

OPENCOURSEWARE
ADVANCED PROGRAMMING
STATISTICS FOR DATA SCIENCE
Ricardo Aler

Rstan

Stan

- Stan is a probabilistic programming language for statistical inference written in C++
- It follows Bayesian statistics principles
- It is typically used to find the distributions of parameters in a model
- Stan can be used from Python (pystan) or R (**Rstan**)
- In order to use Stan fully, specific concepts about Bayesian inference and sampling must be known, therefore this part of the lecture is only a small introduction to Stan.

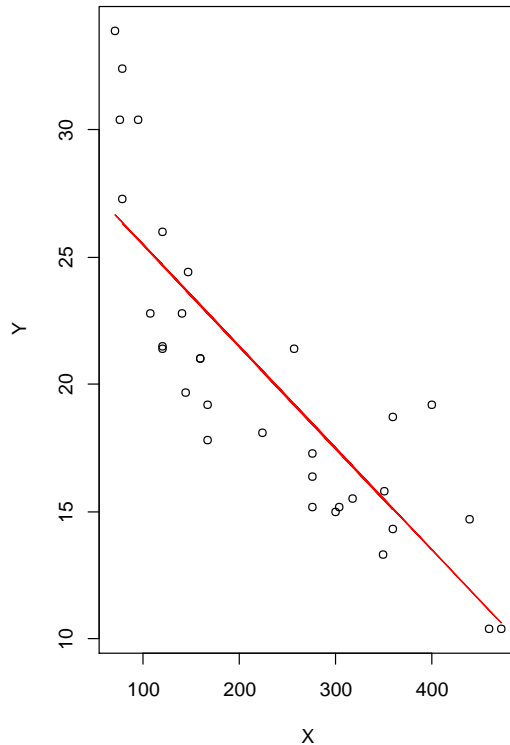
Stan

- Stan is typically used to find the distributions of parameters in a model
- For instance, a linear model can be used to describe some data
$$f(x) = B_0 + B_1 * x + \text{noise}$$
- Standard machine learning or statistical techniques, just fit the model to the data and give some values to the parameters B_0 and B_1 :
$$f(x) = 29.51 - 0.04 * x + \text{noise}$$
- However, those parameters have been estimated from a data sample. If the sample changed, the estimation of coefficients would be slightly different.
- Therefore, what is the uncertainty of the coefficients?
- The advantage of Stan is that it gives the full probability distribution of the model parameters.

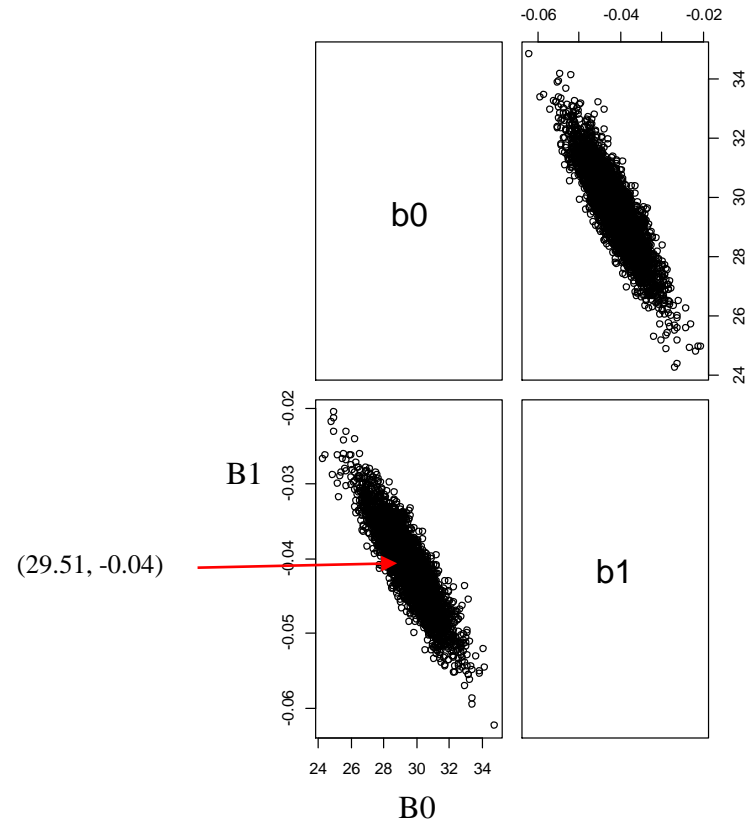
Stan

Stan tells us, not just the most likely values of the coefficients, but also their joint distribution (we can also see that they are correlated)

```
plot(X,Y)  
lines(X,29.51-0.04*X, col="red")
```



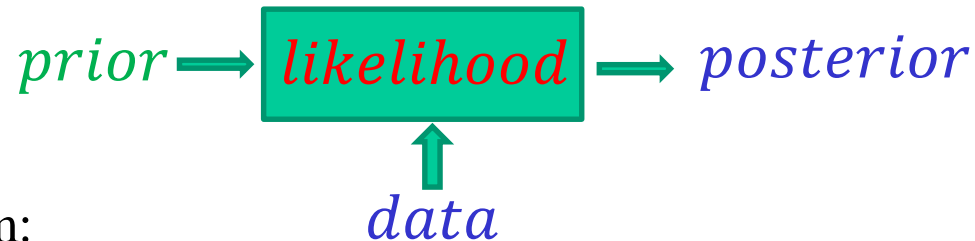
Joint distribution of B0 and B1



Simple example with Rstan

- Stan is typically used to find the distributions of parameters in a model
- Let's suppose that we have some *data* about people's height in a population
- Let's assume that people's height follow a Normal distribution $N(\mu, \sigma)$
- What is the distribution of μ and σ ?
- Bayesian statistics use Bayes theorem

Simple example with Rstan



- Bayes theorem:

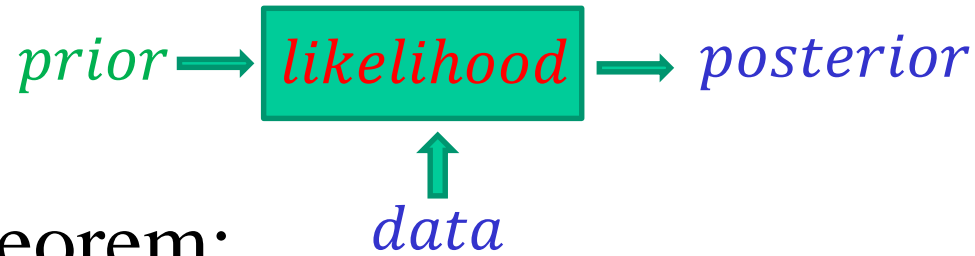
$$posterior \propto likelihood * prior$$

- $prob(\mu, \sigma | data) = \frac{prob(data | \mu, \sigma) * prob(\mu, \sigma)}{prob(data)}$

- **prior**: a-priori beliefs:

- We have no idea: μ follows a uniform distribution between -inf and +inf
- We might have some knowledge about μ . These are some examples with different degrees of uncertainty:
 - It is larger than zero
 - It is uniformly distributed between 0 and 3 meters
 - It is likely to be around 1.6. This can be formulated as: μ follows a normal distribution around 1.6 with sd=0.5
- prior beliefs can be wrong, but they can be corrected if enough data is available.

Simple example with Rstan



- Bayes theorem:

$$\text{posterior} \propto \text{likelihood} * \text{prior}$$

- $\text{prob}(\mu, \sigma | \text{data}) = \frac{\text{prob}(\text{data} | \mu, \sigma) * \text{prob}(\mu, \sigma)}{\text{prob}(\text{data})}$
- **likelihood**: what is the probability of observing some data (list of people's heights in the class) (assuming that our model is a $N(\mu, \sigma)$)?

Simple example with Rstan

- **likelihood**: what is the probability that the data (list of people's heights in the class) has been generated from a $N(\mu, \sigma)$?
- $\text{data} = \{1.8, 1.7, 1.65, 1.5, 1.85, 1.75, \dots\}$
- $\text{prob}(\text{data} | \mu, \sigma)$?
- $= N(1.8, \mu, \sigma) * N(1.7, \mu, \sigma) * \dots * N(1.75, \mu, \sigma) * \dots$

Simple example with Rstan

- Bayes theorem:

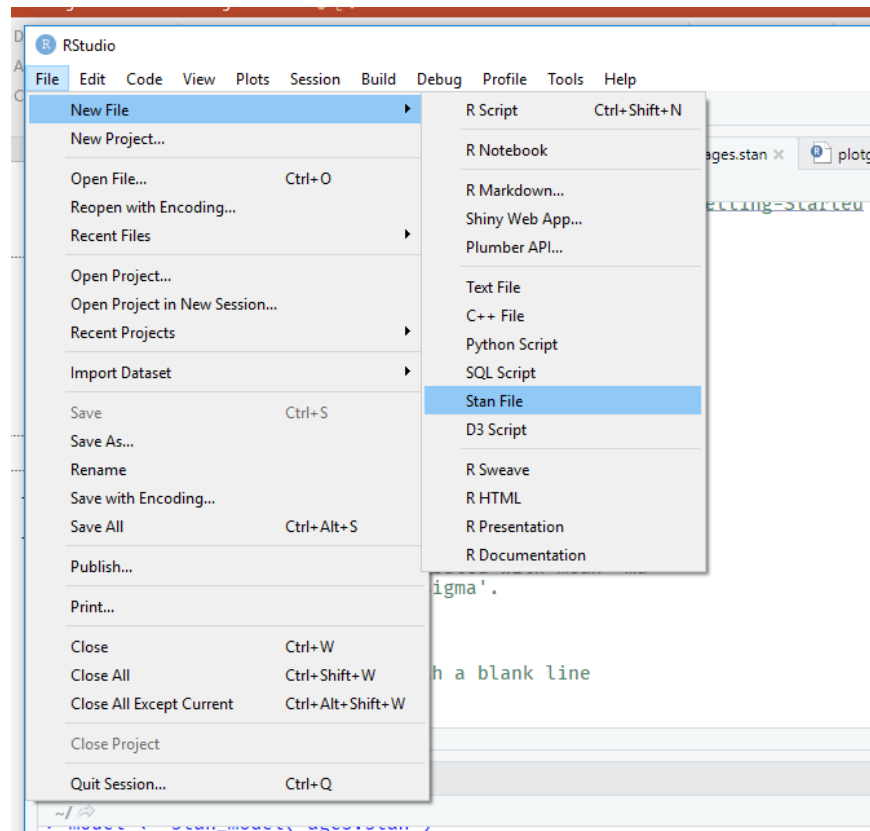
$$\textit{posterior} \propto \textit{likelihood} * \textit{prior}$$

- $\textit{prob}(\mu, \sigma | \textit{data}) = \frac{\textit{prob}(\textit{data} | \mu, \sigma) * \textit{prob}(\mu, \sigma)}{\textit{prob}(\textit{data})}$
- *posterior*: Once we have observed the data, what is now the probability of (μ, σ) ?
- The aim of Stan is to give the posterior distribution of (μ, σ)

Simple Stan Program

- Stan programs have three sections:
 - **data**: description of size and type of the data
 - **parameters**: description of types of parameters of the model (in this case, μ and σ)
 - **model**: description of the model:
 - likelihood
 - priors

- Stan program files are opened like this:



Simple Stan Program

```
// The input data is a vector 'Y' of length 'N'.
data {
  int<lower=0> N;
  vector[N] Y;
}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  real<lower=0> mu;
  real<lower=0> sigma;
}

// The model to be estimated. We model the output
// 'Y' to be normally distributed with mean 'mu'
// and standard deviation 'sigma'.
model {
  for(i in 1:N){
    Y[i] ~ normal(mu, sigma);
  }
}

// Make sure program ends with a blank line
```

Simple Stan Program

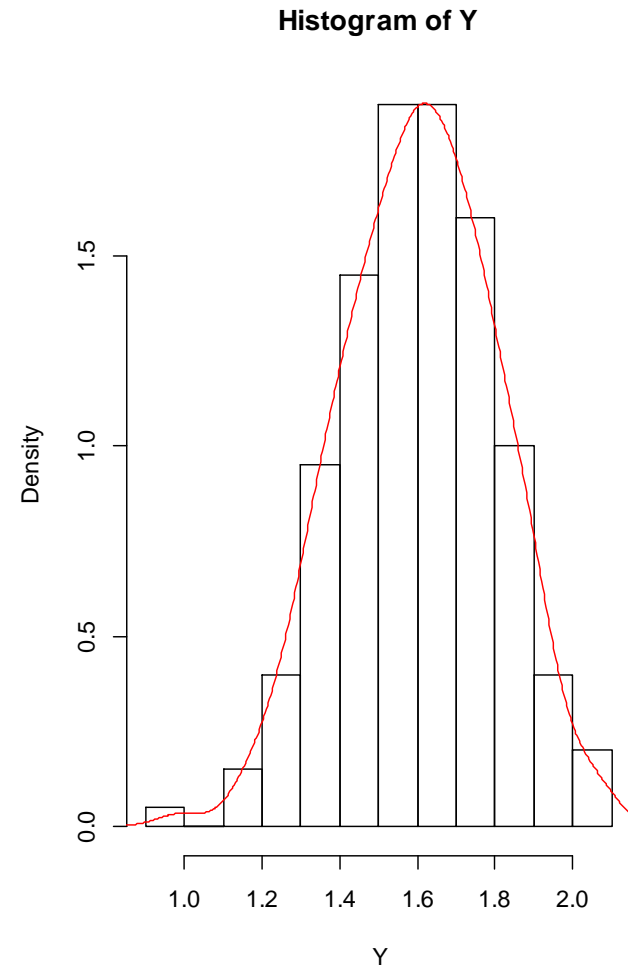
- Stan programs are called from R programs

```
library(ggplot2)
library(rstan)
# Creating the artificial data: people's ages
N <- 200
Y <- rnorm(N, 1.6, 0.2)

# Plotting the artificial data
hist(Y, prob=TRUE)
lines(density(Y), col="red")

# Stan configuration
cores <- parallel::detectCores()
options(mc.cores = cores)
rstan_options(auto_write = TRUE) # Avoid recompilation
# For efficiency (but remove if errors reported)
Sys.setenv(LOCAL_CPPFLAGS = '-march=native')

#Compiling Stan model (it takes some time, seconds to minutes)
model <- stan_model('ages.stan')
fit <- sampling(model, list(N=N, Y=Y), iter = 200, chains=cores)
```



Stan Results

```
> fit
Inference for Stan model: ages.
8 chains, each with iter=200; warmup=100; thin=1;
post-warmup draws per chain=100, total post-warmup draws=800.
```

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
mu	1.60	0.00	0.01	1.58	1.59	1.60	1.61	1.63	749	1.00
sigma	0.20	0.00	0.01	0.18	0.19	0.20	0.20	0.21	165	1.04

- Rhat: measure of convergence. Good if $Rhat < 1.1$
- n_eff: effective sample size. The closer to total_post_warmup_draws the better (800 in this case)
- mu, sigma means are the average values for the two model parameters

Stan Results

- Samples for mu and sigma can be obtained by using

```
params <- extract(fit)
```

- `params$mu` contains the 800 samples for mu

```
[1] 1.623349 1.578908 1.592965 1.611246 1.598419 1.603742 1.593246 1.642225 1.599661 1.618691 1.583435 1.594036 1.602810 1.624624 1.580840 1.616283  
[17] 1.610417 1.595512 1.591667 1.590103 1.616803 1.610667 1.593941 1.613578 1.609404 1.617760 1.613013 1.611529 1.604759 1.606094 1.618307 1.615376  
[33] 1.602298 1.589461 1.628366 1.578119 1.612848 1.615947 1.606042 1.571163 1.596216 1.616349 1.595013 1.587783 1.585090 1.593094 1.609339 1.582615  
[49] 1.566420 1.624919 1.607291 1.598916 1.601412 1.602180 1.622353 1.610767 1.598690 1.596497 1.612590 1.603142 1.616989 1.617132 1.600533 1.611154  
[65] 1.629451 1.602251 1.594384 1.585232 1.602301 1.619595 1.621267 1.612738 1.600781 1.610508 1.587262 1.601109 1.585823 1.599991 1.608005 1.606196  
[81] 1.590657 1.604987 1.608641 1.621516 1.604674 1.593474 1.649275 1.601432 1.620614 1.594151 1.580621 1.587313 1.582009 1.592654 1.608902 1.587119
```

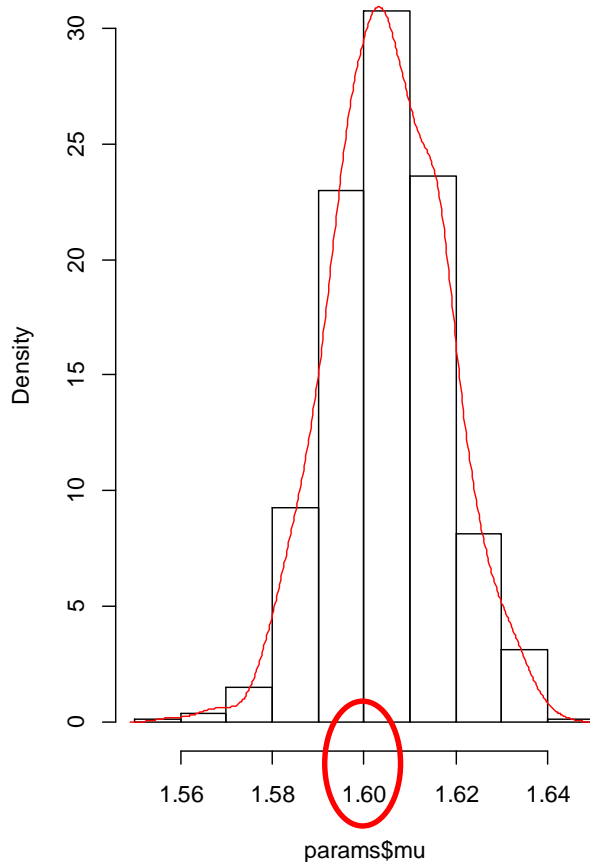


```
hist(params$mu, prob=TRUE)
lines(density(params$mu), col="red")
hist(params$sigma, prob=TRUE)
lines(density(params$sigma), col="red")
```

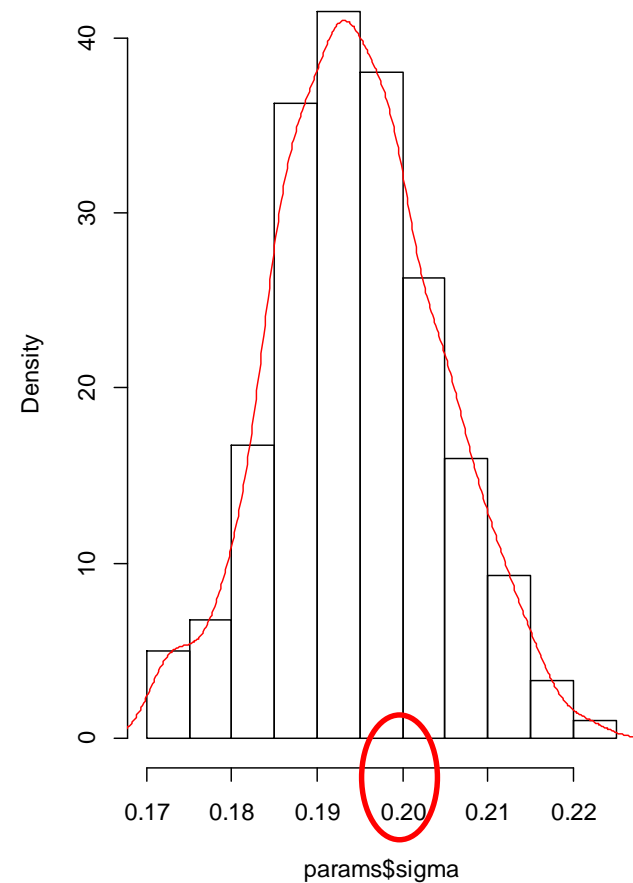
Stan Results

An advantage of Stan is that we not only know the most likely values for the parameters, but also their distribution

Histogram of params\$mu



Histogram of params\$sigma

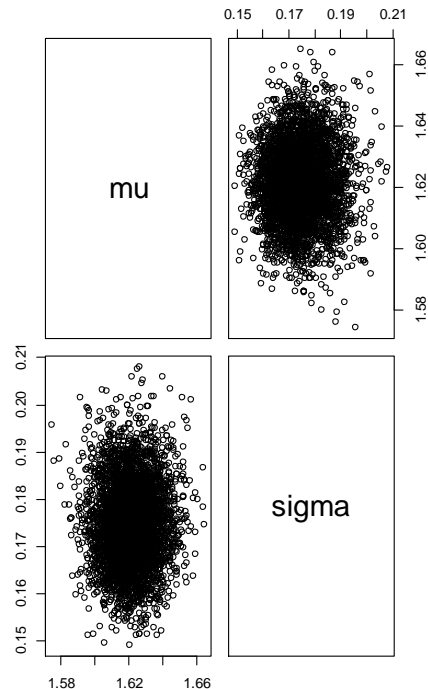


marginal probability distributions

Stan Results

And also their joint distribution. We can see that in this case μ and σ are not correlated.

```
pairs(params[c("mu", "sigma")])
```



joint probability distributions: $\text{prob}(\mu = x, \sigma = y)$

Stan Results

- Let's check with more iterations_

```
fit <- sampling(model, list(N=N, Y=Y), iter = 1000, chains=cores)
```

```
> print(fit)
```

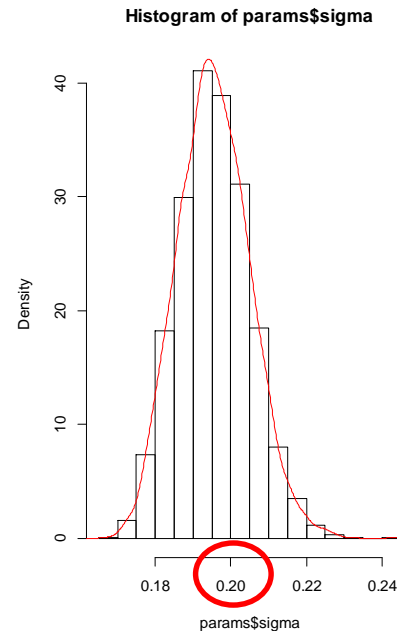
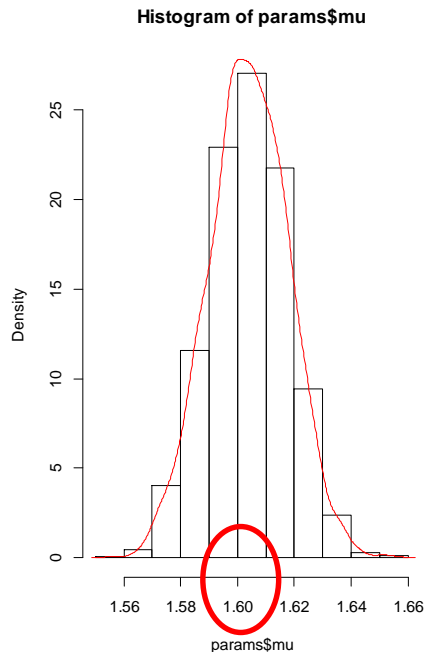
```
Inference for Stan model: ages.
```

```
8 chains, each with iter=1000; warmup=500; thin=1;
```

```
post-warmup draws per chain=500
```

```
total post-warmup draws=4000
```

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
mu	1.60	0.00	0.01	1.58	1.60	1.60	1.61	1.63	3919	1
sigma	0.20	0.00	0.01	0.18	0.19	0.20	0.20	0.22	3072	1

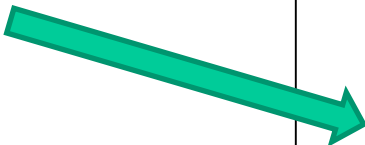


Simple Stan Program: Vectorized

```
// The input data is a vector 'Y' of length 'N'.
data {
  int<lower=0> N;
  vector[N] Y;
}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  real<lower=0> mu;
  real<lower=0> sigma;
}

// The model to be estimated. We model the output
// 'Y' to be normally distributed with mean 'mu'
// and standard deviation 'sigma'.
model {
  for(i in 1:N){
    Y[i] ~ normal(mu, sigma);
  }
}
// Make sure program ends with a blank line
```



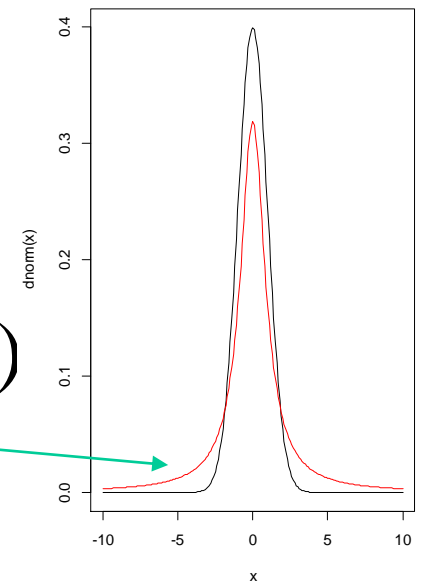
```
// The input data is a vector 'Y' of length 'N'.
data {
  int<lower=0> N;
  vector[N] Y;
}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  real<lower=0> mu;
  real<lower=0> sigma;
}

// The model to be estimated. We model the output
// 'Y' to be normally distributed with mean 'mu'
// and standard deviation 'sigma'.
model {
  Y ~ normal(mu, sigma);
}
// Make sure program ends with a blank line
```

Adding priors to the model

- The program so far had no priors (mu and sigma follow a uniform distribution from 0 to infinity)
- Priors give information where to focus the search
- Let's suppose that:
 - mu follows a Normal(1.7, 0.3) (black)
 - sigma follows a Cauchy(0,1) (red)
- Cauchy has heavier tails (than normal)



Adding priors to the model

// The input data is a vector 'Y' of length 'N'.

```
data {  
  int<lower=0> N;  
  vector[N] Y;  
}
```

// The parameters accepted by the model. Our model

// accepts two parameters 'mu' and 'sigma'.

```
parameters {  
  real<lower=0> mu;  
  real<lower=0> sigma;  
}
```

// The model to be estimated. We model the output

// 'Y' to be normally distributed with mean 'mu'

// and standard deviation 'sigma'.

```
model {  
  Y ~ normal(mu, sigma);  
  mu ~ normal(1.7, 0.3); // Actually, semi-normal and semi-cauchy because mu, sigma >= 0  
  sigma ~ cauchy(0,1);  
}
```

// Make sure program ends with a blank line

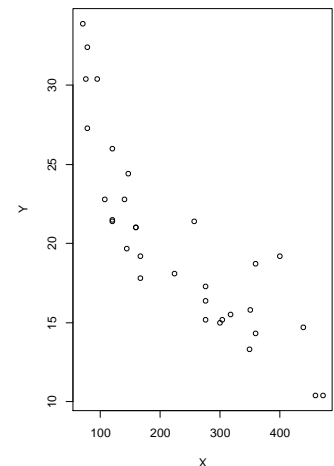
Linear Regression with Stan

- Dataset: mtcars

```
> head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

- Goal: predict mpg (y) as a function of displacement (x)
- $y = f(x) + \text{noise}$:
 - Assume gaussian noise $\sim N(0, \sigma)$
- $f(x) = B_0 + B_1 * x$
- Therefore: $p(y|x) = N(B_0 + B_1 * x, \sigma)$



```
// The input data is a vector 'Y' of length 'N'.
data {
  int<lower=0> N;
  vector[N] Y;
  vector[N] X;
}

// The parameters accepted by the model. Our model
// accepts two parameters 'b0' and 'b1'.
parameters {
  real b0;
  real b1;
  real<lower=0> sigma;
}

// The model to be estimated. We model the output
model {
  Y ~ normal(b0+b1*X, sigma);
}

// Make sure program ends with a blank line
```



```
library(rstan)
# We will use mtcars dataset
N <- nrow(mtcars)
Y <- mtcars$mpg
X <- mtcars$disp
```

```
# Plotting the mpg data
plot(X, Y)
```

```
# Stan configuration
cores <- parallel::detectCores()
options(mc.cores = cores)
# Avoid recompilation
rstan_options(auto_write = TRUE)
# Efficiency (but remove if errors reported)
Sys.setenv(LOCAL_CPPFLAGS = '-march=native')
```

```
#Compiling Stan model (it takes some time, seconds to minutes)
model <- stan_model('linear.stan')
fit <- sampling(model, list(N=N, X=X, Y=Y), iter = 1000, chains=cores)
```

```
print(fit)
params <- extract(fit)
```

```
hist(params$mu, prob=TRUE)
lines(density(params$mu), col="red")
hist(params$sigma, prob=TRUE)
lines(density(params$sigma), col="red")
```

```
> print(fit)
```

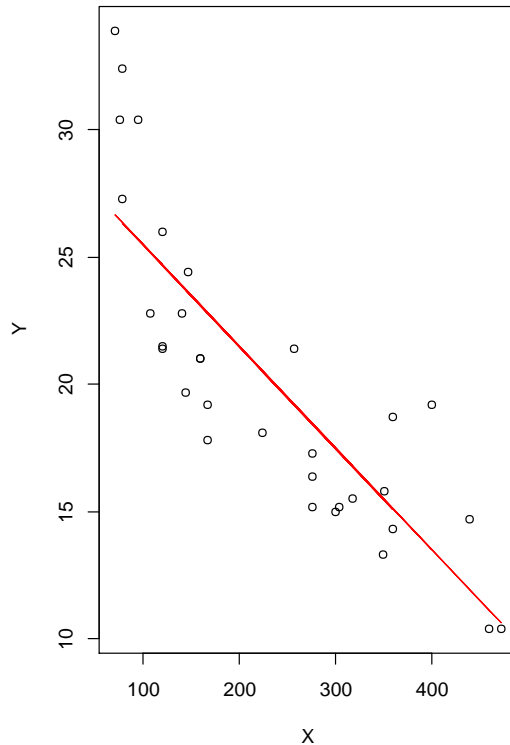
Inference for Stan model: linear.

8 chains, each with iter=1000; warmup=500; thin=1;

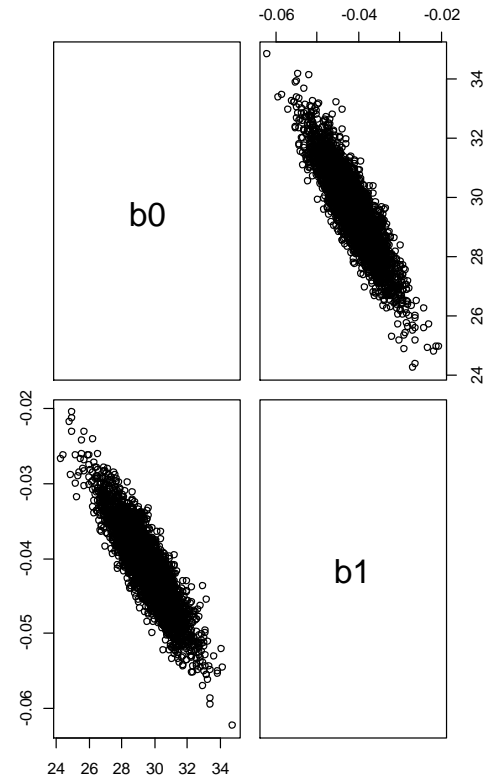
post-warmup draws per chain=500, total post-warmup draws=4000.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
b0	29.51	0.03	1.32	26.91	28.62	29.52	30.39	32.06	1939	1
b1	-0.04	0.00	0.01	-0.05	-0.04	-0.04	-0.04	-0.03	2045	1
sigma	3.40	0.01	0.46	2.65	3.08	3.34	3.66	4.47	1800	1

```
plot(X,Y)  
lines(X,29.51-0.04*X, col="red")
```



Joint distribution of b0 and b1



Exercise: quadratic Regression with Stan

- A quadratic model:
- $Y \sim \text{normal}(b_0 + b_1 * X + b_2 * \text{square}(X), \text{sigma});$

// The input data is a vector 'Y' of length 'N'.

```
data {  
  int<lower=0> N;  
  vector[N] Y;  
  vector[N] X;  
}
```

// The parameters accepted by the model. Our model

// accepts two parameters 'b0' and 'b1'.

```
parameters {  
  real b0;  
  real b1;  
  real b2;  
  real<lower=0> sigma;  
}
```

// The model to be estimated. We model the output

```
model {  
  Y ~ normal(b0+b1*X+b2*square(X), sigma);  
}
```

// Make sure program ends with a blank line

```
library(rstan)
# We will use mtcars dataset
N <- nrow(mtcars)
Y <- mtcars$mpg
X <- mtcars$disp

# Plotting the mpg data
plot(X, Y)

# Stan configuration
cores <- parallel::detectCores()
options(mc.cores = cores)
# Avoid recompilation
rstan_options(auto_write = TRUE)
# Efficiency (but remove if errors reported)
Sys.setenv(LOCAL_CPPFLAGS = '-march=native')

#Compiling Stan model (it takes some time, seconds to minutes)
model <- stan_model('linear.stan')
fit <- sampling(model, list(N=N, X=X, Y=Y), iter = 10000, chains=cores)
```

```
print(fit)
params <- extract(fit)

orderedX <- order(X)
X <- X[orderedX]
Y <- Y[orderedX]
plot(X,Y)
lines(X, 35.81-0.11*X+0.0001252116*X^2, col="red")
```

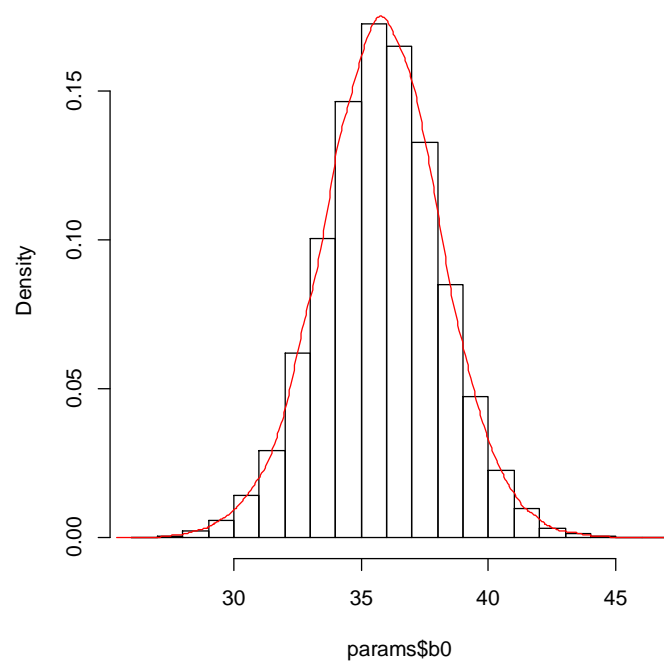
```
hist(params$b0, prob=TRUE)
lines(density(params$b0), col="red")
```

```
hist(params$b1, prob=TRUE)
lines(density(params$b1), col="red")
```

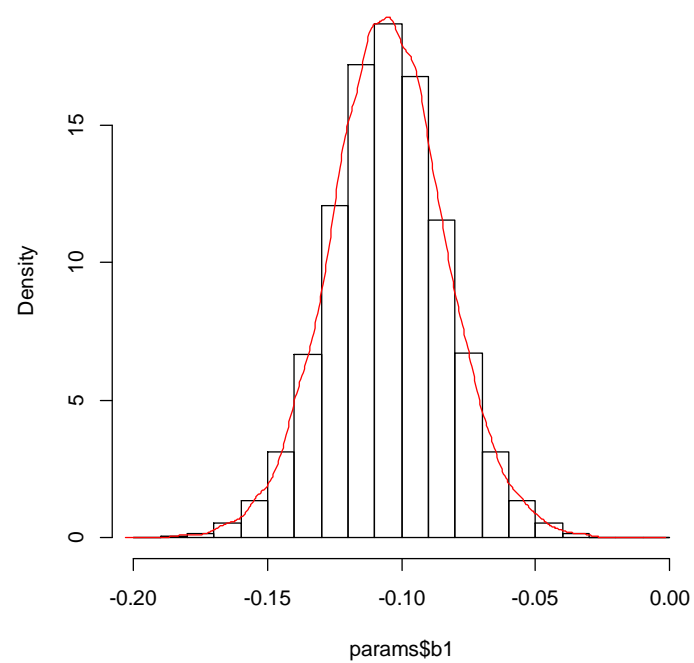
```
hist(params$b2, prob=TRUE)
lines(density(params$b2), col="red")
```

```
hist(params$sigma, prob=TRUE)
lines(density(params$sigma), col="red")
```

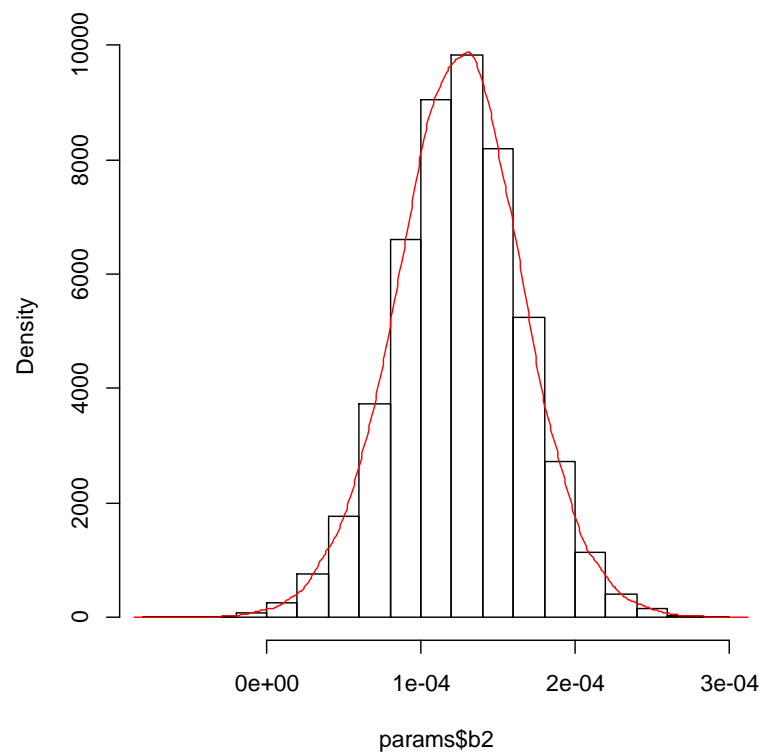
Histogram of params\$b0



Histogram of params\$b1



Histogram of params\$b2



Histogram of params\$sigma

