Testing constrained optimisation part 2

Willem Vervoort

2017-11-07

```r
# 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)
library(nloptr)
library(epiR)
```

testing constrained optimisation using hydromad

This is part 2, that uses the `nloptr` package, rather than simply using optim() as in the first attempt. The package `nloptr` is an implementation of the NLOPT routine.
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

```r
# 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.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)
```

```
##
## Call:
## hydromad(DATA = data.cal, x1 = 196.082, x2 = -19.5597, x3 = 251.615,
```

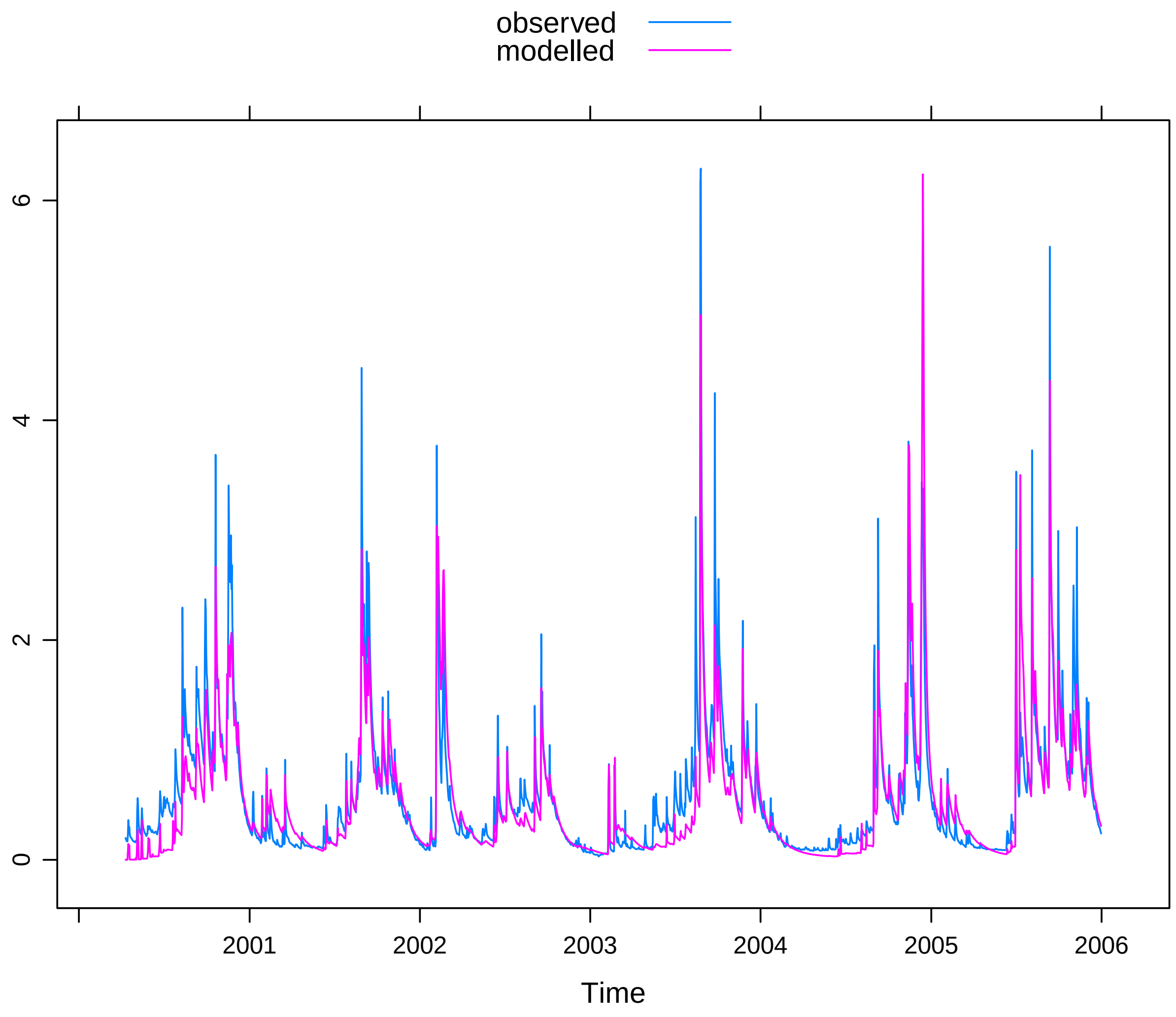

Figure 1: Standard hydromad fit using `FitByOptim()` on the Cotter data

```
##     x4 = 1.18699, etmult = 0.0724253, return_state = TRUE, sma = "gr4j",
##     routing = "gr4jrouting")
##
## Time steps: 2092 (0 missing).
## Runoff ratio (Q/P): (0.5792 / 2.365) = 0.2449
## rel bias: -0.0739
## r squared: 0.7796
## r sq sqrt: 0.7759
## r sq log: 0.7315
##
## For definitions see ?hydromad.stats
```

```
xyplot(Cotter_fit_o)
```

Fitting flow using new objective function and a constraining function using `nloptr`.

To run `nloptr` both an objective function (the function to minimise) and an constraint function need to be defined. This will then be used to refit the Cotter test data, and to check against standard fit.

The objective function

This is fairly easy compared to the first test, as this no longer includes β, but just calculates the SSE to minimise on. I could use a different goal function, but I am simply sticking with SSE at the moment.

```
objfun <- function(par, beta = 0.5, 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

  SSE <- sqrt(sum((error)^2,na.rm=T))/length(error)


  ValToMinimise <- SSE
  return(ValToMinimise)
}
# test
x01 <- objfun(c(600, 0, 100, 2, 0.15),
               mod=Cotter_mod,in_data=data.cal)
x01
```

```
## [1] 0.008692899
```

The constraining function

This again introduces the parameter β, but no longer calibrates on this. We can change beta manually or using a loop. In principle, it still assumes that the constraints:
$error < \beta$
and
$error > -\beta$
Can be rewritten as: $\sqrt(error^2) - \beta$

```
Constr_Mod <- function(par, beta, 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
  output <- sqrt(error^2) - beta
```

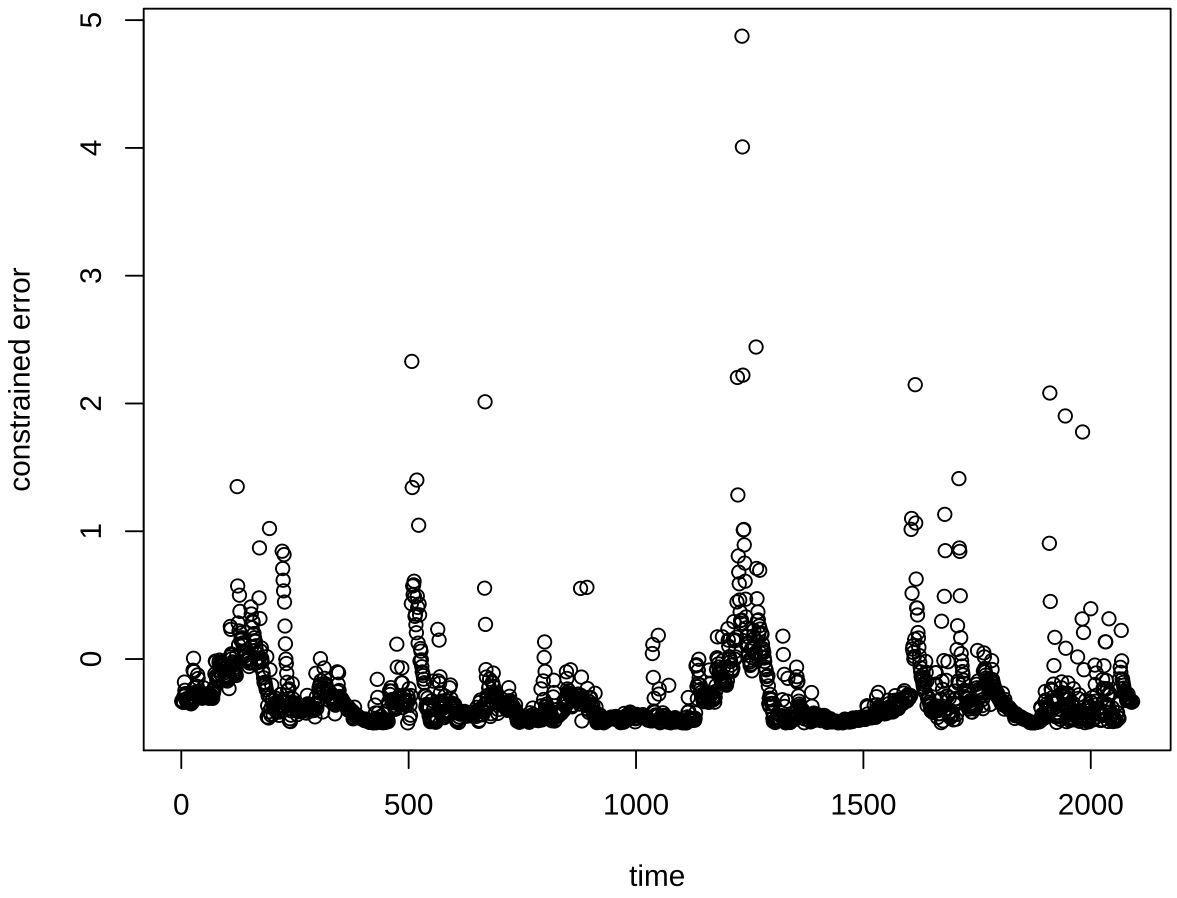

Figure 2: Plot of the constrained errors

```
  return(coredata(output))
}

x02 <- Constr_Mod(par=c(600, 0, 100, 2, 0.15), beta=0.5,
                  mod=Cotter_mod,in_data=data.cal)
plot(x02, xlab="time", ylab="constrained error")
```

Calibrating the model

Now we can use the non-linear constrained optimisation in `nloptr` to constrain the model optimisation. I have used the suggested algorithm "NLOPT_LN_COBYLA" in the vignette, which does not require a gradient to be specified.
At the end, the optimised parameters are reinserted into the model and the optimised model plotted together with the observed data.

```r
res1 <- nloptr( x0=c(600, 0, 100, 2, 0.15),
                 eval_f=objfun,
                 lb = c(100, -30, 5, 0.6 , 0.01),
                 ub = c(1500, 20, 500, 10, 0.5),
                 eval_g_ineq = Constr_Mod,
                # algorithm simply what is suggested in the vignette
                 opts = list("algorithm"="NLOPT_LN_COBYLA",
                             "xtol_rel"=1.0e-7,
                             maxeval=1500),
                 beta = 5,
                in_data = data.cal,
                mod = Cotter_mod
                )

#str(res1)
# plot the solution:
model_fit <- update(Cotter_mod, x1 = res1$solution[1],
                    x2 = res1$solution[2], x3 = res1$solution[3],
                    x4 = res1$solution[4], etmult=res1$solution[5])
# make a plot
par(mfrow=c(1,2))
plot(fitted(model_fit))
lines(data.cal$Q,col="red")
hist(data.cal$Q[101:length(data.cal$Q)]-fitted(model_fit),breaks=50)
```

```r
par(mfrow=c(1,1))
```

This shows a fairly good fit of the data, comparable to the non-constrained optimisation. It can be inspected more formally by calculating the regular statistics.

Varying beta

The key parameter of interest is β, with the idea that this results in some optimal choice of β that will balance the SSE with the local error minimisation. This means a loop through choices of beta should be run and plotted.

```r
# Now we want to run a loop over beta values
# plot the resulting r.sq.sqrt and rel.bias on a figure together
# with the result of the unconstrained optimisation

beta1 <- seq(2,4,length=10)
n <- length(beta1)

result <- data.frame(beta = beta1, mean_error = rep(0,n),
                     var_error = rep(0,n),
                     r.sq.srqt = rep(0,n),
                     Linccc = rep(0,n))

unconstrResults <- rep(0,4)

unc_error <- data.cal$Q[101:length(data.cal$Q)]-fitted(model_fit)
unconstrResults[1:2] <- c(mean(unc_error),var(unc_error))
linsccc <- epi.ccc(data.cal$Q[101:length(data.cal$Q)],
                   fitted(model_fit))
```

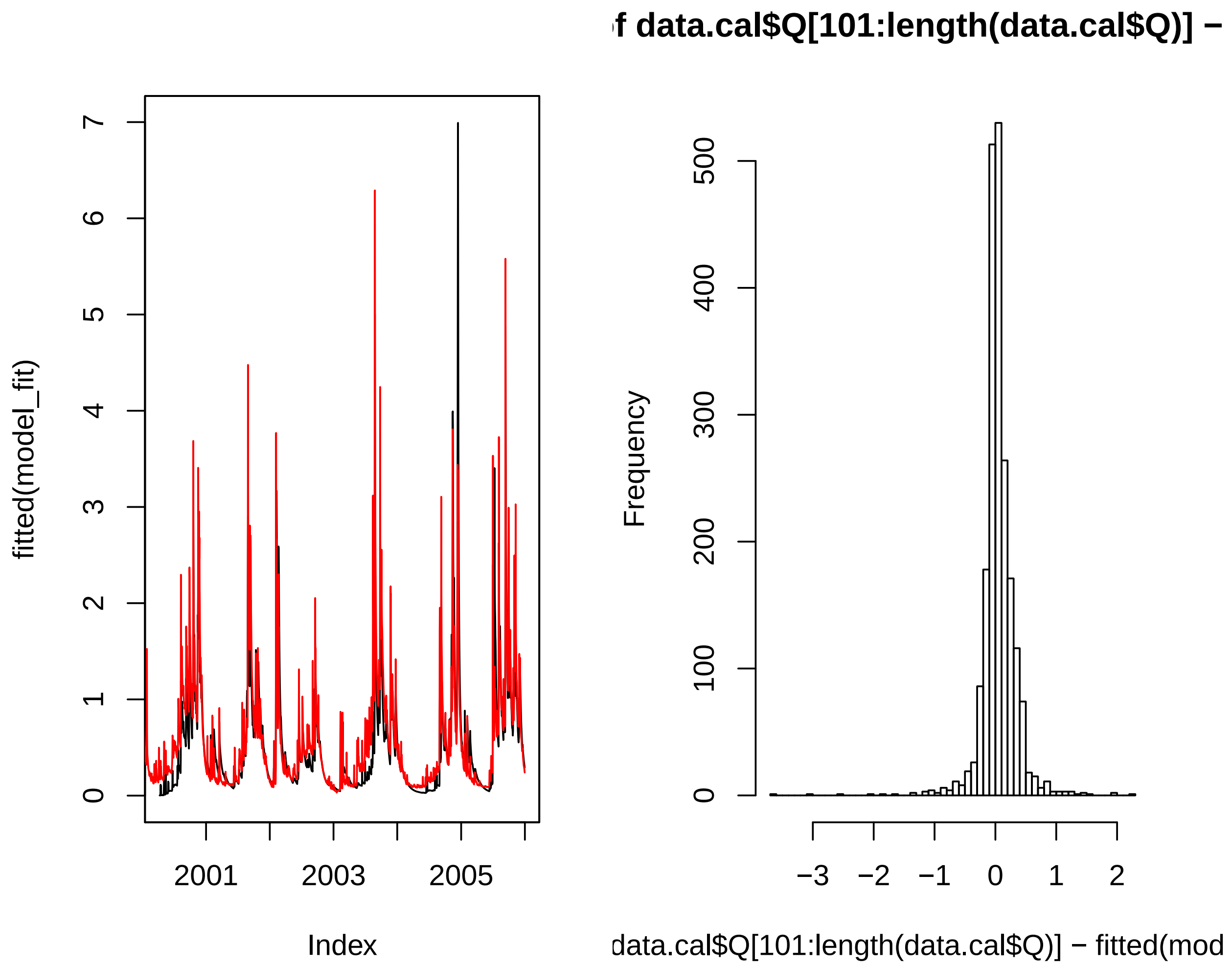

Figure 3: observed versus predicted, non-linear constrained optimisation and residual distribution

```r
unconstrResults[4] <- linsccc$rho.c$est

for (i in 1:length(beta1)) {
  res1 <- nloptr( x0=c(600, 0.1, 100, 2, 0.15),
                  eval_f=objfun,
                  lb = c(100, -30, 5, 0.6 , 0.01),
                  ub = c(1500, 20, 500, 10, 0.5),
                  eval_g_ineq = Constr_Mod,
                  opts = list("algorithm"="NLOPT_LN_COBYLA",
                              "xtol_rel"=1.0e-6,
                              maxeval=1500),
                  beta = beta1[i],
                  in_data = data.cal,
                  mod = Cotter_mod
  )

  model_fit <- update(Cotter_mod, x1 = res1$solution[1],
                        x2 = res1$solution[2],
                      x3 = res1$solution[3],
                        x4 = res1$solution[4],
                      etmult=res1$solution[5])

  # calculate statistics
  result[i,2] <- mean(data.cal$Q[101:length(data.cal$Q)]-
                                   fitted(model_fit))
  result[i,3] <- var(data.cal$Q[101:length(data.cal$Q)]-
                        fitted(model_fit))
  result[i,4] <- hmadstat("r.sq.sqrt")(data.cal$Q[101:length(data.cal$Q)],
                                    fitted(model_fit))
  result[i,5] <- epi.ccc(data.cal$Q[101:length(data.cal$Q)],
                         fitted(model_fit))$rho.c$est
}

unconstrResults[3:4] <- summary(model_fit)[c("rel.bias","r.sq.sqrt")]
```

Now make plots of the resulting statistics relative to the unconstrained fit.

```r
par(mfrow=c(2,2))
plot(result[,1], result[,2], xlab="beta", ylab="mean error",
     ylim=c(-0.5,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[1],n), col="red",lty=2)
plot(result[,1], result[,3], xlab="beta", ylab="var error",
     ylim=c(-0.2,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[2],n), col="red",lty=2)
legend("topright",c("constrained","standard"),
       lty=c(NA,2),pch=c(1,NA),
       lwd=c(5,1),col=c("blue","red"))
plot(result[,1], result[,4], xlab="beta", ylab="r.sq.sqrt flow",
     ylim=c(-0.5,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[4],n), col="red",lty=2)
plot(result[,1], result[,5], xlab="beta", ylab="Lin's ccc flow",
     ylim=c(0,1), col="blue", lwd=5)
lines(beta1,rep(linsccc$rho.c$est,n), col="red",lty=2)
```

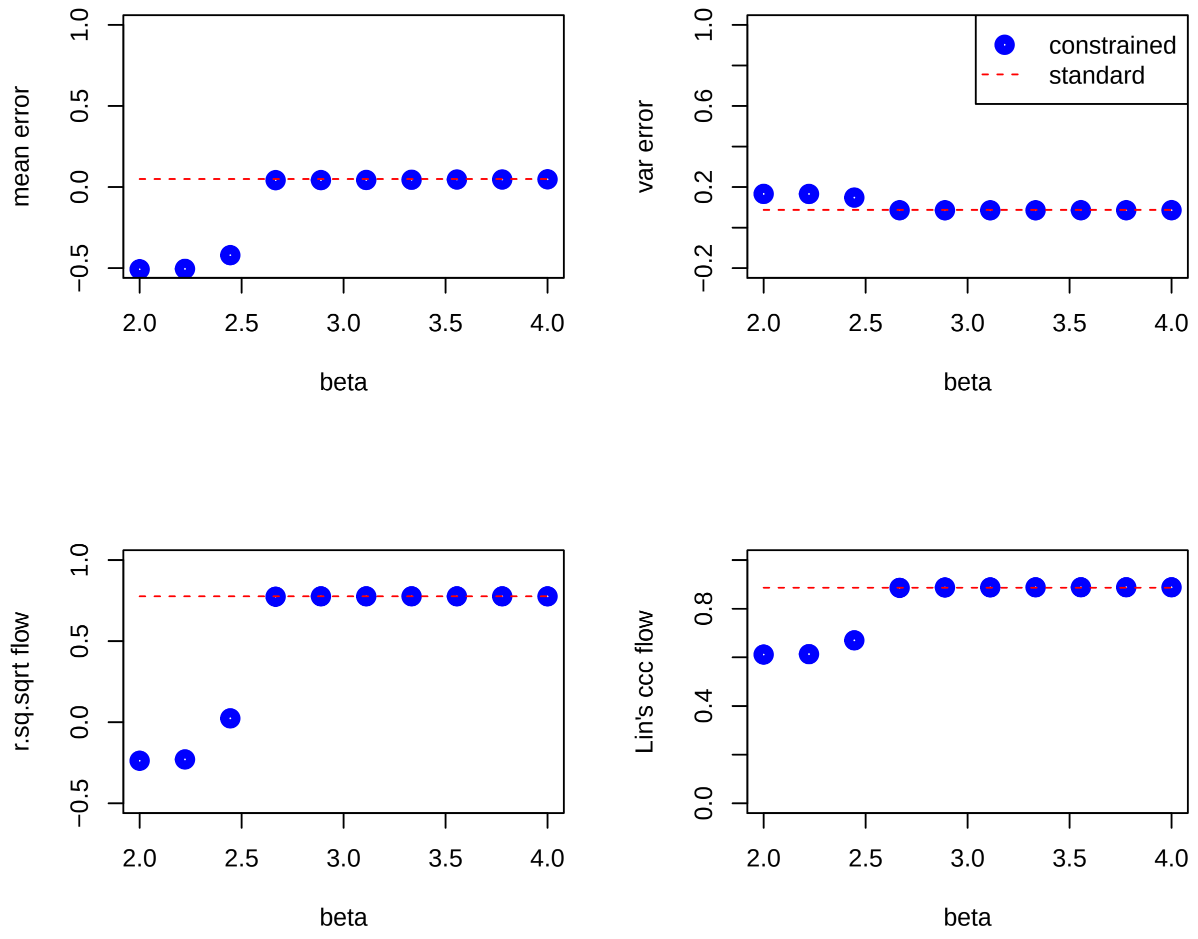

Figure 4: Summary statistics results of the constrained optimisation using a range of beta values from 1 - 10

```
par(mfrow=c(1,1))
```

Including ET in the constrained optimisation

The next step is to redo the above analysis but to include ET as well in the calibration. The idea is that the constrained optimisation will deliver a better result in the ET fit, relative to the unconstrained optimisation.

The unconstrained fit

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.
This is the same as in part 1.

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.

```
# load functions from Satellite calibration
old_dir <- getwd()
setwd("../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
```

```
setwd(old_dir)

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

# 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.

```
# 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   -0.03493251 0.7778544 0.7795779 0.7389605
## Fit only ET   0.20628648 0.5142551 0.6057939 0.5913386
## Fit ET and Q -0.03637193 0.7763757 0.7743895 0.7334586
```

```
# 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.

```
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] -0.2716641
```

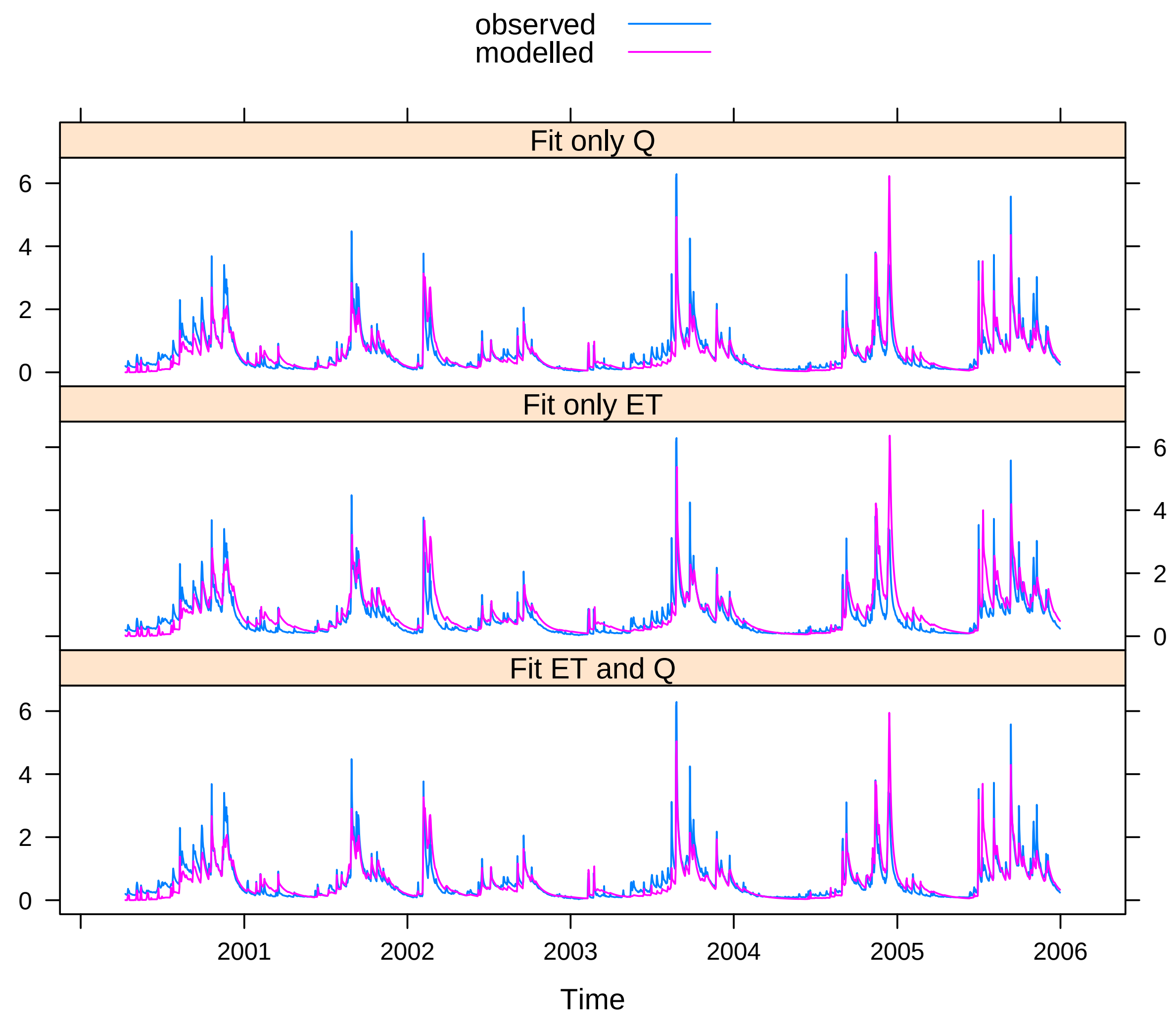

Figure 5: Standard hydromad satellite data fit using FitBySCE() on Cotter and MODIS data

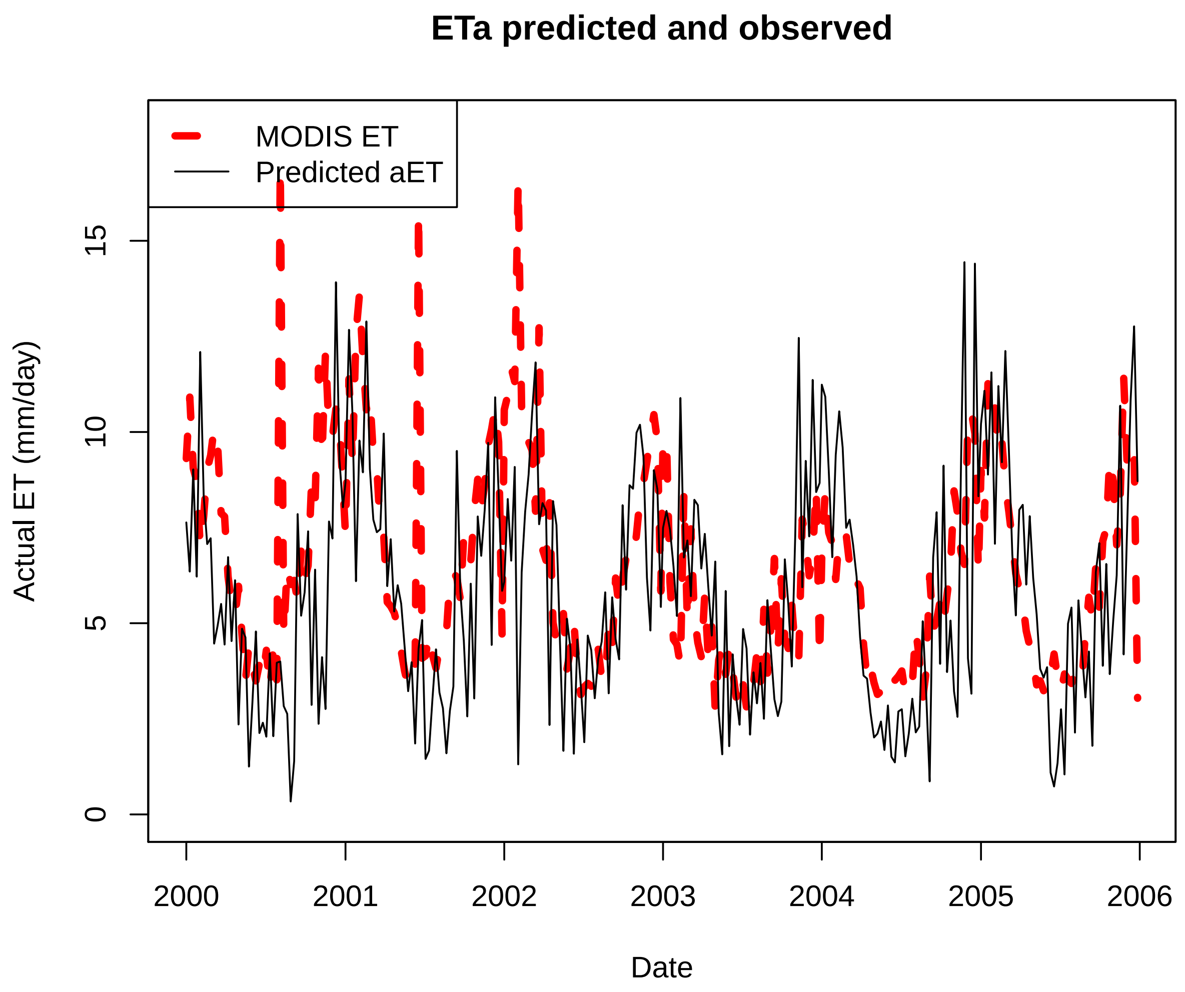

Figure 6: Fit of the model to the MODIS ET data using the standard hydromad fit

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

ET constrained optisation

For this the optimisation and constraining function needs to be rewritten to include ET. This is similar to the above example with only flow data.

```
objfun_ET <- function(par, beta = 0.5, 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])
```

```r
  #browser()
  # Q objective functions
  error_Q <- fitted(mod_run) - in_data$Q

  SSE_Q <- sqrt(sum((error_Q)^2,na.rm=T))/length(error_Q)
  # ET objective function
  #browser()
  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)

  SSE_ET <- sqrt(sum((error_ET)^2,na.rm=T))/length(error_ET)

  w <- 0.5

  ValToMinimise <- w*(SSE_Q) +
    (1-w)*(SSE_ET)
  return(ValToMinimise)
}

test_QET <- objfun_ET(par=c(600, 0, 100, 2, 0.15, 5),
                  mod=Cotter_mod_M,in_data=data.modis.cal)
test_QET
```

```
## [1] 0.1919401
```

One thing to observe is that in terms of magnitude the SSE_ET dominates this overall SSE (check test_QET against x01). However, in terms of variance, probably SSE_Q dominates.

The constraining function can be similarly defined , but it is slightly tricky, as the ET errors are essentially on an 8 day interval, while the flow errors are on a 1 day interval. The trick is to combine these and allow the model to optimise against it.

```r
# Constraint function
Constr_Mod_ET <- function(par, beta, 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()
  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_Q <- coredata(fitted(mod_run)) -
    coredata(in_data$Q[101:length(data.modis.cal$Q)])
  error_ET <- coredata(ET_fin) - coredata(aET_fin)
```

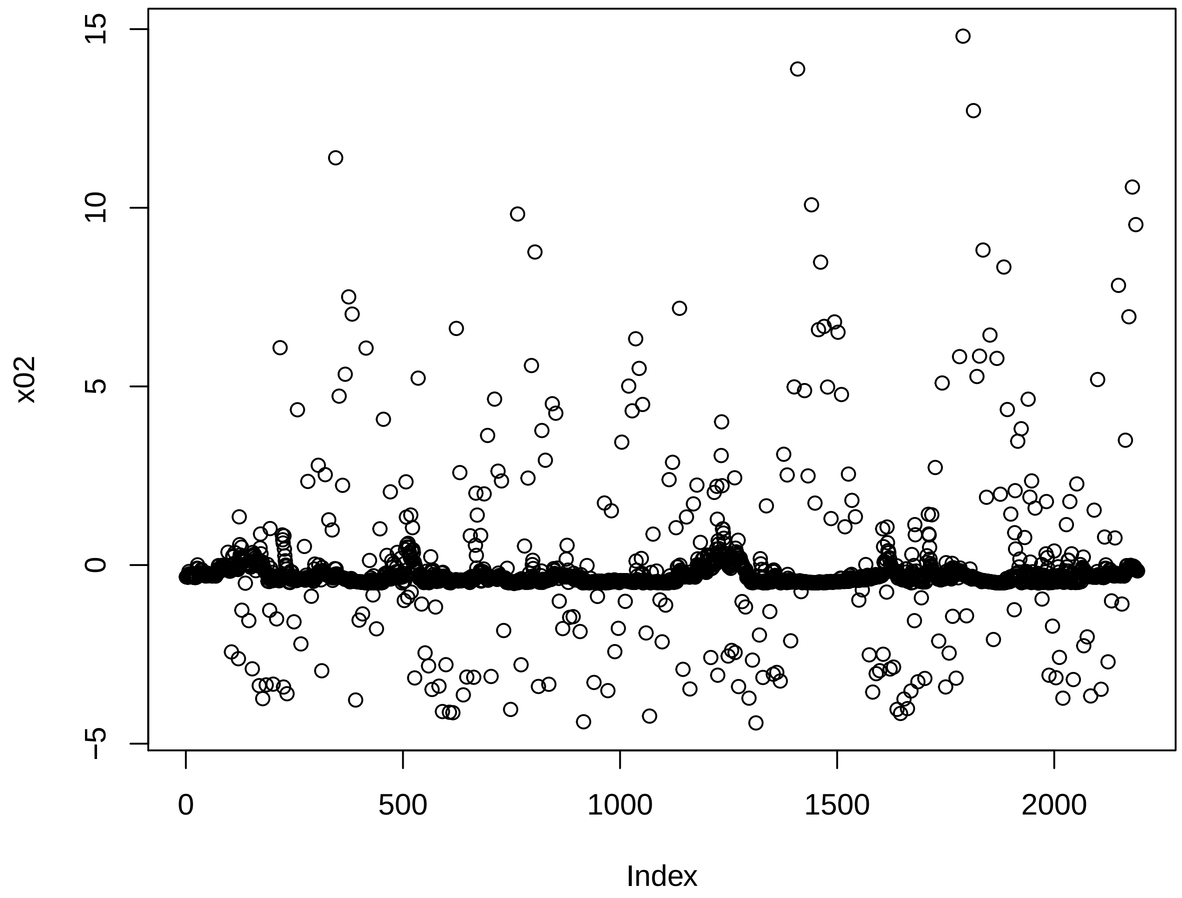

Figure 7: plot of the constrained errors including ET

```
  adj_error_ET <- zoo(sqrt(error_ET^2) - beta*8,
                      order.by = time(time(aET_fin)))
  adj_error_Q <- zoo(sqrt(error_Q^2) - beta,
                     order.by = time(in_data))
  adj_error_fin <- merge(adj_error_Q,adj_error_ET[13:length(adj_error_ET)], all=T)
  out <- apply(adj_error_fin, 1, sum, na.rm=T)

  return(na.omit(out))
}

x02 <- Constr_Mod_ET(par=c(600, 0, 100, 2, 0.15), beta=0.5,
                 mod=Cotter_mod,in_data=data.modis.cal)
plot(x02)
```

doing the calibration on flow and ET and inspecting the statistics

The next step is to simply involke `nloptr` again and to calibrate based on the two functions and inspect the output.

```
res1 <- nloptr( x0=c(600, 0.1, 100, 2, 0.15),
                eval_f=objfun_ET,
                lb = c(100, -30, 5, 0.6 , 0.01),
                ub = c(1500, 20, 500, 10, 0.5),
                eval_g_ineq = Constr_Mod_ET,
                opts = list("algorithm"="NLOPT_LN_COBYLA",
                            "xtol_rel"=1.0e-7,
                            maxeval=2500),
                beta = 5,
                in_data = data.modis.cal,
                mod = Cotter_mod_M
)

#str(res1)
# plot the solution:
model_fit_M <- update(Cotter_mod_M, x1 = res1$solution[1],
                    x2 = res1$solution[2], x3 = res1$solution[3],
                    x4 = res1$solution[4], etmult=res1$solution[5])
# make a plot
par(mfrow=c(1,2))
plot(fitted(model_fit_M))
lines(data.cal$Q,col="red")
hist(data.cal$Q-fitted(model_fit_M),breaks=50)
```

```
par(mfrow=c(1,1))
```

The statistics of the ET calibration can also be shown.

```
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$et.period),
                     sum)
# lins concordance
epi.ccc(coredata(aET_obs),coredata(ET_pred))$rho.c$est
```

```
## [1] 0.4278459
```

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

Varying beta for the constrained ET & Flow calibration

Similar to the flow calibration, the key parameter of interest is β, with the idea that this results in some optimal choice of β that will balance the SSE with the local error minimisation. This means a loop through choices of β should be run and plotted.

Here, I am including Lin's concordance (which is a measure of deviation of the 1:1 line) as a statistic.

```
beta1 <- seq(1,10,by=1)
n <- length(beta1)
```

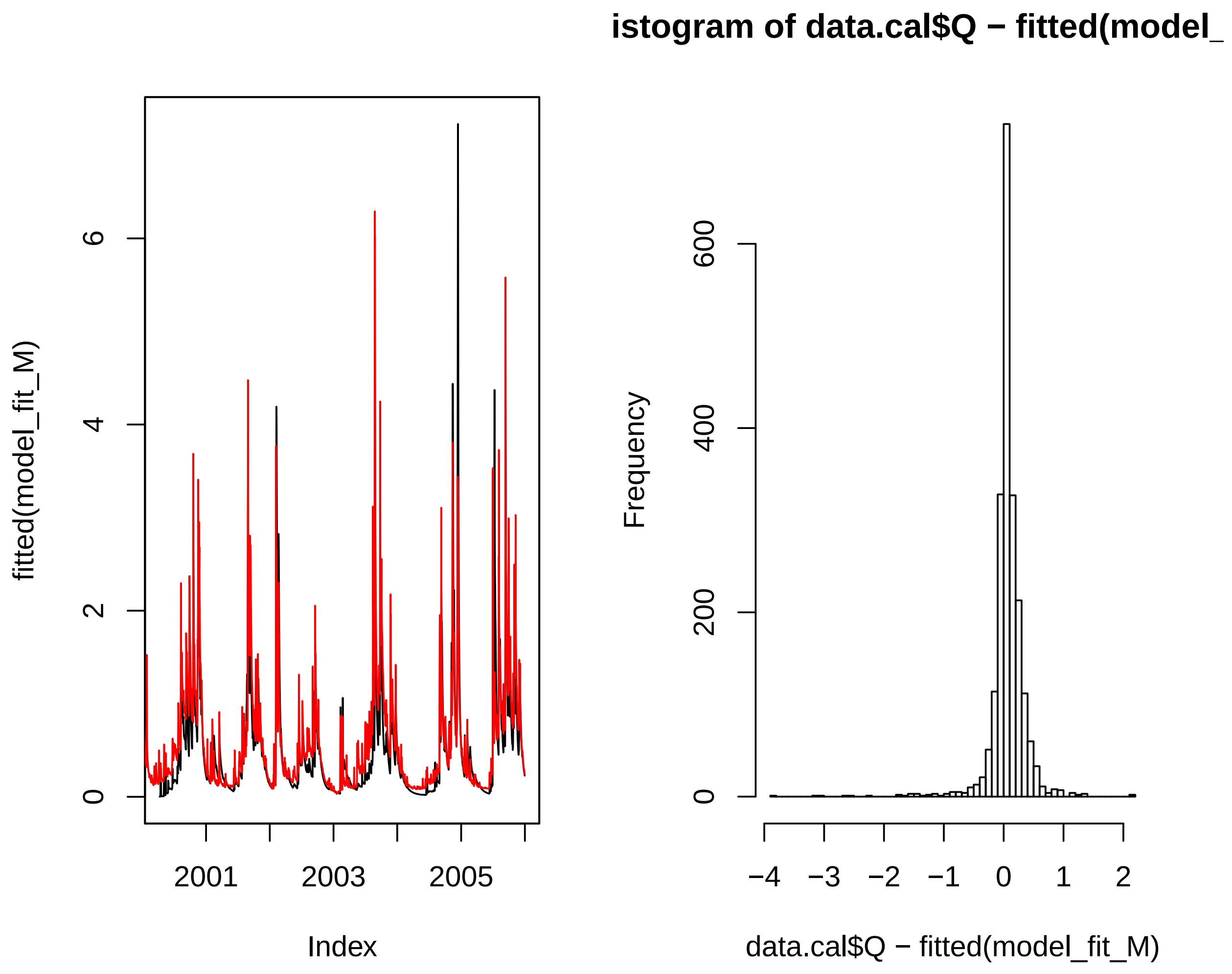

Figure 8: observed versus predicted, non-linear constrained optimisation on flow and ET and the residual distribution for flow

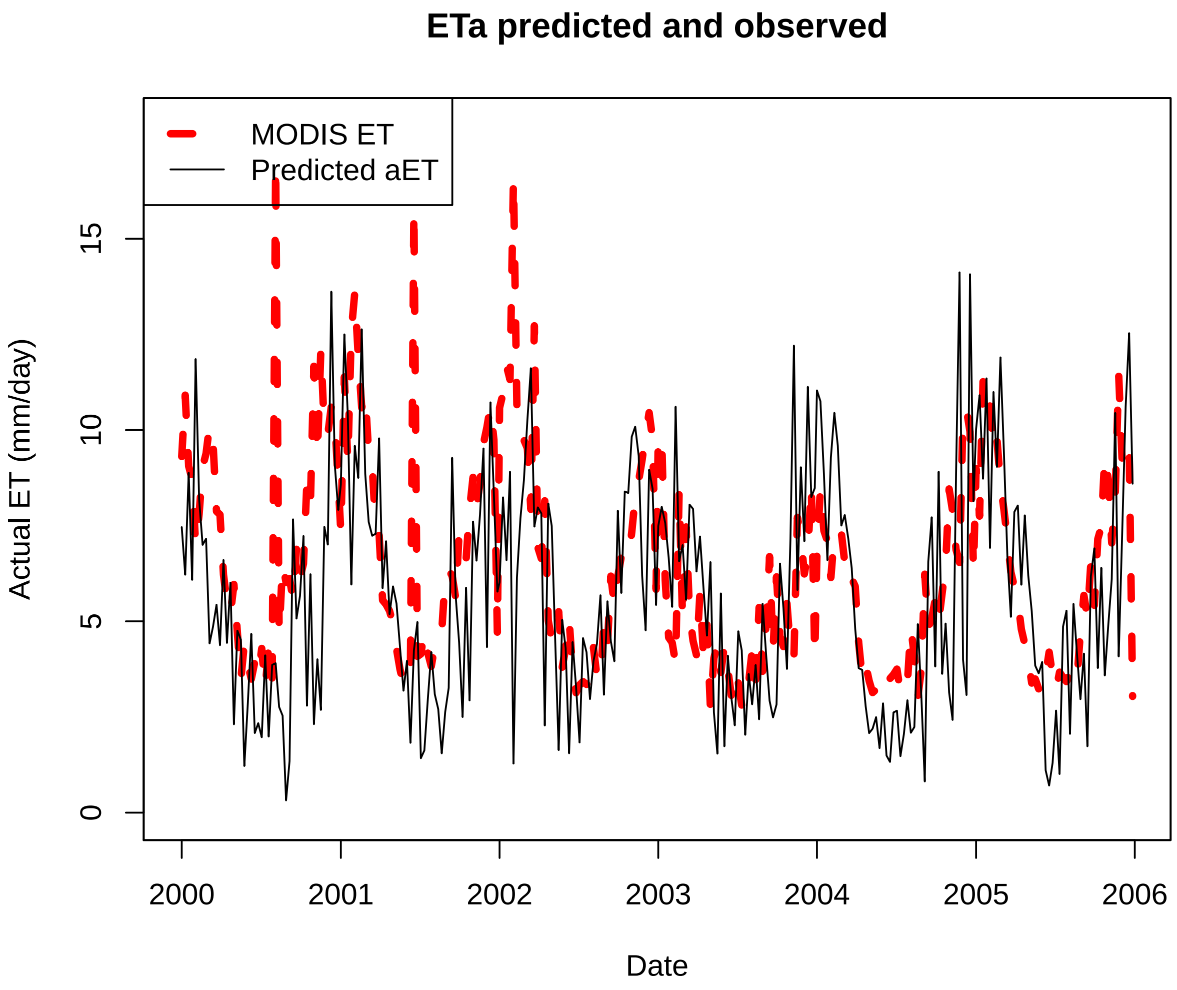

Figure 9: fitted against observed ET for constrained optimisation

```r
result <- data.frame(beta = beta1, mean_error = rep(0,n),
                     var_error = rep(0,n),
                     r.sq.srqt = rep(0,n),
                     Linccc = rep(0,n),
                     mean_error_ET = rep(0,n),
                     var_error_ET = rep(0,n),
                     r.sq.srqt_ET = rep(0,n),
                     Linccc_ET = rep(0,n))

unconstrResults <- rep(0,8)

unc_error <- data.modis.cal$Q[101:length(data.modis.cal$Q)] -
  fitted(Cotter_fit_q_ET[["Fit ET and Q"]])
unconstrResults[1:2] <- c(mean(unc_error),var(unc_error))
unc_errorET <- coredata(aET_obs) - coredata(ET_pred)
unconstrResults[5:6] <- c(mean(unc_errorET),var(unc_errorET))


for (i in 1:length(beta1)) {
  res1 <- nloptr( x0=c(600, 0.1, 100, 2, 0.15),
                  eval_f=objfun_ET,
                  lb = c(100, -30, 5, 0.6 , 0.01),
                  ub = c(1500, 20, 500, 10, 0.5),
                  eval_g_ineq = Constr_Mod_ET,
                  opts = list("algorithm"="NLOPT_LN_COBYLA",
                              "xtol_rel"=1.0e-7,
                              maxeval=2500),
                  beta = beta1[i],
                  in_data = data.modis.cal,
                  mod = Cotter_mod_M
  )

  model_fit_M <- update(Cotter_mod_M, x1 = res1$solution[1],
                        x2 = res1$solution[2],
                        x3 = res1$solution[3],
                        x4 = res1$solution[4],
                        etmult=res1$solution[5])

  # ET calibration
  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)

  result[i,2] <-
    mean((data.modis.cal$Q[101:length(data.modis.cal$Q)]-
                       fitted(model_fit_M)))
  result[i,3] <-
    var((data.modis.cal$Q[101:length(data.modis.cal$Q)]-
                      fitted(model_fit_M)))
  result[i,4] <-
    hmadstat("r.sq.sqrt")(data.modis.cal$Q[101:length(data.modis.cal$Q)],
                                        fitted(model_fit_M))
  result[i,5] <-
    epi.ccc(data.modis.cal$Q[101:length(data.modis.cal$Q)],
                          fitted(model_fit_M))$rho.c$est
  # calculate statistics
  result[i,6] <- mean(coredata(aET_obs) - coredata(ET_pred))
  result[i,7] <- var(coredata(aET_obs) - coredata(ET_pred))
  result[i,8] <- hmadstat("r.sq.sqrt")(coredata(aET_obs),
                                        coredata(ET_pred))
  result[i,9] <- epi.ccc(coredata(aET_obs),
                         coredata(ET_pred))$rho.c$est

}

unconstrResults[3] <- summary(Cotter_fit_q_ET[["Fit ET and Q"]],
                              items = "r.sq.sqrt")[9]
ET_pred_unconstr <- aggregate(Cotter_fit_q_ET[["Fit ET and Q"]]$U$ET,
                                list(date=data.modis.cal$et.period),
                                sum)
unconstrResults[7] <- hmadstat("r.sq.sqrt")(coredata(aET_obs),
                                             coredata(ET_pred_unconstr))
linsET <- epi.ccc(coredata(aET_obs),coredata(ET_pred))
unconstrResults[8] <- linsET$rho.c$est
```

Now make plots of the resulting statistics relative to the unconstrained fit.

```
par(mfrow=c(2,2))
plot(result[,1], result[,2], xlab="beta", ylab="mean error flow",
     ylim=c(-0.5,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[1],n), col="red",lty=2)
plot(result[,1], result[,3], xlab="beta", ylab="var error flow",
     ylim=c(-0.2,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[2],n), col="red",lty=2)
legend("topright",c("constrained","standard"),lty=c(NA,2),pch=c(1,NA),
       lwd=c(5,1),col=c("blue","red"))
plot(result[,1], result[,4], xlab="beta", ylab="r.sq.sqrt flow",
     ylim=c(-0.5,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[3],n), col="red",lty=2)
plot(result[,1], result[,5], xlab="beta", ylab="Lin's ccc flow",
     ylim=c(0,1), col="blue", lwd=5)
lines(beta1,rep(linsccc$rho.c$est,n), col="red",lty=2)
```

```
par(mfrow=c(1,1))
```

And we can look at the same result for ET

```
par(mfrow=c(2,2))
plot(result[,1], result[,6], xlab="beta", ylab="mean error ET",
     ylim=c(-3,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[5],n), col="red",lty=2)
plot(result[,1], result[,7], xlab="beta", ylab="var error ET",
     ylim=c(5,20), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[6],n), col="red",lty=2)
legend("topright",c("constrained","standard"),lty=c(NA,2),pch=c(1,NA),
       lwd=c(5,1),col=c("blue","red"))
```

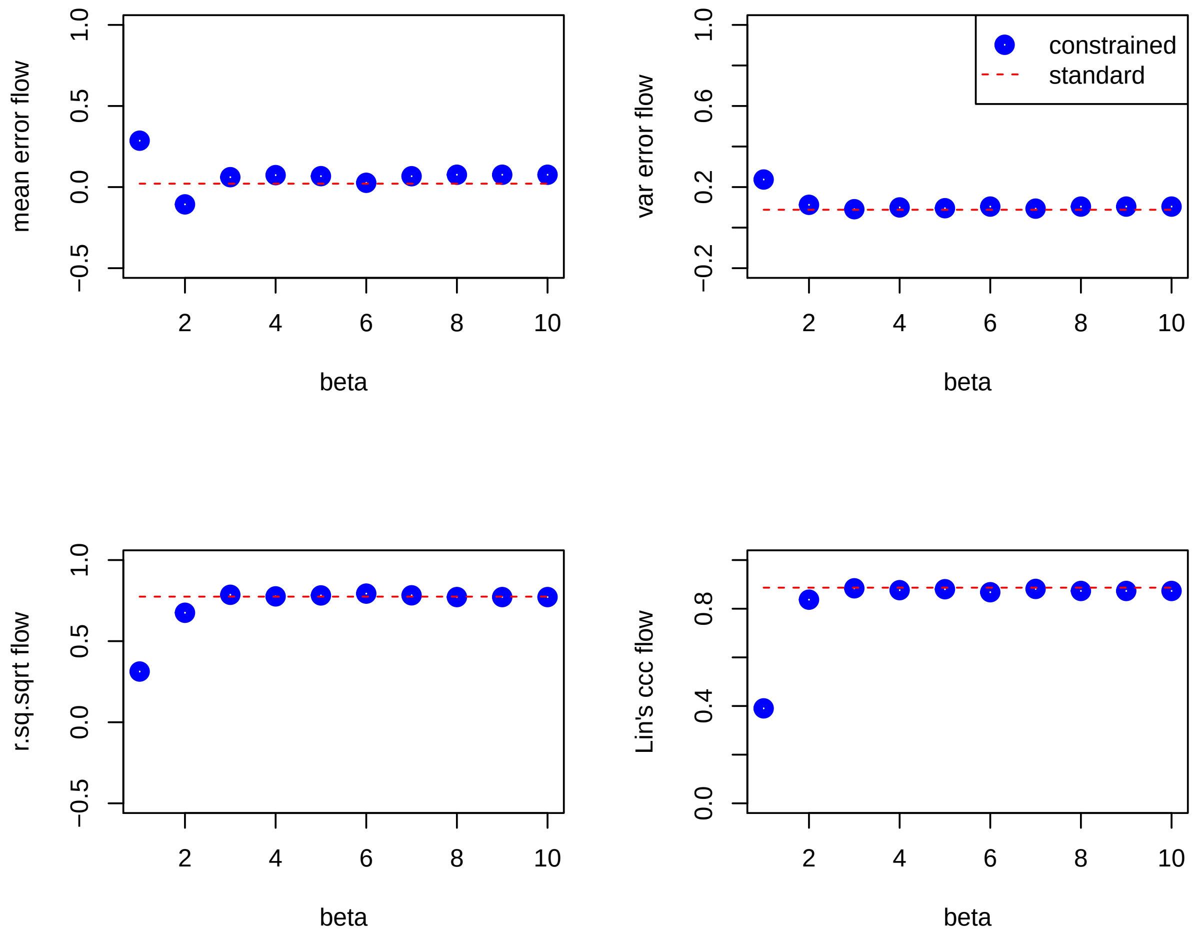

Figure 10: Summary statistics results of the constrained fot for flow and ET, Flow results only

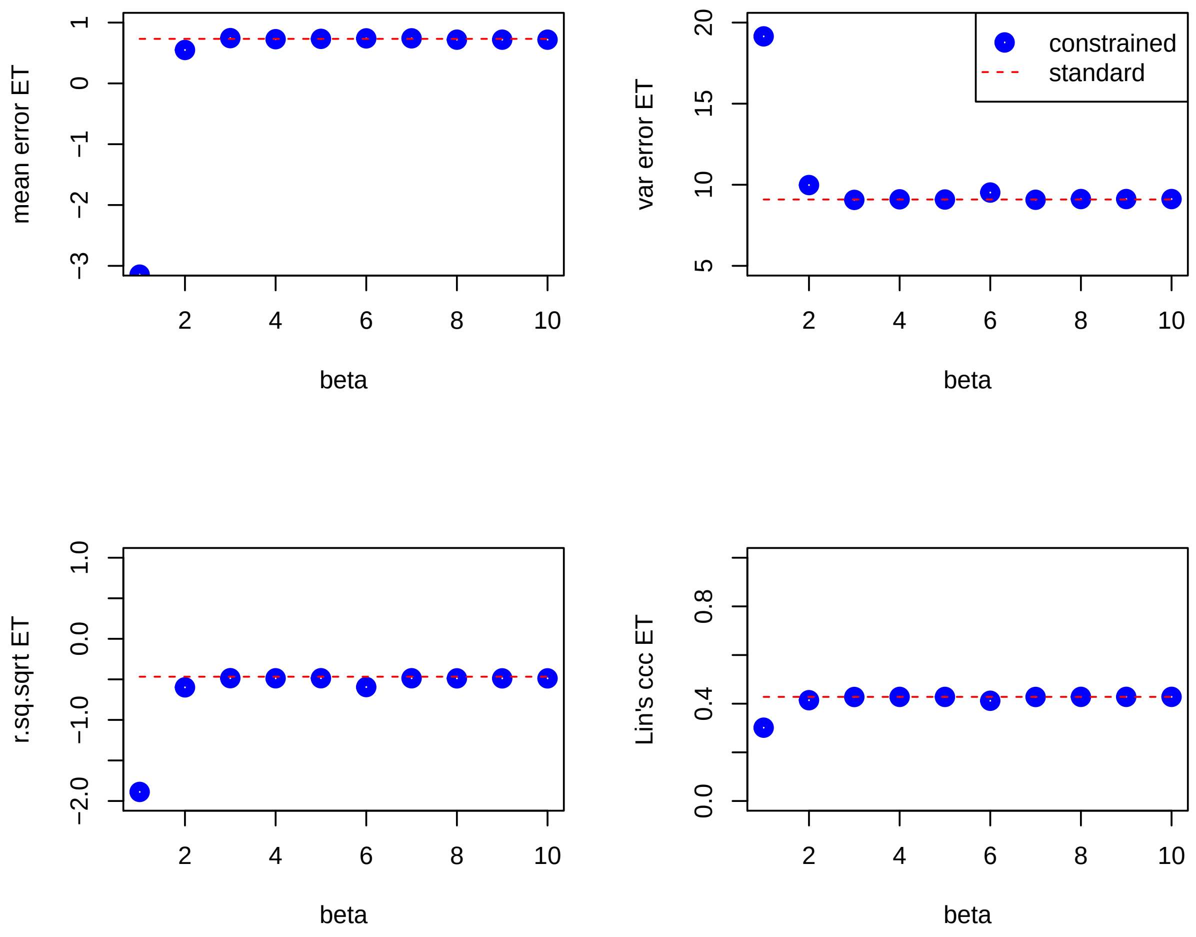

Figure 11: Summary statistics results of the constrained fot for flow and ET, ET results only

```r
plot(result[,1], result[,8], xlab="beta", ylab="r.sq.sqrt ET",
     ylim=c(-2,1), col="blue", lwd=5)
lines(beta1,rep(unconstrResults[7],n), col="red",lty=2)
plot(result[,1], result[,9], xlab="beta", ylab="Lin's ccc ET",
     ylim=c(0,1), col="blue", lwd=5)
lines(beta1,rep(linsET$rho.c$est,n), col="red",lty=2)
```

```r
par(mfrow=c(1,1))
```

The results are somewhat confusing, it seems that for a larger beta the optimisation converges on the unconstrained optimisation. However, I would have expected an optimal point for a smaller beta.