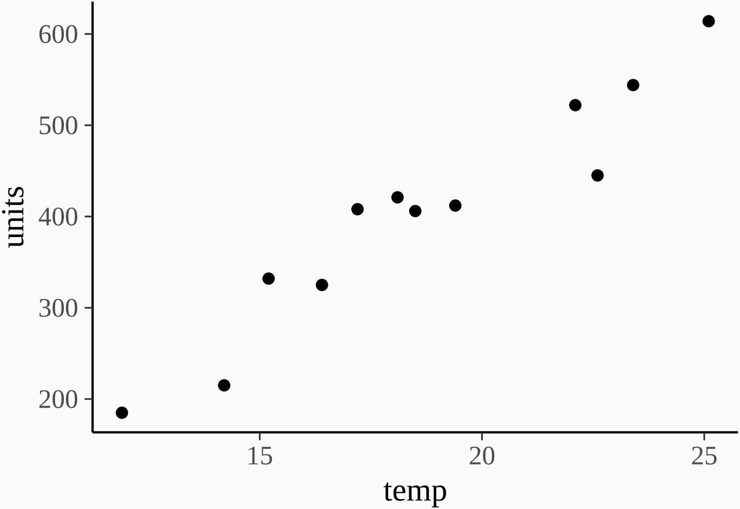
Bayesian Data Analysis Using Stan

Paul Bürkner

Workshop Material

https://github.com/paul-buerkner/2019_DAGStat_Stan_Tutorial

Example: Icecream Sold at Different Temperatures



Linear Models with Stan

Simple Linear Regression

We assume the following generative model (*likelihood*)

$$y_n = \alpha + \beta x_n + \varepsilon_n$$

$$\varepsilon_n \sim \text{Normal}(0, \sigma)$$

or equivalently

$$y_n \sim \text{Normal}(\alpha + \beta x_n, \sigma)$$

Let's vectorize the model

$$y \sim \text{Normal}(\alpha + \beta x, \sigma)$$

Stan Syntax: Simple Linear Regression

```
data {
  int<lower=1> N;  // total number of observations
  vector[N] y;  // response variable
  vector[N] x;  // predictor variable
}
parameters {
  real alpha;  // intercept
  real beta;  // slope
  real<lower=0> sigma;  // residual SD
}
model {
  // likelihood
  for (n in 1:N) {
    y[n] ~ normal(alpha + beta * x[n], sigma);
  }
}
```

Stan Syntax: Simple Linear Regression (Vectorized)

```
data {
  int<lower=1> N;  // total number of observations
  vector[N] y;  // response variable
  vector[N] x;  // predictor variable
}
parameters {
  real alpha;  // intercept
  real beta;  // slope
  real<lower=0> sigma;  // residual SD
}
model {
  // likelihood
  y ~ normal(alpha + beta * x, sigma);
}
```

The Posterior Distribution

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \propto p(y|\theta)p(\theta) = p(y, \theta)$$

What's the matter with all the p functions?

- Likelihood: $p(y|\theta)$
- Prior: $p(\theta)$
- Marginal likelihood: $p(y)$
- Posterior: $p(\theta|y)$
- Joint Model: $p(y, \theta)$

Priors in Stan

```
data {
  ...
}
parameters {
  real alpha;  // intercept
  real beta;  // slope
  real<lower=0> sigma;  // residual SD
}
model {
  // likelihood
  y ~ normal(alpha + beta * x, sigma);
  // priors
  alpha ~ normal(0, 100);
  beta ~ normal(0, 50);
  sigma ~ cauchy(0, 50);
}
```

How to obtain the Posterior Distribution?

Problem: Computing the marginal likelihood

$$p(y) = \int p(y|\theta)p(\theta)d\theta$$

Analytically?

- Only possible for specific models

Numerically?

- Only possible for model with few parameters

Solution: Do not compute $p(y)$ at all

Using Samples to Approximate Expectations

Every quantity of interest is an expectation over $p(\theta|y)$:

$$\mathbb{E}[h(\theta)] = \int h(\theta)p(\theta|y)d\theta$$

Approximate expectations using random samples θ_s from $p(\theta|y)$:

$$\mathbb{E}[h(\theta)] \approx \frac{1}{S}\sum_{s=1}^{S} h(\theta_s)$$

Moreover

$$\frac{1}{S}\sum_{s=1}^{S} f(\theta_s) \xrightarrow{S\to\infty} \mathbb{E}(f(\theta))$$

Markov-Chain Monte-Carlo (MCMC) Sampling

We can't simply draw independent samples from the posterior!

A Markov Chain is a sequence of values where the value at position t is based only on the former value at position $t-1$:

$$\theta_1 \to \theta_2 \to \theta_3 \to \ldots \to \theta_S$$

$$p(\theta_t|\theta_{t-1}, \theta_{t-2}, \ldots, \theta_1) = p(\theta_t|\theta_{t-1})$$

If done correctly, the distribution of the values will converge to the target distribution:

$$p(\theta) = \int p(\theta^\star)\, p(\theta|\theta^\star)\, d\,\theta^\star$$

Icecream Sold: Specification in (R)Stan

Prepare the data:

```r
sdata <- list(
  y = icecream$units,
  x = icecream$temp,
  N = nrow(icecream)
)
```

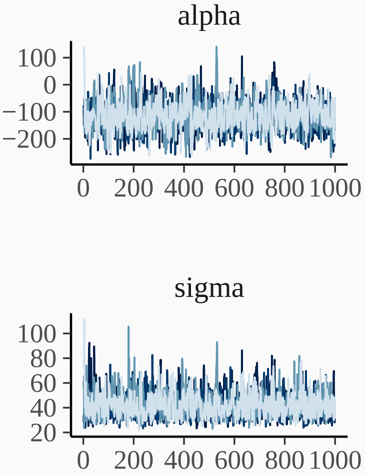
Load rstan:

```r
library(rstan)
```

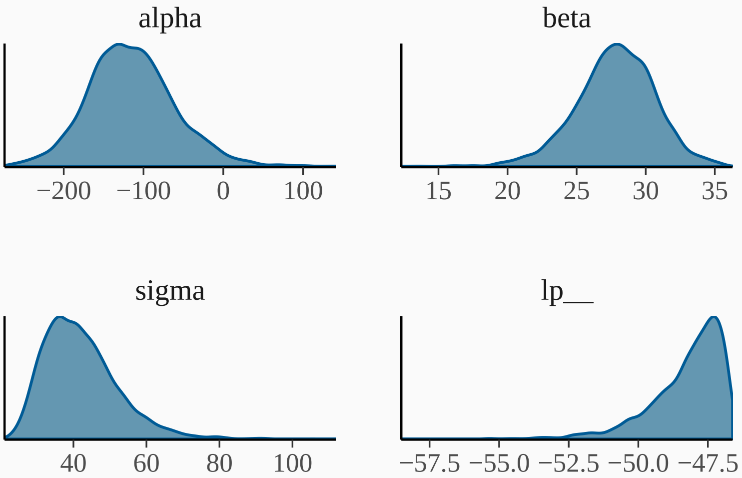
Fit the model:

```r
linreg_model <- stan(file = "linreg.stan", data = sdata)
```

Icecream Sold: Visualize the Chains



Icecream Sold: Visualize the Posterior



Icecream Sold: Summarize the Parameters

```r
print(linreg_model)
```

```
## Inference for Stan model: linreg.
## 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%     50%  97.5% n_eff Rhat
## alpha -114.88    1.61 54.59 -215.62 -117.96   0.23  1151 1.00
## beta    27.80    0.08  2.87   21.64   27.93  33.16  1163 1.00
## sigma   42.04    0.28 10.12   27.18   40.62  65.74  1298 1.00
## lp__   -48.25    0.04  1.30  -51.69  -47.91 -46.75  1025 1.01
##
## Samples were drawn using NUTS(diag_e) at Mon Mar 18 10:40:40 2019.
## 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).
```

Stan overview

- Probabilistic programming language written in C++ ...
- ... to fit open-ended Bayesian models
- Algorithm: (Adaptive) Hamiltonian Monte-Carlo (HMC)
- Automatic differentiation (Stan-Math) library
- Runs on all major platforms (Windows, OS X, Linux)
- Can be called from R, Python, Julia, Stata, and Matlab

Explicitely Constructing the Log-Posterior in Stan

```
data {
  ...
}
parameters {
  real alpha;  // intercept
  real beta;  // slope
  real<lower=0> sigma;  // residual SD
}
model {
  // likelihood
  target += normal_lpdf(y | alpha + beta * x, sigma);
  // priors
  target += normal_lpdf(alpha | 0, 100);
  target += normal_lpdf(beta | 0, 50);
  target += cauchy_lpdf(sigma | 0, 50);
}
```

Stan Syntax: Model Blocks

```
functions
  // user defined Stan functions
data
  // data passed by the user
transformed data
  // variables depending on the data block
  // computed only once before fitting the model
parameters
  // unkown variables to be sampled
transformed parameters
  // variables depending on data and parameter blocks
model
  // specification of the log-posterior density
  // defined variables are local
generated quantities
  // variables to be computed after the model fitting
  // not included in the actual sampling process
```

Why Using Stan?

- Expressive language for probabilistic programming
- Efficient and numerically stable computations
- Powerful MCMC samplers scaling well to high dimensional Bayesian models where other samplers fail
- Continuously developed and improved
- Ecosystem of Stan-related R packages
- Large and friendly community

Learn more about Stan

- Website: http://mc-stan.org/
- Manual: http://mc-stan.org/users/documentation/index.html
- Forums: http://discourse.mc-stan.org/

Selected Publications:

- Carpenter B., Gelman A., Hoffman M. D., Lee D., Goodrich B., Betancourt M., Brubaker M., Guo J., Li P., and Riddell A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*. 76(1). 10.18637/jss.v076.i01
- Gelman A., Lee D., and Guo J. (2015). Stan: A probabilistic programming language for Bayesian inference and optimization. *Journal of Education and Behavioral Statistics*. 40(5):530–543.

The Posterior Predictive Distribution

Distribution of model implied responses $\tilde{y}$ conditional on the existing responses y:

$$p(\tilde{y}|y) = \int p(\tilde{y}|y, \theta)p(\theta|y)\, d\theta$$

For conditionally independent responses:

$$p(\tilde{y}|y) = \int p(\tilde{y}|\theta)p(\theta|y)\, d\theta$$

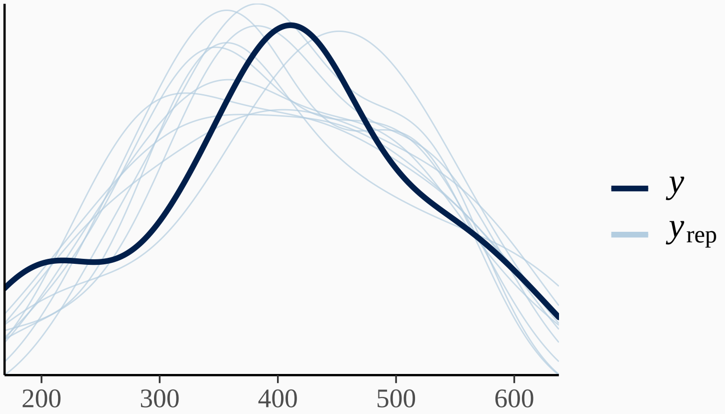
Posterior Predictions in Stan

Sample posterior predictions after model fitting:

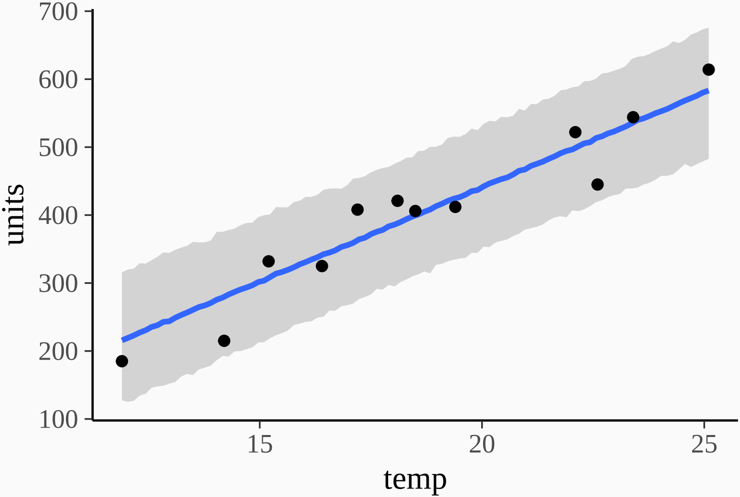
```
...
generated quantites {
  vector[N] yrep;  // posterior predictions
  for (n in 1:N) {
    yrep[n] = normal_rng(alpha + beta * x[n], sigma);
  }
}
```

Icecream Sold: Posterior Predictive Checks

```r
library(bayesplot)
yrep <- as.matrix(linreg_model, pars = "yrep")
yrep <- yrep[sample(1:nrow(yrep), 10), ]
ppc_dens_overlay(y = icecream$units, yrep = yrep)
```



Icecream Sold: Visualize Predictions



Time for exercise ‘stan_linear_regression.R’

What's wrong with our modeling assumptions?

Generalized Linear Models with Stan

Binomial Regression Models

Suppose the icecream market size M is limited

We assume y_n to be binomial distributed with probability θ_n:

$$y_n \sim \text{Binomial}(\theta_n, M)$$

The probability θ_n is predicted via:

$$\theta_n = g(\alpha + \beta x_n)$$

$g(.)$ is a response function for instance

$$g(\eta) = \text{logistic}(\eta) = \frac{\exp(\eta)}{1 + \exp(\eta)}$$

Binomial Model in Stan

```
data {
  int<lower=1> N;  // total number of observations
  int<lower=1> M;  // market size
  int y[N];  // response variable
  vector[N] x;  // predictor variable
}
parameters {
  real alpha;  // intercept
  real beta;  // slope
}
model {
  // likelihood
  for (n in 1:N) {
    real theta = inv_logit(alpha + beta * x[n]);
    y[n] ~ binomial(M, theta);
  }
}
```

Binomial Model in Stan (Optimized)

```
data {
  int<lower=1> N;  // total number of observations
  int<lower=1> M;  // market size
  int y[N];  // response variable
  vector[N] x;  // predictor variable
}
parameters {
  real alpha;  // intercept
  real beta;  // slope
}
model {
  // likelihood
  y ~ binomial_logit(M, alpha + beta * x);;
}
```

Fitting Binomial Models in (R)Stan

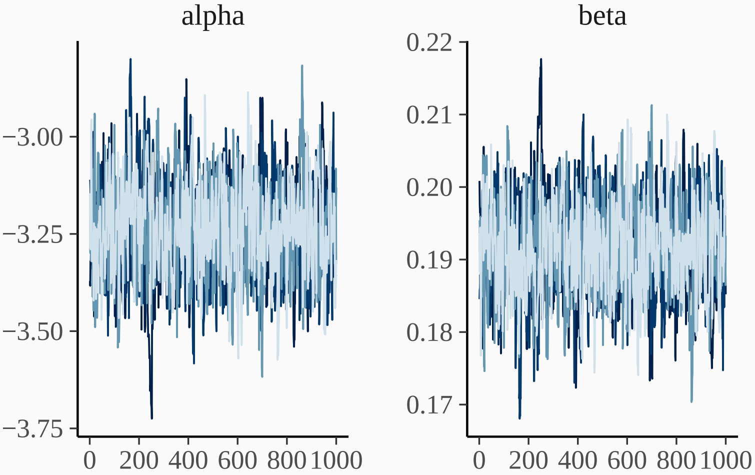
Prepare the data:

```
sdata <- list(
  y = icecream$units,
  x = icecream$temp,
  N = nrow(icecream),
  M = 700
)
```

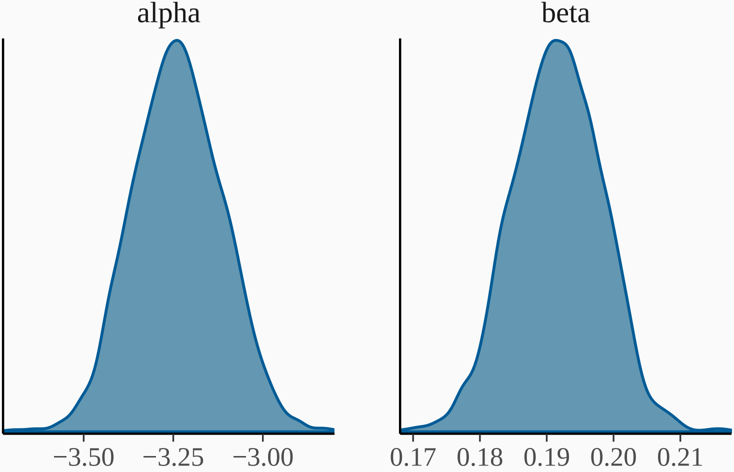
Fit the model:

```
binom_model <- stan(file = "binomial.stan",
                    data = sdata)
```

Binomial Model: Visualize the Chains



Binomial Model: Visualize the Posterior

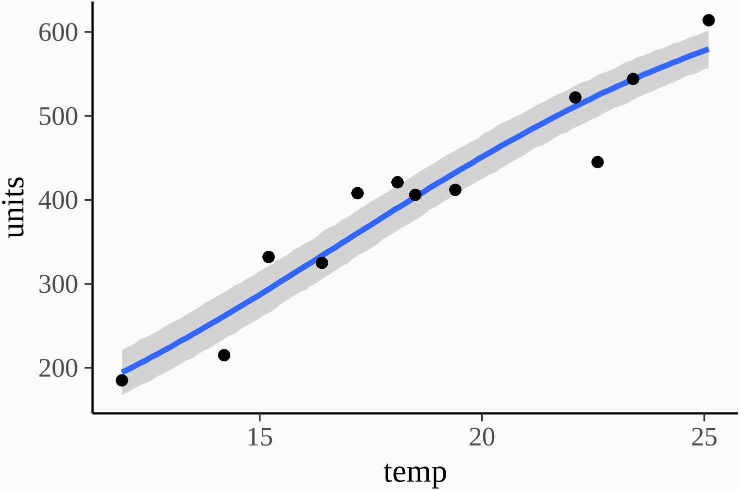


Binomial Model: Summarize the Parameters

```r
print(binom_model)
```

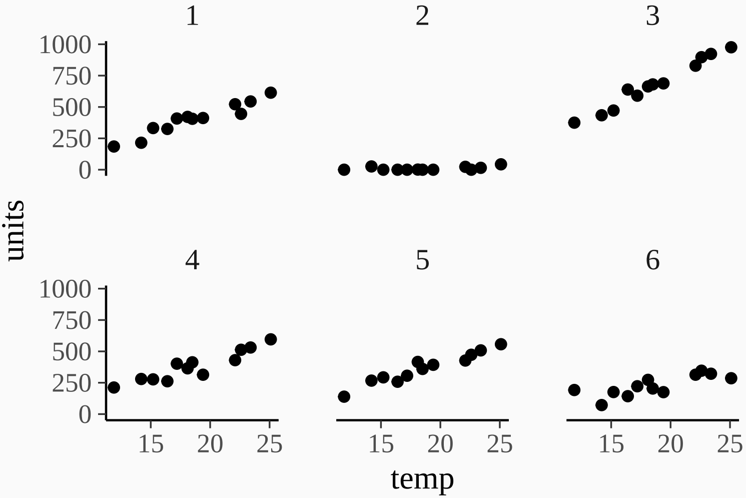
```
## Inference for Stan model: binomial.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean   sd  2.5%   50% 97.5% n_eff Rhat
## alpha -3.24       0 0.12 -3.47 -3.24 -2.99   659 1.01
## beta   0.19       0 0.01  0.18  0.19  0.20   673 1.01
##
## Samples were drawn using NUTS(diag_e) at Mon Mar 18 10:41:29 2019.
## 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).
```

Binomial: Visualize Predictions

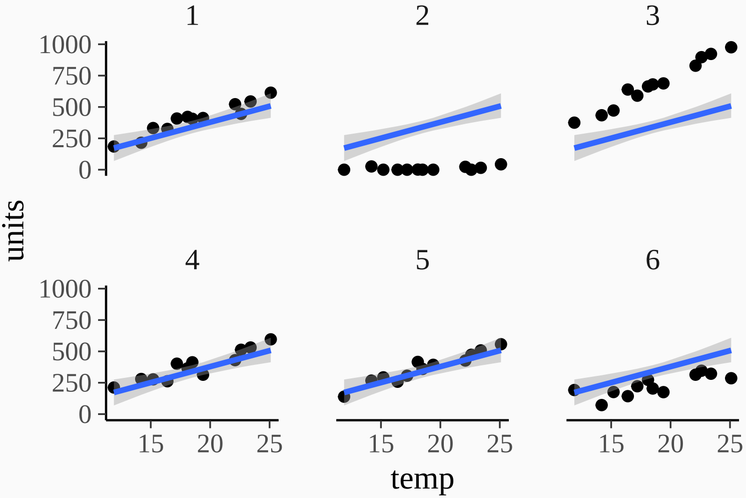


Linear Multilevel Models with Stan

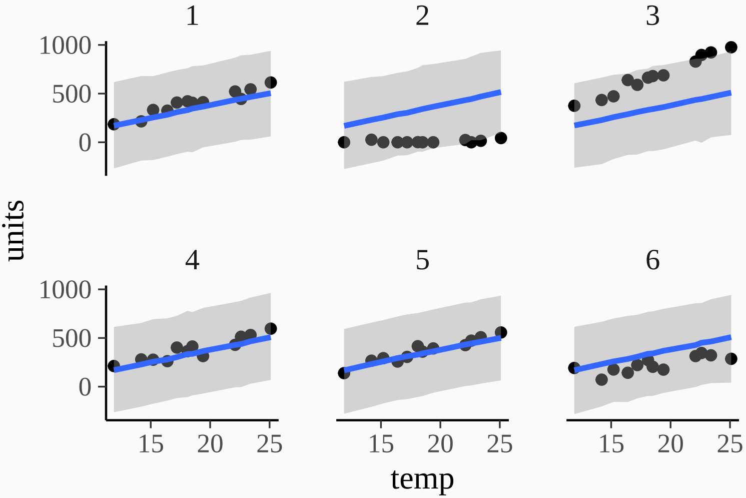
Selling Icecream at Multiple Locations



Simple Linear Model: Visualize Expectations



Simple Linear Model: Visualize Predictions



Varying Intercept Models

We assume the following generative model:

$$y_n \sim \text{Normal}(\alpha_{j_n} + \beta x_n, \sigma)$$

with

$$\alpha_j \sim \text{Normal}(\mu_\alpha, \tau_\alpha)$$

or equivalently

$$\tilde{\alpha}_j \sim \text{Normal}(0, 1)$$

$$\alpha_j = \mu_\alpha + \tau_\alpha \times \tilde{\alpha}_j$$

Varying Intercept Model in Stan (Centered)

```
data {
  ...
  int<lower=1> Nlocation;  // number of locations
  int<lower=1> location[N];  // location index
}
parameters {
  vector[Nlocation] alpha;  // intercepts
  real mu_alpha;  // intercept mean
  real<lower=0> tau_alpha;  // intercept SD
  ...
}
model {
  vector[N] mu;
  for (n in 1:N) {
    mu[n] = alpha[location[n]] + beta * x[n];
  }
  y ~ normal(mu, sigma);
  alpha ~ normal(mu_alpha, tau_alpha);
}
```

Varying Intercept Model in Stan (Non-Centered)

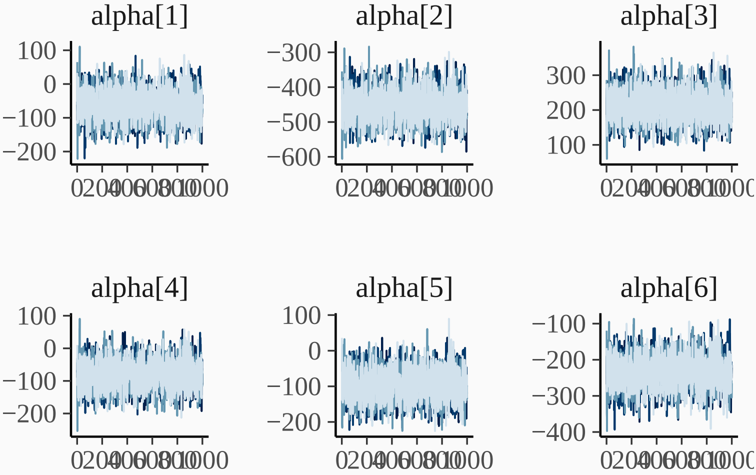
```
...
parameters {
  vector[Nlocation] z_alpha;  // dummy intercepts
  real mu_alpha;  // intercept mean
  real<lower=0> tau_alpha;  // intercept SD
  ...
}
transformed parameters {
  vector[Nlocation] alpha = mu_alpha + tau_alpha * z_alpha;
}
model {
  vector[N] mu;
  for (n in 1:N) {
    mu[n] = alpha[location[n]] + beta * x[n];
  }
  y ~ normal(mu, sigma);
  z_alpha ~ normal(0, 1);
}
```

Fitting Varing Intercept Models in (R)Stan

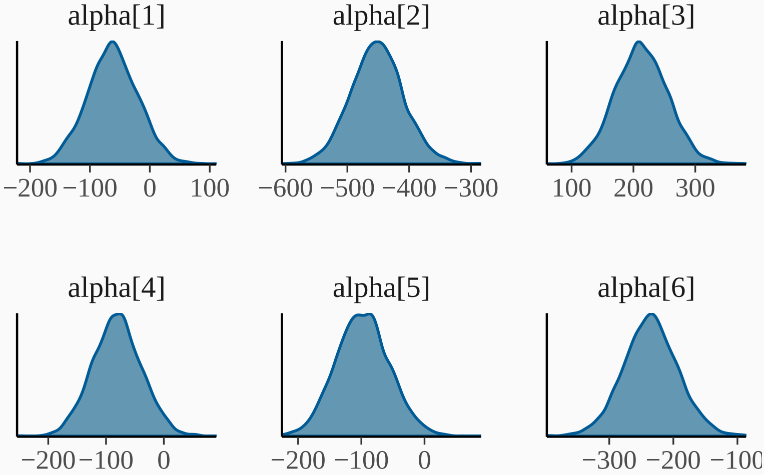
```r
sdata <- list(
  y = icecream2$units,
  x = icecream2$temp,
  location = icecream2$location,
  N = nrow(icecream2),
  Nlocation = length(unique(icecream2$location))
)
```

```r
mlm2 <- stan(
  "stanmodels/multilevel_intercept2.stan",
  data = sdata
)
```

Varying Intercepts: Visualize the Chains



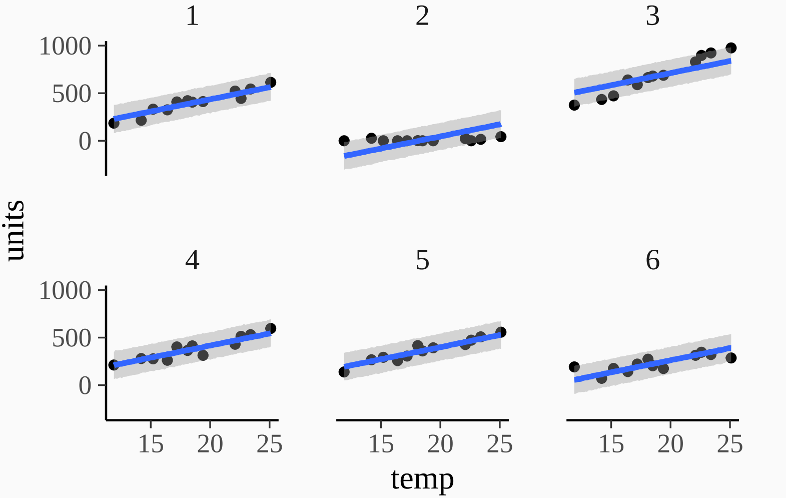
Varying Intercepts: Visualize the Posterior



Varying Intercept Model: Summarize the Parameters

```
## Inference for Stan model: multilevel_intercept2.
## 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%     50%   97.5% n_eff Rhat
## alpha[1]   -61.87    0.81 42.27 -143.98  -62.31   21.68  2730    1
## alpha[2]  -451.74    0.81 42.72 -535.12 -451.99 -364.54  2749    1
## alpha[3]   213.32    0.76 42.40  130.52  212.45  295.75  3089    1
## alpha[4]   -81.09    0.83 42.48 -163.88  -81.36    4.28  2603    1
## alpha[5]   -97.67    0.81 42.27 -178.43  -98.39  -12.67  2690    1
## alpha[6]  -235.23    0.83 42.80 -319.03 -235.51 -149.72  2668    1
## mu_alpha   -65.85    1.82 70.23 -192.87  -68.37   83.86  1494    1
## tau_alpha  228.38    2.45 75.55  128.77  213.31  420.04   951    1
## beta        24.87    0.04  2.02   20.87   24.84   28.80  2592    1
## sigma       68.78    0.12  6.10   57.93   68.45   82.07  2524    1
##
## Samples were drawn using NUTS(diag_e) at Mon Mar 18 10:44:11 2019.
## 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).
```

Varying Intercept Model: Visualize Predictions



Varying Slope Models (Centered)

We assume the following generative model:

$$y_n \sim \text{Normal}(\alpha_{j_n} + \beta_{j_n} x_n, \sigma)$$

with

$$(\alpha_j, \beta_j) \sim \text{MultiNormal}((\mu_\alpha, \mu_\beta), \Sigma)$$

$$\Sigma = \begin{pmatrix} \tau_\alpha^2 & \tau_\alpha \tau_\beta \rho_{\alpha\beta} \\ \tau_\alpha \tau_\beta \rho_{\alpha\beta} & \tau_\beta^2 \end{pmatrix}$$

Varying Slope Models (Non-Centered)

We assume the following generative model:

$$y_n \sim \mathcal{N}(\alpha_{j_n} + \beta_{j_n} x_n, \sigma)$$

with

$$\tilde{\alpha}_j, \tilde{\beta}_j \sim \mathrm{Normal}(0, 1)$$

$$(\alpha_j, \beta_j) = (\mu_\alpha, \mu_\beta) + L \times (\tilde{\alpha}_j, \tilde{\beta}_j)$$

where L is the Cholesky factor of Σ:

$$\Sigma = LL^{\mathsf{T}}$$

We may also write L as:

$$L = \mathrm{Diag}(\tau_\alpha, \tau_\beta) L_\rho$$

Varing Slope Models in Stan (Non-Centered Part 1)

```
...
parameters {
  real mu_alpha;  // intercept mean
  real mu_beta;  // slope mean
  real<lower=0> tau_alpha;  // intercept SD
  real<lower=0> tau_beta;  // slope SD
  // cholesky factor of the correlation matrix
  cholesky_factor_corr[2] L_Cor;
  matrix[2, Nlocation] z_theta;  // dummy varying effects
  real<lower=0> sigma;  // residual SD
}
```

Varing Slope Models in Stan (Non-Centered Part 2)

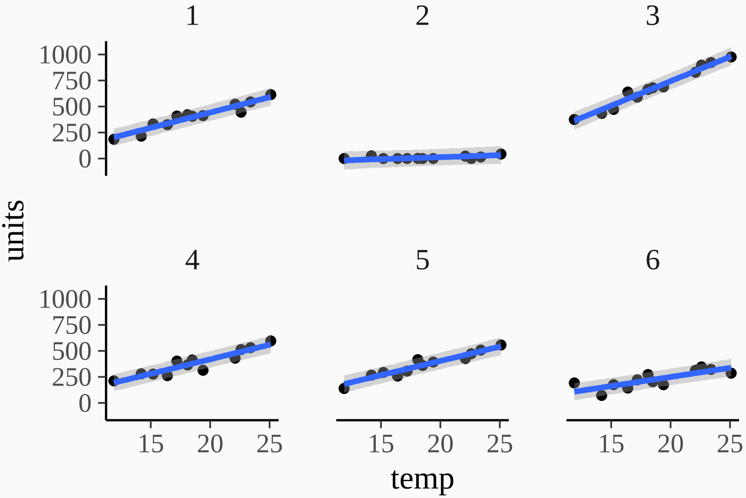
```
...
transformed parameters {
  // cholesky factor of the covariance matrix
  matrix[2, 2] L_Sigma =
    diag_pre_multiply([tau_alpha, tau_beta]', L_Cor);
  matrix[2, Nlocation] theta;  // actual varying effects
  for (j in 1:Nlocation) {
    theta[, j] = [mu_alpha, mu_beta]' + L_Sigma * z_theta[, j];
  }
}
model {
  vector[N] mu;
  for (n in 1:N) {
    mu[n] = theta[1, location[n]] + theta[2, location[n]] * x[n];
  }
  y ~ normal(mu, sigma);
  to_vector(z_theta) ~ normal(0, 1);
}
```

Fitting Varing Slope Models in (R)Stan

```r
sdata <- list(
  y = icecream2$units,
  x = icecream2$temp,
  location = icecream2$location,
  N = nrow(icecream2),
  Nlocation = length(unique(icecream2$location))
)
```

```r
mlm3 <- stan(
  "stanmodels/multilevel_slope.stan",
  data = sdata,
  control = list(adapt_delta = 0.99)
)
```

Varying Slope Model: Visualize Predictions



Cross-Validation

Idea: Estimate the model based on one part of the data (*training* data) to predict the other part of the data (*test* data)

Special case: *Leave-One-Out* cross-validation (LOO-CV):

$$p(y_i|y_{-i}) = \int p(y_i|\theta)p(\theta|y_{-i})\, d\,\theta \approx \frac{1}{S}\sum_{s=1}^{S} p(y_i|\theta_{-i,s})$$

Criterion for model comparison:

$$\text{ELPD}_{\text{loo}} = \sum_{n=1}^{N} \log p(y_i|y_{-i})$$

Disadvantage of CV: Requires fitting the model multiple times

Solution: Replace $p(\theta|y_{-i})$ by $p(\theta|y)$ and adjust the result via Pareto-Smoothed Importance Sampling (PSIS)

Icecream Sold: Compute Log-Likelihood Values

Compute log-likelihoods values after model fitting:

```
...
generated quantities {
  vector[N] ll;  // log-likelihood values
  for (n in 1:N) {
    ll[n] = normal_lpdf(y[n] | alpha + beta * x[n], sigma);
  }
}
```

Approximate LOO-CV (Constant Intercept)

```r
library(loo)
ll <- as.matrix(lm1, pars = "ll")
(loo_lm1 <- loo(ll))
```

```
##
## Computed from 4000 by 72 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -490.6  6.4
## p_loo         2.5  0.5
## looic       981.1 12.8
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

Approximate LOO-CV (Varying Intercepts)

```r
ll <- as.matrix(mlm2, pars = "ll")
(loo_mlm2 <- loo(ll))
```

```
##
## Computed from 4000 by 72 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -411.4  6.4
## p_loo         8.4  1.5
## looic       822.8 12.9
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

Approximate LOO-CV (Varying Intercepts and Slopes)

```r
ll <- as.matrix(mlm3, pars = "ll")
(loo_mlm3 <- loo(ll))
```

```
##
## Computed from 4000 by 72 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -370.4  5.8
## p_loo         9.1  1.5
## looic       740.7 11.6
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     70    97.2%   941
##  (0.5, 0.7]   (ok)        2     2.8%   815
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
```

Comparing Models via Approximate LOO-CV

```r
loo_compare(loo_lm1, loo_mlm2, loo_mlm3)
```

```
##        elpd_diff se_diff
## model3    0.0       0.0
## model2  -41.0       7.6
## model1 -120.2       9.6
```

The R Universe around Stan

- **rstan**: Call Stan from R
- **rstantools**: Development Tools for Stan-Based Packages
- **rstanarm**: Bayesian Applied Regression Modeling via Stan
- **bayesplot**: Plotting for Bayesian models
- **shinystan**: Interactive Plotting for Bayesian Models
- **loo**: Approximation Cross-Validation
- **projpred**: Variable Selection via Projective Predictions
- **brms**: Bayesian Regression Models using Stan

The brms package

A unified framework for Bayesian regression modeling

Selling Icecream with brms

Linear Model:

```r
brm(units ~ temp, data = icecream)
```

Binomial Model:

```r
brm(units | trials(size) ~ temp, data = icecream,
    family = binomial("logit"))
```

Varying Intercept Model:

```r
brm(units ~ temp + (1 | location), data = icecream)
```

Varying Slope Model:

```r
brm(units ~ temp + (temp | location), data = icecream)
```

Learn more about brms

- Help within R: `help("brms")`
- Vignettes: `vignette(package = "brms")`
- List of all methods: `methods(class = "brmsfit")`
- Website: https://github.com/paul-buerkner/brms
- Forums: http://discourse.mc-stan.org/
- Contact me: `paul.buerkner@gmail.com`
- Twitter: @paulbuerkner

Publications

- Bürkner P. C. (2017). brms: An R Package for Bayesian Multilevel Models using Stan. *Journal of Statistical Software*. 80(1), 1-28. doi:10.18637/jss.v080.i01
- Bürkner P. C. (2018). Advanced Bayesian Multilevel Modeling with the R Package brms. *The R Journal*. 10(1), 395–411. doi:10.32614/RJ-2018-017

Questions?

Appendix

Stan syntax: Multiple Linear Regression

```
data {
  int<lower=1> N;  // total number of observations
  vector[N] y;  // response variable
  int<lower=1> K;  // number of regression coefficients
  matrix[N, K] X;  // predictor design matrix
}
parameters {
  vector[K] b;  // regression coeffcients
  real<lower=0> sigma;  // residual SD
}
model {
  vector[N] mu;
  mu = X * b;
  y ~ normal(mu, sigma);  // likelihood
}
```

Importance Sampling

Suppose $f(\theta)$ is the target and $g(\theta)$ the approximating distribution:

$$\begin{aligned}\mathbb{E}[h(\theta)] = \int h(\theta)f(\theta)\,d\,\theta &= \frac{\int[h(\theta)f(\theta)/g(\theta)]g(\theta)\,d\,\theta}{\int[f(\theta)/g(\theta)]g(\theta)\,d\,\theta} \\ &= \frac{\int h(\theta)r(\theta)g(\theta)\,d\,\theta}{\int r(\theta)g(\theta)\,d\,\theta}\end{aligned}$$

with importance ratios

$$r(\theta) = \frac{f(\theta)}{g(\theta)}$$

If $\theta^{(s)}$ are S random draws from $g(\theta)$:

$$\mathbb{E}[h(\theta)] \approx \frac{\sum_{s=1}^{S} h(\theta^{(s)})r(\theta^{(s)})}{\sum_{s=1}^{S} r(\theta^{(s)})}$$

Some Helpful brms Functions (1)

Specify the model using R formulas:

```
brmsformula(formula, ...)
```

Generate the Stan code:

```
make_stancode(formula, ...)
stancode(fit)
```

Generate the data passed to Stan:

```
make_standata(formula, ...)
standata(fit)
```

Handle priors:

```
get_prior(formula, ...)
set_prior(prior, ...)
```

Some Helpful brms Functions (2)

Generate expected values and predictions:

```
fitted(fit, ...)
predict(fit, ...)
marginal_effects(fit, ...)
```

Model comparison:

```
loo(fit1, fit2, ...)
bayes_factor(fit1, fit2, ...)
model_weights(fit1, fit2, ...)
```

Hypothesis testing:

```
hypothesis(fit, hypothesis, ...)
```