Assignment 1

Guolun Li 1004118215

The goal of this assignment is to get you familiar with the basics of decision theory and gradient-based model fitting.

Decision theory [13pts]

One successful use of probabilistic models is for building spam filters, which take in an email and take different actions depending on the likelihood that it's spam.

Imagine you are running an email service. You have a well-calibrated spam classifier that tells you the probability that a particular email is spam: $p(\textnormal spam|\textnormal email)$. You have three options for what to do with each email: You can show it to the user, put it in the spam folder, or delete it entirely.

Depending on whether or not the email really is spam, the user will suffer a different amount of wasted time for the different actions we can take, $L(\textnormal action, \textnormal spam)$:

```
\begin{tabular}{c|cc}
Action & Spam & Not spam \\ \hline
Show   & 10 & 0 \\
Folder & 1  & 50 \\
Delete & 0  & 200
\end{tabular}
```

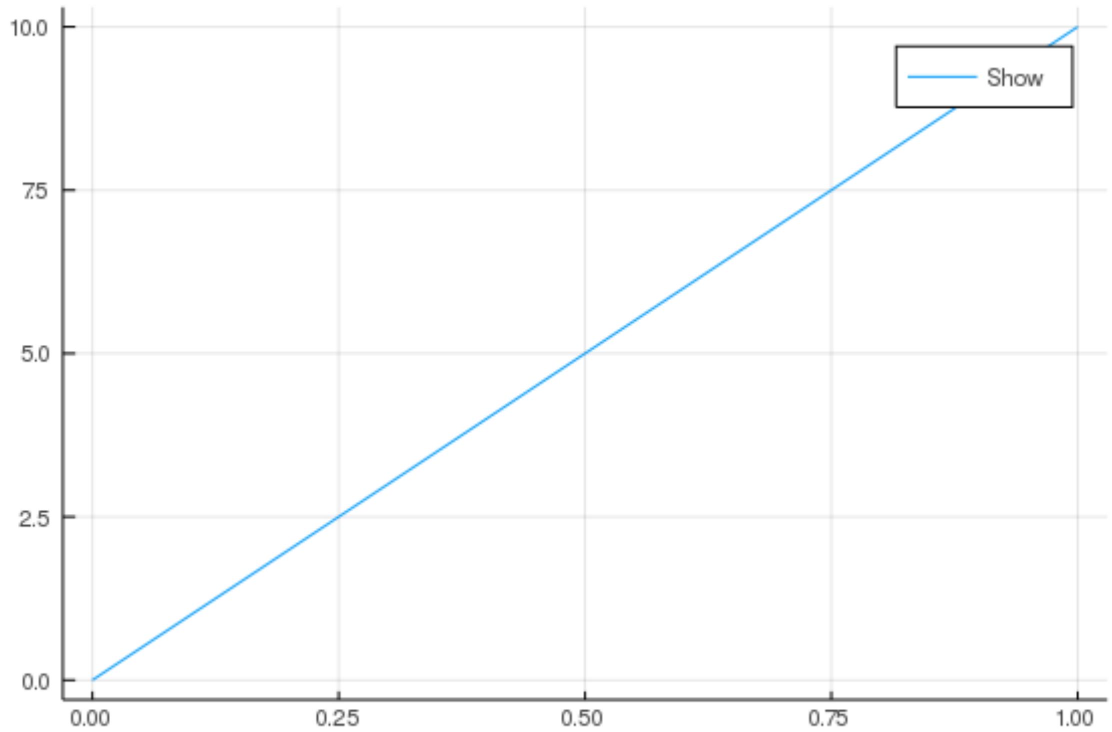
1. [3pts] Plot the expected wasted user time for each of the three possible actions, as a function of the probability of spam: $p(\textnormal spam|\textnormal email)$

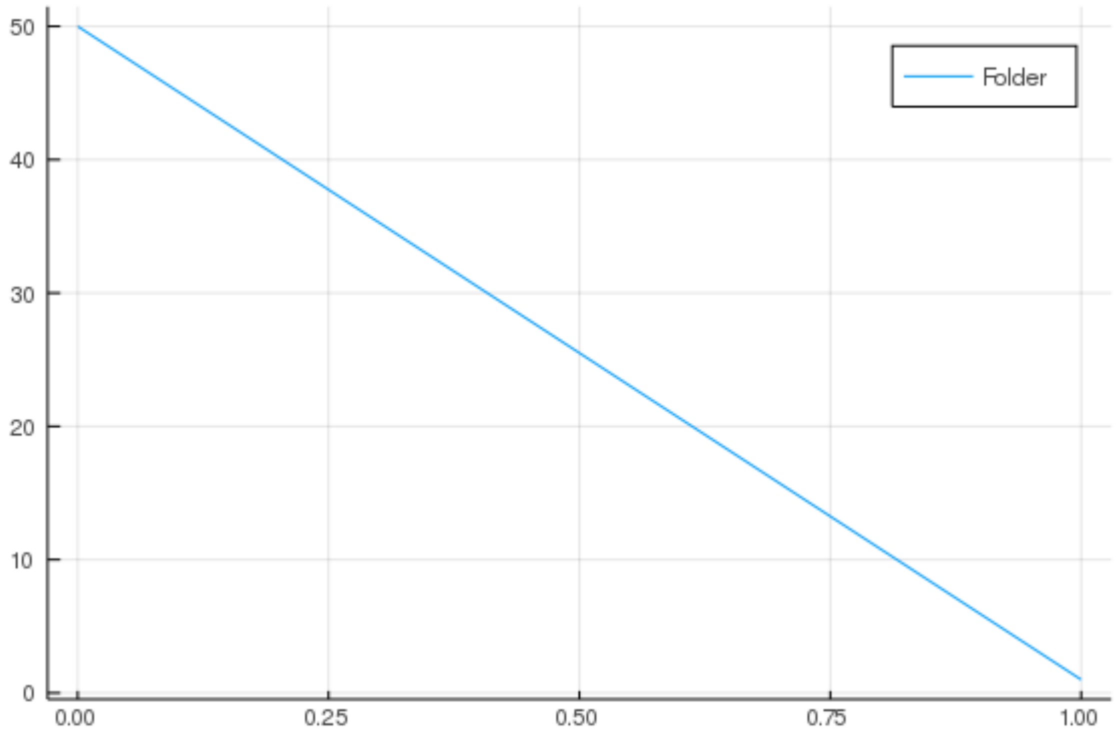
```
losses = [[10, 0],
          [1, 50],
          [0, 200]]

num_actions = length(losses)

function expected_loss_of_action(prob_spam, action)
    #TODO: Return expected loss over a Bernoulli random variable
    #      with mean prob_spam.
    #      Losses are given by the table above.
    return [losses[action][1]*prob_spam[i] + losses[action][2]*(1-prob_spa
m[i])
    for i in 1:length(prob_spam)]
    #return losses[action][1]*prob_spam + losses[action][2]*(1-prob_spam)
end
```

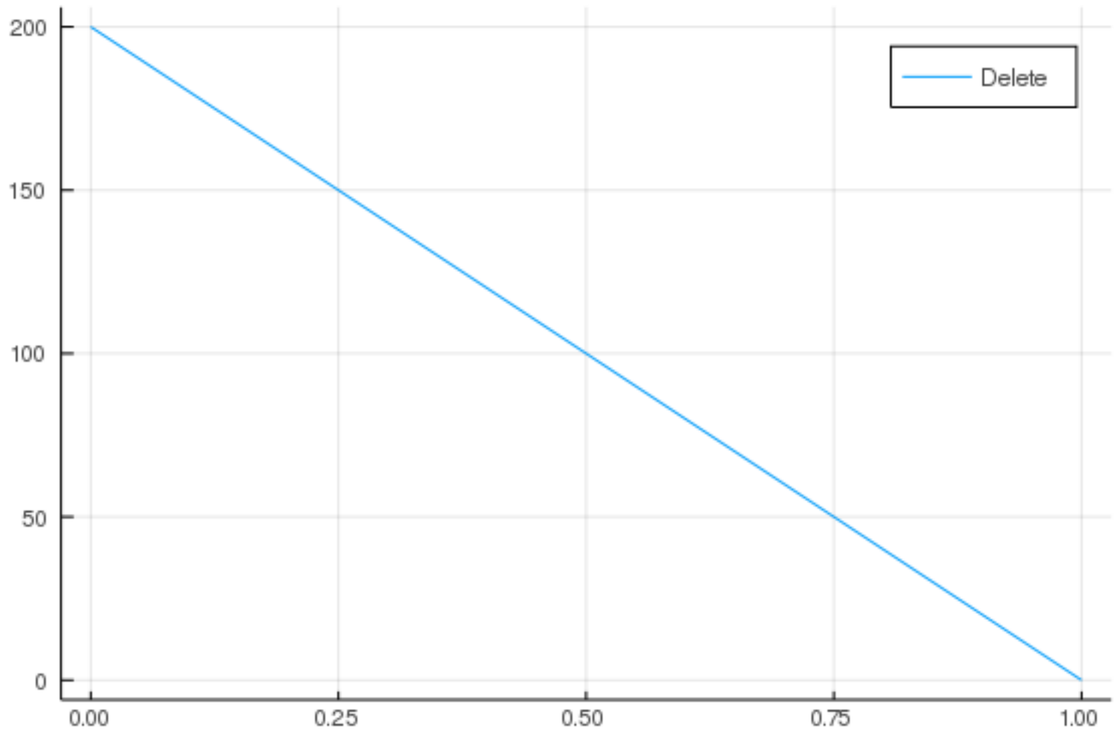
```
using Test
@testset "expected_loss_of_action correct" begin
  n = length(losses)
  @test expected_loss_of_action(0.4, 1) == [4]
  @test expected_loss_of_action([1,0], 2) == [1, 50]
end
```

```
Test Summary:                      | Pass  Total
expected_loss_of_action correct |    2      2
```

```
prob_range = range(0., stop=1., length=500)
action_names = ["Show","Folder","Delete"]
# Make plot
using Plots
for action in 1:num_actions
  display(plot(prob_range, expected_loss_of_action(prob_range, action),
  label = action_names[action]))
end
```







2. [2pts] Write a function that computes the optimal action given the probability of spam.

```
function optimal_action(prob_spam)
  #TODO: return best action given the probability of spam.
  # Hint: Julia's findmin function might be helpful.
  return findmin([expected_loss_of_action(prob_spam, action) for action in
 1:num_actions])[2]
end
```

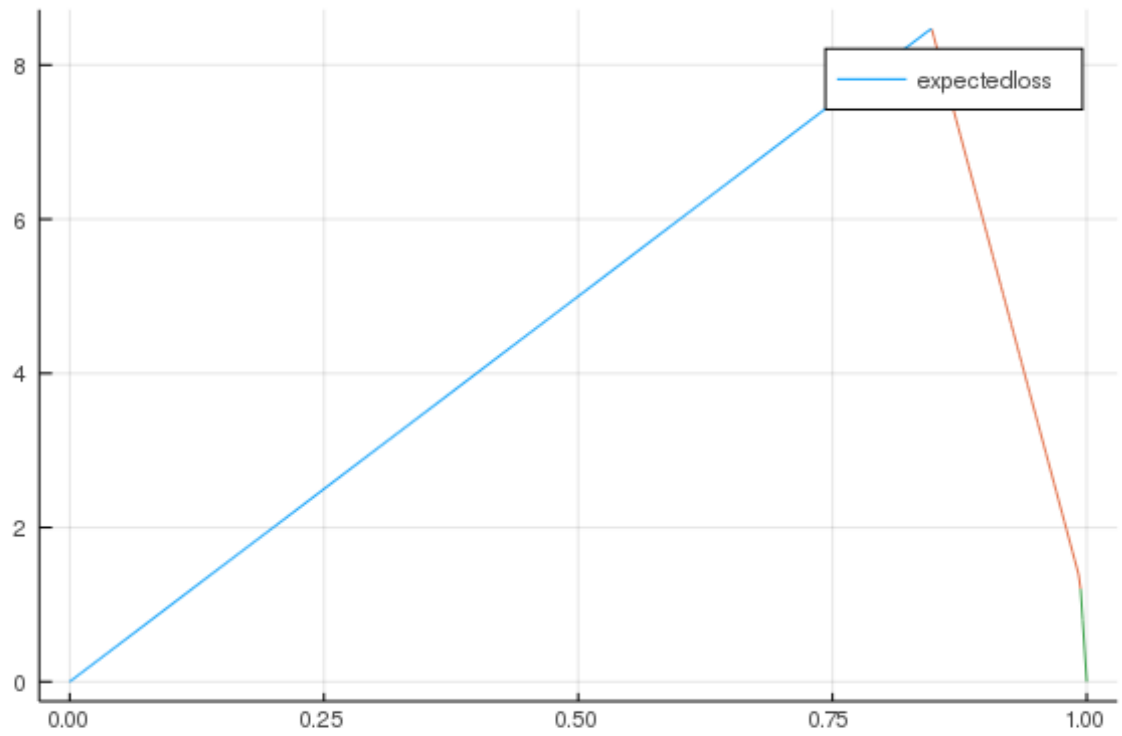
```
using Test
@testset "optimal_action() correct" begin
  @test optimal_action(0.5) == 1
  @test optimal_action(1) == 3
  @test optimal_action(0) == 1
end
```

```
Test Summary:             | Pass  Total
optimal_action() correct |    3      3
Test.DefaultTestSet("optimal_action() correct", Any[], 3, false)
```

3. [4pts] Plot the expected loss of the optimal action as a function of the probability of spam.

Color the line according to the optimal action for that probability of spam.

```
prob_range = range(0, stop=1., length=500)
  optimal_losses = []
  optimal_actions = []
  for p in prob_range
      # TODO:  Compute the optimal action and its expected loss for
      # probability of spam given by p.
      opt = optimal_action(p)
      append!(optimal_actions, opt)
      append!(optimal_losses, expected_loss_of_action(p, opt))
  end

  plot(prob_range, optimal_losses, linecolor=optimal_actions,
  label = "expectedloss")
```



4. [4pts] For exactly which range of the probabilities of an email being spam should we delete an email?

Find the exact answer by hand using algebra. Given probability of spam $p \in [0, 1]$, the expected loss of deleting, foldering and showing such email are respectively $10p, p + 50(1 - p), 200(1 - p)$. Making a decision of deletion is equivalent to expected loss of deletion being smaller than expected loss of the other two actions. We have: $200(1 - p) \leq p + 50(1 - p)$ and $200(1 - p) \leq 10p$, which implies $p \geq \frac{150}{151}$ and $p \geq \frac{20}{21}$. So when $\frac{150}{151} \leq p \leq 1$, we'll delete the mail.

Regression

Manually Derived Linear Regression [10pts]

Suppose that $X \in \mathbb{R}^{m \times n}$ with $n \geq m$ and $Y \in \mathbb{R}^n$, and that $Y \sim \mathcal{N}(X^T \beta, \sigma^2 I)$.

In this question you will derive the result that the maximum likelihood estimate $\hat{\beta}$ of β is given by

$$\hat{\beta} = (XX^T)^{-1} XY$$

1. [1pts] What happens if $n < m$?

In this case $rank(X) = rank(X^T) = n$ so $rank(XX^T) \leq n$ but XX^T is $m \times m$ thus not invertible.

2. [2pts] What are the expectation and covariance matrix of $\hat{\beta}$, for a given true value of β?

$$E(\hat{\beta}) = E((XX^T)^{-1} XY) = (XX^T)^{-1} X E(Y) = (XX^T)^{-1} XX^T \beta = I\beta = \beta$$

3. [2pts] Show that maximizing the likelihood is equivalent to minimizing the squared error $\sum_{i=1}^n (y_i - x_i \beta)^2$. [Hint: Use $\sum_{i=1}^n a_i^2 = a^T a$]

Since $Y \sim N(X^T \beta, \sigma^2 I)$, we have:

$$L(\beta) = f(Y|X, \beta) = \frac{1}{(2\pi)^{n/2} |\sigma^2 I|^{1/2}} exp(-1/2(Y - X^T \beta)^T (\sigma^2 I)^{-1} (Y - X^T \beta))$$

Since exponential function is increasing, maximizing the likelihood function is equivalent to minimizing $(Y - X^T \beta)^T (Y - X^T \beta) = \sum_{i=1}^n (y_i - x_i \beta)^2$ with respect to β, where x_i is the i^{th} column of X.

4. [2pts] Write the squared error in vector notation, (see above hint), expand the expression, and collect like terms. [Hint: Use $\beta^T x^T y = y^T x\beta$ and $x^T x$ is symmetric]

$$\begin{aligned}
\sum_{i=1}^{n}(y_i - x_i\beta)^2 &= (Y - X^T\beta)^T(Y - X^T\beta) && (1 \\
&= (Y^T - \beta^T X)(Y - X^T\beta) && (2 \\
&= Y^T Y - \beta^T XY - Y^T X^T\beta + \beta^T X^T X\beta && (3 \\
&= Y^T Y - 2Y^T X^T\beta + \beta^T X^T X\beta \qquad \text{because } Y^T X^T\beta \text{ is symmetric} && (4
\end{aligned}$$

5. [3pts] Use the likelihood expression to write the negative log-likelihood. Write the derivative of the negative log-likelihood with respect to β, set equal to zero, and solve to show the maximum likelihood estimate $\hat{\beta}$ as above.

$$L(\beta) = f(Y|X,\beta) = \frac{1}{(2\pi)^{n/2}|\sigma^2 I|^{1/2}} exp(-1/(2\sigma^2)I(Y^T Y - 2\beta^T XY + \beta^T X^T X\beta))$$

We have:

$$\begin{aligned}
l(\beta) &= -log(L(\beta)) && (5) \\
&= -log(\frac{1}{(2\pi)^{n/2}|\sigma^2 I|^{1/2}}) - 1/(2\sigma^2)I(Y^T Y - 2\beta^T XY + \beta^T X^T X\beta)) && (6) \\
l'(\beta) &= -1/(2\sigma^2)(0 - 2XY + (XX^T + (XX^T)^T)\beta) && (7) \\
&= -1/(2\sigma^2)(-2XY + 2XX^T\beta) && (8) \\
&= 0 && (9) \\
2XX^T\beta &= 2XY && (10) \\
\hat{\beta} &= (XX^T)^{-1}XY && (11)
\end{aligned}$$

And $l''(\beta) = -1/(2\sigma^2)(2X^T X) < 0$ so $\hat{\beta}$ indeed is a maximum.

Toy Data [2pts]

For visualization purposes and to minimize computational resources we will work with 1-dimensional toy data.

That is $X \in \mathbb{R}^{m\times n}$ where $m = 1$.

We will learn models for 3 target functions

- `target_f1`, linear trend with constant noise.
- `target_f2`, linear trend with heteroskedastic noise.
- `target_f3`, non-linear trend with heteroskedastic noise.

```
using LinearAlgebra
```

```
function target_f1(x, σ_true=0.3)
  noise = randn(size(x))
  y = 2x .+ σ_true.*noise
  return vec(y)
end

function target_f2(x)
  noise = randn(size(x))
  y = 2x + norm.(x)*0.3.*noise
  return vec(y)
end

function target_f3(x)
  noise = randn(size(x))
  y = 2x + 5sin.(0.5*x) + norm.(x)*0.3.*noise
  return vec(y)
end
```

```
target_f3 (generic function with 1 method)
```

1. [1pts] Write a function which produces a batch of data $x \sim \text{Uniform}(0, 20)$ and `y = target_f(x)`

```
function sample_batch(target_f, batch_size)
  x = (20*rand(batch_size))'
  y = target_f(x)
  return (x,y)
end
```

```
sample_batch (generic function with 1 method)
```

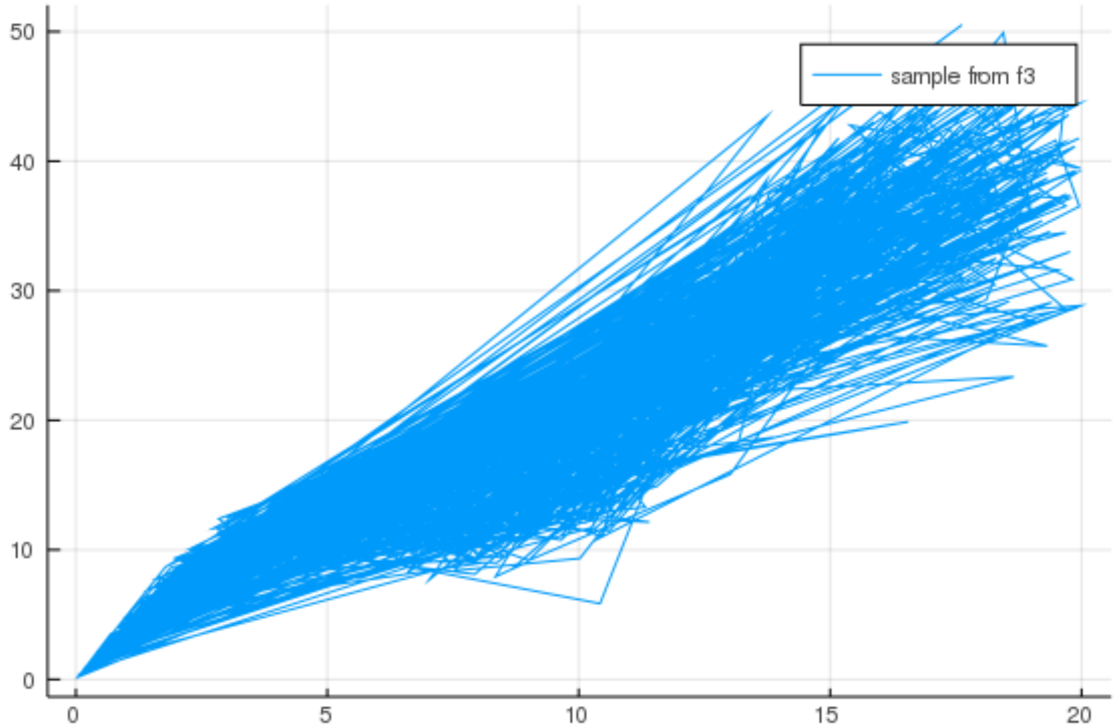
```
using Test
@testset "sample dimensions are correct" begin
  m = 1 # dimensionality
  n = 200 # batch-size
  for target_f in (target_f1, target_f2, target_f3)
    x,y = sample_batch(target_f,n)
    @test size(x) == (m,n)
    @test size(y) == (n,)
  end
end
```

```
Test Summary:                 | Pass  Total
sample dimensions are correct |    6      6
Test.DefaultTestSet("sample dimensions are correct", Any[], 6, false)
```

2. [1pts] For all three targets, plot a $n = 1000$ sample of the data. **Note: You will use these plots later, in your writeup display once other questions are**

complete.

```
using Plots
n = 1000
x1,y1 = sample_batch(target_f1,n)
plot_f1 = plot(x1', y1, label = "sample from f1")

x2,y2 = sample_batch(target_f2,n)
plot_f2 = plot(x2', y2, label = "sample from f2")

x3,y3 = sample_batch(target_f3,n)
plot_f3 = plot(x3', y3, label = "sample from f3")
```



Linear Regression Model with $\hat{\beta}$ MLE [4pts]

1. [2pts] Program the function that computes the the maximum likelihood estimate given X and Y. Use it to compute the estimate $\hat{\beta}$ for a $n = 1000$ sample from each target function.

```
function beta_mle(X,Y)
      beta = inv(X*X')*X*Y
      return beta
    end

    using Test
    @testset "beta_mle computes correctly" begin
        X = [3 4]
```

```
        Y = [1, 2]
        @test beta_mle(X,Y) == [11/25]
    end
```

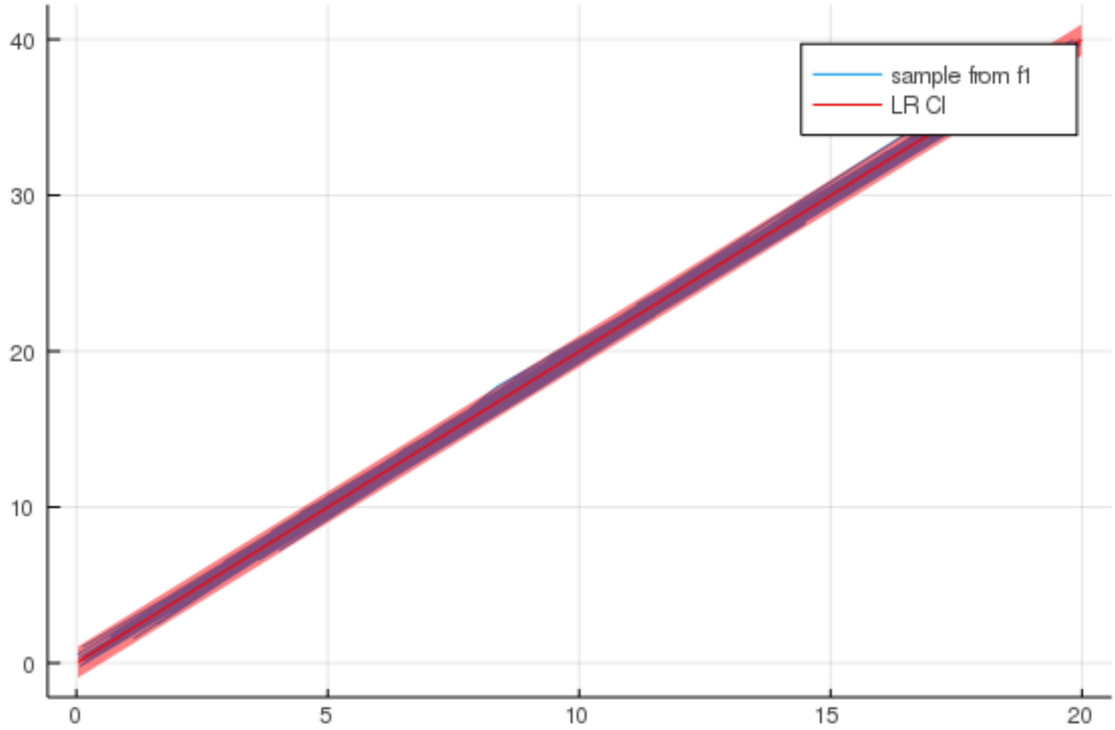
```
Test Summary:                | Pass  Total
beta_mle computes correctly |    1      1
```

```
n=1000 # batch_size

    x_1, y_1 = sample_batch(target_f1,n)
    β_mle_1 = beta_mle(x_1,y_1)

    x_2, y_2 = sample_batch(target_f2,n)
    β_mle_2 = beta_mle(x_2,y_2)

    x_3, y_3 = sample_batch(target_f2,n)
    β_mle_3 = sample_batch(target_f3,n)
```

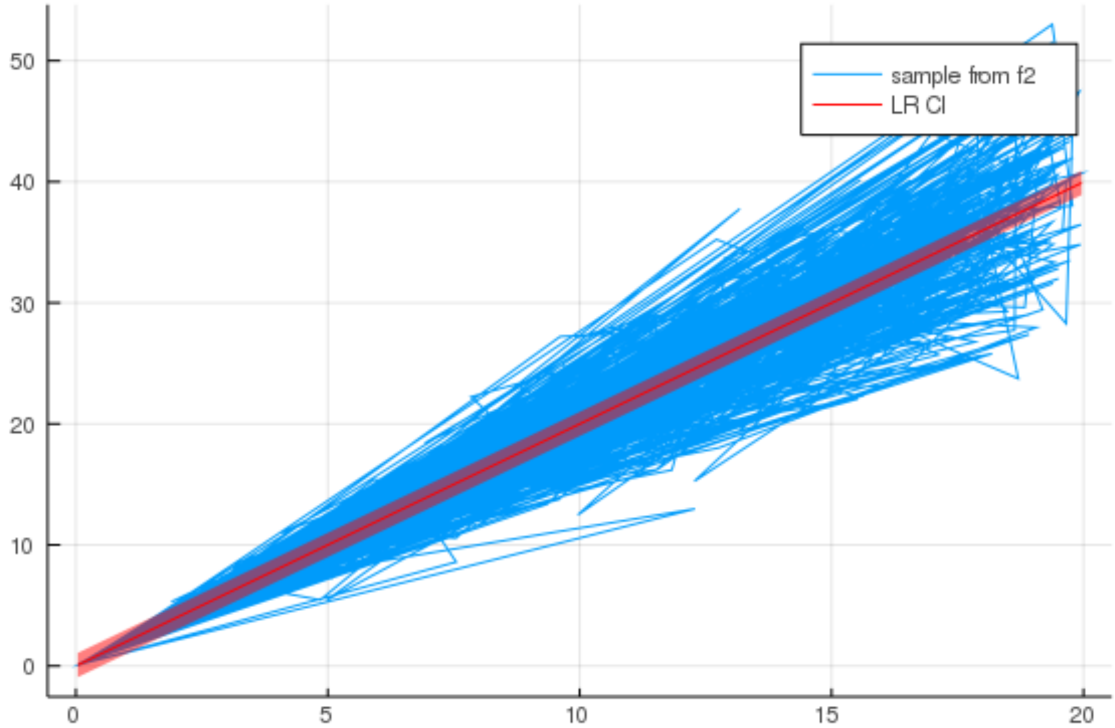
```
([14.260948854266701 10.722549455026424 … 13.849667628918612 1.7723043000
54
2926], [38.47144209603627, 12.075127609109, 32.103677439234886, 35.053252
32
163048, 54.48153679061453, 8.01851751878531, 20.118763825299354, 14.04295
67
90615733, 9.720755041714336, 29.585363355099066  …  3.222014625029063, 12
.2
48829815925092, 27.16716340852801, 30.85397211443513, 10.83000418692313,
27
.780332862741847, 33.963594416567844, 11.339237288725244, 30.554740990584
86
, 6.113399697528932])
```

2. [2pts] For each function, plot the linear regression model given by $Y \sim \mathcal{N}(X^T\hat{\beta}, \sigma^2 I)$ for $\sigma = 1.$. This plot should have the line of best fit given by the maximum likelihood estimate, as well as a shaded region around the line corresponding to plus/minus one standard deviation (i.e. the fixed uncertainty $\sigma = 1.0$). Using `Plots.jl` this shaded uncertainty region can be achieved with the `ribbon` keyword argument. **Display 3 plots, one for each target function, showing samples of data and maximum likelihood estimate linear regression model**

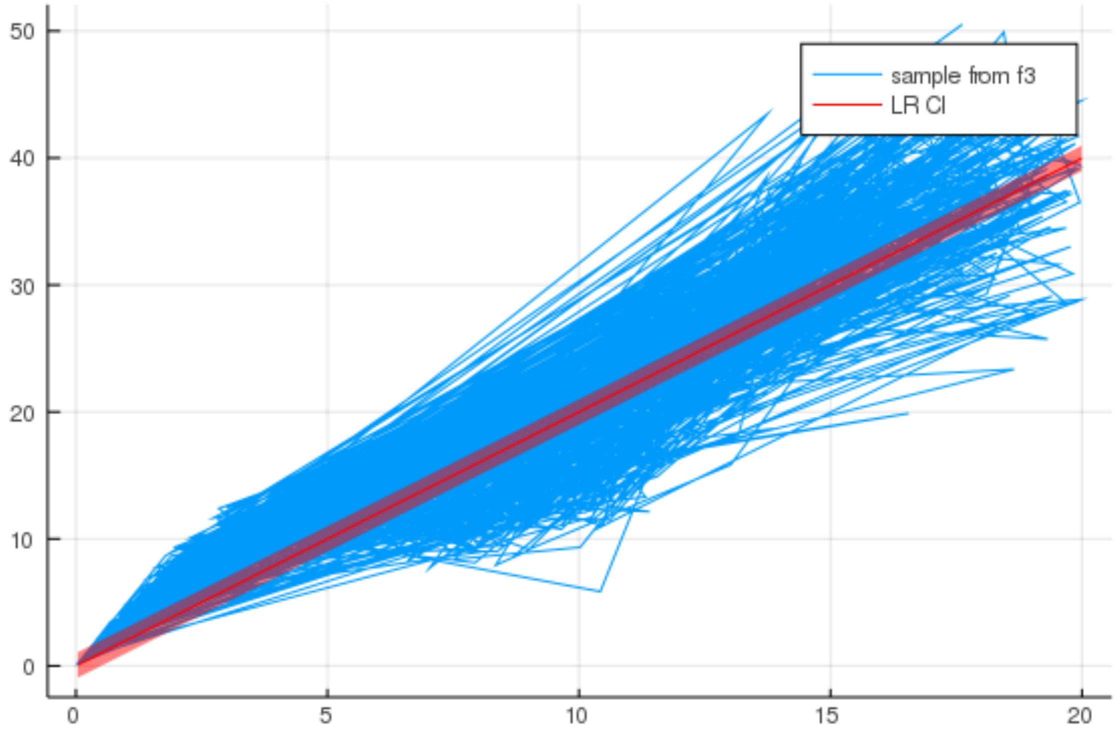
```
sort!(x_1')
plot!(plot_f1, x_1', (x_1)'*β_mle_1,color = "red",ribbon = 1, label = "LR
CI")
```




```
sort!(x2')
plot!(plot_f2, x2', (x2)'*β_mle_1,color = "red",ribbon = 1, label = "LR CI
")
```




```
sort!(x3')
plot!(plot_f3, (x3)', (x3)'*β_mle_1 ,color = "red", ribbon = 1, label = "L
R CI")
```



Log-likelihood of Data Under Model [6pts]

1. [2pts] Write code for the function that computes the likelihood of x under the Gaussian distribution $\mathcal{N}(\mu, \sigma)$. For reasons that will be clear later, this function should be able to broadcast to the case where x, μ, σ are all vector valued and return a vector of likelihoods with equivalent length, i.e., $x_i \sim \mathcal{N}(\mu_i, \sigma_i)$.

```
function gaussian_log_likelihood(μ, σ, x)
  """
  compute log-likelihood of x under N(μ,σ)
  """
  n = length(x)
  return -1/2*log(2*pi) - log(σ) - 1/(2σ^2)*(x-μ)^2
end
```

```
gaussian_log_likelihood (generic function with 1 method)
```

```
# Test Gaussian likelihood against standard implementation
@testset "Gaussian log likelihood" begin
using Distributions: pdf, Normal
# Scalar mean and variance
x = randn()
μ = randn()
σ = rand()
@test size(gaussian_log_likelihood(μ,σ,x)) == () # Scalar log-likelihood
@test gaussian_log_likelihood.(μ,σ,x) ≈ log.(pdf.(Normal(μ,σ),x)) # Correc
```

```
t Value
# Vector valued x under constant mean and variance
x = randn(100)
μ = randn()
σ = rand()
@test size(gaussian_log_likelihood.(μ,σ,x)) == (100,) # Vector of log-like
lihoods
@test gaussian_log_likelihood.(μ,σ,x) ≈ log.(pdf.(Normal(μ,σ),x)) # Correc
t Values
# Vector valued x under vector valued mean and variance
x = randn(10)
μ = randn(10)
σ = rand(10)
@test size(gaussian_log_likelihood.(μ,σ,x)) == (10,) # Vector of log-likel
ihoods
@test gaussian_log_likelihood.(μ,σ,x) ≈ log.(pdf.(Normal.(μ,σ),x)) # Corre
ct Values
end
```

```
Test Summary:            | Pass  Total
Gaussian log likelihood |    6      6
Test.DefaultTestSet("Gaussian log likelihood", Any[], 6, false)
```

2. [2pts] Use your gaussian log-likelihood function to write the code which computes the negative log-likelihood of the target value Y under the model $Y \sim \mathcal{N}(X^T\beta, \sigma^2 * I)$ for a given value of β.

```
function lr_model_nll(β,x,y;σ = 1)
      return -sum(gaussian_log_likelihood.(x'*β, σ, y))
    end
```

```
lr_model_nll (generic function with 1 method)
```

3. [1pts] Use this function to compute and report the negative-log-likelihood of a $n \in \{10, 100, 1000\}$ batch of data under the model with the maximum-likelihood estimate $\hat{\beta}$ and $\sigma \in \{0.1, 0.3, 1., 2.\}$ for each target function.

```
for n in (10,100,1000)
    println("--------  $n  ------------")
    for target_f in (target_f1,target_f2, target_f3)
      println("--------  $target_f  ------------")
      for σ_model in (0.1,0.3,1.,2.)
        println("--------  $σ_model  ------------")
        x,y = sample_batch(target_f, n)
        β_mle = beta_mle(x, y)
        nll = lr_model_nll(β_mle, x, y, σ = σ_model)
        println("Negative Log-Likelihood: $nll")
      end
    end
end
```

```
---------  10  ------------
---------  target_f1  ------------
---------  0.1  ------------
Negative Log-Likelihood: 6.148952499748012
---------  0.3  ------------
Negative Log-Likelihood: -0.38529149271061536
---------  1.0  ------------
Negative Log-Likelihood: 9.396544215505536
---------  2.0  ------------
Negative Log-Likelihood: 16.208496993767056
---------  target_f2  ------------
---------  0.1  ------------
Negative Log-Likelihood: 5047.600566641601
---------  0.3  ------------
Negative Log-Likelihood: 388.50168992821494
---------  1.0  ------------
Negative Log-Likelihood: 127.93116364837448
---------  2.0  ------------
Negative Log-Likelihood: 46.20738450813294
---------  target_f3  ------------
---------  0.1  ------------
Negative Log-Likelihood: 6268.822163738381
---------  0.3  ------------
Negative Log-Likelihood: 954.5128766044058
---------  1.0  ------------
Negative Log-Likelihood: 110.04274597533399
---------  2.0  ------------
Negative Log-Likelihood: 49.63749972540195
---------  100  ------------
---------  target_f1  ------------
---------  0.1  ------------
Negative Log-Likelihood: 369.9345827448195
---------  0.3  ------------
Negative Log-Likelihood: 28.029805589769623
---------  1.0  ------------
Negative Log-Likelihood: 96.34930732336086
---------  2.0  ------------
Negative Log-Likelihood: 162.18528814274373
---------  target_f2  ------------
---------  0.1  ------------
Negative Log-Likelihood: 69835.4850987314
---------  0.3  ------------
Negative Log-Likelihood: 5502.330626964081
---------  1.0  ------------
Negative Log-Likelihood: 668.9728226091892
---------  2.0  ------------
Negative Log-Likelihood: 314.93945599078467
---------  target_f3  ------------
---------  0.1  ------------
Negative Log-Likelihood: 102503.57895022973
---------  0.3  ------------
Negative Log-Likelihood: 12825.921110950843
---------  1.0  ------------
```

```
Negative Log-Likelihood: 1102.0738150190136
--------  2.0  ------------
Negative Log-Likelihood: 440.13565818412553
--------  1000  ------------
--------  target_f1  ------------
--------  0.1  ------------
Negative Log-Likelihood: 3364.0068663156503
--------  0.3  ------------
Negative Log-Likelihood: 235.52988526931057
--------  1.0  ------------
Negative Log-Likelihood: 962.1958699029625
--------  2.0  ------------
Negative Log-Likelihood: 1623.289418676548
--------  target_f2  ------------
--------  0.1  ------------
Negative Log-Likelihood: 652440.9388845653
--------  0.3  ------------
Negative Log-Likelihood: 68009.12404613444
--------  1.0  ------------
Negative Log-Likelihood: 7043.566193212158
--------  2.0  ------------
Negative Log-Likelihood: 3308.00199662571
--------  target_f3  ------------
--------  0.1  ------------
Negative Log-Likelihood: 1.1554648911550445e6
--------  0.3  ------------
Negative Log-Likelihood: 131631.3356157521
--------  1.0  ------------
Negative Log-Likelihood: 13204.708287004929
--------  2.0  ------------
Negative Log-Likelihood: 4607.046543386995
```

4. [1pts] For each target function, what is the best choice of σ?

Please note that σ and batch-size n are modelling hyperparameters. In the expression of maximum likelihood estimate, σ or n do not appear, and in principle shouldn't affect the final answer. However, in practice these can have significant effect on the numerical stability of the model. Too small values of σ will make data away from the mean very unlikely, which can cause issues with precision. Also, the negative log-likelihood objective involves a sum over the log-likelihoods of each datapoint. This means that with a larger batch-size n, there are more datapoints to sum over, so a larger negative log-likelihood is not necessarily worse. The take-home is that you cannot directly compare the negative log-likelihoods achieved by these models with different hyperparameter settings. The best choice of σ for f_1 is 0.3; The best choice of σ for f_2 is 2.0;The best choice of σ for f_1 is 2.0.

Automatic Differentiation and Maximizing

Likelihood [3pts]

In a previous question you derived the expression for the derivative of the negative log-likelihood with respect to β. We will use that to test the gradients produced by automatic differentiation.

1. [3pts] For a random value of β, σ, and $n = 100$ sample from a target function, use automatic differentiation to compute the derivative of the negative log-likelihood of the sampled data with respect β. Test that this is equivalent to the hand-derived value.

```
using Zygote: gradient
using Test
@testset "Gradients wrt parameter" begin
β_test = randn()
σ_test = rand()
x,y = sample_batch(target_f1,100)
ad_grad = gradient((β -> lr_model_nll(β,x,y;σ = σ_test)), β_test)
hand_derivative = 1/(2*σ_test^2)*(-2*x*y + 2*x*x'*β_test)
@test ad_grad[1] ≈ hand_derivative
end
```

```
Test Summary:           | Pass  Total
Gradients wrt parameter |    1      1
Test.DefaultTestSet("Gradients wrt parameter", Any[], 1, false)
```

Train Linear Regression Model with Gradient Descent [5pts]

In this question we will compute gradients of of negative log-likelihood with respect to β. We will use gradient descent to find β that maximizes the likelihood.

1. [3pts] Write a function `train_lin_reg` that accepts a target function and an initial estimate for β and some hyperparameters for batch-size, model variance, learning rate, and number of iterations. Then, for each iteration:

    - sample data from the target function
    - compute gradients of negative log-likelihood with respect to β
    - update the estimate of β with gradient descent with specified learning rate

    and, after all iterations, returns the final estimate of β.

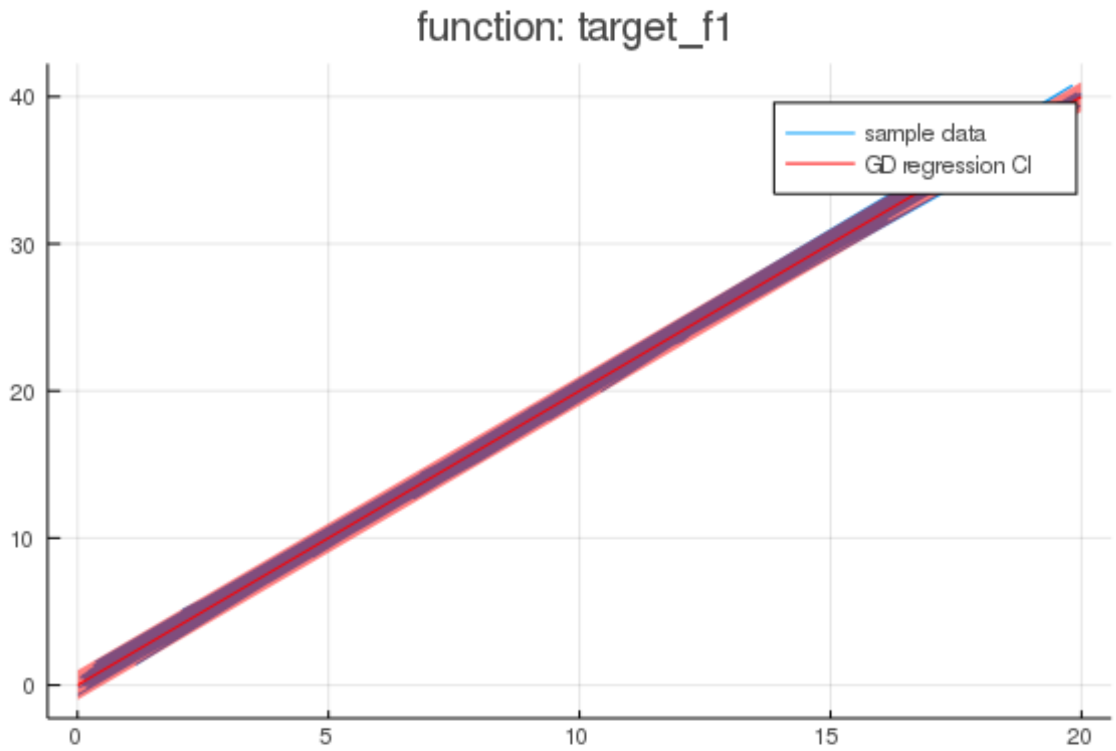
```
using Logging # Print training progress to REPL, not pdf

function train_lin_reg(target_f, β_init; bs= 100, lr = 1e-6, iters=1000, σ
```

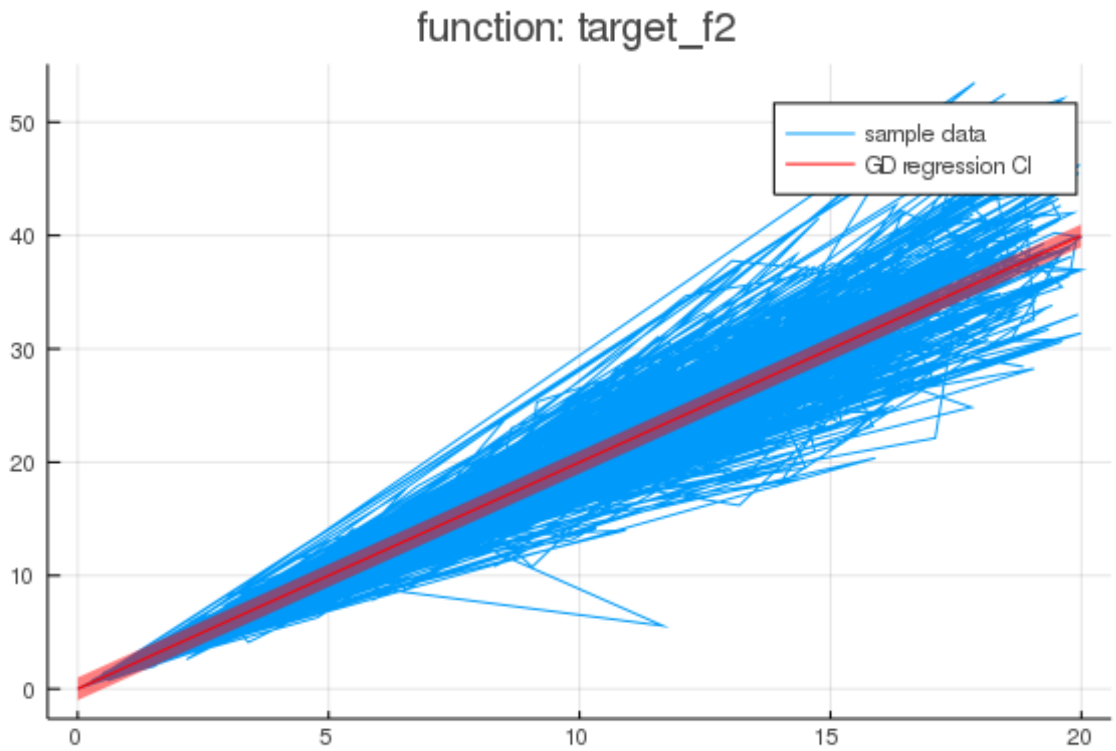
```
_model = 1.)
    @info "function: $target_f"
    β_curr = β_init
    for i in 1:iters
      x,y = sample_batch(target_f, bs)
      log_loss = lr_model_nll(β_curr, x, y)
      @info "loss: $log_loss  β: $β_curr"
      grad_β = gradient(β -> (lr_model_nll(β, x, y)),β_curr)#: compute gra
dients
      β_curr = β_curr - grad_β[1]*lr #gradient descent
    end
    return β_curr
end
```

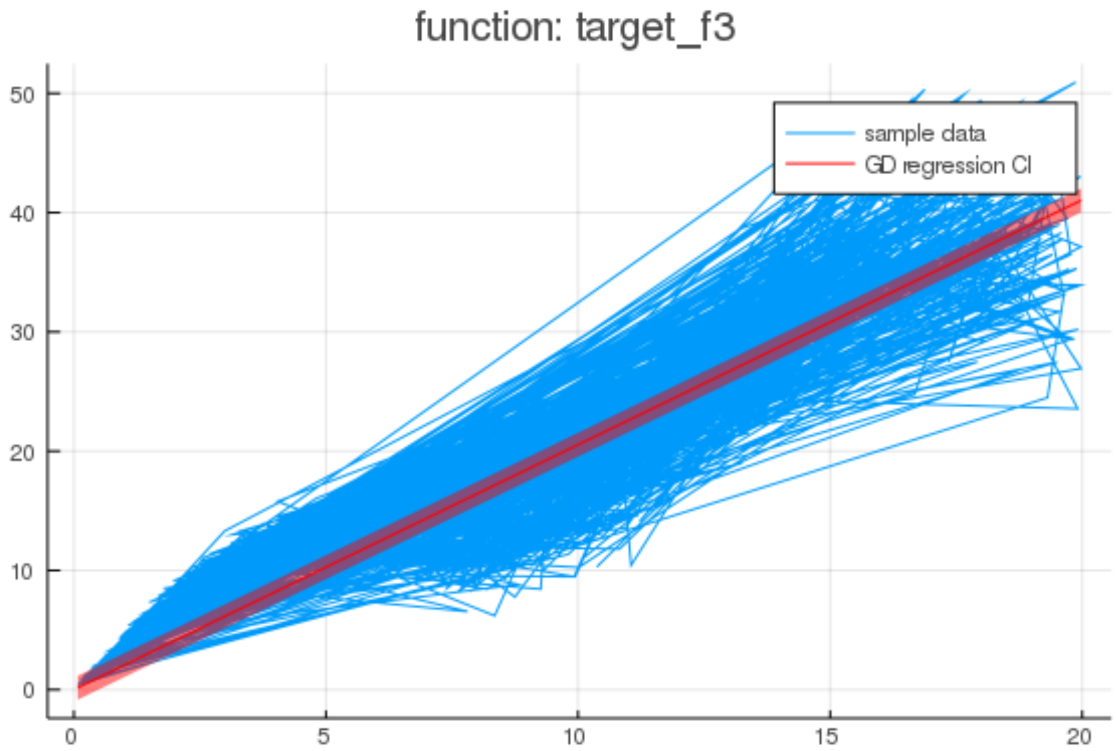
```
train_lin_reg (generic function with 1 method)
```

2. [2pts] For each target function, start with an initial parameter β, learn an estimate for β_{learned} by gradient descent. Then plot a $n = 1000$ sample of the data and the learned linear regression model with shaded region for uncertainty corresponding to plus/minus one standard deviation.

```
using Plots
β_init = 1000*randn() # Initial parameter
#β_learned= train_lin_reg(): #call training function
for target_f in (target_f1, target_f2, target_f3)
    β_learned = train_lin_reg(target_f, β_init)
    n = 1000
    x,y = sample_batch(target_f, n)
    plot(x',y, label = "sample data", title = "function: $target_f")
    sort!(x')
    display(plot!(x', x' * β_learned, color = "red", ribbon = 1, label = "
GD regression CI"))
end
```


function: target_f1
sample data
GD regression CI
0
10
20
30
40
0
5
10
15
20


function: target_f2
sample data
GD regression CI
0
10
20
30
40
50
0
5
10
15
20



Non-linear Regression with a Neural Network [9pts]

In the previous questions we have considered a linear regression model

$$Y \sim \mathcal{N}(X^T\beta, \sigma^2)$$

This model specified the mean of the predictive distribution for each datapoint by the product of that datapoint with our parameter.

Now, let us generalize this to consider a model where the mean of the predictive distribution is a non-linear function of each datapoint. We will have our non-linear model be a simple function called `neural_net` with parameters θ (collection of weights and biases).

$$Y \sim \mathcal{N}(\texttt{neural\_net}(X, \theta), \sigma^2)$$

1. [3pts] Write the code for a fully-connected neural network (multi-layer perceptron) with one 10-dimensional hidden layer and a `tanh` nonlinearirty. You must write this yourself using only basic operations like matrix multiply and `tanh`, you may not use layers provided by a library.
   This network will output the mean vector, test that it outputs the correct shape for some random parameters.

```
# Neural Network Function
#x:1*n
```

```
function neural_net(x,θ)
  W1 = θ[1]
  b1 = θ[2]
  W2 = θ[3]
  b2 = θ[4]
  H = tanh.(W1*x' + b1)
  return W2*H + b2
end



# Random initial Parameters
n = 100
θ = [randn(10,n), randn(10,),randn(n,10),randn(n,)]
x,y = sample_batch(target_f1,n)
@testset "neural net mean vector output" begin
x,y = sample_batch(target_f1,n)
μ = neural_net(x,θ)
@test size(μ) == (n,)
end
```

```
Test Summary:                 | Pass  Total
neural net mean vector output |    1      1
Test.DefaultTestSet("neural net mean vector output", Any[], 1, false)
```

2. [2pts] Write the code that computes the negative log-likelihood for this model where the mean is given by the output of the neural network and $\sigma = 1.0$

```
#x is 1*n, y is n*1
function nn_model_nll(θ,x,y;σ=1)
  μ = neural_net(x,θ)
  return -sum(gaussian_log_likelihood.(μ, σ, y))
end
```

```
nn_model_nll (generic function with 1 method)
```

3. [2pts] Write a function `train_nn_reg` that accepts a target function and an initial estimate for θ and some hyperparameters for batch-size, model variance, learning rate, and number of iterations. Then, for each iteration:

   - sample data from the target function
   - compute gradients of negative log-likelihood with respect to θ
   - update the estimate of θ with gradient descent with specified learning rate

   and, after all iterations, returns the final estimate of θ.

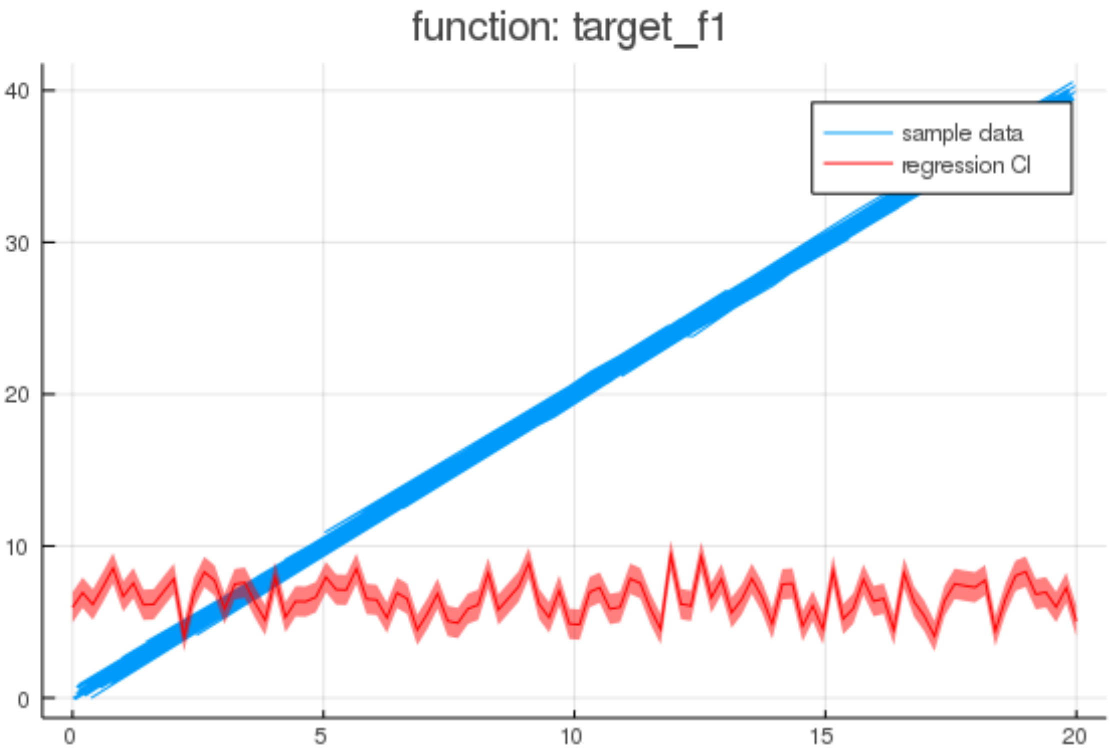
```
using Logging # Print training progress to REPL, not pdf

function train_nn_reg(target_f, θ_init; bs= 100, lr = 1e-5, iters=1000, σ_
model = 1. )
```

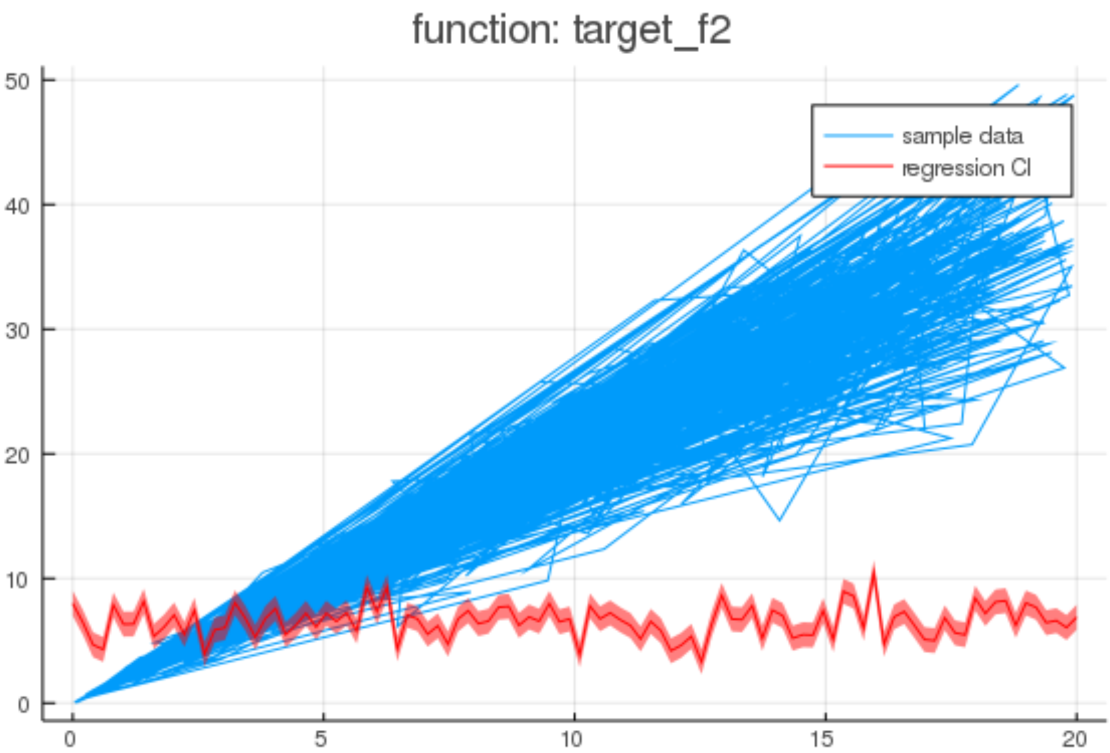
```
    θ_curr = θ_init
    for i in 1:iters
      x,y = sample_batch(target_f, bs)
      log_loss = nn_model_nll(θ_curr, x, y)
      @info "loss: $log_loss" #TODO: log loss, if you want to montior trai
ning
      grad_θ = gradient((θ -> nn_model_nll(θ,x,y; σ = σ_model)), θ_curr)[1
]
      #: compute gradients
      for j in 1:4
        θ_curr[j] = θ_curr[j] - lr .* grad_θ[j]#:gradient descent
      end
    end
    return θ_curr
end
```

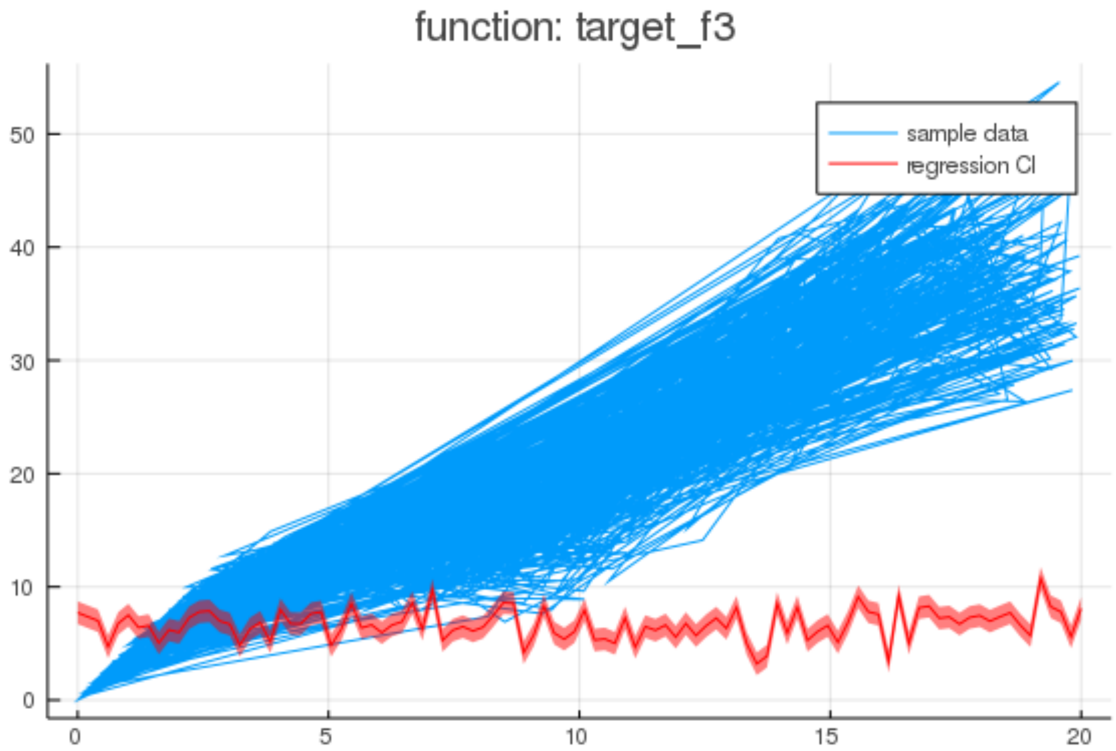
```
train_nn_reg (generic function with 1 method)
```

4. [2pts]

```
using Zygote:gradient
n = 1000
m = 100
xrange = convert(Array, range(0, 20, length=m))
for target_f in (target_f1, target_f2, target_f3)
  θ_init = [rand(10,m), randn(10,),rand(m,10),randn(m,)]
  θ_learned = train_nn_reg(target_f, θ_init)
  x,y = sample_batch(target_f, n)
  plot(x',y, label = "sample data", title = "function: $target_f")
  display(plot!(xrange, neural_net(xrange', θ_learned), color = "red", rib
bon = 1, label = "regression CI"))
end
```


function: target_f1
sample data
regression CI
0
10
20
30
40
0
5
10
15
20


function: target_f2
sample data
regression CI
0
10
20
30
40
50
0
5
10
15
20



Non-linear Regression and Input-dependent Variance with a Neural Network

$$\mu, \log\sigma = \texttt{neural\_net}(X, \theta)$$
$$Y \sim \mathcal{N}(\mu, \exp(\log\sigma)^2)$$

1. [1pts]

```
# Neural Network Function
function neural_net_w_var(x,θ)
  W11 = θ[1]
  b11 = θ[2]
  W12 = θ[3]
  b12 = θ[4]
  W21 = θ[5]
  b21 = θ[6]
  W22 = θ[7]
  b22 = θ[8]
  H1 = tanh.(W11*x' + b11)
  μ_hat = W12*H1 + b12
  H2 = tanh.(W21*x' + b21)
  log_σ = W22*H2 + b22
  return (μ_hat, log_σ)
end


@testset "neural net mean and logsigma vector output" begin
n = 100
θ = [rand(10,n), randn(10,),rand(n,10),randn(n,),
```

```
    rand(10,n), randn(10,),rand(n,10),randn(n,)]
x,y = sample_batch(target_f1,n)
μ, logσ = neural_net_w_var(x,θ)
@test size(μ) == (n,)
@test size(logσ) == (n,)
end
```

```
Test Summary:                              | Pass  Total
neural net mean and logsigma vector output |    2      2
Test.DefaultTestSet("neural net mean and logsigma vector output", Any[],
2,
 false)
```

2. [2pts]

```
#input size:θ = [rand(10,n), randn(10,),rand(n,10),randn(n,),
#    rand(10,n), randn(10,),rand(n,10),randn(n,)]
#x:1*n
#y:n*1
function nn_with_var_model_nll(θ,x,y)
  μ,log_σ = neural_net_w_var(x, θ)
  σ = exp.(log_σ)
  return -sum(gaussian_log_likelihood.(μ, σ, y))
end
```

```
nn_with_var_model_nll (generic function with 1 method)
```

3. [1pts]

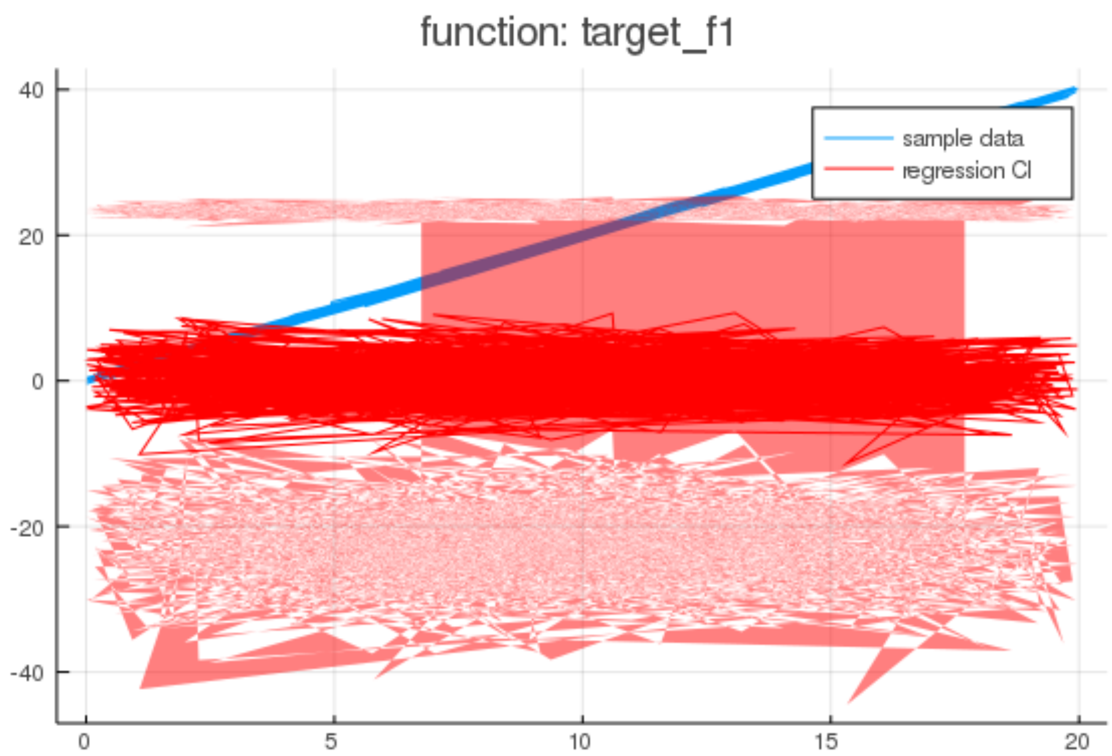
```
function train_nn_w_var_reg(target_f, θ_init; bs= 100, lr = 1e-4, iters=10
000)
    θ_curr = θ_init
    for i in 1:iters
      x,y = sample_batch(target_f, bs)
      log_loss = nn_with_var_model_nll(θ_curr, x, y)
      @info "loss: $log_loss"
      grad_θ = gradient((θ -> nn_with_var_model_nll(θ, x, y)), θ_curr)[1]
#compute gradients
      for j in 1:8
        θ_curr[j] = θ_curr[j] - lr .* grad_θ[j]#:gradient descent end
      end
    end
    return θ_curr
end
```

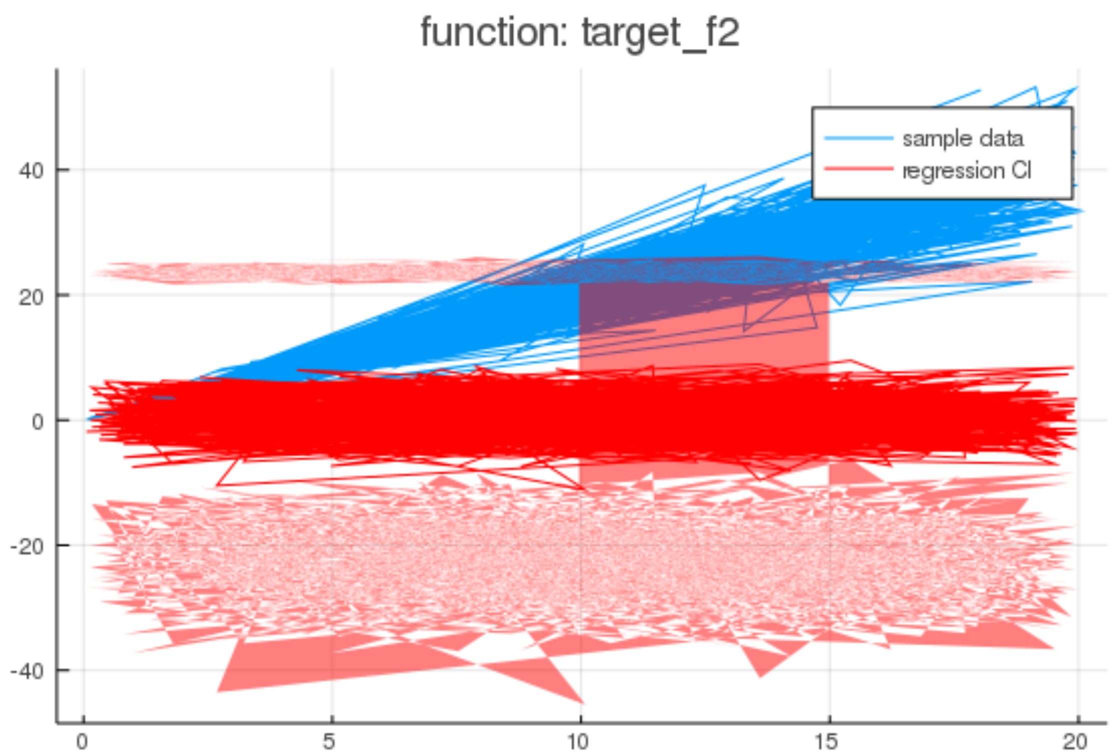
```
train_nn_w_var_reg (generic function with 1 method)
```

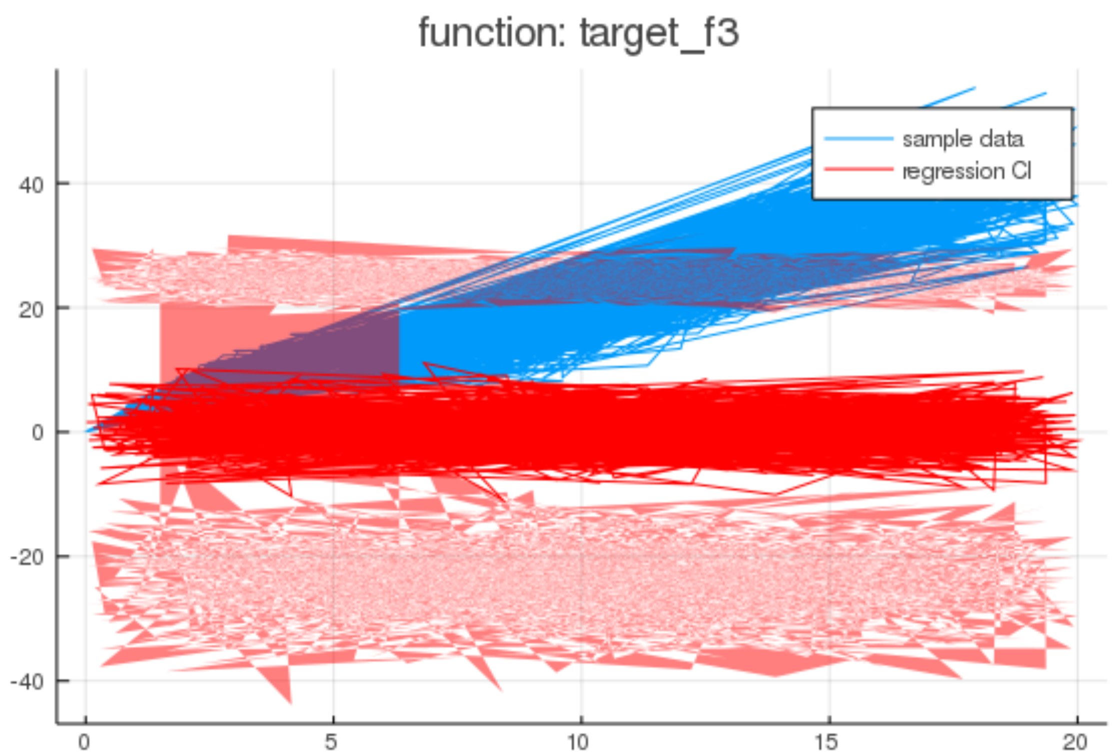
4. [4pts]

```
N = 1000
```

```
m = 1000
xrange = convert(Array, range(0, 20, length=m))
for target_f in (target_f1, target_f2, target_f3)
  θ_init = [randn(10,m), randn(10,),randn(m,10),randn(m,),
            rand(10,m), randn(10,),rand(m,10),randn(m,)]
  θ_learned = train_nn_w_var_reg(target_f, θ_init; bs = m)
  x,y = sample_batch(target_f, m)
  plot(x',y, label = "sample data", title = "function: $target_f")
  μ, log_σ = neural_net_w_var(x,θ_learned)
  σ = exp.(log_σ)
  display(plot!(x', μ, color = "red", ribbon = σ, label = "regression CI")
)
end
```







Published from [A1.jmd]() using [Weave.jl]() on 2020-02-26.