# USING R STAN

R User Group meeting

October 27, 2022

Fayette Klaassen, PhD

# TODAY'S TALK

- Brief introduction Bayesian statistics
- Introduction to stan and rstan
- Workflow & troubleshooting rstan
- Demonstration
- Questions

# WHAT IS BAYESIAN STATISTICS?

- Bayes theorem

$$\frac{P(D \mid \theta)P(\theta)}{P(D)} = P(\theta \mid D)$$

  - $P(D \mid \theta)$ likelihood of the data
  - $P(\theta)$ prior distribution
  - $P(D)$ marginal distribution
  - $P(\theta \mid D)$ posterior distribution

- Combining prior knowledge about $\theta$ to learn the conditional probability of $\theta$ given D

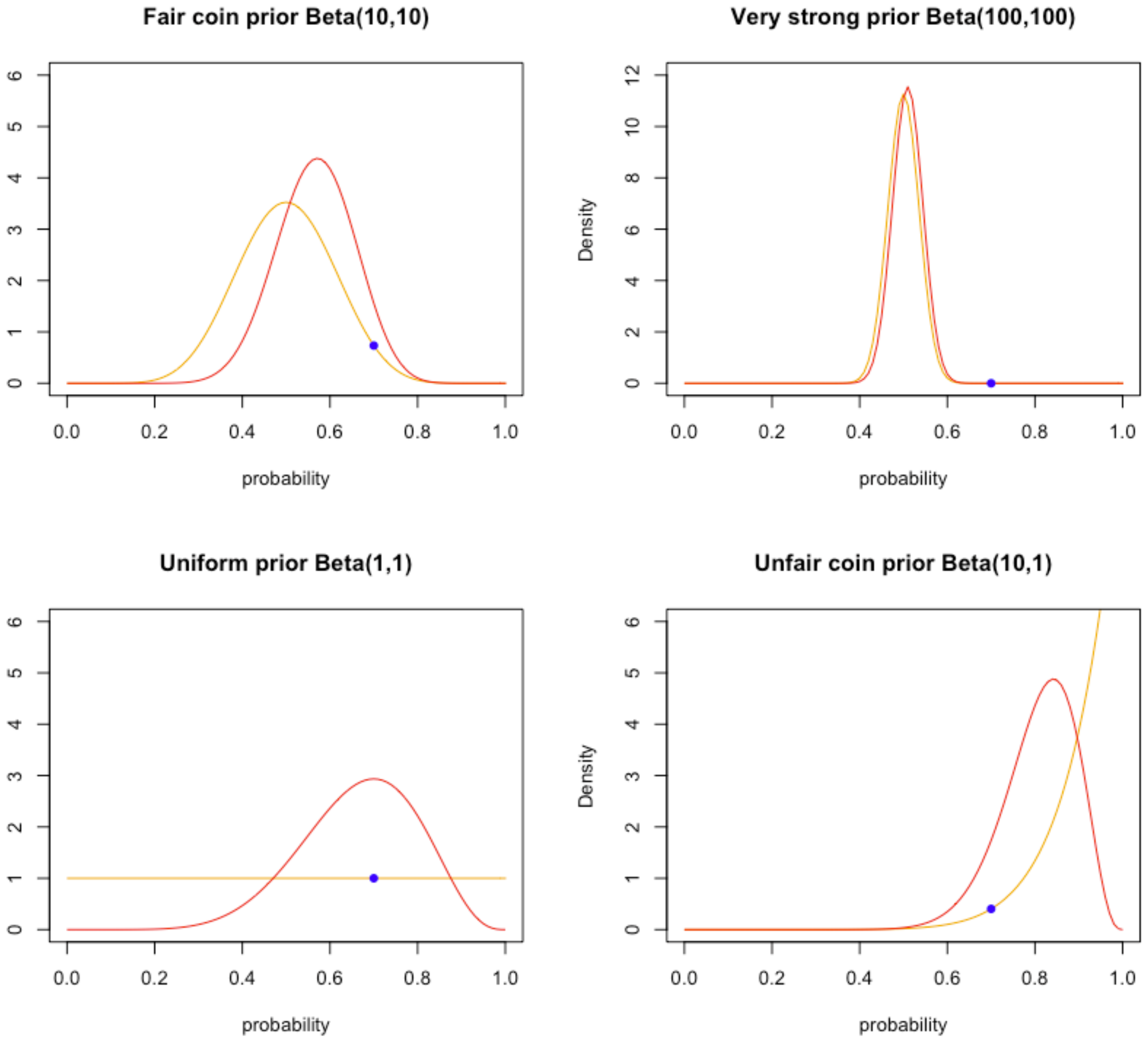
# WHY BAYESIAN STATISTICS?

- Conditional probability (parameters given data)
  - Instead of frequentist probability of the data given a parameter
- Include prior information
  - Accumulate evidence across research
- Facilitate fitting more complex models (decrease the complexity of a model)

# EXAMPLE

- Flipping a coin 10 times
  - 7 heads
- 4 prior distributions
  - Fair coin
  - Very certain fair coin
  - Unknown coin
  - Unfair coin
- Data and prior combined

To posterior



# BAYESIAN ESTIMATION

- Posterior distribution often not closed form solution
- Iterative sampling procedure
  - MCMC
- Determine the posterior probability for a set of prior values
  - Choosing priors + initial values wisely!

| h | $\theta_1$ | $\theta_2$ |
|---|---|---|
| 0 | - | $\theta_2^{(0)}$ |
| 1 | $\theta_1^{(1)} \mid \theta_2^{(0)}$ | $\theta_2^{(1)} \mid \theta_1^{(1)}$ |
| 2 | $\theta_1^{(2)} \mid \theta_2^{(1)}$ | $\theta_2^{(2)} \mid \theta_1^{(2)}$ |
| 3 | $\theta_1^{(3)} \mid \theta_2^{(2)}$ | $\theta_2^{(3)} \mid \theta_1^{(3)}$ |
| ... | | |
| H | $\theta_1^{(H)} \mid \theta_2^{(H-1)}$ | $\theta_2^{(H)} \mid \theta_1^{(H)}$ |

# METHODS FOR BAYESIAN ESTIMATION

| | |
|---|---|
| Direct specification of posterior | only for simple models |
| Gibbs sampling (MCMC algorithm) | implemented in JA**G**S; BU**G**S; requires a closed form of the (conditional) posterior |
| Metropolis Hastings algorithm | time intensive; explore posterior using proposal distribution |
| Hamiltonian Monte Carlo | uses gradient of un-normalized log-posterior; more efficient and complete sampling |
| No U-Turn Sampler | auto-tuning of leapfrog steps |
| Optimization routines (e.g. BFGS in stan) | maximizing the objective function; no uncertainty intervals |

# WHAT IS RSTAN

- Stan
  - C++ language for Bayesian modeling
  - Utilizes  Hamiltonian Monte Carlo (HMC) and No U-Turn Sampler (NUTS) routines
    - Hoffman & Gelman (2014), and stan core development team

- Rstan
  - R interface (package) for stan
  - Requires installation of a C++ complier (gcc/gnu)
  - Requires several R packages
  - Installation can be tricky – follow instructions
    - https://cran.r-project.org/web/packages/rstan/vignettes/rstan.html

# WHY USE RSTAN

- Get the speed and efficiency from the HMC and NUTS in stan
  - Auto-differentiation for HMC, auto-tuning for NUTS
- Ability to specify and fit complex models
- Get the ease of writing the model in R / a .stan file
  - Auto-rendering of C++/stan code

# WORKFLOW USING STAN

1. Formulate the statistical model
   - Write your model in a `.stan` file
2. Sample from the posterior distribution
   - Run the `stan()` function in R
3. Assess (non)—convergence
   - Render plots and diagnostics
4. Draw inference

# BASIC STAN PROGRAM

```
functions { //optional
}

data { //required
// define the format of your data
}

transformed data { //optional
// any manipulations to the data
}

parameters { //required
// define the parameters used in the model
}

transformed parameters { //optional
// define any transformations of the parameters
// define any transformed parameters you want to sample
}

model { //required
// specify the model (likelihood, priors) using (transformed) parameters
}

generated quantities{ //optional
// any secondary output, that does not rely on the model
}
```

# EXAMPLE: STATISTICAL MODEL

Simple linear regression model
1 predictor (x), 1 outcome (y)

$$y = b_0 + b_1 x + \epsilon$$

$$\hat{y} = b_0 + b_1 x$$

$$\epsilon \sim N(0, \sigma^2)$$

$$y \sim N(\hat{y}, \sigma^2)$$

# EXAMPLE: STAN FILE

```
data {
int N;
vector[N] y;
vector[N] x;
}
parameters { //required
real b0;
real b1;
real sigma;
}
transformed parameters { //optional
vector[N] theta;
theta = b0 + b1*x;
}
model { //required
y ~ normal(theta, sigma);
}
```

# NOTES ON STAN CODE

- Data: specify all the data you are feeding in to the model
- Parameters: specify all model parameters
- Transformed parameters: specify all functions/calculations of core parameters that you want sampled as well
- Model: the likelihood and any priors

```
data {
int<lower=0> N;
vector[N] y;
vector[N] x;
}
parameters { //required
real b0;
real b1;
real sigma;
}
transformed parameters { //optional
vector[N] theta;
theta = b0 + b1*x;
}
model { //required
y ~ normal(theta, sigma);
}
```

# NOTES ON STAN CODE

- Specify the type of any quantity in stan
  - int, real, vector, matrix; specify limits
- For loops can be used, indexing starts at 1
- End each statement with a semicolon
- Rstan 'checks' the correctness of your code
  - Not always easy to figure out where the mistake is.

```
data {
int<lower=0> N;
vector[N] y;
vector[N] x;
}
parameters { //required
real b0;
real b1;
real sigma;
}
transformed parameters { //optional
vector[N] theta;
theta = b0 + b1*x;
}
model { //required
for(i in 1:N){
y[i] ~ normal(theta[i], sigma);
} }
```

# ALTERNATIVE FORMULATIONS

```
data {
int N;
vector[N] y;
vector[N] x;
}
parameters { //required
real b0;
real b1;
real sigma;
}
transformed parameters { //optional
vector[N] theta;
theta = b0 + b1*x;
}
model { //required
y ~ normal(theta, sigma);
}
```

```
data {

vector[30] y;
vector[30] x;
}
parameters { //required
real b0;
real b1;
real<lower=0> sigma;
}
transformed parameters { //optional
}
model { //required
for(i in 1:30){
y[i] ~ normal(b0 + b1*x[i], sigma);
}
}
```

# ALTERNATIVE FORMULATIONS

```
data {
int N;
vector[N] y;
vector[N] x;
}
parameters { //required
real b0;
real b1;
real sigma;
}
transformed parameters { //optional
vector[N] theta;
theta = b0 + b1*x;
}
model { //required
y ~ normal(theta, sigma);
}
```

```
data {
vector[30] y;
vector[30] x;
}
parameters { //required
real b0;
real b1;
real<lower=0> sigma;
}
transformed parameters { //optional
}
model { //required
vector[N] theta;
theta = b0 + b1*x;
sigma ~ cauchy(0,1);
target += normal_lpdf(y| theta, sigma);
}
```

# NOTES ON STAN CODE

- Any quantity in `parameters` is sampled
- Any quantity in `transformed paramters` is sampled from the posterior
- Log posterior value is incremented for each model specification
- By omitting priors/likelihoods for parameters, by default improper priors are used

# SAMPLING FROM THE POSTERIOR WITH STAN

```r
library(rstan)

N <- 40
x <- rnorm(N, 100, 20)
res <- rnorm(N, 0, 10)
y <- 10 + x * 5 + res
ex1data <- list("N" = N, "x" = x, "y" = y)

fit1 <- stan(
  file = "example1.stan",  # Stan program
  data = ex1data,    # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  cores = 1,              # number of cores (could use one per chain)
  verbose = TRUE,         # show progress
  refresh = 10,            # number of iterations to refresh for
  pars = "theta",
  include = FALSE
)
```

# NOTES ON RUNNING RSTAN

- Chains
  - how many MCMC chains are you using?
  - used to assess mixing and convergence
  - 3-4 general amount
- Warmup
  - how many iterations should be considered as warmup?
  - these are to 'train' the sampler
  - not used in final samples for estimation
- Iterations
  - total number of iterations
  - Warmup : iteration ratio 2:1 / 1:1 / 1:2
  - at least 1000 iterations for final samples
- Cores
  - if you have a multicore computer, you can starts each chain on an individual core
  - speeds up computing time
  - cannot split up a chain over multiple cores
- Refresh & verbose
  - display intermediate samples

```r
library(rstan)

N <- 40
x <- rnorm(N, 100, 20)
res <- rnorm(N, 0, 10)
y <- 10 + x * 5 + res
ex1data <- list("N" = N, "x" = x, "y" = y)

fit1 <- stan(
  file = "example1.stan",  # Stan program
  data = ex1data,    # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  cores = 1,              # number of cores (could use one per chain)
  verbose = TRUE,         # show progress
  refresh = 10,            # number of iterations to refresh for
  pars = "theta",
  include = FALSE
)
```

# NOTES ON RUNNING RSTAN

- Seed
  - Set the seed that stan uses (set.seed in R does not work)
- Init
  - Provide initial values for the sampler
- Pars
  - Indicate for which parameters you want the sampled output

Advanced options available in `control` argument, with for example the `adapt_delta` and `max_treedepth` (NUTS/HMC settings)

```r
library(rstan)

N <- 40
x <- rnorm(N, 100, 20)
res <- rnorm(N, 0, 10)
y <- 10 + x * 5 + res
ex1data <- list("N" = N, "x" = x, "y" = y)

fit1 <- stan(
  file = "example1.stan",  # Stan program
  data = ex1data,     # named list of data
  chains = 4,              # number of Markov chains
  warmup = 1000,           # number of warmup iterations per chain
  iter = 2000,             # total number of iterations per chain
  cores = 1,               # number of cores (could use one per chain)
  verbose = TRUE,          # show progress
  refresh = 10,             # number of iterations to refresh for
  pars = "theta",
  include = FALSE
)
```

# NOTES ON RUNNING RSTAN

- Using the `stan()` function uses the sampler
  - Alternative: `optimizing()`
- Directly using the stan model file here
  - Also option to first compile model code and then sample

```r
library(rstan)

N <- 40
x <- rnorm(N, 100, 20)
res <- rnorm(N, 0, 10)
y <- 10 + x * 5 + res
ex1data <- list("N" = N, "x" = x, "y" = y)

fit1 <- stan(
  file = "example1.stan",  # Stan program
  data = ex1data,    # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  cores = 1,              # number of cores (could use one per chain)
  verbose = TRUE,         # show progress
  refresh = 10,            # number of iterations to refresh for
  pars = "theta",
  include = FALSE
)
```

```r
stanmod1 <- rstan::stan_model(file = "file.stan")
stanmod2 <- rstan::stan_model(model_code = "data{real y;} parameters{real mu;} model{mu~normal(y,1);}")

samps1 <- rstan::sampling(stanmod1, data = list(y = 0))
```

# ASSESSING CONVERGENCE

- Rstan has a lot of inbuilt functions to assess the convergence of your posterior samples
- Sampled output gives rhat and effective sample size
- Plot additional output using:
  - `plot()`
  - `traceplot()`
  - `pairs()`
  - `stan_diag( [[multiple options]] )`
  - `stan_rhat()`
  - `stan_par(par = "par")`

# ESTIMATION

```
> fit1
Inference for Stan model: example1.
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%     25%     50%     75%   97.5% n_eff Rhat
b0      -3.70    0.24 9.13  -21.58   -9.70   -3.66    2.22   14.95  1424    1
b1       5.15    0.00 0.09    4.97    5.09    5.15    5.21    5.34  1420    1
sigma   12.59    0.04 1.48   10.13   11.56   12.43   13.50   15.90  1594    1
lp__  -120.63    0.04 1.33 -124.07 -121.19 -120.28 -119.68 -119.15  1139    1

Samples were drawn using NUTS(diag_e) at Fri Sep 16 10:27:34 2022.
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).
```

# TROUBLESHOOTING

- Divergent transitions?
  - Inspect the posterior diagnostic figures, assess whether maybe there is 1 or multiple problematic chains
  - Inspect the per chain samples
  - Adjust control parameter – adapt delta, maxtreedepth, objective tolerance…
- Bulk / tail ESS too low? / Rhat above 1 / No mixing of chains?
  - More iterations, using thinning
  - Changing seed / starting values may sometimes do the trick
  - Reparameterization of model (centering, transformations, etc) to reduce strong autocorrelations in sampler

- Inspect the posterior to see if results are making sense!!! All transformation done correctly?

# TROUBLESHOOTING

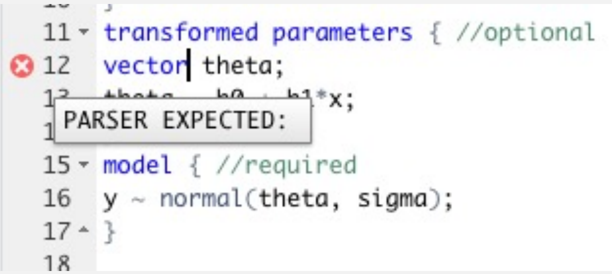
- I made changes to the model but these are not reflected in the output
  - Did you correctly reload the model? Stan makes ‘interim’ model codes, so can sometimes need twice to get updated correctly

# TROUBLESHOOTING

- Stan does not start sampling
  - There is a bug somewhere in your code!
  - Some type mismatch or incorrect use of a function
  - Go back to file, see if the rstan help helps you detect the possible source
  - Reevaluate the functions you are using: are the inputs of the correct type?
  - Use print statements and verbose = TRUE to get interim diagnostics.

```
TRANSLATING MODEL 'example1' FROM Stan CODE TO C++ CODE NOW.
SYNTAX ERROR, MESSAGE(S) FROM PARSER:
 error in 'model75164303c80_example1' at line 12, column 7
  -------------------------------------------------
    10: }
    11: transformed parameters { //optional
    12: vector theta;
              ^
    13: theta = b0 + b1*x;
  -------------------------------------------------

PARSER EXPECTED: <vector length declaration: data-only integer expression in square brackets>
Error in stanc(file = file, model_code = model_code, model_name = model_name,  :
  failed to parse Stan model 'example1' due to the above error.
```



# TROUBLESHOOTING

- Stan user guides
  - User guide, many examples and explanations
    - https://mc-stan.org/docs/2_18/stan-users-guide/index.html
  - Function references
    - Vector, unit, matrix based functions
    - Distributions
    - https://mc-stan.org/docs/2_20/functions-reference/index.html#overview
- Stack exchange // google
- Stan forums

# EXAMPLE IN PRACTICE

- See live demonstration