

Introduction to Bayesian Statistics

Part 7 Introduction to Stan

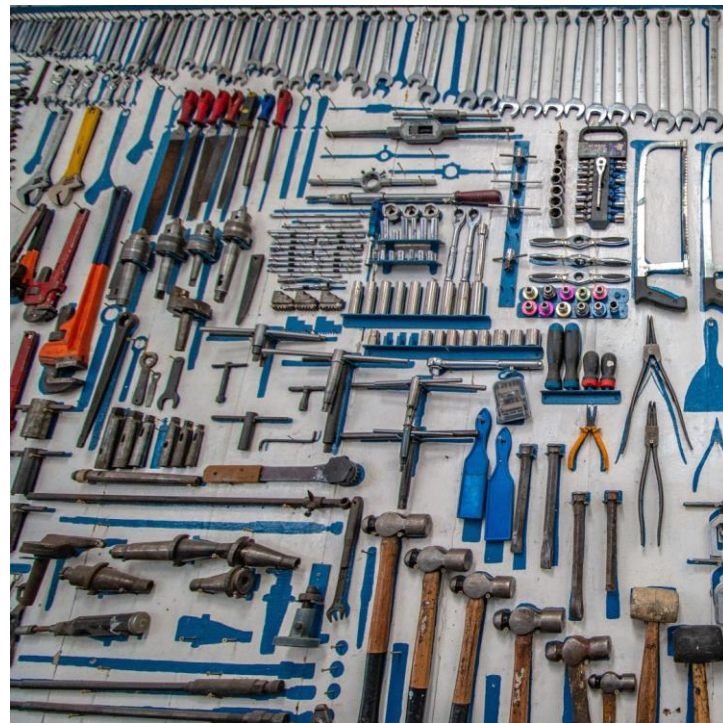
Benjamin Rosenbaum

iDiv 2025




Motivation

- Why using Stan?
- brms actually not a 3d printer:
Very (!) big toolbox
But limited to implemented model classes
- If we want to fit custom statistical models,
we need to code them ourselves



Some history



1700s Bayes' theorem, Laplace formalized it

Bayes impractical
Restricted to simple cases

Early 1800s Gauß: least squares, regression

Late 1800s to early 1900s

Birth of modern statistics. Pearson, Fisher, Neyman ... :
max. likelihood, hypothesis testing, design of experiments

Frequentism superseded Bayes
More practical in most cases

Mid to late 1900s MCMC algorithms

Still a niche topic in statistics

2000s Computational tools for MCMC
BUGS, JAGS, Stan ...

Becoming more popular in sciences

Today Convenient R interfaces
brms, rstanarm ...

Taught in gradschools

Future

Becoming the default
instead of frequentism ??

Who is Stan?

Named after **Stanislaw Ulam (1909-1984)**

- Mathematician, nuclear physicist, computer scientist
- Pioneer of Monte Carlo methods
- But also participant in the Manhattan Project

Biographical movie:

„Adventures of a Mathematician“ 2020

(mixed reviews, watch on own discretion)



By Los Alamos National Laboratory
<https://commons.wikimedia.org/w/index.php?curid=26069369>

What is Stan?

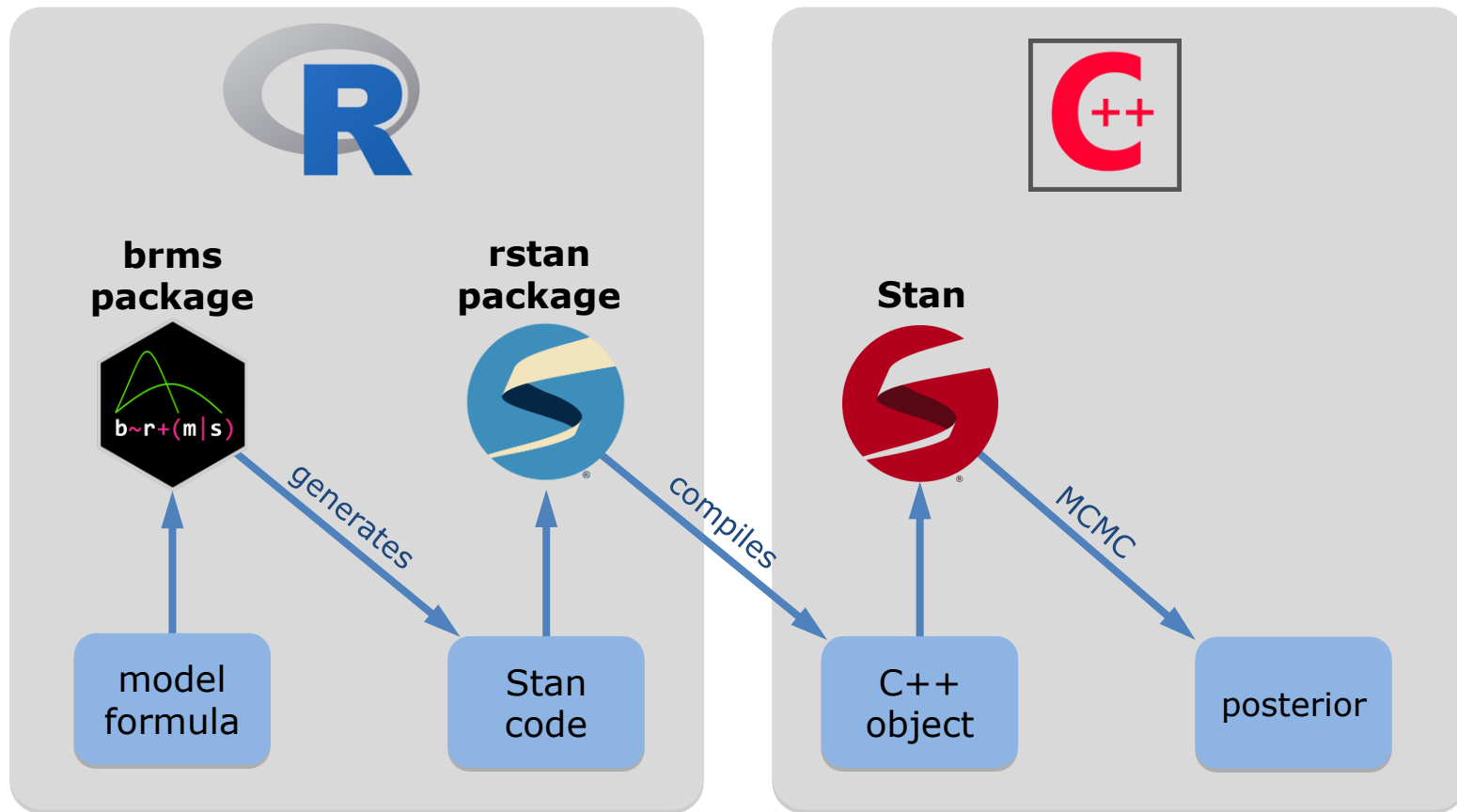
- Started as research project at Columbia University 2011 (Andrew Gelman)
- Written in C++ (fast)
- No-U-turn sampler (NUTS), a version of HMC
- Hamiltonian Monte Carlo (HMC) requires derivatives of posterior
- Uses Automatic Differentiation



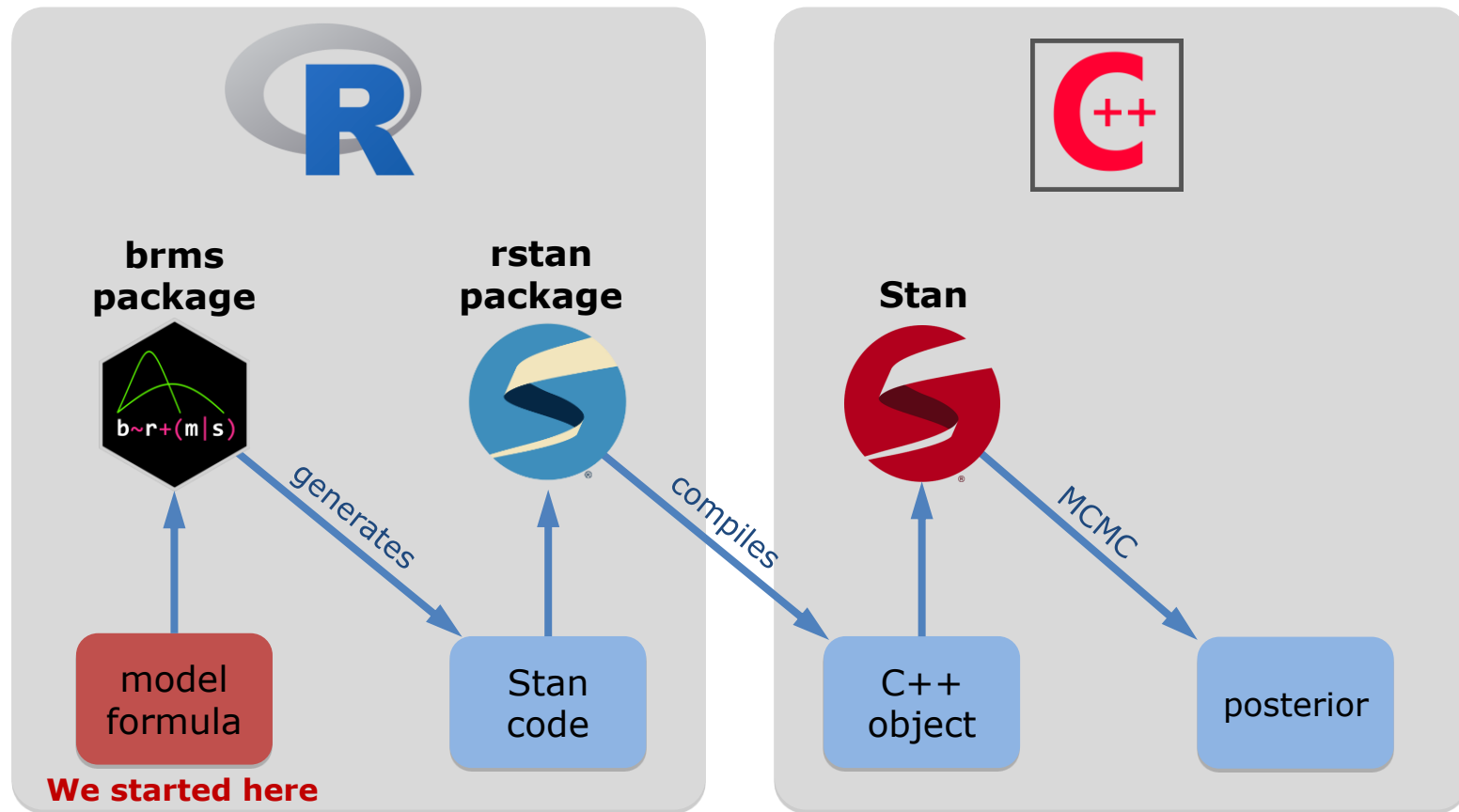
→ Adopted in many fields: science, research, medicine, industry, marketing, ...

OS	Linux		macOS		Windows
	CmdStanPy	CmdStanR	CmdStan	RStan	Stan.jl
	CRAN	R-Universe	conda	GitHub (Source)	

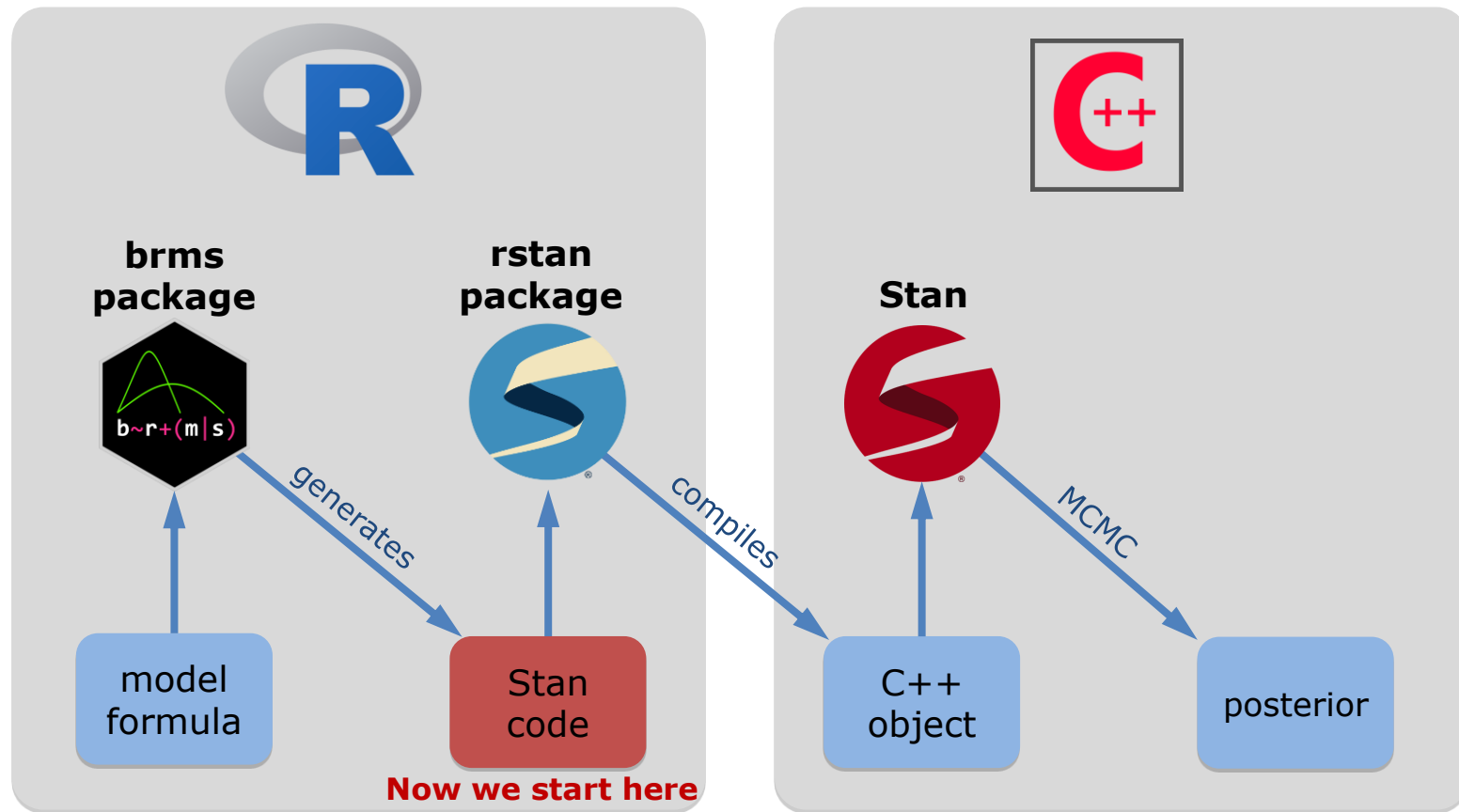
Workflow



Workflow



Workflow



What is Stan code?

Text-object in your R-script

```
stancode = "  
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);  
}  
"
```

Standalone .stan file (supported by RStudio)

```
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);  
}
```

What is Stan code?

Statistical model

Data	x	Predictor
	y	Response
Priors	$a \sim \text{Normal}(0,1)$	
	$b \sim \text{Normal}(0,1)$	
	$\sigma \sim \text{Exponential}(1)$	
Det. part	$\mu_i = a + b \cdot x_i$	
Stoch. part	$y_i \sim \text{Normal}(\mu_i, \sigma)$	

Standalone .stan file (supported by RStudio)

```
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);  
}
```

What is Stan code?

Always 3 blocks:

data{}, **parameters{}**, **model{}**

Optional: functions{}, generated quantities{},
transformed data{}, transformed parameters{}

Coding similar to R, but not always:
some structures different (vectors, arrays)

Each variable (data, parameter, etc)
must be declared with datatype and size

Each operation ends with a semicolon ;

Use //... to comment, not #...

```
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);  
}
```

Data block

Include number of observations N

N determines size of vectors for predictor and response variables

vector is always of datatype real

Count or integer responses must be declared as integers

→ `array[N] int y;`

(Otherwise discrete distributions don't work)

```
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);  
}
```

Parameters block

Declare all datatypes of model parameters

These parameters are sampled by MCMC

`<lower=...>` and `<upper=...>` set hard boundaries

Should only use them if model pars. are logically constrained, e.g. positive sdev

```
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);  
}
```

Model block

- (1) **Prior** statements (\sim) for model parameters
- (2) Arithmetic operations to compute predictions
- (3) **Likelihood** statement for response y_i ($i = 1 \dots N$)

```
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);  
}
```

Model block

- (1) Prior statements (\sim) for model parameters
- (2) Arithmetic operations to compute predictions
- (3) Likelihood statement for response y_i ($i = 1 \dots N$)

Many operations are **vectorized**, but a safe choice is to use a **for-loop** over all N observations

```
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);  
  for(i in 1:N){  
    y[i] ~ normal(a+b*x[i], sigma);  
  }  
}
```


Model block

- (1) Prior statements (\sim) for model parameters
- (2) Arithmetic operations to compute predictions
- (3) Likelihood statement for response y_i ($i = 1 \dots N$)

Many operations are vectorized, but a safe choice is to use a for-loop over all N observations

Can use intermediate steps and **auxiliary variables**. These variables must be declared, but are not sampled

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

What is Stan code?

Recipe for computing prior $p(\theta)$ and likelihood $p(y|\theta)$
and thus **posterior** $p(\theta)p(y|\theta)$ of a sample $\theta = (a, b, \sigma)$

Stan does not know the model structure or
what kind of model we are fitting (LM / GLM / LMM / NLM)

Stan code:

Data & parameter go in, posterior goes out

Stan sampler (NUTS) uses this
to generate samples from posterior distribution

MCMC machinery very sophisticated & efficient:
Uses automatic differentiation to compute curvature
of posterior and to make good proposals for new samples

```
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);  
}
```

rstan, the R interface to Stan

Example

Latitudinal gradient of plant size

Global database with:

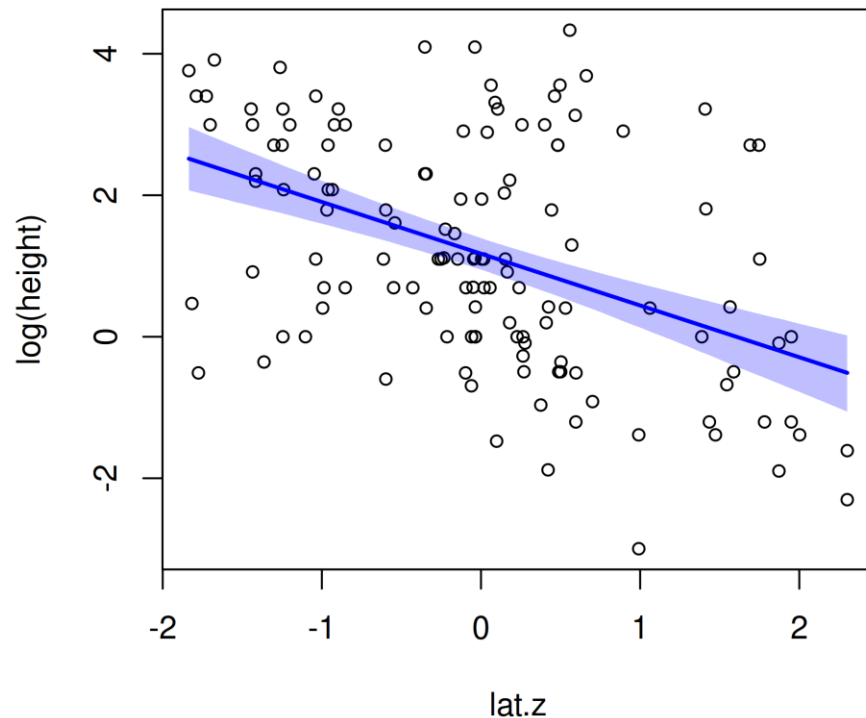
- log of plant height as response
- latitude as predictor (scaled)

Stochastic part:

$$\log(\text{height}) \sim \text{Normal}(\mu, \sigma)$$

Deterministic part:

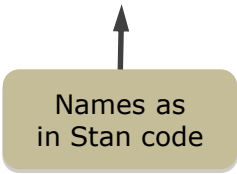
$$\mu = a + b \cdot \text{lat}$$



How to start the engine

(1) Prepare data as named list

```
> stan.data = list(N = nrow(data),  
                  x = scale(data$lat),  
                  y = log(data$height) )
```



Names as
in Stan code

```
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);  
}
```

How to start the engine

(2) Compile model & run MCMC

```
> fit = stan(file = „mymodel.stan“,  
             data = stan.data)
```

```
> fit = stan(model_code = stan.code,  
             data = stan.data)
```

Additional arguments:

```
chains = ...  
iter = ...  
warmup = ...  
cores = ...
```

```
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);  
}
```

How to analyze results

```
> print(fit1, probs=c(0.025, 0.975))  
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	2.5%	97.5%	n_eff	Rhat
a	1.17	0.00	0.14	0.91	1.43	3892	1
b	-0.73	0.00	0.13	-0.99	-0.49	3688	1
sigma	1.49	0.00	0.09	1.32	1.69	3770	1
lp__	-119.77	0.03	1.25	-122.99	-118.31	1862	1

Samples were drawn using NUTS(diag_e) at Mon Feb 3 14:04:04 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).

How to analyze results

From **bayesplot** package

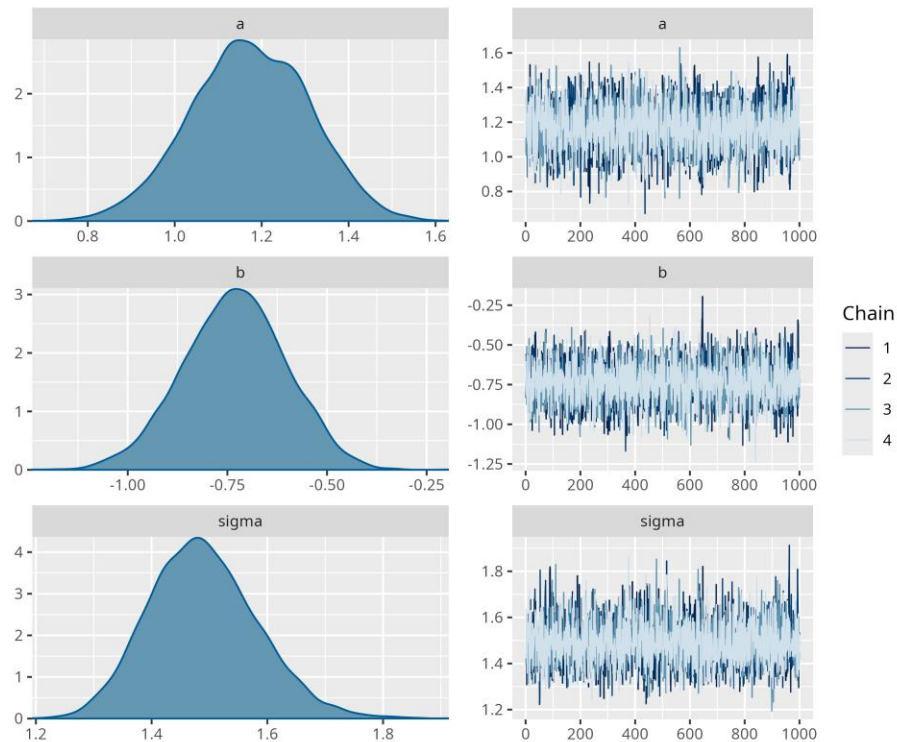
```
> mcmc_combo(fit1)
```

`mcmc_hist`

`mcmc_trace`

`mcmc_dens`

...



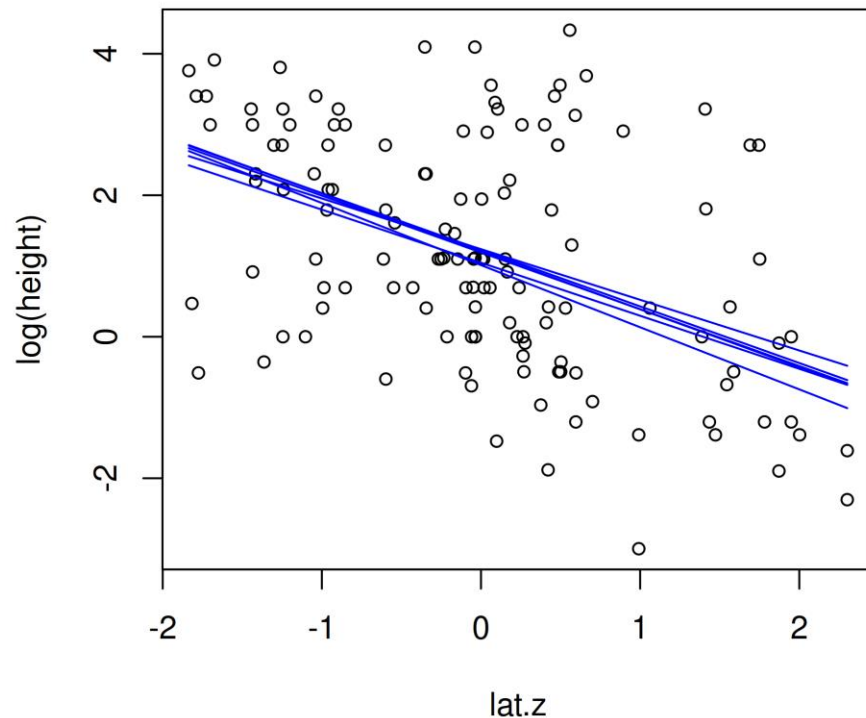
<https://mc-stan.org/bayesplot/reference/index.html#mcmc>

Posterior predictions (as in conditional_effects)

Each posterior sample generates a regression line

```
> post = as.matrix(fit1)
```

	parameters		
iterations	a	b	sigma
[1,]	1.191243	-0.8028239	1.418879
[2,]	1.229759	-0.8005215	1.428119
[3,]	1.206685	-0.8186280	1.498435
[4,]	1.238122	-0.7161273	1.307835
[5,]	1.012098	-0.8779856	1.486404
[6,]	1.046561	-0.7498799	1.444269



Posterior predictions (as in conditional_effects)

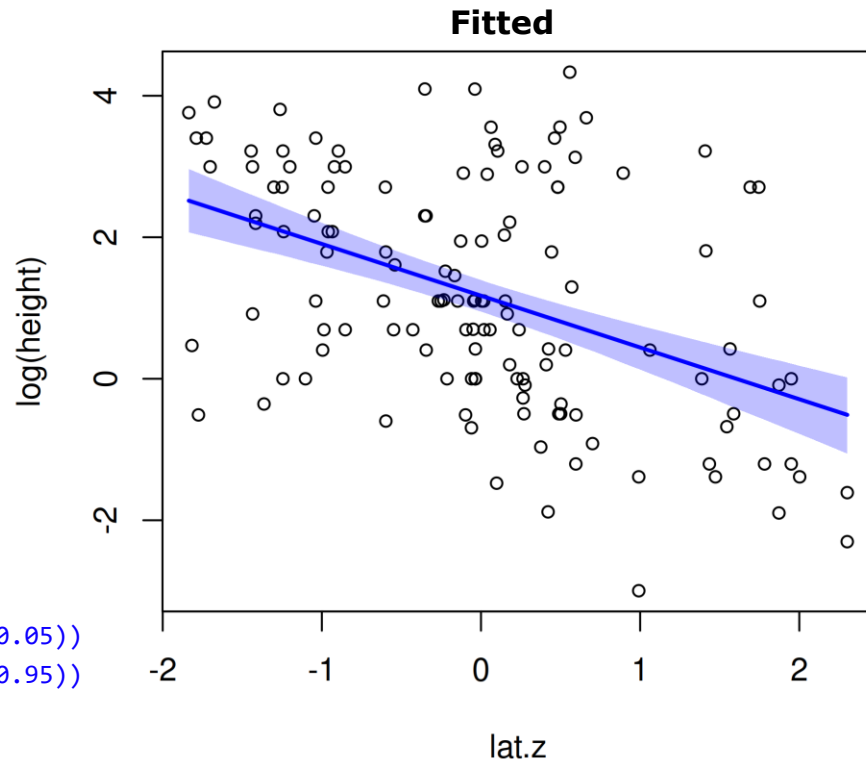
Compute predictions for deterministic model part

```
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, ] = post[i,"a"] + post[i,"b"]*x.pred
}
```

Extract mean fitted curve and credible intervals

```
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))
```



Posterior predictions (as in conditional_effects)

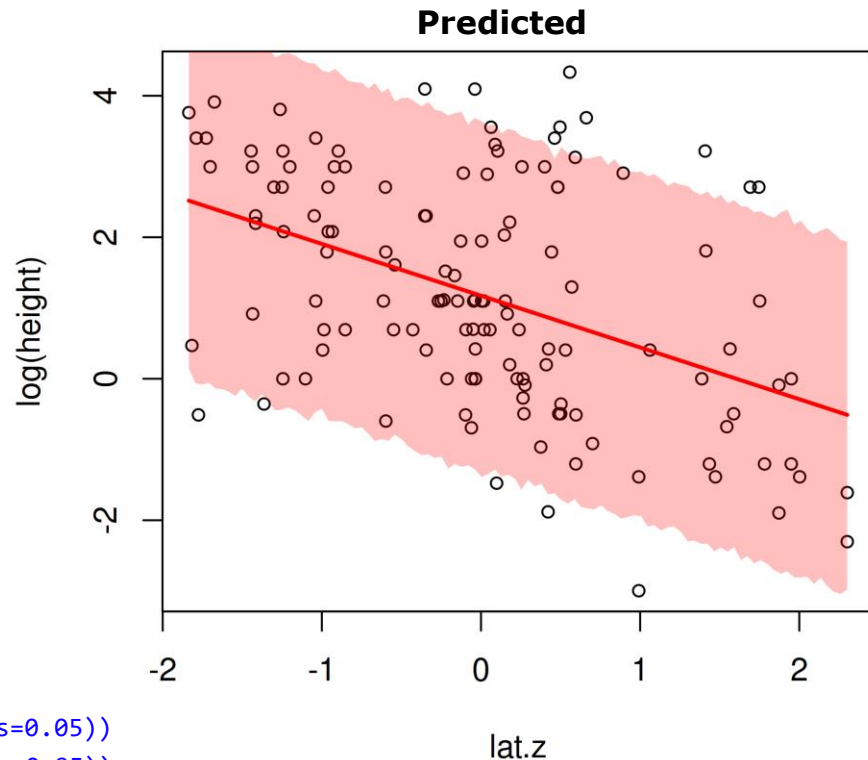
Compute predictions for stochastic model part

```
x.pred = seq(xmin, xmax, length.out=100)
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"] )
}
```

Extract mean predictions and prediction intervals

```
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))
```



Posterior predictions (residuals)

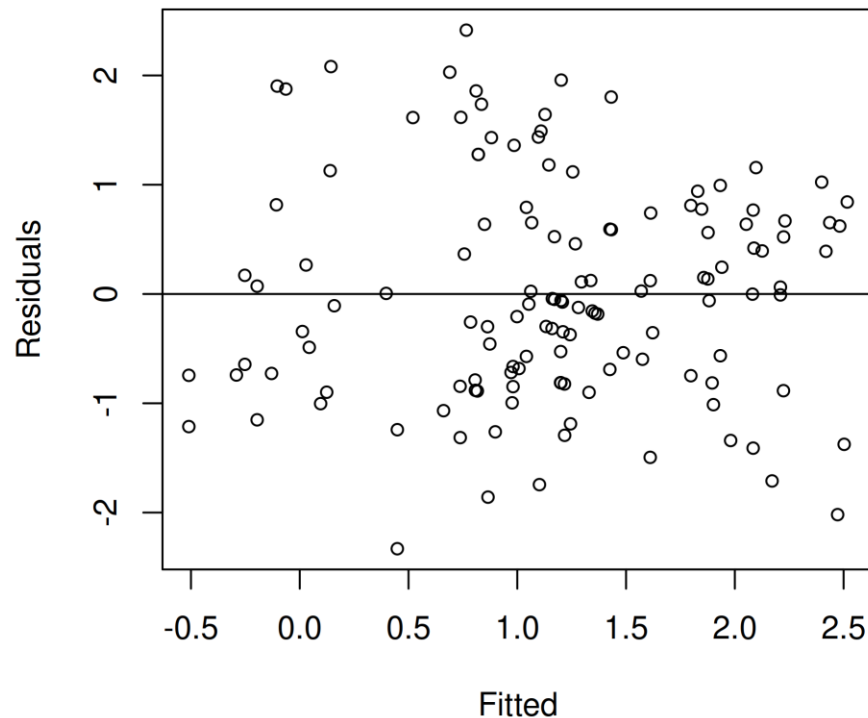
Compute predictions for deterministic model part

```
x.pred = stan.data$x
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
}
```

Extract mean fitted & compute mean **residuals**

```
y.fit.mean= apply(y.fit, 2, function(x) mean(x))
residuals = stan.data$y - y.fit.mean
```



Posterior predictions (pp_checks)

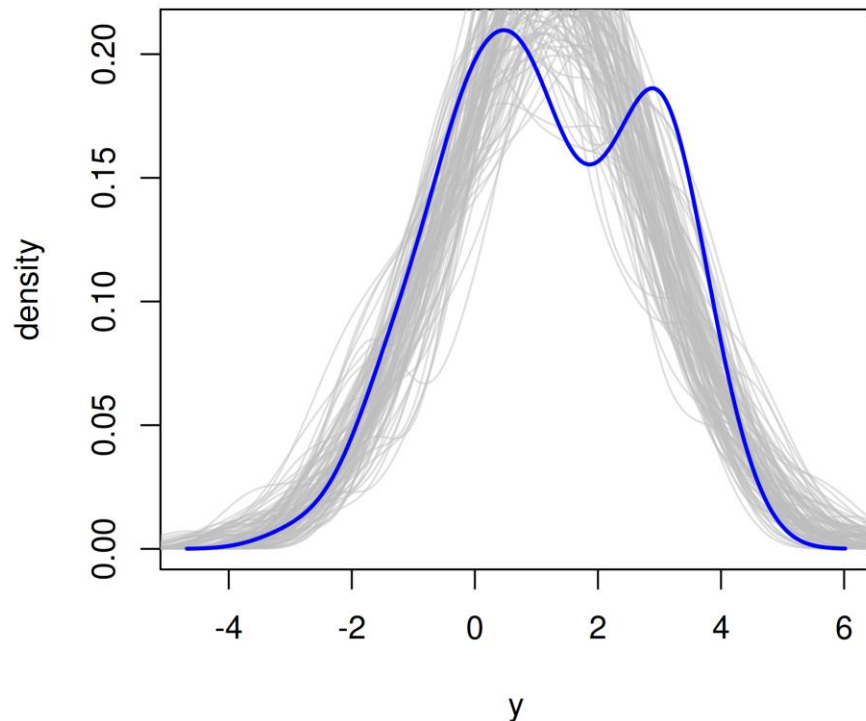
Compute predictions for stochastic model part

```
x.pred = stan.data$x
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"] )
}
```

Posterior predictive check:

Plot histogram of observed response y
vs. some histograms of predicted data y_{pred}



Model comparison with LOO

(1) in Stan code:

Need to save log-likelihood values

of every datapoint $i = 1 \dots N$

$$p(y_i | \theta) = p(y_i | \mu, \sigma) = p(y_i | a + bx_i, \sigma)$$

```
generated quantities {  
  vector[N] log_lik;  
  for (i in 1:N){  
    log_lik[i] = normal_lpdf(y[i] |  
                           a+b*x[i], sigma);  
  }  
}
```

(2) in R:

Extract log-likelihood from fitted model & compute LOO

```
> log_lik_1 = extract_log_lik(fit1)
```

```
> loo(log_lik_1)
```

Computed from 4000 by 131 log-likelihood matrix.

	Estimate	SE
elpd_loo	-239.4	7.1
p_loo	2.7	0.4
looic	478.8	14.2

More examples

Generalized linear model

Example: Occurrence (yes/no) of a butterfly species
versus temperature

Occurrence coded as 1/0 integers

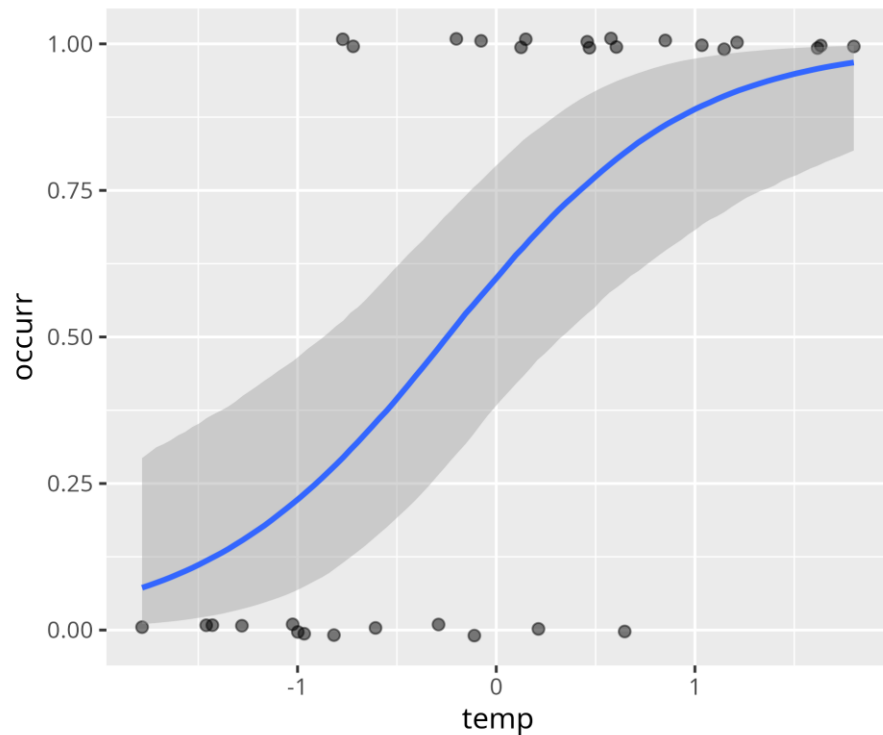
Deterministic part
(linear model & link)

$$\text{logit}(\mu) = a + b \cdot \text{temp}$$

(implicit formulation)

Stochastic part

$$\text{occurrence} \sim \text{Bernoulli}(\mu)$$



Generalized linear model

Example: Occurrence (yes/no) of a butterfly species
versus temperature

Occurrence coded as 1/0 integers

Deterministic part
(linear model & link)

$\mu = \text{inv_logit}(a + b \cdot \text{temp})$
(explicit formulation)

Stochastic part

$\text{occurr} \sim \text{Bernoulli}(\mu)$

```
data {  
  int N;  
  vector[N] temp;  
  array[N] int occurr;  
}  
parameters {  
  real a;  
  real b;  
}  
model {  
  real mu;  
  a ~ normal(0,1);  
  b ~ normal(0,1);  
  for(i in 1:N){  
    mu = inv_logit(a+b*temp[i]);  
    occurr[i] ~ bernoulli(mu);  
  }  
}
```

Categorical predictor

Example: bird species richness vs landscape type

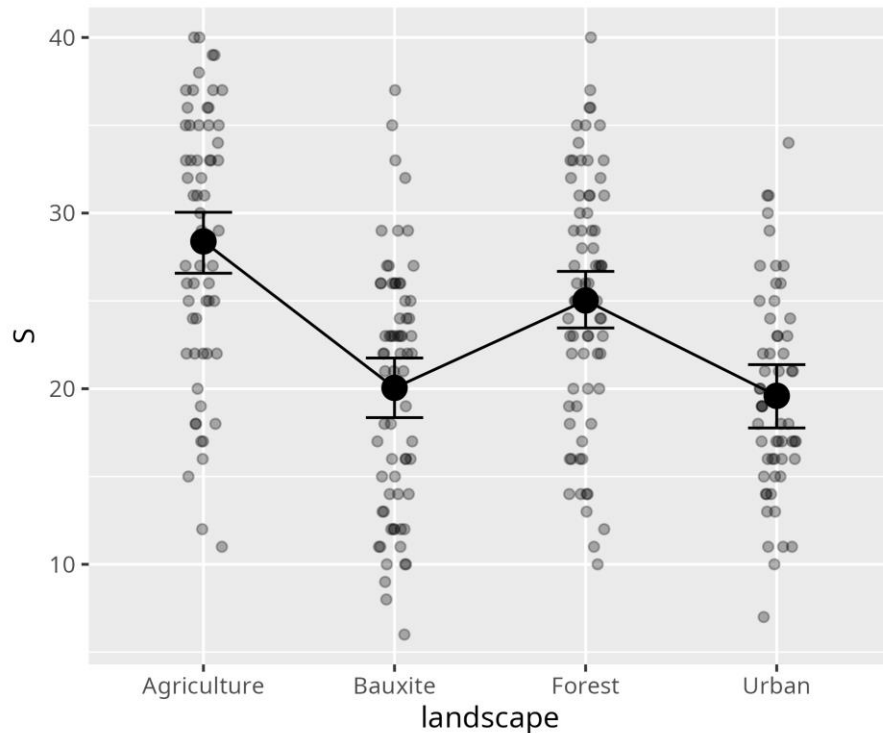
Deterministic part: $\mu = b(\text{landscape})$

Stochastic part: $S \sim \text{Normal}(\mu, \sigma)$

4 means b_{Agri} , b_{Bauxite} , b_{Forest} , b_{Urban}

Problem: Stan does not allow factors variables

→ Code factor *landscape* as integer
(levels 1,2,3,4)



ANOVA

Example: bird species richness vs landscape type

Deterministic part: $\mu_i = b_{\text{landscape}_i}$

Stochastic part: $S_i \sim \text{Normal}(\mu_i, \sigma)$

4 means b_1, b_2, b_3, b_4

Integer predictor *landscape* (levels 1,2,3,4)

used as **index** in Stan

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

Random effects model

Example: bird species richness vs landscape type

Deterministic part: $\mu_i = b_{\text{landscape}_i}$

Hierarchical part: $b_j \sim \text{Normal}(\mu_b, \sigma_b)$

Stochastic part: $S_i \sim \text{Normal}(\mu_i, \sigma)$

Replaced priors for $b_j \rightarrow \mu_b, \sigma_b$ model parameters

Alternative: $\mu_i = \mu_b + \delta_{\text{landscape}_i}$

$\delta_j \sim \text{Normal}(0, \sigma_b)$

```
data {  
  int N;          // i=1:N observations  
  int M;          // j=1:M levels  
  vector[N] S;  
  array[N] int landscape;  
}  
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);  
  }  
  mu_b ~ normal(25,10);  
  sd_b ~ exponential(0.1);  
  sigma ~ exponential(0.1);  
  for(i in 1:N){  
    S[i] ~ normal(b[landscape[i]], sigma);  
  }  
}
```

Summary

What else?

Cool Stan stuff

- brms already pretty versatile ...
... but with Stan, theoretically no limit to model complexity
- Continuous latent variables (state-space models)
- Fit process-based models (population / community dynamics)
- Even differential equations (continuous dynamics)

But also **limitations**

- Deterministic models only
- Could have issues with non-smooth models
- No discrete parameters (workaround via marginalization possible)
- Gets slow with spatial autocorrelation

NIMBLE

An alternative to Stan

- Replaces BUGS / JAGS nowadays
- Originates from the Ecology community (Perry de Valpine)
- Code is more slender, basically just a model block
- Modeling paradigm a bit different:
parameters, data, variables are nodes: “probabilistic graphical models”
- Discrete parameters allowed
Makes possible: HMMs, occupancy models, etc (discrete latent states)
- Also possible: non-exact / random simulation models (Sequential Monte Carlo)



Summary

- Translate almost any statistical model into Stan code
- Not limited to model classes (LM, GLM, GLMM, GAM, etc)
- Fit Stan model from R
- Unfortunately, can't use fancy brms tools for posterior predictions
- Compute predictions manually in R from posterior distribution
- Huge reference manual: <https://mc-stan.org/docs>
- Active community: <https://discourse.mc-stan.org/>

Further reading

Johnson, A. A., Ott, M. Q., Dogucu, M. (2021). Bayes Rules! *CRC Press*. <https://www.bayesrulesbook.com/>

Korner-Nievergelt, F., Roth, T., Von Felten, S., Guélat, J., Almasi, B. and Korner-Nievergelt, P. (2024). Bayesian Data Analysis in Ecology Using Linear Models with R and Stan. <https://tobiasroth.github.io/BDAEcology/>

Lambert, B. (2018). A Student's Guide to Bayesian Statistics. *Sage*. <https://ben-lambert.com/bayesian/>

Stan Development Team (2025). Stan Documentation. <https://mc-stan.org/docs/>