Tutorial

This tutorial aims to fit Mixed Effects Nonlinear Models with Differential Equations using Stan in R. Stan is a highly advanced Bayesian programming language developed in 2012 by Andrew Gelman and his colleagues. Stan employs Hamiltonian Monte Carlo and the No-U-Turn Sampler algorithms, enabling efficient exploration of high-dimensional parameter spaces, making it well-suited for complex models and large datasets.

We will explore the power of Stan as we simulate data, perform Bayesian inference, and analyze the model fit, providing you with essential skills for Bayesian modeling in dynamic systems.

0 - Preliminary step

This document is to be used along *nlme_ode_tutorial_stan.stan* present in the github. In this example, we are using the R package CmdStanR to write in Stan, a lightweight interface compatible with the lastest updates of Stan. The package Rstan, more user-friendly but using older and sometimes deprecated versions of Stan, might also be used.

Import all libraries

```
install.packages(c('dplyr', 'tidyverse','cmdstan', 'ggplot2','gridExtra','bayesplot','posterior','deSolve','mse
m','magrittr','dplyr')) #Install all packages

library(dplyr)    #Load all packages
library(tidyverse)
library(ggplot2)
library(gridExtra)
library(bayesplot)
library(cmdstanr)
library(posterior)
library(deSolve)
library(mse)
library(magrittr)
library(dplyr)
options(mc.cores = parallel::detectCores())

set.seed(123)
```

I - Data Simulation

We are basing this example on the article Timing HIV infection with a simple and accurate population viral dynamics model, by Daniel B. Reeves et al. The detailed parameters given in this paper allows us to simulate data similar to the RV217 study conducted in by MHRP in 2018.

The HIV primary infection model used here is a Holte/Cardozo model, which is a modified version of the standard viral dynamics model.

$$\partial_t S = \alpha_S - \delta_S S - \beta S V$$

$$\partial_t I = \beta S V - \kappa I^{h+1}$$

$$\partial_t V = \pi \mathrm{I} - \gamma \mathrm{V} - \beta \mathrm{SV} \partial_t \mathrm{V} = \pi I - \gamma V - \beta S V$$

The variables are:

- S: concentration of HIV-susceptible cells (cells $\mathrm{ml}^{-1} ml^{-1}$)
- I: infected cells
- V: plasma viral load (viral RNA copies $\mathrm{ml}^{-1} ml^{-1}$)

The model requires eight free parameters:

- $\alpha_S \alpha_S$ the constant growth rate of susceptible cells (cells $\mu \mathrm{l}^{-1} d^{-1} \mu l^{-1} d^{-1}$)
- $\delta_S \delta_S$ the death rate of susceptible cells ($d^{-1} d^{-1}$)
- $\beta \beta$ a mass-action viral infectivity ($\mu \mathrm{l}$ virus $^{-1} \mathrm{d}^{-1}$ $virus^{-1} d^{-1}$)
- $\pi \pi$ the viral production rate (virions cell^{-1} $cell^{-1} d^{-1} d^{-1}$)
- $\gamma \gamma$ the clearance rate of the virus ($d^{-1} d^{-1}$)
- $\kappa \kappa$ the rate that controls the death and killing of infected cells (cells^{-h} $cells^{-h} d^{-1} d^{-1}$)
- h the exponential factor adjusting the density-dependent death rate

Simulate the ODE system

```
ode_model <- function (t, x, params) {
  # State variables
  S <- x[1]
  I <- x[2]
  V <- x[3]
  # Parameters
  alpha_S <- params["alpha_S"]
  delta_S <- params["delta_S"]
  beta <- params["beta"]
  K <- params["K"]
  h <- params["h"]
  pi_ <- params["pi_"]
  gamma <- 23
  # Equations
  dSdt <- alpha_S-delta_S*S-beta*S*V
  dIdt <- beta*S*V- K*I^(h)
  dVdt <- pi_*I-gamma*V-beta*S*V
  # Return
  dxdt <- c(dSdt,dIdt,dVdt)
  list(dxdt)
}
```

Sample the parameters for each subject

Here, all of the parameters vary between subjects. To reduce computational time and have a first look at the inference process, one might start by fixing some of the parameters then increasingly freeing them.

```
N = 30 # number of subjects
Ttot = 90 # number of days
all_params = matrix(0, nrow = N, ncol = 7) # matrix containing all final parameters



# Set the hierarchical means
  mu_alpha_S <- 63.7
  mu_delta_S <- 0.000751
  mu_lbeta <-  -4.18
  mu_K <- 0.356
  mu_h <- 0.148
  mu_lpi_ <- 1.09

# Set the hierarchical variances
  tau_alpha_S <- 0.1
  tau_delta_S <- 0.0023
  tau_lbeta <- 0.1
  tau_K <- 0.1
  tau_h <- 0.01
  tau_lpi_ <- 0.1

for (i in 1:N){

  # Sample each parameter
  params <- list()
  params[1] <- rnorm(1,mu_alpha_S,tau_alpha_S)
  params[2] <- rtnorm(1,mu_delta_S,tau_delta_S, lower=0)
  params[3] <- 10^(rnorm(1,mu_lbeta,tau_lbeta))
  params[4] <- rtnorm(1,mu_K,tau_K, lower=0)
  params[5] <- rtnorm(1,mu_h,tau_h, lower=0)
  params[6] <- 10^(rnorm(1,mu_lpi_,tau_lpi_))



  # Add to matrix containing all parameters

  for (j in 1:6){
    all_params[i,j] <- all_params[i,j]+ as.numeric(params[j][1])
  }
    all_params[i,7] = 23 #gamma is not variable between individuals
}
```

Simulate the data for each subject and time point

```
ytot = matrix(0, nrow = Ttot, ncol = N) # matrix containing all the simulated data for the viral load

for (j in 1:N){
  individual_params <- c(alpha_S=all_params[j,1], # get the individual parameters for each subject
                         delta_S=all_params[j,2],
                         beta=all_params[j,3],
                         K=all_params[j,4],
                         h=all_params[j,5],
                         pi_=all_params[j,6],
                         gamma=all_params[j,7])

  times <- seq(from=1, to=Ttot, by=1) # time range for the simulation
  xstart <- c(S=8.388815e+04, I=1.869510e-02,V=1.0e-02) # initial state of the system


  ode( # using the R general solver for ODEs
    func=ode_model,
    y=xstart,
    times=times,
    parms=individual_params
  ) %>%
    as.data.frame() -> out

  for (i in 1:Ttot){
    ytot[i,j] <- ytot[i,j]+ as.numeric(out$V[i]) # add the viral load for each subject and each time point
  }
}
```

Add noise

```
sigma_noise = 0.1
ytot_test = matrix(0, nrow = Ttot, ncol = N)
for (j in 1:N){
  for (i in 2:Ttot){
    pos = 0
    while (pos==0){
      ytot_test[i,j] <- 10^(log10(ytot[i,j]) + rnorm(1,0,sigma_noise))
      if(ytot_test[i,j]>0.01){
        pos=1
      }
    }
    ytot[i,j] <- ytot_test[i,j]
  }
}
```

Clean simulated data

```
time <- list(1:Ttot)
names(time)[1] <- "days"
RV217_sim <- as.list(as.data.frame(ytot))
RV217_sim <- append(RV217_sim, time, 0)
RV217_simdf <- as.data.frame(RV217_sim)
```

Save simulated data in an external file

```
write.table(RV217_simdf, file = "RV217_simulated_data", append = FALSE, sep = ",", dec = ".",
            row.names = TRUE, col.names = TRUE)
```

Plot the simulated data

Here we plot log10(1000*V) over the days

```
days = list()
VL = list()
ID = list()

for (i in 1:N){
  for (j in 1:Ttot){
    days <- append(days, RV217_sim[[1]][j])
    VL <- append(VL,RV217_sim[[i+1]][j])
    ID <- append(ID, i)
  }
}

days <- as.numeric(unlist(days))
VL <- as.numeric(unlist(VL))
ID <- as.numeric(unlist(ID))


RV217_sim_unpivot = list(days, VL, ID)
names(RV217_sim_unpivot)[1] <- "days"
names(RV217_sim_unpivot)[2] <- "VL"
names(RV217_sim_unpivot)[3] <- "ID"
RV217_sim_unpivot <- data.frame(RV217_sim_unpivot)

plt <- ggplot(RV217_sim_unpivot,aes(y = log10(1000*VL),x = days,group = ID, col = VL)) + geom_line()
plt +scale_color_gradient(low="blue", high="red")
```

II - Stan Inference

Read the data

```
data <- read.table("RV217_simulated_data",sep=',', header=TRUE)
```

Time optimization

Reducing the number of subjects or time points we use to fit the model is one of the first step to reduce the computation time drastically.

```
data_red=data[c(1,2,7,14,21,28,35,42,49,56,63,70,77,84),]
```

Format our data for Stan use

The stan function accepts data as a named list, a character vector of object names, or an environment.

```
n_sub <- 30 #number of subjects
T <- length(data_red$days)-1 #number of time points
y0_V <- as.numeric(data_red[data_red$days == 1, -1]) #initial value
t0 <- 1 #first time point
times <- as.numeric(data_red$days)[-1] #vector of all time points without first one
y <- data_red[data_red$days > 1, -1] #vector of all values without first one
model_data_red <- list(n_sub = n_sub, T = T, y = y, y0_V = y0_V, times = times, t0 = t0)
print(model_data_red)
```

Inference and optimization

Compile the Stan code. Make sure to have the latest version of *R* available. This is where errors may arise.

```
mod_CH_fixed <- cmdstan_model("Reeves_model_fixed.stan")
```

Execute the file. On Windows, there might be some difficulties with executing code via cmdstan. *Refer to* this forum.

```
mod_CH_fixed$exe_file()
```

Fit on data.

```
fit_CH_red <- mod_CH_fixed$sample(data = model_data_red, chains = 2, num_warmup =250, num_samples  =200, cores=2)
```

To decrease computation time, the following arguments can be tuned:

- **Chains**: The default number of chains is 4. In a parallel environment, the best practice would be using one chain/core, depending on how many cores the processor supports.
- **Target proposal acceptance probability**: *adapt_delta* is the target average proposal acceptance probability during Stan's adaptation period. The default value is 0.95 and can be brought closer to 1 as a remedy to divergent transitions, but will result in higher calculation time.
- **Sample size**: Use minimum 200 samples each for warmup and sampling. The default is 1000 each. This can be changed using the *num_warmup* and *num_samples* arguments.

The article Taming divergences in Stan models from Martin Modrak's blog gives a detailed list on how to diagnose divergences.

III - Fit Analysis

To analyze the parameter estimate, start by looking at two diagnostic statistics that appear when printing the fit.

- **Rhat** is a diagnostic tool that can indicate a lack of convergence by comparing the variance between multiple chains to the variance within each chain. If the parameters successfully explored their full space for each chain, Rhat should be close to 1.
- **Ess-bulk** is the number of effectively independent samples that are drawn. It should be over 100/chain.

```
fit_CH_red$time()
print(fit_CH_red, max_rows = 200, digits = 5)
fit_CH_red$draws()
```

Transforming the output into a *stanfit* object makes it easier to use all the plot packages.

```
stanfit <- rstan::read_stan_csv(fit_CH_red$output_files())
```

First, check the trace of the chains. If they are not converging, there is an unidentifiability in the model. When using simulated data, the chains might not mix if the priors are too strong.

```
stan_trace(stanfit)
```

The package *bayesplot* provides an extensive library of plotting functions for use after fitting Bayesian models. One of the advantages of stan is that it gives access to the individual distribution of each parameter for each subject.

Here are some examples of possible plots for parameter analysis.

```
mcmc_trace(fit_CH_red$draws("k"), pars = c("k[27,1]")) #trace plots

mcmc_dens_overlay(fit_CH_red$draws("alpha_S"), pars = c("alpha_S[1,1]","alpha_S[2,1]","alpha_S[3,1]","alpha_S[4,
1]")) #separates the Markov chains

mcmc_areas(fit_CH_red$draws("logbeta"), pars = c("logbeta[27,1]")) #uncertainty intervals as shaded areas under t
he estimated posterior density curves
```