Sequential Calibration of the distributed model

Willem Vervoort & Rafael Navas

2018-07-12

Contents

1 Introduction

Packages needed

```r
require(tidyverse)
require(lubridate)
require(hydromad)
```

This document gives a rough outline of how you might do sequential multi-objective calibration using the distributed GR4J model. The focus is on the principle, not on the coding of optimisation.

The idea of the sequential optimisation is that you first calibrate ony the parameters for the most upstream subbasin (Subbasin1) , then fix the parameters for this subbasin for the next Subbasin calibration, etc.

2 Loading the script, data and model

First load the script with the model.

```r
source("Rcode_IMFIA_course2018/GR4J@distrv2.R")
```

This loads six functions in your environment. The first is the actual model function, `GR4JSubBasins.run()`, the other is a specific objective function, `DistGR4J_objfun()`, which can be used with `optim()` to minimise the required internal objective function (such as "r.squared").

Finally there are the function for muskingum routing, `muskingum()`, and three utility functions: 1) to rewrite the parameters, `rewrite_par`; 2) to plot the results, `plot_results()`; and 3) to calculate statistics `stats_fun()`

2.1 The data

Load the Rdata file that includes all the data and all the parameters for the model.

```r
load("data/SL_distdata.RDATA")
```

Select a window of the data, because it is a zoo data frame, we can use `window()` on the data.

```r
# Use data for 2000 - 2005
Data_in <- window(SL_distdata,
                       start = "2000-01-01",
                       end ="2005-12-31")
```

2.2 Optimisation of the model

To optimise the model, we need to use `optim()`, as the model is not fully integrated in hydromad. So this simply follows the linear regression optimisation example.

This means we need to give initial guesses of the parameters and construct the model input as a list

3 Subbasin 1

3.1 Create the input for the optimisation function

3.1.1 step 1

Define the initial guesses of the parameters to optimise This is the vector x in model optimisation. In this case we only want to fit the GR4J parameter for the first Subbasin

You can just type numbers, but I am using parts of the existing lists to create the vector. They have to be in the right order.

```r
# initial guesses, using the SubX.par in All_par. and
# reachX.par in All_reach_par, where X is a number
x <- c(All_par$Sub1.par[2:5])
```

3.1.2 step 2

Create the vector for par_in, which is all the parameters in the model, again using the parameter lists that we loaded. We can reuse this for Subbasin 2 and 3

```r
# all the parameters of the model
par_in <- c(All_par$Sub1.par[1:5], All_par$Sub2.par[1:5],
  All_par$Sub3.par[1:5], All_reach_par$reach1.par,
  All_reach_par$reach2.par)
```

3.1.3 step 3

Identify which positions in par_in relate to x, so which parameters in the whole series in par_in are actually calibrated.

```
# define which parameters to calibrate
# We want to
Fit_these <- c(2:5)
```

3.1.4 step 4

Finally, we need to construct a list which defines the remaining model input We can reuse this for S2 and S3

```
# Use data for 2000 - 2005
# created earlier as Data_in
# define the model_input (optional, as these are the defaults)
model_input_in <- list(sb = 3, order = c(1,2,3),
                        sbnames = c("PasoTroncos",
                        "FrayMarcos",
                        "PasoPache"))
```

3.2 fit the model using optim()

Now you can run the objective function using optim(). The key here is definition of the weights.

```
Fit_SL <- optim(par = x, # values of the parameters to fit
    DistGR4J_objfun, # function to fit
    calibrate_on = Fit_these, # which positions of parameters
    parM = par_in, # all parameters
    Data = Data_in, # The input data (Q,P,E)
    model_input = model_input_in, # (opt) rest model input
      objective = "r.squared", # objective function(optional)
    weights = c(1,0,0)) # weights

Fit_SL$par
```

```
##          x1          x2          x3          x4
## 1100.319322   -2.369142   42.406006    1.783930
```

```
Fit_SL$value
```

```
## [1] -0.2448957
```

4 Subbasin2

We now want to fix the parameters for subbasin 1 and use this to calibrate the parameters in subbasin 2 and the routing of reach 1.

First we need to rewrite the parameters, I had to rewrite the function

```
# insert fitted values into parameters
# using rewrite_pars() Note the change in definition
# this creates a list of reach and sub parameters
```

```
new_pars <- rewrite_pars(All_par,changed_par = Fit_SL$par,
                          sb = 1, which = 1)
```

4.1 Create the input for the optimisation function

4.1.1 step 1

Define the initial guesses of the parameters to optimise This is the vector x in model optimisation

```
# only subbasin 2 and reach 1
x <- c(All_par$Sub2.par[2:5], All_reach_par$reach1.par)
```

skipping step 2

4.1.2 step 3

Identify which positions in par_in relate to x, so which parameters in the whole series in par_in are actually calibrated. Only subbasin 2 and reach 1

```
# define which parameters to calibrate
# We want to
Fit_these <- c(7:10,16,17)
```

skipping step 4

4.2 fit the model using optim()

Now you can run the objective function using optim(), change weights

```
Fit_SL <- optim(par = x, # values of the parameters to fit
    DistGR4J_objfun, # function to fit
    calibrate_on = Fit_these, # which positions of parameters
    parM = par_in, # all parameters
    Data = Data_in, # The input data (Q,P,E)
    model_input = model_input_in, # (opt) rest model input
      objective = "r.squared", # objective function(optional)
    weights = c(0,1,0)) # weights

Fit_SL$par
```

```
##          x1          x2          x3          x4           k           x
## 208.3610578  -0.8210676  24.2713904   2.5283724   0.3917711   0.4511719
```

```
Fit_SL$value
```

```
## [1] -0.277055
```

5 Subbasin3

We now want to fix the parameters for subbasin 1 & 2 and use this to calibrate the parameters in subbasin 3 and the routing of reach 2.

First rewrite the parameters

```r
# insert fitted values into parameters
# using rewrite_pars()
new_pars <- rewrite_pars(All_par,All_reach_par,
                         changed_par = Fit_SL$par,
                         sb = 2, which = 2)
```

5.1 Create the input for the optimisation function

5.1.1 step 1

Define the initial guesses of the parameters to optimise This is the vector x in model optimisation

```r
# only subbasin 2 and reach 1
x <- c(All_par$Sub3.par[2:5], All_reach_par$reach2.par)
```

skipping step 2

5.1.2 step 3

Identify which positions in par_in relate to x, so which parameters in the whole series in par_in are actually calibrated. Only subbasin 2 and reach 1

```r
# define which parameters to calibrate
# We want to
Fit_these <- c(12:15,18,19)
```

skipping step 4

5.2 fit the model using optim()

Now you can run the objective function using optim(), change weights

```r
Fit_SL <- optim(par = x, # values of the parameters to fit
    DistGR4J_objfun, # function to fit
    calibrate_on = Fit_these, # which positions of parameters
    parM = par_in, # all parameters
    Data = Data_in, # The input data (Q,P,E)
    model_input = model_input_in, # (opt) rest model input
      objective = "r.squared", # objective function(optional)
    weights = c(0,0,1)) # weights

Fit_SL$par
```

```
##            x1            x2            x3            x4             k
## 189.083246239  -2.444682252  59.015021425   3.664749220   0.001216663
##             x
##   0.499655591
```

```r
Fit_SL$value
```

```
## [1] -0.2755332
```

6 Calculating overall performance

6.0.1 step 1

Insert parameters back into the model, we can do this easier using the utility function `rewrite_pars()`, which takes as inputs the old parameters and the new fitted parameters.

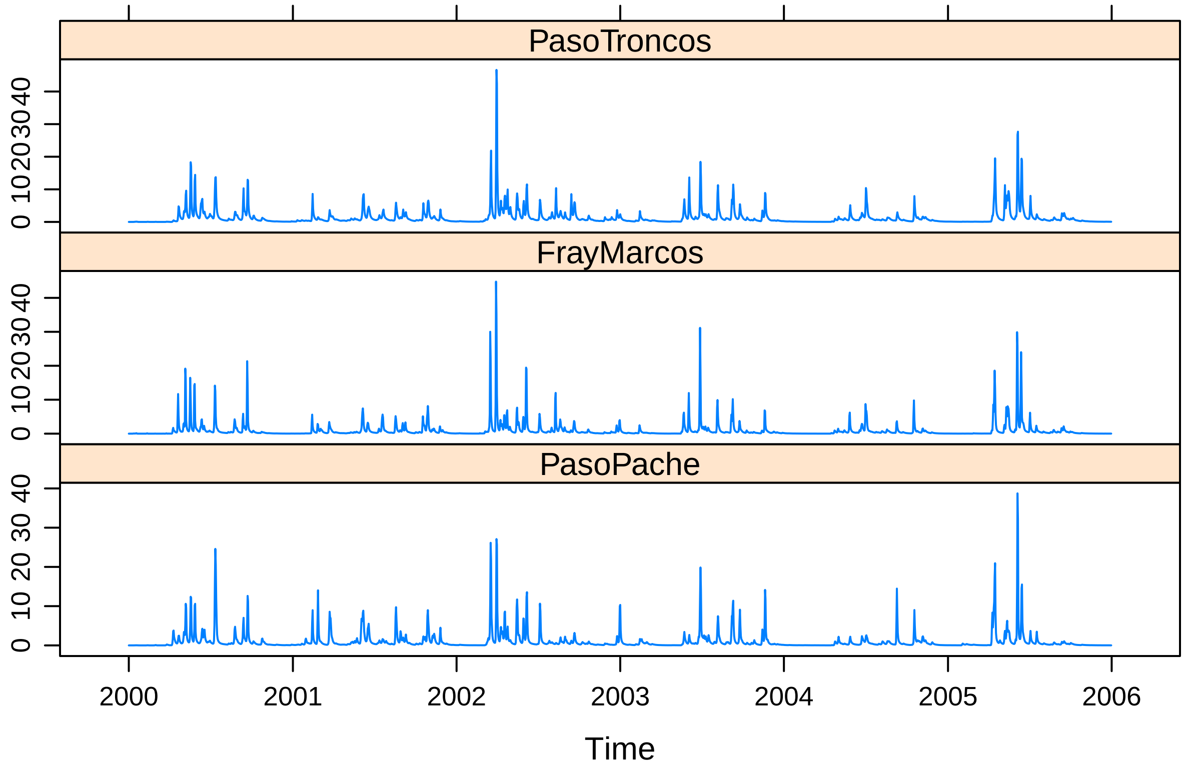
```r
# insert fitted values into parameters
# using rewrite_pars()
# this creates a list of reach and sub parameters
new_pars <- rewrite_pars(All_par,All_reach_par,
                          changed_par = Fit_SL$par,
                          sb = 1, which = 3)
```

6.0.2 step 2

We can now rerun the model with the new parameters, which means we make a prediction. In this case we repredict the data, but you could use the same function to predict new data for a validation.

```r
Fitted_SL <- GR4JSubBasins.run(sb=3,
                          order = c(1,2,3),
                          Data = Data_in,
                          spar = new_pars$sub,
                          rpar = new_pars$reach,
                          sbnames = c("PasoTroncos",
                                      "FrayMarcos",
                                      "PasoPache"))

xyplot(Fitted_SL)
```

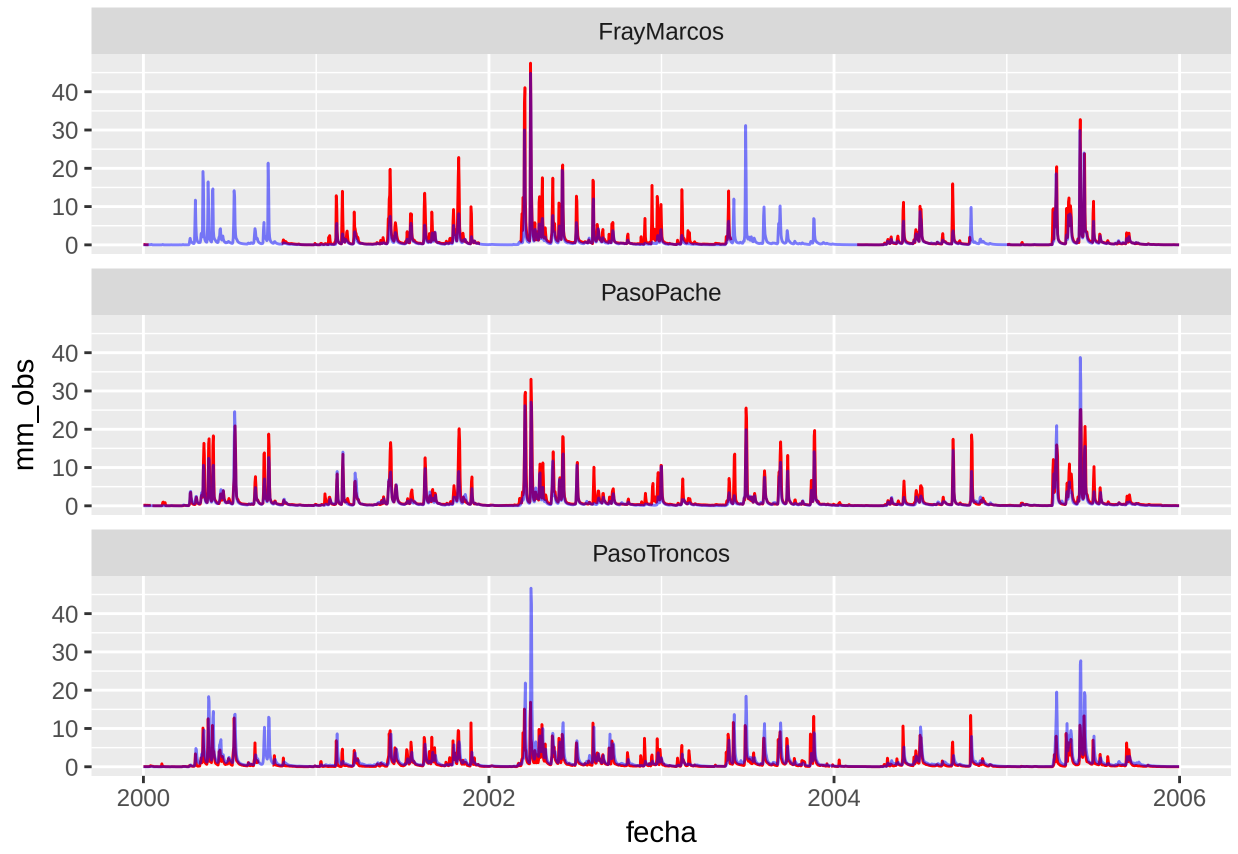


This plot is just the predicted values, we want to know how well it predicts the observed values. For this we want to make a plot and calculate some statistics to show how good the fit is.

6.0.3 step 3

I am here using ggplot and create a dataframe that includes both the fitted and the observed data for each subcatchment. We can make this plot easily using the utility `plot_results()` function. The output of this function is a data frame with the observed and fitted results.

```
# plot using `plot_results()`
results <- plot_results(Fitted_SL,Data_in)
```



```r
# this creates a data frame with observed and fitted Q
```

6.0.4 step 4

We can now calculate the statistics using the output of the last function, and the utility function `stats_fun()`

```r
# using stats_fun on results
stats_fun(results, decimal=2)
```

```
## # A tibble: 3 x 5
##      Subbasin rel.bias r.squared r.sq.sqrt r.sq.log
##         <chr>    <dbl>     <dbl>     <dbl>    <dbl>
## 1  FrayMarcos    -0.38      0.74      0.77     0.78
## 2   PasoPache    -0.28      0.78      0.83     0.81
## 3 PasoTroncos     0.33     -0.38      0.48     0.68
```