Exploring the challenges

Amaël Le Squin

2024-11-29

> ❗ Available on Gitlab
>
> This note is available on gitlab.

Introduction

In order to test my models, I first run dummy tests, with generated data. I build models slowly, from the simplest to the most complex. The first challenge is related to the model used by Christine Deleuze *et al.* (scripts given on 11 October 2024):

```
Modele_mai2 = nlme(formTotNew ~ a + b*hdn + d*hsurd, data = Grdata.PV,
    start = c(a = 0.4, 0, b = 1.5, 0, d = 0.0005, 0),
    fixed = list(a + b + d ~ feuil.res), random = a + d ~ 1|nomessence2)
```

Few things raised a flag in my mind:

1. The R package `nlme`, non-linear mixed effect, is for repeated measured data and is based on Lindstrom and Bates (1990). Here, the repetition occurs within the species level, so I am not sure this package is the best adapted… I feel that `lme4` would have been better
2. `hdn` is the hardiness $\sqrt{c}/h$ while `hsurd` is what I call the 'slenderness', h/c. Therefore, there is a high (negative) correlation between these two explanatory variables!

The model `Modele_mai2` should be understood as follows, for an individual of height h and circumference at breast height c, of species j and functional group (conifer or broadleaf) i:

$$\begin{aligned} \mathcal{F} &\sim \mathcal{N}(\mu, \sigma) \\ \mu_{i,j} &= a_{i,j} + b_i \frac{\sqrt{c}}{h} + d_{i,j} \frac{h}{c} \\ a_{i,j} &\sim \mathcal{N}(\alpha_i, \sigma_\alpha) \\ d_{i,j} &\sim \mathcal{N}(\delta_i, \sigma_\delta), \end{aligned} \tag{1}$$

where α_i and δ_i are the 'group intercepts' (common values within conifers and broadleaves). Therefore, it seems to be a GLMM, and I am not sure `nlme` is relevant here. In the next sections, I will try to reproduce their model with generated data. I will do it step-by-step. Before that, I rewrite the equations with new variable names and without the indices i and j, in order to correspond to stan code:

$$\begin{aligned} \mathcal{F} &\sim \mathcal{N}(\mu, \sigma) \\ \mu &= \beta_0 + b_1 \frac{\sqrt{c}}{h} + \beta_2 \frac{h}{c} \\ \beta_0 &\sim \mathcal{N}(b_0, \sigma_0) \\ \beta_2 &\sim \mathcal{N}(b_2, \sigma_2), \end{aligned} \tag{2}$$

Generating data

Packages and helpers

First, I load the necessary packages:

```
#### Clear space and load packages
rm(list = ls())
graphics.off()

options(max.print = 500)

library(data.table)
library(cmdstanr)
    register_knitr_engine(override = TRUE)
library(stringi)
library(gt)

setHook(packageEvent("grDevices", "onLoad"),
function(...) grDevices::X11.options(type='cairo'))
options(device='x11')
```

> Available on Gitlab
>
> By default, Quarto uses the knitr's built-in stan engine `rstan`. To override it so that all stan chunks are processed with `CmdStanR`, I need to specify:
>
> ```
> register_knitr_engine(override = TRUE)
> ```

Then, I define useful functions:

```r
#### Tool functions
## Tools
source("./toolFunctions.R")

## Generate k integers with constraint they sum to n >= k
generate_random_partition = function(n, k)
{
    if (n < k)
        stop("n should be larger than k")
    parts = c(0, sort(sample(1:(n - 1), k - 1)), n)
    return(diff(parts))
}
```

Parameters

I define the following parameters, that will be used to generate data according to equation (1):

```r
#### Define parameters
set.seed(1969 - 08 - 18) # Woodstock seed

## Fixed effects
b0 = c(conif = 0.2, broad = 5.3)
b1 = c(conif = 0.27, broad = 1.4)
b2 = c(conif = 4, broad = 0.21)

## Variance for random effects and residuals
sigma_beta0 = 4.2
sigma_beta2 = 6.98
sigma = 1.2

## Number of data and species
n = 3e3
S = 30 # Number of species
```

We aim to recover them later with a statiscal model.

Data

```
rep_species = generate_random_partition(n, S)

lim_broadleaf = sum(rep_species[1:19]) + 1

n_conif = lim_broadleaf - 1
n_broad = n - n_conif

fake_dt = data.table(
    species = rep(paste0("sp_", 1:S), times = rep_species),
    type = c(rep("conif", n_conif), rep("broad", n_broad)),
    fake_hdn = runif(n = n, min = -10, max = 50),
    fake_slenderness = rnorm(n = n, mean = 0, sd = 20),
    b0 = c(rep(b0["conif"], n_conif), rep(b0["broad"], n_broad)),
    b1 = c(rep(b1["conif"], n_conif), rep(b1["broad"], n_broad)),
    b2 = c(rep(b2["conif"], n_conif), rep(b2["broad"], n_broad)))

fake_dt[, beta0 := rnorm(1, unique(b0), sigma_beta0), by = species]
fake_dt[, beta2 := rnorm(1, unique(b2), sigma_beta2), by = species]

b0m = fake_dt[, round(mean(unique(beta0)), 3), by = type]
b0m[, sig := fake_dt[, round(sd(unique(beta0)), 3)]]
setkey(b0m, type)

b2m = fake_dt[, round(mean(unique(beta2)), 3), by = type]
b2m[, sig := fake_dt[, round(sd(unique(beta2)), 3)]]
setkey(b2m, type)

fake_dt[, fake_mu := beta0 + b1*fake_hdn + beta2*fake_slenderness]
fake_dt[, fake_vol := rnorm(.N, fake_mu, sigma)]
fake_dt[, fake_vol_no_randeff := rnorm(.N, b0 + b1*fake_hdn + b2*fake_slenderness, sigma)]

ind_species = fake_dt[, .(start = .I[1], end = .I[.N]), by = .(species)]
ind_species[, n_indiv := end - start + 1, by = species]
ind_species[, sum(n_indiv)] == n
ind_species = merge.data.table(ind_species, unique(fake_dt[, .(species, type)]))
setorder(ind_species, start)

n_sp = ind_species[, .N, by = type]
setkey(n_sp, type)

temporary = lm(fake_dt[, fake_vol] ~ 0 + fake_dt[, fake_mu])
```

```
sig_est = round(summary(temporary)$sigma, 3)
```

Here is a summary of the parameters value β_0 and β_2 (simulated), and b_0 and b_2, which are supposed to be the mean of β_0s and β_2s:

Table 1: Parameters value for both functional groups

Parameter	Conifer	Broadleaf
b_0	0.2	5.3
β_0	0.27	6.014
σ_0	4.2	4.2
σ_0 (data)	5.186	5.186
b_1	0.27	1.4
b_2	4	0.21
β_2	4.896	-0.16
σ_2	6.98	6.98
σ_2 (data)	7.974	7.974
σ	1.2	1.2
σ (lm)	1.199	1.199

As can be seen, the data do not necessarily represent the true parameters value very well.

Parameter recovery

> Caching execution
>
> To store some results that could be slow to obtain, or to avoid the recompilation of stan code, use the option `cache = TRUE`.

Simplest model

I first define the simplest possible model, where there is no hierarchy, and I only try to estimate b_0, b_1, b_2, and the residual variance σ:

Prepare the data for Stan,

```
#### Stan data
## Data list
stanData = list(
    N = fake_dt[, .N],
    S = S,
    n_sp_conif = n_sp["conif", N],
    n_sp_broad = n_sp["broad", N],
    ind_start_conif = ind_species[type == "conif", start],
    ind_start_broad = ind_species[type == "broad", start],
    ind_end_conif = ind_species[type == "conif", end],
    ind_end_broad = ind_species[type == "broad", end],
    lim_broadleaf = lim_broadleaf,
    fake_hdn = fake_dt[, fake_hdn],
    fake_slenderness = fake_dt[, fake_slenderness],
    volume_m3 = fake_dt[, fake_vol_no_randeff]
)

## Common variables
n_chains = 4
iter_warmup = 500
iter_sampling = 1500
```

and run the simplest model (see Listing 1):

```
info_dt = pretty_summary(fit = fit, params = fit$metadata()$model_params[-1]) # -1 to remove
info_dt |>
    gt() |>
    cols_label(
        params_name = "Parameter",
        mean_params = "Mean",
        sd_params = "Std. dev",
        r_hat_params = "r hat"
    ) |>
    fmt_number(
        n_sigfig = 2
    ) |>
    tab_style(
        style = cell_borders(sides = "all", style = NULL),
        locations = cells_body()
    ) |>
    tab_style(
        style = cell_text(weight = "bold"),
```

Parameter	Mean	Std. dev	r hat
b0[1]	0.25	0.047	1.0
b0[2]	5.3	0.046	1.0
b1[1]	0.27	0.0018	1.0
b1[2]	1.4	0.0017	1.0
b2[1]	4.0	0.0016	1.0
b2[2]	0.21	0.0015	1.0
sigma	1.2	0.015	1.0

```
        locations = list(cells_column_labels(), cells_column_spanners())
    ) |>
    tab_style(
        style = cell_text(align = "right"),
        locations = cells_body(columns = params_name)
    )
```

Hierarchical model, group effect variances provided

A slightly more complex model, this time with a group effect (*i.e.,* random effect β_0s and β_2s to estimate), but the two variances σ_0 and σ_2 are provided:

```
data {
    // Dimensions and indices
    int N; // Number of individuals
    int S; // Number of species
    int<lower = 0, upper = S> n_sp_conif; // number of conifer species
    int<lower = S - n_sp_conif, upper = S - n_sp_conif> n_sp_broad; // number of broadleaf sp
    array[n_sp_conif] int ind_start_conif; // Conifer species index start
    array[n_sp_broad] int ind_start_broad; // Broadleaf species index start
    array[n_sp_conif] int ind_end_conif; // Conifer species index end
    array[n_sp_broad] int ind_end_broad; // Broadleaf species index end

    // Predictors
    vector [N] fake_hdn;
    vector [N] fake_slenderness;

    // Response variable
    vector[N] volume_m3;
}
```

```
parameters {
    // Fixed effects
    vector[2] b0;
    vector[2] b1;
    vector[2] b2;

    vector[S] beta0;
    vector[S] beta2;

    // Variance
    real<lower = 0> sigma; // sd residuals
}

model {
    // Priors
    target += normal_lpdf(b0 | 0, 10);
    target += normal_lpdf(b1 | 0, 10);
    target += normal_lpdf(b2 | 0, 10);

    target += inv_gamma_lpdf(sigma | 1, 1);

    for (i in 1:n_sp_conif)
    {
        // Hierarchy
        target += normal_lpdf(beta0[i] | b0[1], 4.2);
        target += normal_lpdf(beta2[i] | b2[1], 6.98);

        // Likelihood conifers
        target += normal_lpdf(volume_m3[ind_start_conif[i]:ind_end_conif[i]] | b0[1] +
            b1[1]*fake_hdn[ind_start_conif[i]:ind_end_conif[i]] +
            b2[1]*fake_slenderness[ind_start_conif[i]:ind_end_conif[i]], sigma);
    }

    for (i in 1:n_sp_broad)
    {
        // Hierarchy
        target += normal_lpdf(beta0[n_sp_conif + i] | b0[2], 4.2);
        target += normal_lpdf(beta2[n_sp_conif + i] | b2[2], 6.98);

        // Likelihood broadleaves
        target += normal_lpdf(volume_m3[ind_start_broad[i]:ind_end_broad[i]] | b0[2] +
            b1[2]*fake_hdn[ind_start_broad[i]:ind_end_broad[i]] +
```

Parameter	Mean	Std. dev	r hat
b0[1]	0.25	0.048	1.0
b0[2]	5.3	0.047	1.0
b1[1]	0.27	0.0018	1.0
b1[2]	1.4	0.0017	1.0
b2[1]	4.0	0.0015	1.0
b2[2]	0.21	0.0015	1.0
sigma	1.2	0.016	1.0

```
            b2[2]*fake_slenderness[ind_start_broad[i]:ind_end_broad[i]], sigma);
    }
}
```

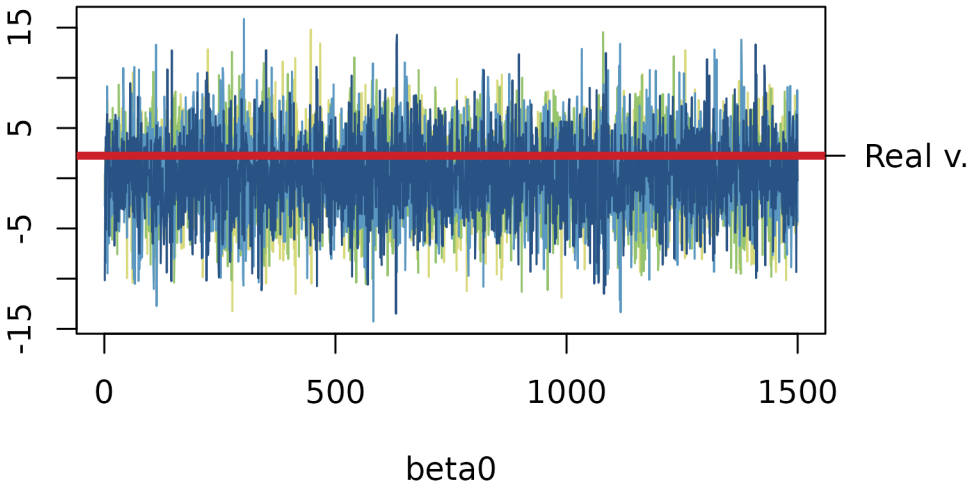
and run the hierarchical model:

This gives the fillowing results:

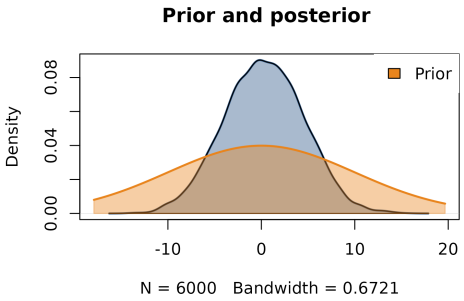
Complete model, where nothing is provided

> **!** Quick and dirty way (which does not work)
>
> Simply adding two priors on σ_0 and σ_2 like bellow will not work:
>
> ```
> target += inv_gamma_lpdf(sigma_beta0 | 1, 1);
> target += inv_gamma_lpdf(sigma_beta2 | 1, 1);
> ```
>
> It seems that this renders the model unidentifiable. Therefore, I am going to try a non-centred parametrisation. Another example is available here.

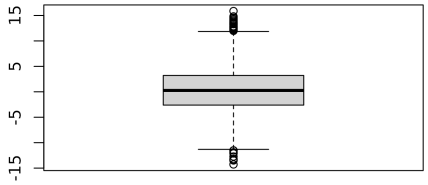
```
data {
    // Dimensions and indices
    int N; // Number of individuals
    int S; // Number of species
    int<lower = 0, upper = S> n_sp_conif; // number of conifer species
    int<lower = S - n_sp_conif, upper = S - n_sp_conif> n_sp_broad; // number of broadleaf sp
    array[n_sp_conif] int ind_start_conif; // Conifer species index start
    array[n_sp_broad] int ind_start_broad; // Broadleaf species index start
    array[n_sp_conif] int ind_end_conif; // Conifer species index end
    array[n_sp_broad] int ind_end_broad; // Broadleaf species index end
```



(a) Trace plot



(b) Posterior



(c) Boxplot

Figure 1: Trace plot and posterior of β_0 for the fifth species

```
    // Predictors
    vector [N] fake_hdn;
    vector [N] fake_slenderness;

    // Response variable
    vector[N] volume_m3;
}

parameters {
    // Fixed effects
    vector[2] b0;
    vector[2] b1;
    vector[2] b2;

    vector[S] beta0;
    vector[S] beta2;

    // Variance
    real<lower = 0> sigma; // sd residuals
    real<lower = 0> sigma_beta0; // sd random effect on intercept
    real<lower = 0> sigma_beta2; // sd random effect on slenderness slope
}

model {
    // Priors
    target += normal_lpdf(b0 | 0, 10);
    target += normal_lpdf(b1 | 0, 10);
    target += normal_lpdf(b2 | 0, 10);

    target += inv_gamma_lpdf(sigma | 1, 1);

    for (i in 1:n_sp_conif)
    {
        // Hierarchy
        target += normal_lpdf(beta0[i] | b0[1], sigma_beta0);
        target += normal_lpdf(beta2[i] | b2[1], sigma_beta2);

        // Likelihood conifers
        target += normal_lpdf(volume_m3[ind_start_conif[i]:ind_end_conif[i]] | b0[1] +
            b1[1]*fake_hdn[ind_start_conif[i]:ind_end_conif[i]] +
            b2[1]*fake_slenderness[ind_start_conif[i]:ind_end_conif[i]], sigma);
    }
```

```
    for (i in 1:n_sp_broad)
    {
        // Hierarchy
        target += normal_lpdf(beta0[n_sp_conif + i] | b0[2], 4.2);
        target += normal_lpdf(beta2[n_sp_conif + i] | b2[2], 6.98);

        // Likelihood broadleaves
        target += normal_lpdf(volume_m3[ind_start_broad[i]:ind_end_broad[i]] | b0[2] +
            b1[2]*fake_hdn[ind_start_broad[i]:ind_end_broad[i]] +
            b2[2]*fake_slenderness[ind_start_broad[i]:ind_end_broad[i]], sigma);
    }
}
```

Lindstrom, Mary J, and Douglas M Bates. 1990. "Nonlinear Mixed Effects Models for Repeated Measures Data." *Biometrics* 46 (3): 673. https://doi.org/10.2307/2532087.

Listing 1

```
data {
    // Dimensions and indices
    int N; // Number of individuals
    int lim_broadleaf; // Number of individuals

    // Predictors
    vector [N] fake_hdn;
    vector [N] fake_slenderness;

    // Response variable
    vector[N] volume_m3;
}

parameters {
    // Fixed effects
    vector[2] b0;
    vector[2] b1;
    vector[2] b2;

    // Variance
    real<lower = 0> sigma; // sd residuals
}

model {
    // Priors
    target += normal_lpdf(b0 | 0, 10);
    target += normal_lpdf(b1 | 0, 10);
    target += normal_lpdf(b2 | 0, 10);

    target += inv_gamma_lpdf(sigma | 1, 1);

    target += normal_lpdf(volume_m3[1:(lim_broadleaf - 1)] | b0[1] +
        b1[1]*fake_hdn[1:(lim_broadleaf - 1)] +
        b2[1]*fake_slenderness[1:(lim_broadleaf - 1)], sigma);

    target += normal_lpdf(volume_m3[lim_broadleaf:N] | b0[2] +
        b1[2]*fake_hdn[lim_broadleaf:N] +
        b2[2]*fake_slenderness[lim_broadleaf:N], sigma);
}
```