UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and, Machine Intelligence 2018-2019

Module Coursework Feedback

Module Title: Probabilistic Machine Learning

Module Code: 4F13

Candidate Number: F606F

Coursework Number: 2

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

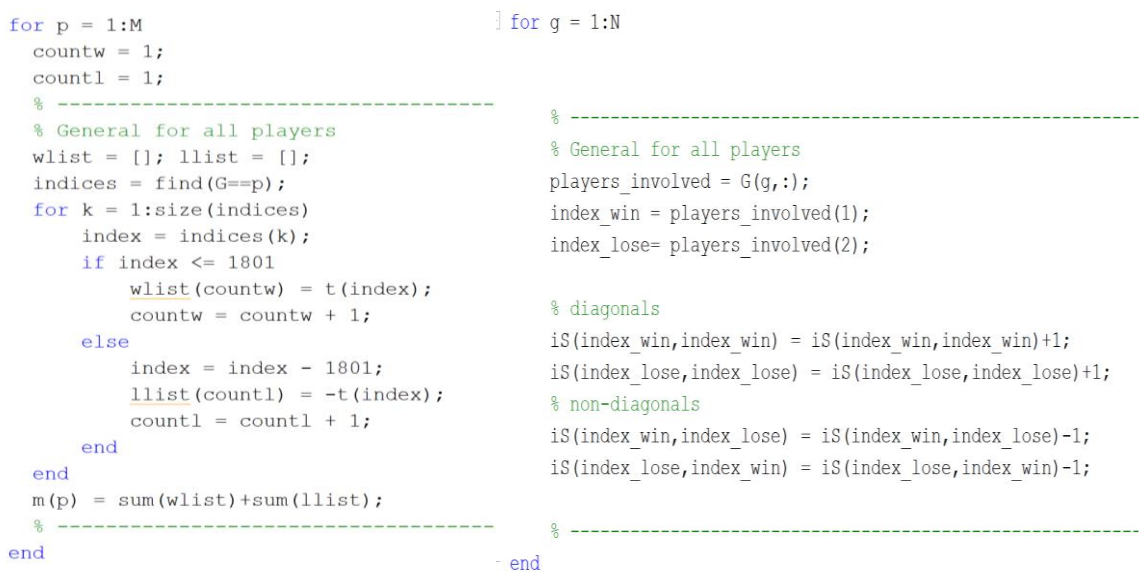
	Distinction			**Pass**			**Fail (C+ - marginal fail)**		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

Probabilistic Machine Learning cw2 – Probabilistic Ranking (960 words)

a.

Mean container and precision matrix are computed in the following loop shown in Figure 1.



```
for p = 1:M
  countw = 1;
  countl = 1;
  % ------------------------------------
  % General for all players
  wlist = []; llist = [];
  indices = find(G==p);
  for k = 1:size(indices)
      index = indices(k);
      if index <= 1801
          wlist(countw) = t(index);
          countw = countw + 1;
      else
          index = index - 1801;
          llist(countl) = -t(index);
          countl = countl + 1;
      end
  end
  m(p) = sum(wlist)+sum(llist);
  % ------------------------------------
end
```

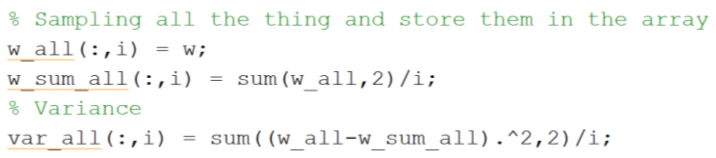
```
for g = 1:N

    % -----------------------------------------------------------
    % General for all players
    players_involved = G(g,:);
    index_win = players_involved(1);
    index_lose= players_involved(2);

    % diagonals
    iS(index_win,index_win) = iS(index_win,index_win)+1;
    iS(index_lose,index_lose) = iS(index_lose,index_lose)+1;
    % non-diagonals
    iS(index_win,index_lose) = iS(index_win,index_lose)-1;
    iS(index_lose,index_win) = iS(index_lose,index_win)-1;

    % -----------------------------------------------------------
end
```

Figure 1: (left) contatiner for the mean of the conditional skill distribution given the t_g samples. (right) container for the sum of the precision matrices distributed by all the games (likelihood terms)



```
% Sampling all the thing and store them in the array
w_all(:,i) = w;
w_sum_all(:,i) = sum(w_all,2)/i;
% Variance
var_all(:,i) = sum((w_all-w_sum_all).^2,2)/i;
```

Figure 2 shows how all the samples from all iterations are stored in a matrix, and how the means and variances are computed for all iterations.

All iterations of skill samples for all players are collected to form a matrix of dimension 107x1100, which the ith column describes the skill samples of all players in ith iteration. The means and variances of all samples of all players are found for all iterations shown in Figure 2.

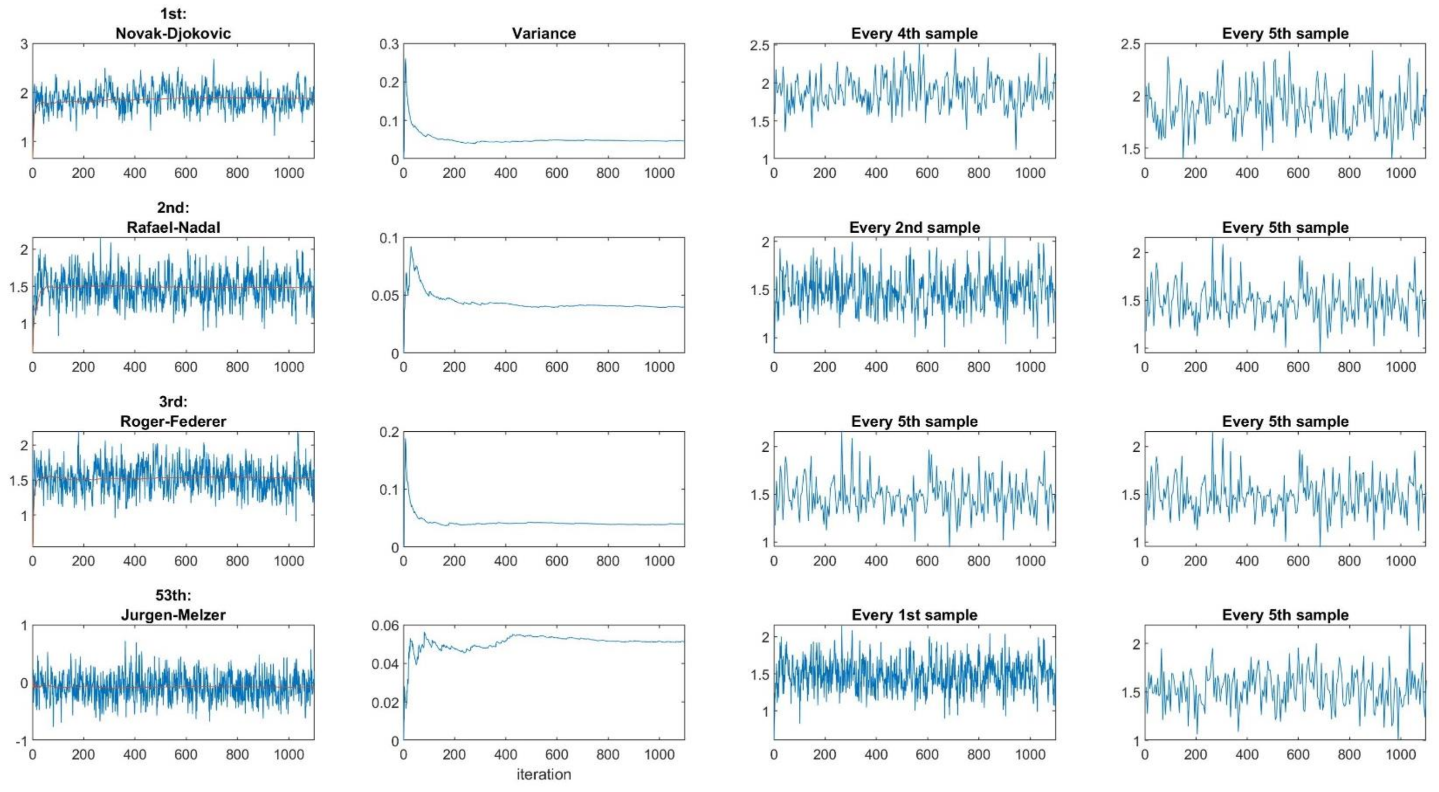
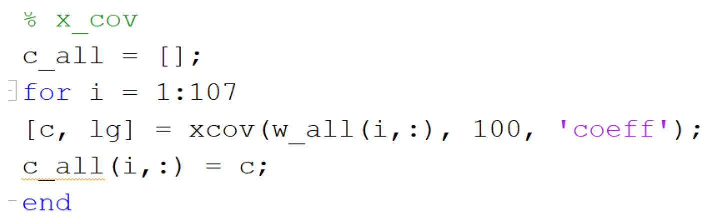


Figure 3: The 1st column shows all the samples of the player for 1100 iterations as the blue lines and the mean of the samples as the red lines. The 2nd column shows the variance changes over iteration. The 3rd column shows the mixing time for each player so that the samples can become independent. The 4th column shows when every 5th sample is sampled, samples from each player will be independent.

From 1st column of Figure 3, if each iteration of skill sample is from the same posterior distribution, the samples for each player should be oscillating according to the mean and variance of the distribution. So, the gibbs sampler is able to move around the whole posterior distribution.



```
% x_cov
c_all = [];
for i = 1:107
[c, lg] = xcov(w_all(i,:), 100, 'coeff');
c_all(i,:) = c;
end
```

Figure 4 shows how the auto-covariance coefficients of each player can be found, and stored into a matrix which contains all the coefficients of all players.

To find the auto-covariance coefficients of each player, the row vector (1x1100) representing the player's skill samples over all iterations is put into the xcov function which outputs the vector (1x201) as the coefficients shown in Figure 4. And the sum of all coefficients in the vector gives the mixing time for each player.

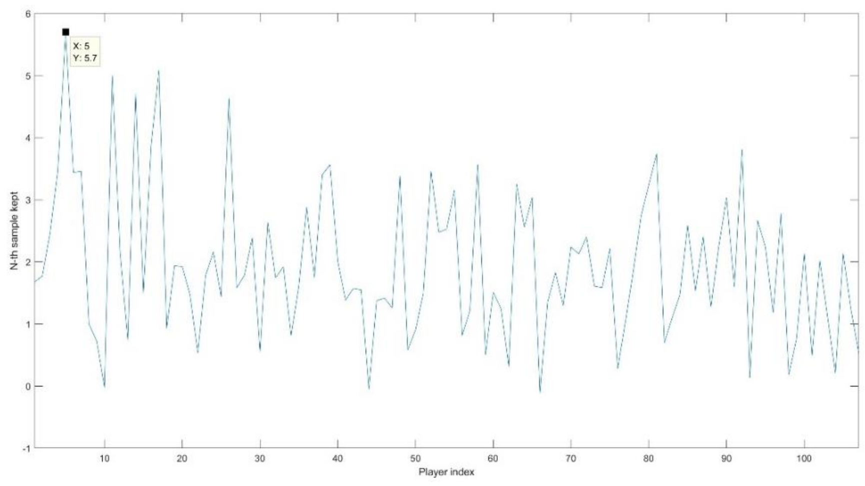


Figure 5 shows the mixing times for all players, and the mixing time peak.

Comparing all these sums from the players in Figure 5, the peak mixing time is 5.7. If the player has the sum of lower than or equal to 1, it means every sample has to be sampled to achieve independence. Even though a higher mixing time is used to sample this player, his samples are still independent. So, the peak mixing time should be taken. And to divide 1100 iterations completely, so every 5th Gibbs sample is kept to achieve independent sampling.

Burn-in describes the practice of throwing away some iterations at the beginning of an Markov Chain Monte Carlo.

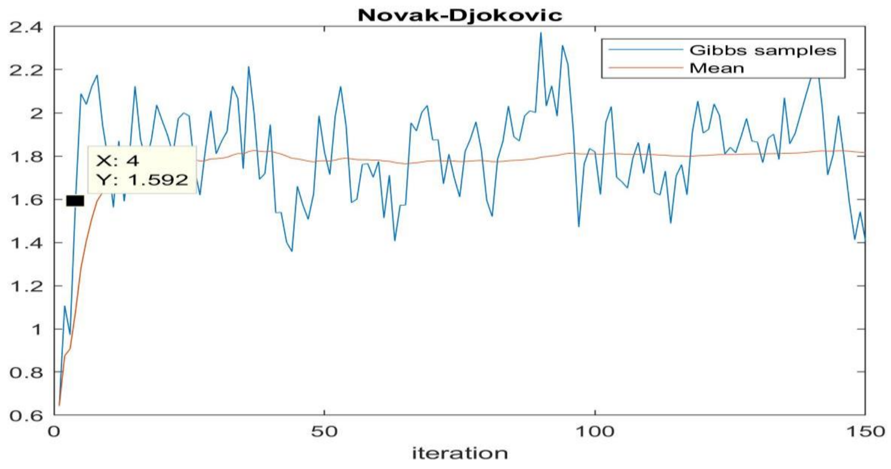


Figure 6 shows the burn-in time of the Markov Chain Monte Carlo algorithm.

When zooming in the samples in Figure 6, there are always the first 4 samples far away from the equilibrium distribution. So, they are the burn-in samples.

b.

For the gibbs sampler, the convergence occurs when the samples from conditional posterior can approximate the joint posterior distribution. It converges when the mean and variance of the samples are stabilized.

From 1st and 2nd columns of the Figure 3, the mean of the player's skill begins to stabilize quite early, around 50th iteration. But the variance of the skill continues to flunctuate until 300th iteration. So the required iterations should be 300.

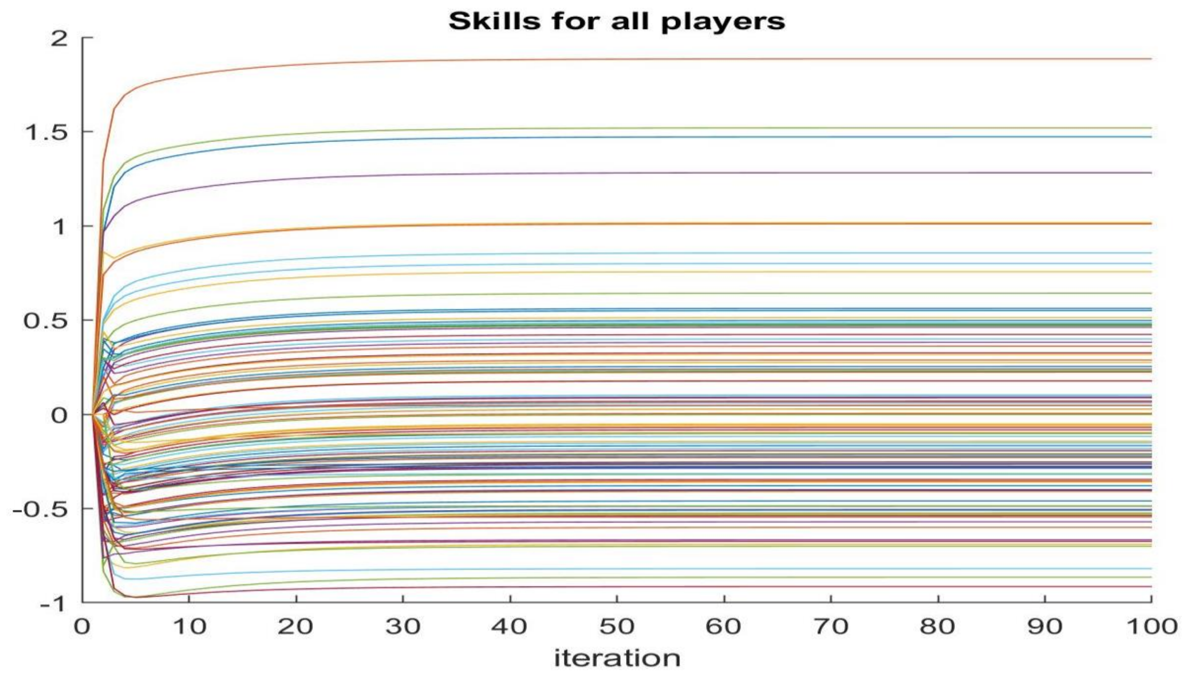


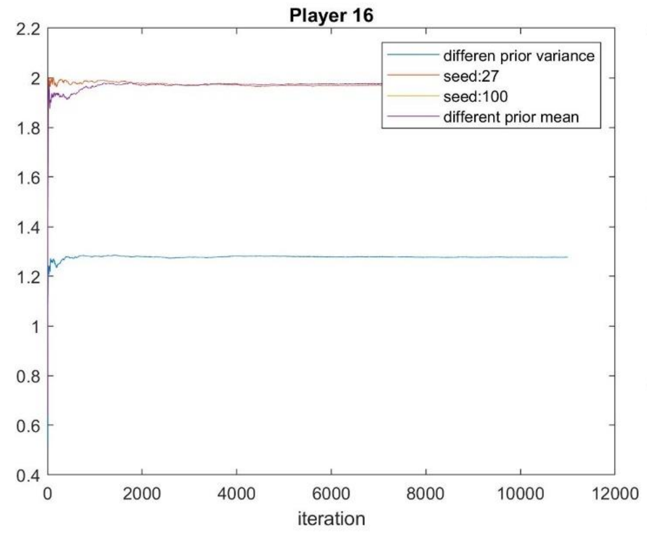
Figure 7 shows the changes of the skills for all players in 100 iterations.

In the message passing algorithm, it converges when the messages propagations can no longer update and change the distribution. From Figure 7 above, the skills of the players are getting updated continuously from the 1st to 20th iteration. But they converge after 30th iteration which is the number of iterations required.

Table 1 compares the ranks and skill means assigned by the gibbs sampler and message passing algorithm to the players.

Player	Gibbs sampler		Message passing	
	Rank	Skill mean	Rank	Skill mean
Novak-Djokovic	1	1.8917	1	1.8854
Rogel-Federer	2	1.5349	2	1.5192
Jurgen-Melzer	53	-0.0604	53	-0.0825

Comparing all the rankings generated from gibbs sampler and message passing, they are found to be not identical. The skill means estimated by the algorithms are different. However, both algorithms assign the same ranks to the strong players, especially for the top 10 players.



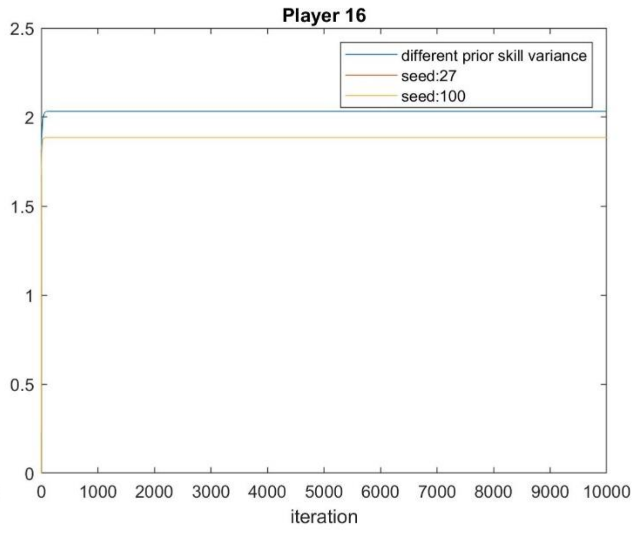


Figure 8: shows how the results of gibbs sampler (left) and messaging passing algorithm (right) are affected by different initializations and pseudo random number seeds.

The results of the gibbs sampler and messaging passing algorithm should be independent of initialization and pseudo random number seeds if the gibbs sampler has infinite time to sample. But the prior skill variance is found to affect the results.

c.

Table 2 shows the skill means and standard deviations of top 4 players computed by the message passing algorithm.

Player	Skill mean	Skill std
Djokovic, Novak	1.8854	0.1991
Nadal, Rafael	1.4716	0.1770
Federer, Roger	1.5192	0.1890
Murray, Andy	1.2811	0.1931

The probailisties that the skill of one player is higher than the other are $\Phi\left(\frac{y(\mu_1-\mu_2)}{\sqrt{\sigma_1^2+\sigma_2^2}}\right) = normcdf\left(\frac{y(\mu_1-\mu_2)}{\sqrt{\sigma_1^2+\sigma_2^2}},0,1\right)$, and the probability of one player winning a match between the two is $\Phi\left(\frac{y(\mu_1-\mu_2)}{\sqrt{1+\sigma_1^2+\sigma_2^2}}\right) = normcdf\left(\frac{y(\mu_1-\mu_2)}{\sqrt{1+\sigma_1^2+\sigma_2^2}},0,1\right)$.

The difference between them is the noise ($n \sim \mathcal{N}(0,1)$) introduced in $t = s+n$ to the latter probability to account for performance inconsistency.

Based on Table 2 and the equations above, Table 3 and 4 are computed.

Table 3 shows the probability that the skill of one player is higher than that of another player. The players listed in the 1st column are assumed to have greater skills than those in the 1st row in the match.

Greater	Djokovic, Novak	Nadal, Rafael	Federer, Roger	Murray, Andy
Djokovic, Novak		0.9398	0.9089	0.9853
Nadal, Rafael	0.0602		0.4271	0.7665
Federer, Roger	0.0911	0.5729		0.8109
Murray, Andy	0.0147	0.2335	0.1891	

Table 4 shows how probable the players in the 1st column can win against those in the 1st row.

Winner	Djokovic, Novak	Nadal, Rafael	Federer, Roger	Murray, Andy
Djokovic, Novak		0.6554	0.6380	0.7198
Nadal, Rafael	0.3446		0.4816	0.5731
Federer, Roger	0.3620	0.5184		0.5909
Murray, Andy	0.2802	0.4269	0.4091	

Table 4 favors players with lower skill means, and assigns them higher probability of being more skilful than the opponent with higher skill means. Table 3 favors stronger players, and assigns them higher probability to win.

d.

Table 5 shows the skill means and standard deviations found by the gibbs sampler for the top 4 players.

Player name	Player index	Skill mean	Skill std
Djokovic, Novak	16	1.8917	0.2032
Nadal, Rafael	1	1.4807	0.1937
Federer, Roger	5	1.5349	0.1928
Murray, Andy	11	1.3021	0.2231

1)

Assume that the marginal skills are defined as,

$$w_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$$

Using the equation that computes Table 3 from part c, the probability that the player 1 has a better skill than player 16 is,

$$\mathrm{p}(w_1 > w_{16}) = 0.0716$$

The probability that the player 16 has a better skill than player 1 is,

$$p(w_{16} > w_1) = 0.9284$$

2)

From 1100 iterations of skills samples from players 1 and 16, the first 4 samples are thrown away for the purpose of burn in time, and every 5th sample is taken because of the mixing time. The remainings are the 220 effective samples for each player. Then the number of times that player 1 has greater skill than player 16 is counted. This infers the probability.

$$p(w_1 > w_{16}) = \frac{\#(w_1 > w_{16})}{220} = \frac{10}{220} \approx 0.0455$$

$$p(w_{16} > w_1) \approx 0.9545$$

These two alternatives describe the same distribution of gibbs samples. The first one is taking all the 220 gibbs samples for each player and computing out the mean and variance. Then put them into the equation to find the probability that one player has a higher skill. The second approach just infers the probability from the gibbs samples. If more gibbs samples are considered, the second approach must converge to the the first approach. Therefore, the first approach is better as it is less sensitive to the outliners.

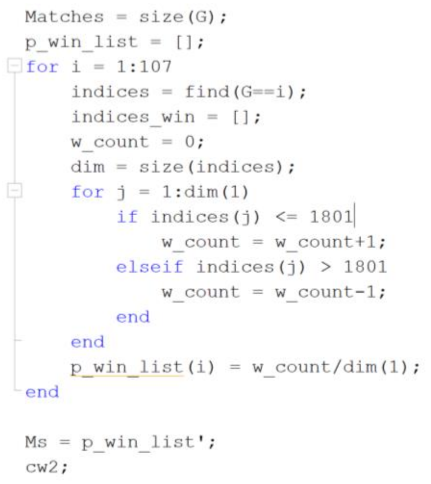
Using the first method, the following table is computed.

Table 6 shows the probability that the skills of 1st column players are greater than the those in 1st row.

Greater	Djokovic, Novak	Nadal, Rafael	Federer, Roger	Murray, Andy
Djokovic, Novak	▨	0.9284	0.8986	0.9746
Nadal, Rafael	0.0716	▨	0.4214	0.7272
Federer, Roger	0.1014	0.5786	▨	0.7851
Murray, Andy	0.0254	0.2728	0.2149	▨

Comparing Table 3 with 6, a similar result is obtained, because message passing algorithem is just an approximation of MCMC gibbs sampling algorithm, and Gibbs sampler is not running long enough. Table 3 seems to give stronger player higher probability that he has a higher skill. But for players with similar skills, such as Nadal-Rafael and Federer-Roger, Table 6 assigns higher probability that a player has a higher skill to the stronger player.

e.



```
Matches = size(G);
p_win_list = [];
for i = 1:107
    indices = find(G==i);
    indices_win = [];
    w_count = 0;
    dim = size(indices);
    for j = 1:dim(1)
        if indices(j) <= 1801
            w_count = w_count+1;
        elseif indices(j) > 1801
            w_count = w_count-1;
        end
    end
    p_win_list(i) = w_count/dim(1);
end

Ms = p_win_list';
cw2;
```

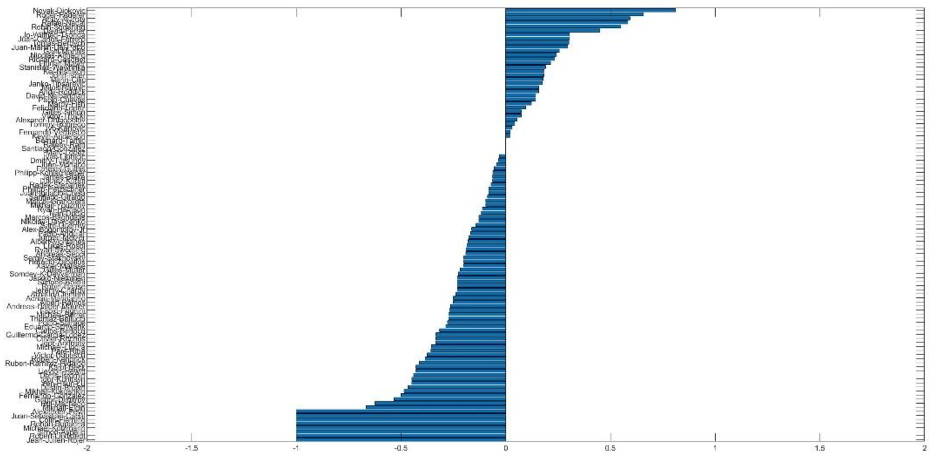


Figure 9 shows the codes used to extract empirical game outcome averages for all players on the left. The ranking of top 50 players is shown on the right.

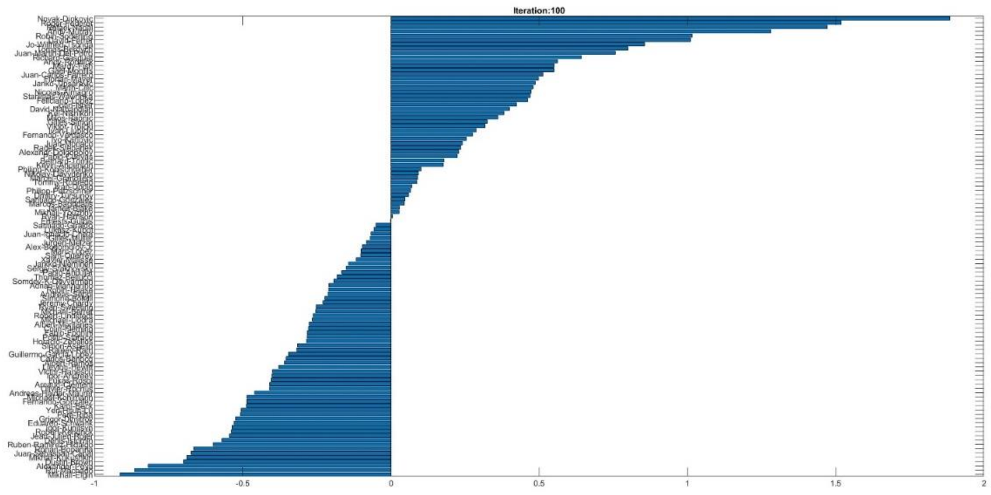


Figure 10 shows the ranking of skill means generated by the message passing algorithm for all players in 100 iterations.

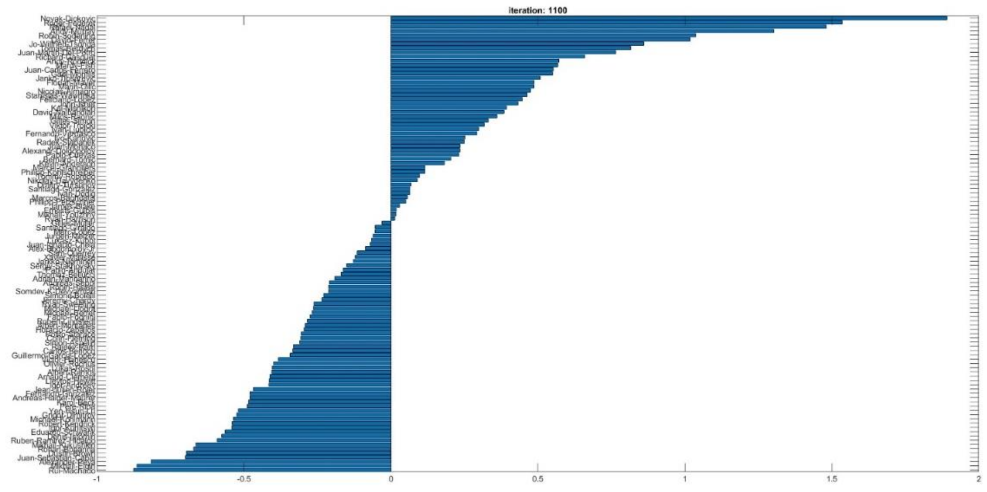


Figure 11 shows the ranking of the skill means generated by the gibbs sampler for all players in 1100 iterations.

Ranking based on empirical game outcome averages (ranged from -1 to 1), shown in Figure 9, shows more negative skills than other ranking systems. It is unfair for the player who played more and lost more, and cannot really distinguish the players at the bottom. Since the range is smaller, it cannot rank the players distinctively.

In Figure 10 and 11, both rankings generated by gibbs sampling and message passing algorithms look similar and use full skill range from -1 to 2. But gibbs sampler has more positive skill means players than the message passing.