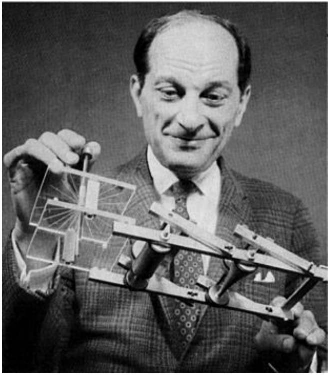
GLMM - Bayesian Inference with Stan

Introduction to Stan

L. Egidi - DEAMS, Units (legidi@units.it)

PhD Course in Statistics - XXXV cycle

Origins



Stanislaw Ulam (1909-1984): Manhattan project, H-Bomb experiments in Los Alamos, MCMC father jointly with John von Neumann.

Indice

What is Stan?

- Probabilistic programming language and inference algorithms.
- Stan program
  - declares data and (constrained) parameter variables
  - defines log posterior (or penalized likelihood)
- Stan inference
  - MCMC for full Bayes
  - Variational Bayes for approximate Bayes
  - Optimization for (penalized) MLE
- Stan ecosystem
  - lang, math library (C++)
  - interfaces and tools (R, Python, Julia, many more)
  - documentation (example model repo, ▸ user guide & reference manual, ▸ case studies, R package vignettes)
  - online community (▸ Stan Forums on Discourse)

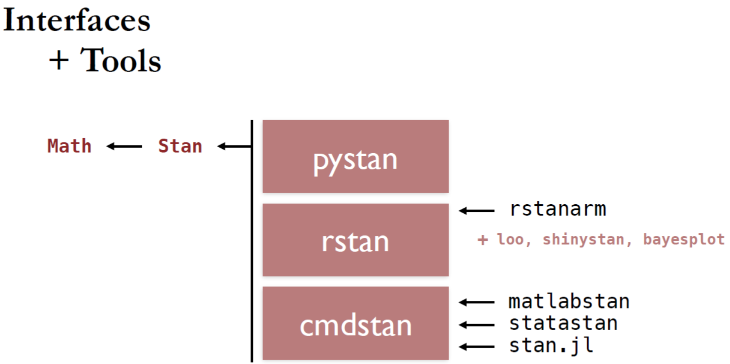
Why Stan?

- Fit rich Bayesian statistical models. Close to the *big data* philosophy.
- Efficiency
    - Hamiltonian Monte Carlo + NUTS
    - Compiled to C++
- Flexible domain specific language
- "Freedom-respecting, open-source"
    - doc & written materials
    - interacting community
    - continuous development
- Interaction with some other R packages designed to explore the Stan output.

Who is using Stan?

- Biological & physical sciences: clinical trials, epidemiology, genomics, population ecology, entomology, ophthalmology, neurology, agriculture, fisheries, cancer biology, astrophysics & cosmology, molecular biology, oceanography, climatology.
- Social sciences: population dynamics. psycholinguistics, social networks, political science, human development, economics.
- Many more: sports analytics, public health, publishing, finance, pharma, actuarial, recommender systems, educational testing, materials engineering.

Interfaces


Interfaces
+ Tools
Math
Stan
pystan
rstan
cmdstan
rstanarm
+ loo, shinystan, bayesplot
matlabstan
statastan
stan.jl

Indice

Improving MCMC performance

With Stan, we aim to provide an MCMC implementation that works robustly for as many target distributions as possible

- Gibbs, RW Metropoilis can be very inefficient, hard to diagnose.
- To explore complicated high-dimensional spaces we need to leverage what we know about the geometry of the typical set.
- For such a reason, Stan enjoys **Hamiltonian Monte Carlo**.

The Stan users may use, analyze and interpret HMC outputs as they were *standard* MCMC outputs.

Indice

Before starting

What is a Bayesian model?

- Building a Bayesian model forces us to build a model for how the data is generated
- We often think of this as specifying a prior and a likelihood, as if these are two separate things
- They are not!

Generative models

The philosophy behind Stan is to think generatively.

The model is expressed as a joint probability distribution of observed and unobserved variables, which may be decomposed as follows:

$$p(y, \theta) = p(y|\theta)\pi(\theta) \qquad (1)$$

The posterior of interest is then proportional to the joint distribution (1):

$$p(\theta|y) \propto p(y|\theta)\pi(\theta) \qquad (2)$$

Generative models

A Bayesian modeller commits to to an a priori joint distribution:

$$p(y,\theta) = \underbrace{p(y|\theta)\pi(\theta)}_{Likelihood \times Prior} = \underbrace{\pi(\theta|y)p(y)}_{Posterior \times Marginal\ \ Likelihood} \tag{3}$$

Generative models and vague priors

What is the problem with *vague/diffuse* priors?

- If we use an improper prior, then we do not specify a joint model for our data and parameters.
- More importantly, we do not specify a data generating mechanism $p(y)$.
- By construction, these priors do not regularize inferences, which is quite often a bad idea
- Proper but diffuse is better than .improper but is still often problematic.

Generative models

- If we disallow improper priors, then Bayesian modeling is generative.
- In particular, we have a simple way to simulate from $p(y)$:

$$\begin{array}{c} \theta^\star \sim \pi(\theta) \\ \downarrow \\ y^\star \sim p(y|\theta^\star) \end{array} \quad \Longleftrightarrow \quad y^\star \sim p(y)$$

Stan computations

Stan works in logarithmic terms: all the computations are actually done on log-scale. So, for the posterior we have.

$$\log(\pi(\theta|y)) = \log(\pi(\theta)) + \log(p(y|\theta)) + \text{constant} \tag{4}$$

Products become sums of logs:

$$p(y|\theta) = \prod_{i=1}^{n} p(y_i|\theta) \rightarrow \log(p(y|\theta)) = \sum_{i=1}^{n} \log(p(y_i|\theta)).$$

Starting point

We are now going to write a Stan program together:

- Open a new empty file in RStudio
- Save it as `linear_regression.stan`

Blocks strategy

Stan programs are organized into blocks:

- data block: declare data types, sizes, and constraints. Read from data source and constraints validated. Evaluated: once.
- parameters block: declare parameter types, sizes, and constraints. Evaluated: every log prob evaluation.
- transformed parameters block: declare those parameters transformed from the original ones declared in the parameters block. Evaluated: every log prob evaluation.
- model block: statements defining the posterior density in log scale. Evaluated: every log prob evaluation.
- generated quantities: declare and define derived variables. (P)RNGs, predictions, event probabilities, decision making. Constraints validated. Evaluated: once per draw.

Data block

```
data {
  // Dimensions
  int<lower=1> N;
  int<lower=1> K;

  // Variables
  matrix[N, K] X;
  vector[N] y;
}
```

```
// single line comment
/* multiple lines of
comments */
```

Parameters' block

```
parameters {
  real alpha;
  vector[K] beta;
  real<lower=0> sigma;
}
```

constraints *required* in **parameters** block

Model block

```
model {
  // priors  (flat, uniform, if omitted)
  sigma ~ exponential(1);
  alpha ~ normal(0, 10);
  for (k in 1:K) beta[k] ~ normal(0, 5);

  for (n in 1:N) {
    y[n] ~ normal(X[n, ] * beta + alpha, sigma);
  }
}
```

Why is the default automatically uniform?

- $\pi(\theta) \propto 1$ (0 on log scale)
- Nothing added to log prob

Generated quantities block

```
generated quantities {
  vector[N] y_rep;
  for (n in 1:N) {
    real y_hat = X[n,] * beta + alpha;  // local/temp
    y_rep[n] = normal_rng(y_hat, sigma);
  }
}
```

Complete Stan model

```
data {
  int<lower=1>  N;
  int<lower=1>  K;
  matrix[N, K] X;
  vector[N] y;
}
parameters {
  real alpha;
  vector[K] beta;
  real<lower=0> sigma;
}
model {
  sigma ~ exponential(1);
  alpha ~ normal(0, 10);
  for (k in 1:K) beta[k] ~ normal(0, 5);

  for (n in 1:N)
    y[n] ~ normal(alpha + X[n, ] * beta, sigma);
}
generated quantities {
  vector[N] y_rep;
  for (n in 1:N)
    y_rep[n] = normal_rng(alpha + X[n,] * beta, sigma);
}
```

Observed variables

Unobserved variables

$$\log \pi(\theta)$$

$$+$$

$$\log p(y \mid \theta)$$

Simulate from generative model

Launching the Stan model from R

Now we may launch the Stan program directly in R:

```
library(rstan)

# passing the data (already stored)
data <- list(N=N, K=K, X=X, y=y)

# fitting the model
fit1 <- stan(
  file = 'linear_regression.stan',
  data = data,
  iter = 2000,
  chains = 4)

# extracting the estimates
sims <- extract(fit1)
```

First example: 8 schools

This example studied coaching effects from eight schools.
We denote with y_{ij} the result of the i-th test in the j-th school. We assume the following model:

$$y_{ij} \sim \mathcal{N}(\theta_j, \sigma_y^2)$$
$$\theta_j \sim \mathcal{N}(\mu, \tau^2)$$

Do some schools perform better/worse according to these coaching effects?
Here is the data, already aggregated by schools:

```
schools_dat <- list(J = 8,
                    y = c(28,  8, -3,  7, -1,  1, 18, 12),
                    sigma = c(15, 10, 16, 11,  9, 11, 10, 18))
```

First Stan model: 8 schools

```
// saved as 8schools.stan
data {
  int<lower=0> J;          // number of schools
  real y[J];               // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}
parameters {
  real mu;               // population treatment effect
  real<lower=0> tau;     // standard deviation in treatment effects
  vector[J] eta;         // unscaled deviation from mu by school
}
transformed parameters {
  vector[J] theta = mu + tau * eta;    // school treatment effects
}
model {
  eta ~ normal(0,1);           // prior
  y ~ normal(theta, sigma);    //likelihood
}
```

First example: 8 schools

To fit the model and visualize the estimates, it is sufficient to type in R the following commands (with 2000 iterations and 4 chains as a default):

```
fit_8schools <- stan(file = '8schools.stan', data = schools_dat)
print(fit_8schools, pars=c("mu", "tau", "theta"))
```

```
          mean   sd   2.5%  25%   50%   75% 97.5% n_eff Rhat
mu        7.89 5.04  -2.31 4.74  7.92 11.05 18.05  2352    1
tau       6.70 5.71   0.24 2.61  5.43  9.19 21.16  1480    1
theta[1] 11.36 8.23  -2.25 6.18 10.29 15.46 31.15  3161    1
theta[2]  7.89 6.21  -4.43 3.96  7.83 11.78 20.47  4923    1
theta[3]  6.05 7.59 -10.81 1.92  6.56 10.81 20.25  4057    1
theta[4]  7.60 6.44  -5.36 3.74  7.63 11.73 20.57  5055    1
theta[5]  5.13 6.23  -8.45 1.35  5.60  9.26 16.37  4346    1
theta[6]  5.95 6.68  -8.21 1.99  6.30 10.21 18.21  4313    1
theta[7] 10.62 6.93  -1.58 6.12 10.14 14.53 25.63  3381    1
theta[8]  8.40 7.77  -7.18 3.84  8.26 12.63 25.78  3854    1
```

Indice

Posterior graphical analysis with bayesplot

Once we fit a model, it is to vital check it via graphical inspection. The `bayesplot` package (for any help, see the vignette) is designed to this task.

The package allows to display:

- Posterior uncertainty intervals
- Univariate marginal posterior distributions
- Bivariate plots
- Trace plots
- Posterior predictive plots

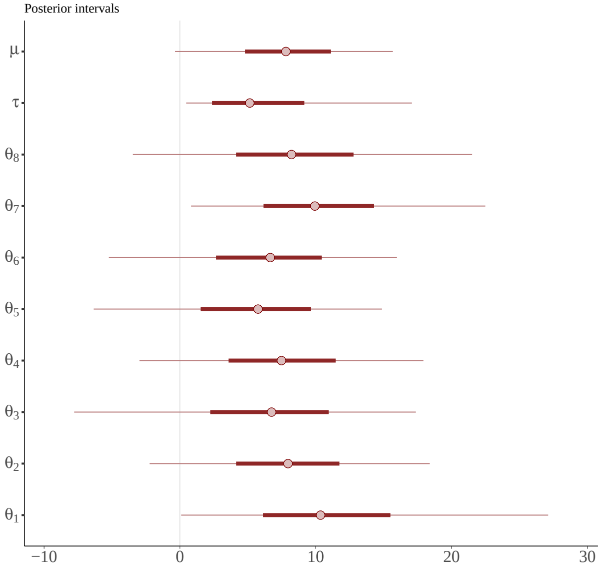
Posterior graphical analysis with bayesplot

The first step is to save the posterior. Then you have many choices:

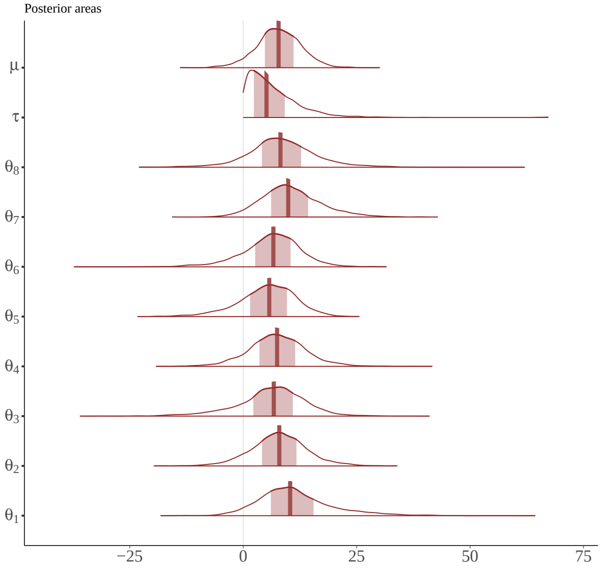
```
library(bayesplot)
posterior <- as.array(fit_8schools)

mcmc_intervals(posterior)     # posterior intervals
mcmc_areas(posterior)         # posterior areas
mcmc_dens(posterior)          # marginal posteriors
mcmc_pairs(posterior)         # bivariate plots
mcmc_trace(posterior)         # trace plots
```

With the arguments pars or regex_pars you may select the desired parameters.

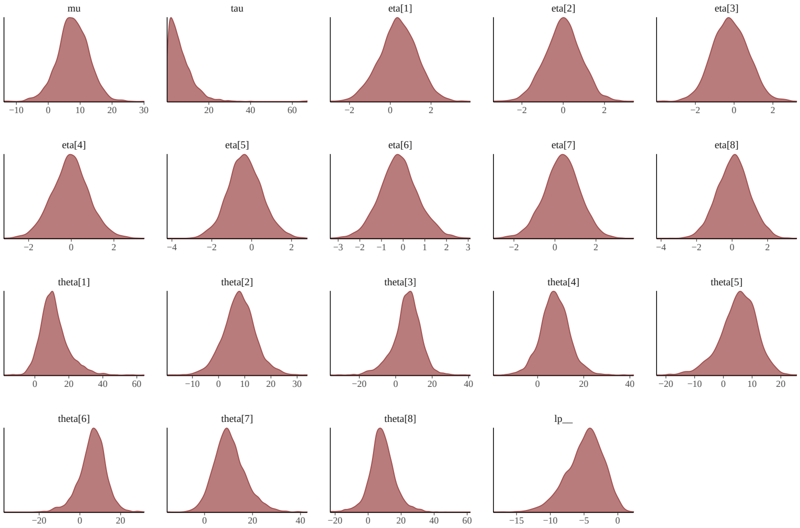
Posterior uncertainty intervals



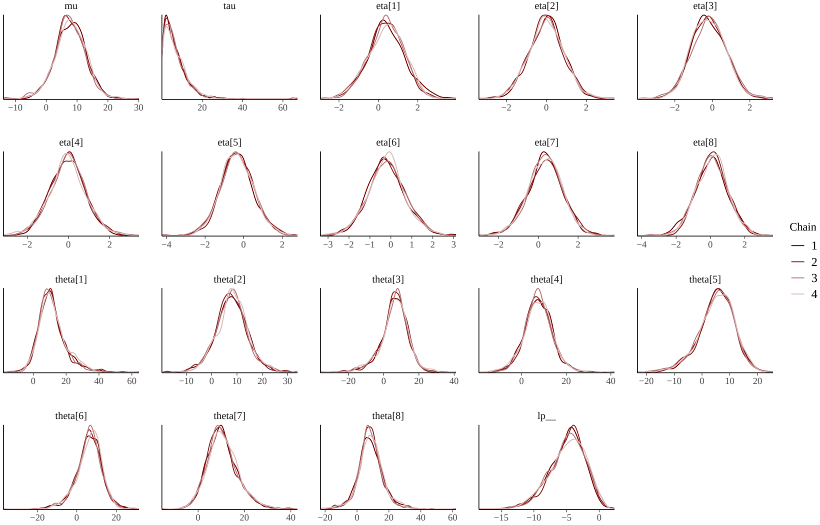
Posterior uncertainty areas



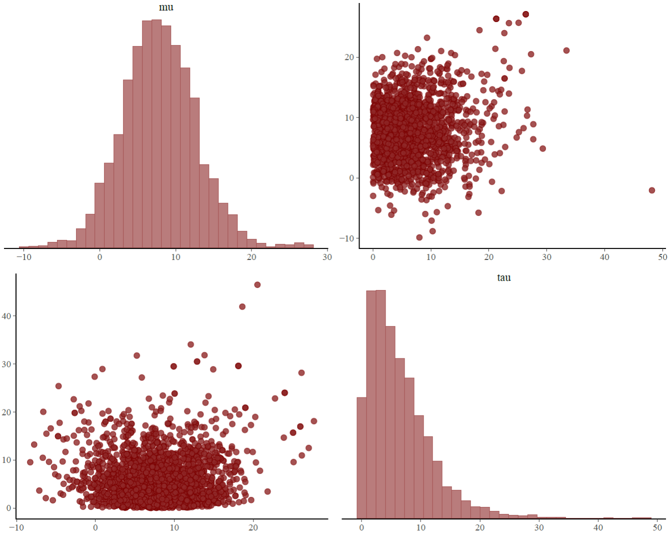
Marginal posteriors



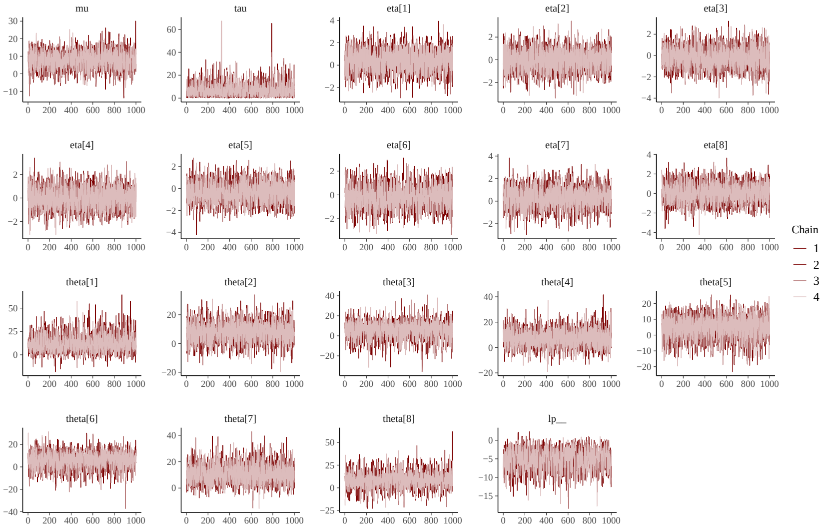
Marginal posteriors separated for each chain



Bivariate posterior plots



Trace plots for the Markov chains



Our challenge with Stan

The Stan shuttle is ready to start! We will learn to:

- write simple and more complex model in Stan: lm, glm, hierarchical models.
- analyze the posterior summaries.
- criticize the model and, eventually, change/reparametrize it.

Further reading

Further reading:

- Carpenter, B, and Gelman, A, Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., Riddell, A. (2017). Stan: A Probabilistic Programming Language, *Journal of statistical software 76(1)*. Here the pdf

Further optional reading about Hamiltonian Monte Carlo:

- Betancourt, M. (2017) A conceptual introduction to Hamiltonian Monte Carlo. Here the pdf