Bayesian Linear Regression

July 2024

Importing the data

We begin by loading the required packages:

```
library(rstan)
library(HDInterval)
library(bayesplot)
library(ggplot2)
library(truncnorm)
```

We now import the data files: after storing the path to the working directory (WD) in the `work_dir` object, we can set the WD and load the auxiliary functions:

```
setwd(work_dir)
```

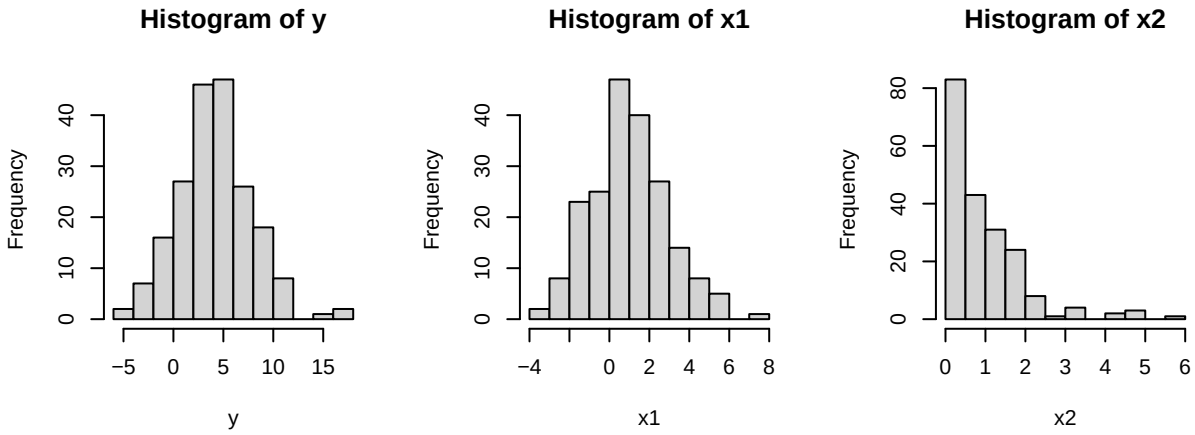
```
load("linear_regression_data.RData")
```

```
source("aux_functions.R")
```

Data Preparation

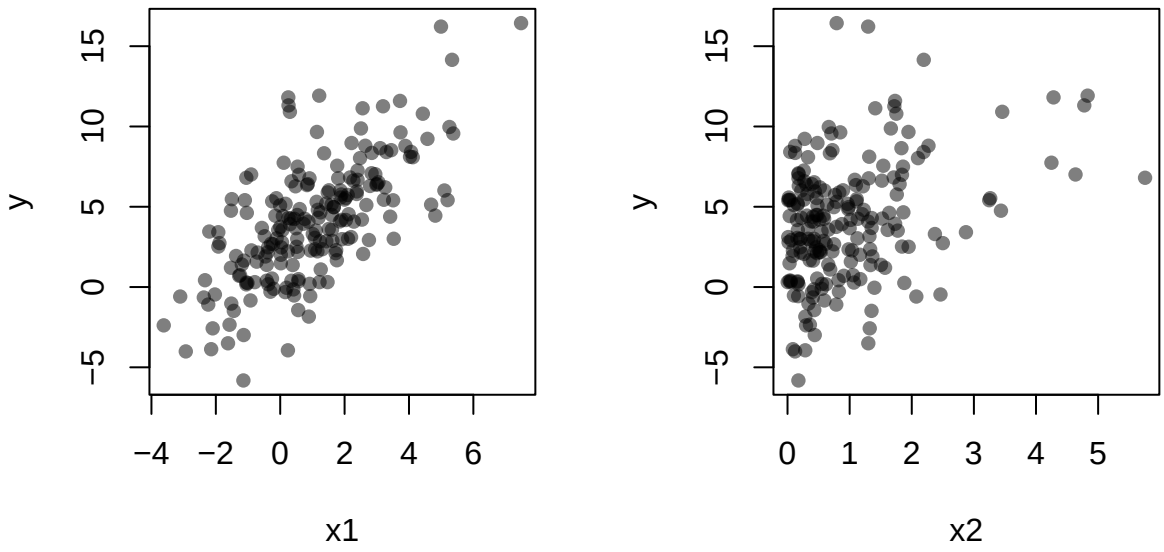
Let us begin by considering an N-sized outcome variable $\mathbf{y}$ and two N-dimensional numeric regressors $\mathbf{x}_1$ and $\mathbf{x}_2$:

```
par(mfrow=c(1,3), pty="s")
hist(y)
hist(x1)
hist(x2)
```



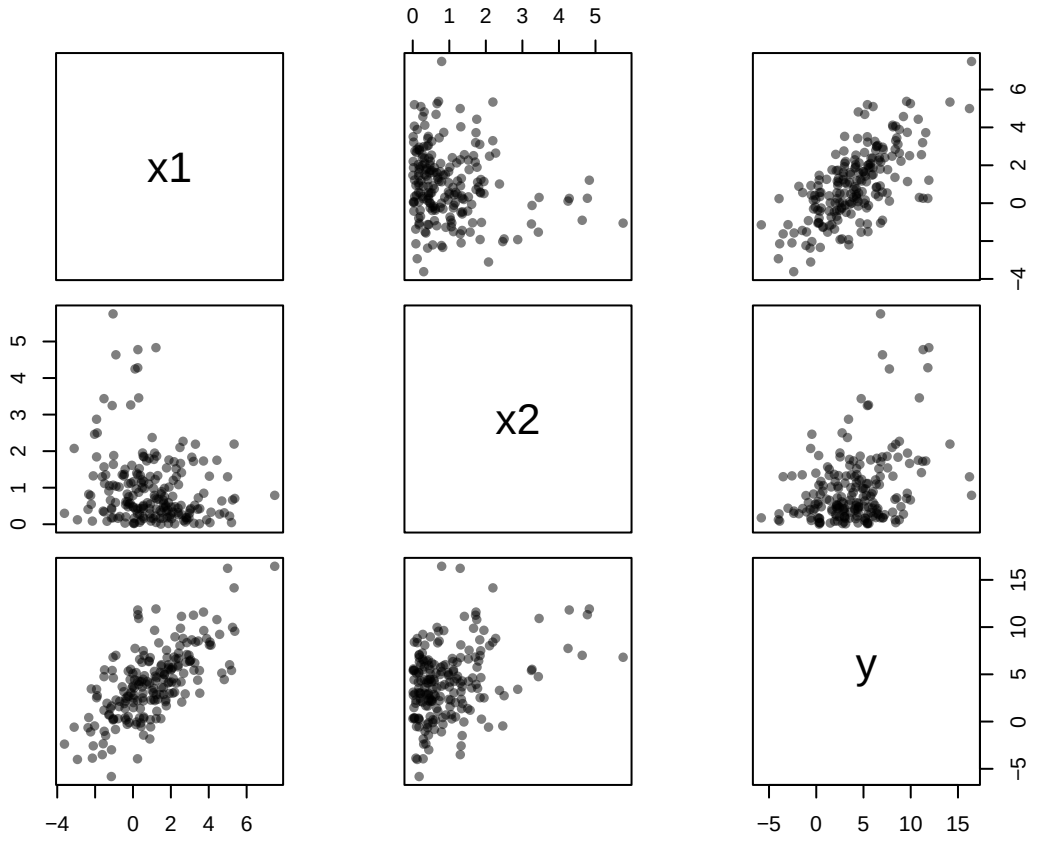

```
par(mfrow=c(1,1), pty="m")

par(mfrow=c(1,2), pty="s")
plot(x1, y, pch = 16, col = rgb(0,0,0,.5))
plot(x2, y, pch = 16, col = rgb(0,0,0,.5))
```




```
par(mfrow=c(1,1), pty="m")

par(pty="s")
pairs(cbind(x1,x2,y), pch = 16, col = rgb(0,0,0,.5))
```




```
par(pty="m")
```

Prior calibration via Prior Predictive Checks

Our goal is to estimate a linear regression model:

$$y_i \sim \mathrm{N}(\mu_i, \sigma^2)$$

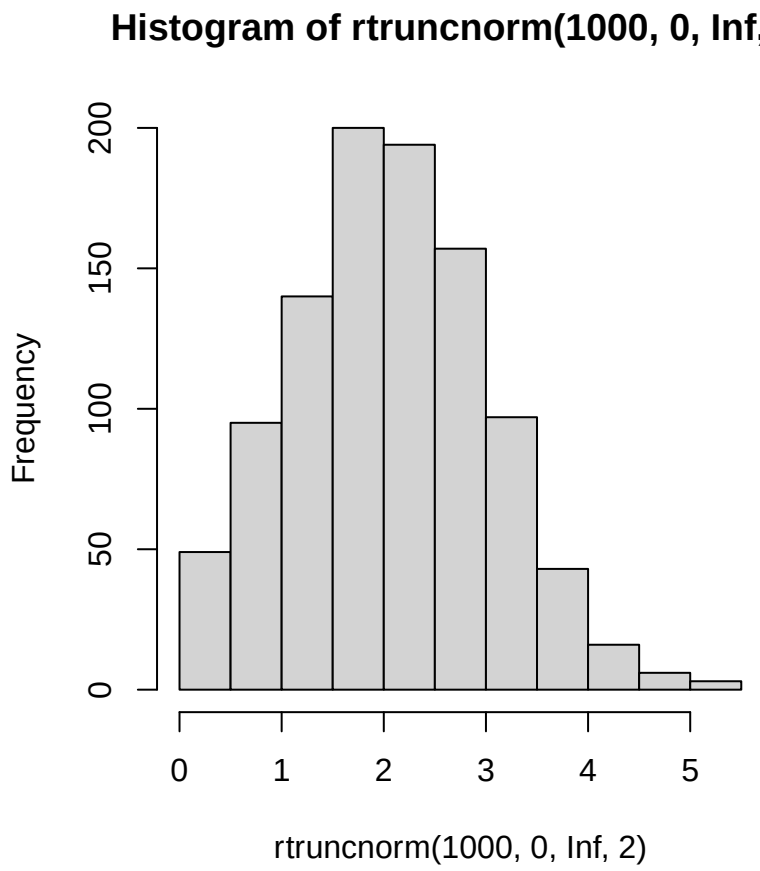
where:

$$\mu_i = \alpha + \beta_1 x_{i,1} + \beta_2 x_{i,2}$$

Therefore, we first need to set priors for α, β_1, β_2 and σ. We begin by studying how choosing a distribution for σ affects the distribution of $y_i \in \mathbf{y}$. To do this, we first randomly draw values of σ from a positive probability density function such as a truncated-Normal or Exponential distribution:

$$\sigma^s \sim \mathrm{N}_+(0, 2)$$

```
par(pty="s")
hist(rtruncnorm(1000, 0, Inf, 2))
```




```
par(pty="m")
```

Next, for each simulated parameter σ^s we generate y_i^s using a Normal distribution with, for the time being, zero mean:

$$\begin{aligned}
\sigma^s &\sim \mathrm{N}_+(0,1) \\
\tilde{y}_1^s &\sim \mathrm{N}(0,\sigma^s) \\
&\vdots \\
\tilde{y}_n^s &\sim \mathrm{N}(0,\sigma^s)
\end{aligned}$$

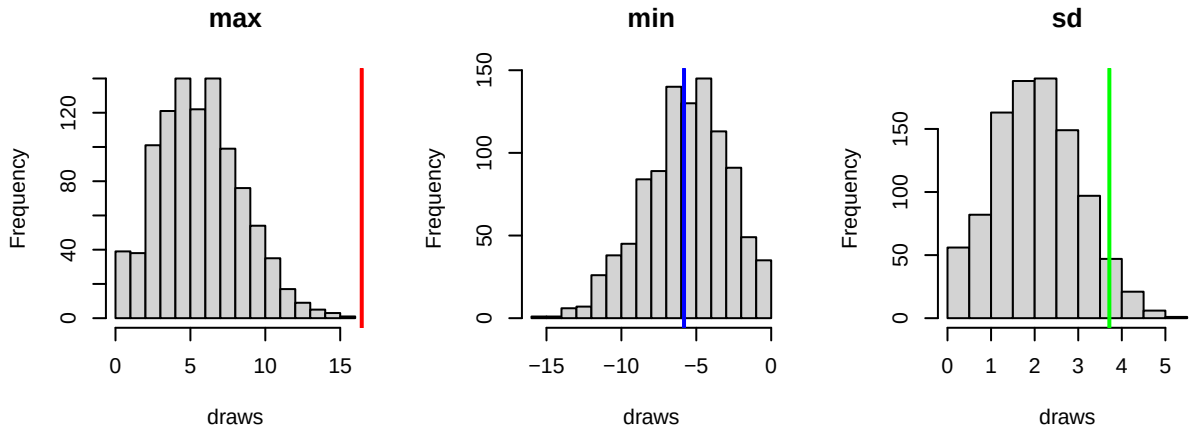
At the end, we obtain S simulation from the Prior Predictive Distribution (PrPD) of $\mathbf{y}$, which we indicate as $\tilde{\mathbf{y}} = \{\tilde{\mathbf{y}}^1, \ldots, \tilde{\mathbf{y}}^S\}$. To understand whether these simulated values look reasonable in face of our observed data, $\mathbf{y}$, we calculate $\min(\tilde{\mathbf{y}}^s)$, $\max(\tilde{\mathbf{y}}^s)$ and $sd(\tilde{\mathbf{y}}^s)$ for all $s \in \{1, \ldots, S\}$ and check whether these are consistent with $\min(\mathbf{y})$, $\max(\mathbf{y})$ and $sd(\mathbf{y})$:

```
set.seed(123)
y_sim <- matrix(nrow=1000, ncol=200)
for (s in 1:1000) {

  std_dev <- rtruncnorm(1, 0, Inf, 2)
```

```
  y_sim[s, ] <- rnorm(200, 0, std_dev)

}
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```




```
par(mfrow=c(1,1), pty="m")
```

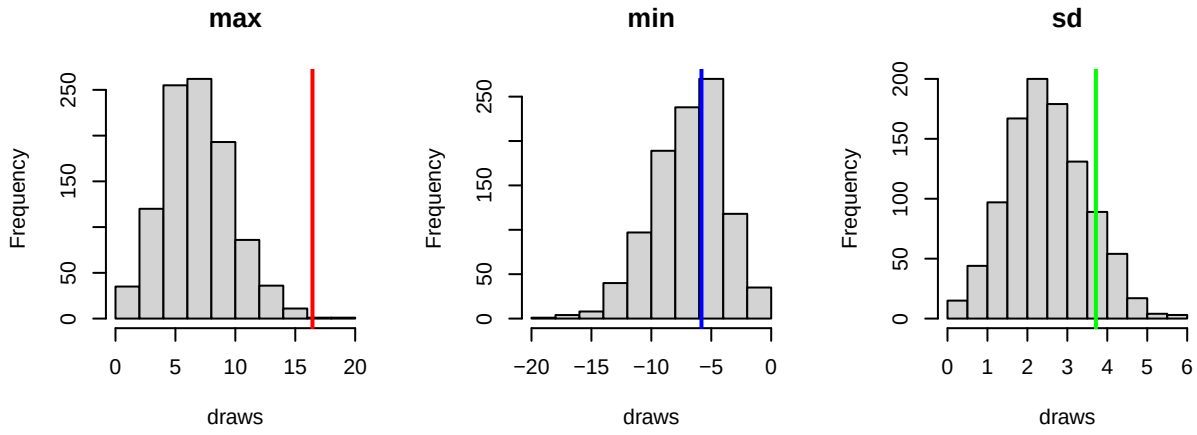
What emerges from these plots, where the vertical lines represent $\min(\mathbf{y})$, $\max(\mathbf{y})$ and $sd(\mathbf{y})$, is that $\tilde{\mathbf{y}}$ does not adequately cover $\mathbf{y}$. Indeed, the simulated maximum values and the simulated standard deviation are too small and too large, respectively, if compared to the calculated statistics for $\mathbf{y}$. We can try to do better by, say, changing the prior on σ to:

$$\sigma^s \sim \mathrm{N}_+(0, 2.5)$$

```r
set.seed(123)
y_sim <- matrix(nrow=1000, ncol=200)
for (s in 1:1000) {

  std_dev <- rtruncnorm(1, 0, Inf, 2.5)
  y_sim[s, ] <- rnorm(200, 0, std_dev)

}
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```



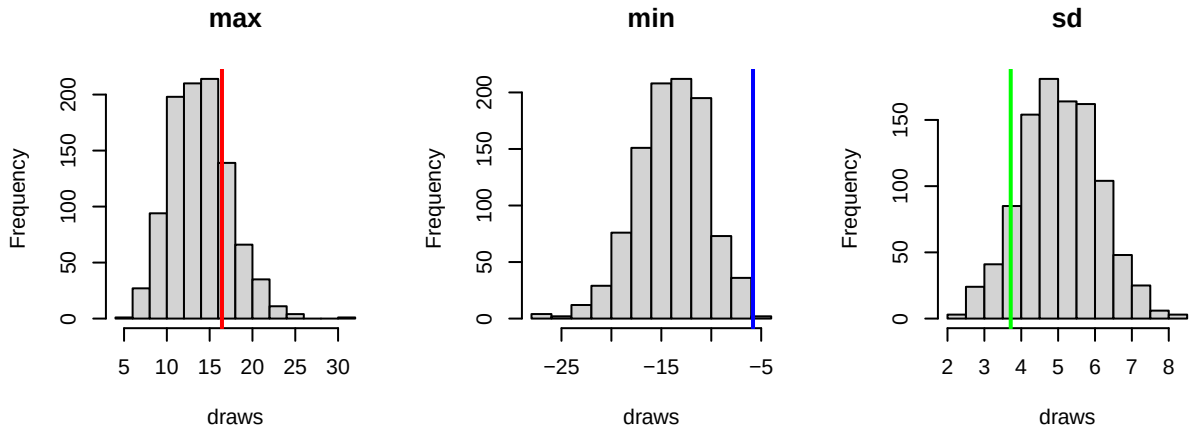

```r
par(mfrow=c(1,1), pty="m")
```

Unfortunately, this still leads to unsatisfactory simulations. What if we try with $\sigma^s \sim \mathrm{N}_+(0, 5)$?

```r
set.seed(123)
y_sim <- matrix(nrow=1000, ncol=200)
for (s in 1:1000) {

  std_dev <- rtruncnorm(1, 0, Inf, 5)
  y_sim[s, ] <- rnorm(200, 0, std_dev)

}
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```




```r
par(mfrow=c(1,1), pty="m")
```

The situation improved for $\max(\mathbf{y})$, but we get worst results for both $\min(\mathbf{y})$ and $sd(\mathbf{y})$. How can we fix this?

Calibrating α

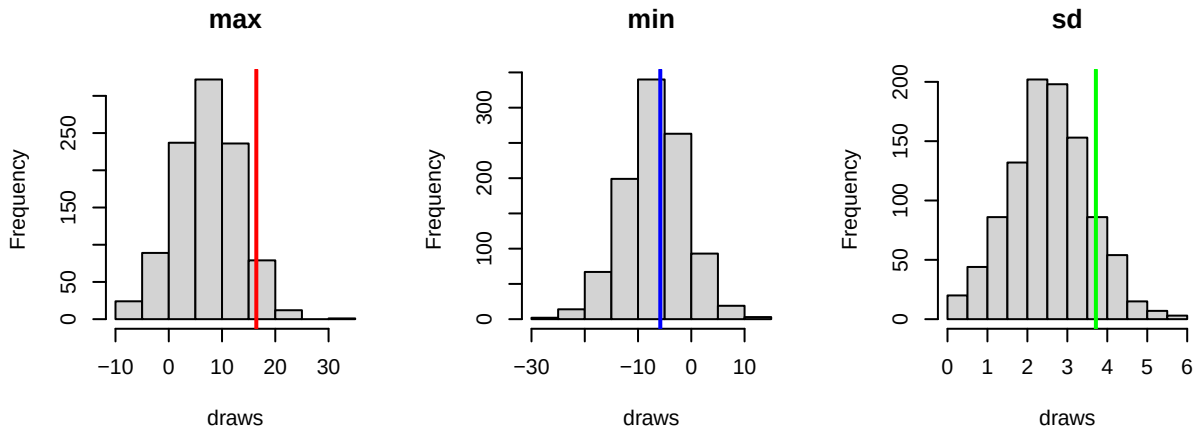
Nudging the prior on σ helps only marginally in trying to simulate reasonable outcome values from the PrPD. The second parameter that comes into play is α, i.e., the intercept. We begin by assigning α the following prior:

$$\alpha \sim \mathrm{N}(0, 5)$$

```
set.seed(123)
y_sim <- matrix(nrow=1000, ncol=200)
for (s in 1:1000) {

  std_dev <- rtruncnorm(1, 0, Inf, 2.5)
  mu <- rnorm(1, 0, 5)
  y_sim[s, ] <- rnorm(200, mu, std_dev)

}
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```




```
par(mfrow=c(1,1), pty="m")
```

All the simulated quantities now look more reasonable when compared to the observed outcome values. If we set

$$\alpha \sim \mathrm{N}(0, 10)$$

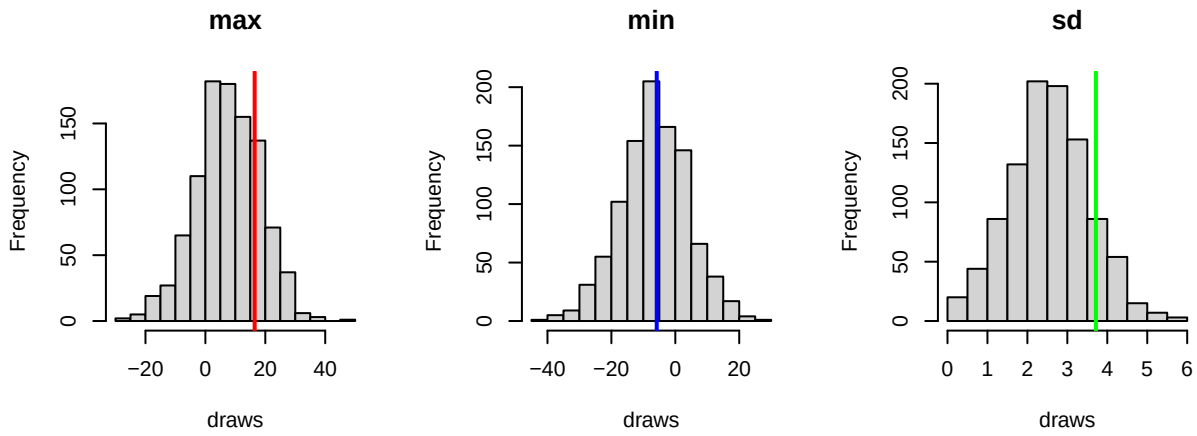
we obtain:

```
set.seed(123)
y_sim <- matrix(nrow=1000, ncol=200)
for (s in 1:1000) {

  std_dev <- rtruncnorm(1, 0, Inf, 2.5)
  mu <- rnorm(1, 0, 10)
  y_sim[s, ] <- rnorm(200, mu, std_dev)

}
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
```

```
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```




```
par(mfrow=c(1,1), pty="m")
```

Calibrating the slope parameters β_1 and β_2

The final step of out prior setting exercise consists of understanding what is a reasonable prior distribution distribution for β_1 and β_2. Since $\beta_j \approx \mathbb{E}[y|x_j] - \mathbb{E}[y|x_j + 1]$ and $\bar{\mathbf{y}} =$`{r} mean(y)`, setting

$$\beta_1 \sim \mathrm{N}(0, 5)$$
$$\beta_2 \sim \mathrm{N}(0, 5)$$

might be a good start. Unlike our simulations for the sole intercept, here we adopt a slightly different approach and collect samples form the PrPD of $\mathbf{y}$ using a `stan` program. This is the approach that we will use moving forward, as simulating from complex model is typically more complicated to hardcode. We begin by loading the necessary scrip:

```
rstan_options(auto_write = TRUE)
lin_reg_prior <- stan_model("linear_regression_prior.stan")
```

We next define the data input and run the simulation:

```
dat_list <- list(
  N = length(y),
  x1 = x1,
  x2 = x2,
  y = y,
  mu_b1 = 0,
  mu_b2 = 0,
  sigma_b1 = 5,
  sigma_b2 = 5,
  mu_alpha = 0,
  sigma_alpha = 10,
  lambda_sigma = 2.5
)
prior_sample <- sampling(lin_reg_prior,
                         data = dat_list,
                         algorithm = "Fixed_param")
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0.027 seconds (Sampling)
## Chain 1:                0.027 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
```

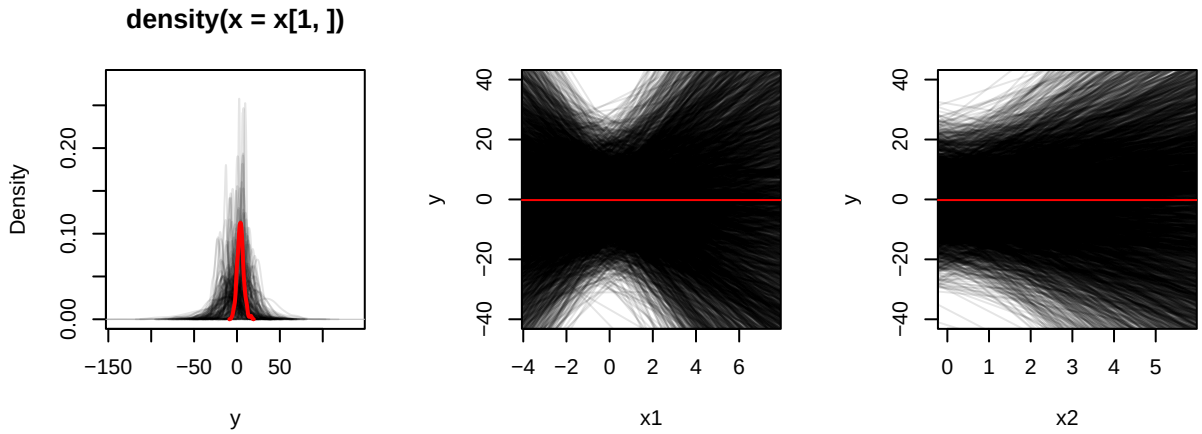
```
## Chain 2:
## Chain 2:  Elapsed Time: 0 seconds (Warm-up)
## Chain 2:                0.029 seconds (Sampling)
## Chain 2:                0.029 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0 seconds (Warm-up)
## Chain 3:                0.029 seconds (Sampling)
## Chain 3:                0.029 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0 seconds (Warm-up)
## Chain 4:                0.029 seconds (Sampling)
## Chain 4:                0.029 seconds (Total)
## Chain 4:
```

```
prior_sample_fit <- extract(prior_sample)
y_sim <- prior_sample_fit$y_pred
```

To visualize the role of $\beta_j \sim \mathrm{N}(0, 5)$ on the PrPD of $\mathbf{y}$ we plot the whole range of regression lines that these priors imply:

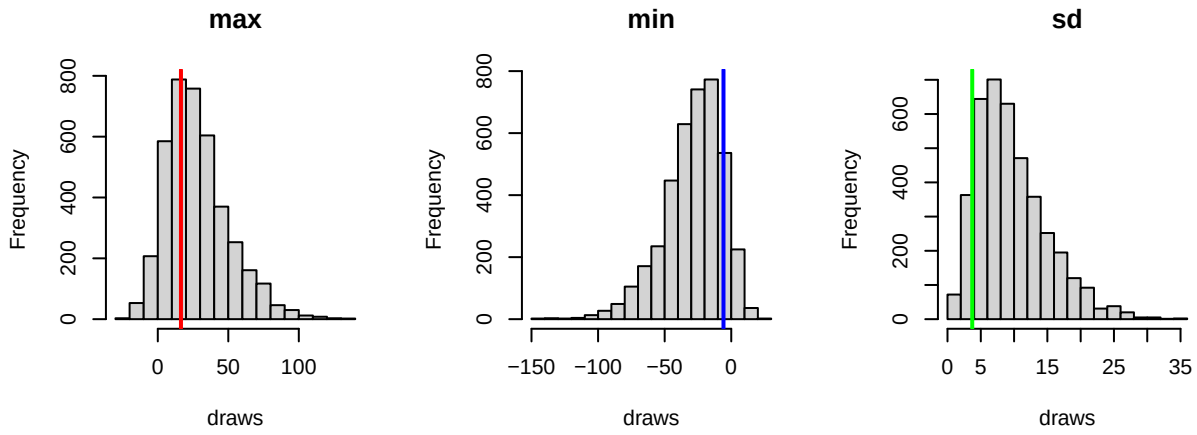
```
par(mfrow=c(1,3))
plot_prior_dens(y_sim, y)
plot_prior_lines(x1,
                 prior_sample_fit$alpha,
                 prior_sample_fit$beta1,
```

```
                  prior_sample_fit$y_pred,
                  ylim = c(-40, 40), xlab = "x1")
plot_prior_lines(x2,
                  prior_sample_fit$alpha,
                  prior_sample_fit$beta2,
                  prior_sample_fit$y_pred,
                  ylim = c(-40, 40), xlab = "x2")
```



Notice how the $\beta_j \sim \mathrm{N}(0, 5)$ prior is truly uninformative as the simulated lines equally represent negatively, positively and zero sloped linear relationships. However, most of these straight lines indicate nonsensical associations, as their projection on the y-axis yields predicted outcomes that are well beyond the expected range of the data. This is also confirmed by the min, max and sd plots:

```
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```




```
par(mfrow=c(1,1), pty="m")
```

Therefore, one might think of giving the β parameters tighter priors, for example:

$$\beta_1 \sim \mathrm{N}(0, 2)$$
$$\beta_2 \sim \mathrm{N}(0, 2)$$

```
dat_list <- list(
  N = length(y),
  x1 = x1,
  x2 = x2,
  y = y,
  mu_b1 = 0,
  mu_b2 = 0,
  sigma_b1 = 2,
  sigma_b2 = 2,
  mu_alpha = 0,
  sigma_alpha = 10,
  lambda_sigma = 2.5
)
prior_sample <- sampling(lin_reg_prior,
```

```
                         data = dat_list,
                         algorithm = "Fixed_param")
```

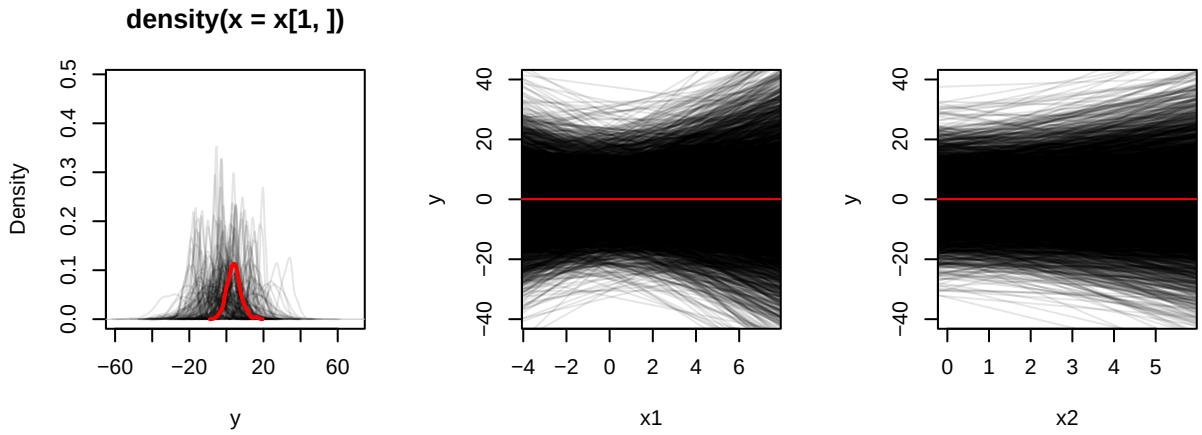
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0.028 seconds (Sampling)
## Chain 1:                0.028 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0 seconds (Warm-up)
## Chain 2:                0.029 seconds (Sampling)
## Chain 2:                0.029 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
```

```
## Chain 3:  Elapsed Time: 0 seconds (Warm-up)
## Chain 3:                0.029 seconds (Sampling)
## Chain 3:                0.029 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0 seconds (Warm-up)
## Chain 4:                0.029 seconds (Sampling)
## Chain 4:                0.029 seconds (Total)
## Chain 4:
```

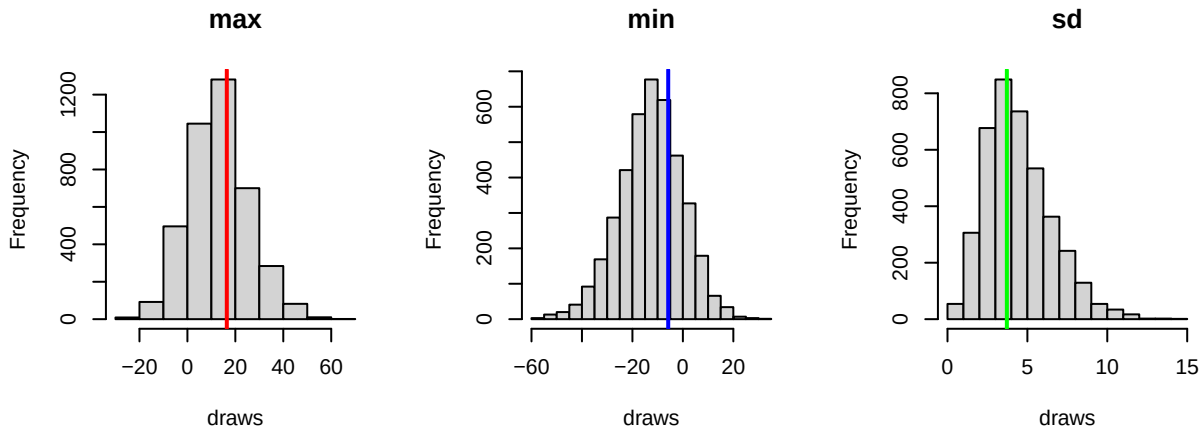
```r
prior_sample_fit <- extract(prior_sample)
y_sim <- prior_sample_fit$y_pred

par(mfrow=c(1,3))
plot_prior_dens(y_sim, y)

plot_prior_lines(x1,
                 prior_sample_fit$alpha,
                 prior_sample_fit$beta1,
                 prior_sample_fit$y_pred,
                 ylim = c(-40, 40), xlab = "x1")
plot_prior_lines(x2,
                 prior_sample_fit$alpha,
                 prior_sample_fit$beta2,
                 prior_sample_fit$y_pred,
                 ylim = c(-40, 40), xlab = "x2")
```




```
par(mfrow=c(1,1))
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```




```
par(mfrow=c(1,1), pty="m")
```

While this simulated regression line look more reasonable, we can see that the prior of corresponding intercepts (i.e., the width of the pivot point) might still be too wide. Therefore, we can try adjusting $\alpha \sim \mathrm{N}(0, 10)$ to, say, $\alpha \sim \mathrm{N}(0, 2.5)$:

```
dat_list <- list(
  N = length(y),
  x1 = x1,
  x2 = x2,
  y = y,
  mu_b1 = 0,
  mu_b2 = 0,
  sigma_b1 = 2,
  sigma_b2 = 2,
  mu_alpha = 0,
  sigma_alpha = 2.5,
  lambda_sigma = 2.5
)
prior_sample <- sampling(lin_reg_prior,
                         data = dat_list,
                         algorithm = "Fixed_param")
```

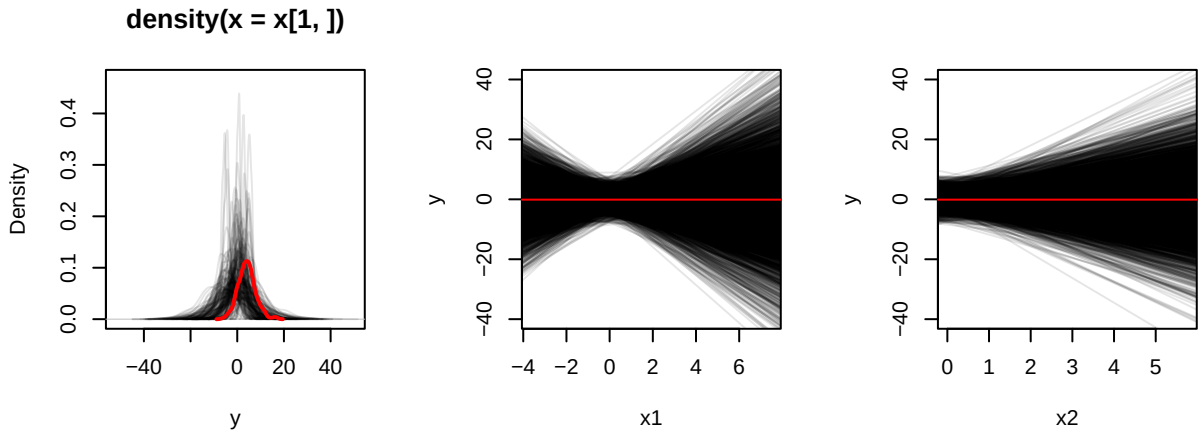
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0.029 seconds (Sampling)
## Chain 1:                0.029 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0 seconds (Warm-up)
## Chain 2:                0.029 seconds (Sampling)
## Chain 2:                0.029 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0 seconds (Warm-up)
## Chain 3:                0.028 seconds (Sampling)
## Chain 3:                0.028 seconds (Total)
## Chain 3:
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0 seconds (Warm-up)
## Chain 4:                0.029 seconds (Sampling)
## Chain 4:                0.029 seconds (Total)
## Chain 4:
```

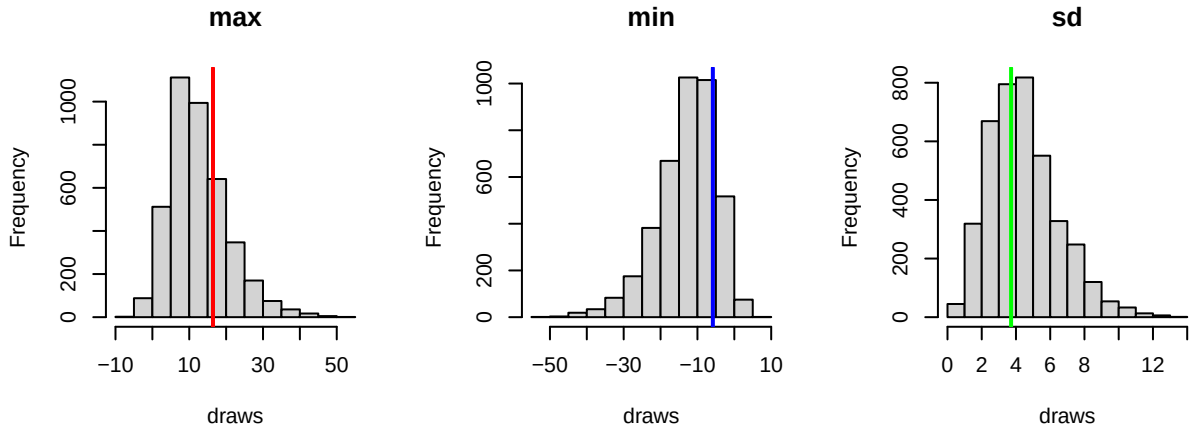
```r
prior_sample_fit <- extract(prior_sample)
y_sim <- prior_sample_fit$y_pred

par(mfrow=c(1,3))
plot_prior_dens(y_sim, y)

plot_prior_lines(x1,
                 prior_sample_fit$alpha,
                 prior_sample_fit$beta1,
                 prior_sample_fit$y_pred,
                 ylim = c(-40, 40), xlab = "x1")
plot_prior_lines(x2,
                 prior_sample_fit$alpha,
                 prior_sample_fit$beta2,
                 prior_sample_fit$y_pred,
                 ylim = c(-40, 40), xlab = "x2")
```




```
par(mfrow=c(1,1))
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y),col="green", lwd=2)
```




```
par(mfrow=c(1,1), pty="m")
```

Default priors

So far, we have tried very hard to come up with reasonable priors for α, β_1, β_2 and σ. However, much of this effort could be relieved by simply putting all the data on the same scale. This is commonly achieved through standardization: for some generic variable z, standardizing means:

$$\dot{z} = \frac{z - \bar{z}}{\text{sd}(z)}$$

As a result, $\dot{z}$ will always have mean zero and variance 1. For example, let us begin by standardizing y and setting the prior for σ:

$$\sigma \sim \text{N}_+(0, 1)$$

Notice that by standardizing y, its mean is zero by construction, so $\alpha = 0$.

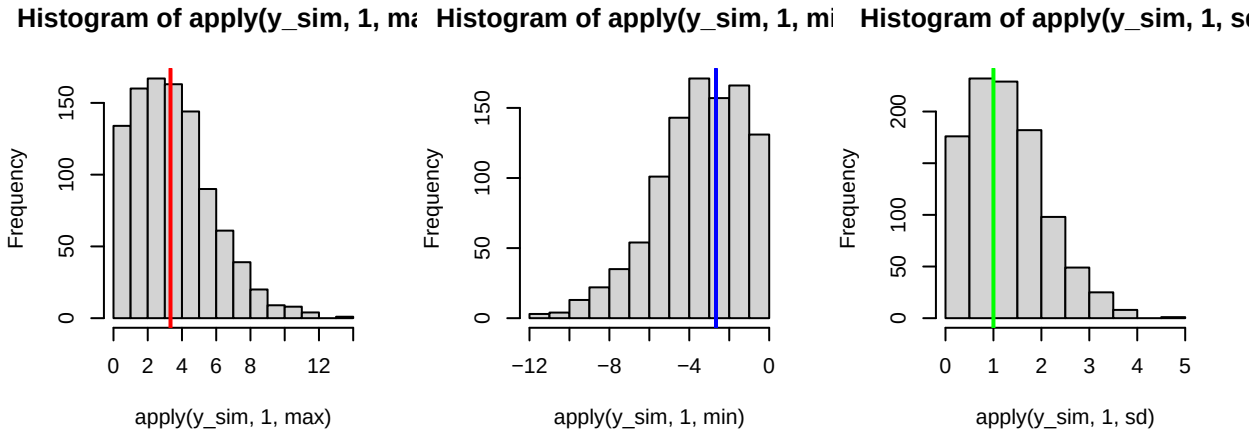
```
y_std <- as.vector(scale(y))

set.seed(123)
y_sim <- matrix(nrow=1000, ncol=200)
```

```r
for (s in 1:1000) {

  std_dev <- rtruncnorm(1, 0, Inf, 1)
  y_sim[s, ] <- rnorm(200, 0, std_dev)

}
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max)); abline(v=max(y_std),col="red", lwd=2)
hist(apply(y_sim, 1, min)); abline(v=min(y_std),col="blue", lwd=2)
hist(apply(y_sim, 1, sd)); abline(v=sd(y_std),col="green", lwd=2)
```




```r
par(mfrow=c(1,1), pty="m")
```

We next complete our prior calibration with with standrdized variables with

$$\beta_1 \sim \mathrm{N}(0, 1)$$
$$\beta_2 \sim \mathrm{N}(0, 1)$$

Since $\mathrm{sd}(\dot{y}) = 1$ by construction, we do not expect that a unit change in either $\dot{x}_1$ or $\dot{x}_2$ will yield shifts in the conditional mean of $\dot{y}$ larger then three standard deviations.

```r
dat_list <- list(
  N = length(y_std),
  x1 = as.vector(scale(x1)),
  x2 = as.vector(scale(x2)),
  y = y_std,
  mu_b1 = 0,
  mu_b2 = 0,
  sigma_b1 = 1,
  sigma_b2 = 1,
  mu_alpha = 0,
  sigma_alpha = 0,
  lambda_sigma = 1
)
prior_sample <- sampling(lin_reg_prior,
                         data = dat_list,
                         algorithm = "Fixed_param")
```

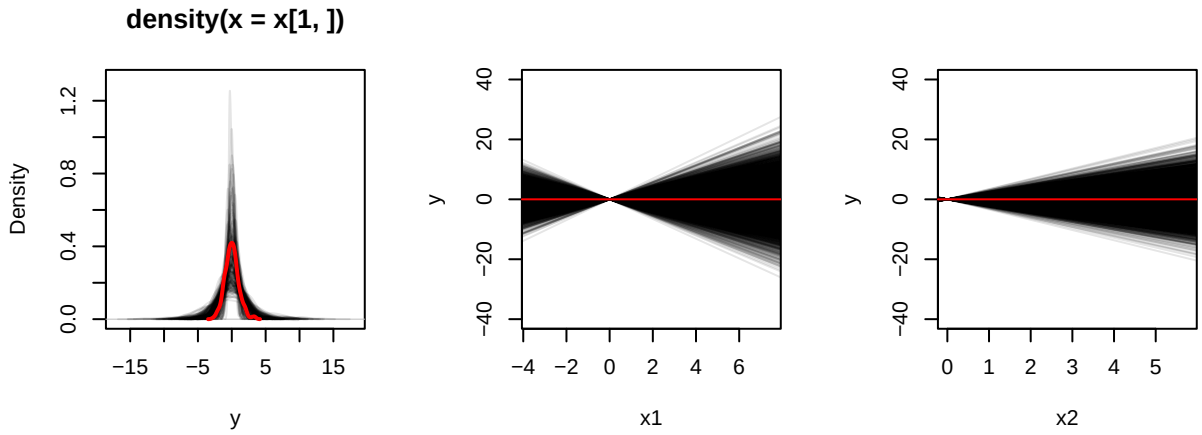
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0.029 seconds (Sampling)
## Chain 1:                0.029 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0 seconds (Warm-up)
## Chain 2:                0.029 seconds (Sampling)
## Chain 2:                0.029 seconds (Total)
## Chain 2:
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0 seconds (Warm-up)
## Chain 3:                0.028 seconds (Sampling)
## Chain 3:                0.028 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4: Iteration:    1 / 2000 [  0%]  (Sampling)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Sampling)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Sampling)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Sampling)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0 seconds (Warm-up)
## Chain 4:                0.028 seconds (Sampling)
## Chain 4:                0.028 seconds (Total)
## Chain 4:
```

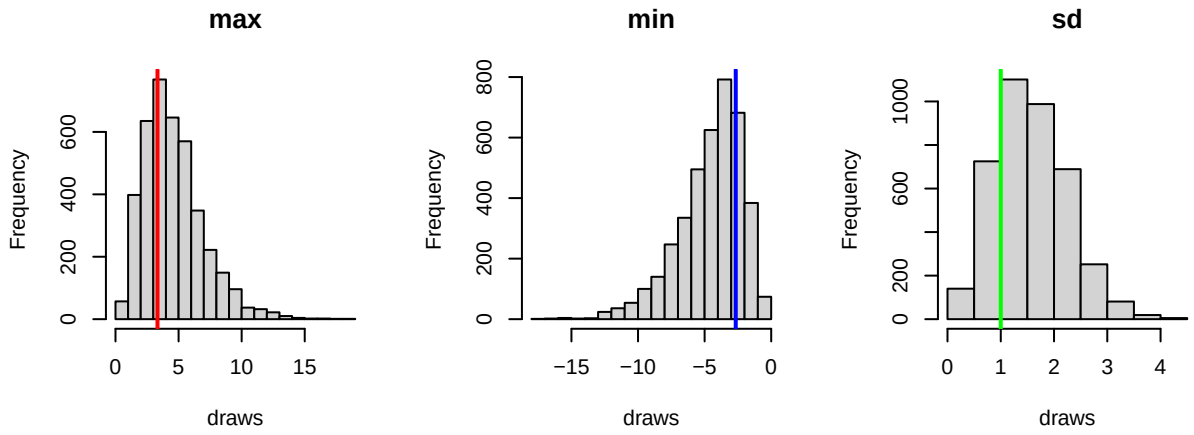
```
prior_sample_fit <- extract(prior_sample)
y_sim <- prior_sample_fit$y_pred

par(mfrow=c(1,3))
plot_prior_dens(y_sim, y_std)

plot_prior_lines(x1,
                 prior_sample_fit$alpha,
                 prior_sample_fit$beta1,
                 prior_sample_fit$y_pred,
                 ylim = c(-40, 40), xlab = "x1")
plot_prior_lines(x2,
                 prior_sample_fit$alpha,
                 prior_sample_fit$beta2,
                 prior_sample_fit$y_pred,
                 ylim = c(-40, 40), xlab = "x2")
```




```
par(mfrow=c(1,1))
par(mfrow=c(1,3), pty="s")
hist(apply(y_sim, 1, max), main = "max", xlab = "draws")
abline(v=max(y_std),col="red", lwd=2)
hist(apply(y_sim, 1, min), main = "min", xlab = "draws")
abline(v=min(y_std),col="blue", lwd=2)
hist(apply(y_sim, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y_std),col="green", lwd=2)
```




```
par(mfrow=c(1,1), pty="m")
```

Estimation

Now that we have defined sensible priors for all the parameters of interest, we can fit the regression model to the data and obtain posterior estimates for the slope coefficients, the intercept and the outcome variance.

Estimation in `Stan` simply requires the definition of a data list (i.e., the one that we have used in the previous section) and to change the sampling algorithm defined above to the HMC sampler. This is simply done by removing the argument `algorithm = "Fixed_param"` and defining the number of Markov Chains and how many of the them we want to run n parallel:

```
lin_reg <- stan_model("linear_regression.stan")

fit <- sampling(lin_reg,
                data = dat_list,
                chains = 4,
                cores = 4)
sample_fit <- extract(fit)
```

To quickly visualize the results, we can call the function `summary` as we would do with the standard `lm` output:

```
params <- c("alpha", "beta1", "beta2", "sigma")
summary(fit, pars = params)$summary
```

```
##            mean      se_mean         sd      2.5%       25%       50%       75%
## beta1 0.7646933 0.0006368860 0.03963645 0.6863018 0.7384181 0.7648391 0.7918249
## beta2 0.4851445 0.0007062893 0.04080874 0.4046954 0.4574228 0.4854879 0.5129838
## sigma 0.5609056 0.0004851969 0.02913474 0.5083348 0.5409774 0.5594048 0.5796840
##           97.5%    n_eff      Rhat
## beta1 0.8402107 3873.166 1.0005519
## beta2 0.5641160 3338.421 1.0011622
## sigma 0.6213890 3605.672 0.9992543
```

Since we standardized both the outcome and the independent variables, we need to transform all the coefficients back to the original scale of the data. To do this, we can simply apply the formula that we (should) have learned for standard linear regression, i.e.:

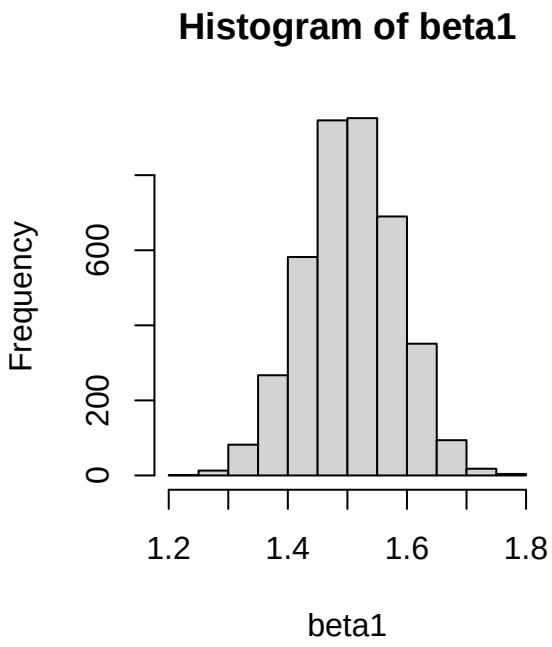
$$\beta_j = \hat{\beta}_j \times \frac{\text{sd}(y)}{\text{sd}(x_j)}$$

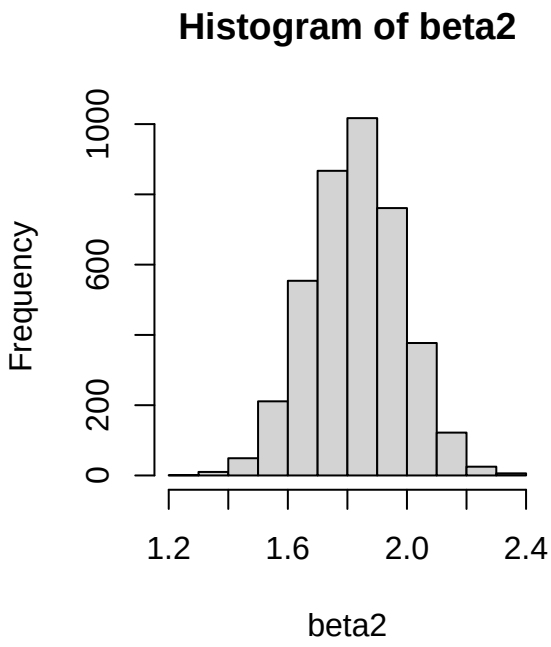
$$\alpha = \bar{y} - \sum_{j=1}^{P} \beta_j \bar{x}_j$$

where $\hat{\beta}_j$ is the regression coefficient obtained by regressing $\dot{y}$ on $\dot{x}_j$.

```
beta1 <- sample_fit$beta1 * (sd(y)/sd(x1))
beta2 <- sample_fit$beta2 * (sd(y)/sd(x2))

par(mfrow=c(1,2), pty="s")
hist(beta1)
hist(beta2)
```






```
par(mfrow=c(1,1), pty="m")

mean(beta1); sd(beta1)
```

```
## [1] 1.505779
```

```
## [1] 0.07804924
```

```
mean(beta2); sd(beta2)
```

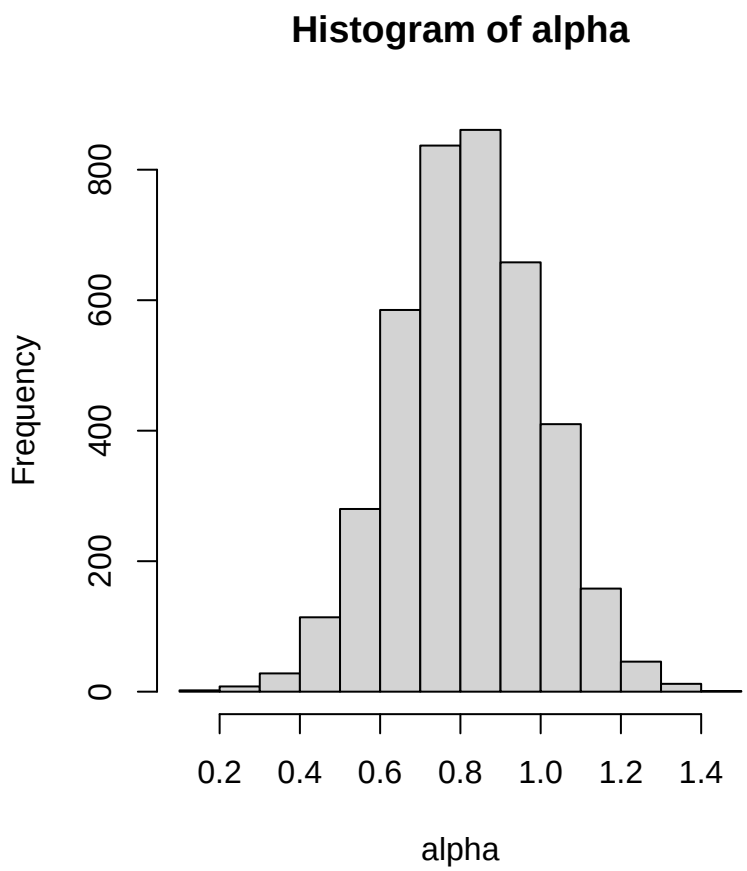
```
## [1] 1.828877
```

```
## [1] 0.153839
```

We can also reconstruct the intercept, if interested:

```
alpha <- mean(y)-(beta1*mean(x1))-(beta2*mean(x2))

par(pty="s")
hist(alpha)
```




```
par(pty="m")

mean(alpha); sd(alpha)
```

```
## [1] 0.8179079
```

```
## [1] 0.1778429
```

Calculating probabilities and Credible Intervals (CrI)

One attractive feature of working with the marginal posterior distribution of α and β_j is that we can make probabilistic statements about the parameter themselves. If the posterior draws of the coefficients are serially uncorrelated, we can then calculate probabilities as simple averages of the extracted samples. For example, suppose that we want to know that is the probability of β_2 being larger than, say 2, we can obtain this value by averaging the count of posterior parameter values $I(\beta_j^s > 2)$:

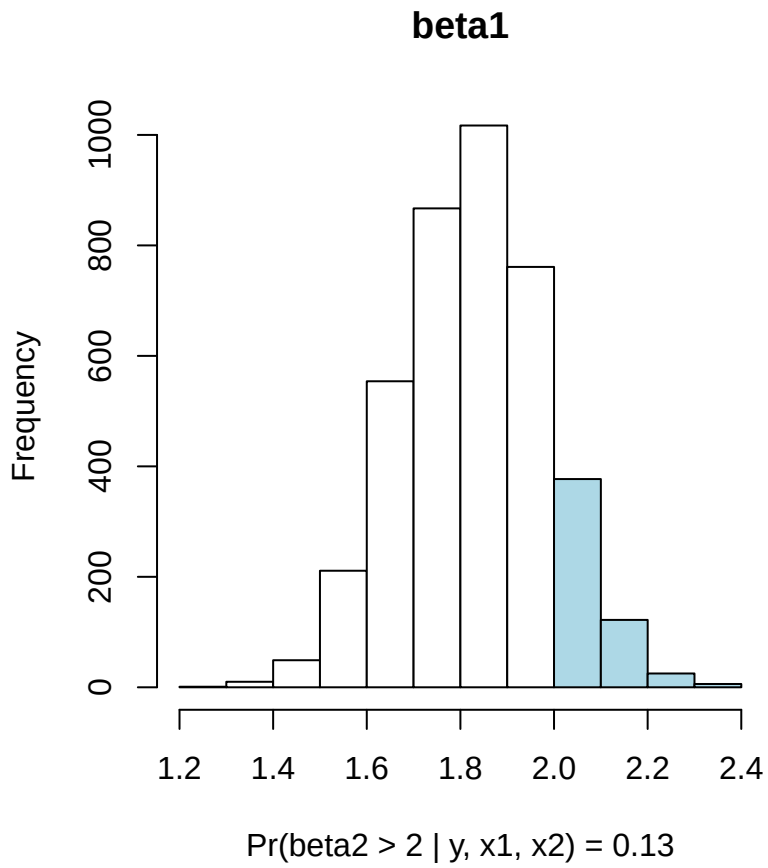
```
thr <- 2
prob <- mean(beta2>thr); prob
```

```
## [1] 0.1325
```

We can also visualize this probability using the histogram above:

```
h_data <- hist(beta2, plot = F)
h_area <- cut(h_data$breaks, c(-Inf, thr, Inf))

par(pty="s")
plot(h_data, col = c("white", "lightblue")[h_area], main = "beta1",
     xlab = paste0("Pr(beta2 > ", thr, " | y, x1, x2) = ", round(prob, 2)))
```




```
par(pty="m")
```

Just as we report Confidence Intervals (CI) in standard frequentest analysis, we can also summaries the marginal posterior of our parameters via intervals. Unlike CI, though, CrI do not describe the range of parameters (a, b) that will include the true value with probability $\gamma\%$. Instead, the latter have more natural interpretation of '$\gamma\%$ of the posterior lies between a and b' or, put it differently, the most likely parameter values lie between a and b.

Defining CrI therefore boils down to calculating an upper and a lower quantile. For example, suppose that we want to get the 95% CrI of β_1 and β_2:

```r
rbind(quantile(beta1, c(.025, .975)),
      quantile(beta2, c(.025, .975)))
```

```
##          2.5%    97.5%
## [1,] 1.351416 1.654482
## [2,] 1.525603 2.126580
```

Model assessment: Posterior Predictive Checks (PoPC)

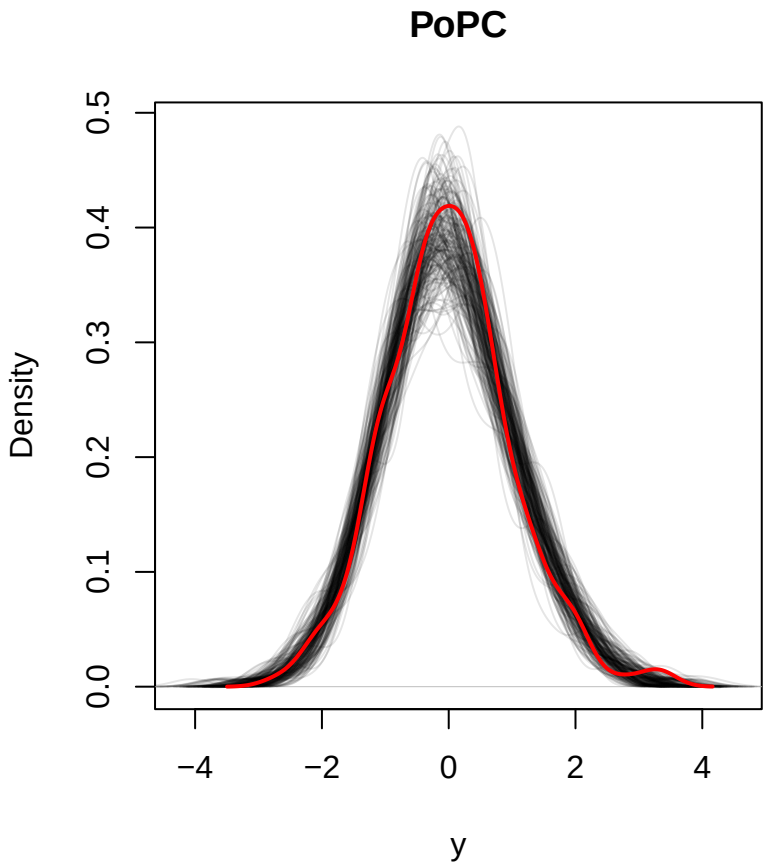
Just as we exploited the PrPD of $\mathbf{y}$ to calibrate the priors, we can use the Posterior Predictive Distribution of $\mathbf{y}$, $\tilde{\mathbf{y}}|\mathbf{y}$ to assess how well our estimated model fir the observed data. The idea is to use the posterior samples for α, β_1 and β_2 to simulate values of $\mathbf{y}$ and compare these predictions with the observed vector of outcomes. In other words, what we do can be formalized as follows:

$$
\begin{aligned}
\alpha^s &\sim p(\alpha|\mathbf{y}, x_1, x_2) \\
\beta_1^s &\sim p(\beta_1|\mathbf{y}, x_1, x_2) \\
\beta_2^s &\sim p(\beta_2|\mathbf{y}, x_1, x_2) \\
\sigma^s &\sim p(\sigma|\mathbf{y}, x_1, x_2) \\
\tilde{\mathbf{y}}^s &\sim \mathbf{N}(\alpha^s + \beta_1^s x_1 + \beta_2^s x_2, \sigma^s)
\end{aligned}
$$

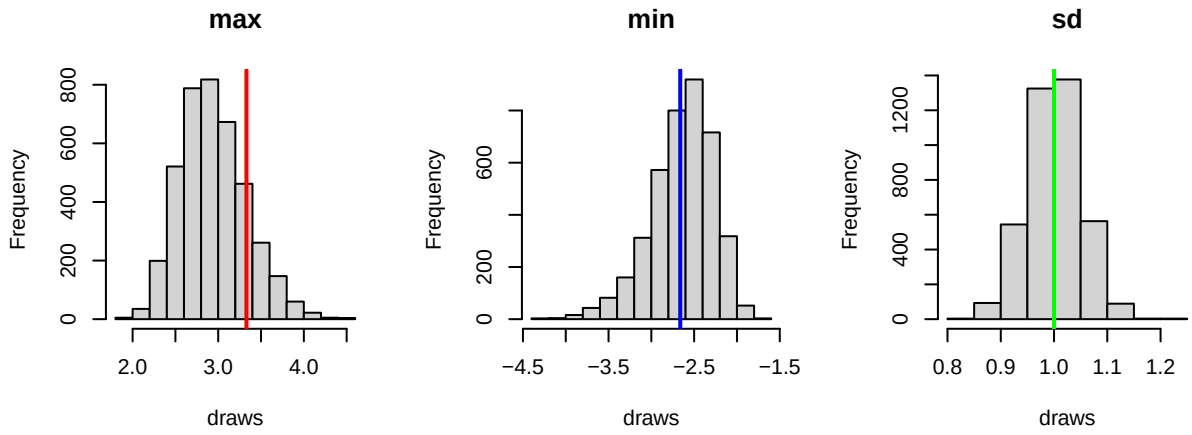
This operation is embedded in the `Stan` file `lin_reg` for simplicity. We can access the full collection of $\tilde{\mathbf{y}}|\mathbf{y}$ from the `sample_fit` object:

```r
y_pred <- sample_fit$y_pred

plot_prior_dens(y_pred, y_std, main = "PoPC")
```




```
par(mfrow=c(1,3), pty="s")
hist(apply(y_pred, 1, max), main = "max", xlab = "draws")
abline(v=max(y_std),col="red", lwd=2)
hist(apply(y_pred, 1, min), main = "min", xlab = "draws")
abline(v=min(y_std),col="blue", lwd=2)
hist(apply(y_pred, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y_std),col="green", lwd=2)
```



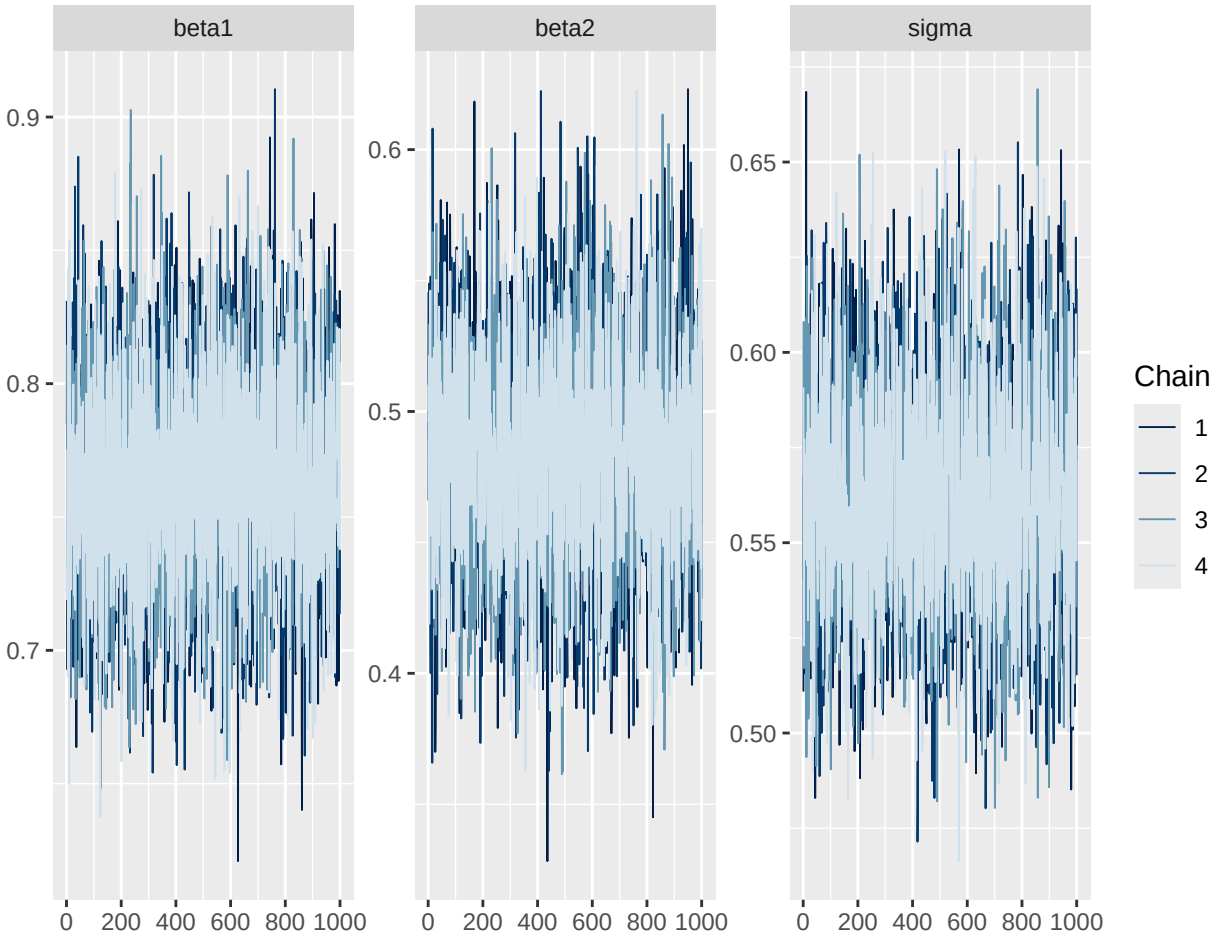

```
par(mfrow=c(1,1), pty="m")
```

These figures indicate very good model fit.

Convergence Diagnostics

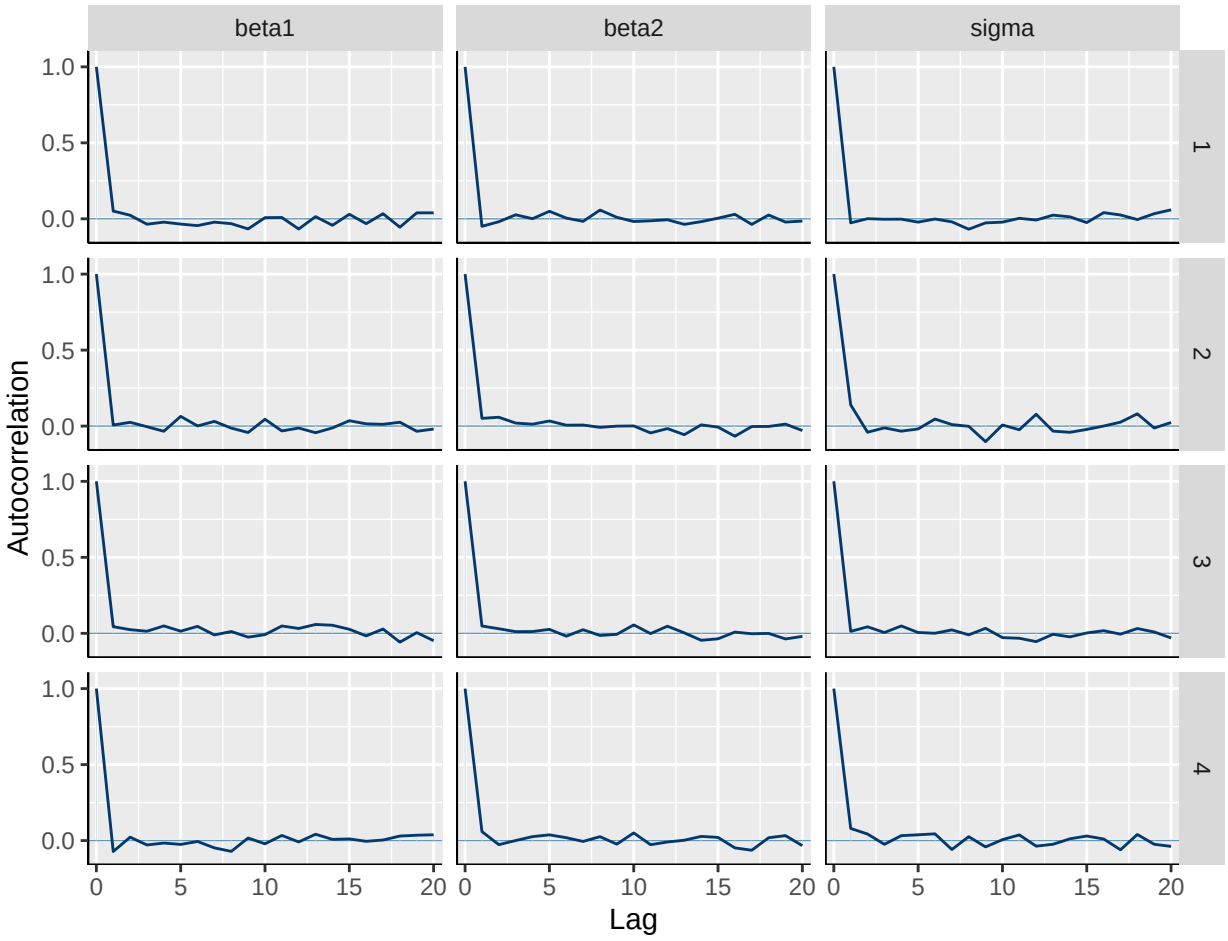
Trace Plots

```
params <- c("beta1", "beta2", "sigma")
mcmc_trace(fit, pars = params)
```



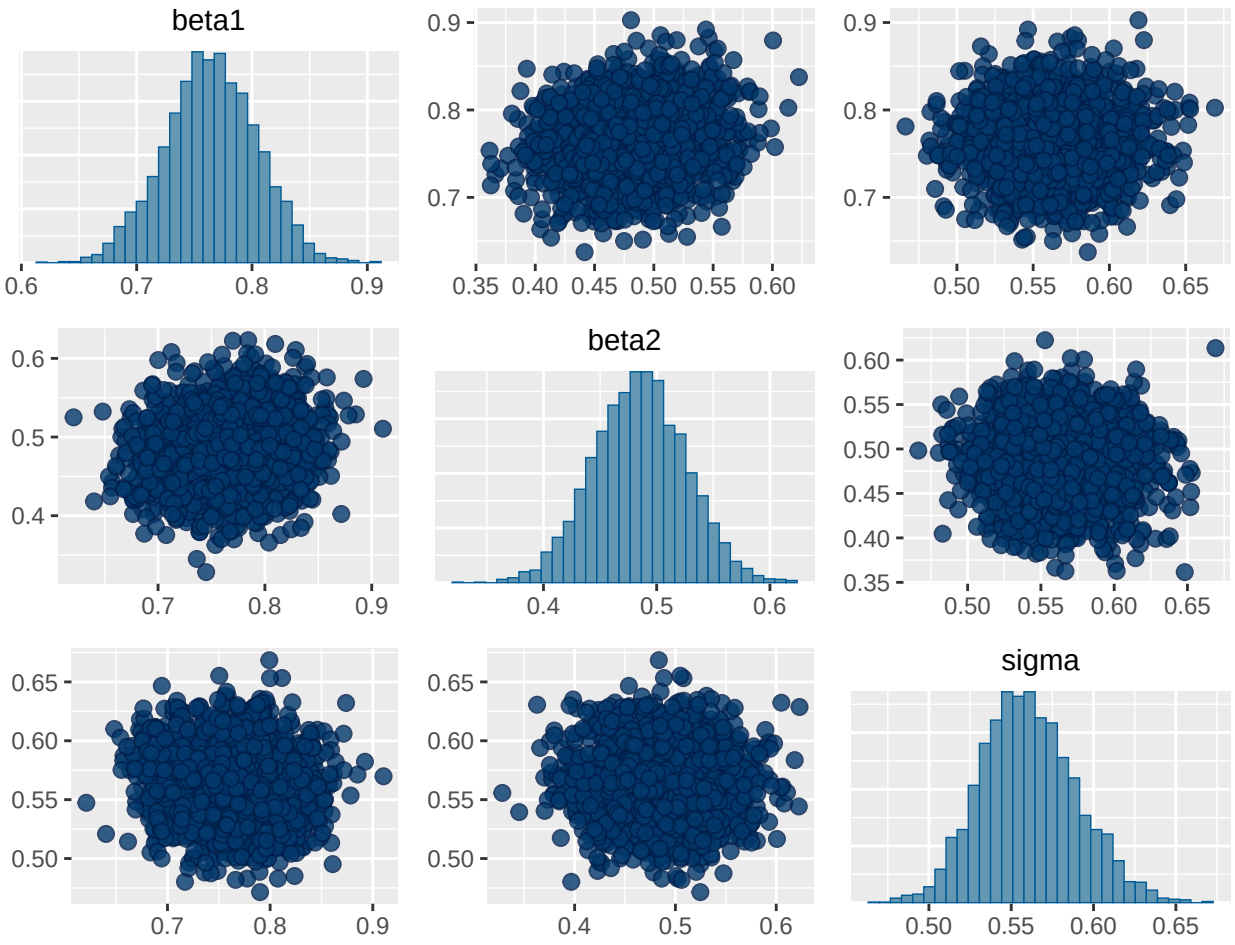
Autocorrelation Plots

```
mcmc_acf(fit, pars = params)
```



Pair Plots

```
mcmc_pairs(fit, pars = params)
```



Effective Sample Size and R-measure

```
neff_ratio(fit, pars = params)
```

```
##     beta1     beta2     sigma
## 0.9682915 0.8346052 0.9014181
```

```
rhat(fit, pars = params)
```

```
##     beta1     beta2     sigma
## 1.0005519 1.0011622 0.9992543
```