Sensitivity Analysis

Martin Hinz

2022-11-20

Setup

The model has 2 predefined parameters whose effect we want to check in a sensitivity analysis. First we load the necessary libraries.

```
library(here)
```

```
## here() starts at /home/martin/r_projekte/bayesian.demographic.reconstruction.2022
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(parallel)
library(doParallel)
```

```
## Lade nötiges Paket: foreach
```

```
##
## Attache Paket: 'foreach'
```

```
## Die folgenden Objekte sind maskiert von 'package:purrr':
##
##     accumulate, when
```

```
## Lade nötiges Paket: iterators
```

```
library(nimble)
```

```
## nimble version 0.12.2 is loaded.
## For more information on NIMBLE and a User Manual,
## please visit https://R-nimble.org.
```

```
##
## Attache Paket: 'nimble'
```

```
## Das folgende Objekt ist maskiert 'package:stats':
##
##     simulate
```

```
library(coda)
library(MCMCvis)
```

Then we set up the basic framework for the model run. To do this, we load the input data and prepare it according to the model (for details on these steps please consult the actual analysis):

```
all_proxies <- read.csv(file = normalizePath(
  file.path(here(), "data","preprocessed_data", "all_proxies.csv")
  ),
  row.names = 1)
all_proxies <- all_proxies %>% arrange(desc(age))
all_proxies[,2:5] <- all_proxies[,2:5] %>% scale() %>% diff() %>% rbind(0,.)
model_data <- all_proxies
```

Next, we incorporate the model, making all potentially sensitive parameters accessible from the outside:

```
model_code <- nimbleCode( {
  # ---- Process Model ----

  # Estimate the initial state vector of population abundances
  nEnd ~ dnorm(nSites[nYears] * MeanSiteSize / AreaSwissPlateau, sd=0.5)

  # Autoregressive process for remaining years
  for(t in 2:(nYears)) {
    # The actual number of sites per year
    nSites[t] ~ dpois(lambda[t])

    # limiting the change to a maximum value, estimated in the model
    constraint_lambda_lower[t] ~ dconstraint(
      nSites[t]/nSites[t-1] < (max_growth_rate + 1)
      )
    constraint_lambda_upper[t] ~ dconstraint(
      nSites[t-1]/nSites[t] < (max_growth_rate + 1)
      )
  }

  # ---- Observational Model ----

  # For all but the first year
  for(t in 2:(nYears)) {
    # lambda depends on the number of sites at the previous year, plus
    # changes in relation to the proxies
    log(lambda[t]) <- log(nSites[t-1]) + (
      p[1] * sumcal[t]  +
        p[2] * openness[t] +
        p[3] * aoristic_sum[t] +
        p[4] * dendro[t]
      )
  }
  # ---- Priors ----
  # Relevance of the proxies is estimated as Dirichlet distribution
  p[1:4] ~ ddirch(alpha[1:4])

  # The parameters for the Dirichlet distribution have a weakly informative prior
  for(j in 1:4) {
```

```
    alpha[j] ~ dlnorm(mu_alpha[j],sdlog=a_alpha[j])
    a_alpha[j] ~ dexp(1)
    mu_alpha[j] ~ dlnorm(1,sdlog=0.1)
  }

  # The maximum growth rate has a prior gamma distributed between 0 and 1
  # by adding 1 in the process model, this becomes 1-2[
  max_growth_rate ~ dgamma(shape = 5, scale=0.05)

  # The mean site size
  MeanSiteSize ~ dpois(50)

  # ---- transformed data ----

  # Population density and total population as function of site number
  PopDens[1:(nYears)] <- PopTotal[1:(nYears)] / AreaSwissPlateau
  PopTotal[1:nYears] <- nSites[1:nYears] * MeanSiteSize
})
```

Parameter Sweep

We will run 20 parameterisations with a range whose centre is our proposed value. To do this, we will run the model until convergence. Afterwards, we will compare the results of each run.

We have externalised the function to run the model, we load it first:

```
source(file = normalizePath(file.path(here(), "code", "sensitivity_helpers.R")))
```

Mean Site Size

First, we vary the mean site size:

```
mean_site_size_sweep <- seq(5,100,length.out = 20)

model_constants_df <- data.frame(
  nEnd = rep(5, length(mean_site_size_sweep)),
  nYears = rep(nrow(model_data), length(mean_site_size_sweep)),
  AreaSwissPlateau = rep(12649, length(mean_site_size_sweep)),
  ParamMeanSiteSize = mean_site_size_sweep
  )

model_constants_list <- purrr::transpose(model_constants_df)
```

Next, we can run the actual analysis and save the results. In the current version this is deactivated to speed up the rendering of the representation of the analysis. The data is loaded from a previous run. However, you are welcome to turn it on to run it yourself.

```
sweep_results <- sweep_run(model_constants_list = model_constants_list,
                           this_model_data = model_data,
                           model_code = model_code)

saveRDS(sweep_results, file = normalizePath(
  file.path(here(), "data","preprocessed_data", "sensitivity_meansitesize.RDS"
            ),
```

```
  mustWork = F)
  )
```

Then we reload the results (this is the entry point to avoid having to do the whole calculation all over again):

```
sweep_results <- readRDS(file = normalizePath(
  file.path(here(), "data","preprocessed_data", "sensitivity_meansitesize.RDS"
            ))
  )
```

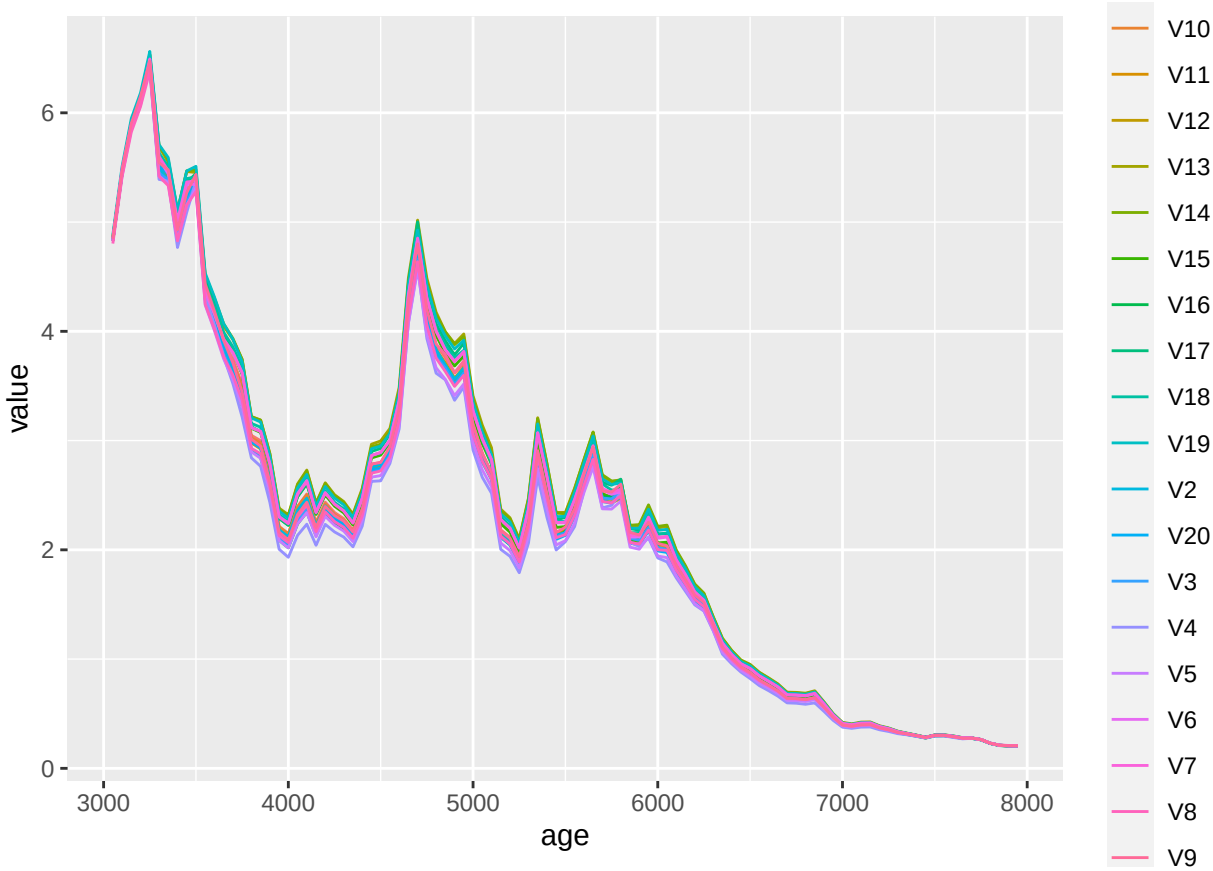
Now we can visualise the results. We will plot the individual mean values on top of each other:

```
res_for_plot <- as.data.frame(sapply(sweep_results, function(x) x$mean))

res_for_plot$age <- model_data$age

res_for_plot <- pivot_longer(res_for_plot, !age)

ggplot(res_for_plot) + geom_line(aes(x = age, y = value, color = name))
```



The specification of a medium settlement size does not have a strong discernible effect on the result of the estimation.

Mean Site Size

Next, we check the dependence on the proposed final population density.

```
nEnd_sweep <- seq(0.5,10,length.out = 20)

model_constants_df <- data.frame(
  nEnd = nEnd_sweep,
  nYears = rep(nrow(model_data), length(nEnd_sweep)),
  AreaSwissPlateau = rep(12649, length(nEnd_sweep)),
  ParamMeanSiteSize = rep(50, length(nEnd_sweep))
  )

model_constants_list <- purrr::transpose(model_constants_df)
```

Here, too, the actual run is currently deactivated in order to speed up the rendering. However, you are welcome to perform it yourself.

```
sweep_results <- sweep_run(model_constants_list = model_constants_list,
                           this_model_data = model_data,
                           model_code = model_code)

saveRDS(sweep_results, file = normalizePath(
  file.path(here(), "data","preprocessed_data", "sensitivity_nend.RDS"
            ),
  mustWork = F)
  )
```

And now (especially if the analysis has not gone through), the results will load in again:

```
sweep_results <- readRDS(file = normalizePath(
  file.path(here(), "data","preprocessed_data", "sensitivity_nend.RDS"
            ))
  )
```

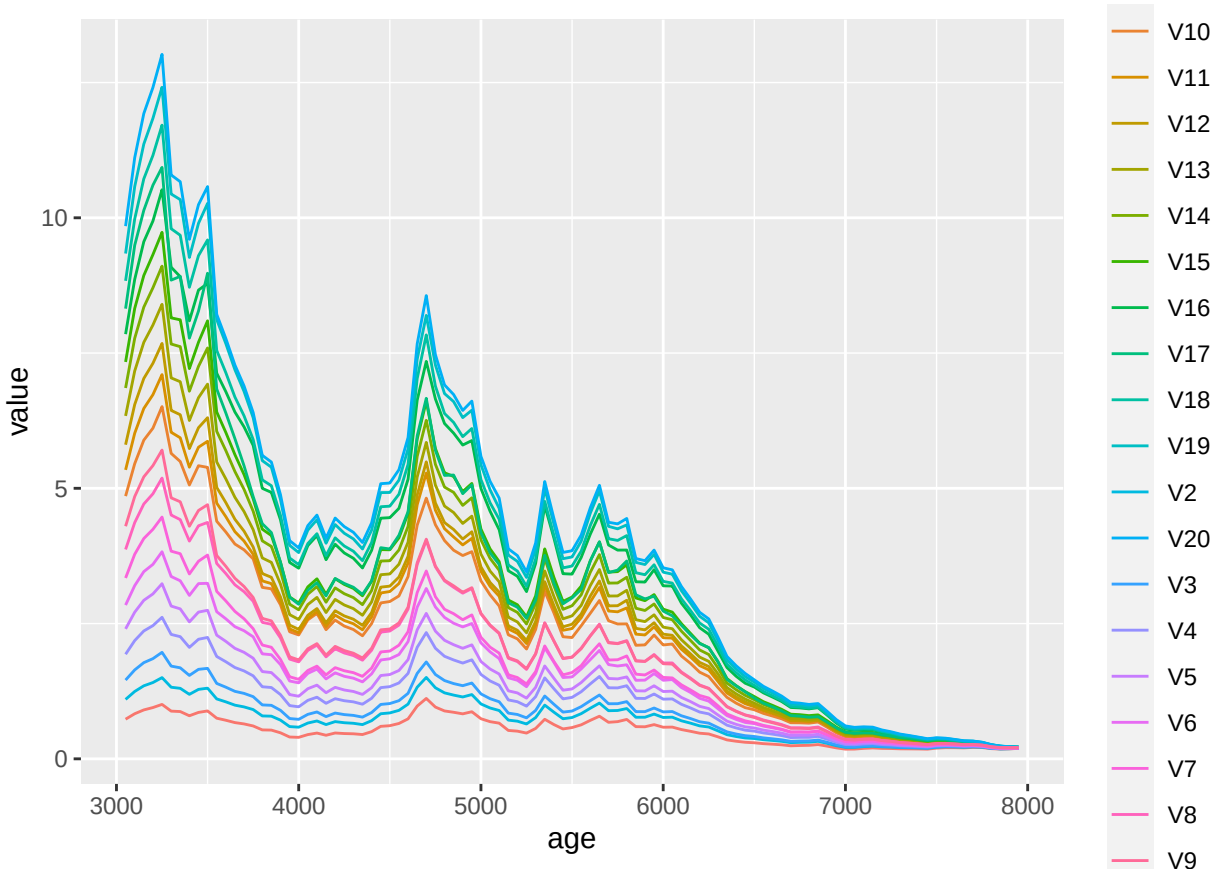
Now we can visualise the results. We will plot the individual mean values on top of each other:

```
res_for_plot <- as.data.frame(sapply(sweep_results, function(x) x$mean))

res_for_plot$age <- model_data$age

res_for_plot <- pivot_longer(res_for_plot, !age)

ggplot(res_for_plot) + geom_line(aes(x = age, y = value, color = name))
```



As expected, this parameter has a decisive influence on the result.