Graphical Methods Assignment 3

Team Members:
- Marika Paraskevopoulou
- Eoghan Flanagan
- Jonathan Smith
- Klaudia Ludwisiak

Work Breakdown:
We approached the questions in groups of two or three, and then came together as a team to brainstorm ideas and compare answers where needed.
The following table shows in green which people took primary responsibility for which questions. It should be emphasised that for the harder questions the entire team contributed ideas.

	Eoghan	Klaudia	Maria	Jonathan
7.4				
7.17				
9.1				
9.9				
9.10				
9.13				
9.14				
10.3				
10.5				
23.4				
23.8				
23.15				
23.16				
23.17				

Exercise 7.4

Part 1:

I approached this as a Markov decision process and applied the Bellman equations recursively to find the optimal path (eqs. 7.5.17 and 7.5.18 from the BRML textbook).

I defined one transition probability matrix for each of the five possible decisions {stay, up, down, left, right}.

I decided to use a finite-horizon approach, as using a temporally-unbounded approach with a discount factor does not reflect the question specification. Even though the question does not have a natural time bound, I justified this approach with the following reasons:

- By inspection, the final 'parking bay' state of the airplane has the highest utility of any location. So:
    - Assuming the 'stay' command is reliable, once in this location the optimal strategy is to remain there.
    - Because the problem is time-invariant, there is no benefit to loitering in an intermediate state on the way to the parking bay.
- To account for the fact that there could be a solution where you sacrifice some initial utility in order to get to the 'parking bay' state as quickly as possible, it is important to have a time horizon sufficiently large that there is time to gain back the utility in the long-term.
- To start with I used a time horizon of 15 x 18 timesteps, to ensure that all possible paths are considered (see 'no loitering' assumption above). Being the area of the airspace, this is the longest path before you have to repeat positions or stay in the same location.
- In the end the 'parking bay' state was reached in under 50 steps due to this particular problem setup, so just those steps are shown.

In the end the particular solution that my code found was to cut across two neutral positions (5,5) and (5,6) on the way to the parking bay. This is because there are two equally good decisions to make from position (5,7):

- Going 'left' delays your arrival at the parking bay by two timesteps, but allows you to gain additional utility of two.
- Going 'down' goes through two squares with utility zero, but gets to the parking bay earlier.
- The choice my code made was arbitrary (due to the Matlab 'max' function implementation), but clearly you could modify the code to prioritise the decision with (e.g.) the higher short-term utility. Here is is not needed so this is not done.

Matlab code and output of optimal path follows below.

Part 2:

In part 2 I simply reused the code from part 1, with a modified transition matrix for the ‘right’ decision.

A different optimal route was found, reflecting the increased risk of trying to move ‘right’ but instead moving ‘up’ into a costly position. The new route is longer, but more conservative.

Again, the particular solution cut across the same two two neutral squares, for the same reasons.

Matlab code and output of optimal path follows below.

Exercise 7.4 part 1

Table of Contents

Environment setup

```
close all
clear all
import brml.*
load airplane.mat   % Gives matrix 'U', which is the state utilities
```

Initialisation

```
% Size of airspace
Gx = 18;
Gy = 15;
```

Set up a transition probability matrix for each action.

I.e. conditional on each decision {stay, up, down, left, right} Keep as generic as possible, so define an entire x,y matrix or probabilitites for each x,y position.

```
% Manage as a cell array or probability matrices for each
% decision.  So trans_prob_stay{x1,y1}(x2,y2) is the probability
% of moving from (x1,y1) to (x2,y2) under decision 'stay'.

% Declare the transition probability matrix for each action.
trans_prob_stay = cell(Gx, Gy);
trans_prob_up = cell(Gx, Gy);
trans_prob_down = cell(Gx, Gy);
trans_prob_left = cell(Gx, Gy);
trans_prob_right = cell(Gx, Gy);

% Initialise with zero probabilities
 for x = 1:Gx
    for y = 1:Gy

        trans_prob_stay{x,y} = zeros(Gx,Gy);
```

```
        trans_prob_up{x,y} = zeros(Gx,Gy);
        trans_prob_down{x,y} = zeros(Gx,Gy);
        trans_prob_left{x,y} = zeros(Gx,Gy);
        trans_prob_right{x,y} = zeros(Gx,Gy);

    end
end
```

Populate transition probability matrices

Implicity here we account for the fact that you can't move outside the airspace. For example, if you are on the edge and try to move out of the airspace, we just set a transition probability to your current location as 1. This is achieved using the max/min expressions below.

```
for x = 1:Gx
   for y = 1:Gy

       % "For stay, the airplane stays in the same x, y position."
       trans_prob_stay{x,y}(x,y) = 1;

       % "For up, the airplane moves to the x, y + 1 position."
       trans_prob_up{x,y}(x,min(y+1,Gy)) = 1;

       % "For down, the airplane moves to the x, y ? 1 position."
       trans_prob_down{x,y}(x,max(y-1,1)) = 1;

       % "For left, the airplane moves to the x ? 1, y position."
       trans_prob_left{x,y}(max(x-1,1),y) = 1;

       % "For right, the airplane moves to the x + 1, y position."
       trans_prob_right{x,y}(min(x+1,Gx),y) = 1;

   end
end
```

Working back through time, apply the Bellman equation recursively

```
T = 50; % up to 18*15; % number of timesteps to evaluate

V = cell(1,T); % Values matrix, with (reverse) cumulative utilities

V{1,T} = U; % Value of the final timestep is the utility value.

for t = T:-1:2
    % This loop implements the Bellman equation here
    % Notation:

    % x_t is x(t)
    % x_t_minus_1 is x(t-1)
    % 'dec' means 'decision'
```

```
    % For each location
    for x_t_minus_1 = 1:Gx
        for y_t_minus_1 = 1:Gy

            % For each decision
            % Work out the expected value resulting from that decision
            % (and subsequent optimal decisions)

            % Initialise totals
            dec_stay_total = 0;
            dec_up_total = 0;
            dec_down_total = 0;
            dec_left_total = 0;
            dec_right_total = 0;

            % In this question it is not strictly necessary to loop
            % over all squares, as we know there are many probabilites
            % of 0 in the transition matrix, but this keeps the
            % solution general.

            % Notation is a little cumbersome here but it matches
            % the Bellman equations in the BRML book.

            for x_t = 1:Gx
                for y_t = 1:Gy

                    dec_stay_total = dec_stay_total ...
                        + trans_prob_stay{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                    dec_up_total = dec_up_total ...
                        + trans_prob_up{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                    dec_down_total = dec_down_total ...
                        + trans_prob_down{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                    dec_left_total = dec_left_total...
                        + trans_prob_left{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                    dec_right_total = dec_right_total...
                        + trans_prob_right{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                end
            end
```

```
                % Calculate the max of the totals
                max_total = max([dec_stay_total, dec_up_total, ...
                    dec_down_total, dec_left_total, dec_right_total]);

                % Then add the utility of the current state.
                V{1,t-1}(x_t_minus_1,y_t_minus_1) = ...
                    max_total + U(x_t_minus_1,y_t_minus_1);

            end
        end
    end
```

Now work forwards from timestep 1 to calculate optimum path

Basically trying to climb the 'steepest' path

```
% Put optimal decision path in a struct to keep track of x, y and
 decision
% Structure is indexed by timestep, and has fields 'x', 'y' and
% 'optimal_decison'.

path = struct;

% initialise start point with timestep 1 information, given
% in question
path(1).x = 1;
path(1).y = 13;

% for each timestep, working forwards
for t = 1:T-1

    % for each decision, keep track of totals
    total_stay = 0;
    total_up = 0;
    total_down = 0;
    total_left = 0;
    total_right = 0;

    x_t = path(t).x;
    y_t = path(t).y;

    % For each location
    for x_t_plus_1 = 1:Gx
        for y_t_plus_1 = 1:Gy

            % for the current location, for each square that you can
            % transition to, calculate the probability  of
            % transitioning to that point, times the value of being in
            % that point at the next step.
```

```
            % Notation is a little cumbersome here but it matches
            % the Bellman equations in the BRML book.

            total_stay = total_stay...
                + trans_prob_stay{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
                * V{1,t+1}(x_t_plus_1,y_t_plus_1);

            total_up = total_up...
                + trans_prob_up{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
                * V{1,t+1}(x_t_plus_1,y_t_plus_1);

            total_down = total_down...
                + trans_prob_down{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
                * V{1,t+1}(x_t_plus_1,y_t_plus_1);

            total_left = total_left...
                + trans_prob_left{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
                * V{1,t+1}(x_t_plus_1,y_t_plus_1);

            total_right = total_right...
                + trans_prob_right{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
                * V{1,t+1}(x_t_plus_1,y_t_plus_1);

        end
    end

    % Now find the highest total value (this is the 'argmax',
    % to find the optimal decision).

    % put totals in an array to be able to take max
    stay = 1; up = 2; down = 3; left = 4; right = 5;

    totals(stay) = total_stay;
    totals(up) = total_up;
    totals(down) = total_down;
    totals(left) = total_left;
    totals(right) = total_right;

    [value, max_index] = max(totals);

    if max_index == stay
       % i.e. best decision is to stay
       path(t).optimal_decision = 'stay';
       path(t+1).x = path(t).x;
       path(t+1).y = path(t).y;
    end

    if max_index == up
       % i.e. best decision is to move up
       % saturate to stop going off the edge of the airspace
       path(t).optimal_decision = 'up';
       path(t+1).x = path(t).x;
       path(t+1).y = min((path(t).y + 1),Gy);
    end
```

```
    if max_index == down
       % i.e. best decision is to move down
       % saturate to stop going off the edge of the airspace
       path(t).optimal_decision = 'down';
       path(t+1).x = path(t).x;
       path(t+1).y = max((path(t).y - 1),0);
    end

    if max_index == left
       % i.e. best decision is to move down
       % saturate to stop going off the edge of the airspace
       path(t).optimal_decision = 'left';
       path(t+1).x = max((path(t).x - 1),0);
       path(t+1).y = path(t).y;
    end

    if max_index == right
       % i.e. best decision is to move down
       % saturate to stop going off the edge of the airspace
       path(t).optimal_decision = 'right';
       path(t+1).x = min((path(t).x + 1),Gx);
       path(t+1).y = path(t).y;
    end
end
```

Display the optimal path

```
for i = 1:length(path)-1
   disp(['timestep: ' num2str(i) ...
       '     x = ' num2str(path(i).x) ...
       '     y = ' num2str(path(i).y) ...
       '     optimal decision = ' ...
       num2str(path(i).optimal_decision)])

end
```

```
timestep: 1     x = 1     y = 13     optimal decision = down
timestep: 2     x = 1     y = 12     optimal decision = down
timestep: 3     x = 1     y = 11     optimal decision = right
timestep: 4     x = 2     y = 11     optimal decision = right
timestep: 5     x = 3     y = 11     optimal decision = right
timestep: 6     x = 4     y = 11     optimal decision = right
timestep: 7     x = 5     y = 11     optimal decision = right
timestep: 8     x = 6     y = 11     optimal decision = right
timestep: 9     x = 7     y = 11     optimal decision = right
timestep: 10     x = 8     y = 11     optimal decision = right
timestep: 11     x = 9     y = 11     optimal decision = right
timestep: 12     x = 10     y = 11     optimal decision = right
timestep: 13     x = 11     y = 11     optimal decision = right
timestep: 14     x = 12     y = 11     optimal decision = right
timestep: 15     x = 13     y = 11     optimal decision = right
timestep: 16     x = 14     y = 11     optimal decision = right
```

```
timestep: 17     x = 15     y = 11     optimal decision = down
timestep: 18     x = 15     y = 10     optimal decision = down
timestep: 19     x = 15     y = 9     optimal decision = down
timestep: 20     x = 15     y = 8     optimal decision = down
timestep: 21     x = 15     y = 7     optimal decision = left
timestep: 22     x = 14     y = 7     optimal decision = left
timestep: 23     x = 13     y = 7     optimal decision = left
timestep: 24     x = 12     y = 7     optimal decision = left
timestep: 25     x = 11     y = 7     optimal decision = left
timestep: 26     x = 10     y = 7     optimal decision = left
timestep: 27     x = 9     y = 7     optimal decision = left
timestep: 28     x = 8     y = 7     optimal decision = left
timestep: 29     x = 7     y = 7     optimal decision = left
timestep: 30     x = 6     y = 7     optimal decision = left
timestep: 31     x = 5     y = 7     optimal decision = down
timestep: 32     x = 5     y = 6     optimal decision = down
timestep: 33     x = 5     y = 5     optimal decision = down
timestep: 34     x = 5     y = 4     optimal decision = right
timestep: 35     x = 6     y = 4     optimal decision = right
timestep: 36     x = 7     y = 4     optimal decision = right
timestep: 37     x = 8     y = 4     optimal decision = stay
timestep: 38     x = 8     y = 4     optimal decision = stay
timestep: 39     x = 8     y = 4     optimal decision = stay
timestep: 40     x = 8     y = 4     optimal decision = stay
timestep: 41     x = 8     y = 4     optimal decision = stay
timestep: 42     x = 8     y = 4     optimal decision = stay
timestep: 43     x = 8     y = 4     optimal decision = stay
timestep: 44     x = 8     y = 4     optimal decision = stay
timestep: 45     x = 8     y = 4     optimal decision = stay
timestep: 46     x = 8     y = 4     optimal decision = stay
timestep: 47     x = 8     y = 4     optimal decision = stay
timestep: 48     x = 8     y = 4     optimal decision = stay
timestep: 49     x = 8     y = 4     optimal decision = stay
```

Published with MATLAB® R2016b

Exercise 7.4 part 2

Table of Contents

Environment setup

```
close all
clear all
import brml.*
load airplane.mat   % Gives matrix 'U', which is the state utilities
```

Initialisation

```
% Size of airspace
Gx = 18;
Gy = 15;
```

Set up a transition probability matrix for each action.

I.e. conditional on each decision {stay, up, down, left, right} Keep as generic as possible, so define an entire x,y matrix or probabilitites for each x,y position.

```
% Manage as a cell array or probability matrices for each
% decision.  So trans_prob_stay{x1,y1}(x2,y2) is the probability
% of moving from (x1,y1) to (x2,y2) under decision 'stay'.

% Declare the transition probability matrix for each action.
trans_prob_stay = cell(Gx, Gy);
trans_prob_up = cell(Gx, Gy);
trans_prob_down = cell(Gx, Gy);
trans_prob_left = cell(Gx, Gy);
trans_prob_right = cell(Gx, Gy);

% Initialise with zero probabilities
 for x = 1:Gx
    for y = 1:Gy

        trans_prob_stay{x,y} = zeros(Gx,Gy);
        trans_prob_up{x,y} = zeros(Gx,Gy);
```

```
        trans_prob_down{x,y} = zeros(Gx,Gy);
        trans_prob_left{x,y} = zeros(Gx,Gy);
        trans_prob_right{x,y} = zeros(Gx,Gy);

    end
end
```

Populate transition probability matrices

Implicity here we account for the fact that you can't move outside the airspace. For example, if you are on the edge and try to move out of the airspace, we just set a transition probability to your current location as 1. This is achieved using the max/min expressions below.

```
for x = 1:Gx
   for y = 1:Gy

       % "For stay, the airplane stays in the same x, y position."
       trans_prob_stay{x,y}(x,y) = 1;

       % "For up, the airplane moves to the x, y + 1 position."
       trans_prob_up{x,y}(x,min(y+1,Gy)) = 1;

       % "For down, the airplane moves to the x, y ? 1 position."
       trans_prob_down{x,y}(x,max(y-1,1)) = 1;

       % "For left, the airplane moves to the x ? 1, y position."
       trans_prob_left{x,y}(max(x-1,1),y) = 1;

       % "For right, the airplane moves to the x + 1, y position."
       trans_prob_right{x,y}(min(x+1,Gx),y) = 1;

   end
end

% Modifications to the 'right' transition for part (b)

% "Provided x + 1, y is in the airspace for current position x, y:"
% "If x,y+1 is out of the airspace, then we go right to x+1,y with
% probability 1." - already covered by part (a) initialise


% "Provided x + 1, y is in the airspace for current position x, y:"
% "If x,y+1 is in the airspace, we go right to x+1,y with probability
% 0.9 and up to x,y+1 with probability 0.1."

for x = 1: (Gx-1)  % i.e. not final row (from question)
   for y = 1:(Gy-1)  % i.e. not final column (from question)

       trans_prob_right{x,y}(x+1,y) = 0.9; % no need for 'min'
       trans_prob_right{x,y}(x,y+1) = 0.1; % no need for 'min'

   end
end
```

Working back through time, apply the Bellman equation recursively

```
T = 50; % up to 18*15; % number of timesteps to evaluate

V = cell(1,T); % Values matrix, with (reverse) cumulative utilities

V{1,T} = U; % Value of the final timestep is the utility value.

for t = T:-1:2
    % This loop implements the Bellman equation here
    % Notation:

    % x_t is x(t)
    % x_t_minus_1 is x(t-1)
    % 'dec' means 'decision'


    % For each location
    for x_t_minus_1 = 1:Gx
        for y_t_minus_1 = 1:Gy

            % For each decision
            % Work out the expected value resulting from that decision
            % (and subsequent optimal decisions)

            % Initialise totals
            dec_stay_total = 0;
            dec_up_total = 0;
            dec_down_total = 0;
            dec_left_total = 0;
            dec_right_total = 0;

            % In this question it is not strictly necessary to loop
            % over all squares, as we know there are many probabilites
            % of 0 in the transition matrix, but this keeps the
            % solution general.

            % Notation is a little cumbersome here but it matches
            % the Bellman equations in the BRML book.

            for x_t = 1:Gx
                for y_t = 1:Gy

                    dec_stay_total = dec_stay_total ...
                        + trans_prob_stay{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                    dec_up_total = dec_up_total ...
                        + trans_prob_up{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
```

```
                        * V{1,t}(x_t, y_t);

                    dec_down_total = dec_down_total ...
                        + trans_prob_down{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                    dec_left_total = dec_left_total...
                        + trans_prob_left{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                    dec_right_total = dec_right_total...
                        + trans_prob_right{x_t_minus_1,y_t_minus_1}
(x_t,y_t)...
                        * V{1,t}(x_t, y_t);

                end
            end

            % Calculate the max of the totals
            max_total = max([dec_stay_total, dec_up_total, ...
                dec_down_total, dec_left_total, dec_right_total]);

            % Then add the utility of the current state.
            V{1,t-1}(x_t_minus_1,y_t_minus_1) = ...
                max_total + U(x_t_minus_1,y_t_minus_1);

        end
    end
end
```

Now work forwards from timestep 1 to calculate optimum path

Basically trying to climb the 'steepest' path

```
% Put optimal decision path in a struct to keep track of x, y and
 decision
% Structure is indexed by timestep, and has fields 'x', 'y' and
% 'optimal_decison'.

path = struct;

% initialise start point with timestep 1 information, given
% in question
path(1).x = 1;
path(1).y = 13;

% for each timestep, working forwards
for t = 1:T-1
```

```
% for each decision, keep track of totals
total_stay = 0;
total_up = 0;
total_down = 0;
total_left = 0;
total_right = 0;

x_t = path(t).x;
y_t = path(t).y;

% For each location
for x_t_plus_1 = 1:Gx
    for y_t_plus_1 = 1:Gy

        % for the current location, for each square that you can
        % transition to, calculate the probability  of
        % transitioning to that point, times the value of being in
        % that point at the next step.

        % Notation is a little cumbersome here but it matches
        % the Bellman equations in the BRML book.

        total_stay = total_stay...
            + trans_prob_stay{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
            * V{1,t+1}(x_t_plus_1,y_t_plus_1);

        total_up = total_up...
            + trans_prob_up{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
            * V{1,t+1}(x_t_plus_1,y_t_plus_1);

        total_down = total_down...
            + trans_prob_down{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
            * V{1,t+1}(x_t_plus_1,y_t_plus_1);

        total_left = total_left...
            + trans_prob_left{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
            * V{1,t+1}(x_t_plus_1,y_t_plus_1);

        total_right = total_right...
            + trans_prob_right{x_t,y_t}(x_t_plus_1,y_t_plus_1)...
            * V{1,t+1}(x_t_plus_1,y_t_plus_1);

    end
end

% Now find the highest total value (this is the 'argmax',
% to find the optimal decision).

% put totals in an array to be able to take max
stay = 1; up = 2; down = 3; left = 4; right = 5;

totals(stay) = total_stay;
totals(up) = total_up;
totals(down) = total_down;
```

```
        totals(left) = total_left;
        totals(right) = total_right;

        [value, max_index] = max(totals);

        if max_index == stay
           % i.e. best decision is to stay
           path(t).optimal_decision = 'stay';
           path(t+1).x = path(t).x;
           path(t+1).y = path(t).y;
        end

        if max_index == up
           % i.e. best decision is to move up
           % saturate to stop going off the edge of the airspace
           path(t).optimal_decision = 'up';
           path(t+1).x = path(t).x;
           path(t+1).y = min((path(t).y + 1),Gy);
        end

        if max_index == down
           % i.e. best decision is to move down
           % saturate to stop going off the edge of the airspace
           path(t).optimal_decision = 'down';
           path(t+1).x = path(t).x;
           path(t+1).y = max((path(t).y - 1),0);
        end

        if max_index == left
           % i.e. best decision is to move down
           % saturate to stop going off the edge of the airspace
           path(t).optimal_decision = 'left';
           path(t+1).x = max((path(t).x - 1),0);
           path(t+1).y = path(t).y;
        end

        if max_index == right
           % i.e. best decision is to move down
           % saturate to stop going off the edge of the airspace
           path(t).optimal_decision = 'right';
           path(t+1).x = min((path(t).x + 1),Gx);
           path(t+1).y = path(t).y;
        end
    end
```

Display the optimal path

```
    for i = 1:length(path)-1
       disp(['timestep: ' num2str(i) ...
           '     x = ' num2str(path(i).x) ...
           '     y = ' num2str(path(i).y) ...
           '     optimal decision = ' ...
           num2str(path(i).optimal_decision)])
```

```
end
```

```
timestep: 1     x = 1     y = 13     optimal decision = up
timestep: 2     x = 1     y = 14     optimal decision = right
timestep: 3     x = 2     y = 14     optimal decision = right
timestep: 4     x = 3     y = 14     optimal decision = right
timestep: 5     x = 4     y = 14     optimal decision = right
timestep: 6     x = 5     y = 14     optimal decision = right
timestep: 7     x = 6     y = 14     optimal decision = right
timestep: 8     x = 7     y = 14     optimal decision = right
timestep: 9     x = 8     y = 14     optimal decision = right
timestep: 10     x = 9     y = 14     optimal decision = right
timestep: 11     x = 10     y = 14     optimal decision = right
timestep: 12     x = 11     y = 14     optimal decision = right
timestep: 13     x = 12     y = 14     optimal decision = right
timestep: 14     x = 13     y = 14     optimal decision = right
timestep: 15     x = 14     y = 14     optimal decision = right
timestep: 16     x = 15     y = 14     optimal decision = right
timestep: 17     x = 16     y = 14     optimal decision = down
timestep: 18     x = 16     y = 13     optimal decision = down
timestep: 19     x = 16     y = 12     optimal decision = down
timestep: 20     x = 16     y = 11     optimal decision = down
timestep: 21     x = 16     y = 10     optimal decision = down
timestep: 22     x = 16     y = 9     optimal decision = down
timestep: 23     x = 16     y = 8     optimal decision = down
timestep: 24     x = 16     y = 7     optimal decision = left
timestep: 25     x = 15     y = 7     optimal decision = left
timestep: 26     x = 14     y = 7     optimal decision = left
timestep: 27     x = 13     y = 7     optimal decision = left
timestep: 28     x = 12     y = 7     optimal decision = left
timestep: 29     x = 11     y = 7     optimal decision = left
timestep: 30     x = 10     y = 7     optimal decision = left
timestep: 31     x = 9     y = 7     optimal decision = left
timestep: 32     x = 8     y = 7     optimal decision = left
timestep: 33     x = 7     y = 7     optimal decision = left
timestep: 34     x = 6     y = 7     optimal decision = left
timestep: 35     x = 5     y = 7     optimal decision = down
timestep: 36     x = 5     y = 6     optimal decision = down
timestep: 37     x = 5     y = 5     optimal decision = down
timestep: 38     x = 5     y = 4     optimal decision = right
timestep: 39     x = 6     y = 4     optimal decision = right
timestep: 40     x = 7     y = 4     optimal decision = right
timestep: 41     x = 8     y = 4     optimal decision = stay
timestep: 42     x = 8     y = 4     optimal decision = stay
timestep: 43     x = 8     y = 4     optimal decision = stay
timestep: 44     x = 8     y = 4     optimal decision = stay
timestep: 45     x = 8     y = 4     optimal decision = stay
timestep: 46     x = 8     y = 4     optimal decision = stay
timestep: 47     x = 8     y = 4     optimal decision = stay
timestep: 48     x = 8     y = 4     optimal decision = stay
timestep: 49     x = 8     y = 4     optimal decision = stay
```

Exercise 7.17

Clearly $E_{25,50} = E_{-25,50} = 1$ where

$E_{s,t}$ is the optimal expected reward for being in position s at time t

Also $E_{s,50} = 0$ for all $s \neq \{-25, +25\}$

At timestep 49

$$E_{-25,49} = E_{+25,49} = 0.8$$ since there

is 20% probability that even having selected "stay", the noise in the environment will thwart you.

Since the noise is random and iid, you cannot achieve a position at timestep 49 with a value higher than 0.8. But only position within 50 steps of +25 or -25 can achieve this position.

Therefore the expected rewards under optimal play are the same whether the first step is down, same or up at timestep $t=1$

Problem 9.1

Table of Contents


```
clear all;
close all;
load('printer.mat');
import brml.*
```

Problem SETUP

```
%PRINTER NIGHTMARE

% Assign index to all 10 variables,
% possible problems/ diagnoses:
PFuse = 1;
PDrum = 2;
PToner = 3;
PPaper = 4;
PRoller = 5;
%  possible consequences/ faults:
PBurning = 6;
PQuality = 7;
PWrinkled = 8;
PMultiplePages = 9;
PPaperJam = 10;

% Assume table provided is such that 1 means fault has occurred and 0
 means
% there was no fault. This is not explicitly stated in question
% formulation. Below assign indexes to false and true
F=1;    % False is equivalent to 0 in table
T=2;    % True is equivalent to 1 in table
```

Part 1

Learn all table entries on the basis of maximum likelihood.

```
%   BELIEF NETWORK:
%   P(B,Q,W..... R) = P(F)*P(D)*P(T)*P(P)*P(R)*P(B|F)*P(Q|D,T,P)...
%                       *P(W|F,P)*P(M|P,R)*P(PJ|R,F)
%
```

```
% Finding each term separately and storing in pot cell:
tempPot = array(PFuse);
tempPot.table(F) = sum(x(PFuse,:) == F)/length(x(PFuse,:));
tempPot.table(T) = 1 - tempPot.table(F);
pot{PFuse} = tempPot;

tempPot = array(PDrum);
tempPot.table(F) = sum(x(PDrum,:) == F)/length(x(PDrum,:));
tempPot.table(T) = 1 - tempPot.table(F);
pot{PDrum} = tempPot;

tempPot = array(PToner);
tempPot.table(F) = sum(x(PToner,:) == F)/length(x(PToner,:));
tempPot.table(T) = 1 - tempPot.table(F);
pot{PToner} = tempPot;

tempPot = array(PPaper);
tempPot.table(F) = sum(x(PPaper,:) == F)/length(x(PPaper,:));
tempPot.table(T) = 1 - tempPot.table(F);
pot{PPaper} = tempPot;

tempPot = array(PRoller);
tempPot.table(F) = sum(x(PRoller,:) == F)/length(x(PRoller,:));
tempPot.table(T) = 1 - tempPot.table(F);
pot{PRoller} = tempPot;

tempPot = array([PBurning PFuse]); %burning given fuse
tempPot.table(F,F) = sum(x(PBurning,x(PFuse,:) == F)==F)/
length(find(x(PFuse,:) == F));
tempPot.table(F,T) = sum(x(PBurning,x(PFuse,:) == T)==F)/
length(find(x(PFuse,:) == T));
tempPot.table(T,:) = 1 - tempPot.table(F,:);
pot{PBurning} = condpot(tempPot,PBurning,PFuse);

tempPot = array([PQuality PDrum PToner PPaper]); %P(Q|D,T,P)
%P(Q=0|D,...,P=0) ETC.:
tempPot.table(F,F,F,F) = sum(x(PQuality,x(PDrum,:) == F & x(PToner,:)
 == F & x(PPaper,:) == F)==F)/length(find((x(PDrum,:) == F &
 x(PToner,:) == F & x(PPaper,:) == F)==F));
tempPot.table(F,F,F,T) = sum(x(PQuality,x(PDrum,:) == F & x(PToner,:)
 == F & x(PPaper,:) == T)==F)/length(find((x(PDrum,:) == F &
 x(PToner,:) == F & x(PPaper,:) == T)==F));
tempPot.table(F,F,T,F) = sum(x(PQuality,x(PDrum,:) == F & x(PToner,:)
 == T & x(PPaper,:) == F)==F)/length(find((x(PDrum,:) == F &
 x(PToner,:) == T & x(PPaper,:) == F)==F));
tempPot.table(F,F,T,T) = sum(x(PQuality,x(PDrum,:) == F & x(PToner,:)
 == T & x(PPaper,:) == T)==F)/length(find((x(PDrum,:) == F &
 x(PToner,:) == T & x(PPaper,:) == T)==F));
tempPot.table(F,T,F,F) = sum(x(PQuality,x(PDrum,:) == T & x(PToner,:)
 == F & x(PPaper,:) == F)==F)/length(find((x(PDrum,:) == T &
 x(PToner,:) == F & x(PPaper,:) == F)==F));
tempPot.table(F,T,F,T) = sum(x(PQuality,x(PDrum,:) == T & x(PToner,:)
 == F & x(PPaper,:) == T)==F)/length(find((x(PDrum,:) == T &
 x(PToner,:) == F & x(PPaper,:) == T)==F));
```

```
tempPot.table(F,T,T,F) = sum(x(PQuality,x(PDrum,:) == T & x(PToner,:)
 == T & x(PPaper,:) == F)==F)/length(find((x(PDrum,:) == T &
 x(PToner,:) == T & x(PPaper,:) == F)==F));
tempPot.table(F,T,T,T) = sum(x(PQuality,x(PDrum,:) == T & x(PToner,:)
 == T & x(PPaper,:) == T)==F)/length(find((x(PDrum,:) == T &
 x(PToner,:) == T & x(PPaper,:) == T)==F));
tempPot.table(T,:,:,:) = 1 - tempPot.table(F,:,:,:);
pot{PQuality} = condpot(tempPot,PQuality,[PDrum PToner PPaper]);

tempPot = array([PWrinkled PFuse PPaper]); %P(W|F,P)
tempPot.table(F,F,F) = sum(x(PWrinkled,x(PFuse,:) == F & x(PPaper,:)
 == F)==F)/length(find((x(PFuse,:) == F & x(PPaper,:) == F)==F));
tempPot.table(F,F,T) = sum(x(PWrinkled,x(PFuse,:) == F & x(PPaper,:)
 == T)==F)/length(find((x(PFuse,:) == F & x(PPaper,:) == T)==F));
tempPot.table(F,T,F) = sum(x(PWrinkled,x(PFuse,:) == T & x(PPaper,:)
 == F)==F)/length(find((x(PFuse,:) == T & x(PPaper,:) == F)==F));
tempPot.table(F,T,T) = sum(x(PWrinkled,x(PFuse,:) == T & x(PPaper,:)
 == T)==F)/length(find((x(PFuse,:) == T & x(PPaper,:) == T)==F));
tempPot.table(T,:,:) = 1 - tempPot.table(F,:,:);
pot{PWrinkled} = condpot(tempPot,PWrinkled,[PFuse PPaper]);

tempPot = array([PMultiplePages PPaper PRoller]); %P(M|R,P)
tempPot.table(F,F,F) = sum(x(PMultiplePages,x(PPaper,:) == F &
 x(PRoller,:) == F)==F)/length(find((x(PPaper,:) == F & x(PRoller,:)
 == F)==F));
tempPot.table(F,F,T) = sum(x(PMultiplePages,x(PPaper,:) == F &
 x(PRoller,:) == T)==F)/length(find((x(PPaper,:) == F & x(PRoller,:)
 == T)==F));
tempPot.table(F,T,F) = sum(x(PMultiplePages,x(PPaper,:) == T &
 x(PRoller,:) == F)==F)/length(find((x(PPaper,:) == T & x(PRoller,:)
 == F)==F));
tempPot.table(F,T,T) = sum(x(PMultiplePages,x(PPaper,:) == T &
 x(PRoller,:) == T)==F)/length(find((x(PPaper,:) == T & x(PRoller,:)
 == T)==F));
tempPot.table(T,:,:) = 1 - tempPot.table(F,:,:);
pot{PMultiplePages} = condpot(tempPot,PMultiplePages,[PPaper
 PRoller]);

tempPot = array([PPaperJam PFuse PRoller]); %P(PJ|R,F)
tempPot.table(F,F,F) = sum(x(PPaperJam,x(PFuse,:) == F & x(PRoller,:)
 == F)==F)/length(find((x(PFuse,:) == F & x(PRoller,:) == F)==F));
tempPot.table(F,F,T) = sum(x(PPaperJam,x(PFuse,:) == F & x(PRoller,:)
 == T)==F)/length(find((x(PFuse,:) == F & x(PRoller,:) == T)==F));
tempPot.table(F,T,F) = sum(x(PPaperJam,x(PFuse,:) == T & x(PRoller,:)
 == F)==F)/length(find((x(PFuse,:) == T & x(PRoller,:) == F)==F));
tempPot.table(F,T,T) = sum(x(PPaperJam,x(PFuse,:) == T & x(PRoller,:)
 == T)==F)/length(find((x(PFuse,:) == T & x(PRoller,:) == T)==F));
tempPot.table(T,:,:) = 1 - tempPot.table(F,:,:);
pot{PPaperJam} = condpot(tempPot,PPaperJam,[PFuse PRoller]);
```

Part 2

```
%Program the belief network using the tables maximum likelihood tables
 and BRMLtoolbox

%initialise consequence and their states for further use in code:
setpotValues = [PBurning PPaperJam PQuality PWrinkled PMultiplePages];
setpotIndex = [T T F F F];

% sum over D,T,R,P of P(F,D,T,R,P|B=T,PJ=T,Q=F,W=F,M=F)
% to get P(F|B=T,PJ=T,Q=F,W=F,M=F):

Jointpot = multpots(pot); % MULTPOTS Multiply potentials into a single
 potential

FuseBad = setpot(sumpot(Jointpot,[PDrum PToner PPaper
 PRoller]),setpotValues,setpotIndex);
    % where %SETPOT sets brml.variables to specified states
    % newpot = setpot(pot,variables,evidstates)

FuseBad.table = FuseBad.table/sum(FuseBad.table);

fprintf('The probability of a fuse assembly malfunction given the
 observed burning smell and paper jam is: %f\n',FuseBad.table(T));
```

```
The probability of a fuse assembly malfunction given the observed
 burning smell and paper jam is: 1.000000
```

Part 3

learning binary vairables using Beta(1,1) prior- flat prior. E.g. P(F)=(1+count)/(2+total) as opposed to previous approach where P(F)=count/total Below code is the repetition of part 1 but with this new prior:

```
tempPot = array(PFuse);
tempPot.table(F) = (1+sum(x(PFuse,:) == F))/(2+length(x(PFuse,:)));
tempPot.table(T) = 1 - tempPot.table(F);
BayPot{PFuse} = tempPot;

tempPot = array(PDrum);
tempPot.table(F) = (1+sum(x(PDrum,:) == F))/(2+length(x(PDrum,:)));
tempPot.table(T) = 1 - tempPot.table(F);
BayPot{PDrum} = tempPot;

tempPot = array(PToner);
tempPot.table(F) = (1+sum(x(PToner,:) == F))/(2+length(x(PToner,:)));
tempPot.table(T) = 1 - tempPot.table(F);
BayPot{PToner} = tempPot;

tempPot = array(PPaper);
tempPot.table(F) = (1+sum(x(PPaper,:) == F))/(2+length(x(PPaper,:)));
tempPot.table(T) = 1 - tempPot.table(F);
BayPot{PPaper} = tempPot;
```

```
tempPot = array(PRoller);
tempPot.table(F) = (1+sum(x(PRoller,:) == F))/
(2+length(x(PRoller,:)));
tempPot.table(T) = 1 - tempPot.table(F);
BayPot{PRoller} = tempPot;

tempPot = array([PBurning PFuse]);
tempPot.table(F,F) = (1+sum(x(PBurning,x(PFuse,:) == F)==F))/
(2+length(find(x(PFuse,:) == F)));
tempPot.table(F,T) = (1+sum(x(PBurning,x(PFuse,:) == T)==F))/
(2+length(find(x(PFuse,:) == T)));
tempPot.table(T,:) = 1 - tempPot.table(F,:);
BayPot{PBurning} = condpot(tempPot,PBurning,PFuse);

tempPot = array([PQuality PDrum PToner PPaper]);
tempPot.table(F,F,F,F) = (1+sum(x(PQuality,x(PDrum,:) == F &
 x(PToner,:) == F & x(PPaper,:) == F)==F))/(2+length(find((x(PDrum,:)
 == F & x(PToner,:) == F & x(PPaper,:) == F)==F)));
tempPot.table(F,F,F,T) = (1+sum(x(PQuality,x(PDrum,:) == F &
 x(PToner,:) == F & x(PPaper,:) == T)==F))/(2+length(find((x(PDrum,:)
 == F & x(PToner,:) == F & x(PPaper,:) == T)==F)));
tempPot.table(F,F,T,F) = (1+sum(x(PQuality,x(PDrum,:) == F &
 x(PToner,:) == T & x(PPaper,:) == F)==F))/(2+length(find((x(PDrum,:)
 == F & x(PToner,:) == T & x(PPaper,:) == F)==F)));
tempPot.table(F,F,T,T) = (1+sum(x(PQuality,x(PDrum,:) == F &
 x(PToner,:) == T & x(PPaper,:) == T)==F))/(2+length(find((x(PDrum,:)
 == F & x(PToner,:) == T & x(PPaper,:) == T)==F)));
tempPot.table(F,T,F,F) = (1+sum(x(PQuality,x(PDrum,:) == T &
 x(PToner,:) == F & x(PPaper,:) == F)==F))/(2+length(find((x(PDrum,:)
 == T & x(PToner,:) == F & x(PPaper,:) == F)==F)));
tempPot.table(F,T,F,T) = (1+sum(x(PQuality,x(PDrum,:) == T &
 x(PToner,:) == F & x(PPaper,:) == T)==F))/(2+length(find((x(PDrum,:)
 == T & x(PToner,:) == F & x(PPaper,:) == T)==F)));
tempPot.table(F,T,T,F) = (1+sum(x(PQuality,x(PDrum,:) == T &
 x(PToner,:) == T & x(PPaper,:) == F)==F))/(2+length(find((x(PDrum,:)
 == T & x(PToner,:) == T & x(PPaper,:) == F)==F)));
tempPot.table(F,T,T,T) = (1+sum(x(PQuality,x(PDrum,:) == T &
 x(PToner,:) == T & x(PPaper,:) == T)==F))/(2+length(find((x(PDrum,:)
 == T & x(PToner,:) == T & x(PPaper,:) == T)==F)));
tempPot.table(T,:,:,:) = 1 - tempPot.table(F,:,:,:);
BayPot{PQuality} = condpot(tempPot,PQuality,[PDrum PToner PPaper]);

tempPot = array([PWrinkled PFuse PPaper]);
tempPot.table(F,F,F) = (1+sum(x(PWrinkled,x(PFuse,:) == F &
 x(PPaper,:) == F)==F))/(2+length(find((x(PFuse,:) == F & x(PPaper,:)
 == F)==F)));
tempPot.table(F,F,T) = (1+sum(x(PWrinkled,x(PFuse,:) == F &
 x(PPaper,:) == T)==F))/(2+length(find((x(PFuse,:) == F & x(PPaper,:)
 == T)==F)));
tempPot.table(F,T,F) = (1+sum(x(PWrinkled,x(PFuse,:) == T &
 x(PPaper,:) == F)==F))/(2+length(find((x(PFuse,:) == T & x(PPaper,:)
 == F)==F)));
```

```
tempPot.table(F,T,T) = (1+sum(x(PWrinkled,x(PFuse,:) == T &
 x(PPaper,:) == T)==F))/(2+length(find((x(PFuse,:) == T & x(PPaper,:)
 == T)==F)));
tempPot.table(T,:,:) = 1 - tempPot.table(F,:,:);
BayPot{PWrinkled} = condpot(tempPot,PWrinkled,[PFuse PPaper]);

tempPot = array([PMultiplePages PPaper PRoller]);
tempPot.table(F,F,F) = (1+sum(x(PMultiplePages,x(PPaper,:) ==
 F & x(PRoller,:) == F)==F))/(2+length(find((x(PPaper,:) == F &
 x(PRoller,:) == F)==F)));
tempPot.table(F,F,T) = (1+sum(x(PMultiplePages,x(PPaper,:) ==
 F & x(PRoller,:) == T)==F))/(2+length(find((x(PPaper,:) == F &
 x(PRoller,:) == T)==F)));
tempPot.table(F,T,F) = (1+sum(x(PMultiplePages,x(PPaper,:) ==
 T & x(PRoller,:) == F)==F))/(2+length(find((x(PPaper,:) == T &
 x(PRoller,:) == F)==F)));
tempPot.table(F,T,T) = (1+sum(x(PMultiplePages,x(PPaper,:) ==
 T & x(PRoller,:) == T)==F))/(2+length(find((x(PPaper,:) == T &
 x(PRoller,:) == T)==F)));
tempPot.table(T,:,:) = 1 - tempPot.table(F,:,:);
BayPot{PMultiplePages} = condpot(tempPot,PMultiplePages,[PPaper
 PRoller]);

tempPot = array([PPaperJam PFuse PRoller]);
tempPot.table(F,F,F) = (1+sum(x(PPaperJam,x(PFuse,:) == F &
 x(PRoller,:) == F)==F))/(2+length(find((x(PFuse,:) == F &
 x(PRoller,:) == F)==F)));
tempPot.table(F,F,T) = (1+sum(x(PPaperJam,x(PFuse,:) == F &
 x(PRoller,:) == T)==F))/(2+length(find((x(PFuse,:) == F &
 x(PRoller,:) == T)==F)));
tempPot.table(F,T,F) = (1+sum(x(PPaperJam,x(PFuse,:) == T &
 x(PRoller,:) == F)==F))/(2+length(find((x(PFuse,:) == T &
 x(PRoller,:) == F)==F)));
tempPot.table(F,T,T) = (1+sum(x(PPaperJam,x(PFuse,:) == T &
 x(PRoller,:) == T)==F))/(2+length(find((x(PFuse,:) == T &
 x(PRoller,:) == T)==F)));
tempPot.table(T,:,:) = 1 - tempPot.table(F,:,:);
BayPot{PPaperJam} = condpot(tempPot,PPaperJam,[PFuse PRoller]);

BayJointPot = multpots(BayPot); %each variable only once
FuseBay = setpot(sumpot(BayJointPot,[PDrum PToner PPaper PRoller]),
[PBurning PPaperJam PQuality PWrinkled PMultiplePages],[T T F F F]);
FuseBay.table = FuseBay.table/sum(FuseBay.table);
fprintf('The probability of a fuse assembly malfunction given the
 observed burning smell and paper jam calculated under the Beysian
 framework is: %f\n',FuseBay.table(T));
```

```
The probability of a fuse assembly malfunction given the observed
 burning smell and paper jam calculated under the Beysian framework
 is: 0.618615
```

Part 4

Given the above information from the secretary, what is the most likely joint diagnosis over the diagnostic variables ? that is the joint most likely p(F use, Drum, T oner, P aper, Roller|evidence)? Use the max-absorption method on the associated junction tree.

```
% Step1
%   Program the belief network as in part 2 but with Bayesian approach
 (part
%   3). Find probability that there is a fuse assembly malfunction
 given
%   that the secretary complains there is a burning smell and that the
%   paper is jammed, and that there are no other problems.
%   Create new potential with evidential variables:
PotP4 = setpot(BayPot,setpotValues,setpotIndex);


[newpot, newvars, uniquevariables,uniquenstates] = squeezepots(PotP4);
%   SQUEEZEPOTS Eliminate redundant potentials (those contained wholly
 within
%   another) by multiplying redundant potentials and renaming
 variables
%   Squeezepots is needed as setpot removes variables so we must
 eliminate
%   redundant variables by squeezing the potentials.

% Apply Junction Tree algorithm to find the joint most likely
% P(F,D,T,P,R|evidence):

[jtpot, jtsep, infostruct] = jtree(PotP4);
%   JTREE Setup a Junction Tree based on a set of potentials
%   [jtpot jtsep infostruct]=jtree(pot)
%   For the potential pot return a set of initialised Junction Tree
%   potentials. Then, perform absprbtion:

[jtpotAbsorb, jtsepAbsorb, logZ] =
 absorption(jtpot,jtsep,infostruct,'max');
%   the max argument ensures max-absorption carried out instead of the
%   default sum absorption.


sumprob2=zeros(1,5);

for i=[1:5]
    jtp=whichpot(jtpotAbsorb,i,1);
    margpot=sumpot(jtpotAbsorb(jtp),i,0);
    someprob2(1,i)=1-(margpot.table(1)./sum(margpot.table));
end

%   get max potential for each absorption potential combination (and
 eliminate
%   repetition of Paper (4) variable)
```

```
maxPot = multpots(jtpotAbsorb);
%max pot has 2row and 5 col in 5 dimensions (2 by 5 by 5 cube)

[maxElem, maxIndex] = max(maxPot.table(:))
%   position of max index is 2 which corresponds to the maxPot table
 (:,:,2)
%   which in turn corresponds to the state where F=1 and the remaining
 variables equal 0.
%   so the most likely joint diagnosis is that the probability of
%   P(F=1,D=0,T=0,P=0,R=0|evidence) with probability  1.5824e-05

fprintf('The most likley joint diagnosis is that there was a fuse
 malfunction, given by: P(F=1,D=0,T=0,P=0,R=0|evidence) = %f.
\n',maxElem);
```

Part 5

```
%Compute the joint most likely state of the distribution
%p(F, D, T, P, R|burning smell, paper jam)

%Explain how to compute this efficiently using the max-absorption
 method.


%repeat as per part 4, but now only set values for 2 var instead of 5:
PotP5 = setpot(BayPot,[PBurning PPaperJam], [T T]);

[newpotP5, newvarsP5, uniquevariablesP5, uniquenstatesP5] =
 squeezepots(PotP5);

[jtpotP5, jtsepP5, infostructP5]=jtree(newpotP5);

%sum over other var as not conditioning on them anymore so can
 marginalise:
jtpotP5= sumpot(jtpotP5,[7 8 9]); % 7,8,9 are indeces of var we want
 to marginalise over

[jtpotAbsorbP5, jtsepAbsorbP5,
 logZP5]=absorption(jtpotP5,jtsepP5,infostructP5,'max');

maxPotP5 = multpots(jtpotAbsorbP5);

[maxElemP5, maxIndexP5] = max(maxPotP5.table(:));

fprintf('The most likely joint diagnosis is that there was a fuse
 malfunction, given by: P(F=1,D=0,T=0,P=0,R=0|Burning smell, paper
 jam) = %0.8f\n',maxElemP5);
```

```
maxElem =

   1.5824e-05
```

```
maxIndex =

     2

The most likley joint diagnosis is that there was a fuse malfunction,
 given by: P(F=1,D=0,T=0,P=0,R=0|evidence) = 0.000016.
The most likely joint diagnosis is that there was a fuse malfunction,
 given by: P(F=1,D=0,T=0,P=0,R=0|Burning smell, paper jam) =
 0.00000001
```

Commentary of results

```
disp('Results from parts 4 and 5 indicate that the most likely
 diagnosis under evidence is a fuse malfunction. Results from both
 parts are conditioned on a different set of evidence, thus the
 numerical results obtained are expected to be different. Probably in
 part 5 is less than in part 4 indicating that given less evidence we
 are also less certain of the diagnosis. ')
```

```
Results from parts 4 and 5 indicate that the most likely diagnosis
 under evidence is a fuse malfunction. Results from both parts are
 conditioned on a different set of evidence, thus the numerical
 results obtained are expected to be different. Probably in part 5 is
 less than in part 4 indicating that given less evidence we are also
 less certain of the diagnosis.
```

Published with MATLAB® R2016b

9.9 $P(D|M) = \prod_k \prod_n P(v_k^u | Pa(v_k^u)) = \prod_k \prod_j \frac{\Gamma(u'(v_k;j))}{\Gamma(u(v_k;j))}$ (1)

note: if $\underset{\sim}{\theta} \sim \text{Dirichlet}(\lambda_1, \lambda_2)$ then $P(\underset{\sim}{\theta}; \lambda_1, \lambda_2) = \frac{\Gamma(\lambda_1+\lambda_2)}{\Gamma(\lambda_1)\Gamma(\lambda_2)} \theta_1^{\lambda_1-1} \theta_2^{\lambda_2-1} =$

$$= \frac{\Gamma(\Sigma \lambda_i)}{\prod \Gamma(\lambda_i)} \prod \theta_i^{\lambda_i - 1} \quad (2)$$

consider first the second product in Equation (1) where we have: $\prod_n P(v_k^u | Pa(v_k^u)) = \prod_n \int_{\theta(u)} P(\theta(u)) P(v^u | pa(v^u), \theta(v)) d\theta(u) =$ $P(\theta(u)) \sim \text{Dirichlet}$

$$= \int_{\theta(u)} P(\theta(u)) \prod_n P(v^u | pa(v^u), \theta(v)) d\theta(u) =$$

$$= \prod_j \left\{ \left[\frac{\Gamma(\sum_i u_i(w;j))}{\prod_i \Gamma(u_i(v;j))} \right] \int_{\theta(u;j)} \left(\prod_i \theta_i(u;j)^{u_i(v;j)-1} \prod_n P(u^u = i | pa(v^u) = j, \theta_i(u;j)) \right) d\theta(v;j) \right\}$$

$$= \prod_j \left\{ \left[\frac{\Gamma(\sum_i u_i(w;j))}{\prod_i \Gamma(u_i(v;j))} \right] \int_{\theta(u;j)} \prod_i \theta_i(v;j)^{u_i(v;j)-1} \prod_n \theta_i(v;j)^{\mathbb{1}[u^u = i, pa(u^u) = j]} d\theta(u;j) \right\}$$

$$= \prod_j \left\{ \left[\frac{\Gamma(\sum_i u_i(v;j))}{\prod_i \Gamma(u_i(v;j))} \right] \int_{\theta(v;j)} \prod_i \theta_i(v;j)^{u_i(v;j)-1} \theta_i(v;j)^{\#(v=i,\, pa(v)=j)} d\theta(u;j) \right\}$$

$$= \prod_j \left\{ \left[\frac{\Gamma(\sum_i u_i(v;j))}{\prod_i \Gamma(u_i(v;j))} \right] \int_{\theta(v;j)} \prod_i \theta_i(v;j)^{u'(v;j)-1} d(\theta(u;j)) \right\} \overset{(2)}{\underset{*}{=}} \int P(\underset{\sim}{\theta}; \underset{\sim}{\lambda}) = 1$$

$$\Rightarrow \frac{\prod \Gamma(\lambda_i)}{\Gamma(\Sigma \lambda_i)} = \int \prod \theta_i^{\lambda_i - 1}$$

$$= \prod_j \left\{ \left[\frac{\Gamma(\sum_i u_i(v;j))}{\prod_i \Gamma(u_i(v;j))} \right] \left[\frac{\prod_i \Gamma(u_i'(v;j))}{\Gamma(\sum_i (u_i'(v;j)))} \right] \right\}$$

hence from (1): $P(D|M) = \prod_k \prod_n P(v_k^u | Pa(v_k^u)) = \prod_k \prod_j \left\{ \frac{\Gamma(\sum_i u_i(v;j))}{\prod_i \Gamma(u_i(v;j))} \frac{\prod_i \Gamma(u_i'(v;j))}{\Gamma(\Sigma u_i'(v;j))} \right\}$

Therefore: $P(D|M) = \prod_k \prod_j \frac{\Gamma(\sum_i u_i(v;j))}{\Gamma(\sum_i u_i'(v;j))} \prod_i \left[\frac{\Gamma(u_i'(v;j))}{\Gamma(u_i(v;j))} \right]$

Exercise 9.10

The question does not specify whether a node is allowed to have no parents We assume that this is not allowed

Therefore possible choices for a_8 are

1) one parent from $\{a_1 \dots a_7\}$

2) two parents from $\{a_1 \dots a_7\}$

This gives a total of $7 + \frac{7 \cdot 6}{2} = 28$ choices for the "parents" of a_8

Similarly a_7 has $6 + \frac{6 \cdot 5}{2} = 21$ choices

Because of the ancestral order these choices are contingent and define different graphs

Therefore the total number of graphs is

$$28 \times 21 \times 15 \times 10 \times 6 \times 3 \times 2$$
$$(a_8)\ (a_7)\ (a_6)\ (a_5)\ (a_4)\ (a_3)\ (a_2)$$

$$= 3,175,200$$

2. If the computation of the BD score "of any member" of N_a takes 1 second then since the model likelihood is the product of the individual

likelihood terms (under local and global param indept. assumptions) we just need to calculate 8 likelihoods (and multiply, which we assume takes zero time)

=> takes $8 \times 3{,}175{,}200 = 25{,}401{,}600$ seconds = 294 days

3.

We estimate the time required for N (without ancestral order) as

$8! \times 294$ days

since there are 8! possible ancestral orders on N (note that this is very clearly an upper bound, rather than any optimal algorithm)

=> time to N $\sim 40{,}320 \times$ ~~years~~

294 days $\approx \sim 32{,}500$ years

Exercise 9.13 - Part 1

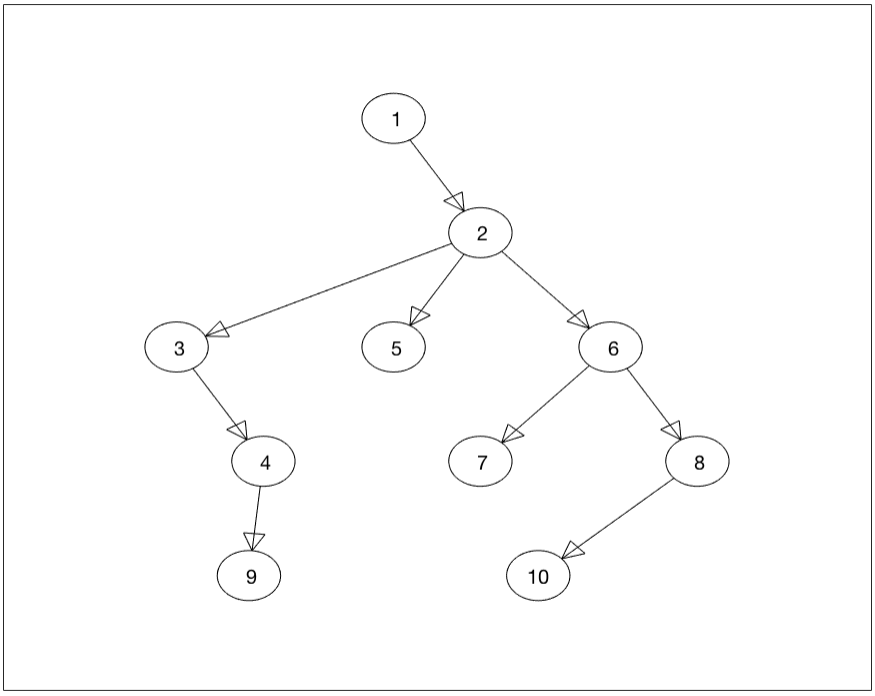
Table of Contents

Environment setup

```
close all
clear all
import brml.*
load ChowLiuData.mat   % Gives matrix 'X', which is the empirical data
```

Calculate Chow-Liu tree for data X

```
A = ChowLiu(X);
draw_layout(A);
```



Published with MATLAB® R2016b

Exercise 9.13 - Part 2

Table of Contents

function A = ChowLiu(X)

```
function A = ChowLiu(X)

%CHOWLIU Calculate ChowLiu Tree from data
% A = ChowLiu(X)
%
% Inputs:
% X : Multidimensional data.  One column per data point, one row per
%     variable corresponding to variables labelled 1,2,3,... according
%     to row number.
%
% Outputs:
% A : Sparse adjacency matrix of ChowLiu Tree, with directions of
%     edges starting from node 1.
```

Environment setup

```
import brml.*
```

Initialisation

```
% D dimensions (i.e. different variables, rows of X)
% N datapoints (i.e. cols of X)
[D, N] = size(X);
```

Find max/min values of each variable

This is needed to have an efficient sum over each variable in order to calculate the mutual information.

```
x_max = max(X,[],2);  % Vector of max values of rows of X
x_min = min(X,[],2);  % Vector of min values of rows of X
```

Count the occurrences of each value of each variable (pair)

Needed for calculation of empirical probabilities

```
single_counts = cell(D,1); % allocate memory

% for each individual variable
for i = 1:D

    % for each value in range of that variable
    for value = x_min(i):x_max(i)

        % count the number of times variable i has this value
        % and store in cell array.
        single_counts{i,1}(value,1) = sum(X(i,:) == value);

    end
end


joint_counts = cell(D,D);  % allocate memory

% for each pair of variables
for i = 1:D
    for j = 1:D

        % for each pair of values in range of that pair of variables
        for i_value = x_min(i):x_max(i)
            for j_value = x_min(j):x_max(j)

                % count the number of times this pair of variable
                % takes this pair of values, and store in a cell
                % array.
                joint_counts{i,j}(i_value,j_value) = ...
                    sum(X(i,:) == i_value & X(j,:) == j_value);
            end
        end
    end
end
```

Convert the counts into emprirical probabilities.

Needed for mutual information calculation

```
single_probs = cell(D,1); % allocate memory

% for each individual variable
for i = 1:D
```

```
        total_counts = sum(single_counts{i,1});

        % for each value in range of that variable
        for value = x_min(i):x_max(i)

            % Divide the counts for that value by the total counts
            % to give an empirical single-variable probability
            % distribution
            single_probs{i,1}(value,1) = ...
                single_counts{i,1}(value,1) / total_counts;

        end
    end


    joint_probs = cell(D,D);  % allocate memory

    % for each pair of variables
    for i = 1:D
        for j = 1:D

            % sum over both rows and cols
            total_counts = sum(sum(joint_counts{i,j}));

            % for each pair of values in range of that pair of
            % variables
            for i_value = x_min(i):x_max(i)
                for j_value = x_min(j):x_max(j)

                    % Divide the counts for that particular pair of
                    % values by the total counts for that pair of
                    % variables to give an empirical joint probability
                    % distribution
                    joint_probs{i,j}(i_value,j_value) = ...
                        joint_counts{i,j}(i_value,j_value) / total_counts;

                end
            end
        end
    end
```

Calculate the mutual information between each pair of variables

```
    mutual_information = zeros(D); % Allocate memory

    % Loop through each pair of variables
    for i = 1:D
        for j = 1:D

            % working with variables i and j
            % sum over all the possible values of i and j
```

```
            % calculate the mutual information using the empirical
            % probabilities calculated using the counts, above.
            % Store in the mutual information matrix.

            total = 0; % init

            for i_value = x_min(i):x_max(i)
                for j_value = x_min(j):x_max(j)

                    % cover case where prob and log are 0
                    if joint_probs{i,j}(i_value,j_value) ~= 0

                        total = total ...
                            + joint_probs{i,j}(i_value,j_value) ...
                            * log(joint_probs{i,j}(i_value,j_value) ...
                            / (single_probs{i,1}(i_value,1) ...
                            * single_probs{j,1}(j_value,1)));
                    end
                end
            end

            mutual_information(i,j) = total;

        end
    end
```

Produce a sorted list of edge weights

Want edges sorted in weight order, highest first. This is needed so that the spantree.m function returns the maximum weight spanning tree

```
% get the edges and weights from the mutual information matrix.
% I have added a tiny value (epsilon) to each element of the mututal
% information matrix to avoid a warning in the brml.spantree
% function.  Epsilon is so small that it will not affect the results.
epsilon = 1e-10;
[edges, weights] = brml.edges(mutual_information + epsilon);

% concatonate edges and weights into rows so that they
% stay together when they are sorted by edge weight.
edges_weights = [edges weights];

% sort by col 3, i.e. edge weight, in descending order
edges_weights_sorted = sortrows(edges_weights,-3);

% extract just the edges
edges_sorted = edges_weights_sorted(:,1:2);
```

Find maximal weight spanning tree

```
[adj_matrix, ~, ~] = spantree(edges_sorted);
```

```
% convert adj_matrix (sparse matrix) to full adjacency matrix
adj_matrix = full(adj_matrix);

% extract list of edges from spanning tree
[edgea, edgeb] = find(adj_matrix);
e = [edgea edgeb];
%weight = adj_matrix(adj_matrix~=0);
```

Convert to DAG

Start at node 1 and successively add more nodes to tree until there are none left to add.

```
connected_nodes = [1]; % init
unconnected_nodes = [2:D]; % init
dag = [];

while (isempty(unconnected_nodes) == false)

    % go through all connected nodes i and find pairs (i,j) where the
    % j is unconnected

    for i = 1:length(connected_nodes)

        % loop through edges
        for edge_index = i:size(e,1)

            node_i = e(edge_index, 1);
            node_j = e(edge_index, 2);

            if (ismember(node_i,connected_nodes) && ...
                    ismember(node_j,unconnected_nodes))

                % Append edge i -> j to the DAG
                dag = [dag ; node_i, node_j];

                % Remove node j from the list of unconnected nodes
                unconnected_nodes(unconnected_nodes==node_j) = [];

                % Add node j to the list of connected nodes
                connected_nodes = [connected_nodes, node_j];

            end
        end
    end
end
```

Construct and return sparse adjacency matrix

```
A = edges2adj(dag,D);
```

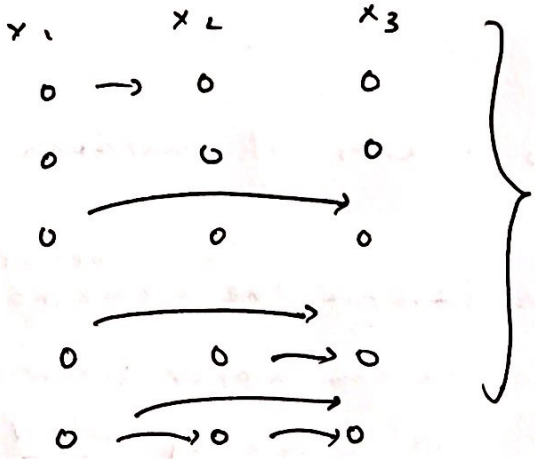
Published with MATLAB® R2016b

9.14

- Suppose we have a graph with $N > 1$

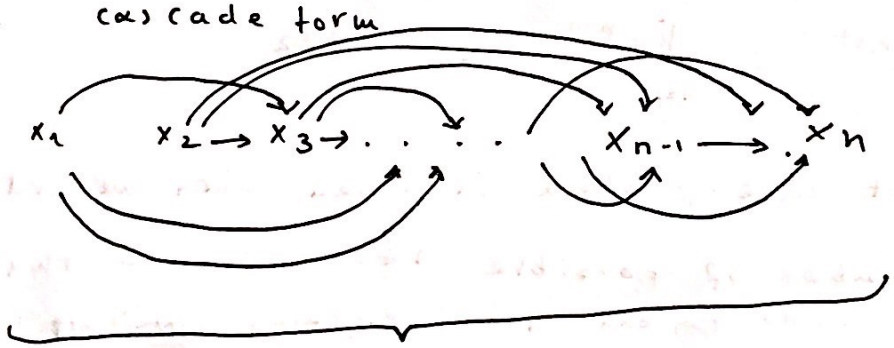
- Possible number of DAGs

Case where we have 3 variables:



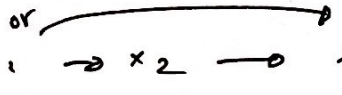
- We can see that there two many different combination
- note: if G is a DAG, then G has a node with no incoming edge

In a more general case we can visualize the graph in a cascade form



from here we can see that the total number of edges in this graph (trivial) is $\frac{N(N-1)}{2}$. (*)

It get's more clear if we go back. in the case where we had only three variables

- 

(figure 1)

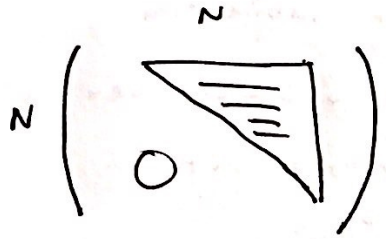
: from 1 -> 3 there two possible way as we can see in (figure 1)
i.e
- two possible choices with present and absent, i.e = 2 x 2
- independently we choose between those two choices, i.e = x2

- Therefore the final result is:

$$2 \times 2 \times 2$$

⇒ Hence we can say that the number of DAG's is at least $\prod_{n=1}^{N} 2^{n-1}$ which will be equal with $2^{N(N-1)/2}$ ⇒ $\prod_{n=1}^{N} 2^{n-1} = 2^{N(N-1)/2}$

(*) Another way to visualize this is in terms of matrixes



← diagonal ⇒ Therefore the number of elements in the upper diagonal is given by $\frac{N^2 - N}{2}$.

- Thus we conclude that for a graph with $N \geq 1$ nodes there at least $\prod_{n=1}^{N} 2^{n-1} = 2^{N(N-1)/2}$

- For the biggest case = where is given when we order the nodes the number of possible DAGs is less that (the highest will include the exact previous highest ... (N-1).N)

$$\frac{N! \left(2^{N(N-1)/2}\right)}{\downarrow}$$

Note: This is not an exact number and is actually this will over-count

Exercise 10.3

Conditioned on each class, the Maximum Likelihood estimates of the probabilities of each attribute (goal, …) correspond to the count weights, as follows:

$P(goal \mid politics) = 2/6$

$P(football \mid politics) = 1/6$

$P(golf \mid politics) = 1/6$

$P(defence \mid politics) = 5/6$

$P(offence \mid politics) = 5/6$

$P(wicket \mid politics) = 1/6$

$P(office \mid politics) = 4/6$

$P(strategy \mid politics) = 5/6$

$P(goal \mid sport) = 5/7$

$P(football \mid sport) = 5/7$

$P(golf \mid sport) = 2/7$

$P(defence \mid sport) = 5/7$

$P(offence \mid sport) = 2/7$

$P(wicket \mid sport) = 1/7$

$P(office \mid sport) = 1/7$

$P(strategy \mid sport) = 1/7$

As for priors, I will assume here that we have been given training data in representative quantities, i.e. the priors are:

$P(politics) = 6/13, \quad P(sport) = 7/13$

The question is asking whether the probability that the document $x = (1, 0, 0, 1, 1, 1, 1, 0)$ is about politics, i.e. what is:

$$P(politics \mid goal, football^c, golf^c, defence, offence, wicket, office, strategy^c)?$$

Applying Bayes' Rule we get:

$$P(politics \mid goal, football^c, golf^c, defence, offence, wicket, office, strategy^c)$$
$$= Z \times P(goal, football^c, golf^c, defence, offence, wicket, office, strategy^c \mid politics)\, P(politics)$$

Where Z can be considered a constant independent of the class. The 'naive' assumption lets us assume that the attributes (goal, defence, etc.) are conditionally independent on the class (politics or sport). So we can factorise the above expression as follows:

$$= Z \times P(goal \mid politics)\, P(football^c \mid politics)\, P(golf^c \mid politics)\, P(defence \mid politics)$$
$$\times P(offence \mid politics)\, P(wicket \mid politics)\, P(office \mid politics)\, P(strategy^c \mid politics)\, P(politics)$$

$= Z \times 2/6 \times 5/6 \times 5/6 \times 5/6 \times 5/6 \times 1/6 \times 4/6 \times 1/6 \times 6/13$

$= Z \times 0.0014$

Similarly, using the same process for the sport class, applying the naive assumption to factorise the expression as follows:

$P(sport \mid goal, football^c, golf^c, defence, offence, wicket, office, strategy^c)$

$= Z \times P(goal \mid sport)\, P(football^c \mid sport)\, P(golf^c \mid sport)\, P(defence \mid sport)$
$\times P(offence \mid sport)\, P(wicket \mid sport)\, P(office \mid sport)\, P(strategy^c \mid sport)\, P(sport)$

where Z has the same value as in the equivalent expression involving politics.

$= Z \times 5/7 \times 2/7 \times 5/7 \times 5/7 \times 2/7 \times 1/7 \times 1/7 \times 7/13$

$= Z \times 0.0003$

Given that politics and sport are the only two class outcomes, we apply the constraint that their probabilities must sum to 1 to get:

$$P(politics \mid goal, football^c, golf^c, defence, offence, wicket, office, strategy^c) = \frac{0.0014}{0.0014 + 0.0003} = 0.8078$$

10.5 Spam filtering

Each email represented by a vector $x = (x_1, \dots, x_D)$, $x_i^n = \begin{cases} 1 \\ 0 \end{cases}$

↑ a particular symbol or word appears in the email

i.e: x_1 = money if $x_1 = 1$ then the word money appears in the mail
x_2 = cash " $x_2 = 1$ " " " cash " "
x_3 = !!!
x_4 = viagra

Training data set: consists of a set of vectors along with the class label, an indicator which takes the value 1 if the mail is spam and the value 0 if else, i.e: $c^n = \begin{cases} 1 & \text{, } n\text{th mail spam} \\ 0 & \text{, } n\text{th mail no spam} \end{cases}$

Naive Bayes Model: $p(c, x) = p(c) \prod_{i=1}^{D} p(x_i | c)$

the training data set consists of a set of pairs (x^n, c^n) $n = 1, \dots, N$ of binary attributed $x_i^n \in \{0, 1\}$, $i = 1..D$ and associated class label c.

let n_0 denote the # of datapoints from class $c = 0 \Rightarrow$ NO SPAM
" n_1 " " " " " c " $= 1 \Rightarrow$ SPAM

Define the probability $p(x_i = 1 | c) = \theta_i^c$ and $p(x_i = 0 | c) = 1 - \theta_i^c$

Therefore; $p(x | c) = \prod_i p(x_i | c) = \prod_i (\theta_i^c)^{x_i} (1 - \theta_i^c)^{1 - x_i}$

based on naive Bayes independence assumption.

and note that if $x_i = \begin{cases} 1, & i \text{ term contributes a factor } \theta_i^c \\ 0 & \text{" " " " } 1 - \theta_i^c \end{cases}$

Thus for iid training data set the likelihood is given by the product $\mathcal{L}(\theta, c; x) = \prod_n p(x^n, c^n)$ and therefore the log-likelihood can be written as:

$$l(\theta, c; x) = \sum_n \log p(x^n, c^n)$$

Hence $l(\theta, p(c); x) = \sum_n \log\left[p(c^n)\prod_{i=1}^{n} p(x_i^n \mid c^n)\right] =$

$= \sum_n \log p(c^n) + \sum_n \log\left[\prod_i p(x_i \mid c^n)\right] =$

$= \sum_n \log p(c^n) + \sum_n \sum_i \log p(x_i^n \mid c^n) =$

$= n_0 \log p(c=0) + n_1 \log p(c=1) + \sum_n \sum_i \log (\theta_i^{c^n})^{x_i^n} (1-\theta_i^{c^n})^{1-x_i^n} =$

$= n_0 \log p(c=0) + n_1 \log p(c=1) + \sum_n \sum_i \left[x_i^n \log \theta_i^{c^n} + (1-x_i^n)\log(1-\theta_i^{c^n})\right] =$

$= n_0 \log p(c=0) + n_1 \log p(c=1) + \sum_n \sum_i \Big[\mathbb{1}[x_i^n=1, c^n=0]\log\theta_i^0 +$

$+ \mathbb{1}[x_i^n=0, c_i^n=1]\log(1-\theta_i^1) + \mathbb{1}[x_i^n=0, c^n=0]\log(1-\theta_i^0) +$

$+ \mathbb{1}[x_i^n=1, c_i^n=1]\log\theta_i^1\Big]$ (1)

Therefore for deriving the expressions for the parameter of this model using the maximum likelihood we simply need to differentiate the likelihood function wrt to θ_i^c and $p(c=c_i^n)$, i.e:

$$\frac{\partial l(\theta_i^c, p(c); x)}{\partial \theta_i^c} \quad \text{and} \quad \frac{\partial l(\theta, p(c=c_i^n); x)}{\partial p(c=c_i^n)}.$$

We have define $p(x_i^n=1 \mid c) = \theta_i^c$ hence differentiating (1) wrt to θ_i^1 the sum over the i's will disapecar since now we have a specific i - all the other will be zero. Hence we have: ← $P(x_i^n=1 \mid c=1) = \theta_i^1$

$$\frac{\partial}{\partial \theta_i^1} l(\theta, p(c); x) = \frac{\partial}{\partial \theta_i^1} \sum_n \Big[\mathbb{1}[x_i^n=0, c^n=1]\log(1-\theta_i^1) + \mathbb{1}[x_i^n=1, c_i^n=1]\log\theta_i^1\Big] =$$

$$= -\frac{1}{1-\theta_i^1}\sum_n \mathbb{1}[x_i^n=0, c^n=1] + \frac{1}{\theta_i^1}\sum_n \mathbb{1}[x_i^n=1, c_i^n=1] = 0$$

$$\Rightarrow -\theta_i^1\left(\sum_n \left[\mathbb{1}[x_i^n=0, c^n=1] + \mathbb{1}[x_i^n=1, c_i^n=1]\right]\right) = -\sum_n \mathbb{1}[x_i^n=1, c_i^n=1]$$

$$\Rightarrow \theta_i^1 = p(x_i=1 \mid c=1) = \frac{\sum \mathbb{1}[x_i^n=1, c_i^n=1]}{\sum_n \left[\mathbb{1}[x_i^n=0, c^n=1] + \mathbb{1}[x_i^n=1, c_i^n=1]\right]} \quad \text{and}$$

therefore for $p(x_i=1 \mid c)$ we have $p(x_i=1 \mid c) = \dfrac{\sum \mathbb{1}[x_i^n=1, c_i^n=c]}{\sum_n \left[\mathbb{1}[x_i^n=0, c^n=c] + \mathbb{1}[x_i^n=1, c^n=c]\right]}$

Hence $p(x_i=1 \mid c=0) = \theta_i^0 = \dfrac{\sum_n \mathbb{1}[x_i^n=1, c_i^n=0]}{\sum_n \left[\mathbb{1}[x_i^n=0, c^n=1] + \mathbb{1}[x_i^n=1, c_i^n=1]\right]}$

let now $p(c=0) = p$ and $p(c=1) = 1-p$

therefore for $\frac{\partial l(\theta, p; x)}{\partial p} = 0$ we have:

$$\frac{n_0}{p} - \frac{n_1}{1-p} = 0 \Rightarrow n_0 - n_0 p - n_1 p = 0 \Rightarrow p = \frac{n_0}{n_0 + n_1}$$

Hence $p(c=0) = \frac{n_0}{n_0+n_1}$ and $p(c=1)) = 1 - p(c=0)) = \frac{n_1}{n_0+n_1}$

2) We classify an input x^{new} as class 1 if

$$p(c=1 \mid x^{new}) > p(c=0 \mid x^{new}).$$

$$\frac{p(c=1, x^{new})}{p(x^{new})} > \frac{p(c=0, x^{new})}{p(x^{new})}$$

$$\frac{p(x^{new} \mid c=1)\, p(c=1)}{p(x^{new})} > \frac{p(x^{new} \mid c=0)\, p(c=0)}{p(x^{new})}$$

$$\log p(x^{new} \mid c=1) + \log p(c=1) - \log p(x^{new}) > \log p(x^{new} \mid c=0) + \log p(c=0) - \log p(x^{new})$$

then using the definition of the classifier the previous inequality is equivalent:

$$\sum_i \log p(x_i^{new} \mid c=1) + \log p(c=1) > \sum_i \log p(x_i^{new} \mid c=0) + \log p(c=0)$$

and then using the binary encoding $x_i = \begin{cases} 1 \\ 0 \end{cases}$ we therefore classify x^{new} as class 1 (bean) if

$$\log p(c=1) + \sum_i \left[x_i^{new} \log \theta_i^1 + (1 - x_i^{new}) \log (1 - \theta_i^1) \right] > \sum_i \left[x_i^{new} \log \theta_i^0 + (1 - x_i^{new}) \log (1 - \theta_i^0) \right] + \log p(c=0)$$

$$\log p(c=1) - \log p(c=0) + \sum_i \left[x_i^{new} \log \theta_i^1 - x_i^{new} \log \theta_i^0 + (1 - x_i^{new}) \log (1 - \theta_i^1) - (1 - x_i^{new}) \log (1 - \theta_i^0) \right] > 0$$

$$\underbrace{\text{const}}_{} + \sum_i w_i x_i^{new} > 0 \quad (1)$$

Therefore for some weights w_i we classify x^{new} as class 1 if (1) holds

↑ and a constant: const

3) x_4 = viagra

- Naive bayes in the case of small count can be overly zealous. If for example the word viagra has no counts for the class $c=1$ (SPAM) then the classifier will say that x cannot be from class $c=1$ (SPAM) This happens because the counts will be 0 and hence the product will be also zero. therefore $P(x_4|c=1)=0$.

- Hence if the word viagra never appears in the spam training data and spam including the word viagra will sender the classifier won't classify it as spam.

- One way to deal with this effect is to use the Bayesian naive Bayes; is simply a Bayesian approach that uses priors on the probabilities $P(x_i^n|c^n)=\theta_i$ to discourage extreme values.

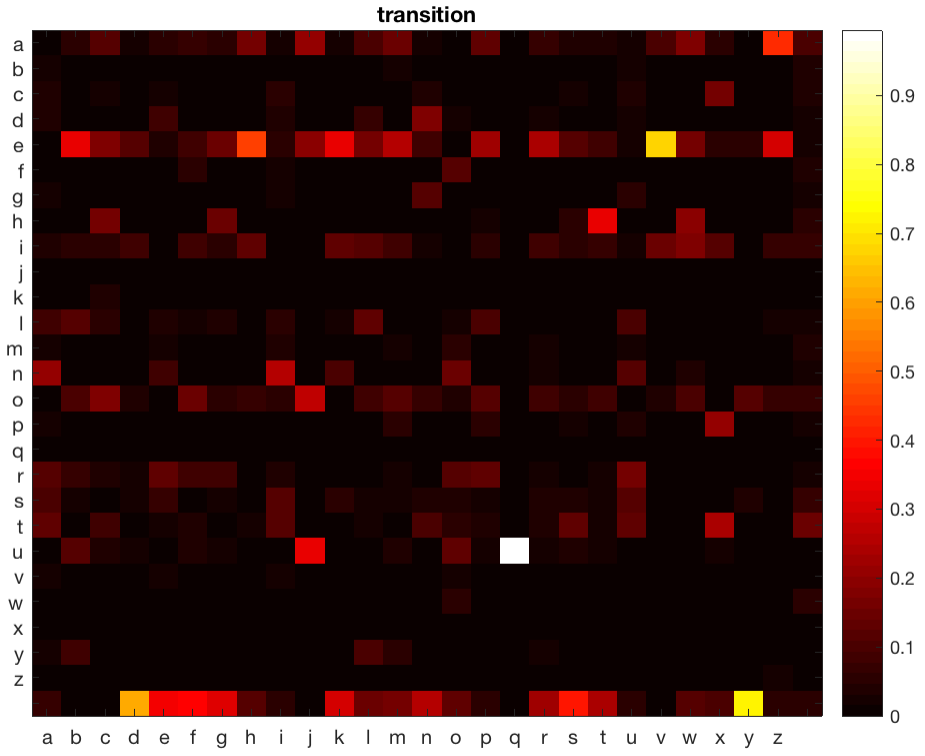
- Finally as we show above, because of this disadvantage of the naive Bayes spam filter namely the fact that does not take into account variable that has no counts for the class $c=1$ (SPAM) in the training data set a spammer might try to fool the naive Bayes spam filter by using words as viagra (that have zero counts) or words that have counts close to zero.

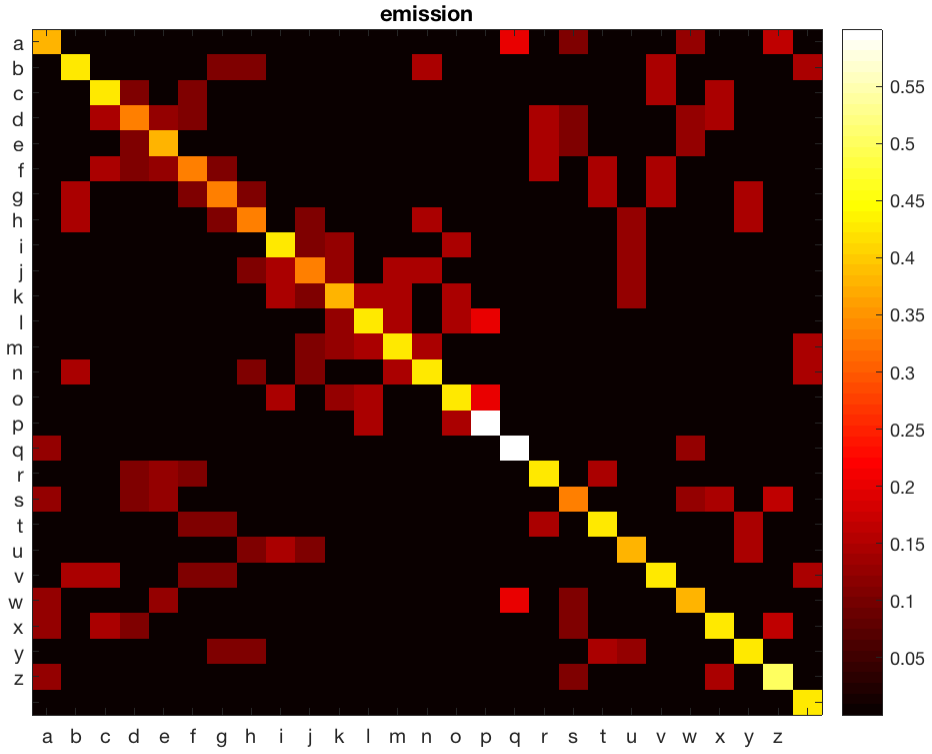
Question 23.4

```
% Question 23.4
% Given the 27 long character string: rgenmonleunosbpnntje vrancg
 typed
% with ?stubby fingers?, what is the most likely correct English
 sentence
% intended?
% In the list of decoded sequences, what value is log p(h1:27|v1:27)
 for this sequence?
clear all;
close all;
%
%The below code is from DEMOHMMBIGRAM demo of HHM for the bigram
 typing scenario
import brml.*
load freq % http://www.data-compression.com/english.shtml
l =
 {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t',
 '};
load typing % get the A transition and B emission matrices
figure(1); imagesc(A); set(gca,'xtick',1:27); set(gca,'xticklabel',l);
 set(gca,'ytick',1:27); set(gca,'yticklabel',l)
colorbar; colormap hot; title('transition')
figure(2); imagesc(B); set(gca,'xtick',1:27); set(gca,'xticklabel',l);
 set(gca,'ytick',1:27); set(gca,'yticklabel',l)
colorbar; colormap hot; title('emission')
ph1=condp(ones(27,1)); % uniform first hidden state distribution

%Modify the imput observed sequence
s = 'rgenmonleunosbpnntje vrancg';  Nmax=1200; % observed sequence
v=double(s)-96; v=replace(v,-64,27); % convert to numbers

% find the most likely hidden sequences by defining a Factor Graph:
T = length(s);
hh=1:T; vv=T+1:2*T;
empot=array([vv(1) hh(1)],B);
prior=array(hh(1),ph1);
pot{1} = multpots([setpot(empot,vv(1),v(1)) prior]);
for t=2:T
    tranpot=array([hh(t) hh(t-1)],A);
    empot=array([vv(t) hh(t)],B);
    pot{t} = multpots([setpot(empot,vv(t),v(t)) tranpot]);
end
FG = FactorGraph(pot);

[maxstate maxval mess]=maxNprodFG(pot,FG,Nmax);
for n=1:Nmax
    maxstatearray(n,:)= horzcat(maxstate(n,1:length(s)).state);
end
strs=char(replace(maxstatearray+96,123,32)); % make strings from the
 decodings
```

```
fid=fopen('brit-a-z.txt','r'); % see http://
www.curlewcommunications.co.uk/wordlist.html for Disclaimer and
 Copyright
w=textscan(fid,'%s'); w=w{1}; % get the words from the dictionary

% discard those decodings that are not in the dictionary:
% (An alternative would be to just compute the probability of each
 word in
% the dictionary to generate the observed sequence.)
for t=1:Nmax
    str = strs(t,:); % current string
    spac = strfind(str,' '); % chop the string into words
    spac = [spac length(str)+1]; % find the spaces
    start=1; val=1;
    for i=1:length(spac) % go through all the words in the string
        wd{i} = str(start:(spac(i)-1));
        start=spac(i)+1;
        if isempty(find(strcmp(wd{i},w))) % check if word is in the
 dictionary
            val=0; break
        end
    end
    if val;
        disp(['after ' num2str(t) ' iterations the intended string is:
 ' str]);
        %add the log computation:
        mylog = log(cell2mat(maxval.table(t)));
        fprintf('\n p(h1:27|v1:27) is: %.3f\n',mylog)
    end
end
```

```
after 659 iterations the intended string is: the monkey is on the
 branch

 p(h1:27|v1:27) is: -94.525
```


transition
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9


emission
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55

Exercise 23.8

(Matlab code that generated these results follows below)

1. The likelihood that ***pnew*** generated sequence S is 8.1781e-13.

2. The likelihood that ***qnew*** generated sequence S is 8.6147e-24.

    a. Yes, it does make sense that sequence S has a much higher likelihood under ***pnew*** than ***qnew***. By inspection the sequence S has many more transitions that are consistent with the cyclic behaviour of the underlying sequence that was used to generate ***pnew*** (e.g. transitions such as A -> C, T -> A).
    b. These transitions have a higher probability in ***pnew*** (0.925) than in ***qnew*** (0.025).
    c. So when you multiply all the transition probabilities together to create the likelihoods, the likelihood under ***pnew*** ends up being much bigger.

3. The posterior results returned in variable phgv are pairs such as {1, 9e-23} and {5e-22, 1}, clearly corresponding to which transition matrix generated each sequence.
    a. Yes, these results make sense. Given the transition probabilities in matrices ***pnew*** and ***qnew*** have values 0.925 and 0.025, the sequences generated by ***pnew*** are highly unlikely to have been generated by ***qnew***, and vice-versa.
    b. Take a sequence generated by ***pnew***. If all the transitions in this sequence have a probability of 0.025 according to ***qnew***, the approximate probability of this sequence according to qnew is 0.025^15 = 9.3132e-25, which is of a similar magnitude to the posterior probabilities calculated by the ***mixMarkov.m*** function.

4. The hidden sequence generated by ***pnew*** is to be preferred. This is because it has a much higher (log) likelihood than the equivalent sequence generated by ***qnew*** (-16.0 vs -27.8).

Exercise 23.8

Table of Contents

Environment setup

```
close all
clear all
import brml.*
```

Global variables

```
A = 1;
C = 2;
G = 3;
T = 4;
```

Define matrices p and pnew

Using the convention that the transition matrix pre-multiplies the vector of probabilities. The given sequence repeats the cycle A,C,G,T,... so a transition matrix that produces this is one that has probabilities of 1 for those transitions, and 0 for all others. i.e.

```
p = [0, 0, 0, 1;
     1, 0, 0, 0;
     0, 1, 0, 0;
     0, 0, 1, 0];

pnew = 0.9*p + 0.1*ones(4)/4;
```

Define matrices q and qnew

Using the same convention as used for p, but for repeating cycle T,G,C,A,...

```
 q = [0, 1, 0, 0;
```

```
      0, 0, 1, 0;
      0, 0, 0, 1;
      1, 0, 0, 0];

qnew = 0.9*q + 0.1*ones(4)/4;
```

Question Part 1

What is the probability that the Markov chain pnew generated the sequence S given by S = (A,A,G,T,A,C,T,T,A,C,C,T,A,C,G,C)

```
S = [A,A,G,T,A,C,T,T,A,C,C,T,A,C,G,C];

% Calculate the probability that pnew generates this sequence.  Do
% this by using the (first order) Markov property and considering
% each pair of consecutive variables in S.

% For length 16, no need to work in log space.

% The first A has (prior) probability 1/4, given in question.

prob = 1/4; %init with prior probability of A

for i = 1:(length(S)-1)

    % element i,j of transition matrix pnew represents the transition
    % probability from state j to state i.
    prob = prob * pnew(S(i+1),S(i));  % transition S(i) -> S(i+1)
end

lik_pnew = prob;

disp(['Likelihood that pnew generated sequence S is: ',
 num2str(lik_pnew)]);
```

Likelihood that pnew generated sequence S is: 8.1781e-13

Question Part 2

Same as Part 1 but for qnew.

```
% The first A has (prior) probability 1/4, given in question.

% Log space found to make no difference to 4th decimal place
% so continuing to work in linear space.

prob = 1/4; %init with prior probability of A

for i = 1:(length(S)-1)

    % element i,j of transition matrix pnew represents the transition
    % probability from state j to state i.
```

```
    prob = prob * qnew(S(i+1),S(i));  % transition S(i) -> S(i+1)
end

lik_qnew = prob;

disp(['Likelihood that qnew generated sequence S is: ',
 num2str(lik_qnew)]);
```

Likelihood that qnew generated sequence S is: 8.6147e-24

Question Part 3

"Using the function randgen.m, generate 100 sequences of length 16 from the Markov chain defined by pnew. Similarly, generate 100 sequences each of length 16 from the Markov chain defined by qnew. Concatenate all these sequences into a cell array v so that v{1} contains the first sequence and v{200} the last sequence.

```
prior = [0.25, 0.25, 0.25, 0.25];

seq = zeros(16,1); % init
v = cell(1,200); % init

for i = 1:100

    % Generate a random sequence using transition matrix pnew

    % First use the prior probabilities to generate the first
    % element of the sequence:
    seq(1) = randgen(prior);

    % Then work through the sequence generating each value using
    % the appropriate column of the transition matrix, i.e.
    % conditioning on the previous sequence value.

    for j = 2:16
       seq(j) = randgen(pnew(:,seq(j-1)));
    end

    % and store this sequence in cell array V
    v{i} = seq;

end


for i = 101:200

    % Generate a random sequence using transition matrix qnew

    % First use the prior probabilities to generate the first
    % element of the sequence:
    seq(1) = randgen(prior);

    % Then work through the sequence generating each value using
```

```
    % the appropriate column of the transition matrix, i.e.
    % conditioning on the previous sequence value.

    for j = 2:16
       seq(j) = randgen(qnew(:,seq(j-1)));
    end

    % and store this sequence in cell array V
    v{i} = seq;

end
```

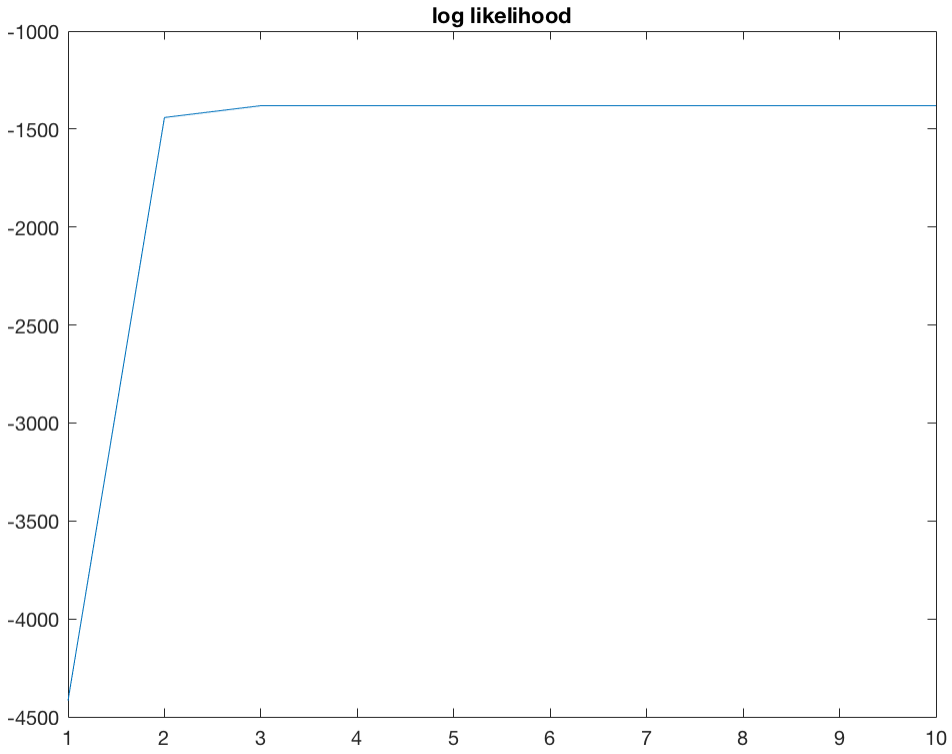
Learn the maximum likelihood parameters

"Use MixMarkov.m to learn the Maximum Likelihood parameters that generated these sequences. Assume that there are H = 2 kinds of Markov chain.

```
data_num_visible_states = 4;
num_mixture_components = 2;
opts.plotprogress = 1; % to display log likelihood
opts.maxit = 10; % the number of iterations of the EM algorithm
[ph,pv1gh,pvgvh,loglikelihood,phgv] = ...
    mixMarkov(v,data_num_visible_states,num_mixture_components,opts);

% The result returned in phgv indicates the posterior
% probability of sequence assignment.

% Display results
format shortEng
format compact   % so that XXe-24 is not displayed as 0.0000

disp(['Sample of posterior probabilities of which hidden model'...
    ' the sequences belong to']);

disp(['phgv{1} = ', num2str(phgv{1}(1)), ' , ', ...
    num2str(phgv{1}(2))]);

disp(['phgv{2} = ', num2str(phgv{2}(1)), ' , ', ...
    num2str(phgv{2}(2))]);

disp(['phgv{101} = ', num2str(phgv{101}(1)), ' , ', ...
    num2str(phgv{101}(2))]);

disp(['phgv{102} = ', num2str(phgv{102}(1)), ' , ', ...
    num2str(phgv{102}(2))]);

format % return number display formatting to normal
```

```
Sample of posterior probabilities of which hidden model the sequences
 belong to
phgv{1} = 2.6055e-18 , 1
phgv{2} = 2.1659e-19 , 1
phgv{101} = 1 , 2.9803e-18
```

```
phgv{102} = 1 , 6.6749e-20
```



Question Part 4

```
% Create emissions distribution matrix according to
% the question:
emission_dist(1:4,1:4) = 0.1;
emission_dist = emission_dist + eye(4) * 0.6;

% Adapting demoHMMinferenceSimple.m
H = 4;  % number of Hidden states
V = 4;  % number of Visible states
T = 16; % length of the time-series

% setup the HMM

% transition distribution p(h(t)|h(t-1))
phghm = condp(pnew);

% emission distribution p(v(t)|h(t))
pvgh = condp(emission_dist);

% initial p(h)
ph1 = condp(ones(H,1));

% Perform Inference tasks:
```

```
[alpha,loglik]=HMMforward(S,phghm,ph1,pvgh); % forward

% Calculate most likely joint state
[viterbimaxstate_pnew, logprob_pnew] = HMMviterbi(S,phghm,ph1,pvgh)

% Repeate calculation for qnew transition matrix
phghm = condp(qnew);
[viterbimaxstate_qnew, logprob_qnew] = HMMviterbi(S,phghm,ph1,pvgh)
```

```
viterbimaxstate_pnew =

  Columns 1 through 13

     1     2     3     4     1     2     3     4     1     2     3
 4     1

  Columns 14 through 16

     2     3     4


logprob_pnew =

  -16.0462


viterbimaxstate_qnew =

  Columns 1 through 13

     1     4     3     2     1     4     3     2     1     4     3
 2     1

  Columns 14 through 16

     4     3     2


logprob_qnew =

  -25.7757
```

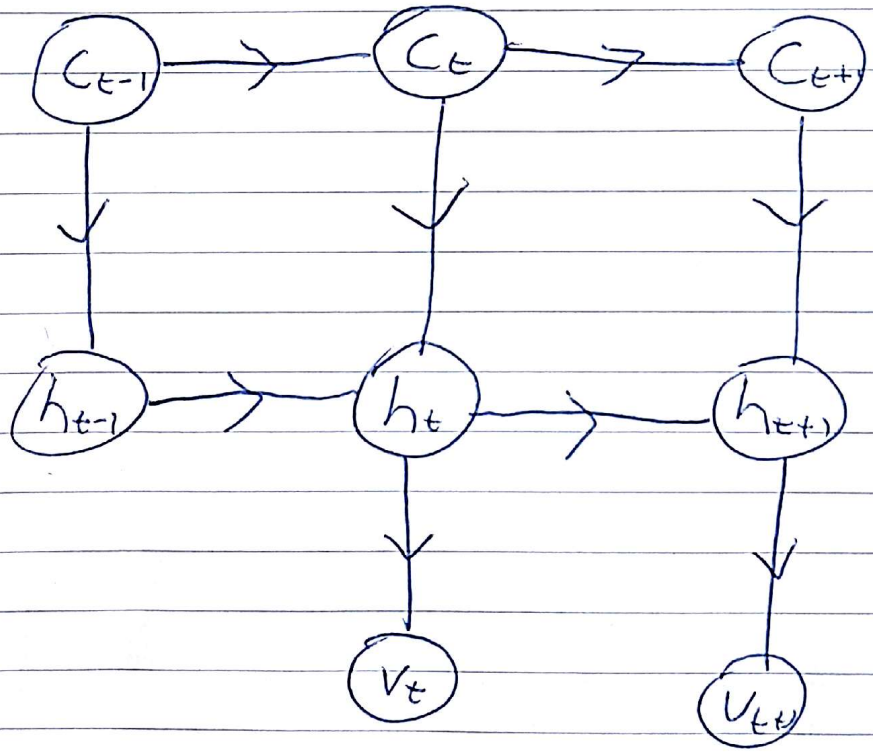
Published with MATLAB® R2016b

Exercise 23.15

23.4.1 states $p(C_t|C_{t-1}) = \begin{cases} \delta(C_t, C_{t-1}-1) & C_{t-1} > 1 \\ p_{dur}(C_t) & C_{t-1} = 1 \end{cases}$

23.4.2 states $p(h_t|h_{t-1}, C_t) = \begin{cases} \delta(h_t, h_{t-1}) & C_t > 1 \\ p_{tran}(h_t|h_{t-1}) & C_t = 1 \end{cases}$

The directed graph representing these assumptions is



$\alpha(h_t, C_t) = p(h_t, C_t, V_{1:t}) =$ [illegible]

(by independence assumptions as shown above)

$= p(V_t|h_t) \sum_{h_{t-1}, C_{t-1}} p(h_t|h_{t-1}, C_t) p(C_t|C_{t-1}) \alpha_{t-1}(h_{t-1}, C_{t-1})$

$\alpha_t(h_t, C_t)$

2.

Splitting the sum on the RHS (prev page) into the cases $c_{t-1} > 1$ and $c_{t-1} = 1$ and dividing both sides by $p(v_t|h_t)$ gives

$$\frac{\alpha(h_t, c_t)}{p(v_t|h_t)} = \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t)\, p(c_t|c_{t-1}=1)\, \alpha(h_{t-1}, c_{t-1}=1)$$

$$+ \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t) \sum_{c_{t-1}=2}^{D_{max}} p(c_t|c_{t-1}) \cdot \alpha_{t-1}(h_{t-1}, c_{t-1})$$

Note that the expression above contains $O(HD_{max})$ terms which agrees with the observation in the book that 'naively' the complexity of inference in the model scales as $O(TH^2D_{max})$

3.

Taking the second term in the RHS of the expression above

$$\sum_{h_{t-1}} p(h_t | h_{t-1}, c_t) \sum_{c_{t-1}=2}^{D_{max}} p(c_t | c_{t-1}) \alpha_{t-1}(h_{t-1}, c_{t-1})$$

$$= \sum_{h_{t-1}} p(h_t | h_{t-1}, c_t) \sum_{c_{t-1}=2}^{D_{max}} \delta(c_t, c_{t-1} - 1) \alpha_{t-1}(h_{t-1}, c_{t-1})$$

$$= \sum_{h_{t-1}} p(h_t | h_{t-1}, c_t) \left[\mathbb{I}(c_t \neq D_{max}) \sum_{c_{t-1}=2}^{D_{max}} \alpha_{t-1}(h_{t-1}, c_{t+1} \right.$$

since otherwise the δ term $= 0$

Rearranging this gives

$$\mathbb{I}(c_t \neq D_{max}) \sum_{h_{t-1}} p(h_t | h_{t-1}, c_t) \alpha_{t-1}(h_{t-1}, c_t + 1)$$

4.

Setting $c_t = 1$ gives

$$\frac{\alpha_t(h,1)}{p(v_t|h_t=h)} = \sum_{h_{t-1}} p(h_t|h_{t-1},1)\, p_{dur}(1)\, \alpha_{t-1}(h_{t-1},1)$$

$$+ \mathbb{I}(D_{max} \neq 1) \sum_{h_t} p_{tran}(h_t|h_{t-1}) \cdot \alpha_{t-1}(h_{t-1}, c_{t+1})$$

by substituting in 23.4.1 and 23.4.2

Taking the terms which do not depend on the h_{t-1} term being summed over out of the summations we get and multiplying both sides by $p(v_t|h_t=h)$ we get

$$\alpha_t(h,1) = p(v_t|h_t=h)\, p_{dur}(1) \sum_{h_{t-1}} p_{tran}(h|h_{t-1})\, \alpha_{t-1}(h_{t-1},1)$$

$$+ \mathbb{I}(D_{max} \neq 1)\, p(v_t|h_t=h) \sum_{h_{t-1}} p_{tran}(h_t|h_{t-1}) \cdot \alpha_{t-1}(h_{t-1},2)$$

as desired for $c=1$ and for $c>1$ we have since

$$p(h_t|h_{t-1}, c) = \delta(h_t, h_{t-1})$$

which equals zero unless $h_t = h_{t-1} = h$

$$\Rightarrow \alpha_t(h,c) = p(v_t/h_t=h)\Big[p_{dur}(c)\,\alpha_{t-1}(h,1) + \mathbb{I}(c \neq D_{max})\,\alpha_{t-1}(h,c+1)\Big]$$

5.
Computing $\alpha_t(h,1)$ is clearly of computational complexity $O(H)$ because of the summation over h_{t-1}

However, for $c>1$ computing $\alpha_t(h,c)$ is clearly $O(1)$

$\Rightarrow$ computing $\alpha_t(h,c)$ for all possible c (all D_{max} of them) is $O(H)$ if $H \gg D_{max}$ and $O(D_{max})$ if $D_{max} \gg H$

$\Rightarrow$ computing α_t messages is clearly $O(H^2)$ if $H \gg D_{max}$ and $O(HD_{max})$ if $D_{max} \gg H$

$\Rightarrow$ The computational complexity of filtered inference is

@/@ $O(TH^2)$ if $H \gg D_{max}$ and $O(THD_{max})$ if $D_{max} \gg H$

6.
An efficient smoothing algorithm is provided by the standard forward-backwards α/β recursions

The β recursion is

$$\beta(h_{t-1}, c_{t-1}) = \sum_{h_t, c_t} p(v_t|h_t, c_t)\, p(h_t|h_{t-1}, c_{t-1})\, p(c_t|c_{t-1}) \cdot \beta(h_t, c_t)$$

For $c_{t-1} = 1$ this equals

$$\sum_{h_t} \sum_{c_t} p(v_t|h_t, c_t)\, p_{tran}(h_t|h_{t-1})\, p_{dur}(c_t)\, \beta(h_t, c_t)$$

For $c_{t-1} > 1$ this equals

$$\sum_{h_t} \sum_{c_t} p(v_t|h_t)\, \delta(h_t, h_{t-1}^{*})\, \delta(c_t, c_{t-1} - 1)\, \beta(h_t, c_t)$$

$$= p(v_t|h_{t-1})\, \beta(h_{t-1}, c_{t-1} - 1)$$

$\Rightarrow$ the $c_{t-1} = 1$ is $O(HD_{max})$ in computational complexity but each of the $c_{t-1} > 1$ calculations is $O(1)$, similarly to the α-recursion case

Exercise 23.16

The standard α-recursion gives

$$\alpha(h_t) = p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1}) \alpha(h_{t-1})$$

Now note that $h_t|h_{t-1} = \delta(h_{t+1}, h_t - 1)$ for $h_t \neq 0$

Therefore instead of $O(L^2)$ entries in the transition matrix there are $O(L)$

$\Rightarrow$ computing the message $\sum_{h_{t-1}} p(h_t|h_{t-1}) \alpha(h_{t-1})$

is $O(L)$ in terms of computational complexity

rather than $O(L^2)$ in the general case

Therefore filtering inference is $O(TL)$
since T messages must be passed

The β-recursion for smoothing is given by

$$\beta(h_{t-1}) = \sum_{h_t} p(v_t|h_t) p(h_t|h_{t-1}) \beta(h_t)$$

Again, because the zero elements in the transition matrix $p(h_t|h_{t-1})$ make the product inside the sum zero, we need only compute $O(L)$ products and again, inference is O for each timestep

and again, inference is $O(TL)$. Note that the arguments refer to the zeros in the transition matrix rather than the emission matrix and thus the argument holds identically in the non-deterministic case

23.16 (2)

For the specific case of the fuzzy string set-up it can be seen that the α-recursion simplifies to

$$\alpha(h_t) = p(v_t|h_t)\,\alpha(h_t+1)$$

for $0 < h_t < L$

$$\alpha(h_t=0) = p(v_t|h_t)\left[\alpha_{t-1}(h_t=1) + \tau\,\alpha_{t-1}(h_t=0)\right]$$

and,

$$\alpha(h_t=L) = p(v_t|h_t)(1-\tau)\,\alpha_{t-1}(h_t=0)$$

Also, the β recursions simplify to

$$\beta(h_{t-1}) = p(v_t|h_{t-1}-1)\,\beta(h_{t-1}-1)$$

for $0 < h_t \leq L$ and

$$\beta(h_{t-1}=0) = p(v_t|h_t=0)\,\tau\,\beta(h_t=0) + p(v_t|h_t=L)(1-\tau)\,\beta(h_t=L)$$

and,

Exercise 23.17

We prove $\alpha(h_t) = \tilde{\alpha}(h_t) \prod_{\tau=1}^{t} z_\tau$ by induction

$\alpha(h_1) = p(v_1, h_1)$ (by definition α)

which equals $p(v_1|h_1)p(h_1)$ (by Bayes)

which equals $\frac{p(v_1|h_1)p(h_1)z_1}{z_1}$ (trivially)

which equals $\tilde{\alpha}(h_1)z_1$ (by def. $\tilde{\alpha}(h_1)$)

Assume that $\alpha(h_t) = \tilde{\alpha}(h_t) \prod_{\tau=1}^{t} z_t$

Then

$$\alpha(h_{t+1}) = p(v_{t+1}|h_{t+1}) \sum_{h_t} p(h_{t+1}|h_t)\alpha(h_t)$$

(by def. α recursion)

which equals $p(v_{t+1}|h_{t+1}) \sum_{h_t} p(h_{t+1}|h_t)\tilde{\alpha}(h_t) \prod_{t=1}^{t} z_t$

(by induction assumption)

$$= \left(\prod_{t=1}^{t} z_t\right) p(v_{t+1}|h_{t+1}) \sum_{h_t} p(h_{t+1}|h_t)\tilde{\alpha}(h_t) \quad (*)$$

(*) equals $\left(\prod_{\tau=1}^{t+1} z_\tau\right) \frac{1}{z_{t+1}} p(v_{t+1}|h_{t+1}) \sum_{h_t} p(h_{t+1}|h_t) \tilde{\alpha}(h_t)$

$$= \tilde{\alpha}(h_{t+1}) \prod_{\tau=1}^{t+1} z_\tau$$

Thus the result follows by induction

Since $\alpha(h_t) = \tilde{\alpha}(h_t) \prod_{\tau=1}^{t} z_\tau$

$$\sum_{h_t} \alpha(h_t) = \prod_{\tau=1}^{t} z_\tau \sum_{h_t} \tilde{\alpha}(h_t) \qquad (*)$$

~~Since $\alpha(h_t) = p(h_t, v_{1:t})$~~

the

Since $\alpha(h_t) = p(h_t, v_{1:t})$ the LHS of (*) equals $p(v_{1:T})$ and since $\tilde{\alpha}(h_t) = p(h_t|v_{1:T})$, $\sum_{h_t} \tilde{\alpha}(h_t) = 1$

$$\Rightarrow p(v_{1:T}) = \prod_{\tau=1}^{t} z_\tau$$

Taking the log of both sides gives the desired result