Jinhan Mei
1003137937

STA414 Assignment 2

Q1 Implementing the model

(a) Implementing log_prior.

```
using Revise # lets you change A2funcs without restarting julia!
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!

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

(b) Implementing logp_a_beats_b.

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

(c) Implementing all_games_log_likelihood.

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

(d) Implementing joint_log_density.

```
function joint_log_density(zs,games)
  return sum(log_prior(zs) .+ all_games_log_likelihood(zs, games), dims=1)
end
```

Jinhan Mei
1003137937

```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.DefaultTestSet
```

REPL

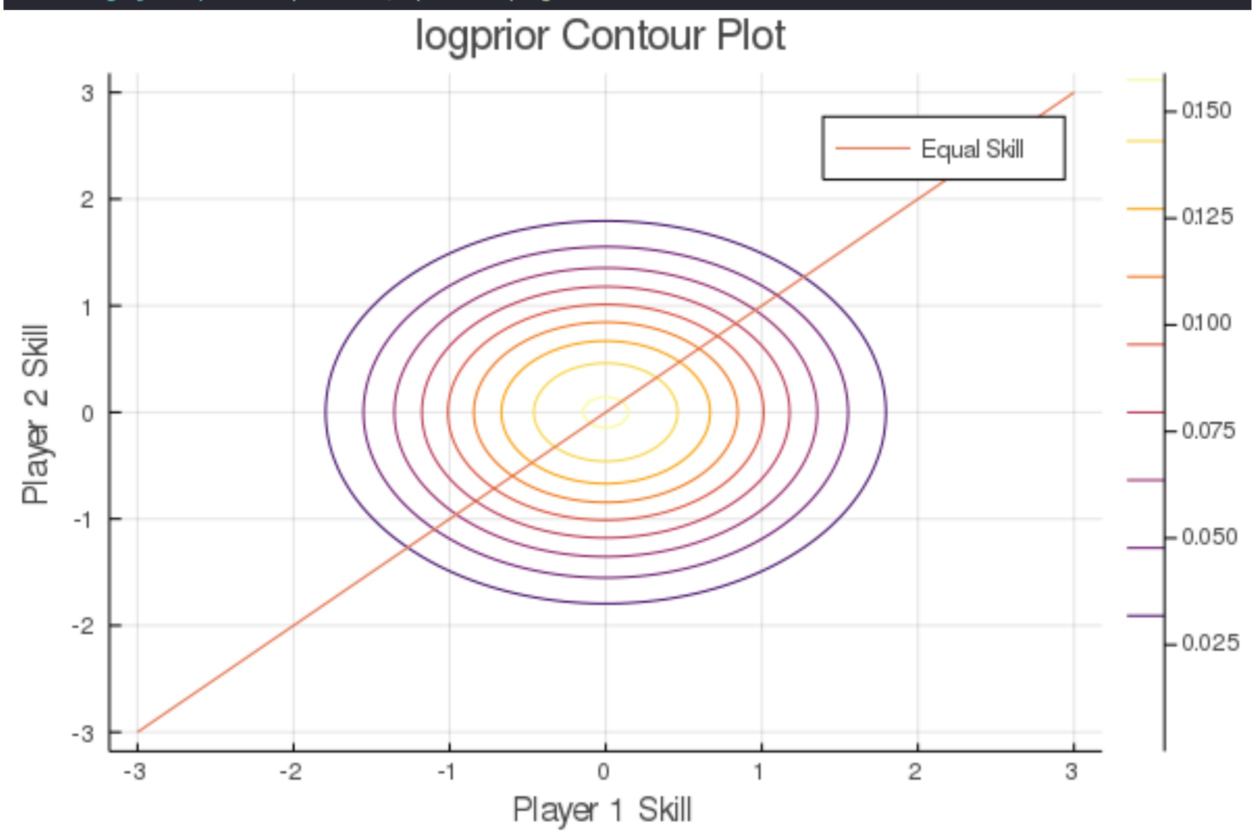
```
Test Summary:                          | Pass  Total
Test shapes of batches for likelihoods |    6      6
```

Jinhan Mei
1003137937

Q2 Examining the posterior for only two players and toy data

(a) plot the log prior graph.

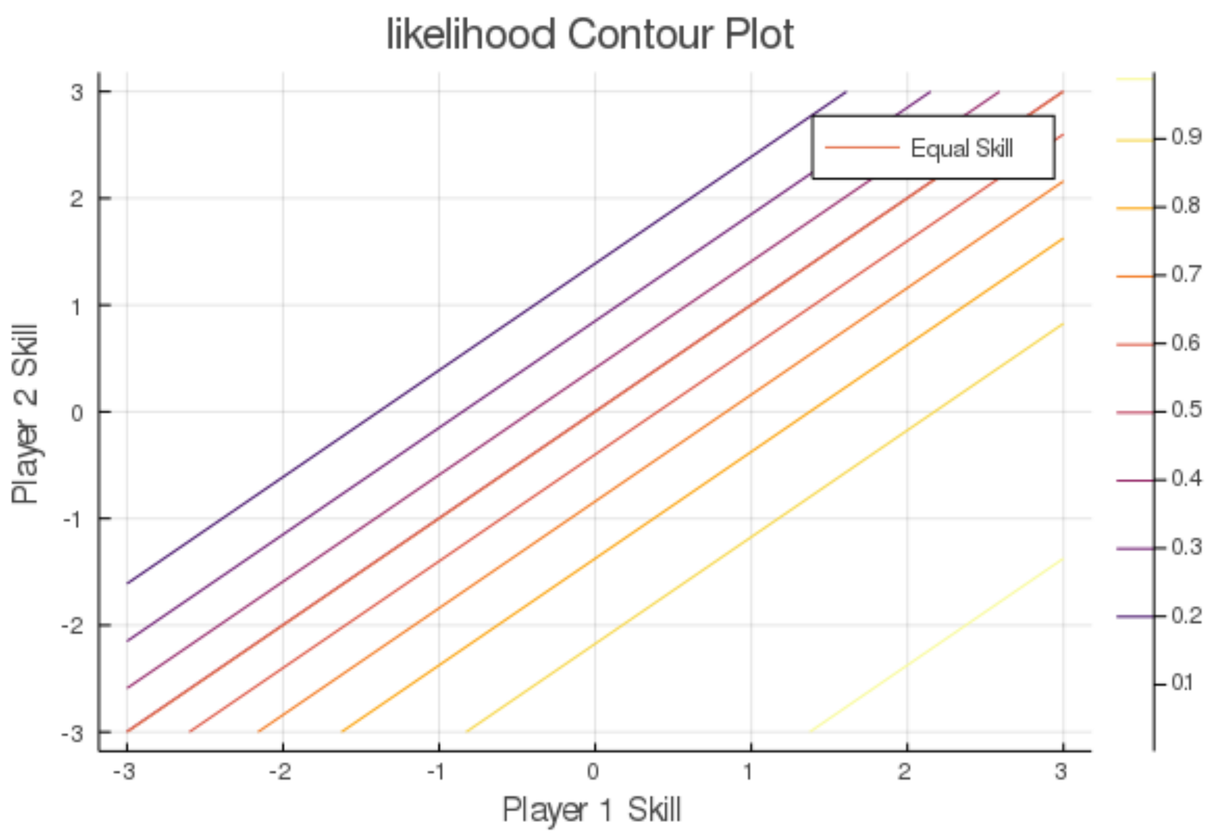
```
# Q2(a)
plot(title="logprior Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )

log_p(zs) = exp(log_prior(zs))
skillcontour!(log_p)
plot_line_equal_skill!()
savefig(joinpath("plots","prior.png"))
```



Jinhan Mei
1003137937

(b) Plot the log likelihood Contour Plot.

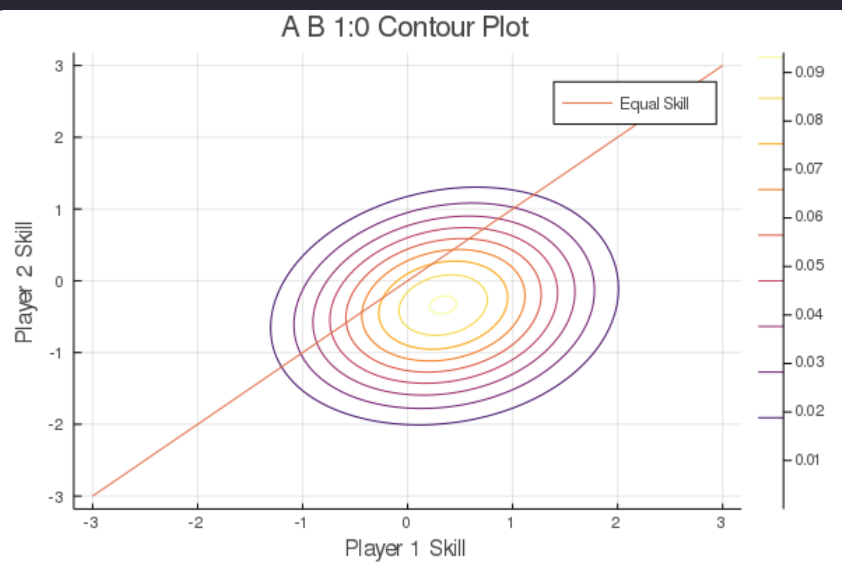
```
plot(title="likelihood Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
skillcontour!(zs -> exp.(logp_a_beats_b(zs[1], zs[2])))
plot_line_equal_skill!()
savefig(joinpath("plots","likelihood.png"))
```



Jinhan Mei
1003137937

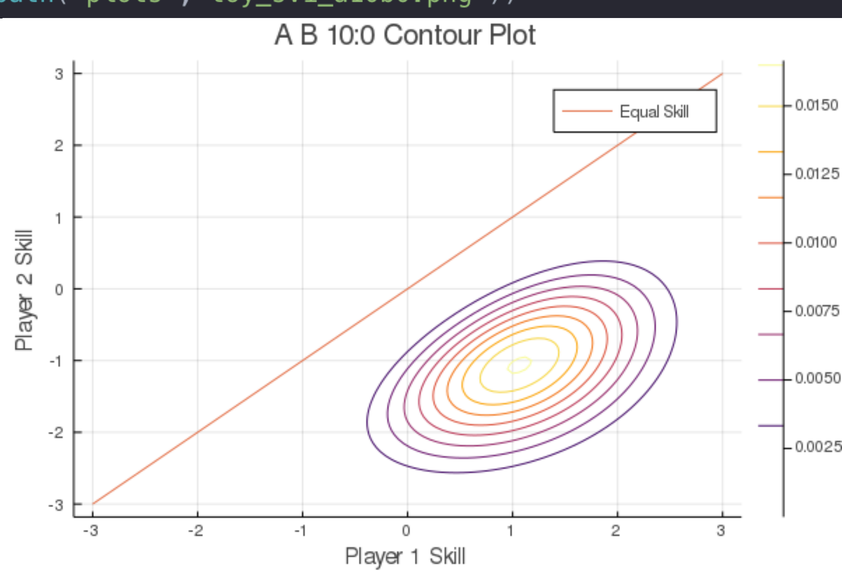
(c) Plot the Contour of A B 1:0.

```
# Q2(c)
plot(title="A B 1:0 Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
skillcontour!(zs -> exp.(joint_log_density(zs, two_player_toy_games(1,0))))
plot_line_equal_skill!()
savefig(joinpath("plots","contour_a1b0.png"))
```



(d) Plot Contour of A B 10:0.

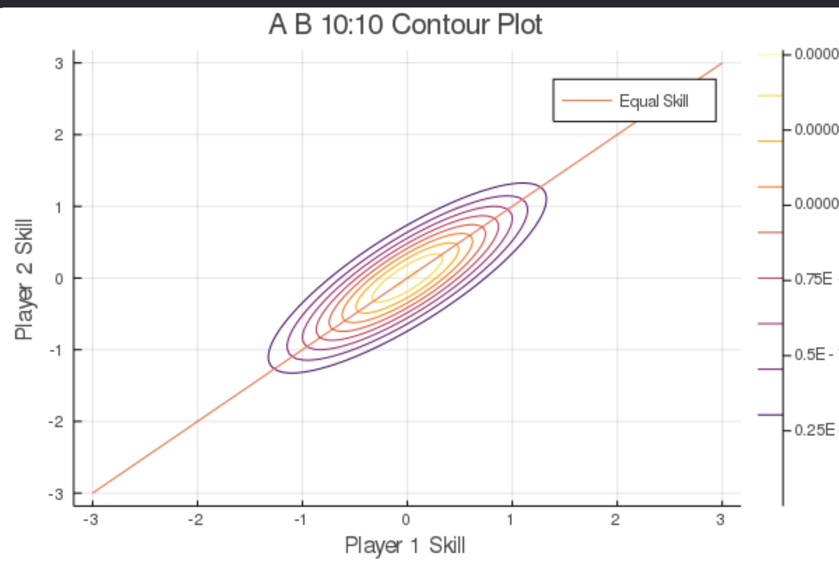
```
plot(title="A B 10:0 Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
skillcontour!(zs -> exp.(joint_log_density(zs, two_player_toy_games(10,0))))
plot_line_equal_skill!()
savefig(joinpath("plots","toy_svi_a10b0.png"))
```



Jinhan Mei
1003137937

(e) Plot Contour of A B 10:10.

```
plot(title="A B 10:10 Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
skillcontour!(zs -> exp.(joint_log_density(zs, two_player_toy_games(10,10))))
plot_line_equal_skill!()
savefig(joinpath("plots","toy_svi_a10b10.png"))
```



Jinhan Mei
1003137937

Q3 Stochastic Variational Inference on Two Players and Toy Data

(a) Implement elbo.

```
function elbo(params,logp,num_samples)
  samples = exp.(params[2]) .*randn(size(params[1])[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
```

(b) Implement neg_toy_elbo.

```
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
```

(c) Implement fit_toy_variational_dist.

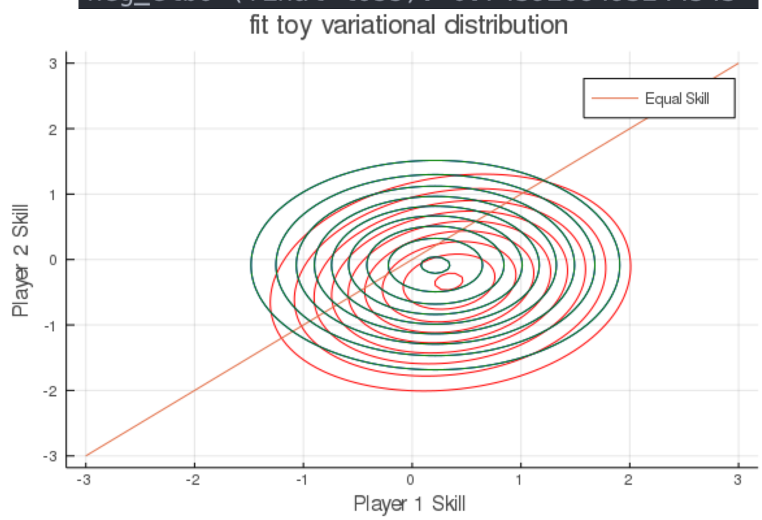
```
function fit_toy_variational_dist(init_params, toy_evidence; num_itrs=200,
  lr= 1e-2, num_q_samples = 10)
  params_cur = init_params
  for i in 1:num_itrs
    grad_params = gradient(params -> neg_toy_elbo(params; games = toy_evidence,
    num_samples = num_q_samples), params_cur)[1]
    mu_new =  params_cur[1] - lr * grad_params[1]
    ls_new =  params_cur[2] - lr * grad_params[2]
    params_cur = (mu_new, ls_new)
    # @info "neg_elbo: $(neg_toy_elbo(params_cur; games = toy_evidence,
    # num_samples = num_q_samples))"
    plot(title="fit toy variational distribution", xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill");

    target_post(zs) = exp.(joint_log_density(zs, toy_evidence))
    skillcontour!(target_post,colour=:red)
    plot_line_equal_skill!()
    mu = params_cur[1]
    logsig = params_cur[2]
    var_log_prior(zs) = factorized_gaussian_log_density(mu, logsig, zs)
    var_post(zs) = exp.(var_log_prior(zs))
    display(skillcontour!(var_post, colour=:blue))
  end
  return params_cur
end
```

Jinhan Mei
1003137937

(d) Optimized variational approximation contours Plot of A B 1:0.

```
#Q1(d) 1:0
plot(title="A B 1:0 Toy SVI",
  xlabel = "Player 1 Skill",
  ylabel = "Player 2 Skill"
   )
# Plot the original graph
target_post(zs) = exp.(joint_log_density(zs, two_player_toy_games(1,0)))
skillcontour!(target_post,colour=:red)
plot_line_equal_skill!()

# Plot SVI
num_q_samples = 10
data = two_player_toy_games(1,0)
opt_params = fit_toy_variational_dist(toy_params_init, two_player_toy_games(1,0), num_itrs=200, lr= 1e-2, num_q_samples = 10)
println("neg_elbo (final loss): $(neg_toy_elbo(opt_params; games = two_player_toy_games(1,0), num_samples = num_q_samples))")
mu = opt_params[1]
logsig = opt_params[2]
var_log_prior(zs) = factorized_gaussian_log_density(mu, logsig, zs)
var_post(zs) = exp.(var_log_prior(zs))
display(skillcontour!(var_post, colour=:green))
savefig(joinpath("plots","toy_svi_a1b0.png"))
```

```
neg_elbo (final loss): 0.7439268493244543
```



(e) Optimized variational approximation contours Plot of A B 10:0.

```
plot(title="A B 10:0 Toy SVI",
  xlabel = "Player 1 Skill",
  ylabel = "Player 2 Skill"
   )
target_post(zs) = exp.(joint_log_density(zs, two_player_toy_games(10,0)))
skillcontour!(target_post,colour=:red)
plot_line_equal_skill!()

# Plot the new graph
opt_params = fit_toy_variational_dist(toy_params_init, two_player_toy_games(10,0); num_itrs=200, lr= 1e-2, num_q_samples = 10)
num_q_samples = 10
println("neg_elbo (final loss): $(neg_toy_elbo(opt_params; games = two_player_toy_games(10,0), num_samples = num_q_samples))")
mu = opt_params[1]
logsig = opt_params[2]
var_log_prior(zs) = factorized_gaussian_log_density(mu, logsig, zs)
var_post(zs) = exp.(var_log_prior(zs))
display(skillcontour!(var_post, colour=:green))
savefig(joinpath("plots","toy_svi_a10b0.png"))
```

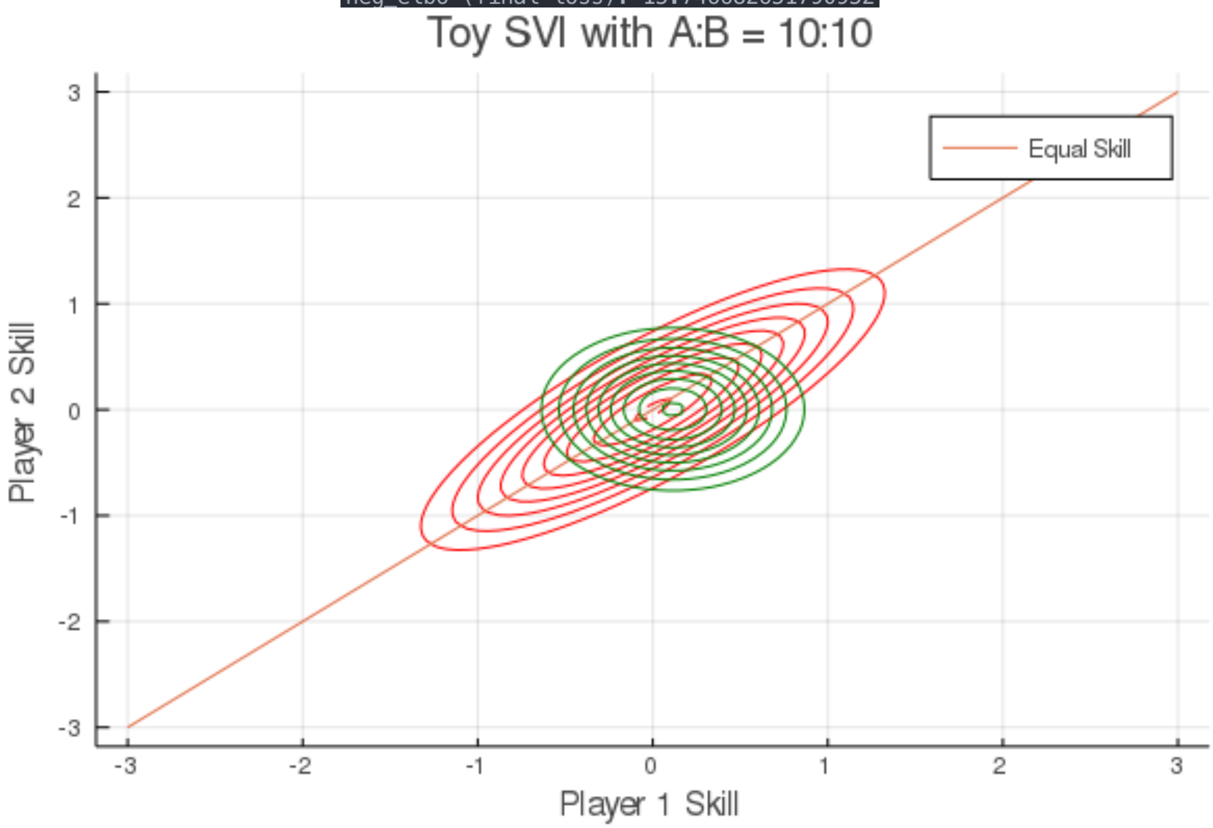
```
neg_elbo (final loss): 2.9131816235518913
```

Jinhan Mei
1003137937

(f) Optimized variational approximation contours Plot of A B 10:10

```
opt_params = fit_toy_variational_dist(toy_params_init, two_player_toy_games(10,10); num_itrs=200, lr= 1e-2, num_q_samples = 10)
num_q_samples = 10
println("neg_elbo (final loss): $(neg_toy_elbo(opt_params; games = two_player_toy_games(10,10), num_samples = num_q_samples))")

plot(title="Toy SVI with A:B = 10:10",
  xlabel = "Player 1 Skill",
  ylabel = "Player 2 Skill"
   )
target_post(zs) = exp.(joint_log_density(zs, two_player_toy_games(10,10)))
skillcontour!(target_post,colour=:red)
plot_line_equal_skill!()

mu = opt_params[1]
logsig = opt_params[2]
var_log_prior(zs) = factorized_gaussian_log_density(mu, logsig, zs)
var_post(zs) = exp.(var_log_prior(zs))
display(skillcontour!(var_post, colour=:green))
savefig(joinpath("plots","toy_svi_a10b10.png"))
```

```
neg_elbo (final loss): 15.746682651790952
```



Q4 Approximate inference conditioned on real data

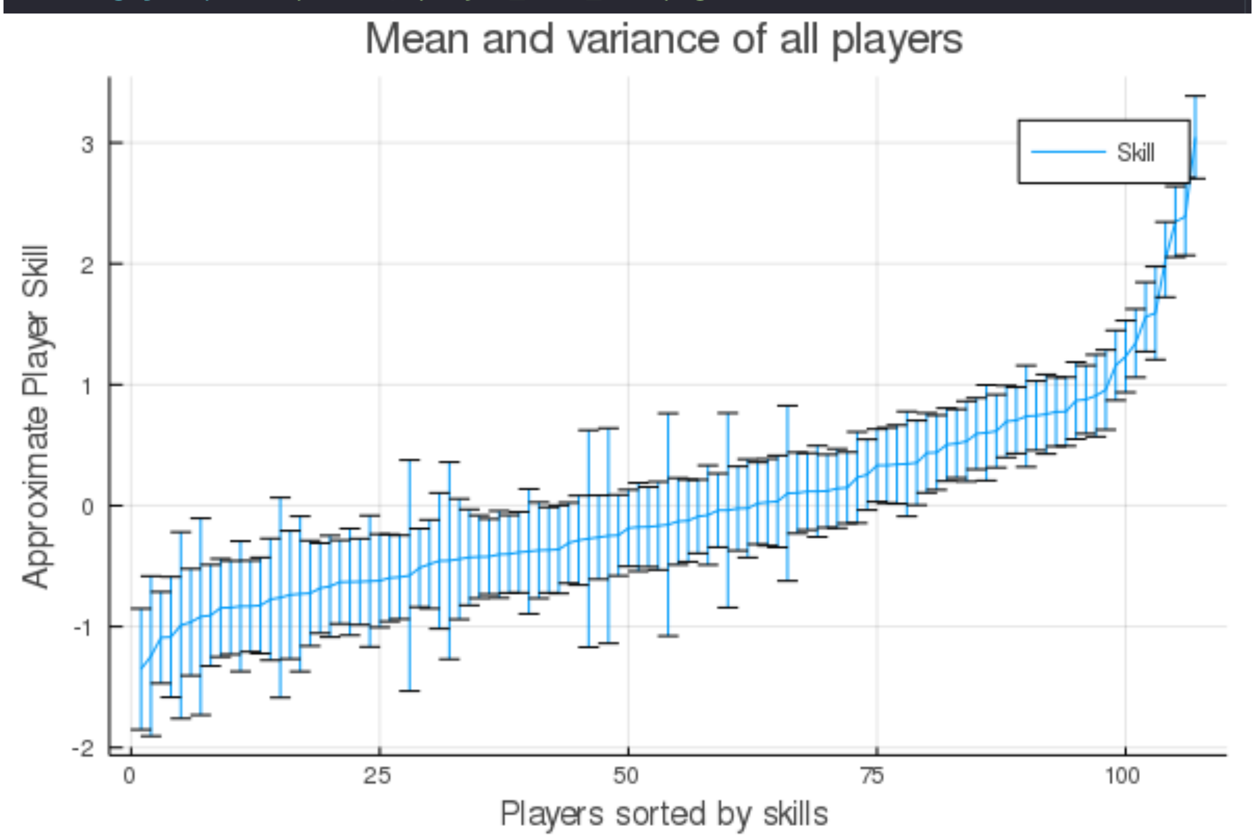
(a) Yes.

(b) loss: 1185.287000374451

```
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]
    mu_new =  params_cur[1] - lr * grad_params[1]
    ls_new =  params_cur[2] - lr * grad_params[2]
    params_cur = (mu_new, ls_new)
    @info "neg_elbo: $(neg_toy_elbo(params_cur; games = tennis_games, num_samples = num_q_samples))"
  end
  println("neg_elbo (final loss): $(neg_toy_elbo(params_cur; games = tennis_games, num_samples = num_q_samples))")
  return params_cur
end  › fit_variational_dist
```

(c)

```
perm = sortperm(means)
plot(title="Mean and variance of all players",
  xlabel = "Players sorted by skills",
  ylabel = "Approximate Player Skill"
   )
plot!(means[perm], yerror=exp.(logstd[perm]), label="Skill")
savefig(joinpath("plots","player_mean_var.png"))
```



Jinhan Mei
1003137937

(d)

```
desc_perm = sortperm(means, rev=true)
print("Top 10 players are: ")
for i in 1:10
  print(player_names[desc_perm][i],"\n")
end
```

```
Top 10 players are: Novak-Djokovic
Roger-Federer
Rafael-Nadal
Andy-Murray
Robin-Soderling
David-Ferrer
Jo-Wilfried-Tsonga
Tomas-Berdych
Juan-Martin-Del-Potro
Richard-Gasquet
```

(f)

f)

$$A\begin{pmatrix} Z_A \\ Z_B \end{pmatrix} = \begin{pmatrix} Z_A - Z_B \\ Z_B \end{pmatrix} = \begin{pmatrix} Y_a \\ Y_b \end{pmatrix}$$

solve for A, $A = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$

since $\begin{pmatrix} Z_a \\ Z_b \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}, \begin{pmatrix} \sigma_{11} & 0 \\ 0 & \sigma_{22} \end{pmatrix}\right) = N(\mu, \Sigma)$

$$\Rightarrow \begin{pmatrix} Y_a \\ Y_b \end{pmatrix} \sim N(A\mu, A\Sigma A') = N\left(\begin{pmatrix} \mu_A - \mu_B \\ \mu_B \end{pmatrix}, \begin{pmatrix} \sigma_{11}+\sigma_{22} & -\sigma_{22} \\ -\sigma_{22} & \sigma_{22} \end{pmatrix}\right)$$

$$\Rightarrow P(Y_a|X) \sim N(\mu_a - \mu_b, \sigma_{11}+\sigma_{22})$$

$$\Rightarrow \text{let } y_a = N(\mu_A - \mu_b, \sigma_{11}+\sigma_{22})$$

$$P(y_a > 0) = 1 - \text{cdf}(y_a, 0)$$

(g)

```
# Q4(g)
# Exact apporach
exact_rf_better = 1 - cdf(Normal(0,1), (mu_RN - mu_RF)/sqrt(var_RF + var_RN))
# SM apporach
MC_size = 10000
samples_RF = randn(MC_size) * exp(-1.1412578223971224) .+ mu_RF
samples_RN = randn(MC_size) * exp(-1.2163985016732244) .+ mu_RN
MC_rf_better = count(x->x==1,samples_RF .> samples_RN) / MC_size
```

```
The result of SM of RF is bette than RN is: 0.5755
```

Jinhan Mei
1003137937

(h)

```
# Q4(h)
lowest_idx = desc_perm[num_players]
mu_lowest = means[lowest_idx]
var_lowest = exp(logstd[lowest_idx])^2
# Exact prob
exact_rf_better_than_worst = 1 - cdf(Normal(0,1), (mu_lowest - mu_RF)/sqrt(var_RF + var_lowest))
# SM apporach
samples_lowest = randn(MC_size) * exp(logstd[lowest_idx]) .+ mu_lowest
MC_h = count(x->x==1,samples_RF .> samples_lowest) / MC_size
```

```
The result of SM of RF is better the worst is: 1.0
```

(i)

(b), (c), and (e)

Jinhan Mei
1003137937

Jinhan Mei
1003137937

Jinhan Mei
1003137937

Jinhan Mei
1003137937

Jinhan Mei
1003137937