Session 4: Introduction to Stan

Contents

1 Main blocks of a Stan program

Suppose that we want to infer the mean height, μ, in a population of interest. To carry out this inference we need some data, so suppose that we have recorded the height, Y_i (in meters), of 10 individuals drawn from that population. Further, suppose that the normal sampling model is appropiate given the multitude of different fators that affect the height of a person, that is:

$$Y_i \sim \mathcal{N}(\mu, \sigma)$$

To complete our specification of a Bayesian model we need priors, which we choose as:

$$\mu \sim \mathcal{N}(1.5, 0.1)$$

$$\sigma \sim \Gamma(1, 1)$$

The completed Stan program implementing the above model is:

```
data {
  real Y[10]; // heights for 10 people
}

parameters {
  real mu; // mean height in population
  real<lower=0> sigma; // sd for height distribution
}

model {
  for (i in 1:10) {
    Y[i] ~ normal(mu, sigma); // likelihood
  }

  mu ~ normal(1.5, 0.1);  // prior for mu
  sigma ~ gamma(1, 1);    // prior for sigma
}
```

1.1 Writing a Stan program

You must create a new text file, making sure to save it suffixed with .stan

1.2 Data

```
data {
  real Y[10]; // heights for 10 people
}
```

The data block is used to declare all the data you will pass to Stan, wich allow it to estimate your model.

You must declare the type of data or parameters which are used in your model. If you declare a variable as one type, you cannot later convert it to another.

Examples of data types declaration:

- `real Y[10]`: array with 10 elements
- `real<lower=0, upper=1> Z`: continuous variable bounded between 0 and 1
- `int<lower=0> Z`: a discrete variable that takes integer values with a minimum value of 0
- `vector[N] Z`: a vector of continuous variables of length `N`

- `matrix[3, 3] Z`: a 3 x 3 matrix of continuous variables
- `matrix[3, 3] Z[5, 2]`: a 5 x 2 array of 3 x 3 matrices

Vectors and **matrices**: only real values. They can be used in *linear algebra* operations and they have *speed* benefits
Arrays: data of any type

##Parameters

In the parameters block we declare all the parameters that we will infer in our model.

```
parameters {
  real mu; // mean height in population
  real<lower=0> sigma; // sd for height distribution
}
```

As with the data block, a range of data types are available (vector, matrix, array...). Other data types:

- `simplex[K] Z`: a vector of K non-negative continuous variables whose sum is 1
- `corr_matrix[K] Z`: a K x K dimensional correlation matrix
- `ordered[K] Z`: a vector of K continuous ordered elements

Stan currently does not support integer-valued parameters. But it is possible to indirectly include discrete parameters by marginalising them out.

1.3 Model

The model block is used to specify the likelihood and priors for a model.

```
model {
  for (i in 1:10) {
    Y[i] ~ normal(mu, sigma); // likelihood
  }

  mu ~ normal(1.5, 0.1);  // prior for mu
  sigma ~ gamma(1, 1);    // prior for sigma
}
```

1.4 Vectorisation

```
for (i in 1:10) {
  Y[i] ~ normal(mu, sigma); // likelihood
}
```

A more efficient and compact way to write the for loop that accesses each element of Y individually: use the array Y on the left-hand side:

```
Y ~ normal(mu, sigma); // likelihood
```

1.5 Probability distributions

Stan have a range of useful probability distributions. Here are a few of the more popular distributions:

- **Discrete**: Bernoulli, binomial, Poisson, beta-binomial, negative-binomial, multinomial
- **Continuous unbounded**: normal, Student-t, Cauchy
- **Continuous bounded**: uniform, beta, log-normal, exponential, gamma, chi-squared, Weibull, Pareto
- **Multivariate continuous**: normal, Student-t

2 Execute a Stan program from R

1. The working directory must contain the script R file that we will use to call Stan and the stan file
2. Load the RStan package:

```
library(rstan)
```

3. (Optional) Set up RStan to run in parallel on multiple processors

```
options(mc.cores = parallel::detectCores())
```

4. Create some generated data in R using:

```
Y <- rnorm(10, 1.5, 0.2)
```

5. Compile and run the MCMC on your Stan program:

```
fit <- stan("stan_models/simple.stan", iter = 200, chains = 4,
  data = list=(Y = Y))
```

The R script should look something like this:

```
library(rstan)
options(mc.cores = parallel::detectCores())


# Simulate data
Y <- rnorm(10, 1.5, 0.2)

# Compile and run the MCMC on the Stan program
fit <- stan("stan_models/simple.stan", iter = 200, chains = 4,
  data = list=(Y = Y))
```

3 Interpreting the results of a Stan run

See the `Session_4.R` script.

Information available:
- Number of chains considered, and how many iterations each
- Warm-up discarded iterations
- Total post-warmup draws
- Summary statistics for the parameters, and for the log probability of the model
- Convergence diagnostics for each parameter:
- `n_eff`: number of effective samples
- `Rhat`: potential scale reduction factor on split chains (at convergence, Rhat=1)

4 Other blocks

4.1 Generated quantities

One of the main uses of the generated quantities block is to obtain the posterior predictive samples and to do posterior predictive checks of a model's fit.

```
generated quantities {
  vector[10] lSimData;
  int aMax_indicator;
  int aMin_indicator;

  // Generate posterior predictive samples
  for (i in 1:10) {
    lSimData[i] = normal_rng(mu, sigma);
  }

  // Compare with real data
  aMax_indicator = max(lSimData) > max(Y);
  aMin_indicator = min(lSimData) > min(Y);
}
```

4.2 Functions

4.3 Transformed parameters

4.4 Transformed data

4.5 Order of the blocks

Everything defined in a block is available for the following blocks

Order of the blocks and number of times that each block is executed in a typical Stan run:

- **functions**: definition - once at the beginning
- **data**: once at the beginning
- **transformed data**: once after the *data block* is executed

- **parameters**: each time the log probability is evaluated
- **transformed parameters**: each time the log probability is evaluated
- **model**: each time the log probability is evaluated
- **generated quantities**: once per sample

5 Loops and conditional statements

For loops iterate through the elements as suggested in the argument:

```
for (i in 1:10) {
  execute this;
}
```

While loops are conditionals:

```
int i = 0;
while (i < 10) {
  execute this;
  i = i + 1;
}
```

Stan allows conditional behaviour via *if* and *else* statements

```
if (i < 2) {
  execute this;
} else if (i == 2) {
  execute that;
} else {
  execute other thing;
}
```

6 References

- Stan documentation:
  - Stan user's guide
  - Stan Reference manual
  - Stan functions reference
  - Tutorials
- Stan user forum

7 Exercise - Discoveries:

The file *evaluation_discoveries.csv* contains data on the numbers of "great" inventions and scientific discoveries (Xt) in each year from 1860 to 1959. In this question you will develop a model to explain the variation in scientific inventions over time. The simplest model here is to assume that (a) one discovery is independent of all others, and (b) the rate of occurrence of discoveries is the same in all years (λ) [Lambert]. Since the data is discrete, these assumptions suggest the use a Poisson likelihood,

$$Y \sim \mathcal{P}\wr\backslash\int\int\wr\backslash(\lambda)$$

1. Open a text editor and create a file called "discoveries.stan" in your working directory. In the file create three parameter blocks:

```
data {

}

parameters {

}

model {

}
```

2. Fill in the data and parameter blocks for the above model.
3. Using a $\updownarrow\wr\}\mathcal{N}(2,1)$ prior for λ code up the model block; making sure to save your file afterwards.
4. From RStudio, load any packages necessary to use Stan.
5. Read the data (`readr::read_csv()`) then put it into a structure that can be passed to Stan.
6. Run your model using Stan, with 4 chains, each with a sample size of 1000, and a warm-up of 500 samples. Set seed=1 to allow for reproducibility of your results. Store your result in an object called "fit".
7. Diagnose whether your model has converged by printing "fit"" ".
8. For your sample what is the equivalent number of samples for an independent sampler?
9. Find the central posterior 80% credible interval for λ.
10. Draw a histogram of your posterior samples for λ.
11. Load the evaluation_discoveries.csv data and graph the data. What does this suggest about our model's assumptions?
12. Create a generated quantities block in your Stan file, and use it to sample from the posterior predictive distribution. (Hint: use the function `poisson_rng` to generate independent samples from your lambda).

A more robust sampling distribution is a negative binomial model:

$$Y_i \sim NB(\mu, \kappa)$$

where μ is the mean number of discoveries per year, and $var(Y) = \mu + \mu^2/\kappa$. Here κ measures the degree of over-dispersion of your model; specifically if κ increases then over-dispersion decreases.

13. Write a new Stan file called "discoveries_negbin.stan" that uses this new sampling model (Hint: use the Stan manual section on discrete distributions to search for the correct negative binomial function name; be careful there are two different parameterisations of this function available in Stan). Assume that we are using the following priors:

$$\mu \sim \text{⇕}\mathcal{N}(2, 1)$$

$$\kappa \sim \text{⇕}\mathcal{N}(2, 1)$$

14. Draw 1000 samples across 4 chains for your new model. Has it converged to the posterior?