Introduction to Bayesian inference using Rstan: practical 1

The aim of this practical is to provide an introduction to the syntax and structure of the Stan programming language and to demonstrate some of the efficiency gains that can be achieved by using vectorized operations.

1 *Getting started*

1.1 *Files in Rstudio*

- Open `Rstudio`
- Open a new text file:

```
File -> New File -> Text File
```

and type the following Stan programme:

```
data {
  int<lower=1> N;
  real<lower=0,upper=1> p;
}
model {
}
generated quantities {
  int x[N];
  for(n in 1:N) {
    x[n] = bernoulli_rng(p);
  }
}
```

Save the text file as `simulation.stan`.

- Open a new R script:

```
File -> New File -> R Script
```

and type the following sequence of commands, omitting the comments if you wish:

```
## Load the rstan package:
library(rstan)
## Allow rstan to exploit parallel computation:
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
## Set up some constants to pass to Stan:
constants = list(N=30, p=0.8)
## Compile and run Stan programme:
output = stan(file="simulation.stan", data=constants, iter=1,
              chains=1, algorithm="Fixed_param")
## Extract simulated data:
out = as.matrix(output)
dim(out)
head(out)
```

```
## Format as vector, removing the log posterior (i.e. "__lp"):
x = out[1, -length(out)]
## Plot data:
barplot(table(x))
```

Save the R script. Note that Rstudio will correctly add the file extension `.R`. We can run our R script by sending the commands to the R console. For example, to send one line at a time, press `Ctrl + Enter` at the end of each line in turn.

1.2 *Course R package*

Installing the course R package is straightforward. First install `drat`:

```
install.packages("drat")
```

Then

```
drat::addRepo("jr-packages")
install.packages("jrRstan")
```

This R package contains copies of the practicals, solutions and data sets for the course in addition to some helpful functions that we will use. To load the package, use

```
library("jrRstan")
```

2 *Simulating data*

In the Stan programme above, we are not attempting to carry out any inference. Instead we're using the `generated quantities` block to sample some data; in this case a random sample of size 30 from the Bernoulli distribution with probability 0.8. In other words a sequence of 30 0s and 1s where the probability of getting a 1 is 0.8. When we simply want to use Stan to simulate data in this way, notice that the (required) model block is empty. We will learn about the `stan` function in Chapter 4, but for now, we simply note that the `file` argument should be the name of the text file where the Stan programme is saved and, if the programme contains a `data` block, the `data` argument should be a `list` with a named element for each variable declared. When it is being used to simulate data, we generally set `algorithm="Fixed_param"`, `iter=1` and `chains=1`.

- What happens if you try to pass a value for `N` or `p` that violates the constraints in the `data` block? For example, what happens if you set:

```
faulty_constants = list(N=30, p=10)
```

- Add a `transformed data` block and include a print statement to check whether the constants, `N` and `p`, have been correctly passed.

- Rather than passing the constants `N` and `p` through the `data` block, we can alternatively omit the `data` block and declare and assign values to these variables in the `transformed data` block. We can then omit the `data` argument in the call to the `stan` function in R. Write a new Stan programme of this form called `simulation_v2.stan` then compile and run the programme in R.
- In R, we can generate uniform random numbers using the `runif` function. For example:

```
runif(5, min=-3, max=3)

## [1]  0.2086775  2.8392929 -2.0527904 -0.7372377  1.5224560
```

  Use the `lookup` function to find a corresponding function in Stan.

- Augment your programme `simulation_v2.stan` to additionally simulate 30 uniform random numbers in a vector `z` with lower bound -3 and upper bound 3. Call the new programme `simulation_v3.stan`. Compile and run it in R and check that the output looks reasonable.
- Modify `simulation_v3.stan`, saving the file as `simulation_v4.stan`, so that instead of representing `x` and `z` as separate objects, you represent them as a length-2 array of vectors called `xz`.
- Modify `simulation_v4.stan`, saving the file as `simulation_v5.stan`, to additionally simulate 30 normal random variables in a vector `y` where $y[n]$ has mean `2.5 * x[n] + 1.5 * z[n]` and standard deviation 0.1. Compile and run your Stan programme in R and check that the output looks reasonable.
- If you wanted your Stan programme to return only `y` and not `xz` how could you modify your code?

3 *Vectorized operations*

In this part of the practical we will investigate the efficiency gains we can realise through the vectorized operations discussed in Chapter 3.

- Copy the Stan programme from Figure 3.1 into a text file and save it as `figure3_1.stan`.
- Create a revised version of this programme which exploits the vectorization tricks described later in the Chapter. Save it as `figure3_1_v2.stan`. Note that in lectures we considered optimising only the `model` block. It is also possible to make the `functions` and `generated quantities` blocks more efficient.
- Load the data to pass to the Stan function. For convenience in this practical, the `list` that we will construct in Chapter 4 has been saved in a data set called `practical1_data` which can be loaded through:

```
## Load data to pass to stan function:
data(practical1_data)
```

We can now compile and run each programme by executing the following code:

```
## Run the programme from Figure 3.1:
fig3_1 = stan("figure3_1.stan", data=practical1_data)
## Run the revised programme:
fig3_1_v2 = stan("figure3_1_v2.stan", data=practical1_data)
```

We will consider the `stan` function and its usage in Chapter 4. For now, simply make a note of the time taken to run each programme. (It will be printed at the bottom of the Stan output). How much speed-up do you gain by using the revised programme?

Solutions

Solutions are available as a vignette:

```
library("jrRstan")
vignette("solutions1", package="jrRstan")
```