STA414 Assignment 2

Jeff Blair: 1002177057
blairjef

Collaborator: Jai Aggarwal

March 2020

1 Implementing the model

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

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

c.
```
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
```

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

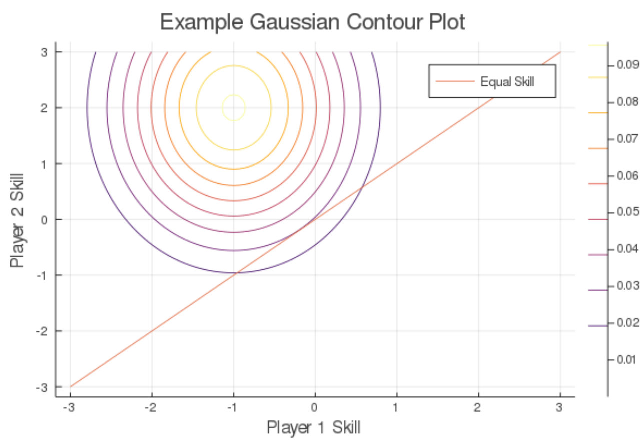
Testing:

```
@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, false)
```

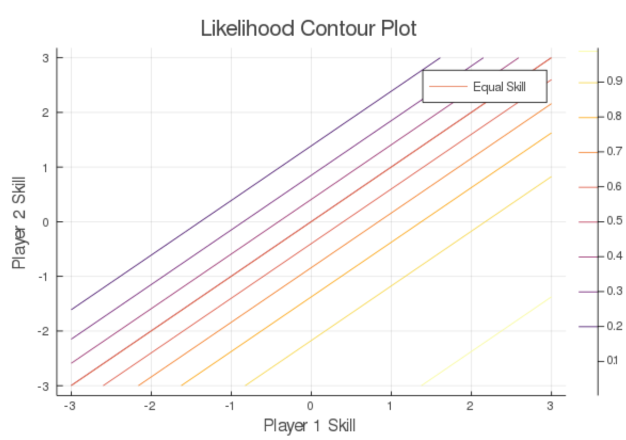
2 Examining the posterior for only two players and toy data

a.
```
# 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"
   )

# TODO: plot prior contours
example_gaussian(zs) = exp(factorized_gaussian_log_density([-1.,2.],[0.,0.5],zs))
skillcontour!(example_gaussian)
plot_line_equal_skill!()
savefig(joinpath("plots","prior_contours"))
```



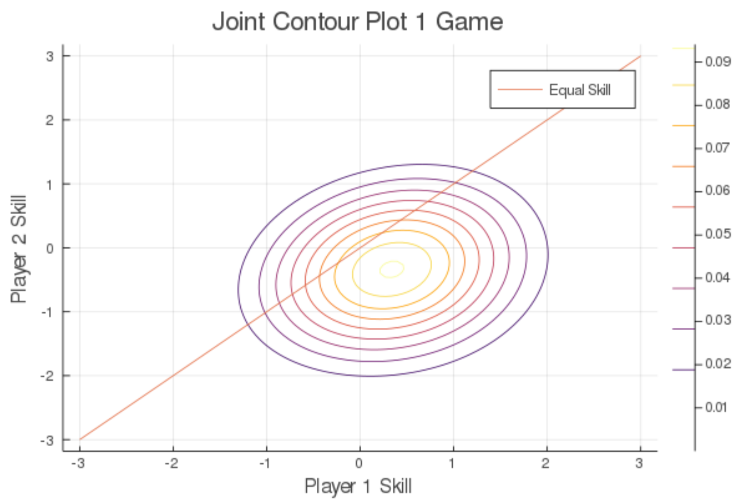
(a) Prior Contours

b.
```
# TODO: plot likelihood contours
plot(title="Likelihood Contour Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill")
likelihood(zs) = exp.(logp_a_beats_b.(zs[1,:], zs[2,:]))
skillcontour!(likelihood)
plot_line_equal_skill!()
savefig(joinpath("plots", "likelihood_contours"))
```



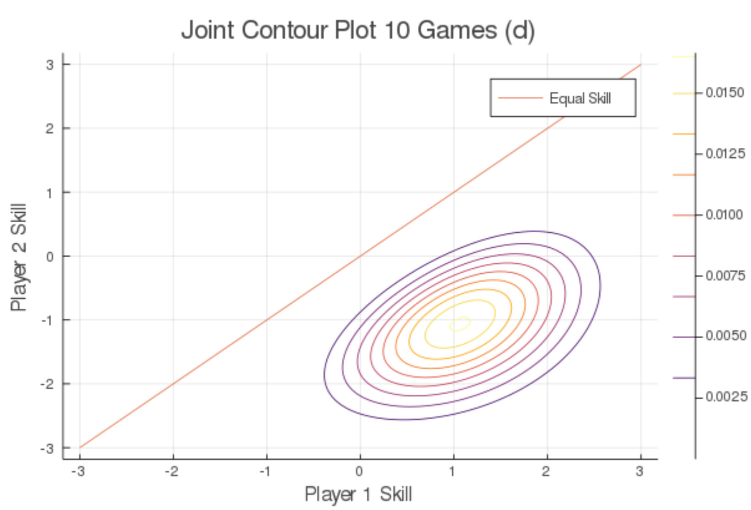
(a) Likelihood Contours

c.
```
# TODO: plot joint contours with player A winning 1 game
plot(title="Joint Contour Plot 1 Game",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill")
one_game = two_player_toy_games(1, 0)
joint_posterior_1(zs) = exp(joint_log_density(zs, one_game))
skillcontour!(joint_posterior_1)
plot_line_equal_skill!()
savefig(joinpath("plots", "posterior_contours_1"))
```



(a) Joint Contours 1-0

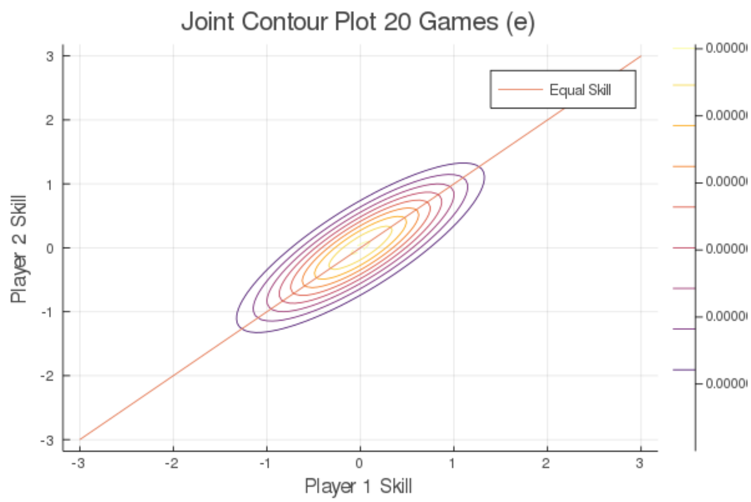
d.
```
# TODO: plot joint contours with player A winning 10 games
plot(title="Joint Contour Plot 10 Games (d)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill")
ten_games = two_player_toy_games(10, 0)
joint_posterior_10d(zs) = exp(joint_log_density(zs, ten_games))
skillcontour!(joint_posterior_10d)
plot_line_equal_skill!()
savefig(joinpath("plots", "posterior_contours_10d"))
```



(a) Joint Contours 10-0

e. *#TODO: plot joint contours with player A winning 10 games and player B winning 10 games*

```
plot(title="Joint Contour Plot 20 Games (e)",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill")
twenty_games = two_player_toy_games(10, 10)
joint_posterior_20e(zs) = exp(joint_log_density(zs, twenty_games))
skillcontour!(joint_posterior_20e)
plot_line_equal_skill!()
savefig(joinpath("plots", "posterior_contours_20e"))
```



(a) Joint Contours 10-10

3 Stochastic Variational Inference on Two Players and Toy Data

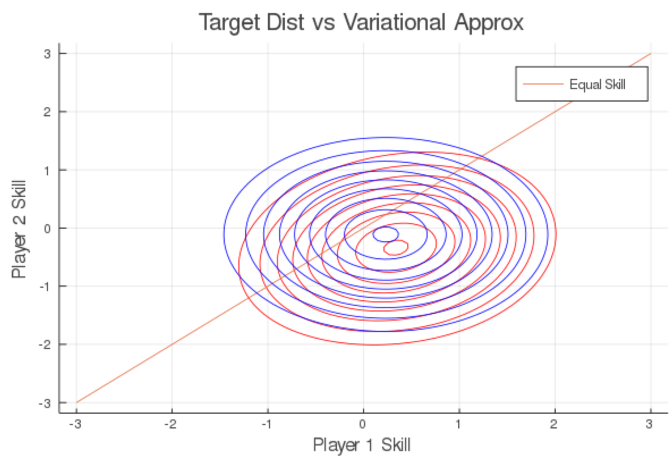
a.
```
function elbo(params, logp, num_samples)
  mu, logsig = params
  samples = mu .+ exp.(logsig) .* randn(size(mu)[1], num_samples)
  logp_estimate = logp(samples)
  logq_estimate = factorized_gaussian_log_density(mu, logsig, samples)
  return mean(logp_estimate .- logq_estimate)#TODO: should return scalar (hint: average over batch)
end
```

b.
```
# 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
```

c.
```
function fit_toy_variational_dist(init_params, toy_evidence; num_itrs=200,
                                  lr= 1e-2, num_q_samples = 10, fp="TvsV")

  params_cur = init_params
  for i in 1:num_itrs
    grad_params = gradient(params_cur -> neg_toy_elbo(params_cur; games=toy_evidence,
                    num_samples=num_q_samples), params_cur)
    mu, logsig = params_cur
    mu -= lr .* grad_params[1][1]
    logsig -= lr .* grad_params[1][2]
    params_cur = mu, logsig  #TODO: update paramters with lr-sized step in descending gradient
    e = -neg_toy_elbo(params_cur; games=toy_evidence, num_samples=num_q_samples)
    @info "ELBO:" e#TODO: report the current elbo 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(title="Target Dist vs Variational Approx",
            xlabel="Player 1 Skill",
            ylabel="Player 2 Skill");

    if i == num_itrs - 1
        true_dist(zs) = exp(joint_log_density(zs, toy_evidence))
        variational_dist(zs) = exp(factorized_gaussian_log_density(mu, logsig, zs))
        # plot likelihood contours for target posterior
        skillcontour!(true_dist,colour=:red)
        plot_line_equal_skill!()
        # plot likelihood contours for variational posterior
        display(skillcontour!(variational_dist, colour=:blue))
        #TODO: save final posterior plots
        savefig(joinpath("plots", fp))
    end
  end
  return params_cur
end
```

d.
```
# 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)

#TODO: fit q with SVI observing player A winning 1 game
one_game = two_player_toy_games(1, 0)
fp = "Toy_vs_Var_1"
fitted = fit_toy_variational_dist(toy_params_init, one_game; fp=fp)
```

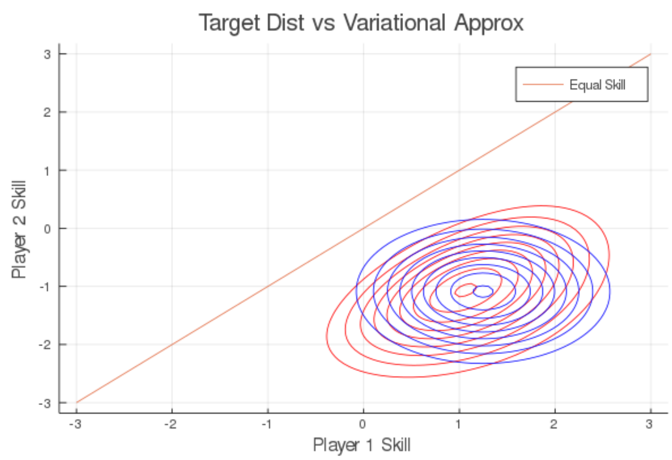
```
Info: ELBO:
  e = -0.8648205601918395
@ Main In[15]:10
```



(a) Variational Dist Contours 1-0

e.
```
#TODO: fit q with SVI observing player A winning 10 games
ten_games = two_player_toy_games(10, 0)
fp = "Toy_vs_Var_10"
fitted = fit_toy_variational_dist(toy_params_init, ten_games, fp=fp)
```

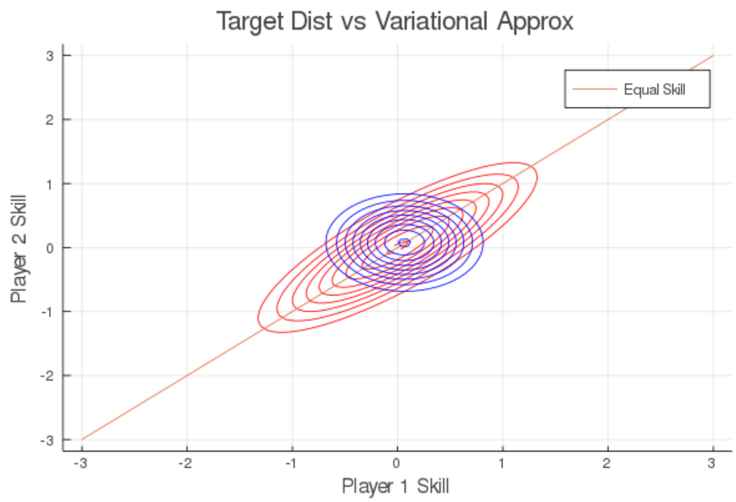
```
Info: ELBO:
  e = -2.857603824101711
@ Main In[15]:10
```



(a) Variational Dist Contours 10-0

f.
```
#TODO: fit q with SVI observing player A winning 10 games and player B winning 10 games
twenty_games = two_player_toy_games(10, 10)
fp = "Toy_vs_Var_20"
fitted = fit_toy_variational_dist(toy_params_init, twenty_games, fp=fp)
```

```
 Info: ELBO:
   e = -15.165728558969192
 @ Main In[15]:10
```



(a) Variational Dist Contours 10-10

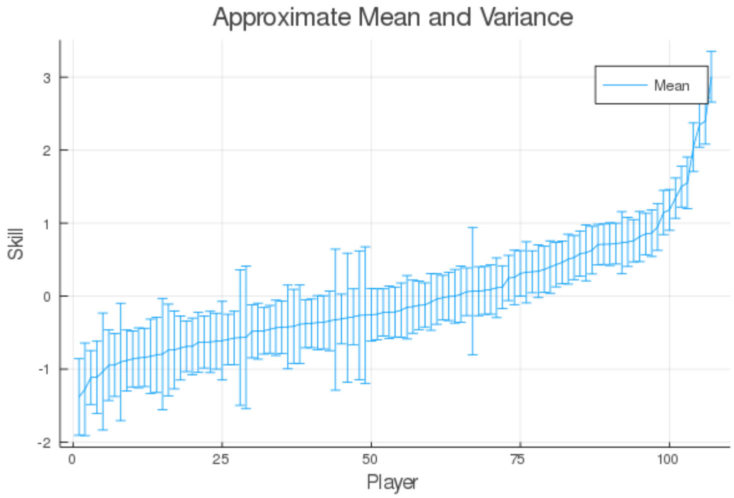
4 Approximate inference conditioned on real data

a. In general, the isocontours of $p(z_i, z_j \mid \text{all games})$ will be different than those of $p(z_i, z_j \mid \text{games between i and j})$

b.
```
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_cur -> neg_toy_elbo(params_cur; games=tennis_games, num_samples=num_q_samples),
    mu, logsig = params_cur
    mu -= lr .* grad_params[1][1]
    logsig -= lr .* grad_params[1][2]
    params_cur = mu, logsig  #TODO: update paramters with lr-sized step in descending gradient
    e = neg_toy_elbo(params_cur; games=tennis_games, num_samples=num_q_samples)
    @info "ELBO:" e#TODO: report the current elbbo during training
  end
  return params_cur
end

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

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

```
 Info: ELBO:
   e = 1142.8211707136102
 @ Main In[46]:10
```

c.
```
means, logstd = trained_params
perm = sortperm(means)
plot(means[perm], yerror=exp.(logstd[perm]))
```

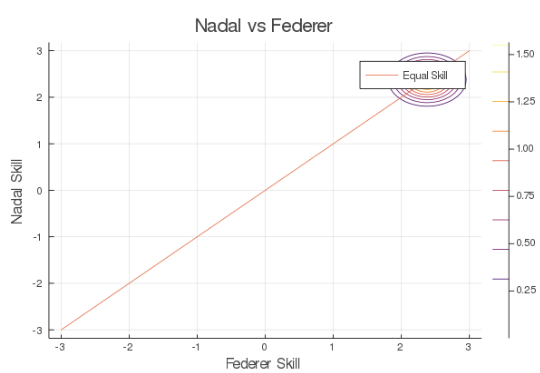


(a) Approx Mean and Variance

d. `reverse(player_names[perm[num_players-9:end]]) # Top ten`

```
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"
```

e.
```
#TODO: joint posterior over "Roger-Federer" and ""Rafael-Nadal""
RF = findall(x -> x == "Roger-Federer", player_names)
RN = findall(x -> x == "Rafael-Nadal", player_names)
mu = means[RF, RN]
logsig = logstd[RF, RN]
variational_dist(zs) = exp(factorized_gaussian_log_density(mu, logsig, zs))
plot(title="Nadal vs Federer",
    xlabel = "Federer Skill",
    ylabel = "Nadal Skill")
skillcontour!(variational_dist)
plot_line_equal_skill!()
savefig(joinpath("plots", "Fed_v_Nad"))
```



(a) Federer vs Nadal

f.
$$\begin{bmatrix} y_a \\ y_b \end{bmatrix} = \begin{bmatrix} z_a - z_b \\ z_b \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} z_a \\ z_b \end{bmatrix}$$

Then $Y = AZ \sim \mathcal{N}\left(A\mu_z, A\Sigma_z A^T\right)$, where $A = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$

$Y \sim \mathcal{N}\left(\begin{bmatrix} \mu_a - \mu_b \\ \mu_b \end{bmatrix} \begin{bmatrix} \sigma_a^2 + \sigma_b^2 & -\sigma_b^2 \\ -\sigma_b^2 & \sigma_b^2 \end{bmatrix}\right) \implies y_a \sim \mathcal{N}(\mu_a - \mu_b, \sigma_a^2 + \sigma_b^2)$

$P(z_a > z_b) = P(z_a - z_b > 0) = P(y_a > 0) = 1 - P(y_a \leq 0) = 1 - F_{y_a}(0) = 1 - \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{\mu_b - \mu_a}{\sqrt{2(\sigma_a^2 + \sigma_b^2)}}\right)\right]$

g. **Exact Probability:**

```
using Distributions
# P(Federer has higher skill than Nadal)
# Exact
mu = means[RF][1] - means[RN][1]
var = exp(logstd[RF][1])^2 + exp(logstd[RN][1])^2
D = Normal(mu, sqrt(var))
p = 1 - cdf(D, 0)
```

```
0.5489200860592095
```

Monte Carlo:

```
# Monte Carlo
count = 0
for i in 1:10000
    z = mu + randn()*sqrt(var)
    if z > 0
        count += 1
    end
end
p = count/10000
```

```
0.5502
```

h. **Exact Probability:**

```
# Federer vs Worst Player
player = perm[1]
mu = means[RF][1] - means[player][1]
var = exp(logstd[RF][1])^2 + exp(logstd[player][1])^2
D = Normal(mu, sqrt(var))
p = 1 - cdf(D, 0)
```

```
0.9999999996664231
```

Monte Carlo:

```
# Monte Carlo
count = 0
for i in 1:10000
    z = mu + randn()*sqrt(var)
    if z > 0
        count += 1
    end
end
p = count/10000
```

```
1.0
```

i. b,c and e.