Assignment 2

Silun Chen 1003163774

March 20, 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.

Background We′ll implement a variant of the TrueSkill model, a player ranking system for competitive games originally developed for Halo 2. It is a generalization of the Elo rating system in Chess. For the curious, the original 2007 NIPS paper introducing the trueskill paper can be found here: http://papers. nips.cc/paper/3079-trueskilltm-a-bayesian-skill-rating-system.pdf This assignment is based on one developed by Carl Rasmussen at Cambridge for his course on probabilistic machine learning: http://mlg.eng.cam.ac.uk/teaching/4f13/1920/

1 Model definition

We′ll consider a slightly simplified version of the original trueskill model. We assume that each player has a true, but unknown skill zi ∈ R. We use N to denote the number of players. The prior. The prior over each player′s skill is a standard normal distribution, and all player′s skills are a prior independent. The likelihood. For each observed game, the probability that player i beats player j, given the player′s skills zA and zB, is: where

$$p(AbeatB|zA, zB) = \sigma(zi - zj)\sigma(y) = \frac{1}{1 + exp(-y)}$$

There can be more than one game played between a pair of players, and in this case the outcome of each game is independent given the players′ skills. We use M to denote the number of games. The data. The data will be an array of game outcomes. Each row contains a pair of player indices. The first index in each pair is the winner of the game, the second index is the loser. If there were M games played, then the array has shape M × 2.

2 Implementing the model [10 points]

1. [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.

```
include("A2_src.jl")
# using Revise # lets you change A2funcs without restarting julia!
# includet("A2_src.jl")
using Plots
using Statistics: mean
using Zygote
using Test
using Logging
using .A2funcs: log1pexp # log(1 + exp(x)) stable
using .A2funcs: factorized_gaussian_log_density
using .A2funcs: skillcontour!
using .A2funcs: plot_line_equal_skill!

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

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

2. [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.

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

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

3. [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(zs_a, zs_b)
  return sum(likelihoods, dims = 1)
end
```

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

4. [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)

```
function joint_log_density(zs,games)
  return log_prior(zs) .+ all_games_log_likelihood(zs,games)
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
Test.DefaultTestSet("Test shapes of batches for likelihoods", Any[], 6, fal
se)
```

3 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.

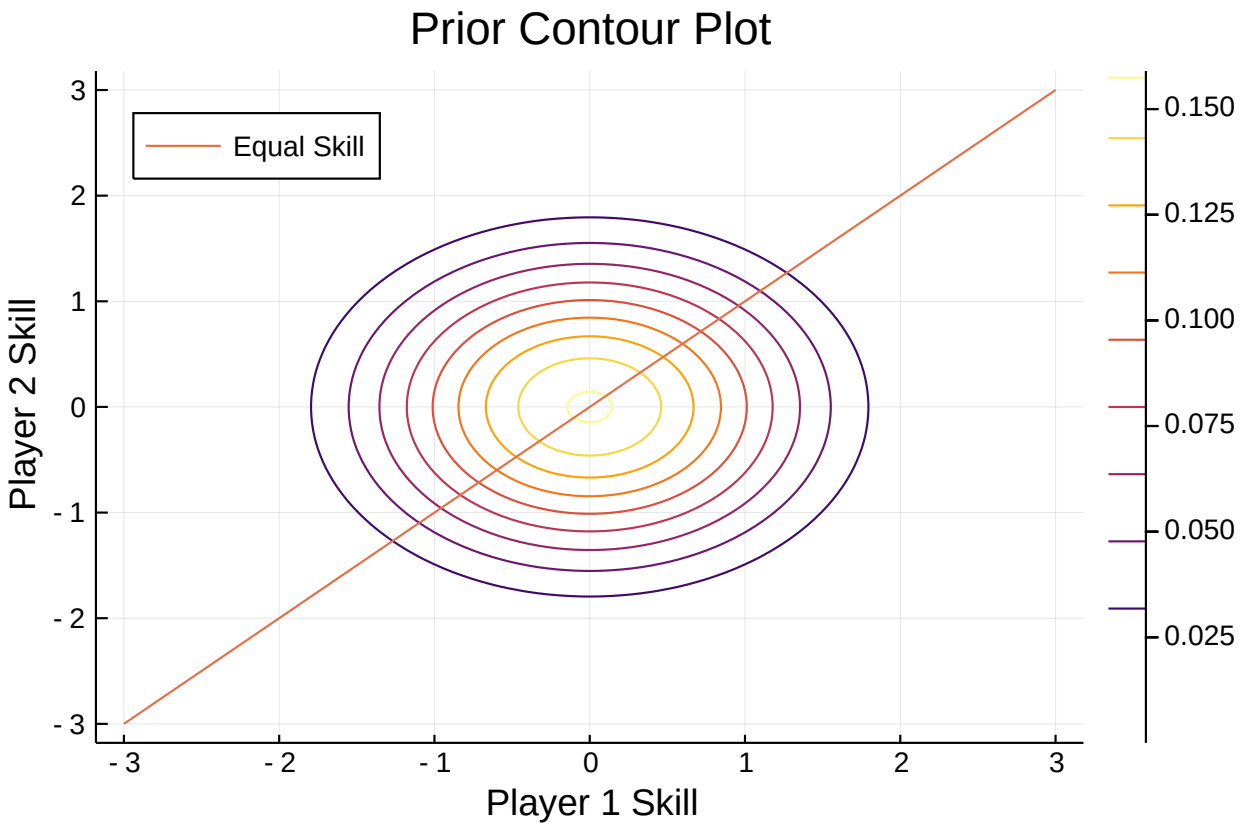
Provided in the starter code is a function skillcontour! which evaluates a provided function on a grid of zA and zB's and plots the isocontours of that function. As well there is a function plot line equal skill!. We have included an example for how you can use these functions.

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.

1. [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.

```
# 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)#; l="example gaussian")
plot_line_equal_skill!()
savefig(joinpath("plots","example_gaussian.pdf"))


# TODO: plot prior contours
plot(title="Prior Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
prior(zs) = exp(log_prior(zs))
skillcontour!(prior)
plot_line_equal_skill!()
```

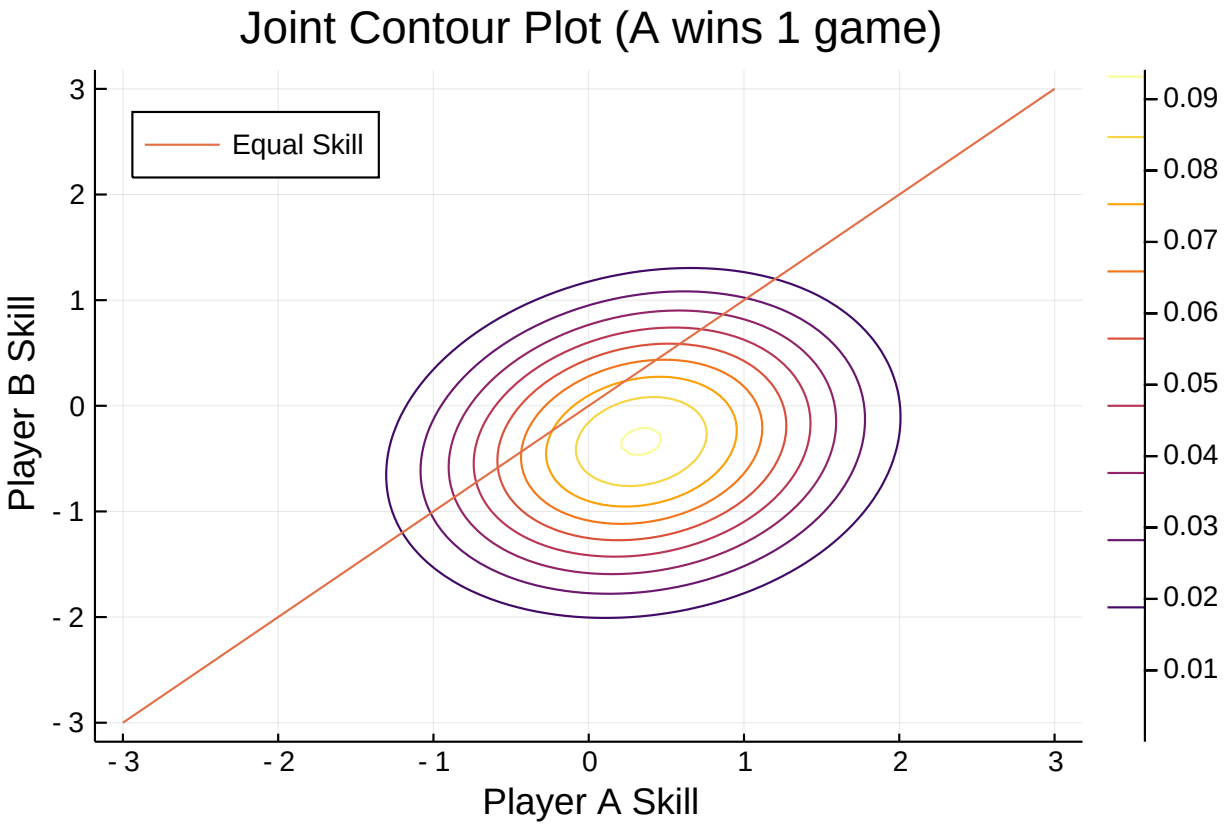


2. [2 points] Plot the isocontours of the likelihood function. Also plot the line of equal skill, zA = zB.

```
plot(title="Likelihood Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
games_1 = two_player_toy_games(1, 0)
likelihood(zs) = exp.(all_games_log_likelihood(zs, games_1))
skillcontour!(likelihood)
plot_line_equal_skill!()
savefig(joinpath("plots","likelihood_contours.pdf"))
```

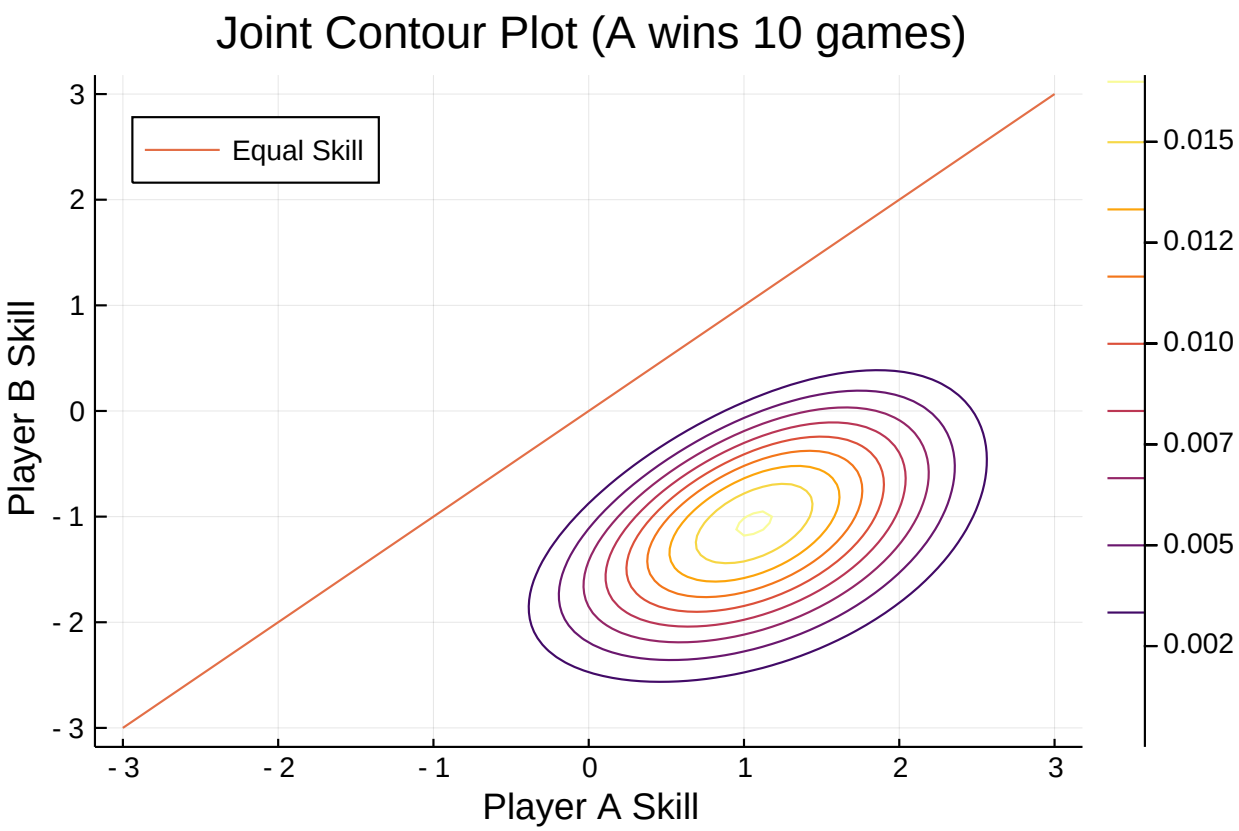
3. [2 points] Plot isocountours of the joint posterior over zA and zB given that player A beat player B in one match. Since the contours don't depend on the normalization constant, you can simply plot the isocontours of the log of joint distribution of p(zA,zB,A beat B) Also plot the line of equal skill, zA =zB.

```
# TODO: plot joint contours with player A winning 1 game
plot(title="Joint Contour Plot (A wins 1 game)",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint(zs) = exp.(joint_log_density(zs, games_1))
skillcontour!(joint)
plot_line_equal_skill!()
```



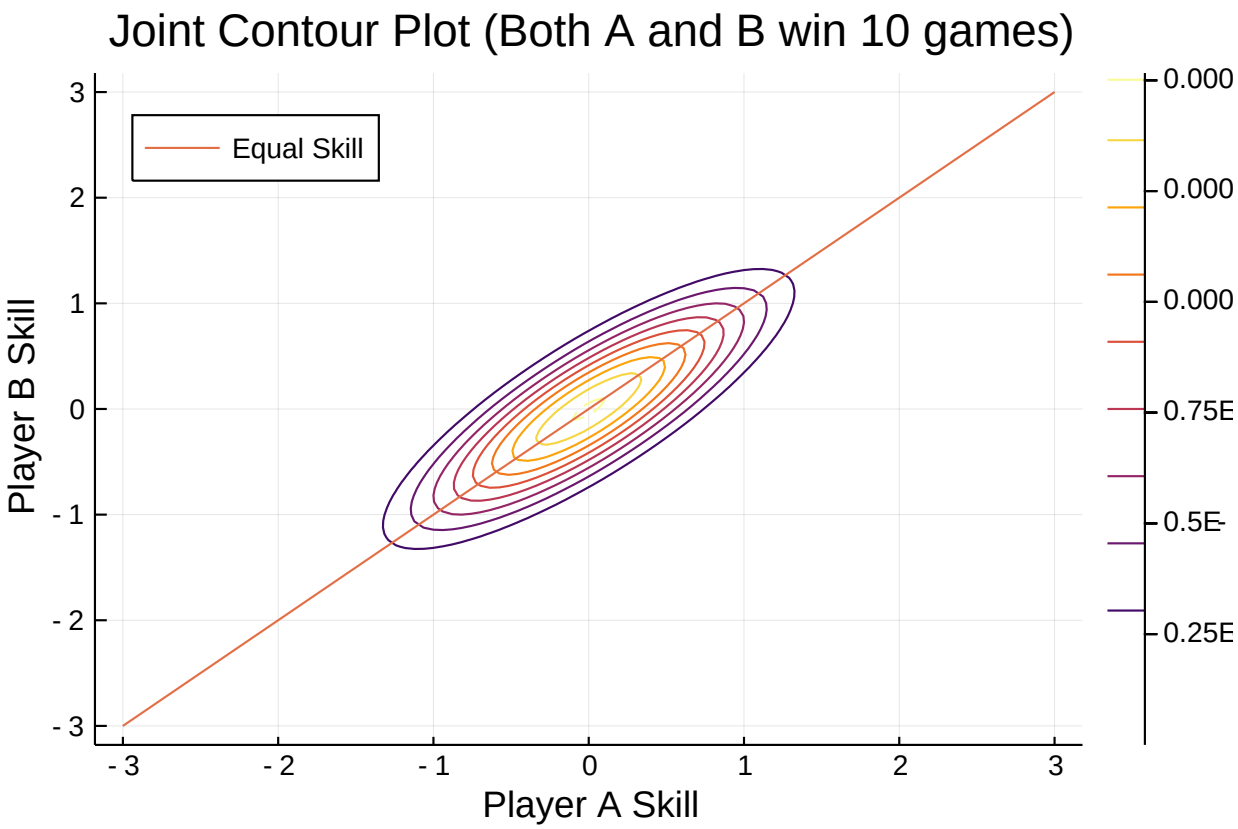
4. [2 points] Plot isocountours of the joint posterior over zA and zB given that 10 matches were played, and player A beat player B all 10 times. Also plot the line of equal skill, zA = zB.

```
# TODO: plot joint contours with player A winning 10 games
plot(title="Joint Contour Plot (A wins 10 games)",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
games_10 = two_player_toy_games(10, 0)
joint_10(zs) = exp.(joint_log_density(zs, games_10))
skillcontour!(joint_10)
plot_line_equal_skill!()
```



5. [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.

```
#TODO: plot joint contours with player A winning 10 games and player B winning 10
games
plot(title="Joint Contour Plot (Both A and B win 10 games)",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
games_10_10 = two_player_toy_games(10, 10)
joint_10_10(zs) = exp.(joint_log_density(zs, games_10_10))
skillcontour!(joint_10_10)
plot_line_equal_skill!()
```



For all plots, label both axes.

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

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

1. [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, qφ(z—data), plus an unknown constant. Use a fully-factorized Gaussian distributionfor qφ(z—data). This estimator takes the following arguments:

• params, the parameters φ of the approximate posterior qφ(z—data).

• A function logp, which is equal to the true posterior plus a constant. This function must take a batch 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)
  mu = params[1]
  ls = params[2]
  s = randn(size(mu)[1], num_samples)
  samples = s .* exp.(ls) .+ mu
  logp_estimate = logp(samples)
  logq_estimate = factorized_gaussian_log_density(mu, ls, samples)
  return sum(logp_estimate .- logq_estimate) ./ num_samples# should return scalar
(hint: average over batch)
end
```

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

2. [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.

```
# Conveinence function for taking gradients
function neg_toy_elbo(params; games = two_player_toy_games(1,0), num_samples = 100)
  # TODO: Write a function that takes parameters for q,
  # evidence as an array of game outcomes,
  # and returns the -elbo estimate with num_samples many samples from q
  logp(zs) = joint_log_density(zs,games)
  return -elbo(params,logp, num_samples)
end
```

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

3. [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.

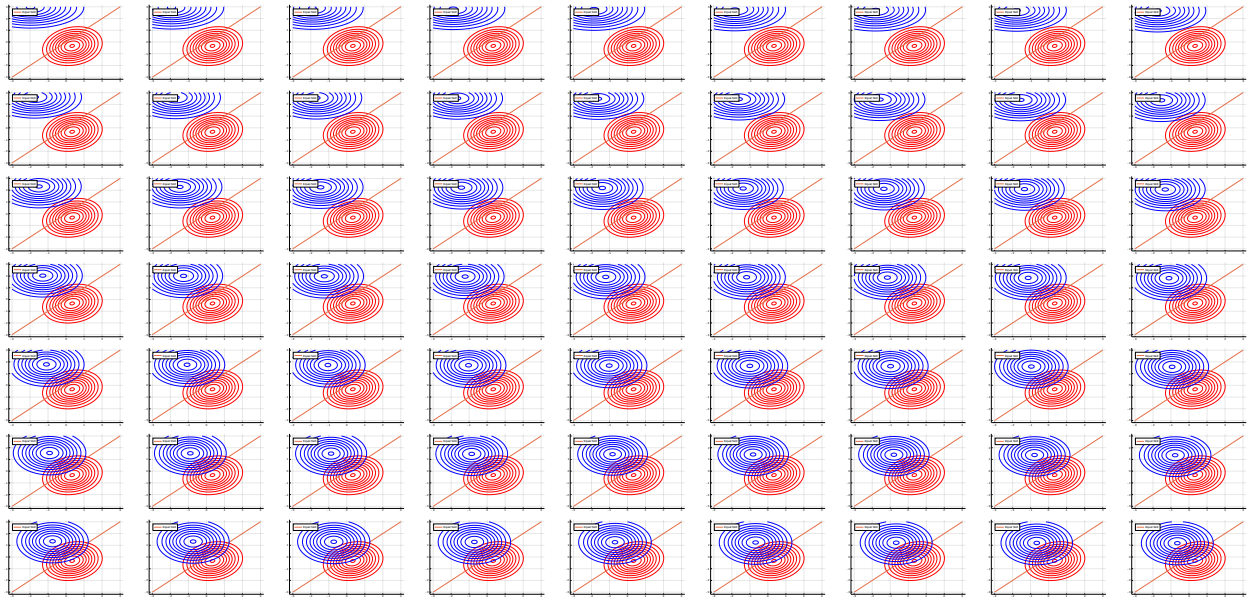
Return the parameters resulting from training.

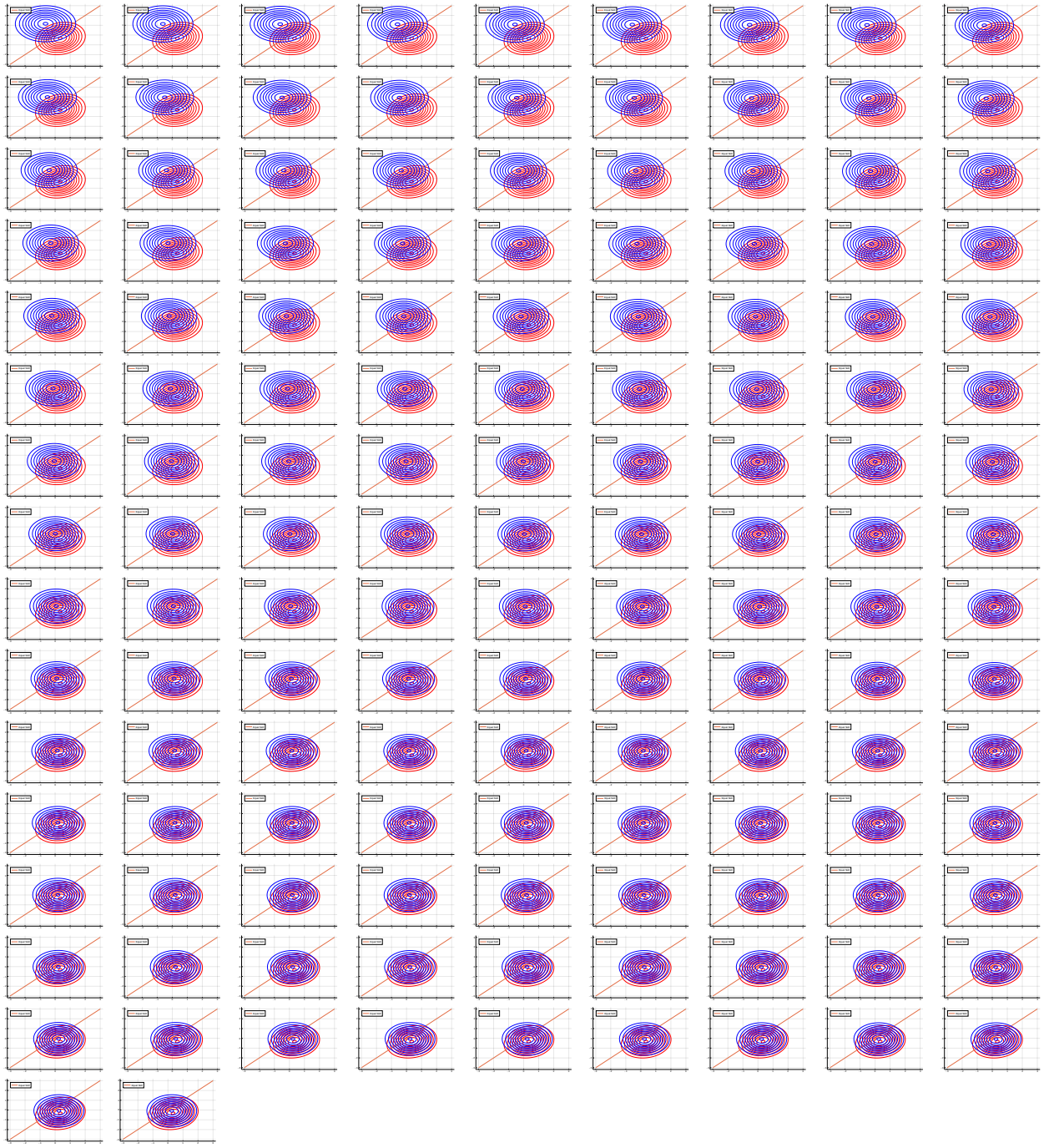
```
num_players_toy = 2
toy_mu = [-2.,3.] # Initial mu, can initialize randomly!
toy_ls = [0.5,0.] # Initual log_sigma, can initialize randomly!
toy_params_init = (toy_mu, toy_ls)

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

fit_toy_variational_dist (generic function with 1 method)
```

4. [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.

```
games_1 = two_player_toy_games(1,0)
init_params = (toy_mu, toy_ls)
fit_toy_variational_dist(init_params, games_1)

([0.2537298754925179, -0.08864656218944625], [-0.08044272290544474, -0.0840
1019736023582])
```

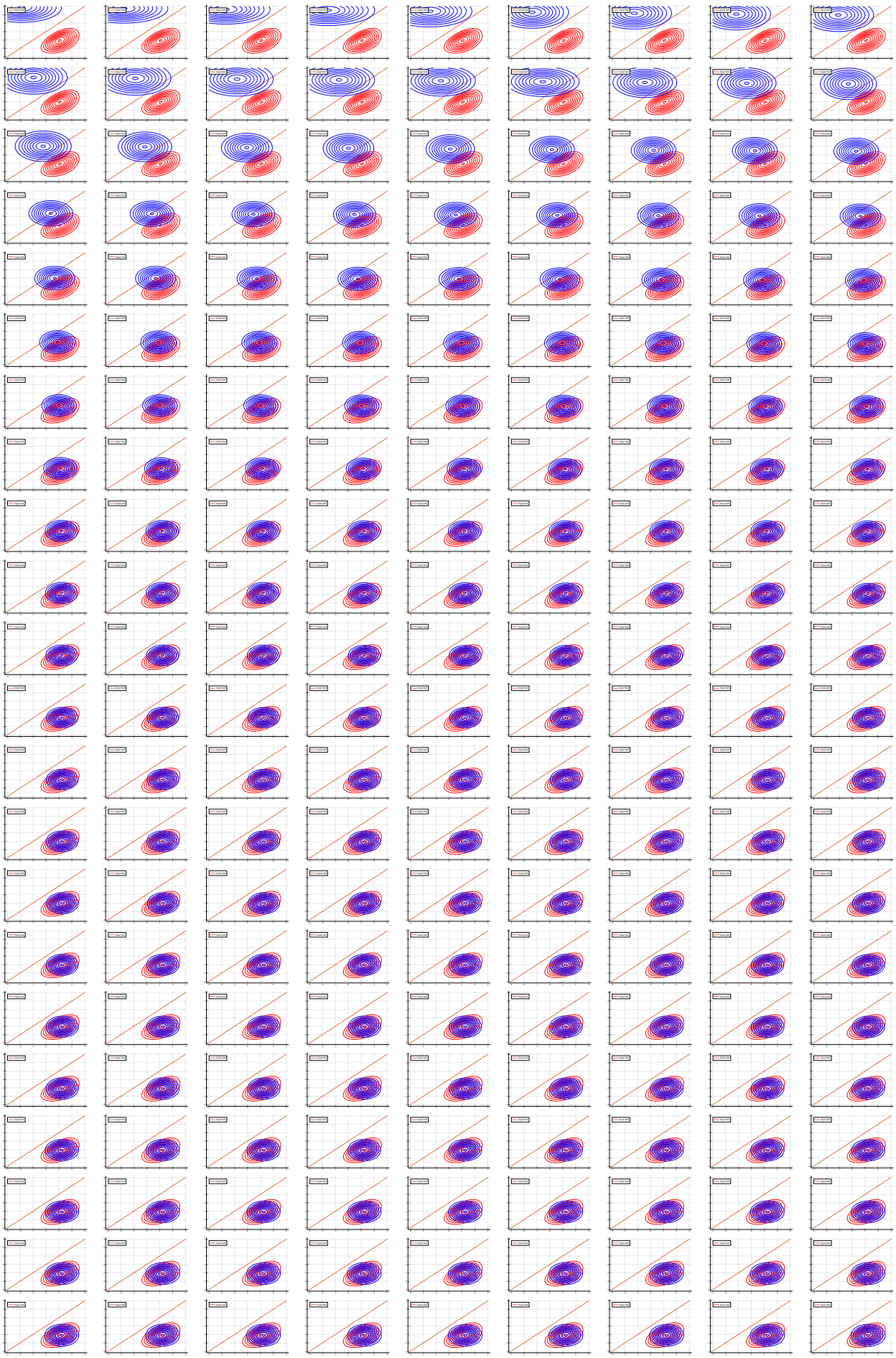


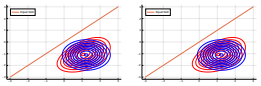


5. [2 points] Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 10 games. Report the final loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
games_10 = two_player_toy_games(10,0)
init_params = (toy_mu, toy_ls)
fit_toy_variational_dist(init_params, games_10)
```

```
([1.2292293961598095, -1.1081897375630698], [-0.2980960436603207, -0.320210
431459923])
```

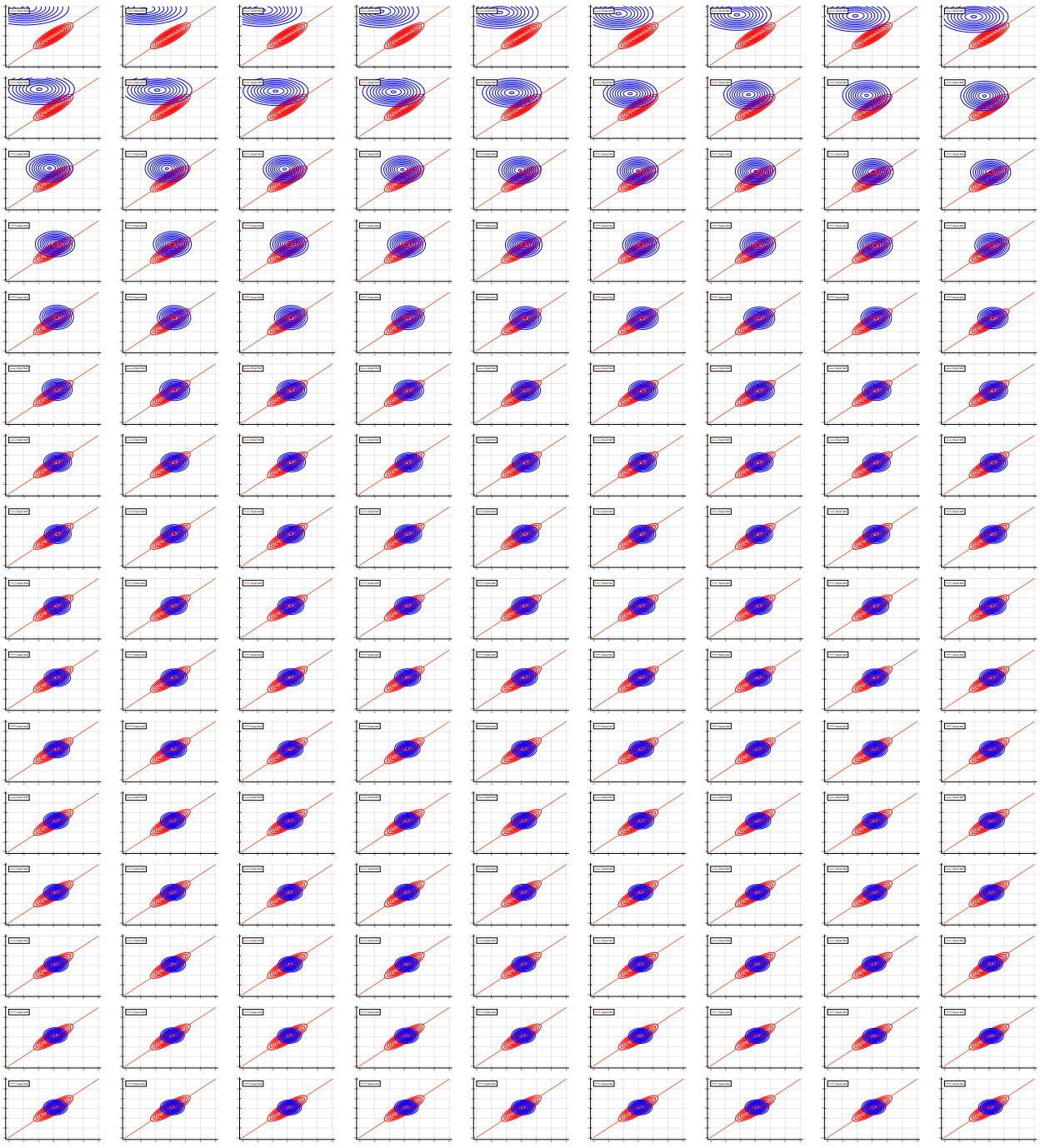


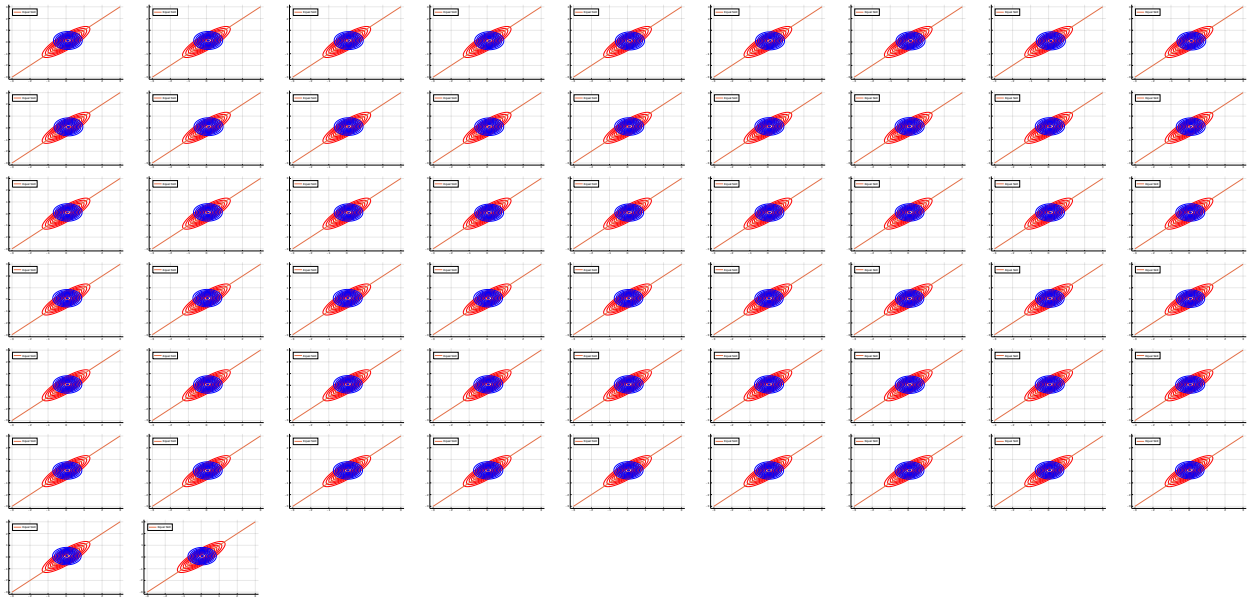


6. [2 points] Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 10 games and player B winning 10 games. Report the final loss. Also plot the optimized variational approximation contours (in blue) aand the target distribution (in red) on the same axes.

```
games_10_10 = two_player_toy_games(10,10)
init_params = (toy_mu, toy_ls)
fit_toy_variational_dist(init_params, games_10_10)
```

```
([0.05096936142182415, 0.0736640610227848], [-0.8260017075658138, -0.883535
885278676])
```





For all plots, label both axes.

5 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:

1. [1point]Foranytwoplayersiandj,p(zi,zj—all games) isalwaysproportionaltop(zi,zj,all games). In general, are the isocontours of p(zi, zj —all games) the same as those of p(zi, zj —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.

No. The isocontours of p(zi, zj —all games) is not the same as those of p(zi, zj —games between i and j). So, the games between other players besides i and j do provide information about the skill of players i and j.

2. [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.

```julia
# Load the Data
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")
```

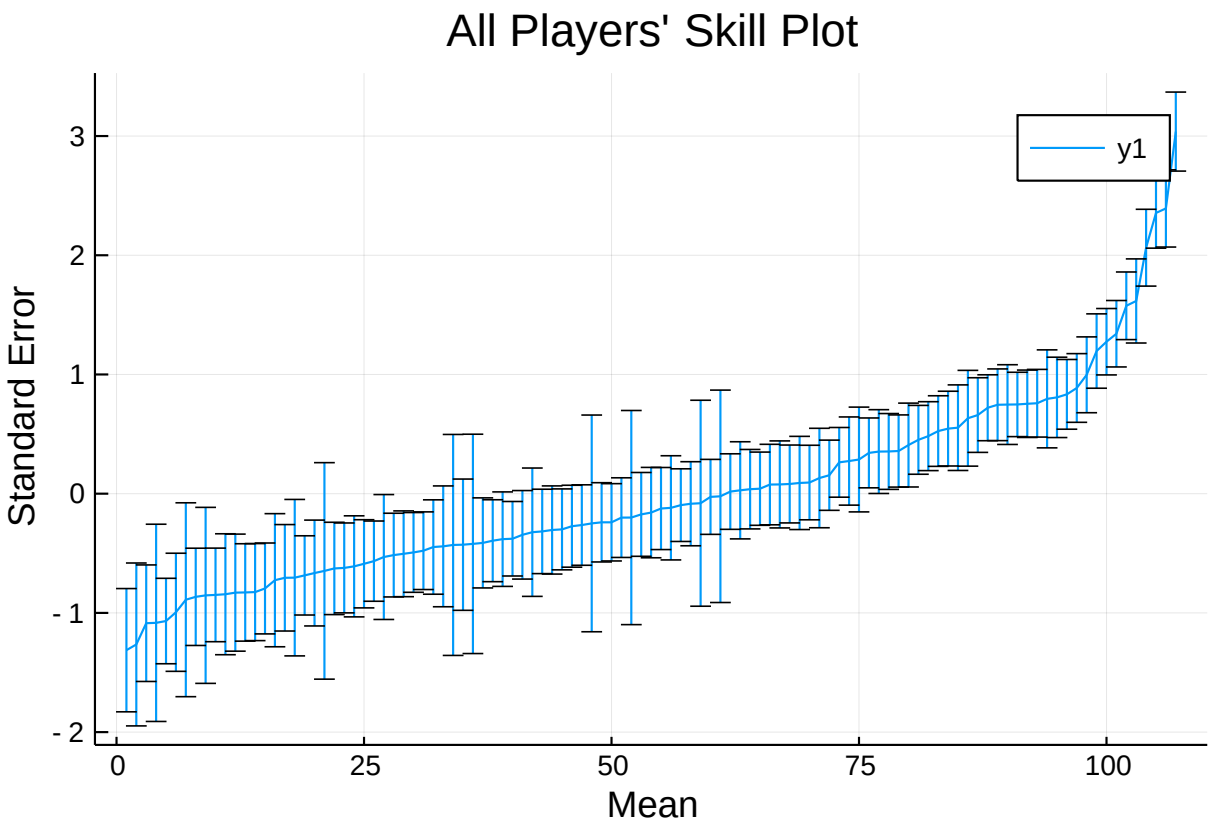
```
Loaded data for 107 players
```

```julia
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)
    #TODO: gradients of variational objective wrt params
    params_cur = params_cur .- lr .* grad_params[1]
    #TODO: update parmaeters wite lr-sized steps in desending gradient direction
    neg_elbo = neg_toy_elbo(params_cur, games = tennis_games, num_samples = num_q_samples)
    @info "elbo: $neg_elbo" #TODO: report objective value with current parameters
  end
  return params_cur
end
```

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

3. [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.

```julia
# TODO: Initialize variational family
init_mu = randn(num_players)#random initialziation
init_log_sigma = randn(num_players)# random initialziation
init_params = (init_mu, init_log_sigma)

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


#TODO: 10 players with highest mean skill under variational model
#hint: use sortperm
perm = sortperm(trained_params[1])
plot(trained_params[1][perm], yerror=exp.(trained_params[2][perm]),
      xlabel = "Mean", ylabel = "Standard Error",
      title = "All Players' Skill Plot")
```



4. [2 points] List the names of the 10 players with the highest mean skill under the variational model.

```
p_names = []
temp = reverse(perm)

for i in 1:10
  p_name = player_names[temp[i]]
  push!(p_names, p_name)
end
p_names
```

```
10-element Array{Any,1}:
 "Novak-Djokovic"
 "Roger-Federer"
 "Rafael-Nadal"
 "Andy-Murray"
 "Robin-Soderling"
 "David-Ferrer"
 "Jo-Wilfried-Tsonga"
 "Tomas-Berdych"
 "Juan-Martin-Del-Potro"
 "Richard-Gasquet"
```

5. [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.

```
#TODO: joint posterior over "Roger-Federer" and ""Rafael-Nadal""
#hint: findall function to find the index of these players in player_names
```

```
roger = findfirst(x -> x == "Roger-Federer", player_names)[1]
rafael = findfirst(x -> x == "Rafael-Nadal", player_names)[1]
mu_fed_Nad = [trained_params[1][roger], trained_params[1][rafael]]
ls_fed_Nad = [trained_params[2][roger], trained_params[2][rafael]]

joint_pos(zs) = exp.(factorized_gaussian_log_density(mu_fed_Nad, ls_fed_Nad, zs))
plot(title="Joint Posterior over Roger Federer and Rafael Nadal",
    xlabel = "Roger Federer Skill",
    ylabel = "Rafael Nadal Skill"
   )
skillcontour!(joint_pos)
plot_line_equal_skill!()
savefig(joinpath("plots","Joint_Posterior_over_Roger_Federer_and_Rafael_Nadal.pdf"))
```

6. [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. Express your answer in terms of the cumulative distribution function of a one-dimensional Gaussian random variable.

• 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 $AX \sim N(A\mu, A\Sigma AT)$ 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 $Xi \sim N(\mu i, \Sigma ii)$

```
using Distributions

function calculate_prob_a_higher_than_b(mu, ls)
  A = [1 -1; 0 1]
  sigma = exp.(ls)
  mu_ya = (A * mu)[1]
  sigma_ya = (A .* sigma .* A')[1][1]
  return 1 - cdf(Normal(mu_ya, sigma_ya), 0)
end
```

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

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \tag{1}$$

$$\mu = [\mu_1 \mu_2] \tag{2}$$

$$\sigma = [\sigma_1 \sigma_2] \tag{3}$$

$$1 - cdf(N(A * mu_1, A. * sigma. * A'_{11}), 0) \tag{4}$$

7. [2 points] Using the formula from part c, compute the exact probability under your approximate posterior that Roger Federer has higher skill than Rafael Nadal. Then, estimate it using simple Monte Carlo with 10000 examples, again using your approximate posterior.

```
# Roger-Federer's skill is higher than Rafael-Nadal's
p1 = calculate_prob_a_higher_than_b(mu_fed_Nad, ls_fed_Nad)
print(p1)
```

```
0.5464209869650729
```

```
sample_Fed = rand(Normal(mu_fed_Nad[1], exp(ls_fed_Nad[1])), 10000)
sample_Nad = rand(Normal(mu_fed_Nad[2], exp(ls_fed_Nad[2])), 10000)

sum_F_N = sum(sample_Fed[i] > sample_Nad[i] for i in 1:10000)

prob_Fed_higher_than_Nad = sum_F_N / 10000
print(prob_Fed_higher_than_Nad)
```

```
0.5319
```

8. [2 points] Using the formula from part c, 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, again using your approximate posterior.

```
lowest_mean = minimum(trained_params[1])
lowest_index = findfirst(x -> x == lowest_mean, trained_params[1])
lowest_ls = trained_params[2][lowest_index]
player_names[lowest_index]

mu_Fed_lowest = [mu_fed_Nad[1], lowest_mean]
ls_Fed_lowest = [ls_fed_Nad[1], lowest_ls]
p2 = calculate_prob_a_higher_than_b(mu_Fed_lowest, ls_Fed_lowest)
print(p2)
```

```
1.0
```

```
sample_lowest = rand(Normal(lowest_mean, exp(lowest_ls)), 10000)
sum_F_lowest = sum(sample_Fed[i] > sample_lowest[i] for i in 1:10000)
prob_Fed_higher_than_lowest = sum_F_lowest / 10000
print(prob_Fed_higher_than_lowest)
```

```
1.0
```

9. [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.

Answer in section c, e would be different.