Question 1

Q1 (Exactly same as tutorial)

15.2 In this exercise, we examine what happens to the probabilities in the umbrella world in the limit of long time sequences.

a. Suppose we observe an unending sequence of days on which the umbrella appears. Show that, as the days go by, the probability of rain on the current day increases monotonically toward a fixed point. Calculate this fixed point.

b. Now consider *forecasting* further and further into the future, given just the first two umbrella observations. First, compute the probability $P(r_{2+k}|u_1, u_2)$ for $k = 1 \ldots 20$ and plot the results. You should see that the probability converges towards a fixed point. Prove that the exact value of this fixed point is 0.5.

Tutor's Note: Question is missing the Umbrella model and Transition Model given in the book/lecture slides, which is needed to solve it.

R_t	$P(U_t)$
t	0.9
f	0.2

- From transition model:

$$\mathbf{T} = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix}$$

R_{t-1}	$P(R_t)$
t	0.7
f	0.3

$\mathbf{T}[1,1]$=$P(R_t$=$true|\ R_{t-1}$=$true)$ $\mathbf{T}[1,2]$=$P(R_t$=$true|\ R_{t-1}$=$false)$

$\mathbf{T}[2,1]$=$P(R_t$=$false|\ R_{t-1}$=$true)$ $\mathbf{T}[2,2]$=$P(R_t$=$false|\ R_{t-1}$=$false)$

The actual answer:

a. For all t, we have the filtering formula

$$\mathbf{P}(R_t|u_{1:t}) = \alpha \mathbf{P}(u_t|R_t) \sum_{R_{t-1}} \mathbf{P}(R_t|R_{t-1}) P(R_{t-1}|u_{1:t-1}) .$$

At the fixed point, we additionally expect that $\mathbf{P}(R_t|u_{1:t}) = \mathbf{P}(R_{t-1}|u_{1:t-1})$. Let the fixed-point probabilities be $\langle \rho,\ 1-\rho \rangle$. This provides us with a system of equations:

$$\begin{aligned} \langle \rho,\ 1-\rho \rangle &= \alpha \langle 0.9,\ 0.2 \rangle \langle 0.7,\ 0.3 \rangle \rho + \langle 0.3,\ 0.7 \rangle (1-\rho) \\ &= \alpha \langle 0.9,\ 0.2 \rangle (\langle 0.4\rho,\ -0.4\rho \rangle + \langle 0.3,\ 0.7 \rangle) \\ &= \frac{1}{0.9(0.4\rho + 0.3) + 0.2(-0.4\rho + 0.7)} \langle 0.9,\ 0.2 \rangle (\langle 0.4\rho,\ -0.4\rho \rangle + \langle 0.3,\ 0.7 \rangle) \end{aligned}$$

Solving this system, we find that $\rho \approx 0.8933$.

Here p is probability it rained yesterday/today/tomorrow/forevermore, and 1-p is probability it did not. The key is that we solve “if there is a fixed point, where will it be?” and so assume p(rain yesterday) = p(rain today). If the solution is impossible (<0, > 1) then we can assume it never converges.

Alpha becomes the big 1/everything because it exists to normalise the probability to add to 1.

0.9(0.4p+0.3)/(0.28p + 0.41) = p

0.36p + 0.27 = 0.28p^2 + 0.41p
0.28p^2 + 0.05p – 0.27 = 0

If you quad formula it, 0.896746 or -1.07532

Cannot be negative as probability

Thus p = 0.896746 (I end up a few decimal places off the textbook answer…. I suspect it is just difference in our calculations and where numbers were rounded to during the quad formula intermediate steps).

Alternatively can do the “bottom half” and solve for p with 1-p on one side.

0.2(-0.4p+0.7)/(0.28p + 0.41) = 1-p

This leads to the same answer for p

Part B

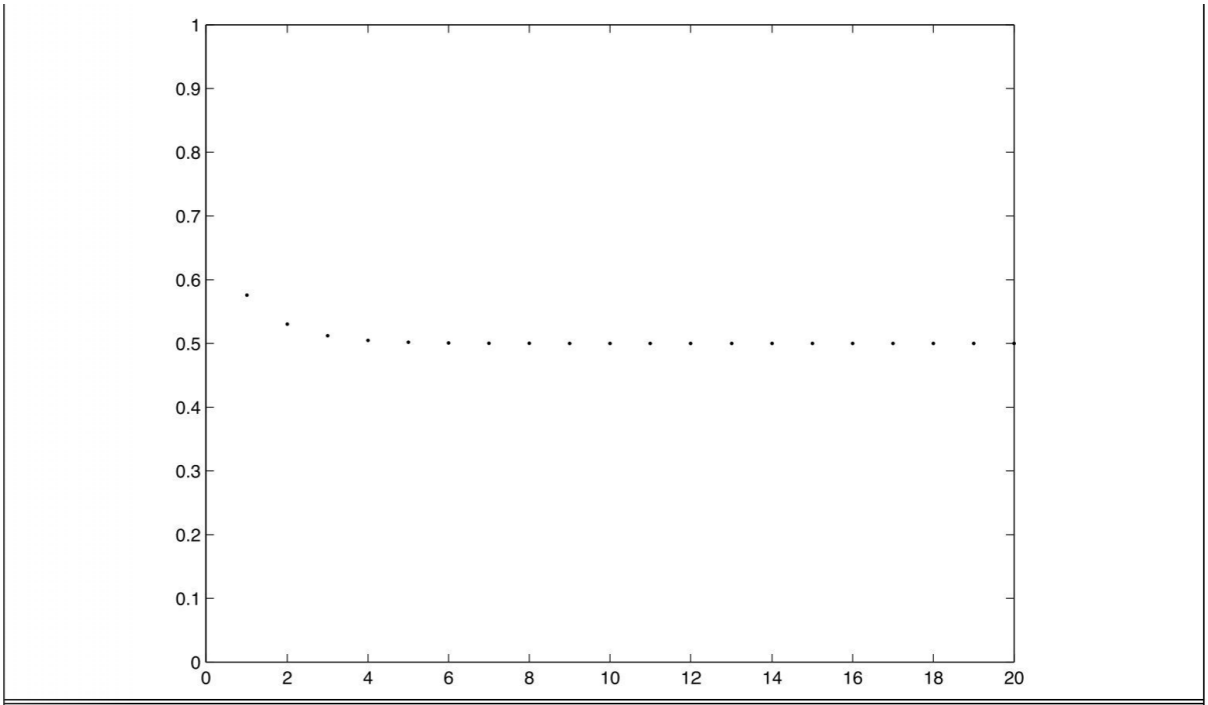


Figure S15.1 A graph of the probability of rain as a function of time, forecast into the future.

Proof:

Transition model says probability of same result as yesterday is 0.7 both ways so

P(Rained today) = 0.7 (P(Rained yesterday)) + 0.3(P(Didn't Rain))

P(Rained today) = P(Rained yesterday) on convergence, and P(didn't rain) = 1-P(Rained) so

P(Rained) = 0.4P(Rained) + 0.3

0.3 = 0.6P(Rained)

0.3/0.6 = P(Rained) = 0.5

Initial evidence is a red herring – it has no effect on convergence point, only how long it takes to reach there.

Question 2

15.3 This exercise develops a space-efficient variant of the forward–backward algorithm described in Figure 15.4 (page 576). We wish to compute $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ for $k = 1, \ldots, t$. This will be done with a divide-and-conquer approach.

a. Suppose, for simplicity, that t is odd, and let the halfway point be $h = (t+1)/2$. Show that $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed for $k = 1, \ldots, h$ given just the initial forward message $\mathbf{f}_{1:0}$, the backward message $\mathbf{b}_{h+1:t}$, and the evidence $\mathbf{e}_{1:h}$.

b. Show a similar result for the second half of the sequence.

c. Given the results of (a) and (b), a recursive divide-and-conquer algorithm can be constructed by first running forward along the sequence and then backward from the end, storing just the required messages at the middle and the ends. Then the algorithm is called on each half. Write out the algorithm in detail.

d. Compute the time and space complexity of the algorithm as a function of t, the length of the sequence. How does this change if we divide the input into more than two pieces?

a. The chapter shows that $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed as

$$\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) = \alpha\,\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \alpha\,\mathbf{f}_{1:k}\mathbf{b}_{k+1:t}$$

The forward recursion (Equation 15.3) shows that $\mathbf{f}_{1:k}$ can be computed from $\mathbf{f}_{1:k-1}$ and $\mathbf{e}_k$, which can in turn be computed from $\mathbf{f}_{1:k-2}$ and $\mathbf{e}_{k-1}$, and so on down to $\mathbf{f}_{1:0}$ and $\mathbf{e}_1$. Hence, $\mathbf{f}_{1:k}$ can be computed from $\mathbf{f}_{1:0}$ and $\mathbf{e}_{1:k}$. The backward recursion (Equation 15.7) shows that $\mathbf{b}_{k+1:t}$ can be computed from $\mathbf{b}_{k+2:t}$ and $\mathbf{e}_{k+1}$, which in turn can be computed from $\mathbf{b}_{k+3:t}$ and $\mathbf{e}_{k+2}$, and so on up to $\mathbf{b}_{h+1:t}$ and $\mathbf{e}_h$. Hence, $\mathbf{b}_{k+1:t}$ can be computed from $\mathbf{b}_{h+1:t}$ and $\mathbf{e}_{k+1:h}$. Combining these two, we find that $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed from $\mathbf{f}_{1:0}$, $\mathbf{b}_{h+1:t}$, and $\mathbf{e}_{1:h}$.

Essentially, as long as we have a starting forwardprop message, a backprop message from an end point, and all the evidence in between, we can calculate any position midway.

b. The reasoning for the second half is essentially identical: for k between h and t, $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed from $\mathbf{f}_{1:h}$, $\mathbf{b}_{t+1:t}$, and $\mathbf{e}_{h+1:t}$.

c. The algorithm takes 3 arguments: an evidence sequence, an initial forward message, and a final backward message. The forward message is propagated to the halfway point and the backward message is propagated backward. The algorithm then calls itself recursively on the two halves of the evidence sequence with the appropriate forward and backward messages. The base case is a sequence of length 1 or 2.

In simpler words: Store the message going forward and evidence going back which you obtain when you hit the midpoint, and suddenly you can start from the middle instead of start or end for easy speedup in future searches.

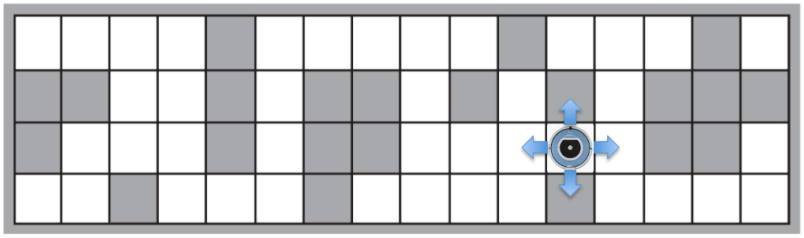
d. At each level of the recursion the algorithm traverses the entire sequence, doing $O(t)$ work. There are $O(\log_2 t)$ levels, so the total time is $O(t \log_2 t)$. The algorithm does a depth-first recursion, so the total space is proportional to the depth of the stack, i.e., $O(\log_2 t)$. With n islands, the recursion depth is $O(\log_n t)$, so the total time is $O(t \log_n t)$ but the space is $O(n \log_n t)$.

Question 3

Question 3 (Based on Question 15.6 of Russell and Norvig, 3ed)
Consider the following map for 2D robot localisation:

Suppose that the robot receives an observation sequence such that, with perfect sensing, there is exactly one possible location it could be in. Is this location necessarily the most probable location under noisy sensing for sufficiently small noise probability ε? Prove your claim or find a counterexample.

Example: robot localisation



- The sensor's error rate is ϵ and error occurs independently in the four directions. This gives the **observation model**

$$P(E_t = e_t \mid X_t = i) = (1-\epsilon)^{4-d_{it}} \epsilon^{d_{it}}$$

where d_{it} is the number of directions that are wrong given location X_t = i and sensor reading $E_t = e_t$

- Example: at the robot's position in the map above, the probability of observing

$$e_t = \textit{NSW} \quad \text{is } (1-\epsilon)^3 \epsilon^1$$

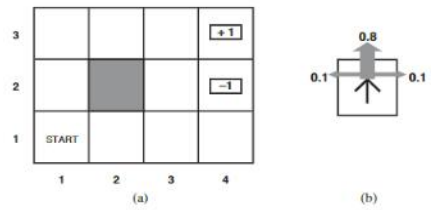
15.6 Let ℓ be the single possible location under deterministic sensing. Certainly, as $\epsilon \to 0$, we expect intuitively that $P(X_t = \ell \mid e_{1:t}) \to 1$. If we assume that all reachable locations are equally likely to be reached under the uniform motion model, then the claim that ℓ is the most likely location under noisy sensing follows immediately: any other location must entail at least one sensor discrepancy—and hence a probability penalty factor of ϵ—on every path reaching it in $t-1$ steps, otherwise it would be logically possible in the deterministic setting. The assumption is incorrect, however: if the neighborhood graph has outdegree k, the probability of reaching any two locations could differ by a factor of $O(k^t)$. If we set ϵ smaller than this, the claim still holds. But for any fixed ϵ, there are neighborhood graphs and observation sequences such that the claim may be false for sufficiently large t. Essentially, if $t-1$ steps of random movement are much more likely to reach m than ℓ—e.g., if ℓ is at the end of a long tunnel of length exactly $t-1$—then that can outweigh the cost of a sensor error or two. Notice that this argument requires an environment of unbounded size; for any bounded environment, we can bound the reachability ratio and set ϵ accordingly.

In human language: For any given error rate E, there is a sequence of measurements long enough that the chance of errors stacks up to the point that the single possible location assuming perfect sensing is not the most likely location. If you have a set limit on the size of the sequence however, then there will be error rates small enough that this does not occur. For example, if the error rate is known to be 1 in 100, then after a billion times it's more likely to have had some errors than to have had none.

Put another way: If we ignore the given map, and imagine one which is a million long. And our sensors say we have moved right a million times in a row, then the *only possible position* with assuming correct sensors is the very right-most square. However it is clearly more likely that a few of those measurements were incorrect, and the robot is actually somewhere between the start and end positions.

Question 4

Question 4 (Based on Question 17.1 of Russell and Norvig, 3ed)



(a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and −1, respectively, and all other states have a reward of −0.04.

For the 4×3 world shown above, calculate which squares can be reached from (1,1) by the action sequence [Up,Up,Right,Right,Right] and with what probabilities. Explain how this computation is related to the prediction task for a hidden Markov model.

17.1 The best way to calculate this is NOT by thinking of all ways to get to any given square and how likely all those ways are, but to compute the occupancy probabilities at each time step. These are as follows:

		Up	*Up*	*Right*	*Right*	*Right*
(1,1)	1	.1	.02	.026	.0284	.02462
(1,2)		.8	.24	.258	.2178	.18054
(1,3)			.64	.088	.0346	.02524
(2,1)		.1	.09	.034	.0276	.02824
(2,3)				.512	.1728	.06224
(3,1)			.01	.073	.0346	.02627
(3,2)				.001	.0073	.04443
(3,3)					.4097	.17994
(4,1)				.008	.0656	.08672
(4,2)					.0016	.01400
(4,3)						.32776

Projection in an HMM involves multiplying the vector of occupancy probabilities by the transition matrix. Here, the only difference is that there is a different transition matrix for each action.

Question 5

17.8 Consider the 3×3 world shown in Figure 17.14(a). The transition model is the same as in the 4×3 Figure 17.1: 80% of the time the agent goes in the direction it selects; the rest of the time it moves at right angles to the intended direction.

Implement value iteration for this world for each value of r below. Use discounted rewards with a discount factor of 0.99. Show the policy obtained in each case. Explain intuitively why the value of r leads to each policy.

a. $r = 100$

b. $r = -3$

c. $r = 0$

d. $r = +3$

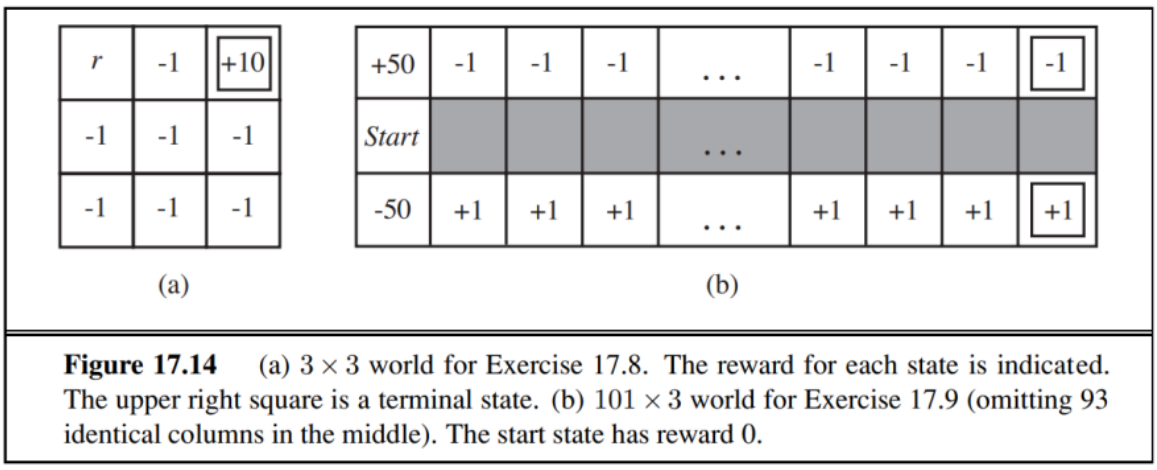


Figure 17.14 (a) 3×3 world for Exercise 17.8. The reward for each state is indicated. The upper right square is a terminal state. (b) 101×3 world for Exercise 17.9 (omitting 93 identical columns in the middle). The start state has reward 0.

17.8

a. $r = 100$.

u	l	.
u	l	d
u	l	l

b. $r = -3$.

r	r	.
r	r	u
r	r	u

Here, the agent again tries to reach the goal as fast as possible while attempting to avoid the square $(1, 3)$, but the penalty for square $(1, 3)$ is not so great that the agent will try to actively avoid it at all costs. Thus, the agent will choose to move Right in square $(1, 2)$ in order to try to get closer to the goal even if it occasionally will result in a transition to square $(1, 3)$.

c. $r = 0$.

r	r	.
u	u	u
u	u	u

Here, the agent again tries to reach the goal as fast as possible, but will try to do so via a path that includes square $(1, 3)$ if possible. This results from the fact that square $(1, 3)$ does not incur the reward of -1 in all other non-goal states, so it reaching the goal via a path through that square can potentially have slightly greater reward than another path of equal length that does not pass through $(1, 3)$.

d. $r = 3$.

u	l	.
u	l	d
u	l	l

This is because the reward is positive and not an end state, so with a discount of 0.99 per iteration, you only need to hug the top left corner a bunch and your score will be above 10. In theory right-most column can be anything as impossible to reach, but good to specify it as down/left in case the robot were to somehow find itself there I suppose. When you try to go left even errors only result up or down, thus it is physically impossible to reach end state with this policy – as intended, for it never wants to reach the end state.

Quick Proof starting from i=2 for the discount factor (the fastest you can top left corner is 2 moves)

0.99^2+0.99^3+0.99^4+0.99^5+0.99^6+0.99^7+0.99^8+0.99^9+0.99^10+0.99^11+0.99^12 = 10.258

Clearly, the reward quickly surpasses that of the end state.

Optional bonus question if time left in tutorial: R=-100

r	r	.
d	r	u
r	r	u

Here, the agent tries to reach the goal quickly, subject to attempting to avoid the square $(1,3)$ as much as possible. Note that the agent will choose to move Down in square $(1,2)$ in order to actively avoid the possibility of "accidentally" moving into the square $(1,3)$ if it tried to move Right instead, since the penalty for moving into square $(1, 3)$ is so great.