

Related code for the assignment can be found at A1.jmd

# 1 Decision Theory

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

1. 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)
    a = Array{Float64}(undef, size(prob_spam)[1], 2)
    a[:,1] = prob_spam
    a[:,2] = 1.0 .- prob_spam
    b = losses[action]
    c = a * b
end

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

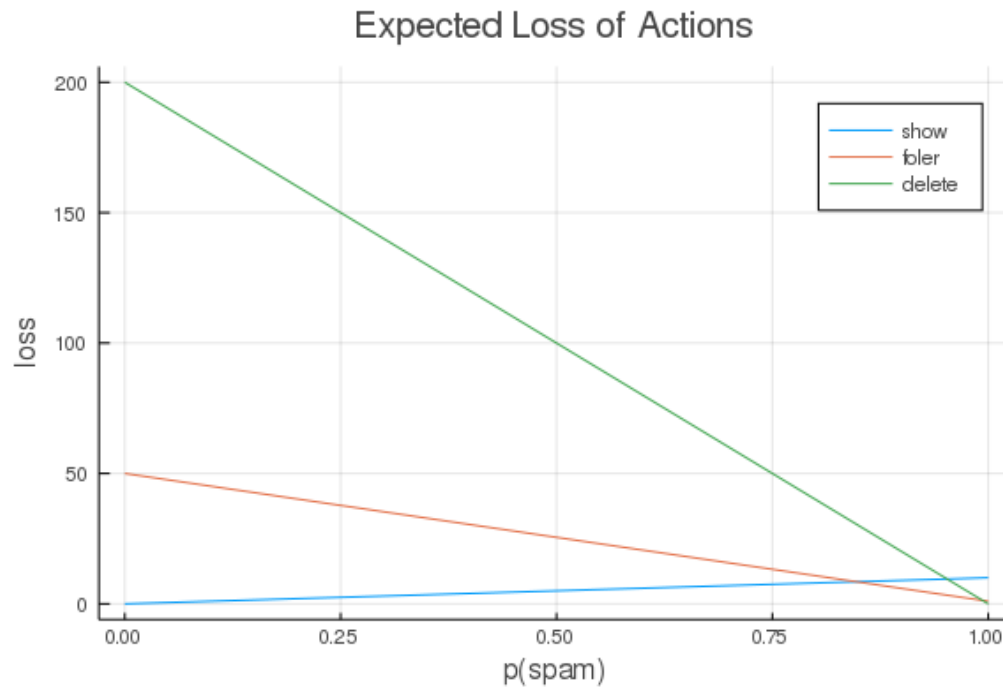
text_actions = ["show", "folder", "delete"]

using Plots

for action in 1:num_actions
    display(plot!(prob_range,
                  expected_loss_of_action(prob_range, action),
                  label=text_actions[action],
                  title="Expected Loss of Actions",
                  xlabel="p(spam)",
                  ylabel="loss"))
end

savefig("1_1_expected_loss_of_action.png")

```

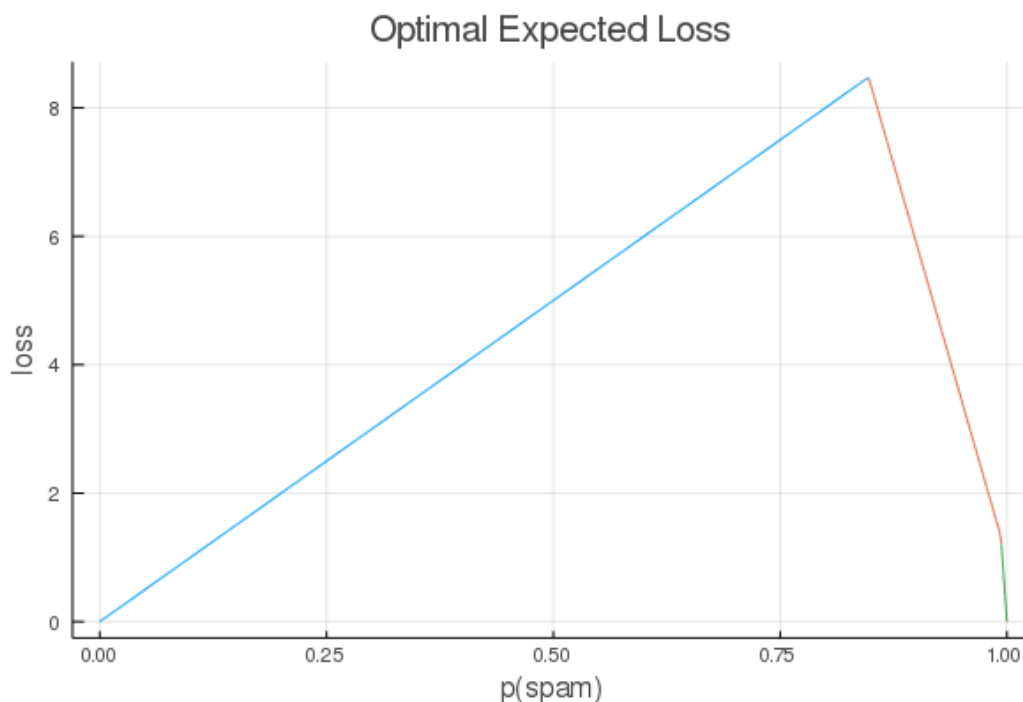


2. Write a function that computes the optimal action given the probability of spam.

```
function optimal_action(prob_spam)
    out = zeros(size(prob_spam)[1], num_actions)
    for action in 1:num_actions
        out[:,action] = expected_loss_of_action(prob_range, action)
    end
    findmin(out, dims=2)
end
```

3. 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)
best = optimal_action(prob_range)
optimal_losses = best[1]
optimal_actions = getindex.(best[2],2)
plot(prob_range, optimal_losses, linecolor=optimal_actions)
```



4. 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.

Solve for intersections of action pair (folder,delete):

$$p + 50(1 - p) = 200(1 - p)$$

$$p + 50 - 50p = 200 - 200p$$

$$151p = 150$$

$$p = 150/151$$

Deleting an email is optimal for  $p(\text{spam}) \in [150/151, 1]$ .

## 2 Regression

### 2.1 Manually Derived Linear Regression

1. What happens if  $n < m$ ?

Then, the problem is underconstrained for  $Y = X^T\beta + \epsilon$  and there exists multiple solutions.

$$Y = X^T(B + B'), \forall B' \in \text{Nullspace}(X^T)$$

$$\beta^* = X(X^T X)^{-1}Y \implies X^T \beta^* = X^T X(X^T X)^{-1}Y = Y$$

$$(\beta - \beta^*)^T \beta^* = (\beta - \beta^*)^T X(X^T X)^{-1}Y = (X^T \beta - X^T \beta^*)(X^T X)^{-1}Y = 0$$

Thus,  $\beta^* = X(X^T X)^{-1}Y$  works.

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

$$\begin{aligned} E[\hat{\beta}] &= E[(X X^T)^{-1} X Y] \\ &= E[(X X^T)^{-1} X X^T \beta + \epsilon], \epsilon \sim \mathcal{N}(0, \sigma^2 I) \\ &= \beta \end{aligned}$$

$$\begin{aligned} \text{Var}(\hat{\beta}) &= \text{Var}((X X^T)^{-1} X X^T \beta + \epsilon) \\ &= \text{Var}(\epsilon) \\ &= \sigma^2 I \end{aligned}$$

3. Show that maximizing the likelihood is equivalent to minimizing the squared error  $\sum_i (y_i - x_i \beta)^2$ .

$$\begin{aligned} l(\theta|D) &= \log p(\theta|D) = \log \prod_i \mathcal{N}(x_i^T \beta, \sigma^2) \\ \max l(\theta|D) &= \min -l(\theta|D) \\ -l(\theta|D) &= -\sum_i \log \frac{1}{2\pi\sigma^2} e^{-\frac{(y_i - x_i^T \beta)^2}{2\sigma^2}} \\ \frac{\partial(-l(\theta|D))}{\partial \beta} &= 0 \\ &= \frac{\partial \sum_i \frac{(y_i - x_i^T \beta)^2}{2\sigma^2}}{\partial \beta} \\ &= \frac{2 \sum_i (y_i - x_i^T \beta)(-x_i)}{2\sigma^2} \end{aligned}$$

Thus, minimizing likelihood is equivalent to minimizing  $\sum_i (y_i - x_i^T \beta)^2$ .

4. Write the squared error in vector notation, (see above hint), expand the expression, and collect like terms.

$$\begin{aligned}
 (y_i - x_i^T \beta)^2 &= (y_i - x_i^T \beta)^T (y_i - x_i^T \beta), y_i \in \mathbb{R}, x_i \in \mathbb{R}^m, \beta \in \mathbb{R}^m \\
 &= (y_i - x_i^T \beta)^T (y_i - x_i^T \beta) \\
 &= y_i^2 - y_i x_i^T \beta - \beta^T x_i y_i + \beta^T x_i x_i^T \beta \\
 &= y_i^2 - 2y_i x_i^T \beta + (x_i^T \beta)^2 \\
 \sum_i (y_i - x_i^T \beta)^2 &= y^T y - 2y^T X^T \beta + (X^T \beta)^T (X^T \beta)
 \end{aligned}$$

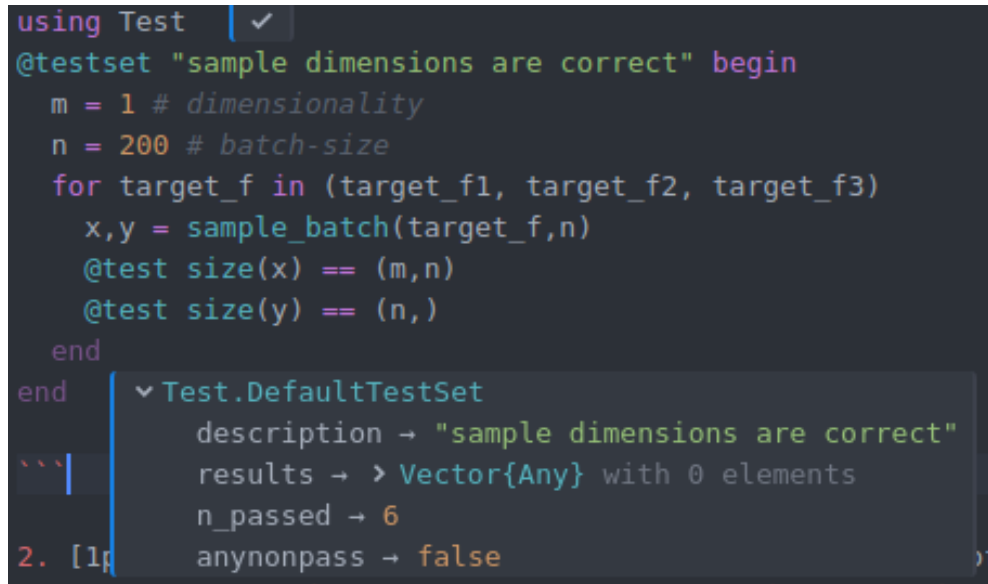
5. Use the likelihood expression to write the negative log-likelihood. Write the derivative of the negative log-likelihood with respect to  $\beta$ , set equal to zero, and solve to show the maximum likelihood estimate  $\hat{\beta}$  as above.

$$\begin{aligned}
 -l(\theta|D) &= -\sum_i \log \frac{1}{2\pi\sigma^2} e^{-\frac{(y_i - x_i^T \beta)^2}{2\sigma^2}} \\
 \frac{\partial(-l(\theta|D))}{\partial\beta} &= 0 \\
 &= \frac{\partial \sum_i \frac{(y_i - x_i^T \beta)^2}{2\sigma^2}}{\partial\beta} \\
 &= \frac{\partial}{\partial\beta} \left( \frac{y^T y - 2y^T X^T \beta + (X^T \beta)^T (X^T \beta)}{2\sigma^2} \right) \\
 &= \frac{\partial}{\partial\beta} (-2y^T X^T \beta + (X^T \beta)^T (X^T \beta)) \\
 &= -2(y^T X^T)^T + \frac{\partial(X^T \beta)^T}{\partial\beta} (X^T \beta) + \frac{\partial(X^T \beta)^T}{\partial\beta} (X^T \beta)^T \\
 &= -2Xy + 2XX^T \beta \\
 XX^T \beta &= Xy \\
 \beta &= (XX^T)^{-1} Xy
 \end{aligned}$$

## 2.2 Toy Data

1. Write a function which produces a batch of data  $x \sim \text{Uniform}(0, 20)$  and  $y = \text{target}_f(x)$

```
function sample_batch(target_f, batch_size)
    x = 20.0 * rand(1, batch_size)
    y = target_f(x)
    return (x,y)
end
```



```
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.DefaultTestSet
description → "sample dimensions are correct"
results → > Vector{Any} with 0 elements
n_passed → 6
anynonpass → false
```

2. For all three targets, plot a  $n = 1000$  sample of the data.

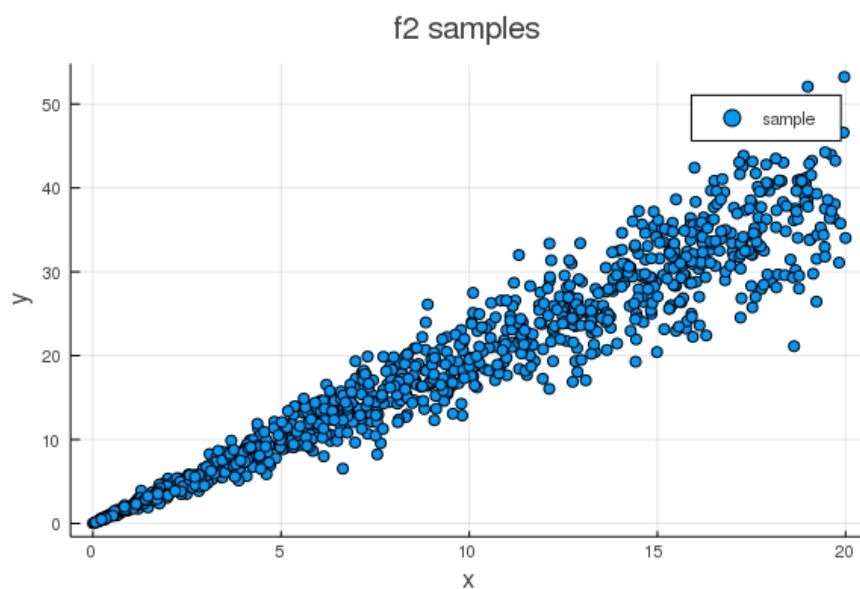
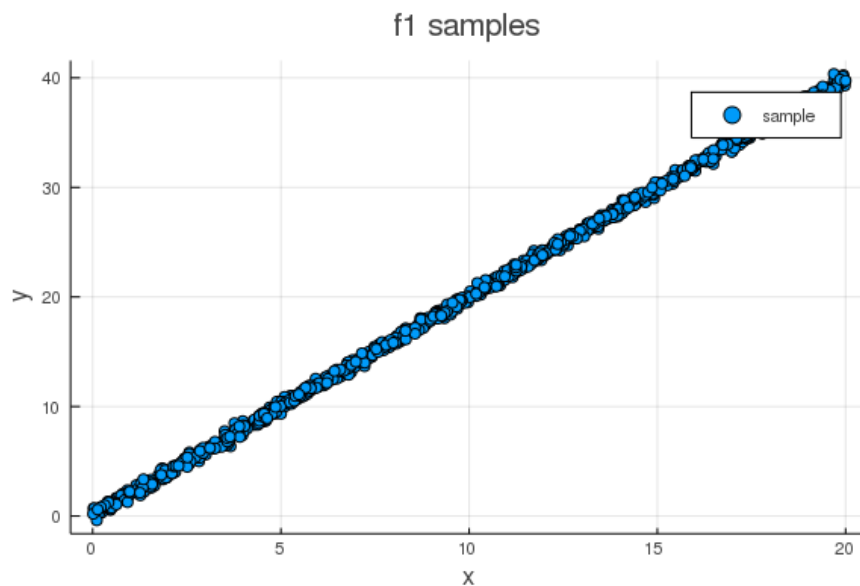
```
using Plots
```

```
x1,y1 = sample_batch(target_f1,1000)
plot_f1 = plot(x1[1,:],y1,seriestype=:scatter,
    title="f1 samples",
    xlabel="x",
    ylabel="y",
    label="sample")
```

```
x2,y2 = sample_batch(target_f2,1000)
plot_f2 = plot(x2[1:],y2,seriestype=:scatter,
    title="f2 samples",
    xlabel="x",
    ylabel="y",
    label="sample")
```

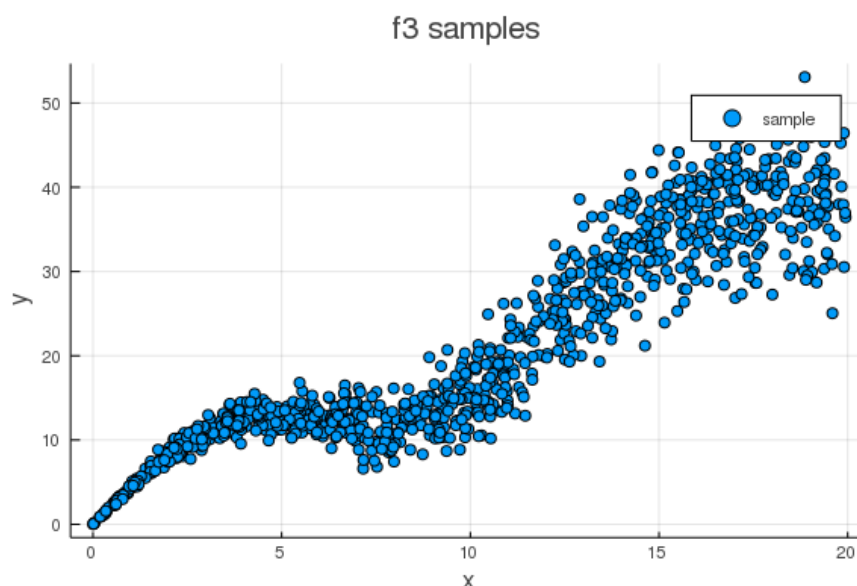
```
x3,y3 = sample_batch(target_f3,1000)
plot_f3 = plot(x3[1:],y3,seriestype=:scatter,
```

```
title="f3 samples",  
xlabel="x",  
ylabel="y",  
label="sample")
```



## 2.3 Linear Regression Model with $\beta$ MLE

1. 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 = sample_batch(target_f1,1000)
beta_mle_1 = beta_mle(x_1, y_1)

x_2, y_2 = sample_batch(target_f2,1000)
beta_mle_2 = beta_mle(x_2, y_2)

x_3, y_3 = sample_batch(target_f3,1000)
beta_mle_3 = beta_mle(x_3, y_3)
```

- For each function, plot the linear regression model given by  $Y \sim \mathcal{N}(X^T\beta, \sigma^2 I)$  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.

Linear regression training with a constant bias, making  $\beta = [\beta_1, \beta_2]$ , was tested and makes very little difference since bias is very close to 0, so a scalar  $\beta$  for estimating slope suffices.



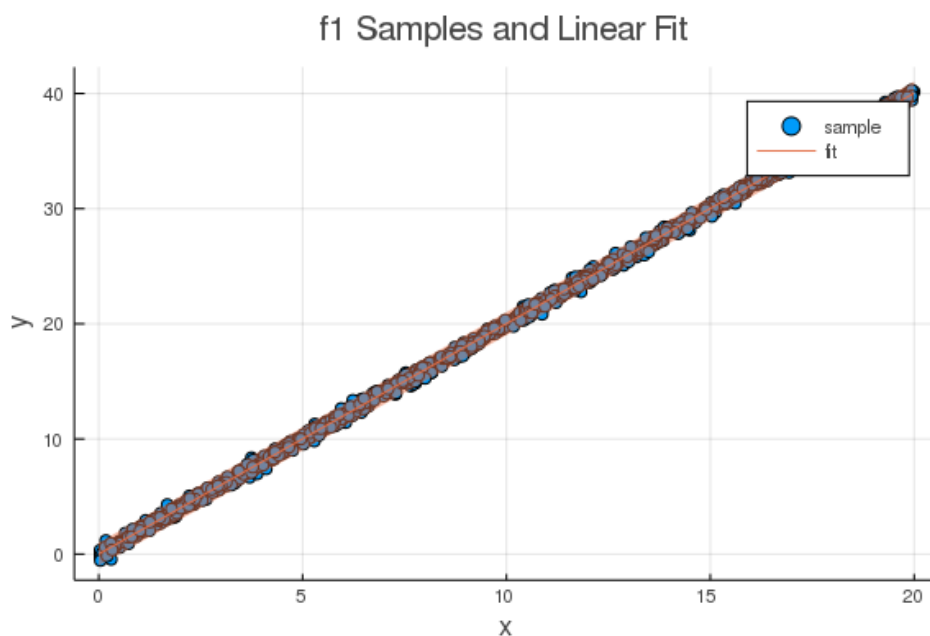
```
using Distributions

x=0:20

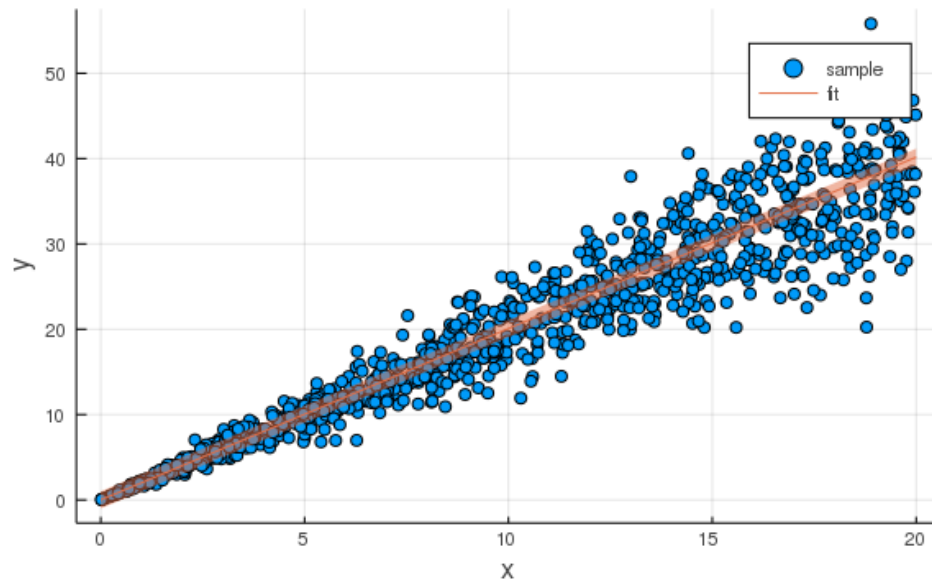
fit_1 = x ->  $\beta_{\text{mle}_1}[1] * x$ 
plot(plot_f1)
plot!(fit_1, 0,20,
      title="f1 Samples and Linear Fit",
      ribbon=1.0,
      label="fit")
savefig("imgs/2_3_2_1.png")

fit_2 = x ->  $\beta_{\text{mle}_2}[1] * x$ 
plot(plot_f2)
plot!(fit_2, 0,20,
      title="f2 Samples and Linear Fit",
      ribbon=1.0,
      label="fit")
savefig("imgs/2_3_2_2.png")

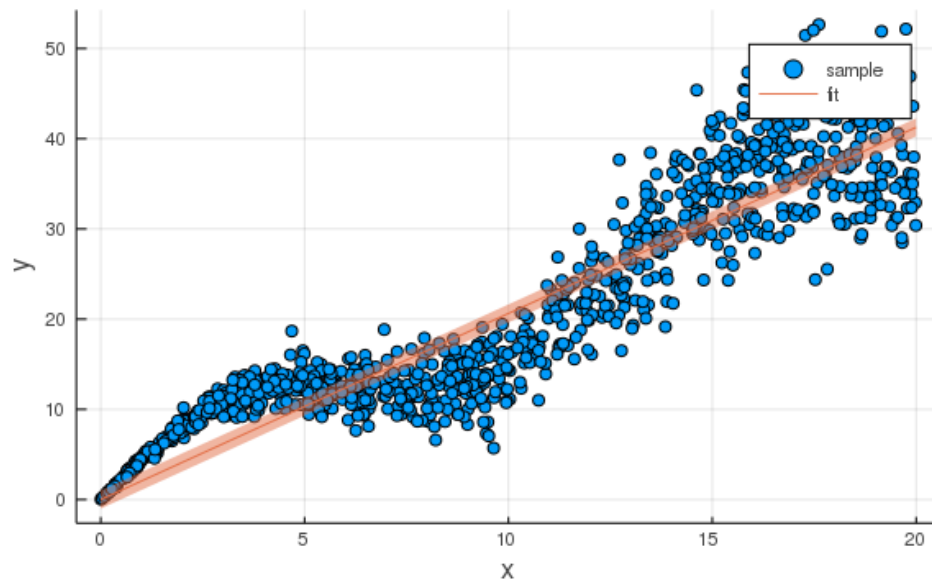
fit_3 = x ->  $\beta_{\text{mle}_3}[1] * x$ 
plot_f3
plot!(fit_3, 0,20,
      title="f3 Samples and Linear Fit",
      ribbon=1.0,
      label="fit")
savefig("imgs/2_3_2_3.png")
```



f2 Samples and Linear Fit



f3 Samples and Linear Fit



## 2.4 Log-likelihood of Data Under Model

1. 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  $x, \mu, \sigma$  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( $\mu$ ,  $\sigma$ , x)
    log.(1 ./ sqrt.(2*pi.* $\sigma$ .^2)) .+ (-0.5 .* ((x.- $\mu$ ).^2)/ $\sigma$ .^2)
end
```

Test Gaussian likelihood against standard implementation.

```
@testset "Gaussian log likelihood" begin
    using Distributions: pdf, Normal
    # Scalar mean and variance
    x = randn()
     $\mu$  = randn()
     $\sigma$  = rand()
    @test size(gaussian_log_likelihood( $\mu$ , $\sigma$ ,x)) == () # Scalar log-likelihood
    @test gaussian_log_likelihood( $\mu$ , $\sigma$ ,x)  $\approx$  log.(pdf.(Normal( $\mu$ , $\sigma$ ),x)) # Correct Value
    # Vector valued x under constant mean and variance
    x = randn(100)
     $\mu$  = randn()
     $\sigma$  = rand()
    @test size(gaussian_log_likelihood( $\mu$ , $\sigma$ ,x)) == (100,) # Vector of log-likelihoods
    @test gaussian_log_likelihood( $\mu$ , $\sigma$ ,x)  $\approx$  log.(pdf.(Normal( $\mu$ , $\sigma$ ),x)) # Correct Values
    # Vector valued x under vector valued mean and variance
    x = randn(10)
     $\mu$  = randn(10)
     $\sigma$  = rand(10)
    @test size(gaussian_log_likelihood( $\mu$ , $\sigma$ ,x)) == (10,) # Vector of log-likelihoods
    @test gaussian_log_likelihood( $\mu$ , $\sigma$ ,x)  $\approx$  log.(pdf.(Normal.( $\mu$ , $\sigma$ ),x)) # Correct Values
end
```

2. 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  $\beta$ .

```
function lr_model_nll( $\beta$ ,x,y; $\sigma$ =1.)
    mu = (x' *  $\beta$ )
    sum(-1 .* gaussian_log_likelihood(mu,  $\sigma$ , y))
end
```

3. 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  $\sigma_{\text{model}}$  in (0.1,0.3,1.,2.)
      println("-----  $\sigma_{\text{model}}$  -----")
      x,y = sample_batch(target_f,n)
       $\beta_{\text{mle}}$  = beta_mle(x,y)
      nll = lr_model_nll( $\beta_{\text{mle}}$ ,x,y, $\sigma=\sigma_{\text{model}}$ )
      println("Negative Log-Likelihood: $nll")
    end
  end
end
end

```

4. For each target function, what is the best choice of  $\sigma$ ?

Using N=10:

	$\sigma = 0.1$	$\sigma = 0.3$	$\sigma = 1.0$	$\sigma = 2.0$
target_f1	29.33	4.91	9.49	16.24
target_f2	3855.20	679.16	53.88	31.19
target_f3	18084.52	805.28	136.68	31.27

We observed best choice of  $\sigma_{\text{target\_f1}} = 0.3$ ,  $\sigma_{\text{target\_f2}} = 2.0$ ,  $\sigma_{\text{target\_f3}} = 2.0$ .

Using N=100:

	$\sigma = 0.1$	$\sigma = 0.3$	$\sigma = 1.0$	$\sigma = 2.0$
target_f1	290.15	29.81	96.2	162.2
target_f2	53752.83	6605.53	489.89	281.52
target_f3	123286.74	14524.38	1109.84	413.30

We observed best choice of  $\sigma_{\text{target\_f1}} = 0.3$ ,  $\sigma_{\text{target\_f2}} = 2.0$ ,  $\sigma_{\text{target\_f3}} = 2.0$ .

Using N=1000:

	$\sigma = 0.1$	$\sigma = 0.3$	$\sigma = 1.0$	$\sigma = 2.0$
target_f1	2840.99	205.14	965.24	1623.16
target_f2	588157.09	64999.86	7775.36	3134.97
target_f3	1286740.39	128972.94	12401.60	4612.70

We observed best choice of  $\sigma_{\text{target\_f1}} = 0.3$ ,  $\sigma_{\text{target\_f2}} = 2.0$ ,  $\sigma_{\text{target\_f3}} = 2.0$ .

## 2.5 Automatic Differentiation with Maximizing Likelihood

For a random value of  $\beta$ ,  $\sigma$ , 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 to  $\beta$ . Test that this is equivalent to the hand-derived value.

using Zygote: gradient

```
@testset "Gradients wrt parameter" begin
   $\beta_{\text{test}}$  = randn()
   $\sigma_{\text{test}}$  = rand()
  x,y = sample_batch(target_f1,100)
  ad_grad = gradient( (bb, xx,yy, sigma) -> lr_model_nll(bb,xx,yy, $\sigma$ =sigma),  $\beta_{\text{test}}$ , x, y,
     $\sigma_{\text{test}}$ )
  hand_derivative = ((-2 * x * y .+ 2* x * x' *  $\beta_{\text{test}}$ )./(2* $\sigma_{\text{test}}$ ^2))[1]
  @test ad_grad[1]  $\approx$  hand_derivative
end
```

```
@testset "Gradients wrt parameter" begin
   $\beta_{\text{test}}$  = randn()  [-0.213...]
   $\sigma_{\text{test}}$  = rand()  [0.124...]
  x,y = sample_batch(target_f1,100)  [( > 1x100 Array{Float64,2}::, > Vector{Float64} with 100 elements)
  ad_grad = gradient( (bb, xx,yy, sigma) -> lr_model_nll(bb,xx,yy, $\sigma$ =sigma),  $\beta_{\text{test}}$ , x, y,  $\sigma_{\text{test}}$ )
  hand_derivative = ((-2 * x * y .+ 2* x * x' *  $\beta_{\text{test}}$ )./(2* $\sigma_{\text{test}}$ ^2))[1]  [-1.84e+6...]
  @test ad_grad[1]  $\approx$  hand_derivative  [Test Passed]
end
```

## 2.5.1 Train Linear Regression Model with Gradient Descent

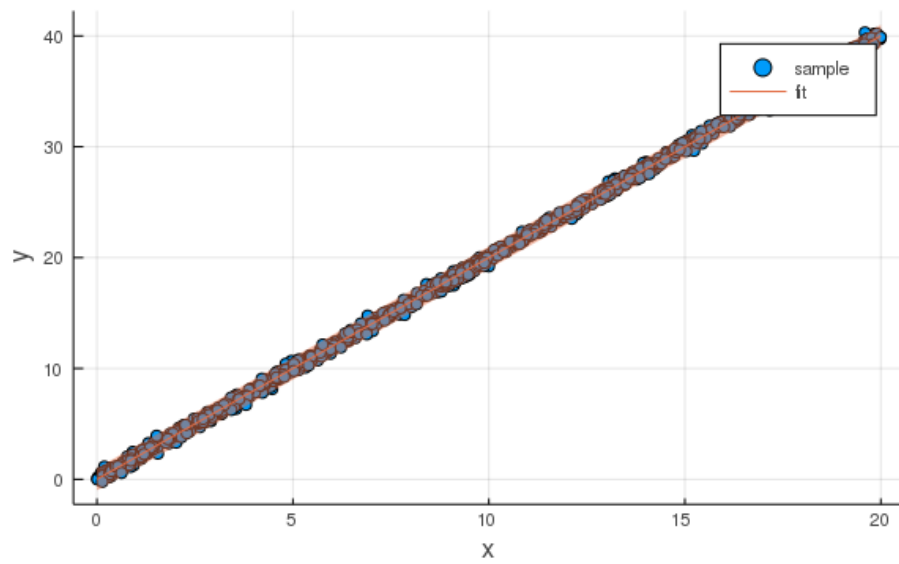
1. Write a function `train_lin_reg` that accepts a target function and an initial estimate for  $\beta$  and some hyperparameters for batch-size, model variance, learning rate, and number of iterations.

```
function train_lin_reg(target_f, _init; bs= 100, lr = 1e-6, iters=1000,  $\sigma_{\text{model}}$  =
    1. )
     $\beta_{\text{curr}}$  =  $\beta_{\text{init}}$ 
    for i in 1:iters
        x,y = sample_batch(target_f,bs)
        grad_ $\beta$  = gradient((bb, xx,yy, sigma) -> lr_model_nll(bb,xx,yy, $\sigma$ =sigma),  $\beta$ 
            _curr, x, y,  $\sigma_{\text{model}}$ )[1]
         $\beta_{\text{curr}}$  =  $\beta_{\text{curr}}$  - grad_ $\beta$  * lr
    end
     $\beta_{\text{curr}}$ 
end
```

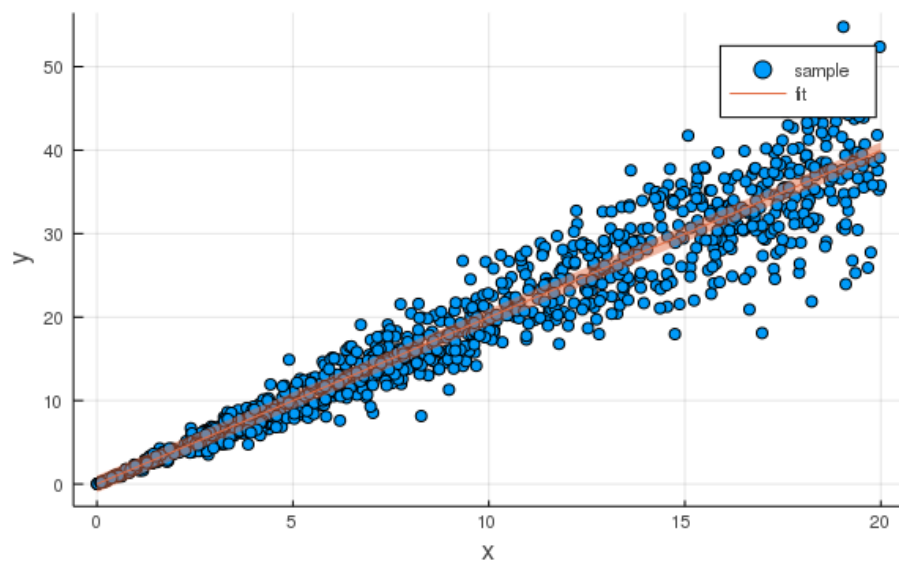
2. For each target function, start with an initial parameter  $\beta$ , learn an estimate for  $\beta$  learned by gradient descent. Then plot a  $n = 1000$  sample of the data and the learned linear regression model with shaded region for uncertainty corresponding to plus/minus one standard deviation.

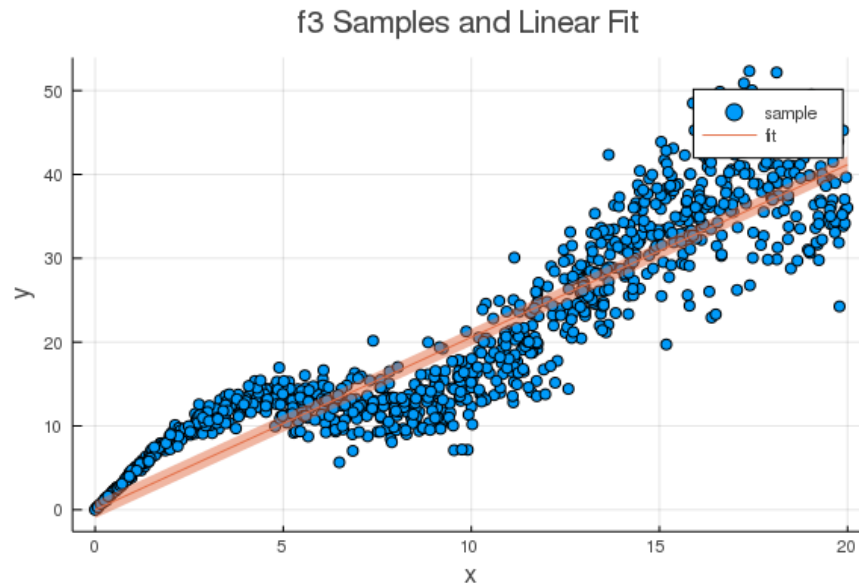
```
 $\beta_{\text{init}}$  = [randn(), randn(), randn()] # Initial parameter
targets = [target_f1, target_f2, target_f3]
 $\beta_{\text{learned}}$  = train_lin_reg.(targets,  $\beta_{\text{init}}$ ; bs= 100, lr = 1e-6, iters=1000,  $\sigma_{\text{model}}$ 
    = 1. )
x=0:20
plot(plot_f1)
plot!(x-> $\beta_{\text{learned}}$ [1]*x, x,
    title="f1 Samples and Linear Fit",
    xlabel="x",
    ylabel="y",
    ribbon=1.0,
    label="fit")
savefig("imgs/2_5_1_2_1.png")
plot(plot_f2)
plot!(x-> $\beta_{\text{learned}}$ [2]*x, x,
    title="f2 Samples and Linear Fit",
    xlabel="x",
    ylabel="y",
    ribbon=1.0,
    label="fit")
savefig("imgs/2_5_1_2_2.png")
plot(plot_f3)
plot!(x-> $\beta_{\text{learned}}$ [3]*x, x,
    title="f3 Samples and Linear Fit",
    xlabel="x",
    ylabel="y",
    ribbon=1.0,
    label="fit")
savefig("imgs/2_5_1_2_3.png")
```

f1 Samples and Linear Fit



f2 Samples and Linear Fit





## 2.5.2 Non-linear Regression with a Neural Network

1. Write the code for a fully-connected neural network (multi-layer perceptron) with one 10-dimensional hidden layer and a tanh nonlinearity. 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
function neural_net(x,θ)
    hidden = 0.5 .* tanh.(θ[1] * x .+ θ[2]) .+ 0.5
    out = θ[3] * (hidden .+ θ[4])
    out[:]
end
```

```
# Neural Network Function
function neural_net(x,θ)
    hidden = tanh.(θ[1] * x .+ θ[2])
    out = θ[3] * (hidden .+ θ[4])
    out[:]
end

n = 100
h = 10

# [ weights1, bias1, weights2, bias2 ]
θ = [randn((h,1)), randn((h,1)), randn(1,h), randn(h,1)]

@testset "neural net mean vector output" begin
    x,y = sample_batch(target_f1,n)
    μ = neural_net(x,θ)
    @test size(μ) == (n,)
end
```



2. 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(theta,x,y;sigma=1)
    mu = neural_net(x,theta)
    sum(-1 .* gaussian_log_likelihood.(mu, sigma, y))
end
```

3. Write a function, `train_nn_reg`, that accepts a target function and an initial estimate for  $\theta$  and some hyperparameters for batch-size, model variance, learning rate, and number of iterations.

```
function train_nn_reg(target_f, theta_init; bs= 200, lr = 1e-5, iters=1000, sigma_model =
    1. )
    # momentum
    b = 0.99
    v = theta_init .* 0
    theta_curr = theta_init
    for i in 1:iters
        x,y = sample_batch(target_f,bs)
        grad_theta = gradient(
            (theta, xx, yy, sigma) -> nn_model_nll(theta,xx,yy,sigma),
            theta_curr, x, y, sigma_model)[1]
        v = b .* v + (1.0 - b) .* grad_theta
        theta_curr = theta_curr - lr .* v
    end
    theta_curr
end
```

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

```
h=10
theta_init = [ 0.001*[randn((h,1)), randn((h,1)), randn(1,h), randn(h,1)],
               0.001*[randn((h,1)), randn((h,1)), randn(1,h), randn(h,1)],
               0.001*[randn((h,1)), randn((h,1)), randn(1,h), randn(h,1)],
               ]

targets = [target_f1, target_f2, target_f3]

theta_learned = train_nn_reg.(targets, theta_init; bs= 300, lr = 1e-5, iters=4000, sigma_model
    = 1.0 )

# plot data samples and learned regression

x = reshape([i for i =0:1:20],1, :)

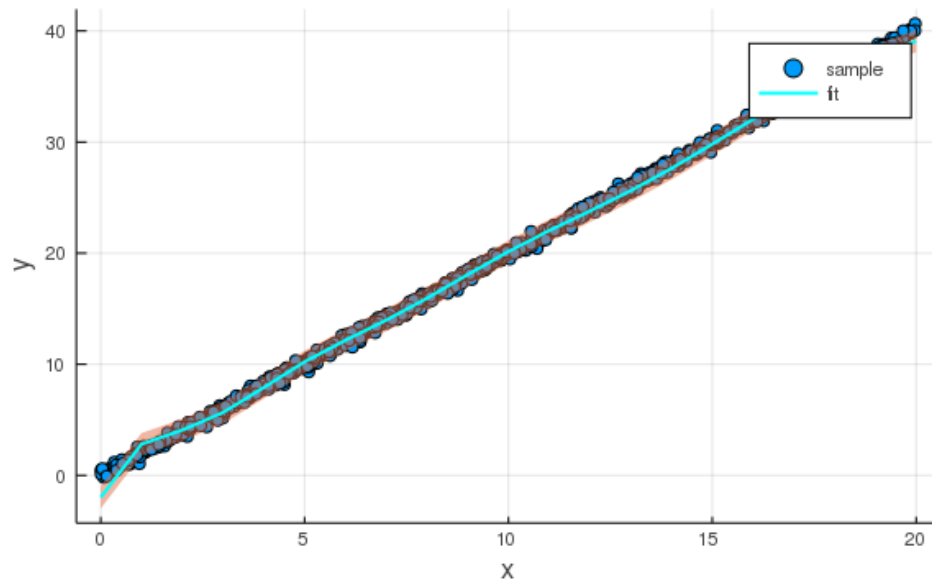
mu = neural_net(x,theta_learned[1])
p1 = plot(x1[1,:],y1,seriestype=:scatter,
    title="f1 Samples and Non-Linear Mean Fit, sigma=1",
    xlabel="x",
    ylabel="y",
```

```
        label="sample")
plot(x[:,mu[:],
      linewidth = 2,
      linecolor = :cyan,
      ribbon=1.0,
      label="fit")
savefig("imgs/2_5_2_4_1.png")

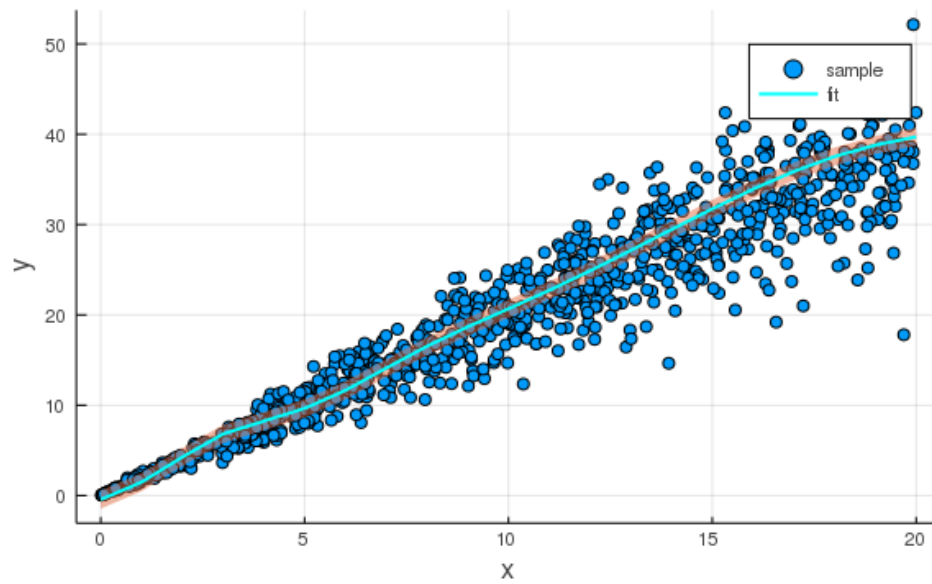
mu = neural_net(x, $\theta$ _learned[2])
plot(x2[1,:],y2,seriestype=:scatter,
      title="f2 Samples and Non-Linear Mean Fit, sigma=1",
      xlabel="x",
      ylabel="y",
      label="sample")
plot(x[:,mu[:],
      linewidth = 2,
      linecolor = :cyan,
      ribbon=1.0,
      label="fit")
savefig("imgs/2_5_2_4_2.png")

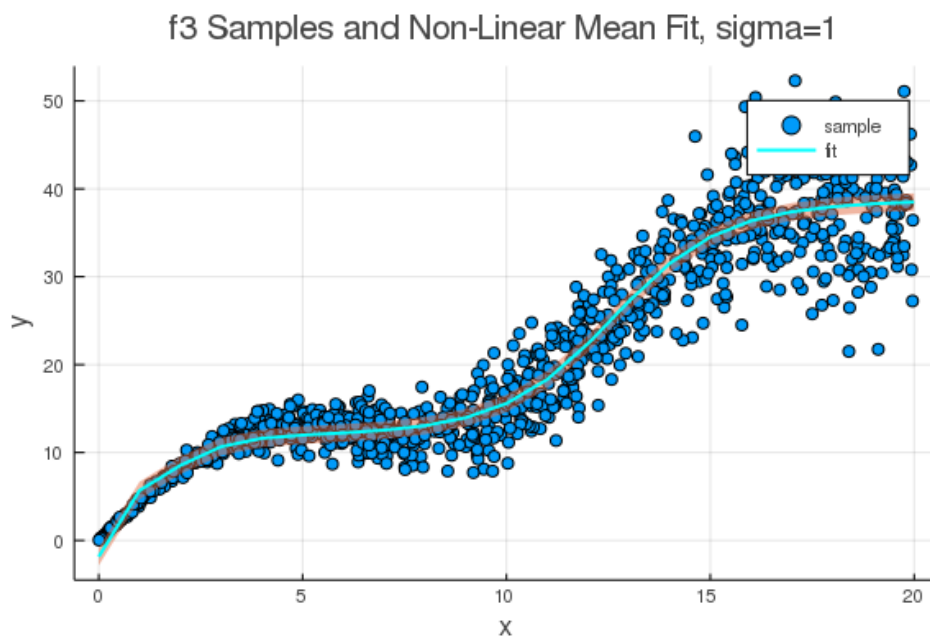
mu = neural_net(x, $\theta$ _learned[3])
plot(x3[1,:],y3,seriestype=:scatter,
      title="f3 Samples and Non-Linear Mean Fit, sigma=1",
      xlabel="x",
      ylabel="y",
      label="sample")
plot(x[:,mu[:],
      linewidth = 2,
      linecolor = :cyan,
      ribbon=1.0,
      label="fit")
savefig("imgs/2_5_2_4_3.png")
```

f1 Samples and Non-Linear Mean Fit, sigma=1



f2 Samples and Non-Linear Mean Fit, sigma=1





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

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

```

function neural_net_w_var(x,  $\theta$ , stat_count, stat_mean, stat_var; training=true)

    hidden_theta_1 =  $\theta$ [1][1] * x .+  $\theta$ [1][2]

    new_count = stat_count + size(x)[2]

    #batch normalization for nodes responsible for mean estimation
    if training == true
        avg1 = sum(hidden_theta_1, dims=2) ./ size(x)[2]
        var1 = sum((hidden_theta_1 .- avg1).^2, dims=2) ./ size(x)[2]
        hidden_theta_1_normalized = (hidden_theta_1 .- avg1) ./ sqrt.(var1 .+ 1e-10)
        hidden_theta_1_activation =  $\theta$ [1][9] * ( $\theta$ [1][5] .* (tanh.(
            hidden_theta_1_normalized .-1.1)) .+  $\theta$ [1][6])

        # println(size(stat_mean[1][1]))
        stat_mean_new_1 = stat_mean[1][1] .* 0.98 .+ avg1 .* 0.02
        stat_var_new_1 = stat_var[1][1] .* 0.98 .+ var1 .* 0.02

        avg12 = sum(hidden_theta_1_activation, dims=2) ./ size(hidden_theta_1_activation
            )[2]
        var12 = sum((hidden_theta_1_activation .- avg12).^2, dims=2) ./ size(
            hidden_theta_1_activation)[2]
        hidden_theta_2_normalized = (hidden_theta_1_activation .- avg12) ./ sqrt.(var12
            .+ 1e-10)
        hidden_theta_2_activation =  $\theta$ [1][7] .* (tanh.(hidden_theta_2_normalized .-1.1))
            .+  $\theta$ [1][8]

        stat_mean_new_12 = stat_mean[1][2] .* 0.98 .+ avg12 .* 0.02
        stat_var_new_12 = stat_var[1][2] .* 0.98 .+ var12 .* 0.02

    else
        hidden_theta_1_normalized = (hidden_theta_1 .- stat_mean[1][1]) ./ sqrt.(
            stat_var[1][1] .+ 1e-10)
        hidden_theta_1_activation =  $\theta$ [1][9] * ( $\theta$ [1][5] .* (tanh.(
            hidden_theta_1_normalized .-1.1)) .+  $\theta$ [1][6])

        hidden_theta_2_normalized = (hidden_theta_1_activation .- stat_mean[1][2]) ./
            sqrt.(stat_var[1][2] .+ 1e-10)
        hidden_theta_2_activation =  $\theta$ [1][7] .* (tanh.(hidden_theta_2_normalized .-1.1))
            .+  $\theta$ [1][8]

        stat_mean_new_1 = stat_mean[1][1]
        stat_var_new_1 = stat_var[1][1]

        stat_mean_new_12 = stat_mean[1][2]
        stat_var_new_12 = stat_var[1][2]
    end

    out_theta =  $\theta$ [1][4] * (hidden_theta_2_activation .+  $\theta$ [1][3])

    hidden_log_variance_1 =  $\theta$ [2][1] * x .+  $\theta$ [2][2]

```

```

#batch normalization for nodes responsible for log variance estimation
if training == true
    avg2 = sum(hidden_log_variance_1, dims=2) ./ size(x)[2]
    var2 = sum((hidden_log_variance_1 .- avg2).^2, dims=2) ./size(x)[2]
    hidden_log_variance_1_normalized = (hidden_log_variance_1 .- avg2) ./ sqrt.(var2
        .+ 1e-10)
    hidden_log_variance_1_activation =  $\theta[2][9]$  * ( $\theta[2][5]$  .* (tanh.(
        hidden_log_variance_1_normalized .-1.1)) .+  $\theta[2][6]$ )

    stat_mean_new_2 = stat_mean[2][1] .* 0.98 .+ avg2 .* 0.02
    stat_var_new_2 = stat_var[2][1] .* 0.98 .+ var2 .* 0.02

    avg22 = sum(hidden_log_variance_1_activation, dims=2) ./ size(
        hidden_log_variance_1_activation)[2]
    var22 = sum((hidden_log_variance_1_activation .- avg2).^2, dims=2) ./size(
        hidden_log_variance_1_activation)[2]
    hidden_log_variance_2_normalized = (hidden_log_variance_1_activation .- avg22)
        ./ sqrt.(var22 .+ 1e-10)
    hidden_log_variance_2_activation =  $\theta[2][7]$  .* (tanh.(
        hidden_log_variance_2_normalized .-1.1)) .+  $\theta[2][8]$ 

    stat_mean_new_22 = stat_mean[2][2] .* 0.98 .+ avg22 .* 0.02
    stat_var_new_22 = stat_var[2][2] .* 0.98 .+ var22 .* 0.02

else
    hidden_log_variance_1_normalized = (hidden_log_variance_1 .- stat_mean[2][1]) ./
        sqrt.(stat_var[2][2] .+ 1e-10)
    hidden_log_variance_1_activation =  $\theta[2][9]$  * ( $\theta[2][5]$  .* (tanh.(
        hidden_log_variance_1_normalized .-1.1)) .+  $\theta[2][6]$ )

    hidden_log_variance_2_normalized = (hidden_log_variance_1_activation .-
        stat_mean[2][2]) ./ sqrt.(stat_var[2][2] .+ 1e-10)
    hidden_log_variance_2_activation =  $\theta[2][7]$  .* (tanh.(
        hidden_log_variance_2_normalized .-1.1)) .+  $\theta[2][8]$ 

    stat_mean_new_2 = stat_mean[2][1]
    stat_var_new_2 = stat_var[2][1]

    stat_mean_new_22 = stat_mean[2][2]
    stat_var_new_22 = stat_var[2][2]
end

out_log_variance =  $\theta[2][4]$  * (hidden_log_variance_2_activation .+  $\theta[2][3]$ )

return (out_theta[:,
    out_log_variance[:,
    new_count,
    [[stat_mean_new_1, stat_mean_new_12], [stat_mean_new_2, stat_mean_new_22]],
    [[stat_var_new_1, stat_var_new_12], [stat_var_new_2, stat_var_new_22]])

end

 $\theta$  = [[randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1), randn(h,1),
    randn(h,1), randn(h,1), randn(h,h)],

```

```

[randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1), randn(h,1),
 randn(h,1), randn(h,1), randn(h,h)]]

h = 10

@testset "neural net mean and logsigma vector output" begin
n = 10
x,y = sample_batch(target_f1,n)
stat_count =  $\theta$ 
stat_mean = [ [zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))] ]
stat_var = [ [zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))] ]
, log, _ = neural_net_w_var(x, $\theta$ , stat_count, stat_mean, stat_var)
@test size() == (n,)
@test size(log) == (n,)
end

```

```

@testset "neural net mean and logsigma vector output" begin
n = 100 [ 100 ]
x,y = sample_batch(target_f1,n) [ (> 1x100 Array{Float64,2}:, > Vector{Float64,1}) ]
stat_count = 0 [ 0 ]
stat_mean = [ [zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))] ] [ 2x100 Array{Float64,1} ]
stat_var = [ [zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))] ] [ 2x100 Array{Float64,1} ]
 $\mu$ , log $\sigma$ , _ = neural_net_w_var(x, $\theta$ , stat_count, stat_mean, stat_var) [ (> Vector{Float64,1}) ]
@test size( $\mu$ ) == (n,) [ Test Passed ]
@test size(log $\sigma$ ) == (n,) [ Test Passed ]
end

```

- 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.

```

function nn_with_var_model_nll( $\theta$ ,x,y, stat_count, stat_mean, stat_var; training=
true)
mu, log_variance, new_count, new_mean, new_var = neural_net_w_var(x, $\theta$ , stat_count,
stat_mean, stat_var; training=true)
sum(-1 .* gaussian_log_likelihood(mu, sqrt.(exp.(log_variance)), y), new_count,
new_mean, new_var)
end

```

3. Write a function 'train\_nn\_w\_var\_reg' that accepts a target function and an initial estimate for  $\theta$  and some hyperparameters for batch-size, learning rate, and number of iterations.

```
function train_nn_w_var_reg(target_f,  $\theta_{init}$ , stat_count, stat_mean, stat_var; bs=
    100, lr = 1e-5, iters=10000)
    # update method: SGD with momentum
    b = 0.97
    v =  $\theta_{init}$  .* 0
     $\theta_{curr}$  =  $\theta_{init}$ 

    final_loss = Inf64

    for i in 1:iters
        x,y = sample_batch(target_f,bs)

        function ff(theta, xx, yy, s_count, s_mean, s_var)
            loss, s_count, s_mean, s_var = nn_with_var_model_nll(theta,xx,yy, s_count,
                s_mean, s_var; training=true)
            stat_count = s_count
            stat_mean = s_mean
            stat_var = s_var

            final_loss = loss

            if i % 100 == 0
                println("iter: ", i, ", loss: ", loss)
            end

            loss
        end

        grad_ $\theta$  = gradient(ff,  $\theta_{curr}$ , x, y, stat_count, stat_mean, stat_var)[1]

        v = b .* v + (1.0 - b) .* grad_ $\theta$ 
         $\theta_{curr}$  =  $\theta_{curr}$  - lr .* v

    end
    ( $\theta_{curr}$ , stat_count, stat_mean, stat_var, final_loss)
end
```



4. For each target function, start with an initialization of the network parameters,  $\theta$ , learn an estimate for  $\theta$  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).

```

h=10

theta_init = 0.001 * [ [randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1),
    , randn(h,1), randn(h,1), randn(h,1), randn(h,h)],
    [randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1),
    randn(h,1), randn(h,1), randn(h,1), randn(h,h)],
    [randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1),
    , randn(h,1), randn(h,1), randn(h,1), randn(h,h)],
    [randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1),
    randn(h,1), randn(h,1), randn(h,1), randn(h,h)],
    [randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1),
    , randn(h,1), randn(h,1), randn(h,1), randn(h,h)],
    [randn((h,1)), randn((h,1)), randn(h,1), randn(1,h), randn(h,1),
    randn(h,1), randn(h,1), randn(h,1), randn(h,h)] ]

targets = [target_f1, target_f2, target_f3]

global stat_mean = [ [zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))],
    [[zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))]],
    [[zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))]], ]

global stat_var = [ [zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))],
    [[zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))]],
    [[zeros((h,1)), zeros((h,1))], [zeros((h,1)), zeros((h,1))]], ]

stat_count = [0, 0, 0]

ret = train_nn_w_var_reg(targets, theta_init, stat_count, stat_mean, stat_var; bs=
    2000, lr = 4e-5, iters=10000)

global theta_learned = map(x->x[1], ret)
global scount = map(x->x[2], ret)
global smean = map(x->x[3], ret)
global svar = map(x->x[4], ret)

# plot data samples and learned regression

support = reshape([i for i =0:0.01:20],1, :)

mu_1, log_variance_1 = neural_net_w_var(support,theta_learned[1], scount[1], smean[1],
    svar[1]; training=false)

plot(x1[1,:],y1,seriestype=:scatter,
    title="f1 Samples and Non-Linear Fit",
    xlabel="x",
    ylabel="y",
    label="sample")

```

```
plot!(support[:,mu_1[:],
        linewidth = 2,
        linecolor = :cyan,
        ribbon=sqrt.(exp.(log_variance_1)),
        label="fit")

savefig("imgs/2_5_3_4_1.png")

mu_2, log_variance_2 = neural_net_w_var(support, $\theta$ _learned[2], scount[2], smean[2],
        svar[2]; training=false)

plot(x2[1,:],y2,seriestype=:scatter,
        title="f2 Samples and Non-Linear Fit",
        xlabel="x",
        ylabel="y",
        label="sample")
plot!(support[:,mu_2[:],
        linewidth = 2,
        linecolor = :cyan,
        ribbon=sqrt.(exp.(log_variance_2)),
        label="fit")

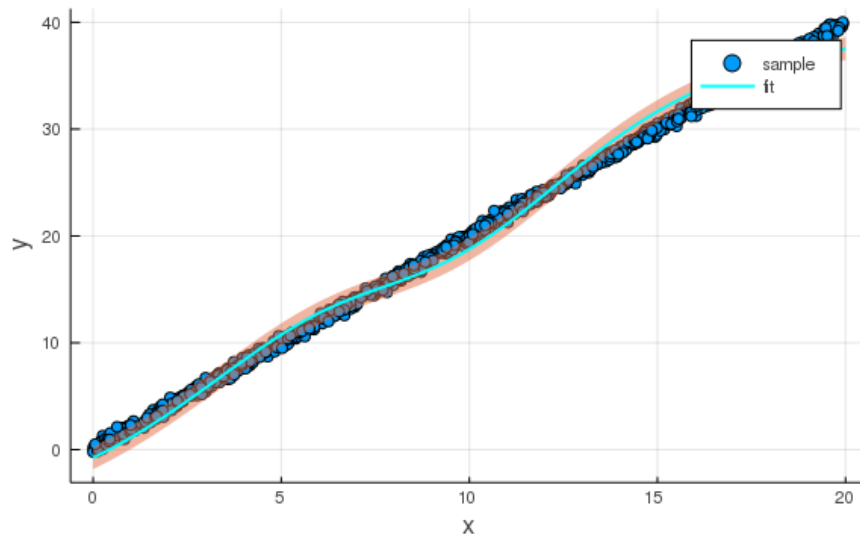
savefig("imgs/2_5_3_4_2.png")

mu_3, log_variance_3 = neural_net_w_var(support, $\theta$ _learned[3], scount[3], smean[3],
        svar[3]; training=false)

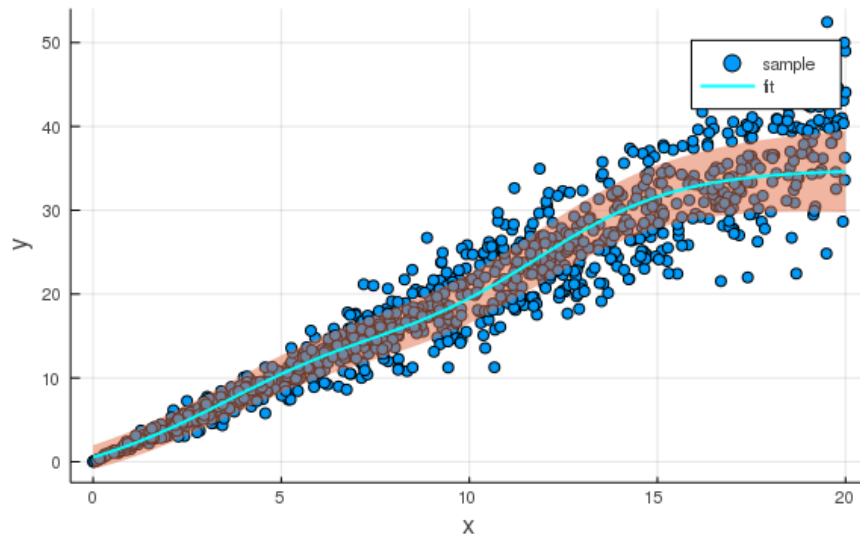
plot(x3[1,:],y3,seriestype=:scatter,
        title="f3 Samples and Non-Linear Fit",
        xlabel="x",
        ylabel="y",
        label="sample")
plot!(support[:,mu_3[:],
        linewidth = 2,
        linecolor = :cyan,
        ribbon=sqrt.(exp.(log_variance_3)),
        label="fit")

savefig("imgs/2_5_3_4_3.png")
```

f1 Samples and Non-Linear Fit



f2 Samples and Non-Linear Fit



f3 Samples and Non-Linear Fit

