Testing constrained optimisation

Willem Vervoort

2017-11-01

```
# root dir
knitr::opts_knit$set(root.dir = "C:/Users/rver4657/ownCloud/working/Constr_optim")
knitr::opts_chunk$set(echo = TRUE)
# LOAD REQUIRED PACKAGES
library(tidyverse)
library(zoo)
library(hydromad)
library(lattice)
```

testing constrained optimisation in hydromad

The idea is to include a constraint on the normal optimisation objective function (such as NSE or RMSE). The constraint needs to be that the individual error (pointwise error) should not be $> \beta$, β is a small number that will be optimised. In essence this is the same as a regularisation, but in this case we use the β as a constraint.

Hydromad standard fit

Load the hydromad Cotter data from the hydromad satellite project and fit GR4J using simple optimisation

```
# use the satellite hydromad example data
load("C:/Users/rver4657/ownCloud/working/SatelliteHydromad/Examples/Cotter.rdata")
load("C:/Users/rver4657/ownCloud/working/SatelliteHydromad/Examples/CotterMODISET.rdata")
#data(Cotter)


data.cal <- window(Cotter, start = "2000-01-01",
                   end = "2005-12-31")

# Define the model, important to define return_state=T
Cotter_mod <- hydromad(DATA=data.cal,
                       sma = "gr4j", routing = "gr4jrouting",
                       x1 = c(100,1500), x2 = c(-30,20),
                       x3 = c(5,500),
                       x4 = c(0.5,10), etmult=c(0.01,0.5),
                       return_state=TRUE)
# Fit traditional fit
# Using Optim algorithm for fitting
Cotter_fit_o<- fitByOptim(Cotter_mod,
                          objective=~hmadstat("r.squared")(Q,X))
# Extract the coefficients and the summary
summary(Cotter_fit_o)
```

```
##
```

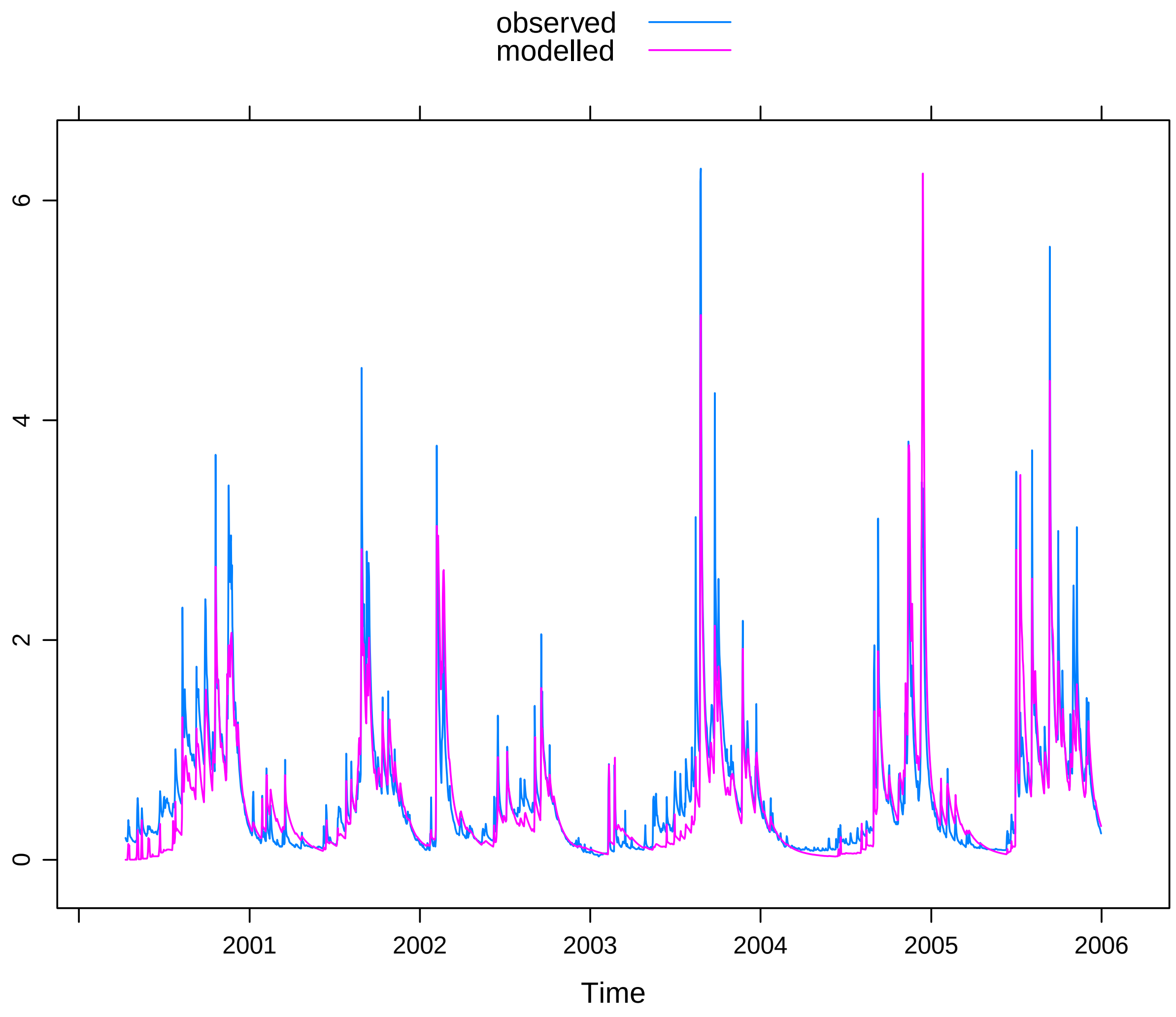


Figure 1: Standard hydromad fit using `FitByOptim()` on the Cotter data

```
## Call:
## hydromad(DATA = data.cal, x1 = 196.776, x2 = -19.5731, x3 = 250.954,
##     x4 = 1.18792, etmult = 0.0722753, return_state = TRUE, sma = "gr4j",
##     routing = "gr4jrouting")
##
## Time steps: 2092 (0 missing).
## Runoff ratio (Q/P): (0.5792 / 2.365) = 0.2449
## rel bias: -0.07403
## r squared: 0.7796
## r sq sqrt: 0.7763
## r sq log: 0.7322
##
## For definitions see ?hydromad.stats
```

```
xyplot(Cotter_fit_o)
```

A new objective function

Define a new objective function that includes the constraint, refit the Cotter test data, check against standard fit.

defining the objective function

This might be the trickiest bit, as this requires introducing a parameter β, but also calibrate on this. Originally I wanted to rewrite the hydromad optimisation function, but I have started much easier by defining my own function that can be optimised using the standard optim function

```
objfun <- function(par, in_data, mod, ...) {
  # par is vector of initial values of parameters to be optimised
  # this includes beta
  # in_data are the input data
  # mod is a hydromad model
  mod_run <- update(mod, x1 = par[1], x2 = par[2], x3 = par[3],
                    x4 = par[4], etmult=par[5])
  #browser()
  error <- fitted(mod_run) - in_data$Q

  adj_error <- abs(abs(error) - par[6])
  SSE <- sqrt(sum((error)^2,na.rm=T))/length(error)


  ValToMinimise <- SSE + sum((adj_error),na.rm=T)
  return(ValToMinimise)
}

test <- objfun(par=c(600, 0, 100, 2, 0.15, 5),
               mod=Cotter_mod,in_data=data.cal)
# this just tests whether the function can be loaded correctly
print(test) # the objective function value
```

```
## [1] 10001.05
```

Calibrating the model

Now write an optimisation routine that includes this objective function and uses `optim()`. I am using the L-BFGS-B routine so I can constrain the upper and lower boundaries of the parameters as well. This is different from the constrained optimisation component.

```
# lower boundaries
# x1, x2, x3, x4, etmult, beta
lb <- c(100, -30, 5, 0.6 , 0.01, 0.1)
# upper boundaries
up <- c(1500, 20, 500, 10, 0.5, 10)

# initial values
par_in <- c(600, 0.1, 100, 2, 0.15, 5)
```

Note that β is also fitted.

Now fit the the actual model with the `optim()` routine.

```r
# use optim and L-BFGS-B
sol <- optim(par_in, objfun,method="L-BFGS-B",
             in_data=data.cal, mod=Cotter_mod,
             lower = lb, upper = up)
# actual fit result for the parameters
sol$par
```

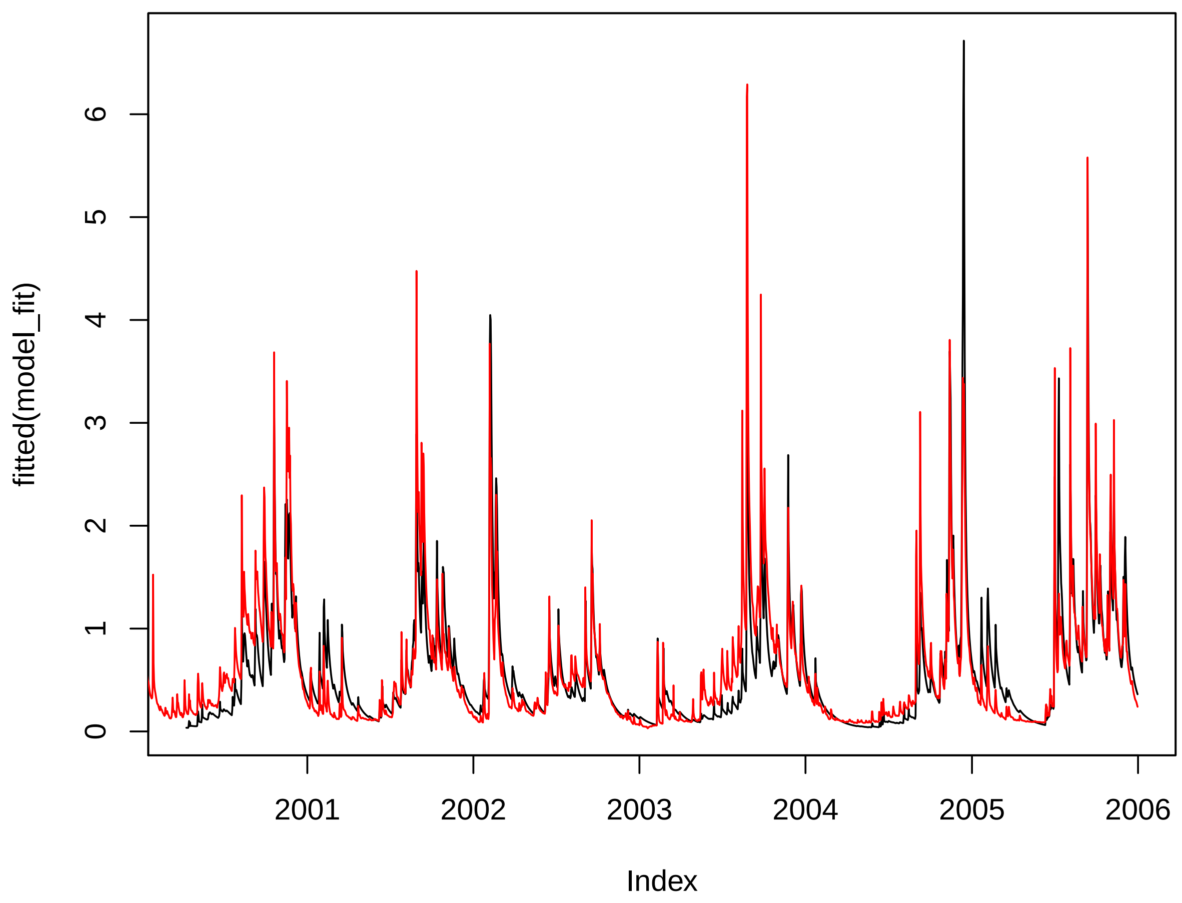
```
## [1] 599.98354640  -4.62920876  99.72281593   1.19719473   0.09730208
## [6]   0.11271805
```

Here the parameter β equals 0.1127.

Inspect the results

We can now inspect the results from the fit by looking at `sol`. But we also need to update the model and calculate the statistics

```r
# apply solution to model and plot
model_fit <- update(Cotter_mod, x1 = sol$par[1], x2 = sol$par[2], x3 = sol$par[3],
                    x4 = sol$par[4], etmult=sol$par[5])
# make a plot
plot(fitted(model_fit))
lines(data.cal$Q,col="red")
```



```r
# calculate statistics
hmadstat("rel.bias")(data.cal$Q[101:length(data.cal$Q)],
                      fitted(model_fit))
```

```
## [1] -0.02415582
```

```r
hmadstat("r.squared")(data.cal$Q[101:length(data.cal$Q)],
                       fitted(model_fit))
```

```
## [1] 0.7457519
```

```r
hmadstat("r.sq.sqrt")(data.cal$Q[101:length(data.cal$Q)],
                       fitted(model_fit))
```

```
## [1] 0.7731539
```

```r
hmadstat("r.sq.log")(data.cal$Q[101:length(data.cal$Q)],
                      fitted(model_fit))
```

```
## [1] 0.7492858
```

This is slightly better in low flow fit and similar in r sq sqrt and has a lower bias then the original fit. That would be correct given the way we fit β, which will take away the focus from the high flows that the standard fit is causing.

Including Satellite data

The next test would be to include the satellite data and to check if the fit on the satellite data improves relative to the hydromad satellite fit developed by Joseph Guillaume and myself: SatelliteHydromad.

Satellite hydromad fit

We start with simply using the functions in the satellite hydromad fitting example as we already have loaded the data. Here I am using the `SCEOptim()` function, which is slightly different from the basic `optim()`, but should get a fairly similar result.

First step is to prepare the data and load the functions from the satellite hydromad project.

```r
# load functions from Satellite calibration
setwd("C:/Users/rver4657/ownCloud/working/SatelliteHydromad/Examples")
source("setup.R")
```

```
## Loading required package: raster

## Loading required package: sp

##
## Attaching package: 'raster'

## The following object is masked from 'package:dplyr':
##
##     select

## The following object is masked from 'package:tidyr':
##
##     extract

## Loading required package: maptools

## Checking rgeos availability: TRUE

## Loading required package: rgeos

## rgeos version: 0.3-25, (SVN revision 555)
##  GEOS runtime version: 3.6.1-CAPI-1.10.1 r0
##  Linking to sp version: 1.2-5
##  Polygon checking: TRUE

## Loading required package: rgdal

## rgdal: version: 1.2-13, (SVN revision 686)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.0, released 2017/04/28
##  Path to GDAL shared files: C:/Users/rver4657/Documents/R/win-library/3.4/rgdal/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: C:/Users/rver4657/Documents/R/win-library/3.4/rgdal/proj
##  Linking to sp version: 1.2-5
```

```r
# using ETa.merge()
Flow.Modis.zoo <- ETa.merge(Flowdata=Cotter,ETdata=Cot_MODISET)
```

```r
# remake the calibration data
data.modis.cal <- window(Flow.Modis.zoo, start = "2000-01-01",end = "2005-12-31")
```

We can now fit the model following the example in the github repository, and inspect the results and compare to the older results.

```r
# Because we have rebuilt data.cal, redefine the model
Cotter_mod_M <- hydromad(DATA=data.modis.cal,
                         sma = "gr4j", routing = "gr4jrouting",
                         x1 = c(100,1500), x2 = c(-30,20),
                         x3 = c(5,500), x4 = c(0.5,10),
                         etmult=c(0.01,0.5),
                         return_state=TRUE)

Cotter_fit_q_ET <- FitMODbySCE(mod=Cotter_mod_M, FIT_Q=T,
                    FIT_Q_ET = T, FIT_ET = T)
summary(Cotter_fit_q_ET, items = c("rel.bias", "r.squared","r.sq.sqrt", "r.sq.log"))
```

```
##                   rel.bias r.squared r.sq.sqrt  r.sq.log
## Fit only Q   -0.03492301 0.7778536 0.7794961 0.7388268
## Fit only ET   0.04767446 0.7368913 0.7406832 0.6983755
## Fit ET and Q -0.03637840 0.7763753 0.7743296 0.7333664
```

```r
# essentially the same solution as earlier, ET calibration has no effect
# plot
xyplot(Cotter_fit_q_ET)
```

```
## Warning in formals(fun): argument is not a function

## Warning in formals(fun): argument is not a function
```

These results and statistics suggests there is little difference between fitting with ET and fitting without ET for this catchment, as I have earlier shown with the SWAT results. The question is whether this will change with the constrained optimisation.

But that is only the fit on Q that is shown, and the statistics calculated. We actually want to know the fit on ET for the last fit in the table. There is a little trick here, as the ET data need to be aggregated back to the 8 day periods of the original data. In other words, we need a function that calculates the NSE stat, but on the aggregated data.

Using the `plot.ET()` function these zero values are automatically removed.

```r
aET_obs <- aggregate(data.modis.cal$aET,
                     list(date=data.modis.cal$et.period),sum)
ET_pred <- aggregate(Cotter_fit_q_ET[["Fit ET and Q"]]$U$ET,
                     list(date=data.modis.cal$et.period),
                      sum)

nseStat(coredata(aET_obs),coredata(ET_pred))
```

```
## [1] -0.2716778
```

```r
# now make a plot using the plot.ET() function
plot.ET(data.modis.cal,Cotter_fit_q_ET[["Fit ET and Q"]])
```

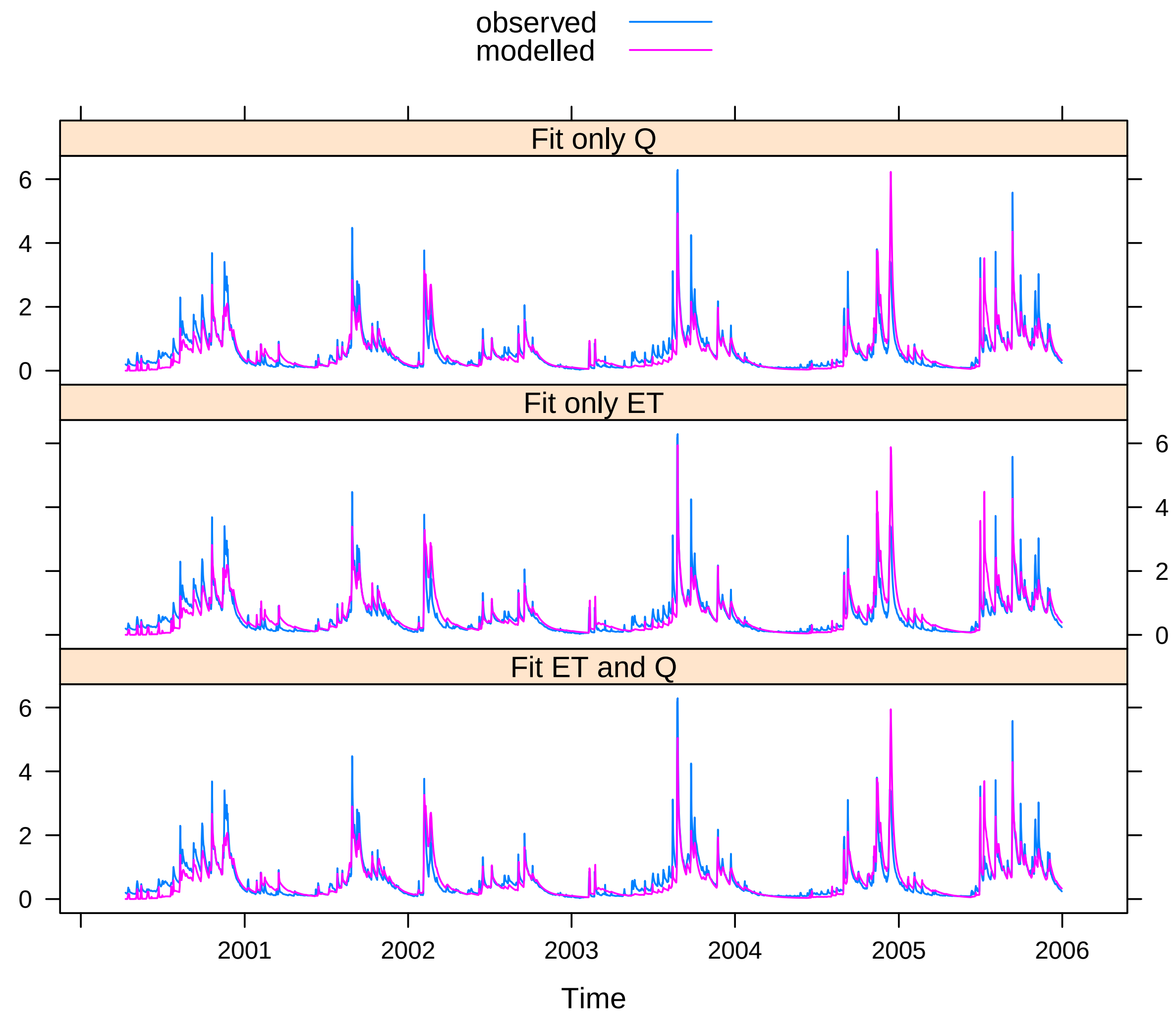


Figure 2: Standard hydromad satellite data fit using FitBySCE() on Cotter and MODIS data

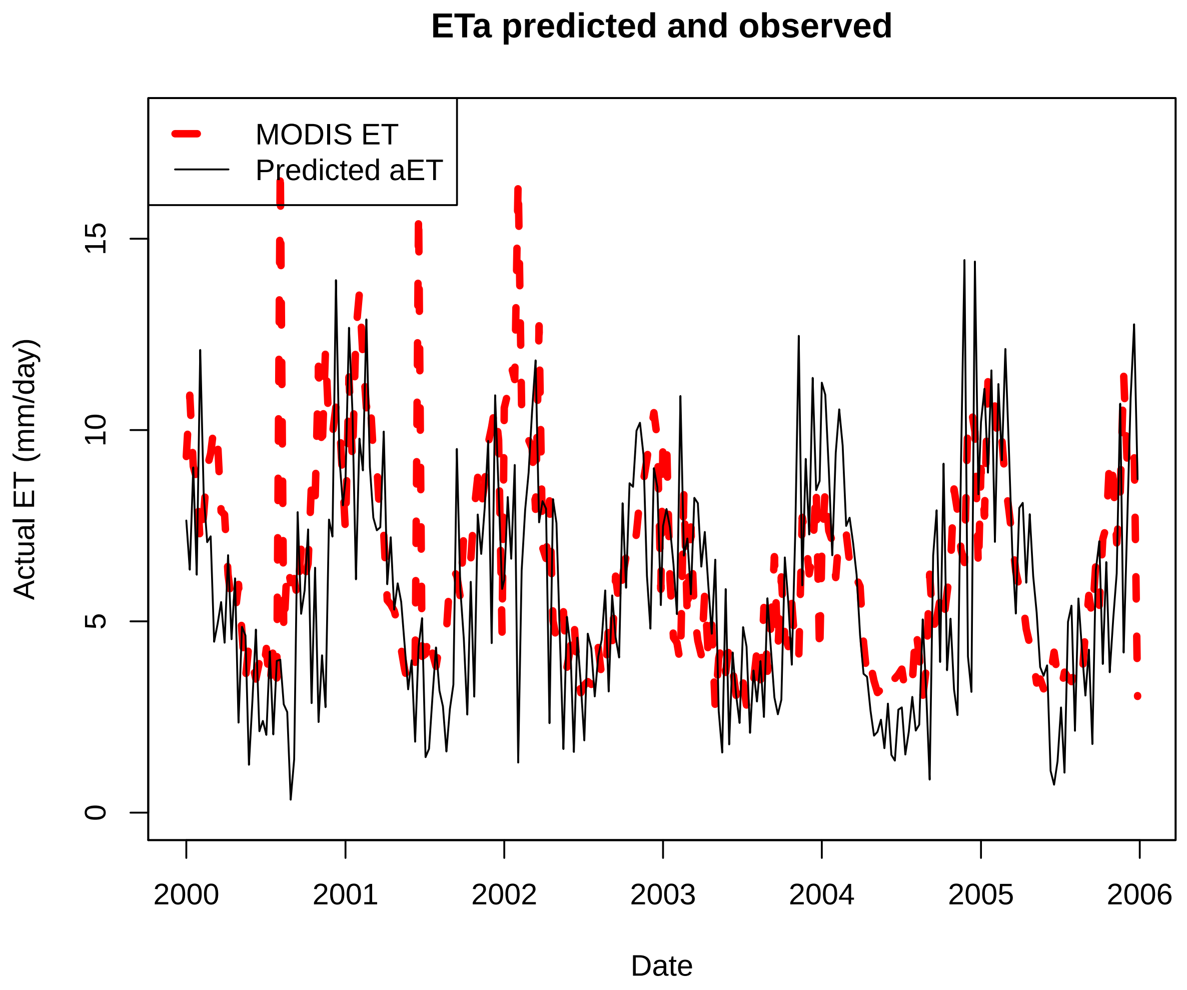


Figure 3: Fit of the model to the MODIS ET data using the standard hydromad fit

fitting the constrained optimisation

For this we need to rewrite the whole optimisation function to include ET. I will be making use of some of the functions from the satellite hydromad. The first step is to once again define a function to fit with `optim()`. For this we need to have the objective function being a function of beta and the error, similar to the example above with just the Q data.

```r
objfun <- function(par, in_data, mod, ...) {
  # par is vector of initial values of parameters to be optimised
  # this includes beta
  # in_data are the input data, which includes MODIS ET
  # mod is a hydromad model
  mod_run <- update(mod, x1 = par[1], x2 = par[2], x3 = par[3],
                    x4 = par[4], etmult=par[5])
  #browser()
  # Q objective functions
  error_Q <- fitted(mod_run) - in_data$Q

  adj_error_Q <- abs(abs(error_Q) - par[6])
  SSE_Q <- sqrt(sum((error_Q)^2,na.rm=T))/length(error_Q)
  # ET objective function
  # aggregate the ET data to the 8 day totals
  aET_fin <- aggregate(in_data$aET,list(date=in_data$et.period),sum)
  ET_fin <- aggregate(mod_run$U$ET,
                      list(date=in_data$et.period),
                      sum)
  #browser()
  error_ET <- coredata(ET_fin) - coredata(aET_fin)

  adj_error_ET <- abs(abs(error_ET) - par[6])
  SSE_ET <- sqrt(sum((error_ET)^2,na.rm=T))/length(error_ET)

  # initially set w = 0.5
  w <- 0.5 # can we in some way set this to SSE_ET/SSE_Q?

  ValToMinimise <- w*(SSE_Q + sum((adj_error_Q),na.rm=T)) +
                   (1-w)*(SSE_ET + sum((adj_error_ET),na.rm=T))
  return(ValToMinimise)
}

test_QET <- objfun(par=c(600, 0, 100, 2, 0.15, 5),
               mod=Cotter_mod_M,in_data=data.modis.cal)
# again, just test if the function is loaded correctly
```

Using the boundaries already defined earlier, the model can now be fitted to both ET and Q using constrained optimisation to both.

```r
# use optim and L-BFGS-B
sol <- optim(par_in, objfun,method="L-BFGS-B",
             in_data=data.modis.cal, mod=Cotter_mod_M,
             lower = lb, upper = up)
# sol$par should be the results
sol$par
```

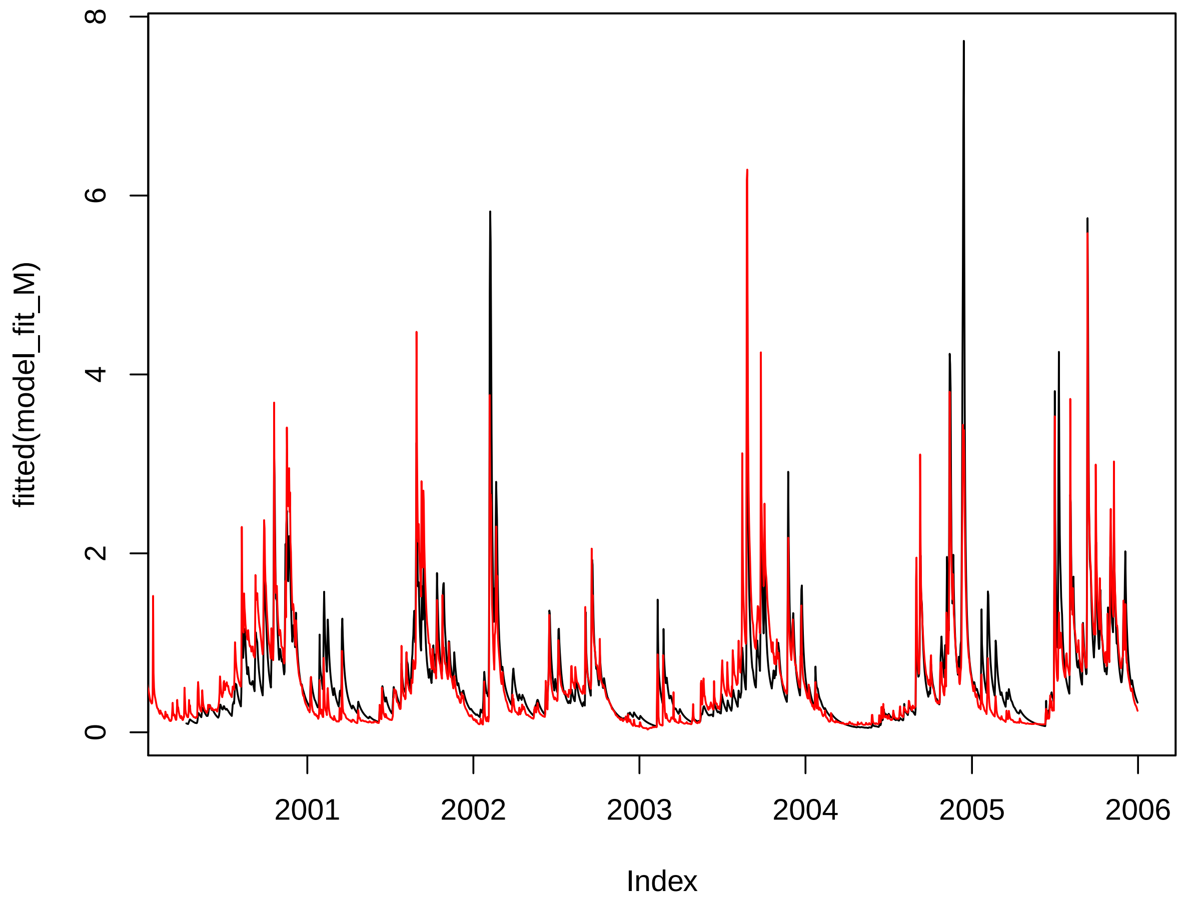
```
## [1] 600.03491317  -9.38021382 100.05863070   1.27690078   0.06301507
## [6]    0.13254386
```

Here the parameter β equals 0.1325.

Inspect the ET fit results

Similar to the earlier calibration the model needs to be updated to calculate the statistics.

```r
# apply solution to model and plot
model_fit_M <- update(Cotter_mod_M, x1 = sol$par[1], x2 = sol$par[2], x3 = sol$par[3],
                      x4 = sol$par[4], etmult=sol$par[5])
# make a plot
plot(fitted(model_fit_M))
lines(data.modis.cal$Q,col="red")
```



```r
# calculate statistics
nseStat(data.modis.cal$Q,fitted(model_fit_M))
```

```
## [1] 0.6937957
```

```
hmadstat("rel.bias")(data.modis.cal$Q[101:length(data.modis.cal$Q)],
                     fitted(model_fit_M))
```

```
## [1] 0.05661371
```

```
hmadstat("r.squared")(data.modis.cal$Q[101:length(data.modis.cal$Q)],
                      fitted(model_fit_M))
```

```
## [1] 0.6937957
```

```
hmadstat("r.sq.sqrt")(data.modis.cal$Q[101:length(data.modis.cal$Q)],
                      fitted(model_fit_M))
```

```
## [1] 0.7643437
```

```
hmadstat("r.sq.log")(data.modis.cal$Q[101:length(data.modis.cal$Q)],
                     fitted(model_fit_M))
```

```
## [1] 0.7582866
```

This suggests the actual calibration has again reduced the influence on the peaks and the calibration on Q is reduced compared to calibration on Q alone. And compared to the unconstrained calibration, the calibration on Q has decreased, but is still similar in the low and overall flow (`r.sq.log` and `r.sq.sqrt`).

We can also recalculate the fit on ET using the model output. Again, I am removing the zero values to calculate the correct NSE for the calibration on ET.

```
# summarise ET to 8 day values
aET_obs <- aggregate(data.modis.cal$aET,
                     list(date=data.modis.cal$et.period),sum)
ET_pred <- aggregate(model_fit_M$U$ET,
                     list(date=data.modis.cal$et.period),
                      sum)

nseStat(coredata(aET_obs),coredata(ET_pred))
```

```
## [1] -0.3463939
```

```
plot.ET(caldata=data.modis.cal,model_fit_M)
```

This shows again improved the low flow and general flow calibration, reduced the peaks and surprisingly a reduced ET calibration. This could be to the value of w that I assigned, but some testing with this didn't seem to make any difference.

It could be because β should be different for Q and ET, and therefore I need to possibly calibrate on two different β values.

Finally, the model is not spatial, so it is not really taking advantage of the ET data features. This needs to be further tested with SWAT.

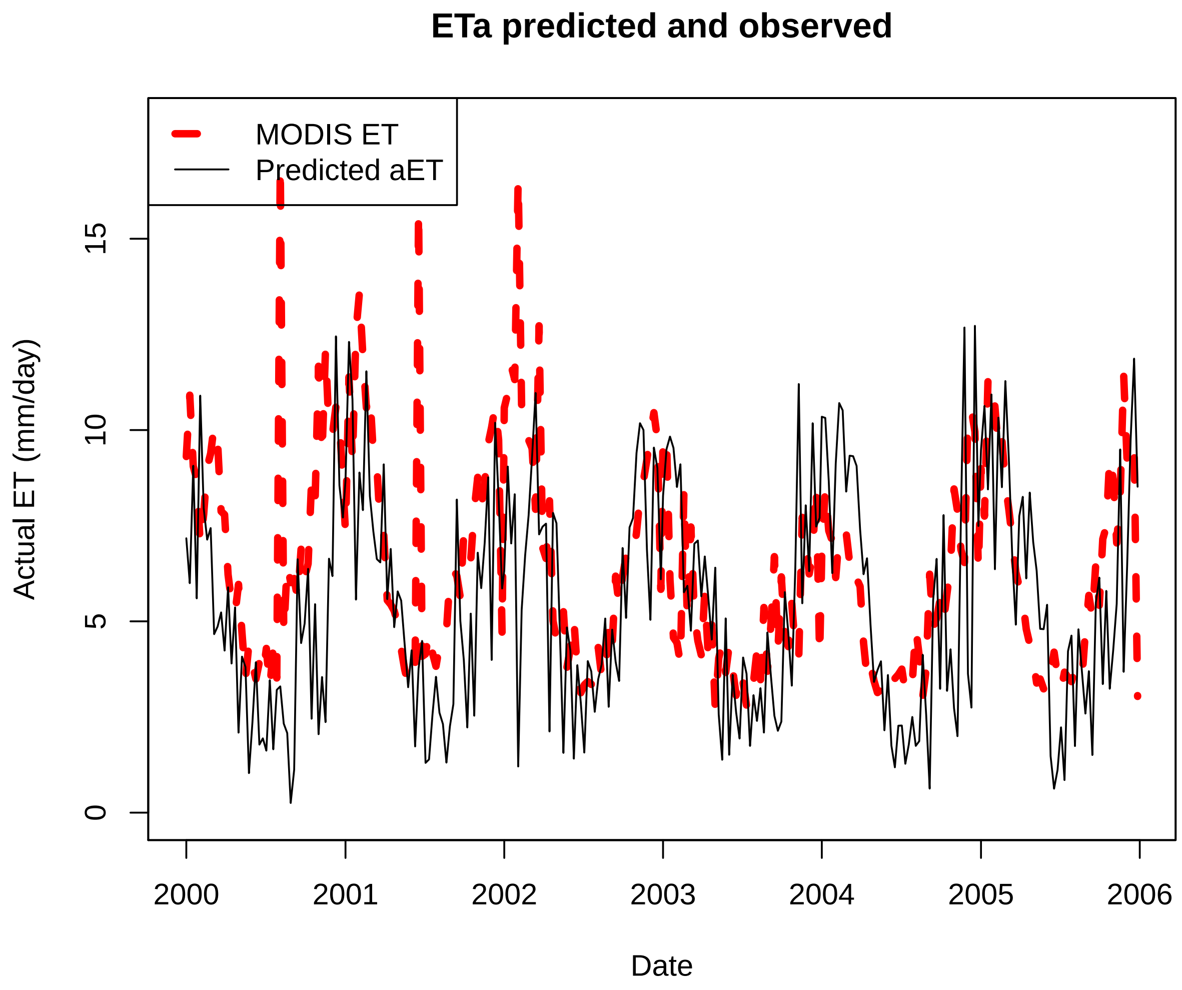


Figure 4: Fit of the model to the MODIS ET data using the constrained optimisation