Activity 1: Estimating the mean

Learning outcomes

1. Experience simulating stochastic data for analysis
2. Understanding the Stan workflow using the CmdStanR interface from R
3. Introduction to the Stan language syntax for coding a Bayesian model
4. Introduction to essential model diagnostics

Source files

1. Stan model:
   ./practicals/1_model_of_the_mean/1_model.stan
2. R script to simulate data and run MCMC:
   ./practicals/1_model_of_the_mean/1_script.R

Model description

The model of the mean can be written as:

$$y_i \sim Normal(\mu, \sigma)$$

$$\mu \sim Normal(0,\ 100)$$

$$\sigma \sim Uniform(0,\ 1000)$$

where y_i is the value of the dependent variable for the i^{th} sample from the data. The mean value is μ and the standard deviation is σ. The first line is the likelihood and the next two lines are the priors for our two parameters.

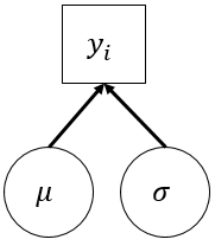
Q: What are some assumptions that are inherent in this model?

Q: What is an example of a measurement that y_i might represent? Be sure to consider the data type (e.g. continuous, discrete) and the range of support.

Q: Can you match the likelihood and priors from our model with elements from Bayes theorem below? Where is the posterior in Bayes theorem and how is it determined from our model?

$$P(\mu, \sigma | y_i) = \frac{P(y_i|\mu, \sigma)P(\mu)P(\sigma)}{P(y_i)}$$

We can represent our model with the following Directed Acyclic Graph (DAG):



In the DAG, the squares represent data and the circles are parameters. Arrows point towards a node that is dependent on another node (i.e. the arrows point towards variables that are on the left side of an expression in the model statement above). A prior must be specified for every parameter in the DAG that does not have an arrow pointing into it. Solid arrows represent stochastic relationships (~) and dashed arrows would represent deterministic relationships (=) if we had any in the model.

Q: Have we specified priors in our model for every node that does not have any arrows pointing into it?

Models cannot contain any node that is indirectly dependent on itself. This would appear as a loop in the DAG (i.e. A -> B -> C-> A). This is why DAGs are referred to as Directed **Acyclic** Graphs–the models cannot contain "cycles".

Q: Do we have any cycles in our model?

Specify the model in Stan

Please open our Stan model of the mean in RStudio or Positron.

A few basics of Stan syntax to be aware of:

- Stan models saved with the file extension `*.stan` will include helpful markup and syntax checking when opened within RStudio.
- A Stan script must be divided into program blocks (e.g. `data{ }`). We will dive into these more below.
- Each line of code must end with a semicolon.
- `//` precedes any comments that you want to be ignored by the Stan interpreter. Do not use `#` as a comment character because this will return [often cryptic] errors.
- The script must end with a blank line.

Q: Does RStudio identify any syntax errors within our model? Hint: Click the "Check" button in RStudio that should appear when you are viewing the Stan model code.

- *Try deleting a semicolon and check again*
- *Try removing the blank line at the end of the script and check again*
- *Try replacing a Stan comment symbol `//` with an R comment symbol `#` and check again*

You will notice that we have three program blocks within our Stan model:
`data{ }`, `parameters{ }`, and `model{ }`.

Note: The Stan language has other program blocks that are described in the Stan Reference Manual, but we do not need them now.

The **data** block is where we define each variable that we will provide as observed data. We need to define the data type, dimensions, and range of support (optional) for all data. We provide two variables as data for this model: the sample size (*n*) which is a positive integer, and the data (*y*) which is a vector of length *n* with values that are continuous numbers with a range of support from negative infinity to infinity.

Note: The Stan Reference Manual describes all data types available.

The **parameters** block is where we define each of the parameters that we want to estimate. In our case, this is the mean μ and the standard deviation σ. . Again, we need to define the data type, dimensions, and range of support (optional) for every parameter. Notice that we defined an appropriate range of support for σ because the standard deviation cannot be negative.

The **model** block is where we specify our model and priors. You will notice that the code is very similar to how we wrote our model in this document (above).

Simulate data

See SECTION 1 of the R script: `.../1_script.R`

We would normally use a real dataset for analysis, but we will simulate a dataset for many or our exercises. This will allow you to re-simulate data with different characteristics and to modify your model to explore how things work.

Use the default simulation settings to generate data for your first model.

Once you have completed the first model (i.e. the rest of Activity 1) with default settings, you can come back to this point and simulate new data to explore how different data characteristics or model specifications might affect your results. Here are some questions that may be interesting to explore:

Q: What happens to our estimation accuracy for μ and σ if the sample size is very small? Do you think we would under- or over-estimate σ? Can you test this?

Q: How is our posterior parameter estimate affected if the simulated true value of σ is very close to (or outside) the bounds of the prior that we specified for this parameter?

Q: What happens if the distribution used for the data generating process process (i.e. the simulated data, in this case) differs from the likelihood function in our model? For example, what if the data generating was a log-normal distribution:
`y <- rlnorm(n=n, meanlog=log(5), sdlog=0.6)`.

Running MCMC to fit the Stan model in R

See SECTION 2 of the R script: `.../1_script.R`

We will use the *cmdstanr* package to run an MCMC simulation in R that samples from the posterior distribution $P(\mu, \sigma|y_i)$ for our model.

Compile the model. We begin by compiling our Stan model (`…/1_model.stan`) into an executable file. This step is one of several features of Stan that helps it run much faster than other MCMC software (e.g. BUGS, JAGS). The model only needs to be compiled once, and then the same executable will be called (in the background) every time you re-run the same model.

Prepare data. Next, we need to format our data as a list object in R. This data format allows flexibility for variables to have different dimensions (e.g. vector, matrix, array). All data that have been specified in the Stan model code must be included in this data list. We can also include variables here that are not used by the model, so we will include the true simulated values for each parameter to keep for future reference.

The data also includes a "seed" which will be used by the random number generator in the MCMC algorithm. It is important to set this seed so that your MCMC simulations are reproducible—you will get identical results only if you re-run MCMC using the same seed, model, data, and initial values.

Specify MCMC initials. We will specify initial values to be used by the MCMC simulation for each parameter in the model. This defines the first value for each parameter in each MCMC chain. It is good practice to create a function to generate random initial values for each parameter so that the initials are different for each chain. We want to generate initials that are over-dispersed compared to the true values of the parameters (i.e. they shouldn't always be close to the correct answer).

Run MCMC. Lastly, we will configure some MCMC settings and then run the MCMC simulation. We will run four MCMC chains with a 1000 iteration warmup period (a.k.a. burn-in) that will be discarded, plus 2000 additional iterations that we will keep. This will give us 8000 samples (4 chains x 2000 iterations) drawn from the posterior distribution of our model. These MCMC draws provide our parameter estimates. These default settings will

usually work for simple models, but you may need to increase your warmup period or keep more iterations if you have MCMC convergence problems (we will look at diagnostics next).

Model diagnostics

See SECTION 3 of the R script: .../1_script.R

This code will introduce you to initial tools to diagnose MCMC problems and to assess model fit. We will look at these issues in more depth in later sessions (e.g. proper cross validation to assess model fit). Our goals today are to extract the MCMC samples, check for potential problems, retrieve our parameter estimates, and see how they compare to the true parameters that we used to generate the data.

Extract MCMC draws. We begin by seeing how to extract the MCMC draws that were sampled from our posterior distribution. We can extract these values as a data.frame and/or as an array. The MCMC draws are the values that were saved for each parameter from each iteration of each MCMC chain. When we plot the samples for multiple chains together, we should see that all chains have converged on the same parameter space. We can diagnose many convergence issues visually.

Q: Was our warmup period long enough? In other words, have the chains converged before the end of our warmup?

Q: Did we save enough iterations after warmup? In other words, do we see strong and sustained convergence on the same parameter estimates for the entire post-warmup period?

Q: Do we see any evidence that different chains settled on different parameter estimates? What might this indicate (we can discuss this as a group)?

Convergence and MCMC sampler diagnostics. We will take a look at the rhat statistic to quantify convergence. Rhat values of 1 (or very close to 1; e.g. rhat < 1.01) indicate that chains have converged for a parameter estimate. We can have convergence for some parameters but not others, so we will look at rhat for all of them.

We will also take a look at some basic quantitative measures that can be used to diagnose other MCMC issues, including divergent transitions, exceeding max tree depth, and EBFMI. Stan will return warnings when these types of issues arise along with a link to documentation about potential solutions. We will look at how to extract these metrics manually, but we will not go into detail about them today.

Check Stan documentation for more info on diagnostics and warnings: https://mc-stan.org/misc/warnings.html

Conclusion

We have now completed the most complicated process known to human-kind to estimate the mean of a data set! Yay! The goal was to go through the essentials of a basic Bayesian

workflow to familiarise you with the key components. I would certainly not recommend this process for actually estimating a mean, but I hope you gained valuable experience to demystify the process and build confidence to start working on more complex models together in our next sessions.

We have included a few additional activities in this document that we would recommend working through in class if there is time remaining or as a self-guided exercise later.

If you have time, you can now scroll back up to the box in the "Simulate Data" section to explore some of those questions.