Assignment 2

Kevin Shi, 1002519134

March 21, 2020

```
# using Revise # lets you change A2funcs without restarting julia!
# includet("A2_src.jl")
include("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!
```

1 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 $K \times N$ array where each row is a setting of the skills for all N players, it returns a $K \times 1$ 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
```

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

2. [**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
```

```
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 $K \times N$ array where each row is a setting of the skills for all N players, and an $M \times 2$ array of game outcomes, it returns a $K \times 1$ 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 =  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 $p(z_1, z_2, \ldots, z_N$, all game outcomes)

```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)
```

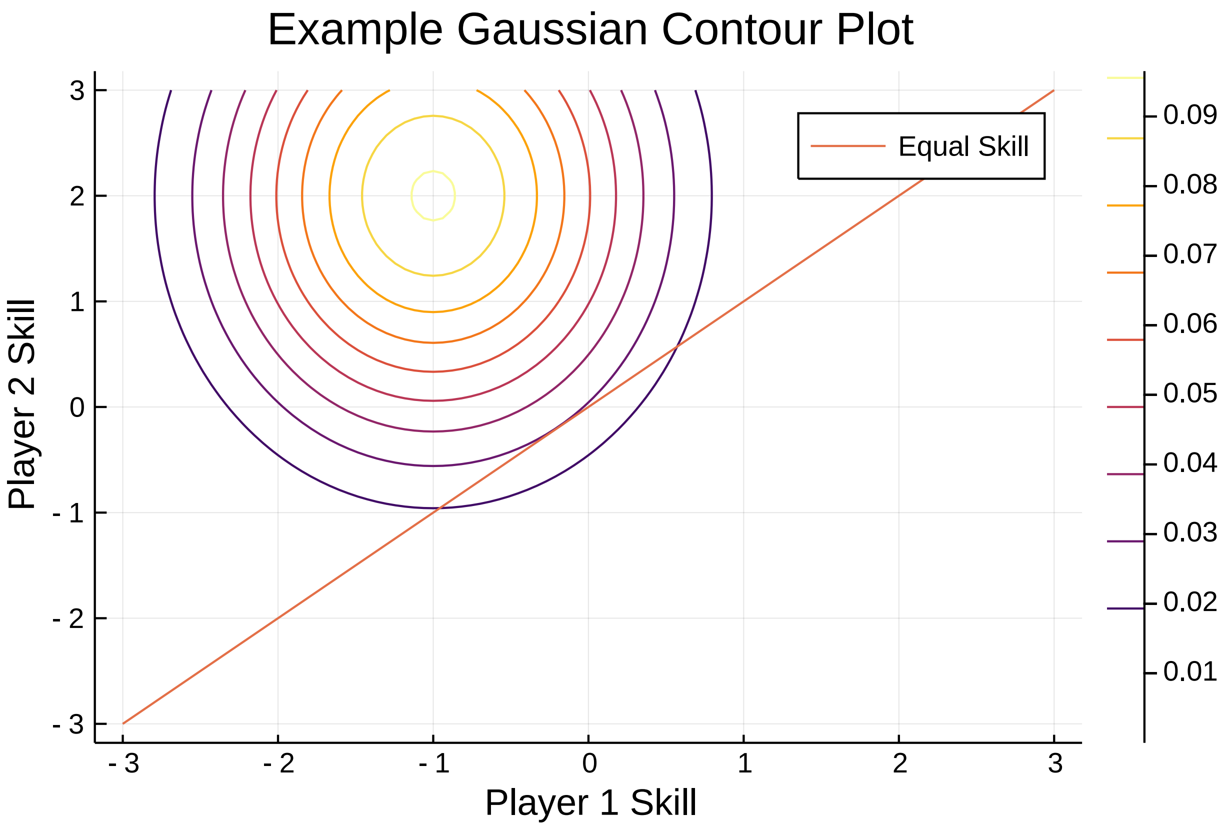
```julia
@testset "Test shapes of batches for likelihoods" begin
  B = 15 # number of elements in batch
  N = 4 # Total Number of Players
  test_zs = randn(4,15)
  test_games = [1 2; 3 1; 4 2] # 1 beat 2, 3 beat 1, 4 beat 2
  @test size(test_zs) == (N,B)
  #batch of priors
  @test size(log_prior(test_zs)) == (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 [10points]

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 z_A and z_B'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.

```
# 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; label="example gaussian")
skillcontour!(example_gaussian)
display(plot_line_equal_skill!())
```

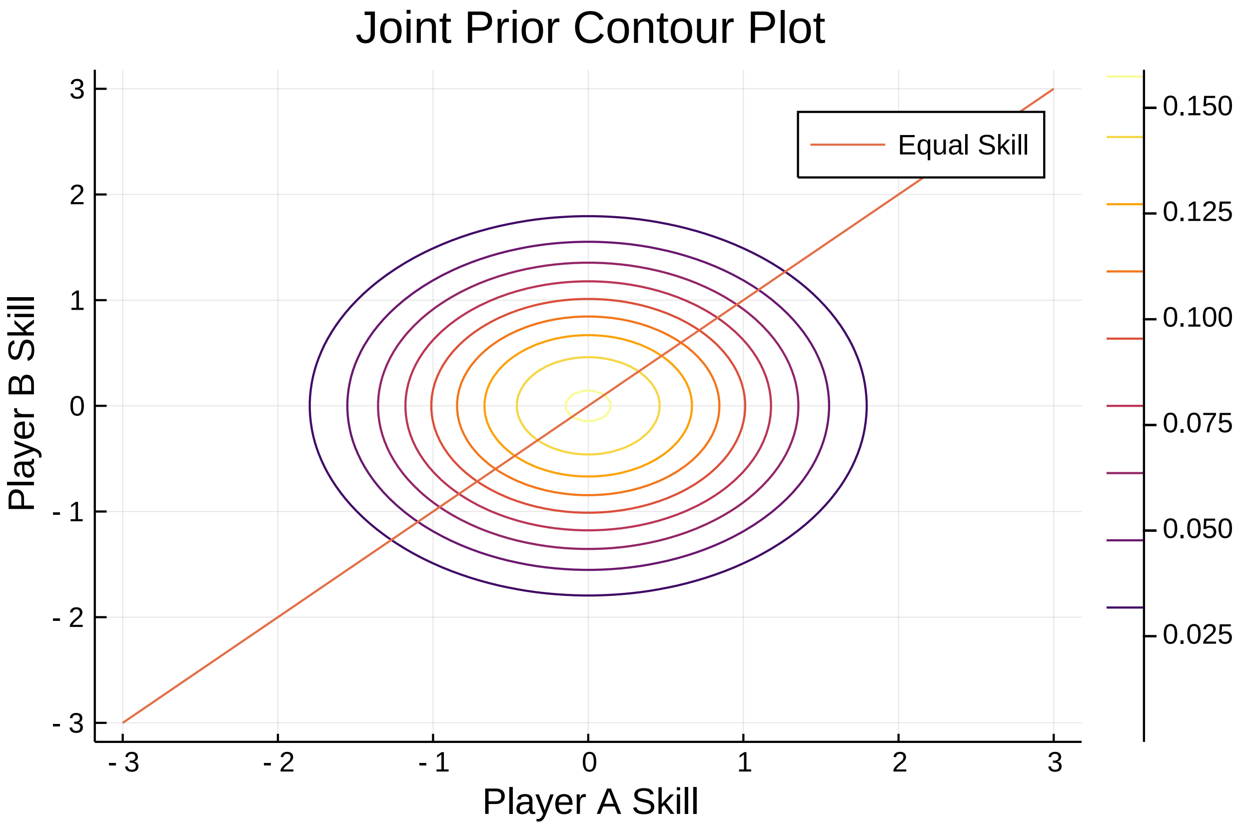



```
savefig(joinpath("plots","example_gaussian.pdf"))
```

1. **[2 points]** For two players A and B, plot the isocontours of the joint prior over their skills. Also plotthe line of equal skill,$z_A = z_B$. Hint: you've already implemented

thelogof the likelihood function.

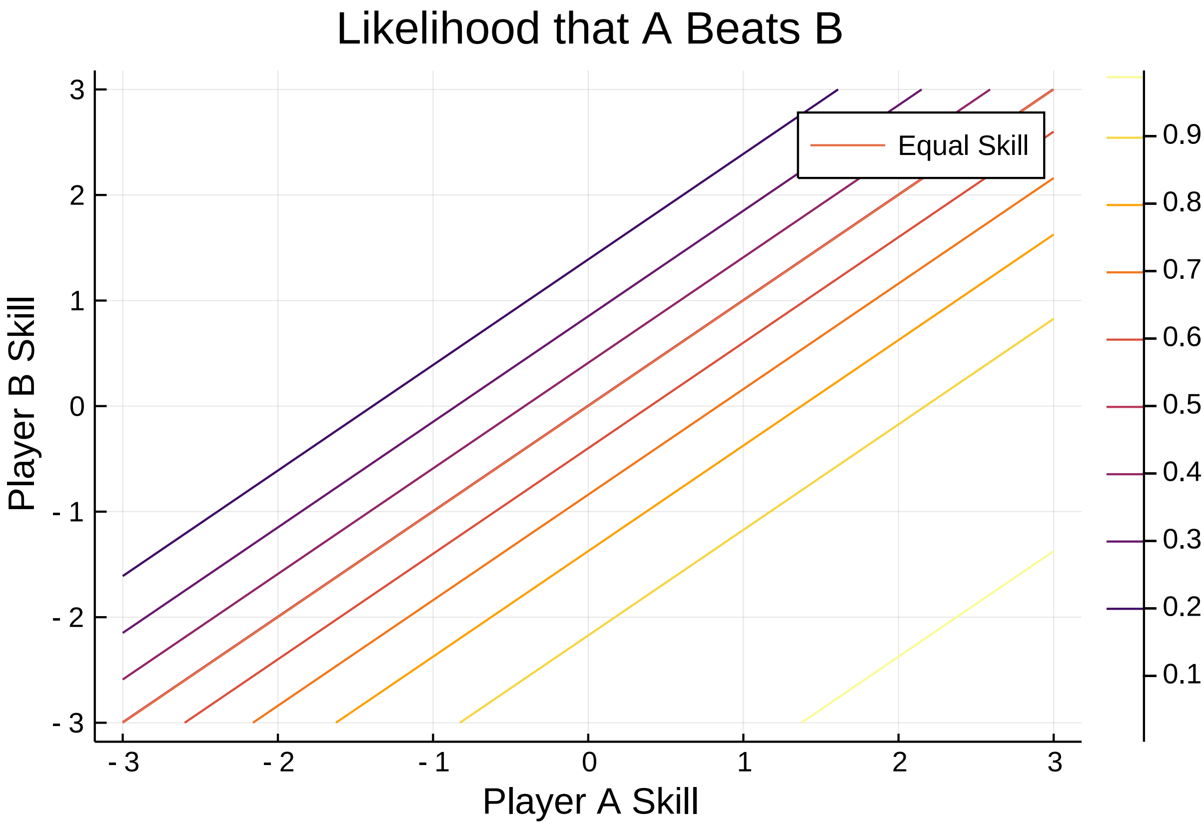
```
plot(title = "Joint Prior Contour Plot",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint_prior(zs) = exp(log_prior(zs))
skillcontour!(joint_prior)
display(plot_line_equal_skill!())
```




```
savefig(joinpath("plots","2a_prior_contours.pdf"))
```

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

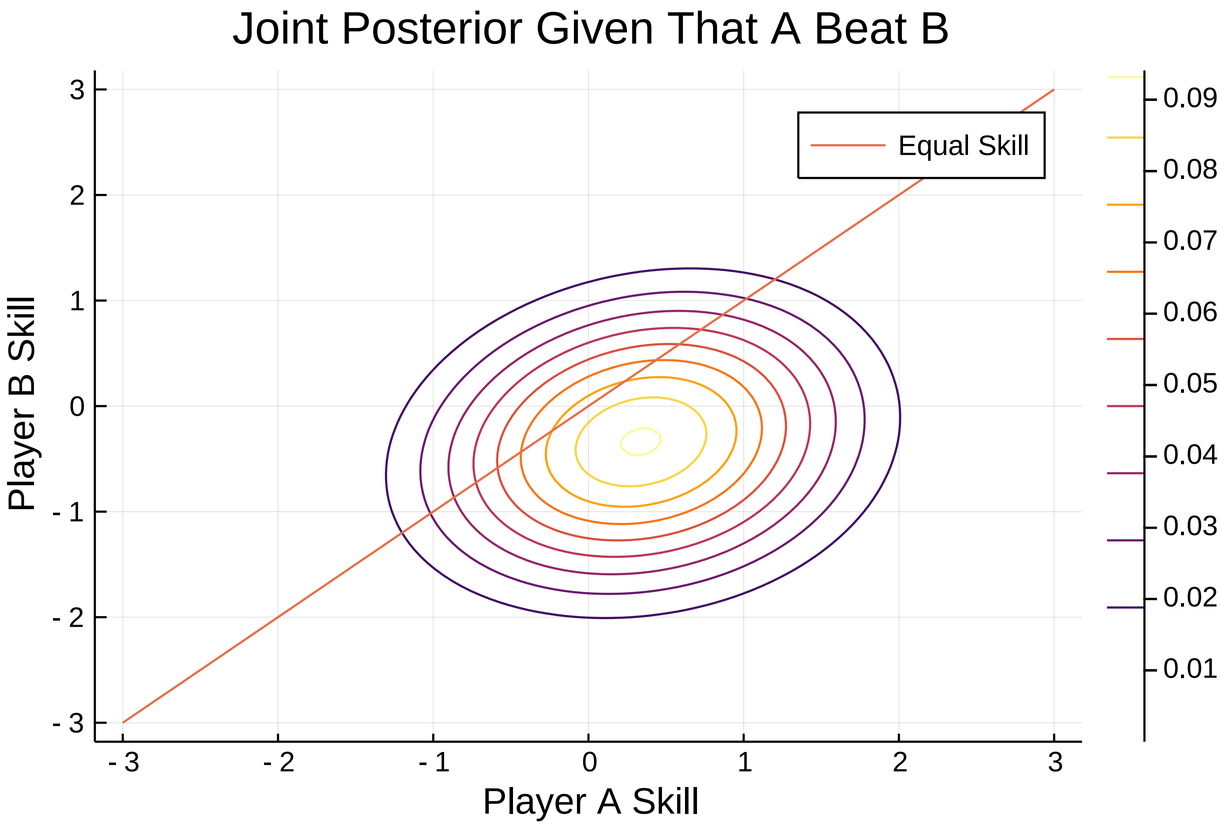
```
plot(title = "Likelihood that A Beats B",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
likelihood_a_beats_b(zs) = exp.(logp_a_beats_b.(zs[1,:],zs[2,:]))
skillcontour!(likelihood_a_beats_b)
display(plot_line_equal_skill!())
```




```
savefig(joinpath("plots","2b_likelihood_contours.pdf"))
```

3. **[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, $z_A = z_B$.

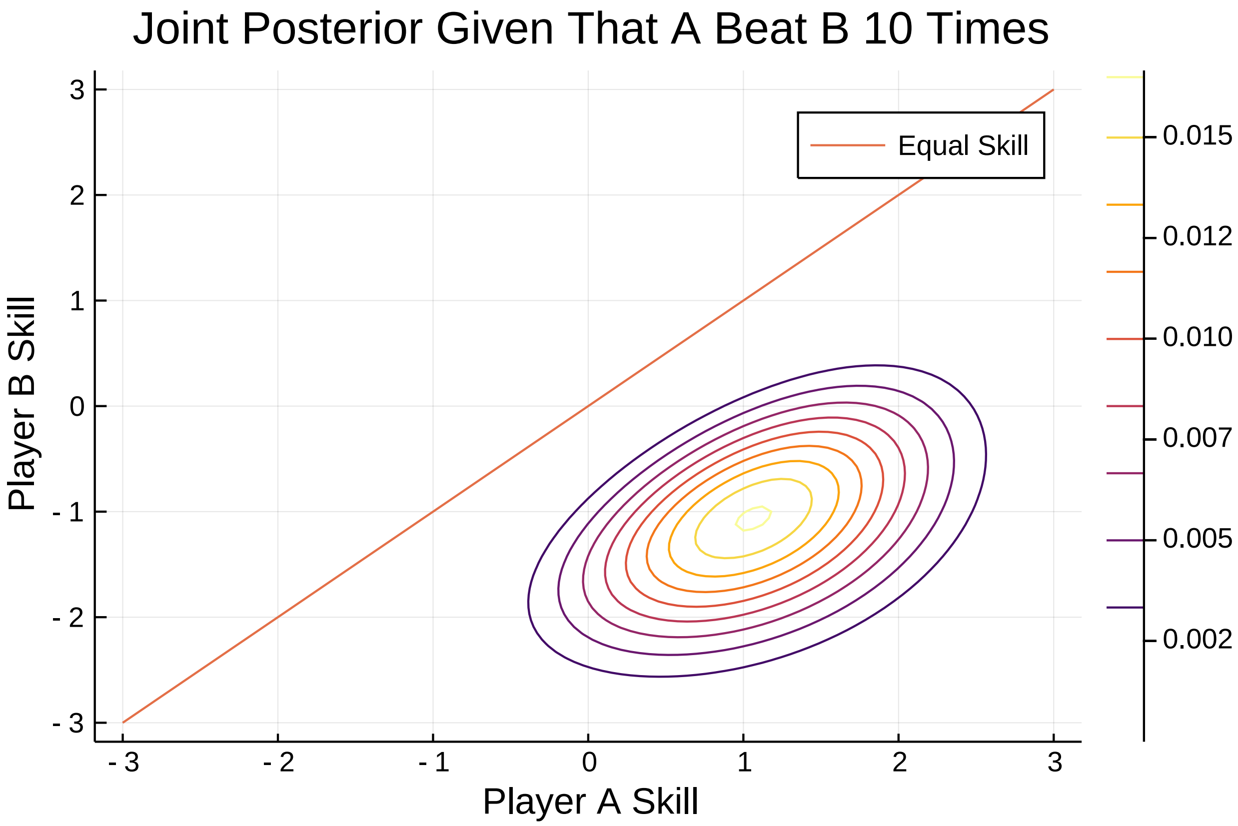
```
plot(title = "Joint Posterior Given That A Beat B",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint_posterior(zs) = exp(joint_log_density(zs, two_player_toy_games(1,0)))
skillcontour!(joint_posterior)
display(plot_line_equal_skill!())
```




```
savefig(joinpath("plots","2c_A_beat_B_once_contours.pdf"))
```

4. [**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, $z_A = z_B$.

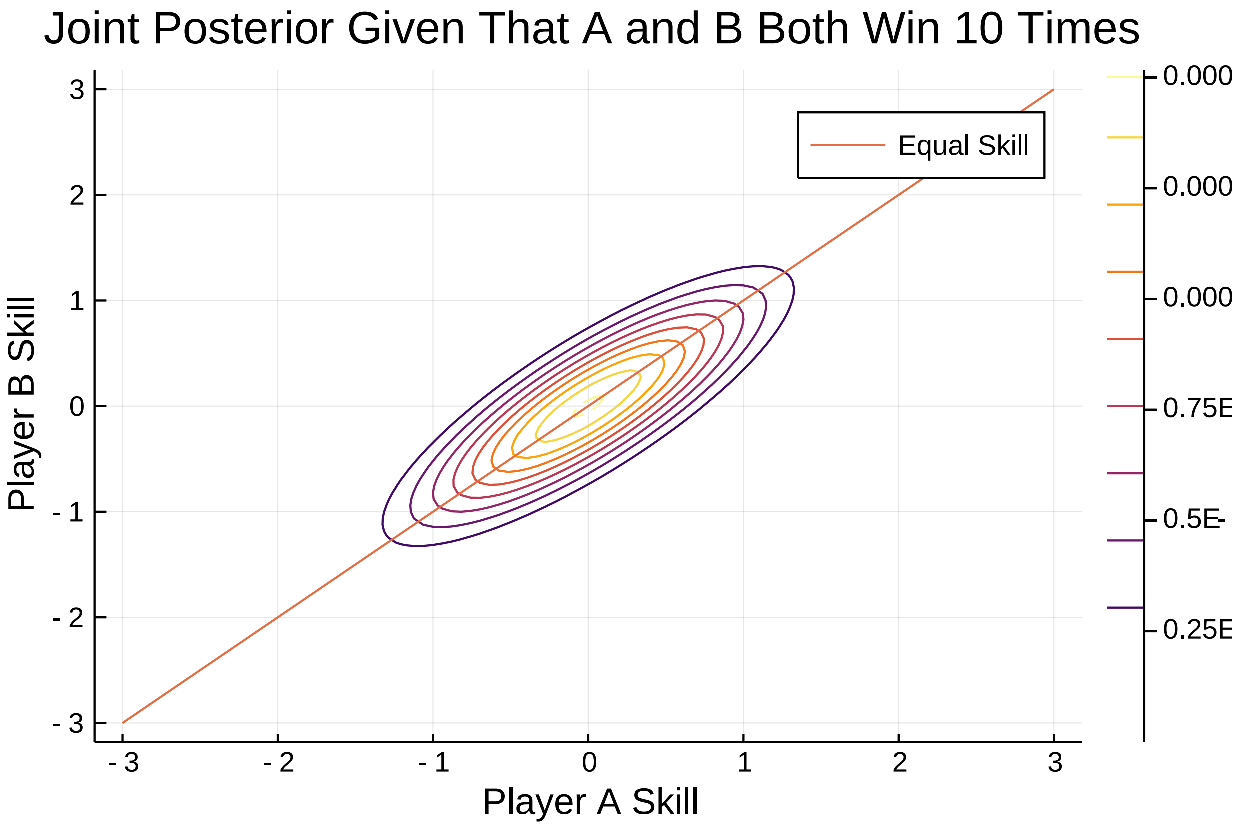
```
plot(title = "Joint Posterior Given That A Beat B 10 Times",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint_posterior(zs) = exp(joint_log_density(zs, two_player_toy_games(10,0)))
skillcontour!(joint_posterior)
display(plot_line_equal_skill!())
```




```
savefig(joinpath("plots","2d_A_beat_B_ten_times_contours.pdf"))
```

5. [**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, $z_A = z_B$.

```
plot(title = "Joint Posterior Given That A and B Both Win 10 Times",
    xlabel = "Player A Skill",
    ylabel = "Player B Skill"
   )
joint_posterior(zs) = exp(joint_log_density(zs, two_player_toy_games(10,10)))
skillcontour!(joint_posterior)
display(plot_line_equal_skill!())
```




```
savefig(joinpath("plots","2e_A_and_B_win_ten_times_contours.pdf"))
```

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 specification from the approximate inference strategy. The original Trueskill paper from 2007 used message passing. Carl Rasmussen′sassignment uses Gibbs sampling, a form of Markov Chain Monte Carlo. We′ll use gradient-based stochasticvariational inference, which wasn′t invented until around 2014.In this question we will optimize an approximate posterior distribution with stochastic variational inference to approximate the true posterior.

1. [**5 points**] Implement a function `elbo` which computes an unbiased estimate of the evidence lowerbound. 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 overall players′ skills. This batch of samples `zs` will be an array with dimensions (N,B).

- `numsamples`

  , 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)
  # μ = params[1], log(σ) = params[2]
  num_players = size(params[1])[1]
  samples = exp.(params[2]) .* randn(num_players,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 #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.

```
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 variational parameters, and the evidence. Inside it will perform a number of iterations of gradient descentwhere for each iteration :

- Compute the gradient of the loss with respect to the parameters using automatic differentiation.
- Update the parameters by taking anlr-scaled step in the direction of the descending gradient.
- Report the loss with the new parameters (using@infoor print statements)
- On the same set of axes plot the target distribution in red and the variational approximation inblue.

```
# Toy game
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)
```

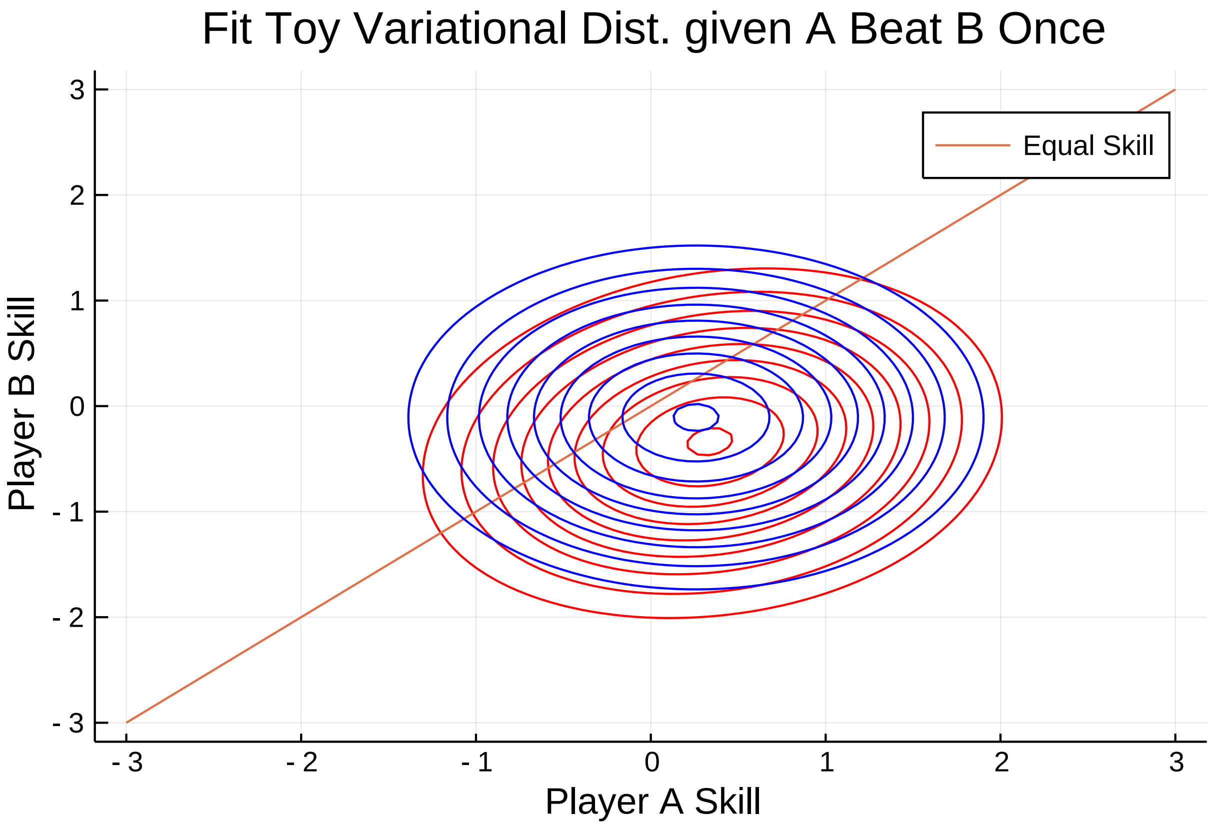
```
function fit_toy_variational_dist(init_params, toy_evidence; num_itrs=200, lr= 1e-2,
num_q_samples = 10, title = "Fit Toy Variational Dist.")
  params_cur = init_params
  for i in 1:num_itrs
    grad_params = gradient(params->neg_toy_elbo(params; games = toy_evidence, num_samples
= num_q_samples),params_cur)[1]
    params_cur = params_cur .- grad_params .* lr
    @info neg_toy_elbo(params_cur; games = toy_evidence, num_samples = num_q_samples)

    # This is commented out for the report so that only the final image will show
    # plot(title = title,
    #     xlabel = "Player A Skill",
    #     ylabel = "Player B Skill"
    #    )
    # joint_posterior(zs) = exp(joint_log_density(zs, toy_evidence))
    # skillcontour!(joint_posterior;colour=:red)
    # plot_line_equal_skill!()
    # iter_gaussian(zs) =
exp(factorized_gaussian_log_density(params_cur[1],params_cur[2],zs))
    # display(skillcontour!(iter_gaussian;colour=:blue))  # run this line to see the
model train
  end

  plot(title = title,
      xlabel = "Player A Skill",
      ylabel = "Player B Skill"
     )
  joint_posterior(zs) = exp(joint_log_density(zs, toy_evidence))
  skillcontour!(joint_posterior;colour=:red)
  plot_line_equal_skill!()
  iter_gaussian(zs) = exp(factorized_gaussian_log_density(params_cur[1],params_cur[2],zs))
  display(skillcontour!(iter_gaussian;colour=:blue))  # run this line to see the model
train
  print("Final Loss: ", neg_toy_elbo(params_cur; games = toy_evidence, num_samples =
num_q_samples))
  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 variationalapproximation contours (in blue) and the target distribution (in red) on the same axes.

```
toy_games_1_0 = two_player_toy_games(1,0)
final_params = fit_toy_variational_dist(toy_params_init, toy_games_1_0; title = "Fit Toy
Variational Dist. given A Beat B Once")
```

```
Final Loss: 0.6089712214733692
```

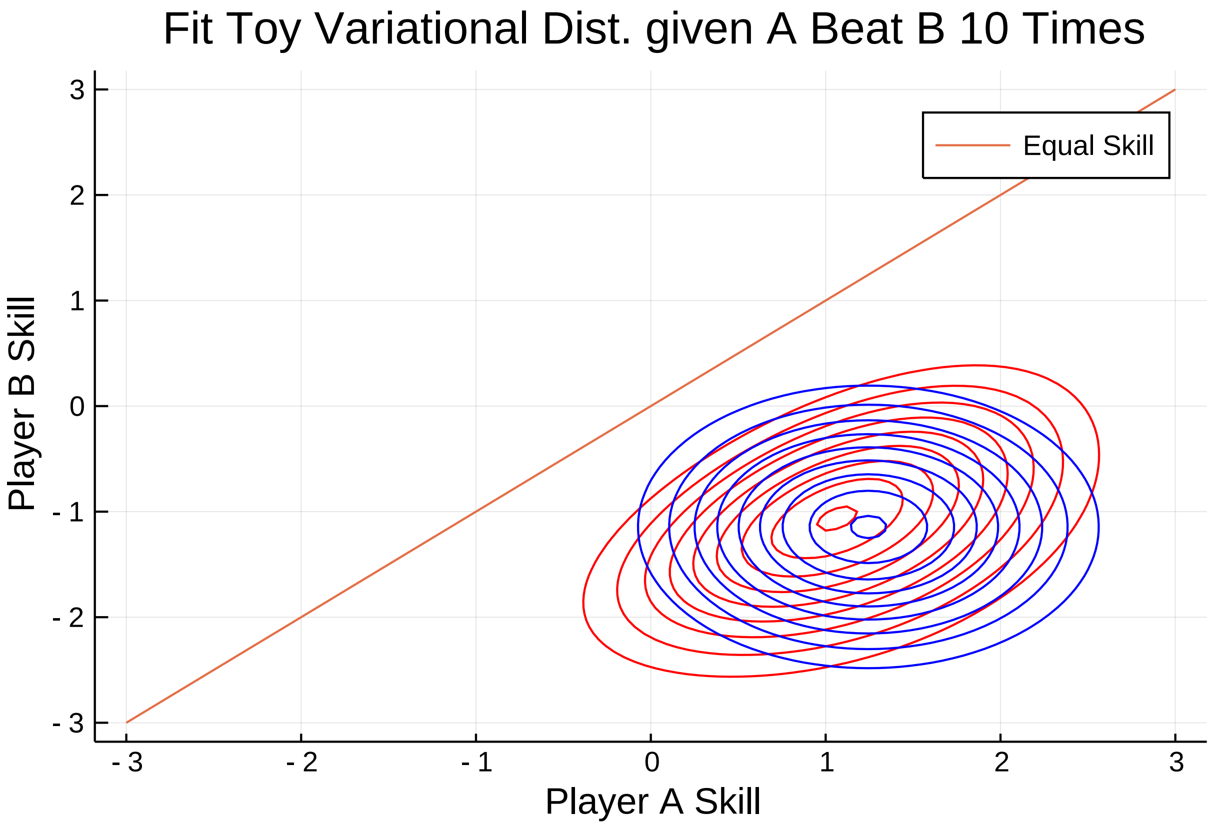



```
savefig(joinpath("plots","3d_fit_toy_A_beat_B_once.pdf"))
```

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

```
toy_games_10_0 = two_player_toy_games(10,0)
final_params = fit_toy_variational_dist(toy_params_init, toy_games_10_0; title = "Fit Toy
Variational Dist. given A Beat B 10 Times")
```

```
Final Loss: 2.8642139881396833
```

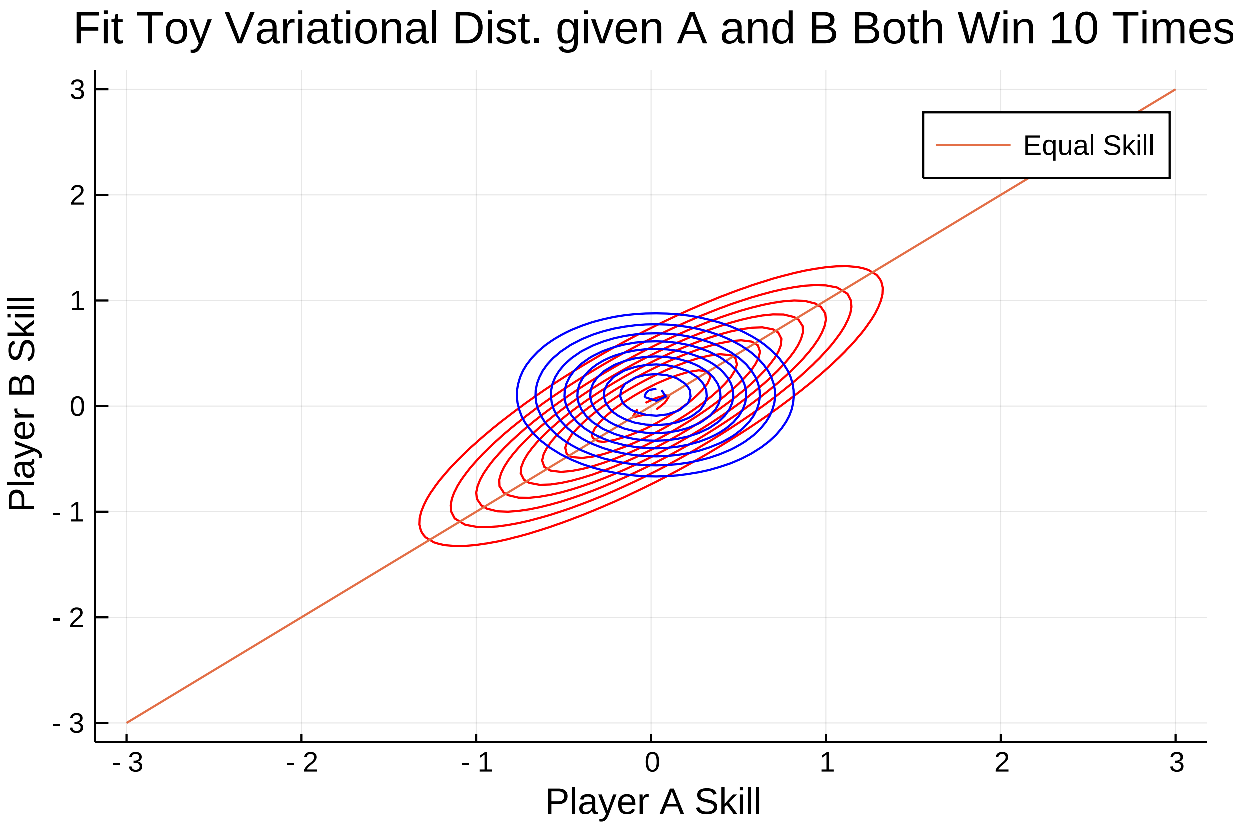



```
savefig(joinpath("plots","3e_fit_toy_A_beat_B_ten_times.pdf"))
```

6. [**2 points**] Initialize a variational distribution parameters and optimize them to approximate the jointwhere 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.

```
toy_games_10_10 = two_player_toy_games(10,10)
final_params = fit_toy_variational_dist(toy_params_init, toy_games_10_10; title = "Fit Toy
Variational Dist. given A and B Both Win 10 Times")
```

```
Final Loss: 15.48536289137503
```




```
savefig(joinpath("plots","3f_fit_toy_A_and_B_win_ten_times.pdf"))
```

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:

```
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
```

1. **[1 point]** For any two players i and j, $p(zi, zj|allgames)$ is always proportional to $p(zi, zj, allgames)$. In general, are the isocontours of $p(zi, zj|allgames)$ the same as

those of $p(zi, zj|gamesbetweeniandj)$? That is, do the games between other players besidesiandjprovide information about the skill ofplayers 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.

Yes the games between players other than i and j 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 previousquestion except it does not plot anything. Initialize a variational distribution and fit it to the jointdistribution with all the observed tennis games from the dataset. Report the final negative `ELBO` estimate after optimization.

```
init_mu = vec(zeros(num_players, 1))
init_log_sigma = vec(ones(num_players, 1))
init_params = (init_mu, init_log_sigma)

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 .- grad_params .* lr
    @info neg_toy_elbo(params_cur; games = tennis_games, num_samples = num_q_samples)
  end
  print("Final Loss:", neg_toy_elbo(params_cur; games = tennis_games, num_samples =
num_q_samples))
  return params_cur
end
```

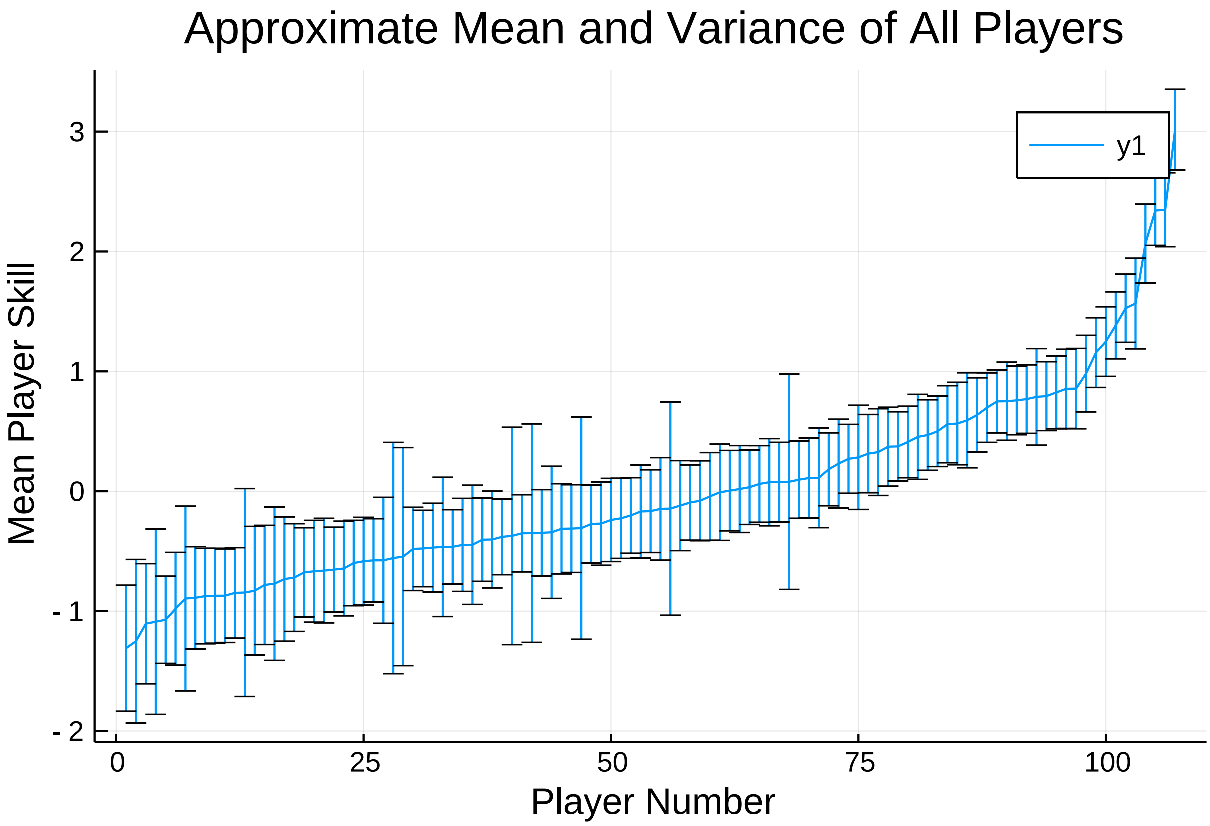
```
fit_variational_dist (generic function with 1 method)
```

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.(lo`
There′s no need to include the names of the players.

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

```
Final Loss:1142.8409093770138
```

```
perm = sortperm(trained_params[1]);
display(plot(trained_params[1][perm], yerror=exp.(trained_params[2][perm]),
    title = "Approximate Mean and Variance of All Players",
    xlabel = "Player Number",
    ylabel = "Mean Player Skill"))
```



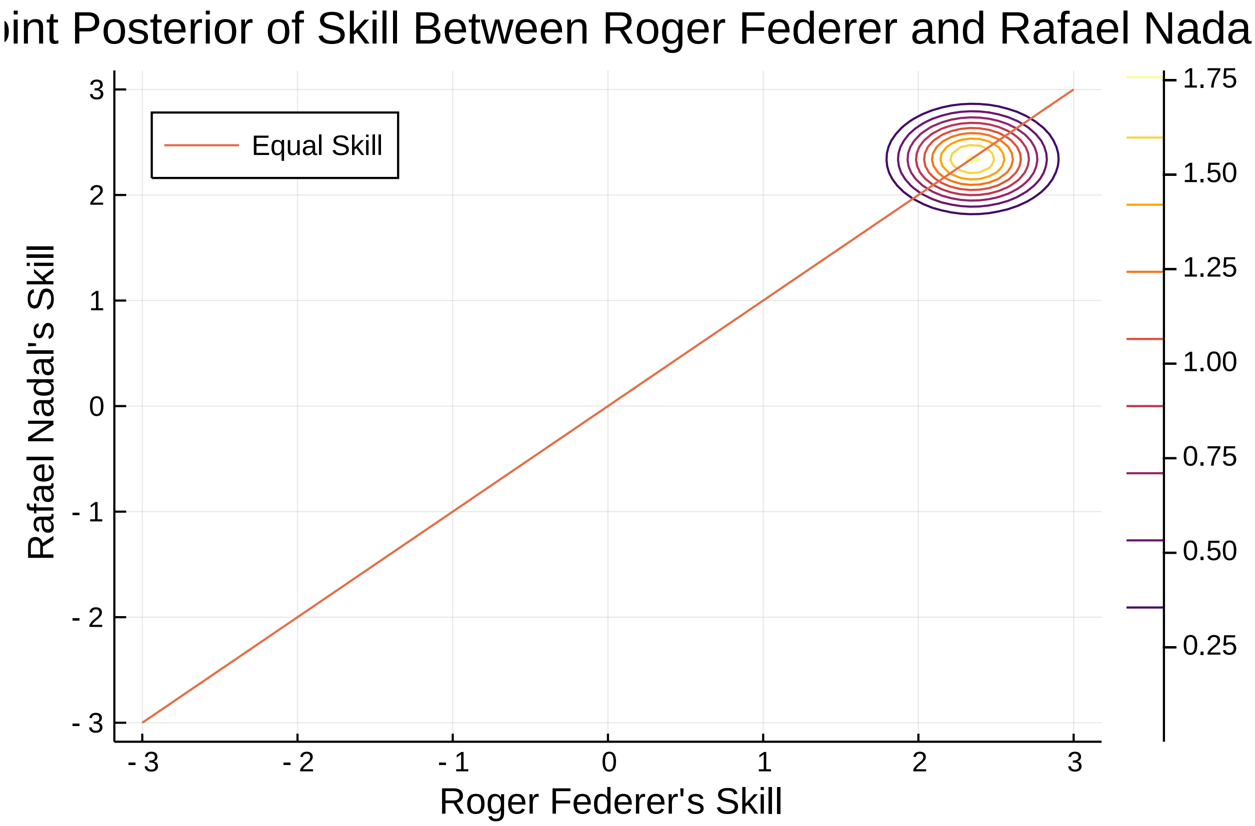

```
savefig(joinpath("plots","4c_approx_mean_var_all_players.pdf"))
```

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

```
top_ten_player_indices = reverse(perm)[1:10]
top_ten_player_names = player_names[top_ten_player_indices]
print(top_ten_player_names)

Any["Novak-Djokovic", "Roger-Federer", "Rafael-Nadal", "Andy-Murray", "Robi
n-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.

```
i_roger_federer = findall(x -> x == "Roger-Federer",player_names)[1][1]
i_rafael_nadal  = findall(x -> x == "Rafael-Nadal", player_names)[1][1]

rf_rn_mu  = [trained_params[1][i_roger_federer]; trained_params[1][i_rafael_nadal]]
rf_rn_logsig = [trained_params[2][i_roger_federer]; trained_params[2][i_rafael_nadal]]

plot(title = "Joint Posterior of Skill Between Roger Federer and Rafael Nadal",
    xlabel = "Roger Federer's Skill",
    ylabel = "Rafael Nadal's Skill",
    legend=:topleft)
rf_rn_joint_posterior(zs) = exp(factorized_gaussian_log_density(rf_rn_mu, rf_rn_logsig, zs))
skillcontour!(rf_rn_joint_posterior)
display(plot_line_equal_skill!())
```




```
savefig(joinpath("plots","4e_joint_posterior_rf_rn.pdf"))
```

6. [**5 points**] Derive the exact probability under a factorized Guassian over two players' skills that onehas 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 randomvariable.

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

since $p(z_A > z_B) = p(z_A - z_B > 0)$

let $y_A = z_A - z_B, y_B = z_B$

$$\begin{bmatrix} y_A \\ y_B \end{bmatrix} = \begin{bmatrix} z_A - z_B \\ z_B \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_A \\ z_B \end{bmatrix}$$

Then we have a linear transformation $A = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$

since $z \sim N(\mu, \Sigma)$,

then $y \sim N(A\mu, A\Sigma A^T)$

$$\begin{bmatrix} y_A \\ y_B \end{bmatrix} \sim N(\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma_2^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}^T)$$

$$\begin{bmatrix} y_A \\ y_B \end{bmatrix} \sim N(\begin{bmatrix} \mu_1 - \mu_2 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 + \sigma_2^2 & -\sigma_2^2 \\ -\sigma_2^2 & \sigma_2^2 \end{bmatrix})$$

which implies that $y_A \sim N(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2)$

therefore $p(y_A > 0) = 1 - p(y_A < 0) = 1 - \dfrac{1}{\sigma\sqrt{2\pi}} \displaystyle\int_{-\infty}^{x} e^{-\dfrac{x-\mu}{2\sigma^2}} dx$

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

```
using Random, Distributions
Random.seed!(123)

rf_rn_mu  = [trained_params[1][i_roger_federer]; trained_params[1][i_rafael_nadal]]
rf_rn_logsig = [trained_params[2][i_roger_federer]; trained_params[2][i_rafael_nadal]]

y_mu = rf_rn_mu[1] - rf_rn_mu[2]
y_sig = sqrt(exp(rf_rn_logsig[1])^2 + exp(rf_rn_logsig[2])^2)
p_fr_gt_rn = 1 - cdf(Normal(y_mu, y_sig), 0)

num_samples = 10000
fd_rn_samples = exp.(rf_rn_logsig) .* randn(2, num_samples) .+ rf_rn_mu
mc_fd_gt_rn = sum(fd_rn_samples[1, :] .> fd_rn_samples[2, :])/num_samples

println("Exact probability under approximate posterior:", p_fr_gt_rn)

Exact probability under approximate posterior:0.506807686178188

println("Simple Monte Carlo with 10000 samples under approximate posterior:", mc_fd_gt_rn)

Simple Monte Carlo with 10000 samples under approximate posterior:0.5015
```

8. [**2 points**] Using the formula from part c, compute the probability that Roger Federer is better thanthe player with the lowest mean skill. Compute this quantity exactly, and then estimate it using simpleMonte Carlo with 10000 examples, again using your approximate posterior.

```
rf_worst_mu  = [trained_params[1][i_roger_federer]; trained_params[1][perm[1]]]
rf_worst_logsig = [trained_params[2][i_roger_federer]; trained_params[2][perm[1]]]

y_mu = rf_worst_mu[1] - rf_worst_mu[2]
y_sig = sqrt(exp(rf_worst_logsig[1])^2 + exp(rf_worst_logsig[2])^2)
p_fr_gt_worst = 1 - cdf(Normal(y_mu, y_sig), 0)

num_samples = 10000
```

```
fd_worst_samples = exp.(rf_worst_logsig) .* randn(2, num_samples) .+ rf_worst_mu
mc_fd_gt_worst = sum(fd_worst_samples[1, :] .> fd_worst_samples[2, :])/num_samples

println("Exact probability under approximate posterior:", p_fr_gt_worst)

Exact probability under approximate posterior:0.9999999990273677

println("Simple Monte Carlo with 10000 samples under approximate posterior:",
mc_fd_gt_worst)

Simple Monte Carlo with 10000 samples under approximate posterior: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 thischange? No need to show your work, just list the letters of the questions whose answers would bedifferent in expectation.

This will change the answers for part (c).