Getting Started

2018-04-30

Using the ShinyStan app with different types of objects

stanfit objects

If `my_stanfit` is a stanfit object (the result of fitting a model with **rstan**), then to launch the ShinyStan app simply use

```
library(shinystan)
my_sso <- launch_shinystan(my_stanfit)
```

and ShinyStan will launch. Here `my_sso` is the name you want to use for the shinystan object that will be returned. If you simply run

```
launch_shinystan(my_stanfit)
```

then ShinyStan will launch but no shinystan object will be saved upon quitting the app.

Sometimes keeping only a subset of parameters before can improve performance. This can be done by creating an object with `as.shinystan` and specifying the `pars` argument. The resulting shinystan object can then be passed to `launch_shinystan()`.

stanreg and brmsfit objects

The **rstanarm** and **brms** packages provide `launch_shinystan` methods for stanreg and brmsfit objects, respectively. For example, the method for stanreg objects is documented at http://mc-stan.org/rstanarm/reference/launch_shinystan.stanreg.html

mcmc.list objects

If you have an `mcmc.list` object called `my_mcmc` then you can use the `as.shinystan` function to convert `my_mcmc` into a shinystan object that can then be used with `launch_shinystan`:

```
my_sso <- launch_shinystan(as.shinystan(my_mcmc, model_name = "my_model"))
```

If, for example, the first 100 iterations in each chain in `my_mcmc` are warmup iterations, you should add the `warmup` argument when you call `as.shinystan`:

```
my_sso <- launch_shinystan(as.shinystan(my_mcmc, model_name = "my_model", warmup =
100))
```

However, you should only use the `warmup` argument if the warmup iterations have been saved and included in `my_mcmc`.

Other types of objects

3-D array

To convert a 3-D array to a shinystan object make sure that the three dimensions of the array correspond to the *number of iterations* **x** *number of chains* **x** *number of parameters*. You can then `as.shinystan` exactly how it's used in the examples for `mcmc.list` objects above (but you don't need to convert your array to an `mcmc.list`).

List of matrices

If you have separate chains that are each a matrix (with iterations as rows and parameters as columns) you can combine them in a list to pass to `as.shinystan`

```
# Generate some fake data
chain1 <- cbind(beta1 = rnorm(100), beta2 = rnorm(100), sigma = rexp(100))
chain2 <- cbind(beta1 = rnorm(100), beta2 = rnorm(100), sigma = rexp(100))
chain_list <- list(chain1, chain2)
my_sso <- launch_shinystan(as.shinystan(X = list(chain1, chain2), model_name =
"my_model"))
```

Other functions in the shinystan package

Generating new quantities

You can add a new parameter/quantity as a function of one or two existing parameters to your shinystan object by using the `generate_quantity` function. For example, assume `sso` is a shinystan object and two of the parameters are `alpha` and `beta`. We could add a parameter `gamma` that is the inverse logit of `beta` using the code

```
inv_logit <- function(x) 1/(1 + exp(-x))
sso <- generate_quantity(sso, fun = inv_logit, param1 = "beta", new_name = "gamma")
```

Here, `fun` is the function we want to use, `param1` is the name of the parameter to apply the function to, and `new_name` is the name to give the new parameter.

Adding a parameter as a function of two parameters just requires specifying the `param2` argument and providing a function of two variables. For example, we can add a parameter `delta` to `sso` that is the squared difference of `alpha` and `beta` like this

```
sso <- generate_quantity(sso, fun = function(x,y) (x-y)^2,
                         param1 = "alpha", param2 = "beta", new_name = "delta")
```

Storing your model code in a shinystan object

For models fit using **rstan** the model code will automatically be stored in the `model_code` slot of your shinystan object. When ShinyStan is open you can view your model code in the *Model Code* tab.

If you did not use **rstan** fit your model then you can add your model code by using the `model_code()` function. For example, you may have used Bugs or JAGS or some other software and want to add the following code

```
  for (i in 1:length(Y)) {
     Y[i] ~ dpois(lambda[i])
     log(lambda[i]) <- inprod(X[i,], theta[])
   }
   for (j in 1:J) {
     theta[j] ~ dt(0.0, 1.0, 1.0)
   }
 }
```

to your shinystan object. To add that code you can simply include it as the `code` argument to the `model_code` function

```
my_code <- "
 model {
  for (i in 1:length(Y)) {
     Y[i] ~ dpois(lambda[i])
     log(lambda[i]) <- inprod(X[i,], theta[])
   }
   for (j in 1:J) {
     theta[j] ~ dt(0.0, 1.0, 1.0)
   }
 }
"

# Add the code to a shinystan object sso
sso <- model_code(sso, code = my_code)
```

Renaming a model

On the home page ShinyStan will display the model name associated with the shinystan object being used. This name can be set by adding the `model_name` argument to `as.shinystan` when creating a shinystan object. For an existing shinystan object you can use the `model_name` function like this:

```
sso <- model_name(sso, "new_model_name")
```

where `"new_model_name"` is the new name you want to give your model.