1.6 Exercise: first Stan model

Benjamin Rosenbaum

November 2, 2021

Exercise 1.

For the same dataset and statistical model as before, code the intercept and slope as a parameter vector of length 2: `vector[2] b;`

Exercise 2.

Fit a quadratic regression.

Exercise 3.

Fit the linear regression using different numbers of MCMC samples (few, many) on the same dataset. How does the posterior change?

Exercise 4.

Fit the linear regression using different numbers of observations (few, many), i.e. on different datasets. How does the posterior change?

Exercise 1: linear regression, code parameters as vector

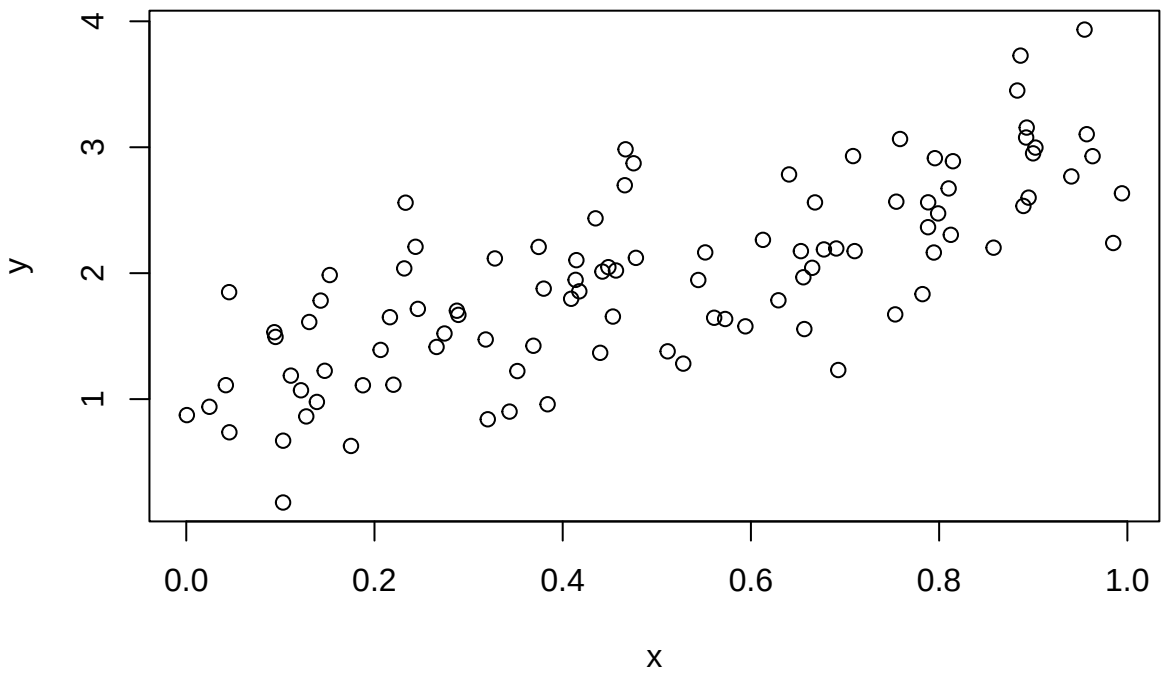
Setup:

```
rm(list=ls())
library(rstan)
library(coda)

rstan_options(auto_write = TRUE)
options(mc.cores = 4)
```

First, we have to generate the data

```
set.seed(123) # initiate random number generator for reproducibility

n=100

a=1
b=2
sigma=0.5

x = runif(n=n, min=0, max=1)
```

```r
y = rnorm(n=n, mean=a+b*x, sd=sigma)

df = data.frame(x=x,
                y=y)

plot(df)
```



The statistical model now reads

$$y_i \sim \text{normal}(\mu_i, \sigma)$$
$$\mu_i = b_1 + b_2 \cdot x_i$$

The parameter is defined as vector b in the parameter block: `vector[2] b;`

`b[1]` is the intercept, `b[2]` is the slope.

When defining the **prior distribution**, `b ~ normal(0, 10);` is now short version of:

```
b[1] ~ normal(0, 10);
b[2] ~ normal(0, 10);
```

Again, the likelihood definition `y ~ normal(b[1]+b[2]*x,sigma);` is short for `for(i in 1:n){ y[i]~normal(b[1]+b[2]*x[i],sigma); }`

```r
stan_code = '
data {
  int n;
  vector[n] x;
  vector[n] y;
}
parameters {
  vector[2] b;
  real<lower=0> sigma;
}
model {
  // priors
```

```
  b ~ normal(0, 10);
  sigma ~ normal(0, 10);
  // likelihood
  y ~ normal(b[1]+b[2]*x, sigma);
}
'
```

We wrap the data in a named list, compile the Stan code and fit the model

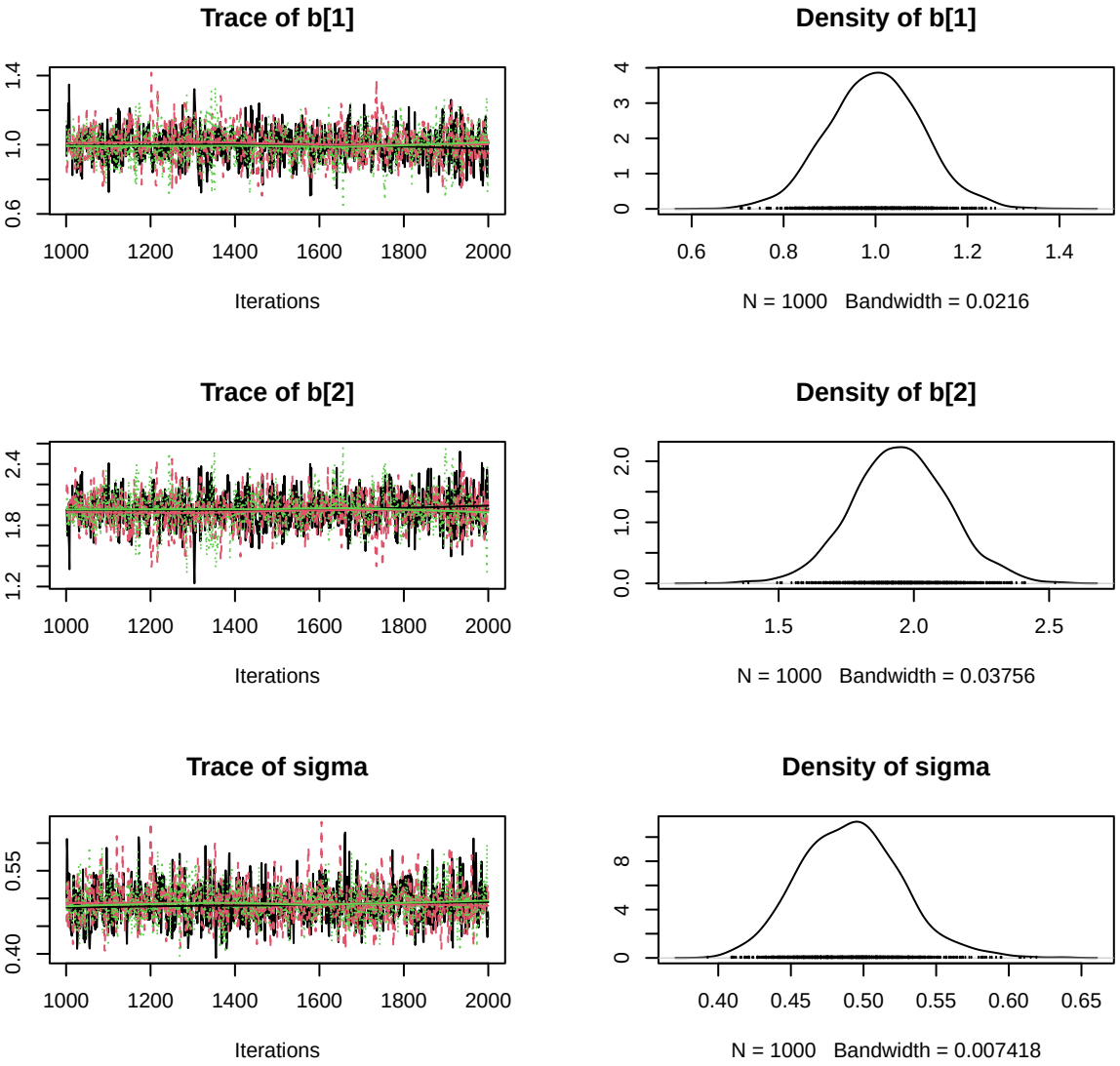
```
data = list(n=n,
            x=df$x,
            y=df$y)

stan_model = stan_model(model_code=stan_code)

fit  = sampling(stan_model,
                data=data,
                chains=3,
                iter=2000,
                warmup=1000
)
```

The results are the same as before, we just renamed the parameters.

```
print(fit, probs=c(0.025, 0.975))
```

```
## Inference for Stan model: 4dfb0786bf27b7170a78018ed44715ab.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##        mean se_mean   sd  2.5% 97.5% n_eff Rhat
## b[1]   1.00    0.00 0.10  0.80  1.20  1177    1
## b[2]   1.95    0.01 0.18  1.60  2.31  1216    1
## sigma  0.49    0.00 0.03  0.43  0.57  1410    1
## lp__  21.19    0.04 1.25 17.87 22.57  1010    1
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:42:30 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
posterior = As.mcmc.list(fit)
plot(posterior[, c("b[1]", "b[2]", "sigma")])
```



Exercise 2: quadratic regression.

The statistical model for quadratic regression reads

$$y_i \sim \text{normal}(\mu_i, \sigma)$$
$$\mu_i = b_1 + b_2 \cdot x_i + b_3 \cdot x_i^2$$

We define `vector[3] b;`in the parameters block containing intercept, effects of linear and quadratic term.

Again, in the prior definition `b ~ normal(0, 10);` is short for

```
b[1] ~ normal(0, 10);
b[2] ~ normal(0, 10);
b[3] ~ normal(0, 10);
```

Attention: we can't use the formulation `b[3]*x^2` or `b[3]*x*x` because these operations aren't defined for a vector `x` in Stan.

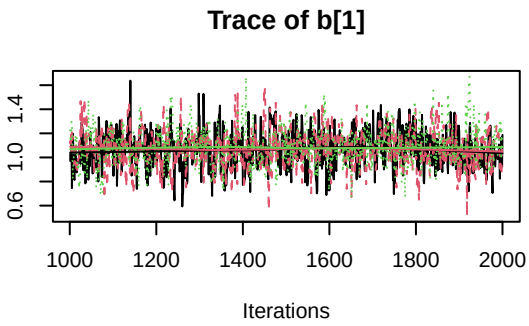
Either use the pointwise vector operation `x .* x` or a for loop!

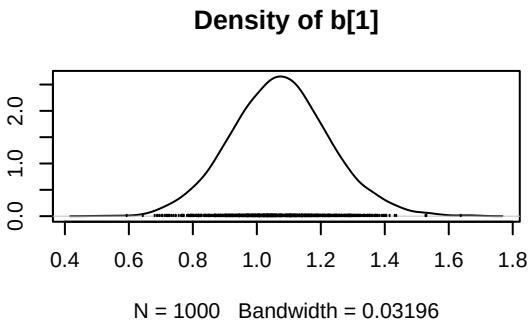
```
stan_code_2 = '
data {
  int n;
  vector[n] x;
  vector[n] y;
}
parameters {
  vector[3] b;
  real<lower=0> sigma;
}
model {
  // priors
  b ~ normal(0, 10);
  sigma ~ normal(0, 10);
  // likelihood
  // for(i in 1:n){
  //   y[i] ~ normal(b[1]+b[2]*x[i]+b[3]*x[i]^2, sigma);
  // }
  y ~ normal(b[1] + b[2]*x + b[3] * x .* x, sigma);
}
'

stan_model_2 = stan_model(model_code=stan_code_2)
```

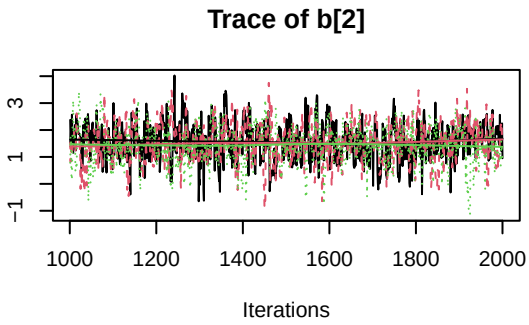
```
fit_2  = sampling(stan_model_2,
                  data=data,
                  chains=3,
                  iter=2000,
                  warmup=1000
)

print(fit_2, probs=c(0.025, 0.975))
```

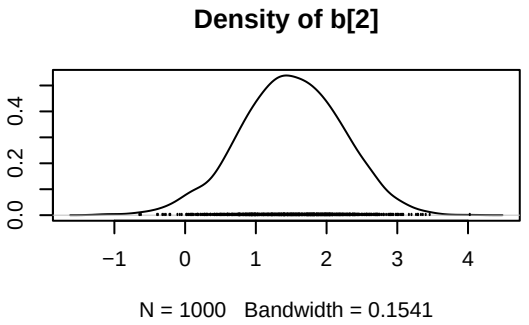
```
## Inference for Stan model: ff2beaa58bd7f85389051e018ed667cb.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##        mean se_mean   sd  2.5% 97.5% n_eff Rhat
## b[1]   1.07    0.01 0.15  0.78  1.39   809    1
## b[2]   1.49    0.03 0.72  0.03  2.88   734    1
## b[3]   0.46    0.02 0.70 -0.86  1.86   790    1
## sigma  0.49    0.00 0.04  0.43  0.56  1203    1
## lp__  20.91    0.05 1.45 17.49 22.68   891    1
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:43:09 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

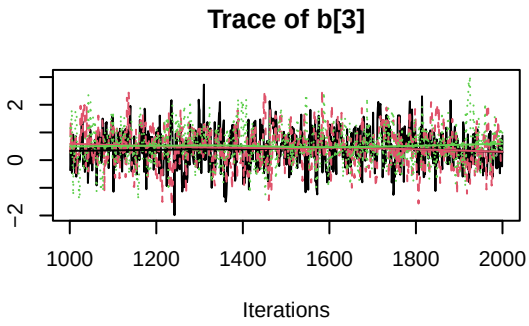
```
posterior_2 = As.mcmc.list(fit_2)

plot(posterior_2[, c("b[1]", "b[2]", "b[3]", "sigma")])
```

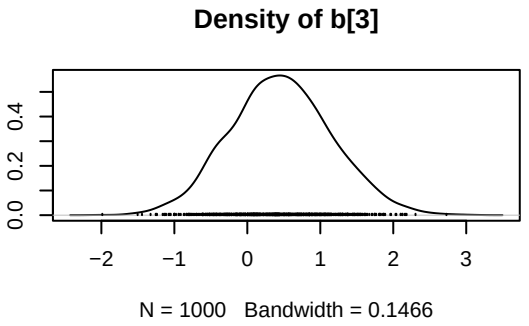


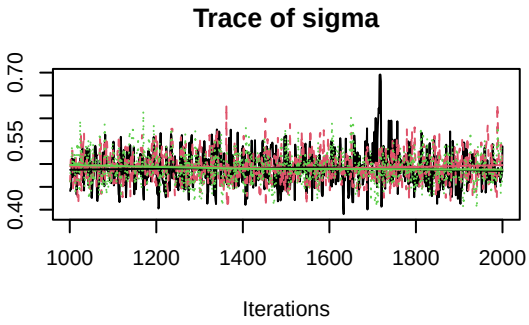


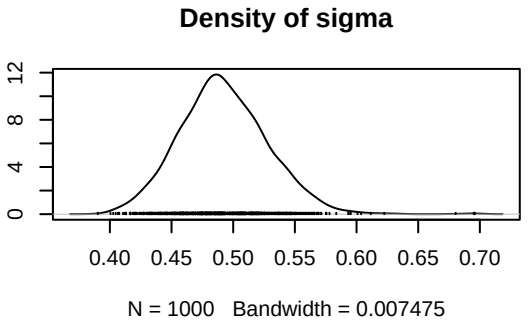












There does not seem to be much evidence for a quadratic effect, there is a lot of probability mass distributed around zero (more on that tomorrow).

Reminder: Do not interpret lower order effects ("main effects") independently from higher order effects!!

Exercise 3: linear regression, change number of MCMC samples

We fit the same model to the same data, using different number of posterior samples per chain (100, 1000, 10000).

(1000 already saved in `fit` object)

We don't have to recompile the model!

With larger sample size, we get a better approximation of the true posterior distribution.

```
fit_few_samples  = sampling(stan_model,
                            data=data,
                            chains=3,
                            iter=1100,
                            warmup=1000
)
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

```
fit_many_samples  = sampling(stan_model,
                             data=data,
                             chains=3,
                             iter=11000,
                             warmup=1000
)

print(fit_few_samples)
```

```
## Inference for Stan model: 4dfb0786bf27b7170a78018ed44715ab.
## 3 chains, each with iter=1100; warmup=1000; thin=1;
## post-warmup draws per chain=100, total post-warmup draws=300.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b[1]   0.98    0.01 0.10  0.81  0.91  0.98  1.05  1.16   193 1.00
## b[2]   1.97    0.01 0.17  1.64  1.84  1.98  2.09  2.28   182 1.00
## sigma  0.49    0.00 0.04  0.43  0.47  0.49  0.52  0.57   153 1.00
## lp__  21.18    0.12 1.24 17.93 20.64 21.49 22.04 22.59   106 1.01
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:43:10 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
print(fit)
```

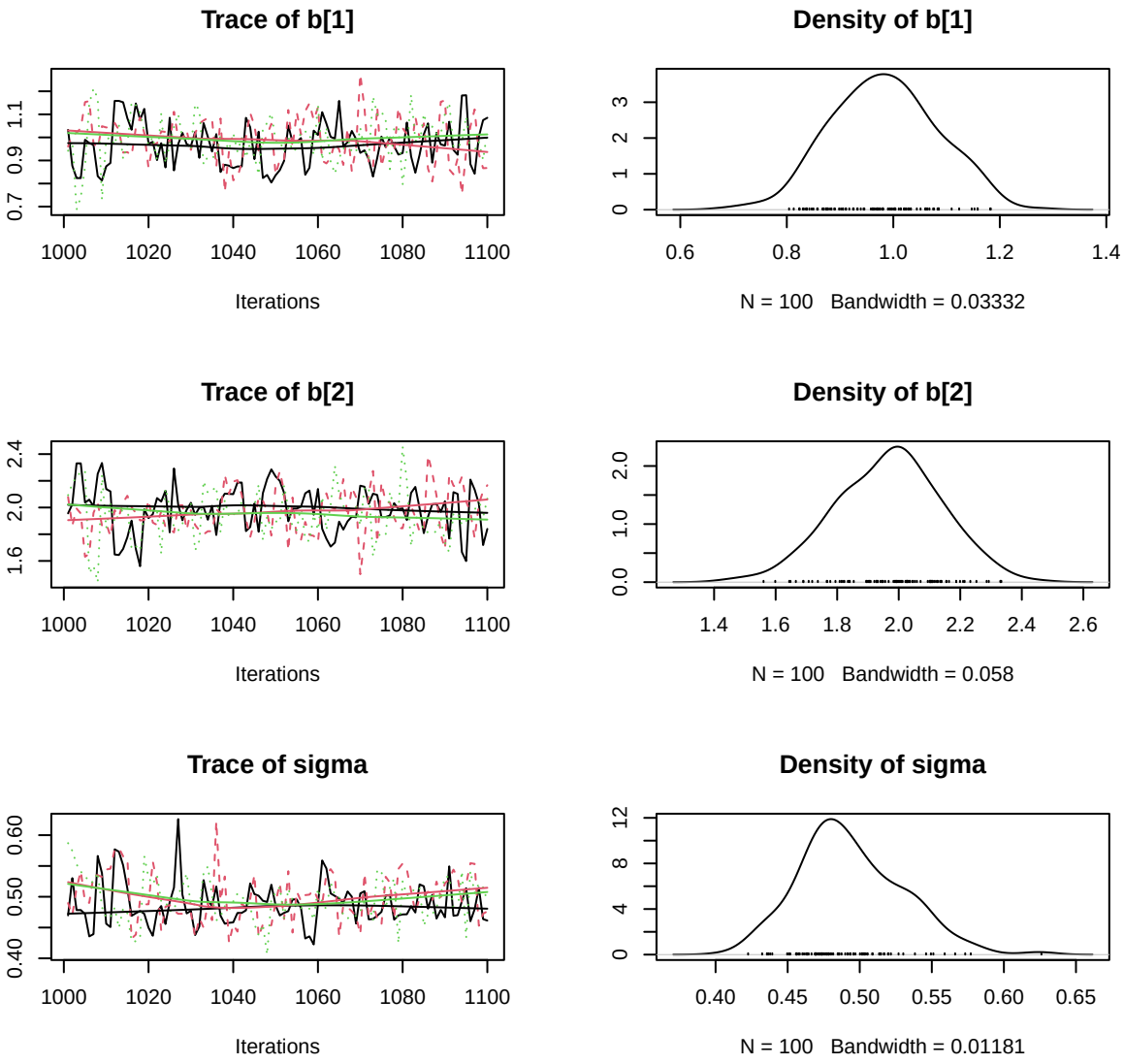
```
## Inference for Stan model: 4dfb0786bf27b7170a78018ed44715ab.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b[1]   1.00    0.00 0.10  0.80  0.93  1.00  1.07  1.20  1177    1
```

```
## b[2]   1.95    0.01 0.18  1.60  1.83  1.95  2.07  2.31  1216    1
## sigma  0.49    0.00 0.03  0.43  0.47  0.49  0.51  0.57  1410    1
## lp__  21.19    0.04 1.25 17.87 20.68 21.54 22.09 22.57  1010    1
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:42:30 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

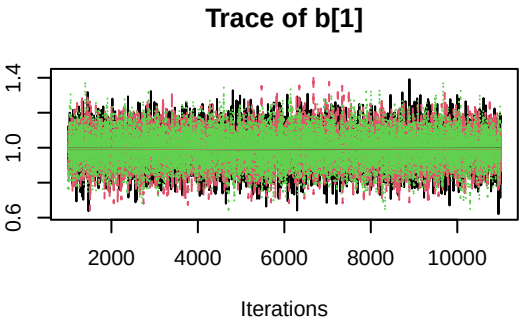
```
print(fit_many_samples)
```

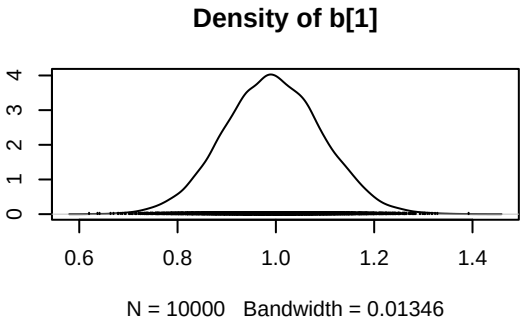
```
## Inference for Stan model: 4dfb0786bf27b7170a78018ed44715ab.
## 3 chains, each with iter=11000; warmup=1000; thin=1;
## post-warmup draws per chain=10000, total post-warmup draws=30000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b[1]   0.99    0.00 0.10  0.80  0.93  0.99  1.06  1.19 11991    1
## b[2]   1.96    0.00 0.17  1.61  1.84  1.96  2.07  2.29 12137    1
## sigma  0.49    0.00 0.04  0.43  0.47  0.49  0.51  0.57 15691    1
## lp__  21.17    0.01 1.23 17.98 20.61 21.49 22.07 22.58 11249    1
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:43:12 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

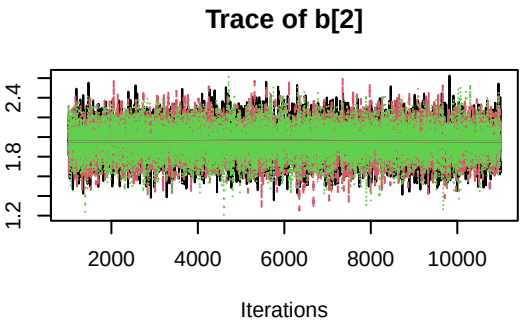
```
plot(As.mcmc.list(fit_few_samples)[, c("b[1]", "b[2]", "sigma")])
```

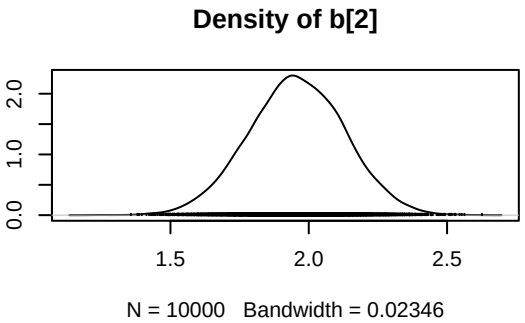


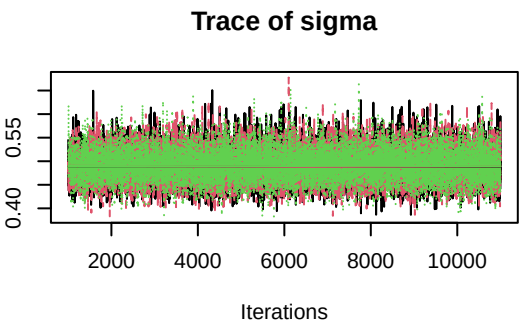

```
plot(As.mcmc.list(fit_many_samples)[, c("b[1]", "b[2]", "sigma")])
```

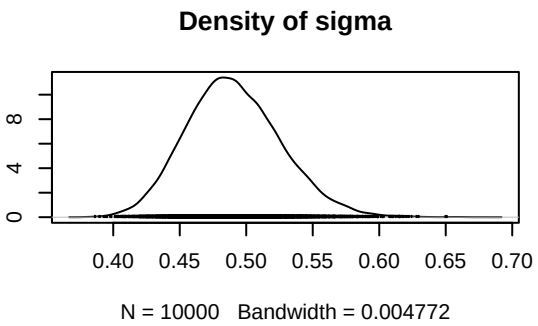












Exercise 4: linear regression, change number of observations (data)

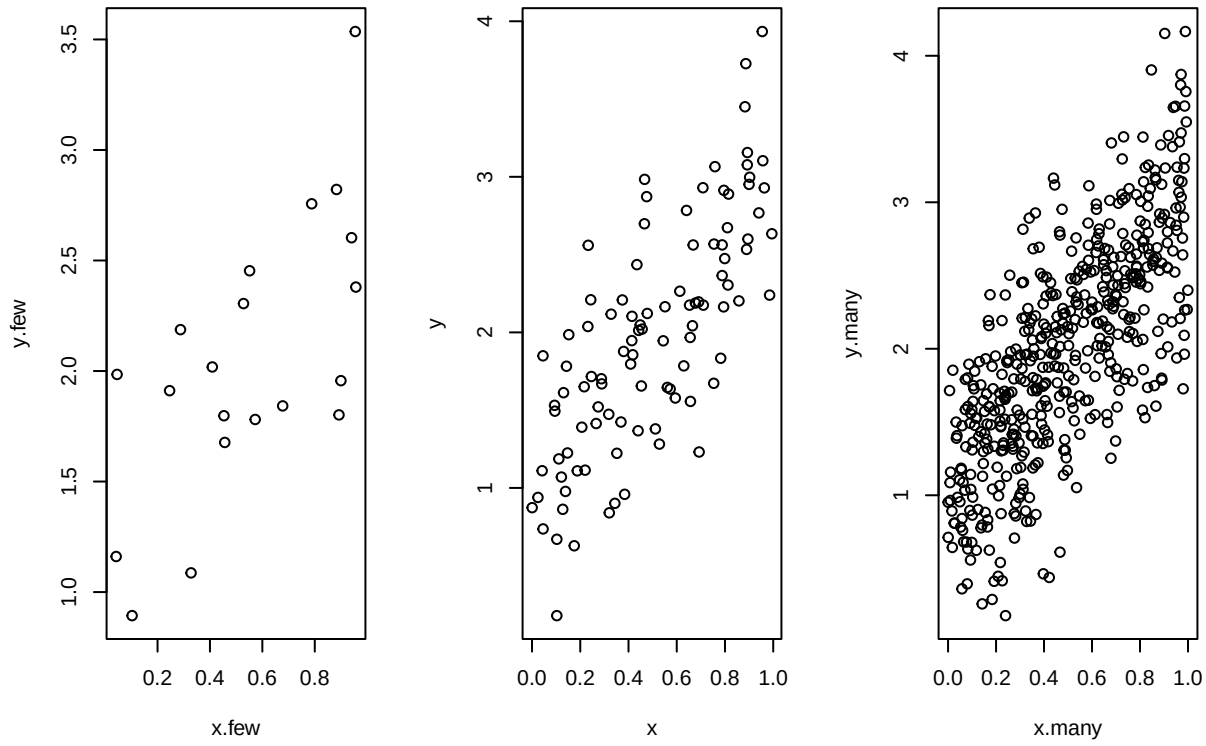
We generate new datasets using identical (underlying) parameters, but using different numbers of observations (20, 100, 500).

With larger number of observations, we get lower posterior standard deviations, i.e. lower uncertainty (higher certainty) in parameter estimation.

More information in the data (in the likelihood) means a more narrow (more informative) posterior distribution.

```
set.seed(123) # initiate random number generator for reproducability

n.few = 20
x.few = runif(n=n.few, min=0, max=1)
y.few = rnorm(n=n.few, mean=a+b*x.few, sd=sigma)

n.many = 500
x.many = runif(n=n.many, min=0, max=1)
y.many = rnorm(n=n.many, mean=a+b*x.many, sd=sigma)
```

```
par(mfrow=c(1,3))
plot(x.few,y.few)
plot(x,y)
plot(x.many,y.many)
```



We wrap the datasets in named lists and fit the linear model. (`n=100` already saved in `fit` object)

```
data.few = list(n=n.few,
                x=x.few,
                y=y.few)

data.many = list(n=n.many,
                 x=x.many,
                 y=y.many)

fit_few_obs  = sampling(stan_model,
                        data=data.few,
                        chains=3,
                        iter=2000,
                        warmup=1000
)

fit_many_obs  = sampling(stan_model,
                         data=data.many,
                         chains=3,
                         iter=2000,
                         warmup=1000
)

print(fit_few_obs)
```

```
## Inference for Stan model: 4dfb0786bf27b7170a78018ed44715ab.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##       mean se_mean   sd  2.5%  25%  50%  75% 97.5% n_eff Rhat
## b[1]  1.30    0.01 0.25  0.82 1.15 1.31 1.46  1.79  1137    1
## b[2]  1.35    0.01 0.39  0.56 1.10 1.34 1.59  2.13  1097    1
## sigma 0.52    0.00 0.10  0.37 0.45 0.50 0.57  0.74  1132    1
## lp__  3.45    0.05 1.40 -0.05 2.84 3.81 4.47  5.00   854    1
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:43:17 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
print(fit)
```

```
## Inference for Stan model: 4dfb0786bf27b7170a78018ed44715ab.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b[1]   1.00    0.00 0.10  0.80  0.93  1.00  1.07  1.20  1177    1
## b[2]   1.95    0.01 0.18  1.60  1.83  1.95  2.07  2.31  1216    1
## sigma  0.49    0.00 0.03  0.43  0.47  0.49  0.51  0.57  1410    1
## lp__  21.19    0.04 1.25 17.87 20.68 21.54 22.09 22.57  1010    1
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:42:30 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

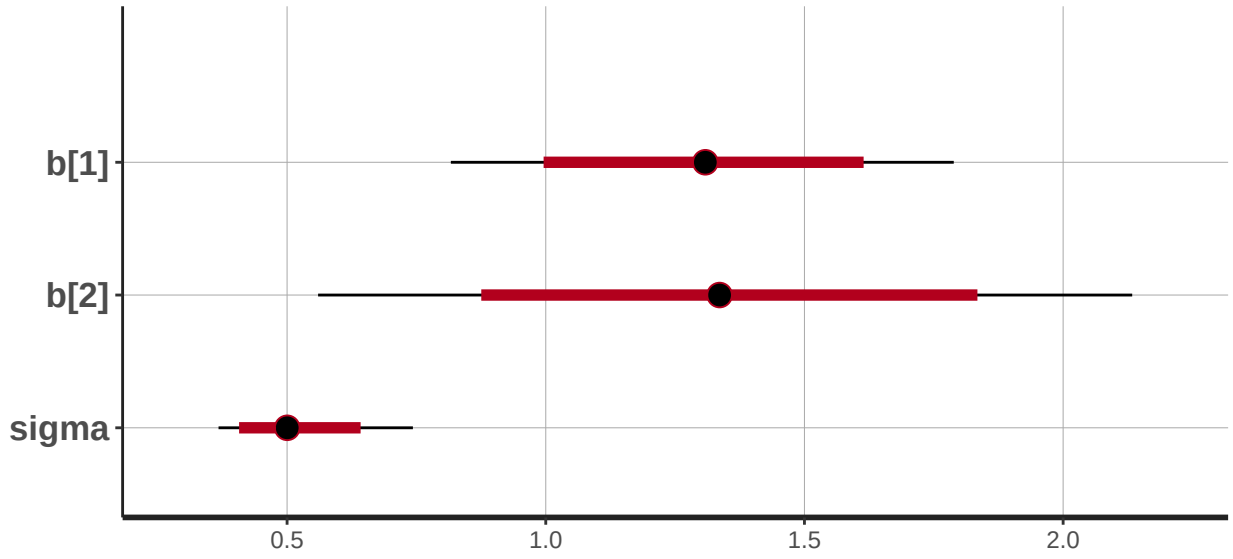
```
print(fit_many_obs)
```

```
## Inference for Stan model: 4dfb0786bf27b7170a78018ed44715ab.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b[1]   1.04    0.00 0.05  0.95  1.01  1.04  1.07  1.13  1260    1
## b[2]   1.93    0.00 0.08  1.78  1.88  1.93  1.99  2.09  1189    1
## sigma  0.50    0.00 0.02  0.47  0.49  0.50  0.51  0.54  1494    1
## lp__  91.73    0.04 1.22 88.46 91.16 92.04 92.63 93.16  1120    1
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 19 14:43:18 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
plot(fit_few_obs)
```

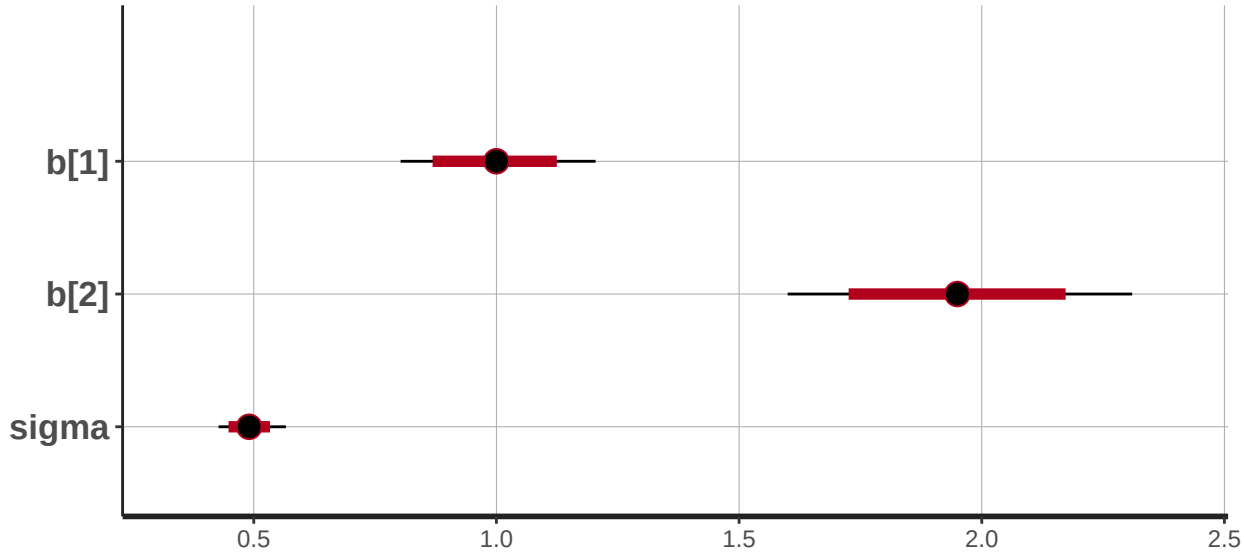
```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```



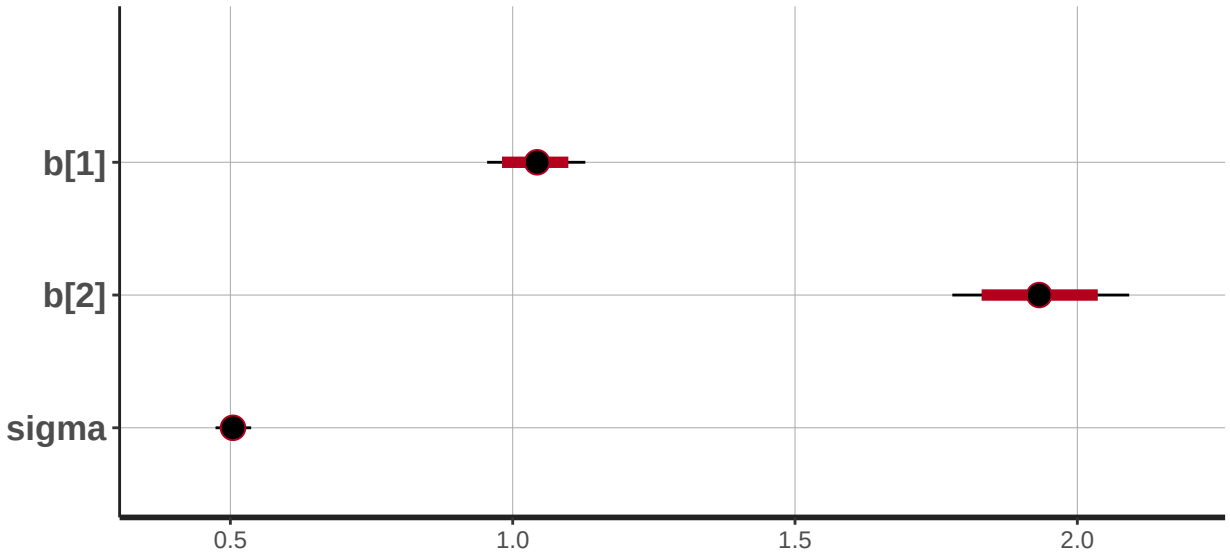

```
plot(fit)
```

```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```




```
plot(fit_many_obs)
```

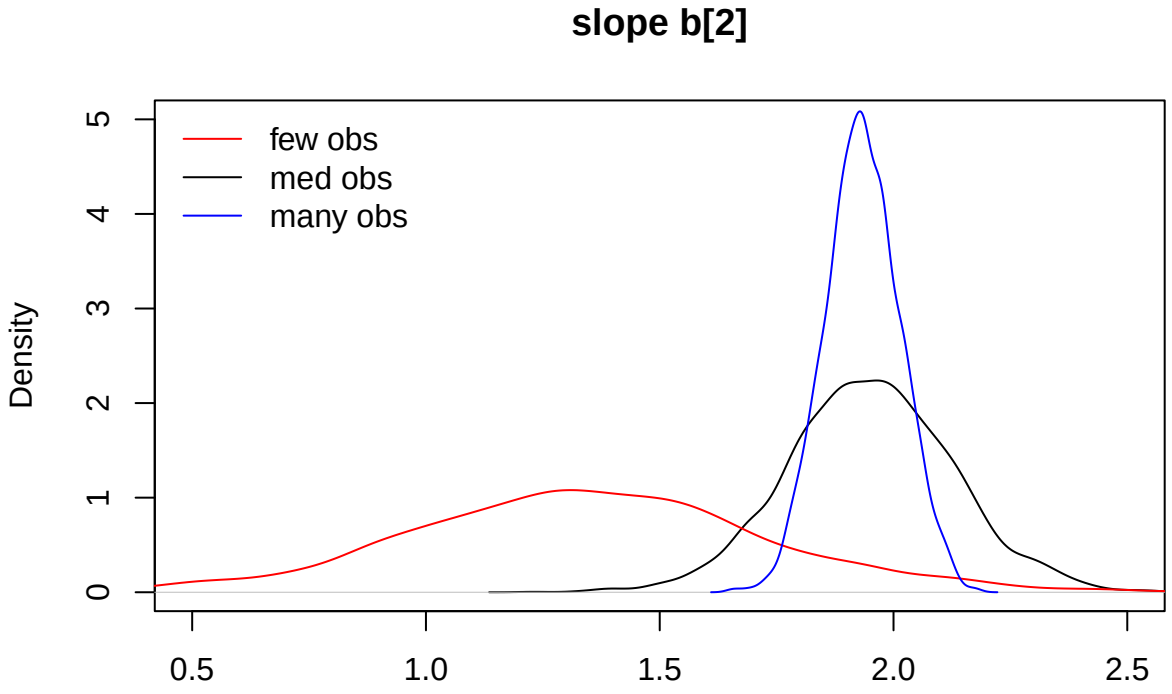
```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```



We can also extract the posterior density for single parameters, e.g. the slope `b[2]`, and see how the posterior changes with size of the data.

(More on handling the posterior distribution tomorrow)

```
posterior=as.matrix(fit)
posterior_few_obs=as.matrix(fit_few_obs)
posterior_many_obs=as.matrix(fit_many_obs)

density_1=density(posterior[, "b[2]"])
density_few_obs=density(posterior_few_obs[, "b[2]"])
density_many_obs=density(posterior_many_obs[, "b[2]"])

par(mfrow=c(1,1))

plot(density_1, xlim=c(0.5,2.5), ylim=c(0,5), main="slope b[2]", xlab="")
lines(density_few_obs, col="red")
lines(density_many_obs, col="blue")
legend("topleft", bty="n", legend=c("few obs","med obs","many obs"), lty=c(1,1,1), col=c("red","black",
```


slope b[2]
few obs
med obs
many obs
Density
0
1
2
3
4
5
0.5
1.0
1.5
2.0
2.5