```
import Pkg;Pkg.add("Revise")
```

```
Resolving package versions...
  Updating `~/.julia/environments/v1.3/Project.toml`
 [no changes]
  Updating `~/.julia/environments/v1.3/Manifest.toml`
 [no changes]
```

```
import Base.hex
using Revise # lets you change A2funcs without restarting julia!
includet("A2_src.jl")
using Plots
using Statistics: mean
using Zygote
using Test
using Logging
using Distributions
import Pkg;Pkg.add("StatsFuns")
```

```
Resolving package versions...
  Updating `~/.julia/environments/v1.3/Project.toml`
 [no changes]
  Updating `~/.julia/environments/v1.3/Manifest.toml`
 [no changes]
```

```
#module A2funcs
#using Plots
using StatsFuns: log1pexp

@info "Guolun Li A2"
@info "For all plots, see the plot file."
function factorized_gaussian_log_density(mu,logsig,xs)
  #mu and logsig either same size as x in batch or same as whole batc
h
  #returns a 1 x batchsize array of likelihoods
  #each col is a group of observation
  σ = exp.(logsig)
  return sum((-1/2)*log.(2π*σ.^2) .+ -1/2 * ((xs .- mu).^2)./(σ.^2),d
ims=1)
end

function skillcontour!(f; colour=nothing)
  n = 100
  x = range(-3,stop=3,length=n)
  y = range(-3,stop=3,length=n)
  z_grid = Iterators.product(x,y) # meshgrid for contour
  z_grid = reshape.(collect.(z_grid),:,1) # add single batch dim
  z = f.(z_grid)
```

```
  z = getindex.(z,1)'
  max_z = maximum(z)
  levels = [.99, 0.9, 0.8, 0.7,0.6,0.5, 0.4, 0.3, 0.2] .* max_z
  if colour==nothing
  p1 = contour!(x, y, z, fill=false, levels=levels)
  else
  p1 = contour!(x, y, z, fill=false, c=colour,levels=levels,colorbar=
false)
  end
  plot!(p1)
end

function plot_line_equal_skill!()
  plot!(range(-3, 3, length=200), range(-3, 3, length=200), label="Eq
ual Skill")
end

#Q1(a)
function log_prior(zs)
  #inputs a N*k matrix and outputs a 1*K matrix
  return factorized_gaussian_log_density(0,0, zs)
end

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

#Q1(c)
function all_games_log_likelihood(zs,games)
  za = zs[games[:,1],:]
  zb = zs[games[:,2],:]
  return sum(logp_a_beats_b.(za, zb),dims = 1)
end

#Q1(d)
function joint_log_density(zs,games)
  return log_prior(zs) .+ all_games_log_likelihood(zs, games)
end

@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
```

```
# 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"
   )
#correct size
example_gaussian(zs) = exp(factorized_gaussian_log_density([-1.,2.],[
0.,0.5],zs))
#second dimension copies first dimension parameter
example_gaussian1(zs) = exp(factorized_gaussian_log_density([1],[0],z
s))
#will produce error
example_gaussian2(zs) = exp(factorized_gaussian_log_density([1,2,3],[
0,4,8],zs))
skillcontour!(example_gaussian)
plot_line_equal_skill!()
savefig(joinpath("/Users/liguolun/Desktop/COURSES/Fourth year/STA414/
STA414-2020-A2-polo2444172276/plots","example_gaussian1.pdf"))

#Q2(a)
# plot of joint prior contours
plot(title="Joint prior over skills Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
joint_prior(zs) = exp.(log_prior(zs))
skillcontour!(joint_prior)
plot_line_equal_skill!()
savefig(joinpath("plots","2a-joint_prior.pdf"))

#Q2(b)
# plot of likelihood contours
plot(title="Probability of p1 beating p2 over skills Plot",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
```

```
   )
likelihood_skill(zs) = exp(logp_a_beats_b(zs[1],zs[2]))
skillcontour!(likelihood_skill; colour =:blue)
plot_line_equal_skill!()
savefig(joinpath("plots","2b-likelihood_over_skills.pdf"))

#Q2(c)
#plot of joint contours with player A winning 1 game
plot(title="Joint posterior of skills given p1 beats p2 one time",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
onegame_posterior(zs) = exp.(joint_log_density(zs,two_player_toy_game
s(1,0)))
skillcontour!(onegame_posterior)
plot_line_equal_skill!()
savefig(joinpath("plots","2c-Joint posterior of skills given p1 beats
 p2 one time.pdf"))

#Q2(d)
#plot of joint contours with player A winning 10 games
plot(title="Joint posterior of skills given p1 beats p2 ten times",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
tengame_posterior(zs) = exp.(joint_log_density(zs,two_player_toy_game
s(10,0)))
skillcontour!(tengame_posterior)
plot_line_equal_skill!()
savefig(joinpath("plots","2d-Joint posterior of skills given p1 beats
 p2 ten times.pdf"))

#Q2(e)
#plot of joint contours with player A winning 10 games and player B w
inning 10 games
plot(title="Joint posterior of skills given p1 and p2 beat each other
 ten times",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
bothtengame_posterior(zs) = exp.(joint_log_density(zs,two_player_toy_
games(10,10)))
skillcontour!(bothtengame_posterior)
plot_line_equal_skill!()
savefig(joinpath("plots","2e-Joint posterior of skills given p1 and p
2 beat each other ten times.pdf"))

#Q3(a)
function elbo(params,logp,num_samples)
  #regard params as a ((N*1), (N*1)) matrix
```

```
  #N = length(params[:,1])
  #mu = repeat(params[:,1]', num_samples)' #a N*B matrix
  #log_sigma = repeat((sqrt.(params[:,2]))', num_samples)' #a N*B mat
rix
  #samples = exp.(log_sigma) .* randn(N,num_samples) .+ mu
  samples = exp.(params[2]) .* randn(length(params[1]), num_samples)
.+ params[1]
  logp_estimate = logp(samples)
  logq_estimate = factorized_gaussian_log_density(params[1],params[2]
, samples)
  return 1/num_samples * sum((logp_estimate .- logq_estimate))
end

#Q3(b)
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


# 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)
  new_loss(p) = neg_toy_elbo(p; games = toy_evidence, num_samples = n
um_q_samples)
  params_cur = init_params
  for i in 1:num_itrs
    grad_params = gradient(new_loss,params_cur)[1]
    params_cur = (params_cur[1] .- lr .* grad_params[1],params_cur[2]
 .- lr .* grad_params[2])
    curr_loss = new_loss(params_cur)
    @info "loss = $curr_loss"
    plot(title="Joint posterior of skills given data",
        xlabel = "Player 1 Skill",
        ylabel = "Player 2 Skill"
       )
    #skillcontour!(zs -> exp.(joint_log_density(zs, toy_evidence));co
lour =:red)
    #plot_line_equal_skill!()
    #display(skillcontour!(zs -> exp.(factorized_gaussian_log_density
(params_cur[1],params_cur[2],zs));colour =:blue))
  end
  return params_cur
```

```
end

#Q3(d)
toy_evidence = two_player_toy_games(1,0)
trained_par = fit_toy_variational_dist(toy_params_init,toy_evidence)
plot(title="Joint posterior of skills given p1 beats p2 one time",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
skillcontour!(zs -> exp.(joint_log_density(zs, toy_evidence)); colour
 =:red)
plot_line_equal_skill!()
skillcontour!(zs -> exp.(factorized_gaussian_log_density(trained_par[
1],trained_par[2],zs));colour =:blue)
@info final_loss = 0.93
savefig(joinpath("plots","3d-trained model - joint posterior given p1
 beats p2 one time.pdf"))

#Q3(e)
toy_evidence = two_player_toy_games(10,0)
#trained_par = fit_toy_variational_dist(toy_params_init,toy_evidence)
plot(title="Joint posterior of skills given p1 beats p2 ten times",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
skillcontour!(zs -> exp.(joint_log_density(zs, toy_evidence)); colour
 =:red)
plot_line_equal_skill!()
skillcontour!(zs -> exp.(factorized_gaussian_log_density(trained_par[
1],trained_par[2],zs));colour =:blue)
@info final_loss = 2.865
savefig(joinpath("plots","3e-trained model - joint posterior given p1
 beats p2 ten times.pdf"))

#Q3(f)
toy_evidence = two_player_toy_games(10,10)
#trained_par = fit_toy_variational_dist(toy_params_init,toy_evidence)
plot(title="Joint posterior of skills given p1p2 beat each other ten
times",
    xlabel = "Player 1 Skill",
    ylabel = "Player 2 Skill"
   )
'''
skillcontour!(zs -> exp.(joint_log_density(zs, toy_evidence)); colour
 =:red)
plot_line_equal_skill!()
skillcontour!(zs -> exp.(factorized_gaussian_log_density(trained_par[
1],trained_par[2],zs));colour =:blue)
@info final_loss = 15.93
savefig(joinpath("plots","3f-trained model - joint posterior given p1
```

```
p2 beat each other ten times.pdf"))

## Question 4
# Load the Data
import Pkg;Pkg.add("MAT")
```

```
Resolving package versions...
  Updating `~/.julia/environments/v1.3/Project.toml`
 [no changes]
  Updating `~/.julia/environments/v1.3/Manifest.toml`
 [no changes]
```

```
using MAT
vars = matread("tennis_data.mat")
player_names = vars["W"]
tennis_games = Int.(vars["G"])
num_players = length(player_names)
num_games = length(tennis_games[:,1])
print("Loaded data for $num_players players")
```

```
Loaded data for 107 players
```

```
#4(a)
@info "No. Games not involving both players provide no other info."

#4(b)
function fit_variational_dist(init_params, tennis_games; num_itrs=400
, lr= 1e-2, num_q_samples = 10)
  params_cur = init_params
  new_loss(p) = neg_toy_elbo(p; games = tennis_games, num_samples = n
um_q_samples)
  for i in 1:num_itrs
    grad_params = gradient(new_loss, params_cur)[1]
    params_cur = (params_cur[1] - lr * grad_params[1], params_cur[2]
- lr * grad_params[2])
    curr_loss = new_loss(params_cur)
    @info "loss = $curr_loss"
  end
  return params_cur
end

init_mu = randn(num_players)#random initialziation
init_log_sigma = randn(num_players)# random initialziation
init_params = (init_mu, init_log_sigma)

# Train variational distribution
trained_params = fit_variational_dist(init_params, tennis_games)
@info "final neg_elbo is 1143.11"
```

```
#4(c)
#plot sorted skills of all players
means = trained_params[1]
logstd = trained_params[2]
perm = sortperm(means)
plot(means[perm], yerror = exp.(logstd[perm]),
    title = "Approximated sorted players' skill", ylabel = "player's
skill",
    label = "sorted skill")
savefig(joinpath("plots","4c-Approximated sorted players' skill"))

#4(d):
#10 players with highest mean skill under variational model
perm_des = [perm[num_players + 1 - i] for i in 1: num_players]
top_ten = player_names[perm_des[1:10]]
#listed in descending order
@info top_ten_names = ["Novak-Djokovic", "Roger-Federer", "Rafael-Nad
al", "Andy-Murray", "Robin-Soderling"
, "David-Ferrer", "Jo-Wilfried-Tsonga", "Tomas-Berdych", "Juan-Martin
-Del-Potro", "Richard-Gasquet"]

#4(e)
#finding player info
function find_player_info(name)
  #RF_num = [i for i in 1: num_players if player_names[i] == name]
  RF_num = findall(x -> x==name, player_names)[1][1][1]
  RF_rank = [i for i in 1: num_players if player_names[perm_des][i] =
= name][1]
  RF_games = [tennis_games[k,:] for k in 1:num_games if RF_num[1] in
tennis_games[k,:]]
  RF_win = sum([1 for k in 1:length(RF_games) if RF_games[k][1] == RF
_num[1]])
  return (RF_num,RF_rank, RF_win, length(RF_games))
end
RF_num = find_player_info("Roger-Federer")[1]
RN_num = find_player_info("Rafael-Nadal")[1]
# approximate joint posterior over "Roger-Federer" and ""Rafael-Nadal
""
plot(title="approximated joint posterior of skills between Roger and
Rafael",
    xlabel = "Roger's Skill",
    ylabel = "Rafael's Skill"
   )
mean_RFRN = [means[RF_num],means[RN_num]]
logstd_RFRN = [logstd[RF_num],logstd[RN_num]]
skillcontour!(zs -> exp.(factorized_gaussian_log_density(mean_RFRN, l
ogstd_RFRN, zs)))
plot!(range(-3, 3, length=200), range(-3, 3, length=200), label="Equa
l Skill",legend=:bottomleft)
```

```
savefig(joinpath("plots","4e-approximated joint posterior of skills b
etween Roger and Rafael"))

@info "See plots section for Q4(f)"

#4(g)
p = 1 - cdf(Normal(0,1), (mean_RFRN[2] - mean_RFRN[1])/
    sqrt(exp(logstd_RFRN[1])^2 + exp(logstd_RFRN[2])^2) )
@info "exact prob that RF has higher skill than RN is
      0.553"
N = 10000
RF_sample = randn(N) .* exp(logstd_RFRN[1]) .+ mean_RFRN[1]
RN_sample = randn(N) .* exp(logstd_RFRN[2]) .+ mean_RFRN[2]
est_RFbeatsRN = 1/N * (sum(RF_sample .> RN_sample))
@info "exact prob that RF has higher skill than RN is
      0.556"

#4(h)
lowest_mean = means[perm[1]]
lowest_logstd = logstd[perm[1]]
p = 1 - cdf(Normal(0,1), (lowest_mean - mean_RFRN[1])/
    sqrt(exp(logstd_RFRN[1])^2 + exp(lowest_logstd)^2) )
@info "exact prob that RF has higher skill than lowest skill player i
s
      1.00"
N = 10000
RF_sample = randn(N) .* exp(logstd_RFRN[1]) .+ mean_RFRN[1]
lowest_sample = randn(N) .* exp(lowest_logstd) .+ lowest_mean
est_RFbeatsRN = 1/N * (sum(RF_sample .> lowest_sample))
@info "exact prob that RF has higher skill than lowest skill player i
s
      1.00"
#4(i)
@info "We would change answer to (c),(e)"
```

Published from A2_starter.jl using Weave.jl on 2020-03-22.