Lab5

Jun Rao

9/25/2019

Linear Regression in R and Stan

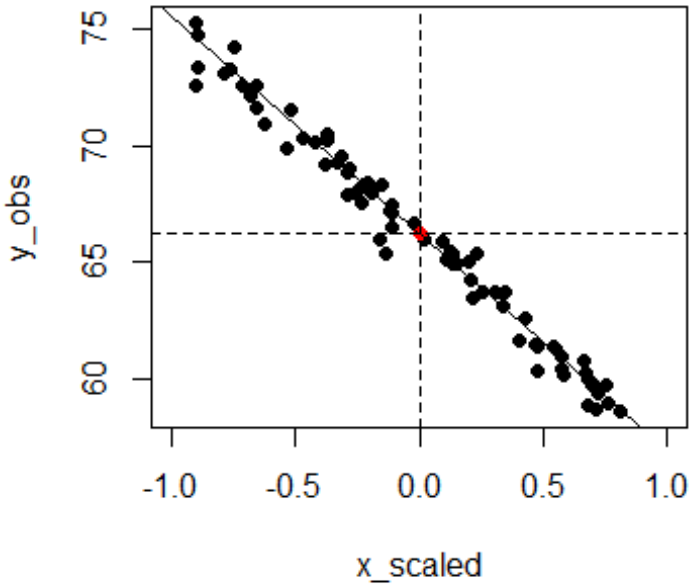
Today we are going to simulate data following the assumptions of linear regression, and we're going to conduct a Bayesian linear regression using Stan.

Simulating the data

Each time you run this code chunk, you'll come up with a new data set and a new linear relationship. However, each time you will still be following the assumptions of linear regression.

```
# Here are the required parameters:
alpha = rnorm(1, 0, 50)
beta = rnorm(1, 0, 50)
sigma = abs(rcauchy(1, 0, 2.5))

# We also need to know how many data points to generate:
n_sample = 75

# What are the values of our X variable?
# Let's assume we're dealing with elevation, measured in meters:
x_raw = runif(n_sample, 0, 3500)

# Now center and standardize:
x_mean = mean(x_raw)
x_sd = sd(x_raw)

x_scaled = (x_raw - x_mean) / (2 * x_sd)

# Create our residuals:
epsilon = rnorm(n_sample, 0, sigma)

# Now we can finally create our data:
y_hat = alpha + x_scaled * beta
y_obs = y_hat + epsilon

# A quick plot:
plot(y_obs ~ x_scaled, type = "p", pch = 19, xlim = c(-1, 1))
points(x = 0, y = alpha, pch = 19, col = "red")
```

```
abline(v = 0, lty = 2)
abline(h = alpha, lty = 2)
abline(a = alpha, b = beta, lty = 1, col = "black")
```



Set up the Stan analysis

```
#---------------------------------------
#---------------------------------------
# Import/Load the rstan library:
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
#---------------------------------------
#---------------------------------------

# Specify the data inputs:


stan_data = list(n_sample = n_sample,
                 x_vec = x_scaled,
                 y_vec = y_obs)

# Now we have to specify which parameters to "monitor"
# in the sampler.

params_monitor = c("beta", "alpha", "sigma")
```

```r
# First, we will test whether our code compiles correctly:
test_fit =
  stan(file = "LinReg.stan",
       data = stan_data,
       pars = params_monitor,
       chains = 1, # How many chains to run
       iter = 10, # How many iterations per chain
       # Algorithm: No-U-Turn Sampler (NUTS)
       # Variant of Hamiltonian Monte Carlo
       algorithm="NUTS")
```

```
##
## SAMPLING FOR MODEL 'LinReg' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would tak
e 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 10 [ 10%]  (Warmup)
## Chain 1: Iteration: 2 / 10 [ 20%]  (Warmup)
## Chain 1: Iteration: 3 / 10 [ 30%]  (Warmup)
## Chain 1: Iteration: 4 / 10 [ 40%]  (Warmup)
## Chain 1: Iteration: 5 / 10 [ 50%]  (Warmup)
## Chain 1: Iteration: 6 / 10 [ 60%]  (Sampling)
## Chain 1: Iteration: 7 / 10 [ 70%]  (Sampling)
## Chain 1: Iteration: 8 / 10 [ 80%]  (Sampling)
## Chain 1: Iteration: 9 / 10 [ 90%]  (Sampling)
## Chain 1: Iteration: 10 / 10 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0.001 seconds (Total)
## Chain 1:
```

```r
## Now we will run our full model:

# How many samples do we want of each parameter, from each chain?
n_mc_samples = 1000
# How much burn-in?
n_burn = 500
# How much thinning? (take the ith value of the chain)
n_thin = 3
# Total iterations needed:
n_iter_total = (n_mc_samples * n_thin) + n_burn
```

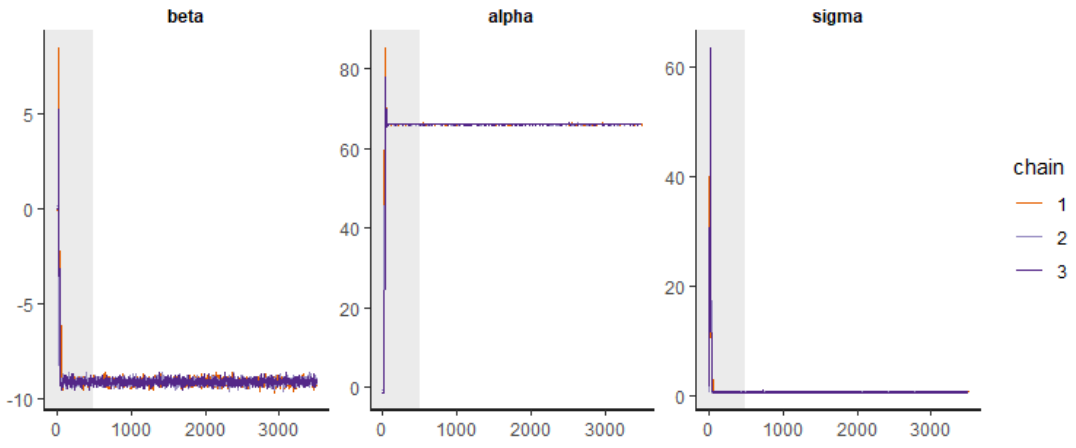
```
model_fit =
  stan(fit = test_fit, # So it knows we're already compiled
       file = "LinReg.stan",
       data = stan_data,
       pars = params_monitor,
       chains = 3,
       warmup = n_burn,
       thin = n_thin,
       iter = n_iter_total,
       algorithm="NUTS")
```

Check the output of the sampler

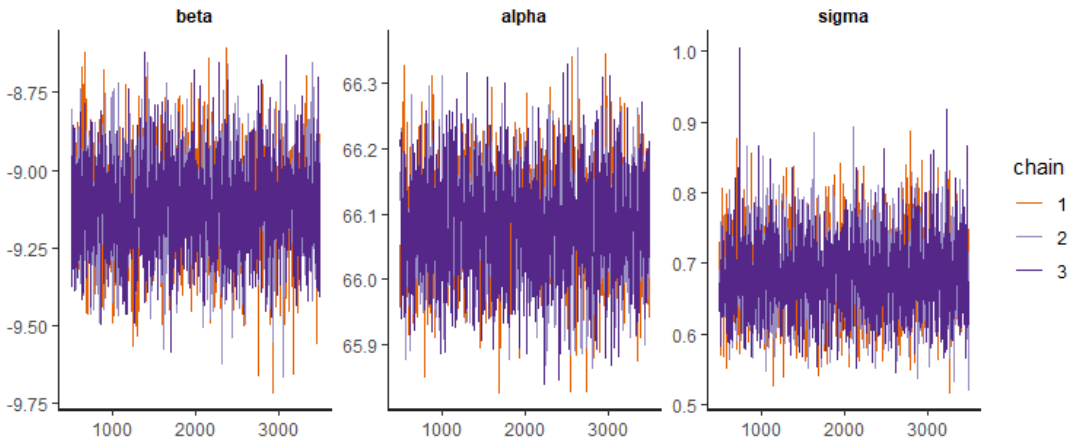
```
model_out = rstan::extract(model_fit)
str(model_out)
```

```
## List of 4
##  $ beta : num [1:3000(1d)] -9.23 -8.96 -8.98 -9.06 -9.15 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##  $ alpha: num [1:3000(1d)] 66 66.1 65.9 66.1 66.1 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##  $ sigma: num [1:3000(1d)] 0.714 0.676 0.721 0.619 0.624 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##  $ lp__ : num [1:3000(1d)] -90.3 -89.1 -91.6 -89.1 -88.8 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
```

```
# Traceplot with burn-in:
rstan::traceplot(model_fit, pars = params_monitor, inc_warmup = TRUE)
```




```
# Traceplot without burn-in:
rstan::traceplot(model_fit, pars = params_monitor, inc_warmup = FALSE)
```




```
# Summary Report:
# Note the 2.5 and 97.5 quantiles give you the 95% credible interval
# Are are "true" values within these bounds?
# "lp__" represents log-posterior (the joint posterior),
#  up to a constant of proportionality
summary(model_fit)$summary
```

```
##              mean     se_mean         sd        2.5%         25%
## beta   -9.1132786 0.002799411 0.15633242  -9.4181188  -9.2206557
## alpha  66.0820790 0.001532737 0.07946263  65.9314507  66.0296513
## sigma   0.6810568 0.001102266 0.05717647   0.5797594   0.6413683
## lp__  -89.9539286 0.024589208 1.19589665 -92.9313924 -90.5368861
##               50%         75%       97.5%    n_eff      Rhat
## beta   -9.1125286  -9.0083770  -8.8025733 3118.637 0.9997222
## alpha  66.0804417  66.1355948  66.2377625 2687.762 1.0001642
## sigma   0.6780642   0.7185916   0.8007651 2690.677 0.9993470
## lp__  -89.6531030 -89.0809488 -88.5687958 2365.365 1.0001323
```

```
# Check Convergence:
summary(model_fit)$summary[, "Rhat"]
```

```
##      beta     alpha     sigma      lp__
## 0.9997222 1.0001642 0.9993470 1.0001323
```

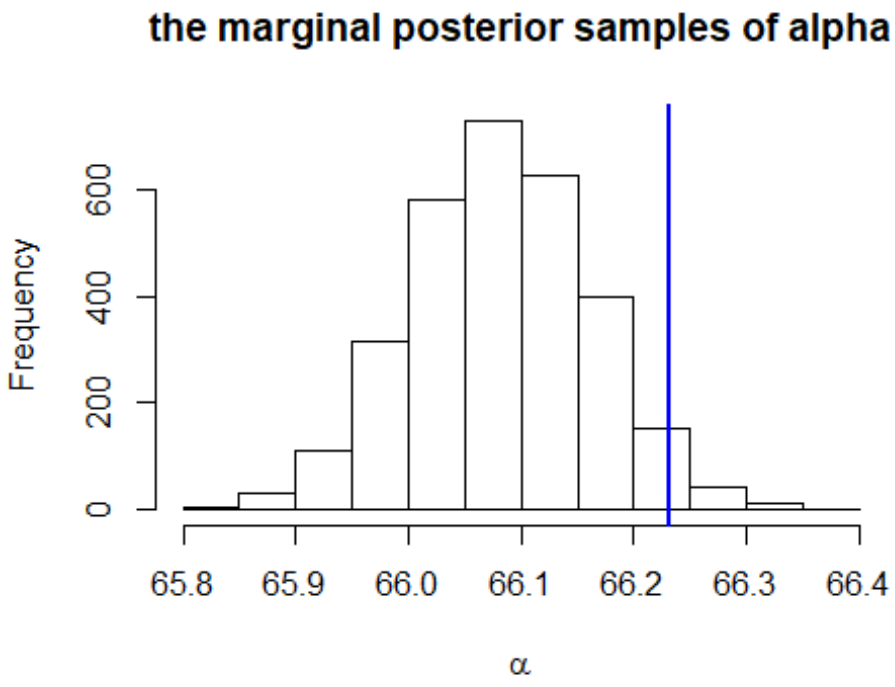
```
# Note that our n_eff < n_iter_total
```

Task 1 (5 points)

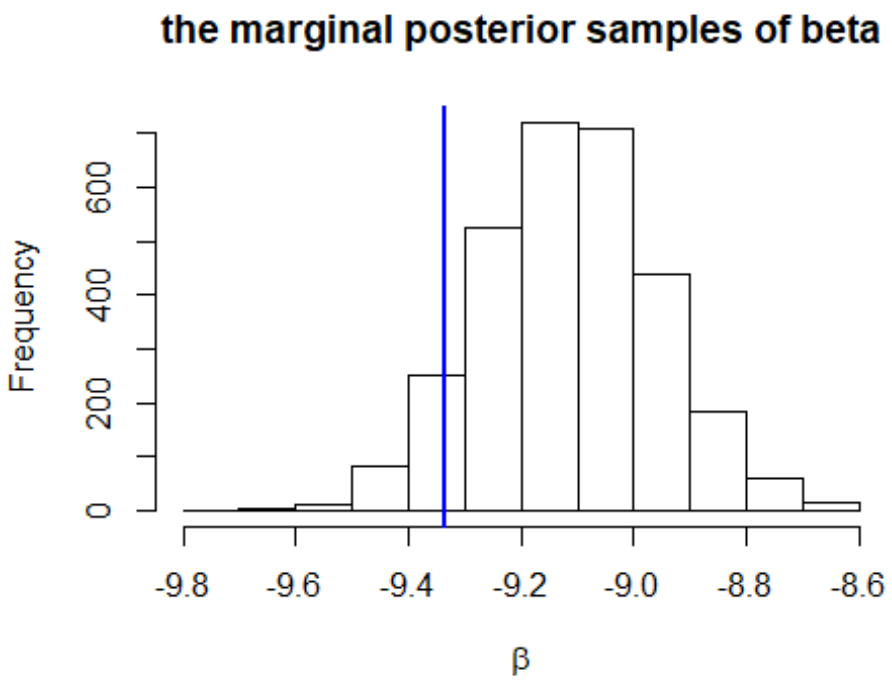
For each parameter:

1. Plot a histogram of the marginal posterior samples from the Stan output.
2. Use the () function to draw a vertical line that represents the value of each parameter.

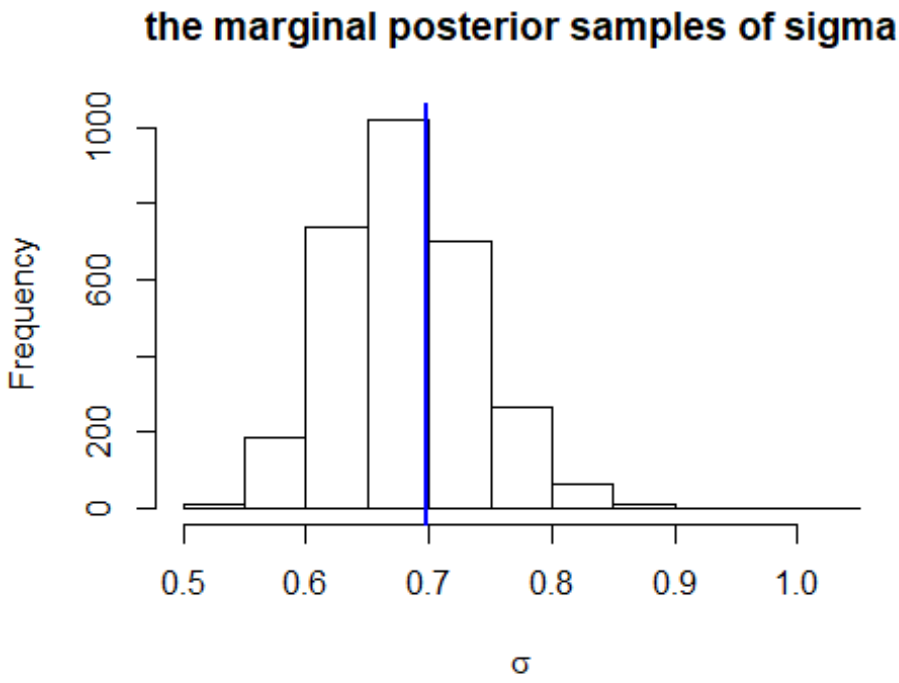
```
hist(model_out$alpha,main="the marginal posterior samples of alpha",xlab = ex
pression(alpha))
abline(v = alpha, lty = 1, lwd = 2, col = "blue")
```




```
hist(model_out$beta,main="the marginal posterior samples of beta",xlab = expr
ession(beta))
abline(v = beta, lty = 1, lwd = 2, col = "blue")
```



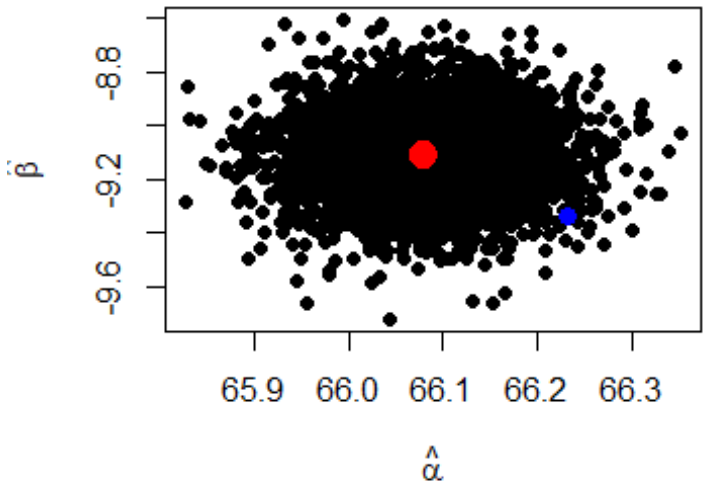

```r
hist(model_out$sigma,main="the marginal posterior samples of sigma",xlab = expression(sigma))
abline(v = sigma, lty = 1, lwd = 2, col = "blue")
```



3. Think about what this tells you. The blue verrTical line in each histogram is very close to the center.The generated posterior value based on the give value is very close to the true vale and acceptable.

The joint marginal

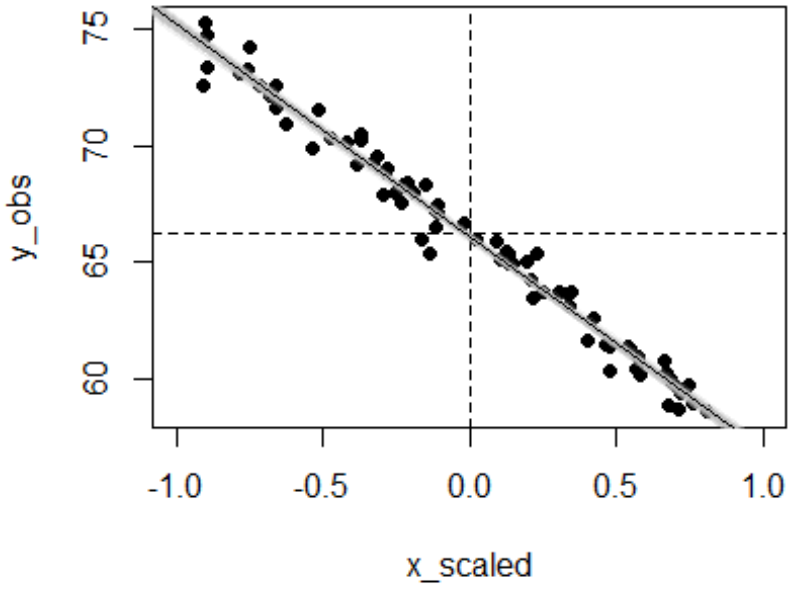
```
# Plot the joint marginal samples of slope and intercept:

beta_est = model_out$beta
alpha_est = model_out$alpha
sigma_est = model_out$sigma

plot(beta_est ~ alpha_est, type = "p", pch = 19,
     xlab = expression(hat(alpha)), ylab = expression(hat(beta)))
points(x = median(alpha_est), y = median(beta_est), col = "red", pch = 19, ce
x = 2)
# Does this match the true?
points(x = alpha, y = beta, col = "blue", pch = 19, cex = 1.25)
```



Visualizing the estimated model fit, with error

```
plot(y_obs ~ x_scaled, type = "p", pch = 19, xlim = c(-1, 1))
abline(v = 0, lty = 2)
abline(h = alpha, lty = 2)

# How many estimated lines to plot?
n_lines = 100

for(i in 1:n_lines){
  this_sample = sample(c(1:length(alpha_est)), size = 1)
```

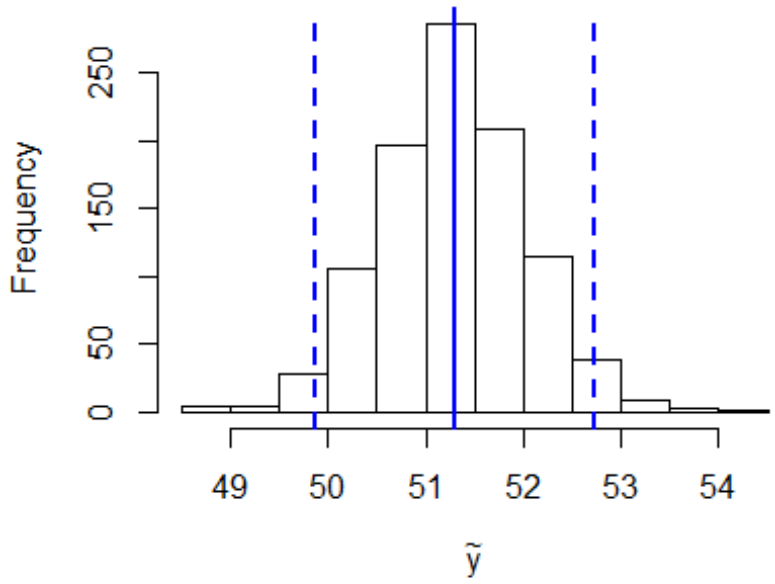
```
  alpha_temp = alpha_est[this_sample]
  beta_temp = beta_est[this_sample]

  abline(a = alpha_temp, b = beta_temp, col = scales::alpha("gray", alpha = 0
.2))


}

# Add the median estimate:
abline(a = median(alpha_est), b = median(beta_est), col = "black")
```



Predicting outcome ($\tilde{y}$) at unobserved values of input

```
# A future observation of X, outside the original range of X (forecasting)
x_future = 5000
# Scale it:
x_future_scaled = (x_future - x_mean) / (2 * x_sd)

# Now, generate possible values of unobserved y, given our observed y
n_pred = 1000
y_pred = NULL
for(i in 1:n_pred){

  this_sample = sample(c(1:length(alpha_est)), size = 1)
```

```
  alpha_temp = alpha_est[this_sample]
  beta_temp = beta_est[this_sample]
  sigma_temp = sigma_est[this_sample]

  y_pred[i] = alpha_temp + beta_temp * x_future_scaled + rnorm(1, 0, sigma_te
mp)

}

hist(y_pred, ylab = "Frequency", xlab = expression(tilde(y)), main = "")
abline(v = median(y_pred), lty = 1, lwd = 2, col = "blue")
abline(v = quantile(y_pred, probs = c(0.025, 0.975)), lty = 2, lwd = 2, col =
"blue")
```

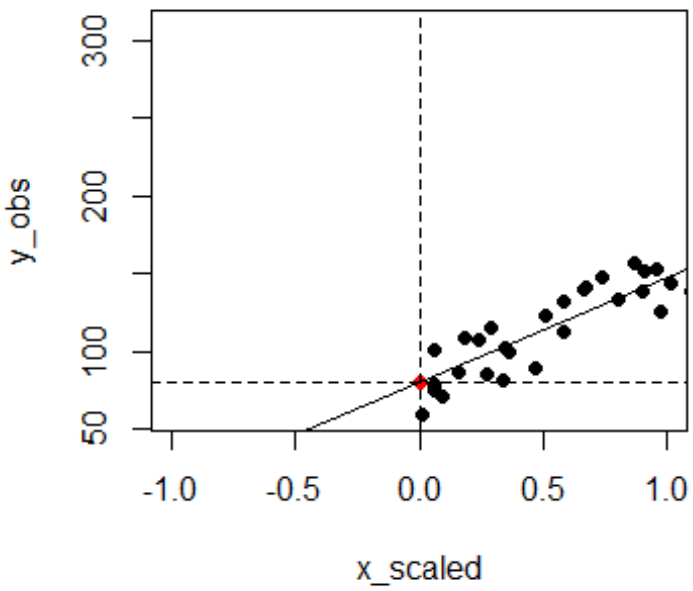


Task 2 (10 points)

1. Re-code your data simulation so that you are not centering your input variable.

2. Instead of standardizing by dividing by 2 standard deviations, standardize by dividing the input variable by 1000.

```
# Here are the required parameters:
alpha = rnorm(1, 0, 50)
beta = rnorm(1, 0, 50)
sigma = abs(rcauchy(1, 0, 2.5))

# We also need to know how many data points to generate:
n_sample = 75

# What are the values of our X variable?
# Let's assume we're dealing with elevation, measured in meters:
x_raw = runif(n_sample, 0, 3500)

# Now center and standardize:
x_mean = mean(x_raw)
x_sd = sd(x_raw)

x_scaled = x_raw / 1000

# Create our residuals:
epsilon = rnorm(n_sample, 0, sigma)

# Now we can finally create our data:
y_hat = alpha + x_scaled * beta
y_obs = y_hat + epsilon

# A quick plot:
plot(y_obs ~ x_scaled, type = "p", pch = 19, xlim = c(-1, 1))
points(x = 0, y = alpha, pch = 19, col = "red")
abline(v = 0, lty = 2)
abline(h = alpha, lty = 2)
abline(a = alpha, b = beta, lty = 1, col = "black")
```



3. Re-run your Bayesian analysis on this new data set. ## Set up the Stan analysis

```
#------------------------------------
#------------------------------------
# Import/Load the rstan library:
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
#------------------------------------
#------------------------------------

# Specify the data inputs:


stan_data = list(n_sample = n_sample,
                 x_vec = x_scaled,
                 y_vec = y_obs)

# Now we have to specify which parameters to "monitor"
# in the sampler.

params_monitor = c("beta", "alpha", "sigma")

# First, we will test whether our code compiles correctly:
test_fit =
  stan(file = "LinReg.stan",
       data = stan_data,
```

```
       pars = params_monitor,
       chains = 1, # How many chains to run
       iter = 10, # How many iterations per chain
       # Algorithm: No-U-Turn Sampler (NUTS)
       # Variant of Hamiltonian Monte Carlo
       algorithm="NUTS")

##
## SAMPLING FOR MODEL 'LinReg' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would tak
e 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 10 [ 10%]  (Warmup)
## Chain 1: Iteration: 2 / 10 [ 20%]  (Warmup)
## Chain 1: Iteration: 3 / 10 [ 30%]  (Warmup)
## Chain 1: Iteration: 4 / 10 [ 40%]  (Warmup)
## Chain 1: Iteration: 5 / 10 [ 50%]  (Warmup)
## Chain 1: Iteration: 6 / 10 [ 60%]  (Sampling)
## Chain 1: Iteration: 7 / 10 [ 70%]  (Sampling)
## Chain 1: Iteration: 8 / 10 [ 80%]  (Sampling)
## Chain 1: Iteration: 9 / 10 [ 90%]  (Sampling)
## Chain 1: Iteration: 10 / 10 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0 seconds (Total)
## Chain 1:

## Now we will run our full model:

# How many samples do we want of each parameter, from each chain?
n_mc_samples = 1000
# How much burn-in?
n_burn = 500
# How much thinning? (take the ith value of the chain)
n_thin = 3
# Total iterations needed:
n_iter_total = (n_mc_samples * n_thin) + n_burn

model_fit =
  stan(fit = test_fit, # So it knows we're already compiled
       file = "LinReg.stan",
       data = stan_data,
```

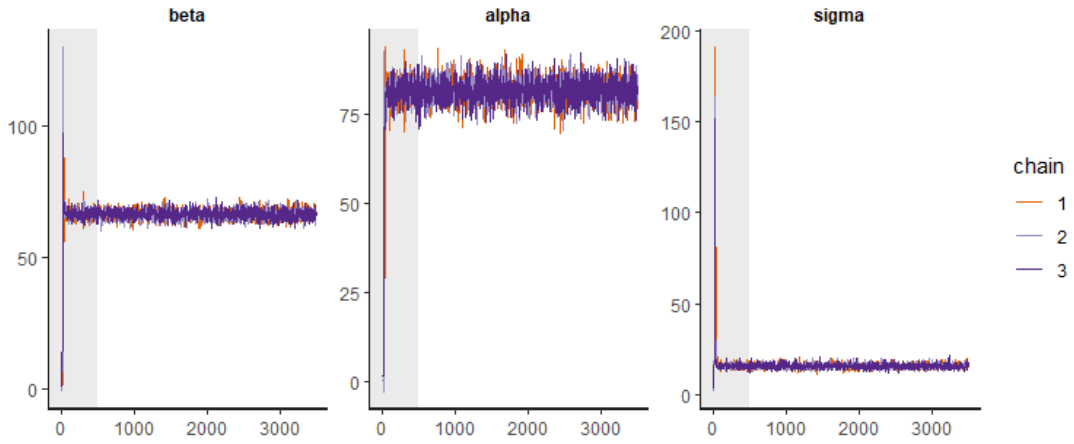
```
        pars = params_monitor,
        chains = 3,
        warmup = n_burn,
        thin = n_thin,
        iter = n_iter_total,
        algorithm="NUTS")
```

Check the output of the sampler

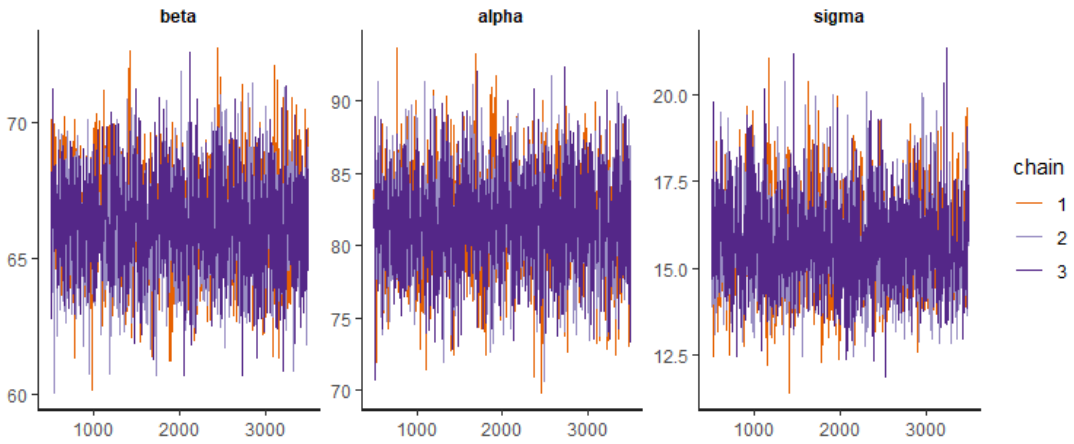
```
model_out = rstan::extract(model_fit)
str(model_out)
```

```
## List of 4
##  $ beta : num [1:3000(1d)] 69.6 66.2 62.4 65.5 70.8 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##  $ alpha: num [1:3000(1d)] 75.7 81 86.2 85.5 75.8 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##  $ sigma: num [1:3000(1d)] 15.2 16 15 14.9 16.1 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##  $ lp__ : num [1:3000(1d)] -328 -326 -329 -327 -329 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
```

```
# Traceplot with burn-in:
rstan::traceplot(model_fit, pars = params_monitor, inc_warmup = TRUE)
```




```
# Traceplot without burn-in:
rstan::traceplot(model_fit, pars = params_monitor, inc_warmup = FALSE)
```




```
# Note that our n_eff < n_iter_total
```

4. Look at the number of effective samples in each chain. How does this compare to when you properly centered and scaled?

```
# Summary Report:
# Note the 2.5 and 97.5 quantiles give you the 95% credible interval
# Are are "true" values within these bounds?
# "lp__" represents log-posterior (the joint posterior),
#  up to a constant of proportionality
summary(model_fit)$summary
```

```
##             mean    se_mean        sd       2.5%        25%        50%
## beta    66.26997 0.03661323 1.856145   62.75135   64.96848   66.25827
## alpha   81.73594 0.06742776 3.406587   74.89169   79.47588   81.82035
## sigma   15.64319 0.02561276 1.334347   13.32900   14.72965   15.56593
## lp__  -327.78359 0.02635896 1.303031 -331.21088 -328.38193 -327.43139
##              75%      97.5%    n_eff     Rhat
## beta    67.48743   69.86583 2570.085 1.002180
## alpha   84.04939   88.35113 2552.476 1.001703
## sigma   16.43991   18.59381 2714.095 1.000372
## lp__  -326.82729 -326.32268 2443.728 1.001116
```

```
# Check Convergence:
summary(model_fit)$summary[, "Rhat"]
```

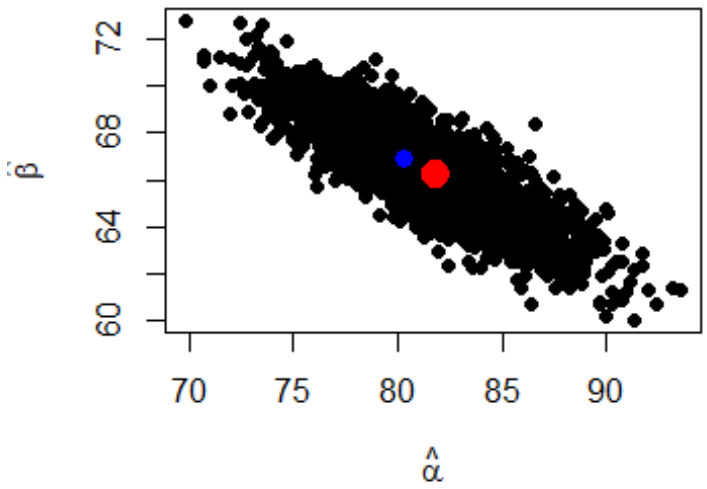
```
##     beta    alpha    sigma     lp__
## 1.002180 1.001703 1.000372 1.001116
```

```
# Note that our n_eff < n_iter_total
```

5. Plot the joint marginal posterior samples of α and β. What differences do you observed compared to when we properly centered and scaled the input? ## The joint marginal

```
# Plot the joint marginal samples of slope and intercept:
```

```r
beta_est = model_out$beta
alpha_est = model_out$alpha
sigma_est = model_out$sigma

plot(beta_est ~ alpha_est, type = "p", pch = 19,
     xlab = expression(hat(alpha)), ylab = expression(hat(beta)))
points(x = median(alpha_est), y = median(beta_est), col = "red", pch = 19, ce
x = 2)
# Does this match the true?
points(x = alpha, y = beta, col = "blue", pch = 19, cex = 1.25)
```



When we didn't center and scale the input, the joint marginal posterior samples of alpha and beta are different. The graphic od the paired alpha-beta in the question 1 is a circle, however, at here is a ellipsoid. It looks like alpha has a hihger effect on posterior.