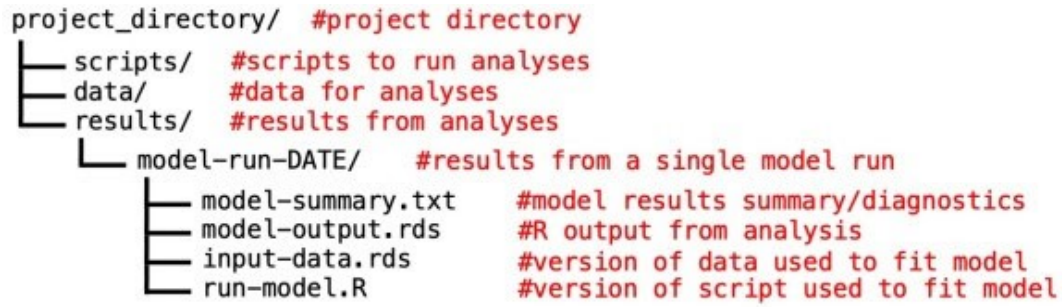
The background

Results from Bayesian analyses are often the result of many hours of study design, data processing, model construction, evaluation, checking, and validation (see here for an excellent overview of a typical Bayesian workflow). Because model building/fitting/checking is an iterative process, we often are left with many versions of scripts, objects, and files associated with an analysis. While this issue isn't specific to Bayesian analyses, that's what I focus on here (as that is the focus of the MCMCvis R package, which aims to facilitate the visualization and wrangling of MCMC output).

The problem

This overflow of information can make it difficult to stay organized, keep track of the model development process, and refer back to past results. A number of strategies can be used to help combat the deluge of code and results, one of which being version control, such as git. However, given the large file sizes that model results often generate, version control is often not an effective strategy for tracking changes over time (given that the file size limit for many online version control platforms, such as Github, is several hundred MB). Moreover, wading through one's git history to find a particular set of model results and the code associated with them can be quite arduous (speaking from personal experience here).

As a user of Bayesian models, I want an archive of my model results, a general summary/list of diagnostics of those results, and the scripts used to fit these models in a single location. I've found that creating a new directory (typically with the date the analysis was run) for each model run, and storing all the relevant files associated with that analysis there to be helpful. A typical project directory for me might look like the following:



Archiving results in this way provides a quick and easy way to reference a given model run (how this model fit, what the results were). However, this is somewhat of a pain to produce and code to achieve this can quickly clutter scripts!

A solution

The `MCMCdiag` function in MCMCvis 0.15.0, now available on CRAN, aims to provide some structure to do the tedious tasks outlined above with a single function call. Using the default options, `MCMCdiag` takes an object output from one of the various packages used to fit Bayesian models in R (including `rstan`, `nimble`, `rjags`, `jagsUI`, `R2jags`, `rstanarm`, and `brms`) as an argument, and produces a quick and basic summary of the model results and diagnostics in the form of a .txt file. This .txt file serves as a point of reference for the results from this particular model.

Before getting to the other bells and whistles in this function, let's first take a look at that file, using results from a model fit using `rstan` to interface with `Stan`. Code to simulate the data and fit the model can be found at the end of this post.

```
library(MCMCvis)

#fit is an object produced from rstan::stan
#round = 2 used to round output for readability in this post
```

```
MCMCdiag(fit, round = 2)
```

Based on the information in the .txt file (below), it looks like our model ran quickly, with good numbers for some basic diagnostics. Note that some fields in this file will vary depending on which software was used to fit the model. That is, output from models fit with `rstan` will have different fields compared to output from models fit with `nimble` (as relevant information such as diagnostics differs slightly between these two programs). More on Stan-specific diagnostics can be found in the Stan documentation. .txt file output:

```
Information and diagnostics
===========================
Run time (min):                    0.01
Total iter:                        2000
Warmup:                            1000
Thin:                              1
Num chains:                        4
Adapt delta (specified):           0.95
Max tree depth (specified):        13
Initial step size (specified):     0.01
Mean accept stat:                  0.96
Mean tree depth:                   2.6
Mean step size:                    0.5856
Num divergent transitions:         0
Num max tree depth exceeds:        0
Num chains with BFMI warnings:     0
Max Rhat:                          1
Min n.eff:                         1530


Model summary
=============
          mean   sd     2.5%      50%    97.5% Rhat n.eff
alpha     1.92 0.23     1.49     1.92     2.37    1  4020
beta      2.93 0.08     2.78     2.93     3.09    1  3598
sigma     7.26 0.16     6.93     7.26     7.58    1  3006
lp__  -2478.96 1.22 -2482.08 -2478.65 -2477.55    1  1530
```

However, while this is helpful for summarizing the model output, we still have some work to do to reproduce the organizational structure outlined above. Fortunately, by giving `MCMCdiag` several arguments we can do the following (in addition to creating the above .txt summary/diagnostic file):

1. **create a new directory for the files associated with this analysis (`mkdir` argument)**
2. **save the object from `rstan` as a .rds file for later use (`save_obj` argument)**
3. **save the data used to fit this model and `sessionInfo` as .rds objects (`add_obj` argument)**
4. **create a copy of the .stan model file and script used to fit this model (`cp_file` argument)**

```
MCMCdiag(fit,
         mkdir = 'blog-model-2021-03-25',
         file_name = 'blog-model-summary',
         dir = '~/project_directory/results',
         save_obj = TRUE,
         obj_name = 'blog-model-output',
         add_obj = list(DATA, sessionInfo()),
         add_obj_names = c('blog-model-data',
                           'session-info'),
         cp_file = c('model.stan', 'blog-model.R'))
```

My `results/` directory now looks as follows:

```
results/
└── blog-model-2021-03-25/
    ├── blog-model-summary.txt   #summary/diagnostic file above
    ├── blog-model-output.rds    #output from rstan
    ├── blog-model-data.rds      #data used to fit model
    ├── blog-model.R             #script used to fit model
    ├── model.stan               #.stan file used to fit model
    └── session-info.rds         #sessionInfo when model was fit
```

If you aren't familiar with .rds objects, they can be read into R in the following way:

```
fit <- readRDS('blog-model-output.rds')
```

Additional arguments to `MCMCdiag` can be used to save additional objects as .rds files, add additional fields to the .txt file (useful for making notes about a particular model run), and control how the summary information is presented (as with other functions in `MCMCvis`). See the package vignette for a more complete overview of `MCMCvis` functionality.

As a workflow

As I make changes to my analyses (e.g., data processing, model parameterization, priors), I can create an catalog of results by making `MCMCdiag` calls after each model run. Importantly, all the information required to reproduce results from a given run (i.e., data to fit the model, code to fit the model, and environment information from `sessionInfo`) is contained within each one of these directories, sparing one the pain of combing back through version control history to find out how the data were structured, what model was run, and what the results were at a given point in a project.

When looking back at model results, I generally want to know: 1) what the model was, 2) whether it fit properly, and 3) what the parameter estimates were. This organizational structure facilitated by `MCMCdiag` makes this achievable with minimal code.

Code to simulate data and fit model

Simulate data

```
#set seed
set.seed(1)

#number of data points
N <- 1000

#simulated predictor
x_sim <- seq(-5, 5, length.out = N)

#intercept generating value
alpha_sim <- 2

#slope generating value
beta_sim <- 3

#generated error (noise)
eps_sim <- rnorm(length(x_sim), 0, 7)

#simulate response
y_sim <- alpha_sim + beta_sim * x_sim + eps_sim
```

Stan model (model.stan)

```
data {
int N;
vector[N] y;
vector[N] x;
}

parameters {
real alpha;
real beta;
real sigma;
}

model {
alpha ~ normal(0, 30);
beta ~ normal(0, 30);
sigma ~ normal(0, 30);

y ~ normal(alpha + beta * x, sigma);
}
```

Fit model

```
library(rstan)

DATA <- list(N = length(y_sim),
             y = y_sim,
             x = x_sim)

rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

fit <- stan('model.stan',
            data = DATA,
            iter = 2000,
            chains = 4,
            cores = 4,
            pars = c('alpha',
                     'beta',
                     'sigma'),
            control = list(adapt_delta = 0.95,
                           max_treedepth = 13,
                           stepsize = 0.01))…
```