ENGINEERING TRIPOS PART IIB

4F13 Coursework #2: Probabilistic Ranking

Candidate 5606G

970 Words

November 17, 2021

1 Part A: Gibbs Ranking

Python A.1:

```
for g in range(N):
    j = G[g,0]
    k = G[g,1]
    summed[j,g], summed[k,g] = 1, -1    # a matrix used in calculating the mean
    iS[k,k], iS[j,j] = iS[k,k] + 1, iS[j,j] + 1
    iS[k,j], iS[j,k] = iS[k,j] - 1, iS[j,k] - 1
m = np.matmul(summed, t)
```

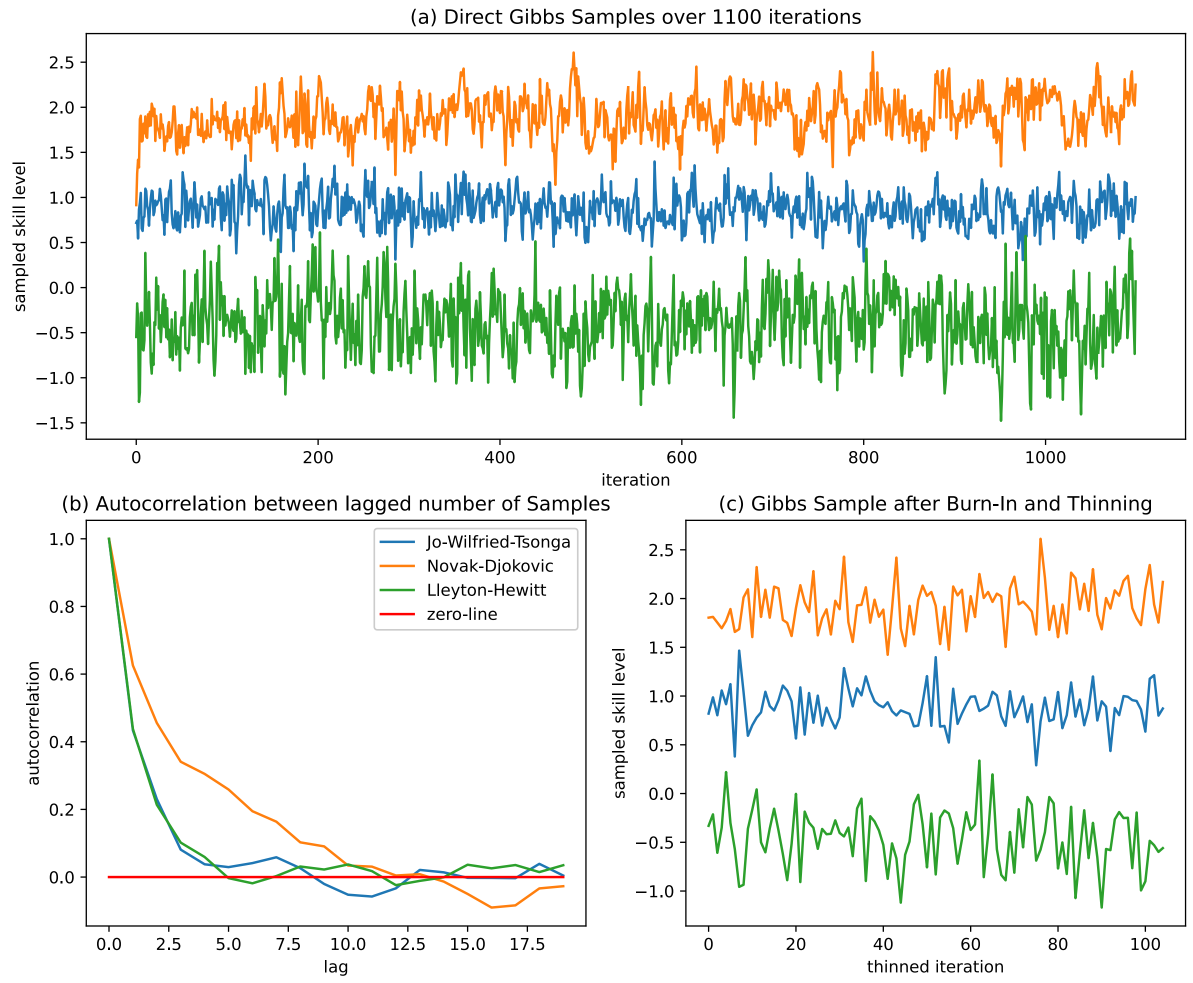

Figure 1: Results of Gibbs Sampling for 3 Players (legend in (b) is consistent across figures)

Fig. 1(a) shows the resulting sampled skills from Gibbs Sampling. The burn-in is the number of samples at the start of a run that we throw away. We do this because our starting point is random, and can be well into the tails of our modelled distribution. Fig. 1(a) shows that our players maintain a similar spread after less than 50 iterations, so we choose 50 as our burn-in. Gibbs Sampling inherently has dependence on one sample to the next, as each is picked based on the previous one. To reduce this effect, we 'thin' our samples by picking every n^{th} sample, where we hope the autocorrelation between sample i and $n + i$ is negligible.

We can see from plot 1(b) that the autocorrelation for each player's samples is close to 0 after a lag of 10, so we choose $n = 10$ as our interval. Fig 1(c) shows our resulting Gibbs Sample of 105 samples with burn-in and thinning. We can judge when the algorithm has produced enough samples to provide reliable results by when the marginal mean and variance for each player over all kept samples is stable. In fig 2, this occurs after 500 samples for the mean and variance.

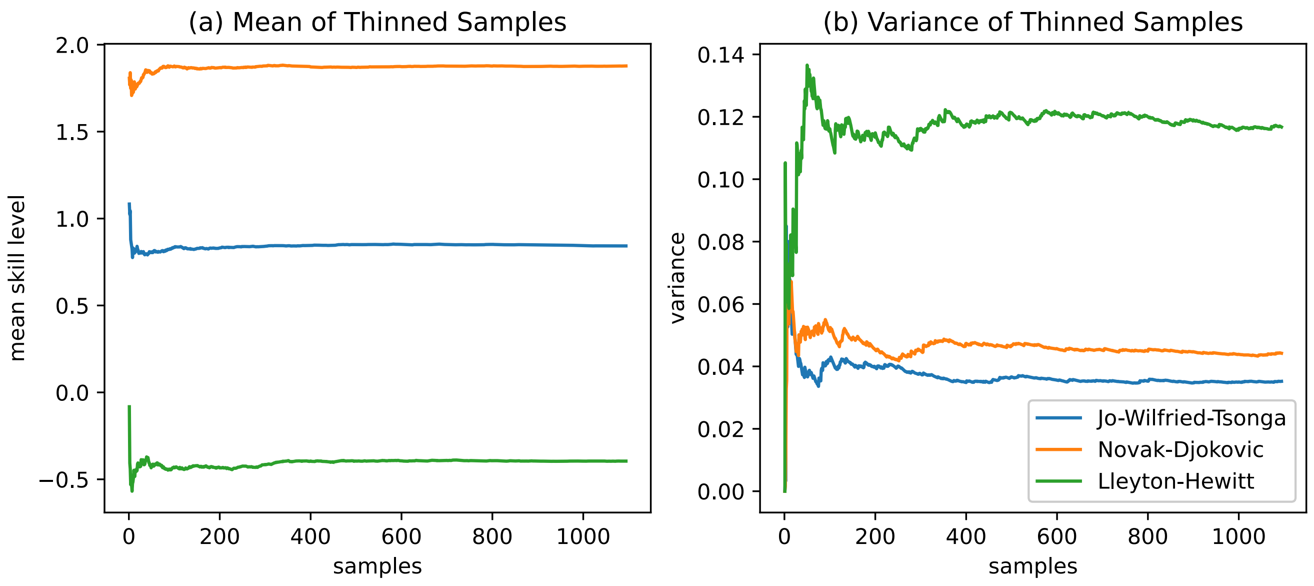

Figure 2: Mean and Variance over Increasing number of Thinned and Burned-In Samples

2 Part B: Expectation Propogation (EP)

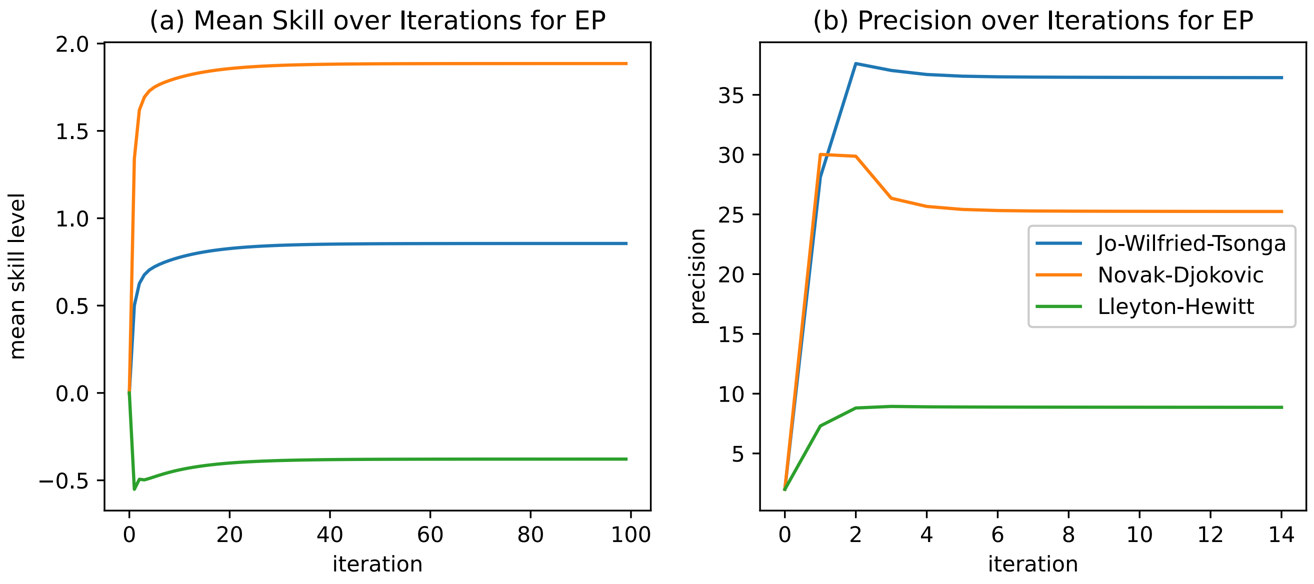

Figure 3: Results of EP for 3 Players (legend in (b) is consistent across figures)

In the EP algorithm, we are trying to converge to the best approximation of the true distribution $p(x)$ of the players' skill by a marginal gaussian distribution of skill for each player. In this case, convergence is when our mean and variance for each player becomes constant over iterations. From fig 3 we can see that our means converge after 50 iterations and our precisions (inverse of variances) converge after 8.

The Gibbs sampler is a Markov Chain Monte-Carlo (MCMC) method to produce samples from $p(x)$. The stationary distribution of the Markov Chain is $p(x)$ and convergence is when the probability distribution over the states is non-changing i.e. has reached the stationary distribution. Since we can't look at this distribution directly, we are converging to a set of samples that can reflect the intractable joint distribution for a particular purpose. One purpose could be to find a marginal mean and variance for each player, as discussed in part a. Another could be to find suitable mean and covariance matrices for a joint gaussian across all players. We will see in part d that this requires many more samples.

3 Part C: EP Probabilities for the Top 4 Players

Python C.1:

```
for i in range(4):
    for j in range(4):
        player1 = players[i]
        player2 = players[j]
        mean = mean_player_skills[player2] - mean_player_skills[player1]
        sd_skill = np.sqrt((1./precision_player_skills[player1]) +
                    (1./precision_player_skills[player2]))
        skill_probabilities[i, j] = norm.cdf(0, mean, sd_skill)
```

Python C.2:

```
sd_win = np.sqrt(sd_skill**2 + 1)
win_probabilities[i, j] = norm.cdf(0, mean, sd_win)
```

P1↓ P2→	**Nadal**	**Federer**	**Murray**	**Djokovic**
Nadal	-	0.42717016	0.7665184	0.06017779
Federer	0.57282984	-	0.81083525	0.09111473
Murray	0.2334816	0.18916475	-	0.01467851
Djokovic	0.93982221	0.90888527	0.98532149	-

Table 1: Table of Probabilities that Player 1 (P1) has higher skill than Player 2 (P2) from EP algorithm

P1↓ P2→	**Nadal**	**Federer**	**Murray**	**Djokovic**
Nadal	-	0.4816481	0.57310992	0.34463295
Federer	0.5183519	-	0.59087902	0.36197303
Murray	0.42689008	0.40912098	-	0.28017431
Djokovic	0.65536705	0.63802697	0.71982569	-

Table 2: Table of Probabilities that Player 1 (P1) beats Player 2 (P2) from EP algorithm

Table 1 is computed directly from the converged means and variances of each player from the message passing. Each player's skill follows the distribution in eq. 1, so we can compute the probability that player i's skill is higher than player j's by creating a new variable $d_{ij} = w_j - w_i$. This probability is then equal to $p(d_{ij} < 0)$, so we evaluate the cdf at 0 for the distribution in eq 2.

Table 2 includes the effect of adding a normally distributed noise n to the difference between player skills to predict the outcome of a game between players. We compute this using the variable $t_{ij} = d_{ij} + n$, and the distribution of t is the same as d, except we add 1 to the variance to account for the noise. The effect of increasing the variance means we are flattening and spreading out our distribution, which is why all probabilities move closer to 0.5. Game outcomes are not deterministic from player skills, and the noise represents a variety of other factors that increase the uncertainty of the outcome.

$$p(w_i) = N(w_i; \mu_i, \sigma_i^2) \qquad (1)$$

$$p(d_{ij}) = N(d; \mu_i - \mu_j, \sigma_i^2 + \sigma_j^2) \qquad (2)$$

4 Part D: Skill Approximation Methods

Python D.1:

```
skill_prob = 1 - norm.cdf(0, mu_d - mu_n, np.sqrt(var_d + var_n))
```

Python D.2:

```
var = cov[0,0] + cov[1,1] - 2*cov[0,1]
skill_prob = 1 - norm.cdf(0, mu_d - mu_n, np.sqrt(var))
```

Python D.3:

```
skill_prob = np.mean(djokovic > nadal)
```

Number of Samples	**(1) Gaussian Marginal Skill**	**(2) Joint Gaussian Skill**	**(3) Direct From Samples**
100	0.9264	0.9662	0.9619
1000	0.9218	0.9482	0.9479
10000	0.9256	0.9497	0.9518
100000	0.9230	0.9488	0.9512

Table 3: Results of Approximation Methods for the Probability that Djokovic has Higher Skill than Nadal

The resulting probabilities for Method 2 and 3 are closer than for Method 1. Method 3 involved finding the percentage of iterations where Djokovic's sampled skill was higher than Nadal's. This is essentially a maximum-likelihood estimate, so should converge to the true distribution over enough iterations. It is highly sample dependent, and doesn't take into account the magnitude of difference between skills, so it will take much longer to converge and be noisier. However, it converging closer to the result from the joint gaussian suggests the join gaussian more accurately resembles the true distribution. The difference between method 1 and 2 is that the joint gaussian takes into account the covariance between samples of Nadal and Djokovic, which was 0.01039228.

We found few of the covariances between any players to be negative. 100000 iterations gave less than 1% of covariances as negative. Fig. 4(b) shows we are tending to 0. A positive covariance means that if Nadal's skill is sampled to be higher than his mean, then Djokovic's skill is also expected to be higher than his own mean. This makes sense for preserving the rank order of the players that we have, making this model more accurate. All covariances were also very small, so this effect is subtle. Table 4 shows this method being used across the top four players. It resembles the EP method from table 1 well. The differences are influenced by the EP method being a marginal gaussian compared to a joint gaussian for table 4.

P1↓ P2→	**Nadal**	**Federer**	**Murray**	**Djokovic**
Nadal	-	0.42720372	0.77672884	0.0512275
Federer	0.57279628	-	0.80912915	0.08090558
Murray	0.22327116	0.19087085	-	0.01251343
Djokovic	0.9487725	0.91909442	0.98748657	-

Table 4: Probabilities that P1 has higher skill than P2 from Joint Gaussian Approximated using Gibbs Samples

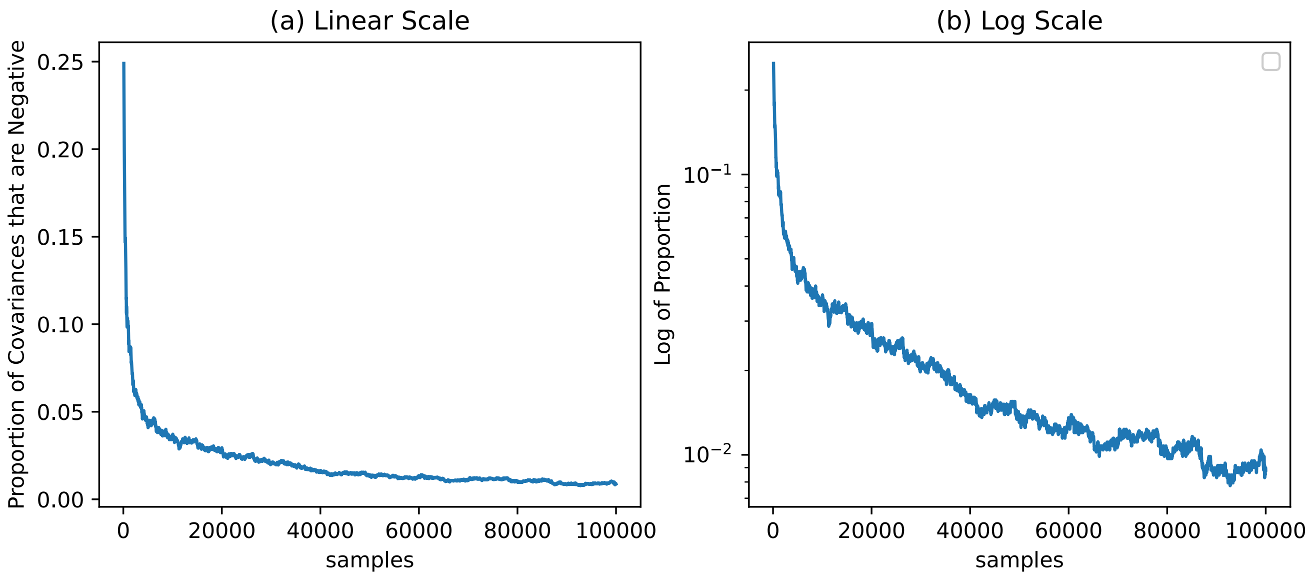

Figure 4: Proportion of Covariances between Players that are Negative with increasing Number of Samples

5 Part E: Player Rankings

Python E.1:

```
m = np.mean(skill_samples[:, 50::10], axis = 1)
cov = np.cov(skill_samples[:, 50::10], rowvar = 1, bias = True)
P = np.zeros((107,107))
for i in range(M):
    for j in [j for j in range(M) if j!=i]:
        mu = m[j] - m[i]
        var = cov[i,i] + cov[j,j] - 2*cov[i,j] + 1
        P[i,j] = norm.cdf(0, mu, np.sqrt(var))
P2 = np.mean(P, axis=1)*(107/106)
```

Python E.2:

```
for i in range(M):
    for j in [j for j in range(M) if j!=i]:
        mu = m[j] - m[i] # means from eprank
        var = v[i]+v[j]+1 # variances from eprank
        P[i,j] = norm.cdf(0, mu, np.sqrt(var))
P3 = np.mean(P, axis=1)*(107/106)
```

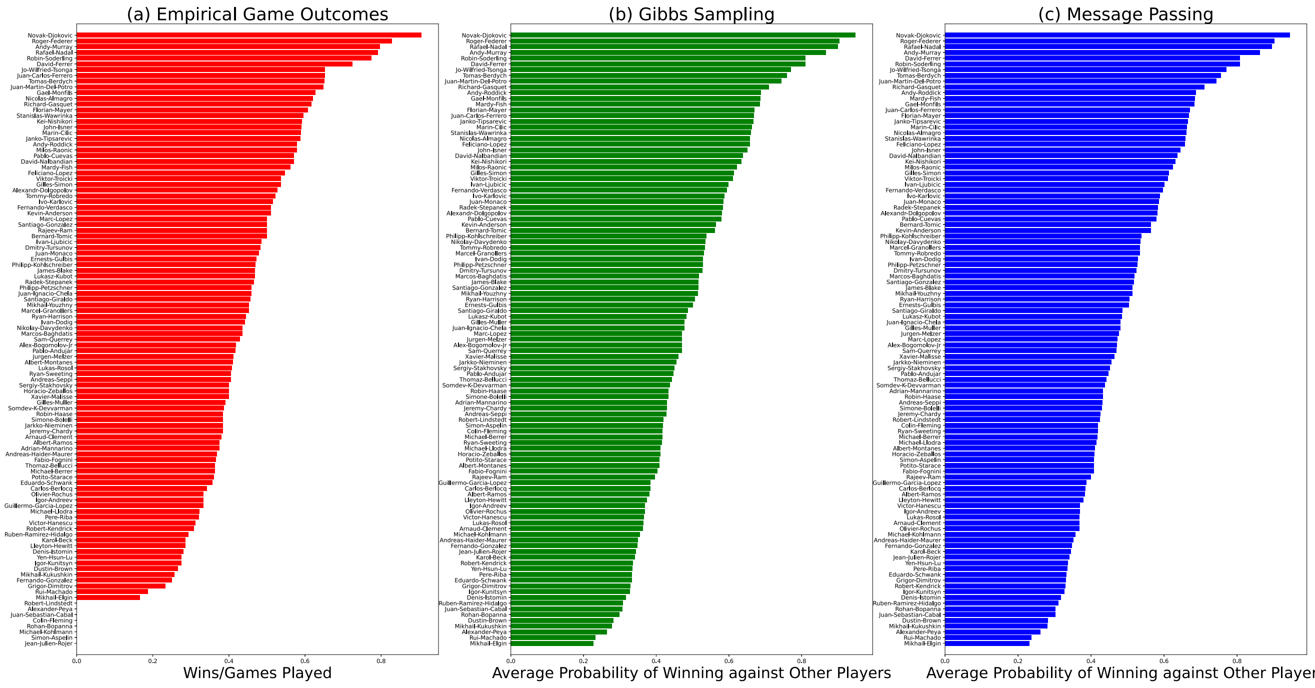

Figure 5: Player Rankings Based on Different Algorithms

We ranked the players empirically from game outcomes by the percentage of games they played that they won. Our ranking from Gibbs Sampling averaged probabilities of a player winning against each of the other players using an approximated joint gaussian, like in part d. The resulting rankings are shown in fig. 5. The empirical method is the worst method of ranking, as it doesn't take into account the level of the opponent in each game, and is highly dependent on the number of games each player has played. A player that has only played one game and won will rank the highest. All players with no wins are ranked equally, independently of how many losses they each had.

The rankings produced from Gibbs and EP are very similar. This gives more evidence that the marginal means and variances produced agree well. The Gibbs Sampling method takes much longer to converge but accounts for covariances between players. The EP method has quick convergence and the predicted parameters are robust to more iterations, but does not include joint correlations.