PMR Assignment: Modelling the skills of Go Players

29/3/2017

s1686853

1 Question 1

1.a.)
See the hand-drawn factor graph associated with this question.

b.)
Between s_1 and s_2 there are two possible paths in the graph. They are: $s_1 - \phi_1 - s_2$, and $s_1 - \phi_3 - s_3 - \phi_2 - s_2$.

When only r_2 is conditioned upon, both paths are blocked, and therefore s_1 and s_2 are conditionally independent. The first path is blocked because it contains a factor with two incoming edges - ϕ_1 which is not in the conditioning set. The other path is blocked because it also contains a factor with two incoming edges - ϕ_3 which is not in the conditioning set.

When both r_2 and r_3 are known, however, then s_1 and s_2 are not conditionally independent. This is because the second path is no longer blocked as both colliders - ϕ_2 and ϕ_3 are in the conditioning set. Intuitively this makes sense as although we don't know the result of the game between s_1 and s_2 directly, we know how s_1 and s_3 compare and how s_3 and s_2 compare, and this therefore gives us some information about the relative skills of s_1 and s_2.

c.)
See the hand-drawn graphical model associated with this question.

d.)
We can make no conditional independence statements about the skills of the players in this undirected Markov network. This is because the graph consisting of the skills is fully connected. So, there are no variables that can be removed which will cause there to be no path between any two skill variables. By the u-separation criterion, this means that we can make no conditional independence about the skill variables in the graph. The undirected representation of the factor graph has thus lost some information about the independences of the variables making it up.

We can make conditional independence statements about the r variables, or the results of each game, as these are not fully connected. We can say, for instance, that r_1 is conditionally independent of the other variables in the graph given s_1 and s_2. More generally, we can say that the result of any given game is conditionally independent of the other results and skills given the skills of the players who played that game.

In general, the conditional independence relationships represented in the two factor graphs are different, and converting between a directed and an undirected graph does not necessarily maintain all of the information encoded into the original graph.

2 Question 2

a.)

From the factor graph, we know that the probability distribution over s_w, p_w, r, s_b, p_b factorises as follows:

$$P(r, p_w, p_b, s_w, s_b) = \frac{1}{Z}\phi_1(s_w)\phi_2(p_w, s_w)\phi_3(s_b)\phi_4(p_b, s_b)\phi_5(r, p_w, p_b)$$

We also know that:

$$P(r, p_w, p_b, s_w, s_b) = \frac{\sum_r P(r, p_w, p_b, s_w, s_b)}{\sum_r \int_{p_w, p_b} P(r, p_w, p_s, s_w, s_b)}$$

These can be written in terms of the factors defined above. The integrals are distributed through the factors:

$$\frac{\phi_1(s_w)\phi_2(s_b)\phi_3(p_w, s_w)\phi_4(p_b, s_b)\sum_r \phi_5(r, p_w, p_b)}{\phi_1(s_w)\phi_2(s_b)\int_{p_w, p_b} \phi_3(p_w, s_w)\phi_4(p_b, s_b)\sum_r \phi_5(r, p_w, p_b)}$$

Factors ϕ_1, ϕ_2, and ϕ_5 cancel, giving us:

$$\frac{\phi_3(p_w, s_w)\phi_4(p_b, s_b)}{\int_{p_w, p_b} \phi_3(p_w, s_w)\phi_4(p_b, s_b)}$$

We now plug in the values of the factors:

$$= \frac{N(p_w; s_w, 1)N(p_b; s_b, 1)}{\int_{p_w} N(p_w; s_w, 1)\int_{p_b} N(p_b; s_b, 1)}$$

The two integrals in the denominator both integrate to one as they are integrals of a univariate probability distribution in that variable. Therefore we obtain:

$$P(p_w, p_b | s_w, s_b) = N(p_w; s_w, 1)N(p_b; s_b, 1)$$

b.)

$$E[\psi | s_w, s_b] = E[\frac{(p_w - s_w) - (p_b - s_b)}{\sqrt{2}}]$$

By the linearity of expectations and the fact that the expectation of a constant is that constant, we know that:

$$= \frac{1}{\sqrt{2}}(E[(p_w - s_w)] - E[p_b - s_b]) = \frac{1}{\sqrt{2}}(E[p_w] - s_w - E[p_b] - s_b)$$

However:

$$E[p_w] = E[N(p_w; s_w, 1)] = s_w$$

And:

$$E[p_b] = E[N(p_b; s_b, 1)] = s_b$$

So:

$$E[\psi|s_w, s_b] = \frac{1}{\sqrt{2}}((s_w - s_w) - (s_b - s_b)) = 0$$

By a similar argument:

$$E[\theta|s_w, s_b] = \frac{1}{\sqrt{2}}((E[p_w] - s_w) + (E[p_b] - s_b)) = \frac{1}{\sqrt{2}}(s_w - s_w + s_b - s_b)) = 0$$

We now deal with the squared expectations.

First $P(\theta^2|s_w, s_b)$:

$$E[\theta^2|s_w, s_b] = \frac{E[((p_w - s_w) + (p_b - s_b))]^2}{2} = \frac{E[(p_w - s_w)^2] + 2E[(p_w - s_w)(p_b - s_b)] + E[(p_b - s_b)^2]}{2}$$

I deal with the squared terms first:

$$E[(p_w - s_w)^2] = E[p_w^2] - 2E[p_w]s_w + s_w^2$$

$$= E[p_w^2] - s_w^2 = Var(p_w) = 1$$

By symmetry, $E[(p_b - s_b)^2]$ also equals 1
Now for the cross term:

$$2E[(p_w - s_w)(p_b - s_b)] = 2(E[p_w p_b] - s_w E[p_b] - s_b E[p_w] + s_w s_b)$$

$$= 2(E[p_w p_b] - s_w s_b - s_b s_w + s_w s_b) = 2(E[p_w p_b] - s_w s_b)$$

This can be rewritten as:

$$2(E[p_w p_b] - E[p_w]E[p_b]) = 2 * Cov(p_w, p_b) = 0$$

Which is just the covariance between p_w and p_b which is 0, as we know from the factor graph that p_w and p_b are independent.

So

$$E[\theta^2|s_w, s_b] = \frac{1 + 0 + 1}{2} = 1$$

Now $P(\psi^2|s_w s_b)$:

$$E[\psi^2|s_w, s_b] = \frac{E[((p_w - s_w) - (p_b - s_b))]^2}{2} = \frac{E[(p_w - s_w)^2] - 2E[(p_w - s_w)(p_b - s_b)] + E[(p_b - s_b)^2]}{2}$$

From the previous answer, we know that

$$E[(p_w - s_w)^2] = E[(p_b - s_b)^2] = 1$$

And

$$2E[(p_w - s_w)(p_b - s_b)] = 0$$

So:

$$E[\psi^2|s_w, s_b] = \frac{E[(p_w - s_w)^2] - 2E[(p_w - s_w)(p_b - s_b)] + E[(p_b - s_b)^2]}{2} = \frac{1 - 0 + 1}{2} = 1$$

Finally, we have :

$$E[\theta\psi|s_w, s_b] = \frac{E[((p_w - s_w) - (p_b - s_b))((p_w - s_w) + (p_b - s_b))]}{2}$$

$$= \frac{E[(p_w - s_w)^2] - E[(p_w - s_w)(p_b - s_b)] + E[(p_b - s_b)(p_w - s_w)] - E[(p_b - s_b)^2]}{2}$$

$$= \frac{E[(p_w - s_w)^2] - E[(p_b - s_b)^2]}{2}$$

As we know from previous questions:

$$E[(p_w - s_w)^2] = E[(p_b - s_b)^2] = 1$$

So:

$$E[\theta\psi|s_w s_b] = \frac{E[(p_w - s_w)^2] - E[(p_b - s_b)^2]}{2} = \frac{1 - 1}{2} = 0$$

Therefore, to summarise, we have derived:

$$E[\theta|s_w, s_b] = 0$$

$$E[\psi|s_w, s_b] = 0$$

$$E[\theta^2|s_w, s_b] = 1$$

$$E[\psi^2|s_w, s_b] = 1$$

$$E[\theta\psi|s_w, s_b] = 0$$

From this we can determine that the means of θ and ϕ are 0. The variances of θ and ϕ are 0, and their covariance is 0. From this we know that they form a bivariate Gaussian distribution of the form:

$$P(\theta, \psi|s_w, s_b) = N([\theta, \psi]; [0, 0], [[1, 0], [0, 1]])$$

Or a zero mean bivariate Gaussian with an identity covariance matrix.

c.)

The random variable r is 1 when $p_b > p_w$ and zero when $p_b < p_w$ (we ignore the case of a draw here, as the probability of that happening is zero, as both p_b and p_w are continuous probability distributions, where the density at any point is zero).

Therefore, r can be expressed using an indicator function as follows:

$$r = I[p_b > p_w]$$

Now we need to get p_b and p_w in terms of ψ and θ.

We know that:

$$\psi = \frac{(p_w - s_w) - (p_b - s_b)}{\sqrt{2}}$$

We can then rearrange to get p_b:

$$p_b = p_w - s_w + s_b - \sqrt{2}\psi$$

We also know that:

$$\theta = \frac{(p_w - s_w) + (p_b - s_b)}{\sqrt{2}}$$

So:

$$p_b = \sqrt{2}\theta - p_w - s_w + s_b$$

We then set the two expressions we derived for p_b equal to each-other, and rearrange for p_w:

$$p_w - s_w + s_b - \sqrt{2}\psi = \sqrt{2}\theta - p_w - s_w + s_b$$

$$2p_w = \sqrt{2}\theta + \sqrt{2}\psi$$

Substituting this back into the expression for p_b we derived earlier, we get:

$$p_b = \frac{\sqrt{2}\theta + \sqrt{2}\psi}{2} - s_w + s_b - \sqrt{2}\psi = \frac{\sqrt{2}\theta - \sqrt{2}\psi}{2} - s_w + s_b$$

Remembering that $r = I[p_b > p_w]$. This means that:

$$r = I[\frac{\sqrt{2}\theta}{2} - \frac{\sqrt{2}\psi}{2} - s_w + s_b > \frac{\sqrt{2}\theta}{2} + \frac{\sqrt{2}\psi}{2}]$$

which simplifies to:

$$r = I[s_b - s_w > \sqrt{2}\psi] = I[\frac{s_b - s_w}{\sqrt{2}} > \psi]$$

So:

$$P(r = 1|\psi, \theta, s_w, s_b) = I[\frac{s_b - s_w}{\sqrt{2}} > \psi]$$

d.)

We know that:

$$P(r = 1|s_w, s_b) = \int_{\psi,\theta} P(r = 1, \psi, \theta|s_w, s_b)$$

Which we can write as:

$$\frac{\int_{\psi,\theta} P(r = 1, \psi, \theta, s_w, s_b)}{P(s_w, s_b)}$$

Which we can rewrite using the chain rule as:

$$= \frac{\int_{\psi,\theta} P(r=1|\psi,\theta,s_w,s_b)P(\psi,\theta|s_w,s_b)P(s_w,s_b)}{P(s_w,s_b)}$$

As the $P(s_w, s_b)$ term in the numerator does not contain any ϕs or θs, it can be taken out of the integral and cancelled with the denominator, thus leaving us with:

$$P(r=1|s_w,s_b) = \int_{\psi,\theta} P(r=1|\psi,\theta,s_b,s_w)P(\psi,\theta|s_w,s_b)$$

Both of these probability distributions have been derived in earlier questions.

Because earlier we have proven that θ and ψ are independent of each other, we can split the integral as follows:

$$P(r=1|s_w,s_b) = \int_{\psi} P(r=1|\psi,\theta,s_b,s_w)P(\psi|s_w,s_b) \int_{\theta} P(\theta|s_w,s_b)$$

We now plug in the values we have derived:

$$P(r=1|s_w,s_b) = I[\frac{s_b - s_w}{\sqrt{2}} > \psi]N(\psi;0,1) \int_{\theta} N(\theta;0,1)$$

The integral in θ is that of a univariate Gaussian in θ, so it integrates so one. So we can ignore that term.

Therefore:

$$P(r=1|s_w,s_b) = \int_{\psi} I[\frac{s_b - s_w}{\sqrt{2}} > \psi]N(\psi;0,1)$$

$$= \Phi[\frac{s_b - s_w}{\sqrt{2}}]$$

e.)

As there are only two possible outcomes of a single game (we ignore draws), we know that:

$$P(r=0|s_w,s_b) = 1 - P(r=1|s_w,s_b)$$

As there are only two outcomes, we can model the outcome of a single game as a Bernoulli distribution

$$P(r|s_w,s_b) = (\Phi[\frac{s_b - s_w}{\sqrt{2}}])^r(1 - \Phi[\frac{s_b - s_w}{\sqrt{2}}])^{1-r}$$

Where $r = \{0, 1\}$. If we assume that games are independent of one another, then we can model the distribution over the results of a set of games, as simply the product of the result of single games. Thus we can write:

$$P(\{r^k\}_{k=1}^N|\{s\}_{i=1}^M) = \prod_{k=1}^{N} \{(\Phi[\frac{s_b - s_w}{\sqrt{2}}])^r(1 - \Phi[\frac{s_b - s_w}{\sqrt{2}}])^{1-r}\}$$

f.)

Using Bayes rule, we can rewrite the equation derived above as:

$$P(\{s\}_{i=1}^{M}|\{r^k\}_{k=1}^{N}) = \frac{P(\{r^k\}_{k=1}^{N}|\{s\}_{i=1}^{M})P(\{s\}_{i=1}^{M})}{P(\{r^k\}_{k=1}^{N})}$$

We can drop the denominator and just rewrite the equation as proportional rather than equal as follows:

$$P(\{s\}_{i=1}^{M}|\{r^k\}_{k=1}^{N}) \propto P(\{r^k\}_{k=1}^{N}|\{s\}_{i=1}^{M})P(\{s\}_{i=1}^{M})$$

We now have two terms - a likelihood and a prior. We will deal with the prior first: $P(\{s\}_{i=1}^{M})$.

This term is simply a set of individual skills of the players. Each individual player skill is defined as a univariate Gaussian, so it makes sense to define the prior over all skills as a multivariate Gaussian. From the factor graph at the start we know that the skills of the players are independent of each other. Thus, the covariance matrix of the Gaussian will be diagonal. The skill of each player is a Gaussian with a mean of 0 and a variance of σ^2 so we can define the mean of the multivariate skills Gaussian to consist of a vector of zeros, and the covariance to be a diagonal matrix with all σs on the diagonal. We write this as:

$$P(\{s\}_{i=1}^{M}) = N(s; \mu_0, V_0)$$

Where μ_0 is a vector of 0s M long. And V_0 is a MxM diagonal matrix with $sigma^2$s along the diagonal.

The likelihood term is one that we derived earlier:

$$\prod_{k=1}^{N}\{(\Phi[\frac{s_b - s_w}{\sqrt{2}}])^r(1 - \Phi[\frac{s_b - s_w}{\sqrt{2}}])^{1-r}\}$$

Using the fact that $1 - \Phi(x) = \Phi(-x)$, we can rewrite this term as:

$$\prod_{k=1}^{N}\{\Phi[s_b - s_w]^r\Phi[s_w - s_b)]^{1-r}\}$$

Note that we have absorbed the square-root-of-two denominators into the constant of proportionality.

Now notice that the calculation we do is practically the same regardless of whether the result is a win or a loss, except that we switch the order of subtraction. If the result is 1, which is defined as black winning. The probability of this is white's skill subtracted from black's. If r = 0, when white wins, we subtract black's skill from white's.

We could simplify this procedure by having a list, or vector, of skills,and the operation performed by the function is to simply perform addition. However,

the loser of each match is multiplied by -1, so that the effect is that the skill of the loser is subtracted from that of the winner. This scheme is identical mathematically to the form given above. This new form can be written as:

$$\prod_{k=1}^{N}\{\Phi[s^T x]\}$$

Where s is the list of player skills, and x is a vector of all players. The elements corresponding to players not involved in the game are 0. The element corresponding to the winner is 1, and the element corresponding to the loser is -1. The dot product operation will thus ensure that the effective operation done by the formula for each game is $s_{winner} - s_{loser}$, which is what the original formula implemented.

Using this form we can now rewrite the full equation as:

$$P(\{s\}_{i=1}^{M}|\{r^k\}_{k=1}^{N}) \propto \prod_{k=1}^{N}\{\Phi[s^T x]\}N(s;\mu_0,V_0)$$

3 Question 3

a.)

The green line represents the true factor which is not a Gaussian or of the exponential distribution. This means that the approximate (blue and orange) factors cannot approximate it exactly, as the EP algorithm assumes the approximating distribution is Gaussian. Therefore, the approximate distribution will be the closest projection of the true factor into the space of Gaussian distributions, where "closest" is defined in terms of the KL divergence between the true and approximate factors. For a single factor, this is the blue line, which manages to match the mode of the true factor, while also spreading probability mass relatively widely to cover a lot of the mass of the true distribution. Due to the symmetry of the Gaussian and the asymmetry of the distribution it is approximating, however, it is forced to spread mass over regions where the true factor has relatively little probability mass.
This approximate factor is fitted iteratively to the true factor where each factor is first fitted locally, in the context of all the other approximate factors. At first this approximation may be relatively poor - especially if the factor is one of the first one fitted, but after several iterations this will converge towards a reasonable value. The approximation here actually seems quite successful as the approximate factor has at least managed to moment-match the true factor, and has a reasonable distribution.

The approximate posterior is created by combining the Gaussian approximate factor with the prior, which is also a Gaussian. As the product of two Gaussians is Gaussian, the approximate posterior is also Gaussian, which means it is more tractable to analytic methods than the non-Gaussian true posterior distribution. The Approximate posterior has a slightly smaller mean and a

much smaller variance than the approximate factor. This is likely due to the influence of the prior which, as a prior over the skill has a mean of zero and a variance of σ^2. As the mean and variance of the prior is smaller than the mean and variance of the approximate factor density, the mean and variance of the approximate posterior will shrink.

The approximate posterior also appears to be a fairly good approximation. It has the same rough mean as the true factor, and it's smaller variance means that it does not assign probability mass to regions where the true factor does not, unlike the approximate factor. However, its smaller variance may mean that it is unable to model the true variance of the true factor.

b.)

The posterior marginal skill densities for Alpha-go, Lee Sedol, and Ke Jie for the full covariance and diagonal covariance models are plotted below:

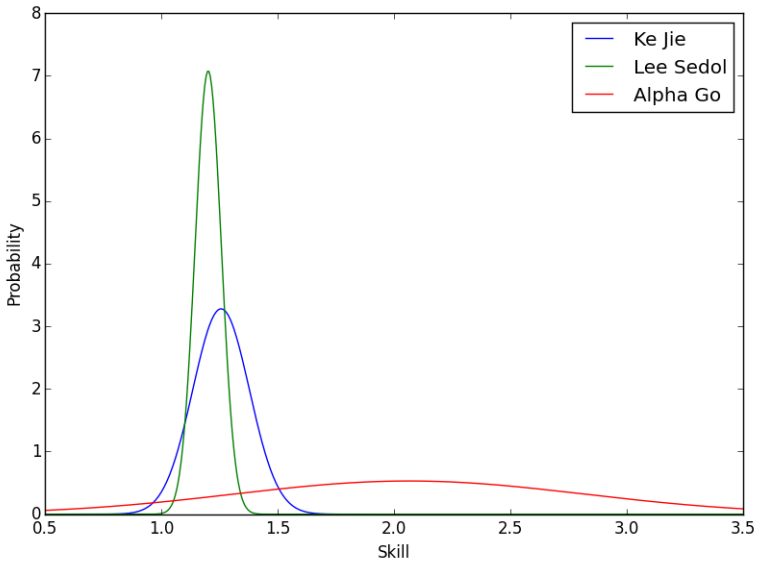

Figure 1: The marginal skill densities for Alpha-Go, Ke Jie and Lee Sedol under the diagonal approximation

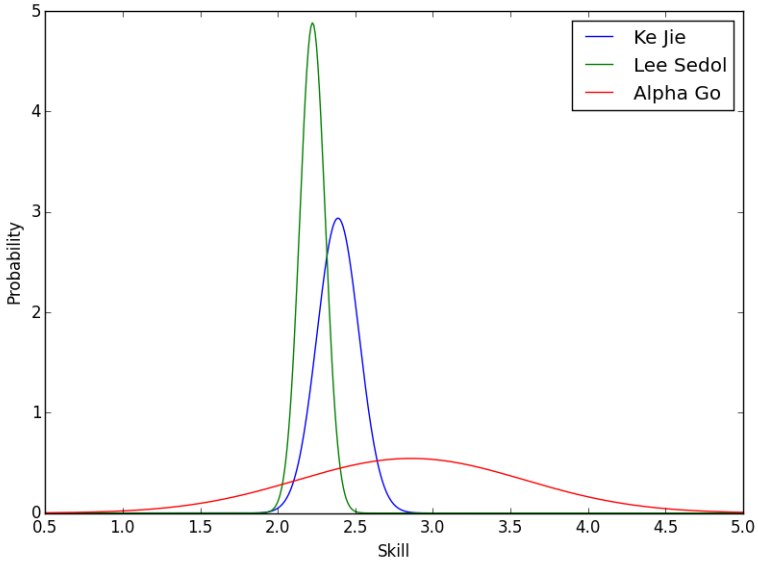

Figure 2: The marginal skill densities for Alpha-Go, Ke Jie and Lee Sedol under the full-covariance approximation

In both plots the variance of the two human players - Ke Jie and Lee Sedol - is relatively small and their Gaussian are obviously peaked around a single value where Ke Jie is shown to have a slightly higher estimated mean skill than Lee Sedol. AlphaGo's estimated skill, by contrast, is a very spread out Gaussian with high variance which is probably due to the fact that it has played relatively few games rather than any large inherent variability in AlphaGo's skill level. The estimated mean skill of AlphaGo is higher than both of the human players by quite a significant margin, although due to the larger variance, there is still lots of overlap in the distributions.

The variance of all the skill estimates for both human players and AlphaGo is reduced in the full-covariance approximation. This is probably because the approximation can use information about the covariance between the skills of players rather than solely relying on the variance of that player, which will lead to less uncertain estimates. Interestingly, in the full-covariance approximation, the means of the human players move closer to AlphaGo's estimated mean than in the diagonal approximation, and the means of all players are higher overall. This could be because the full covariance matrix includes more information about how each player ranks against every other player, rather than simply how much their skill varies in general.

The code that generated these plots is shown below:

```
from __future__ import division
```

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as mlab
import math
from scipy.stats import norm

full_covar = np.load("full_covar.npz")
diag_covar = np.load("diag_covar.npz")


def diag():
        globals().update(diag_covar)
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        x = np.linspace(0.5,3.5,500)
        plt.plot(x, mlab.normpdf(x,approx_mean[ke_jie_id], approx_covar[ke_jie_i
        **0.5), label = "Ke Jie")
        plt.plot(x,mlab.normpdf(x,approx_mean[lee_sedol_id], approx_covar[lee_se
        **0.5), label = "Lee Sedol")
        plt.plot(x,mlab.normpdf(x,approx_mean[alpha_go_id], approx_covar[alpha_g
        **0.5), label = "Alpha Go")
        ax.set_xlabel("Skill")
        ax.set_ylabel("Probability")
        ax.legend()
        fig.tight_layout()
        plt.show()
        return fig
def full():
        globals().update(full_covar)
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        x = np.linspace(0.5,5,500)
        # The marginal distribution of a gaussian is just a gaussian
        plt.plot(x, mlab.normpdf(x,approx_mean[ke_jie_id], approx_covar[ke_jie_i
        [ke_jie_id]**0.5), label = "Ke Jie")
        plt.plot(x,mlab.normpdf(x,approx_mean[lee_sedol_id], approx_covar[lee_se
        [lee_sedol_id]**0.5), label = "Lee Sedol")
        plt.plot(x,mlab.normpdf(x,approx_mean[alpha_go_id], approx_covar[alpha_g
        [alpha_go_id]**0.5), label = "Alpha Go")
        ax.set_xlabel("Skill")
        ax.set_ylabel("Probability")
        ax.legend()
        fig.tight_layout()
        plt.show()
        return fig
```

c.)

Using the equations given, the probability of AlphaGo winning when playing as black in a single game against Ke Jie was calculated for both the full and diagonal approximations.

The probability of AlphaGo winning under the full covariance approximation was: 0.599

The probability of AlphaGo winning under the diagonal covariance approximation was: 0.665

The code to generate these probabilities is shown below:

```
def predictive_probabilities():
        #for the full covariance
        globals().update(full_covar)
        #set up our x* vector:
        x = np.zeros([n_players,1])
        x[alpha_go_id] = 1/math.sqrt(skill_prior_var)
        x[ke_jie_id] = -1/math.sqrt(skill_prior_var)

        print x.T.shape
        print x.shape
        print approx_covar.shape
        numerator = np.dot(approx_mean.T, x)
        denom = np.dot(np.dot(x.T, approx_covar), x) +1
        denom = np.sqrt(denom)
        frac = numerator/denom
        full_prob = norm.cdf(frac)
        print full_prob

        #for the diagonal covariance
        globals().update(diag_covar)
        #set up our x* vector:
        x = np.zeros([n_players,1])
        x[alpha_go_id] = 1/math.sqrt(skill_prior_var)
        x[ke_jie_id] = -1/math.sqrt(skill_prior_var)

        #create the diagonal matrix
        #diag = np.zeros([n_players, n_players])
        diag = np.diag(approx_covar)

        #calculate the probability
        numerator = np.dot(approx_mean.T, x)
        denom = np.dot(np.dot(x.T, diag), x) +1
        denom = np.sqrt(denom)
```

```
frac = numerator/denom
diag_prob = norm.cdf(frac)
print diag_prob
return full_prob, diag_prob
```

d.)

Using the full covariance matrix to approximate factors and posterior probabilities in the network should lead to more accurate approximations as it is able to take into the covariances of the skill of a player with other players when estimating the posterior probability. These covariances, if large, may be able to significantly sway the estimate away from the simple variance of that player's own skill. The diagonal approximation loses all of this information and thus may be expected to produce a worse and less accurate approximation. However, the key advantage of the diagonal approximation is that it is significantly easier to compute. The computational complexity of inverting the full covariance matrix is $O(n^3)$, which may be computationally infeasible when calculating the posteriors of a huge number of Go players. Moreover the full covariance matrix must be stored in memory which, for a large matrix, may also be difficult. Using the diagonal variances only is much easier as inverting the diagonal to calculate probabilities is the same operation as inverting a vector, which can take place in linear time, while the memory demands are the square root of those of storing the full covariance matrix. This means that for situations where there are huge numbers of players, sacrificing some accuracy for computational feasibility may be worthwhile.

It is worth noting that the diagonal approximation does not appear to be particularly bad, despite throwing away a large amount of information. The shapes and arrangements posterior probabilities plotted in 3b are very similar in the diagonal and full covariance approximation. Importantly, the ordering of the means of the Gaussians are maintained, so that a simple ranking of players would not be altered in this case by using the diagonal covariance.

4 Question 4

a.)

Our model is symmetric in that it does not model any disadvantages or advantages accruing from playing black or white, and thus going first or second. Although the *komi* points are added to the game to balance this out, it is possible that they are not entirely sufficient. Thus, adding a way of modelling there being an advantage to starting first or second in the game would add realism and perhaps usefulness to the model.

A simple model is to assume that there is a fixed benefit to going first which is constant across players. To model this we could simply add a small extra factor to the projected skill of black (who goes first). Mathematically this model would

be:

$$P(r = 1|s_b, s_w) \propto \Phi[(s_b + \frac{\alpha}{s_b}) - s_w]$$

Where α is the expected benefit of going first. This could perhaps be estimated from game statistics such as the difference in total wins as black vs as white. The α factor is divided by s_b as a simple measure to normalise α. If this were not done, the effect of going first would have a much greater effect on the projected skill of someone with a small skill compared to someone with a larger skill.

A more sophisticated model might take into account the fact that the benefit from going first may be different between players. Some players might have an aggressive play-style so that going first benefits them more, while others might be more defensive and reactive so that they actually perform better while going second. To model this we would simply estimate a α factor for each player separately over the games they have played, perhaps starting from the population average, and then tuning it as the player plays more games.

b.)

Our current model assumes that the player's skill is a static quantity which is fixed across time. It also assumes that all games, no matter how long ago they occured, are equally valuable for inferring the player's current skill. Both of these assumptions seem questionable. In reality, the player's skill is a dynamic and evolving quantity both in the short and the long run. In the long run, the player's skill hopefully improves as they play more games and learn from them, or temporarily decreases if they do not play for a long time and become "rusty". Moreover, in the short run, players can go on "streaks" where they either perform much better than their baseline skill for a while, or else perform much worse.

One simple adjustment to the model is to discount games that occured a long time ago, so they affect our estimate of the player's skill less. This makes sense as we would naturally expect observations that took place long ago to provide a worse estimate of the current environment than observations that occurred recently. We can write the posterior distribution over all player's skills given all occuring games as:

$$P(\{s\}_{i=1}^{M}|\{r^k\}_{k=1}^{N}) \propto \prod_{k=1}^{N}\{\Phi[\mathbf{s}^T\mathbf{x}]\}N(s;\mu_0,V_0)$$

From this formula, we can write the posterior distribution over a single player's skill given all games as:

$$P(s|\{r^k\}_{k=1}^{M}) \propto \prod_{k=1}^{N}\Phi[\mathbf{s}^T x^s]N(s;m_s,V_s)$$

Where x^s is a vector of zeros whenever the player s is not involved in that game, and is the same as the vector $\mathbf{x}$ when they are not. And $N(s;m_s,V_s)$ is the prior distribution solely of that player's skills.

Here the estimated posterior skill is simply the product of all games the player has been involved in. A game played a long time ago has the same contribution to the estimate as one played recently. To model the discounting of games over time, we simply add a discount factor - $\beta(t)$ - inside the product so that older games affect the estimate less. This can be expressed mathematically as:

$$P(s|\{r^k\}_{k=1}^M, t) \propto \prod_{k=1}^{N} \beta(t)\Phi[\mathbf{s}^T x^s]N(s; m_s, V_s)$$

Where t can be the actual time, for instance the number of days since the game was played, or else could be the number of games this game was played before the most recent one. $\beta(t)$ is a monotonically decreasing function in t. One obvious possibility is to use an exponential, so that:

$$\beta(t) \propto e^{-t}$$

This would have the effect that the effect of games on our estimate are discounted exponentially the longer ago they were. This approach has the effect that the variance in the estimated player skill will be larger, and it will fluctuate to a greater degree depending on current performance. This is because our estimate is effectively based on fewer games than previously, as older games are discounted. Depending on the underlying nature of the game to be modelled, and the use to which the model is put, this may either be positive or negative overall. If the aim of the system is to provide an adaptive ranking for matching players, then more rapid fluctuations of the skill may be beneficial, on the other hand, if the purpose of the system is to create a relatively static leader-board for a sport with well established players, then the greater variance may mean mostly pointless upheaval.

Another factor we might wish our model to represent is that we become increasingly uncertain about the true skill of the player, the longer it has been since they played a game. A simple way to model this increase in uncertainty would be through the prior. Currently, the prior has the form $N(s; \mu_s, V_s)$. Under this model, we make the variance of the prior time-dependent, so that generally, it increases the longer it has been since the player played a game. We denote the time since the player last played a game as τ. The new form of the prior can thus be written as:

$$N(s; \mu_s, V_s(\tau))$$

Where $V_s(\tau)$ is some function that increases with time. It makes sense that we want this function to increase rapidly initially, but then asymptote to some maximum when the time since the player last played becomes very large. A logarithmic model thus seems to be appropriate. This would be of the form:

$$V_s(\tau) = V_s(0) + \gamma log(\tau)$$

Where $V_s(0)$ is some fixed inherent uncertainty in the skill calculation, and the γ is a hyperparameter which can be adjusted to control the rate of increase of the uncertainty. This γ could be estimated from data.

c.)

One limitation of Expectation Propagation is that it explicitly relies on the factorial structure of the distribution which it hopes to model, and can only model it with Gaussian factors. Thus, if we want the approximation derived by EP to be accurate, the model needs to have a probability distribution which can be neatly factorised into Gaussian factors. This is largely the case in our model, however it constrains the possible choice of model more generally. In addition, EP is not guaranteed to converge except in tree-structured graphs, such as our model, and may diverge in more general cases. EP is also an approximate algorithm, and will never necessarily converge to the correct value of the integral, even if it does converge to something.

The reason we are forced to do approximate inference in the first place is that we cannot analytically evaluate the integral:

$$\int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} P[\{s_i\}_{i=1}^{M}, \{r^k\}_{k=1}^{N}] ds_1 ... ds_M$$

Which is the joint probability over all the skills of the players and all the games they play. Nor are we able to calculate analytically integrals like:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P[s_b, s_w | \{r^k\}_{k=1}^{N}] ds_b ds_w$$

Which are necessary in order to make inferences like calculating the chance of a given player winning, which is calculated probabilistically as:

$$P[r = 1 | \{r^k\}_{k=1}^{N}] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P[r = 1 | s_w, s_b] P[s_b, s_w | \{r^k\}_{k=1}^{N}] ds_b ds_w$$

Although the first integral is relatively easy to calculate the second is not, and needs to be approximated. The EP way to approximate this is to first define some tractable family of approximate distributions - Gaussian in the case of EP - and then fit those distributions to the true distribution by minimising some divergence metric such as the KL divergence. Another method that does something similar is Variational Message Passing (VMP). However, there is a simpler method of modelling the distribution - sample from it using a Markov Chain Monte Carlo (MCMC) method.

One MCMC method which is commonly used is Gibbs sampling. In Gibbs sampling we sample from the posterior by sweeping through each variable in the posterior and sampling from the conditional distribution of that variable given all the other variables (which we hold fixed). Initially the samples generated through this approach may be completely wrong, but as multiple sweeps

progress, the distribution of the samples slowly converges towards the true posterior distribution.

One major advantage of the MCMC sampler over EP is that it is guaranteed to converge to the true distribution eventually (in the limit of infinite samples) as long as certain conditions about the Markov chain hold (such as detailed balance) - and in all sampling algorithms it is made certain that the conditions do hold. This is unlike other methods which can only minimise the distance between the true distribution and a family of approximate distributions - which will never be zero unless the true distribution is of that family.

The big flaw in MCMC methods, however, is that although they theoretically converge to a solution with a higher accuracy, they are significantly slower than EP. This will be especially the case as the number of dimensions in the integral increases as the volume, and thus the space in which the MCMC sampler must sample from increases exponentially. As it appears in this case that the EP approximations get very close to the true distributions (as in the graph in 3a of the assignment), the additional accuracy provided by the MCMC sampling methods does not seem to provide a particularly large return compared to the additional time and computational cost it takes to run them.