Assignment 2 Stochastic Variational Inference in the TrueSkill Model

Gang Peng # 1002961921

March 15, 2020

The goal of this assignment is to get you familiar with the basics of Bayesian inference in large models with continuous latent variables, and the basics of stochastic variational inference.

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

We are given The prior over each player′s skill is a standard normal distribution, and all player′s skills are a prior independent. That is each z_i is iid N(0,1).

```
using Statistics: mean

function factorized_gaussian_log_density(mu,logsig,xs)
  """
  mu and logsig either same size as x in batch or same as whole batch
  returns a 1 x batchsize array of likelihoods
  """
  σ = exp.(logsig)
  return sum((-1/2)*log.(2π*σ.^2) .+ -1/2 * ((xs .- mu).^2)./(σ.^2),dims=1)
end

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

log_prior (generic function with 1 method)
```

(b) [3 points] Implement a function logp a beats b that, given a pair of skills za and zb evaluates the log-likelihood that player with skill za beat player with skill zb 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.

```
using StatsFuns: log1pexp
function logp_a_beats_b(za,zb)
  return log.(1 ./exp.(log1pexp.(-(za .- zb))))
end
```

```
logp_a_beats_b (generic function with 1 method)
```

(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].

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

```
all_games_log_likelihood (generic function with 1 method)
```

(d) [2 points] Implement a function joint log density which combines the log-prior and log-likelihood of the observations to give p(z1,z2,...,zN,all game outcomes)

Again by given indpendency, the joint density is the product of independent density while the joint log density is the sum of independent density.

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

```
joint_log_density (generic function with 1 method)
```

```julia
using Test
@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
Test.DefaultTestSet("Test shapes of batches for likelihoods", Any[], 6, fal
se)
```

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

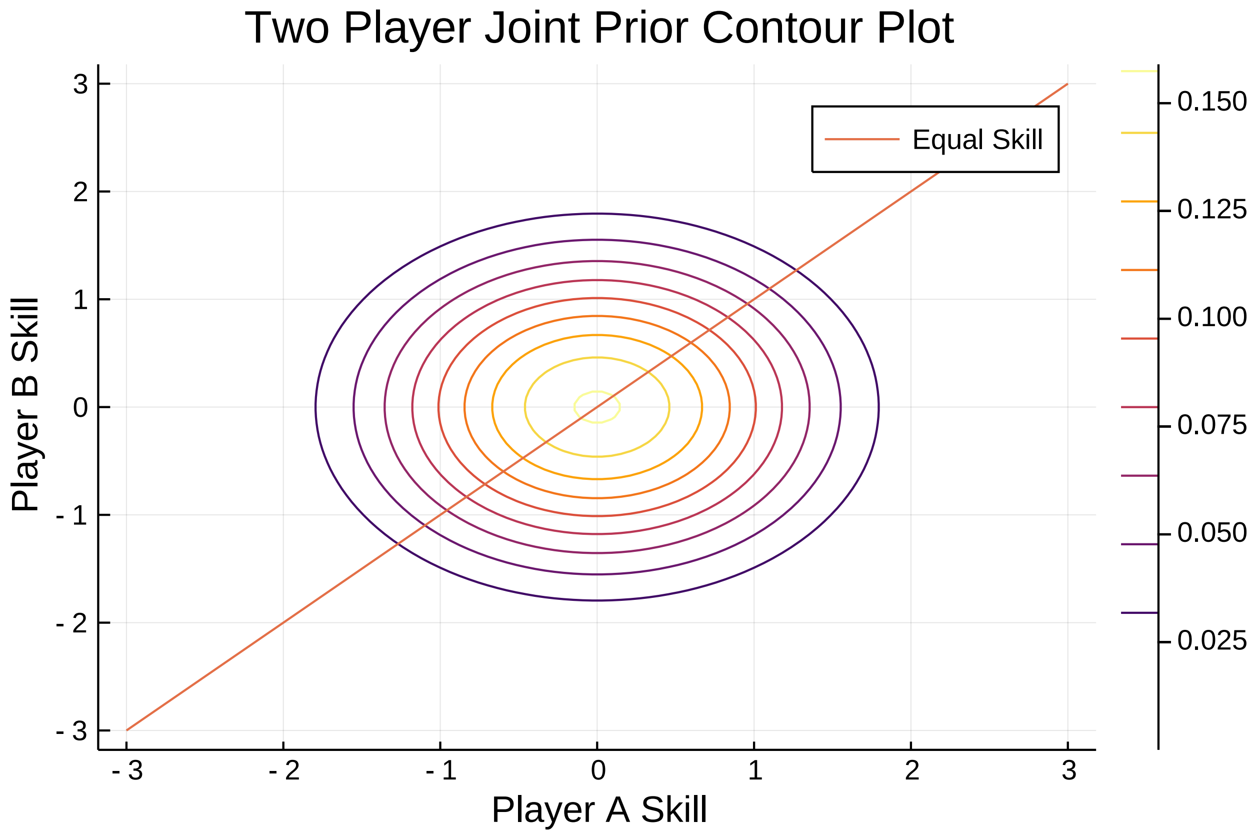
To get a feel for this model, we'll first consider the case where we only have 2 players, A and B. We'll examine how the prior and likelihood interact when conditioning on different sets of games.

Provided in the starter code is a function skillcontour! which evaluates a provided function on a grid of zA and zB's and plots the isocontours of that function. As well there is a function plot line equal skill!. We have included an example for how you can use these functions.

We also provided a function two player toy games which produces toy data for two players. I.e. two player toy games(5,3) produces a dataset where player A wins 5 games and player B wins 3 games.

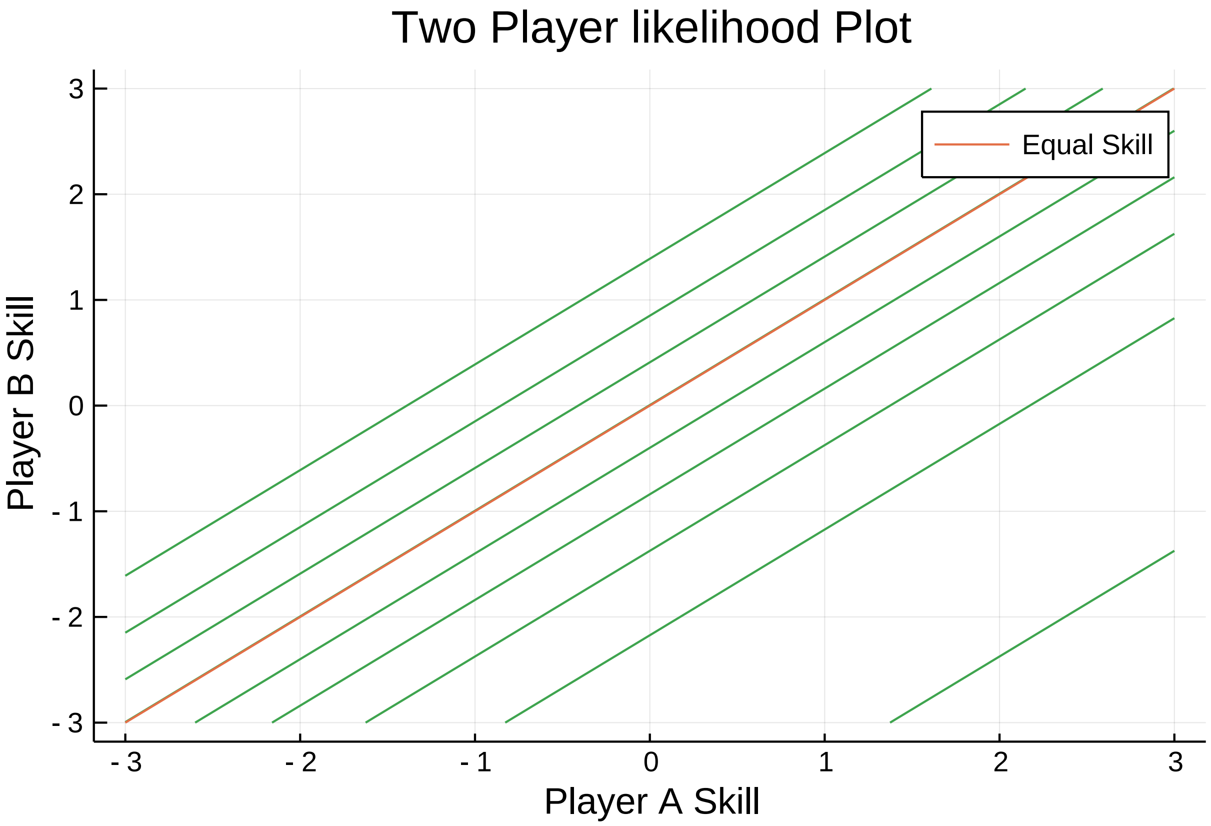
(a) [2 points] For two players A and B, plot the isocontours of the joint prior over their skills. Also plot the line of equal skill, zA = zB. Hint: you've already implemented the log of the likelihood function.

```
using Plots
function skillcontour!(f; colour=nothing)
  n = 100
  x = range(-3,stop=3,length=n)
  y = range(-3,stop=3,length=n)
  z_grid = Iterators.product(x,y) # meshgrid for contour
  z_grid = reshape.(collect.(z_grid),:,1) # add single batch dim
  z = f.(z_grid)
  z = getindex.(z,1)'
  max_z = maximum(z)
  levels = [.99, 0.9, 0.8, 0.7,0.6,0.5, 0.4, 0.3, 0.2] .* max_z
  if colour==nothing
  p1 = contour!(x, y, z, fill=false, levels=levels)
  else
  p1 = contour!(x, y, z, fill=false, c=colour,levels=levels,colorbar=false)
  end
  plot!(p1)
end

function plot_line_equal_skill!()
  plot!(range(-3, 3, length=200), range(-3, 3, length=200), label="Equal Skill")
end

# 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)]...)

jointPrior(zs) = exp.(log_prior(zs))
plot(title="Two Player Joint Prior Contour Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(jointPrior)
plot_line_equal_skill!()
```



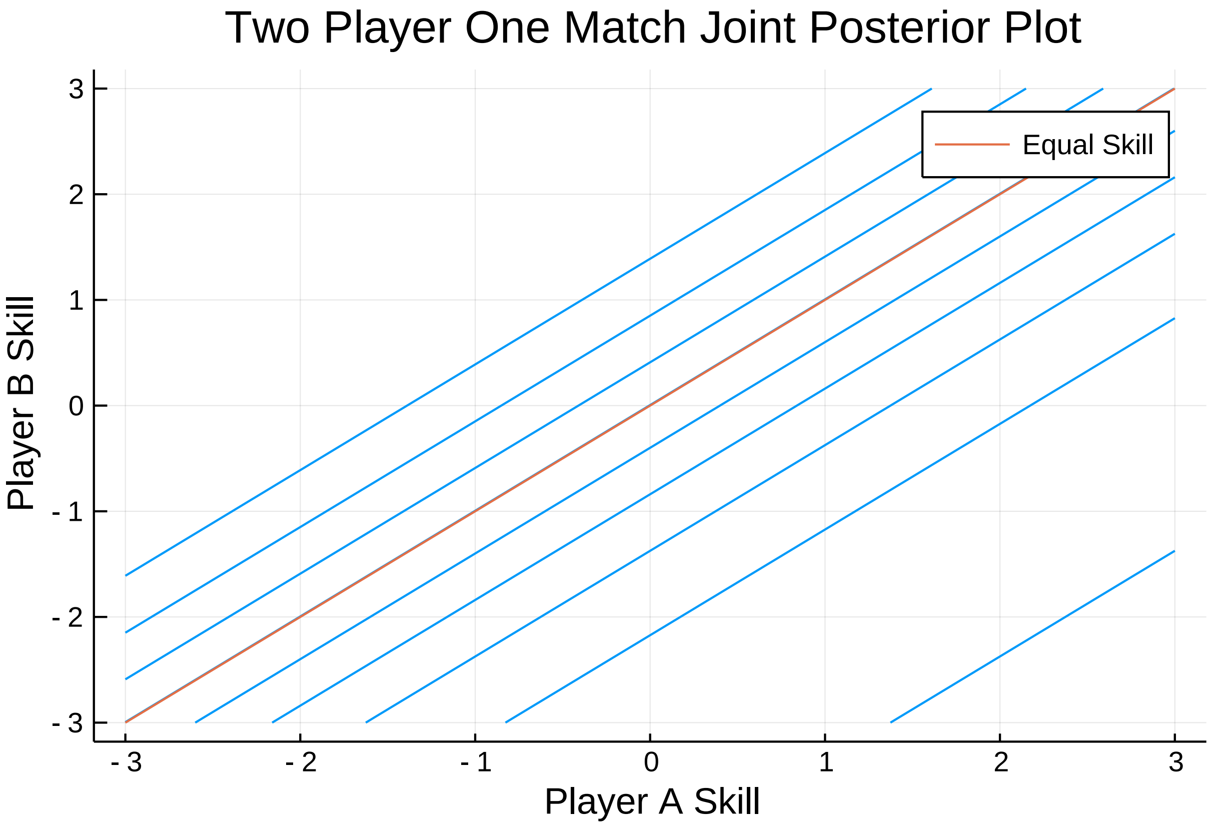
(b) [2 points] Plot the isocontours of the likelihood function. Also plot the line of equal skill, zA = zB.

```
likelihood(zs)=exp.(logp_a_beats_b(zs[1],zs[2]))
plot(title="Two Player likelihood Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(likelihood, colour=3)
plot_line_equal_skill!()
```



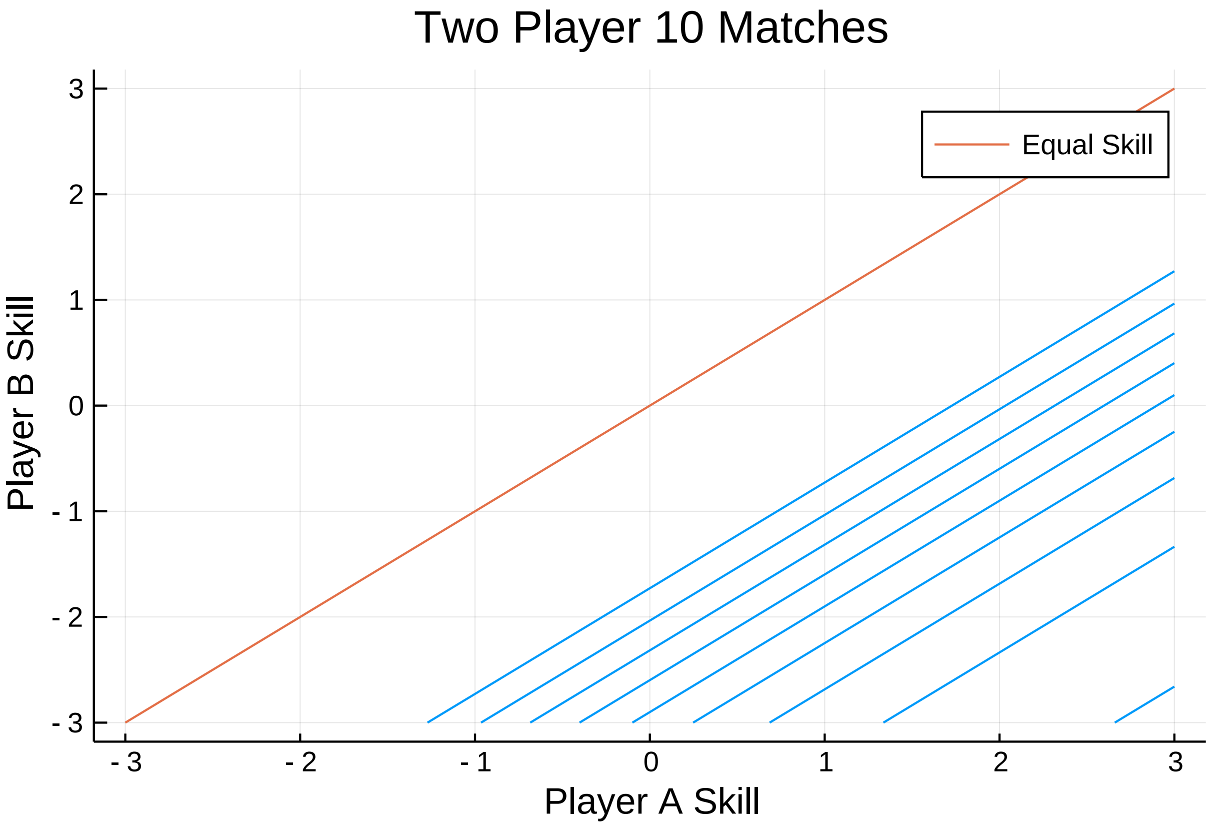
(c) [2 points] Plot isocountours of the joint posterior over zA and zB given that player A beat player B in one match. Since the contours don′t depend on the normalization constant, you can simply plot the isocontours of the log of joint distribution of p(zA,zB,A beat B) Also plot the line of equal skill, zA = zB.

```
games=two_player_toy_games(1, 0)
jt(zs)=exp(all_games_log_likelihood(zs,games))
plot(title="Two Player One Match Joint Posterior Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(jt,colour=1)
plot_line_equal_skill!()
```



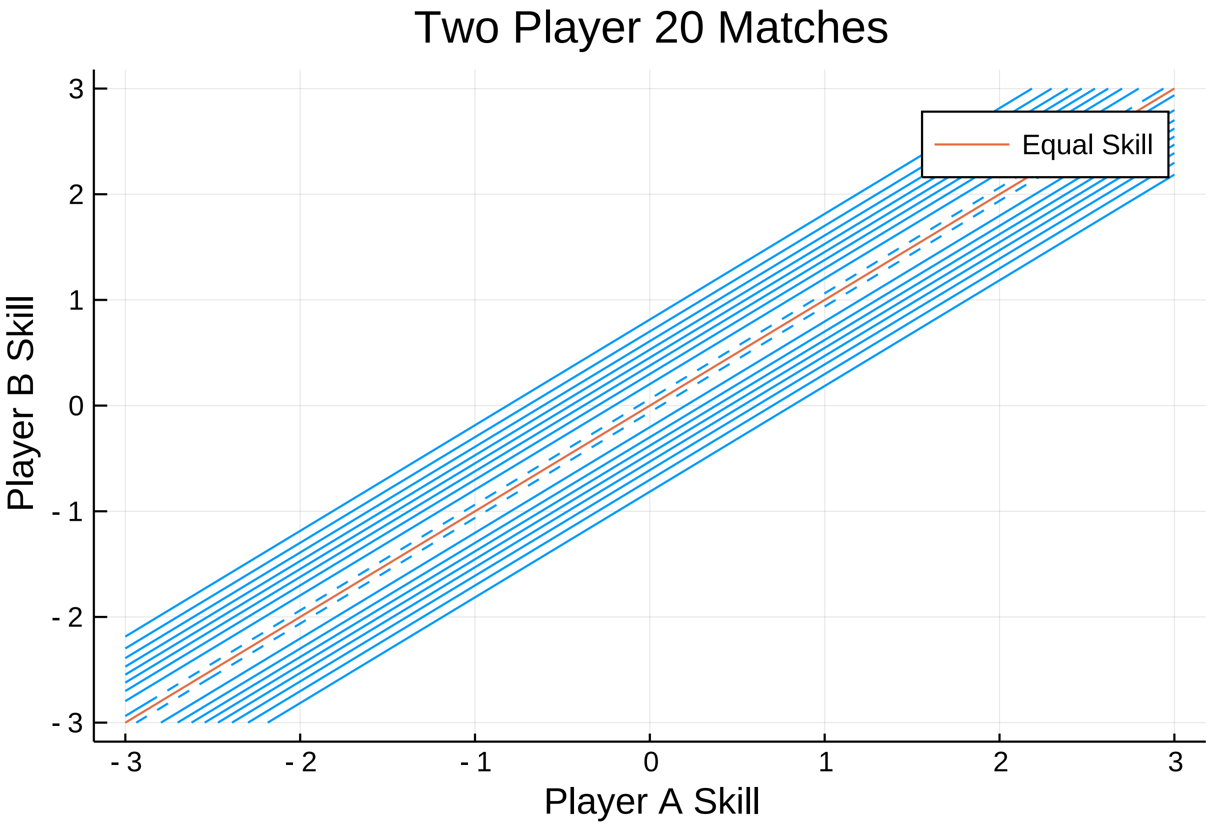
(d) [2 points] Plot isocountours of the joint posterior over zA and zB given that 10 matches were played, and player A beat player B all 10 times. Also plot the line of equal skill, zA = zB.

```
games=two_player_toy_games(10, 0)
jt10(zs)=exp.(all_games_log_likelihood(zs,games))
plot(title="Two Player 10 Matches",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(jt10,colour=1)
plot_line_equal_skill!()
```



(e) [2 points] Plot isocountours of the joint posterior over zA and zB given that 20 matches were played, and each player beat the other 10 times. Also plot the line of equal skill, zA = zB. For all plots, label both axe

```
games=two_player_toy_games(10, 10)
jt20(zs)=exp.(all_games_log_likelihood(zs,games))
plot(title="Two Player 20 Matches",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(jt20,colour=1)
plot_line_equal_skill!()
```



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

One nice thing about a Bayesian approach is that it separates the model speci

cation from the approxi- mate inference strategy. The original Trueskill paper from 2007 used message passing. Carl Rasmussen's assignment uses Gibbs sampling, a form of Markov Chain Monte Carlo. We'll use gradient-based stochastic variational inference, which wasn't invented until around 2014.

In this question we will optimize an approximate posterior distribution with stochastic variational infer- ence to approximate the true posterior.

(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, $q_\Phi(z|data)$, plus an unknown constant. Use a fully-factorized Gaussian distribution for $q_\Phi(z|data)$. This estimator takes the following arguments:

- params, the parameters ϕ of the approximate posterior $q_\Phi(z|data)$.
- A function logp, which is equal to the true posterior plus a constant. This function must take abatch of samples of z. If we have N players, we can consider B-many samples from the joint over

all players' skills. This batch of samples zs will be an array with dimensions (N;B).

- num samples, the number of samples to take.

This function should return a single scalar. Hint: You will need to use the reparamterization trick when sampling zs.

```
function elbo(params,logp,num_samples)
  #Generate random samples from uniform distribution
  U=rand(size(params[1])[1],num_samples)
  #Reparametrization to genearte Gaussian of desired parameter
  zs = sqrt.(-2.0 .* log.(U)) .* cos.(2*pi .* U) .* exp.(params[2]) .+ params[1]
  log_z=factorized_gaussian_log_density(0,0,zs)
  logp_estimate = logp
  log_data=logp .- log_z #Separate data from logp
  #estimate $q_{Φ}(z|data)$
  logq_estimate = factorized_gaussian_log_density(params[1],params[2],zs) .+ log_data
  return sum(logp_estimate .- logq_estimate) ./ num_samples
end

elbo (generic function with 1 method)
```

(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)
  zs=randn(size(params[1])[1],num_samples)
  logp = joint_log_density(zs,games)
  return -elbo(params,logp, num_samples)
end

neg_toy_elbo (generic function with 1 method)
```

(c) [5 points] Write an optimization function called fit toy variational dist which takes initial vari- ational parameters, and the evidence. Inside it will perform a number of iterations of gradient descent where for each iteration :

(a) Compute the gradient of the loss with respect to the parameters using automatic differentiation.

(b) Update the parameters by taking an lr-scaled step in the direction of the descending gradient.

(c) Report the loss with the new parameters (using @info or print statements)

(d) On the same set of axes plot the target distribution in red and the variational approximation in blue. Return the parameters resulting from training.

```
using Zygote
using Logging

function fit_toy_variational_dist(init_params, toy_evidence; num_itrs=200, lr= 1e-2,
num_q_samples = 10)
  params_cur = init_params
  elbo_val = neg_toy_elbo(params_cur; games = toy_evidence, num_samples = num_q_samples)
  #Generate true prior
  pzs=randn(size(init_params[1])[1],num_q_samples)
  jointp(pzs)=exp.(joint_log_density(pzs,toy_evidence)) #function for contour plot

  #Initialize plot, comment out during compiling jmd
  #plot(title="Fit Toy Variational Dist",
  #    xlabel = "Player A Skill",
  #    ylabel = "Player B Skill"
  #   )

  for i in 1:num_itrs
    f(params) = neg_toy_elbo(params; games = toy_evidence, num_samples = num_q_samples)
    grad_params = gradient(f, params_cur)[1]
    params_cur =  params_cur .- grad_params .* lr
    elbo_val = neg_toy_elbo(params_cur; games = toy_evidence, num_samples = num_q_samples)
    #@info "loss: $(elbo) "
    #Note: the following code do the required ploting but comment out during compile
jmd to avoid too much plots in the final file
    #U=rand(size(init_params[1])[1],num_q_samples)
    #qzs = sqrt.(-2.0 .* log.(U)) .* cos.(2*pi .* U) .* exp.(params_cur[2]) .+
params_cur[1]
    #jointq(qzs)=exp.(factorized_gaussian_log_density(params_cur[1],params_cur[2],qzs) .+
    #all_games_log_likelihood(qzs,toy_evidence))
    #display(skillcontour!(jointq,colour=1))
  end
  #plot_line_equal_skill!()

  return params_cur, elbo_val
end
```

```
fit_toy_variational_dist (generic function with 1 method)
```

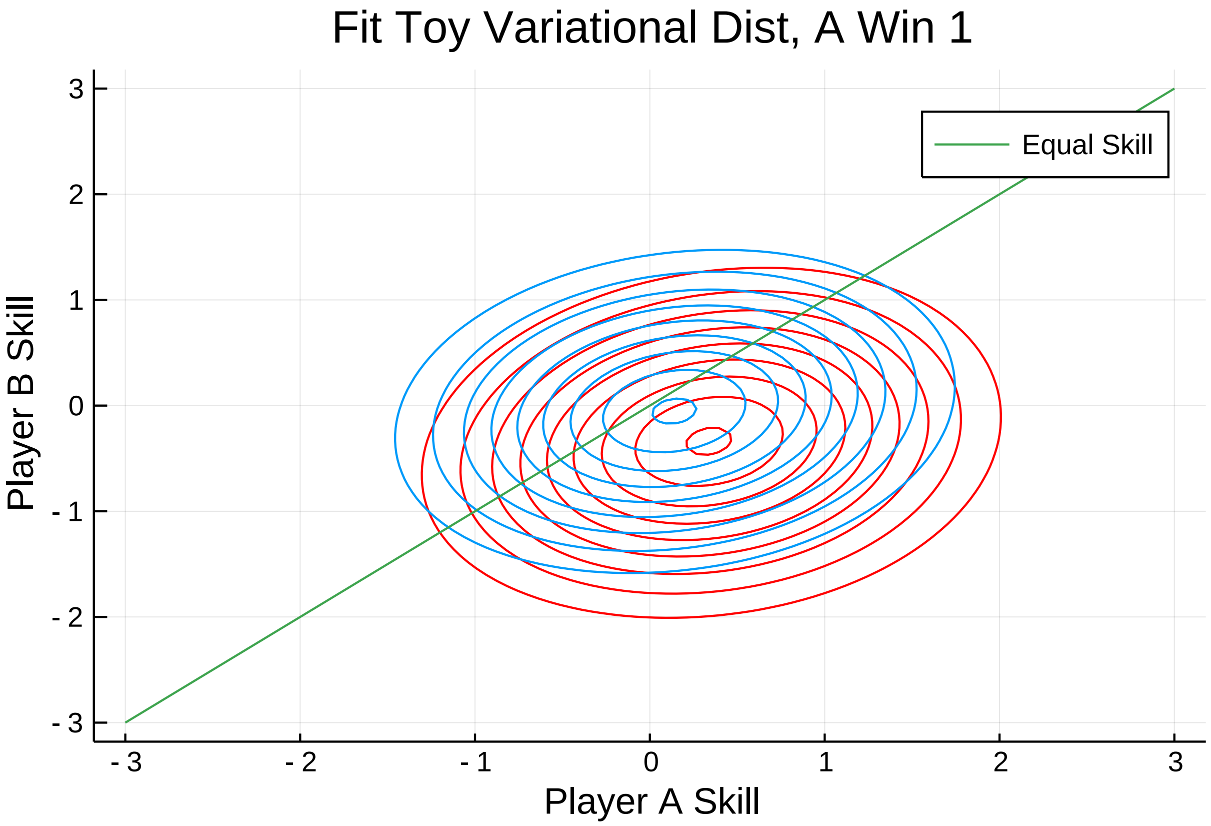
(d) [2 points] Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 1 game. Report the

final loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
num_players_toy = 2
toy_mu = [-2.,3.] # Initial mu, can initialize randomly!
toy_ls = [0.5,0.] # Initual log_sigma, can initialize randomly!
toy_params_init = (toy_mu, toy_ls)
toy_evidence=two_player_toy_games(1,0)
fit=fit_toy_variational_dist(toy_params_init, toy_evidence; num_itrs=200, lr= 1e-2,
num_q_samples = 10)
opt_params=fit[1]
pzs=randn(size(toy_params_init[1])[1],10)
jointp(pzs)=exp.(joint_log_density(pzs,toy_evidence)) #function for contour plot
U=rand(size(toy_params_init[1])[1],10)
qzs = sqrt.(-2.0 .* log.(U)) .* cos.(2*pi .* U) .* exp.(opt_params[2]) .+ opt_params[1]
jointq(qzs)=exp.(factorized_gaussian_log_density(opt_params[1],opt_params[2],qzs) .+
all_games_log_likelihood(qzs,toy_evidence))

# plot result
print("Final loss:",fit[2])
```

```
Final loss:-0.17031184800645582
```

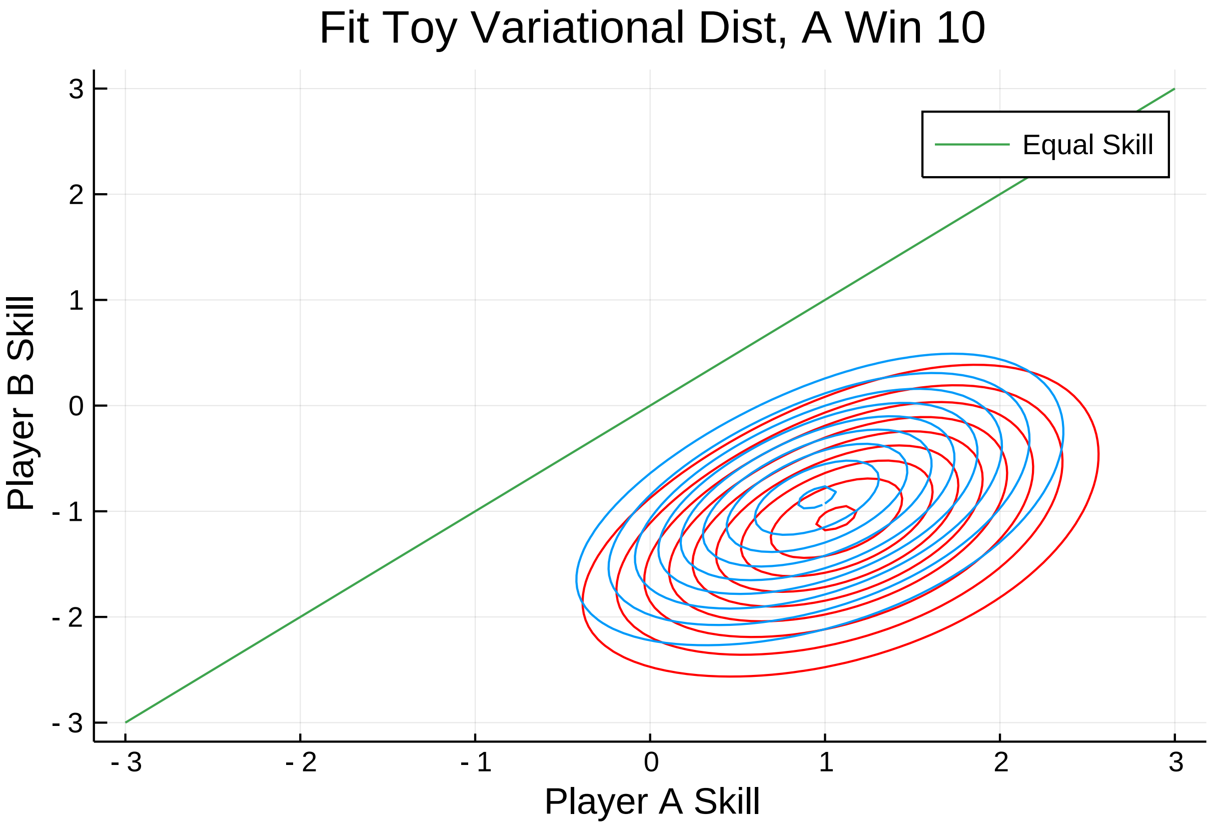
```
plot(title="Fit Toy Variational Dist, A Win 1",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(jointp,colour="red")
skillcontour!(jointq,colour=1)
plot_line_equal_skill!()
```



(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

nal loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
num_players_toy = 2
toy_mu = [-2.,3.] # Initial mu, can initialize randomly!
toy_ls = [0.5,0.] # Initual log_sigma, can initialize randomly!
toy_params_init = (toy_mu, toy_ls)
toy_evidence=two_player_toy_games(10,0)
fit=fit_toy_variational_dist(toy_params_init, toy_evidence; num_itrs=200, lr= 1e-2,
num_q_samples = 10)
opt_params=fit[1]
pzs=randn(size(toy_params_init[1])[1],10)
jointp(pzs)=exp.(joint_log_density(pzs,toy_evidence)) #function for contour plot
U=rand(size(toy_params_init[1])[1],10)
qzs = sqrt.(-2.0 .* log.(U)) .* cos.(2*pi .* U) .* exp.(opt_params[2]) .+ opt_params[1]
jointq(qzs)=exp.(factorized_gaussian_log_density(opt_params[1],opt_params[2],qzs) .+
all_games_log_likelihood(qzs,toy_evidence))

# plot result
print("Final loss:",fit[2])

Final loss:0.4459762929383396

plot(title="Fit Toy Variational Dist, A Win 10",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(jointp,colour="red")
skillcontour!(jointq,colour=1)
plot_line_equal_skill!()
```

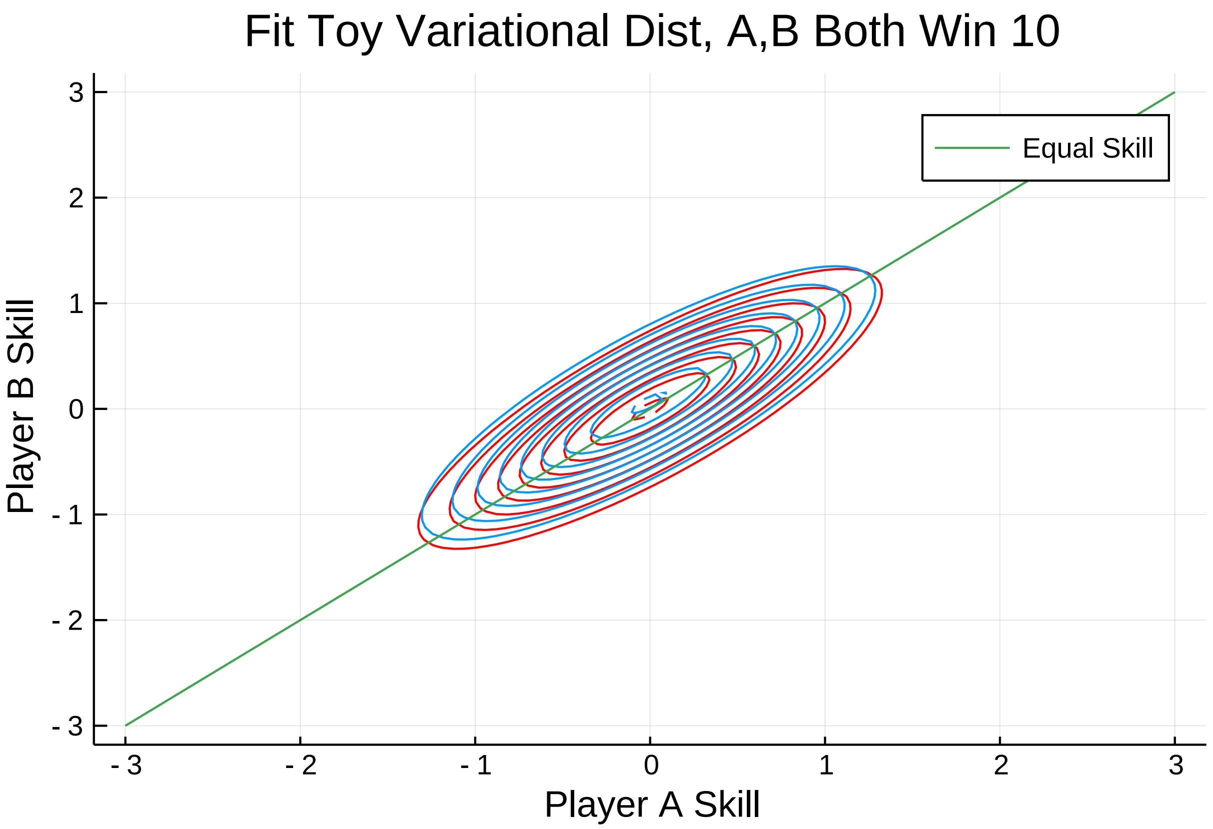


(f) [2 points] Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 10 games and player B winning 10 games. Report the

nal loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
num_players_toy = 2
toy_mu = [-2.,3.] # Initial mu, can initialize randomly!
toy_ls = [0.5,0.] # Initual log_sigma, can initialize randomly!
toy_params_init = (toy_mu, toy_ls)
toy_evidence=two_player_toy_games(10,10)
fit=fit_toy_variational_dist(toy_params_init, toy_evidence; num_itrs=200, lr= 1e-2,
num_q_samples = 10)
opt_params=fit[1]
pzs=randn(size(toy_params_init[1])[1],10)
jointp(pzs)=exp.(joint_log_density(pzs,toy_evidence)) #function for contour plot
U=rand(size(toy_params_init[1])[1],10)
qzs = sqrt.(-2.0 .* log.(U)) .* cos.(2*pi .* U) .* exp.(opt_params[2]) .+ opt_params[1]
jointq(qzs)=exp.(factorized_gaussian_log_density(opt_params[1],opt_params[2],qzs) .+
all_games_log_likelihood(qzs,toy_evidence))

# plot result
print("Final loss:",fit[2])

Final loss:0.07697729601796191

plot(title="Fit Toy Variational Dist, A,B Both Win 10",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
skillcontour!(jointp,colour="red")
skillcontour!(jointq,colour=1)
```

```
plot_line_equal_skill!()
```



4 Approximate inference conditioned on real data [24 points]

Load the dataset from tennis data.mat containing two matrices:

- W is a 107 by 1 matrix, whose i'th entry is the name of player i.
- G is a 1801 by 2 matrix of game outcomes (actually tennis matches), one row per game. The first column contains the indices of the players who won. The second column contains the indices of the player who lost.

Compute the following using your code from the earlier questions in the assignment, but conditioning on the tennis match outcomes:

(a) [1 point] For any two players i and j, $p(z_i, z_j|$ all games) is always proportional to $p(z_i, z_j$,all games). In general, are the isocontours of p($z_i, z_j|$ all games) the same as those of p($z_i, z_j|$ 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.

Hint: One way to answer this is to draw the graphical model for three players, i, j, and k, and the results of games between all three pairs, and then examine conditional independencies. If you do this, there's no need to include the graphical models in your assignment.

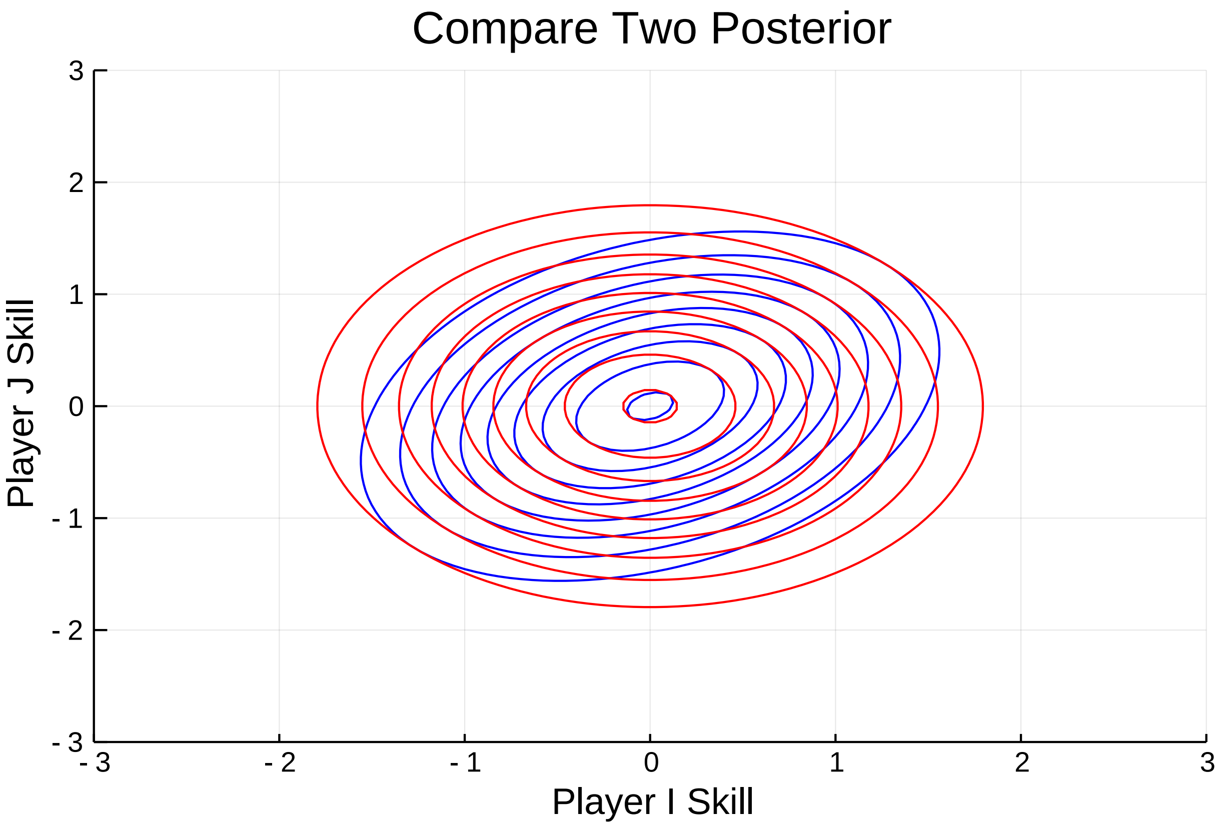
Answer: No, the two posterior are not the same.

For the comparison I used data of three players, i, j, and k. The game records are i and j both win once but i win k twice and k beat j. The posterior based on all games is not the same as the posterior only based on games bewteen i and j.

```
using MAT
vars = matread("tennis_data.mat")
player_names = vars["W"]
tennis_games = Int.(vars["G"])
num_players = length(player_names)
print("Loaded data for $num_players players")
```

```
Loaded data for 107 players
```

```
three_player_games=vcat([1,2]',[2,1]',[1,3]',[1,3]',[3,2]')
pair_games=vcat([1,2]',[2,1]')

zs=randn(3,10)
z2=zs[1:2,:]
all_games=all_games_log_likelihood(zs,three_player_games)
jointp3(zs)=exp.(log_prior(zs) .+all_games)
jointp2(z2)=exp.(log_prior(z2) .+ all_games_log_likelihood(z2,pair_games))

plot(title="Compare Two Posterior",
    xlabel = "Player I Skill",
    ylabel = "Player J Skill"
   )
skillcontour!(jointp2,colour="blue")
skillcontour!(jointp3,colour="red")
```

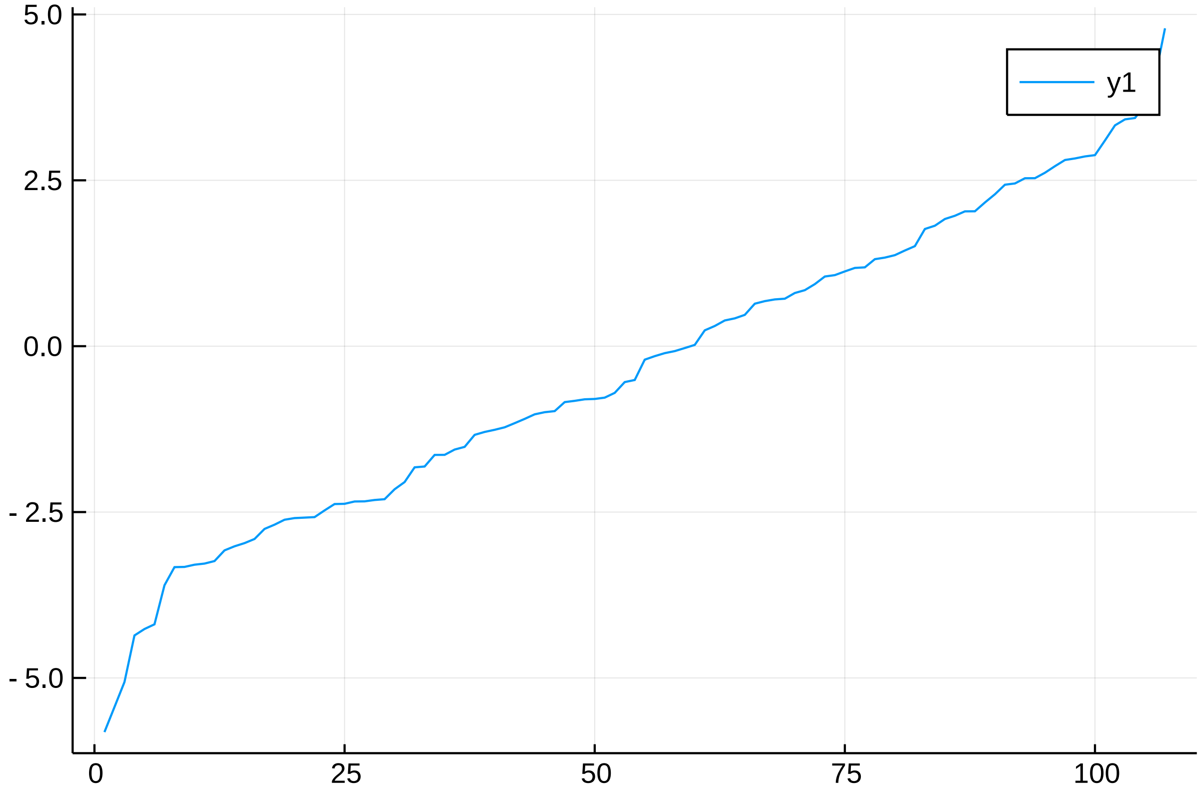


(b) [5 points] Write a new optimization function fit variational dist like the one from the previous question except it does not plot anything. Initialize a variational distribution and fit it to the joint distribution with all the observed tennis games from the dataset. Report the nal negative ELBO 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
  elbo_val=neg_toy_elbo(params_cur; games = tennis_games, num_samples = num_q_samples)
  for i in 1:num_itrs
    f(params)=neg_toy_elbo(params; games = tennis_games, num_samples = num_q_samples)
    grad_params = gradient(f, params_cur)[1]
    params_cur =  params_cur .- grad_params .* lr
    elbo_val=neg_toy_elbo(params_cur; games = tennis_games, num_samples = num_q_samples)
    @info "loss: $(elbo_val) "
  end
  return params_cur, elbo_val
end

num_q_samples = 10
init_mu = randn(num_players, num_q_samples)
init_log_sigma = randn(num_players, num_q_samples)
init_params = (init_mu, init_log_sigma)

# Train variational distribution
trained_params = fit_variational_dist(init_params, tennis_games,lr= 1e-2)
print("Final negative ELBO:", trained_params[2])

Final negative ELBO:67.45408209076622
```

(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.

```
opt_params=trained_params[1]
means=vec(sum(opt_params[1], dims=2))
perm = sortperm(means)
plot(means[perm])
```



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

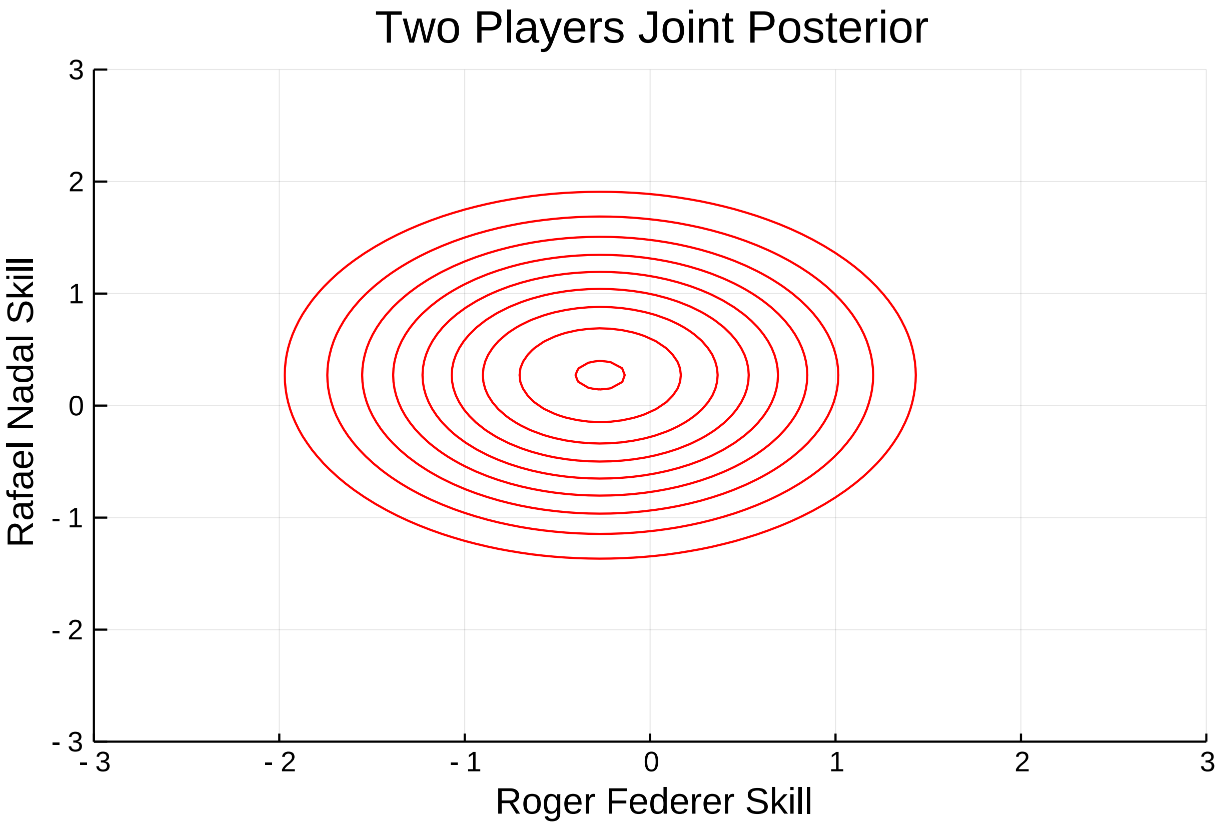
```
top10=perm[98:107]
top10names=[]
for i in 1:10
  push!(top10names,player_names[top10[11-i]] )
end
print(top10names)
```

```
Any["Andy-Murray", "Rohan-Bopanna", "Bernard-Tomic", "Juan-Martin-Del-Potro
", "Robert-Lindstedt", "Novak-Djokovic", "Colin-Fleming", "Ivan-Dodig", "Mi
chael-Llodra", "Ernests-Gulbis"]
```

(e) [3 points] Plot the joint posterior over the skills of Roger Federer and Rafael Nadal.

Here I ploted the posterior conditioned on all the games.

```
meanp1=mean(opt_params[1][1,:])
logsp1=mean(opt_params[2][1,:])
meanp2=mean(opt_params[1][5,:])
logsp2=mean(opt_params[2][5,:])
meanp=vcat(meanp1,meanp2)
logsp=vcat(logsp1,logsp2)
U=rand(2,10)
z2 = sqrt.(-2.0 .* log.(U)) .* cos.(2*pi .* U) .* exp.(logsp) .+ meanp
zs = randn(107,10)
jointp(z2)=exp.(factorized_gaussian_log_density(meanp,logsp,z2) .+
exp.(all_games_log_likelihood(zs, tennis_games)))
plot(title="Two Players Joint Posterior",
    xlabel = "Roger Federer Skill",
    ylabel = "Rafael Nadal Skill"
   )
skillcontour!(jointp,colour="red")
```



(f) [5 points] Derive the exact probability under a factorized Guassian over two players' skills that one has higher skill than the other, as a function of the two means and variances over their skills.

- Hint 1: Use a linear change of variables yA; yB = zA - zB, zB. What does the line of equal skill look like after this transformation?
- Hint 2: If $X \sim N(\mu, \Sigma)$, then AX $\sim N(A\mu, A^T \Sigma A)$ where A is a linear transformation.
- Hint 3: Marginalization in Gaussians is easy: if X $\sim N(\mu, \Sigma)$, then the ith element of X has a marginal distribution $X_i \sim N(\mu_i, \Sigma_{ii})$

(g) [2 points] Compute the probability under your approximate posterior that Roger Federer has higher skill than Rafael Nadal. Compute this quantity exactly, and then estimate it using simple Monte Carlo with 10000 examples.

(h) [2 points] Compute the probability that Roger Federer is better than the player with the lowest mean skill. Compute this quantity exactly, and then estimate it using simple Monte Carlo with 10000 examples.

(i) [2 points] Imagine that we knew ahead of time that we were examining the skills of top tennis players, and so changed our prior on all players to Normal(10, 1). Which answers in this section would this change? No need to show your work, just list the letters of the questions whose answers would be different in expectation.