Assignment 2

Arina Sitnikova, 1005863585

2020-03-22

1. Implementing the model [10 points]

a. **[2 points]** Implement a function log prior that computes the log of the prior over all player's skills. Specifically, given a K × N array where each row is a setting of the skills for all N players, it returns a K × 1 array, where each row contains a scalar giving the log-prior for that set of skills.

```
function log_prior(zs)
  return  factorized_gaussian_log_density(0, 0, zs)
end
```

b. **[3 points]** Implement a function logp a beats b that, given a pair of skills z_a and z_b evaluates the log-likelihood that player with skill z_a beat player with skill z_b under the model detailed above. To ensure numerical stability, use the function log1pexp that computes log(1 + exp(x)) in a numerically stable way. This function is provided by StatsFuns.jl and imported already, and also by Python's numpy.

```
function logp_a_beats_b(za,zb)
  return -log1pexp(-(za-zb))
end
```

c. **[3 points]** Assuming all game outcomes are i.i.d. conditioned on all players' skills, implement a function all games log likelihood that takes a batch of player skills zs and a collection of observed games games and gives a batch of log-likelihoods for those observations. Specifically, given a K × N array where each row is a setting of the skills for all N players, and an M × 2 array of game outcomes, it returns a K × 1 array, where each row contains a scalar giving the log-likelihood of all games for that set of skills. Hint: You should be able to write this function without using for loops, although you might want to start that way to make sure what you've written is correct. If A is an array of integers, you can index the corresponding entries of another matrix B for every entry in A by writing B[A].

```
function all_games_log_likelihood(zs,games)
  zs_a = zs[games[:, 1], :]
  zs_b = zs[games[:, 2], :]
  likelihoods = logp_a_beats_b.(sum(zs_a, dims = 1), sum(zs_b, dims = 1))
  return likelihoods
end
```

d. **[2 points]** Implement a function joint log density which combines the log-prior and log-likelihood of the observations to give p($z_1, z_2, \ldots, z_N$, all game outcomes)

```
function joint_log_density(zs,games)
  return log_prior(zs) + all_games_log_likelihood(zs, games)
end
```

Tests:

```
@testset "Test shapes of batches for likelihoods" begin
  B = 15 # number of elements in batch
  N = 4 # Total Number of Players
  test_zs = randn(4,15)
  test_games = [1 2; 3 1; 4 2] # 1 beat 2, 3 beat 1, 4 beat 2
  @test size(test_zs) == (N,B)
  #batch of priors
  @test size(log_prior(test_zs)) == (1,B)
  # loglikelihood of p1 beat p2 for first sample in batch
  @test size(logp_a_beats_b(test_zs[1,1],test_zs[2,1])) == ()
  # loglikelihood of p1 beat p2 broadcasted over whole batch
  @test size(logp_a_beats_b.(test_zs[1,:],test_zs[2,:])) == (B,)
  # batch loglikelihood for evidence
  @test size(all_games_log_likelihood(test_zs,test_games)) == (1,B)
  # batch loglikelihood under joint of evidence and prior
  @test size(joint_log_density(test_zs,test_games)) == (1,B)
end
```

All 6 tests were passed.

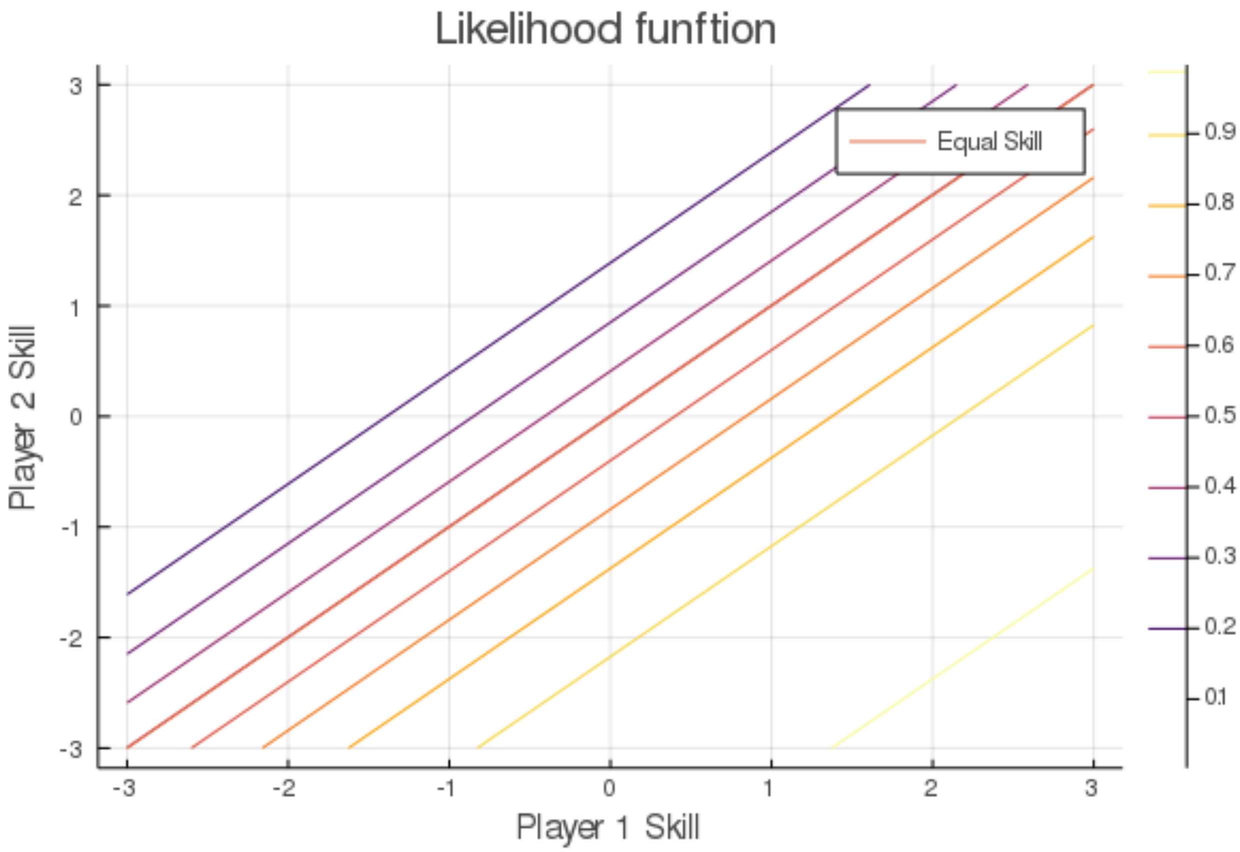
2. Examining the posterior for only two players and toy data [10 points]

a. **[2 points]** For two players A and B, plot the isocontours of the joint prior over their skills. Also plot the line of equal skill, $z_A = z_B$. Hint: you've already implemented the log of the likelihood function.

```
plot(title="Joint prior over players' skills", xlabel = "Player 1 Skill",
ylabel = "Player 2 Skill")
prior(zs) = exp(log_prior(zs))
skillcontour!(prior)
plot_line_equal_skill!()
savefig("2a.pdf")
```

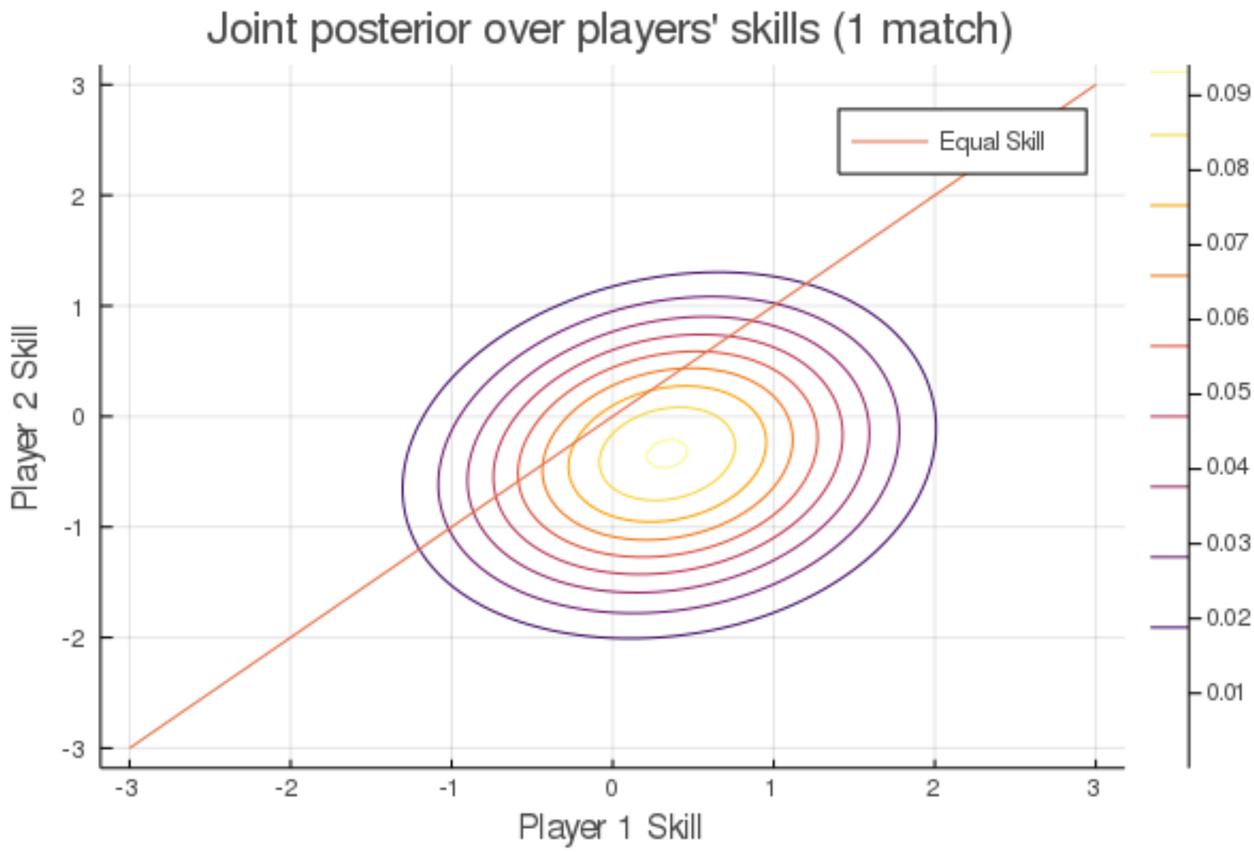
b. **[2 points]** Plot the isocontours of the likelihood function. Also plot the line of equal skill, $z_A = z_B$.

```
plot(title="Likelihood funftion",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
likelihood(zs) = exp.(logp_a_beats_b(zs[1], zs[2]))
skillcontour!(likelihood)
plot_line_equal_skill!()
savefig("2b.png")
```



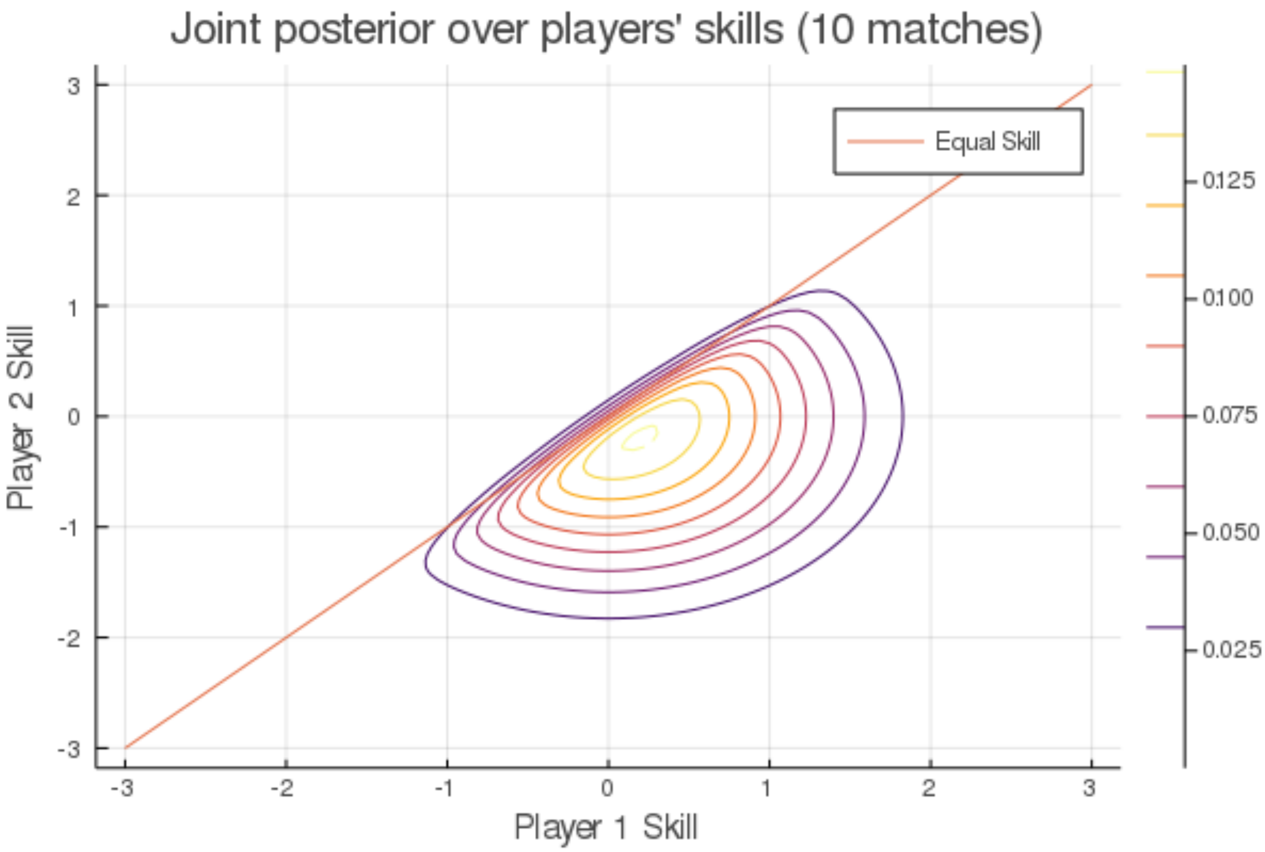
c. **[2 points]** Plot isocountours of the joint posterior over z_A and z_B given that player A beat player B in one match. Since the contours don't depend on the normalization constant, you can simply plot the isocontours of the log of joint distribution of p(z_A, z_B,A beat B). Also plot the line of equal skill, $z_A = z_B$.

```
game1 = two_player_toy_games(1, 0)
plot(title="Joint posterior over players' skills (1 match)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
posterior1(zs) = exp(joint_log_density(zs,game1))
skillcontour!(posterior1)
plot_line_equal_skill!()
savefig("2c.png")
```



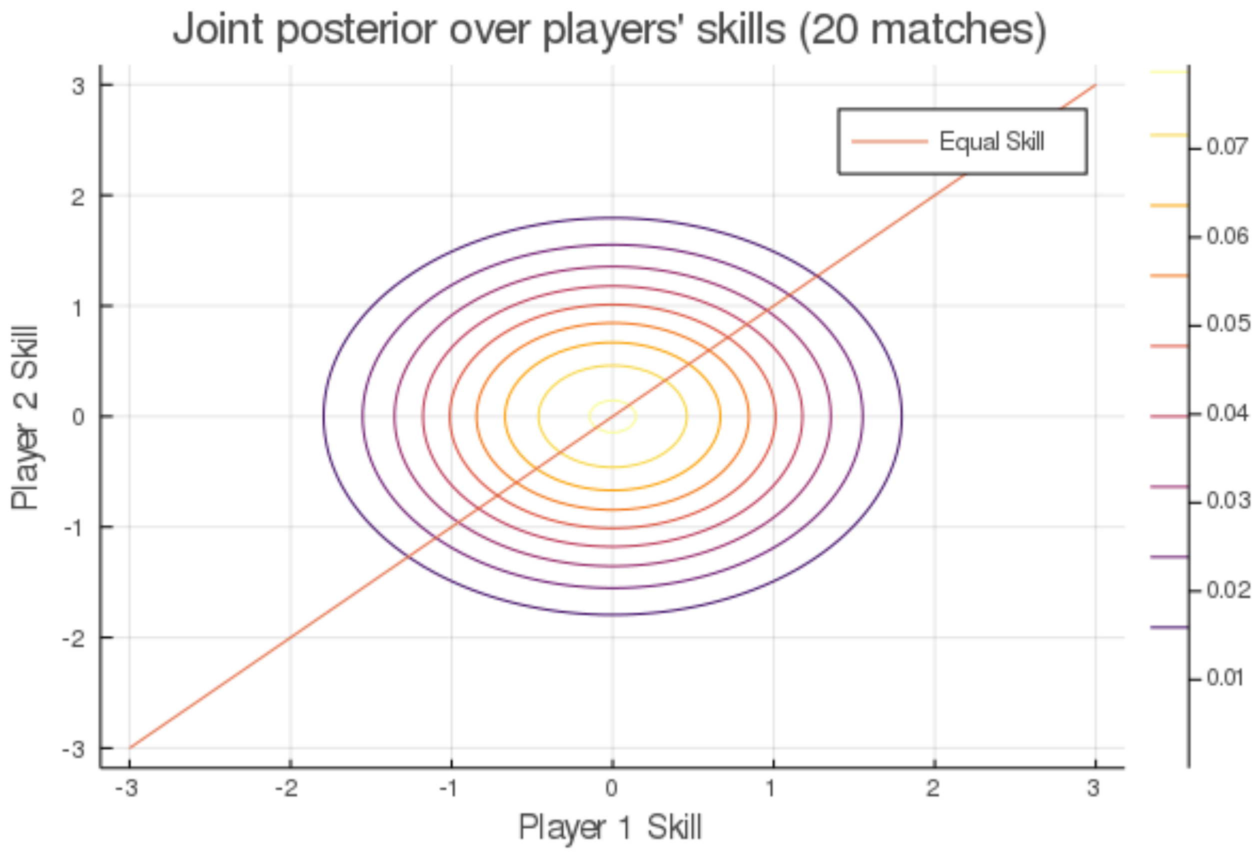
d. **[2 points]** Plot isocountours of the joint posterior over z_A and z_B given that 10 matches were played, and player A beat player B all 10 times. Also plot the line of equal skill, $z_A = z_B$.

```
game2 = two_player_toy_games(10, 0)
plot(title="Joint posterior over players' skills (10 matches)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
posterior2(zs) = exp(joint_log_density(zs,game2))
skillcontour!(posterior2)
plot_line_equal_skill!()
savefig("2d.png")
```



e. **[2 points]** Plot isocountours of the joint posterior over z_A and z_B given that 20 matches were played, and each player beat the other 10 times. Also plot the line of equal skill, $z_A = z_B$.

```
game3 = two_player_toy_games(10, 10)
plot(title="Joint posterior over players' skills (20 matches)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
posterior3(zs) = exp(joint_log_density(zs,game3))
skillcontour!(posterior3)
plot_line_equal_skill!()
savefig("2e.png")
```



3. Stochastic Variational Inference on Two Players and Toy Data [18 points]

a. **[5 points]** Implement a function elbo which computes an unbiased estimate of the evidence lower bound. As discussed in class, the ELBO is equal to the KL divergence between the true posterior $p(z|data)$, and an approximate posterior, $q_\varphi(z|data)$, plus an unknown constant.

```
function elbo(params,logp,num_samples)
  samples = exp.(params[2]).*randn(length(params[1]),num_samples) .+ params[1]
  logp_estimate = logp(samples)
  logq_estimate = factorized_gaussian_log_density(params[1],params[2],samples)
  return mean(logp_estimate - logq_estimate)
end
```

b. **[2 points]** Write a loss function called neg_toy_elbo that takes variational distribution parameters and an array of game outcomes, and returns the negative elbo estimate with 100 samples.

```
function neg_toy_elbo(params; games = two_player_toy_games(1,0), num_samples = 100)
  logp(zs) = joint_log_density(zs,games)
  return -elbo(params,logp, num_samples)
end
```

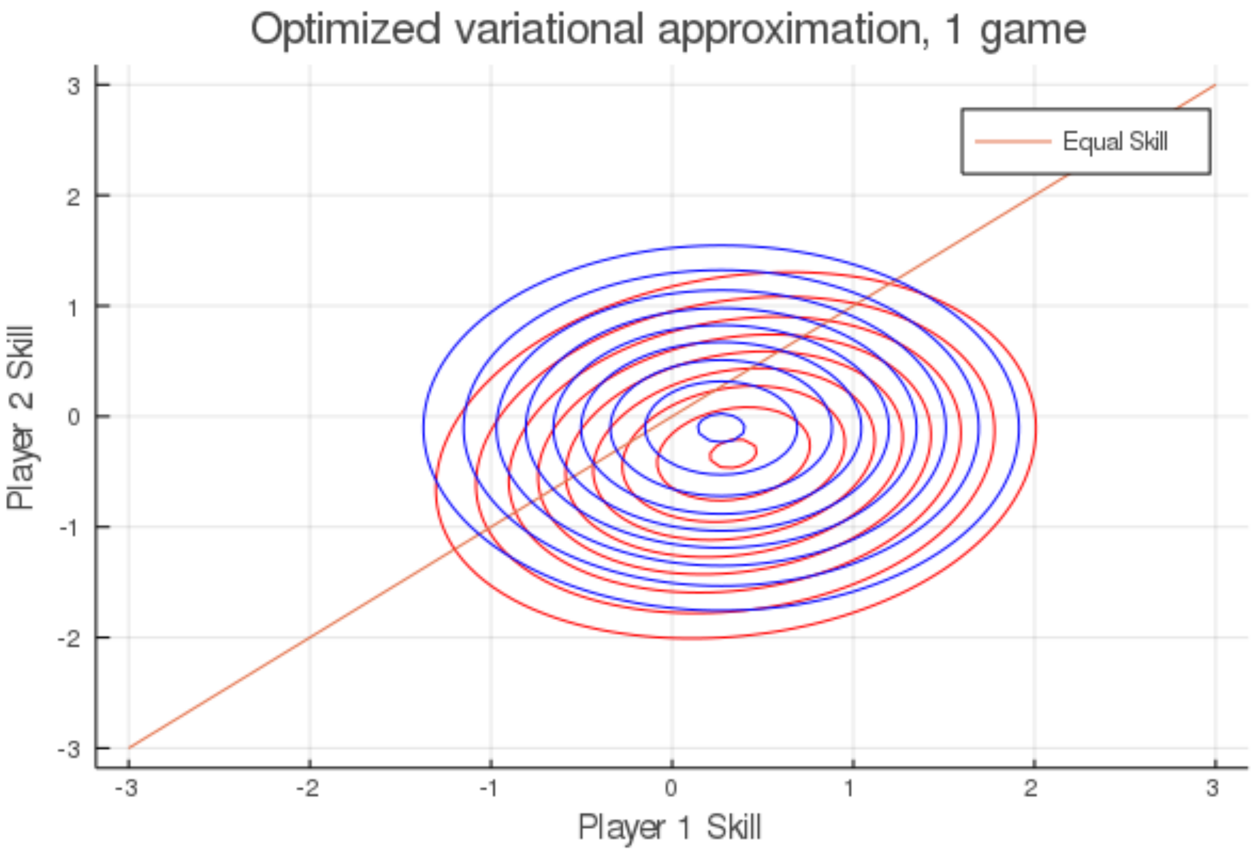
c. **[5 points]** Write an optimization function called fit_toy_variational_dist which takes initial variational parameters, and the evidence. Inside it will perform a number of iterations of gradient descent. Return the parameters resulting from training.

```
function fit_toy_variational_dist(init_params, toy_evidence; num_itrs=200, lr= 1e-2,
num_q_samples = 10)
  params_cur = init_params
  for i in 1:num_itrs
    grad_params = gradient(params -> neg_toy_elbo(params, games = toy_evidence, num_s
amples = num_q_samples),
    params_cur)[1]
    params_cur =  params_cur .- lr.*grad_params
    @info "Current ELBO" neg_toy_elbo(params_cur, games = toy_evidence, num_samples =
num_q_samples)
    plot();
    skillcontour!(zs -> exp.(joint_log_density(zs, toy_evidence)),colour=:red)
    plot_line_equal_skill!()
    display(skillcontour!(zs -> exp.(factorized_gaussian_log_density(params_cur[1], p
arams_cur[2], zs)),
    colour=:blue))
  end
  return params_cur
end
```

d. **[2 points]** Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 1 game. Report the final loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
fit_toy_variational_dist(toy_params_init, two_player_toy_games(1, 0))
title!("Optimized variational approximation, 1 game")
xlabel!("Player 1 Skill")
ylabel!("Player 2 Skill")
savefig("fig_game1.png")
```

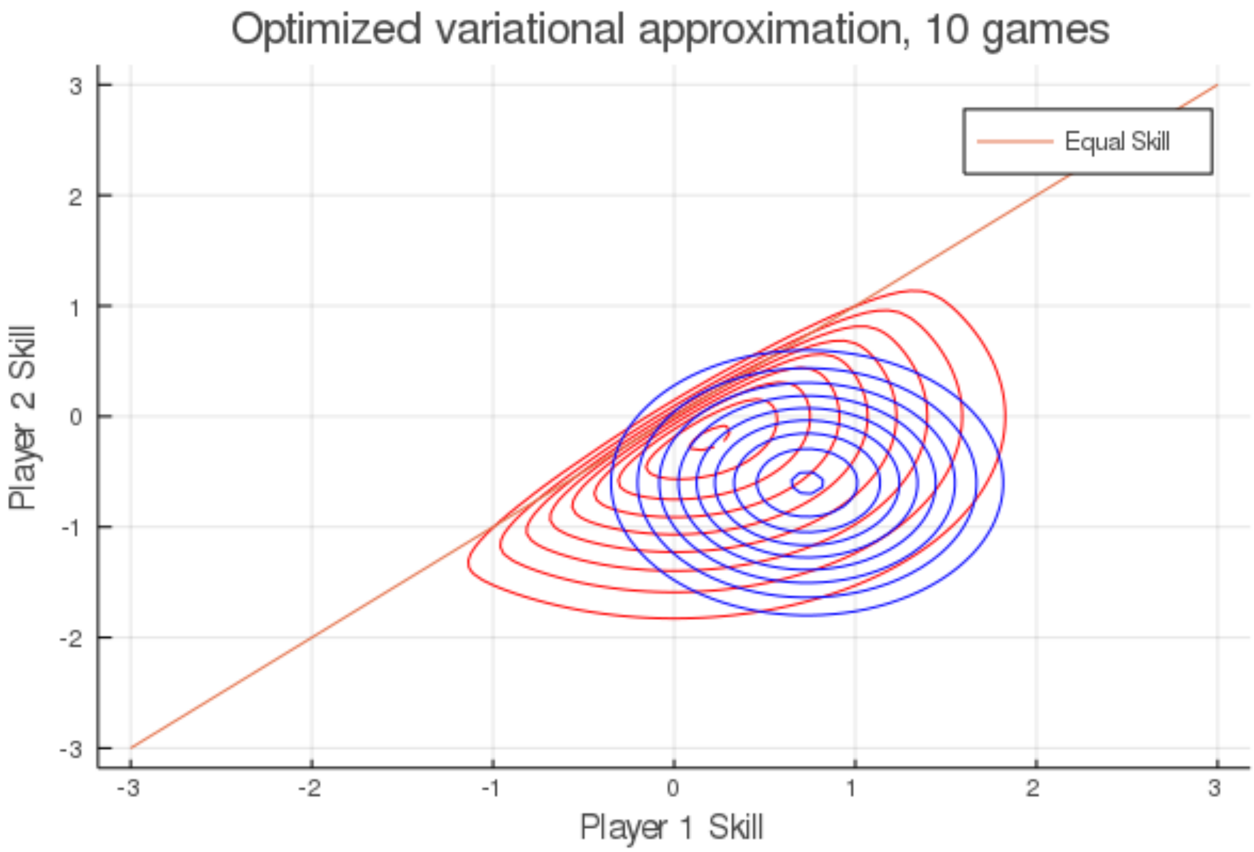
```
## Final loss = 0.73323
```



e. **[2 points]** Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 10 games. Report the final loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
fit_toy_variational_dist(toy_params_init, two_player_toy_games(10, 0))
title!("Optimized variational approximation, 10 games")
xlabel!("Player 1 Skill")
ylabel!("Player 2 Skill")
savefig("fig_game10.png")
```

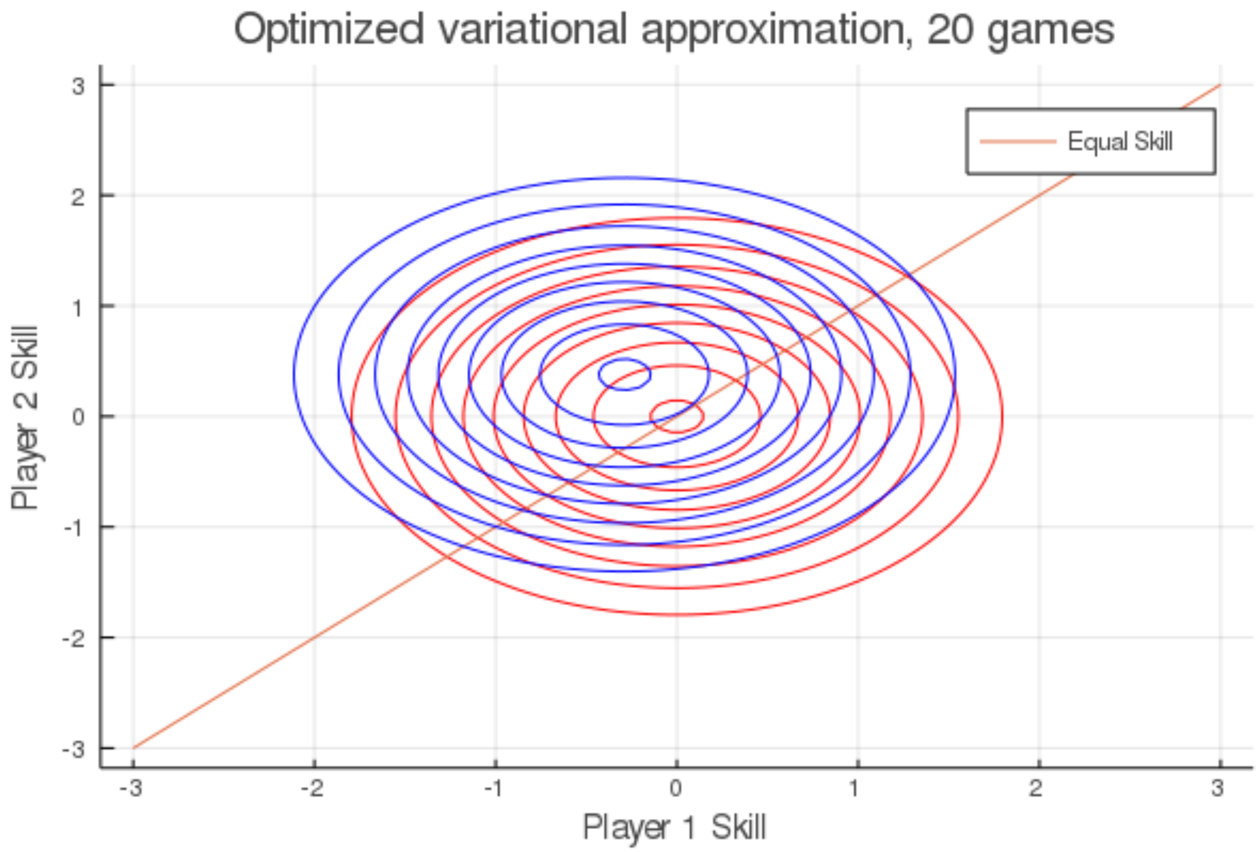
```
## Final loss = 1.09858
```



f. **[2 points]** Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 10 games and player B winning 10 games. Report the final loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
fit_toy_variational_dist(toy_params_init, two_player_toy_games(10, 10))
title!("Optimized variational approximation, 20 games")
xlabel!("Player 1 Skill")
ylabel!("Player 2 Skill")
savefig("fig_game20.png")
```

```
## Final loss = 0.733639
```



4. Approximate inference conditioned on real data [24 points]

a. **[1point]** For any two players i and j, $p(z_i, z_j|\text{all games})$ is always proportional to $p(z_i, z_j, \text{all games})$. In general, are the isocontours of $p(z_i, z_j\ |\text{all games})$ the same as those of $p(z_i, z_j\ |\text{games between i and j})$? That is, do the games between other players besides i and j provide information about the skill of players i and j? A simple yes or no suffices.

Yes, they do. Among others, we have multiple observations of games between i and k or j and k (k is any other player), which will give us some information regarding the skills of i and j.

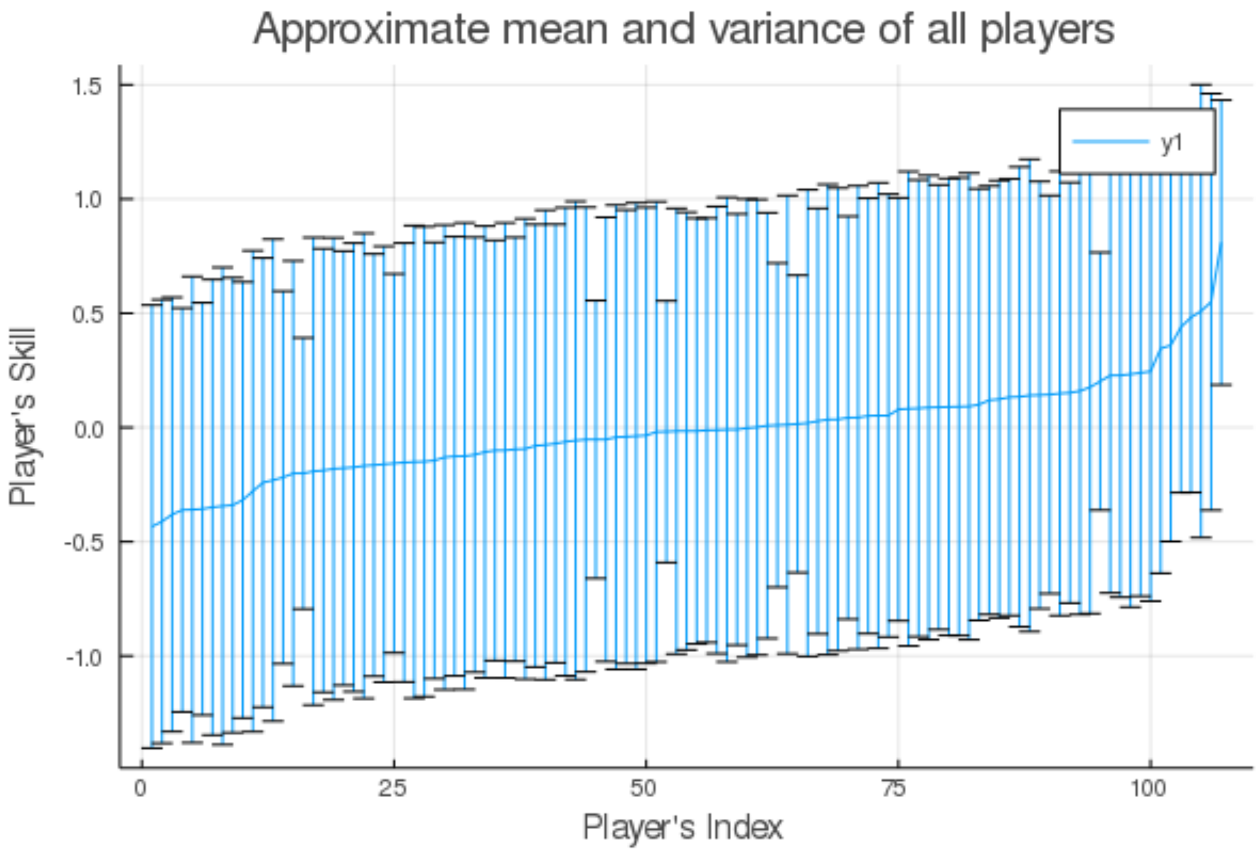
b. **[5 points]** Write a new optimization function fit variational dist like the one from the previous question except it does not plot anything. Initialize a variational distribution and fit it to the joint distribution with all the observed tennis games from the dataset. Report the final negative ELBO estimate after optimization.

```
function fit_variational_dist(init_params, tennis_games; num_itrs=200, lr= 1e-2, num_
q_samples = 10)
  params_cur = init_params
  for i in 1:num_itrs
    grad_params =  gradient(params -> neg_toy_elbo(params, games = tennis_games, num_
samples = num_q_samples),
    params_cur)[1]
    params_cur = params_cur .- lr.*grad_params
    @info "Current ELBO" neg_toy_elbo(params_cur, games = tennis_games, num_samples =
num_q_samples)
  end
  return params_cur
end

#Initialize variational family
init_mu = randn(num_players)
init_log_sigma = randn(num_players)
init_params = (init_mu, init_log_sigma)

# Train variational distribution
trained_params = fit_variational_dist(init_params, tennis_games)
```

```
## Final loss = 3.941687
```

c. **[2 points]** Plot the approximate mean and variance of all players, sorted by skill. For example, in Julia, you can use: **perm = sortperm(means); plot(means[perm], yerror=exp.(logstd[perm]))**. There's no need to include the names of the players.

```
perm = sortperm(trained_params[1])
plot(trained_params[1][perm], yerror=exp.(trained_params[2][perm]))
title!("Approximate mean and variance of all players")
xlabel!("Player's Index")
ylabel!("Player's Skill")
savefig("players_mean_var.png")
```



d. **[2 points]** List the names of the 10 players with the highest mean skill under the variational model.

```
reverse(player_names[perm])[1:10]
```

The output:

“Novak-Djokovic”

“Roger-Federer”

“Rafael-Nadal”

“Andy-Murray”

“Robin-Soderling”

“Kei-Nishikori”

“David-Ferrer”

“Juan-Martin-Del-Potro”

“Mardy-Fish”

“Nicolas-Almagro”

“Jo-Wilfried-Tsonga”

e. **[3 points]** Plot the joint approximate posterior over the skills of Roger Federer and Rafael Nadal. Use the approximate posterior that you fit in question 4 part b.

```
RF_idx = findall(x -> x == "Roger-Federer", player_names)[1][1]
RN_idx = findall(x -> x == "Rafael-Nadal", player_names)[1][1]
print("Roger Federer's index is ", RF_idx, "; Rafael Nadal's index is ", RN_idx, ".")
```

```
## Roger Federer's index is 5 ; Rafael Nadal's index is 1 .
```

```
RF_RN(zs) = exp.(factorized_gaussian_log_density([trained_params[1][RN_idx], trained_
params[1][RF_idx]],[trained_params[2][RN_idx], trained_params[2][RF_idx]],zs))
plot(title="Joint approximate posterior",
    xlabel = "Rafael Nadal",
    ylabel = "Roger Federer");
skillcontour!(RF_RN)
plot_line_equal_skill!()
savefig("RF_RN.png")
```