Math 538: Bayesian Statistics

Lecture Slides 6: Multiple Parameter Models with STAN

Department of Mathematics, California State University, Fullerton

STAN: A Brief Introduction

What is STAN?

- STAN is a probablistic programming language that implements Bayesian approaches to statistical modeling
- Written in C++
- High-performance
- Interfaces with: R, Julia, Python, shell, Matlab, and STATA
- See http://mc-stan.org/ for more information
- Many presentation slides from StanConn2023 available at https://github.com/stan-dev/stancon2023

Installing STAN through R (RSTAN)

For installing RSTAN on your operating system:

- For Windows, Mac, or Linux Users: https://github.com/stan-dev/rstan/wiki/#installing-rstan

Note that you will want to:

1. Update your operating system
2. Using R version 4.2.0 or later is strongly recommended
3. RStudio version 1.4x or later
4. You will need an updated C++ compiler on your computer (I use XCode)

Configuring C Toolchain

- Mac https://github.com/stan-dev/rstan/wiki/Configuring-C---Toolchain-for-Mac
- Windows https://github.com/stan-dev/rstan/wiki/Configuring-C---Toolchain-for-Windows
- Linux https://github.com/stan-dev/rstan/wiki/Configuring-C-Toolchain-for-Linux

```
# install macrotools
install.packages("remotes")
remotes::install_github("coatless-mac/macrtools")

# installs: Xcode CLI, gfortran,R Development
# binaries will ask for your computer password
macrtools::macos_rtools_install()
```

```
Congratulations!
Xcode CLI, Gfortran, and R developer binaries have been installed
  successfully.
```

Optimize and then install rstan

```
# Once the toolchain is installed you can then
# enable some compiler optimizations to improve
# the estimation speed of the model:
dotR <- file.path(Sys.getenv("HOME"), ".R")
if (!file.exists(dotR)) dir.create(dotR)
M <- file.path(dotR, "Makevars")
if (!file.exists(M)) file.create(M)
arch <- ifelse(R.version$arch == "aarch64", "arm64",
    "x86_64")
cat(paste("\nCXX14FLAGS += -O3 -mtune=native -arch",
    arch, "-ftemplate-depth-256"), file = M, sep = "\n",
    append = FALSE)
#----------------------------------
#-----Installing rstan -----------
remove.packages("rstan")
if (file.exists(".RData")) file.remove(".RData")
Sys.setenv(MAKEFLAGS = "-j4")  # four cores used
install.packages(c("Rcpp", "RcppEigen", "RcppParallel",
    "StanHeaders"), type = "source")
install.packages("rstan", type = "source")
```

```r
# Run this example to make sure your rstan works
example(stan_model, package = "rstan", run.dontrun = TRUE)
```

```
tn_md> stancode <- 'data {real y_mean;} parameters {real y;}
  model {y ~ normal(y_mean,1);}'

stn_md> mod <- stan_model(model_code = stancode, verbose = TRUE)

TRANSLATING MODEL '' FROM Stan CODE TO C++ CODE NOW.

stn_md> fit <- sampling(mod, data = list(y_mean = 0))

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 3e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition
  would take 0.03 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
```

```r
# Now you can load

library("rstan")

# If you are running on a local multi-core
# processor do
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
```

Bioassay Example Using rstan

Bioassay Example Using rstan

- Recall the Bioassay Problem p.74 of BDA

Dose (logscale)	# animals	# deaths
x_i	n_i	y_i
–0.86	5	0
–0.30	5	1
–0.05	5	3
0.73	5	5

- A standard model for these data are given by:

$$\begin{aligned} y_i|\theta_i &\sim Bin(n_i, \theta_i) \text{ for } i = 1, ..., K \\ \text{with} \quad & logit(\theta_i) = \alpha + \beta x_i \\ p(\alpha, \beta) &\propto 1 \end{aligned}$$

In RStudio open a stan file and write:

```
data {
  int<lower=0> J;          // number of batches
  int<lower=0> n[J]; // number in each batch
  int<lower=0> y[J]; // number of deaths in each batch
  vector[J] x;         // log-dose of each batch
}
parameters {
  real alpha;      // intercept
  real beta;      // coeffient of dose effect
}
transformed parameters {
    vector[J] theta = inv_logit(alpha+beta*x);  // death rate by dosage
}
model {
  y ~ binomial(n,theta); // log-likelihood
  //Note a uniform prior is assumed when prior not specified
}
```

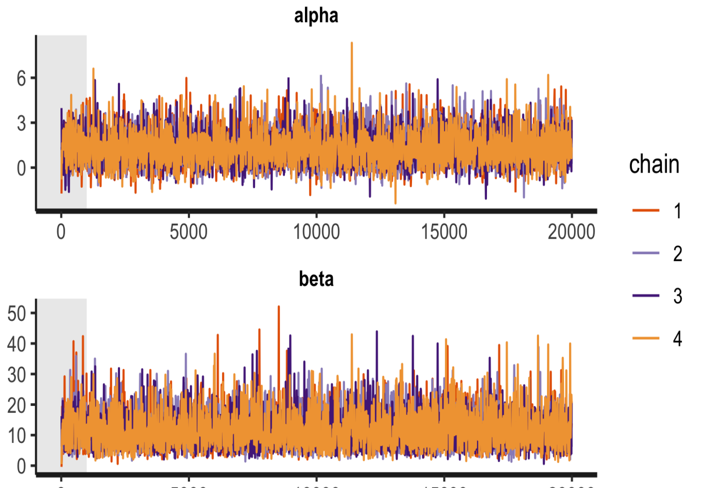
```
#create data
bioassay_dat <- list(
  J = 4,
  n = c(5, 5, 5, 5),
  y = c(0, 1, 3, 5),
  x = c(-0.86, -0.30, -0.05, 0.73))

fit1 <- stan(
  file = "bioassay.stan", # Stan program
  data = bioassay_dat,    # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 20000,           # total number of iterations per chain
  cores = 2,              # number of cores
  refresh = 1000,         # show progress every 'refresh' iterations
  thin = 10               # number of thinning
)
```

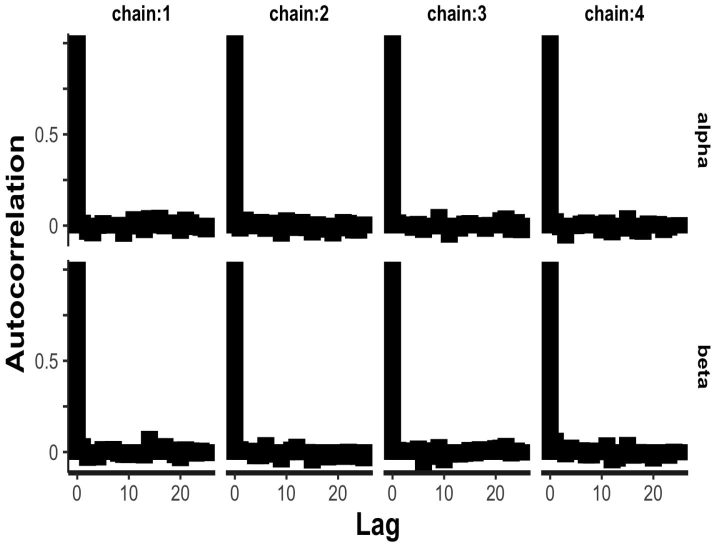
```
Chain 4: Iteration: 20000 / 20000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.072 seconds (Warm-up)
Chain 4:                1.184 seconds (Sampling)
```

Some useful automatic plotting functions in rstan

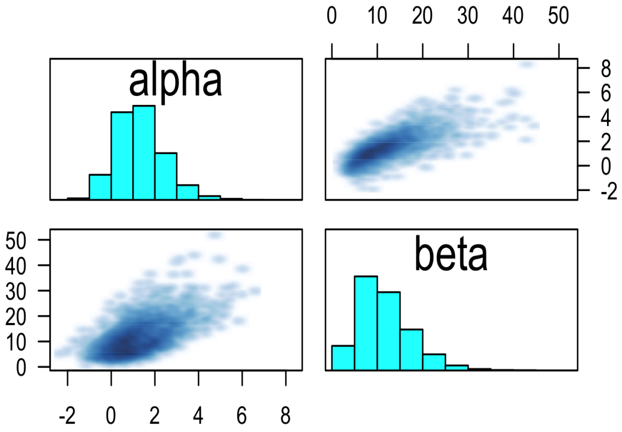
```
stan_trace(fit1, pars = c("alpha", "beta"), inc_warmup = TRUE,
    nrow = 2)
```



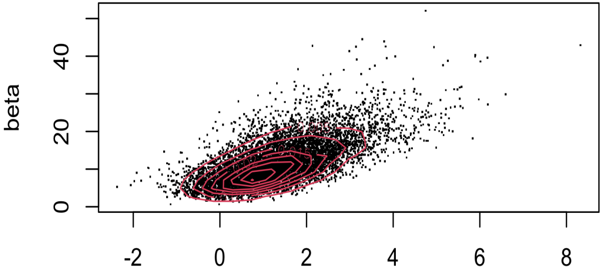
```r
stan_ac(fit1, pars = c("alpha", "beta"), inc_warmup = FALSE,
    nrow = 2, separate_chains = TRUE, lags = 25)
```



```
pairs(fit1, pars = c("alpha", "beta"), las = 1)
```



```r
fit_ss <- extract(fit1)
names(fit_ss)
# [1] 'alpha' 'beta' 'theta' 'lp__'
biv_kde <- MASS::kde2d(fit_ss$alpha, fit_ss$beta)
plot(fit_ss$alpha, fit_ss$beta, pch = ".", xlab = "alpha",
    ylab = "beta")
contour(biv_kde, col = 2, add = T)
```



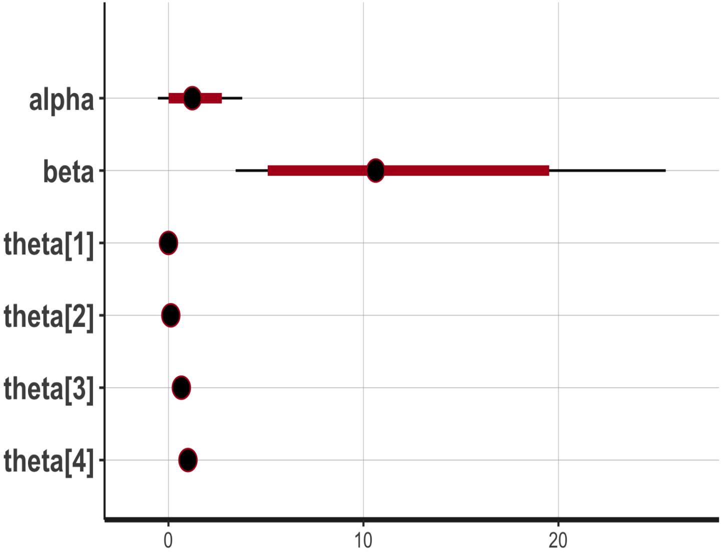
```r
print(fit1, pars = c("alpha", "beta", "theta"), probs = c(0.025,
    0.5, 0.975))
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=20000; warmup=1000; thin=10;
post-warmup draws per chain=1900, total post-warmup draws=7600.

          mean se_mean   sd  2.5%   50% 97.5% n_eff Rhat
alpha     1.32    0.01 1.10 -0.54  1.23  3.78  7448    1
beta     11.66    0.07 5.87  3.45 10.64 25.51  7139    1
theta[1]  0.01    0.00 0.02  0.00  0.00  0.07  7443    1
theta[2]  0.15    0.00 0.13  0.01  0.12  0.47  6931    1
theta[3]  0.65    0.00 0.18  0.29  0.66  0.94  7569    1
theta[4]  0.99    0.00 0.03  0.92  1.00  1.00  7468    1

Samples were drawn using NUTS(diag_e) at Sun Oct 29 22:50:20 2023.
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).
```

```
plot(fit1)
```

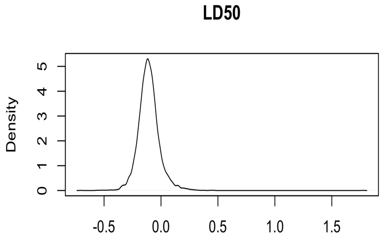


Inference for LD50

Recall: LD50: $E\left(\frac{y_i}{n_i}\right) = logit^{-1}(\alpha + \beta x_i) = 0.5 \Rightarrow$ occurs when $x_i = -\alpha/\beta$

```
######## Posterior of LD50
LD50 = -fit_ss$alpha/fit_ss$beta
quantile(LD50, probs = c(0.25, 0.5, 0.975))
plot(density(LD50), type = "l", main = "LD50")
```

```
        25%          50%        97.5% 
-0.16217623 -0.11249260  0.09887022 
```



Bioassay Example Using brms

brms: Bayesian Regression Models Using Stan

The brms package provides an interface to fit Bayesian generalized (non)linear multivariate multilevel models using Stan.

```
install.packages("brms")
library("brms")

bioassay_dat = data.frame(n = c(5, 5, 5, 5), y = c(0,
    1, 3, 5), x = c(-0.86, -0.3, -0.05, 0.73))

fit2 <- brm(y | trials(n) ~ x, data = bioassay_dat,
    family = binomial())
```

```
Chain 4:  Elapsed Time: 0.091 seconds (Warm-up)
Chain 4:                0.08 seconds (Sampling)
Chain 4:                0.171 seconds (Total)
```

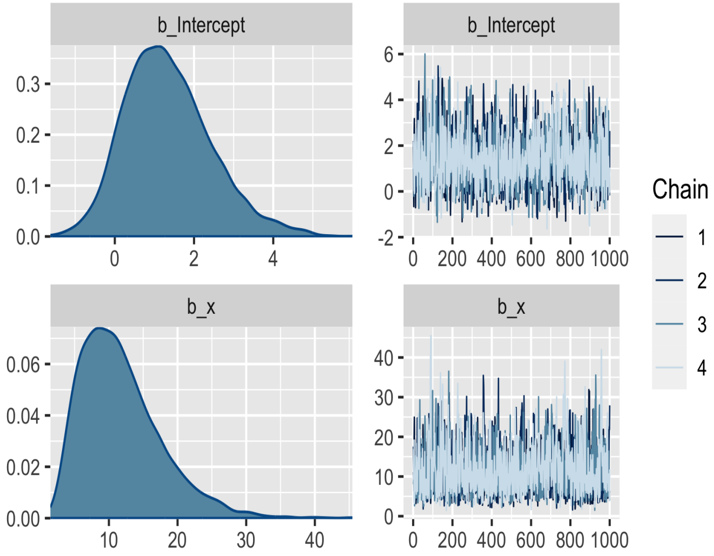
```
summary(fit2)
```

```
Family: binomial
  Links: mu = logit
Formula: y | trials(n) ~ x
   Data: bioassay_dat (Number of observations: 4)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
Intercept     1.35      1.09    -0.48     3.82 1.00     1606
x            11.75      5.87     3.54    25.82 1.00     1414
          Tail_ESS
Intercept     1868
x             1378

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the
potential scale reduction factor on split chains
(at convergence, Rhat = 1).
```

```r
plot(fit2)
```


b_Intercept
0.3
0.2
0.1
0.0
0
2
4
b_Intercept
6
4
2
0
-2
0
200
400
600
800
1000
b_x
0.06
0.04
0.02
0.00
10
20
30
40
b_x
40
30
20
10
0
0
200
400
600
800
1000
Chain
1
2
3
4