1 (a)

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

1 (b)

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

1 (c)

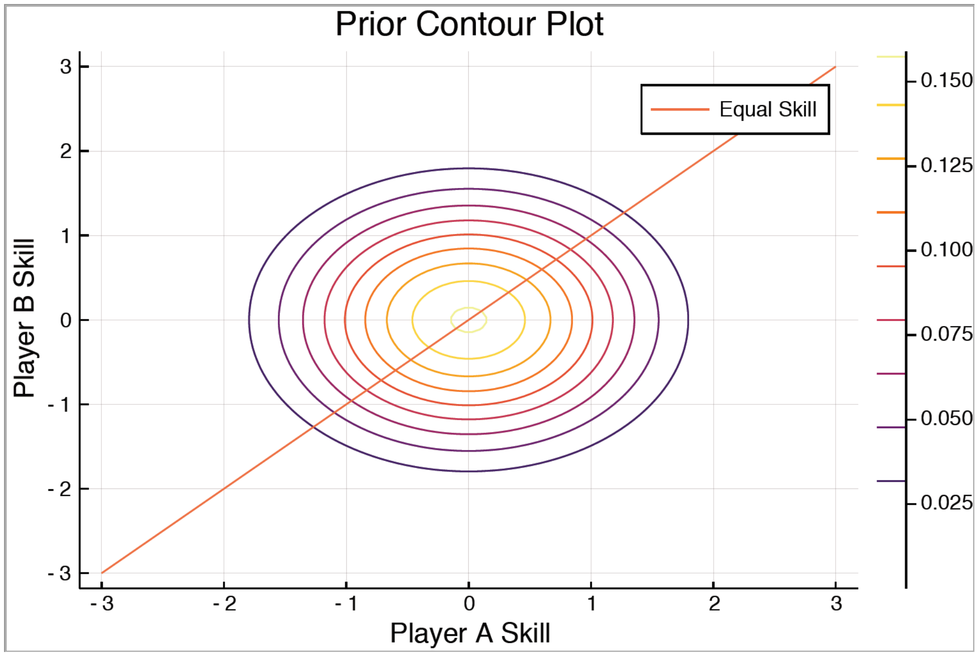
```
function all_games_log_likelihood(zs,games)
  zs_a = zs[games[:,1],:]
  zs_b = zs[games[:,2],:]
  likelihoods = logp_a_beats_b.(zs_a,zs_b)
  return  sum(likelihoods, dims=1)
end
```

1 (d)

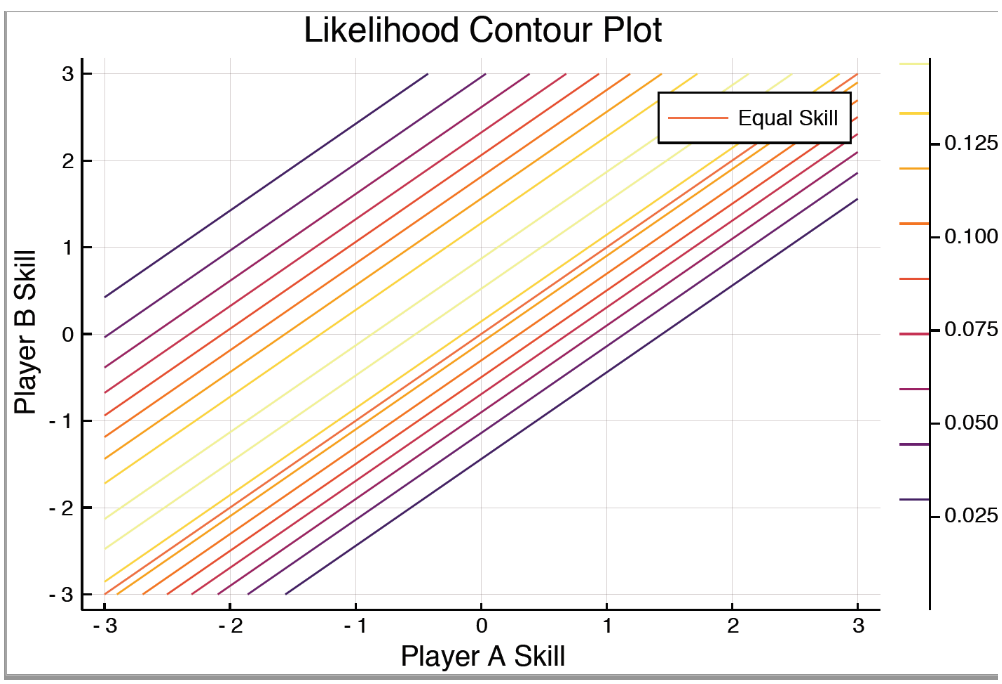
```
function joint_log_density(zs,games)
  return sum(all_games_log_likelihood(zs,games).+log_prior(zs),dims=1)
end
@testset "Test shapes of batches for likelihoods" begin
  B = 15 # number of elements in batch
  N = 4 # Total Number of Players
  test_zs = randn(4,15)
  test_games = [1 2; 3 1; 4 2] # 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
```

```
Test Summary:                          | Pass  Total
Test shapes of batches for likelihoods |    6      6
```

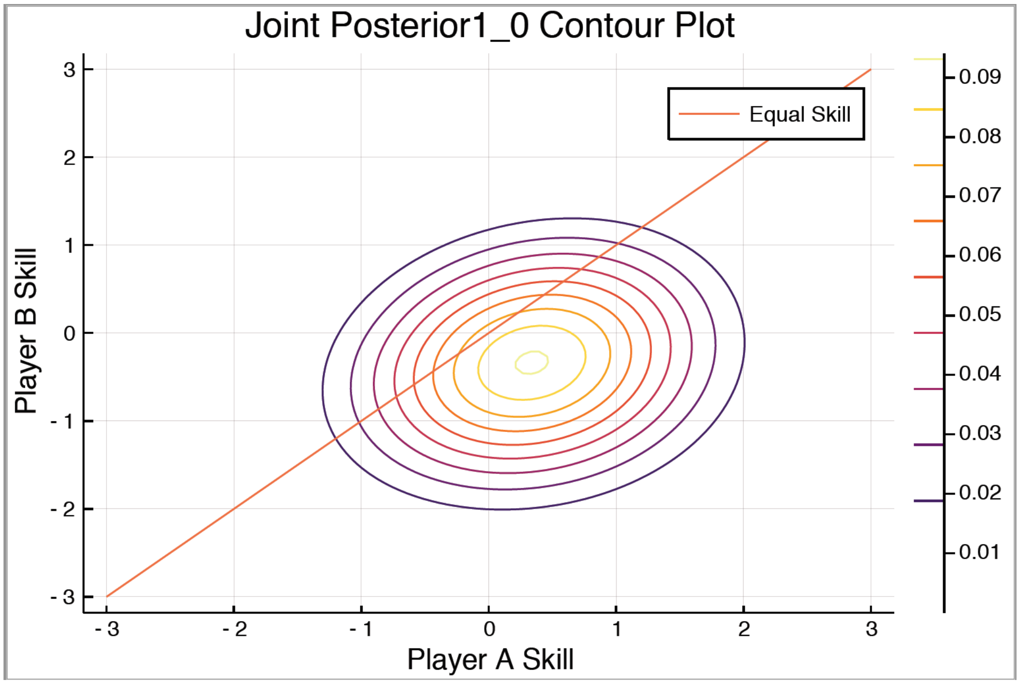
2 (a)


Prior Contour Plot
Equal Skill
Player B Skill
Player A Skill
3
2
1
0
- 1
- 2
- 3
0.150
0.125
0.100
0.075
0.050
0.025

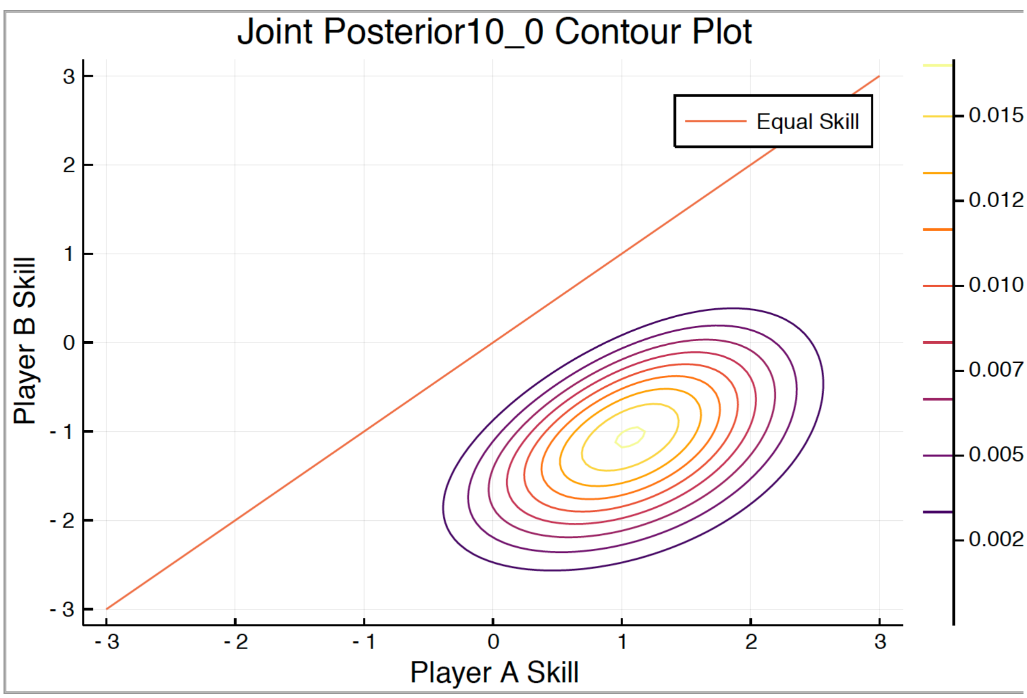
2 (b)


Likelihood Contour Plot
Equal Skill
Player B Skill
Player A Skill
3
2
1
0
- 1
- 2
- 3
0.125
0.100
0.075
0.050
0.025

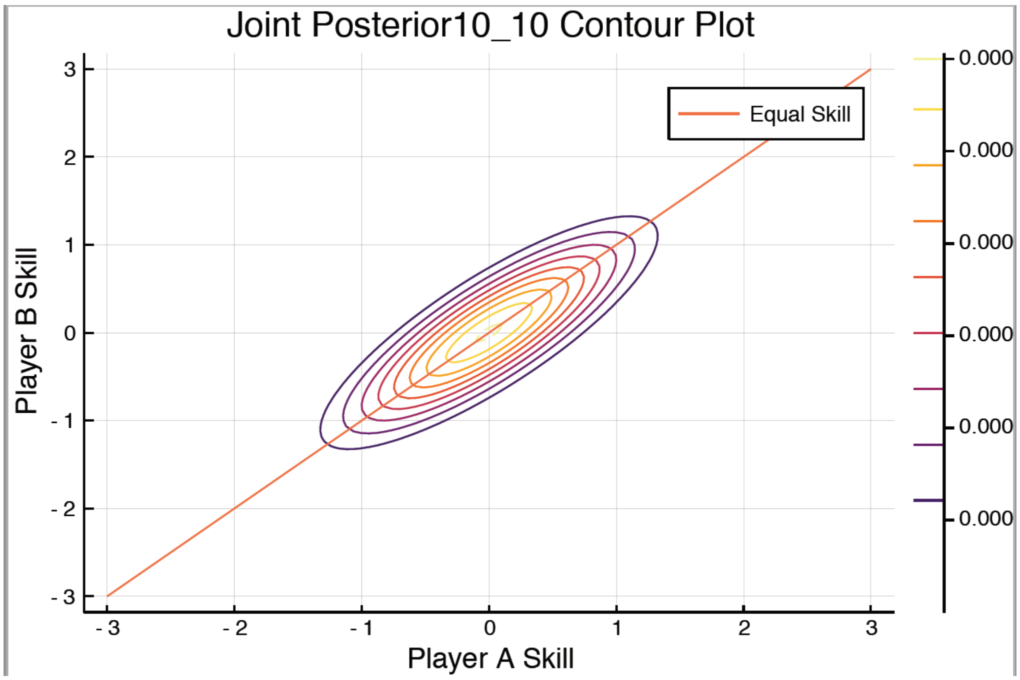
2 (c)


Joint Posterior1_0 Contour Plot
Equal Skill
Player B Skill
Player A Skill
3
2
1
0
- 1
- 2
- 3
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01

2 (d)


Joint Posterior10_0 Contour Plot
Equal Skill
Player B Skill
Player A Skill
3
2
1
0
- 1
- 2
- 3
0.015
0.012
0.010
0.007
0.005
0.002

2 (e)



3 (a)

```
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 sum(logp_estimate-logq_estimate)/num_samples
end
```

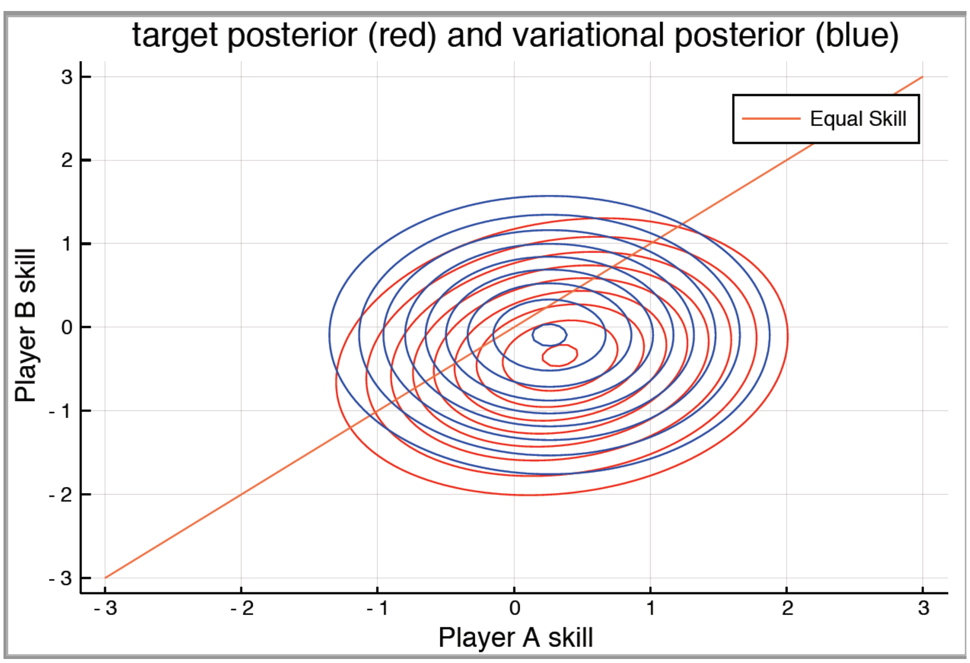
3 (b)

```
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
```

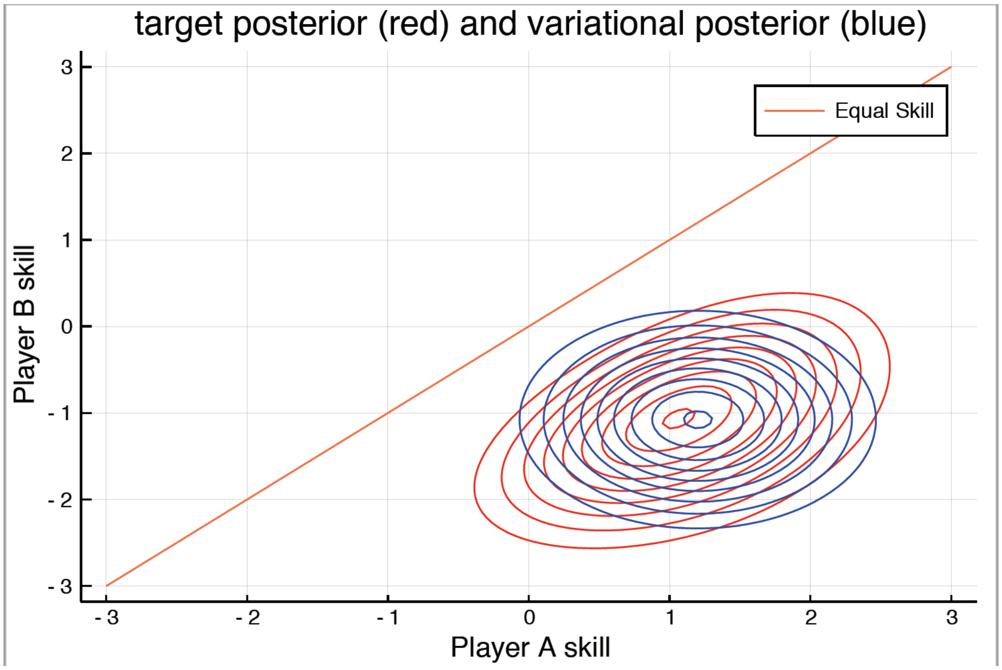
3 (c)

```
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_samples = num_q_samples), params_cur)[1]#gradients of variational objective with respect to parameters
    params_cur =  params_cur .- lr.* grad_params #update paramters with lr-sized step in descending gradient
    @info "nelbo: $(neg_toy_elbo(params_cur; games=toy_evidence, num_samples=num_q_samples))"#report the current elbbo during training
    # plot true posterior in red and variational in blue
    plot();
    #plot likelihood contours for target posterior
    display(skillcontour!(zs->exp(joint_log_density(zs, toy_evidence)),colour=:red))
    plot_line_equal_skill!()
    #plot likelihood contours for variational posterior
    display(skillcontour!(zs -> exp(factorized_gaussian_log_density(params_cur[1], params_cur[2],zs)),colour=:blue))
  end
  return params_cur
end
```

3 (d)

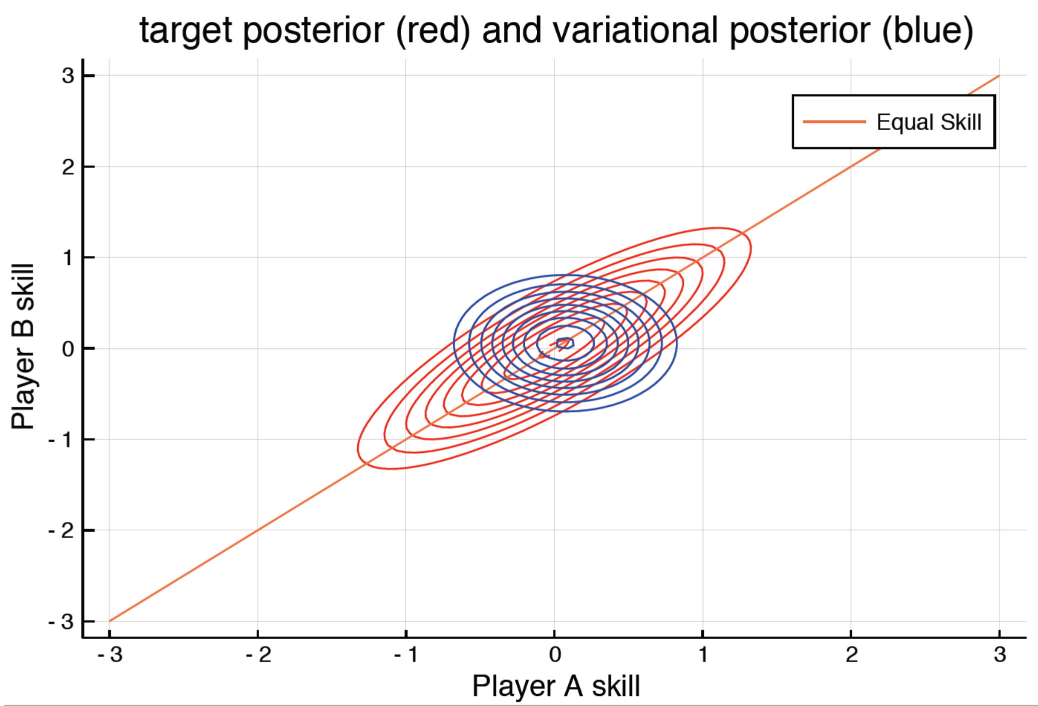


3 (e)



target posterior (red) and variational posterior (blue)
Equal Skill
Player B skill
Player A skill
3
2
1
0
- 1
- 2
- 3
- 3
- 2
- 1
0
1
2
3

3 (f)



target posterior (red) and variational posterior (blue)
Equal Skill
Player B skill
Player A skill
3
2
1
0
- 1
- 2
- 3
- 3
- 2
- 1
0
1
2
3

4 (a)

Question: Are the isocontours of $P(z_i, z_j | \text{all games})$ the same as those of $P(z_i, z_j | \text{games between i and j})$?
Answer: Yes

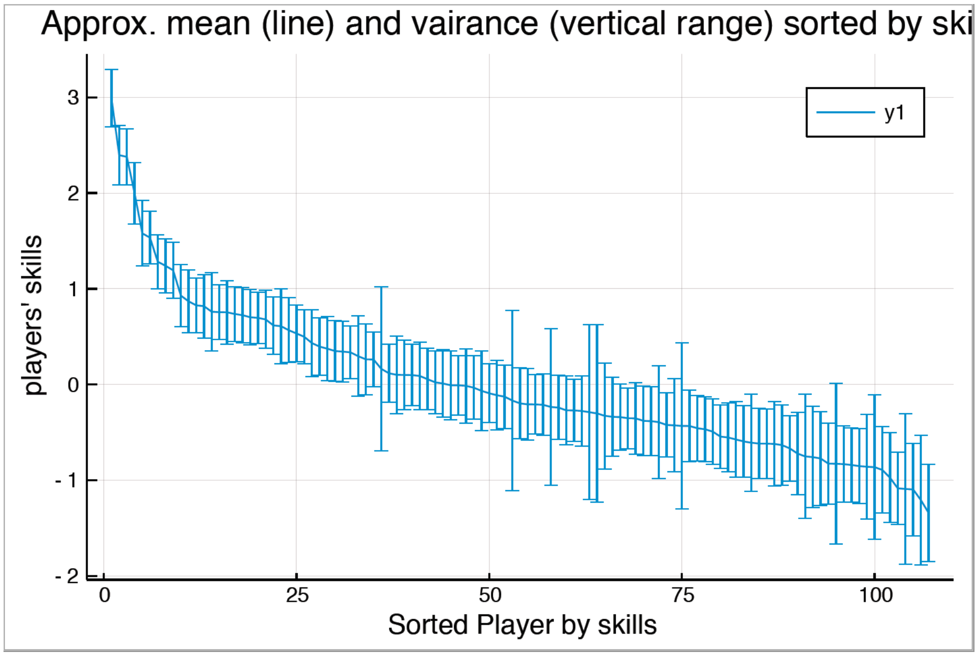
Question: do the games between other players besides i and j provide information about the skill of players i and j?
Answer: No

4(b)

```
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 "nelbo: $(neg_toy_elbo(params_cur; games=tennis_games, num_samples=num_q_samples))"#report objective value with current parameters
  end
  return params_cur
end
init_mu = randn(107)#random initialziation
init_log_sigma = randn(107)# random initialziation
init_params = (init_mu, init_log_sigma)
trained_params = fit_variational_dist(init_params, tennis_games)
```

The final negative ELBO estimate after optimization is 1144.024976806034.

4 (c)

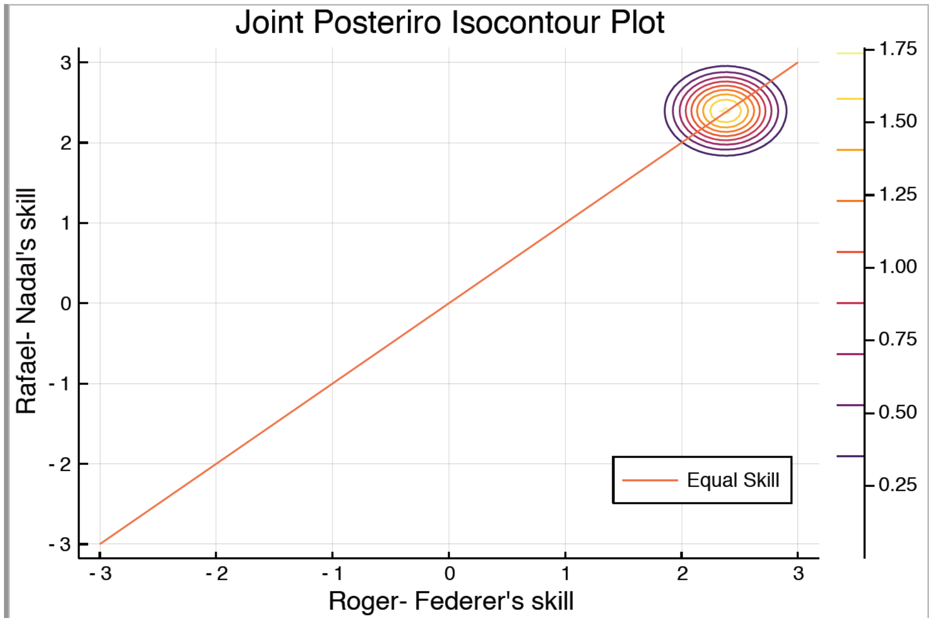


4 (d)

The top 10 players ranked by the highest mean skills are:

Novak-Djokovic
Roger-Federer
Rafael-Nadal
Andy-Murray
Robin-Soderling
David-Ferrer
Jo-Wilfried-Tsonga
Tomas-Berdych
Juan-Martin-Del-Potro
Richard-Gasquet

4 (e)



4 (f)

Since $y_a = z_a - z_b$ and $y_b = z_b$
We can write $Y = AZ$ where $Y = \begin{pmatrix} y_a \\ y_b \end{pmatrix}$, $A = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$ and $Z = \begin{pmatrix} z_a \\ z_b \end{pmatrix}$
By hint 2, since $Z \sim N(\mu_z, \Sigma_z)$ where $\mu_z = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}$ and $\Sigma_z = \begin{pmatrix} \sigma_{aa} & 0 \\ 0 & \sigma_{bb} \end{pmatrix}$, we can get $Y \sim N(A\mu_z, A\Sigma_z A^T)$.

$$A\mu_z = \begin{pmatrix} \mu_a - \mu_b \\ \mu_b \end{pmatrix} \text{ and } A\Sigma_z A^T = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \sigma_{aa} & 0 \\ 0 & \sigma_{bb} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} \sigma_{aa} + \sigma_{bb} & -\sigma_{bb} \\ -\sigma_{bb} & \sigma_{bb} \end{pmatrix}$$

By hint 3, we can get $y_a \sim N(\mu_a + \mu_b, \sigma_{aa} + \sigma_{bb})$

We want to know $P(z_a > z_b) \equiv P(z_a - z_b > 0) \equiv P(y_a > 0)$

We need to use cdf function to get the probability.

1-cdf(Normal(mean, sigma),0)

4 (g)

The exact probability under the approximate posterior that Roger Federer has higher skill than Rafael Nadal is 0.5421.

Calculated by the following code:

```
# P(zRF>zRN)
print(1-cdf(Normal(means[5]-means[1], sqrt((exp(logsd[1]))^2+(exp(logsd[5]))^2)),0))
```

The estimated probability by simple Monte Carlo with 10000 examples is 0.5448.

Calculated by the following code:

```
# P(zRF>zRN)
zs_RF = rand(Normal(means[5], exp(logsd[5])),10000)
zs_RN = rand(Normal(means[1], exp(logsd[1])),10000)
print(sum(zs_RF .> zs_RN,dims=1) / 10000)
```

4 (h)

The exact probability under the approximate posterior that Roger Federer is better than the player with the lowest mean skill is 0.999 ≈ 1.

Calculated by the following code:

```
# P(zRF>zb)
print(1-cdf(Normal(means[5]-means[75], sqrt((exp(logsd[75]))^2+(exp(logsd[5]))^2)),0))
```

The estimated probability by simple Monte Carlo with 10000 examples is 1.

Calculated by the following code:

```
# P(zRF>zb)
lowestindex = perm[107]
zs_RF = rand(Normal(means[5], exp(logsd[5])),10000)
zs_b = rand(Normal(means[75], exp(logsd[75])),10000)
print(sum(zs_RF .> zs_b,dims=1) / 10000)
```

4 (i)

If we change the prior to Normal(10,1), then the posterior will be changed based on the prior. Since “elbo” function is designed based on the prior then “neg_toy_elbo” function and “fit_variational_dist” function will change as well. So all questions related with above functions will change.

Therefore, Q4 b, c, e, g will change.

For Q4 b, the info neg elbo changed if we change the prior distribution.

For Q4 c, the mean of all players will increase by approximate 10.

For Q4 e, the plot will shift to upper right corner.

For Q4 g, the probability increases from 0.54 to 0.59.

For Q4 h, the probability is still 1 since the difference between the lowest player and RF is too large. Hard to see the change.

Appendix

The following code is for plotting the above plots.

```
# Convenience function for producing toy games between two players.
two_player_toy_games(p1_wins, p2_wins) = vcat([repeat([1,2]',p1_wins), repeat([2,1]',p2_wins)]...)

# Example for how to use contour plotting code
#plot(title="Example Gaussian Contour Plot",
#    xlabel = "Player 1 Skill",
#    ylabel = "Player 2 Skill"
#   )
#example_gaussian(zs) = exp(factorized_gaussian_log_density([-1.,2.],[0.,0.5],zs))
#skillcontour!(example_gaussian)
#plot_line_equal_skill!()
#savefig(joinpath("plots","example_gaussian.pdf"))


zs = randn(2,15)
# plot prior contours
plot(title="Prior Contour Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
prior(zs) = exp(log_prior(zs))
skillcontour!(zs -> prior(zs))
plot_line_equal_skill!()
savefig(joinpath("plots","joint_prior.pdf"))

# plot likelihood contours
```

```
game12 = two_player_toy_games(1,2)
plot(title="Likelihood Contour Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
likelihood(zs) = exp(all_games_log_likelihood(zs,game12))
skillcontour!(zs -> likelihood(zs))
plot_line_equal_skill!()
savefig(joinpath("plots","likelihood.pdf"))

# plot joint contours with player A winning 1 game
games = two_player_toy_games(1,0)
plot(title="Joint Posterior1_0 Contour Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint1(zs) = exp(joint_log_density(zs,games))
skillcontour!(zs -> joint1(zs))
plot_line_equal_skill!()
savefig(joinpath("plots","joint1.pdf"))
# plot joint contours with player A winning 10 games
games = two_player_toy_games(10,0)
plot(title="Joint Posterior10_0 Contour Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint10(zs) = exp(joint_log_density(zs,games))
skillcontour!(zs -> joint10(zs))
plot_line_equal_skill!()
savefig(joinpath("plots","joint10.pdf"))

# plot joint contours with player A winning 10 games and player B winning 10 games
games = two_player_toy_games(10,10)
plot(title="Joint Posterior10_10 Contour Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint20(zs) = exp(joint_log_density(zs,games))
skillcontour!(zs -> joint20(zs))
plot_line_equal_skill!()
savefig(joinpath("plots","joint20.pdf"))

#fit q with SVI observing player A winning 1 game
game10 = two_player_toy_games(1,0)
fit_toy_variational_dist(toy_params_init,game10)
xlabel!("Player A skill")
ylabel!("Player B skill")
title!("target posterior (red) and variational posterior (blue)")
savefig(joinpath("plots","posterior 1.pdf"))

#fit q with SVI observing player A winning 10 games
game100 = two_player_toy_games(10,0)
fit_toy_variational_dist(toy_params_init,game100)
xlabel!("Player A skill")
ylabel!("Player B skill")
title!("target posterior (red) and variational posterior (blue)")
savefig(joinpath("plots","posterior 10.pdf"))

#fit q with SVI observing player A winning 10 games and player B winning 10 games
game1010 = two_player_toy_games(10,10)
fit_toy_variational_dist(toy_params_init,game1010)
```

```
xlabel!("Player A skill")
ylabel!("Player B skill")
title!("target posterior (red) and variational posterior (blue)")
savefig(joinpath("plots","posterior 20.pdf"))

#10 players with highest mean skill under variational model
#hint: use sortperm
means = trained_params[1]
logsd = trained_params[2]
perm = sortperm(means, rev=true)
plot(means[perm], yerror = exp.(logsd[perm]))
xlabel!("Sorted Player by skills")
ylabel!("players' skills")
title!("Approx. mean (line) and vairance (vertical range) sorted by skills")
savefig(joinpath("plots","mean_variance under variational model.pdf"))

#joint posterior over "Roger-Federer" and ""Rafael-Nadal""
#hint: findall function to find the index of these players in player_names
indexRF = findall(name-> name == "Roger-Federer", player_names)
indexRN = findall(name-> name == "Rafael-Nadal", player_names)
plot(legend=:bottomright)
skillcontour!(zs -> exp(factorized_gaussian_log_density([means[1], means[5]],[logsd[1],logsd[5]],zs)))
plot_line_equal_skill!()
xlabel!("Roger-Federer skill")
ylabel!("Rafael-Nadal skill")
title!("Joint Posterior Isocontour Plot")
savefig(joinpath("plots","Joint Posterior Isocontour Plot.pdf"))
```