Linear regression using Stan

Alessandro Varacca, Thomas Heckelei

July 2024

But what does linear regression mean? (1)

$$y \sim \mathcal{D}(\boldsymbol{\theta})$$

- where $\boldsymbol{\theta}$ indicates a vector of parameters;
- In standard **linear** regression models, we want to specify a functional from for $\mathbb{E}[y|X = x]$ **independently** of the the (conditional) variance $\mathbb{V}(y|X = x)$;
- There are only a few choices of $\mathcal{D}$ that allow to do so;
- The most popular one is the **Normal** distribution. Other options include the **Student's t** and the **Double exponential** distribution;
- The former is by far the most widely adopted;
- Therefore, $\boldsymbol{\theta} = [\mu, \sigma]$ and:

$$y \sim \mathsf{N}(\mu, \sigma)$$

But what does linear regression mean? (2)

- For $i =, 1 \ldots, N$ independent observations, we typically write:

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

$$\mu_i = \mathbb{E}[y|X = x] = \alpha + \sum_{p=1}^{P} \beta_p x_{p,i}$$

- or, equivalently:

$$y_i = \mu_i + \varepsilon_i$$

$$\mu_i = \mathbb{E}[y|X = x] = \alpha + \sum_{p=1}^{P} \beta_p x_{p,i}$$

$$\varepsilon_i \sim \mathrm{N}(0, \sigma^2)$$

- This is the classic homoscedastic linear regression model.

How many parameters?

- The classical linear regression model has $P + 2$ parameters:
  1. σ
  2. α
  3. all the β_p
- We therefore need to set a prior on each of these parameters;
- To understand what kind of priors we need, it is important to understand what each of these parameters governs in the distribution on y_i;
- However, since y is binary, we are better off reasoning in terms of π;
- In particular,
  1. $\sigma = \sqrt{\mathbb{V}(y)}$ governs the **spread** of y, marginally;
  2. $\alpha = \mathbb{E}[y|X = 0]$ governs the **location** of y, marginally;
  3. $\beta_p = \beta_p \approx \mathbb{E}[y|X_p = x_p + 1] - \mathbb{E}[y|X_p = x_p]$ describes the average change in y when we shoft x_p by one unit.

Calibrating the prior(s) (1)

- If we have any knowledge about y_i, we can use this information to set reasonable prior distributions on each of these parameters;
- Going back to Thomas' example on yield, we might know what the expected yield of some crop might be, and how volatile this average value can be;
- In other words, we might have **prior** information on both α and σ;
- Then, in case $P = 1$ and x_i represents, for example, soil moisture, then the literature (or common sense/agronomic expertise) can provide upper and lower bounds for the prior on β;
- The same line of reasoning also apply in case of $P > 1$.

Calibrating the prior(s) (2)

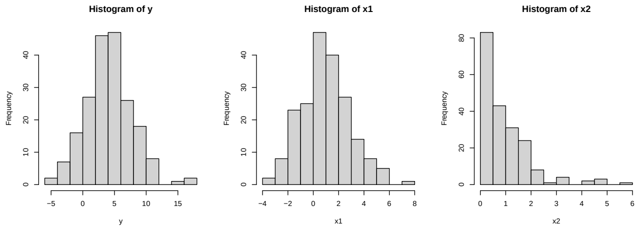
- There are however cases in which such details are not or only partially known;
- Then how do we set reasonable priors?
- In the next slides we are going to explore a simulation-based calibration approach to solve this puzzle;
- The idea is to:
  1. choose a prior for σ, α and β_p and draw potential parameter values from these distribution;
  2. plug these parameter values into $\mathrm{N}(\alpha + \sum_{p=1}^{P} \beta_p x_{p,i}, \sigma)$ and simulate potential values of y_i.

Load data and libraries

```r
library(truncnorm)
library(rstan)

setwd("your working directory")

load("linear_regression_data.RData")
source("aux_functions.R")

set.seed(123)
```

Visualize the data (1)

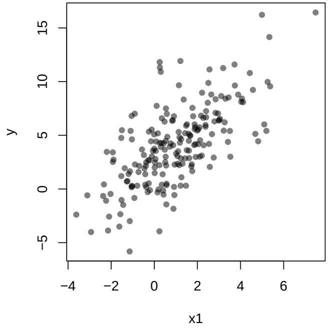
```
par(mfrow=c(1,3), pty="s")
hist(y)
hist(x1)
hist(x2)
```

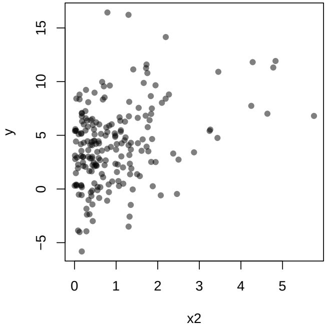


```
par(mfrow=c(1,1), pty="m")
```

Visualize the data (2)

```r
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))
```

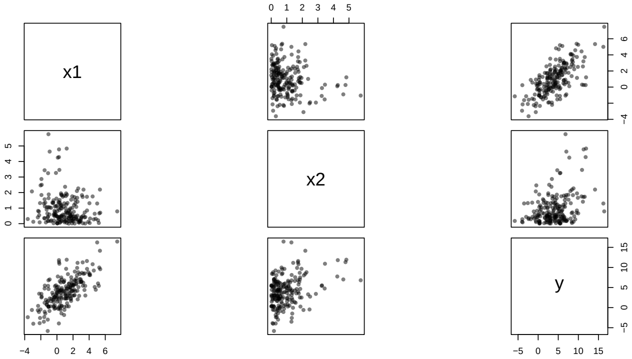




```r
par(mfrow=c(1,1), pty="m")
```

Visualize the data (3)

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



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

Begin sampling from the prior: σ (1)

- We begin by studying how σ may impact $\mathbf{y} = [y_1, \ldots, y_N]$;
- Since the σ can only take up positive values, the **prior distribution** must satisfy this restriction;
- There are several options: **Exponential** distribution, **Gamma** distribution, **Truncated Normal** distribution and others;
- Here we use the Truncated Normal:

$$\sigma \sim \mathsf{N}_+(0, \lambda)$$

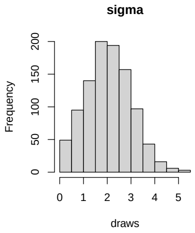
- Calibrating the prior for σ consist in choosing a sensible value for λ.

Begin sampling from the prior: σ (2)

- For example, for $\lambda = 2$, this is how 1000 draws from $N_+(0, 2)$ look like:

```r
std_dev <- rtruncnorm(1000, 0, Inf, 2)
```

```r
par(pty="s")
hist(std_dev, main = "sigma", xlab = "draws")
```



```r
par(pty="s")
```

Begin sampling from the prior: σ (3)

- We can now begin to study what $\mathrm{N}_+(0, 2)$ means in terms of potential values of $\mathbf{y}$:
  1. Sample one parameter value from $\mathrm{N}_+(0, \lambda)$, and get σ_y^s;
  2. Sample one observation from $\mathrm{N}(0, \sigma_y^s)$ and get $\tilde{\mathbf{y}}^s$;
  3. Repeat the process $S = 1000$ times.

```r
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)

}
```

- The $N \times S$ matrix `y_sim`$= [\mathbf{y}^1, \ldots, \mathbf{y}^S]$ approximates the so-called **Prior Predictive Distribution** (PrPD) of y;
- We can then use visualization to study the properties of the PrPD and check whether our assumption makes any sense in face of the observed data, $\mathbf{y}$.

Visualizing the PrPD (1)

1. For each draw $\tilde{\mathbf{y}}^s$ calculate $max^s = \max(\tilde{\mathbf{y}}^s)$;
2. For each draw $\tilde{\mathbf{y}}^s$ calculate $min^s = \min(\tilde{\mathbf{y}}^s)$;
3. For each draw $\tilde{\mathbf{y}}^s$ calculate $sd^s = \text{sd}(\tilde{\mathbf{y}}^s)$

```r
max_y <- apply(y_sim, 1, max)
min_y <- apply(y_sim, 1, min)
sd_y <- apply(y_sim, 1, sd)
```

These are obviously $S \times 1$ vectors:

```r
length(max_y)
```

```
## [1] 1000
```

```r
length(min_y)
```

```
## [1] 1000
```

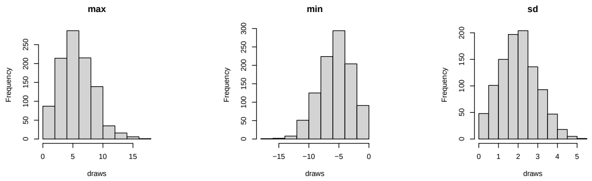
```r
length(sd_y)
```

```
## [1] 1000
```

Visualizing the PrPD (2)

Plot these quantities:

```
par(mfrow=c(1,3), pty="s")
hist(max_y, main = "max", xlab = "draws")
hist(min_y, main = "min", xlab = "draws")
hist(sd_y, main = "sd", xlab = "draws")
```



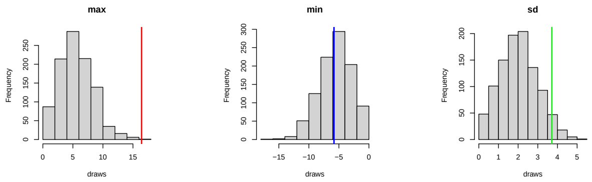
```
par(mfrow=c(1,1), pty="m")
```

Each histogram represents the extent of maximum/minumim values, and standard deviations that can be obtained setting $\lambda = 2$.

Visualizing the PrPD (3)

We can check consistency between the summary statistics of the PrPD and out observed data points by overlaying min(**y**), max(**y**) and *sd*(**y**) to the corresponding histogram:

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



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

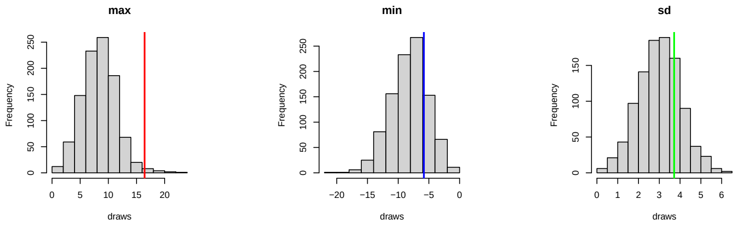
Visualizing the PrPD (3)

- The standard deviation (SD) of the PrPD is smaller than the observed SD;
- Also the maximum value does not quite match;
- We can try changing *lambda* to, say, 3:

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

  std_dev <- rtruncnorm(1, 0, Inf, 3)
  y_sim[s, ] <- rnorm(200, 0, std_dev)

}
```

Visualizing the PrPD (4)

```r
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")
```

- SD is now well calibrated, but to have better max and min we will need to tweak α and β.

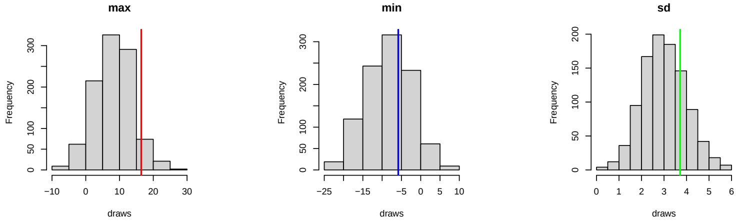
Calibrating α (1)

- Recall that $\alpha = \mathbb{E}[y|X = 0]$;
- Therefore, the range of plausible value of α should reflect the expected range of average values for y;
- Since y includes both positive and negative values, using a **Normal** prior for α should do;
- We begin with $\alpha \sim \text{N}(0, 5)$;
- Notice that this prior implies that the range of $\mathbb{E}[y|X = 0]$ is roughly $[-15, +15]$ and values between $[-5, 5]$ are more likely.

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

  std_dev <- rtruncnorm(1, 0, Inf, 3)
  mu <- rnorm(1, 0, 5)
  y_sim[s, ] <- rnorm(200, mu, std_dev)

}
```

Calibrating α (2)

```
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)
```

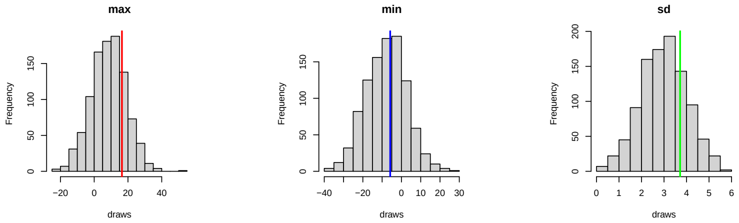


```
par(mfrow=c(1,1), pty="m")
```

Calibrating α (3)

- Setting α to N(0, 10), instead leads to:

```r
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)
```



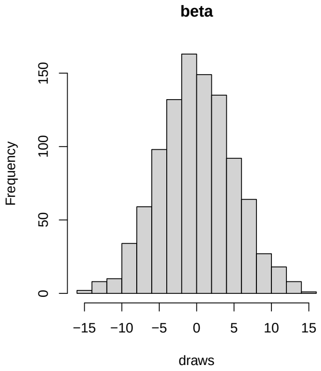
```r
par(mfrow=c(1,1), pty="m")
```

Calibrating α (3)

- Observe that, although these priors might seem at first quite informative, they are not!
- In fact, although the most likely values under the priors set so far are close to those observed in the dataset, the range of plausible y extends quite beyond that range;
- Prior distributions of this nature are typically referred to as **weakly informative**;
- There is a very large chunk of the literature suggesting that weakly informative priors should be preferred to completely uninformative ones;
- There are several reasons for this, but one very important one in probabilistic programming is that they usually trigger computational issues;
- Examples of uninformative priors that should not be used include the historically very popular Uniform$(-\infty, +\infty)$.

Calibrating β_1 and β_2 (1)

- Recall that $\beta_p \approx \mathbb{E}[y|X_p = x_p + 1] - \mathbb{E}[y|X_p = x_p]$. Since $\bar{\mathbf{y}} = 4.07$, setting $\beta_p \sim \text{N}(0, 5)$ implies that shifting x_p by one unit would change the expected value of y by:



Calibrating β_1 and β_2 (2)

- Notably, by using a normal prior centered at zero, we are implicitly assuming that the most likely change in $\mathbb{E}[y|X_p = x_p]$ **before seeing the data** is small;
- This is a **conservative** choice and is typically adopted in regression problems;
- However, notice also that the $\mathrm{N}(0, 5)$ prior allows the 'slopes' to potentially extend up to roughly $[-15, +15]$ and beyond;
- It is the likelihood-prior interaction that will cause a change in the **posterior** distribution of β_p.

Calibrating β_1 and β_2 (3)

- To simulate the impact of
  1. $\sigma \sim \mathrm{N}_+(0, 3)$
  2. $\alpha \sim \mathrm{N}(0, 10)$
  3. $\beta_1 = \beta_2 \sim \mathrm{N}(0, 5)$

  on the PrPD of y, I have prepared a small `Stan` script;
- When we need to calibrate more than one prior, wiring down the whole data generating process in `Stan` is a convenient way to avoid hardcoding the entire scheme in `R`;
- In the next slides I will go through this program very briefly, just so you have an understanding of how it works.

Stan program for prior calibration (1)

- We first load the script using the R function `stan_model()`:

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

- Before using using the `lin_reg_prior` object to simulate the data, let us have a look at what this code looks like!

Stan program for prior calibration (2)

- The input to **any** `Stan` program is a data **list**;
- This list contains all the (hyper)parameter values, the data and any additional variable necessary to run the code contained in the script

```r
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 = 3
)
```

Stan program for prior calibration (3)

- `N` indicates the sample size;
- `x1` and `x2` are $N \times 1$ vector for the two covariates;
- `y` indicates the $N \times 1$ vector $\mathbf{y}$;
- `mu_b1` and `mu_b1` indicate the mean of the priors on β_1 and β_2, respectively;
- `sigma_b1` and `sigma_b1` indicate the standard deviation of the priors on β_1 and β_2, respectively;
- `mu_alpha` indicates the mean of the prior on α;
- `sigma_alpha` indicates the standard deviation of the prior on α;
- `lambda_sigma` indicates the standard deviation of the prior on σ.

Stan program for prior calibration (4)

- We can now simulate $\tilde{\mathbf{y}}^s$!

```
prior_sample <- sampling(lin_reg_prior,
                         data = dat_list,
                         algorithm = "Fixed_param")
```

- and extract the $N \times S$ matrix of simulated outcome values:

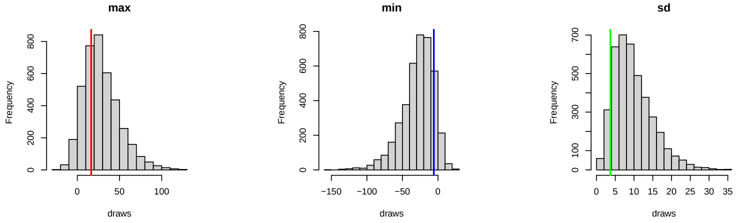
```
prior_sample_fit <- extract(prior_sample)
y_sim <- prior_sample_fit$y_pred
```

- We can next repeat the visual assessment introduced in the earlier slides. We begin by calculating our summary statistics:

```
max_y <- apply(y_sim, 1, max)
min_y <- apply(y_sim, 1, min)
sd_y <- apply(y_sim, 1, sd)
```

Calibrated PrPD (1)

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



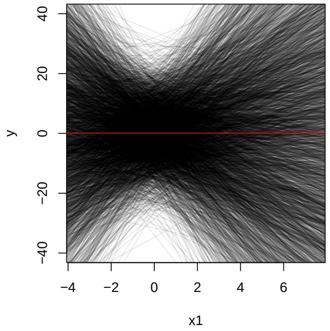
```r
par(mfrow=c(1,1), pty="m")
```

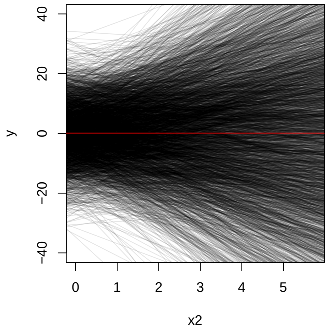
Calibrated PrPD (2)

- We can also visualize the bundle of regression lines implied by $\beta_1 = \beta_2 \sim \mathrm{N}(0, 5)$;
- To do this we can use the

```r
par(mfrow=c(1,2))
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))
```

Calibrated PrPD (3)





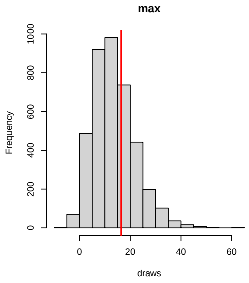
Calibrated PrPD (4)

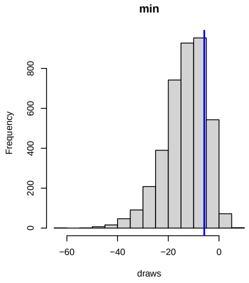
- Things to observe:
  1. This model is general. Perhaps too general... It implies very extreme modelling assumption (i.e., potentially nonsensical regression lines!);
  2. Conditional on the two covariates x_1 and x_2, the prior on the intercept maybe also too wide (look at how wide the bundle of regression lies is around zero!);
- Let us see what changes by adjusting both distributions!

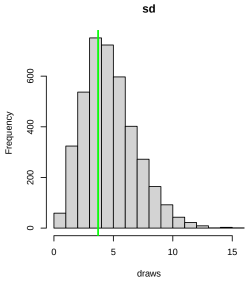
Calibrated PrPD (5)

- Adjusting β_1 and β_2 to N$(0, 2)$, α to N$(0, 2.5)$ and re-running the simulation yields:

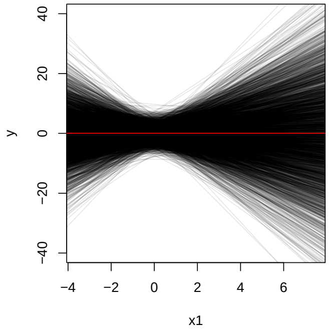
```
dat_list$sigma_b1 <- 2
dat_list$sigma_b2 <- 2
dat_list$sigma_alpha <- 2.5
```

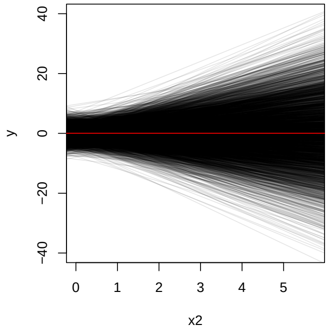






Calibrated PrPD (6)





Default priors (1)

- This calibration exercise demonstrates that, when knowledge on y and x_p is limited, prior calibration can be tedious;
- Moreover, when the number of regressors increases and/or the model includes non-linearities and interactions, choosing the most suitable distribution for β_p becomes very hard;
- Also, new sample information could yield rather different (and bad!) PrPD if the distribution of the new y, x_1 and x_2 changes too much;
- A commonly adopted solution to this issue consists of **standardizing** our variables;
- These transformation remove scale-dependence and allow to defined one-fits-all priors on the parameters of interest;
- Let's see how!

Default priors (2)

- Standardizing some numeric variable z means:

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

- Therefore, any $\dot{z}$ will have mean zero and standard deviation 1 by construction;
- Applying this straightforward transformation to our data simplifies enormously the calibration procedure discussed above;
- To begin with, by standardizing y, we force its mean to zero, so $\alpha = 0$ by construction;
- We therefore do not need to calibrate the prior for α anymore;
- Moreover, since $\mathbb{V}(\dot{y}) = 1$, setting $\sigma \sim \mathrm{N}_{+}(0, 1)$ already provides a good default for the prior on the standard deviation!

Default priors (3)

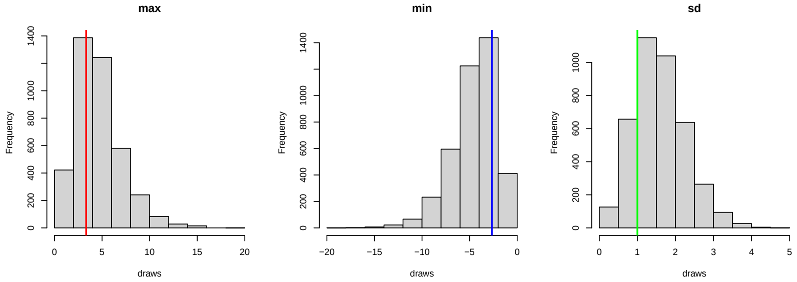
- Finally, since $\mathbb{V}(\dot{y}) = 1$ and the range of $\dot{y}$ is artificially constraint to roughly $[-3, +3]$ (three SD of $\dot{y}$), giving both β_1 and β_2 a $\mathrm{N}(0, 1)$ prior seems a sensible choice;
- To see what these new values yield in terms of PrPD, let us redefine out data list accordingly:

```r
y_std <- as.vector(scale(y))
x1_std <- as.vector(scale(x1))
x2_std <- as.vector(scale(x2))

dat_list <- list(
  N = length(y_std),
  x1 = x1_std, x2 = x2_std,
  y = y_std,
  mu_b1 = 0,
  mu_b2 = 0,
  sigma_b1 = 1, sigma_b2 = 1,
  mu_alpha = 0, sigma_alpha = 0,
  lambda_sigma = 1
)
```

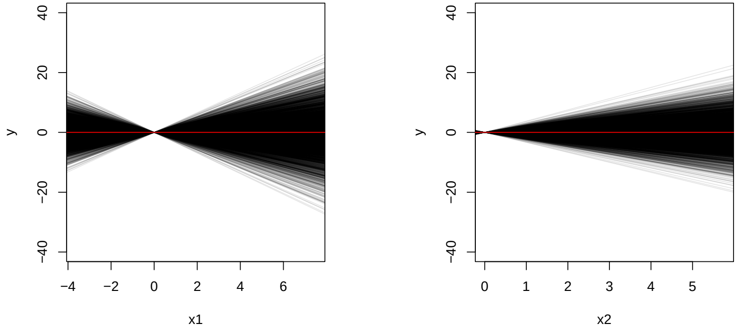
Default priors (4)

```r
prior_sample <- sampling(lin_reg_prior,
                         data = dat_list,
                         algorithm = "Fixed_param")
prior_sample_fit <- extract(prior_sample)
y_sim <- prior_sample_fit$y_pred

max_y <- apply(y_sim, 1, max)
min_y <- apply(y_sim, 1, min)
sd_y <- apply(y_sim, 1, sd)
```

Default priors (5)



Default priors (6)



Estimation

- Once all the priors have been setup, we can use Stan to sample from the implied joint posterior distribution:

$$f(\sigma, \beta_1, \beta_2 | \mathbf{y}, X_1 = x_1, X_2 = x_2)$$

- To do this, we need to load the second program:

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

- Fitting the model to the data now only requesires a slighly different sampling statement:

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

Exploring the results (1)

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

```r
params <- c("beta1", "beta2", "sigma")
summary(fit, pars = params)$summary[,1:3]
```

```
##            mean      se_mean         sd
## beta1 0.7648427 0.0007385151 0.04126889
## beta2 0.4857550 0.0007458866 0.04067350
## sigma 0.5618872 0.0004903962 0.02896625
```

```r
params <- c("beta1", "beta2", "sigma")
summary(fit, pars = params)$summary[,4:8]
```

```
##            2.5%       25%       50%       75%     97.5%
## beta1 0.6822056 0.7378445 0.7646251 0.7917823 0.8481729
## beta2 0.4045790 0.4586613 0.4854272 0.5127030 0.5666763
## sigma 0.5080042 0.5416115 0.5608089 0.5806676 0.6213943
```

Exploring the results (2)

- Because of standardization, we need to transform all the coefficients back to the original scale of the data;
- Otherwise, $\hat{\beta}_p \approx \mathbb{E}[y|X_p = x_p + \text{sd}(y)] - \mathbb{E}[y|X_p = x_p]$;
- To do this, it suffices to rescale the standardized coefficients $\hat{\beta}_p$ by the standard deviation of x_p and y:

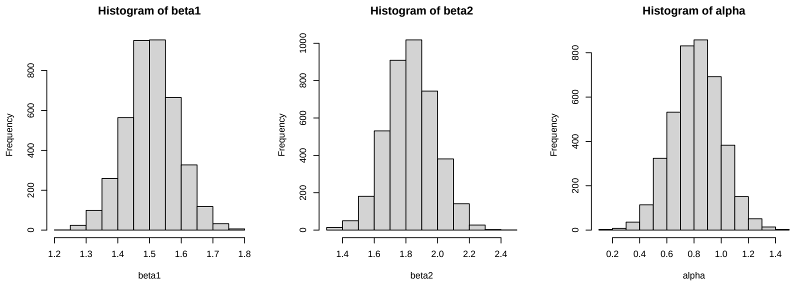
$$\beta_p = \hat{\beta}_p \times \frac{\text{sd}(y)}{\text{sd}(x_p)}$$

- Moreover, we can retrieve the intercept through:

$$\alpha = \bar{y} - \sum_{p=1}^{P} \beta_p \bar{x}_p$$

Exploring the results (3)

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

par(mfrow=c(1,3), pty="s")
hist(beta1)
hist(beta2)
hist(alpha)
```



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

Exploring the results (4)

```r
summary(beta1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   1.223   1.453   1.506   1.506   1.559   1.789
```

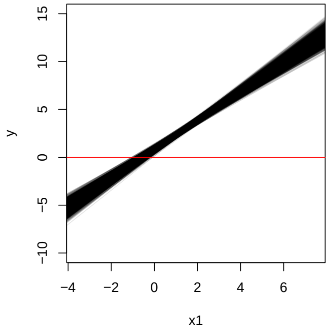
```r
summary(beta2)
```

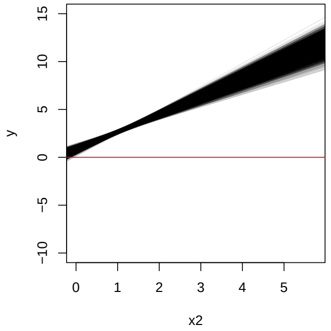
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   1.303   1.729   1.830   1.831   1.933   2.423
```

```r
summary(alpha)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.1201  0.6977  0.8171  0.8154  0.9368  1.4617
```

Exploring the results (5)





What if we used `lm()` instead?

```r
summary(lm(y ~ x1 + x2))
```

```
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.6352 -1.4044  0.1549  1.3290  5.4982
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.81122    0.23174    3.50 0.000574 ***
## x1           1.50803    0.07953   18.96  < 2e-16 ***
## x2           1.83350    0.15226   12.04  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.082 on 197 degrees of freedom
## Multiple R-squared:  0.689,  Adjusted R-squared:  0.6859
## F-statistic: 218.3 on 2 and 197 DF,  p-value: < 2.2e-16
```

Leveraging the posterior

- So far, it might seems that all the extra effort to setup a probabilistic program to estimate a linear regression might be worthless in fact of a simple `lm()` call;
- However, having samples from the joint posterior unlocks several features that only belong in the Bayesian realm;
- First of all, it is worth stressing that having $f(\beta_p|\mathbf{y}, X = x)$ is different from having just $\mathbb{E}[\beta_p]$ and its standard error;
- If the parameters in $f(\sigma, \beta_1, \beta_2|\mathbf{y}, X_1 = x_1, X_2 = x_2)$ are uncorrelated, then extracting any component from the $S \times (P + 2)$ matrix $[\boldsymbol{\sigma}, \boldsymbol{\alpha}, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2]$, provides S samples from the marginal posterior of that parameter;
- These are the quantities that we are going to exploit.

Posterior probabilities (1)

- Consider the marginal posterior of β_2, which we store in the vector `beta2`;
- We can use this object to calculate probabilities. For example, what is the probability that $\beta_2 > 2$?
- This can be easily calculated as the average count of posterior parameter values that are greater than 2:

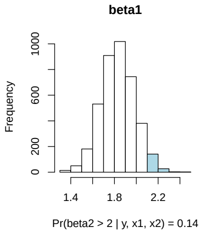
```r
thr <- 2
prob <- mean(beta2>thr); prob
```

```
## [1] 0.13825
```

Posterior probabilities (2)

- We can also visualize this probability using the histogram above:

```r
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,
```



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

Credible Intervals

- We can summarize the marginal posterior of our parameters via intervals;
- These are called Credible Intervals (CrI);
- They are naturally interpreted as the as the proportion of parameter values that fall between (i.e., what is the range of most likely values) an upper and lower value;
- Therefore, CrI are simply computed as the γ and $1 - \gamma$ **quantiles** of β_2. For $\gamma = 0.025$ we have the 95% CrI:

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

```
##     2.5%    97.5%
## 1.525164 2.136232
```

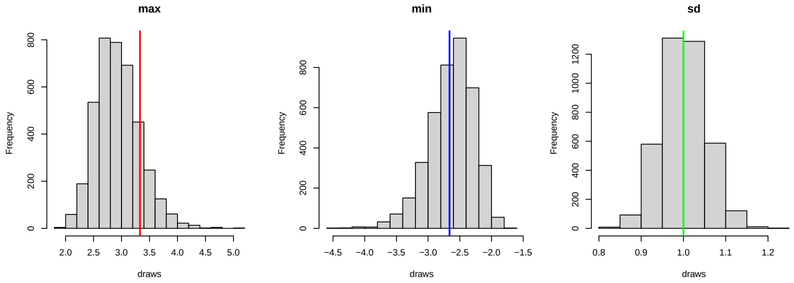
Model checking (internal consistency - 1)

- Another useful tool in the Bayesian regression toolbox is the sp-called Posterior Predictive Distribution (PoPD);
- Much like we used the PrPD to simulate y under different prior configurations, we use the PoPD to simulate y under the estimated posterior;
- The idea is to check the consistency between the simulated y and the observed counterpart;
- This automatically computed in the `Stan` program above and can be easily accessed by extracting `y_pred` from the fitted model:

```
y_pred <- sample_fit$y_pred
```

Model checking (internal consistency - 2)

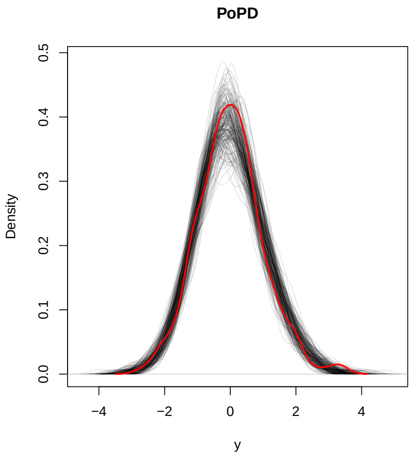
```r
par(mfrow=c(1,3), pty="s")
hist(apply(y_pred, 1, max), main = "max", xlab = "draws")
abline(v=max(y_std),col="red", lwd=2)
hist(apply(y_pred, 1, min), main = "min", xlab = "draws")
abline(v=min(y_std),col="blue", lwd=2)
hist(apply(y_pred, 1, sd), main = "sd", xlab = "draws")
abline(v=sd(y_std),col="green",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



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

Model checking (internal consistency - 3)

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


PoPD
Density
y