Introduction to Bayesian Statistics

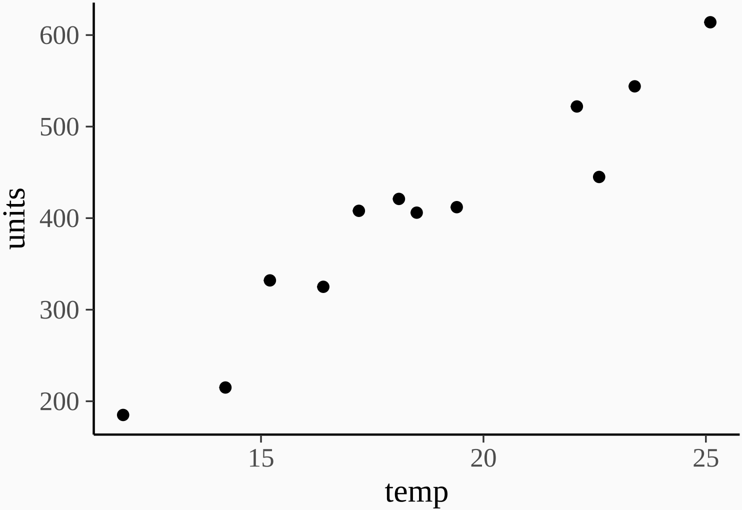
with practical examples in Stan

Paul Bürkner

Bayesian Data Analysis

> “If you quantify uncertainty with probability, you are a Bayesian.”
>
> Michael Betancourt

Example: Icecream Sold at Different Temperatures



Simple Linear Regression

We assume the following data 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

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

Bayesian Simple Linear Regression

We assume the following likelihood:

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

We assume the following *prior distributions*:

$$\alpha \sim \text{normal}(0, 100)$$

$$\beta \sim \text{normal}(0, 50)$$

$$\sigma \sim \text{exponential}(1/50)$$

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: $p(y,\theta)$

Stan: A Probabilistic Programming Language

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);
}
```

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 ~ exponential(1 / 50);
}
```

Log-Distributions and Loss-Functions

Log-Posterior:

$$\begin{aligned}\log(p(\theta|y)) &= \log(p(y|\theta)) + \log(p(\theta)) + C \\ &= \log(p(y|\theta)) + \log(p(\theta_1)) + \log(p(\theta_2)) + C\end{aligned}$$

for independent priors on θ_1 and θ_2

Regularized Loss-Functions:

$$\begin{aligned}C(y, \theta) &= L(y, \theta) + R(\theta) \\ &= L(y, \theta) + R_1(\theta_1) + R_2(\theta_2)\end{aligned}$$

for independent regularizing terms on θ_1 and θ_2

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 += exponential_lpdf(sigma | 1 / 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

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

$$\mathbb{E}_p(h) = \int h(\theta)\, p(\theta \mid y)\, \mathrm{d}\theta$$

Having obtained exact random samples $\{\theta_s\}$ from $p(\theta \mid y)$:

$$\frac{1}{S}\sum_{s=1}^{S} h(\theta_s) \sim \text{Normal}\left(\mathbb{E}_p(h), \sqrt{\frac{\text{Var}_p(h)}{\text{S}}}\right)$$

Rejection Sampling

(1) Sample parameter values from the prior

(2) Sample data from the likelihood based on the sampled parameters

(3) Only keep those parameter values, which produced data consistent with our observed data

(4) Repeat steps (1) – (3) many times

The kept parameter values are exact random samples from the posterior!

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 $t-1$:

$$\theta_1 \rightarrow \theta_2 \rightarrow \theta_3 \rightarrow \ldots \rightarrow \theta_S$$

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

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

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

Example: The Metropolis-Algorithm

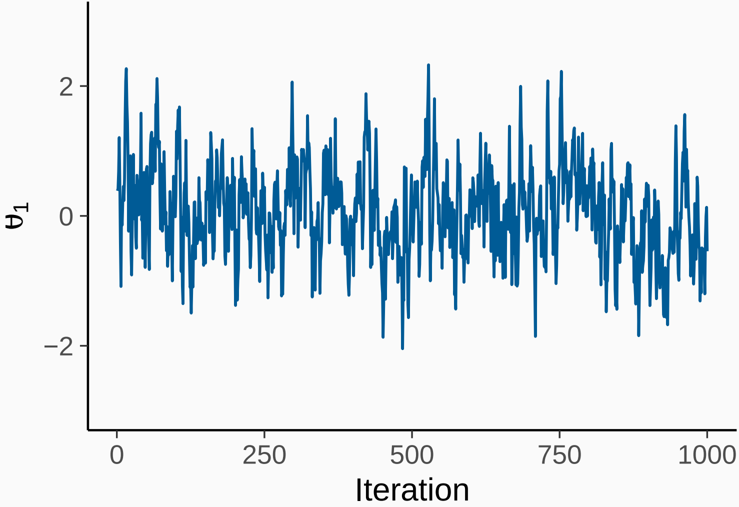
- Choose an initial value θ_1. Set $t = 1$.
- Sample a possible new value θ_p based on a *proposal distribution* $g(\theta_p|\theta_t)$ – usually use $N(\theta_t, \tau)$ as the proposal distribution
- (τ serves as a tuning parameter controlling the *step-size*)
- Compute the ratio $\alpha = p(\theta_p|y)/p(\theta_t|y)$
- If $\alpha \geq 1$, set $\theta_{t+1} = \theta_p$.
- If $\alpha < 1$, set $\theta_{t+1} = \theta_p$ with probability α
- Else, go back to step 2 and sample new value θ_p

Markov-Chain Monto-Carlo Estimator

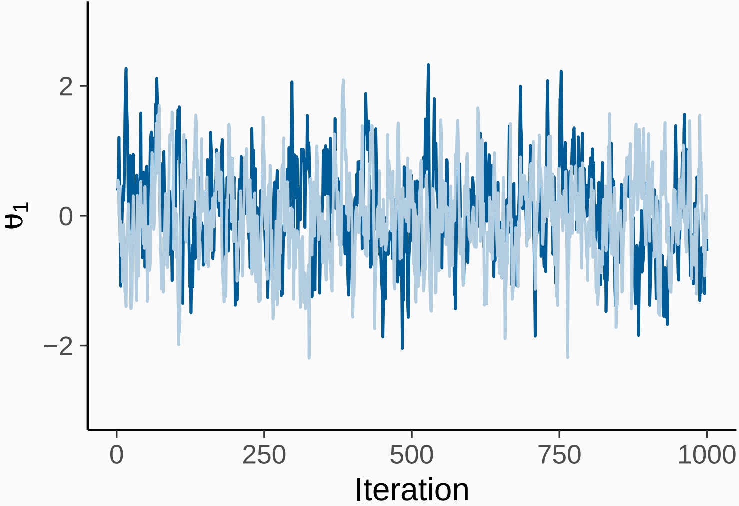
Assuming *geometric ergodicity* of a Markov Chain $\{\theta_s\}$:

$$\frac{1}{S}\sum_{s=1}^{S} h(\theta_s) \sim \text{Normal}\left(\mathbb{E}_p(h), \sqrt{\frac{\text{Var}_p(h)}{\text{ESS}}}\right)$$

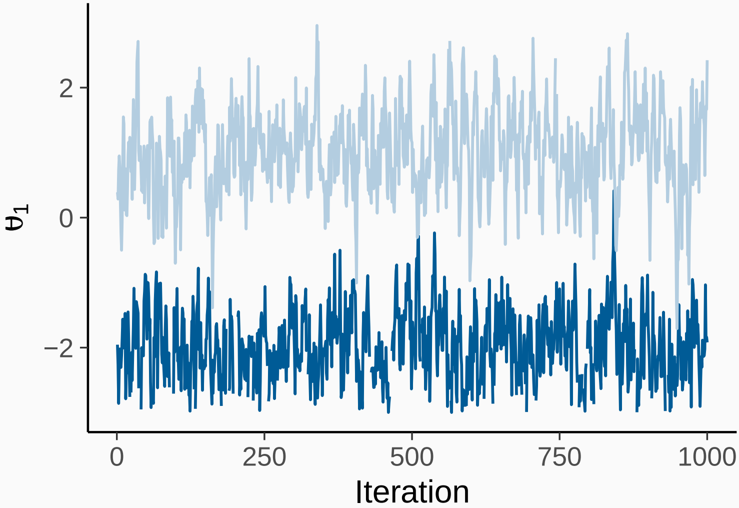
Trace Plots: Visualizing a Single Chain



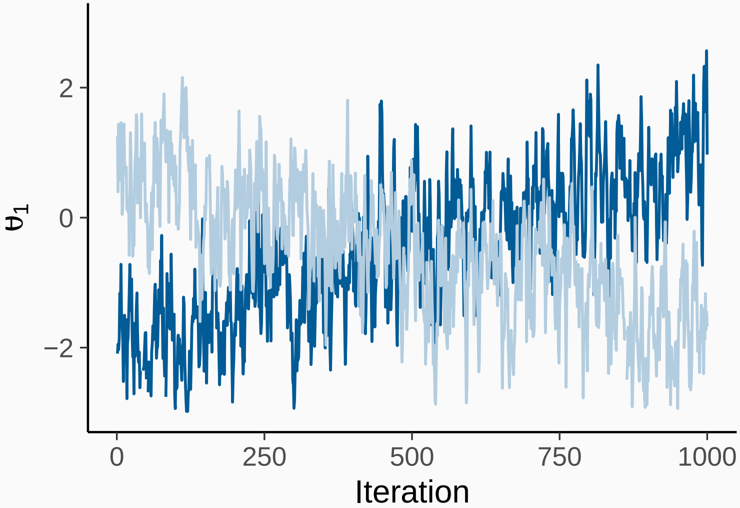
Trace Plots: Visualizing Multiple Chains



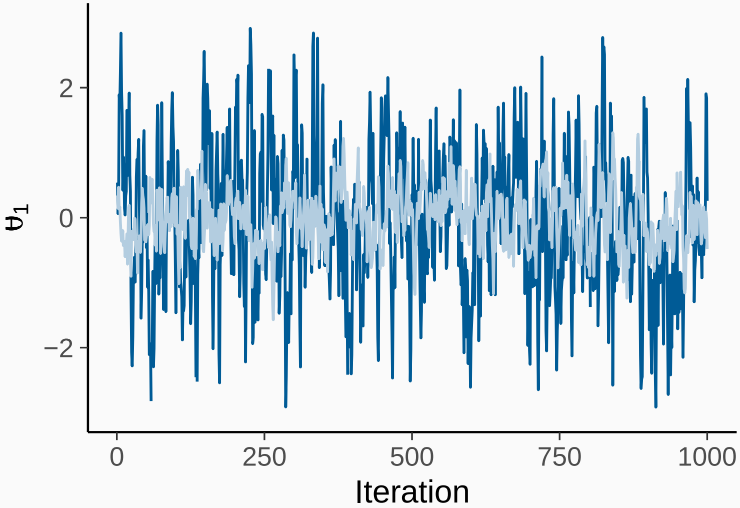
Chains with Different Locations



Non-Stationary Chains



Chains with Different Variances



Traditional MCMC Diagnostics

Between Chain Variance:

$$B = \frac{N}{M-1} \sum_{m=1}^{M} (\overline{\theta}^{(.m)} - \overline{\theta}^{(..)})^2$$

Within Chain Variance:

$$W = \frac{1}{M(N-1)} \sum_{m=1}^{M} \sum_{n=1}^{N} (\theta^{(nm)} - \overline{\theta}^{(.m)})^2$$

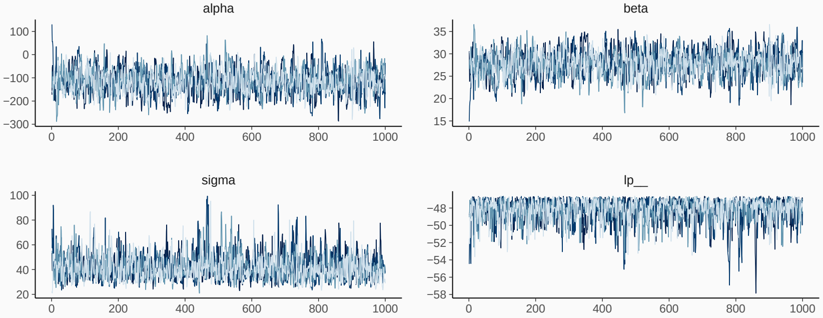
Potential Scale Reduction Factor:

$$\widehat{R} = \sqrt{\frac{\frac{N-1}{N} W + \frac{1}{N} B}{W}}$$

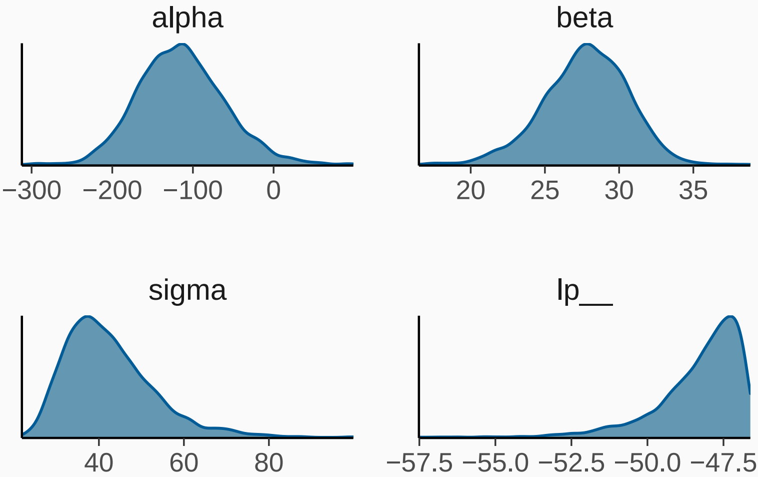
Effective Sample Size:

$$\mathrm{ESS} = \frac{N\,M}{\hat{\tau}}$$

Icecream Sold: Visualize the Chains



Icecream Sold: Visualize the Posterior



Icecream Sold: Summarize the Parameters

```
##   variable mean median   sd  mad   q5 q95 rhat ess_bulk ess_tail
## 1    alpha -114   -116 54.1 51.9 -198 -21    1      996     1206
## 2     beta   28     28  2.9  2.8   23  32    1      983     1130
## 3    sigma   42     41 10.4  9.5   29  62    1     1314     1276
## 4     lp__  -48    -48  1.4  1.1  -51 -47    1      975     1401
```

Posterior Probabilities

Applicable to interval hypotheses – examples:

If $H : \theta > 0$ then

$$P(H) = P(\theta > 0) = \frac{1}{S} \sum_{s=1}^{S} 1_{>0}(\theta_s)$$

If $H : \theta \in [10, 20]$ then

$$P(H) = P(\theta \in [10, 20]) = \frac{1}{S} \sum_{s=1}^{S} 1_{[10,20]}(\theta_s)$$

- $S =$ Number of posterior samples
- $\theta_s =$ Posterior sample number s of parameter θ
- $1_I(x) = 1$ if x is in the interval I and $1_I(x) = 0$ otherwise

Transformation of Parameters

The Posterior does not only contain information of each parameter, separately, but also about the *dependencies* of the parameters.

The dependencies are reflected in the posterior draws which can be transformed arbitrarily

Simple example: Difference δ of two parameters θ_1 and θ_2

For every posterior sample *s* compute:

$$\delta_s = \theta_{1s} - \theta_{2s}$$

Then, the set $\{\delta_s\}$ forms the posterior of δ

The computation of summary statistics should always be done *after* all parameter transformation!

Transformation Example: Selling Icecream

Let `alpha` and `beta` be vectors of posterior samples

Compute posterior prediction for 30 degree celsius:

```
pred = alpha + beta * 30
```

```
##   variable mean median   sd  mad  q5 q95 rhat ess_bulk ess_tail
## 1     pred  718    720 35.9 33.6 657 773    1     1140     1395
```

Advantages and Disadvantages of Bayesian Statistics

Advantages:

- Natural approach to expressing uncertainty
- Ability to incorporate prior information
- Increased modeling flexibility
- Full posterior distribution of parameters
- Natural propagation of uncertainty

Disadvantages:

- Slow Speed of model estimation

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:

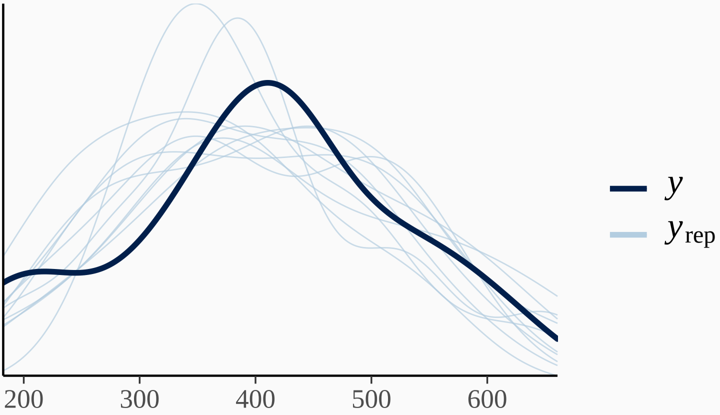
$$p(\tilde{y}|y) = \int p(\tilde{y}|\theta)p(\theta|y)\,d\theta$$

Posterior Predictions in Stan

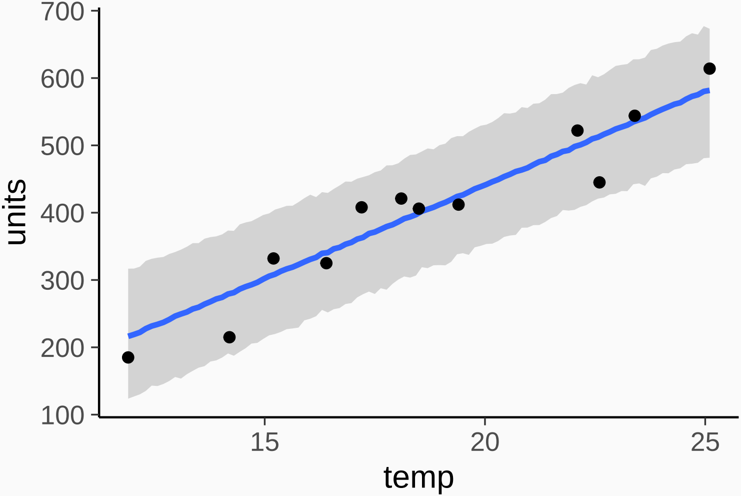
Sample posterior predictions after model fitting:

```
...
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



Icecream Sold: Visualize Predictions



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 coefficients
  real<lower=0> sigma;  // residual SD
}
model {
  vector[N] mu;
  mu = X * b;
  y ~ normal(mu, sigma);  // likelihood
}
```

What's wrong with our modeling assumptions?

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

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

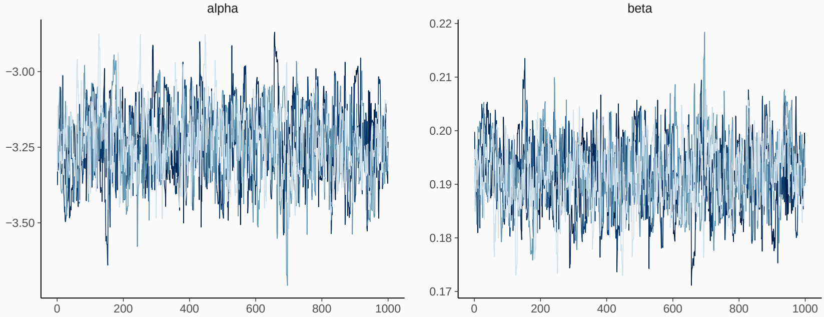
Binomial Model in Stan

```
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);
  }
}
```

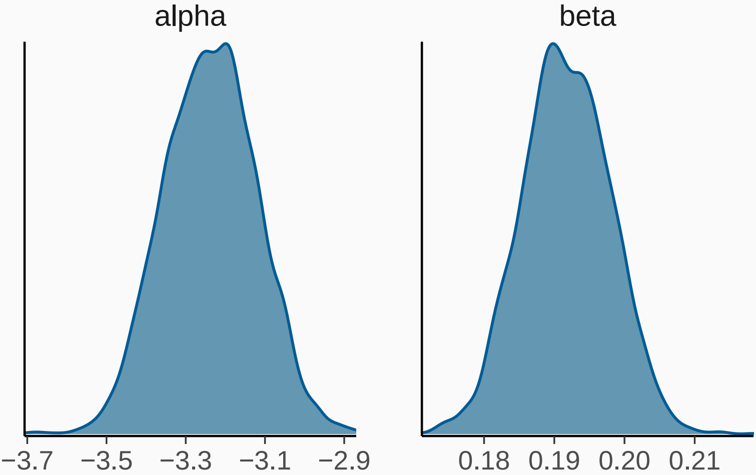
Binomial Model in Stan (Optimized)

```
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);;
}
```

Binomial Model: Visualize the Chains



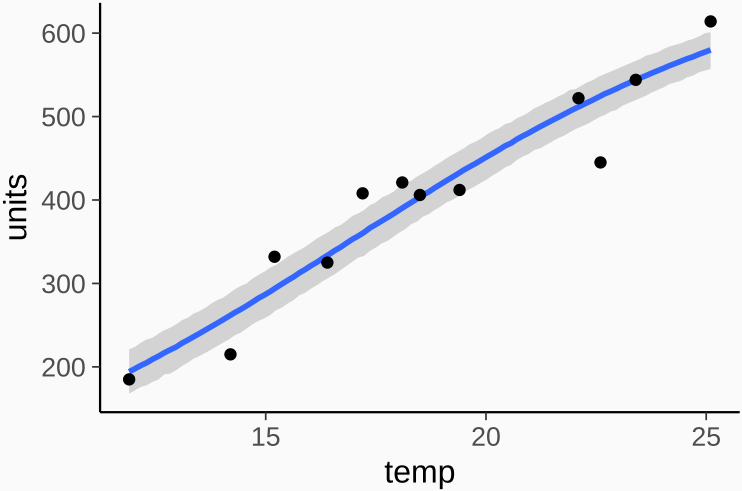
Binomial Model: Visualize the Posterior



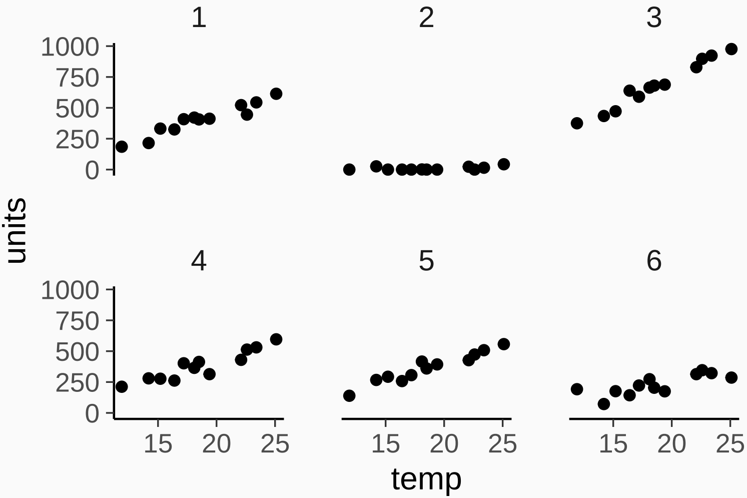
Binomial Model: Summarize the Parameters

```
##   variable  mean median     sd    mad    q5  q95 rhat ess_bulk ess_tail
## 1    alpha -3.24  -3.24 0.1166 0.1197 -3.43 -3.0    1      567      769
## 2     beta  0.19   0.19 0.0063 0.0064  0.18  0.2    1      566      818
```

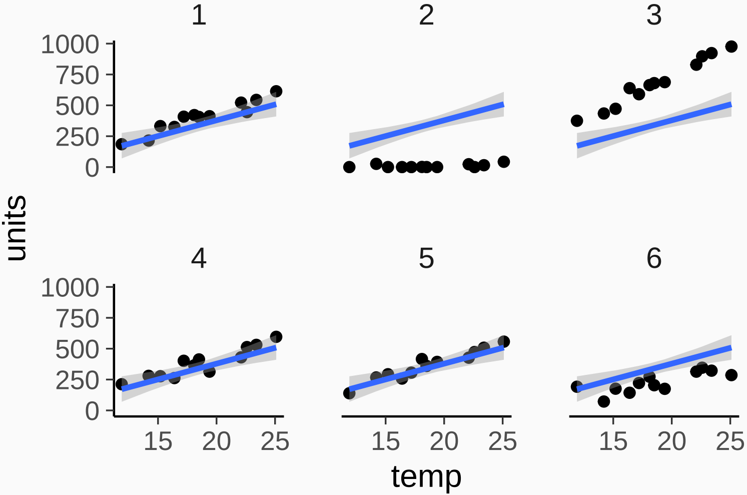
Binomial: Visualize Predictions



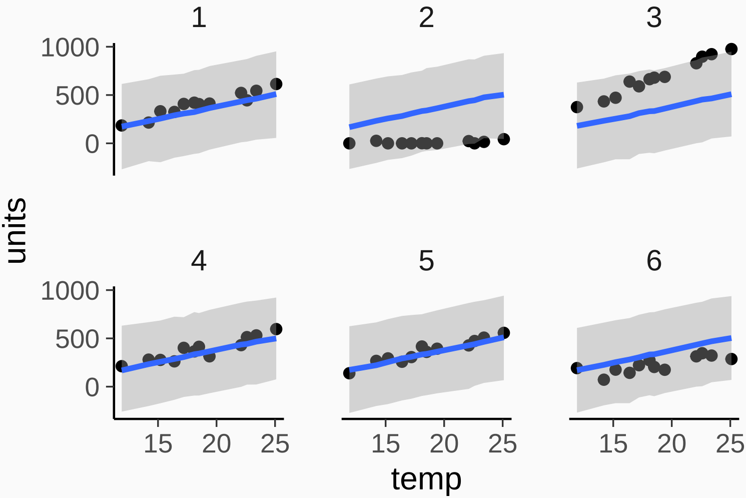
Selling Icecream at Multiple Locations



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

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

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

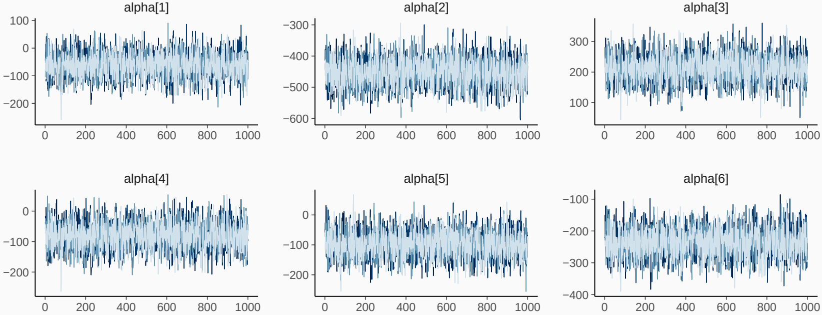
Varying Intercept Model in Stan (Centered)

```
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);
}
```

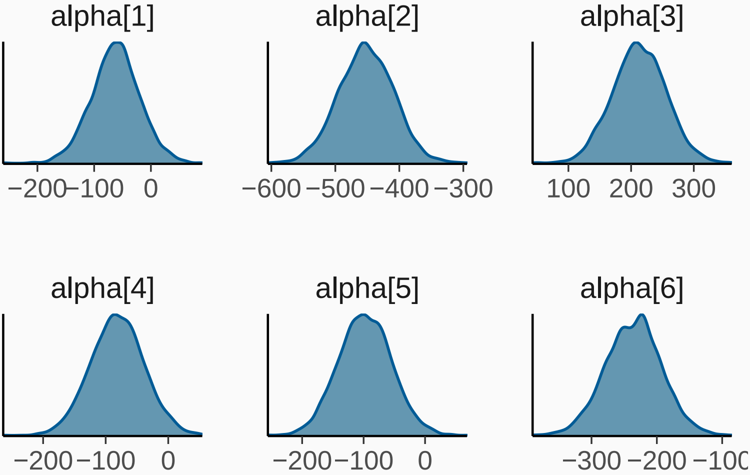
Varying Intercept Model in Stan (Non-Centered)

```
...
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);
}
```

Varying Intercepts: Visualize the Chains



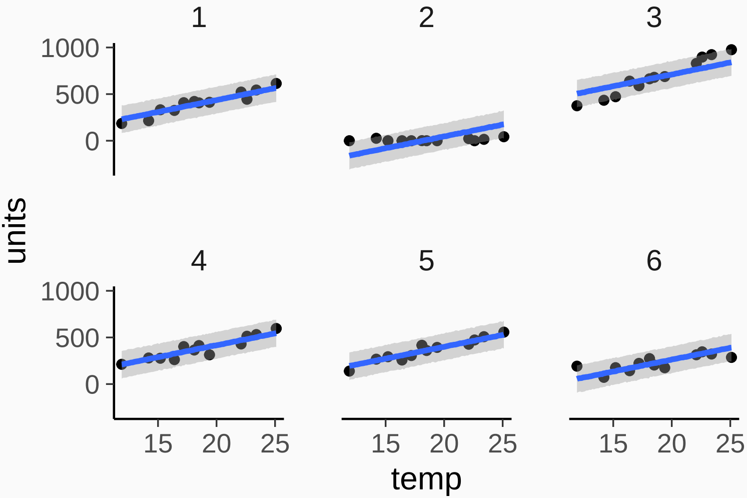
Varying Intercepts: Visualize the Posterior



Varying Intercept Model: Summarize the Parameters

```
##     variable mean median   sd mad   q5    q95 rhat ess_bulk ess_tail
## 1   alpha[1]  -62    -62 42.7  41 -133    7.3    1     2732     2084
## 2   alpha[2] -452   -452 43.2  43 -523 -382.6    1     2849     2651
## 3   alpha[3]  213    213 42.8  42  142  281.9    1     2759     2486
## 4   alpha[4]  -81    -81 42.8  43 -151  -10.4    1     2876     2521
## 5   alpha[5]  -98    -98 43.1  43 -169  -27.5    1     2800     2596
## 6   alpha[6] -236   -235 42.8  42 -307 -166.9    1     2869     2442
## 7   mu_alpha  -70    -72 73.5  73 -186   54.2    1     1249     1527
## 8  tau_alpha  228    214 76.7  62  136  375.7    1      932     1379
## 9       beta   25     25  2.0   2   22   28.2    1     2531     1970
## 10     sigma   69     68  6.1   6   60   79.6    1     2618     2181
```

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

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

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

where L is the Cholesky factor of Σ:

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

We may also write L as:

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

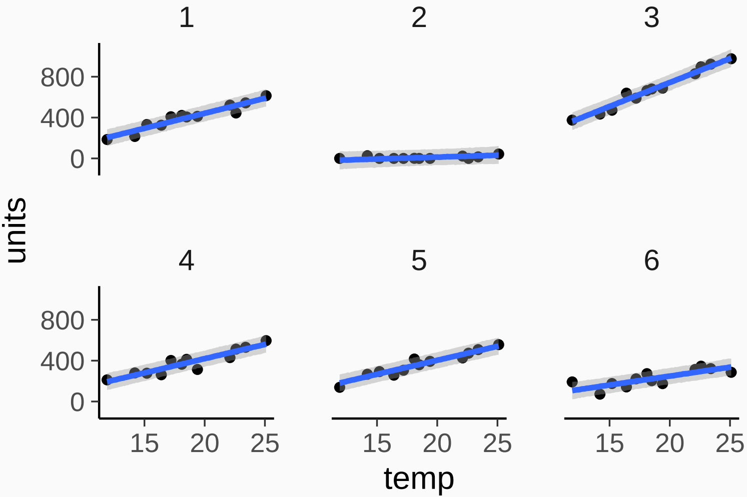
Varing Slope Models in Stan (Non-Centered Part 1)

```
...
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);
}
```

Varying Slope Model: Visualize Predictions



Does including ‘location’ improve model fit?

In-sample vs. out-of-sample fit

In-sample fit:

- How close are the model's predictions to the data it was estimated on?
- Problem: High danger of overfitting

Out-of-sample fit:

- How close are the model's predictions to new data?
- Balances under- and overfitting
- Problem: How do we evaluate predictions on new data without actual new data?

Cross-Validation

Steps in cross-validation:

(1) Split the data into two Subsets: training data and test data
(2) Fit the model on the training data
(3) Evaluate the predictions on the test data
(4) Repeat (1) to (3) with multiple data splits
(5) Summarize the results of all splits

Types of cross-validation (selection):

- Leave-one-out cross-validation (LOO-CV)
- K-fold cross-validation (K-fold-CV)
- Leave-group-out cross-validation (LGO-CV)
- Leave-future-out cross-validation (LFO-CV)

Measures of Predictive Accuracy / Utility

Example measures for a single data split:

$$\text{ELPD} = \log\ p(y|y_{\mathrm{Tr}}) = \log \int p(y|\theta)\ p(\theta|y_{\mathrm{Tr}})\ d\theta \approx \log\ \frac{1}{S}\sum_{s=1}^{S} p(y|\theta^{(s)})$$

$$\text{RMSE} = \sqrt{\int (y-\hat{y})^2\ p(\hat{y}|y_{\mathrm{Tr}})\ d\hat{y}} \approx \sqrt{\frac{1}{S}\sum_{s=1}^{S}(y-\hat{y}^{(s)})^2}$$

$$\text{MAE} = \int |y-\hat{y}|\ p(\hat{y}|y_{\mathrm{Tr}})\ d\hat{y} = \frac{1}{S}\sum_{s=1}^{S}|y-\hat{y}^{(s)}|$$

Leave-One-Out Cross-Validation

Leave out a single observation y_i and predict by all other observations y_{-i} using the ELPD:

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

(other measures are possible as well)

Important properties of LOO-CV:

- All possible N splits can be evaluated
- Can be approximated using the full model

Importance Sampling

Approximate expectations over a target distribution $f(\theta)$ using an approximating proposal distribution $g(\theta)$:

$$\mathbb{E}_f(h) = \int h(\theta) f(\theta)\, d\theta = \frac{\int h(\theta) f(\theta)\, d\theta}{\int f(\theta)\, d\theta} = \frac{\int h(\theta) r(\theta) g(\theta)\, d\theta}{\int r(\theta) g(\theta)\, d\theta}$$

Raw importance ratios:

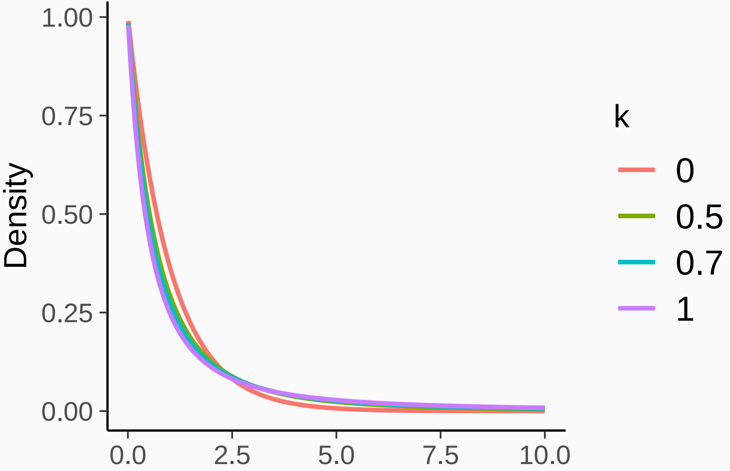
$$r(\theta) = \frac{f(\theta)}{g(\theta)}$$

Approximation via $\theta^{(s)} \sim g(\theta)$:

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

Pareto Smoothed Importance Sampling (PSIS)

Replace the largest importance ratios with quantiles of the generalized Pareto distribution (GPD)



The $\hat{k}$-Diagnostic

The number of existing moments of the GPD is

$$\#\text{moments} = \begin{cases} \text{if } k > 0: \ \text{floor}\left(\frac{1}{k}\right) \\ \text{else: } \ \infty \end{cases}$$

Relevant thresholds:

- $k < 0.5$: Finite variance and fast convergence rate
- $0.5 \leq k \leq 0.7$: Convergence rate is still ok
- $k > 0.7$: Preasymptotic behavior gets in your way
- $k > 1$: All is lost

PSIS-LOO-CV

Compute the raw LOO importance ratios:

$$r_i^{(s)} = \frac{f_i(\theta^{(s)})}{g(\theta^{(s)})} \propto \frac{1}{p(y_i \mid \theta^{(s)})}$$

Obtain smoothed importance weights $w_i^{(s)}$ via PSIS

Approximate the *i*th posterior preditive density (PPD):

$$p(y_i \mid y_{-i}) \approx \frac{\sum_{s=1}^{S} w_i^{(s)} \, p(y_i \mid \theta^{(s)})}{\sum_{s=1}^{S} w_i^{(s)}}$$

Sum over the log pointwise contributions:

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

Icecream Sold: Compute Log-Likelihood Values

Compute log-likelihoods values after model fitting (example shown for linear regression):

```
...
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)

```
## 
## Computed from 4000 by 72 log-likelihood matrix
## 
##          Estimate   SE
## elpd_loo   -490.6  6.4
## p_loo         2.6  0.5
## looic       981.2 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)

```
##
## Computed from 4000 by 72 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -411.3  6.4
## p_loo         8.3  1.5
## looic       822.6 12.8
## ------
## 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)

```
##
## Computed from 4000 by 72 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -370.3  5.8
## p_loo         9.0  1.4
## 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)     71    98.6%   931
##  (0.5, 0.7]   (ok)        1     1.4%   2575
##    (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).
## See help('pareto-k-diagnostic') for details.
```

Comparing Models via Approximate LOO-CV

```
##         elpd_diff se_diff
## model3    0.0       0.0
## model2  -40.9       7.6
## model1 -120.3       9.5
```

A Look into the Future

Improve speed of Bayesian Inference:

- Improved sampling algorithms
- Use GPUs/TPUs for matrix algebra
- Use of within-chain parallelization
- Use asymptotically biased approximations?

Improve feasibility of simulation-based Bayesian inference:

- Move away from Approximate Bayesian Computation (ABC)
- Develop fast auto-differentiable (O/P)DE solver
- Leverage the power of normalizing flows

Amortize Bayesian inference over data sets:

- Train the model once after which inference is almost instant

Appendix

Bayes Factors

Used to compare two models M_1 and M_2:

$$BF_{12} = \frac{p(y|M_1)}{p(y|M_2)}$$

- where $p(y|M_1)$ denotes the marginal likelihood of M_1

Closely related to the posterior Odds:

$$\frac{p(M_1|y)}{p(M_2|y)} = \frac{p(M_1)}{p(M_2)} BF_{12}$$

- $p(M_1)$ and $p(M_2)$ are the prior probabilities of the models M_1 and M_2
- Usually $p(M_1) = p(M_2) = 1/2$

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

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
  // unknown 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.