PGMs session 3 Representation: Markov Random Fields

1 Preparation: Watch the Koller videos on Markov network fundamentals

Watch the following videos in the Coursera course Probabilistic Graphical Models 1: Representation, Week 3:

- *Markov Network Fundamentals: Pairwise Markov Networks (10:59)*
- *Markov Network Fundamentals: General Gibbs Distribution (15:52)*
- *Markov Network Fundamentals: Conditional Random Fields (22:22)*
- *Independencies in Markov Networks and Bayesian Networks: Independencies in Markov Networks (4:48)*
- *Independencies in Markov Networks and Bayesian Networks: I-maps and Perfect Maps (20:59)*
- *Local Structure in Markov Networks: Log-Linear Models (22:08)*
- *Local Structure in Markov Networks: Shared Features In Log-Linear Models (8:28)*

Comments on the videos:

- In an MRF, the edges (between nodes) become undirected and factors/potentials are no longer limited to being a probability distributions – it can be any non-negative function.
- Flow of influence is simpler than in a Bayes net[1] (BN) since there are no edge directions to take into account. The nodes surrounding a specific node is its Markov blanket. A node is independent of nodes outside its Markov blanket given that the Markov blanket is observed.
- Conditional random fields (CRFs, *Markov Network Fundamentals: Conditional Random Fields (22:22)*) avoid modelling the distribution of the observations explicitly; instead, we model only how other variables depend on them. Barber does not discuss them in Chapter 4, but they are very useful and do form part of later work.
- Log-linear models (*Local Structure in Markov Networks: Log-Linear Models (22:08)* and *Local Structure in Markov Networks: Shared Features In Log-Linear Models (8:28)*) are also not discussed in Barber Chapter 4. Once again they are very useful and do form part of later work in Barber.

It is common to transform BNs into MRFs. After the BN is moralised, we can simply replace the directed edges with undirected ones. In the process we may lose some of the independencies present in the BN – specifically the unconditional independence of the parents of colliders are destroyed by the moralisation (!). To transform this to a factor graph (FG), we need to identify one or more factors with each clique (a fully connected subset of nodes).

2 Preparation: Read Barber Chapter 4 – Markov random fields (MRFs).

- Figure 4.9 provides an example where we can't replace an MRF with a BN without destroying some conditional independencies.

[1] To emphasise that BNs are applicable to both the Bayesian and frequentist approaches to probability theory, I prefer this name over the *Bayesian* net often found in literature.

- For the first time in Barber we meet the I-map concept, but as we have seen from the Koller videos, it is also applicable to BNs.
- Chain graphs (Section 4.3) are graphical models containing both directed and undirected edges. The resultant model is more powerful than either BNs and MRFs (and of course contains them as special cases). This concept is not discussed in the Koller videos (although it is in her book). We will not focus on them in this module – the interested reader is referred to Koller's book for more detail (Chain graphs has its own moralisation procedure, and flow of influence can be ascertained via a process related to d-separation, now called 'c-separation').
- Barber explicitly discusses factor graphs, a finer grained model than MRFs. In the error-correction coding community this is known as a Tanner graph – see MacKay's book for more detail. In video *Message Passing in Cluster Graphs: Properties of Cluster Graphs (15:00)* (later, in Probabilistic Graphical Models 2: Inference, Week 2) Koller mentions them by another name: *Bethe* cluster graphs. Although they are very popular, they are a special/constrained case of a cluster graph and are in general not as powerful as a full cluster graph.

3 Practical: The Hamming (7,4) error correction code

3.1 Background

The Hamming (7,4) error-correction code is applied to digital information transmitted over a noisy communications channel. This simple code extends a 4-bit input sequence with a further 3 parity check bits to result in a 7-bit sequence to transmit. This provides redundancy that allows automatic correction of any *one* wrongly received bit. It works as follows:

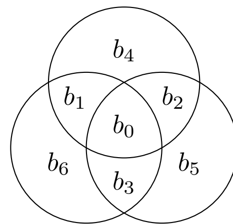


Figure 1: Creating parity checks in the Hamming (7,4) error correcting code. Bits $b_0, \ldots, b_3$ are the original four information bits to be transmitted. Bits b_4, b_5 and b_6 are chosen to create even parity in each of the three circles they occur in.

Transmitter side: Figure 1 describes the coding of the original data to be transmitted. b_0, b_1, b_2 and b_3 are the original 4 information bits to be transmitted, while b_4, b_5 and b_6 are so-called parity check bits. These are chosen so that there will be an even number of 1's in each of the three circles – so-called 'even' parity.

Receiver side: The decoder will receive seven bits, $r_0, \ldots, r_6$. Let us consider the three corresponding circles from Figure 1, but now with $r_0, \ldots, r_6$:

- If all three circles have even parity, the most likely conclusion is that $r_0, \ldots, r_6$ were decoded (recreated) correctly. The alternative would be that there is an even number of errors in each of the three circles, which common sense should indicate as a much less likely outcome. (You will soon be able to calculate these probabilities.)
- If exactly one of these circles fails its parity check, the most likely conclusion is that the bit unique to that circle – i.e., the parity bit – is the one that is in error.
- If exactly two of these circles fail their parity checks, the most likely conclusion is that the single bit that is common to both these circles, but not also common to the third circle, is the one that is in error.
- If all three of them fail their parity checks, the most likely conclusion is that the bit common to them all – i.e. r_0 – is in error.

- Note that if there are more than one error, the decoding will give an incorrect answer. This code can only correct one error in the seven transmitted bits (of which only four are the original information bits we wanted to transmit).

This is enough information to be able to implement such a coder/decoder pair. A few simple rules on the transmit side can generate the check bits for any arbitrary $b_0, \ldots, b_3$. On the receive side, one can calculate the three parities (one for each circle) and then use the logic above to figure out which bit (if any) is in error. However, we will now tackle this via PGMs. Instead of explicitly coding these particular rules, we are going to capture their logic in probability distributions (you can think of this as declarative coding) and then let the PGM figure out what goes for what.

3.2 A PGM to calculate the check bits b_4, b_5 and b_6 given input bits $b_0, \ldots, b_3$.

(a) (On paper) Your first task is to convert Figure 1 into a BN and make a drawing of it. Try to do this on your own – a primary skill you want to develop is the art of transforming logical constraints into a corresponding probability distribution. You can confirm your answer by looking at Figure 2(a) (further down).

(b) (On paper) Directly from this BN, write down an expression for the joint distribution $p(b_0, \ldots, b_6)$. Specifically note how the joint distribution, representing the whole coding subsystem, factorises into a number of smaller distributions where each of them describes a very specific "local" aspect (or piece of knowledge) pertaining to the system. Isn't this a beautiful way of determining the full probabilistic description just about directly from our logical understanding of the situation?
(You should get: $p(b_0, \ldots, b_6) = p(b_4|b_0, b_1, b_2)p(b_5|b_0, b_2, b_3)p(b_6|b_0, b_1, b_3)p(b_0)p(b_1)p(b_2)p(b_3)$.)

Note the equivalence of two viewpoints on the task – initially we described the procedure from a logical viewpoint. Now we have compiled a set of probability distributions that captures that same logic.

(c) (On paper) Now you need to flesh out the actual probability tables for all those conditional probability distributions that you found via the BN. For instance consider $p(b_4|b_0, b_1, b_2)$: For this you need to think carefully about which combinations of b_0, b_1, b_2 and b_4 have even parity and assign a probability of 1.0 to those cases. All the uneven parity cases get a probability of zero. You should end up with half of the 16 cases having a probability of one, and the other half with a probability of zero.

The distributions for the other parity check factors are similar to this, you only need to substitute to the new sets of random variables.

(d) (On paper) From this, use the moralisation procedure to transform this BN into an Markov random field (MRF). Your MRF should look something like Figure 2(b) (further down).

Notice that the conditional distributions of the BN morphs into "cliques" in the MRF. They are somewhat more difficult to spot now. You should be able to easily identify the three parity bit cliques. However, there is a fourth maximal clique in this diagram, see if you can spot that – it will surface again later in this course.

(e) (On paper) In the next assignment we will introduce the concept of message passing or belief propagation. In doing this we will almost exclusively focus on two other PGM representations: the FG and the CG.

In preparation for the next assignment, sketch the FG version of the above graphs. The FG makes the relationship between the factors (probability distributions here) and the random variables they operate on, explicit. To sketch this you list your random variables (each shown inside a circle), and then connect them to the factors they appear in (each shown inside a box). Figure 2(c) shows this.

(f) (In code) Use the `emdw DiscreteTable` class to implement the three factors – one for each parity check circle.

(g) (In code) Multiply these three[2] distributions to get the distribution jointly describing $b_0, \ldots, b_6$, using the `absorb` operator in `emdw`). Use the `normalize` operator to ensure that the result still sums to one.

(h) (In code) Choose some arbitrary combination for the input bits, such as $b_0 = 1, b_1 = 0, b_2 = 1, b_3 = 0$. Use these values as observation or evidence values; do this with `observeAndReduce` in `emdw`.

[2]From the BN, you might have noticed that there also are four other factors describing for $b_0, \ldots, b_3$ their prior probability of being either a one or zero. In general we need to include them. However, if these prior probabilities are "flat" – i.e., equally likely for both cases – they wont affect anything. If you still feel uneasy about leaving them out, feel free to include them. In that case they will also have to be included in the product to determine the joint distribution.

(i) (In code) The above product should result in a single/unique combination of $b_4, \ldots, b_6$ with a non-zero probability (unity after normalisation). You can confirm this by simply printing this (reduced) distribution to the screen. This provide the appropriate values to use for the check bits given the original four information bits. Congratulations, you have built the coding stage for the Hamming(7,4) code.

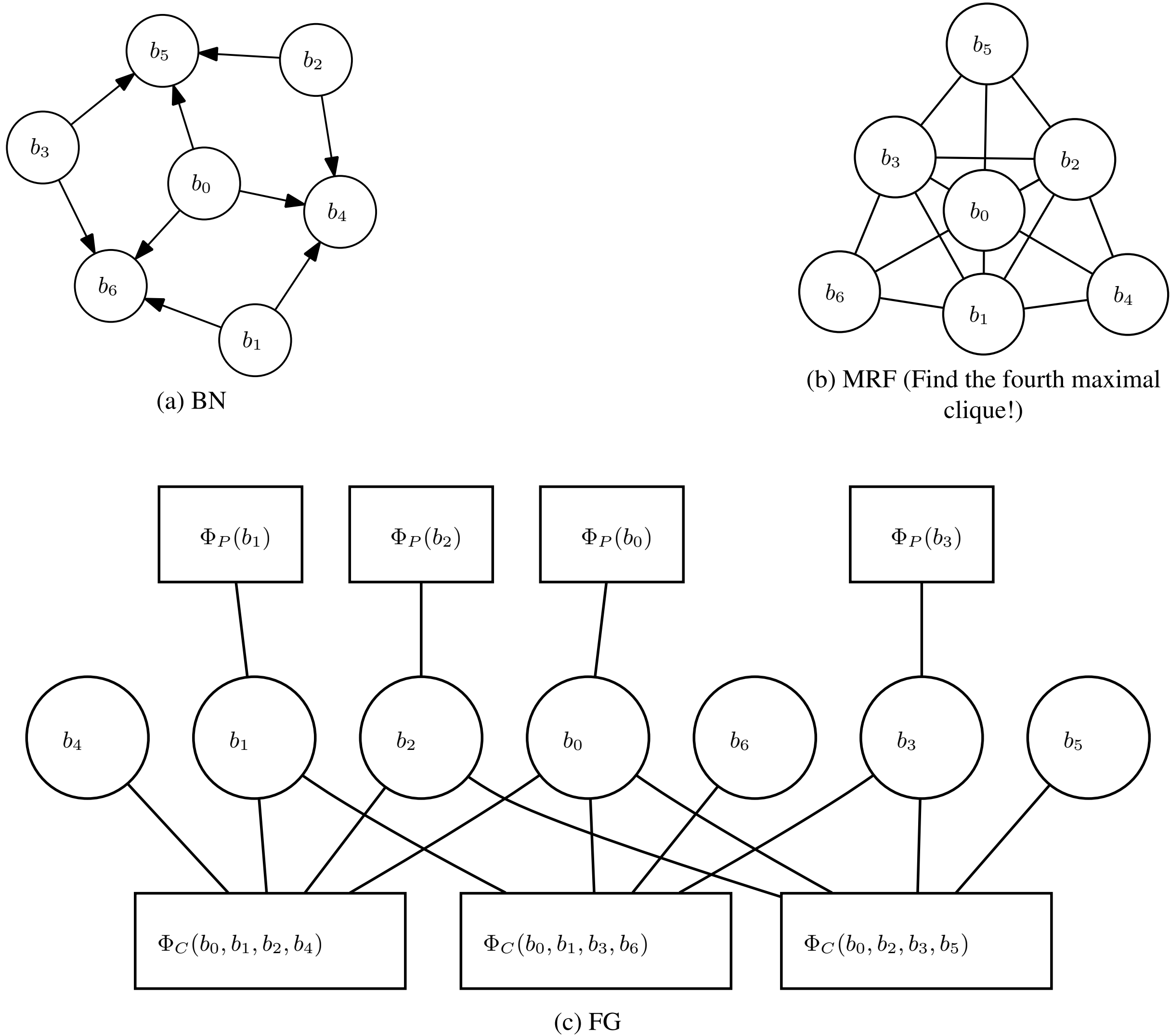


Figure 2: Various PGM representations for the Hamming (7,4) coding stage.

3.3 A PGM to decode the received $r_0, \ldots, r_6$ bits: binary symmetric channel (BSC) case.

On the receiving side, the demodulator will receive degraded versions of the transmitted waveforms and apply some decision circuitry to directly translate it into a received binary sequence $r_0, \ldots, r_6$[3]. The error correction logic is not involved in determining this initial received sequence – each received r_i depends directly and solely on what the corresponding transmitted b_i was. Our purpose now is to take a look at these received r_i bits, take into account the legitimate combinations implied by the check-bit coding, and from all of this determine what the most likely transmitted sequence $b_0, \ldots, b_6$ is. Lets approach this systematically:

(a) (On paper) Our first objective is to determine the full BN that describes the receiver side – it therefore should

[3]If this does not make sense to you, just assume that the received message will be a sequence of bits, $r_0, \ldots, r_6$, which will be the same as the sequence of transmitted bits, $b_0, \ldots, b_6$, but some bits may have "flipped" in the process.

describe the joint distribution for *both* the b_i's as well as the r_i's. Before reading further, first give it a try right now. If you do get stuck, here is some help:

- At the receiver side our knowledge about the parity behaviour between the various b_i's remains unchanged. That part of the network therefore should also remain unchanged, but with the (important) difference that we now of course do not know what the values of the b_i's are (they remain unobserved or latent).
- Each r_i causally depends on its corresponding b_i. We represent this new knowledge by adding an arrow from each b_i to its corresponding r_i.

  Ponder: We know from probability theory that we should not confuse correlation with causality. In principle, we should also have been able to somehow work with $p(b_i|r_i)$. Why did we specifically choose the causal direction?

Easy isn't it? We simply told the system of the applicable facts. Your BN should look something like Figure 3(a) (further down).

(b) (On paper) Directly from this BN, write down an expression for the joint distribution $p(b_0, \ldots, b_6, r_0, \ldots, r_6)$. You should see that the bits that describe the transmitted check bits are exactly as they were on the transmitted side. This makes sense, our knowledge about that hasn't changed. However, there now appears seven $p(r_i|b_i)$ terms. Choose an error probability P_e giving the chances for r_i to differ from the transmitted b_i. Low values implies a good channel, high values a bad channel.

(c) (On paper) You already have the probability tables for the parity checks. Now compile a table that describes the relationship between r_i and b_i. Initially you can set $P_e = 0.01$.

(d) (On paper) Drawing the MRF for the receiving side should now also be quite easy. Seeing that each r_i only has a single parent, no further moralisation is required. It should look something like Figure 3(b) (further down). If you wish to you can also similarly create an FG version.

(e) (In code) Use the `DiscreteTable` class and to your existing parity check factors, add seven more for the $p(r_i|b_i)$'s.

(f) (In code) Simulate observed values for these r_i's by setting them to the same values as their corresponding b_i's (which of course the receiver is ignorant about), and then flip the value of one r_i to create a single error in those seven received bits. Reduce the various $p(r_i|b_i)$'s with these observed values.

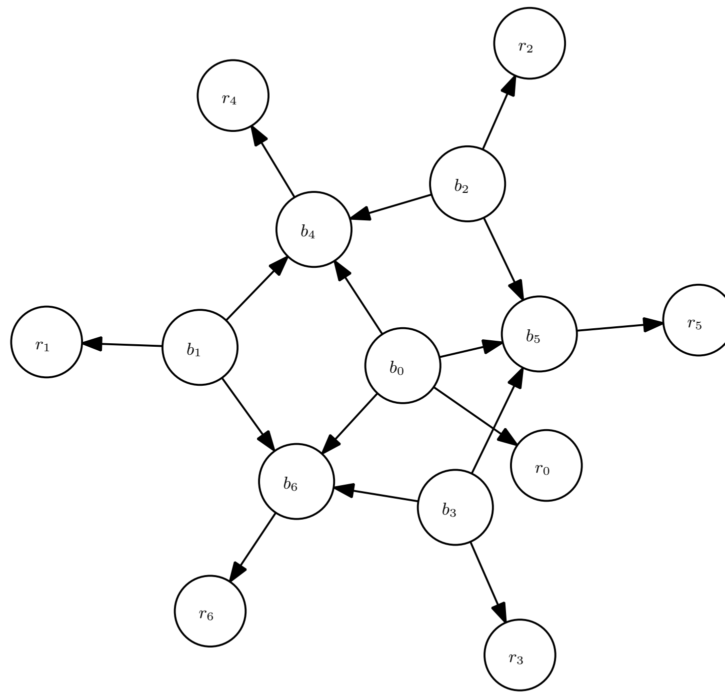
Ponder: Notice that we are doing the observing on the left side of the conditioning bar (i.e., the | symbol). This is quite handy – in your previous probability theory education you went through a bit of Bayes rule gymnastics to achieve this. Another thing you might notice: we now reduced these factors fairly early on, while earlier (with the coding side of things) we did it only after we multiplied the factors. Both are valid (here we wanted to keep the tables small).

(g) (In code) Now multiply all your factors together and normalise. Because this product already takes the observed r_i's into account, you have now calculated $p(b_0, \ldots, b_6|r_0, \ldots, r_6 = \text{the observed values})$. Find the combination with maximum probability, that should be your decoded sequence. You should see that the introduced error was automatically corrected.

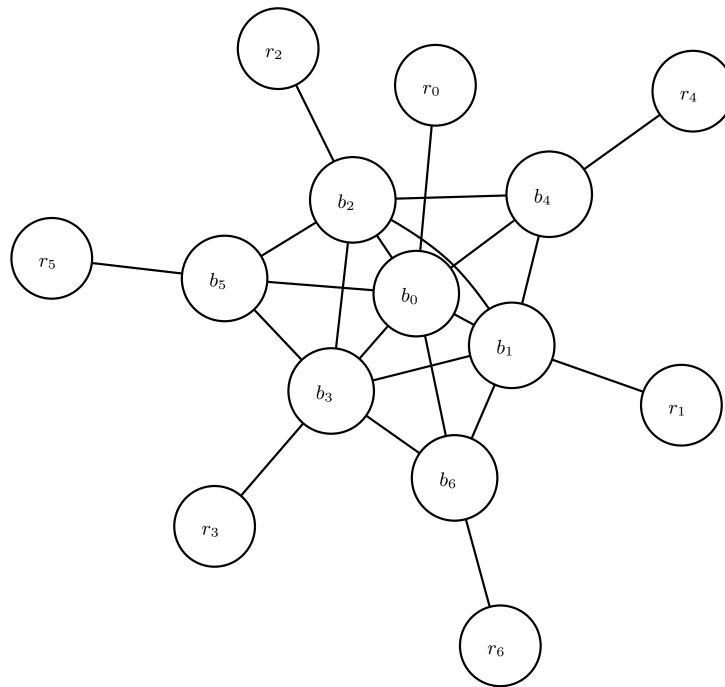
Ponder: In contrast to the transmitter side of things where there was only one non-zero probability, there now are 16 cases with non-zero probability. Why is this so?

(h) (In code) Play a bit with this. For instance, change P_e and observe what happens to the decoding probabilities. Check whether the maximum probability depends on which particular bit went bad. Flip a second one of the received bits and confirm that this system cannot correct two wrongly received bits. And reflect a moment on how much easier this is than doing the manual calculation.

(i) **Ponder:** What happens when P_e is unreasonably (literally so) high such as in $P_e = 0.9$? Does this make sense to you?



(a) BN



(b) MRF

Figure 3: Some PGM representations for the Hamming (7,4) decoding stage.

3.4 Decoding directly on unquantised/continuously valued received $r_0, \ldots, r_6$: additive white Gaussian noise (AWGN) case. A first foray into the world of hybrid (i.e. discrete and continuous) random variables.

In the BSC case (section 3.3 above) we assumed that the demodulator first decides on the value of the binary r_i's, and then we fix possible errors by making use of the parity check logic. We also saw that this procedure can correct up to a maximum of one error.

But certainly this can't be optimal? To make these r_i decisions the system compared received (analogue) waveforms to reference waveforms resulting in real-valued correlations. The final decision was then made by comparing these correlations against thresholds. Shouldn't we also take into account how certain the system was when making each of these decisions?

We can improve on this situation by getting rid of these premature decisions and move the r_i's one level deeper into the system – they now directly become the underlying real-valued / continuous correlation values. Depending on what the transmitted bit was, a continuous probability density function (PDF) can describe the r_i correlation values we might see. The nature and severity of the channel noise will dictate the shape of the PDF. For instance, the relationship might be something like:

$$p(r_i|b_i) = \begin{cases} \sqrt{\frac{1}{2\sigma^2}}e^{-\frac{(r_i-\mu_0)^2}{2\sigma^2}} \text{ with } & b_i = 0 \\ \sqrt{\frac{1}{2\sigma^2}}e^{-\frac{(r_i-\mu_1)^2}{2\sigma^2}} & b_i = 1, \end{cases}$$

i.e. two Gaussian PDFs with means respectively μ_0, μ_1 and common variances σ^2. (For this simulation you can work with $\mu_0 = -1$, $\mu_1 = 1$ and $\sigma^2 = 0.25$).

Now this might seem to be a bit daunting – we have (so far) not at all discussed how to handle continuous valued random variables. And that is indeed a whole topic on its own. However, help is close at hand. Remember, the moment we observe a random variable it reduces the table to whatever we are left with after eliminating everything not compatible with this observed value? Similarly, when we have a hybrid factor (i.e. involving both discrete and continuous random variables), and we observe *all* the continuous ones, we simply instantiate those observed continuous values and we are left with a factor consisting purely of (scaled) discrete probabilities. In Figure 4, with the dashed line indicating the observed value for r_i, the two circled values directly translates to these (scaled) probabilities. You might want to protest that Bayes' rule should be involved here somewhere? Already taken care of elsewhere in the graph. Nifty heh!

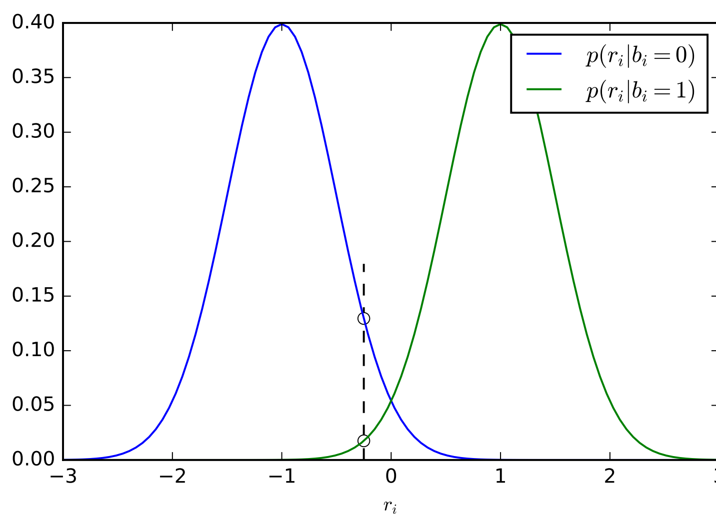


Figure 4: Plot of a hybrid factor. For any given r_i this reduces to a table with two scaled probabilities for b_i.

The important new aspect we introduced here is that, where-as the BSC channel situation above worked with fixed P_e values, in this hybrid case each one of the b_i's has its own individualised table reflecting its particular

uncertainties. This allows this version of the system to, in some cases, be able to correct more than one wrong bit.

Code it: You can test these ideas by using the transmitted sequence of the previous section. For erroneous bits, use a received value $r_i = \pm 0.01$, choosing the sign such that it is *just* to the wrong side of the decision threshold. For correctly received bits, use a received value of $r_i = \pm 1$, with the sign chosen so that it is (clearly) to the correct side of the threshold. As you confirmed before, the above BSC code would not have been able to cope with more than one error. Now compile the individualised $p(r_i|b_i)$ (using the example PDF described above), redo the exercise and compare.

Remark: In this case we explicitly knew the two functions relating the continuous variables to the discrete ones. Quite often we do not. In that case we often use some machine learning technique (logistic regression, neural net etc.) to train such a function for us. This simple technique greatly expands the use of discrete PGMs to also include the cases where we observe real-valued evidence from the environment. However, there also are situations where some continuous-valued random variables remain in the PGM. For this one needs more advanced techniques (not discussed here).

4 Something to ponder: Image denoising

Consider applying what you have already learnt to the image denoising example Barber gives as Example 4.2 on p.66. What exactly makes this task difficult to implement?