All central commands for parts A-E can be found in the appendix.

1 A

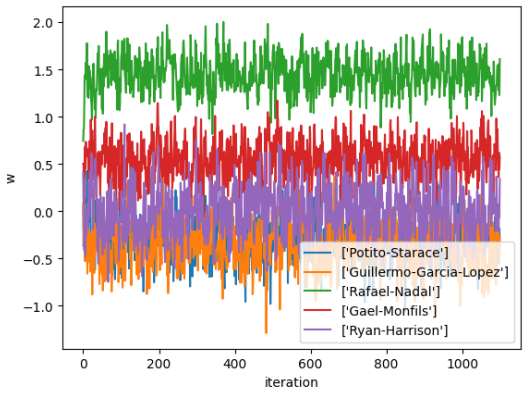


Figure 1: Players skills of 5 different players as a function of the Gibbs iteration.

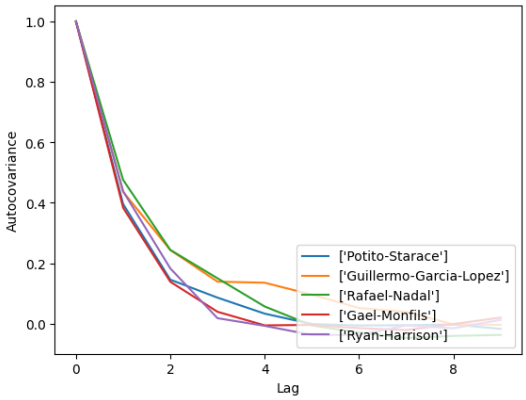
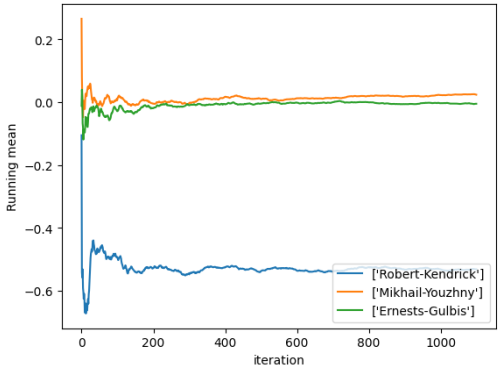
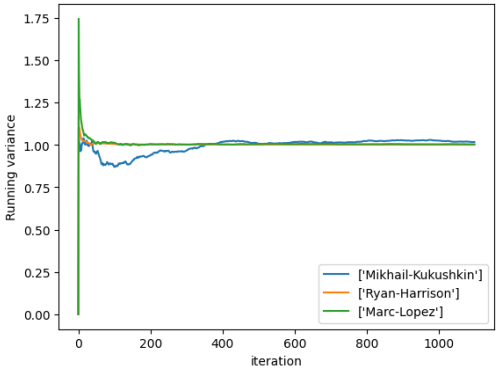


Figure 2: Auto covariance coefficient

As observed in figure 3, the running mean and variance settles after around 200 iteration. Therefore the burn-in period = 200. When generating samples from the conditional distribution, we'd ideally like the samples to be independently generated, however, that is not the case. Looking at figure 2 suggests that sample approximately 5 iterations apart are effectively independent.
Therefore, run the Gibbs sampler for at least 200 iterations for burn-in period, then perform thinning by taking every 5th sample afterwards. For 100 'independent' samples we ideally like at least 700 iterations.



(a) Running mean of players skills.



(b) Running variance of players skills.

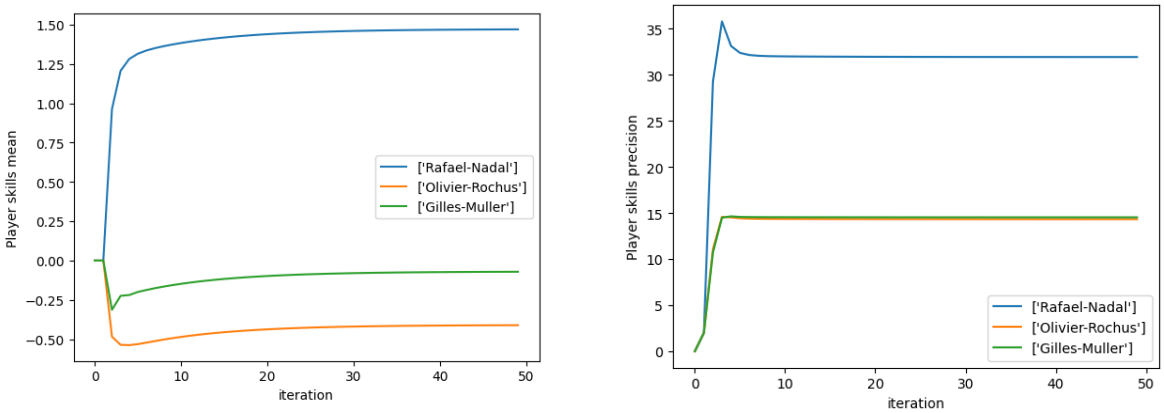
Figure 3: Running moments for Gibbs Sampling

Due to the Cholesky factorization used in computing the inverse of the posterior precision matrix to generate the next sample, the computation complexity of running the Gibbs Sampler is $O(N^3 * iteration)$.

2 B

In Gibbs sampling, convergence is the point when the generating chain reaches the stationary distribution. At the point of convergence, we have discarded the effects of initial values, since it may have been a low

probability state. Convergence in this case can be observe when the mean/variance settles. We saw that at least 200 iterations were necessary.
In message passing, we conceptualize the marginals of variables as nodes as and the likelihood or posterior as factors, of a graph. We initialize the skill marginals as some value that propagates through the graph to give us the performance marginals, then we propagate the performance marginals back to update our skill marginals, repeating back and forth until convergence occurs when the marginal values settle. We see in figure 4 that it takes around 40 iteration for the mean players skills to settle.



(a) Mean players skills

(b) Precision of players skills

Figure 4: Mean and precision of 3 players as a function of message passing iteration

3 C

i/j	Djokovic	Nadal	Federer	Murray
Djokovic	0.5	0.940	0.909	0.985
Nadal	0.0602	0.5	0.427	0.766
Federer	0.0911	0.573	0.5	0.811
Murray	0.0147	0.234	0.189	0.5

Table 1: Probability that the skill of player i is greater than player j using EP

i/j	Djokovic	Nadal	Federer	Murray
Djokovic	0.5	0.660	0.643	0.727
Nadal	0.340	0.5	0.481	0.576
Federer	0.357	0.519	0.5	0.594
Murray	0.273	0.424	0.406	0.5

Table 2: Probability that player i wins player j in a match using EP

The probability of player i's skill, w_i being greater than player j's skill, w_j, is simply the probability that the random variable $w_i - w_j$ is greater than 0.

$$\begin{aligned} P(w_i > w_j) &= P(w_i - w_j > 0) \\ &= \int_0^\infty N(z; \mu_i - \mu_j, \sigma_i^2 + \sigma_j^2) dz \end{aligned} \tag{1}$$

Where μ_i is the mean skill of player i and $\sigma_i^2 = v_i^{-1}$, the inverse of the precision of player i.
The probability that player i wins a match with player j is however:

$$\begin{aligned} P(y=1) &= P(t>0) \\ &= \int_0^\infty N(z;\mu_i-\mu_j,1)dz \\ &= \Phi(w_i-w_j) \end{aligned} \tag{2}$$

Interpret it as, given the mean skills of two players, take into account any real-life randomness in a game to determine how likely who is going to win.

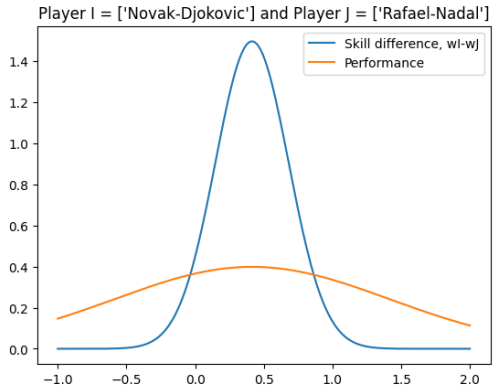
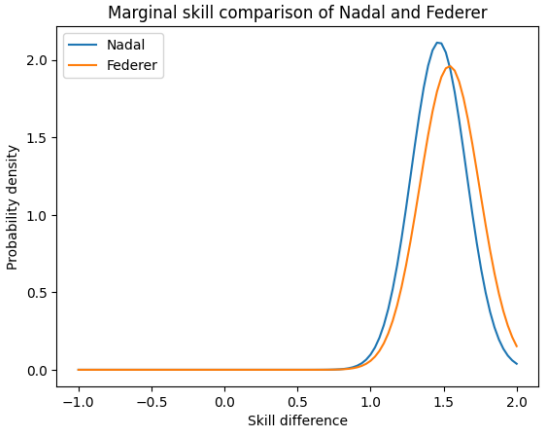


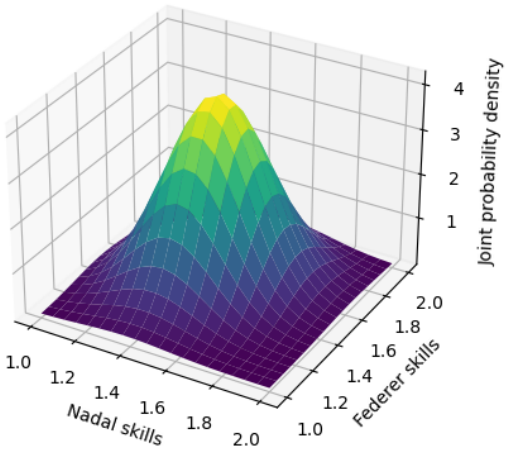
Figure 5: Skills probability density v.s. game outcome probability density

Due to the noise (variance of 1) we can see how performance has a much greater variance than skill difference in figure 5.

4 D



(a) Marginal skills of Nadal and Federer



(b) Joint skills of Nadal and Federer

Figure 6: Methods to compare the skills of Nadal and Federer using the Gibbs sampler

First, we pre-process the samples returned by the Gibbs sampler to discard the first 200 samples (burn-in) and we keep every 5th sample subsequently (thinning).
To compute the marginal skills, calculate the mean of the samples and variance of the samples. Then plot a Gaussian distribution with said mean and variance for each player.
For joint skills, we assume a multivariate Gaussian where the mean vector:

$$\tilde{\mu} = \begin{bmatrix} \mu_{Nadal} \\ \mu_{Federer} \end{bmatrix} \tag{3}$$

and the covariance matrix $\sum$ is computed using the sampled skills for Nadal and Federer,
To compare directly from the samples, just count the number of times $w_{Nadal} > w_{Federer}$ and divide by the number of samples.

Method	$P(w_{Nadal} > w_{Federer})$
Marginal gaussian	0.396
Jointly gaussian	0.386
Averaged over samples	0.367

Table 3: Probability that Nadal has greater skills than Federer

Marginal skills by Gaussian and directly from samples were very easy to compute. The joint Gaussian was a lot more complex to compute and computationally more expensive, since we had to evaluate the Gaussian at N^2 datapoints (where N is the number of datapoints we are evaluating for each x axis) and even more computation to approximate the area under the curve where $w_{Nadal} > w_{Federer}$.
However, the join Gaussian takes into account the covariance between Nadal and Federer skills, which was 0.007, not close to 0 at all. This extra information is worthy consider it is just a bivariate Gaussian and modern computers can compute it easily. Also, approximating directly from samples also requires many iteration to converge anyways. Therefore we pick the joint skill Gaussian.

i/j	**Djokovic**	**Nadal**	**Federer**	**Murray**
Djokovic	0.5	0.954	0.936	0.984
Nadal	0.046	0.5	0.386	0.781
Federer	0.0640	0.614	0.5	0.824
Murray	0.0160	0.219	0.176	0.5

Table 4: Probability that the skill of player i is greater than player j using joint skills by a Gaussian

Comparing table 1 and table 4, we see that the values are not too far off. The Gibbs method required many more iterations to converge, but allows us to compute the skill comparisons using different methods, whereas message passing only allowed marginal skills. However in real life, with so many players to rank, message passing seems efficient and gives a good enough result.

5 E

To rank by empirical game outcome average, we let skill= no. game won/ total games played by a player. Based on Gibbs sampling and message passing, the rankings are the same. Using the empirical method, rankings were mostly the same but some would be swapped around a small ranking distance away. This is expected since ranking by empirical means considers each win for a player equally, winning against a skilled player or beginners makes no difference in this method. Although the empirical ranking method is the simplest and fastest, the message passing algorithm may take a bit of effort to implement but was inexpensive to compute, therefore is the best ranking method. The Gibbs sampling method produced the same results but was a lot more computationally expensive, however, it can let us obtain some joint distributions.

A Python Code: A

```
# Jointly sample skills given performance differences
    m = np.zeros((M, 1))
    for p in range(M):
        m[p] = sum([t[g] for g in range(N) if p==G[g,0]])
                - sum([t[g] for g in range(N) if p==G[g,1]])
    iS = np.zeros((M, M))  # Container for sum of precision matrices (likelihood terms)

    for g in range(N):
        Ig=G[g,0]
        Jg=G[g,1]
        iS[Ig][Ig]+=1
        iS[Jg][Jg]+=1
        iS[Ig][Jg]-=1
        iS[Jg][Ig]-=1
```

B Python Code: C

topfour is the array of top 4 player's index in W.

```
# Using message passing algorithm
skill_matrix=[[0 for _ in range(4)] for _ in range(4)]
winning_matrix=[[0 for _ in range(4)] for _ in range(4)]
for x,i in enumerate(topfour):
    for y,j in enumerate(topfour):
        skill_matrix[x][y]=stats.norm.cdf(mean_player_skills[i]-
        mean_player_skills[j],0,(1/precision_player_skills[i]
        +1/precision_player_skills[j])**0.5)
        winning_matrix[x][y]=stats.norm.cdf(mean_player_skills[i]-mean_player_skills[j],
        0,1)
```

C Python Code: D

```
means=[mean_Djokovic,mean_Nadal,mean_Federer,mean_Murray]
skills = [skill_samples_Djokovic, skill_samples_Nadal,
skill_samples_Federer, skill_samples_Murray]

for i in range(4):
    skill_matrix[i][i]=0.5
    for j in range(i+1,4):
        iteration=100000
        count=0
        cov_matrix=np.cov(skills[i],skills[j])

        for iter in range(iteration):
            sample = np.random.multivariate_normal([means[i], means[j]], cov_matrix)
            if sample[0]>sample[1]:
                count+=1

        skill_matrix[i][j] = count/iteration
        skill_matrix[j][i]=1-skill_matrix[i][j]
```

D Bar plots for Section E Rankings

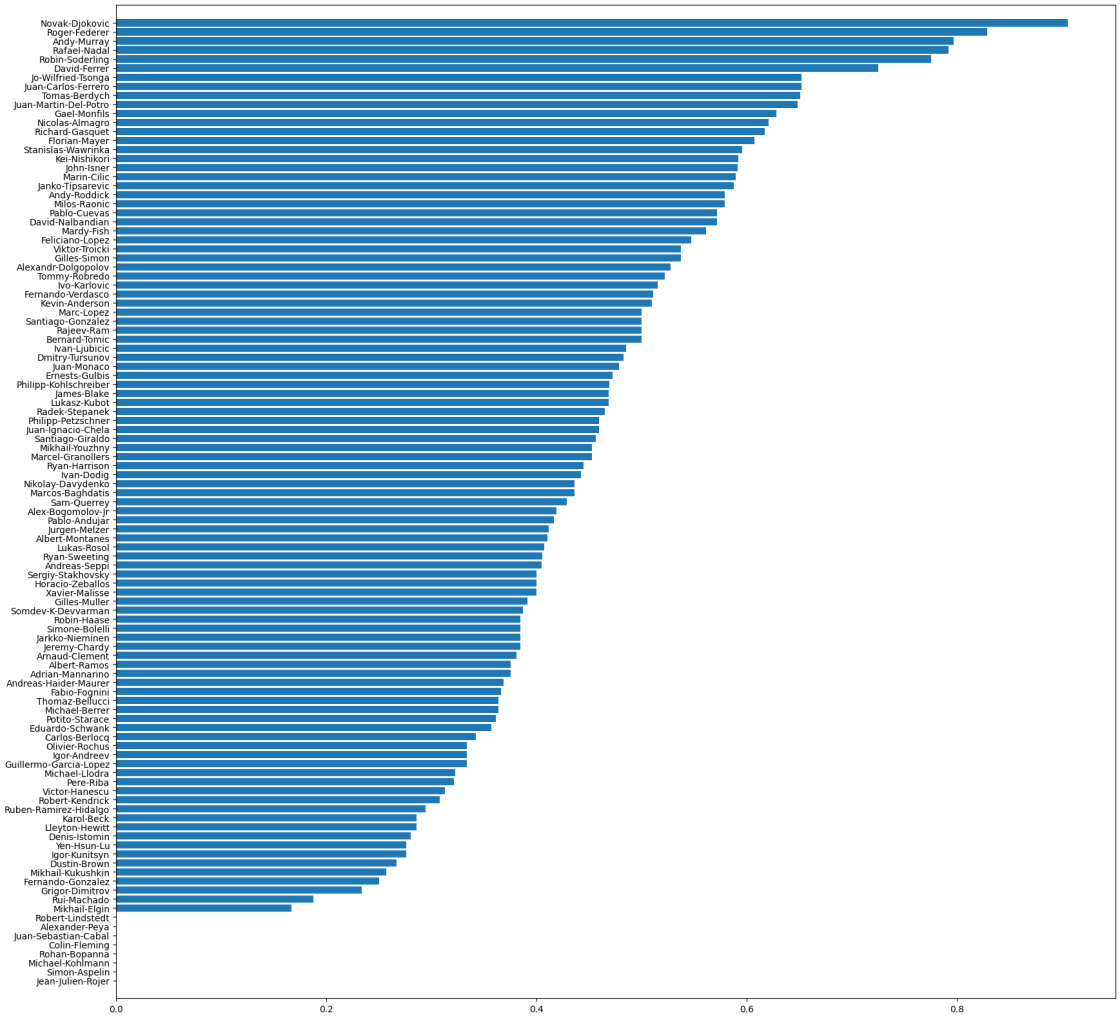


Figure 7: Ranking with empirical outcome averages

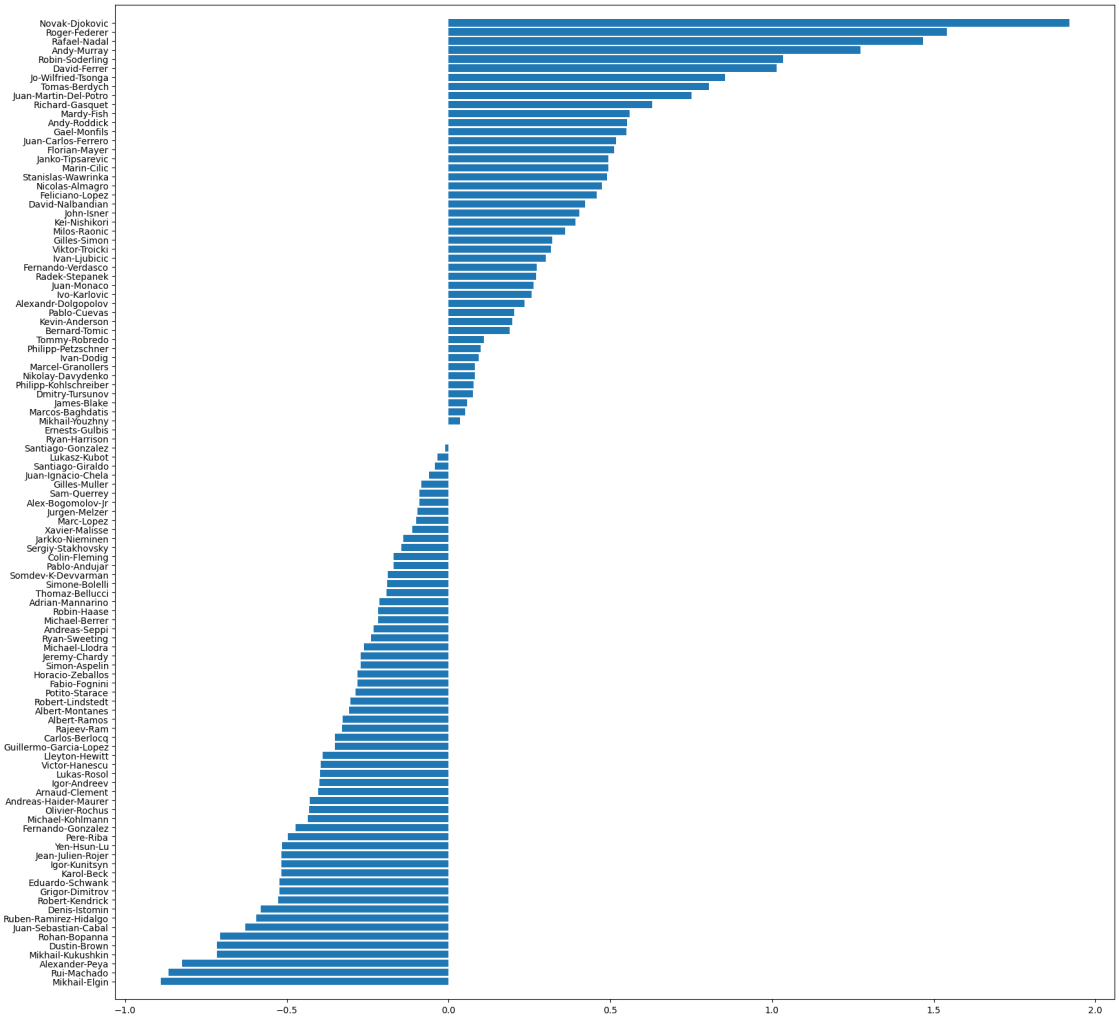


Figure 8: Ranking based on Gibbs sampling

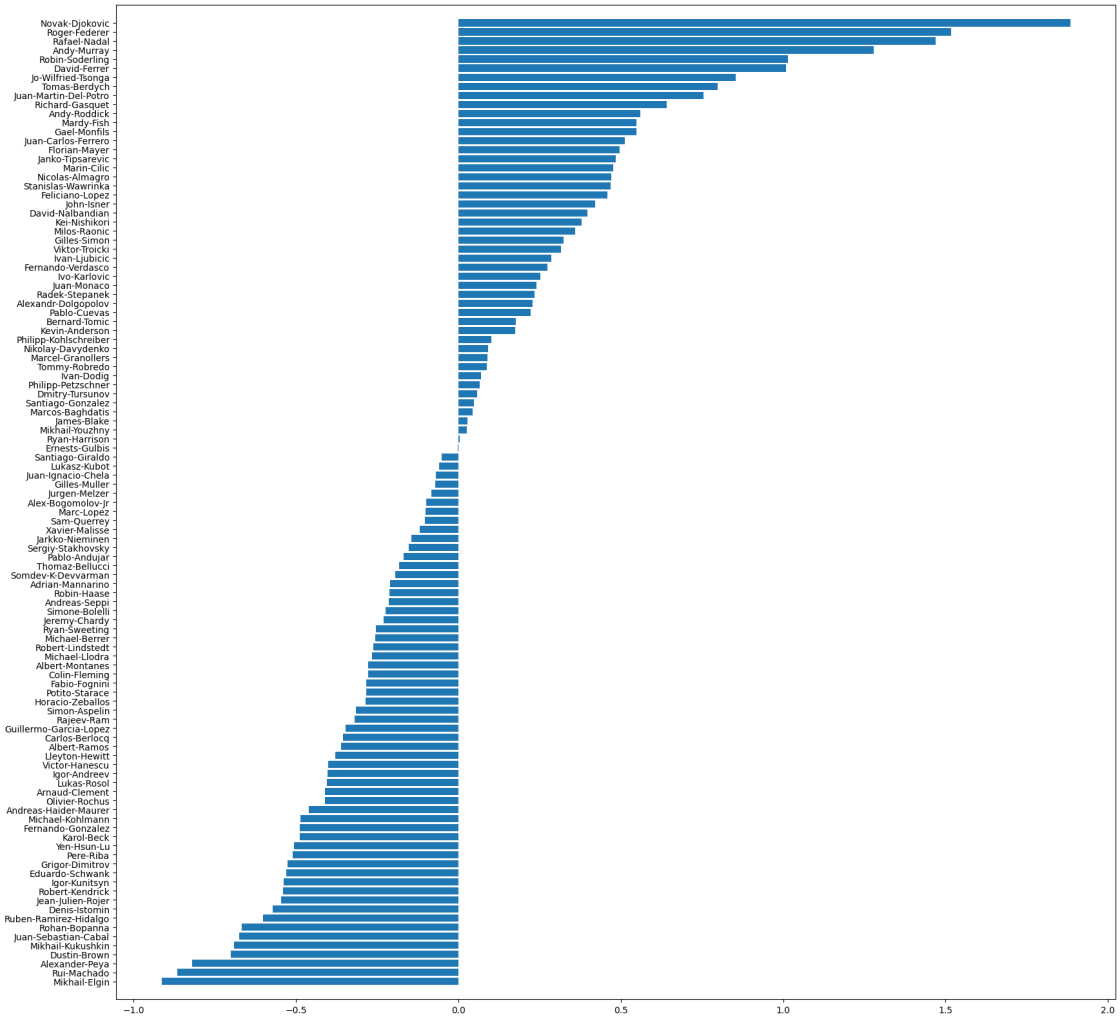


Figure 9: Ranking based on message passing algorithm