Assignment 2

Mohammad Saad Khan - 1004138818

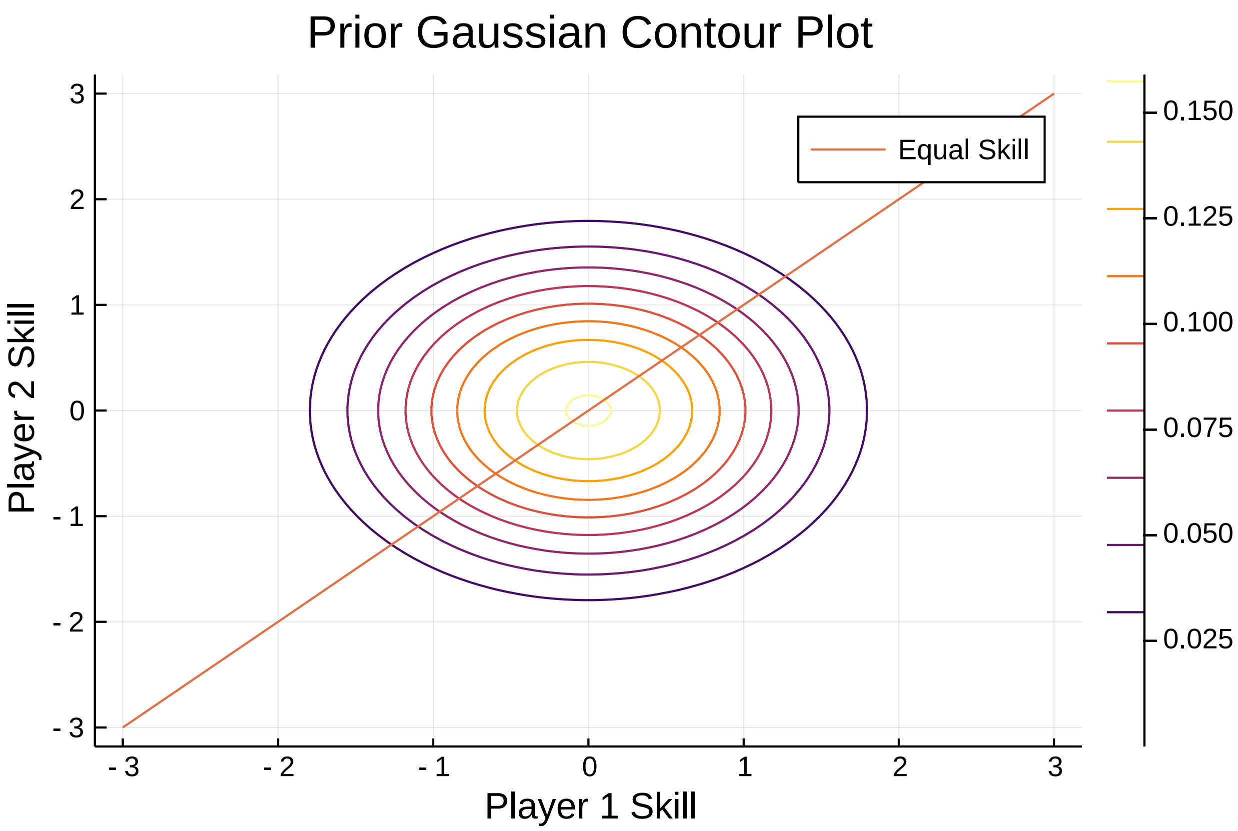
March 21, 2020

1 Implementing the Model

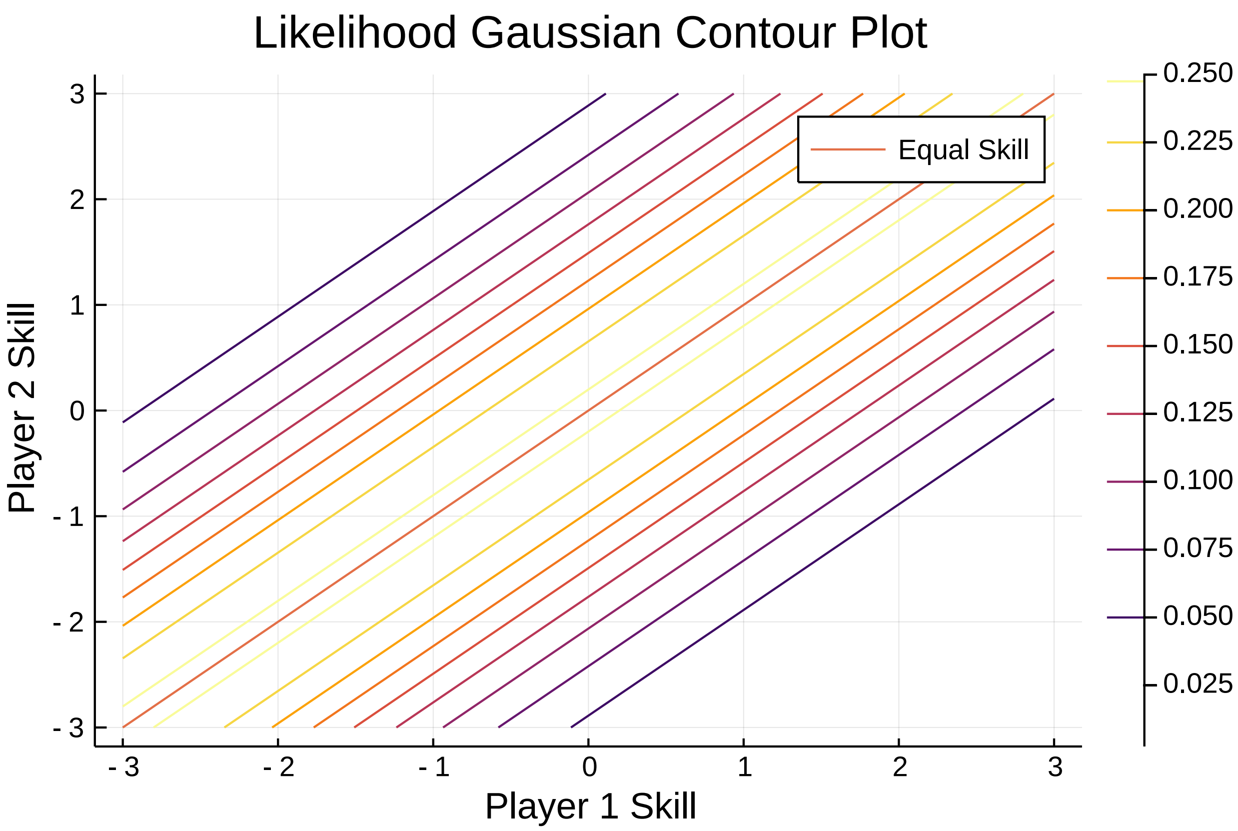
```
function log_prior(zs)
  return factorized_gaussian_log_density(0, 0, zs)
end


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


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


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

2 Examining the Posterior

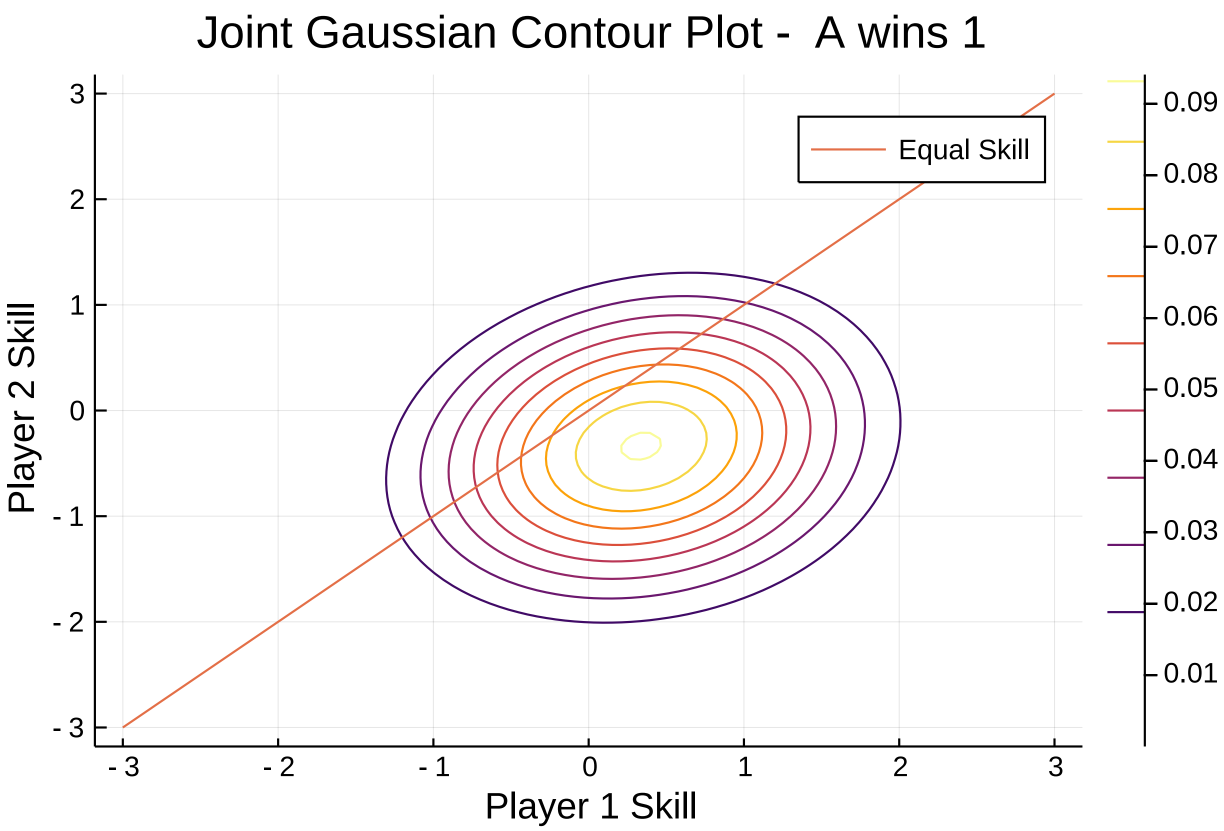
```
plot(title="Prior Gaussian Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
prior_gaussian(zs) = exp(log_prior(zs))
skillcontour!(prior_gaussian)
plot_line_equal_skill!()
```



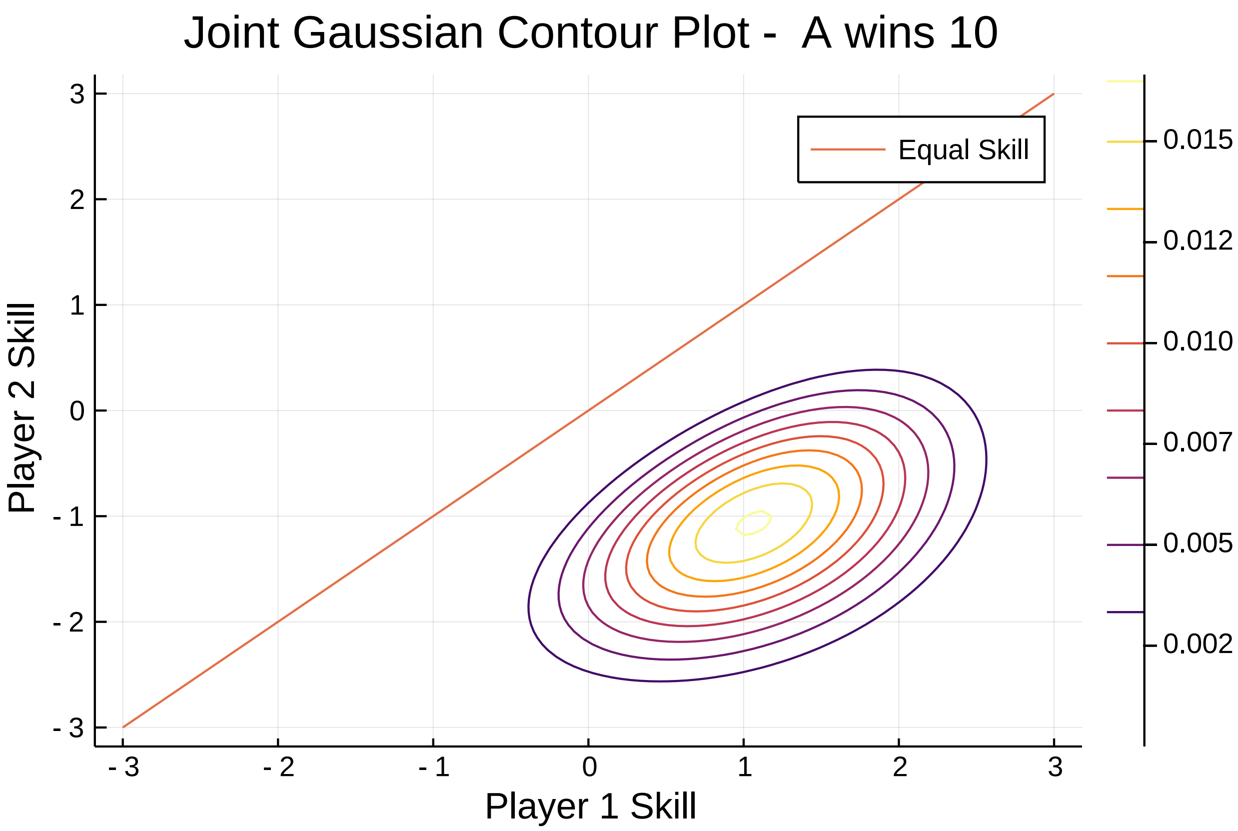

```
plot(title="Likelihood Gaussian Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
likelihood_gaussian(zs) = exp(all_games_log_likelihood(zs, two_player_toy_games(1, 1)))
skillcontour!(likelihood_gaussian)
plot_line_equal_skill!()
```



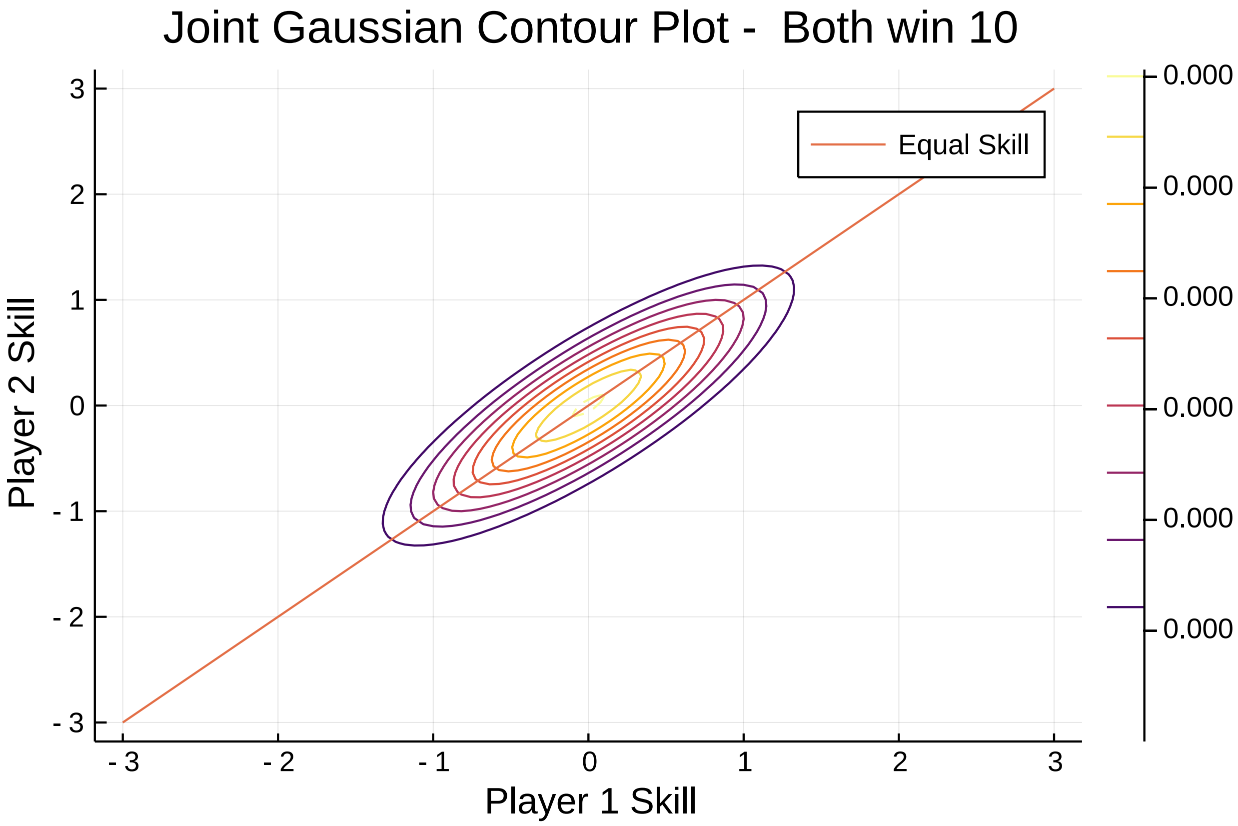

```
plot(title="Joint Gaussian Contour Plot - A wins 1",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
joint_gaussian_A1(zs) = exp(joint_log_density(zs, two_player_toy_games(1, 0)))
skillcontour!(joint_gaussian_A1)
plot_line_equal_skill!()
```




```
plot(title="Joint Gaussian Contour Plot - A wins 10",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
joint_gaussian_A10(zs) = exp(joint_log_density(zs, two_player_toy_games(10, 0)))
skillcontour!(joint_gaussian_A10)
plot_line_equal_skill!()
```




```
plot(title="Joint Gaussian Contour Plot - Both win 10",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
joint_gaussian_A10B10(zs) = exp(joint_log_density(zs, two_player_toy_games(10, 10)))
skillcontour!(joint_gaussian_A10B10)
plot_line_equal_skill!()
```



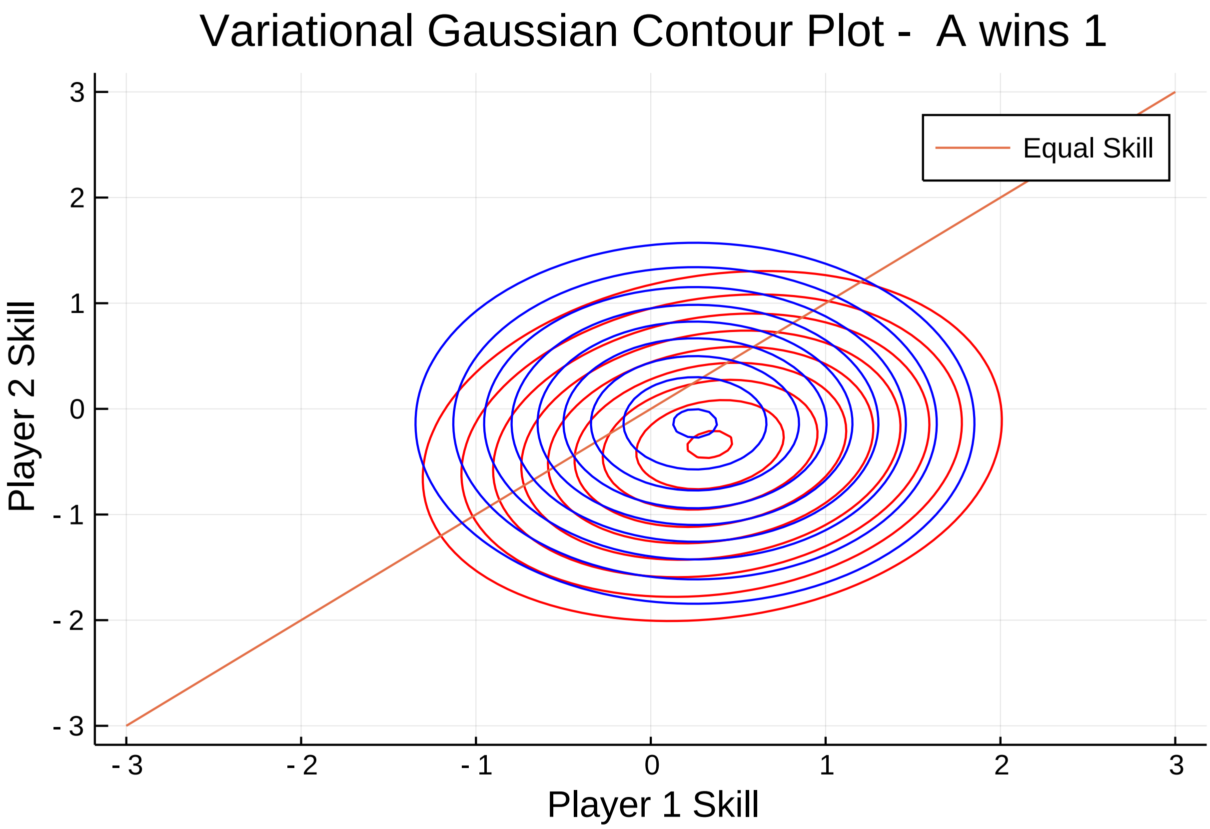
3 Stochastic Variational Inference

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

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

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]

    params_cur =  params_cur .- (lr .* grad_params)
    @info neg_toy_elbo(params_cur, games=toy_evidence, num_samples=num_q_samples)
    plot();
    p(zs) = exp(joint_log_density(zs,toy_evidence))
    skillcontour!(p,colour=:red) # plot likelihood contours for target posterior
    plot_line_equal_skill!()
    q(zs) = exp(factorized_gaussian_log_density(params_cur[1], params_cur[2], zs))
    # display(skillcontour!(q, colour=:blue)) # plot likelihood contours for variational
posterior
  end
```

```
  return params_cur, neg_toy_elbo(params_cur, games=toy_evidence,
num_samples=num_q_samples)
end

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)

games = two_player_toy_games(1, 0)
fitted_A1_params, loss = fit_toy_variational_dist(toy_params_init, games)
print(loss)
```

```
0.8173925834300657
```

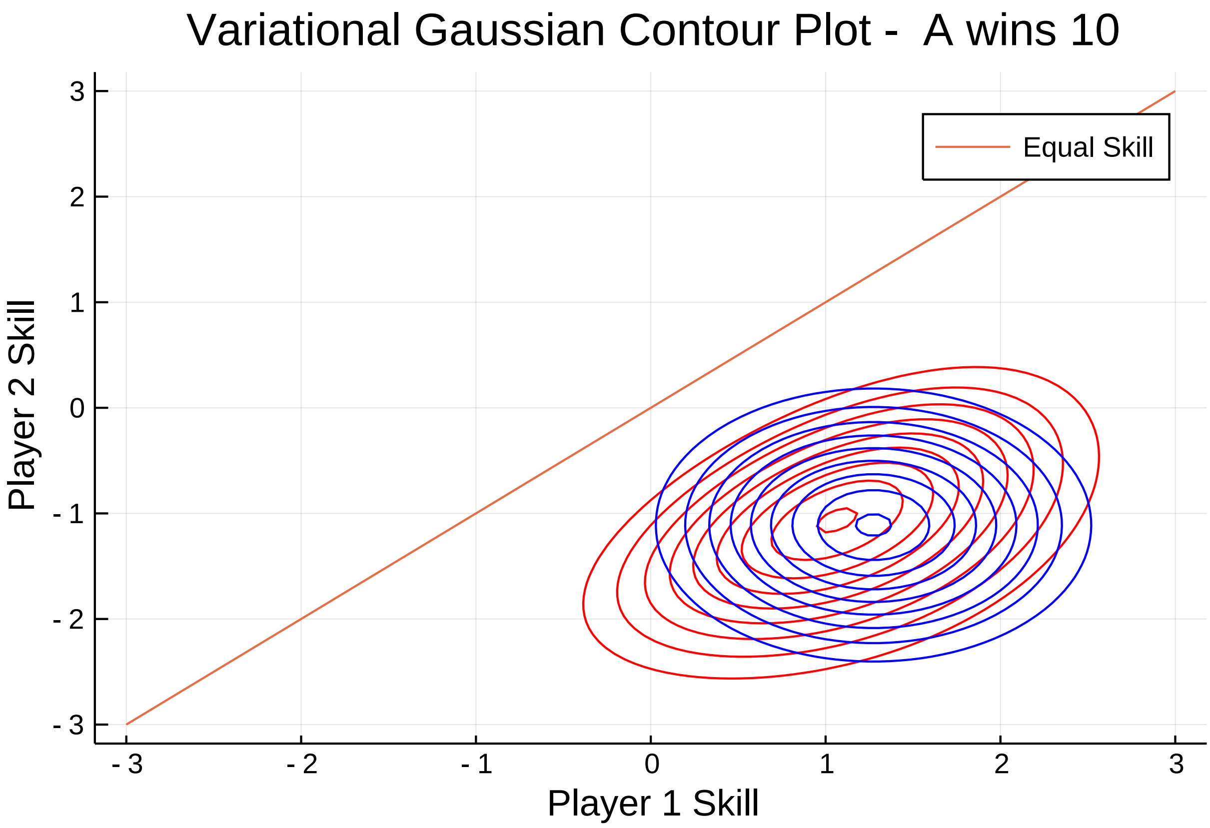
```
plot(title="Variational Gaussian Contour Plot - A wins 1",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   );
p(zs) = exp(joint_log_density(zs,games))
skillcontour!(p,colour=:red) # plot likelihood contours for target posterior
plot_line_equal_skill!()
q(zs) = exp(factorized_gaussian_log_density(fitted_A1_params[1], fitted_A1_params[2],
zs))
skillcontour!(q, colour=:blue)
```




```
games = two_player_toy_games(10, 0)
fitted_A10_params, loss = fit_toy_variational_dist(toy_params_init, games)
print(loss)
```

```
2.987927859284215
```

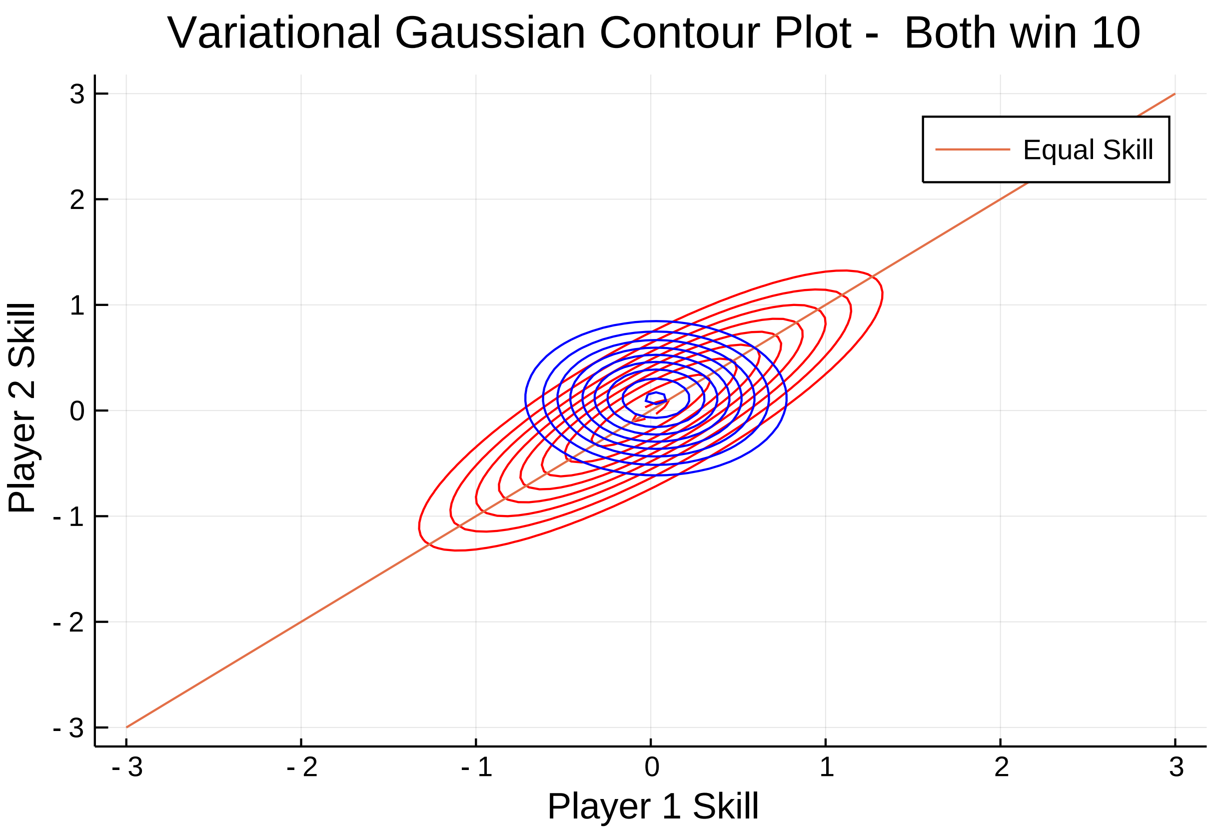
```
plot(title="Variational Gaussian Contour Plot - A wins 10",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   );
p(zs) = exp(joint_log_density(zs,games))
skillcontour!(p,colour=:red) # plot likelihood contours for target posterior
plot_line_equal_skill!()
q(zs) = exp(factorized_gaussian_log_density(fitted_A10_params[1], fitted_A10_params[2],
zs))
skillcontour!(q, colour=:blue)
```




```
games = two_player_toy_games(10, 10)
fitted_A10B10_params, loss = fit_toy_variational_dist(toy_params_init, games)
print(loss)
```

```
15.666049643567101
```

```
plot(title="Variational Gaussian Contour Plot - Both win 10",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   );
p(zs) = exp(joint_log_density(zs,games))
skillcontour!(p,colour=:red) # plot likelihood contours for target posterior
plot_line_equal_skill!()
q(zs) = exp(factorized_gaussian_log_density(fitted_A10B10_params[1],
fitted_A10B10_params[2], zs))
skillcontour!(q, colour=:blue)
```



4 Approximate Inference

(a) Yes, the games of other players besides i and j provides information about their skills

```
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 neg_toy_elbo(params_cur, games=tennis_games, num_samples=num_q_samples)
  end
  return params_cur, loss
end
```

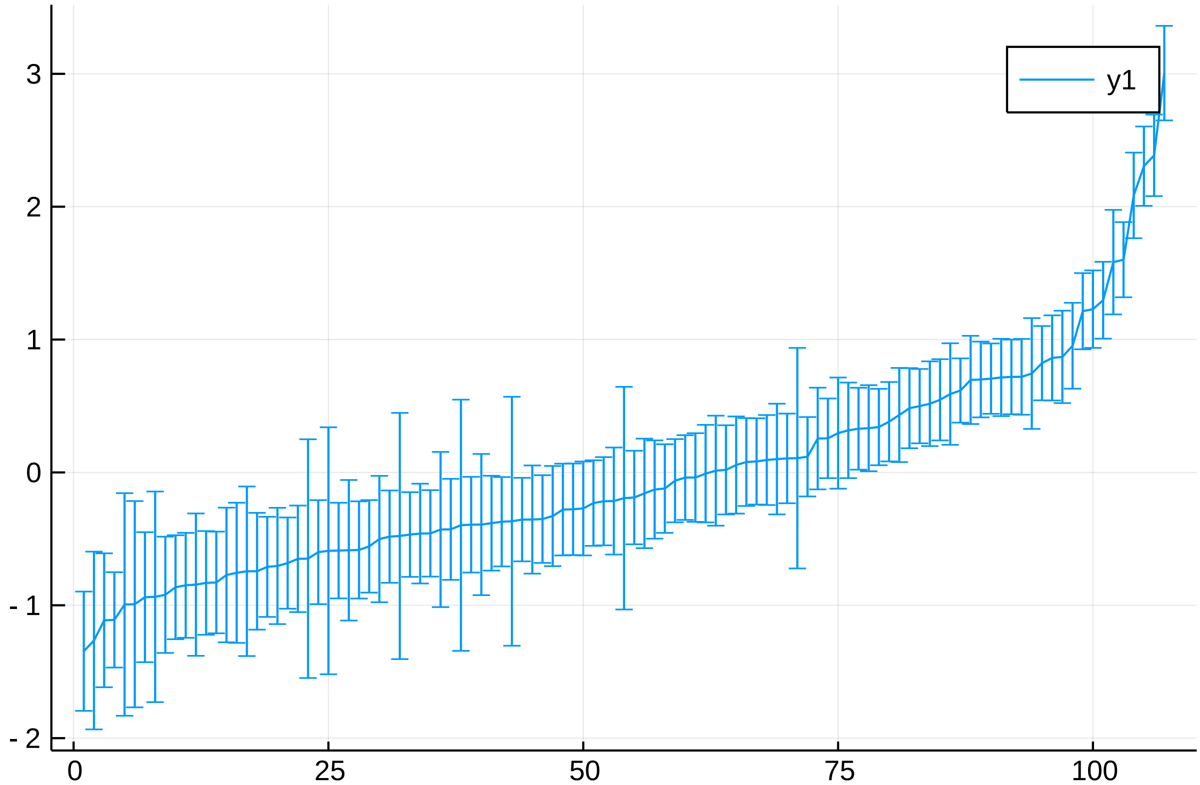
Fitting distribution to observed dataset

```
init_mu = randn(length(player_names))
init_log_sigma = randn(length(player_names))
init_params = (init_mu, init_log_sigma)

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

```
15.666049643567101
```

Plotting approximate mean and variance of all players

```
perm = sortperm(trained_params[1])
plot(trained_params[1][perm], yerror=exp.(trained_params[2][perm]))
```



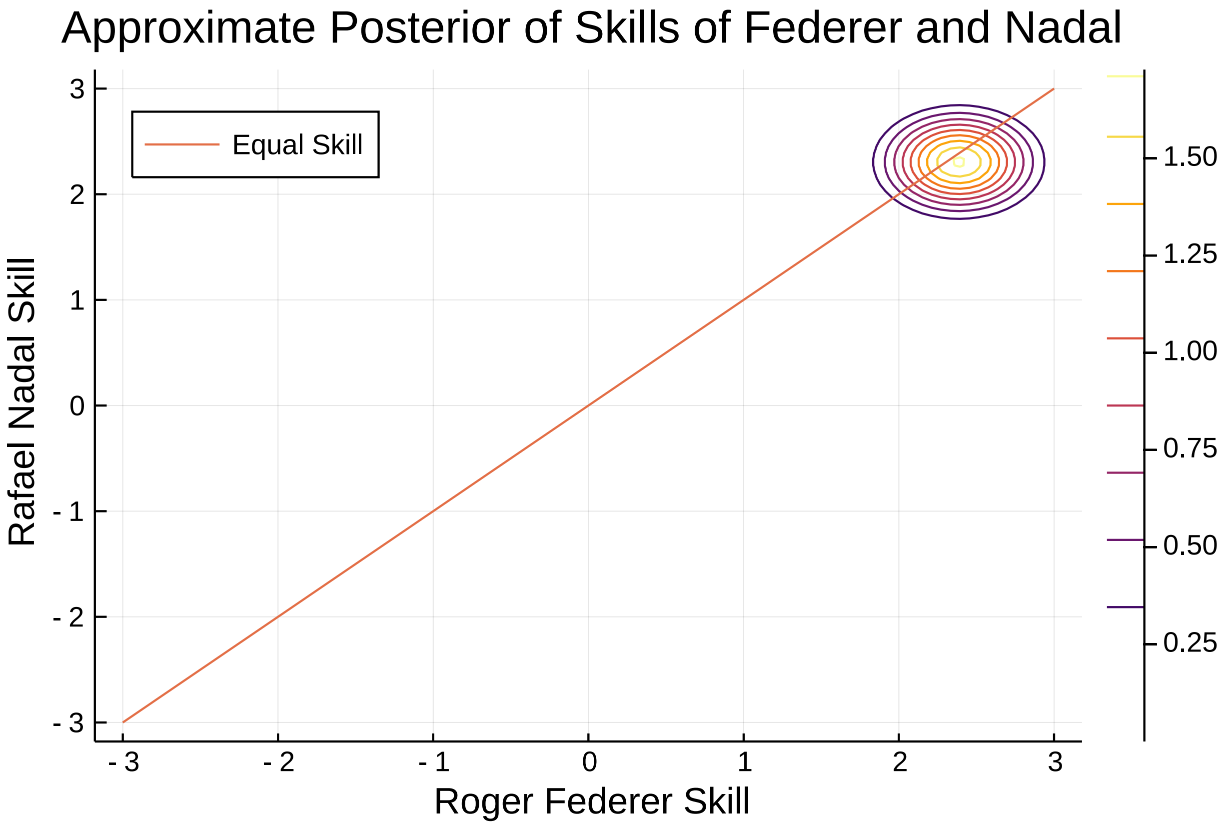
Printing top 10 players by mean skill

```
print(player_names[perm][end-9:end])
```

```
Any["Richard-Gasquet", "Juan-Martin-Del-Potro", "Tomas-Berdych", "Jo-Wilfri
ed-Tsonga", "Robin-Soderling", "David-Ferrer", "Andy-Murray", "Rafael-Nadal
", "Roger-Federer", "Novak-Djokovic"]
```

Plotting joint distribution over Nadal and Federer

```
rog_index = findall(x->x=="Roger-Federer", player_names)
raf_index = findall(x->x=="Rafael-Nadal", player_names)

plot(legend=:topleft,
    title="Approximate Posterior of Skills of Federer and Nadal",
    xlabel = "Roger Federer Skill",
    ylabel = "Rafael Nadal Skill"
   );
means = [trained_params[1][rog_index]; trained_params[1][raf_index]]
logsigs = [trained_params[2][rog_index]; trained_params[2][raf_index]]
dist(zs) = exp.(factorized_gaussian_log_density(means, logsigs, zs))
skillcontour!(dist)
plot_line_equal_skill!()
```



To find exact probability that one player has higher skill than another. Consider two players with skills z_A and z_B and from a distribution with mean μ and variance Σ. Firstly, consider the matrix $A = \begin{matrix} 1 & -1 \\ 0 & 1 \end{matrix}$.

As in the hint, we can use this matrix transformation to create new variables $y = Az$. We know that $y_A = z_A - z_B$ and $y_B = z_B$ because of the matrix above.

Also, because A is a linear transformation and $z \sim N(\mu, \Sigma)$, we know $y \sim N(A\mu, A\Sigma A^T)$ because $y = Az$.

Then, we specifically want to find $P(z_A - z_B > 0) = P(y_A > 0)$. Since we are using a factorized gaussian distribution, we know $y_A \sim N(y_A, [A\Sigma A^T]_{11})$

We can use the reparameterization trick to express this as $P(y_A + [A\Sigma A^T]_{11}N > 0)$ where N is the standard normal.

Finally, $P(y_A + [A\Sigma A^T]_{11}N > 0) = P[A\Sigma A^T]_{11}N > -y_A) = 1 - P(N \leq \frac{-y_A}{[A\Sigma A^T]_{11}})$

```
function transform_skills(mu, logsig)
  A = [1 -1; 0 1]
  ya = mu[1] - mu[2]
  sig = Diagonal(exp.(logsig))
  cov = A * sig * A'
  var = sqrt(cov[1])
  threshold = -ya/var
  return 1 - cdf(Normal(0, 1), threshold)
end

function monte_carlo(mu, logsigs)
  num_samples = 10000
  samples = (randn(2, num_samples) .* exp.(logsigs)) .+ mu
  diff = samples[1,:] - samples[2,:]
```

```
  return count(x -> x > 0, diff)/num_samples
end
```

First, we can find the probability that Federer is better than Nadal:

```
print(transform_skills(means, logsigs))
```

```
0.5416807784390549
```

```
print(monte_carlo(means, logsigs))
```

```
0.5818
```

Then, we can compare Federer to the worst player

```
means = [trained_params[1][rog_index]; trained_params[1][perm[1]]]
logsigs = [trained_params[2][rog_index]; trained_params[2][perm[1]]]

print(transform_skills(means, logsigs))
```

```
0.9999911763364223
```

```
print(monte_carlo(means, logsigs))
```

```
1.0
```

If we changed the prior to Normal(10, 1), then the sections that would change are (b), (c), (e), (g), (h)