Checking FLBEIA inputs

26 febrero, 2020

Aim

FLBEIA (Garcia et al. 2017) provides a battery of tutorials for learning how to use this software. This tutorial of **FLBEIA** is a practical guide about how to check if **FLBEIA** inputs have been correctly defined.

In this tutorial the functions available for checking input objects of `FLBEIA` function are presented. It is recommemded to use these functions prior to do the call to the `FLBEIA` function, in order to avoid some common errors.

The procedure to create the objects used to run the simulation is described in the **FLBEIA** manual. This manual can be downloaded from GitHub, within the 'doc' folder of the package installation or typing `help(package = FLBEIA)` in the R console. For details on these objects, see FLBEIA manual, Section 5.2, see tutorial on FLBEIA - Simple example or type `?FLBEIA` in the R console.

Nevertheless, it is not neccesary to create the objects to set the simulation because the datasets `one`, `oneIt` and `multi`, available in FLBEIA package will be used.

To see all the datasets available in the **FLBEIA** package:

```
data(package="FLBEIA")
```

Required packages to run this tutorial

To follow this tutorial you should have installed the following packages:

- FLR: FLCore, FLFleet

```
install.packages( c("FLCore", "FLFleet", "FLBEIA"),
                  repos="http://flr-project.org/R")
```

It has to be noted that packages `FLCore`, `FLFleet` and `FLBEIA` have to be installed in this exact order, as alternative orders can cause some problems.

Load all thenecessary packages.

```
library(FLBEIA)
```

Checking functions

In order to avoid commonly occurring mistakes when conditioning FLBEIA, several functions have been created for checking the input objects of the `FLBEIA` function.

There are two types of functions to run the checkings. On the one hand, there are specific functions for some of the input objects of the `FLBEIA` function (e.g. `checkBiols`). And on the other hand, a function that allows checking all the input objects at the same time (`checkFLBEIAData`).

Checking specific input objects one by one:

Available functions for checking an specific input object:

- `checkBiols` : for checking biols argument of FLBEIA function (of class `FLBiols`).
- `checkFleets` : for checking fleets argument of FLBEIA function (of class `FLFleetsExt`).
- `checkSRs` : for checking SRs argument of FLBEIA function (list of `FLSRsim` objects).
- `checkBDs` : for checking BDs argument of FLBEIA function (list of `FLBDsim` objects).
- `checkAdvice` : for checking advice argument of FLBEIA function (of class `list` with two `FLQuant` elements, TAC and quota.share).
- `checkObsctrl`: for checking obs.ctrl argument of FLBEIA function (of class `list`).

Let's see some examples with the FBLEIA datasets. Firstly, we need to load the datasets.

```
rm(list=ls()) # empty the workspace

data(one)     # load the datasets
data(oneIt)
data(multi)
```

In each of the following subsections, the different arguments of the FLBEIA function will be checked.

biols (`FLBiols`)

FLBEIA data objects `oneBio`, `oneItBio` and `multiBio` should pass the checkings.

```
checkBiols(oneBio)
```

```
## [1] TRUE
```

```
checkBiols(oneItBio)
```

```
## [1] TRUE
```

```
checkBiols(multiBio)
```

```
## [1] TRUE
```

They do, so let's see some examples with incorrect input data.

```
obj1 <- obj2 <- oneBio

mat(obj1$stk1)[1,1,] <- -0.5 # mat < 0
checkBiols(obj1)             # returns an error

mat(obj2$stk1)[1,1,] <- 5    # mat > 1
checkBiols(obj2)             # returns an error
```

fleets (`FLFleetsExt`)

We are going now to check if `oneFl`, `oneItFl` and `multiFl` pass the checkings.

```
checkFleets(oneFl)
```

```
## [1] TRUE
```

```
  checkFleets(oneItFl)
```

```
## [1] TRUE
```

```
  checkFleets(multiFl)
```

```
## Warning in checkFleets(multiFl): Check capacity if
## fleets.ctrl[['fl2']]$effort.model != 'fixedEffort'
```

```
## [1] TRUE
```

In this case `multiFl` passess the checkings, but it returns a warning because there are `NA` values for capacity in some years. As if we do not fix the effort in the projection, then always an effort threshold is required (the capacity). So if you include an additonal argument in the call to the function (`ctrl` argument, which corresponds to the fleets.ctrl object), then the function checks which is you effort function in reality and will return an error only if the effort function is different to fixed effort.

```
  checkFleets(multiFl, ctrl = multiFlC) # returns an error
```

In this case we get an error because we have `NA` values for capacity in some of the initial years. However, if we restrict to the projection period, the object will pass the check.

```
  sim.years <- as.numeric(multiMainC$sim.years)
  checkFleets( window(multiFl, start = sim.years[1]-1, end = sim.years[2]),
               ctrl = multiFlC)
```

```
## [1] TRUE
```

We are going to see some examples with incorrect data that makes the function return an error:

```
  obj1 <- obj2 <- obj3 <- obj4 <- multiFl

  obj1$fl1@effort[,ac(1990),,1,]
  obj1$fl1@metiers$met1@effshare[,ac(1990),,1,] <- NA # sum != 1, but effort = 0
  checkFleets(obj1)                                   # pass the check

  obj1$fl1@effort[,ac(1999),,1,]
  obj1$fl1@metiers$met1@effshare[,ac(1999),,1,] <- 5 # sum != 1, and effort > 0
  checkFleets(obj1)                                  # returns an error

  obj2$fl1@metiers$met1@catches$stk1@landings.sel[] <-
    obj2$fl1@metiers$met1@catches$stk1@discards.sel[] <- 0 # landins.sel + discards.sel != 1
  checkFleets(obj2)                                        # returns an error

  obj3$fl1@metiers$met1@catches$stk1@landings.wt[,5,] <- NA   # landings.wt == NA
  checkFleets(obj3)                                          # returns an error

  obj3$fl1@metiers$met1@catches$stk1@landings.wt[,5,] <- -0.7 # landings.wt < 0
  checkFleets(obj3)                                          # returns an error

  obj4$fl1@metiers$met1@catches$stk1@discards.wt[,5,] <- NA   # discards.wt == NA
  checkFleets(obj4)                                          # returns an error

  obj4$fl1@metiers$met1@catches$stk1@discards.wt[,5,] <- -0.1 # discards.wt < 0
  checkFleets(obj4)                                          # returns an error
```

SRs and BDs (list of FLSRsim or FLBDsim)

We are going now to check if oneSR, oneItSR, multiSR and multiBD pass the checkings.

```
  checkSRs(oneSR)
  checkSRs(oneItSR)
  checkSRs(multiSR)
  checkBDs(multiBD)
```

We get errors in some of the cases, but these are due to missing values for uncertainty. These values are due to missing values in the covariate included in the stock-recruitment model, that lead to NA values in the fitted values. However, if we just have a look at the simulation period, objects will be ok.

```
  checkSRs(lapply(oneSR, window, start = sim.years[1]-1, end = sim.years[2]))
```

```
## [1] TRUE
```

```
  checkSRs(lapply(oneItSR, window, start = sim.years[1]-1, end = sim.years[2]))
```

```
## [1] TRUE
```

Now we follow with some more examples returning an error:

```
# BDs

  obj1 <- obj2 <- obj3 <- multiSR

  obj1$stk1@proportion[,,,1,] <- -1000   # proportions > 0
  checkSRs(obj1)                         # returns an error

  obj1$stk1@proportion[,,,1,] <- 1000    # proportions < 1
  checkSRs(obj1)                         # returns an error

  obj2$stk1@proportion[,,,1:4,] <- 0.5   # sum proportions = 1
  checkSRs(obj2)                         # returns an error

  obj3$stk1@uncertainty[1,1,,1,] <- -0.5 # uncertainty> 0
  checkSRs(obj3)                         # returns an error

# SRs

  obj1 <- obj2 <- obj3 <- multiBD

  obj1$stk2@alpha[1,1,] <- 10          # alpha < 1
  checkBDs(obj1)                       # returns an error

  obj2$stk2@alpha[1,1,] <- (obj2$stk2@params["p",1,1,] / obj2$stk2@params["r",1,1,]+1) ^
    (1/obj2$stk2@params["p",1,1,]) - 1 # alpha > (p/r+1)^(1/p)
  checkBDs(obj2)                       # returns an error
```

advice (list)

We check now if oneAdv, oneItAdv and multiAdv pass the checkings.

```
  checkAdvice(oneAdv)
```

```
## [1] TRUE
```

```
  checkAdvice(oneItAdv)
```

```
## [1] TRUE
```

```
  checkAdvice(multiAdv)
```

```
## [1] TRUE
```

And we see some examples with incorrect data that makes the function return an error:

```
  obj1 <- multiAdv
  obj1$quota.share$stk1[,1,] <- 2 # sum quota shares != 1
  checkAdvice(obj1)               # returns an error
```

obs.ctrl (list)

Finally, we will check if `oneObsC`, `oneObsCIndAge`, `oneObsCIndBio`, `oneItObsC`, `oneItObsCIndAge`, `oneItObsCIndBio` and `multiObsC` pass the checkings.

```
  checkObsctrl(oneObsC)
```

```
## [1] TRUE
```

```
  checkObsctrl(oneObsCIndAge)
```

```
## [1] TRUE
```

```
  checkObsctrl(oneObsCIndBio)
```

```
## [1] TRUE
```

```
  checkObsctrl(oneItObsC)
```

```
## [1] TRUE
```

```
  checkObsctrl(oneItObsCIndAge)
```

```
## [1] TRUE
```

```
  checkObsctrl(oneItObsCIndBio)
```

```
## [1] TRUE
```

```
  checkObsctrl(multiObsC)
```

```
## [1] TRUE
```

And see some examples with incorrect data that makes the function return an error:

```
# Index: total biomass

  obj1 <- oneObsCIndBio

  obj1$stk1$stkObs$land.bio.error[,1,] <- -0.7 # error < 0
  checkObsctrl(obj1)                           # returns an error

# Index: numbers at age

  obj2 <- oneObsCIndAge

  obj2$stk1$stkObs$ages.error[1,,,] <- 2 # sum ages.error by age != 1
  checkObsctrl(obj2)                     # returns an error
```

Checking all the FLBEIA input objects at once:

For checking all the FLBEIA inputs at once, we should use `checkFLBEIAData` function. In this function, inputs should be the same as the FLBEIA function inputs. And it just calls internally to all the functions detailed in previous section, but restricting the objects to the projection period.

For example, lets check the inputs from FLBEIA datasets:

```
checkFLBEIAData( biols = oneBio, SRs = oneSR, BDs = NULL, fleets = oneFl,
                 covars = oneCv, indices = NULL, advice = oneAdv,
                 main.ctrl = oneMainC, biols.ctrl = oneBioC, fleets.ctrl = oneFlC,
                 covars.ctrl = oneCvC, obs.ctrl = oneObsC, assess.ctrl = oneAssC,
                 advice.ctrl = oneAdvC)
```

```
## [1] TRUE
```

```
checkFLBEIAData( biols = oneItBio, SRs = oneItSR, BDs = NULL, fleets = oneItFl,
                 covars = oneItCv, indices = NULL, advice = oneItAdv,
                 main.ctrl = oneItMainC, biols.ctrl = oneItBioC, fleets.ctrl = oneItFlC,
                 covars.ctrl = oneItCvC, obs.ctrl = oneItObsC, assess.ctrl = oneItAssC,
                 advice.ctrl = oneItAdvC)
```

```
## [1] TRUE
```

```
checkFLBEIAData( biols = multiBio, SRs = multiSR, BDs = multiBD, fleets = multiFl,
                 covars = multiCv, indices = NULL, advice = multiAdv,
                 main.ctrl = multiMainC, biols.ctrl = multiBioC, fleets.ctrl = multiFlC,
                 covars.ctrl = multiCvC, obs.ctrl = multiObsC, assess.ctrl = multiAssC,
                 advice.ctrl = multiAdvC)
```

```
## [1] TRUE
```

More information

- You can submit bug reports, questions or suggestions on this tutorial at https://github.com/flr/doc/issues.
- Or send a pull request to https://github.com/flr/doc/
- For more information on the FLR Project for Quantitative Fisheries Science in R, visit the FLR webpage, http://flr-project.org.
- You can submit bug reports, questions or suggestions specific to **FLBEIA** to flbeia@azti.es.

Software Versions

- R version 3.5.2 (2018-12-20)
- FLCore: 2.6.13
- FLBEIA: 1.15.4
- FLFleet: 2.6.1
- FLash: 2.5.11
- FLAssess: 2.6.3
- FLXSA: 2.6.3
- ggplotFL: 2.6.7
- ggplot2: 3.2.1
- **Compiled**: Wed Feb 26 12:28:16 2020

License

Author information

Sonia Sanchez. AZTI, Marine Research Unit. Herrera Kaia, Portualdea z/g, 20110, Pasaia, Gipuzkoa, Spain. https://www.azti.es/.

FLBEIA team. AZTI. Marine Reserach Unit. Txatxarramendi Ugartea z/g, 48395, Sukarrieta, Basque Country, Spain. http://flbeia.azti.es/. **Mail** flbeia@azti.es

References

Garcia, Dorleta, Sonia Sánchez, Raúl Prellezo, Agurtzane Urtizberea, and Marga Andrés. 2017. "FLBEIA: A Simulation Model to Conduct Bio-Economic Evaluation of Fisheries Management Strategies." *SoftwareX* 6: 141–47.