_i (1 - \theta_{ij}^h)$$

Differentiating *w.r.t.* θ_{ij}^h and setting to zero, we obtain

$$\sum_n \sum_{t=1}^{T-1} q^n(h) \mathbb{I}[v_t^n = j, v_{t+1}^n = i] \frac{1}{\theta_{ij}^h} - \lambda_j = 0$$

Rearranging, this gives

$$\theta_{ij}^h = \frac{\sum_n q^n(h) \sum_{t=1}^{T-1} \mathbb{I}[v_t^n = j, v_{t+1}^n = i]}{\lambda_j}$$

Due to normalisation, $\sum_i \theta_{ij}^h = 1$, therefore

$$\theta_{ij}^h = \frac{\sum_n q^n(h) \sum_{t=1}^{T-1} \mathbb{I}[v_t^n = j, v_{t+1}^n = i]}{\sum_i \sum_n q^n(h) \sum_{t=1}^{T-1} \mathbb{I}[v_t^n = j, v_{t+1}^n = i]}$$

This has the interpretation of the weighted number of j to i transitions across the sequences. Similarly, the optimum of the energy *w.r.t.* the initial distribution $p(v_1 = i|h) \equiv \theta_i$ is

$$\theta_i = \frac{\sum_n q^n(h) \mathbb{I}[v_1^n = i]}{\sum_i \sum_n q^n(h) \mathbb{I}[v_1^n = i]}$$

The optimal prior $p(h)$ is

$$p(h) = \frac{\sum_n q^n(h)}{\sum_h \sum_n q^n(h)}$$

One then iterates these equations to convergence.

2. See `mixMarkov.m`.
 3. See `demoMixMarkov.m`. Running this several times, we end up with the highest log likelihood of -483.635 . For this the sequences assignments are:

Group 1	CATAGGCATTCTATGTGCTG	Group 2	TGGAACCTTAAAAAAAAAAAA
	CCAGTTACGGAACGCCGAAAG		GTCTCCTGCCCTCTCTGAAC
	CGGCCGCGCTCCGGGAACG		GTGCCTGGACCTGAAAAGCC
	ACATGAACTACATAGTATAA		AAAGTGCTCTGAAAACCTCAC
	GTTGGTCAGCACACGGACTG		CCTCCCCCTCCCCCTTTCCTGC
	CACATACGGCTACCTGGGCAA		TAAGTGTCTCTGCTCCTAA
	CGGTCCGTCCGAGGCACTCG		AAAGAACTCCCCCTCCCTGCC
	CACCATCACCTTGCTAAGG		AAAAAAACGAAAAACCTAAG
	CAAATGCCTCACGCGTCTCA		GCGTAAAAAAAGTCCTGGGT
	GCCAAGCAGGTCTCAACTT		
	CATGGACTGCTCCACAAAGG		

11.5 To be done.

- 11.6 1. In the three layered general case, this is in principle more powerful than the two-layered case since when we marginalise over \mathbf{h}_3 we end up with a non-factorised contribution in \mathbf{h}_2 , which is not in the exponential family. Only if the form of $\phi(\mathbf{h}_1, \mathbf{h}_2)$ were completely unrestricted (not of a BM form), does the extra layer play no role.
 2. For the restricted BM part of the question, the same holds as well. In this case it is even more clear since the additional layer has the effect of inducing links between the nodes in the \mathbf{h}_2 layer.
- 11.7 1. A belief network in which the top row represents $p(\mathbf{x})$, and then each layer below $p(\mathbf{x}^{l-1}|\mathbf{x}^l)$. The variables \mathbf{x}^0 are in the bottom layer.
 2. The graph is multiply-connected. However, one can collect all variables in each layer and use message passing from layer to layer. The number of entries of each table $p(\mathbf{x}^{l-1}|\mathbf{x}^l)$ is $2^w \times 2^w = 2^{2w}$. Since we have a linear-chain factor graph on the cluster variables \mathbf{x} , then we can compute the marginal in $O(L2^{2w})$ time using message passing from layer to layer.

3. The energy term decomposes as

$$\begin{aligned} \left\langle \log p(\mathbf{x}^0, \mathbf{x}^1, \dots, \mathbf{x}^L) \right\rangle_{q(\mathbf{x})} &= \sum_l \sum_i \left\langle \log p(x_i^{l-1} | \mathbf{x}^l) \right\rangle_{q(x_i^{l-1}, \mathbf{x}^l)} \\ &= \sum_l \sum_i \left\langle \log \sigma \left((2x_i^{l-1} - 1) \mathbf{w}_{i,l}^\top \mathbf{x}^l \right) \right\rangle_{q(x_i^{l-1}, \mathbf{x}^l)} \end{aligned}$$

To compute each average, we therefore need to know the distribution of $(2x_i^{l-1} - 1) \mathbf{w}_{i,l}^\top \mathbf{x}^l$. If we first assume $x_i^{l-1} = 1$, we would therefore need to compute the distribution of the projection $\mathbf{w}_{i,l}^\top \mathbf{x}^l$. Whilst for certain distributions, such as Gaussian \mathbf{x}^l , this is straightforward (the distribution of a linear projection of a Gaussian is another Gaussian), for binary \mathbf{x}^l , there is no simple way to compute the distribution of this projection. In general, it would require that we compute $O(2^w)$ summations to find this exactly. For moderate w , this may still be feasible. The complexity of computing the variational EM bound is therefore $O(LW2^w)$, compared to $O(L2^{2w})$ in the exact EM case.

11.8

$$L(\theta_b, \theta_g, \theta_p) = \log \theta_b + \log \theta_g + \lambda(1 - \theta_b - \theta_g - \theta_p)$$

Differentiating *w.r.t.* θ_b , and equating to zero, we have

$$\frac{1}{\theta_b} - \lambda = 0, \Rightarrow \theta_b = \frac{1}{\lambda}$$

Similarly, optimising *w.r.t.* θ_g gives $\theta_g = 1/\lambda = \theta_b$. Since $\theta_b \geq 0$ then $\lambda \geq 0$. Hence, for θ_p , the optimal setting is given when $\theta_p = 0$ (since then we are subtracting the minimal amount from L). The normalisation criterion $\theta_b + \theta_g + \theta_p = 1$ then gives that $\theta_b = \theta_g = 1/2$.

11.9

$$p(x_1 = i) = \sum_{x_2} p(x_1 = i, x_2) = \sum_{j=1}^2 \theta_{1,j}$$

This means that θ only contributes to the likelihood via $\sum_{j=1}^2 \theta_{1,j}$. Any θ which has this same ‘projection’ is therefore maximum likelihood equivalent. In terms of the matrix θ , this means that two different θ ’s which have the same row sum will have the same likelihood. This is the case in the example since in both cases, the row sum of the two matrices is

$$\begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix}$$

12.1 To avoid numerical underflow we can use the `logsumexp.m` function. See `demoCoin.m`.

12.2 See MATLAB code `exerciseBayesModelHedge.m`.

most likely model has weights at: 1 1 1 1 0 0

```
function exerciseBayesModelHedge
import brml.*
load dodder
plot(y); title('time-series y(t)');

model = ind2subv(repmat(2,1,6),1:2^6-1)-1; % all the models
for m=1:size(model,1)
    mask=model(m,:);
    M=sum(mask); % number of parameters in this model
    A = zeros(M,M); b=zeros(M,1); c=0;
    for t=2:T
        xtm = x(mask==1,t-1);
        A = A + xtm*xtm'/sigma(t)^2;
        b = b + xtm*y(t)/sigma(t)^2;
        c=c+y(t).^2/sigma(t)^2;
    end
    A=eye(M) + A;
    logl(m) = 0.5*(-c+b'*inv(A)*b-log(det(A))+M*log(2*pi)+sum(log(2*pi*sigma(2:T).^2)));
end
post=condexp(logl'); % posterior, assuming that the prior p(m)=const
figure; bar(post); title('posterior over the models')
disp(['most likely model has weights at: ', num2str(ind2subv(repmat(2,1,6),argmax(post))-1)]);
```

12.4

$$p(\mathbf{o}_a, \mathbf{o}_b, \mathbf{o}_c) p(H_{\text{indep}} | \mathbf{o}_a, \mathbf{o}_b, \mathbf{o}_c) = p(H_{\text{indep}}) \frac{Z(\mathbf{u} + \#^a)}{Z(\mathbf{u})} \frac{Z(\mathbf{u} + \#^b)}{Z(\mathbf{u})} \frac{Z(\mathbf{u} + \#^c)}{Z(\mathbf{u})} \quad (\text{B.0.108})$$

where $Z(\mathbf{u})$ is given by equation (12.6.10).

$$\begin{aligned} p(\mathbf{o}_a, \mathbf{o}_b, \mathbf{o}_c) p(H_{\text{same}} | \mathbf{o}_a, \mathbf{o}_b, \mathbf{o}_c) &= p(H_{\text{same}}) \int p(\mathbf{o}_a | \boldsymbol{\alpha}, H_{\text{same}}) p(\mathbf{o}_b | \boldsymbol{\alpha}, H_{\text{same}}) p(\mathbf{o}_c | \boldsymbol{\alpha}, H_{\text{same}}) p(\boldsymbol{\alpha} | H_{\text{same}}) d\boldsymbol{\alpha} \\ &= p(H_{\text{same}}) \frac{Z(\mathbf{u} + \#^a + \#^b + \#^c)}{Z(\mathbf{u})} \end{aligned}$$

If we assume no prior preference for either hypothesis, $p(H_{\text{indep}}) = p(H_{\text{same}})$, then

$$\frac{p(H_{\text{indep}} | \mathbf{o}_a, \mathbf{o}_b, \mathbf{o}_c)}{p(H_{\text{same}} | \mathbf{o}_a, \mathbf{o}_b, \mathbf{o}_c)} = \frac{Z(\mathbf{u} + \#^a) Z(\mathbf{u} + \#^b) Z(\mathbf{u} + \#^c)}{Z^2(\mathbf{u}) Z(\mathbf{u} + \#^a + \#^b + \#^c)} = \frac{Z([14, 4, 5]) Z([5, 10, 8]) Z([9, 9, 5])}{Z^2([1, 1, 1]) Z([26, 21, 16])} = 2.7585863899362$$

This is computed using `logZdirichlet.m`. Hence it is around 2.7 times more probable that the people are classifying the images using the same distribution, as opposed to each using their own distribution.

12.5 1.

$$p(\mathcal{D} | \mathcal{H}_{\text{random}}) = \prod_{n=1}^{R+W} 0.5 = 0.5^{R+W}$$

2.

$$p(\mathcal{D} | \mathcal{H}_{\text{non random}}) = \int_{\theta} \theta^R (1-\theta)^W \theta^{a-1} (1-\theta)^{b-1} / B(a, b) = B(R+a, W+b) / B(a, b)$$

3.

$$p(\mathcal{H}_{\text{random}} | \mathcal{D}) = \frac{p(\mathcal{D} | \mathcal{H}_{\text{random}}) p(\mathcal{H}_{\text{random}})}{p(\mathcal{D} | \mathcal{H}_{\text{non random}}) p(\mathcal{H}_{\text{non random}}) + p(\mathcal{D} | \mathcal{H}_{\text{random}}) p(\mathcal{H}_{\text{random}})}$$

Using $p(\mathcal{H}_{\text{non random}}) = p(\mathcal{H}_{\text{random}})$ and the definitions above, gives the required result.

4.

$$0.5^2 2 / (0.5^2 2 + \text{beta}(11, 13) / \text{beta}(1, 1)) = 0.780024735721027$$

$$0.5^2 20 / (0.5^2 20 + \text{beta}(101, 121) / \text{beta}(1, 1)) = 0.827528384491262$$

This is slightly higher than in the first instance.

5. Let $N = R + W$. For $x_n \in \{0, 1\}$:

$$\left\langle \sum_{n=1}^N x_n \right\rangle = 0.5N$$

$$\left\langle \left(\sum_{n=1}^N x_n \right)^2 \right\rangle = \sum_{n, n'} \langle x_n x_{n'}' \rangle = 0.5N + 0.5^2 N(N-1)$$

Hence the variance is

$$0.5N + 0.5^2 N(N-1) - 0.5^2 N^2 = 0.5N - 0.25N = 0.25N$$

and standard deviation is $\sqrt{0.25N} = 0.5\sqrt{R+W}$. In the above computations, we have $N = 22$ and $N = 220$. In the first case, for a random classifier, we would therefore expect to see deviations from 11 (the mean) of around 1.65. In the second case, we would expect deviations from the mean of 110 of around 5.24.

12.6 1. The expression is an extension of the standard derivation. It is an integral over the parameters of the expression

$$\begin{aligned} &B(\theta_y | \alpha, \beta) B(\theta_{1|0} | \alpha_{1|0}, \beta_{1|0}) B(\theta_{1|1} | \alpha_{1|1}, \beta_{1|1}) \\ &\times \theta_y^{\#(y=1)} (1-\theta_y)^{\#(y=0)} \theta_{1|0}^{\#(x=1, y=0)} (1-\theta_{1|0})^{\#(x=0, y=0)} \theta_{1|1}^{\#(x=1, y=1)} (1-\theta_{1|1})^{\#(x=0, y=1)} \end{aligned}$$

Using the definition of the Beta function and Beta distribution, and the conjugacy property, we immediately arrive at the result.

2. If we assume symmetry for the parameters we have $p(\mathcal{D} | M_{x \rightarrow y})$ given by

$$\begin{aligned} &\frac{B(\#(y=1, x=0) + \alpha_{1|0}, \#(x=0, y=0) + \beta_{1|0})}{B(\alpha_{1|0}, \beta_{1|0})} \frac{B(\#(x=1, y=1) + \alpha_{1|1}, \#(y=0, x=1) + \beta_{1|1})}{B(\alpha_{1|1}, \beta_{1|1})} \\ &\times \frac{B(\#(x=1) + \alpha, \#(x=0) + \beta)}{B(\alpha, \beta)} \end{aligned}$$

3. Taking the ratio, we obtain the Bayes' factor

$$\frac{B(\#(x=1, y=0) + \alpha_{1|0}, \#(x=0, y=0) + \beta_{1|0}) B(\#(x=1, y=1) + \alpha_{1|1}, \#(x=0, y=1) + \beta_{1|1}) B(\#(y=1) + \alpha, \#(y=0) + \beta)}{B(\#(y=1, x=0) + \alpha_{1|0}, \#(x=0, y=0) + \beta_{1|0}) B(\#(x=1, y=1) + \alpha_{1|1}, \#(y=0, x=1) + \beta_{1|1}) B(\#(x=1) + \alpha, \#(x=0) + \beta)}$$

4. If we assume that a deterministic distribution $p(y|x)$ in which the state of y is quite certain (given the state of x) yet $p(x)$ is not necessarily certain, we can encode this by setting:
- $\alpha_{1|0} = \beta_{1|0}$ to be less than 1. This means that when $y = 0$ then $p(x = 1|y = 0)$ will be high or $p(x = 0|y = 0)$ will be high (since the beta distribution with these settings favours more deterministic distributions).
 - Similarly, if we set $\alpha_{1|1} = \beta_{1|1}$ to be less than 1 (say 0.1), we encode that, given $y = 1$ then we will have a quite deterministic $p(x|y = 1)$ distribution (see the form of the Beta distribution for $\alpha = \beta = 0.1$).
 - Setting $\alpha = \beta = 1$ is a generic setting to not bias $p(y)$ towards any particular shape.

For example

```
N00=10; % number of times x=0,y=0
N01=10; % number of times x=0,y=1
N10=0;  % number of times x=1,y=0
N11=0;  % number of times x=1,y=1

A=1; B=1; % uniform prior
A10=0.1; B10=0.1; % bias towards deterministic table
A11=0.1; B11=0.1; % bias towards deterministic table

r1=beta(N10+A10,N00+B10)*beta(N11+A11,N01+B11)*beta(N01+N11+A,N10+N00+B);
r2=beta(N01+A10,N00+B10)*beta(N11+A11,N10+B11)*beta(N10+N11+A,N01+N00+B);

BayesFactor=r1/r2
```

This gives a Bayes' Factor of around 16.8, which says that $M_{y \rightarrow x}$ is much more likely. This is because in this case, the count data N is consistent with a deterministic mapping $y \rightarrow x$ (since the state of x is deterministic in this case, once the state of y is known, but $p(y)$ is uniform). This shows that we can learn the direction of the arrow in the Belief Network under assumptions such as how deterministic the tables of the distributions are.

It's interesting also to consider the situation in which the priors on the tables are uniform:

```
N00=10; % number of times x=0,y=0
N01=10; % number of times x=0,y=1
N10=0;  % number of times x=1,y=0
N11=0;  % number of times x=1,y=1

A=1; B=1; % uniform prior
A10=1; B10=1; % uniform prior
A11=1; B11=1; % uniform prior

r1=beta(N10+A10,N00+B10)*beta(N11+A11,N01+B11)*beta(N01+N11+A,N10+N00+B);
r2=beta(N01+A10,N00+B10)*beta(N11+A11,N10+B11)*beta(N10+N11+A,N01+N00+B);

BayesFactor=r1/r2
```

This gives a Bayes' Factor of around 0.17, suggesting that the model $M_{x \rightarrow y}$ is more probable. The explanation is that, under a flat prior, it's relatively unlikely that such an extreme table as a deterministic mapping from y to x would occur. Since, for this data $p(y|x)$ is uniform for both states of x , this outweighs the single deterministic table $p(x)$.

The exercise demonstrates how we can use a heuristic such as how deterministic a table is to help determine the direction of an edge in an otherwise Markov equivalent Belief Network.

13.1 The decision boundary is given by

$$p(\text{class 1}|x) = p(\text{class 2}|x)$$

Ignoring the common denominator and taking logs, this is equivalent to

$$\log p(x|\text{class 1}) + \log p_1 = \log p(x|\text{class 2}) + \log p_2$$

which is

$$\begin{aligned} -\frac{1}{2\sigma_1^2}(x - \mu_1)^2 - \frac{1}{2}\log(2\pi\sigma_1^2) + \log p_1 &= -\frac{1}{2\sigma_2^2}(x - \mu_2)^2 - \frac{1}{2}\log(2\pi\sigma_2^2) + \log p_2 \\ -\frac{1}{\sigma_1^2}(x - \mu_1)^2 - \log(2\pi\sigma_1^2) + 2\log p_1 &= -\frac{1}{\sigma_2^2}(x - \mu_2)^2 - \log(2\pi\sigma_2^2) + 2\log p_2 \end{aligned}$$

This can be written in the form

$$ax^2 + bx + c = 0$$

where

$$\begin{aligned} a &= \frac{1}{\sigma_2^2} - \frac{1}{\sigma_1^2} \\ b &= 2\left(\frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}\right) \\ c &= \frac{\mu_2^2}{\sigma_2^2} - \frac{\mu_1^2}{\sigma_1^2} - \log\sigma_1^2 + 2\log p_1 + \log\sigma_2^2 - 2\log p_2 \end{aligned}$$

Since we have a quadratic equation, there are two solutions,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Using this one may create a solution in which the two Gaussians intersect where they both have appreciable mass, but another solution in which the Gaussians intersect when both are in the tails of the distribution. See `plotGaussianDecBoundary.m`. This is a natural, but potentially unintended, consequence of the Gaussian distribution. It's unlikely we will have confidence in estimating the tails of a distribution well, but the decision boundary can change there too.

13.2 Define the shorthand $p_j = p(\text{class } j|\mathbf{x})$, $q_j = q(\text{class } j|\mathbf{x})$. The classifier makes an error if the state sampled from q does not match the state sampled from p . Since we sample q and p independently, the expected error is given by

$$E = \sum_{i,j} \mathbb{I}[i \neq j] p(\text{class } i|\mathbf{x}) q(\text{class } j|\mathbf{x})$$

Since $\mathbb{I}[i \neq j] = 1 - \mathbb{I}[i = j]$, we can equivalently consider the q that maximises the accuracy

$$A = \sum_{i,j} \mathbb{I}[i = j] p(\text{class } i|\mathbf{x}) q(\text{class } j|\mathbf{x}) = \sum_j p(\text{class } j|\mathbf{x}) q(\text{class } j|\mathbf{x}) = \sum_j p_j q_j$$

Our task is then to find the distribution q that maximises $\sum_j p_j q_j$. Since $\sum_j p_j q_j$ is the average of p with respect to q , the optimal setting of q is to place all the mass in the highest valued component of p . That is, find that component i^* which corresponds to the highest value of p_1, \dots, p_K , and then set

$$q_i = \begin{cases} 1 & \text{if } i = i^* \\ 0 & \text{otherwise} \end{cases}$$

This is Bayes' decision rule.

13.3

$$U(c^{pred}) = \sum_{c^{true}} U(c^{true}, c^{pred}) p(c^{true}) = \begin{cases} 3.2 & \text{class 1} \\ 2.9 & \text{class 2} \\ 1.3 & \text{class 3} \end{cases}$$

So the best decision is to choose class 1. See `ExerciseUtilpred.m`.

13.4

$$p(c = 1|x) = \frac{p(x|c = 1)p(c = 1)}{p(x|c = 1)p(c = 1) + p(x|c = 2)p(c = 2)} = \frac{p(x|c = 1)}{p(x|c = 1) + p(x|c = 2)} = \frac{1}{1 + \frac{p(x|c=1)}{p(x|c=2)}}$$

Using the Gaussians, we have

$$\frac{p(x|c = 1)}{p(x|c = 2)} = e^{-\frac{1}{2\sigma^2}(x-m_1)^2 + \frac{1}{2\sigma^2}(x-m_2)^2} = e^{\frac{1}{2\sigma^2}(2x(m_1-m_2) + m_2^2 - m_1^2)}$$

so

$$a = -\frac{1}{\sigma^2}(m_1 - m_2), \quad b = \frac{m_2^2 - m_1^2}{2\sigma^2}$$

13.5 What's not clear is how 'lucky' WowCo is in finding the solution. In principle, it could attempt a great number of solutions until one just hits on the solution that has minimal test error. The point is that if one is allowed to search repeatedly for an algorithm that gets minimal test error, this is effectively finding an algorithm that is trained to minimise the test error itself. There is no reason to have any confidence that such an algorithm would perform well on any novel problem.

13.6

$$\text{KL}(q|\hat{p}) = \langle \log q \rangle_q - \langle \log \hat{p} \rangle = \langle \log q \rangle_q - \langle \log p \rangle_q - \langle \log \tilde{p} \rangle - \log A \geq 0$$

This gives

$$\log A \geq \langle \log \tilde{p}(y|x) \rangle_{q(x,y)} - \text{KL}(q(x,y)|p(x,y))$$

Then for the particular empirical,

$$q(x,y) = \frac{1}{N} \sum_{n=1}^N \delta(x, x^n) \delta(y, y^n)$$

the first term reduces to enumerating the integrand at the training points:

$$\log A \geq \frac{1}{N} \sum_{n=1}^N \log \tilde{p}(y^n|x^n) - \text{KL}(q(x,y)|p(x,y))$$

We can write

$$p(x, y) = p(y|x)p(x)$$

so that

$$\text{KL}(q(x, y)|p(x, y)) = \langle \log q(x, y) \rangle_{q(x, y)} - \langle \log p(y|x) \rangle_{q(x, y)} - \langle \log p(x) \rangle_{q(x, y)}$$

Similarly, we can write

$$q(x, y) = q(y|x)q(x)$$

so that

$$\text{KL}(q(x, y)|p(x, y)) = \langle \log q(x) \rangle_{q(x)} + \langle \log q(y|x) \rangle_{q(x, y)} - \frac{1}{N} \sum_n \log p(y^n|x^n) - \langle \log p(x) \rangle_{q(x)}$$

Assuming that there is a unique output y for each training input x , then $q(y|x)$ has probability 1 and that $p(y^n|x^n) = 1$, then

$$\log A \geq \frac{1}{N} \sum_{n=1}^N \log \tilde{p}(y^n|x^n) - \text{KL}(q(x)|p(x))$$

In the case that the predictor classifies each training point with certainty, the first term is zero. Exponentiating, we obtain

$$A \geq e^{-\text{KL}(q(x)|p(x))}$$

13.7 $p(c|x, y) \propto p(y|c)p(c|x)$.

14.1 See `exerciseNearNeigh.m`.

14.2 To be written.

14.3 1. This can be achieved using validation in which a portion of the training data is used to assess the performance of the classifier for a given number of neighbours.

2. Those elements i corresponding to small w_i will have little impact on the BQ. If we assume that all the pixels are in greyscale (so x_i is positive). Therefore one might be able to explain BQ based on highlighting the image \mathbf{x} with each pixel x_i shaded green for $w_i > 0$ and red for $w_i < 0$. A drawback of this approach is that it is evaluating each pixel individually in terms of its contribution to the BQ. In reality we would like a more global feature (such as say the symmetry of the face) to assess the BQ.

15.2 Because of the symmetry, the covariance matrix is the identity. This means that the principal component is not well defined – one can take any direction. A suitable one dimensional representation in this case is given by the non-linear mapping $x_i = \cos \theta, y_i = \sin \theta$.

15.3 Let $\mathbf{y}^a = \mathbf{c} + \sum_{i=1}^M a_i \mathbf{e}^i$, $\mathbf{y}^b = \mathbf{c} + \sum_{i=1}^M b_i \mathbf{e}^i$

$$(\mathbf{y}^a - \mathbf{y}^b)^2 = \left(\sum_{i=1}^M (a_i - b_i) \mathbf{e}^i \right)^2 = \sum_{ij} (a_i - b_i)(a_j - b_j) \mathbf{e}^i \mathbf{e}^j = \sum_i (a_i - b_i)^2$$

15.4

$$\mathbf{S}_{xy} \mathbf{S}_{yy}^{-1} \mathbf{S}_{yx} \mathbf{a} = \lambda^2 \mathbf{S}_{xx} \mathbf{a}$$

We can write this as

$$\mathbf{S}_{xy} \mathbf{S}_{yy}^{-\frac{1}{2}} \mathbf{S}_{yy}^{-\frac{1}{2}} \mathbf{S}_{yx} \mathbf{S}_{xx}^{-\frac{1}{2}} \mathbf{S}_{xx}^{\frac{1}{2}} \mathbf{a} = \lambda^2 \mathbf{S}_{xx}^{\frac{1}{2}} \mathbf{S}_{xx}^{\frac{1}{2}} \mathbf{a}$$

Multiplying by $\mathbf{S}_{xx}^{-\frac{1}{2}}$ we obtain

$$\underbrace{\mathbf{S}_{xx}^{-\frac{1}{2}} \mathbf{S}_{xy} \mathbf{S}_{yy}^{-\frac{1}{2}} \mathbf{S}_{yx} \mathbf{S}_{xx}^{-\frac{1}{2}}}_{\mathbf{UDV}^T} \underbrace{\mathbf{S}_{xx}^{\frac{1}{2}} \mathbf{S}_{xx}^{\frac{1}{2}}}_{\mathbf{VDU}^T} \mathbf{a} = \lambda^2 \mathbf{S}_{xx}^{\frac{1}{2}} \mathbf{a}$$

Defining $\tilde{\mathbf{a}} = \mathbf{S}_{xx}^{\frac{1}{2}} \mathbf{a}$ we have then

$$\mathbf{UD}^2 \mathbf{U}^T \tilde{\mathbf{a}} = \lambda^2 \tilde{\mathbf{a}}$$

This is an eigen-equation. The optimal solution for $\tilde{\mathbf{a}}$ is given by the principal eigen vector \mathbf{u}_1 , with \mathbf{D} ordered to have $D_{ii} \geq D_{jj}$ for $i < j$. Hence

$$\mathbf{a} = \mathbf{S}_{xx}^{-\frac{1}{2}} \mathbf{u}_1$$

By symmetry, the solution for \mathbf{b} is given by interchanging x and y throughout. This is equivalent to the above on interchanging \mathbf{V} and \mathbf{U} , giving the desired result.

15.5 Using the approximations, we have

$$(\mathbf{x}^a - \mathbf{x}^b)^\top \mathbf{S}^{-1} (\mathbf{x}^a - \mathbf{x}^b) \approx \left(\sum_i a_i \mathbf{e}^i - \sum_i b_i \mathbf{e}^i \right)^\top \mathbf{S}^{-1} \left(\sum_j a_j \mathbf{e}^j - \sum_j b_j \mathbf{e}^j \right) \quad (\text{B.0.109})$$

Due to the orthonormality of the eigenvectors, this is $\sum_i a_i^2 / \lambda_i - 2a_i b_i / \lambda_i + b_i^2 / \lambda_i = (\mathbf{a} - \mathbf{b})^\top \mathbf{D}^{-1} (\mathbf{a} - \mathbf{b})$ where \mathbf{D} is a diagonal matrix containing the eigenvalues.

15.6 To optimise equation (15.11.6), it is straightforward to show that we should first transform the data to be zero mean : $\sum_n \mathbf{x}^n = \mathbf{0}$ and $\sum_n v_k^n = 0$, $k = 1, \dots, K$. We may assume, without loss of generality, that the \mathbf{b}^j are orthonormal (since we could rescale the y_j^n if not). However, we cannot assume that the \mathbf{c}^k , $k = 1, \dots, K$ are orthonormal, since we cannot rescale the v . Similarly, we assume nothing, a priori, regarding the relationship between the vectors \mathbf{b}^j and \mathbf{c}^k . Differentiating equation (15.11.6) with respect to \mathbf{w}^n gives (using the orthonormality constraint on the \mathbf{b}^i)

$$\mathbf{w}^n = \sum_i (\mathbf{b}^i)^\top \left(\mathbf{x}^n - \sum_l v_l^n \mathbf{c}^l \right) \quad (\text{B.0.110})$$

The residual vector (difference between \mathbf{x}^n and the linear reconstruction) is then

$$\mathbf{r}^n = \mathbf{x}^n - \sum_i (\mathbf{b}^i)^\top \left(\mathbf{x}^n - \sum_l v_l^n \mathbf{c}^l \right) \mathbf{b}^i - \sum_j v_j^n \mathbf{c}^j \quad (\text{B.0.111})$$

By defining $\tilde{B} \equiv I - \sum_i \mathbf{b}^i (\mathbf{b}^i)^\top \equiv \mathbf{I} - \mathbf{U}\mathbf{U}^\top$, (using the notation of the previous section), the residual is

$$\mathbf{r}^n = \tilde{B} \left(\mathbf{x}^n - \sum_j v_j^n \mathbf{c}^j \right) \quad (\text{B.0.112})$$

Differentiating $E = \sum_n (\mathbf{r}^n)^\top \mathbf{r}^n$ with respect to \mathbf{c}^i , we get

$$\sum_n v_i^n \tilde{B} \mathbf{x}^n = \sum_j \sum_n v_j^n v_i^n \tilde{B} \mathbf{c}^j \quad (\text{B.0.113})$$

Define

$$[\tilde{V}]_{ij} = \sum_n v_i^n v_j^n, \quad [\tilde{\mathbf{X}}]_{ij} = \sum_n v_i^n \mathbf{x}_j^n, \quad \mathbf{C} = [\mathbf{c}^1, \dots, \mathbf{c}^K] \quad (\text{B.0.114})$$

Then the above has solution

$$\mathbf{C} = \tilde{\mathbf{X}} \tilde{V}^{-1} \quad (\text{B.0.115})$$

Hence, the objective involves the matrix

$$\tilde{S} = \sum_n (\mathbf{x}^n - \mathbf{d}^n) (\mathbf{x}^n - \mathbf{d}^n)^\top \quad (\text{B.0.116})$$

where

$$\mathbf{d}^n = \sum_j v_j^n \mathbf{c}^j \quad (\text{B.0.117})$$

Hence, the optimal solution is given by taking the principal eigenvectors of \tilde{S} , with \mathbf{C} set as above.

15.7 See `exercisePCA2D.m`.

15.8

$$\tilde{\mathbf{B}} = \mathbf{U} \mathbf{D} \mathbf{V}^\top \mathbf{V} \mathbf{D}^{-1} = \mathbf{U}$$

Since $\mathbf{U}^\top \mathbf{U} = \mathbf{I}$, the result follows.

15.9 Let $\tilde{p}(x, y, z) = \tilde{p}(x|z) \tilde{p}(z|y)$,

$$\tilde{p}(x|y) = \frac{\sum_z \tilde{p}(x|z) \tilde{p}(z|y)}{\sum_{xz} \tilde{p}(x|z) \tilde{p}(z|y)}$$

and consider

$$\sum_y \text{KL}(p(x|y) | \tilde{p}(x|y)) = \sum_y \left(\langle \log p \rangle_p - \langle \log \tilde{p} \rangle_p \right)$$

Since p is fixed, minimising the Kullback-Leibler divergence with respect to the approximation \tilde{p} is equivalent to maximising the ‘likelihood’ term $\langle \log \tilde{p} \rangle_p$. This is

$$\sum_{x,y} p(x|y) \log \tilde{p}(x|y)$$

Consider

$$\text{KL}(q(z|x, y) | \tilde{p}(z|x, y)) = \sum_z q(z|x, y) \log q(z|x, y) - \sum_z q(z|x, y) \log \tilde{p}(z|x, y) \geq 0$$

where \sum_z implies summation over all states of the variable z . Using

$$\tilde{p}(z|x, y) = \frac{\tilde{p}(x, y, z)}{\tilde{p}(x, y)}$$

and rearranging, this gives the bound,

$$\log \tilde{p}(x|y) \geq - \sum_z q(z|x, y) \log q(z|x, y) + \sum_z q(z|x, y) \log \tilde{p}(z, x, y)$$

Plugging this into the ‘likelihood’ term above, we have the bound

$$\sum_{x,y} p(x|y) \log \tilde{p}(x|y) \geq - \sum_{x,y} p(x|y) \sum_z q(z|x, y) \log q(z|x, y) + \sum_{x,y} p(x|y) \sum_z q(z|x, y) [\log \tilde{p}(x|z) + \log \tilde{p}(z|y)]$$

M-step

Optimally,

$$\tilde{p}(x|z) \propto \sum_y p(x|y) q(z|x, y)$$

and similarly,

$$\tilde{p}(z|y) \propto \sum_x p(x|y) q(z|x, y)$$

E-step

The optimal setting for the q distribution at each iteration is

$$q(z|x, y) = \tilde{p}(z|x, y)$$

which is fixed throughout the M-step.

16.1 In this case, the matrix \mathbf{B} is not formally invertible. A simple solution is to constrain \mathbf{w} to lie in the space spanned by the data. This can be achieved by finding a basis for the data, as in section(16.3.1). The derivation is analogous, and one obtains

$$\mathbf{w} \propto \mathbf{Q} \left(\mathbf{Q}^\top \mathbf{B} \mathbf{Q} \right)^{-1} \mathbf{Q}^\top (\mathbf{m}_1 - \mathbf{m}_2) \quad (\text{B.0.118})$$

16.2 See `exerciseCanonVarDigits.m`.

16.3 Zero derivative conditions are

$$\sum_{n_1} (y_1 - b - \mathbf{w}^\top \mathbf{x}_{n_1}) + \sum_{n_2} (y_2 - b - \mathbf{w}^\top \mathbf{x}_{n_2}) = 0 \quad (\text{B.0.119})$$

and

$$\sum_{n_1} (y_1 - b - \mathbf{w}^\top \mathbf{x}_{n_1}) \mathbf{x}_{n_1}^\top + \sum_{n_2} (y_2 - b - \mathbf{w}^\top \mathbf{x}_{n_2}) \mathbf{x}_{n_2}^\top = 0 \quad (\text{B.0.120})$$

Let $N = N_1 + N_2$. Then

$$b = \frac{N_1 y_1 + N_2 y_2}{N_1 + N_2} - \mathbf{w}^\top \mathbf{m} \quad (\text{B.0.121})$$

where \mathbf{m} is the mean of all the data. Under the setting of y_1 and y_2 , this gives

$$b = -\mathbf{w}^\top \mathbf{m} \quad (\text{B.0.122})$$

Plugging this into the zero derivative for \mathbf{w} we have

$$N_1 y_1 \mathbf{m}_1 + N_2 y_2 \mathbf{m}_2 = \mathbf{w}^\top \left(N \mathbf{B} + N_1 \mathbf{m}_1 \mathbf{m}_1^\top + N_2 \mathbf{m}_2 \mathbf{m}_2^\top - N_1 \mathbf{m}_1 \mathbf{m}^\top - N_2 \mathbf{m}_2 \mathbf{m}^\top \right) \quad (\text{B.0.123})$$

Now using $\mathbf{m} - \mathbf{m}_1 = (\mathbf{m}_2 - \mathbf{m}_1)N_2/N$ and the setting for y_1, y_2 we have

$$N(\mathbf{m}_1 - \mathbf{m}_2) = \left(N\mathbf{B} + \frac{N_1 N_2}{N} (\mathbf{m}_1 - \mathbf{m}_2) (\mathbf{m}_1 - \mathbf{m}_2)^\top \right) \mathbf{w} \quad (\text{B.0.124})$$

which can be written as

$$f(\mathbf{w})(\mathbf{m}_1 - \mathbf{m}_2) = \mathbf{B}\mathbf{w} \quad (\text{B.0.125})$$

for some scalar function $f(\mathbf{w})$. This means that

$$\mathbf{w} \propto \mathbf{B}^{-1}(\mathbf{m}_1 - \mathbf{m}_2) \quad (\text{B.0.126})$$

which is Fisher's solution.

16.4 See `exerciseCanonVarDigits57.m`. Interestingly, the classification performance of the CCA projected data is slightly worse than for PCA. For projecting down to a very low number of dimensions, say 2 or 3, then CCA performs better. A possible explanation is that whilst CCA attempts to find well separated data, it is not designed to find a projection that is explicitly tailored for KNN. In this sense, there exists potentially more suitable projections.

16.5 Since $B(w)$ is positive, we have

$$\frac{A(w)}{B(w)} > \lambda^{old} = \frac{A(w^{old})}{B(w^{old})} \quad (\text{B.0.127})$$

Hence, this procedure cannot decrease the quotient $F(w)$. This suggests a simple algorithm:

- Initialise w^1 randomly.
- (1) Set $\lambda^i \equiv \frac{A(w^i)}{B(w^i)}$
- (2) find $w^{i+1} = \arg \max_w A(w) - \lambda^i B(w)$
- Repeat (1) and (2) to convergence.

17.1 1. In two dimensions, $(0, 0), (1, 0)$ in class 1 and $(2, 0)$ in class 2.

2. Data that is linearly independent is linearly separable, as discussed in section(17.4.1).

17.2 From equation (17.1.5) we see that if the y data has zero mean, and x data has zero mean, then $a = 0$. Hence, if we were to shift the data so that it has zero mean, then the ordinary least squares fit must go through zero – in other words it goes through the mean in the non-shifted data. We showed in chapter(15) that PCA goes through the mean of the data.

17.3 1. When $C = 2$ we have

$$p(c = 1 | \mathbf{x}) = \frac{e^{\mathbf{w}_1^\top \mathbf{x}}}{e^{\mathbf{w}_1^\top \mathbf{x}} + e^{\mathbf{w}_2^\top \mathbf{x}}} = \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^\top \mathbf{x}}} = \sigma(\mathbf{w}_2 - \mathbf{w}_1^\top \mathbf{x})$$

For $C = 2$, this is therefore logistic regression parameterised by the difference $\mathbf{w} \equiv \mathbf{w}_2 - \mathbf{w}_1$. Note that there is therefore redundancy in the softmax parameterisation since only this difference plays a role in computing the probability. The same hold for $C > 2$ – by multiplying and dividing by $\exp(-\mathbf{w}_d^\top \mathbf{x})$, for some chosen $d \in \{1, \dots, C\}$ only the differences $\mathbf{w}'_c \equiv \mathbf{w}_c - \mathbf{w}_d$ are required to specify the probability. This is potentially important to realise for optimisation purposes since there will therefore be equivalent solutions in the redundant parameterisation.

2.

$$L(\mathcal{D}) = \sum_n \left[\mathbf{w}_{c^n}^\top \mathbf{x}^n - \log \sum_{d=1}^C e^{\mathbf{w}_d^\top \mathbf{x}^n} \right]$$

3.

$$\frac{\partial}{\partial \mathbf{w}_c} L = \sum_n \left[\mathbb{I}[c^n = c] \mathbf{x}^n - \frac{\mathbf{x}^n e^{\mathbf{w}_c^\top \mathbf{x}^n}}{\sum_{d=1}^C e^{\mathbf{w}_d^\top \mathbf{x}^n}} \right]$$

$$\frac{\partial^2}{\partial \mathbf{w}_c \partial \mathbf{w}_{c'}^\top} L = - \sum_n \mathbf{x}^n (\mathbf{x}^n)^\top \frac{Z_n \delta(c, c') e^{\mathbf{w}_c^\top \mathbf{x}^n} - e^{\mathbf{w}_c^\top \mathbf{x}^n} e^{\mathbf{w}_{c'}^\top \mathbf{x}^n}}{Z_n^2}$$

where $Z_n \equiv \sum_{d=1}^C e^{\mathbf{w}_d^\top \mathbf{x}^n}$. Defining $\pi_c^n = p(c^n | \mathbf{x}^n)$, we have

$$\frac{\partial^2}{\partial \mathbf{w}_c \partial \mathbf{w}_{c'}^\top} L = - \sum_n \mathbf{x}^n (\mathbf{x}^n)^\top (\delta(c, c') \pi_c^n - \pi_c^n \pi_{c'}^n)$$

Then for a vector

$$\mathbf{z} = \begin{pmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_C \end{pmatrix}$$

$$\begin{aligned}
\mathbf{z}^\top \mathbf{H} \mathbf{z} &= \sum_c \mathbf{z}_c \mathbf{H}_{cc} \mathbf{z}_c + \sum_{c \neq c'} \mathbf{z}_c \mathbf{H}_{cc'} \mathbf{z}_{c'} \\
&= - \sum_n \left[\sum_c (\mathbf{z}_c)^\top \mathbf{x}^n (\mathbf{x}^n)^\top \mathbf{z}_c \left(\pi_c^n - (\pi_c^n)^2 \right) - \sum_{c \neq c'} (\mathbf{z}_c)^\top \mathbf{x}^n (\mathbf{x}^n)^\top (\mathbf{z}_{c'}) \pi_c^n \pi_{c'}^n \right]
\end{aligned}$$

Let $y_c^n \equiv (\mathbf{x}^n)^\top \mathbf{z}_c$. Then

$$\begin{aligned}
\mathbf{z}^\top \mathbf{H} \mathbf{z} &= - \sum_n \left[\sum_c (y_c^n)^2 \left(\pi_c^n - (\pi_c^n)^2 \right) - \sum_{c \neq c'} y_c^n y_{c'}^n \pi_c^n \pi_{c'}^n \right] \\
&= \sum_n \left[- \sum_c (y_c^n)^2 \pi_c^n + \sum_{c, c'} y_c^n y_{c'}^n \pi_c^n \pi_{c'}^n \right]
\end{aligned}$$

Since π_c^n is a distribution in c , we have

$$\mathbf{z}^\top \mathbf{H} \mathbf{z} = - \sum_n \left(\langle (y^n)^2 \rangle - \langle y^n \rangle^2 \right) \leq 0$$

The last step follows since each variance is ≥ 0 .

17.4 Using

$$C \xi^n = \alpha^n$$

Then

$$\xi^n = \alpha^n / C \rightarrow \sum_n (\xi^n)^2 = \sum_n (\alpha^n)^2 / C^2$$

and

$$\sum_n \alpha^n \xi^n = \frac{1}{C} \sum_n (\alpha^n)^2$$

Also

$$\sum_n \alpha y^n \mathbf{w}^\top \mathbf{x}^n = \mathbf{w}^\top \mathbf{w}$$

Hence the objective is to optimise

$$\left(\frac{1}{2} - 1 \right) \mathbf{w}^\top \mathbf{w} + \frac{1}{C} \left(\frac{1}{2} - 1 \right) \sum_n (\alpha^n)^2 + \sum_n \alpha^n$$

Using $\mathbf{w} = \sum_n \alpha^n y^n \mathbf{x}^n$,

$$\mathbf{w}^\top \mathbf{w} = \sum_n \alpha^n y^n \mathbf{x}^n \sum_m \alpha^m y^m \mathbf{x}^m = \sum_{n, m} y^n y^m \alpha^n \alpha^m,$$

we obtain the desired form. Note that provided $C \geq 0$, the constraint $\xi^n \geq 0$ is automatically satisfied since $\xi^n = \alpha^n / C \geq 0$. Also $L(\alpha)$ is negative semidefinite wrt α so that the Lagrange stationary requirement means that we need to find a minimum *w.r.t.* α .

17.5

$$\mathbf{y} = \mathbf{M} \mathbf{x} \tag{B.0.128}$$

The argument of the sigmoid $\sigma(h)$ is

$$h = \mathbf{w}^\top \dot{\mathbf{x}} + b = \mathbf{w}^\top \mathbf{M} \dot{\mathbf{x}} + b = (\mathbf{M}^\top \mathbf{w})^\top \mathbf{x} + b = \tilde{\mathbf{w}}^\top \mathbf{x} + b \tag{B.0.129}$$

PCA is a linear projection of the data and, since the argument of the logistic function is also a linear function of the data, overall using PCA first results still in a linear decision boundary. However, the problem is constrained since we have forced linear regression to work in the subspace spanned by the PCA vectors. Consider 100 training vectors randomly positioned in a 1000 dimensional space each with a random class 0 or 1. With very high probability, these 100 vectors will be linearly separable. Now project these vectors onto a 10 dimensional space: with very high probability, 100 vectors plotted in a 10 dimensional space will *not* be linearly separable. Hence, attention should be paid to the fact that using PCA first could potentially transform a linearly separable problem into a non-linearly separable problem.

17.6

$$\sigma^{-1}(x) = \log \left(\frac{\sigma(x)}{\sigma(x)(1 - \sigma(x))} \right)$$

17.7

$$L = \sum_n c^n \log \sigma(\mathbf{w}^\top \mathbf{x}^n + b) + (1 - c^n) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}^n + b))$$

Using the derivative result $\sigma' = \sigma(1 - \sigma)$, and $\sigma_n \equiv \sigma(\mathbf{w}^\top \mathbf{x}^n + b)$ we have

$$\nabla_{\mathbf{w}} L = \sum_{n=1} c^n (1 - \sigma_n) \mathbf{x}^n - (1 - c^n) \sigma_n \mathbf{x}^n = \sum_{n=1} (c^n - \sigma_n) \mathbf{x}^n$$

17.8 1. Since the data is linearly separable, for class +1 data and class -1 data

$$\mathbf{w}^\top \mathbf{x}_+ + b \geq 0$$

and

$$\mathbf{w}^\top \mathbf{x}_- + b \leq 0$$

Multiplying by a positive scalar preserves the inequalities:

$$\lambda \mathbf{w}^\top \mathbf{x}_+ + \lambda b \geq 0$$

and

$$\lambda \mathbf{w}^\top \mathbf{x}_- + \lambda b \leq 0$$

which demonstrates the required result.

2. In terms of maximum likelihood this means that we can make \mathbf{w} and b as large as we like, once we have a separable solution. For logistic regression, the most probable settings are when the σ functions saturate to 1 (or 0 in the case of negative datapoints). In order for this to happen the argument $\mathbf{w}^\top \mathbf{x} + b$ needs to go to $\pm\infty$, which means that \mathbf{w} and b need to become infinite. This means that for any incremental training procedure for maximum likelihood, this will never terminate since the weights will continue to increase without bound. We therefore need to introduce a termination criterion.

17.9 In general, we may assume that the data is linearly separable since there are N points in an N dimensional space. For a given set of $\epsilon^n > 0$, we can therefore set $b = 0$ and use $\mathbf{w} = \mathbf{X}^{-\top} \boldsymbol{\epsilon}$ where $\mathbf{X} = [\mathbf{x}^1, \dots, \mathbf{x}^N]$. This will result in a finite length \mathbf{w} . This cannot be equivalent to the maximum likelihood solution for logistic regression, which would in principal have infinite length.

17.10 1.

$$L = \sum_n c^n \log \sigma(f(\mathbf{x}^n)) + (1 - c^n) \log(1 - \sigma(f(\mathbf{x}^n)))$$

where

$$f(\mathbf{x}) \equiv v_0 + v_1 g(\mathbf{w}_1^\top \mathbf{x} + b_1) + v_2 g(\mathbf{w}_2^\top \mathbf{x} + b_2)$$

2.

$$\frac{\partial}{\partial \theta} L = \sum_n (c^n (1 - \sigma^n) - (1 - c^n) \sigma^n) \frac{\partial}{\partial \theta} f(\mathbf{x}^n) = \sum_n (c^n - \sigma^n) \frac{\partial}{\partial \theta} f(\mathbf{x}^n)$$

$$\frac{\partial}{\partial \mathbf{w}_i} f(\mathbf{x}^n) = -v_i (\mathbf{w}_i^\top \mathbf{x}^n + b_i) e^{-(\mathbf{w}_i^\top \mathbf{x}^n + b_i)^2 / 2} \mathbf{x}^n$$

For

$$\frac{\partial}{\partial b_i} f(\mathbf{x}^n) = -v_i (\mathbf{w}_i^\top \mathbf{x}^n + b_i) e^{-(\mathbf{w}_i^\top \mathbf{x}^n + b_i)^2 / 2}$$

For $i = 1, 2$

$$\frac{\partial}{\partial v_i} f(\mathbf{x}^n) = g(\mathbf{w}_i^\top \mathbf{x}^n + b_i)$$

$$\frac{\partial}{\partial v_0} f(\mathbf{x}^n) = 1$$

3. This is a non-linear form of logistic regression.
 4. Given a trained set of parameters, the decision boundary is given by those \mathbf{x} that satisfy:

$$0 = v_0 + v_1 g(\mathbf{w}_1^\top \mathbf{x} + b_1) + v_2 g(\mathbf{w}_2^\top \mathbf{x} + b_2)$$

Given any solution \mathbf{x} , then $\mathbf{x} + \mathbf{w}_1^\perp + \mathbf{w}_2^\perp$, where \mathbf{w}_1^\perp is orthogonal to \mathbf{w}_1 and \mathbf{w}_2^\perp is orthogonal to \mathbf{w}_2 will also satisfy this equation. This means that the decision boundary will be an $N - 2$ dimensional non-linear manifold.

18.1 1.

$$p(f|\mathbf{x}) = \int \int \delta(f - \mathbf{w}^\top \mathbf{x}) p(\mathbf{w}) d\mathbf{w}$$

From our general results, in chapter(8), we know that f is Gaussian distributed since it is linear in \mathbf{w} and \mathbf{w} itself is Gaussian. The mean and variance are given by

$$\langle f \rangle = \mathbf{x}^\top \langle \mathbf{w} \rangle = 0, \quad \langle f^2 \rangle = \mathbf{x}^\top \langle \mathbf{w} \mathbf{w}^\top \rangle \mathbf{x} = \mathbf{x}^\top \Sigma \mathbf{x}$$

2. This is a Gaussian distribution, which we can find from conditioning:

$$\langle t \rangle = \langle f \rangle = 0, \quad \langle t^2 \rangle = \langle f^2 \rangle + \sigma^2, \quad \langle tf \rangle = \langle f^2 \rangle$$

Using Gaussian conditioning (the partitioned Gaussian results),

$$p(f|t, \mathbf{x}) = \mathcal{N}(f | (t \langle f^2 \rangle / (\langle f^2 \rangle + \sigma^2), \langle f^2 \rangle - \langle f^2 \rangle \langle f^2 \rangle / (\langle f^2 \rangle + \sigma^2))$$

18.3

$$p(\mathbf{y}_{val} | \mathcal{X}_{train}, \mathcal{Y}_{train}, \mathcal{X}_{val}) = \int p(\mathbf{y}_{val} | \mathcal{X}_{val}, \mathbf{w}) p(\mathbf{w} | \mathcal{X}_{train}, \mathcal{Y}_{train})$$

Using

$$\Phi_{val}^\top = [\phi(\mathbf{x}_{val}^1), \dots, \phi(\mathbf{x}_{val}^M)]$$

we can write

$$\mathbf{y}_{val} = \Phi_{val} \mathbf{w} + \boldsymbol{\eta}_{val}$$

Since this is linearly related to \mathbf{w} which itself is Gaussian, then \mathbf{y}_{val} is Gaussian. Its mean is

$$\langle \mathbf{y}_{val} \rangle = \Phi_{val} \underbrace{\langle \mathbf{w} \rangle}_{\mathbf{m}} + \underbrace{\langle \boldsymbol{\eta}_{val} \rangle}_{\mathbf{0}}$$

The covariance is obtained from

$$\langle [\mathbf{y}_{val} - \langle \mathbf{y}_{val} \rangle] [\mathbf{y}_{val} - \langle \mathbf{y}_{val} \rangle]^\top \rangle = \Phi_{val} \langle (\mathbf{w} - \mathbf{m})(\mathbf{w} - \mathbf{m})^\top \rangle \Phi_{val}^\top + \langle \boldsymbol{\eta}_{val} \boldsymbol{\eta}_{val}^\top \rangle$$

The cross terms involving $\langle \boldsymbol{\eta}_{val} \mathbf{w} \rangle$ are zero since $\boldsymbol{\eta}_{val}$ and \mathbf{w} are uncorrelated and $\boldsymbol{\eta}_{val}$ has zero mean. Thus

$$\mathbf{C}_{val} = \Phi_{val} \Sigma \Phi_{val}^\top + \sigma^2 \mathbf{I}_M$$

Hence

$$p(\mathbf{y}_{val} | \mathcal{X}_{train}, \mathcal{Y}_{train}, \mathcal{X}_{val}) = \mathcal{N}(\mathbf{y}_{val} | \Phi_{val} \mathbf{m}, \mathbf{C}_{val})$$

Taking logs gives the required result.

18.4 1.

$$\frac{\partial}{\partial w_i} E = \alpha w_i - \sum_n \frac{1}{\sigma_n} \sigma_n (1 - \sigma_n) h_i^n = \alpha w_i - \sum_n (1 - \sigma_n) h_i^n$$

$$\frac{\partial^2}{\partial w_i \partial w_j} E = \alpha \delta_{ij} + \sum_n \sigma_n (1 - \sigma_n) h_i^n h_j^n = \left[\alpha \mathbf{I} + \sum_{n=1}^N \sigma^n (1 - \sigma^n) \mathbf{h}^n (\mathbf{h}^n)^\top \right]_{ij}$$

Since $\mathbf{h}^n = (2c^n - 1)\phi^n$

$$\mathbf{h}^n (\mathbf{h}^n)^\top = \underbrace{(2c^n - 1)^2}_1 \phi^n (\phi^n)^\top$$

This gives the desired result.

2. Since $\sigma(1 - \sigma) \geq 0$ all data terms in the Hessian are positive sums of outer products of vectors and hence positive semidefinite. The additional prior term $\alpha \mathbf{I}$ is positive definite, so that \mathbf{H} is overall positive definite.

18.5

$$\int f(\mathbf{x}^\top \mathbf{w}) p(\mathbf{w}) d\mathbf{w} = \int f(\mathbf{x}^\top v \mathbf{w}) p(h, \mathbf{w}) d\mathbf{w} dh = \int f(\mathbf{x}^\top \mathbf{w}) p(h | \mathbf{w}) p(\mathbf{w}) d\mathbf{w} dh \quad (\text{B.0.130})$$

$$= \int f(\mathbf{x}^\top \mathbf{w}) \delta(h - \mathbf{x}^\top \mathbf{w}) p(\mathbf{w}) d\mathbf{w} dh = \int f(h) \left\{ \delta(h - \mathbf{x}^\top \mathbf{w}) p(\mathbf{w}) d\mathbf{w} \right\} dh = \int f(h) p(h) dh \quad (\text{B.0.131})$$

18.6

$$\frac{\partial}{\partial \alpha} L = -\frac{1}{2}(\mathbf{w}^*)^T \mathbf{w}^* - \frac{1}{2} \text{trace}((\alpha \mathbf{I} + \mathbf{J})^{-1}) + \frac{B}{2\alpha}$$

To find the optimum of $L(\alpha)$ we set the derivative to zero and solve for α . This gives that the optimum α satisfies:

$$\alpha = \frac{N}{(\mathbf{w}^*)^T \mathbf{w}^*} + \text{trace}((\alpha \mathbf{I} + \mathbf{J})^{-1})$$

We can make a fixed point equation from this, as described in the text.

19.1 1.

$$\sum_{ij} y_i K_{ij}^+ y_j = \sum_{ij} y_i (K_{ij}^1 + K_{ij}^2) y_j = \underbrace{\sum_{ij} y_i K_{ij}^1 y_j}_{\geq 0} + \underbrace{\sum_{ij} y_i K_{ij}^2 y_j}_{\geq 0} \geq 0$$

2.

$$\sum_{ij} y_i K_{ij}^* y_j = \sum_{lm} \sum_{ij} y_i u_{il} u_{jl} v_{im} v_{jm} y_j = \sum_{lm} \underbrace{\left(\sum_i y_i u_{il} v_{im} \right)}_{z_{lm}} \underbrace{\left(\sum_j u_{jl} v_{jm} y_j \right)}_{z_{lm}} \geq 0$$

19.2

$$\sum_{ij} y_i S_{ij} y_j = \sum_{ij} \left(\frac{1}{N} \sum_n x_i^n y_i x_j^n y_j - \frac{1}{N^2} \sum_n x_i^n y_i \sum_m x_j^m y_j \right)$$

Let $z_n \equiv \sum_i y_i x_i^n$. Then

$$\sum_{ij} y_i S_{ij} y_j = \frac{1}{N} \sum_n z_n^2 - \frac{1}{N^2} \sum_n z_n \sum_m z_m = \frac{1}{N} \sum_n \left(z_n - \frac{1}{N} \sum_m z_m \right)^2 \geq 0$$

19.3 Consider the vector

$$\mathbf{u}(x) = \lambda \begin{pmatrix} \cos \omega x \\ \sin \omega x \end{pmatrix}$$

Then a valid kernel is given by

$$k(x, x') = e^{-|\mathbf{u}(x) - \mathbf{u}(x')|}$$

We can simplify the exponent as follows:

$$\begin{aligned} |\mathbf{u}(x) - \mathbf{u}(x')|^2 &= \mathbf{u}(x)^T \mathbf{u}(x) + \mathbf{u}(x')^T \mathbf{u}(x') - 2\mathbf{u}(x)^T \mathbf{u}(x') \\ &= 2\lambda^2 (1 - \mathbf{u}(x)^T \mathbf{u}(x')) = 2\lambda^2 (1 - \cos(\omega(x - x'))) \\ &= 4\lambda^2 \sin^2 \left(\frac{\omega(x - x')}{2} \right) \end{aligned}$$

Hence, setting $\lambda = 1/2$ and $\omega = 2$, we obtain the covariance function

$$k(x - x') = e^{-|\sin(x - x')|}$$

19.4 [Renato Vicente]

$$e^{x_i x_j} = \sum_{k=0}^{\infty} \frac{1}{k!} (x_i x_j)^k = \sum_{k=0}^{\infty} \left(\frac{x_i}{\sqrt{k!}} \right)^k \left(\frac{x_j}{\sqrt{k!}} \right)^k \quad (\text{B.0.132})$$

Note that this expansion has infinite radius of convergence. Using this expansion, we have

$$f(x_i, x_j) = \sum_{k=0}^{\infty} e^{-\frac{1}{2} x_i^2} \underbrace{\left(\frac{x_i}{\sqrt{k!}} \right)^k \left(\frac{x_j}{\sqrt{k!}} \right)^k}_{g_k(x_j)} e^{-\frac{1}{2} x_j^2} \quad (\text{B.0.133})$$

Thus

$$\sum_{ij} z_i f(x_i, x_j) z_j = \sum_{k=0}^{\infty} \left(\sum_i z_i g_k(x_i) \right) \left(\sum_j z_j g_k(x_j) \right) = \sum_{k=0}^{\infty} \left(\sum_i z_i g_k(x_i) \right)^2 \geq 0 \quad (\text{B.0.134})$$

This construction also gives an explicit infinite dimensional feature representation, corresponding to a non-orthogonal basis.

19.5 Since the product of two covariance functions is a covariance function, by repeatedly applying this rule, we have that the integer power of a covariance function, $k^n(x, x')$ is a covariance function. Also, since the sum of two covariance functions is a covariance function and $\lambda k(x, x')$ is a covariance function for $\lambda \geq 0$, then the power series

$$\sum_n \lambda_n k^n(x, x')$$

is a covariance function for $\lambda_n \geq 0$. Since

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Then $e^{k(x, x')}$ is expressible as an infinite sum of positive coefficient powers of the covariance function $k(x, x')$ and hence itself a covariance function. The same holds for $\tan k_1(x, x')$ since

$$\tan x = x + \frac{x^3}{3} + \frac{2x^5}{15} + \dots$$

19.6 By defining $\mathbf{y} = \mathbf{A}^{1/2} \mathbf{x}$, $\mathbf{y}' = \mathbf{A}^{1/2} \mathbf{x}'$,

$$k_2(\mathbf{x}, \mathbf{x}') = f\left((\mathbf{y} - \mathbf{y}')^\top (\mathbf{y} - \mathbf{y}')\right) = k_1(\mathbf{y}, \mathbf{y}')$$

Since k_1 is a covariance function, so is k_2 .

19.8 1. First we define a set of substrings of interest, for example 'goal', 'score', 'policy', 'fiscal', *etc.*, and a set of positive weights \mathbf{w} for these substrings. Then for two documents x and x' , we can compute the covariance function $k(x, x')$ using the string kernel. This can then be used as part of the standard GP classifier approach, when faced with a novel example x^* .

2. The Laplace method approximates the log likelihood of the training data, as in equation (19.5.30). The \mathbf{w} can then be treated as hyperparameters, as in section(19.5.3). The derivatives are obtained using the formula in exercise(19.7), where

$$\frac{\partial}{\partial w_s} \mathbf{K}_{\mathbf{x}, \mathbf{x}} = \phi_s(x_i) \phi_s(x_j)$$

19.9

$$p(\mathbf{y}^* | \mathbf{x}^*, \mathcal{X}, \mathcal{Y}) \propto p(\mathbf{y}^*, \mathcal{Y} | \mathbf{x}^*, \mathcal{X})$$

This requires us to specify a Gaussian covariance over the joint outputs for two difference \mathbf{y} , \mathbf{y}' . Since the dimensions are assumed independent, then

$$p(\mathbf{y}, \mathbf{y}' | \mathbf{x}, \mathbf{x}') = \prod_i p(y_i, y'_i | \mathbf{x}, \mathbf{x}')$$

Hence we can form a GP predictor based on $p(y_i, y'_i | \mathbf{x}, \mathbf{x}')$ for each input dimension alone.

19.10 1. By definition, \mathbf{x}_1 is Gaussian. Then $p(\mathbf{x}_2, \mathbf{x}_1) = p(\mathbf{x}_2 | \mathbf{x}_1) p(\mathbf{x}_1)$ is also jointly Gaussian since \mathbf{x}_2 depends linearly on \mathbf{x}_1 , $\mathbf{x}_2 = \mathbf{A} \mathbf{x}_1 + \boldsymbol{\eta}_2$. Since the sum of two Gaussian distributed variables is Gaussian distributed, then \mathbf{x}_2 is Gaussian distributed. Hence $p(\mathbf{x}_2, \mathbf{x}_1)$ is Gaussian. By repeating this, one sees that $\mathbf{x}_1, \dots, \mathbf{x}_t$ is jointly Gaussian.

2. By iterating, we have

$$\mathbf{x}_t = \mathbf{A} (\mathbf{A} \mathbf{x}_{t-2} + \boldsymbol{\eta}_{t-1}) + \boldsymbol{\eta}_t = \mathbf{A}^2 \mathbf{x}_{t-2} + \mathbf{A} \boldsymbol{\eta}_{t-1} + \boldsymbol{\eta}_t = \mathbf{A}^{t-1} \mathbf{x}_1 + \sum_{\tau=2}^t \mathbf{A}^{t-\tau} \boldsymbol{\eta}_\tau$$

Then

$$\langle \mathbf{x}_{t'} \mathbf{x}_t \rangle = \left\langle \left[\mathbf{A}^{t'-1} \mathbf{x}_1 + \sum_{\tau=2}^{t'} \mathbf{A}^{t'-\tau} \boldsymbol{\eta}_\tau \right] \left[\mathbf{A}^{t-1} \mathbf{x}_1 + \sum_{\rho=2}^t \mathbf{A}^{t-\rho} \boldsymbol{\eta}_\rho \right]^\top \right\rangle$$

Since $\langle \mathbf{x} \boldsymbol{\eta}_\tau^\top \rangle = \mathbf{0}$ and $\langle \boldsymbol{\eta}_\tau \boldsymbol{\eta}_\rho^\top \rangle = \sigma^2 \mathbf{I} \delta_{\tau, \rho}$, this reduces to

$$\langle \mathbf{x}_{t'} \mathbf{x}_t \rangle = \mathbf{A}^{t'-1} \boldsymbol{\Sigma} (\mathbf{A}^{t-1})^\top + \sigma^2 \sum_{\tau=2}^{t'} \sum_{\rho=2}^t \delta_{\tau, \rho} \mathbf{A}^{t'-\tau} (\mathbf{A}^{t-\rho})^\top = \mathbf{A}^{t'-1} \boldsymbol{\Sigma} (\mathbf{A}^{t-1})^\top + \sigma^2 \sum_{\tau=2}^{\min(t, t')} \mathbf{A}^{t'-\tau} (\mathbf{A}^{t-\tau})^\top$$

This therefore specifies the covariance matrix explicitly in terms of \mathbf{A} and $\boldsymbol{\Sigma}$. This is a Gaussian Process since the joint distribution of $\mathbf{x}_1, \dots, \mathbf{x}_t$ is Gaussian, with covariance function given by equation (19.8.18).

3. Since the \mathbf{x} are jointly Gaussian, and $\mathbf{y}_1, \dots, \mathbf{y}_t$ is linearly dependent on $\mathbf{x}_1, \dots, \mathbf{x}_t$, then $\mathbf{y}_1, \dots, \mathbf{y}_t$ is Gaussian. This indeed a zero mean Gaussian Process with covariance function

$$\langle \mathbf{y}_{t'} \mathbf{y}_t^\top \rangle = \langle (\mathbf{B} \mathbf{x}_{t'} + \boldsymbol{\epsilon}_{t'}) (\mathbf{B} \mathbf{x}_t + \boldsymbol{\epsilon}_t)^\top \rangle = \mathbf{B} \langle \mathbf{x}_{t'} \mathbf{x}_t^\top \rangle \mathbf{B}^\top$$

20.1 Since the data is assumed i.i.d., we can concentrate on the observations n . In this case

$$p(v_1^n, \dots, v_{i-1}^n, v_{i+1}^n, \dots, v_N^n) = \int_{v_i^n} p(v) = \sum_h p(h) \underbrace{\left[\int_{v_i^n} p(v_i|h) \right]}_1 \left[\prod_{j \neq i} p(v_j^n|h) \right]$$

Hence

$$p(v_1^n, \dots, v_{i-1}^n, v_{i+1}^n, \dots, v_N^n) = \sum_h p(h) \prod_{j \neq i} p(v_j^n|h)$$

which is equivalent to ‘ignoring’ the i^{th} component of datapoint n .

20.2 In this case one can readily adjust equation (20.3.12) to give

$$\text{trace} \left(\mathbf{S}_i^{-1} \sum_{n=1}^N p^{old}(i|\mathbf{x}^n) \Delta_i^n (\Delta_i^n)^\top \right) - \log \det \left(\mathbf{S}_i^{-1} \right) \sum_{n=1}^N p^{old}(i|\mathbf{x}^n) \quad (\text{B.0.135})$$

where $\mathbf{S}_i = \text{diag}(d_1^i, \dots, d_D^i)$. The determinant is the product of the diagonal entries for a diagonal matrix. The contribution from d_j^i is therefore

$$\sum_{n=1}^N p^{old}(i|\mathbf{x}^n) (\Delta_i^n)^\top \Delta_i^n \frac{1}{d_j^i} + \log d_j^i \sum_{n=1}^N p^{old}(i|\mathbf{x}^n) \quad (\text{B.0.136})$$

Differentiating with respect to d_j^i and equating to zero gives

$$d_j^i = \sum_{n=1}^N p^{old}(i|\mathbf{x}^n) (\Delta_i^n)^\top \Delta_i^n \quad (\text{B.0.137})$$

20.3

$$p(n|i) = \frac{p(i|\mathbf{x}^n)}{\sum_n p(i|\mathbf{x}^n)} = \frac{p(\mathbf{x}^n|i)p(i)}{\sum_n p(\mathbf{x}^n|i)p(i)}$$

Using the GMM form this is

$$p(n|i) = \frac{e^{-\frac{1}{2\sigma^2}(\mathbf{x}^n - \mathbf{m}_i)^2}}{\sum_n e^{-\frac{1}{2\sigma^2}(\mathbf{x}^n - \mathbf{m}_i)^2}}$$

As σ^2 becomes very small, the numerator term becomes exponentially small. However, given i , there will be an \mathbf{x}^n that is closest to \mathbf{m}_i . Whilst the numerator is small for this n , it is exponentially larger than for any other $n' \neq n$. By normalisation, it is therefore the case that $p(n|i)$ tends to 1 for that n such that \mathbf{x}^n is closest to \mathbf{m}_i , with all other elements of $p(n|i)$ tending to zero.

20.4

$$\frac{\partial}{\partial p(h)} L = \sum_n \frac{\partial}{\partial p(h)} \sum_{h'} p^{old}(h'|v^n) \log p(h') - \lambda = \sum_n p^{old}(h|v^n) \frac{1}{p(h)} - \lambda$$

Equating to zero for the optimum, we have

$$0 = \frac{1}{p(h)} \sum_n p^{old}(h|v^n) - \lambda$$

hence

$$p(h) = \frac{1}{\lambda} \sum_n p^{old}(h|v^n)$$

Since $\sum_h p(h) = 1$, we have the desired result.

20.5 The maximum likelihood solution for the covariance is

$$\mathbf{\Sigma} = \frac{1}{N} \sum_n (\mathbf{x}^n - \boldsymbol{\mu})(\mathbf{x}^n - \boldsymbol{\mu})^\top$$

where $\boldsymbol{\mu}$ is given the data mean. Hence $\mathbf{\Sigma}$ always remains finite (for bounded data).

20.7 Firstly, any matrix formed from an outerproduct $\mathbf{Z}_* \mathbf{Z}_*^\top$ is positive semidefinite. Since \mathbf{Z} is a clique matrix for \mathbf{A} , we must have, for $S_{ij} = 0$

$$\sum_k Z_{ik}^* Z_{kj}^* = \sum_k Z_{ik} Z_{kj} \theta_{ik} \theta_{kj}$$

Since for $S_{ij} = 0$, $\sum_k Z_{ik} Z_{kj} = 0$ and $Z_{ik} \in \{0, 1\}$, then for all k , $Z_{ik} Z_{kj} = 0$. This means that for $S_{ij} = 0$

$$[\mathbf{S}_*]_{ij} = \sum_k Z_{ik}^* Z_{kj}^* = 0$$

Hence \mathbf{S}_* has the same zeros structure as \mathbf{S} and is a valid semidefinite covariance matrix for any θ .

21.1 For a factor analysis model

$$\mathbf{x} = \mathbf{M}\mathbf{h} + \boldsymbol{\eta}$$

where $\boldsymbol{\eta}$ is noise from a diagonal matrix. Then

$$\mathbf{C}\mathbf{x} = \mathbf{C}\mathbf{M}\mathbf{h} + \mathbf{C}\boldsymbol{\eta}$$

or

$$\mathbf{y} = \mathbf{M}'\mathbf{h} + \boldsymbol{\eta}'$$

If $\boldsymbol{\eta}$ has diagonal covariance \mathbf{D} then $\boldsymbol{\eta}'$ has diagonal covariance $\text{diag}(C_{ii}^2 D_{ii})$.

21.2 By writing

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_A \\ \mathbf{x}_B \end{pmatrix}$$

The contribution of \mathbf{A} and \mathbf{B} to the energy term is proportional to

$$\sum_n \langle (\mathbf{x}_A - \mathbf{A}\mathbf{h})^\top \boldsymbol{\Psi}_A (\mathbf{x}_A - \mathbf{A}\mathbf{h}) \rangle_n + \sum_n \langle (\mathbf{x}_B - \mathbf{B}\mathbf{h})^\top \boldsymbol{\Psi}_B (\mathbf{x}_B - \mathbf{B}\mathbf{h}) \rangle_n$$

Since the contributions of \mathbf{A} and \mathbf{B} are separate, and each is exactly of the standard FA form, we can use the standard M-step to arrive at

$$\mathbf{A} = \mathbf{M}_A \mathbf{H}^{-1}$$

where

$$\mathbf{M}_A \equiv \frac{1}{N} \sum_n (\mathbf{x}_A^n - \bar{\mathbf{x}}_A) \langle \mathbf{h} \rangle_{q(\mathbf{h}|\mathbf{x}^n)}^\top, \quad \mathbf{H} = \frac{1}{N} \sum_n \langle \mathbf{h}\mathbf{h}^\top \rangle_{q(\mathbf{h}|\mathbf{x}^n)}$$

An analogous update holds for \mathbf{B} .

21.3 Under this prior the marginal distribution of the visible variables is

$$\mathbf{v} \sim \mathcal{N}(\mathbf{v}|\mathbf{c}, \mathbf{F}\boldsymbol{\Sigma}_H\mathbf{F}^\top + \boldsymbol{\Psi}) \quad (\text{B.0.138})$$

By defining a new covariance matrix

$$\tilde{\mathbf{F}} \equiv \mathbf{F}\boldsymbol{\Sigma}_H^{\frac{1}{2}} \quad (\text{B.0.139})$$

where we define a square root (Cholesky factor for example) such that $\boldsymbol{\Sigma}_H^{\frac{1}{2}} \boldsymbol{\Sigma}_H^{\frac{1}{2}\top} = \boldsymbol{\Sigma}_H$, then

$$\mathbf{v} \sim \mathcal{N}(\mathbf{v}|\mathbf{c}, \tilde{\mathbf{F}}\tilde{\mathbf{F}}^\top + \boldsymbol{\Psi}) \quad (\text{B.0.140})$$

This is of the same for as the FA for an uncorrelated prior, simply with a renaming of \mathbf{F} . There is therefore nothing gained from using a correlated Gaussian prior $p(\mathbf{h})$.

21.4

$$\begin{aligned} \boldsymbol{\Sigma}_D^{-1}\mathbf{F} &= (\mathbf{F}\mathbf{F}^\top + \boldsymbol{\Psi})^{-1}\mathbf{F} \\ &= \left[\boldsymbol{\Psi}^{-1} - \boldsymbol{\Psi}^{-1}\mathbf{F}(\mathbf{I} + \mathbf{F}^\top\boldsymbol{\Psi}^{-1}\mathbf{F})^{-1}\mathbf{F}^\top\boldsymbol{\Psi}^{-1} \right] \mathbf{F} \\ &= \boldsymbol{\Psi}^{-1}\mathbf{F} - \boldsymbol{\Psi}^{-1}\mathbf{F}(\mathbf{I} + \mathbf{F}^\top\boldsymbol{\Psi}^{-1}\mathbf{F})^{-1}(\mathbf{F}^\top\boldsymbol{\Psi}^{-1}\mathbf{F} + \mathbf{I} - \mathbf{I}) \\ &= \cancel{\boldsymbol{\Psi}^{-1}\mathbf{F}} - \cancel{\boldsymbol{\Psi}^{-1}\mathbf{F}} + \boldsymbol{\Psi}^{-1}\mathbf{F}(\mathbf{I} + \mathbf{F}^\top\boldsymbol{\Psi}^{-1}\mathbf{F})^{-1} \end{aligned}$$

21.5 Differentiating *w.r.t.* σ^2 and equating to zero immediately gives the result.

21.6 1. The dependence of the energy on \mathbf{W} is proportional to

$$\left\langle (\mathbf{y}^n - \mathbf{W}\mathbf{x}^n)^2 \right\rangle_{q(\mathbf{x}|\mathbf{y}^n, \mathbf{W}^{old})}$$

Since this is quadratic in \mathbf{x} we only need the statistics $\langle \mathbf{x} \rangle$, $\langle \mathbf{x}\mathbf{x}^T \rangle$.

2.

$$p(\mathbf{x}|\mathbf{y}, \mathbf{W}) \propto p(\mathbf{y}|\mathbf{x}, \mathbf{W})p(\mathbf{x}) = e^{-\frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{W}\mathbf{x})^2} \prod_i p(x_i)$$

This is non-Gaussian in \mathbf{x} since the priors $p(x_i)$ are non-Gaussian. In general, this distribution is not tractable since there is no coordinate rotation that will render this as a product of one dimensional integrals.

3. In the limit $\sigma^2 \rightarrow 0$, the term

$$e^{-\frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{W}\mathbf{x})^2}$$

dominates and

$$p(\mathbf{x}|\mathbf{y}, \mathbf{W}) = \delta(\mathbf{x} - \mathbf{x}_y)$$

where \mathbf{x}_y is that vector \mathbf{x} for which $(\mathbf{y} - \mathbf{W}\mathbf{x})^2$ is minimal.

Let's look at the M-step for \mathbf{w}_j . This is given by the minimum of

$$\sum_n \langle (\mathbf{y}^n - \mathbf{W}\mathbf{x}^n)^2 \rangle_{p(\mathbf{x}|\mathbf{y}^n, \mathbf{W}^{old})}$$

However, this is minimal for $\mathbf{W} = \mathbf{W}^{old}$, meaning that no update occurs and EM fails.

22.1

20, 1, 11, 13, 2, 16, 6, 8, 5, 7

See `exerciseRaschBronco.m`.

22.3 1. See `exerciseLaReine.m`

2.

59, 8, 77, 44, 39, 48, 78, 24, 98, 42

22.4 1. TBD

2. TBD

3. The highest positive weights indicate the most valuable player.

4. This can be done by looking at the probability that team 1 beats team 2 as a function of the binary factors of team 2. One can maximise this probability by searching over the space of all the binary factors.

23.1

$$\sum_{ij} M_{ij}e_j = \sum_j e_j = \lambda \sum_i e_i$$

Hence, provided $\sum_i e_i \neq 0$, any stochastic matrix \mathbf{M} has an eigenvector with unit eigenvalue.

23.2 The effect of this transition is simply to deterministically the state at each time step. Hence

$$p(x_t = 1) = \mathbb{I}[x_1 = 1] \int \text{tis odd} + \mathbb{I}[x_1 = 2] \int \text{tis even}$$

Hence, no matter how large t is, there remains a dependence on the initial condition. This means that there is no equilibrium distribution. For a stationary distribution we require

$$\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \mathbf{M} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

The distribution $p(x = 1) = p(x_2) = 1/2$ satisfies this and is therefore a stationary distribution.

23.3 This problem is small enough to be computed 'by hand'. It is of course easier to do this numerically, which can be achieved using the standard forward and backward routines. An inefficient, but perhaps instructive way to do this is to use `BRMLTOOLBOX`, see `exerciseHMMsimple.m`. This gives:

1. $p(v_{1:3}) = 0.153823$

2. $p(h_1|v_{1:3}) = \begin{pmatrix} 0.930933605507629 \\ 0.069066394492372 \\ 0 \end{pmatrix}$

3. $\arg \max_{h_{1:3}} p(h_{1:3}|v_{1:3}) = (1, 2, 3)$

23.4 The 659th most likely joint sequence is the first that is a correct sentence ‘the monkey is on the branch’, with $\log p(h_{1:27}|v_{1:27}) = -94.5250$. See `demoHMMbigramMonkey.m`.

23.5 Under these initial settings, the visible variables have no influence on the posterior $p^{old}(h_{1:T}|v_{1:T})$, which is uniform. Similarly, all the marginals such as $p^{old}(h_t, h_{t+1}|v_{1:T})$ are also uniform. From equation (23.3.5) we see that setting $p^{old}(i, i') = 1/H$ for a constant k gives $A_{i',i}^{new} = 1/H$. Similarly, $p^{new}(v_t|h_t)$ does not depend on h_t , $v_t \perp\!\!\!\perp h_t$ meaning that the visible variables still play no role in the updates. Hence the M-step retains the pure symmetry of the initialisation and no sensible update is made.

23.6 1. We need to carry out the computation:

$$\max_{v_{1:T}} \sum_{h_{1:T}} \prod_{t=1}^T p(v_t|h_t)p(h_t|h_{t-1})$$

In general, we are not able to interchange the order of the sum and the max, so that we cannot distribute all the required computations to make a recursive algorithm. Carrying out the summation first means that the resulting $p(v_{1:T})$ does not have any structure, being a single clique on $v_{1:T}$. If each observation as V states, the complexity of computing the most likely observation sequence is therefore $O(V^T)$.

2. The idea is to treat $v_{1:T}$ as ‘parameters’ in an EM algorithm:

$$\operatorname{argmax}_{v_{1:T}} \log p(v_{1:T}) = \operatorname{argmax}_{v_{1:T}} \log \sum_{h_{1:T}} p(v_{1:T}|h_{1:T})p(h_{1:T})$$

From the EM algorithm, we need to consider the M-step:

$$v_{1:T}^{new} = \operatorname{argmax}_{v_{1:T}} \sum_t \langle \log p(v_t|h_t)p(h_t|h_{t-1}) \rangle_{p(h_{1:T}|v_{1:T}^{old})} = \operatorname{argmax}_{v_{1:T}} \sum_t \langle \log p(v_t|h_t) \rangle_{p(h_t|v_{1:T}^{old})}$$

Hence the M-step is given by

$$v_t^{new} = \operatorname{argmax}_{v_t} \langle \log p(v_t|h_t) \rangle_{p(h_t|v_{1:T}^{old})}$$

In order to compute this update, we require the smoothed marginal $p(h_t|v_{1:T}^{old})$ which can be computed efficiently using the forward-backward algorithm. By virtue of this being an EM algorithm, updating in this way guarantees to find a more likely $v_{1:T}$ at each stage until we reach a local optimum.

23.7 In general, one may add a Lagrange term to the energy to enforce the constraint. In the case of an upper triangular structure, we may simply optimise the energy with respect to the upper triangular parts alone, setting the lower triangular parts to zero. This is equivalent to the standard M-step but setting the lower triangular part to zero and renormalising.

23.8 See `exerciseACGTHMM.m`

23.9 Write

$$\max_{h_1, \dots, h_T} \prod_{t=2}^T \phi_t(h_t, h_{t-1}) = \max_{h_1, \dots, h_{T-1}} \prod_{t=2}^{T-1} \phi_t(h_t, h_{t-1}) \underbrace{\max_{h_T} \phi_T(h_T, h_{T-1})}_{\gamma_{T-1 \leftarrow T}(h_{T-1})} \quad (\text{B.0.141})$$

Similarly

$$\max_{h_1, \dots, h_T} \prod_{t=2}^T \phi_t(h_t, h_{t-1}) = \max_{h_1, \dots, h_{T-2}} \prod_{t=2}^{T-2} \underbrace{\max_{h_{T-1}} \phi_T(h_{T-1}, h_{T-2}) \gamma_{T-1 \leftarrow T}(h_{T-1})}_{\gamma_{T-2 \leftarrow T-1}(h_{T-2})} \quad (\text{B.0.142})$$

which describes the recursion. Once we have repeated the recursion, we will have

$$\max_{h_1, \dots, h_T} \prod_{t=2}^T \phi_t(h_t, h_{t-1}) = \max_{h_1} \gamma_{1 \leftarrow 2}(h_1) \quad (\text{B.0.143})$$

We can compute the most probable state h_1^* then explicitly. For h_2 , knowing h_1^* we have

$$h_2^* = \operatorname{argmax}_{h_2} \phi(h_2, h_1^*) \gamma_{2 \leftarrow 3}(h_2) \quad (\text{B.0.144})$$

we continue to backtrack in this manner.

23.11 One can arrive at the solution by following the same procedure as for the first order HMM. One writes down a suitable potential $\phi_t = p(v_t|h_t)p(h_t|h_{t-1}, h_{t-2})$. One now performs a maximisation over h_t , which defines a message

$$\gamma_{t-1 \leftarrow t}(h_{t-1}, h_{t-2}) = \max_{h_t} \phi(h_t, h_{t-1}, h_{t-2}) \gamma_{t \leftarrow t+1}(h_t, h_{t-1}) \quad (\text{B.0.145})$$

Everything is the same as before except that a vector message is replaced with a matrix message. Once the recursion down to h_1, h_2 is completed, the optimal h_1, h_2 can be computed explicitly. Backtracking then continues.

23.12 Just use the general formula for the gradient give in section(11.6).

23.13 First we note that any distribution can be written in the ‘cascade’ form

$$p(\mathcal{H}|\mathcal{V}) = \prod_t p(h_t|h_{1:t-1}, \mathcal{V})$$

and because of the HMM structure, this simplifies to

$$p(\mathcal{H}|\mathcal{V}) = \prod_t p(h_t|h_{t-1}, \mathcal{V})$$

One can then find these terms by first computing the joint marginal $p(h_t, h_{t-1}|\mathcal{V})$ and then conditioning on h_{t-1} .

An alternative approach is to write

$$p(\mathcal{H}|\mathcal{V}) \propto \prod_t \underbrace{p(h_t|h_{t-1})p(v_t|h_t)}_{\phi(h_t, h_{t-1})}$$

This is therefore an undirected distribution that is a chain. If we start at the end of the chain we have the term

$$\phi(h_T, h_{T-1})$$

which is the only place that h_T appears. We can therefore write

$$\tilde{p}(h_T|h_{T-1}) = \frac{\phi(h_T, h_{T-1})}{\sum_{h_T} \phi(h_T, h_{T-1})}$$

Let $z(h_{T-1}) = \sum_{h_T} \phi(h_T, h_{T-1})$

$$p(\mathcal{H}|\mathcal{V}) \propto \left[\prod_{t=1}^{T-1} \phi(h_t, h_{t-1}) \right] z(h_{T-1}) \tilde{p}(h_T|h_{T-1}) = \left[\prod_{t=1}^{T-2} \phi(h_t, h_{t-1}) \right] \underbrace{\phi(h_{T-1}, h_{T-2}) z(h_{T-1})}_{\tilde{p}(h_{T-1}|h_{T-2}) z(h_{T-2})} \tilde{p}(h_T|h_{T-1})$$

In general,

$$z(h_{t-1}) = \sum_{h_t} \phi(h_t, h_{t-1}) z(h_t), \quad \tilde{p}(h_t|h_{t-1}) = \frac{\phi(h_t, h_{t-1}) z(h_t)}{z(h_{t-1})}$$

23.15 All parts are straightforward. The last part can be achieved using a β recursion.

23.16 1. A HMM would normally take $O(TL^2)$ operations. This can be reduced here since the transition is deterministic for all states h_t that are not 0. This means that no sum is required over the $h_t \neq 0$ states and the computation is reduced to linear scaling in L .

2. See `teststringsearch.m`.

24.1 1. The eigenvalues are given by $\cos \theta \pm i \sin \theta$, which are in general imaginary. Intuitively, this has to be the case since the definition of an eigenvalue-eigenvector is that the matrix operating on the eigenvector simply rescales the eigenvector. For a rotation matrix, clearly this cannot simply rescale the eigenvector – therefore there can be no real solution (unless the rotation corresponds to flipping the direction, $\theta = \pi$).

2. This is explained in the text: we can take use a two dimensional latent \mathbf{h}_t with transition matrix given by \mathbf{R}_θ . Then for the emission matrix, we can take the projection so that $v_t = [10]\mathbf{h}_t$. This takes then only the first component of \mathbf{h}_t , which will trace out a sinusoid.

3. Using $a = \cos \theta$, $b = -\sin \theta$, $c = \sin \theta$, $d = \cos \theta$,

$$x_t = ax_{t-1} + by_{t-1}, y_t = cx_{t-1} + dy_{t-1} \quad (\text{B.0.146})$$

The first equation also means

$$x_{t+1} = ax_t + by_t = ax_t + b(cx_{t-1} + dy_{t-1}) = ax_t + b \left(cx_{t-1} + \frac{d}{b} [x_t - ax_{t-1}] \right) \quad (\text{B.0.147})$$

4. This shows that one may express a sinusoidal curve exactly as a second order difference equation.

5. The solution of this continuous time harmonic oscillator is a sinusoid. This is interesting since it shows that there exists in an exact time discretisation for a second order harmonic oscillator. That is, the discrete time system exactly matches the continuous time system at the discrete time points.

24.2

$$\mathbf{A}^\top \mathbf{A} = e^{\mathbf{M}} e^{\mathbf{M}^\top} = e^{\mathbf{M} + \mathbf{M}^\top} = e^{\mathbf{M} - \mathbf{M}} = e^{\mathbf{0}} = \mathbf{I}$$

Note that $\mathbf{A}^\top = e^{\mathbf{M}^\top}$ since $e^{\mathbf{M}}$ is obtained from the Taylor series expansion of exp. This means that it is a function of powers of \mathbf{M} , and since $\mathbf{M}^{k^\top} = \mathbf{M}^{\top k}$, the result follows. We can have some control over the orthogonal matrix by adjusting \mathbf{M} . If the elements of \mathbf{M} are very small, then \mathbf{A} will be orthogonal but close to the identity. The larger \mathbf{M} is the larger the rotation enacted by \mathbf{A} will be.

24.5 TBD

25.1 See `exerciseSLDSprice.m`.

25.2 The solution is in `exerciseSLDSmeanrev.m`: 0.8298 and 0.9142

26.1 Since

$$p(v_i(t+1)|\mathbf{v}(t)) = \sigma \left(v_i(t+1) \left(b_i + \sum_j w_{ij} v_j(t) \right) \right)$$

as we saw in equation (26.3.19),

$$\frac{dL}{dw_{ij}} = \beta \sum_{t=1}^{T-1} \gamma_i(t) v_i(t+1) v_j(t)$$

which means that if we start gradient ascent from $w_{ij} = 0$, we must end up with w of the form

$$w_{ij} = \sum_{t=1}^{T-1} u_i(t) v_i(t+1) v_j(t)$$

for some suitable ‘dual’ parameters $u_i(t)$. In this case we can then learn the dual parameters rather than the weights. Note that there are VT dual parameters. Since we assume $V \ll T$, then this requires much less storage than w . One can proceed to learn the u by maximum likelihood. Then

$$p(v_i(t+1)|\mathbf{v}(t)) = \sigma \left(v_i(t+1) \left(b_i + \sum_{j,\tau} u_j(\tau) v_i(\tau+1) v_j(\tau) v_j(t) \right) \right)$$

26.2 There are N neurons. If we assume all images are independent, and no restriction on w , we can store maximally N patterns. Hence we can store

$$\frac{10^6}{10 \times 60 \times 60} \approx 27.8$$

hours of binary video.

26.3

$$\langle v_i(t+1) \rangle = \sigma(a_i(t)) - (1 - \sigma(a_i(t)))$$

Hence

$$\sigma(a_i(t)) = \frac{1}{2}(1 + \langle v_i(t+1) \rangle)$$

Also

$$\gamma_i(t) = 1 - \frac{v_i(t+1) + 1}{2} \sigma(a_i(t)) + \frac{v_i(t+1) - 1}{2} (1 - \sigma(a_i(t)))$$

Combining these gives the required result.

26.4

$$\begin{aligned} \sum_{i,j,k,l} x_{ij} x_{kl} \frac{d^2 L}{dw_{ij} dw_{kl}} &= -\beta^2 \sum_{i,j,k,l} x_{ij} x_{kl} \sum_{t=1}^{T-1} (v_i(t+1) v_j(t)) \gamma_i(t) (1 - \gamma_i(t)) v_k(t+1) v_l(t) \delta_{ik} \\ &= -\beta^2 \sum_{i,j,l} x_{ij} x_{il} \sum_{t=1}^{T-1} v_j(t) \gamma_i(t) (1 - \gamma_i(t)) v_l(t) \\ &= -\beta^2 \sum_{t,i} \gamma_i(t) (1 - \gamma_i(t)) \underbrace{\sum_j x_{ij} v_j(t)}_{z_i(t)} \underbrace{\sum_l v_l(t) x_{il}}_{z_i(t)} \\ &= -\beta^2 \sum_{t,i} \gamma_i(t) (1 - \gamma_i(t)) z_i^2(t) \leq 0 \end{aligned}$$

Which follows since $0 \leq \gamma_i(t) \leq 1$.

26.5 TBD

26.6 Based on maximum likelihood training, I estimate that one can store around 20 patterns robustly. See `exerciseHopfieldSelfTran.m`.

27.1 Writing $\mathbf{y} = (y_1, y_2)$ and $\mathbf{x} = (x_1, x_2)$, we have

$$p(\mathbf{y}) = \int \delta(\mathbf{y} - f(\mathbf{x})) d\mathbf{x} = \int \delta(\mathbf{x} - f^{-1}(\mathbf{y})) \left| \frac{d}{d\mathbf{y}} \mathbf{x} \right| d\mathbf{y}$$

From exercise(8.10), we need to find the absolute value of the determinant of the Jacobian:

$$\begin{vmatrix} \frac{\partial}{\partial x_1} y_1 & \frac{\partial}{\partial x_2} y_1 \\ \frac{\partial}{\partial x_1} y_2 & \frac{\partial}{\partial x_2} y_2 \end{vmatrix} = \begin{vmatrix} -(-2 \log x_1)^{-\frac{1}{2}} \frac{\cos 2\pi x_2}{x_1} & -2\pi(-2 \log x_1)^{\frac{1}{2}} \sin 2\pi x_2 \\ -(-2 \log x_1)^{-\frac{1}{2}} \frac{\sin 2\pi x_2}{x_1} & 2\pi(-2 \log x_1)^{\frac{1}{2}} \cos 2\pi x_2 \end{vmatrix} = -\frac{2\pi}{x_1} (\cos^2 2\pi x_2 + \sin^2 2\pi x_2) = -\frac{2\pi}{x_1}$$

From the transformation, we also have

$$x_1 = e^{-\frac{1}{2}(y_1^2 + y_2^2)}$$

Hence, since $p(x_1) = p(x_2) = 1$,

$$p(\mathbf{y}) = \frac{1}{2\pi} e^{-\frac{1}{2}(y_1^2 + y_2^2)} = \mathcal{N}(y_1|0, 1) \mathcal{N}(y_2|0, 1)$$

27.2

$$\frac{p^*(x)}{q(x)} = e^{\sin(x) + \frac{x^2}{2\sigma^2}} \sqrt{2\pi\sigma^2}$$

Since $-\pi \leq x \leq \pi$ this quantity certainly cannot be greater than

$$e^{1 + \frac{\pi^2}{2\sigma^2}} \sqrt{2\pi\sigma^2} = M$$

See `demoRejSamp.m`.

27.3

$$p'(x_1, x_2, x_3, x_4, x_6) = \frac{p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)p(x_5|x_3)p(x_6|x_4, x_5)}{\sum_{x_1, x_2, x_3, x_4, x_6} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)p(x_5|x_3)p(x_6|x_4, x_5)}$$

Since

$$\sum_{x_1, x_2, x_3, x_4, x_6} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)p(x_5|x_3)p(x_6|x_4, x_5) = \sum_{x_1, x_2, x_3} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_5|x_3)$$

Hence

$$p'(x_1, x_2, x_3, x_4, x_6) = p'(x_1, x_2, x_3)p(x_4|x_3)p(x_6|x_4, x_5)$$

where

$$p'(x_1, x_2, x_3) = \frac{p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_5|x_3)}{\sum_{x_1, x_2, x_3} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_5|x_3)}$$

To perform ancestral sampling, we can therefore sample a joint state x_1, x_2, x_3 from $p'(x_1, x_2, x_3)$ and then a state x_4 from $p(x_4|x_3)$ and a state x_6 from $p(x_6|x_4, x_5)$.

27.4 Consider a 4×4 example:

$$\begin{pmatrix} w_1 & b_1 & w_2 & b_2 \\ b_3 & w_3 & b_4 & w_4 \\ w_5 & b_5 & w_6 & b_6 \\ b_7 & w_7 & b_8 & w_8 \end{pmatrix}$$

Since all neighbours of any white variable are all black, and similarly, all neighbours of any black variable are all white, when we condition on all the black variables, the white variables become independent, and vice versa. This means that in the Gibbs step to sample from

$$p(b_1, b_2, \dots, |w_1, w_2, \dots) = p(b_1|w_1, w_2, \dots)p(b_2|w_1, w_2, \dots) \dots$$

we can draw a sample from the joint distribution by independently drawing a sample from each of the black variables conditioned on its neighbouring white variables. The converse is true when we sample from

$$p(w_1, w_2, \dots, |b_1, b_2, \dots) = p(w_1|b_1, b_2, \dots)p(w_2|b_1, b_2, \dots) \dots$$

This is advantageous since we are able to truly draw independent samples from all black variables conditioned on all white variables, and vice versa.

27.5

$$\left\langle \log \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right\rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})} = \frac{1}{2\sigma_p^2} \left(\mathbf{x}^2 - \langle (\mathbf{x}')^2 \rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})} \right)$$

Since

$$\langle (\mathbf{x}')^2 \rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})} = \mathbf{x}^2 + N\sigma_q^2$$

the result follows. The ratio

$$\frac{p(\mathbf{x}')}{p(\mathbf{x})}$$

is the Metropolis acceptance probability for a proposed \mathbf{x}' generated by adding random Gaussian vector onto the current sample \mathbf{x} . What the result shows is that the rough scaling of this acceptance probability is that it will be exponentially small in the dimension N . This suggests that the acceptance probability for this proposal distribution will be impractically small in high dimensions. To counter this effect one would need to make σ_q^2 extremely small.

More closely related to the question, the log acceptance probability can be bounded

$$\log \min \left(1, \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right) \leq \log \frac{p(\mathbf{x}')}{p(\mathbf{x})}$$

so that we have shown that the average log acceptance probability is exponentially small.

Note that we can also calculate a bound on the average acceptance probability. Since

$$\min \left(1, \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right) \leq \frac{p(\mathbf{x}')}{p(\mathbf{x})}$$

Then

$$\left\langle \min \left(1, \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right) \right\rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})} \leq \left\langle \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right\rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})}$$

Since the ratio of two Gaussians is a Gaussian, this calculation requires computing the average of a Gaussian with respect to another Gaussian. This is straightforward and gives the bound

$$\left\langle \min \left(1, \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right) \right\rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})} \leq \left(\frac{\sigma_p^2}{\sigma_p^2 + \sigma_q^2} \right)^{\frac{N}{2}} \exp \left(\frac{\frac{\sigma_q^2}{\sigma_p^2} \mathbf{x}^2}{2(\sigma_q^2 + \sigma_p^2)} \right)$$

Using $\log x \leq x - 1$, we then can then write

$$\log \left\langle \min \left(1, \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right) \right\rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})} \leq -\frac{N\sigma_q^2}{\sigma_q^2 + \sigma_p^2} \left(1 - \frac{\mathbf{x}^2}{N\sigma_p^2} \right)$$

Hence, if we are currently at a point \mathbf{x} with $\mathbf{x}^2 < N\sigma_p^2$ (i.e. a point \mathbf{x} that is reasonably close to the mode of $p(\mathbf{x})$, namely $\mathbf{x} = \mathbf{0}$), then the acceptance rate is exponentially small and we will effectively never move from this high probability point.

27.6 See `exerciseSampleHMM.m`. As λ increases, the variables in the posterior become more dependent. This means that this becomes far from the condition under which single variable Gibbs updating is valid. As such, the accuracy tends to go down as we increase the dependence in the posterior.

```
function exerciseSampleHMM
import brml.*
H=2; V=2; T=10;
% make a HMM
lambdas=[0.1 1 10 20];

for la=1:length(lambdas)
    lambda=lambdas(la);
    fprintf(1,'lambda=%f\n',lambda)
    for ex=1:20
        fprintf(1,'experiment=%d\n',ex)
        A=condp(rand(H,H).^lambda);
        B=condp(rand(V,H).^1);
        a=condp(rand(H,1));

        % draw some samples for v:
        h(1)=randgen(a); v(1)=randgen(B(:,h(1)));
        for t=2:T
            h(t)=randgen(A(:,h(t-1))); v(t)=randgen(B(:,h(t)));
        end
    end
end
```

```

[logalpha,loglik]=HMMforward(v,A,a,B); logbeta=HMMbackward(v,A,B);
gamma=HMMsmooth(logalpha,logbeta,B,A,v); % exact marginal

% single site Gibbs updating
hsamp(:,1)=randgen(1:H,1,T);
hv=1:T; vv=T+1:2*T; % hidden and visible variable indices

num_samples=100;
for sample=2:num_samples
    h = hsamp(:,sample-1);
    emiss=array([vv(1) hv(1)],B);
    trantm=array(hv(1),a);
    trant=array([hv(2) hv(1)],A);
    h(1) = randgen(table(setpot(multipots([trantm trant emiss]),[vv(1) hv(2)],[v(1) h(2)])));

    for t=2:T-1
        trantm=array([hv(t) hv(t-1)],A);
        trant=array([hv(t+1) hv(t)],A);
        emiss=array([vv(t) hv(t)],B);
        h(t) = randgen(table(setpot(multipots([trantm trant emiss]),[vv(t) hv(t-1) hv(t+1)],[v(t) h(t-1) h(t+1)])));
    end

    trantm=array([hv(T) hv(T-1)],A);
    emiss=array([vv(T) hv(T)],B);
    h(T) = randgen(table(setpot(multipots([trantm emiss]),[vv(T) hv(T-1)],[v(T) h(T-1)])));

    hsamp(:,sample)=h; % take the sample after a forward sweep through time
end
for t=1:T
    gamma_samp(:,t) = count(hsamp(t,:),H)/num_samples;
end
er(ex,la) = mean(abs(gamma(:)-gamma_samp(:)));
end
end
fprintf(1,'lambdas and mean absolute error in marginals:\n')
disp([lambdas' mean(er)'])

```

27.8 For the last part we need

$$\frac{1}{\sigma^4} \left\langle \left[(\mathbf{x}^i - \mathbf{x})^2 - (\mathbf{x}^j - \mathbf{x})^2 \right]^2 \right\rangle \quad (\text{B.0.148})$$

The average is straightforward based on moments of Gaussians. This can be made a little easier by defining

$$\mathbf{y}_i = \mathbf{x} - \mathbf{x}^i$$

which is Gaussian with zero mean and variance $2\nu^2\mathbf{I}$, and $\langle y_{ik}y_{jl} \rangle = \nu^2$ for $i \neq j$. Then the required computation is

$$2 \left(\langle (\mathbf{y}_i^2)^2 \rangle - \langle (\mathbf{y}_i)^2 (\mathbf{y}_j)^2 \rangle \right) = 2D (\langle y^4 \rangle - \langle y_i^2 y_j^2 \rangle) = 2D (3(2\nu^2)^2 - 6\nu^4)$$

27.9 See `solutionSoccer.m`. Let's label the game data \mathcal{D} . We are then interested in the posterior

$$p(\mathbf{a}, \mathbf{b} | \mathcal{D}) \propto p(\mathcal{D} | \mathbf{a}, \mathbf{b}) p(\mathbf{a}, \mathbf{b})$$

We will assume that the prior on skill levels for the two teams and all players are independent and that each player has a priori equal skill level, *i.e.* $p(\mathbf{a}, \mathbf{b}) = \text{const.}$ Then

$$p(\mathbf{a}, \mathbf{b} | \mathcal{D}) \propto \prod_{g=1}^{20} \sigma \left(\sum_{i=1}^{10} (a_{t_i^a(g)} - b_{t_i^b(g)}) \right)$$

where $t_i^a(n)$ is the team of Aces in game g , and similarly for Bruisers. We can draw samples from this posterior. I chose to use Gibbs sampling.

Given this posterior, we can calculate the expected result for a game in which we know the team players of Bruisers is players 1 to 10. This is

$$\left\langle \sigma \left(\sum_{i=1}^{10} (a_{t_i^a} - b_i) \right) \right\rangle_{p(\mathbf{a}, \mathbf{b} | \mathcal{D})}$$

To find the best Aces team, we then need to search through all the possible team arrangements, approximating the above expectation using the samples from the posterior. Note that this is not necessarily the same as simply choosing the best Aces team based on their individual abilities $\langle a_i \rangle_{p(a_i | \mathcal{D})}$ since, in the posterior $p(\mathbf{a}, \mathbf{b} | \mathcal{D})$, \mathbf{a} and \mathbf{b} will be dependent. (You might want to field what seems to be a good Aces player, but historically he might have performed poorly when say player 2 is in the Bruiser's team. Similarly, in principle, we could have a posterior with a high mean skill, but also very high variance, in which case it might be preferable to take a player with a lower expected skill if they also have a lower variance.) One thing to note is that, based on Gibbs sampling, the posterior ability of each player has quite high variance, meaning that we cannot be particularly confident of their abilities, given the limited training game data. Nevertheless, based on 100,000 Gibbs samples (and then discarding the first 1000 and subsampling, taking only every 20th sample), the best Aces team given that Bruisers field players 1 to 10 is

```
bestAteamGivenB1to10 =
    1     2     7    10    11    12    13    14    16    20
```

There is no easy answer to the question ‘which is the best player’. One can to answer this is to rank players by their posterior mean abilities. This gives

```
bestA =
    1     2     7    10    11    12    13    14    16    20
bestB =
    2     5     6     7    10    12    13    14    16    20
```

In this case, it happens that the best Aces team is the same as simply selecting the 10 best players based on their mean performance (I need to make a better dataset). I don’t think this would generally be true. Another approach would be to find the best team A assuming that we do not know the team B selection. This would require summing over all team B selections for each team A choice. This is computationally very expensive.

```
clear all; close all
load Soccer
import brml.randgen
% sample some game data
S=20 % number of players in each squad
T=10 % number of players in a team
G=20; % number of games

for g=1:G
    ta(:,g)=game(g).teamAces;
    tb(:,g)=game(g).teamBruisers;
    outab(g)=game(g).AcesWin;
end

% now do some Gibbs sampling to get the posterior skill distribution,
% assuming uniform priors on skill levels, p(a,b|data)\propto p(out|a,b):
levels=-2:2; nsamples=100000;
aa=zeros(1,S); bb=zeros(1,S);
asample=zeros(S,nsamples);
bsample=zeros(S,nsamples);
for sample=2:nsamples
    asample(:,sample)=asample(:,sample-1);
    bsample(:,sample)=bsample(:,sample-1);
    for player=randperm(S)
        [~,games]=find(ta==player);
        aacand=asample(:,sample);
        tmp=ones(5,1);
        ind=0;
        for skill=levels
            ind=ind+1;
            aacand(player)=skill;
            for g=games(:)'
                tmp(ind)=tmp(ind)*sigmoid(outab(g)*(sum(aacand(ta(:,g)))-sum(bsample(tb(:,g),sample))));
            end
        end
        asample(player,sample)=levels(randgen(tmp));
    end
    for player=randperm(S)
        [~,games]=find(tb==player);
        bbcand=bsample(:,sample);
        tmp=ones(5,1);
        ind=0;
        for skill=levels
            ind=ind+1;
            bbcand(player)=skill;
            for g=games(:)'
                tmp(ind)=tmp(ind)*sigmoid(outab(g)*(sum(asample(ta(:,g),sample))-sum(bbcand(tb(:,g)))));
            end
        end
        bsample(player,sample)=levels(randgen(tmp));
    end
end

aasample=asample(:,1000:20:end); bbsample=bsample(:,1000:20:end); % subsample

hold on; sd=diag(sqrt(cov(aasample')));
errorbar(1:20,mean(aasample,2),sd,'o')
sd=diag(sqrt(cov(bbsample')));
errorbar(1.25:20.25,mean(bbsample,2),sd,'ro'); legend('Aces','Bruisers')

teamb=1:10;
teams=nchoosek(1:S,T);
for t=1:size(teams,1)
```



```

    teama=teams(t,:);
    pAbeatsB(t)= mean(sigmoid(sum(aasample(teama,:)-sum(bbsample(teamb,:))));
end

[val ind]=max(pAbeatsB);
bestAteamGivenB1to10=teams(ind,:) % best A team

% There is no easy answer to the question 'which is the best player'. One
% can to answer this is to rank players by their posterior mean abilities:
[~, indA]=sort(mean(aasample'),'descend');
[~, indB]=sort(mean(bbsample'),'descend');
bestA=sort(indA(1:10))
bestB=sort(indB(1:10))
% another approach would be to find the best team A assuming that we do
% not know the team B selection. This would require summing over all team
% B selections for each team A choice. This is computationally very
% expensive.

27.10 clear all; close all
load SymptomDisease.mat

import brml.*

[S,D]=size(W);

% use gibbs sampling to estimate p(s|d)

% Gibb's sampling:
Nsamps=10000;
d=rand(D,1)<p;
samps=zeros(D,Nsamps);
for samp=1:Nsamps
    r=randperm(D); % random ordering for the Gibbs sweep through the variables
    for i=r
        d(i)=0;
        tmp=sigmoid(W*d+b,1);
        E0 = sum(d.*log(p)+(1-d).*log(1-p)) + sum(s.*log(tmp))+sum((1-s).*log(1-tmp));

        d(i)=1;
        tmp=sigmoid(W*d+b,1);
        E1 = sum(d.*log(p)+(1-d).*log(1-p)) + sum(s.*log(tmp))+sum((1-s).*log(1-tmp));

        if rand<sigmoid(E1-E0,1) % sample from the conditional p(d_i=1|d_-\i,s)
            d(i)=1;
        else
            d(i)=0;
        end
    end
    samps(:,samp)=d;
end
burnin=1000;
%imagesc(samps)
mean(samps(:,burnin:end),2)' % sample everage

% running the above code gives the output:
%
% ans =
%
% Columns 1 through 10
%
%    0.0026    0.9982    0.0210    1.0000    0.6495    0.0111    0.0157    0.0006    0.0070    0.9999
%
% Columns 11 through 20
%
%    0.0049    1.0000    1.0000    1.0000    0.9883    0.9698    0.9857    0.9045    0.9999    0.9936
%
% Columns 21 through 30
%
%    0.0843    0.7478    0.9998    0.9948    0.0039    0.9996    0.0158         0    0.9989         0
%
% Columns 31 through 40
%
%    0.0001    0.0885    0.0106    1.0000         0    0.0023    0.9941    0.0013         0    0.0003
%
% Columns 41 through 50
%
%    0.9923    0.9977    1.0000    0.9998    0.0001    0.0157    0.9967    0.9934         0    0.9998

```

Nearly all the marginals are almost deterministic, except for disease 5, which is relatively uncertain.

27.11 This follows directly from Bayes rule and the iid assumption. We would need to choose a sensible prior for the parameters, for example independent priors for each parameter block \mathbf{W} , \mathbf{b} , \mathbf{p} . One could then use Gibbs sampling to draw parameter samples from $p(\mathbf{W}, \mathbf{b}, \mathbf{p} | \mathcal{D})$ and use these samples to draw samples:

$$p(\mathbf{d} | \mathbf{s}, \mathcal{D}) \approx \frac{1}{L} \sum_{l=1}^L p(\mathbf{d} | \mathbf{s}, \mathbf{W}^l, \mathbf{b}^l, \mathbf{p}^l)$$

where here l is the sample index. For each parameter sample l , we then need to run another sampling scheme to draw samples \mathbf{d} .

28.1 0.0160. See `exerciseKLfit.m`.

28.2 mean error BP = 0.0402797 mean error MF = 0.0501128 See `exerciseMFBP.M`. Both work quite well for this small problem, with BP working slightly better.

28.3 TBD

28.5 1.

$$\begin{aligned} \int e^x &\geq \int 0 \\ e^x - e^a &\geq 0 \quad (x \geq a) \\ \int (e^x - e^a) &\geq \int 0 \\ e^x - e^a - (x - a)e^a &\geq 0 \end{aligned}$$

Rearranging, we obtain

$$e^x \geq e^a(1 + x - a)$$

2.

$$\begin{aligned} e^{\mathbf{s}^\top \mathbf{W} \mathbf{s}} &\geq e^{\mathbf{h}^\top \mathbf{s}}(1 + \mathbf{s}^\top \mathbf{W} \mathbf{s} - \mathbf{h}^\top \mathbf{s}) \\ Z = \sum_{\mathbf{s}} e^{\mathbf{s}^\top \mathbf{W} \mathbf{s}} &\geq \sum_{\mathbf{s}} e^{\mathbf{h}^\top \mathbf{s} + \theta} (1 + \mathbf{s}^\top \mathbf{W} \mathbf{s} - \mathbf{h}^\top \mathbf{s} - \theta) \end{aligned}$$

3. Differentiating *w.r.t.* θ , and equating to zero we have

$$\sum_{\mathbf{s}} e^{\mathbf{h}^\top \mathbf{s} + \theta} (-1 + 1 + \mathbf{s}^\top \mathbf{W} \mathbf{s} - \mathbf{h}^\top \mathbf{s} - \theta) = 0$$

Hence

$$\sum_{\mathbf{s}} e^{\mathbf{h}^\top \mathbf{s} + \theta} (\mathbf{s}^\top \mathbf{W} \mathbf{s} - \mathbf{h}^\top \mathbf{s} - \theta) = 0$$

and

$$\theta = \frac{1}{\sum_{\mathbf{s}} e^{\mathbf{h}^\top \mathbf{s}}} \sum_{\mathbf{s}} e^{\mathbf{h}^\top \mathbf{s}} (\mathbf{s}^\top \mathbf{W} \mathbf{s} - \mathbf{h}^\top \mathbf{s})$$

Plugging these into the bound and taking the logarithm, we have

$$\log Z \geq \log Z_q + \langle \mathbf{s}^\top \mathbf{W} \mathbf{s} - \mathbf{h}^\top \mathbf{s} \rangle = H_q + \langle \mathbf{s}^\top \mathbf{W} \mathbf{s} \rangle$$

where we define a factorised MF distribution

$$q(\mathbf{s}) = \frac{e^{\mathbf{h}^\top \mathbf{s}}}{\sum_{\mathbf{s}} e^{\mathbf{h}^\top \mathbf{s}}}$$

and $H_q = -\langle \log q(\mathbf{s}) \rangle_q$. This is then in the form of a standard naive mean field bound.

4. If we perform another double integration, on the bound

$$e^x \geq e^a(1 + x - a)$$

we obtain a cubic lower bound on e^x :

$$e^x - e^b \geq e^a(1 - a)(x - b) + \frac{1}{2}e^a(x^2 - b^2) \quad x \geq b$$

$$e^x - e^b - (x - b)e^b \geq -b(x - b)e^a(1 - a) + \frac{1}{2}e^a(1 - a)(x^2 - b^2) - \frac{1}{2}b^2e^a(x - b) + \frac{1}{3}e^a(x^3 - b^3)$$

If we use $a = b$, then we can rearrange to get

$$e^x \geq e^a f(x)$$

where $f(x)$ is a cubic polynomial in x . By using the same procedure as before, replacing $a \rightarrow \mathbf{h}^\top \mathbf{s} + \theta$, we can optimise the bound wrt to a factorised approximation.

28.6

$$Z = \sum_{\mathbf{x}} e^{\mathbf{b}^\top \mathbf{x}} e^{\mathbf{x}^\top \langle W \rangle \mathbf{x}} \leq \sum_i p_i \sum_{\mathbf{x}} e^{\mathbf{x}^\top W_i \mathbf{x} + \mathbf{b}^\top \mathbf{x}}$$

Each term $\sum_{\mathbf{x}} e^{\mathbf{x}^\top W_i \mathbf{x} + \mathbf{b}^\top \mathbf{x}}$ is the normalisation of a tree distribution, which is therefore tractable. Naively, one can therefore optimise the bound then with respect to the parameters p_i .

28.7 A special case of the IM framework is to use a linear Gaussian decoder

$$\begin{aligned} \langle \log q(\mathbf{x}|\mathbf{y}) \rangle_{p(\mathbf{x},\mathbf{y})} &= -\frac{1}{2} \left\langle (\mathbf{x} - \mathbf{U}\mathbf{y})^\top \Sigma^{-1} (\mathbf{x} - \mathbf{U}\mathbf{y}) \right\rangle_{p(\mathbf{x},\mathbf{y})} - \frac{1}{2} \log \det (2\pi \Sigma) \\ &= -\frac{1}{2} \text{trace} \left(\Sigma^{-1} \left\langle (\mathbf{x} - \mathbf{U}\mathbf{y}) (\mathbf{x} - \mathbf{U}\mathbf{y})^\top \right\rangle_{p(\mathbf{x},\mathbf{y})} \right) - \frac{1}{2} \log \det (2\pi \Sigma) \end{aligned}$$

Writing, more compactly $\langle \cdot \rangle$ for $\langle \cdot \rangle_{p(\mathbf{x},\mathbf{y})}$, and optimising *w.r.t.* Σ and \mathbf{U} we obtain:

$$\Sigma = \left\langle (\mathbf{x} - \mathbf{U}\mathbf{y}) (\mathbf{x} - \mathbf{U}\mathbf{y})^\top \right\rangle, \quad \mathbf{U} = \left\langle \mathbf{x}\mathbf{y}^\top \right\rangle \left\langle \mathbf{y}\mathbf{y}^\top \right\rangle^{-1}$$

Plugging these into the bound we obtain

$$I(X, Y) \geq H(X) - \frac{1}{2} \text{trace}(\mathbf{I}) - \frac{1}{2} \log \det (2\pi \Sigma)$$

We can write this directly in terms of the statistics of $p(\mathbf{x}, \mathbf{y})$ since

$$\Sigma = \left\langle (\mathbf{x} - \mathbf{U}\mathbf{y}) (\mathbf{x} - \mathbf{U}\mathbf{y})^\top \right\rangle = \left\langle \mathbf{x}\mathbf{x}^\top - \mathbf{x}\mathbf{y}^\top \mathbf{U}^\top - \mathbf{y}^\top \mathbf{U}^\top \mathbf{x} + \mathbf{U}\mathbf{y}\mathbf{y}^\top \mathbf{U}^\top \right\rangle = \left\langle \mathbf{x}\mathbf{x}^\top \right\rangle - \left\langle \mathbf{x}\mathbf{y}^\top \right\rangle \left\langle \mathbf{y}\mathbf{y}^\top \right\rangle^{-1} \left\langle \mathbf{y}\mathbf{x}^\top \right\rangle$$

28.8 1.

$$r(x) = \frac{p(x)f(x)}{Z}$$

where, by construction, therefore, $J = \log Z$. Taking $\text{KL}(r|q)$ gives the bound

$$\log Z = J \geq -\langle \log q \rangle_q + \langle \log p(x) \rangle_q + \langle \log f(x) \rangle_q = -\text{KL}(q|p) + \langle \log f(x) \rangle_q$$

2. The result follows simply from adding and subtracting the entropy $H(q)$. The result is particularly interpretable for a function $f(x)$ that is positive and integrates to 1, although the result holds more generally, even when f is not a distribution (on using the standard expression for the KL divergence). Essentially the three requirements translate into q being equal, optimally, to pf .

28.9

$$p(x_i) = \sum_{x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_D} p(\mathbf{x}) = \frac{1}{Z} \sum_{x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_D} e^{\mathbf{x}^\top \mathbf{W} \mathbf{x}}$$

If we wish to lower bound $p(x_i)$ we need a lower bound on $Z_{\setminus i}$ and a lower bound on $1/Z$ since the product of two lower bounds is a lower bound. In other words we need an upper bound on Z . To upper bound $p(x_i)$, we need the converse.

28.10 TBD

28.11 For any $y \in \mathcal{T}$ let's look at the flow $\gamma(y)$ through this node. This is the amount coming in minus the amount going out.

$$\begin{aligned} \gamma(y) &= \sum_x f(x, y) - \sum_x x f(y, x) \\ &= \sum_{x \in \mathcal{S}} f(x, y) + \sum_{x \in \mathcal{T}} f(x, y) - \sum_{x \in \mathcal{S}} f(y, x) - \sum_{x \in \mathcal{T}} f(y, x) \end{aligned}$$

Now sum this over $y \in \mathcal{T}$:

$$\sum_{y \in \mathcal{T}} \gamma(y) = \sum_{x \in \mathcal{S}, y \in \mathcal{T}} f(x, y) + \sum_{x \in \mathcal{T}, y \in \mathcal{T}} f(x, y) - \sum_{x \in \mathcal{S}, y \in \mathcal{T}} f(y, x) - \sum_{x \in \mathcal{T}, y \in \mathcal{T}} f(y, x) \quad (\text{B.0.149})$$

Since from flow conservation $\gamma(y) = 0$ except for $y = t$, when $\gamma(t) = \text{val}(f)$, namely the flow across the network. Hence

$$\text{val}(f) = \sum_{x \in \mathcal{S}, y \in \mathcal{T}} f(x, y) - \sum_{x \in \mathcal{S}, y \in \mathcal{T}} f(y, x) \quad (\text{B.0.150})$$

The second part follows since the term $\sum_{x \in \mathcal{S}, y \in \mathcal{T}} f(y, x) \geq 0$, showing that the maximum flow cannot be greater than the minimum cut. The max-flow min s - t -cut theorem goes further and states that the maximum flow is actually equal to the minimum cut. See for example [40] for a concise proof.

28.12

$$\mathbb{I}[x_i = x_j] = \mathbb{I}[s_i \alpha + (1 - s_i)x_i^{old} = s_j \alpha + (1 - s_j)x_j^{old}] \quad (\text{B.0.151})$$

Since $s_i \in \{0, 1\}$, this can be written as

$$\begin{aligned} \mathbb{I}[x_i = x_j] &= (1 - s_i)(1 - s_j)\mathbb{I}[x_i^{old} = x_j^{old}] + (1 - s_i)s_j\mathbb{I}[x_i^{old} = \alpha] + s_i(1 - s_j)\mathbb{I}[x_j^{old} = \alpha] + s_i s_j \\ &= s_i s_j u_{ij} + a_i s_i + b_j s_j + \text{const.} \end{aligned}$$

with

$$u_{ij} \equiv 1 - \mathbb{I}[x_i^{old} = \alpha] - \mathbb{I}[x_j^{old} = \alpha] + \mathbb{I}[x_i^{old} = x_j^{old}] \quad (\text{B.0.152})$$

and similarly defined a_i, b_i . By enumeration it is straightforward to show that u_{ij} is either 0, 1 or 2. Using the mathematical identity

$$s_i s_j = \frac{1}{2} (\mathbb{I}[s_i = s_j] + s_i + s_j - 1) \quad (\text{B.0.153})$$

we can write, for $x_i = s_i \alpha + (1 - s_i)x_i^{old}$,

$$\mathbb{I}[x_i = x_j] = \frac{u_{ij}}{2} (\mathbb{I}[s_i = s_j] + s_i + s_j) + a_i s_i + b_j s_j + \text{const.} \quad (\text{B.0.154})$$

Hence terms $w_{ij}\mathbb{I}[x_i = x_j]$ translate to positive interaction terms $w_{ij}u_{ij}/2\mathbb{I}[s_i = s_j]$. All the unary terms are easily exactly mapped into corresponding unary terms $c'_i s_i$ for c'_i defined as the sum of all unary terms in s_i . This shows that the positive interaction w_{ij} in terms of the original variables x maps to a positive interaction in the new variables s . Hence we can find the maximal state of s using a graph cut algorithm.

28.13 1.

$$\text{KL}(p|q) = \langle \log p \rangle_p - \langle \log \rangle_p = -\phi^\top \langle x \rangle_p + \log Z(\phi) \text{const.}$$

Differentiating *w.r.t.* ϕ_i and equating to zero, the optimal ϕ satisfies

$$\langle g_i(x) \rangle_{p(x)} - \frac{1}{Z(\phi)} \int g_i(x) e^{\phi^\top \mathbf{g}(x)} = 0$$

which is the desired result.

2. This is clearly true since a Gaussian is quadratic in the exponential.

$$-\frac{1}{2\sigma^2} (x - \mu)^2 = -\frac{1}{2\sigma^2} x^2 + \frac{\mu}{\sigma^2} x + \text{const.}$$

3. Optimally

$$\langle g_1(x) \rangle_p = \langle g_1(x) \rangle_q$$

$$\langle g_2(x) \rangle_p = \langle g_2(x) \rangle_q$$

and the result follows.

28.14 These results just follow directly. Consider

$$\frac{d\mathcal{B}}{db_i} = \frac{\partial \mathcal{B}}{\partial b_i} + \sum_j \frac{d\mathcal{B}}{d\alpha_j} \frac{\partial \alpha_j}{\partial b_i}$$

At the optimum of the bound,

$$\frac{d\mathcal{B}}{d\alpha_j} = 0$$

Hence at the optimum the mean is given by

$$\frac{d\mathcal{B}}{db_i} = \frac{\partial \mathcal{B}}{\partial b_i} = \tanh(\alpha_i) = \langle x_i \rangle_q$$

Similarly, for $i \neq j$, the covariance term is

$$\frac{d^2}{db_i db_j} \log Z(w, b) = \frac{d \tanh(\alpha_i)}{db_j} = 0$$

which is consistent with using a factorised (independent) approximation q .

28.15 Since $x_i \in \{0, 1\}$

$$\langle x_i x_j \rangle = 0 \times p(x_i = 0, x_j = 0) + 0 \times p(x_i = 0, x_j = 1) + 0 \times p(x_i = 1, x_j = 0) + 1 \times p(x_i = 1, x_j = 1) = p(x_i = 1, x_j = 1) = p(x_i = 1 | x_j = 1) p(x_j = 1),$$

where the last step follows from Bayes' rule. We can use this 'algebraic perturbation' to obtain an improved approximation to $\langle x_i x_j \rangle_p$ by running

$$\langle x_i x_j \rangle_p \approx \langle x_i | x_j = 1 \rangle_q \langle x_j \rangle_q$$

For each j we now clamp x_j into state 1 and then run a factorised approximation on the remaining variables, giving $\langle x_i | x_j = 1 \rangle$. One repeats for all j . This method gives an improved approximation to $\langle x_i x_j \rangle$, but is more expensive since a separate mean field optimisation is required for each i .

28.16 Defining

$$E(\tilde{\phi}_{3 \rightarrow 2}(x_2)) = \log \tilde{Z} - \left\langle \log \tilde{\phi}_{3 \rightarrow 2}(x_2) \tilde{\phi}_{2 \rightarrow 3}(x_3) \right\rangle_{\tilde{p}_*(x_2, x_3)} = \log \tilde{Z} - \left\langle \log \tilde{\phi}_{3 \rightarrow 2}(x_2) \right\rangle_{\tilde{p}_*(x_2)} + \text{const.}$$

and using

$$\tilde{Z} \propto \sum_{x_2} \tilde{\phi}_{1 \rightarrow 2}(x_2) \tilde{\phi}_{3 \rightarrow 2}(x_2) \sum_{x_3} \tilde{\phi}_{2 \rightarrow 3}(x_3) \tilde{\phi}_{4 \rightarrow 3}(x_3)$$

Then differentiating and equating to zero, we have at the optimum

$$\frac{\partial E}{\partial \tilde{\phi}_{3 \rightarrow 2}(x_2)} = \frac{1}{\tilde{Z}} \tilde{\phi}_{1 \rightarrow 2}(x_2) - \frac{\tilde{p}_*(x_2)}{\tilde{\phi}_{3 \rightarrow 2}(x_2)} = 0$$

rearranging

$$\tilde{p}_*(x_2) = \frac{1}{\tilde{Z}} \tilde{\phi}_{1 \rightarrow 2}(x_2) \tilde{\phi}_{3 \rightarrow 2}(x_2)$$

Since we are free to normalise the messages as we wish, we may define a marginal distribution

$$\tilde{p}_*(x_2) = \frac{\tilde{\phi}_{1 \rightarrow 2}(x_2) \tilde{\phi}_{3 \rightarrow 2}(x_2)}{\sum_{x_2} \tilde{\phi}_{1 \rightarrow 2}(x_2) \tilde{\phi}_{3 \rightarrow 2}(x_2)}$$

A similar derivation holds for $\tilde{p}_*(x_3)$.