Practical 7: Introduction to Stan

Benjamin Rosenbaum

Table of contents

We code some classical models in Stan, using the rstan package. We learn how to interpret and assess Stan model output. Posterior predictions unfortunately have to be coded manually.

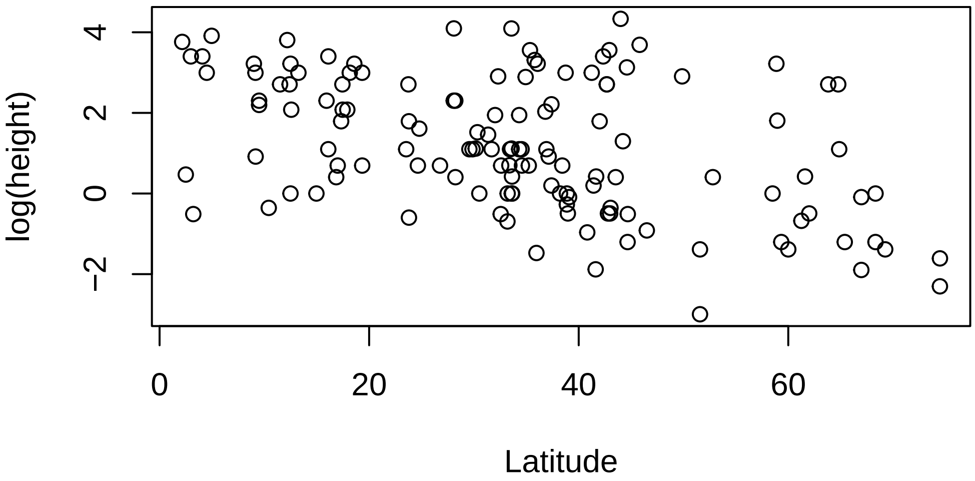
```r
rm(list=ls())
library("rstan")
library("loo")
library("ecostats")
library("Data4Ecologists")
library("bayesplot")
library("ggplot2")
library("arm")
library("RColorBrewer")
try(dev.off())
```

Linear regression

Data preparation, model coding & fitting

We go back to the globalPlants dataset and perform a simple linear regression of log(height) vs latitude (scaled predictor).

```
data("globalPlants")
data = globalPlants
plot(data$lat, log(data$height), xlab="Latitude", ylab="log(height)")
```



Data must be but in a named list for Stan. Names must be identical with names in Stan code.

```
stan.data = list(N = nrow(data),
                 x = as.vector(scale(data$lat)), # convert to vector to get rid of additional
                 y = log(data$height))
str(stan.data)
```

```
List of 3
 $ N: int 131
 $ x: num [1:131] 0.592 -1.262 -0.598 -0.097 0.421 ...
 $ y: num [1:131] 3.129 3.807 -0.598 -0.511 -1.881 ...
```

This is the stan code stored in a file `lm.stan`:

```r
cat(readChar("lm.stan", 1e6))
```

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real a;
  real b;
  real<lower=0> sigma;
}
model {
  a ~ normal(0,1);
  b ~ normal(0,1);
  sigma ~ exponential(1.0);
  y ~ normal(a+b*x, sigma);
}
```

The `stan()` function reads the Stan code, compiles it to C++ and runs MCMC.

```r
fit1 = stan(file="lm.stan", data=stan.data)
```

Model output

Model output shows estimated parameters & their statistics only. It does not show infos on model structure as brms summary output, because Stan does not know anything about model structure. Stan code is just a recipe for calculating the posterior value: parameters values go in, posterior value goes out.

```r
print(fit1, probs=c(0.05, 0.95))
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd      5%     95% n_eff Rhat
a         1.18    0.00 0.13    0.97    1.40  4165    1
b        -0.73    0.00 0.13   -0.94   -0.51  4197    1
sigma     1.49    0.00 0.09    1.34    1.65  3812    1
```

```
lp__  -119.73    0.03 1.24 -122.17 -118.38  1670    1

Samples were drawn using NUTS(diag_e) at Tue Feb  4 16:24:43 2025.
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).
```

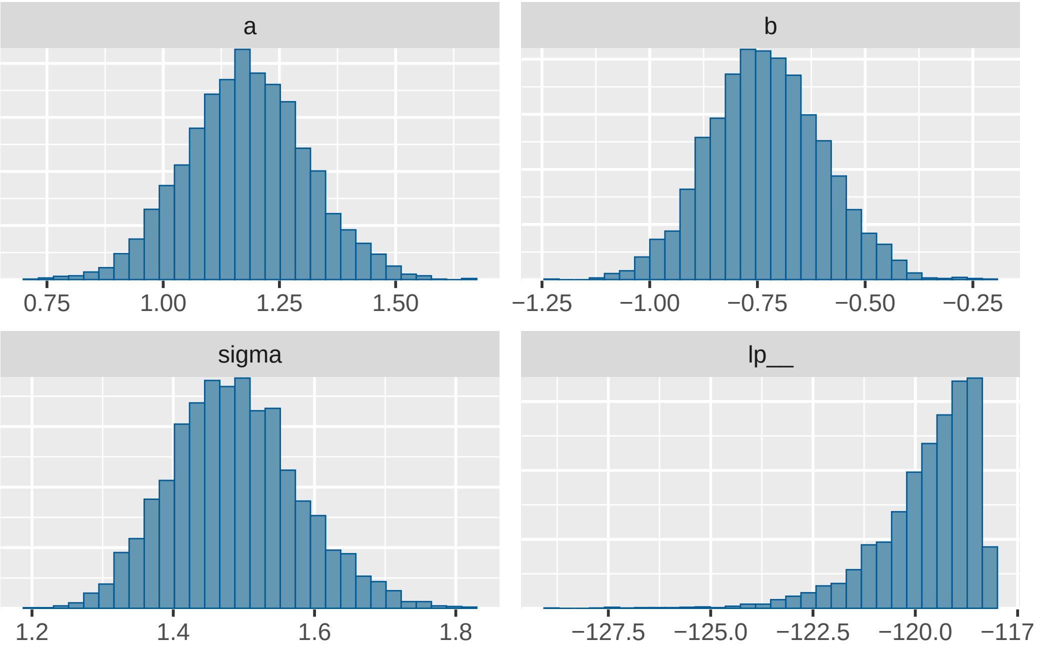
Summary statistics can be extracted in a dataframe / matrix.

```r
summ.table = summary(fit1, probs=c(0.05, 0.95))$summary
print(summ.table, digits=3)
```

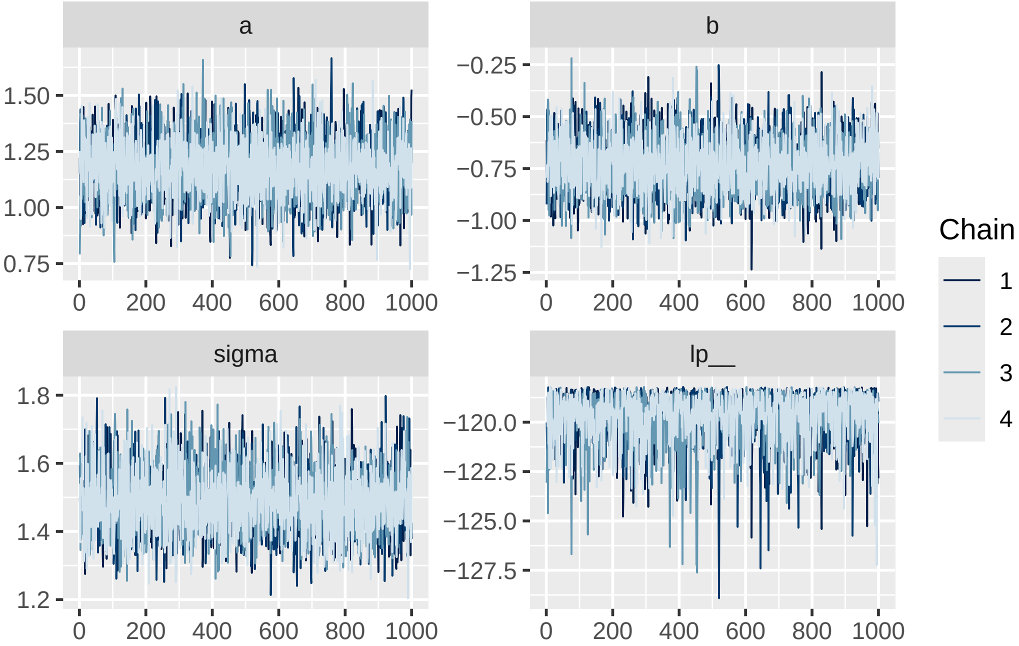
```
          mean se_mean     sd      5%      95% n_eff  Rhat
a        1.180 0.00202 0.1303    0.967    1.400  4165 1.000
b       -0.729 0.00204 0.1320   -0.939   -0.507  4197 1.000
sigma    1.487 0.00147 0.0909    1.344    1.645  3812 0.999
lp__  -119.727 0.03045 1.2441 -122.173 -118.377  1670 1.002
```

The bayesplot package offers plotting functions which are also used in brms. Alternatively, you can use `stan_trace()`, `stan_hist()`, `stan_dens()` … from the rstan package.

```r
mcmc_hist(fit1)
```

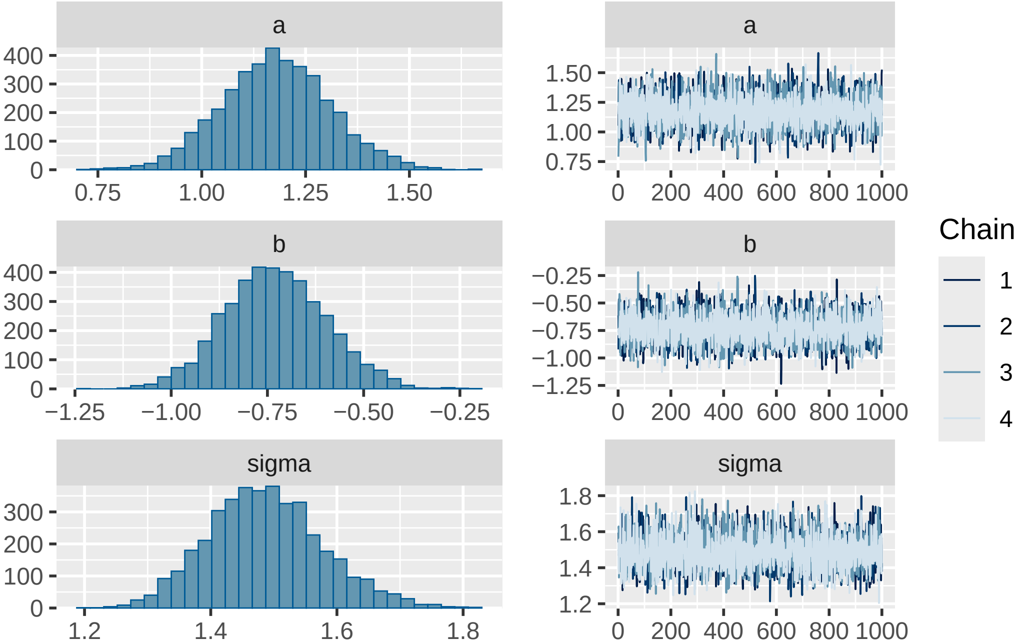


```r
mcmc_trace(fit1)
```



Side-by-side as in brms output looks better:

```r
mcmc_combo(fit1, combo=c("hist","trace"), pars=c("a","b","sigma"))
```



Posterior predictions

Unfortunately, there is no fancy `conditional_effects()` or `predict()` function for rstan (remember Stan does not know about model structure). So we have to extract the posterior distribution and compute posterior predictions manually.

Each row = 1 sample, each column = 1 parameter

```r
post = as.matrix(fit1)
head(post[, 1:3])
```

```
          parameters
iterations         a          b    sigma
      [1,] 1.2210534 -0.6133009 1.499114
      [2,] 1.0549519 -0.7954653 1.467821
      [3,] 1.1160180 -0.7121111 1.577292
      [4,] 1.0445726 -0.4826395 1.484276
      [5,] 0.9567048 -0.5379022 1.476656
      [6,] 1.3082696 -0.9188406 1.541856
```

Start by setting up a range of predictor values for plotting. Here we have only 1 predictor, latitude. For multiple predictors, a fixed level for other predictor(s) has to be chosen.

```r
xmin = min(stan.data$x)
xmax = max(stan.data$x)
x.pred = seq(xmin, xmax, length.out=100)
```

Fitted (deterministic part)

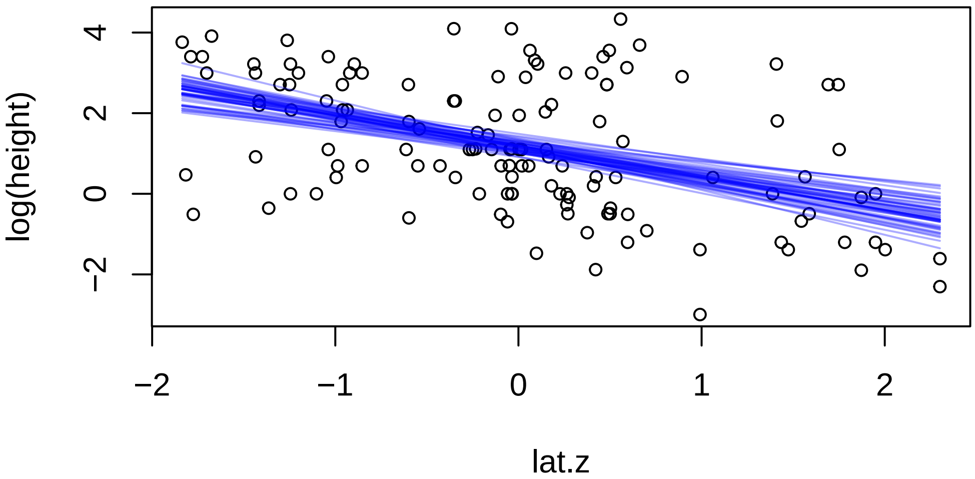
We initialize an empty matrix. Number of rows = number of posterior samples, number of columns = number of predictions. Each row will contain 1 regression line, each column will contain the posterior fitted distribution for 1 predictor value. A loop over all posterior samples makes predictions for all predictor values.

```r
y.fit = matrix(NA, nrow=nrow(post), ncol=length(x.pred) )
for(i in 1:nrow(post)){
  y.fit[i, ] = post[i,"a"] + post[i,"b"]*x.pred
}
str(y.fit)
```

```
 num [1:4000, 1:100] 2.35 2.52 2.42 1.93 1.94 ...
```

Use 50 random samples for a spaghetti plot

```
plot(stan.data$x, stan.data$y, xlab="lat.z", ylab="log(height)", cex=0.8)
draws = sample(1:nrow(post), size=50)
for(i in draws){
  lines(x.pred, y.fit[i, ], col=adjustcolor("blue", alpha.f=0.33))
}
```

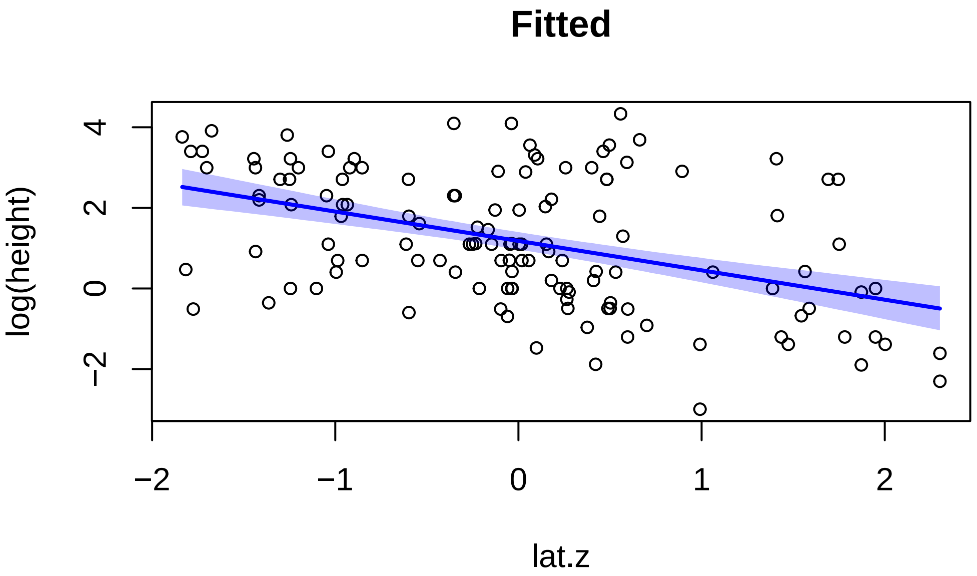


We extract mean and 90%-CIs of fitted vales. Each column in the prediction matrix is summarized: `apply(..., margin=2, function() ... )`

```
y.fit.mean = apply(y.fit, 2, function(x) mean(x))
y.fit.q05 = apply(y.fit, 2, function(x) quantile(x, probs=0.05))
y.fit.q95 = apply(y.fit, 2, function(x) quantile(x, probs=0.95))
```

A classical fitted plot with mean and 90%-CIs.

```
plot(stan.data$x, stan.data$y, xlab="lat.z", ylab="log(height)", cex=0.8, main="Fitted")
polygon(c(x.pred, rev(x.pred)),
        c(y.fit.q05, rev(y.fit.q95)),
        border = NA,
        col = adjustcolor("blue", alpha.f=0.25))
lines(x.pred, y.fit.mean, col="blue", lwd=2)
```



Predicted (deterministic & stochastic part)

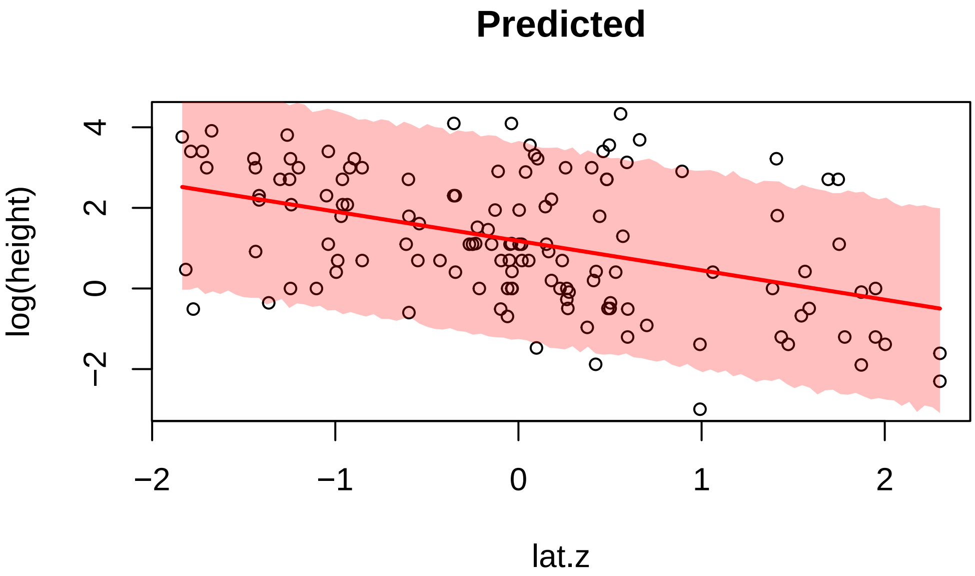
Again, we start with an empty matrix and predict data based on statistical model (normal distr. around fitted value `y.fit` from above):

```
y.pred = matrix(NA, nrow=nrow(post), ncol=length(x.pred) )
for(i in 1:nrow(post)){
  y.pred[i, ] = rnorm(n = length(x.pred),
                      mean = y.fit[i, ],
                      sd = post[i, "sigma"] )
}
```

Just as with the fitted values, we extract mean & CIs for a classical predicted plot

```
y.pred.mean = apply(y.pred, 2, function(x) mean(x))
y.pred.q05 = apply(y.pred, 2, function(x) quantile(x, probs=0.05))
y.pred.q95 = apply(y.pred, 2, function(x) quantile(x, probs=0.95))

plot(stan.data$x, stan.data$y, xlab="lat.z", ylab="log(height)", cex=0.8, main="Predicted")
polygon(c(x.pred, rev(x.pred)),
        c(y.pred.q05, rev(y.pred.q95)),
        border = NA,
        col = adjustcolor("red", alpha.f=0.25))
lines(x.pred, y.fit.mean, col="red", lwd=2)
```

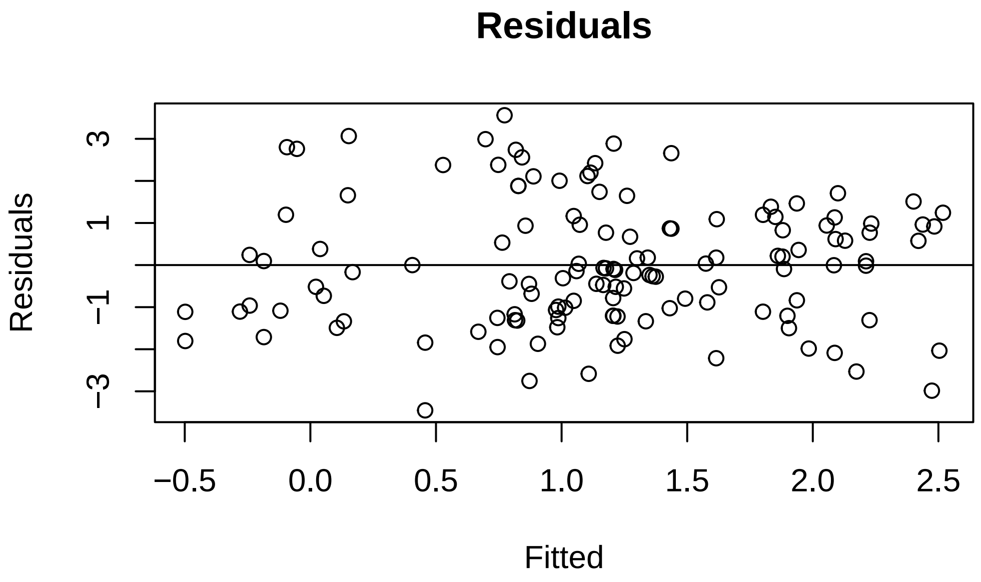


Predictions (on data level)

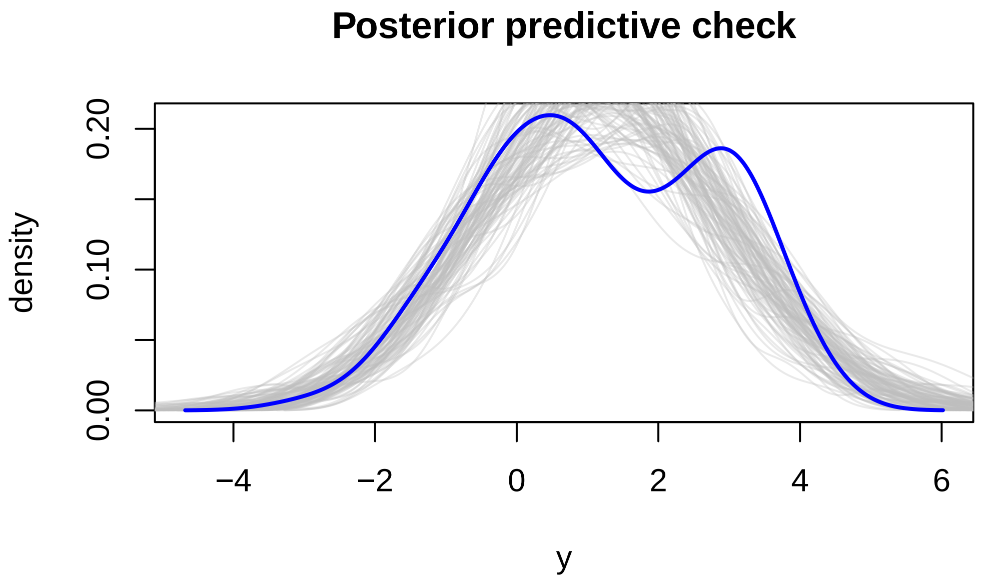
For computing predictions and residuals for the actual datapoints, we use the same code as above:

```
x.pred = stan.data$x
y.fit  = matrix(NA, nrow=nrow(post), ncol=length(x.pred) )
y.pred = matrix(NA, nrow=nrow(post), ncol=length(x.pred) )
for(i in 1:nrow(post)){
  y.fit[i, ]  = post[i,"a"] + post[i,"b"]*x.pred
  y.pred[i, ] = rnorm(n=length(x.pred),
                      mean = y.fit[i, ],
                      sd = post[i,"sigma"])
}
```

```
y.fit.mean = apply(y.fit, 2, function(x) mean(x))
residuals = stan.data$y - y.fit.mean
plot(y.fit.mean, residuals, xlab="Fitted", ylab="Residuals", main="Residuals")
abline(0,0)
```



```
dens.data = density(stan.data$y)
plot(dens.data$x, dens.data$y, type="n", xlab="y", ylab="density", main="Posterior predictive
draws = sample(1:nrow(post), size=100)
for(i in draws){
  dens = density(y.pred[i, ])
  lines(dens$x, dens$y, col=adjustcolor("grey", alpha.f=0.33))
}
lines(dens.data$x, dens.data$y, col="blue", lwd=2)
```



Generalized linear model

We go back to the mice dose-infection model, where number of inoculated and infected mice are recorded together with the parasite infection dose.

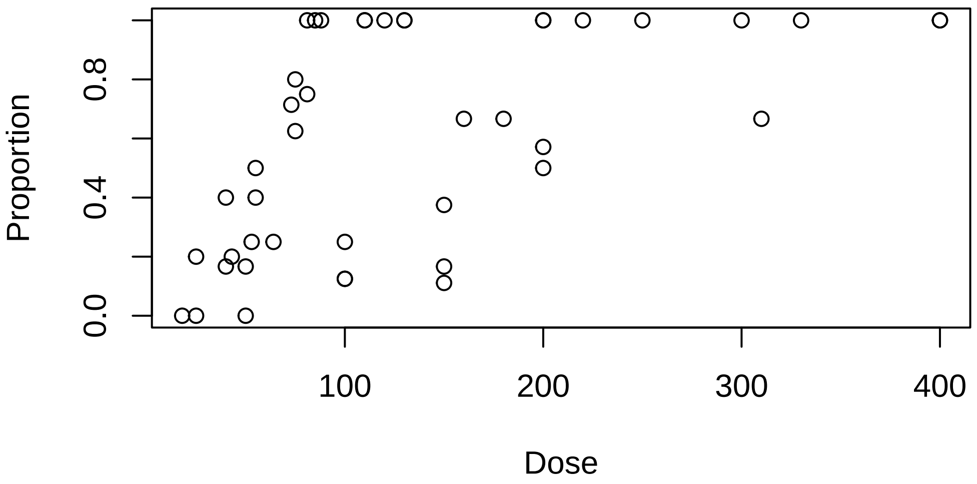
This is a classical logistic regression (linear model, logit-link, Binomial distribution). Stan actually doesn't care about linearity, we just have to provide code that computes the deterministic part of the model based on predictor values, and a distribution for the stochastic part of the model.

Deterministic part: $\text{logit}(\mu) = a + b \cdot Dose$
Stochastic part: $y \sim \text{Binomial}(Total, \mu)$

The deterministic part is written in its implicit formulation (linear function on right hand side), but we need the explicit formulation for coding it. Stan has a function for the inverse-logit:

Deterministic part: $\mu = \text{inv\_logit}(a + b \cdot Dose)$

```
df = read.csv("https://raw.githubusercontent.com/songsqian/eesR/refs/heads/master/R/Data/cryp
              sep=" ")
data = subset(df, Source=="Finch")
plot(data$Dose, data$Y/data$N, xlab="Dose", ylab="Proportion")
```



```
stan.data = list(N = nrow(data),
                 dose = as.vector(scale(data$Dose)), # get rid of additional infos from scale
                 total = as.integer(data$N),
                 y = as.integer(data$Y))
str(stan.data)
```

```
List of 4
 $ N    : int 43
 $ dose : num [1:43] -1.181 -1.11 -1.11 -0.958 -0.958 ...
 $ total: int [1:43] 9 10 6 6 5 5 8 6 4 5 ...
 $ y    : int [1:43] 0 2 0 1 2 1 0 1 1 2 ...
```

It's important to declare `total` and `y` as integers in Stan, otherwise the Binomial distribution doesn't work.

In Stan, we loop over all datapoints. Deterministic model part `mu` is computed using the `inv_logit` function, and the likelihood is evaluated. Alternatively, there is a vectorized form that automatically applies the `inv_logit` to the linear model part and evaluates likelihood for all observations: `y ~ binomial_logit(total, dose, a, b);`

```r
cat(readChar("glm.stan", 1e6))
```

```
data {
  int<lower=0> N;
  vector[N] dose;
  array[N] int total;
  array[N] int y;
}
parameters {
  real a;
  real b;
}
model {
  real mu;
  a ~ normal(0,1);
  b ~ normal(1,1);
  for(i in 1:N){
    mu = inv_logit(a+b*dose[i]);
    y[i] ~ binomial(total[i], mu);
  }
  // short: y ~ binomial_logit(total, dose, a, b);
}
```

```r
fit2 = stan(file="glm.stan", data=stan.data)
```

Check model output and convergence

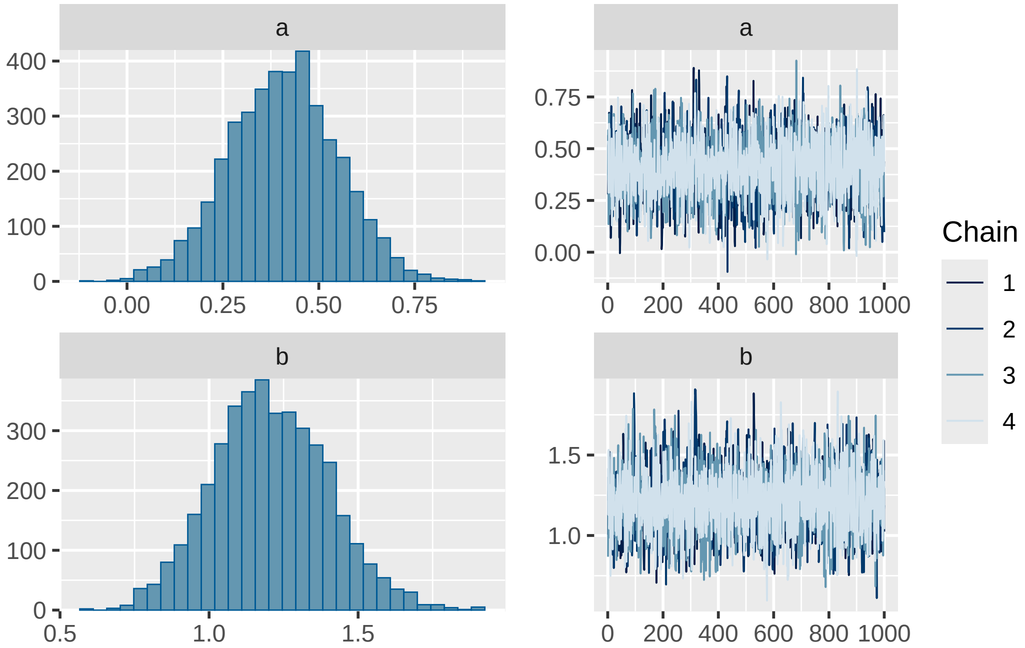
```
print(fit2, probs=c(0.05, 0.95))
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean   sd      5%     95% n_eff Rhat
a       0.41    0.00 0.14    0.17    0.64  2649    1
b       1.21    0.00 0.19    0.90    1.53  2812    1
lp__ -164.23    0.03 1.02 -166.27 -163.26  1567    1

Samples were drawn using NUTS(diag_e) at Wed Feb  5 10:49:43 2025.
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).
```

```
mcmc_combo(fit2, combo=c("hist","trace"), pars=c("a","b"))
```



Posterior predictions. Here we use the `invlogit()` function from the arm package

```
post = as.matrix(fit2)
head(post[, 1:2])
```

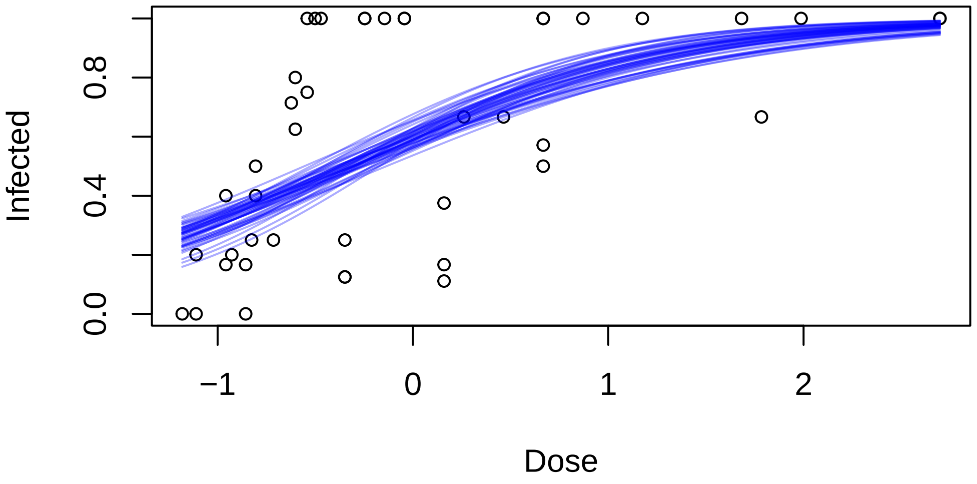
```
          parameters
iterations         a        b
      [1,] 0.2791467 1.005077
      [2,] 0.3974638 1.276939
      [3,] 0.3314378 1.132588
      [4,] 0.2009294 1.491483
      [5,] 0.6737583 1.475006
      [6,] 0.4979883 1.521034
```

```
xmin = min(stan.data$dose)
xmax = max(stan.data$dose)
x.pred = seq(xmin, xmax, length.out=100)
y.fit = matrix(NA, nrow=nrow(post), ncol=length(x.pred) )
for(i in 1:nrow(post)){
  y.fit[i, ] = invlogit(post[i,"a"] + post[i,"b"]*x.pred)
}
str(y.fit)
```

```
 num [1:4000, 1:100] 0.287 0.248 0.268 0.173 0.256 ...
```

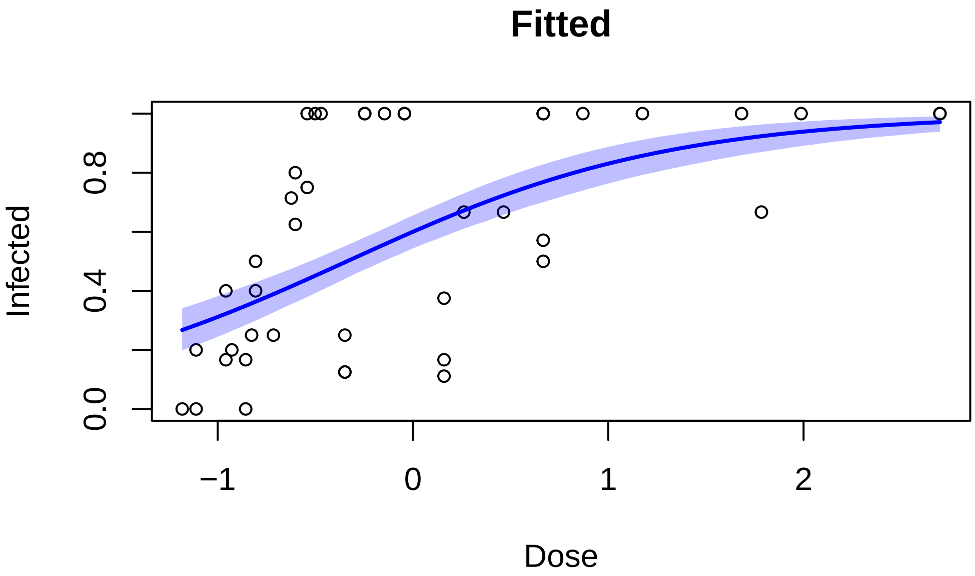
Spaghetti plot. Each posterior sample generates a regression curve

```
plot(stan.data$dose, stan.data$y/stan.data$total, xlab="Dose", ylab="Infected", cex=0.8)
draws = sample(1:nrow(post), size=50)
for(i in draws){
  lines(x.pred, y.fit[i, ], col=adjustcolor("blue", alpha.f=0.33))
}
```



Extract mean and CIs for the fitted plot

```
y.fit.mean = apply(y.fit, 2, function(x) mean(x))
y.fit.q05 = apply(y.fit, 2, function(x) quantile(x, probs=0.05))
y.fit.q95 = apply(y.fit, 2, function(x) quantile(x, probs=0.95))
plot(stan.data$dose, stan.data$y/stan.data$total, xlab="Dose", ylab="Infected", cex=0.8, mai
polygon(c(x.pred, rev(x.pred)),
        c(y.fit.q05, rev(y.fit.q95)),
        border = NA,
        col = adjustcolor("blue", alpha.f=0.25))
lines(x.pred, y.fit.mean, col="blue", lwd=2)
```



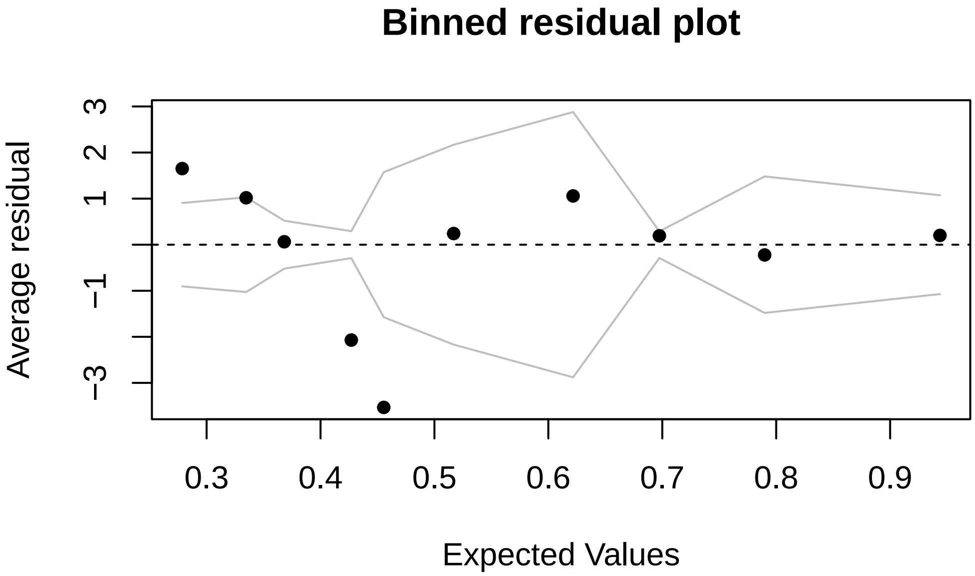
Binned residuals plot with predictions for actual observations

```
x.pred = stan.data$dose
y.fit = matrix(NA, nrow=nrow(post), ncol=length(x.pred) )
for(i in 1:nrow(post)){
  y.fit[i, ] = invlogit(post[i,"a"] + post[i,"b"]*x.pred)
}
str(y.fit)
```

```
 num [1:4000, 1:43] 0.287 0.248 0.268 0.173 0.256 ...
```

```
y.fit.mean = apply(y.fit, 2, function(x) mean(x))
y.fit.response = y.fit.mean*stan.data$total # scale up from proportion to response
y.residuals = y.fit.response-stan.data$y

binnedplot(y.fit.mean, y.residuals)
```

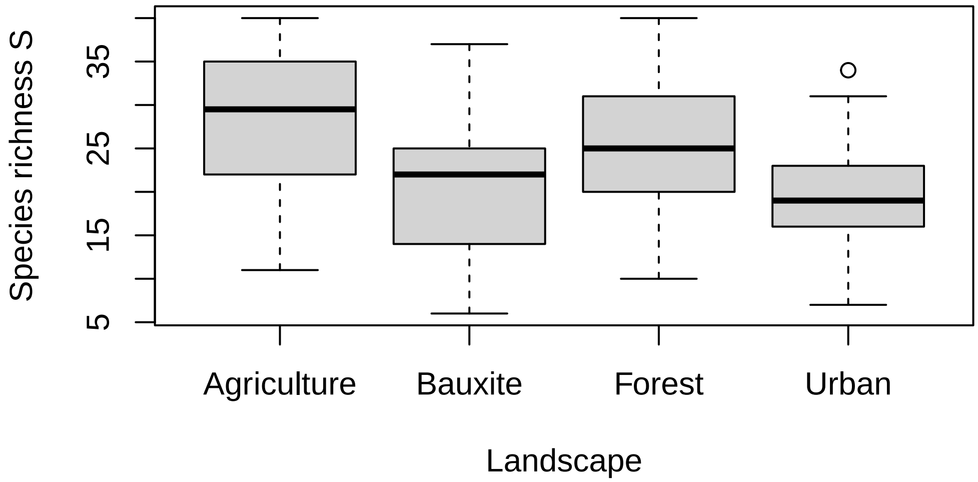


ANOVA

We redo the bird species richness vs landscape type analysis. We use effects-coding, meaning we directly fit the group-specific intercepts (mean species richness in each landscape), instead of dummy-coding.

Deterministic part: $\mu_i = b(landscape_i)$
Stochastic part: $S_i \sim \text{Normal}(\mu_i, \sigma)$
$i = 1, ..., n$

```
data(birds)
data = birds[, c("S", "landscape", "log.area.")] # use area later
data$landscape = as.factor(data$landscape)
data = data[complete.cases(data), ]
plot(data$landscape, data$S, xlab="Landscape", ylab="Species richness S")
```



We convert the factor `landscape` into an integer, so we can use them as an index variable in Stan

```
stan.data = list(N = nrow(data),
                 M = 4,
                 y = data$S,
                 group = as.integer(data$landscape))
str(stan.data)
```

```
List of 4
 $ N    : int 257
 $ M    : num 4
 $ y    : int [1:257] 24 15 25 35 32 40 27 37 40 36 ...
 $ group: int [1:257] 1 1 1 1 1 1 1 1 1 1 ...
```

In the Stan code, we loop through all datapoints and use the grouping variable landscape as an index variable (1,2,3,4) to select the correct intercept for each observation `b[group[i]]`. A vectorized version would just use `b[group]` with the whole vector `group`.

```
cat(readChar("lm_anova.stan", 1e6))
```

```
data {
  int N;        // i=1:N observations
  int M;        // j=1:M levels
  vector[N] y;
  array[N] int group;
}
```

```
parameters {
  vector[M] b;
  real<lower=0> sigma;
}
model {
  for(j in 1:M){
    b[j] ~ normal(25,10);
  }
  // or short: b ~ normal(25,10);
  sigma ~ exponential(0.1);
  for(i in 1:N){
    y[i] ~ normal(b[group[i]], sigma);
  }
  // or short: y ~ normal(b[group], sigma);
}
```

```
fit3 = stan(file="lm_anova.stan", data=stan.data)
```

Check model output and convergence

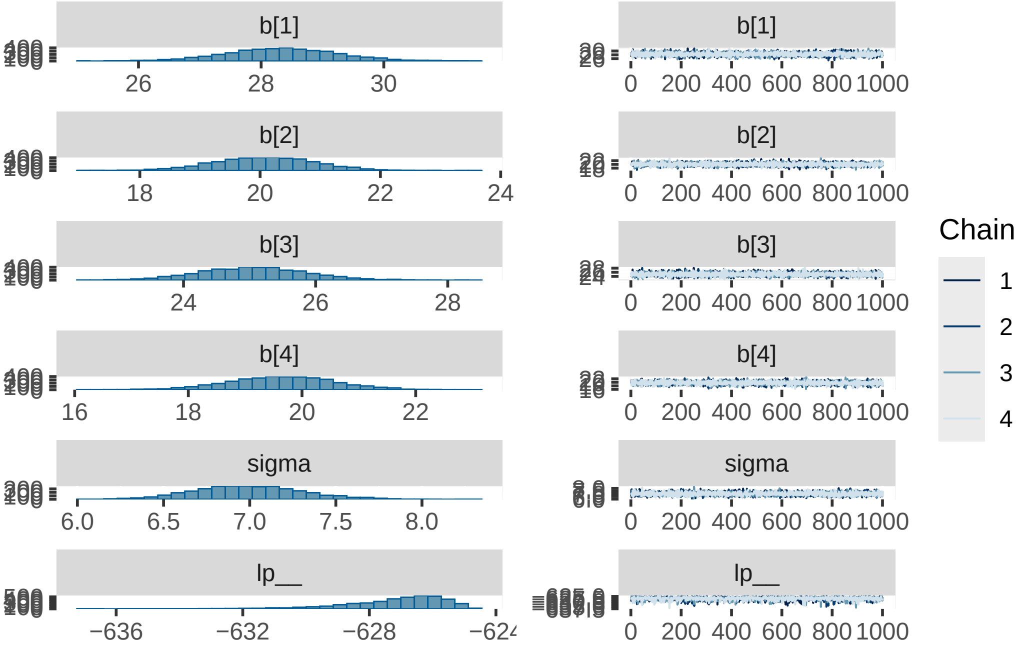
```
print(fit3, probs=c(0.05, 0.95))
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd      5%     95% n_eff Rhat
b[1]    28.35    0.01 0.91   26.85   29.86  4913    1
b[2]    20.08    0.01 0.84   18.69   21.45  5246    1
b[3]    25.06    0.01 0.82   23.74   26.39  5066    1
b[4]    19.65    0.01 0.93   18.12   21.16  5544    1
sigma    7.00    0.00 0.31    6.51    7.53  5217    1
lp__  -627.17    0.03 1.60 -630.22 -625.25  2106    1

Samples were drawn using NUTS(diag_e) at Tue Feb 18 13:42:17 2025.
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).
```

```
mcmc_combo(fit3, combo=c("hist","trace"))
```

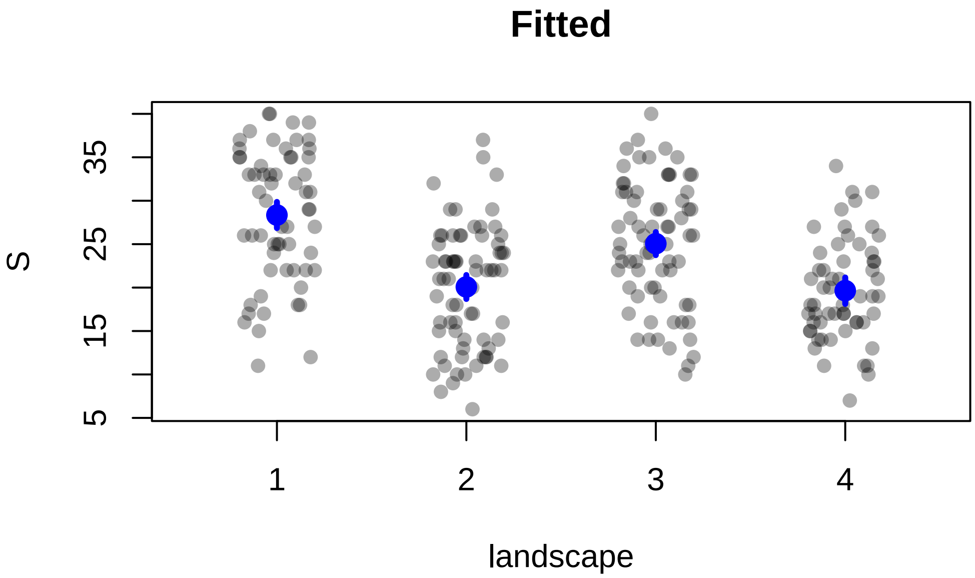


For plotting fitted values versus data, we here can just extract group-level intercepts from the summary table

```
summ.table = summary(fit3, probs=c(0.05, 0.95))$summary
print(summ.table, digits=3)
```

```
         mean se_mean    sd      5%     95% n_eff  Rhat
b[1]     28.3 0.01299 0.910   26.85   29.86  4913 1.000
b[2]     20.1 0.01157 0.838   18.69   21.45  5246 1.000
b[3]     25.1 0.01145 0.815   23.74   26.39  5066 1.000
b[4]     19.6 0.01250 0.931   18.12   21.16  5544 1.000
sigma     7.0 0.00434 0.313    6.51    7.53  5217 0.999
lp__   -627.2 0.03486 1.600 -630.22 -625.25  2106 1.000
```

```
plot(jitter(stan.data$group), stan.data$y,
     xlim=c(0.5,4.5),
     pch=16,
     col=adjustcolor(1, alpha.f=0.33),
     xlab="landscape", ylab="S", main="Fitted")
for(i in 1:4){
  points(i, summ.table[i, "mean"], col="blue", pch=16, cex=1.5) # mean
  lines(c(i, i), summ.table[i, 4:5], col="blue", lwd=3)          # CI
}
```



Mixed effects model

Same data as ANOVA. ANOVA uses no pooling for landscape type. Linear mixed model uses partial pooling for landscape type.

This would be an appropriate model if we wanted to estimate overall mean species richness, but acknowledge non-independence of residuals. Here this would make sense if the actual sites of observations were not geographically mixed, but 1 geographical cluster per landscape type with multiple sites (1 site = 1 obs). The mixed effects model accounts for spatial autocorrelation (with grouping factor landscape).

Deterministic part: $\mu_i = b(landscape_i)$
Stochastic part: $S_i \sim \text{Normal}(\mu_i, \sigma)$
Hierarchical part: $b_j \sim \text{Normal}(\mu_b, \sigma_b)$
$i = 1 \dots n, \quad j = 1 \dots m$

Random effects parameters are the actual group means b_j, not deviation from grand mean μ_b.

In the Stan code, we have to replace the priors for b_j with hierarchical parameters μ_b, σ_b, which have priors themselves.

```
cat(readChar("lmm_intercepts.stan", 1e6))
```

```
data {
  int N;        // i=1:N observations
  int M;       // j=1:M levels
  vector[N] y;
  array[N] int group;
}
parameters {
  real mu_b;
  real<lower=0> sd_b;
  vector[M] b;
  real<lower=0> sigma;
}
model {
  for(j in 1:M){
    b[j] ~ normal(mu_b,sd_b);
  }
  // or short: b ~ normal(mu_b,sd_b);
  mu_b ~ normal(25,10);
  sd_b ~ exponential(0.1);
  sigma ~ exponential(0.1);
  for(i in 1:N){
    y[i] ~ normal(b[group[i]], sigma);
  }
  // or short: y ~ normal(b[group], sigma);
}
```

```
fit4 = stan(file="lmm_intercepts.stan", data=stan.data)
```

Check model output and convergence

```
print(fit4, probs=c(0.05, 0.95))
```

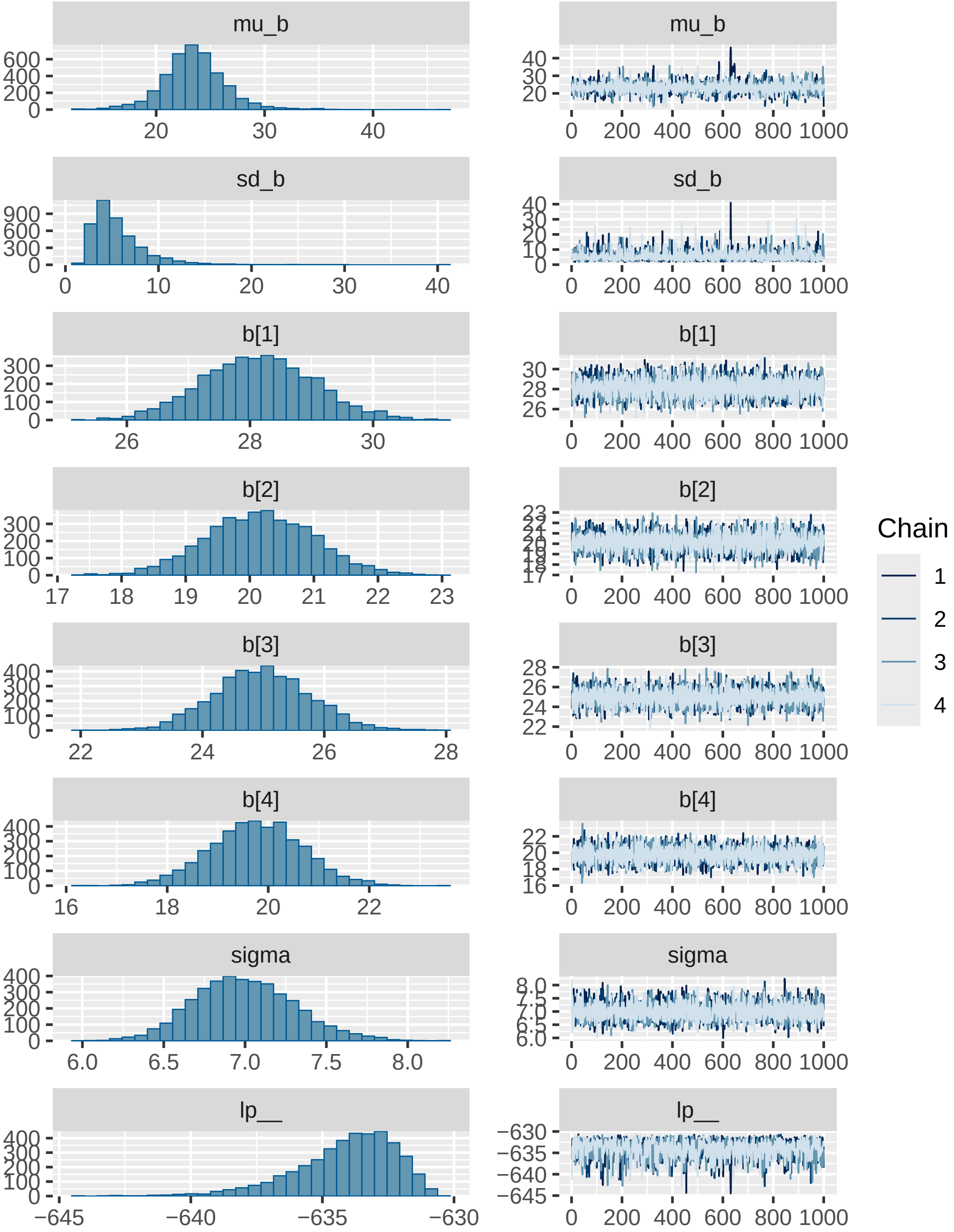
```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd     5%    95% n_eff Rhat
mu_b    23.48    0.07 2.94  19.02  28.14  1957    1
sd_b     5.66    0.06 3.04   2.56  11.22  2532    1
b[1]    28.14    0.01 0.91  26.66  29.64  5370    1
b[2]    20.16    0.01 0.85  18.77  21.54  5853    1
b[3]    24.98    0.01 0.80  23.68  26.29  5491    1
b[4]    19.77    0.01 0.91  18.26  21.26  6101    1
```

```
sigma    7.00    0.00 0.31    6.52    7.54  4753    1
lp__  -634.12    0.05 1.94 -637.79 -631.57  1654    1

Samples were drawn using NUTS(diag_e) at Wed Feb  5 13:49:59 2025.
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).
```

```r
mcmc_combo(fit4, combo=c("hist","trace"))
```

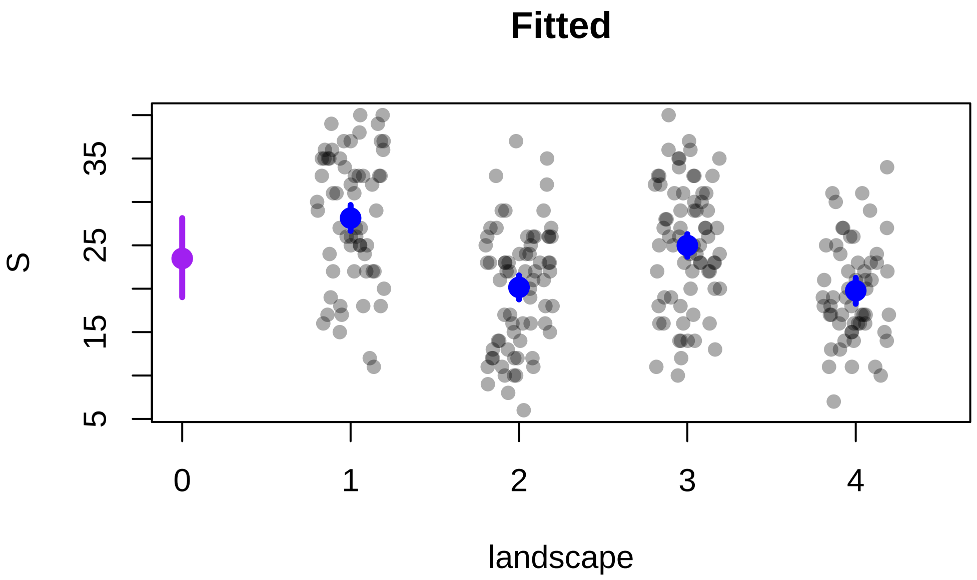


For plotting fitted values with fixed effects (1 population-level intercept) and random effects (4 group-level intercepts) versus data, we here can just extract from the summary table

```
summ.table = summary(fit4, probs=c(0.05, 0.95))$summary
print(summ.table, digits=3)
```

```
          mean se_mean    sd      5%     95% n_eff Rhat
mu_b     23.48 0.06635 2.935   19.02   28.14  1957    1
sd_b      5.66 0.06050 3.044    2.56   11.22  2532    1
b[1]     28.14 0.01237 0.906   26.66   29.64  5370    1
b[2]     20.16 0.01105 0.846   18.77   21.54  5853    1
b[3]     24.98 0.01077 0.798   23.68   26.29  5491    1
b[4]     19.77 0.01160 0.906   18.26   21.26  6101    1
sigma     7.00 0.00452 0.312    6.52    7.54  4753    1
lp__   -634.12 0.04766 1.939 -637.79 -631.57  1654    1
```

```
plot(jitter(stan.data$group), stan.data$y,
     xlim=c(0,4.5),
     pch=16,
     col=adjustcolor(1, alpha.f=0.33),
     xlab="landscape", ylab="S", main="Fitted")
for(i in 1:4){
  points(i, summ.table[i+2, "mean"], col="blue", pch=16, cex=1.5)
  lines(c(i, i), summ.table[i+2, 4:5], col="blue", lwd=3)
}
points(0, summ.table[1, "mean"], col="purple", pch=16, cex=1.5)
lines(c(0,0), summ.table[1, 4:5], col="purple", lwd=3)
```



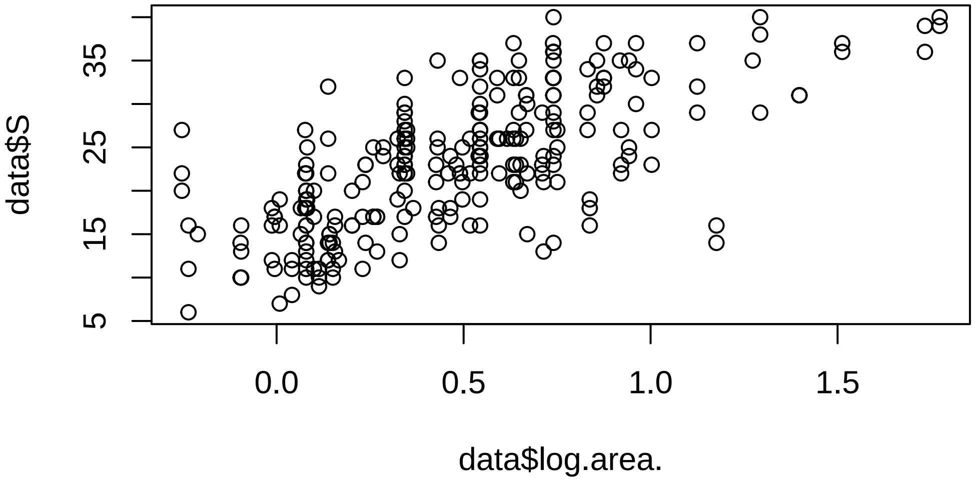
ANCOVA & model comparison

Exercise: re-analyze the bird species richness data, but this time add a continuous predictor area (ANCOVA), `S ~ landscape + log.area`

Bonus: Do a model comparison with LOO for the ANOVA and the ANCOVA model. You must save the pointwise log-likelihood values in the generated quantities block, which later can be extracted by the loo-package.

https://mc-stan.org/loo/articles/loo2-with-rstan.html

```
data(birds)
data = birds[, c("S", "landscape", "log.area.")]
data$landscape = as.factor(data$landscape)
data = data[complete.cases(data), ]
plot(data$log.area., data$S)
```



We add the predictor *log.area* to the ANOVA model, meaning we have 4 different intercepts (landscape type), but 1 joint slope (area)

Deterministic part: $\mu_i = b(landscape_i) + c \cdot logarea$
Stochastic part: $S_i \sim \text{Normal}(\mu_i, \sigma)$
$i = 1, ..., n$

```
stan.data = list(N = nrow(data),
                 M = 4,
                 y = data$S,
                 group = as.integer(data$landscape),
                 x = as.vector(scale(data$log.area.)
                  )
)
str(stan.data)
```

```
List of 5
 $ N    : int 257
 $ M    : num 4
 $ y    : int [1:257] 24 15 25 35 32 40 27 37 40 36 ...
 $ group: int [1:257] 1 1 1 1 1 1 1 1 1 1 ...
 $ x    : num [1:257] 0.153 -1.782 -0.349 1.107 -0.89 ...
```

In Stan, we simply include a slope `c` and add the area effect to the prediction.

For model comparison with LOO, we need the log-likelihood value of each datapoint. We compute them in the `generated quantities` block. The function `normal_lpdf( observation | prediction, sigma)` computes log values ("l" in "lpdf" is for log) of a normal distribution likelihood.

```
cat(readChar("lm_ancova_loo.stan", 1e6))
```

```
data {
  int N;       // i=1:N observations
  int M;       // j=1:M levels
  vector[N] y;
  vector[N] x; // continuous predictor
  array[N] int group; // categorical predictor
}
parameters {
  vector[M] b; // M intercepts
  real c;      // joint slope
  real<lower=0> sigma;
}
model {
  for(j in 1:M){
    b[j] ~ normal(25,10);
  }
  // or short: b ~ normal(25,10);
  c ~ normal(5, 10);
  sigma ~ exponential(0.1);
  for(i in 1:N){
    y[i] ~ normal(b[group[i]]+c*x[i], sigma);
  }
  // or short: y ~ normal(b[group]+c*x, sigma);
}
generated quantities{
  vector[N] log_lik;
  for(i in 1:N){
    log_lik[i] = normal_lpdf(y[i] | b[group[i]]+c*x[i], sigma);
  }
}
```

```
fit5 = stan(file="lm_ancova_loo.stan", data=stan.data)
```

Check model output and convergence. Specify parameters, otherwise you will get an output for all 257 pointwise log-likelihoods.

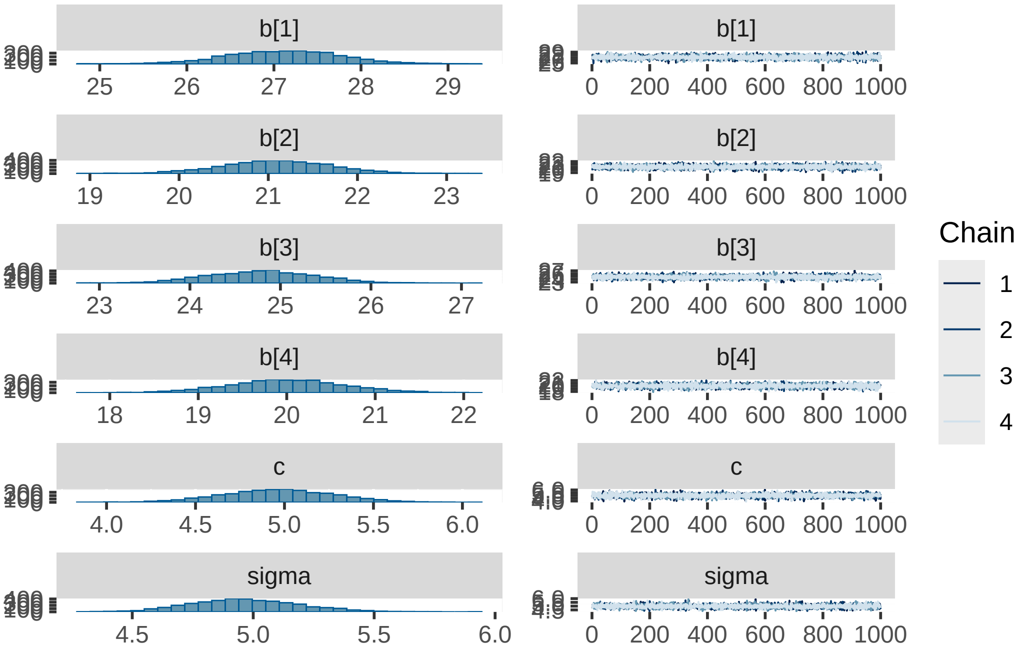
```
print(fit5, probs=c(0.05, 0.95), pars=c("b","c","sigma"))
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

       mean se_mean   sd    5%   95% n_eff Rhat
b[1]  27.12    0.01 0.66 26.03 28.18  5239    1
b[2]  21.10    0.01 0.61 20.08 22.10  5800    1
b[3]  24.79    0.01 0.59 23.84 25.75  5175    1
b[4]  20.00    0.01 0.65 18.92 21.09  5263    1
c      4.97    0.00 0.32  4.45  5.49  4795    1
sigma  4.97    0.00 0.23  4.62  5.35  4785    1

Samples were drawn using NUTS(diag_e) at Wed Feb  5 11:35:20 2025.
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).
```

```
mcmc_combo(fit5, combo=c("hist","trace"),
           pars=c("b[1]","b[2]","b[3]","b[4]","c","sigma"))
```



For plotting fitted values, we use the same steps as before, but wrap it in a for-loop over all levels of landscape type. `post[i,k]` selects the correct intercept `b[k]`.

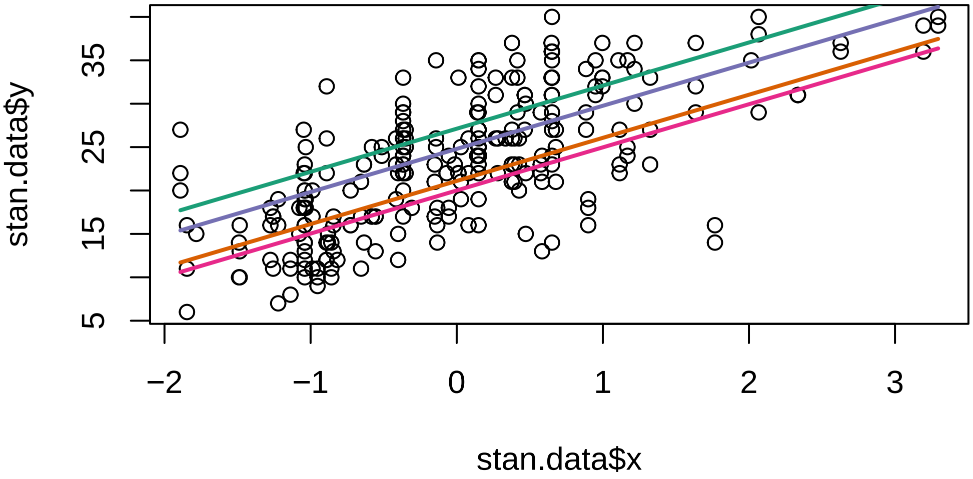
```r
post = as.matrix(fit5)
head(post[, 1:5])
```

```
          parameters
iterations     b[1]     b[2]     b[3]     b[4]        c
      [1,] 26.24975 21.94404 24.71915 19.95800 4.757185
      [2,] 28.06879 20.24766 24.78926 20.23378 5.304887
      [3,] 27.17458 20.37799 24.57714 19.74945 5.322666
      [4,] 27.37366 21.22623 25.27899 20.17855 4.848027
      [5,] 26.56795 20.79091 25.78383 19.65466 4.602620
      [6,] 27.58373 20.78474 23.61422 19.72103 5.346899
```

```r
xmin = min(stan.data$x)
xmax = max(stan.data$x)
x.pred = seq(xmin, xmax, length.out=100)

colors = brewer.pal(4, "Dark2")

plot(stan.data$x, stan.data$y)
for(k in 1:4){
  y.fit = matrix(NA, nrow=nrow(post), ncol=length(x.pred) )
  for(i in 1:nrow(post)){
    y.fit[i, ] = post[i,k] + post[i,"c"]*x.pred
  }
  # extract mean and CI
  y.fit.mean = apply(y.fit, 2, function(x) mean(x))
  y.fit.q05 = apply(y.fit, 2, function(x) quantile(x, probs=0.05))
  y.fit.q95 = apply(y.fit, 2, function(x) quantile(x, probs=0.95))
  # polygon(c(x.pred, rev(x.pred)),
  #         c(y.fit.q05, rev(y.fit.q95)),
  #         border = NA,
  #         col = adjustcolor(colors[k], alpha.f=0.25))
  lines(x.pred, y.fit.mean, col=colors[k], lwd=2)
}
```



Finally, we recompute the ANOVA, saving the log-likelihood values in the generated quantities block.

```
cat(readChar("lm_anova_loo.stan", 1e6))
```

```
data {
  int N;        // i=1:N observations
  int M;        // j=1:M levels
  vector[N] y;
  array[N] int group;
}
parameters {
  vector[M] b;
  real<lower=0> sigma;
}
model {
  for(j in 1:M){
    b[j] ~ normal(25,10);
  }
  // or short: b ~ normal(25,10);
  sigma ~ exponential(0.1);
  for(i in 1:N){
    y[i] ~ normal(b[group[i]], sigma);
  }
  // or short: y ~ normal(b[group], sigma);
}
generated quantities{
  vector[N] log_lik;
  for(i in 1:N){
    log_lik[i] = normal_lpdf(y[i] | b[group[i]], sigma);
  }
}
```

```
fit6 = stan(file="lm_anova_loo.stan", data=stan.data)
```

```
lik.anova = extract_log_lik(fit6)
lik.ancova = extract_log_lik(fit5)

loo.anova = loo(lik.anova)
loo.ancova = loo(lik.ancova)

loo_compare(loo.anova, loo.ancova)
```

```
       elpd_diff se_diff
model2   0.0       0.0   
model1 -86.9      12.6   
```

The model including the additional continuous predictor *log.area* is cleary preferred.