STA2104 Assignment 2
Stochastic Variational Inference in the TrueSkill Model
Chang Liu, 1001106386

1. Implementing the model [10 points]

1(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.

Code:

```
#Q1(a)
function log_prior(zs)
  logprob=factorized_gaussian_log_density(0, 0, zs)
  return logprob
end;
```

1(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.

Code:

```
#Q1(b)
function logp_a_beats_b(za,zb)
  return -log1pexp.(zb .- za)
end;
```

1(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].

Code:

```
#Q1(c)
function all_games_log_likelihood(zs,games)
  zs_a = zs[games[:,1],:]
  zs_b = zs[games[:,2],:]
  loglike_a_beats_b = logp_a_beats_b(zs_a,zs_b)
  likelihoods =  sum.(eachcol(loglike_a_beats_b))
  return  vec(likelihoods)'
end;
```

1(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)

Code:

```
#Q1(d)
function joint_log_density(zs,games)
  prior = log_prior(zs)
  likelihoods = all_games_log_likelihood(zs,games)
  return prior + likelihoods
end;
```

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.

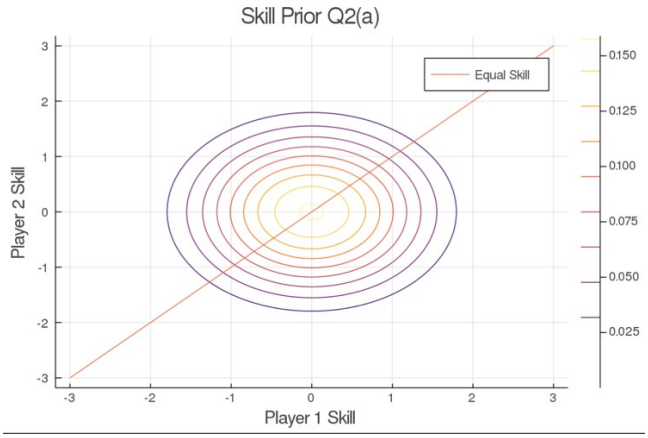
We also provided a function ***two_player_toy_games*** which produces toy data for two players. I.e. ***two_player_toy_games(5,3)*** produces a dataset where player A wins 5 games and player B wins 3 games.

2(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.

Code:

```
#Q2(a)
plot_log_prior(zs) = exp.(log_prior(zs));
plot(title="Skill Prior Q2(a)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
skillcontour!(plot_log_prior)
plot_line_equal_skill!()
```

Plot:

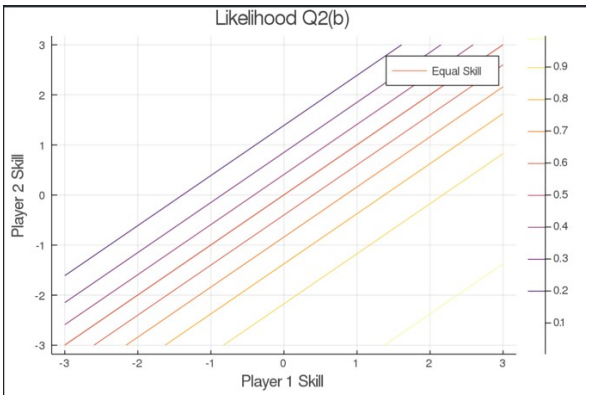


2(b). [2 points] Plot the isocontours of the likelihood function. Also plot the line of equal skill, zA = zB.

Code:

```
#Q2(b)
plot_likelihood(zs) = exp.(logp_a_beats_b(zs[1],zs[2])); | ✓
plot(title="Likelihood Q2(b)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   ) | Plot{Plots.GRBackend() n=0}
skillcontour!(plot_likelihood) | Plot{Plots.GRBackend() n=1}
plot_line_equal_skill!() | Plot{Plots.GRBackend() n=2}
```

Plot:

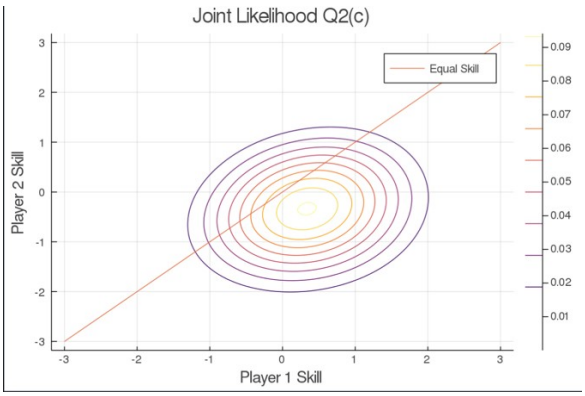


2(c). [2 points] Plot isocountours of the joint posterior over zA and zB given that player A beat player B in one match. Since the contours don't depend on the normalization constant, you can simply plot the isocontours of the log of joint distribution of p(zA,zB,A beat B) Also plot the line of equal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

Code:

```
#Q2(c)
# player A wins 1 match
games=two_player_toy_games(1,0);
plot_joint_likelihood(zs) = exp.(joint_log_density(zs,games));
plot(title="Joint Likelihood Q2(c)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   ) Plot{Plots.GRBackend() n=0}
skillcontour!(plot_joint_likelihood) Plot{Plots.GRBackend() n=2}
plot_line_equal_skill!() Plot{Plots.GRBackend() n=2}
```

Plot:

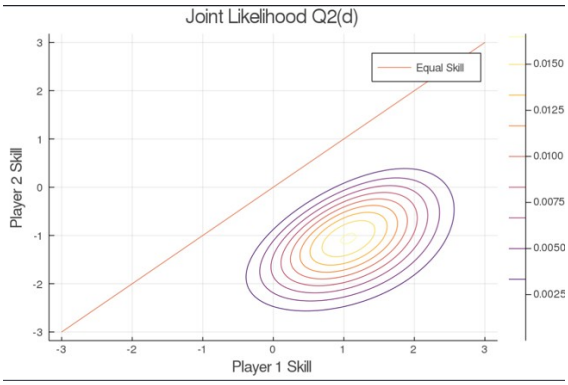


2(d). [2 points] Plot isocountours of the joint posterior over zA and zB given that 10 matches were played, and player A beat player B all 10 times. Also plot the line of equal skill, zA = zB.

Code:

```
# Q2(d)
# plot joint contours with player A winning 10 games
games=two_player_toy_games(10,0);
plot_joint_likelihood(zs) = exp.(joint_log_density(zs,games));
plot(title="Joint Likelihood Q2(d)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   ) Plot{Plots.GRBackend() n=0}
skillcontour!(plot_joint_likelihood) Plot{Plots.GRBackend() n=1}
plot_line_equal_skill!() Plot{Plots.GRBackend() n=2}
```

Plot:

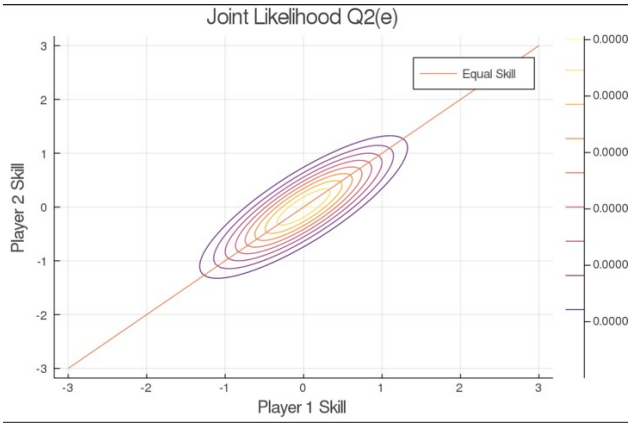


2(e). [2 points] Plot isocountours of the joint posterior over zA and zB given that 20 matches were played, and each player beat the other 10 times. Also plot the line of equal skill, zA = zB.

Code:

```
# Q2(e)
# plot joint contours with player A winning 10 games and player B winning 10 games
games=two_player_toy_games(10,10);
plot_joint_likelihood(zs) = exp.(joint_log_density(zs,games));
plot(title="Joint Likelihood Q2(e)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   ) Plot{Plots.GRBackend() n=0}
skillcontour!(plot_joint_likelihood) Plot{Plots.GRBackend() n=1}
plot_line_equal_skill!()
```

Plot:



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'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 inference to approximate the true posterior.

3(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, qϕ(z|data), plus an unknown constant. Use a fully-factorized Gaussian distribution for 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.

Code:

```
# Q3(a)
function elbo(params,logp,num_samples)
  mu = params[1]
  logσ = params[2]
  N = size(mu)[1]
  a = Diagonal(exp.(logσ)) * randn(N, num_samples)
  b = reshape(mu, N, 1) * ones(1, num_samples)
  samples = a .+ mu
  logp_estimate = logp(samples)
  logq_estimate = factorized_gaussian_log_density(mu, logσ, samples)
  return mean(logp_estimate .- logq_estimate)
end;
```

3(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.

Code:

```
# Q3(b)
# Conveinence function for taking gradients
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;
```

3(c). [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 descent where for each iteration:

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

Return the parameters resulting from training.

Code:

```
# Q3(c)
# 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);
```

```
# Q3(c)
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
    loss(params) = neg_toy_elbo(params; games=toy_evidence, num_samples=num_q_samples)
    # gradients of variational objective with respect to parameters
    grad_params = gradient(loss, params_cur)[1]
    mu_grad = grad_params[1]
    logσ_grad = grad_params[2]
    mu = params_cur[1] - lr .* mu_grad
    logσ = params_cur[2] - lr .* logσ_grad
    params_cur = (mu, logσ)
    # println(loss(params_cur))
  end
```

```
  p(zs) = exp.(joint_log_density(zs, games));
  q(zs) = exp.(factorized_gaussian_log_density(params_cur[1], params_cur[2], zs));
  plot(title="Compare p & q",
      xlabel = "Player 1 Skill",
      ylabel = "Player 2 Skill"
     )
  display(plot_line_equal_skill!())
  display(skillcontour!(p; colour=:red))
  display(skillcontour!(q; colour=:blue))
  return params_cur
end; ✓
```

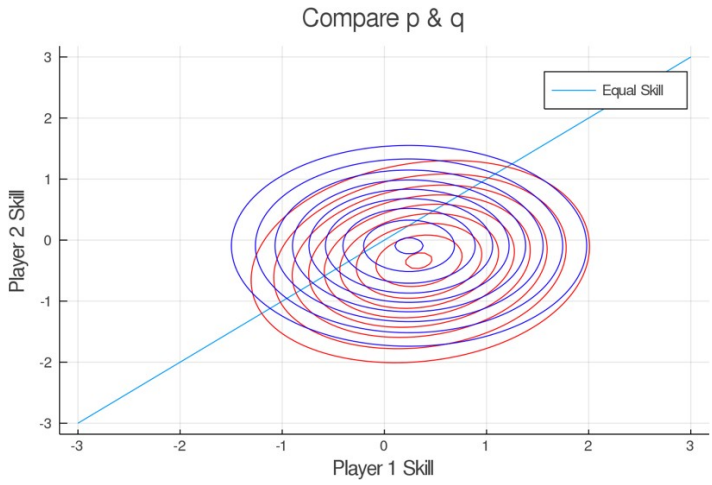
3(d). [2 points] Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 1 game. Report the final loss. Also plot the optimized variational approximation contours (in blue) and the target distribution (in red) on the same axes.

Code:

```
# Q3(d)
# fit q with SVI observing player A winning 1 game
games = two_player_toy_games(1,0);
params_1 = fit_toy_variational_dist(toy_params_init, games);
# savefig("Q3_d.pgn")
```

Output:

Final loss: 0.667



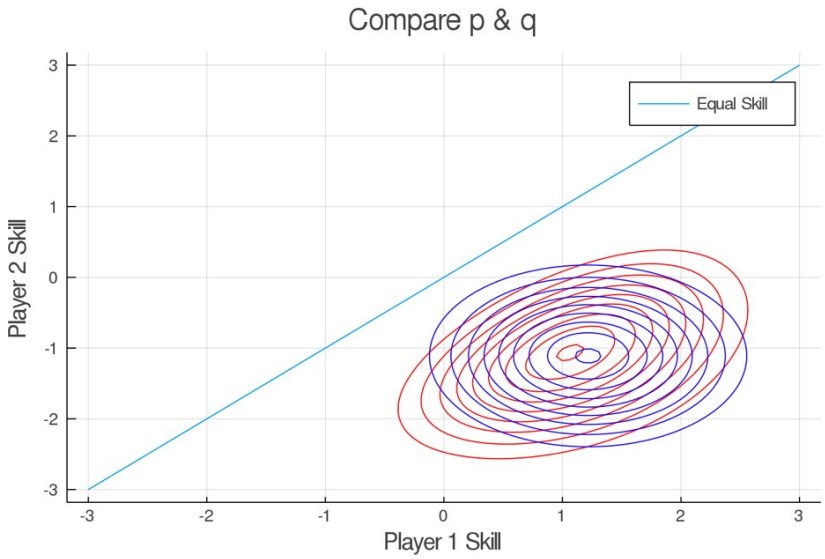
3(e). [2 points] Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 10 games. Report the final loss. Also plot the optimized variational approximation contours (in blue) and the target distribution (in red) on the same axes.

Code:

```
# Q3(e)
# fit q with SVI observing player A winning 10 games
games = two_player_toy_games(10,0);
params_1 = fit_toy_variational_dist(toy_params_init, games);
# savefig("Q3_e.pgn")
```

Output:

Final loss: 2.9051



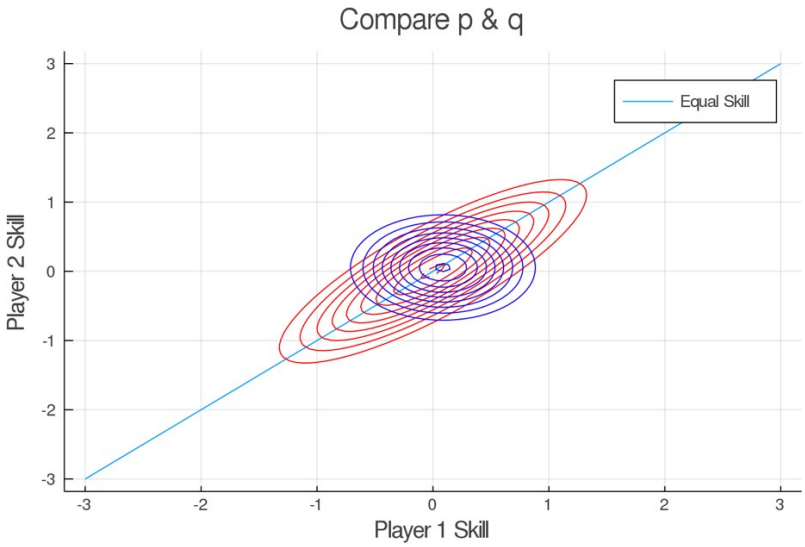
3(f). [2 points] Initialize a variational distribution parameters and optimize them to approximate the joint where we observe player A winning 10 games and player B winning 10 games. Report the final loss. Also plot the optimized variational approximation contours (in blue) and the target distribution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

Code:

```
# Q3(f)
# fit q with SVI observing player A winning 10 games and player B winning 10 games
games = two_player_toy_games(10,10);
params_1 = fit_toy_variational_dist(toy_params_init, games);
# savefig("Q3_f.pgn")
```

Output:

Final loss: 15.73



4. Approximate inference conditional on real data [24 points]

Load the dataset from ***tennis_data.mat*** containing two matrices:

• W is a 107 by 1 matrix, whose ith 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:

4(a). [1point] For any two players I and j,p(zi,zj|all games)is always proportional to p(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.

Answer: yes

4(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 final negative ELBO estimate after optimization.

Code:

```
# Q4(b)
function fit_variational_dist(init_params, tennis_games; num_itrs=500, lr= 1e-2, num_q_samples = 10)
  params_cur = init_params
  for i in 1:num_itrs
    loss(params) = neg_toy_elbo(params; games=tennis_games, num_samples=num_q_samples)
    grad_params = gradient(loss, params_cur)[1]
    mu_grad = grad_params[1]
    logσ_grad = grad_params[2]
    mu = params_cur[1] - lr .* mu_grad
    logσ = params_cur[2] - lr .* logσ_grad
    params_cur = (mu, logσ)
    println(loss(params_cur))
  end
  return params_cur
end;
```

```
# Initialize variational family
init_mu = rand(-2:3, size(player_names)[1]);
init_log_sigma = rand(107);
init_params = (init_mu, init_log_sigma);

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

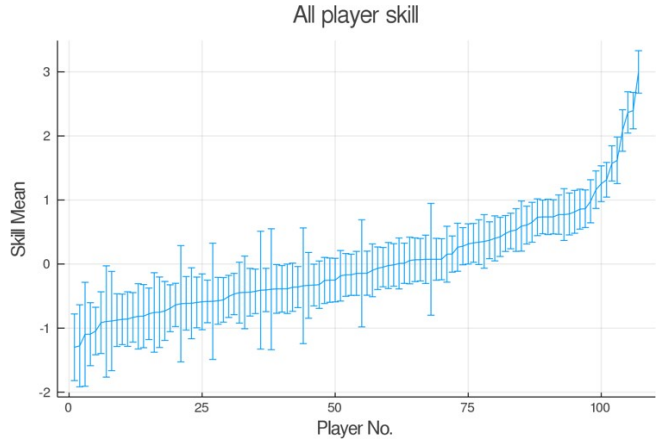
Final loss/negative elbo = 1143.29

4(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.

Code:

```
# Q4(c)
skill_μ = trained_params[1];
skill_σ = trained_params[2];
perm = sortperm(skill_μ);
plot(title="All player skill",
    xlabel = "Player No.",
    ylabel = "Skill Mean",
    legend = false,
   )
plot!(skill_μ[perm], yerror=exp.(skill_σ[perm]))
```

Plot:



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

Answer:

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

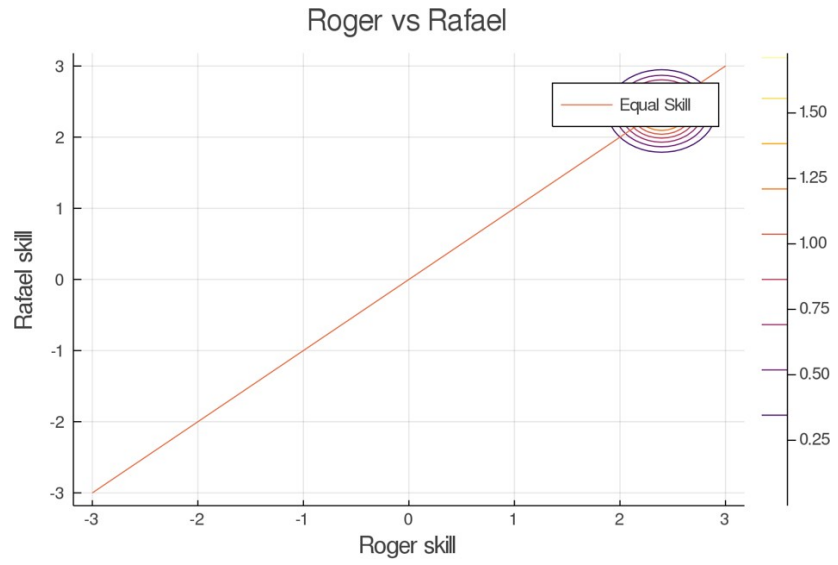
4(e). [3 points] Plot the joint posterior over the skills of Roger Federer and Rafael Nadal.

Code:

```
# Q4(e)
sorted_μ = reverse(skill_μ[perm]);
sorted_σ = reverse(skill_σ[perm]);
μ1 = sorted_μ[2];
μ2 = sorted_μ[3];
logσ1 = sorted_σ[2];
logσ2 = sorted_σ[3];
plot_μ = [μ1, μ2];
plot_logσ = [logσ1, logσ2];

# find index of Roger-Federer and Rafael-Nadal
f1(x) = x=="Roger-Federer";
car1 = findall(f1, player_names);
f2(x) = x=="Rafael-Nadal";
car2 = findall(f2, player_names);
L = LinearIndices(player_names);
ind1 = L[car1][1];
ind2 = L[car2][1];
```

```
f3(x) = (x[1] == ind1) & (x[2] == ind2);
f3_ind = f3.(eachrow(tennis_games));
f3_result = tennis_games[f3_ind, :];

f4(x) = (x[1] == ind2) & (x[2] == ind1);
f4_ind = f4.(eachrow(tennis_games));
f4_result = tennis_games[f4_ind, :];

games = vcat(f3_result, f4_result);
plot_joint(zs) = exp.(factorized_gaussian_log_density(plot_μ, plot_logσ, zs));
plot(title="Roger vs Rafael",
    xlabel = "Roger skill",
    ylabel = "Rafael skill",
)
skillcontour!(plot_joint)
plot_line_equal_skill!()
```

Plot:



4(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 ∼ N (μ, Σ), then AX ∼ N (Aμ, AT ΣA) where A is a linear transformation.
- Hint 3: Marginalization in Gaussians is easy: if X ∼ N (μ, Σ), then the ith element of X has a marginal distribution Xi ∼ N(μi,Σii)

$$\vec{\mu}_z = [\bar{z}_a, \bar{z}_b], \quad \Sigma_z = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$

$$\vec{\mu}_y = \begin{bmatrix} \bar{z}_a - \bar{z}_b \\ \bar{z}_b \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{z}_a \\ \bar{z}_b \end{bmatrix} \Rightarrow A = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

$$\Sigma_y = A^T \Sigma_z A = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 \\ -\sigma_1 & \sigma_2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1 & -\sigma_1 \\ -\sigma_1 & \sigma_1 + \sigma_2 \end{bmatrix}$$

$$z_{a-b} \sim N(\bar{z}_a - \bar{z}_b, \sigma_1)$$

$$P(z_a - z_b > 0) = P(z_{a-b} > 0) = \int_0^\infty \frac{1}{\sigma_1 \sqrt{2\pi}} \exp\left(\frac{-(x - (\bar{z}_a - \bar{z}_b)^2)}{2\sigma_1^2}\right) dx$$

4(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.

Code:

```
# Q4(g)
mu_g = μ1 - μ2;
sigma_g = exp(logσ1);
dist_g = Normal(mu_g, sigma_g);
prob_exact_g = 1- cdf(dist_g, 0)  0.539…

sample_g = randn(10000) .* sigma_g .+ mu_g;
f5(x) = x .> 0;
prob_mc_g = mean(f5(sample_g))  0.541…
```

Answer:

- Approximate posterior: 0.539
- Monte Carlo: 0.541

4(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.

Code:

```
# Q4(h)
skill_μ = trained_params[1];
skill_σ = trained_params[2];
perm = sortperm(skill_μ);
sorted_μ = reverse(skill_μ[perm]);
sorted_σ = reverse(skill_σ[perm]);

μ107 = sorted_μ[107];
mu_h = μ1 - μ107;
sigma_h = exp(logσ1);
dist_h = Normal(mu_h, sigma_h);
prob_exact_h = 1- cdf(dist_h, 0)  1.00

sample_h = randn(10000) .* sigma_h .+ mu_h;
prob_mc_h = mean(f5(sample_h))  1.00
```

Answer:

- Exact: 1
- Monte Carlo: 1

4(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.

Answer: (b), (c)