STA2104: Assignment 1

Justin Leo| 1006376459

24 February 2020

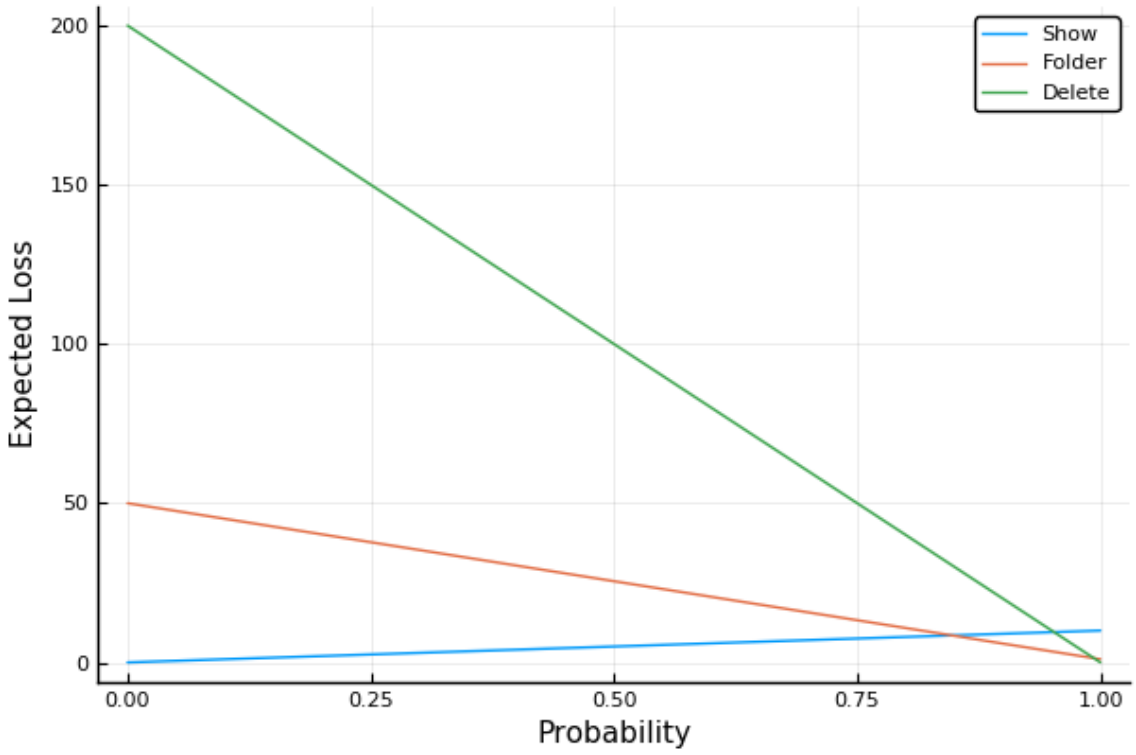
1. Decision Theory

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(spam|email)

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

num_actions = length(losses)

Show = losses[1]
Folder = losses[2]
Delete = losses[3]

function expected_loss_of_action(p_spam, action)
#TODO: Return expected loss over a Bernoulli random variable
# with mean prob spam.
# Losses are given by the table above.
    if action == 1
        time = Show
    elseif action == 2
        time = Folder
    else
        time = Delete

    end



    list = []
    for i in p_spam
        value = i.*(time[1]) .+ (1-i).*(time[2])
        #print(value)
        append!(list,value)
    end
```

```
    return list
end

prob_range = range(0., stop=1., length=500)



p1 = collect(prob_range)

# Make plot


plot(xaxis = ("Probability"), yaxis = ("Expected Loss"))
plot()
using Plots
for action in 1:num_actions
    names = ("Show", "Folder", "Delete")
    display(plot!(p1, expected_loss_of_action((p1), action),label = (names[action]),
    xaxis = ("Probability"), yaxis = ("Expected Loss")))

end
```



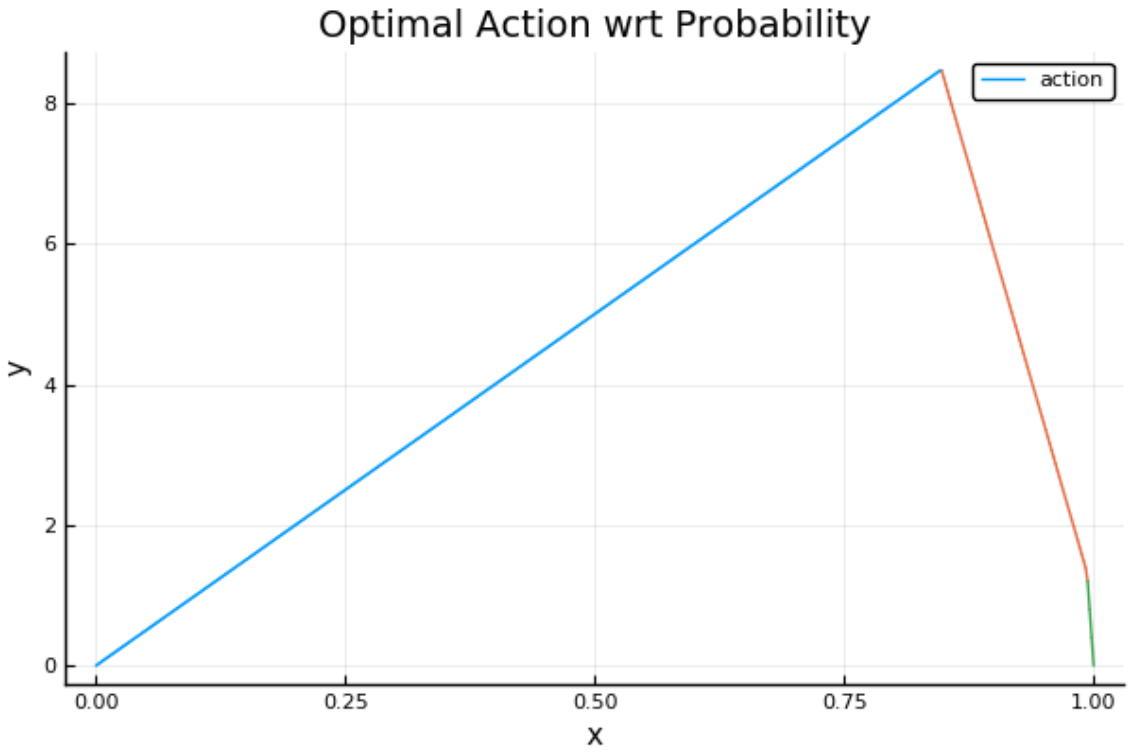
2. [2pts] Write a function that computes the optimal action given the probability of spam.

```
function optimal_action(prob_spam)
  list = []
  for i in 1:num_actions
    value = expected_loss_of_action(prob_spam,i)
    append!(list, value)
  end
```

```
  min = findmin(list)[2]
  #actions =  ["Show","Folder","Delete"]
  return min
end
```

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
    v = optimal_action(p)
    append!(optimal_actions,v)
    append!(optimal_losses,expected_loss_of_action(p,v))

end
plot(prob_range, optimal_losses, linecolor=optimal_actions, title = "optimal action wrt probability",
xlabel = "x", ylabel = "y")
```



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.

We want to find the probability for which the expected loss of "Delete" is less than the expected loss of "Folder":

$$p(1)+(1-p)50 > p(0)+(1-p)200 \Rightarrow p+50-50p > 200-200p \Rightarrow 151p > 150 \Rightarrow p > \frac{150}{151} \Rightarrow p > 0.993 \quad (1)$$

2. Regression

2.1 Manually Derived Linear Regression

$$\hat{\beta} = (XX^T)^{-1}XY \tag{2}$$

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

   If m>n, then the system of equations will be under-identified since observations are less than the number of variables to solve, and hence there exists no unique solution to $\beta_{m\times 1}$

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

   The expected value of $\hat{\beta}$ is given by

$$E(\hat{\beta}) = E[(XX^T)^{-1}XY] = E[(XX^T)^{-1}X(X^T\beta)] = (XX^T)^{-1}XX^T\beta = (X^T)^{-1}X^{-1}XX^T\beta = \boldsymbol{I}\times\beta = \beta$$

   The variance of $\hat{\beta}$ is given by

$$\sigma^2_{\hat{\beta}} = Var(\hat{\beta}) = Var[(XX^T)^{-1}XY] = E(\hat{\beta}^2) - E(\hat{\beta})^2 = \sigma^2(X^TX)^{-1}$$

   Note that:

$$\hat{\beta_1} = (XX^T)^{-1}XY_1, \hat{\beta_2} = (XX^T)^{-1}XY_2$$

$$\begin{aligned} Cov(\hat{\beta_1}, \hat{\beta_2}) &= Cov((X^TX)^{-1}X^TY_1(X^TX)^{-1}X^TY_2) \\ &= X^TX^{-1}X^TCov(Y_1, Y_2)X(X^TX)^{-1} \\ &= X^TX^{-1}\sum_{12}X(X^TX)^{-1} \end{aligned}$$

   A similar calculation shows $Cov(\hat{\beta_1}, \hat{\beta_1}) = Var(\hat{\beta_1})$

   The covariance matrix of β is then given by:

$$\sigma^2_{\beta}\times\boldsymbol{I}_{n\times n} + ((\boldsymbol{J}_{n\times n} - \boldsymbol{I}_{n\times n})\times Cov(\beta)) = \sigma^2_{\beta}\times\boldsymbol{J}_{n\times n}$$

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

$$\hat{\beta}_{OLS} = argmin_{\beta\in\theta}\sum_{i=1}^{n}\epsilon_i^2$$

$$\hat{\beta}_{OLS} = argmin_{\beta\in\theta}\sum_{i=1}^{n}\epsilon_i^T\epsilon_i$$

$$\hat{\beta}_{OLS} = argmin_{\beta\in\theta}\sum_{i=1}^{n}(y_i - x_i^T\beta)^T(y_i - x_i^T\beta)$$

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

$$argmin_\beta \sum_{i=1}^{n}(y_i - x_i\beta)^2$$
$$argmin_\beta (y - x^T\beta)^T(y - x^T\beta)$$
$$argmin_\beta y^T y - y^T x^T\beta - \beta^T xy + \beta^T xx^T\beta$$
$$argmin_\beta y^T y - 2y^T x\beta + \beta^T x^T x\beta$$

Note that $\beta^T x_i y_i$ is a scalar so it is the same as its transpose, $y_i^T x_i^T \beta$. Taking the derivative of this expression with respect to β and setting it equal to zero, we have the following first order condition:

$$2y_i^T x_i^T = 2\beta^T x_i x_i^T$$

Now we can derive the optimal beta OLS by solving for β:

$$\beta^T = y_i^T x_i^T (x_i^T x_i)^{-1}$$
$$\hat{\beta}_{OLS} = (x_i^T x_i)^{-1} x_i y_i$$

This beta OLS estimate is what we wanted to obtain given the format of equation (1) given in the question.

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.

For maximum likelihood estimator:

$$L(y_{i\,i=1}^n \mid \beta_l, \sigma^2) = (\frac{1}{\sqrt{2\pi\sigma^2}})^n exp(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - x_i\beta)^2) = (\frac{1}{\sqrt{2\pi\sigma^2}})^n exp(-\frac{1}{2\sigma^2}(y_i - x_i^T\beta)^T(y_i - x_i^T\beta))$$

$$l(y_{i\,i=1}^n \mid \beta_l, \sigma^2) = nlog(\frac{1}{\sqrt{2\pi\sigma^2}})[-\frac{1}{2\sigma^2}(y_i - x_i^T\beta)^T(y_i - x_i^T\beta)]$$

Since neither $nlog(\frac{1}{\sqrt{2\pi\sigma^2}})$ nor $-\frac{1}{2\sigma^2}$ are functions of beta, they will be constants when we derive with respect to β. So to make things easier, we will represent $nlog(\frac{1}{\sqrt{2\pi\sigma^2}}) \times -\frac{1}{2\sigma^2}$ as k in our likelihood expression.

$$l(\cdot) = k(y_i - x_i^T\beta)^T(y_i - x_i^T\beta)$$

Hence we have that,

$$\hat{\beta}_{MLE} = argmax_{\beta\in\theta}\sum_{i=1}^{n} l(y_{i\,i=1}^n \mid \beta_l, \sigma^2) = argmax_{\beta\in\theta}\sum_{i=1}^{n} k(y_i - x_i^T\beta)^T(y_i - x_i^T\beta)$$

Note that $\frac{\partial l(\cdot)}{\partial\theta} = 0$ is interchangeable with $\frac{\partial\epsilon^T\epsilon}{\partial\beta}$

Therefore, $\hat{\beta}_{MLE} = \hat{\beta}_{OLS}$

$$\begin{aligned} l(\cdot) &= k(y_i - x_i^T\beta)^T(y_i - x_i^T\beta) \\ &= k(y_i^T y_i - y_i^T x_i^T\beta - \beta^T x_i y_i + \beta^T x_i^T x_i\beta) \end{aligned}$$

Take derivative of l(·) with respect to β

$$2ky^T y - 2k\beta^T xx^T = 0$$

Finally, using a similar process to solve for beta as in part 4, we arrive at the previously proven result, as desired.

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

2.2 Toy Data

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:

linear trend with constant noise.

linear trend with heteroskedastic noise.

non-linear trend with heteroskedastic noise.

```
using LinearAlgebra

function target_f1(x, σ_true=0.3)[escapeinside=,mathescape=true]{julia}
  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
```

‘‘‘
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 = rand(Uniform(0,20),batch_size)'
  y = target_f(x)
  return x,y
end

@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,)
```

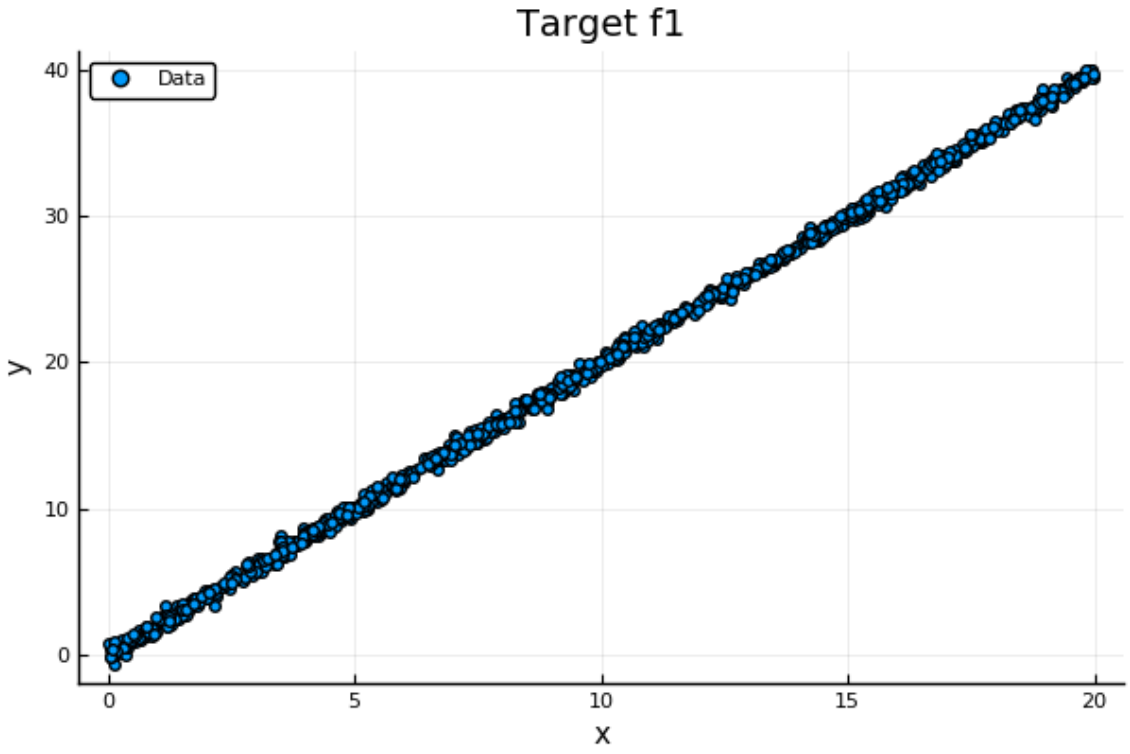
```
  end
end
```

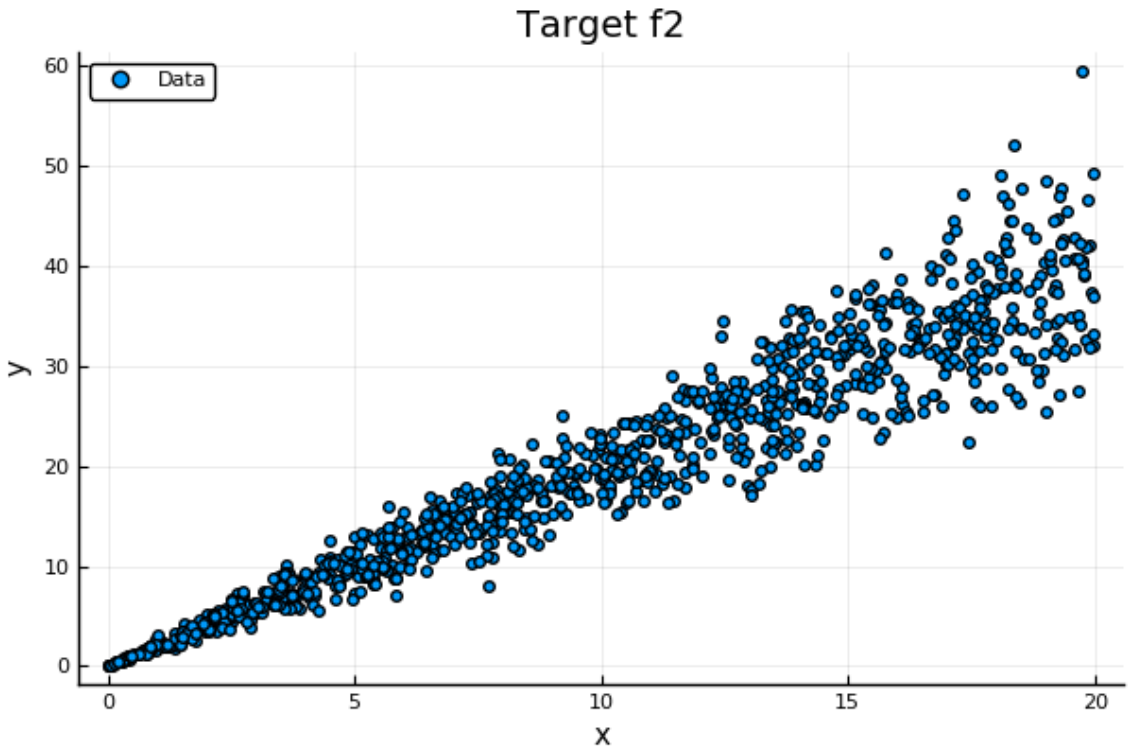
```
Test Summary:                   Pass  Total
sample dimensions are correct      6      6
```

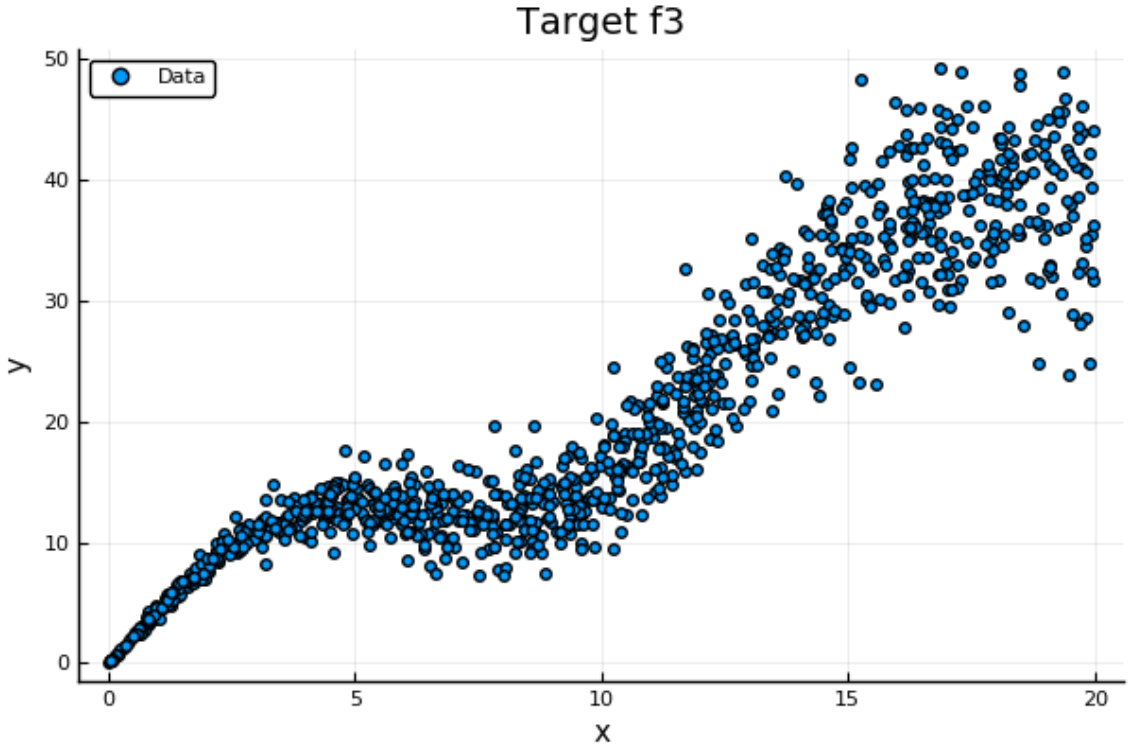
2. [1pts] For all three targets, plot a $n = 1000$ sample of the data. **Note: You will use these plots later, in your writeup display once other questions are complete.**

```
Using Plots
```

```
n=1000
x1,y1 = sample_batch(target_f1,n)
  plot_f1 = scatter(x1',y1,title = "Target f1", xlabel = "x", ylabel = "y",label = "Data")
savefig("A1q2.1")
x2,y2 =sample_batch(target_f2,n)
  plot_f2 = scatter(x2',y2,title = "Target f2", xlabel = "x", ylabel = "y",label = "Data")
savefig("A1q2.2")
x3,y3 = sample_batch(target_f3,n)
  plot_f3 = scatter(x3',y3,title = "Target f3", xlabel = "x", ylabel = "y",label = "Data")
savefig("A1q2.3")
```







2.3 Linear Regression Model with $\hat{\beta}$ MLE

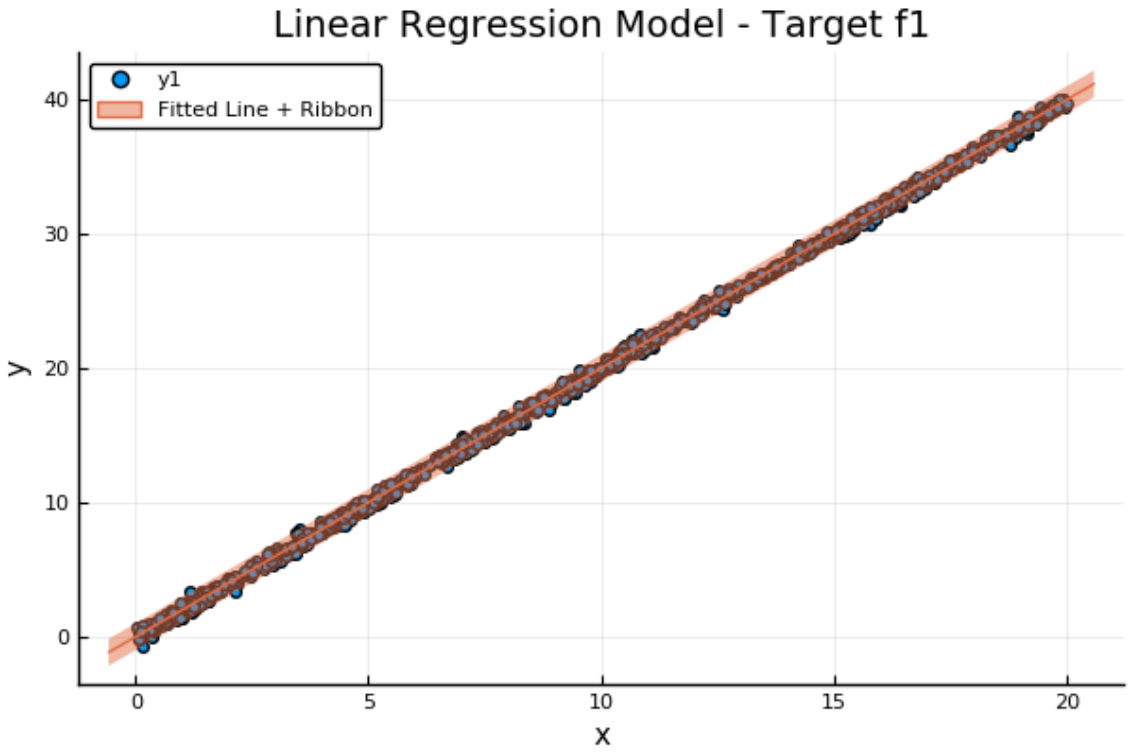
Linear Regression Model with $\hat{\beta}$ MLE [4pts]

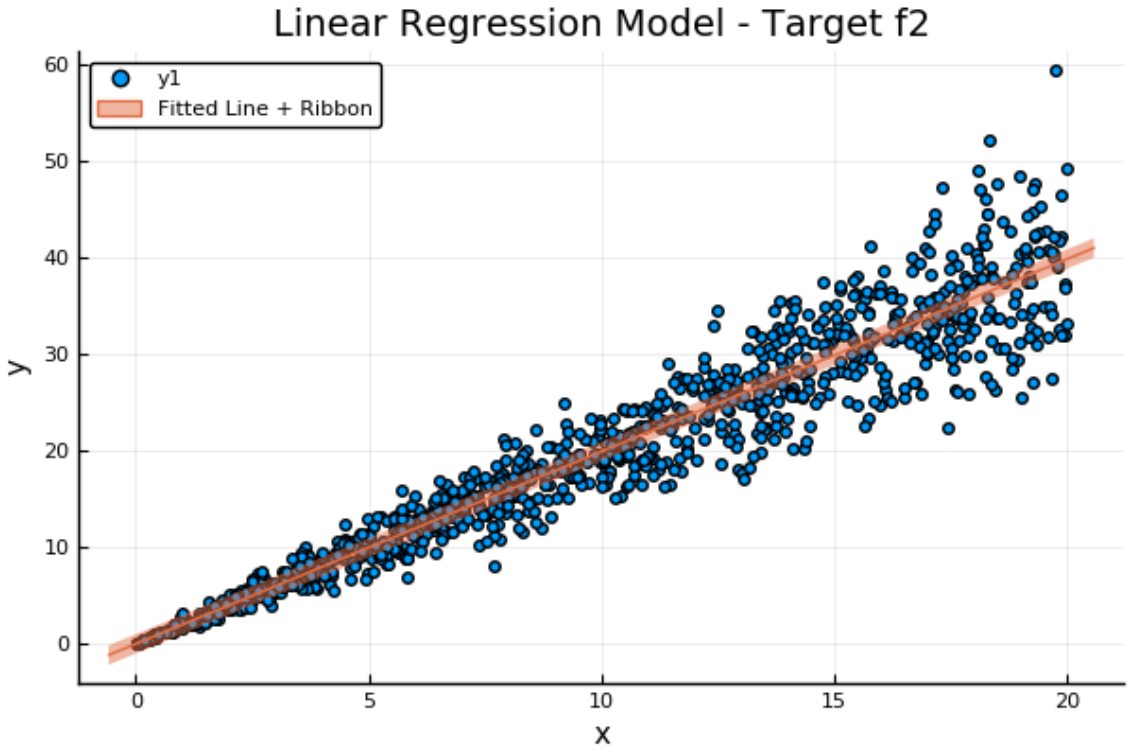
1. [2pts] Program the function that computes the the maximum likelihood estimate given X and Y. Use it to compute the estimate $\hat{\beta}$ for a $n = 1000$ sample from each target function.

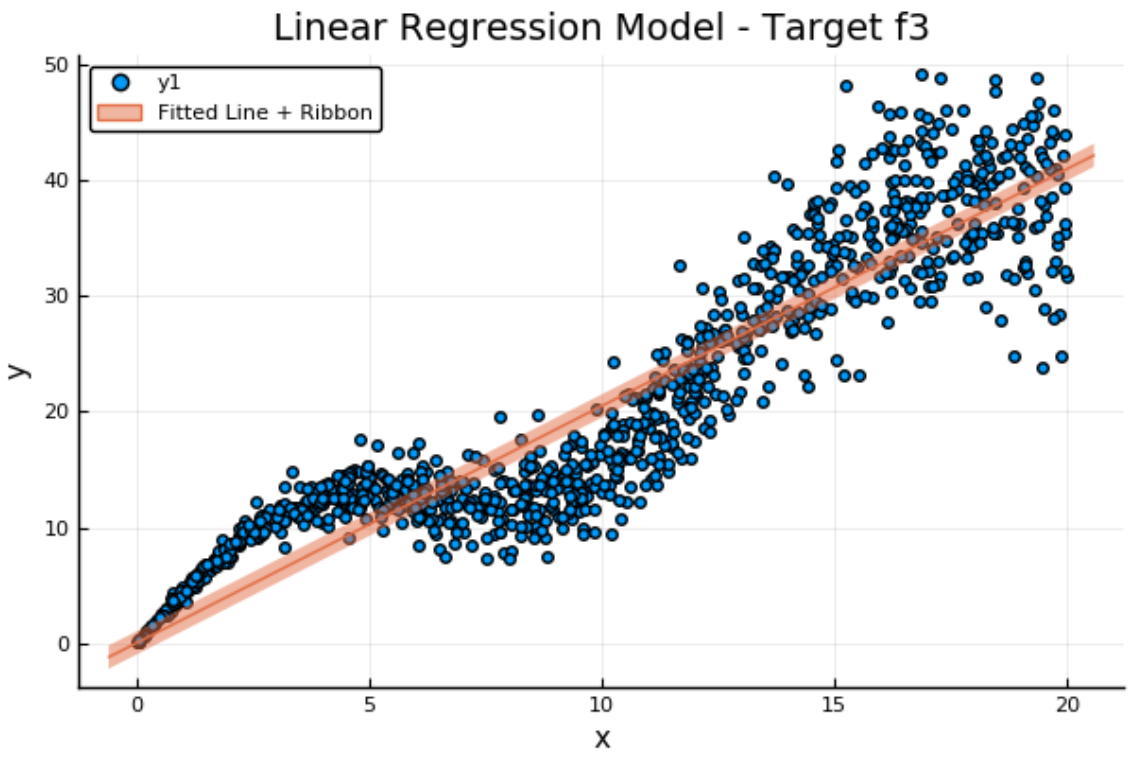
```
function beta_mle(x,y)
  beta = inv(x*x')*(x*y)
  return beta
end

n=1000


x1,y1 = sample_batch(target_f1,n)

x2,y2 =sample_batch(target_f2,n)

x3,y3 = sample_batch(target_f3,n)


lobf_mle_1(x) = x'.*beta_mle(x1,y1)
lobf_mle_2(x) = x'.*beta_mle(x2,y2)
lobf_mle_3(x) = x'.*beta_mle(x3,y3)
```

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**

```
plot_f1_v2 = scatter(x1',y1)
  display(plot!(plot_f1_v2, lobf_mle_1, ribbon= 1.,title = "Linear Regression Model - Target f1", xlabel
savefig("A1q2.4")
plot_f2_v2 = scatter(x2',y2)
  display(plot!(plot_f2_v2, lobf_mle_2, ribbon= 1.,title = "Linear Regression Model - Target f2", xlabel
savefig("A1q2.5")
plot_f3_v2 = scatter(x3',y3)
  display(plot!(plot_f3_v2, lobf_mle_3, ribbon= 1.,title = "Linear Regression Model - Target f3", xlabel
savefig("A1q2.6")
```


Linear Regression Model - Target f1
y1
Fitted Line + Ribbon
y
x
0
10
20
30
40
0
5
10
15
20


Linear Regression Model - Target f2
y1
Fitted Line + Ribbon
y
x
0
10
20
30
40
50
60
0
5
10
15
20



2.4 Log-likelihood of Data Under Model

Log-likelihood of Data Under Model [6pts]
1. [2pts] Write code for the function that computes the likelihood of x under the Gaussian distribution $\mathcal{N}(,s)$. 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)
  n = length(x)
  return (-n/2)*log(2*pi*σ^2)-(1/(2*(σ^2)))*sum((x-μ)^2)
end

# Test Gaussian likelihood against standard implementation
@testset "Gaussian log likelihood" begin
  using Random #TODO: added to fix -inf problem
  Random.seed!(123)
  # 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))
   # 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) ≈ log.(pdf.(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) ≈ log.(pdf.(Normal.(μ,σ),x)) # Correct Values
end
```

```
Test Summary:            Pass  Total
Gaussian log likelihood     6      6
```

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((-1)*(gaussian_log_likelihood.(x'*β, σ, y)))
end
```

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  ------------")
      nll_vec = []
      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)[1]
        nll = lr_model_nll(β_mle,x,y; σ=σ_model)
        println("Negative Log-Likelihood: $nll")
        append!(nll_vec,nll)
      end
    nLog =   findmin(nll_vec)
    op_nll = nLog[1]
    op_sig = (0.1,0.3,1.,2.)[nLog[2]]
```

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.

```
--------  10  ------------
--------  target_f1  ------------
--------  0.1  ------------
Negative Log-Likelihood: 22.339808058275505
--------  0.3  ------------
Negative Log-Likelihood: 0.9801890627384118
```

```
--------  1.0  ------------
Negative Log-Likelihood: 9.872316668237415
--------  2.0  ------------
Negative Log-Likelihood: 16.28260087900393
Min. Negative Log Likelihood: 0.9801890627384118  Optimal Sigma: 0.3
--------  target_f2  ------------
--------  0.1  ------------
Negative Log-Likelihood: 4569.804432443775
--------  0.3  ------------
Negative Log-Likelihood: 808.6101530890096
--------  1.0  ------------
Negative Log-Likelihood: 83.698203062815
--------  2.0  ------------
Negative Log-Likelihood: 24.19663742052348
Min. Negative Log Likelihood: 24.19663742052348  Optimal Sigma: 2.0
--------  target_f3  ------------
--------  0.1  ------------
Negative Log-Likelihood: 16953.23226953066
--------  0.3  ------------
Negative Log-Likelihood: 1047.1597006225077
--------  1.0  ------------
Negative Log-Likelihood: 86.5744715218029
--------  2.0  ------------
Negative Log-Likelihood: 38.379978091761316
Min. Negative Log Likelihood: 38.379978091761316  Optimal Sigma: 2.0
--------  100  ------------
--------  target_f1  ------------
--------  0.1  ------------
Negative Log-Likelihood: 281.4711380082697
--------  0.3  ------------
Negative Log-Likelihood: 28.93098528177236
--------  1.0  ------------
Negative Log-Likelihood: 96.58070543457596
--------  2.0  ------------
Negative Log-Likelihood: 162.10676707206432
Min. Negative Log Likelihood: 28.93098528177236  Optimal Sigma: 0.3
--------  target_f2  ------------
--------  0.1  ------------
Negative Log-Likelihood: 71219.65564803002
--------  0.3  ------------
Negative Log-Likelihood: 7017.015572790145
--------  1.0  ------------
Negative Log-Likelihood: 680.8393550227454
--------  2.0  ------------
Negative Log-Likelihood: 311.64692780069123
Min. Negative Log Likelihood: 311.64692780069123  Optimal Sigma: 2.0
--------  target_f3  ------------
--------  0.1  ------------
Negative Log-Likelihood: 103585.29451849911
--------  0.3  ------------
Negative Log-Likelihood: 10390.52926759204
--------  1.0  ------------
Negative Log-Likelihood: 943.4570793115731
--------  2.0  ------------
```

```
Negative Log-Likelihood: 444.6601264702941
Min. Negative Log Likelihood: 444.6601264702941  Optimal Sigma: 2.0
--------  1000  ------------
--------  target_f1  ------------
--------  0.1  ------------
Negative Log-Likelihood: 3087.5986499607916
--------  0.3  ------------
Negative Log-Likelihood: 180.35655866359065
--------  1.0  ------------
Negative Log-Likelihood: 964.6561571787815
--------  2.0  ------------
Negative Log-Likelihood: 1623.8142161151418
Min. Negative Log Likelihood: 180.35655866359065  Optimal Sigma: 0.3
--------  target_f2  ------------
--------  0.1  ------------
Negative Log-Likelihood: 559608.4950206816
--------  0.3  ------------
Negative Log-Likelihood: 63426.76140934152
--------  1.0  ------------
Negative Log-Likelihood: 7658.88529092344
--------  2.0  ------------
Negative Log-Likelihood: 3021.0272572913927
Min. Negative Log Likelihood: 3021.0272572913927  Optimal Sigma: 2.0
--------  target_f3  ------------
--------  0.1  ------------
Negative Log-Likelihood: 1.0829641655336618e6
--------  0.3  ------------
Negative Log-Likelihood: 128565.141083667
--------  1.0  ------------
Negative Log-Likelihood: 13124.478318320016
--------  2.0  ------------
Negative Log-Likelihood: 4564.285416963065
Min. Negative Log Likelihood: 4564.285416963065  Optimal Sigma: 2.0
```

2.5 Automatic Differentiation and Maximizing Likelihood

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
  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 = -(x *y - x * x'*β_test) / (σ_test^2) #TODO: the new correct hand_derivative based o
  @test ad_grad[1] ≈ hand_derivative
end
```

```
Test Summary:            Pass  Total
Gradients wrt parameter     1      1
```

2.5.1 Train Linear Regression Model with Gradient Descent

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 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
  #grad_β = 0 TODO: Remove this line
  for i in 1:iters
    x,y = sample_batch(target_f,bs)
    @info "loss: $(lr_model_nll(β_curr,x,y;σ=σ_model)) β: $β_curr"
    grad_β = gradient((β) -> lr_model_nll(β,x,y;σ=σ_model), β_curr)[1]
    β_curr -= (grad_β) * lr
  end
  return β_curr
end
```

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()
β1_learned = train_lin_reg(target_f1,β_init)
β2_learned= train_lin_reg(target_f2,β_init)
β3_learned= train_lin_reg(target_f3,β_init)

v1(x) = x'.*β1_learned
v2(x) = x'.*β2_learned
v3(x) = x'.*β3_learned

plot_f1_new = scatter(x1',y1, label = "Data")
  display(plot!(plot_f1_new, v1, ribbon=1., title = "Trained Linear Regression Model - Target f1",
  xlabel = "x", ylabel = "y",label = "Fitted Line"))
savefig("A1q2.7")

plot_f2_new = scatter(x2',y2,label = "Data")
  display(plot!(plot_f2_new, v2, ribbon=1.,title = "Trained Linear Regression Model - Target f2",
  xlabel = "x", ylabel = "y",label = "Fitted Line"))
savefig("A1q2.8")

plot_f3_new = scatter(x3',y3,label = "Data")
  display(plot!(plot_f3_new, v3, ribbon=1.,title = "Trained Linear Regression Model - Target f3",
  xlabel = "x", ylabel = "y",label = "Fitted Line"))
savefig("A1q2.9")
```

2.5.2 Non-linear Regression with a Neural Network

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$_n$et'withparametersθ (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)
  w1 = θ[1]
  b1 = θ[2]
  w2 = θ[3]
  b2 = θ[4]
  return tanh.(x'*w1 .+ b1)*w2 .+ b2
end

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

@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
```

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)
  n_Net = neural_net(x,θ)
  return sum(-1*gaussian_log_likelihood.(n_Net, σ, y))
end
```

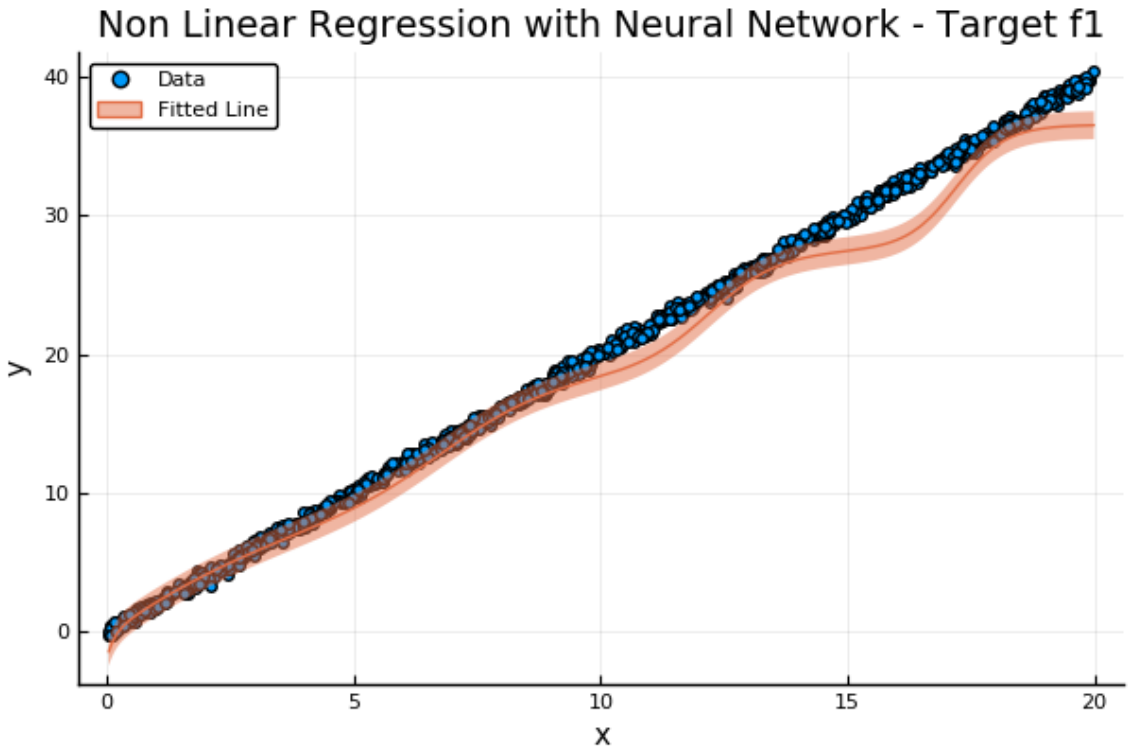
3. [2pts] Write a function train_nn_reg that accepts a target function and an initial estimate for θ and some hyperparameters for batch-size, model variance, learning rate, and number of iterations. Then, for each iteration: * sample data from the target function * compute gradients of negative log-likelihood 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.

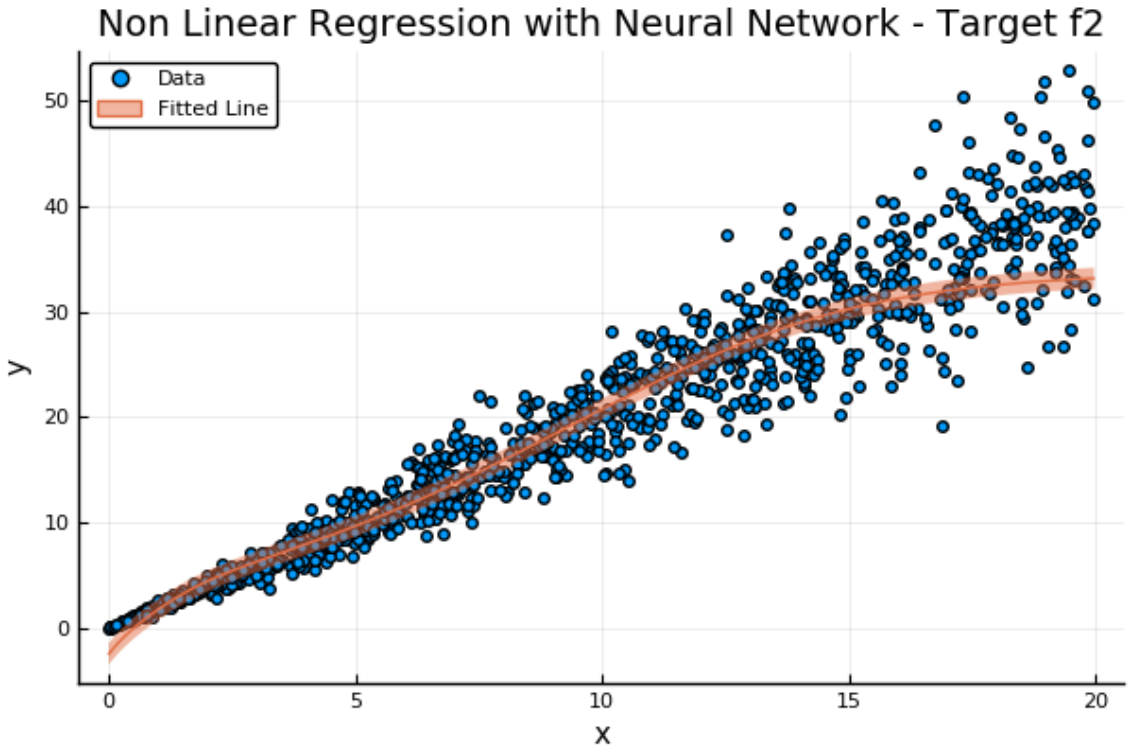
```
using Logging # Print training progress to REPL, not pdf
function train_nn_reg(target_f, θ_init; bs= 100, lr = 1e-5, iters=100, σ_model = 1. )
    θ_curr = θ_init
    for i in 1:iters
```

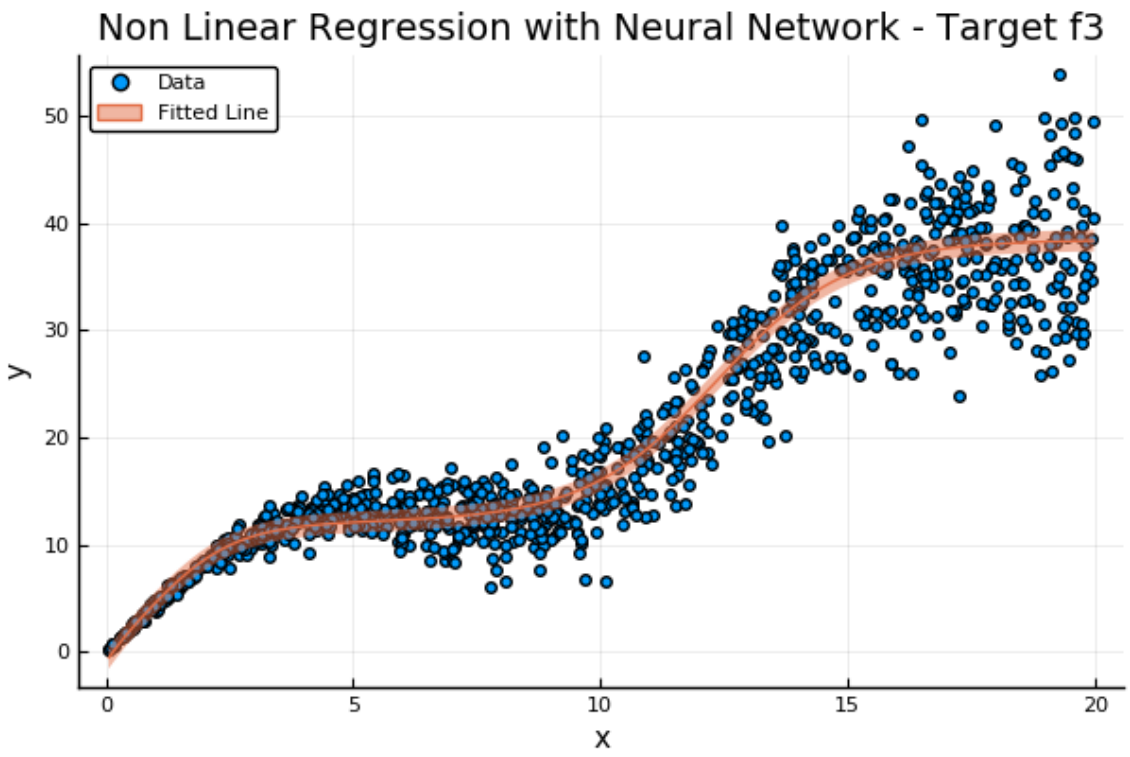
```
    x,y = sample_batch(target_f,bs)
```

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 $\sigma = 1.0$

```
θ_init = θ
θ1_learned =  train_nn_reg(target_f1, θ_init; bs=1000, iters=7000, lr=1e-5)
θ2_learned =  train_nn_reg(target_f2, θ_init; bs=1000, iters=10000, lr=1e-6)
θ3_learned =  train_nn_reg(target_f3, θ_init; bs=1000, iters=10000, lr=1e-6)



x1_new,y1_new = sample_batch(target_f1, 1000)
plot_f1_new = scatter(x1_new', y1_new,label = "Data")
  display(plot!(plot_f1_new, sort(x1_new'),sort(neural_net(x1_new, θ1_learned)), ribbon=1., title = "Nor
savefig("A1q2.10")

x2_new,y2_new = sample_batch(target_f2, 1000)
plot_f2_new = scatter(x2_new', y2_new,label = "Data")
  display(plot!(plot_f2_new, sort(x2_new'),sort(neural_net(x2_new, θ2_learned)), ribbon=1., title = "Nor
savefig("A1q2.11")

x3_new,y3_new = sample_batch(target_f3, 1000)
plot_f3_new = scatter(x3_new', y3_new,label = "Data")
  display(plot!(plot_f3_new, sort(x3_new'),sort(neural_net(x3_new, θ3_learned)), ribbon=1.,title = "Non
savefig("A1q2.12")
```


Non Linear Regression with Neural Network - Target f1
Data
Fitted Line
y
x
0
10
20
30
40
0
5
10
15
20


Non Linear Regression with Neural Network - Target f2
Data
Fitted Line
y
x
0
10
20
30
40
50
0
5
10
15
20



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

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)

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

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.

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.

```
# Neural Network Function
function neural_net_w_var(x,θ)
  w1 = θ[1]
  b1 = θ[2]
  w2 = θ[3]
  b2 = θ[4]
  m = tanh.(x'*w1 .+ b1)*w2 .+ b2
  μ = m[:,1]
  logσ = m[:,2]
  return μ, logσ
```

```
end

# Random initial Parameters
Random.seed!(4)
θ_new = (randn(1, 10), randn(1,10), randn(10,2), randn(1,2))

@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_new)
  @test size(μ) == (n,)
  @test size(logσ) == (n,)
end
```

```
Test Summary:                                 Pass  Total
neural net mean and logsigma vector output     2      2
```

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 $\exp \log \sigma$)

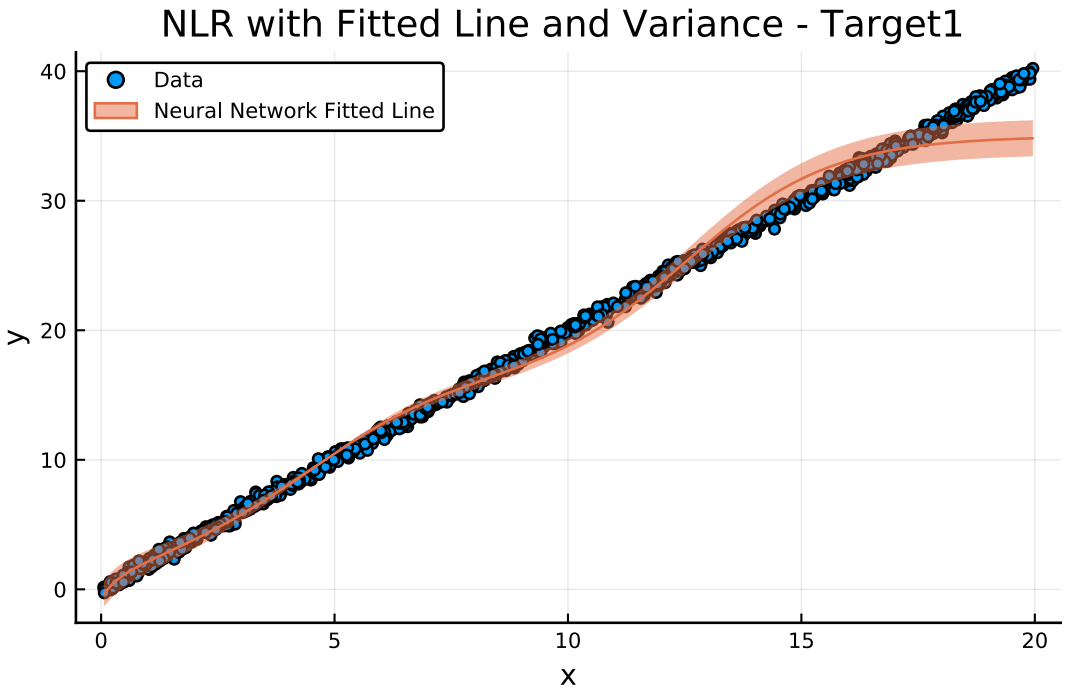
```
function nn_with_var_model_nll(θ,x,y)
  μ,σ = neural_net_w_var(x,θ)
  return sum(-1*gaussian_log_likelihood.(μ, exp.(σ), y))
end
```

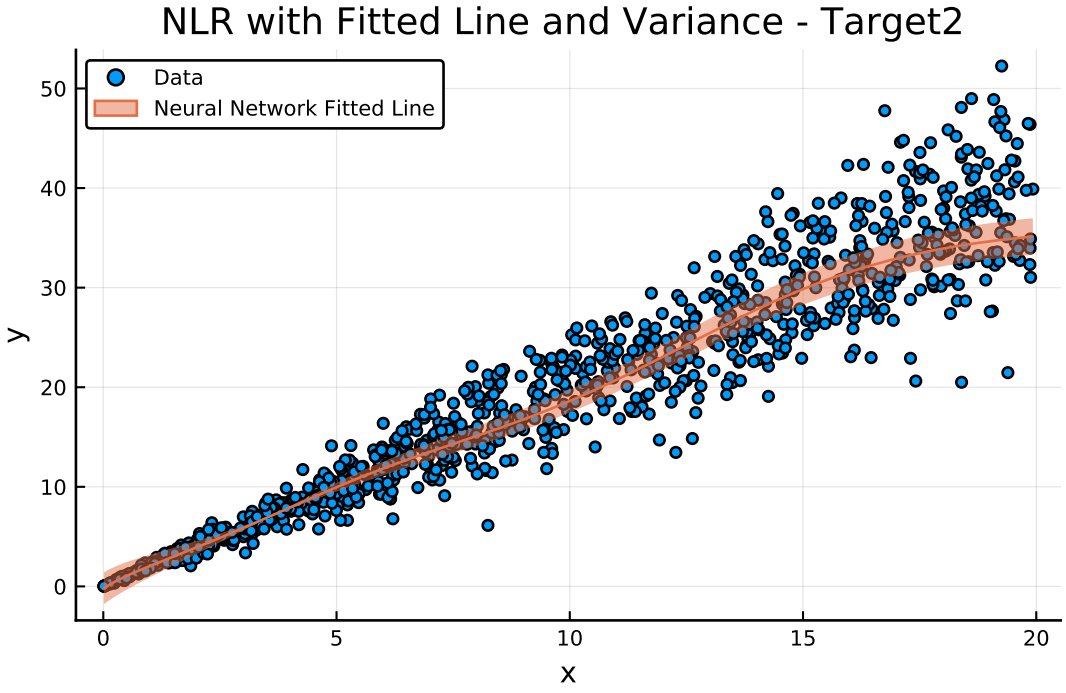
3. [1pts] Write a function train_nn_w_var_reg that accepts a target function and an initial estimate for θ and some hyperparameters for batch-size, learning rate, and number of iterations. Then, for each iteration: * sample data from the target function * compute gradients of negative log-likelihood 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.

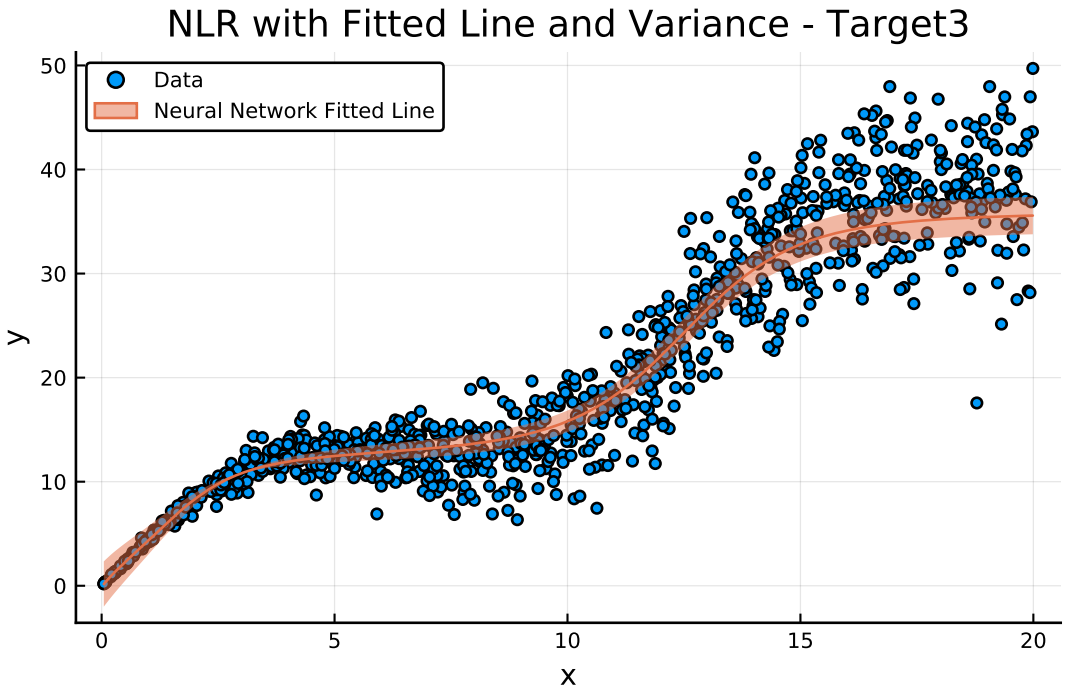
```
function train_nn_w_var_reg(target_f, θ_init; bs= 1000, lr = 1e-4, iters=10000)
    θ_curr = θ_init
    for i in 1:iters
      x,y = sample_batch(target_f,bs)
      @info "loss: $(nn_with_var_model_nll(θ_curr,x,y))"
      grad_θ = gradient((θ) -> nn_with_var_model_nll(θ,x,y), θ_curr)[1]
      θ_curr = θ_curr .- lr .*grad_θ
    end
    return θ_curr
end
```

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.

```
Random.seed!(4)
θ_init = θ_new
θ1_new_learned =  train_nn_w_var_reg(target_f1, θ_init; bs=500, lr = 1e-5, iters=15000)
θ2_new_learned =  train_nn_w_var_reg(target_f2, θ_init; lr = 1e-5, iters=15000)
θ3_new_learned =  train_nn_w_var_reg(target_f3, θ_init; lr = 1e-5, iters=20000)


x1_new2,y1_new2 = sample_batch(target_f1, 1000)
plot_f1_new2 = scatter(x1_new2', y1_new2, label="Data"
  ,xlabel="x",ylabel="y",margin=5mm, title="NLR with Fitted Line and Variance - Target1 ")
  display(plot!(plot_f1_new2, vec(x1_new2), neural_net_w_var(x1_new2, θ1_new_learned)
    , label="Neural Network Fitted Line", seriestype=:line
    , ribbon =neural_net_w_var(x1_new2, θ1_new_learned)[2]))
      savefig("A1q2.13")

x2_new2,y2_new2 = sample_batch(target_f2, 1000)
plot_f2_new2 = scatter(x2_new2', y2_new2, legend=:topleft, label="Data"
  ,xlabel="x",ylabel="y",margin=5mm, title="NLR with Fitted Line and Variance - Target2 ")
  display(plot!(plot_f2_new2, vec(x2_new2), neural_net_w_var(x2_new2, θ2_new_learned)[1]
    , label="Neural Network Fitted Line",seriestype=:line
    , ribbon =neural_net_w_var(x2_new2, θ2_new_learned)[2]))
      savefig("A1q2.14")

x3_new2,y3_new2 = sample_batch(target_f3, 1000)
plot_f3_new2 = scatter(x3_new2', y3_new2, legend=:topleft, label="Data"
  ,xlabel="x",ylabel="y",margin=5mm, title="NLR with Fitted Line and Variance - Target3")
  display(plot!(plot_f3_new2, vec(x3_new2), neural_net_w_var(x3_new2, θ3_new_learned)[1]
    , label="Neural Network Fitted Line",seriestype=:line
    , ribbon =neural_net_w_var(x3_new2, θ3_new_learned)[2]))
      savefig("A1q2.15")
```


NLR with Fitted Line and Variance - Target1
Data
Neural Network Fitted Line
y
x
0
10
20
30
40
0
5
10
15
20


NLR with Fitted Line and Variance - Target2
Data
Neural Network Fitted Line
y
x
0
10
20
30
40
50
0
5
10
15
20


NLR with Fitted Line and Variance - Target3
Data
Neural Network Fitted Line
y
x
0
10
20
30
40
50
0
5
10
15
20