Assignment 1

Student Name Sheng Gao and # 1002093584

February 24, 2020

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

1 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: $p(\text{spam}|\text{email})$. 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, $L(\text{action}, \text{spam})$:

Action	Spam	Not spam
Show	10	0
Folder	1	50
Delete	0	200

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

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

num_actions = length(losses)

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

prob_range = range(0., stop=1., length=500)
# Make plot
```

```
using Plots
for action in 1:num_actions
  if action == 1
    plot(prob_range, expected_loss_of_action(prob_range, action), label=action)
  elseif action == 2
    plot!(prob_range, expected_loss_of_action(prob_range, action), label=action)
  else
    display(plot!(prob_range, expected_loss_of_action(prob_range, action), label=action,
xaxis="prob of spam", yaxis="expected_loss", title="expected_loss_of_action"))
  end
end
```

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

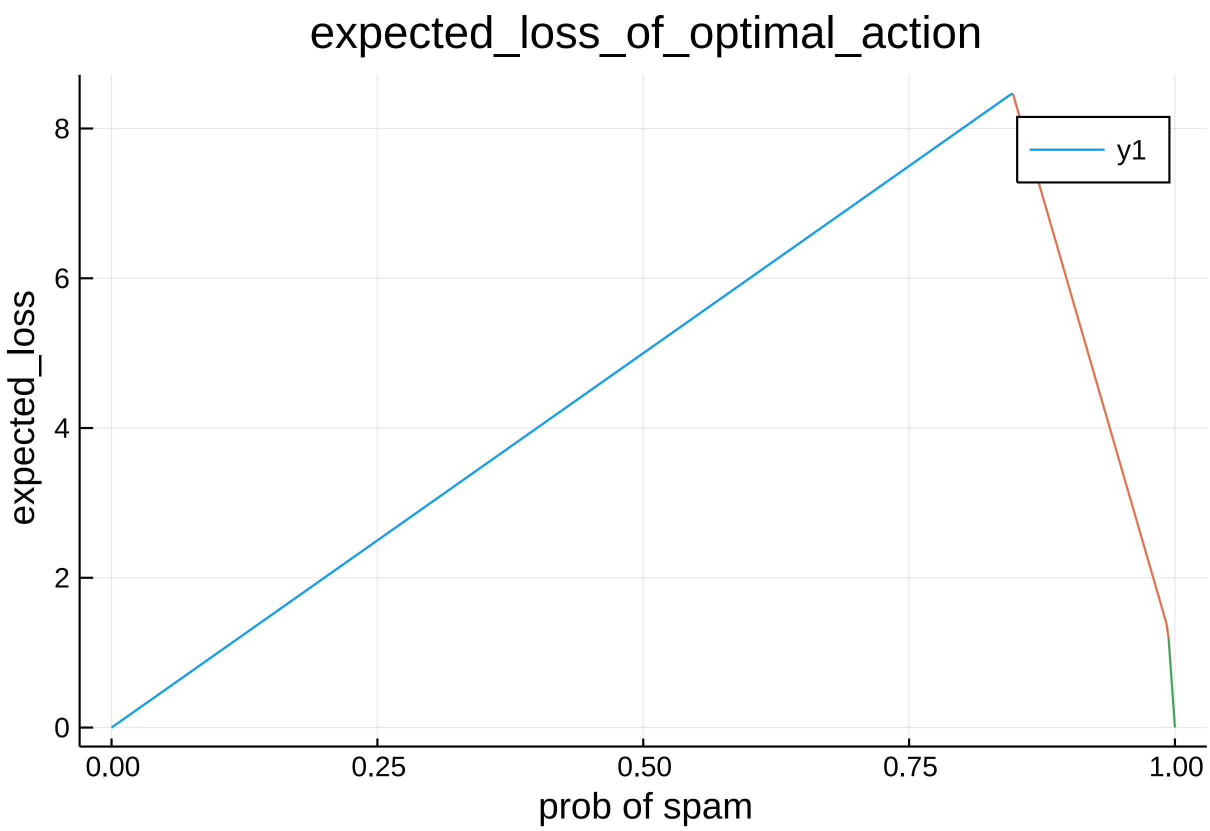
```
function optimal_action(prob_spam)
    x = prob_range
    for action in 1:num_actions
        x = hcat(x, expected_loss_of_action(prob_range, action))
    end
    for row in eachrow(x)
        if row[1] == prob_spam
            return findmin(row[2:4])[2]
        end
    end
    #TODO: return best action given the probability of spam.
    # Hint: Julia's findmin function might be helpful.
end
```

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

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
    append!(optimal_actions, optimal_action(p))
    append!(optimal_losses, expected_loss_of_action(p, optimal_action((p))))
    # TODO:  Compute the optimal action and its expected loss for
    # probability of spam given by p.
end
plot(prob_range, optimal_losses, linecolor=optimal_actions, xaxis="prob of spam",
yaxis="expected_loss", title="expected_loss_of_optimal_action")
```



Blue line is action 1, orange line is action 2, green line is action 3.

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.

$$\begin{cases} 0 * p + 200 * (1-p) \leqslant 10 * p + 0 * (1-p) \\ 0 * p + 200 * (1-p) \leqslant 1 * p + 50 * (1-p) \\ 0 \leqslant p \leqslant 1 \end{cases}$$

We solve $\frac{150}{151} \leqslant p \leqslant 1$. Therefore, when probability of an email being spam in the range of $\frac{150}{151} \leqslant p \leqslant 1$, we should delete an email.

2 Regression

2.1 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 $n < m$?

This is a problem is known as Perfect multicollinearity, which means too little data compared to number of parameters. Finally, β would be non-identifiable, which means β has no unique solution.

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

$$\begin{aligned}
&\mathbb{E}(\hat{\beta}) = \mathbb{E}((XX^T)^{-1}XY) \\
&= \mathbb{E}((XX^T)^{-1}XX^T\beta) \text{ since: } Y \sim \mathcal{N}(X^T\beta, \sigma^2 I) \\
&= \mathbb{E}(\beta) \text{ since: } (XX^T)^{-1}XX^T = I \\
&= \beta \\
&\mathrm{Cov}(\hat{\beta}) = \mathrm{Cov}((XX^T)^{-1}XY) \\
&= (XX^T)^{-1}X\mathrm{Cov}(Y)((XX^T)^{-1}X)^{-1} \text{ since } (XX^T)^{-1}X \text{is constant} \\
&= (XX^T)^{-1}X\sigma^2 I((XX^T)^{-1}X)^{-1} \\
&= (XX^T)^{-1}X\sigma^2 IX^T(XX^T)^{-1} \qquad (XX^T)^{-1}\text{is symmetric due to } (XX^T) \text{ is symmetric.} \\
&= \sigma^2(XX^T)^{-1}XX^T(XX^T)^{-1} \\
&= \sigma^2(XX^T)^{-1}
\end{aligned}$$

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 $\sum_{i=1}^{n} a_i^2 = a^T a$]

We are trying to maximize the likelihood:

$$-\frac{n}{2}log(\frac{1}{2\pi\sigma^2 I})-\frac{(Y-X^T\beta)^T(Y-X^T\beta)}{2\sigma^2 I}$$
$$=-(\frac{n}{2}log(\frac{1}{2\pi\sigma^2 I})+\frac{(Y-X^T\beta)^T(Y-X^T\beta)}{2\sigma^2 I})$$

Which is equivalent to minimize the below:

$$\frac{n}{2}log(\frac{1}{2\pi\sigma^2 I})+\frac{(Y-X^T\beta)^T(Y-X^T\beta)}{2\sigma^2 I}$$

The first term is constant, we just need to minimize the second term:

$$\frac{(Y-X^T\beta)^T(Y-X^T\beta)}{2\sigma^2 I}$$
$$=\frac{\sum_{i=1}^n(y_i-x_i\beta)^2}{2\sigma^2 I} \text{ By hint}$$

We just need to minimize the: $\sum_{i=1}^n(y_i-x_i\beta)^2$ Since denominator is also constant.

Therefore, maximizing the likelihood is equivalent to minimizing square error

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]

$$\begin{aligned}\sum_{i=1}^n(y_i-x_i\beta)^2 &= (Y-X^T\beta)^T(Y-X^T\beta)\\ &= Y^TY - Y^TX^T\beta - \beta^TXY + \beta^TXX^T\beta\\ &= Y^TY - 2\beta^TXY + \beta^TXX^T\beta\end{aligned}$$

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.

Negative likelihood is below:

$$\frac{n}{2}log(\frac{1}{2\pi\sigma^2 I}) + \frac{(Y - X^T\beta)^T(Y - X^T\beta)}{2\sigma^2 I}$$

Minimizing above is equivalent to minimizing below since the first term is constant.

$$\frac{(Y - X^T\beta)^T(Y - X^T\beta)}{2\sigma^2 I}$$

$$\propto (Y - X^T\beta)^T(Y - X^T\beta)$$

$$= Y^T Y - 2\beta^T XY + \beta^T XX^T\beta$$

Let's take derivative of it with respect to β

$$\frac{\partial}{\partial\beta} = -2XY + 2XX^T\beta = 0$$

$$XX^T\beta = XY$$

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

2.2 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 = prevfloat(20.0)*(1 .- rand(batch_size))
  y = target_f(x)
  return (vec(x)',y)
end
```

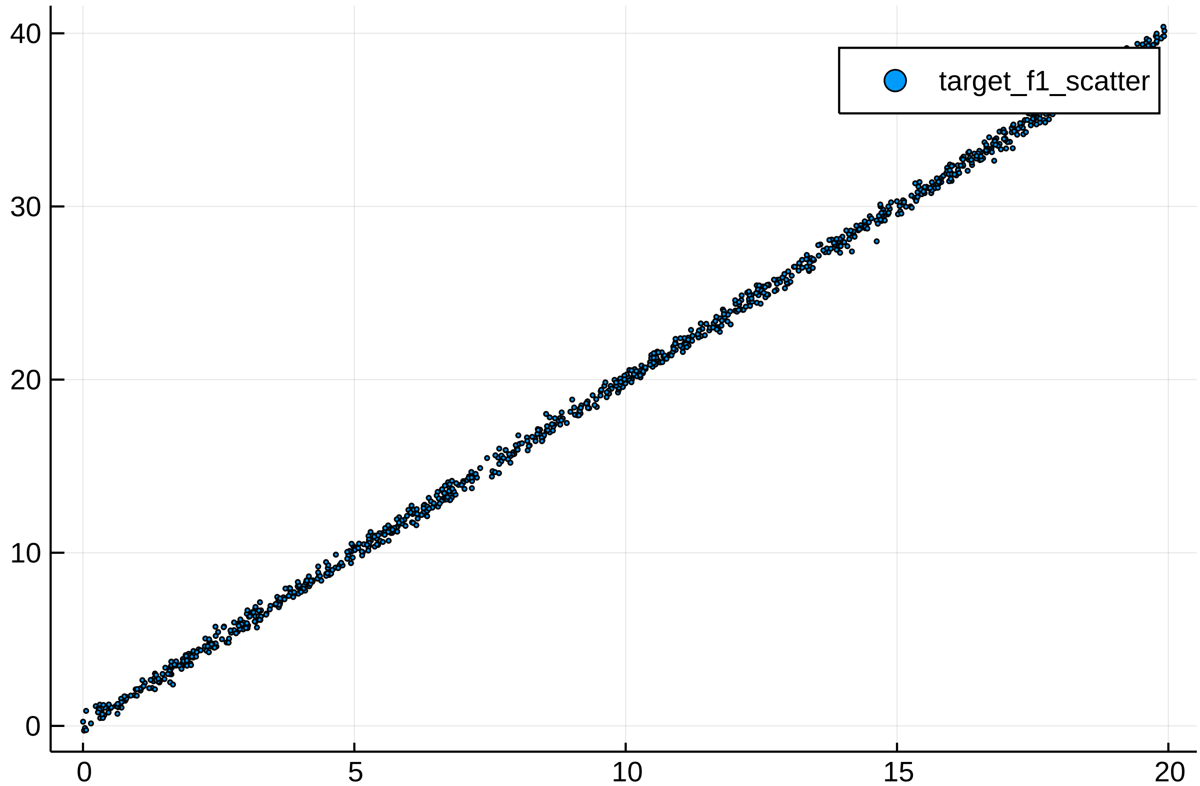
```
sample_batch (generic function with 1 method)
```

```
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
```

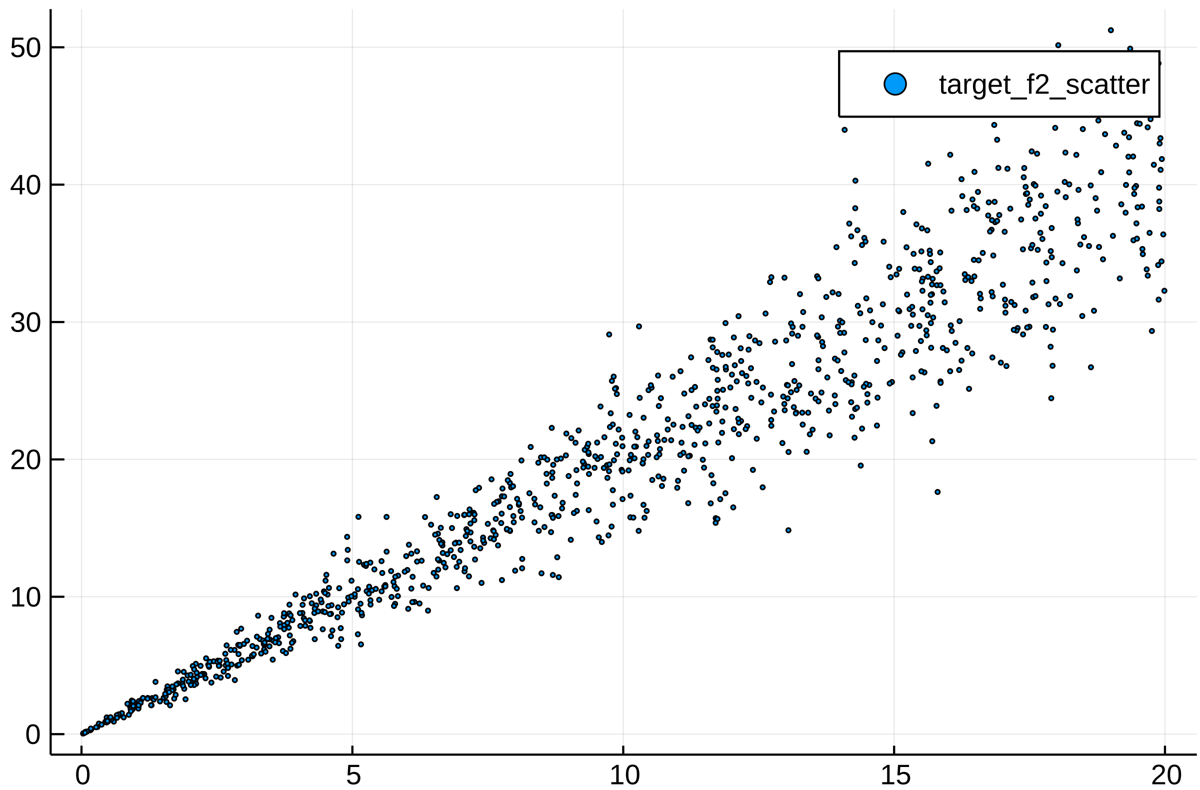
```
Test Summary:                  | Pass  Total
sample dimensions are correct |    6      6
Test.DefaultTestSet("sample dimensions are correct", Any[], 6, false)
```

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

x1,y1 = sample_batch(target_f1, 1000)
plot_f1 = scatter(vec(x1), y1, markersize = 2, label="target_f1_scatter")
display(plot_f1)
```

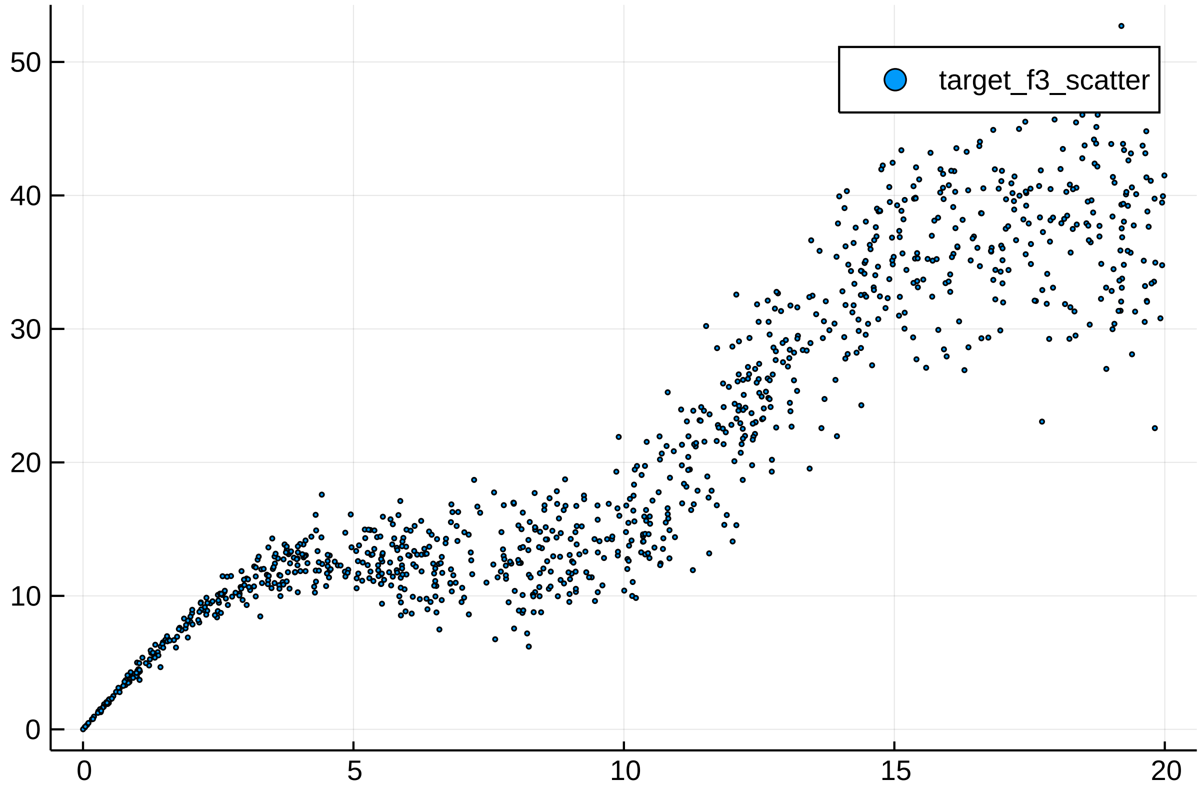



```
x2,y2 = sample_batch(target_f2, 1000)
plot_f2 = scatter(vec(x2), y2, markersize = 2, label="target_f2_scatter")
display(plot_f2)
```




```
x3,y3 = sample_batch(target_f3, 1000)
plot_f3 = scatter(vec(x3), y3, markersize = 2, label="target_f3_scatter")
```

```
display(plot_f3)
```



2.3 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

n=1000 # batch_size

x_1, y_1 = x1,y1
β_mle_1 = beta_mle(x_1, y_1)

x_2, y_2 = x2,y2
β_mle_2 = beta_mle(x_2, y_2)

x_3, y_3 = x3,y3
β_mle_3 = beta_mle(x_3, y_3)
```

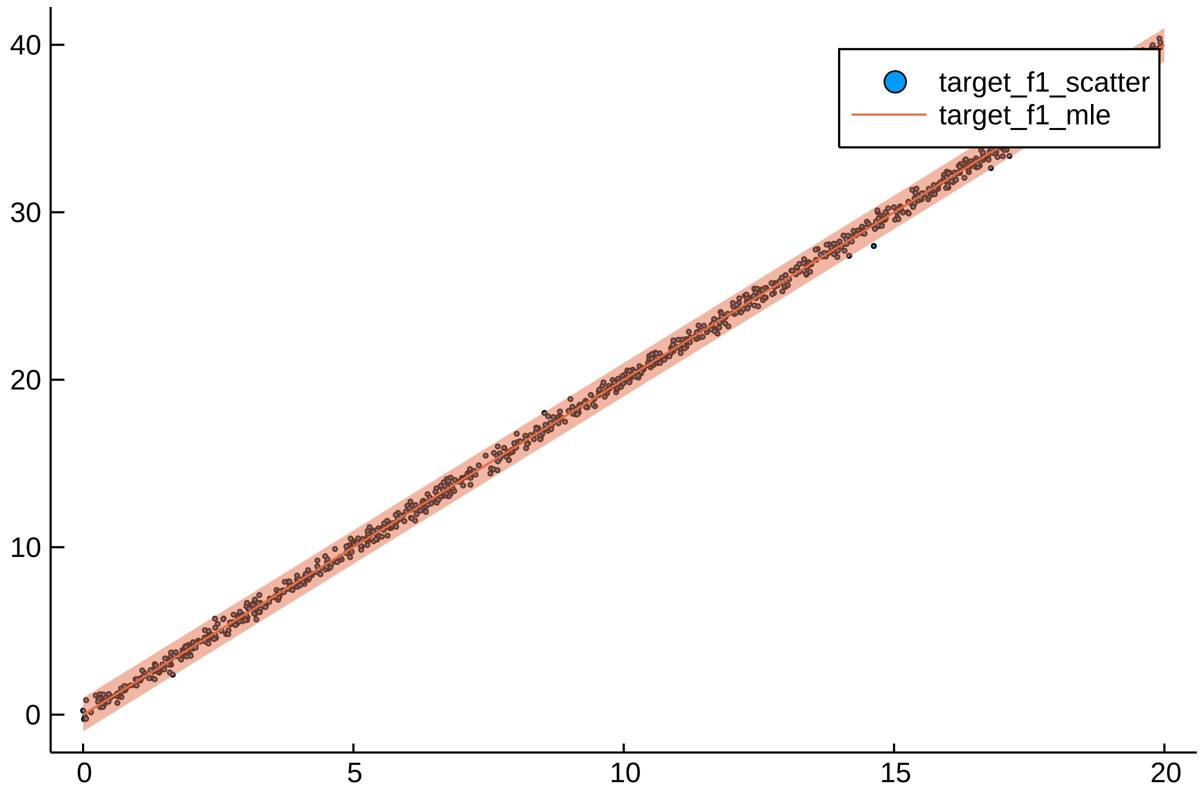
```
2.06520858148626
```

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

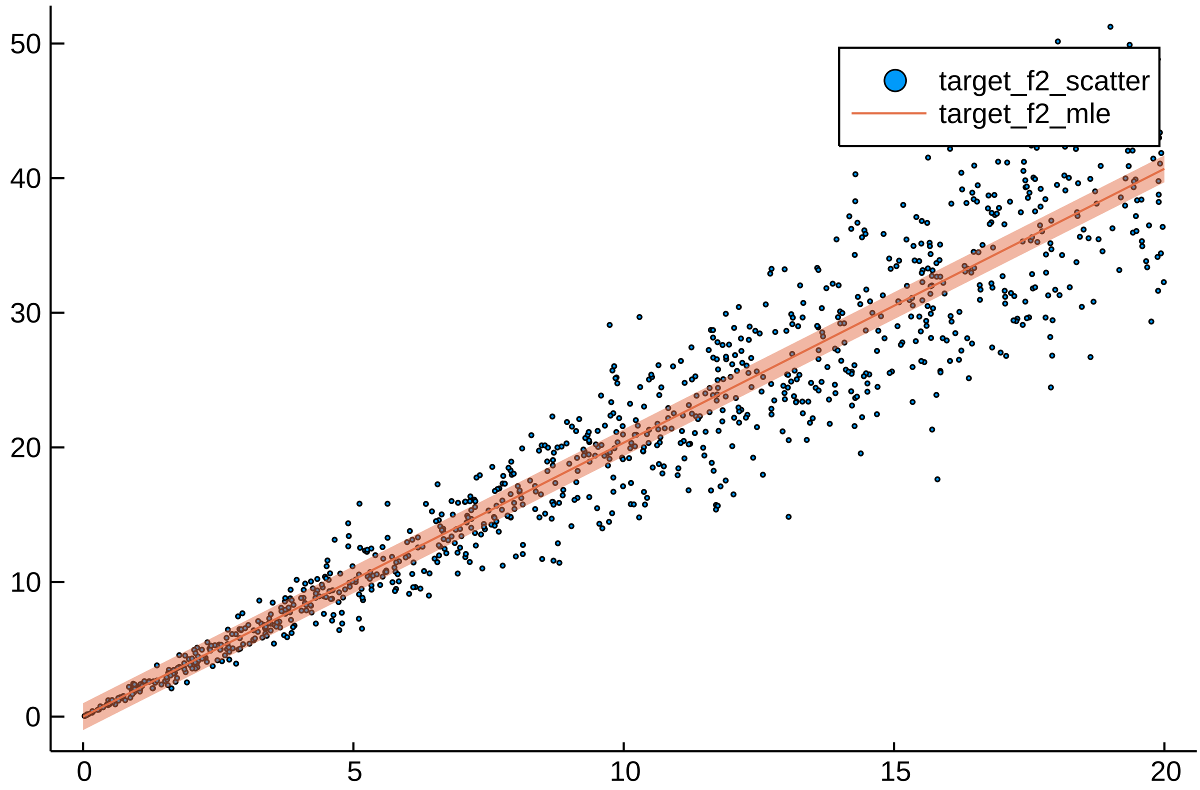
estimate, as well as a shaded region around the line corresponding to plus/minus one standard deviation (i.e. the fixed uncertainty $\sigma = 1.0$). Using `Plots.jl` this shaded uncertainty region can be achieved with the `ribbon` keyword argument. **Display 3 plots, one for each target function, showing samples of data and maximum likelihood estimate linear regression model**

```
abc = 0.00:0.01:20
```

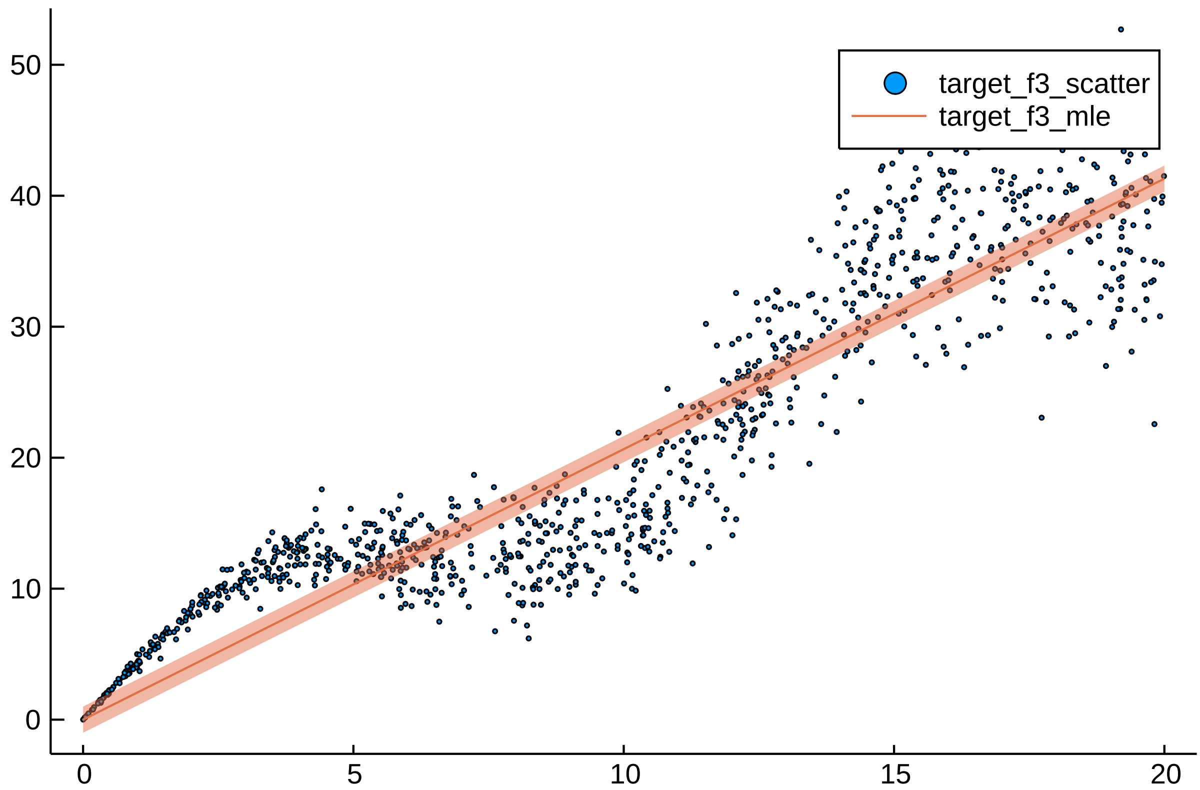
```
plot!(plot_f1, abc, β_mle_1'*abc, grid=false, ribbon=1.0, label="target_f1_mle")
```




```
plot!(plot_f2, abc, β_mle_2'*abc, grid=false, ribbon=1.0, label="target_f2_mle")
```




```
plot!(plot_f3, abc, β_mle_3'*abc, grid=false, ribbon=1.0, label="target_f3_mle")
```



2.4 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)
  """
  return (-1/2)*log.(2*π*(σ^2))+((-(x-μ)^2))/(2*(σ^2))#TODO: log-likelihood function
end
```

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

```
# Test Gaussian likelihood against standard implementation
@testset "Gaussian log likelihood" begin
# using Random
# Random.seed!(123)
using Distributions: logpdf, 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) ≈ logpdf.(Normal(μ,σ),x) # Correct 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-likelihoods
@test gaussian_log_likelihood.(μ,σ,x) ≈ logpdf.(Normal(μ,σ),x) # Correct 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-likelihoods
@test gaussian_log_likelihood.(μ,σ,x) ≈ logpdf.(Normal.(μ,σ),x) # Correct 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 -gaussian_log_likelihood.(x'* β, σ, y)#TODO: Negative Log Likelihood
  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: 1.427833810810382
--------  0.3  ------------
Negative Log-Likelihood: 4.16001375623686
--------  1.0  ------------
Negative Log-Likelihood: 9.77983358494867
--------  2.0  ------------
Negative Log-Likelihood: 16.267455290459147
--------  target_f2  ------------
--------  0.1  ------------
Negative Log-Likelihood: 7358.119733024863
--------  0.3  ------------
Negative Log-Likelihood: 493.1946299427762
--------  1.0  ------------
Negative Log-Likelihood: 19.51544670722729
--------  2.0  ------------
Negative Log-Likelihood: 31.00207673137666
--------  target_f3  ------------
--------  0.1  ------------
Negative Log-Likelihood: 16467.070135631275
--------  0.3  ------------
Negative Log-Likelihood: 378.57970774047146
--------  1.0  ------------
Negative Log-Likelihood: 119.10582417679771
--------  2.0  ------------
Negative Log-Likelihood: 48.352323131147934
--------  100  ------------
--------  target_f1  ------------
--------  0.1  ------------
Negative Log-Likelihood: 384.8046496758032
--------  0.3  ------------
Negative Log-Likelihood: 17.199935496874787
--------  1.0  ------------
Negative Log-Likelihood: 95.72045320681647
--------  2.0  ------------
Negative Log-Likelihood: 162.29924438977523
--------  target_f2  ------------
--------  0.1  ------------
Negative Log-Likelihood: 57673.183959873466
```

```
-------- 0.3 ------------
Negative Log-Likelihood: 4330.787956226285
-------- 1.0 ------------
Negative Log-Likelihood: 549.297719095662
-------- 2.0 ------------
Negative Log-Likelihood: 306.199898384406
-------- target_f3 ------------
-------- 0.1 ------------
Negative Log-Likelihood: 113996.11960713238
-------- 0.3 ------------
Negative Log-Likelihood: 12401.165893590585
-------- 1.0 ------------
Negative Log-Likelihood: 991.3340147239508
-------- 2.0 ------------
Negative Log-Likelihood: 426.09392640936267
-------- 1000 ------------
-------- target_f1 ------------
-------- 0.1 ------------
Negative Log-Likelihood: 3138.9454965022273
-------- 0.3 ------------
Negative Log-Likelihood: 190.98260504624147
-------- 1.0 ------------
Negative Log-Likelihood: 964.8679061608393
-------- 2.0 ------------
Negative Log-Likelihood: 1622.5392937082825
-------- target_f2 ------------
-------- 0.1 ------------
Negative Log-Likelihood: 598083.0693896158
-------- 0.3 ------------
Negative Log-Likelihood: 64708.632037624266
-------- 1.0 ------------
Negative Log-Likelihood: 6495.662034289212
-------- 2.0 ------------
Negative Log-Likelihood: 3268.6396949861005
-------- target_f3 ------------
-------- 0.1 ------------
Negative Log-Likelihood: 1.20266624470533e6
-------- 0.3 ------------
Negative Log-Likelihood: 128516.54849918313
-------- 1.0 ------------
Negative Log-Likelihood: 12114.148598492375
-------- 2.0 ------------
Negative Log-Likelihood: 4600.1141595626395
```

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

In target function 1, best σ is 0.3. In target function 2 and 3, best σ is 2.0.

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.

2.5 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

@testset "Gradients wrt parameter" begin
using Random
Random.seed!(123)
β_test = randn()
σ_test = rand()
x,y = sample_batch(target_f1,100)
f(β) = lr_model_nll(β,x,y;σ=σ_test)
ad_grad = gradient(f, β_test)
lr_model_nll(β_test,x,y;σ=σ_test)
hand_derivative = (-x*(y-x'*β_test)/(σ_test^2))[1]
@test ad_grad[1] ≈ hand_derivative
end

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

2.5.1 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.

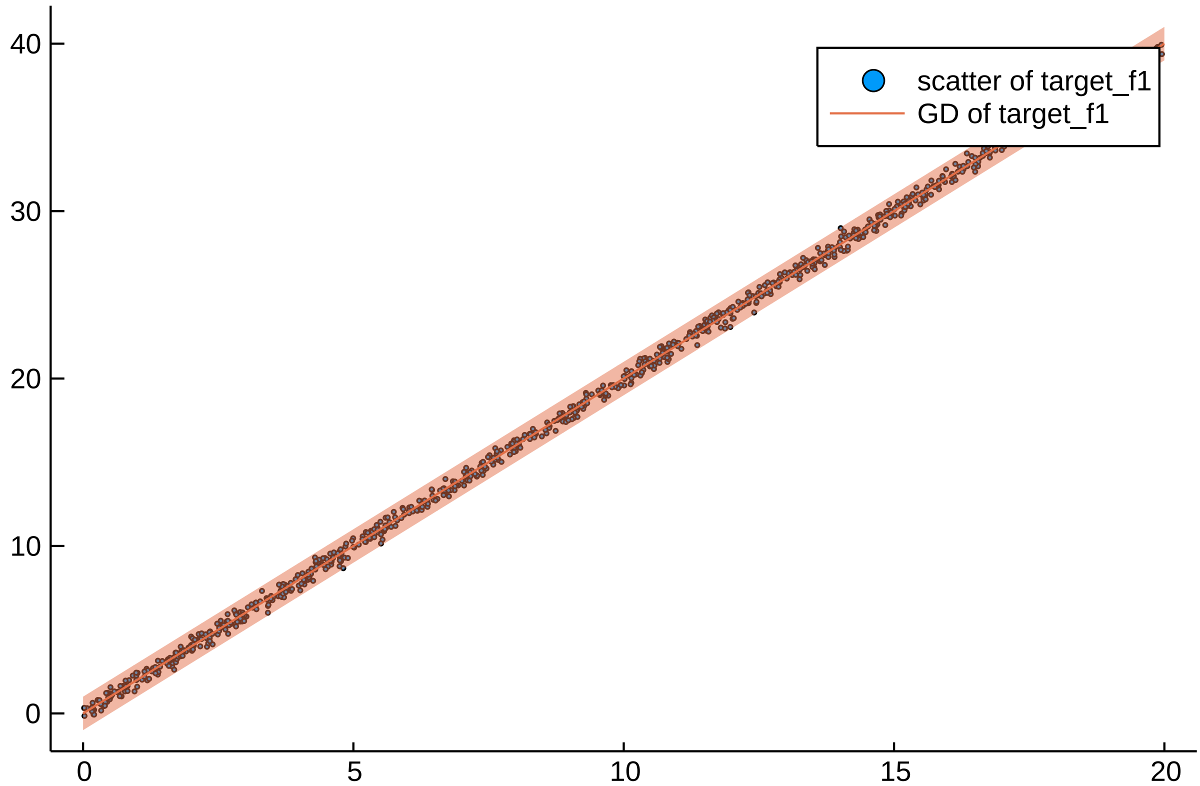
```
using Logging # Print training progress to REPL, not pdf

function train_lin_reg(target_f, β_init; bs= 100, lr = 1e-6, iters=1000, σ_model = 1. )
    β_curr = β_init
    for i in 1:iters
      x,y = sample_batch(target_f,bs)#TODO
      @info "loss: $(lr_model_nll(β_curr,x,y;σ=σ_model))  β: $β_curr" #TODO: log loss,
if you want to monitor training progress
```

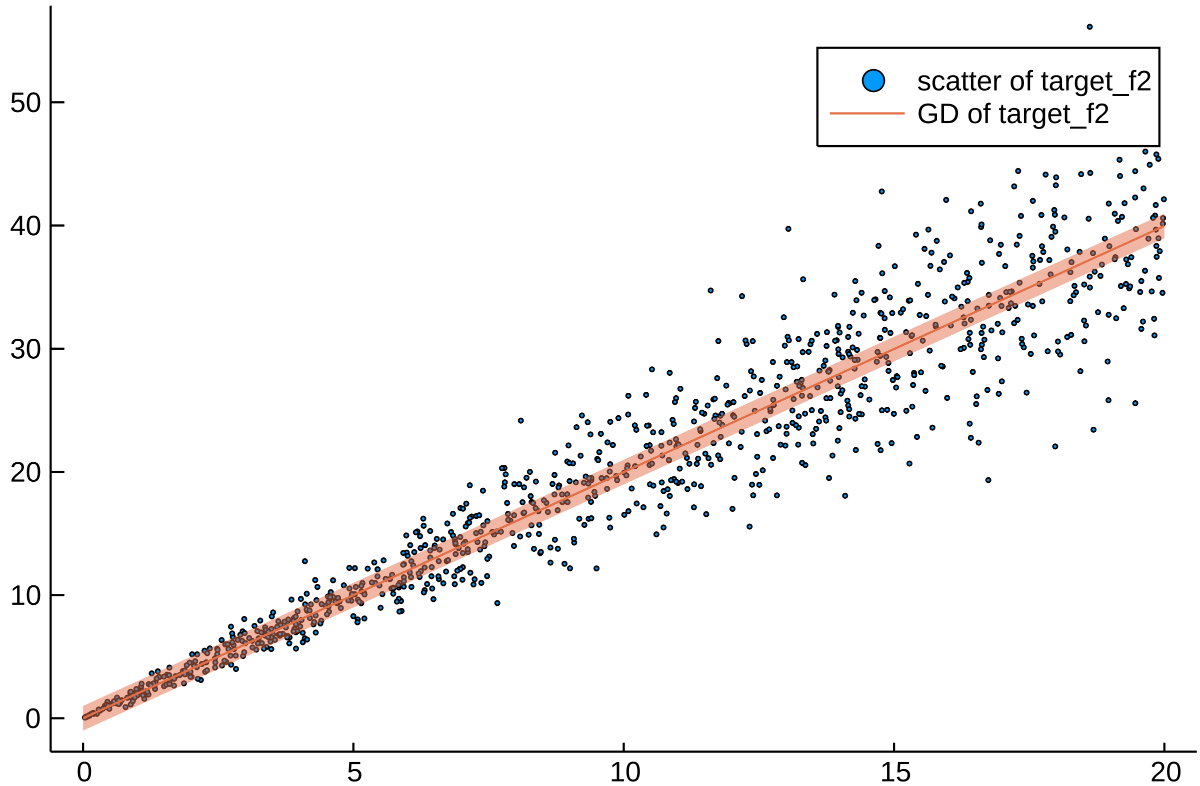
```
        grad_β = (-x*(y-x'*β_curr)/(σ_model^2))[1]#TODO: compute gradients
        β_curr = β_curr-lr*grad_β#TODO: gradient descent
    end
    return β_curr
end
```

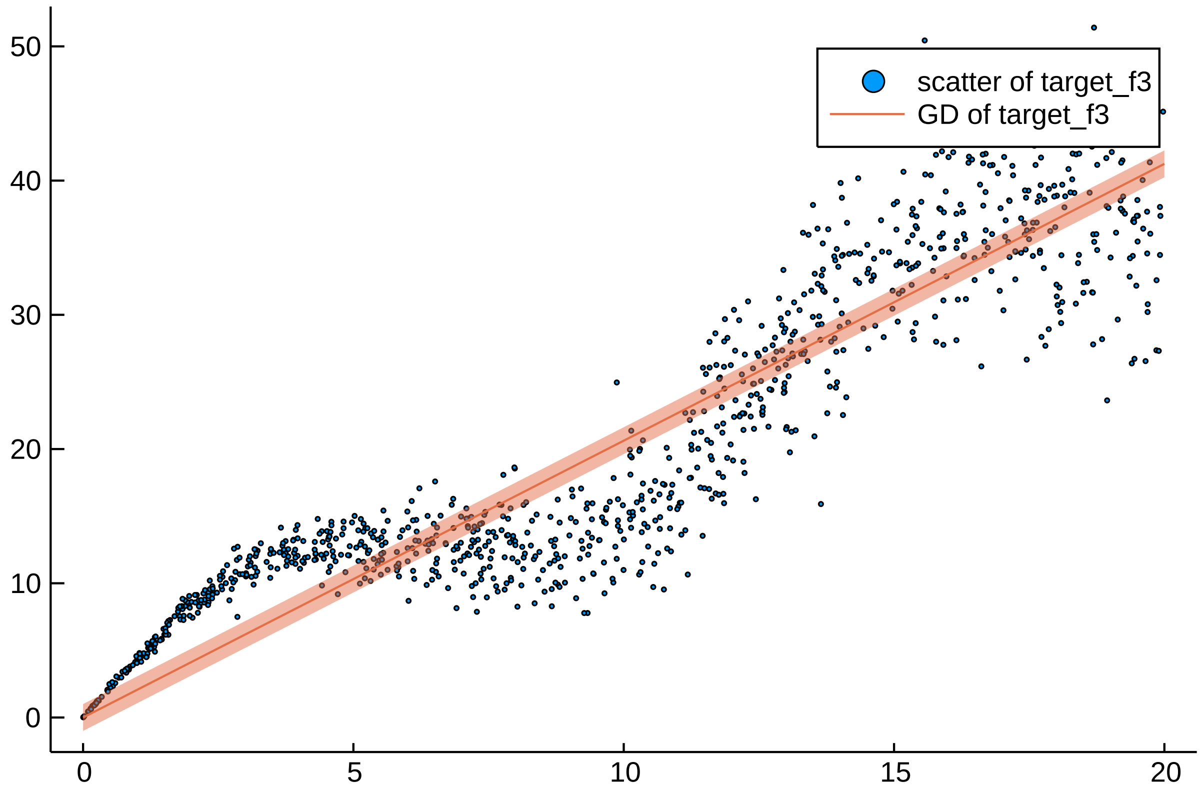
```
train_lin_reg (generic function with 1 method)
```

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.

```
β_init = 1000 * randn() # Initial parameter
for target_f in (target_f1, target_f2, target_f3)
    x,y = sample_batch(target_f,n)
    β_learned= train_lin_reg(target_f, β_init; bs= 1000, lr = 1e-6, iters=1000, σ_model
= 1.)#TODO: call training function
    plot_f = scatter(vec(x), y, markersize = 2, label="scatter of $target_f")
    display(plot!(abc, β_learned'*abc, grid=false, ribbon=1.0, label="GD of $target_f"))
end
```








```
#TODO: For each target function, plot data samples and learned regression
```

2.5.2 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.

```
function neural_net(x,θ)

  return vec(((θ[3])'*tanh.((θ[1])'*x.+θ[2]).+θ[4])') #TODO
end

# Random initial Parameters

θ = (randn(1,10), randn(10,1),randn(10,1), randn(1,1)) #TODO

@testset "neural net mean vector output" begin
n = 100
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 $\sigma = 1.0$

```
function nn_model_nll(θ,x,y;σ=1)
  return sum(-gaussian_log_likelihood.(neural_net(x,θ), σ, y)) #TODO
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

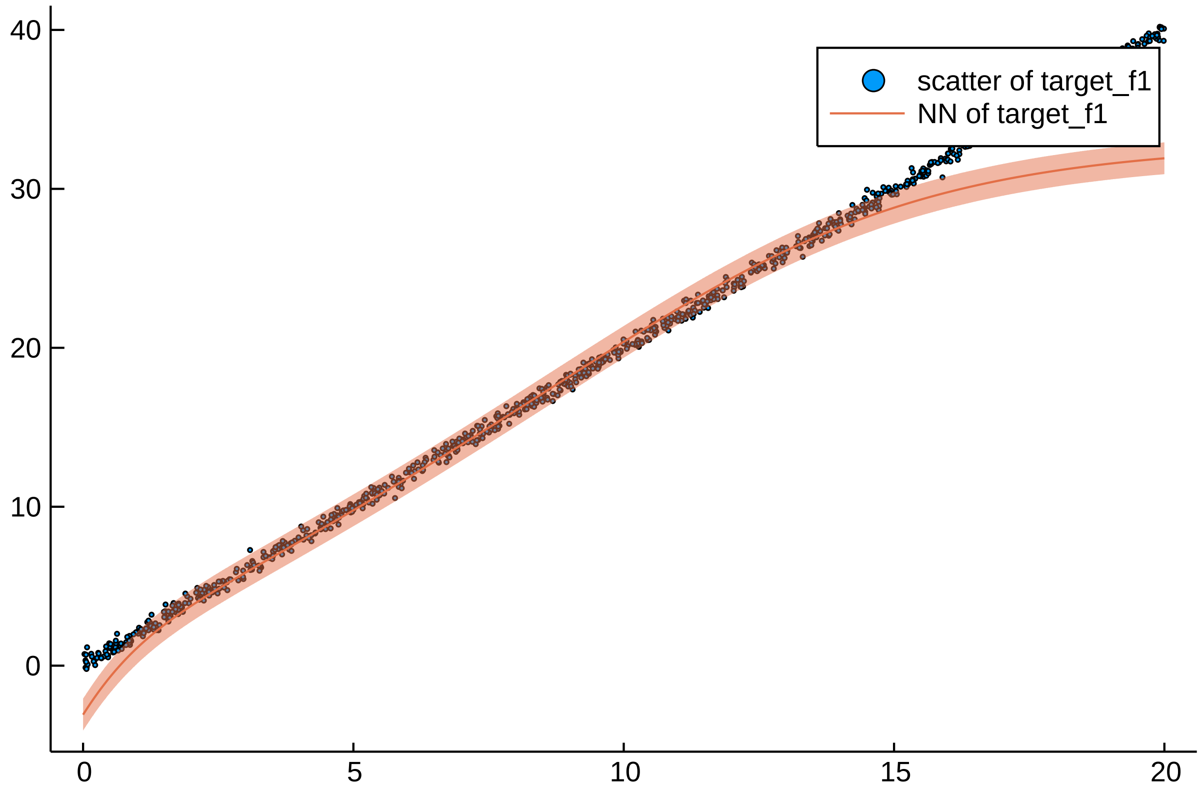
and, after all iterations, returns the final estimate of θ.

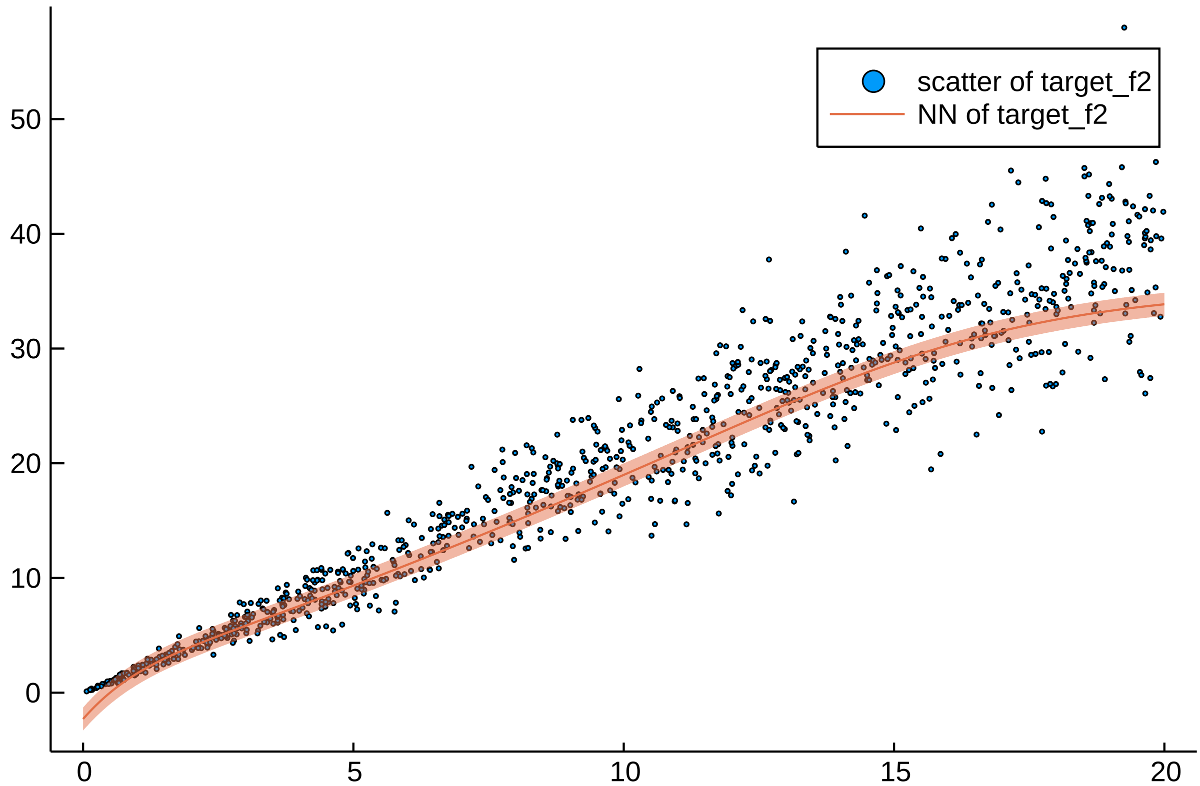
```
using Logging # Print training progress to REPL, not pdf

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) #TODO
      @info "loss: $(nn_model_nll(θ_curr,x,y;σ=σ_model))" #TODO: log loss, if you want
to montior training
      grad_θ = gradient(θ_curr -> nn_model_nll(θ_curr,x,y;σ=σ_model), θ_curr)
      # grad_θ = (-gradient(f,θ_curr)*(y-neural_net(x,θ_curr))/(σ_model^2)) #TODO:
compute gradients
      θ_curr[1] .= θ_curr[1]-lr*grad_θ[1][1] #TODO: gradient descent
      θ_curr[2] .= θ_curr[2]-lr*grad_θ[1][2]
      θ_curr[3] .= θ_curr[3]-lr*grad_θ[1][3]
      θ_curr[4] .= θ_curr[4]-lr*grad_θ[1][4]
    end
    return θ_curr
end
```

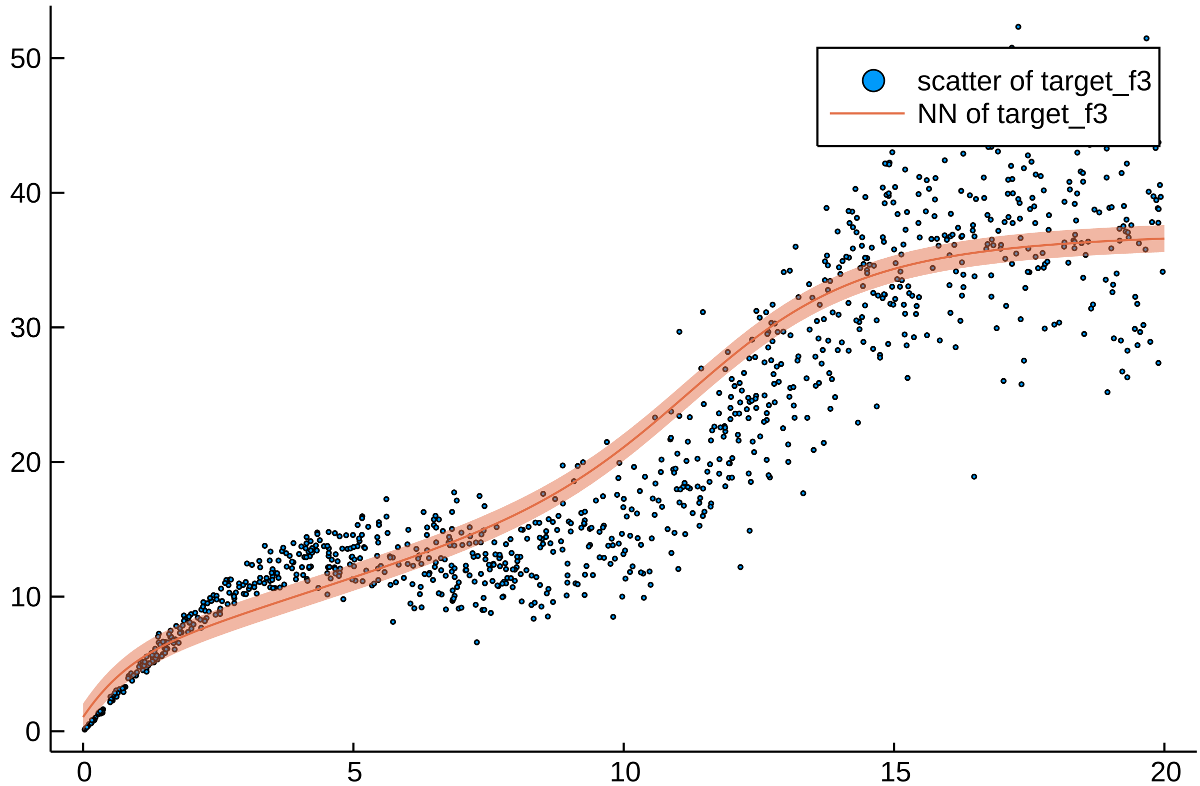
```
train_nn_reg (generic function with 1 method)
```

4. [2pts] For each target function, start with an initialization of the network parameters, θ, use your train function to minimize the negative log-likelihood and find an estimate for θ_{learned} by gradient descent. Then plot a $n = 1000$ sample of the data and the learned regression model with shaded uncertainty bounds given by $\sigma = 1.0$

```
#TODO: For each target function
n = 1000
abc = 0.00:0.01:20
θ_init = (rand(1,10), rand(10,1),rand(10,1), rand(1,1))  #TODO

for target_f in (target_f1, target_f2, target_f3)
    x,y = sample_batch(target_f,n)
    θ_learned = train_nn_reg(target_f, θ_init; bs= 100, lr = 1e-5, iters=1000, σ_model =
1. ) #TODO
    plot_f = scatter(vec(x), y, markersize = 2, label="scatter of $target_f")
    display(plot!(plot_f, abc, neural_net(abc', θ_learned), grid=false, ribbon=1.0,
label="NN of $target_f"))
end
```


scatter of target_f1
NN of target_f1
40
30
20
10
0
0
5
10
15
20


scatter of target_f2
NN of target_f2
50
40
30
20
10
0
0
5
10
15
20




```
#TODO: plot data samples and learned regression
```

2.5.3 Non-linear Regression and Input-dependent Variance with a Neural Network [8pts]

In the previous questions we've gone from a gaussian model with mean given by linear combination

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

to gaussian model with mean given by non-linear function of the data (neural network)

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

However, in all cases we have considered so far, we specify a fixed variance for our model distribution. We know that two of our target datasets have heteroscedastic noise, meaning any fixed choice of variance will poorly model the data.

In this question we will use a neural network to learn both the mean and log-variance of our gaussian model.

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

1. [1pts] Write the code for a fully-connected neural network (multi-layer perceptron) with one 10-dimensional hidden layer and a `tanh` nonlinearirty, and outputs both a vector for mean and $\log\sigma$. Test the output shape is as expected.

```julia
function neural_net_w_var(x,θ)
  μ = ((θ[3])'*tanh.((θ[1])'*x.+θ[2]).+θ[4])[1,:]
  logσ = ((θ[3])'*tanh.((θ[1])'*x.+θ[2]).+θ[4])[2,:]
  return μ, logσ #TODO
end

# Random initial Parameters
n=100
x,y = sample_batch(target_f1,n)
θ = (randn(1,10), randn(10,1),randn(10,2), randn(2,1)) #TODO

@testset "neural net mean and logsigma vector output" begin
n = 100
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] Write the code that computes the negative log-likelihood for this model where the mean and $\log \sigma$ is given by the output of the neural network. (Hint: Don't forget to take $\exp \log \sigma$)

```julia
function nn_with_var_model_nll(θ,x,y)
  μ, logσ = neural_net_w_var(x,θ)
  return sum(-gaussian_log_likelihood.(μ, exp.(logσ), y)) #TODO
end
```

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

3. [1pts] Write a function `train_nn_w_var_reg` that accepts a target function and an initial estimate for θ and some hyperparameters for batch-size, 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.

```julia
function train_nn_w_var_reg(target_f, θ_init; bs= 100, lr = 1e-4, iters=10000)
    θ_curr = θ_init
    for i in 1:iters
      x,y = sample_batch(target_f, bs) #TODO
      @info "loss: $(nn_with_var_model_nll(θ_curr,x,y))" #TODO: log loss
      grad_θ = grad_θ = gradient(θ_curr -> nn_with_var_model_nll(θ_curr,x,y), θ_curr)
#TODO compute gradients
      θ_curr[1] .= θ_curr[1]-lr*grad_θ[1][1] #TODO: gradient descent
      θ_curr[2] .= θ_curr[2]-lr*grad_θ[1][2]
```

```
        θ_curr[3] .= θ_curr[3]-lr*grad_θ[1][3]
        θ_curr[4] .= θ_curr[4]-lr*grad_θ[1][4]
    end
    return θ_curr
end
```

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

4. [4pts] For each target function, start with an initialization of the network parameters, θ, learn an estimate for θ_{learned} by gradient descent. Then plot a $n = 1000$ sample of the dataset and the learned regression model with shaded uncertainty bounds corresponding to plus/minus one standard deviation given by the variance of the predictive distribution at each input location (output by the neural network). (Hint: `ribbon` argument for shaded uncertainty bounds can accept a vector of σ)

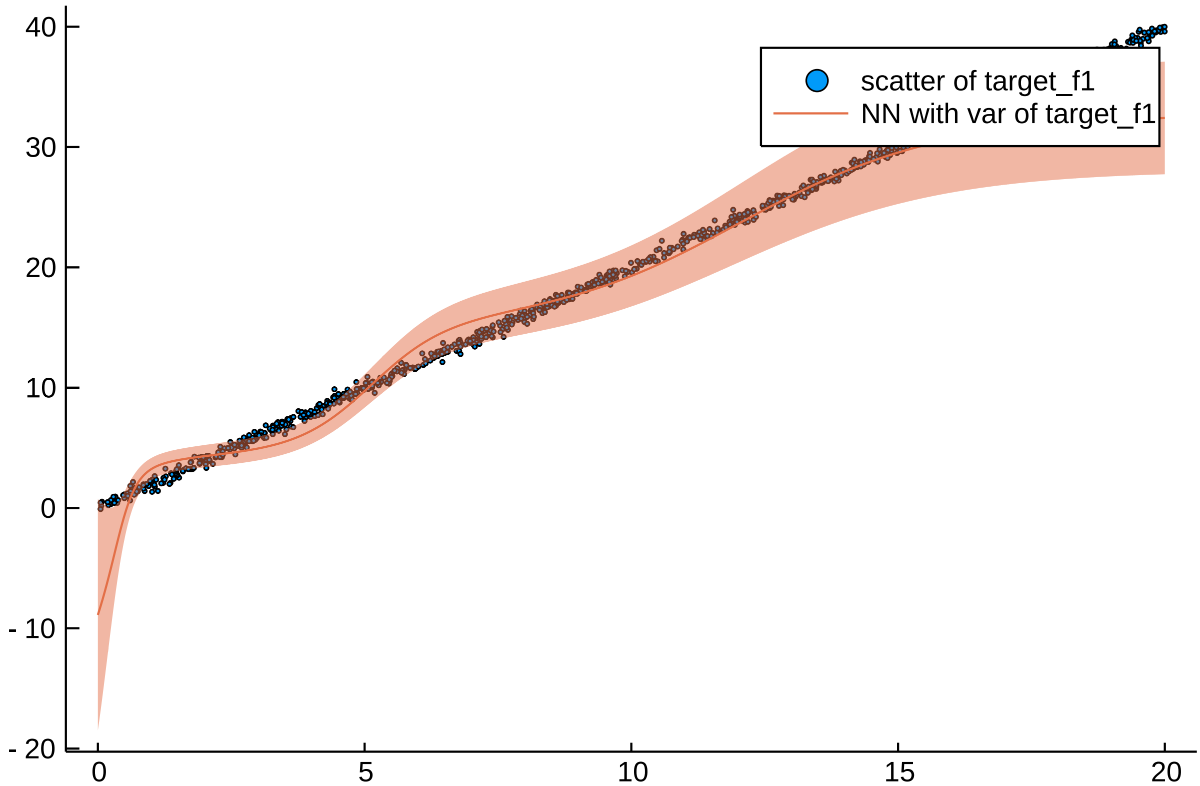
   Note: Learning the variance is tricky, and this may be unstable during training. There are some things you can try:

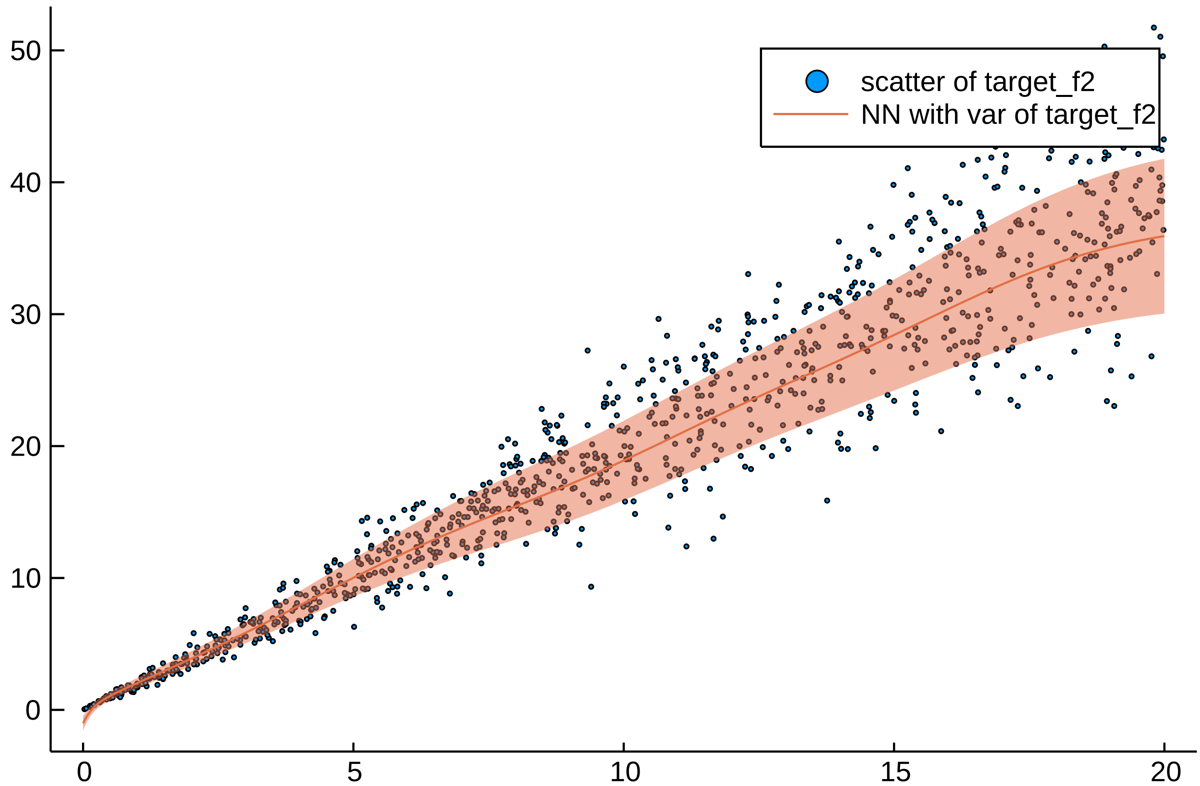
   - Adjusting the hyperparameters like learning rate and batch size
   - Train for more iterations
   - Try a different random initialization, like sample random weights and bias matrices with lower variance.

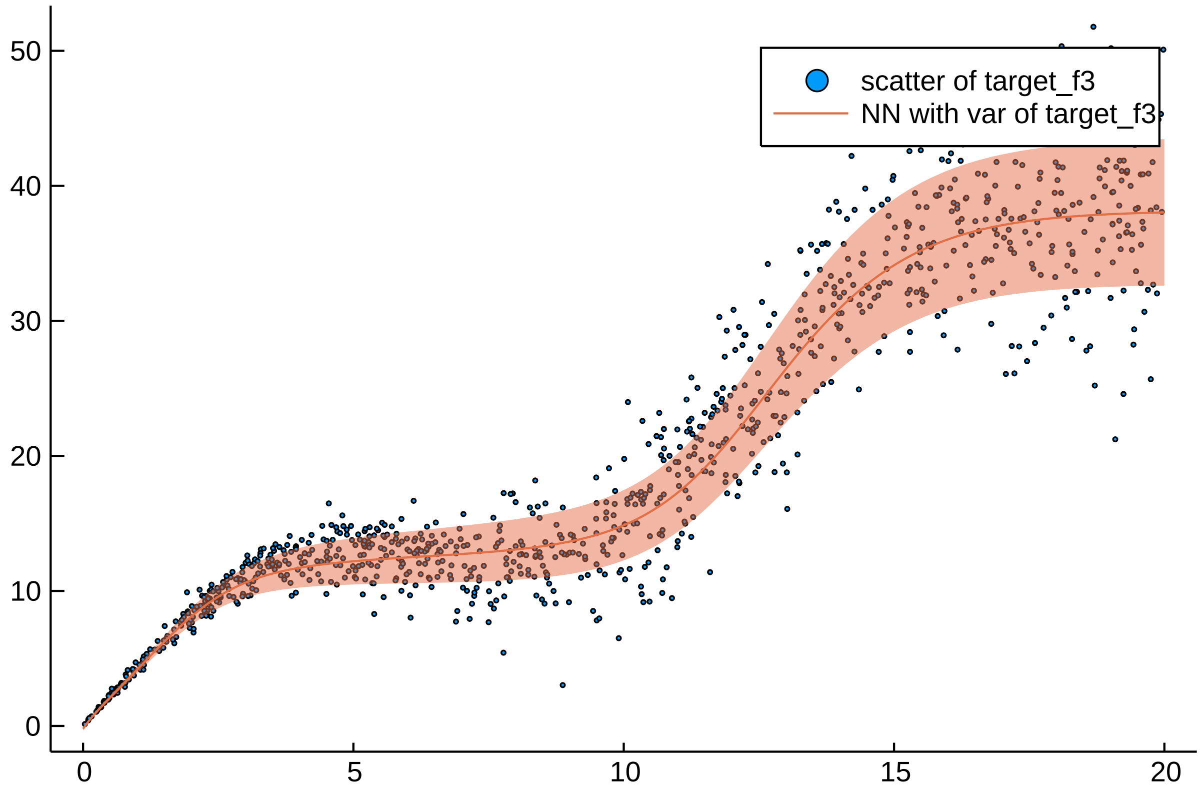
   For this question **you will not be assessed on the final quality of your model**. Specifically, if you fails to train an optimal model for the data that is okay. You are expected to learn something that is somewhat reasonable, and **demonstrates that this model is training and learning variance**.

   If your implementation is correct, it is possible to learn a reasonable model with fewer than 10 minutes of training on a laptop CPU. The default hyperparameters should help, but may need some tuning.

```
#TODO: For each target function
n=1000
θ_init =  (rand(1,10), rand(10,1),rand(10,2), rand(2,1)) #TODO
for target_f in (target_f1, target_f2, target_f3)
    x,y = sample_batch(target_f,n)
    θ_learned = train_nn_w_var_reg(target_f, θ_init; bs= 128, lr = 1e-4, iters=10000)
#TODO
    plot_f = scatter(vec(x), y, markersize = 2, label="scatter of $target_f")
    μ, logσ = neural_net_w_var(abc', θ_learned)
    display(plot!(plot_f, abc, μ, grid=false, ribbon=exp.(logσ), label="NN with var of
$target_f"))
end
```


scatter of target_f1
NN with var of target_f1
40
30
20
10
0
- 10
- 20
0
5
10
15
20


scatter of target_f2
NN with var of target_f2
50
40
30
20
10
0
0
5
10
15
20




```
#TODO: plot data samples and learned regression
```

If you would like to take the time to train a very good model of the data (specifically for target functions 2 and 3) with a neural network that outputs both mean and $\log \sigma$ you can do this, but it is not necessary to achieve full marks. You can try

- Using a more stable optimizer, like Adam. You may import this from a library.
- Increasing the expressivity of the neural network, increase the number of layers or the dimensionality of the hidden layer.
- Careful tuning of hyperparameters, like learning rate and batchsize.