FLBEIA a R package to conduct Bio-Economic Impact assessments using FLR

Dorleta García, Raúl Prellezo, Sonia Sánchez & Marga Andrés

April 18, 2013

Abstract

FLBEIA (FL Bio-Economic Impact Assessment) is an R package build on top of FLR libraries. The purpose of the package is to provide a flexible and generic simulation model to conduct Bio-Economic Impact Assessments of harvest control rule based management strategies under a Management Strategy Evaluation (MSE) framework. As such, the model is divided into two main blocks, the operating model (OM) and the management procedure model (MPM). In turn, these two blocks are divided in 3 components. The OM is formed by the biological, the fleet and the covariables components and the MPM by the observation, the assessment and the management advice components.

The model is multistock, multifleet and seasonal and uncertainty is introduced by means of montecarlo simulation. The algorithm has been coded in a modular way to ease its checking and to make it flexible. The package provides functions that describe the dynamics of the different model components, under certain assumptions. In each specific model implementation the user chooses which of the functions are used. Furthermore, if in a specific case study or scenario, for some of the components, the functions provided within FLBEIA do not fulfill the requirements, the user can code the functions that adequately describe the dynamics of those components and use existing ones for the rest of the components. As the user can construct its own model, selecting existing submodels and constructing new ones, we can define it as a framework more than as a model. The package is still under development but most of its functionalities are already available. At the moment there are no functions to model trophic interactions but it is something planed in the short term. Main limitations of the model are that the stocks must be age structured or aggregated in biomass (length structure is not allowed) and that spatial dimension is not considered explicitly. Spatial characteristics could be modeled assigning stocks and/or fleets/metiers to specific areas.

Contents

1 Introduction **4**

2 The Concept of `BEIA` **7**
2.1 Operating Model 7
Biological component: 7
Fleet component: 7
Covariates: 7
Links among and within components: 8
2.2 The management procedure model 8
Observation component: 8
Assessment component: 8
Management advice component: 9

3 Running `BEIA` **9**
3.1 First level function: `BEIA` 9
3.2 Second level functions 10
3.2.1 Biological Component: `biols.om` 10
3.2.2 Fleets Component: `fleets.om` 11
3.2.3 Covariables Component: `covars.om` 14
3.2.4 Observation Component: `observation.mp` 15
3.2.5 Assessment Component: `assessment.mp` 16
3.2.6 Management Advice Component: `advice.mp` 17
3.3 Third level functions 18
3.3.1 Population growth functions 18
`fixedPopulation`: Fixed Population function. 18
`ASPG`: Age Structured Population Growth function. . 18
`BDPG`: Biomass Dynamic Population Growth function. 19
3.3.2 Effort models 19
`fixedEffort`: Fixed Effort model. 19
`SMFB`: Simple Mixed Fisheries Behavior. 19
`SSFB`: Simple Sequential Fisheries Behavior. 22
`MaxProfit.stkCnst`: Maximization of profit under a TAC constraint. 23
3.3.3 Price Models 24
`fixedPrice`. 24
`elasticPrice`. 24
3.3.4 Capital Models 25
`fixedCapital`. 25
`SCD`: Simple capital Dynamics. 25
3.3.5 Covariables Models 26
`fixedCovar`. 26
`ssb.get`. 26
3.3.6 Observation Models: Catch and biological parameters 26
`age2ageDat`. 26
`bio2bioDat`. 28
`age2bioDat`. 28
3.3.7 Observation Models: Population 28
`NoObsStock`. 28

perfectObs . . . 28
age2agePop . . . 29
bio2bioPop . . . 29
age2bioPop . . . 29
3.3.8 Observation Models: Abundance Indices . . . 29
ageInd . . . 30
bioInd . . . 30
NoObsInd . . . 30
3.3.9 Observation Model: Fleets . . . 30
3.3.10 Management Advice Models . . . 30
fixedAdvice . . . 30
annualTAC . . . 30
IcesHCR . . . 32
FroeseHCR . . . 32
annexIVHCR . . . 32
ghlHCR . . . 32
aneHCRE . . . 32
neaMAC_ltmp . . . 32
3.4 Fourth level functions . . . 32
3.4.1 Stock-Recruitment relationships . . . 32
3.4.2 Catch production functions . . . 34
CobbDouglasBio . . . 34
CobbDouglasAge . . . 35
3.4.3 Costs functions . . . 36
TotalCostsPower . . . 36

A New FLR - S4 classes **37**
A.1 FLBDsim class . . . 37
A.2 FLSRSim class . . . 38

B Graphical representation of FLR Objects **39**

1 Introduction

The idea of `FLBEIA` comes from many applications developed to perform bio-economic analysis in AZTI-Tecnalia to which pieces of code were re-written in order to match with a specific case study or fishery. These pieces, in many cases, reflect exactly the same processes with similar dynamics, that have to be slightly, adapted to these different case studies.

In order to avoid these cases we decided to develop not a model but a framework in which a model is built. This model can be constructed combining already existing functions or new ones can be constructed and combined with existing ones. This idea comes from the fact that there is no an universal model that can be applied to address all fishery management issues. The choice of the model to be used is dependent on the questions asked, which implies that any model can be considered valid for all purposes.

Furthermore, big advances have been seen in the last years in bio-economic modelling, in which models such as Fishrent [Salz et al., 2011], Fcube [Ulrich et al., 2011], FcubeEcon [Hoff et al., 2010] and many others can be cited. But also theoretical and sometimes partial assessments have been developed. In that sense, `FLBEIA` pretends not to create new models or processes but to integrate many of them in a common bio-economic impact assessment framework as a package of FLR [Kell et al., 2007].

`FLR` [Kell et al., 2007] was built with the goal of developing a common framework to facilitate collaboration within and across disciplines (e.g. biological, ecological, statistical, mathematical, economic, and social) and, in particular, to ensure that new modelling methods and software are more easily validated and evaluated, as well as becoming widely available once developed.

`FLBEIA` package is built on top of existing `FLR` packages. It is an `R` package [R Development Core Team, 2010] developed to conduct *B*io-*E*conomic *I*mpact *A*ssessments, that is, to identify the potential economic and biological consequences of a proposed policy action, to support policy making.

It has been built under Management Strategy Evaluation framework [Butterworth and Punt, 1999, Butterworth, 2007, De la Mare, 1998, Punt and Donovan, 2007, Rademeyer et al., 2007]. It contains a collection of functions and new `S4` classes developed to facilitate the simulation of fishery systems response to different types of management strategies.

The main characteristics of `FLBEIA` package are:

- It is coded in a generic, flexible and extensible way.
- Provides functions to condition the simulations, to run them and to analyze the results.

In fact, a mayor effort has been set on the second functionality, namely the simulation model.

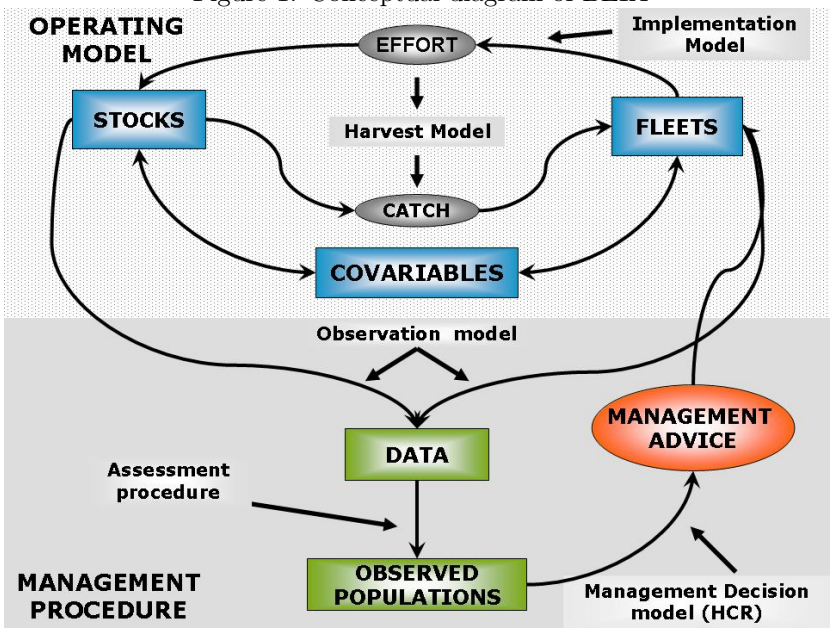
The main characteristics of the `BEIA` simulation model are:

- The model is fully biological-economic coupled and provides fully integrated bio-economic assessment.

- The model deals with multi-species, multi-fleet and multi-*metier* situations.
- The model can be run using seasonal steps (smaller or equal to one year).
- It is generic, flexible and extensible.
- Uncertainty can be introduced in almost any of the parameters used.

A conceptual diagram of the model is shown in figure 1. The simulation is divided in two main blocks: the Operating Model (OM) and the Management Procedure Model (MPM). The OM is the part of the model that simulates the real dynamics of the fishery system and the MPM is the part of the model that simulates the whole management process.

Figure 1: Conceptual diagram of BEIA



The OM has three components that can interact among themselves:

1. The biological populations or stocks.
2. The fleets.
3. The covariables. They can be of any nature; environmental, economical or technical.

The MPM has also three components:

1. The data collected from the OM.
2. The observed population obtained through the application of a set of assessment models to the observed data.

3. The management advice obtained from the application of harvest control rules (HCR) to the observed populations.

The model is built modularly with a top-down structure that has, at least, four levels:

1. In the first level (top level), there is only one function, `BEIA` function. It calls the functions on the second level in a determined order and it links the main components (stocks, fleets, covariates, data, observed population and management advice) of the OM and MPM.
2. The functions in the second level correspond with the components in the figure 1. The OM components project the objects one season forward: `biols.om` projects the stocks, `fleets.om` projects the fleets and `covars.om` projects the covariables. The MPM components generate the objects necessary to produce the management advice, they generate the objects based on OM objects and they operate at most once a year: `observation.mp` generates the data, `assessment.mp` generates the observed population and `advice.mp` generates the management advice. They take the input objects and returns only those related to the component they belong to.
3. The functions in the third level define the specific dynamics of each component and they are chosen by the user in each simulation. They are always called by a second level function and in some cases they call fourth level functions, for example a function that describes the dynamics of an age structured population can call a stock recruitment function. In this way, a function used to describe age structured populations can be combined with different stock recruitment relationships.
4. The functions in the fourth level are called by functions in the third level and are used to model the most basic processes in the simulation. They are coded as a function and selected by the user because it could be interesting to use the same third level function together with different fourth level functions, as in the case of age structured population and stock recruitment functions.

This top down structure allows us to avoid the classical structure of separated biological and economic (and social) modules (that could be integrated or not). When designing the model, we can think out only at what level do we want to include a particular characteristic, and this decision is independent of being a biological or an economic characteristic.

`FLBEIA` is prepared to incorporate new third and lower level components or to modify them, while first and second level ones are fixed. Changing first or second level functions would imply a different approach, but existing third and lower level functions would be usable.

In the next two sections, `BEIA`'s conceptual model and its specifications will be explained. The conceptual model will characterize the main components as well as the feedbacks and loops among them. The model specification, will describe the components, the functions by level, the currently available third and four level functions and how to use them within the `FLBEIA` package.

2 The Concept of BEIA

As commented above the simulation model has been divided in two main blocks, the Operating Model (OM) and the Management Procedure Model (MPM). This division is part of the requirements of the MSE approach, that is, mathematical representations of the *real world* (OM), the *observed world* (MPM) and the interactions between them.

2.1 Operating Model

The OM is the part of the model that simulates the real dynamics of the fishery system. It is divided in three components or operating models, the biological operating model, the fleets operating model, and the covariables operating model.

It runs in seasonal steps, projecting the components in each step. Firstly, the biological component is updated, secondly the fleet component and finally the covariable component.

Biological component: The biological component simulates the population dynamics of the biological populations, the stocks. The number of populations is, in principle, unlimited. The limitation could come from memory problems with R and/or the operating system. The stocks can be described as age structured populations or as biomass dynamic populations, length structured populations models are not supported by the simulation algorithm. Each stock can follow a different population dynamics model and is projected independently. It does not mean that there cannot interdepend between them but the order in which these biological components are updated has to be decided and it will affect the results obtained.

Fleet component: The fleet component simulates the behaviour and dynamics of the individual fleets. As the number of the stocks, the number of fleets is in principle unlimited. The limitation could come from memory problems with R and/or the operating system used. The activity of the fleets is divided into metiers. The metiers are formed by trips that share the same catchability for all the stocks caught. Fishing effort of the fleets and their effort share among metiers are independently updated for each fleet in each season.

There is also defined a capital dynamics that annually updates the capacity and/or catchability of the fleets according to their economic performance, independently for each fleet.

Covariates: This part of the model is intended to incorporate all the variables that are not part of the biological or fleet components and that affect any of the operating model components or the management process. The number of covariates is, in principle, unlimited. The limitation could come from memory problems with R and/or the operating system used.

Links among and within components: The links within the OM components are not restricted by the general settings of the simulation model. If the modeller decides to include these links, we will have:

- The link within the *biological* component is that catch affects abundance.
- The link within the *fleets* component is that fleets' capacity affects fishing effort.
- The link between the *biological and fleets* components is that fishing effort and fish abundance affects catches.

+++++ REPASAR ESTE PARRAFO SOBRE LINKS ES CONFUSO

Again, it should be reminded that the use or not of these links is under the discretion of the modeller.

2.2 The management procedure model

The Management Procedure Model (MPM) is divided into 3 components: the observation, the assessment and the management advice. The observation component produces the necessary data to run the assessment. Then, the assessment component is applied to the data to obtain the observed populations. Finally, the management advice component produces a management advice based on the observed populations. MPM procedure is applied yearly in the appropriate season of the year. Not necessarily in the last season. For example, it can be simulated management going from the mid-season of the one year to the mid-season of the next year (as occurs in the case of the Bay of Biscay anchovy). Performing multi-annual advice is also possible.

Observation component: The observation component generates the necessary objects to run the assessments. Three types of objects can be generated:

- Stocks.
- Fleets.
- Abundance indices.

Stocks and abundance indices objects are generated independently, stock by stock, whereas fleets are observed jointly. These observed objects are generated based on the components of the OM to which a variation is introduced. This variation can be due to:

- Introducing uncertainty to the OM variables, or
- adjusting the OM variables to the assessment model requirements which is going to be used in the next step (e.g. collapsing the dimensions -age, season,...)
- adjusting the OM variables to the legal conditionings (TACs, quotas, TAE, discards,...)

Assessment component: Assessment models are applied on a stock by stock basis and they can vary from stock to stock.

Management advice component: The management advice component produces a set of indicators (determined by the user) useful for policy making. The management advice is produced based on the output obtained from the observation and assessment components. The advice is first applied at single stock level and then (after that) it can be applied at fleet level.

3 Running BEIA

3.1 First level function: BEIA

The simulations are run using `BEIA` function. This function is a multistock, multifleet and seasonal simulation algorithm coded in a generic, flexible and extensible way. It is generic in the sense that can be applied to any case study that fit into the model restrictions. The algorithm is made up by third and fourth level functions specified by the user. Apart from existing functions new ones can be defined and used if necessary. This is the reason to define it as flexible and extensible.

To determine the simulation the third- and fourth-level functions must be specified. For this end, in the main function call, there is a control argument associated to each second level function. These control arguments are lists which include, apart from the name of the functions to be used in the simulations, any extra argument needed by those functions that are not contained in the main arguments.
`BEIA` function is called as follows:

```
BEIA(biols, SRs, BDs, fleets, covars, indices, advice,
        main.ctrl, biols.ctrl, fleets.ctrl, covars.ctrl,
         obs.ctrl, assess.ctrl, advice.ctrl)
```

Main arguments:

`biols` : An `FLBiols` object. The object must be named and the names must coincide with those used in `SRs` object, `BDs` object and `catches` slots within `FLFleetExts` object.

`SRs` : A list of `FLSRsim` objects. This object is a simulation version of the original `FLSR` object. The object must be named and the names must coincide with those used in `FLBiols` object. For details on this object see the figure in the annex.

`BDs` : A list of `FLBDsim` objects. This object is similar to `FLSRs` object but oriented to simulate population growth in biomass dynamic populations. The object must be named and the names must coincide with those used in `FLBiols` object. For details on this object see the figure in the annex.

`fleets` : An `FLFleetExts` object. This object is almost equal to the original `FLFleet` object but the `FLCatch` object in `catch` slot has been replaced by `FLCatchExt` object. The difference between `FLCatch` and `FLCatchExt` objects is that `FLCatchExt` has two extra slots `alpha` and `beta` used to store Cobb-Douglas production function parameters, α and β, [Cobb and Douglas, 1928, Clark, 1990]. α corresponds with

effort's exponent and β with that of biomass. The `FLFleetExts` object must be named and the names used must be consistently used in the rest of the arguments. For details on this object see the figure in the annex.

`covars` : An `FLQuants` object. This object is not used in the most basic configuration of the algorithm. Its content is totally dependent on the third or lower level functions that make use of it.

`indices` : A list of `FLIndices` objects. Each element in the list corresponds with one stock. The list must be named and the names must coincide with those used in `FLBiols` object.

`advice` : A list. The class and content of its elements depends on functions used in `fleet.om` to simulate fleets' effort and the functions used to produce advice in `advice.mp`.

Control arguments:

`main.ctrl` : Controls the behavior of the main function, `BEIA`.

`biols.ctrl` : Controls the behavior of the second level function `biols.om`.

`fleets.ctrl` : Controls the behavior of the second level function `fleets.om`.

`covars.ctrl` : Controls the behavior of the second level function `covars.om`.

`obs.ctrl` : Controls the behavior of the second level function `observation.mp`.

`assess.ctrl` : Controls the behavior of the second level function `assessment.mp`.

`advice.ctrl` : Controls the behavior of the second level function `advice.mp`.

3.2 Second level functions

3.2.1 Biological Component: `biols.om`

The call to the function within `BEIA` is done as:

```
biol.om(biols, fleets, SRs, BDs, covars, biols.ctrl, year,
                              season)
```

This function projects the stocks one season forward. The projection is done independently stock by stock by the third level function specified for each stock in `biols.ctrl` object. Currently, there are three population dynamics functions implemented, one corresponding to age structured populations, `ASPG`, the second one to biomass dynamic populations, `BDPG` and another one to fixed populations given as input, `fixedPopulation`. These functions do not include predation among stocks, but this kind of models could be implemented and used in the algorithm if necessary.
Control arguments:

`biols.control` : This argument is a list which contains the necessary information to run the third level functions that are called by `biols.om`. The elements depend on the third and lower level functions

used to describe the dynamics of the stocks. The list must contain at least one element per stock and the name of the element must coincide exactly with the name used in `biols` argument so it can be used to link the population with its dynamics model. At the same time, each of these elements must be a list with at least one element, `dyn.model`, which specifies the name of the function used to describe population dynamics.

When only `ASPG` and/or `BDPG` functions are used the `biols.ctrl` object is just a list with one element per stock. And these elements are lists with just one element, `dyn.model`, specifying the name of the function used to describe population dynamics, `ASPG`, `BDPG` or `fixedPopulation`. For example:

```
> biols.ctrl

$NHKE
$NHKE$dyn.model
[1] "ASPG"


$CMON
$CMON$dyn.model
[1] "BDPG"


$FAKE
$FAKE$dyn.model
[1] "ASPG"
```

3.2.2 Fleets Component: `fleets.om`

The call to `fleets.om` function within `BEIA` is done as:

```
fleets.om(fleets, biols, covars, advice,
       fleets.ctrl, year, season)
```

This function projects the fleets one season forward. The main argument, `fleets`, is an object of class `FLFleetsExt` (for more detail see section 3.1)

The function is divided in three processes related to fleet dynamics: the effort model, the price model and the capital model. Effort and capital models are fleet specific, whereas price model is fleet and stock specific. First, `fleets.om` calls the effort model and it updates the slots related to effort and catch. The effort models are called independently fleet by fleet. Then, `fleets.om` calls the price model in fleet by fleet and stock by stock basis. The price model updates the `price` slot in the `fleets` object. Finally, the function calls the capital model, but the call is done in the last season of the year. Thus, investment and disinvestment is only done annually. The capital model is called independently fleet by fleet.

Effort model: This part of the model simulates the tactical behaviour of the fleet every season and iteration. In each time step and iteration, the effort exerted by each individual fleet and its effort-share

among metiers is calculated depending on the stock abundance, management restrictions or others. After that, the catch produced by the combination of effort and effort-share is calculated and `discards, discards.n, landings, landings.n` slots are filled. Other variables stored in `fleets.ctrl` could also be updated here, for example `quota.share`, as a result of the exerted effort.

The effort model is specified at fleet level, so each fleet can follow a different effort model. At the moment there are 4 functions available, `fixedEffort, SMFB, SSFB` and `MaxProf.StkCnst`. To write new functions for effort, it must be taken into account that the input arguments must be found among `fleets.om` function arguments and that the output must be a list with updated `FLFleetsExt` and `fleet.ctrl` objects, i.e.:

```
list(fleets = my_fleets_obj, fleet.ctrl = my_fleet.ctrl_obj)
```

Price Model: The price model updates the price-at-age at stock, metier and fleet level in each time step and iteration.

At the moment, there are 2 functions available, `fixedPrice` and `elasticPrice`. To write new functions for price it must be taken into account that the input arguments must be found among `fleets.om` function arguments and that the output must be a list with an updated `FLFleetsExt` object.

Capital Model: This module is intended to simulate the strategic behaviour of the fleets, namely, the investment and disinvestment dynamics. The model is applied at fleet level and in an annual basis and can affect fleets' capacity and catchability. Catchability could be modified through investment in technological improvement and capacity as a result of an increase (investment) or decrease (disinvestment) in the number of vessels. Changes in fleets' capacities could produce a variation in quota share among fleets, for example. Thus, the corresponding change would have to be done in `fleets.ctrl` object.

At the moment, there are 2 functions available, `fixedCapital` and `SCD`. To write new functions for capital dynamics, as for effort and price, it must be taken into account that the input arguments must be found among `fleets.om` function arguments and that the output must be a list with updated `FLFleetsExt` and `fleets.ctrl` objects.

Control arguments:

`fleets.ctrl` : The most simple example of fleet dynamics model and hence the most simple `fleets.ctrl` object correspond with the model where all the parameters in `fleets` object are given as input and maintained fixed within the simulation. This is obtained using the third level functions, `fixedEffort, fixedPrice` and `fixedCapital` which do not need any extra arguments. In the case of two fleets, `FL1` and `FL2`, where `FL1` catches 3 stocks, `ST1, ST2` and `ST3` and `FL2` catches `ST1` and `ST3` stocks, the `fleets.ctrl` could be created using the following code:

```
>fleets.ctrl <- list()

# The fleets
>fleets.ctrl[['FL1']] <- list()
>fleets.ctrl[['FL2']] <- list()

# Effort model per fleet.
>fleets.ctrl[['FL1']]$effort.dyn <- 'fixedEffort'
>fleets.ctrl[['FL2']]$effort.dyn <- 'fixedEffort'

# Price model per fleet and stock.
>fleets.ctrl[['FL1']][['ST1']]$price.dyn  <- 'fixedPrice'
>fleets.ctrl[['FL1']][['ST2']]$price.dyn  <- 'fixedPrice'
>fleets.ctrl[['FL1']][['ST3']]$price.dyn  <- 'fixedPrice'

>fleets.ctrl[['FL2']][['ST1']]$price.dyn  <- 'fixedPrice'
>fleets.ctrl[['FL2']][['ST3']]$price.dyn  <- 'fixedPrice'

# Capital model by fleet.
>fleets.ctrl[['FL1']]$capital.dyn  <- 'fixedCapital'
>fleets.ctrl[['FL2']]$capital.dyn  <- 'fixedCapital'

> fleets.ctrl
$FL1
$FL1$effort.dyn
[1] "fixedEffort"

$FL1$ST1
$FL1$ST1$price.dyn
[1] "fixedPrice"


$FL1$ST2
$FL1$ST2$price.dyn
[1] "fixedPrice"


$FL1$ST3
$FL1$ST3$price.dyn
[1] "fixedPrice"


$FL1$capital.dyn
[1] "fixedCapital"


$FL2
$FL2$effort.dyn
[1] "fixedEffort"
```

```
$FL2$ST1
$FL2$ST1$price.dyn
[1] "fixedPrice"


$FL2$ST3
$FL2$ST3$price.dyn
[1] "fixedPrice"


$FL2$capital.dyn
[1] "fixedCapital
```

3.2.3 Covariables Component: `covars.om`

`covars.om` projects `covars` object one season forward. `covars` object is a named list and the class and dimension of each element will depend on the function used to project it into the simulation.

The call to `covars.om` function within `BEIA` is done as:

```
covars.om(biols, fleets, covars, advice,
       covars.ctrl, year, season)
```

Internally, for each element in the `covars` list, it calls to the third level functions specified in the `covars.ctrl` object. At the moment, there exist 2 third level functions: `fixedCovar`, which is used to work with variables that are input parameters not updated within the simulation and `ssb.get`, which is used to get the real abundance of one of the simulated stocks.

This way of working could be useful, for example, for environmental variables such as sea surface temperature that could affect catchability or recruitment in the fleet and biological operating models respectively and that are external to fishery system.

A covariable with a non-trivial dynamics could be the abundance of certain animal which is not commercially exploited by the fleet, but which abundance affects the natural mortality of any of the exploited stocks. In this case, 2 extra functions will be needed, the function that defines the dynamics of the covariable and the function that models the natural mortality of the stock as a function of the abundance of the animal. The first function should be declared in `covars.ctrl` argument and the former one in `biols.ctrl` argument as a stock dynamics model.
Control arguments:

`covars.ctrl` : This argument is a named list with one element per covariable and the names of the list must match those used to name the `covars` object. Each of the elements is, at the same time, a list with, at least, one element, `dyn.model`, which defines the dynamics of the covariable in question.

3.2.4 Observation Component: `observation.mp`

The observation component generates the necessary data to run the assessment models. The main function is `observation.mp` and it calls third level functions which generate 3 possible objects, a `FLStock`, a `FLIndices` or a `FLFleetsExt` object. The `FLStock` and `FLIndices` objects are generated independently for each stock and the `FLFleetsExt` object jointly for all the fleets.

The call to `observation.mp` function within `BEIA` is done as:

```
observation.mp(biols, fleets, covars, indices, advice,
                    obs.ctrl, year)
```

The output of `observation.mp` is a list with 3 elements. The first element, `stocks`, is a named list with one element per stock and its names correspond with those used in the `biols` object. The elements of the `stocks` list are of class `FLStock` or `NULL`, if a `FLStock` is not needed to run the assessment. The second element, `indices`, is a named list with one element per stock and its names correspond with those used in `biols` object. The elements of the `indices` list are of class `FLIndices` or `NULL`, if a `FLIndices` is not needed to run the assessment. The third element, `fleets.obs`, is an observed version of the original `fleets` object. The segmentation of the fleet in the observed version would be different to the real one (in this moment there is no third level function implemented to generate observed fleets). +++++ LO VAMOS A IMPLEMENTAR A CORTO PLAZO O CAMBIAMOS LA REDACCION DEL TEXTO?

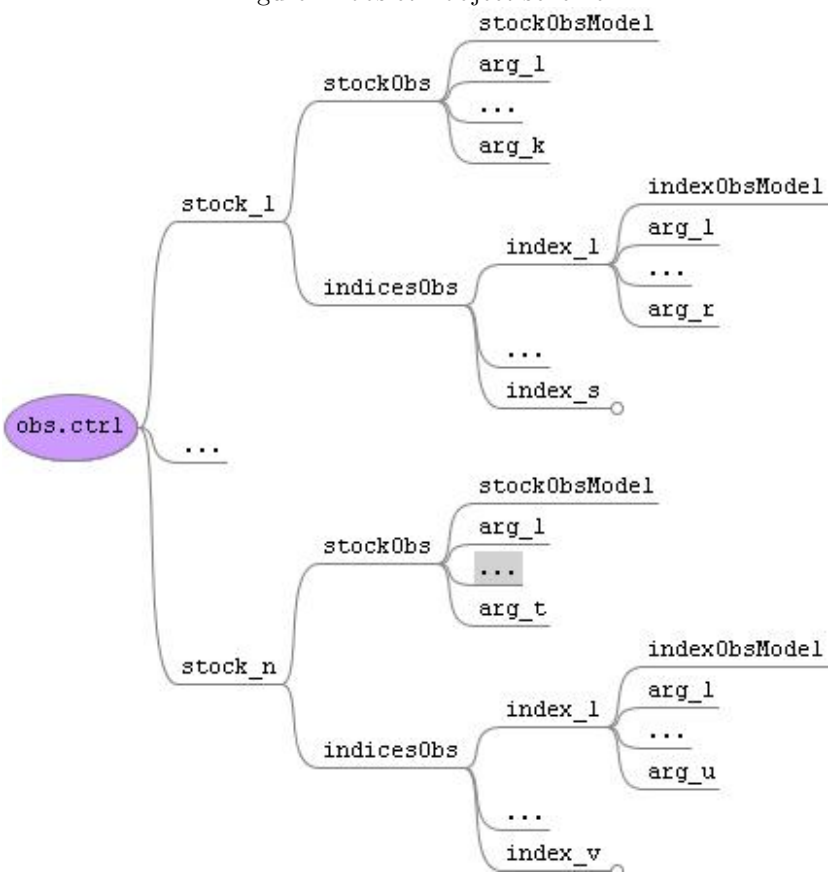
As the management process is currently run in a yearly basis, the `unit` and `season` dimensions are collapsed in all the observed objects. Moreover, if the management process is being conducted at the end of year `y` the observed objects extend up to year `y-1`, whereas they extend up to year `y` in the cases when management process is conducted in any other season as it happens in reality.

Control arguments:

`obs.ctrl` : This argument is a list with one element per stock. If fleets were observed the object should have also one element per fleet but as at the moment there are no functions that provide observed version of `FLFleetsExt` object this option is not described here. The `obs.ctrl` object must be a named list where the names used correspond with those used in the `FLBiols` object. Each stock element is, at the same time, a list with two elements (`stockObs` and `indicesObs`) and this two elements are once again lists. A scheme of `obs.ctrl` object is presented in Figure 2. +++++ REVISAR FIGURA (notación incorrecta)

The `stockObs` element is a list with the arguments necessary to run the third level function used two generate the `FLStock` object. In the list there must be at least one element, `stockObsModel`, with the name of the third level function that will be used two generate the `FLStock` object. If it is not required to generate a `FLStock` object `NoObject` should be assigned to `stockObsModel` argument and this function will return the `NULL` object.

Figure 2: obs.ctrl object scheme



The `indicesObs` element is a list with one element per index in the `FLIndices` object. Each element of the list is, at the same time, a list with the arguments necessary to run the third level function used to generate the `FLIndex` object. In the list there must be at least one element, `indexObsModel`, with the name of the third level function that will be used two generate the `FLIndex` object. If it is not required to generate a `FLIndices` object `indicesObs` element will be set equal to `NoObject` instead of a list and this will return the `NULL` object instead of a `FLIndices` for the corresponding stock.

3.2.5 Assessment Component: `assessment.mp`

The assessment component applies an existing assessment model to the stock data objects generated by the observation model (`FLStock` and `FLIndices`). The assessment models are applied stock by stock, independently.

The call to `assessment.mp` function within `BEIA` is done as follows:

```
assessment.mp(stocks, fleets.obs, indices, assess.ctrl,
                                datayr)
```

The output of the function is a list of `FLStocks` with `harvest`, `stock.n` and `stock` slots updated. Within `FLBEIA` no new assessment models are provided, but the models already available in `FLR` can be used.
Control arguments:

`assess.ctrl` : This argument is a named list with one element per stock, where the names must coincide with those used in `biols` object. The elements must have at least one element, `assess.model`, which defines the name of the assessment model to be used for each stock. Furthermore, if the assessment model to be used is non-trivial (not `NoAssessment`), the list must contain a second argument `control` with the adequate control object to run the assessment model.

3.2.6 Management Advice Component: `advice.mp`

The Management Advice component generates an advice based on the output of assessment and/or observation components.

The call to `advice.mp` function within `BEIA` is done as follows:

```
advice.mp(stocks, fleets.obs, indices, covars, advice,
                advice.ctrl, year, season
```

First, the advice is generated stock by stock, independently. Later a function that generates advice based on the single stock advices, observed fleets and others could be applied, `FCube` like approaches [Ulrich et al., 2011]. The output of the function is an updated `advice` object.

Depending on the structure of the third level functions used to generate advice and to simulate fleet dynamics, the advice could be an input advice (effort, temporal closures, spatial closures -implicitly through changes in catchability-...) or an output advice (catch).

`advice` **object** : The structure of this object is open and it is completely dependent on the third level functions used to describe fleet dynamics and to generate the advice. For example, if `SMFB` and `annualTAC` are used to describe fleet dynamics and generate the advice respectively, `advice` is a list with two elements, `TAC` and `quota.share`. `TAC` is an annual `FLQuant` with the `quant` dimension used to store stock specific TACs and, `quota.share` is a named list with one element per stock being the elements `FLQuant`-s with `quant` dimension used to store fleet specific annual quota share.

Control arguments:

`advice.ctrl` : This argument is a named list with one element per stock and one more element for the whole fleet. The names must coincide with those used to name `biols` object and the name of the extra argument must be `fleets`. The elements of the list are, at the same time, lists with at least one element, `HCR`, with the name of the model used to generate the single stock and fleet advice depending on the case.

3.3 Third level functions

3.3.1 Population growth functions

`fixedPopulation`: Fixed Population function. In this function all the parameters are given as input, because there is not any population dynamics simulated.

`ASPG`: Age Structured Population Growth function. The function `ASPG` describes the evolution of an age structured population using an exponential survival equation for existing age classes and a stock-recruitment relationship to generate the recruitment. The recruitment can occur in one or more seasons. However, the age is measured in integer years and the seasonal cohorts are tracked separately. The seasonal cohorts and their corresponding parameters are stored in the '`unit (u)`' dimension of the `FLQuant`-s. And all the individuals move from one age group to the following one in the 1st of January. Thus, being ϕ the recruitment function, RI the reproductive index, N the number of individuals, M the natural mortality, C the catch, a_0 the age at recruitment, s_0 the season when the recruitment was spawn, and a, y, u, s the subscripts for age, year, unit and season respectively, the population dynamics can be written mathematically as:

If $s = 1$,

$$N_{a,y,u,1} = \begin{cases} \phi\left(RI_{y=y-a_0,s=s-s_0}\right) & a = a_0 \\ (N_{i_a} \cdot e^{-\frac{M_{i_a}}{2}} - C_{i_a}) \cdot e^{-\frac{M_{i_a}}{2}} & a_0 < a < A \\ (N_{i_{A-1}} \cdot e^{-\frac{M_{i_{A-1}}}{2}} - C_{i_{A-1}}) \cdot e^{-\frac{M_{i_{A-1}}}{2}} + & \\ (N_{i_A} \cdot e^{-\frac{M_{i_A}}{2}} - C_{i_A}) \cdot e^{-\frac{M_{i_A}}{2}} & a = A, \end{cases} \tag{1}$$

where $i_a = (a-1, y-1, u, ns)$, $i_{A-1} = (A-1, y-1, u, ns)$ and $i_A = (A, y-1, u, ns)$.

If $s > 1$,

$$N_{a,y,u,s} = \begin{cases} \phi\left(RI_{y=y-a_0,s=s-s_0}\right) & a = a_0 \\ (N_{i_a} \cdot e^{-\frac{M_{i_a}}{2}} - C_{i_a}) \cdot e^{-\frac{M_{i_a}}{2}} & a_0 < a < A \end{cases} \tag{2}$$

where $i_a = (a, y, u, s-1)$

And the reproductive index RI is given by:

$$RI_{y-a_0,s} = \sum_a \sum_u (N \cdot wt \cdot fec \cdot spwn)_{a,y-a_0,u,s} \tag{3}$$

where wt is the mean weight, fec +++++ and $spwn$ +++++.

+++++ DORLETA: por favor revisa la formula anterior (e.g. $SSB = \sum_a \sum_u (N \cdot wt \cdot fec \cdot exp - (M \cdot M_{spwn} + F \cdot F_{spwn}))$), otros ejemplos?????

Depending on what is stored in the fec slot, RI can be SSB or any other reproductive index of the population. The stock-recruitment relationship ϕ is specified in the `model` slot of corresponding `FLSRsim` object. `FLSRsim` object enables modeling a great variety of stock-recruitment relationships depending on its functional form and seasonal dynamics.

`BDPG`**: Biomass Dynamic Population Growth function.** The function `BDPG` describes the evolution of a biomass dynamic population, i.e. a population with no age, stage or length structure. The population is aggregated in biomass, B, and the growth of the population, g is a function of the current biomass and the catch C. The model is mathematically described in equation 4:

$$B_{s,y} = \begin{cases} B_{s-1,y} + g(B_{s-1,y}) - C_{s-1,y} & s \neq 1, \\ B_{ns,y-1} + g(B_{ns,y-1}) - C_{ns,y-1} & s = 1. \end{cases} \tag{4}$$

NOTATION!!! Because `BEIA` is seasonal the equation depends on season. The growth model g and its parameters are specified, respectively, in the `model` and `params` slot of corresponding `FLBDsim` class. Currently only Pella and Tomlinson model [Pella and Tomlinson, 1969] is implemented to model growth, but new models can be defined if needed. The following parameterization of the growth model has been implemented:

$$g(B) = B \cdot \frac{r}{p} \cdot \left[1 - \left(\frac{B}{K}\right)^p\right] \tag{5}$$

3.3.2 Effort models

`fixedEffort`**: Fixed Effort model.** In this function all the parameters are given as input except discards and landings (total and at age). The only task of this function is to update the discards and landings (total and at age) according to the catch production function specified in `fleets.ctrl` argument.

Two arguments need to be declared as elements of `fleets.ctrl` if this function is used, `effort.dyn = 'fixedEffort'` and `catch.equation`. The last argument is used to specify the catch production function that will be used to generate the catch. Note that both arguments must be declared at fleet level (i.e `fleets.ctrl[[fleet.name]]$effort.dyn` and `fleets.ctrl[[fleet.name]]$catch.model`) and that catch production model corresponds with a fourth level function.

`SMFB`**: Simple Mixed Fisheries Behavior.** This model is a simplified version of the behavior of fleets that work in a mixed fisheries framework. The function is seasonal and assumes that effort share among metiers is given as input parameter.

In each season, the effort of each fleet, f, is restricted by the seasonal landing quotas or catch quotas of the stocks that are caught by the fleet. The following steps are followed in the calculation of effort:

1. Compare the overall seasonal quota, $\sum_f Q_{f,s,st} \cdot TAC$, with the abundances of the stocks. If the ratio between overall quota and abundance exceeds the seasonal catch threshold, $\gamma_{s,st}$, reduce the quota share in the same degree. Mathematically:

$$Q'_{f,s,st} = \begin{cases} Q_{f,s,st} & \text{if } \frac{\sum_f Q_{f,s,st} \cdot TAC}{B_{s,st}} \leq \gamma_{s,st}, \\ Q_{f,s,st} \cdot \frac{B_{s,st} \cdot \gamma_{s,st}}{\sum_f Q_{f,s,st} \cdot TAC} & \text{otherwise.} \end{cases} \tag{6}$$

2. According to the catch production function calculate the efforts corresponding to the landing or catch quotas, $Q'_{f,s,st} \cdot TAC$, of the individual stocks, $\{E_{f,s,st_1}, \ldots, E_{f,s,st_n}\}$.
3. Based on the efforts calculated in the previous step, calculate an unique effort, $E_{f,s}$. To calculate this effort there are the following options:

   `max:` The maximum among possible efforts, $\hat{E}_{f,s} = \max_{j=1,\ldots,n} E_{f,s,st_j}$

   `min:` The minimum among possible efforts, $\hat{E}_{f,s} = \min_{j=1,\ldots,n} E_{f,s,st_j}$

   `mean:` The mean of possible efforts, $\hat{E}_{f,s} = \text{mean}_{j=1,\ldots,n} E_{f,s,st_j}$

   `previous:` The effort selected is the effort most similar to previous year effort on that season,

   $$\hat{E}_{f,s} = \left\{ E_{f,s,st} : \left| 1 - \frac{E_{f,s,st}}{E_{f,y-1,s}} \right| = \min_{j=1,\ldots,n} \left| 1 - \frac{E_{f,s,st_j}}{E_{f,y-1,s}} \right| \right\}$$

   `stock.name:` The effort corresponding to `stock.name` is selected: $\hat{E}_{f,s} = E_{f,s,\texttt{stock.name}}$
4. Compare the effort, $\hat{E}_{f,s}$, with the capacity of the fleet, κ_f (capacity must be measured in the same units as effort and it must be stored in the `capacity` slot of the `FLFLeetsExt` object). If the capacity is bigger, then the final effort is unchanged and if the capacity is smaller, the effort is set qual to the capacity, i.e.:

   $$E_{f,s} = \begin{cases} \kappa_f & \text{if } \kappa < \hat{E}_{f,s}, \\ \hat{E}_{f,s} & \text{if } \kappa \geq \hat{E}_{f,s}. \end{cases} \tag{7}$$
5. The catch corresponding to the effort selected is calculated for each stock and compared with the corresponding quota. If the catch is not equal to the quota and the season is not the last one, the seasonal quota shares of the rest of the seasons are reduced or increased proportionally to their weight in the total share. The shares are changed in such a way that the resultant annual quota share is equal to the original one. In case the difference between actual catch and that corresponding to the quota exceeds the quota left over in the rest of the seasons, the quota in the rest of the seasons is canceled. Mathematically for season i where $s \leq i \leq ns'$:

   $$Q''_{f,i,st} = \max\left(0, Q'_{f,i,st} + (Q'_{f,s,st} - Q''_{f,s,st}) \cdot \frac{Q'_{f,i,st}}{\sum_{j>s} Q'_{f,j,st}}\right) \tag{8}$$

   where Q' denotes the quota share obtained in the first step and Q'' the new quota share.

The `fleets.ctrl` argument in `SMFB` function

`SMFB` function requires several arguments at global and fleet level that are described below.

Global arguments:

`catch.threshold` : This element is used to store $\gamma_{s,st}$ parameter described in the first step of `SMFB` function algorithm. The elements must be a `FLQuant` object with dimension `[stock = nstk, year = ny, unit = 1, season = ns, area = 1, iter = nit]`, where the names in the first dimension must match with those used to name `FLBiols` object. Thus, the thresholds may vary between stocks, seasons, years and iterations. The elements of the object are proportions between 0 and 1 that indicate the maximum percentage of the stock that can be caught in each season. The reason to use this argument is that it is reasonable to think that it is impossible to fish all the fish in the sea. Thus, although the TAC is very large the actual catch will be restricted to $\gamma_{s,st} \cdot B_{s,st}$.

`seasonal.share` : A named `FLQuants` object, one per stock, with the proportion of the fleets' TAC share that 'belongs' to each season, so the sum along seasons for each fleet, year and iteration is equal 1. The elements must be `FLQuant` objects with dimension `[fleet = nf, year = ny, unit = 1, season = ns, area = 1, iter = nit]`, where the names in the first dimension must match with those used to name `FLFleetsExt` object. The names of the `FLQuants` must match stock names.

Fleet level arguments:

`effort.dyn` : `'SMFB'`.

`effort.restr` :`'max'`, `'min'`, `'mean'`,`'previous'` or `'stock.name'` (the name of one of the stocks caught by the fleet).

- `max` : The fleet will continue fishing until the catch quotas of all the stocks are exhausted.
- `min` : The fleet will stop fishing when the catch quota of any of the stocks is exhausted.
- `previous` : Among the efforts obtained under each stock restriction the effort most similar to the previous year effort will be selected.
- `stock` : The fleet will continue fishing until the catch quota of '`stock`' is exhausted. (This could correspond, for example, with a situation where the catch of one stock is highly controlled.)

These options are explained mathematically above when the `SMFB` function is described step by step.

`restriction` : `'catch'` or `'landings'`. Are the efforts calculated according to catch or landings restriction? *(for the moment only catch restriction is available).*

Fleet/Stock level arguments:

`catch.model` : The name of the fourth level function which gives the catch production given effort and biomass (aggregated or at age). The function must be coherent with `SMFB` and the function used to simulate the population growth. At the moment, two functions are available `CobbDouglasAge` and `CobbDouglasBio`.

`SSFB`**: Simple Sequential Fisheries Behavior.** Simple sequential fisheries behaviour is related to those fleets whose the fishing profile changes with the season of the year. `SSFB` function models the behaviour of fleets that work in a sequential fisheries framework. It is assumded that, in each season, the fleet, f, has only one target species or stock, st, thus the metier, m, is defined on the basis of the season and target species, resulting only in one target species per each metier.

In each season s, the effort allocated to each species st or metier m follows the historical trend (in order to capture the seasonality of each species fishing season), but is restricted to the remaining catch quota of the fleet.

Therefore, production function is applied at metier level, but the production has some restrictions, in both catches C and effort E, that are described through following steps:

1. Calculate the total quota that corresponds to each fleet, CQ, from the historical data and estimate remaining quota for the fleet, $RQ_{s,f,st}$, deducting the catches from previous seasons.

$$RQ_{s,f,st} = CQ_{f,st} - \sum_{ss<s} C_{ss,f,st} = TAC \cdot QS_{f,st} - \sum_{ss<s} C_{ss,f,st}$$

   Where QS is the quota share and C are the catches.

2. Compare the total remaining quotas with the abundances of the stocks. If the ration between remaining quotas and abundance exceeds the seasonal catch thershold, $\gamma_{s,st}$, reduce the remaining quota in the same degree.

$$RQ'_{s,f,st} = \begin{cases} RQ_{s,f,st} & \text{, if } \frac{\sum_f Q_{s,f,st}}{Bs,st} \leq \gamma_{s,st}; \\ RQ_{s,f,st} \cdot \frac{Bst,s \cdot \gamma_{s,st}}{\sum_f RQ_{s,f,st}} & \text{, otherwise.} \end{cases}$$

3. Initially expected effort, $\hat{E}_{s,f}$, is shared between different metiers (i.e. species) month by month on the basis of historical seasonal effort pattern.

$$\hat{E}_{s,m,st} = \hat{E}_{s,f} \cdot E_{s,m} = \kappa_f \cdot PED_{s,f} \cdot E_{s,m}$$

   Where $E_{s,m}$ is the effort share by metier, $PED_{s,f}$ is the percentage of effective days and κ_f is the fleet's capacity.

4. Expected catches ,$\hat{C}_{s,m,st}$, corresponding to that initial effor, are calculated through tha catch production function at metier and stock level, seasonally. The production function is the well-known Cobb Douglas.

5. If the expected catches resulting from the previous step is higher than the remaining quota corresponding to each metier (step 2), there is extra effort which has to be reallocated between the other species.

   If $\hat{C}_{s,f,st} > RQ_{s,f,st} \Rightarrow \hat{C}_{s,f,st} = RQ_{s,f,st} \Rightarrow E_{s,m,st} < \hat{C}_{s,m,st}$;

   else $\hat{C}_{s,f,st} \leq RQ_{s,f,st} \Rightarrow \hat{C}_{s,f,st} = C_{s,f,st} \Rightarrow E_{s,m,st} = \hat{C}_{s,m,st}$.

6. The reallocation of remaining effort, $\hat{E}_{s,m,st} - E_{s,m,st}$, can be performed in differnt ways:
   - Proportionally to the price and availability of the species in a given season
   - Proportionally to the effort allocated to the remaining metiers
7. This is repeated stock by stock until no effort remains to be allocated or all the TACs are exhausted

+++++ SONIA: comprobar si descripción del objeto ADVICE por si redundancia The `advice` argument in `SSFB` function

`quota.share`: A named `FLQuants` object, one per stock, with the total proportion of TAC that 'belongs' to each fleet each year. The '`fleet`' dimension names must match fleets' names. And the `FLQuants` must match stock names. For each and iteration the sum of the proportions must be equal to 1. (`FLQuant (fleet = nf, year = ny, unit = 1, season = 1, area =1, iter =1)`).

+++++
The `fleets.ctrl` argument in `SSFB` function:
Global arguments:

`catch.threshold` : A `FLQuant` object with dimension `[stock=nst, year = ny, unit = 1, season = ns, area =1, iter =nit]`, which contains the proportion of biomass that total catch of stock cannot exceed, i.e. the previously mentioned $\gamma_{s,st}$ parameter.

Fleet level arguments:

`fleet.dyn` : '`SSFB`'

`restriction` : '`catch`'. Relate to quota threshold.

`effectiveDay.perc` : A `FLQuant` object with dimension `[quant=1, year = ny, unit = 1, season = ns, area =1, iter =nit]`, which contains the proportion of days expected to be effective in a season (i.e. in which the fleet will go out fishing), the previously mentioned PED parameter.

`effort.realoc` : `NULL` or '`curr.eff`'. Element used to describe how does the remaining effort have to be reallocated between the rest of the metiers which already have remaining quota for the target stock.

`NULL` : The same proportion to each metier.

`month.price` : Proportionally to the expected effort share.

`MaxProfit.stkCnst`**: Maximization of profit under a TAC constraint.** Calculates the total effort and the effort share along metiers that maximizes the profits of the fleet. The maximization is done under two constrainst (the capacity of the fleet and the quota-share of one of the stocks caught by the fleet):

- The total effort can be higher than the fleet capacity (measured in the same units as the effort).

- The catch of one of the stocks can exceed the quota share of that stock.

Objective function:

$$\max_{E,\gamma_1,\ldots,\gamma_{nmt}} \sum_m \sum_{st} \sum_a \left(q_{m,st,a} \cdot B_{st,a}^{\beta_{m,st,a}} \cdot (E \cdot \gamma_m)^{\alpha_{m,st,a}} \right) \cdot pr_{m,st,a} - E \cdot \gamma_m \cdot VC_m - FC \tag{9}$$

+++++ DORLETA: describir los parámetros de la ecuación
With the constraints:

$$\begin{aligned} & 0 \leq \gamma_i \leq 1, \\ & E \leq \kappa, \\ & C_{st} \leq QS_{st} \end{aligned}$$

3.3.3 Price Models

`fixedPrice.` The prices are given as input data and are unchanged within the simulation. Only the function name, `FixedPrice`, must be specified in `price.dyn` element in `fleets.ctrl` object.

```
fleets.ctrl[[fleet.name]][[stock.name]][['price.dyn']] <-
        'FixedPrice'
```

`elasticPrice.` This function implements the price function used in Kraak et al. [2004]:

$$P_{a,y,s,f} = P_{a,0,s,f} \cdot \left(\frac{L_{a,0,s,f}}{L_{a,y,s,f}} \right)^{e_{a,s,f}} \tag{10}$$

This function uses base price, $P_{a,0,s,f}$, and base landings, $L_{a,0,s,f}$ to calculate the new price $P_{a,y,s,f}$ using a elasticity parameter $e_{a,s,f}, (e \geq 0)$. If the base landings are bigger than current landings the price is increased and decreased if the contrary occurs. a, y, s, f correspond to the subscripts for age, year, season and fleet respectively. For simplicity, the iteration subscripts has been obviated, but all the elements in the equation are iteration dependent. As prices could depend on total landings instead of on fleet's landings, there is an option to use $L_{a,0,s}$ instead of $L_{a,0,s,f}$ in the formula above.

Although price is stored at metier and stock level in `FLFleetsExt`, this function assumes that price is common to all metiers within a fleet and it is calculated at fleet level.

The `fleets.ctrl` argument in `fixedPrice` function

When `elasticPrice` is used, the following arguments must be specified, at fleet and stock level (i.e. `fleets.ctrl[[fleet.name]][[stock.name]]`):

`price.dyn` : `'fixedPrice'`.

`pd.Pa0` : An array with dimension `[age = na, season = ns, iter = it]` to store base price, P_{a0sf}.

`pd.La0` : An array with dimension `[age = na, season = ns, iter = it]` to store base landings, L_{a0sf}.

`pd.els` : An array with dimension `[age = na, season = ns, iter = it]` to store price elasticity, e_{asf}.

`pd.total` : Logical. If `TRUE` the price is calculated using total landings and if `FALSE` the landings of the fleet in question are used to estimate the price.

3.3.4 Capital Models

`fixedCapital`. The capacity and catchability are given as input data and are unchanged within the simulation. Only the function name, `FixedCapital`, must be specified in `capital.dyn` element of `fleets.ctrl` object.

```
fleets.ctrl[[fl.name]][[stk.name]][['capital.dyn']] <-
                'FixedCapital'
```

`SCD`: Simple capital Dynamics. In this simple function catchability is not updated, it is an input parameter, and only capacity is updated depending on some economic indicators. The following variables and indicators are defined at fleet and year level (fleet and year subscripts are omitted for simplicity):

FuC : Fuel Cost.

CrC : Crew Cost.

VaC : Variable Costs.

FxC : Fixed Costs (repair, maintenance and other).

CaC : Capital Costs (depreciation and interest payment).

Rev : Revenue:

$$Rev_f = \sum_m \sum_s \sum_a L_{m,s,a} \cdot P_{a,s}$$

where L is the total landings, P the price and m, s, a the subscripts for metier, season and age respectively.

BER : Break Even Revenue, the revenues that make profit equal to 0.

$$BER = \frac{FxC + CaC}{1 - \frac{FuC}{Rev} - \frac{CrC}{Rev - FuC} + \frac{FuC \cdot CrC}{Rev \cdot (Rev - FuC)} - \frac{VaC}{Rev}}$$

In principle the investment, Inv, is determined by:

$$Inv_0 = \frac{Rev - BER}{Rev}$$

But not all the profits are dedicated to increase the fleet, thus:

$$Inv = \eta \cdot \frac{Rev - BER}{Rev}$$

where η is the proportion of the profits that is used to buy new vessels. Furthermore, investment in new vessels will only occur if the operational days of existing vessels is equal to maximum days. If this occurs, the investment/disinvestment decision, Ω, will follow the rule below:

$$\Omega_y = \begin{cases} Inv & \text{if } Inv_0 < 0 \text{ and } \eta \cdot |Inv_0| < \omega_1, \\ -\omega_1 * \kappa_{y-1} & \text{if } Inv_0 < 0 \text{ and } \eta \cdot |Inv_0| > \omega_1, \\ Inv & \text{if } Inv_0 > 0 \text{ and } \eta \cdot |Inv_0| < \omega_2, \\ \omega_2 * \kappa_{y-1} & \text{if } Inv_0 > 0 \text{ and } \eta \cdot |Inv_0| > \omega_2. \end{cases} \tag{11}$$

where ω_2 stands for the limit on the increase of the fleet relative to the previous year, and ω_1 for the limit on the decrease of the fleet relative to the previous year.

3.3.5 Covariables Models

`fixedCovar.` The covariables that follow this model are given as input data and are unchanged within the simulation. Only the function name, `FixedCovar`, must be specified in `covar.dyn` element of `covars.ctrl` object.

```
covars.ctrl[[covar.name]]<- 'FixedCovar'
```

`ssb.get.` +++++ SONIA: completar

3.3.6 Observation Models: Catch and biological parameters

The functions in this section are used to generate a `FLStock` from `FLBiol` and `FLFleetsExt` objects. The former is used to fill the slots relative to biology, (`***.wt`, `mat` and `m` slots), and the last to fill the slots relative to catch, landings and discards. `harvest, stock` and `stock.n` slots are leave empty and `harvest.spwn` and `m.spwn` are set equal to 0.

`age2ageDat.` This function creates an age structured `FLStock` from age structured `FLBiol` and `FLFleetsExt` objects. The slots `catch`, `catch.n`, `catch.wt`, `discards`, `discards.n`, `discards.wt`, `landings`, `landings.n`, `landings.wt`, `m`, `mat`, `harvest.spwn` and `'m.spwn'` of the `FLStock` object are filled in the following way:

`m` : `m` slot in `FLBiol` object multiplied by `varia.mort` where `varia.mort` is an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = it]`. `varia.mort` is used to introduce multiplicative uncertainty in the observation of natural mortality.

`mat` : `fec` slot in `FLBiol` object multiplied by `varia.fec` where `varia.fec` is an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = it]`. `varia.fec` is used to introduce multiplicative uncertainty in the observation of fecundity.

`landings.n` : Landings at age are obtained from `fleets` object, summing them up along seasons, units, metiers and fleets. After summing up, two sources of uncertainty are introduced, one related to aging error and the second one related to misreporting. Aging error is specified through `error.ages` argument, an array with dimension `[age = na, age = na, year = ny, iter = it]`. For each year and iteration, each element `(i,j)` in the first 2 dimensions indicates the proportion of individuals of age `i` that are wrongly assigned to age `j`, thus the sum of the elements along the first dimension must be equal to 1. For each year and iteration, the real landings at age are multiplied matricially with the corresponding sub-matrix of `error.ages` object. Afterwards, the second source of uncertainty is introduced multiplying the obtained landings at age by `varia.ltot`, an `FLQuant` with dimension `[age = na, year = ny,unit = 1, season = 1, area = 1, iter = it]`. Once uncertainty is introduced in landings at age and weight at age, the total landings are computed an compared with the TAC. If landings are lower than $'\text{TAC} \cdot \texttt{TAC.ovrsht}'$ the observed landings at age are unchanged, but if they were higher, the landings at age would be reduced by $\frac{1}{\texttt{TAC.ovrsht}}$ where `TAC.ovrsht` is a positive real number.

`landings.wt` : Landings weight at age is derived from `fleets` object, averaging it along seasons, units, metiers and fleets. After averaging, 2 sources of uncertainty are introduced, one related to aging error and the second one related to misreporting. Aging error is the same as the one used in the landings at age. For each year and iteration, the real weight at age is weighted by the proportion of landings in each age group and multiplied matricially with the corresponding sub-matrix of `error.ages` object. Afterwards, the second source of uncertainty is introduced multiplying the obtained weight at age by `varia.dwgt` an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = it]`.

`discards.n` : Observed discards at age are obtained in the same way as the landings but summing up the discards instead of landings and using, in the second source of error, the object `varia.dtot`, an `FLQuant` with dimension `[age = na, year = ny,unit = 1, season = 1, area = 1, iter = it]`. The object `error.ages` is the same as the one used in the derivation of landings at age.

`discards.wt` : Observed weight at age is obtained in the same way as the landings but averaging along discards weight instead of landings weight and using, in the second source of error, the object `varia.dwgt`, an `FLQuant` with dimension `[age = na, year = ny,unit = 1, season = 1, area = 1, iter = it]`. The object `error.ages` is the same as the one used in the derivation of landings at age.

`discards, landings` : Observed discards and landings are derived from observed landings and discards at age and their corresponding weight.

`catch, catch.n, catch.wt` : These slots are derived from the observed landings and discards at age and their corresponding weight.

bio2bioDat. This function creates a `FLStock` aggregated in biomass from `FLBiol` and `FLFleetsExt` objects aggregated in biomass.

`m, mat, landings.n, landings.wt, discards.n, discards.wt` , `catch.n, catch.wt: NA`

`discards` : The discards are summed up along fleets and metiers and then uncertainty (observation error) is introduced using a multiplicative error. This multiplicative error is specified through `varia.tdisc` argument an `FLQuant` with dimension `[quant = 1, year = ny,unit = 1, season = 1, area = 1, iter = it]`.

`landings` : Observed landings are derived in the same way as discards but the argument used to introduce uncertainty is called `varia.tland` in this case. Once uncertainty is introduced in landings, they are compared with the TAC. If the landings are lower than $'\text{TAC} \cdot \texttt{TAC.ovrsht}'$ the observed landings are unchanged but if there were higher the landings would be reduced by $\frac{1}{\texttt{TAC.ovrsht}}$, where `TAC.ovrsht` is a positive real number.

`catch` : This slot is equal to the sum of landings and discards.

age2bioDat. This function creates a `FLStock` aggregated in biomass from age structured `FLBiol` and `FLFleetsExt` objects. The function works exactly in the same way as `bio2bioDat` function.

3.3.7 Observation Models: Population

This type of models are useful when no assessment model is used in the next step of the MPM and management advice is just based on the population 'observed' in this step. `age2agePop`, `bio2bioPop` and `age2bioPop` are equal to their relatives in the previous section but in this case stock numbers, stock biomass and harvest are observed, with or without error, depending on the arguments given.

NoObsStock. This function is used when the advice is given independently to stock status. Therefore, we do not need to observ the population.

perfectObs. This function creates a `FLStock` from `FLBiol` and `FLFleetsExt` objects. The `FLBiol` and `FLFleetsExt` objects can be either aggregated in biomass or age structured and the returned `FLStock` object will have the same structure, but with unit and season dimensions collapsed. This function does not introduce any observation uncertainty in the observation of the different quantities stored in the `FLStock` or `FLFLeetsExt` objects. Slots relative to biological parameters are calculated averaging across units and seasons, those relative to catch summing up across units and seasons and numbers at age or biomass are taken from the start of the first season, except recruitment that is obtained summing up the recruitment produced along seasons. Finally, fishing mortality is calculated numerically from numbers at age and natural mortality.

`age2agePop.` This function operates exactly in the same way as its counterpart in the previous section, `age2ageDat`, but it also fills `stock.n`, `stock.wt`, `stock` and `harvest` slots:

`stock.n` : First, the numbers at age are calculated as in `perfectObs` function and then 2 sources of uncertainty are introduced, as it is done in landings and discards at age. The error attributed to aging error is given by the same argument as in landings and discards at age, `error.ages`. The second uncertainty is introduced in the same way but by different argument, `varia.ntot`.

`stock.wt` : First, the weight at age is calculated as in `perfectObs` function and then 2 sources of uncertainty are introduced, as it is done in weight at age of landings but replacing landings by stock numbers at age. The error attributed to aging error is given by the same arguments as in landings, `error.ages`. The second uncertainty is introduced in the same way but by different argument, `varia.ntot`.

`stock` : This is equal to the sum of the product of `stock.n` and `stock.wt`.

`harvest` : Harvest is numerically calculated from stock numbers at age and natural mortality.

`bio2bioPop.` This function operates exactly in the same way as its counterpart in the previous section `bio2bioDat` but it also fills `stock` and `harvest` slots:

`stock` : Stock biomass is calculated multiplying `n` andt `wt` slots in the `FLBiol` object and summing up along seasons (note that unit dimension is always equal to 1 in populations aggregated in biomass). After, that uncertainty in the observation is introduced multiplying the obtained biomass by the argument `varia.btot`, which is an `FLQuant` with dimension `[quant = 1, year = ny, unit = 1, season = 1, area = 1, iter = it]`

`harvest` : Harvest is calculated as the ratio between catch and stock biomass.

`age2bioPop.` This function operates exactly in the same way as its counterpart in the previous section `age2bioDat`, but it also fills `stock` and `harvest` slots. These two slots are calculated as in `bio2bioPop` function but summing up along ages in the case of `stock` slot.

3.3.8 Observation Models: Abundance Indices

Currently, there are 2 functions that simulate abundance indices, one that generates age structured abundance indices `ageInd` and a second one that generates abundance indices in biomass `bioInd`. The last one can be applied to both age structured and biomass dynamic populations. In both cases a linear relationship between the index and the abundance is assumed being the catchability q the slope, i.e:

$$I = q \cdot N \quad \text{or} \quad I = q \cdot B$$

ageInd. Age structured abundance indices are obtained multiplying the slot `n` of `FLBiol` with the catchability of the index (`catch.q` in `FLIndex` object). The `FLIndex` is an input object and the `index` slot is yearly updated. Two sources of uncertainty are introduced, one related to aging error and a second one related to random variation. Aging error is the same as in the observation of landings at age and the argument is the same `error.ages`. Afterwards the second source of uncertainty is introduced multiplying the index by the slot `index.var` of the `FLIndex` object. The indices do not need to cover the full age or year ranges.

bioInd. Biomass abundance indices are generated in the same way as age structured indices but without the error associated to age.

NoObsInd. This function is used when abundance indices are not required.

3.3.9 Observation Model: Fleets

At this point there are no functions to observe the fleets, their catch or catch at age is just observed in an aggregated way in the functions defined in previous section. In the short term it is not planned to write such a function. This function would be useful to be able to test Fcube [Ulrich et al., 2011] like approaches in management advice module.

3.3.10 Management Advice Models

Different management advice models have been implemented. Some of them are methods generally applicable (e.g. `fixedAdvice`, `annualTAC`, `IcesHCR` and `annexIVHCR`), whereas others are designed specifically for specific case studies (e.g. `FroeseHCR`, `ghlHCR`, `aneHCRE` and `neaMAC_ltmp`)

fixedAdvice. This function is used when the advice is fixed and independent to the stock status. TAC or TAE values should be given as input in the `advice` object.

annualTAC. This function mimics the typical harvest control rule (HCR) used in recovery and management plans implemented in Europe. The function is a wrapper of the `fwd` function in `FLash` library. As `fwd` is only defined for age structured populations within `FLBEIA` a new function `fwdBD` has been coded. `fwdBD` is a tracing of `fwd` but adapted to work with populations aggregated in biomass. The advice is produced in terms of catch, i.e TAC. The call to `annualTAC` function within `BEIA` is done as:

```
annualTAC(stocks, advice, advice.ctrl, year, stknm, ...)
```

If the management is being running in year `y`, the function works as follows:

1. Project the observed stock one year forward from 1st of January of year y up to 1st of January of year `y+1` (intermediate year).

2. Apply the HCR and get the TAC for year `y+1`. Depending on the definition of the HCR the stock could be projected several years forward.

`advice.ctrl` for `annualTAC`

`HCR` : `'annualTAC'`.

`nyears` : Number of years to project the observed stock from year `y-1`.

`wts.nyears` : Number of historic years to be used in the average of biological parameters. The average is used in the projection of biological parameters.

`fbar.nyears` : Number of historic years to be used in the average of selection pattern. The average is used in the projection of selection pattern.

`f.rescale` : Logical. If `TRUE` rescale to status quo fishing mortality.

`disc.nyears` : Number of years over which to calculate mean for `discards.n` and `landings.n` slots.

`fwd.ctrl` : Element of class `fwdControl`. For details on this look at the help page in `FLash` object. The only difference is the way the years are introduced. As this object is defined before simulation and it is applied year by year, the definition of the year should be dynamic. Thus the following convention has been taken:

- `year = 0` indicates the year when management is taking place, (intermediate year).
- `year = -1` corresponds with one year before the year when management is taking place. In this case, whithin `annualTAC` function, coincides with the year up to which data is available, (data year). Then, -2 would indicate 2 years before,-3 would indicate 3 years before and so on.
- `year = 1` corresponds with one year after the year when management is taking place. In this case, whithin `annualTAC` function, coincides with the year for which management advice is going to be produced, (TAC year). Then, 2 would indicate 2 years after the year when management is taken place, 3 would indicate 3 years after and so on.

In this way, within the simulation, each year, the intermediate year is summed up to the `year` in the original control argument and the correct year names are obtained.

`advice` : `catch` or `landings`. Is the TAC given in terms of catch or landings?

`sr` : The stock recruitment relationship used to project the observed stock forward, not needed in the case of population aggregated in biomass. `sr` is a list with 3 elements, `model, params` and `years`. `model` is mandatory and the other 2 are complementary, if `params` is given `years` is not necessary. `model` can be any stock-recruitment model defined for `FLSR` class. `params` is a `FLPar` model an if specified it is used to parameterized the stock-recruitment model. `years` is a

numeric named vector with 2 elements `'y.rm'` and `'num.years'`, for example `c(y.rm = 2, num.years = 10)`. This element is used to determine the observeds years to be used to estimate the parameters of the stock recruitment relationship. In the example the last 2 observations will be removed and starting from the year before to the last 2 observed years 10 years will be used to estimate the stock-recruitment parameters.

`growth.years` : This argument is used only for stocks aggregated in biomass and it indicates the years to be used in the estimation of annual population growth. This growth is used to project the population forward.
`growth.years` is a numeric named vector with 2 elements `'y.rm'` and `'num.years'` which play the same role played in `sr[['years']]` argument defined in the previous point.

`IcesHCR.` +++++

`FroeseHCR.` +++++

`annexIVHCR.` +++++

`ghlHCR.` +++++

`aneHCRE.` ++++

`neaMAC_ltmp.` +++++

3.4 Fourth level functions

These functions are called by the third level functions and, for the time being, are the functions in the lowest level within `FLBEIA`.

3.4.1 Stock-Recruitment relationships

Stock-recruitment relationships are used, for example, within `ASPG` and `annualTAC` functions. The stock-recruitment relationship used in `ASPG` is defined in the slot `model` of `FLSRsim` and it defines the true recruitment dynamics of the stocks. Within `annualTAC`, the stock-recruitment relationship used is defined in:

```
advice.ctrl[['stknm']][['sr']][['model']]
```

element and it describes the 'observed' stock-recruitment dynamics (used) in the management process.

In `FLCore` package there are several stock-recruitment relationships already defined and all can be used within `FLBEIA`. Some of the functions available are:

`bevholt` : Beverton and Holt model with the following parameterization:

$$R = \frac{\alpha \cdot S}{(\beta + S)}$$

where α is the maximum recruitment (asymptotically) and β is the stock level needed to produce the half of maximum recruitment $\alpha/2$. $(\alpha, \beta > 0)$.

`bevholt.ar1, ricker.ar1, shepherd.ar1` : Beverton and Holt, Ricker and Shepherd stock-recruitment models with autoregressive normal log residuals of first order. In the model fit the corresponding stock-recruitment model is combined with an autoregressive normal log likelihood of first order for the residuals. If R_t is the observed recruitment and $\hat{R}_t$ is the predicted recruitment, an autoregressive model of first order is fitted to the log-residuals, $x_t = log(R_t/\hat{R}_t)$.

$$x_t = \rho \cdot x_{t-1} + \varepsilon$$

where $\varepsilon \sim N(0, \sigma^2_{ar})$.

`geomean` : Recruitment is independent of the stock and equal to the geometric mean of historical period.

$$R = \alpha = \sqrt[n]{R_1 \cdot \ldots \cdot R_n}$$

`ricker` : Ricker stock-recruitment model fit with the following parameterization:

$$R = \alpha \cdot S \cdot e^{-\beta \cdot S}$$

where α is related to productivity and β to density dependence. α is the recruit per stock unit at small stock levels. $(\alpha, \beta > 0)$.

`segreg` : Segmented regression stock-recruitment model fit:

$$R = \begin{cases} \alpha \cdot S & \text{if } S < \beta, \\ \alpha \cdot \beta & \text{if } S \geq \beta. \end{cases}$$

α is the slope of the recruitment for stock levels below β, and $\alpha \cdot \beta$ is the mean recruitment for stock levels above β. $(\alpha, \beta > 0)$.

`shepherd` : Shepherd stock-recruitment model fit:

$$R = \alpha \cdot \frac{S}{(1 + (S/\beta)^{\gamma})}$$

This model generalizes Beverton and Holt and Ricker models, ($\gamma = 1$ corresponds with Beverton and Holt model, $\gamma > 1$ takes a ricker-like shape and with $\gamma < 1$ the curve rises indefinitely).

There could be more stock-recruitment relationships defined in `FLCore`, thus, if you are interested in using a model not defined here take a look at `SRModels` help page in `FLCore` package. New stock-recruitment models to be used in `FLSRsim` class can be defined in two ways:

1. Using a formula in slot `model`:

$$rec \sim \Phi(X)$$

   where Φ is a function of `ssb` and parameters and covariables stored in `params` and `covar` slots respectively.

2. Defining a function in `R`, `foo <- function(X)`, and using the name of the function, `foo`, in slot `model`. The function arguments must be among `ssb` and parameters and covariables stored in `params` and `covar` slots respectively.

+++++ Tenemos definidas varias relaciones SR para nuestros casos de estudio (e.g. `hockstick`, `redfishRecModel`, `aneRec_pil`, `pilRec_ane`, `ctRec_alb`). Las metemos también en el manual??? +++++

3.4.2 Catch production functions

The catch production functions can be different for the same third level effort model. Currently, there are two catch production functions available, both correspond with Cobb-Douglas production functions [Clark, 1990, Cobb and Douglas, 1928] but in one case the model operates at stock level and in the second one at age class level.

`CobbDouglasBio`: Cobb-Douglas production function at stock level. The total catch of the fleet is calculated according to the Cobb-Douglas production function:

$$C = q \cdot E^{\alpha} \cdot B^{\beta} \tag{12}$$

where C denotes total catch and B total biomass, both in weight, q the catchability and E the effort. α and β are the elasticity parameters associated to labor and capital (biomass in this case), respectively. These parameters are associated to the existing technology.

As α and β parameters depend on the stock and the technology, Cobb-Douglas function is applied at metier level. Thus, the catch of a certain fleet f is given by:

$$C_f = \sum_{m \in M_f} q_{f,m} \cdot B^{\beta_{f,m}} \cdot (E_f \cdot \delta_{f,m})^{\alpha_{f,m}} \tag{13}$$

where M_f represents the set of metiers of fleet f and δ the effort share among metiers.

Derivation of Catch-at-age. Once the total catch is calculated, it is divided into catch at age using selectivity at age, $s_{a,f,m}$, and biomass at age in the population, B_a:

$$C_{a,f,m} = \frac{C_{f,m}}{\sum_a s_{a,f,m} \cdot B_a} \cdot s_{a,f,m} \cdot B_a \tag{14}$$

Derivation of equation 14:

- If the whole population were accessible to the gear, the catch of age a would be:

$$s_{a,f,m} \cdot B_a$$

- Thus, if the whole population were accessible to the gear, the total catch we could obtain would be:

$$\sum_a s_{a,f,m} \cdot B_a$$

- But, the actual total catch is C_f, so theoretically the proportion of the population that have been accessible is [1]:

$$C_{a,f,m} = \frac{C_{f,m}}{\sum_a s_{a,f,m} \cdot B_a}$$

- Then, if we assume the population is homogeneously distributed we arrive to equation 14.

The catch at age is then further disaggregated in landings- and discards-at-age using landings' and discards' specific selectivity:

$$L_{a,f,m} = \frac{sl_{a,f,m}}{s_{a,f,m}} \cdot C_{a,f,m} \quad \text{and} \quad D_{a,f,m} = \frac{sd_{a,f,m}}{s_{a,f,m}} \cdot C_{a,f,m} \tag{15}$$

`CobbDouglasAge`: Cobb-Douglass production function at age-class level. The catch of the fleets is calculated according to the Cobb-Douglas production function applied at age-class level, i.e.:

$$C = \sum_a C_a = q_a \cdot E^{\alpha_a} \cdot B_a^{\beta_a} \tag{16}$$

where C denotes catch and B biomass, both in weight, q the catchability, E the effort and a the subscript for age. α and β are the elasticity parameters associated to labor and capital (biomass in this case) respectively. These parameters are associated to the existing technology.

As α and β parameters dependent on age classes and technology, Cobb-Douglas function is applied at metier level. Thus, the catch of a certain fleet f is given by:

$$C_f = \sum_a C_{a,f} = \sum_{m \in M_f} \sum_a q_{a,f,m} \cdot B_a^{\beta_{a,f,m}} \cdot (E_f \cdot \delta_{f,m})^{\alpha_{a,f,m}} \tag{17}$$

where M_f represents the set of metiers of fleet f, δ the effort share among metiers and m is the subscript that indicates the metier.

[1] If all the age classes were not accessible or completely accessible we would replace $s_{a,f,m}$ by $s_{a,\acute{f},m} = \gamma_{a,f,m} \cdot s_{a,f,m}$ where $\gamma_{a,f,m}$ is the proportion of individuals of age a accessible to metier m in fleet f.

3.4.3 Costs functions

Cost functions have been developed in order to be used within `fleets.om`. As cost structure could differ among fleets it has been defined as fourth level function and it works at fleet level. In principle, it could be useful in both tactic and strategic dynamics of fleets.

`TotalCostsPower.` This function sums up the fixed costs (FxC) and the power functions of cost per unit of effort ($CostPUE$), crew share per unit of landings ($CSPUL$) and capital cost per unit capital ($CapCostPUC$), mathematically:

$$\begin{aligned} Cost_f =& FxC_f + CostPUE_{f,m} \cdot \sum_m (E_f \cdot \tau_{f,m})^{\gamma_{1f,m}} + \\ & \sum_{st} \sum_m CSPUL_{f,m,st} \cdot L_{f,m,st}^{\gamma_{2f,m,st}} + CapCostPUC_f \cdot Cap_f^{\gamma_{3f}} \end{aligned} \tag{18}$$

The fixed cost are given at fleet level, f, cost per unit of effort at metier level, m, and crew share at fleet, metier and stock, st, level. $\gamma_{1f,m}$ is the exponent of effort at fleet and metier level in cost of effort addend, $\gamma_{2f,m,st}$ the exponent of landing at fleet, metier and stock level in crew share cost addend and γ_{3f} is the exponent of capital at fleet level in capital cost addend.

References

D. S. Butterworth and A. E. Punt. Experiences in the evaluation and implementation of management procedures. *ICES Journal of Marine Science*, 56(6):985–998, 1999. TY - JOUR.

Doug S. Butterworth. Why a management procedure approach? some positives and negatives 10.1093/icesjms/fsm003. *ICES J. Mar. Sci.*, 64 (4):613–617, 2007.

Colin W. Clark. *Mathematical Bioeconomics: The Optimal Management of Renewable Resources*. John Wiley & Sons, 1990.

C.W. Cobb and P.H. Douglas. A theory of production. *American Economic Reviews*, 18:139–165, 1928.

William K. De la Mare. Tidier fisheries management requires a new mop (management-oriented paradigm). *Reviews in Fish Biology and Fisheries*, 8:349–356, 1998.

Ayoe Hoff, Hans Frost, Clara Ulrich, Dimitrios Damalas, Christos D. Maravelias, Leyre Goti, and Marina SanturtÃžn. Economic effort management in multispecies fisheries: the fcubecon model 10.1093/icesjms/fsq076. *ICES Journal of Marine Science: Journal du Conseil*, 2010.

L. T. Kell, I. Mosqueira, P. Grosjean, J-M. Fromentin, D. Garcia, R. Hillary, E. Jardim, S. Mardle, M. A. Pastoors, J. J. Poos, F. Scott, and R. D. Scott. Flr: an open-source framework for the evaluation and development of management strategies 10.1093/icesjms/fsm012. *ICES J. Mar. Sci.*, 64(4):640–646, 2007.

S.B.M. Kraak, F.C. Buisman, M Dickey-Collas, Poos J.J., M.A. Pastoors, J.G.P. Smit, and N. Daan. How can we manage mixed fisheries? a simulation study of the effect of management choices on the sustainability and economic performance of a mixed fishery. . Technical report, 2004.

J.J Pella and P.K. Tomlinson. A generalized stock-production model. *Bulletin of the Inter-American Tropical Tuna Comission*, 13:421–458, 1969.

Andre E. Punt and Greg P. Donovan. Developing management procedures that are robust to uncertainty: lessons from the international whaling commission 10.1093/icesjms/fsm035. *ICES J. Mar. Sci.*, 64(4):603–612, 2007.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Rebecca A. Rademeyer, Eva E. Plaganyi, and Doug S. Butterworth. Tips and tricks in designing management procedures 10.1093/icesjms/fsm050. *ICES J. Mar. Sci.*, 64(4):618–625, 2007.

Pavel Salz, Erik Buisman, Katrine Soma, Hans Frost, Paolo Accacia, and Raúl Prellezo. Fishrent: Bio-economic simulation and optimization model for fisheries. Technical report, 2011.

Clara Ulrich, Stuart A. Reeves, S. Holmes, and W. Vanhee. Reconciling single-species tacs in the north sea demersal fisheries using the fcube mixed-fisheries advice framework. *ICES J. Mar. Sci.*, In press, 2011.

A New `FLR` - `S4` classes

A.1 `FLBDsim` class

`FLBDsim` class has been created in order to facilitate the simulation of population growth in populations aggregated in biomass, i.e. $g(.)$ in equation 4. The population dynamics are simulated as follows:

$$B_{y,s} = B_{y0,s0} + g(B_{y0,s0}) \cdot \varepsilon_{y,s} - C_{y0,s0} \tag{19}$$

where B is the biomass, C the catch, $y0$ and $s0$ are the subscripts of previous season's year and season and ε is the uncertainty value in year y and season s. It is a `S4` class and has 10 slots:

`name, desc, range` : Slots common to all `FLR` objects.

`model` : `character` or `formula`. If character, it must coincide with an already existing growth model. If formula, the parameters must be slots in the object or elements of `covar` slot. Currently, there is only one growth model available, `'PellaTom'` that corresponds with Pella-Tomlinson growth model (`?`).

`biomass` : `FLQuant` to store biomass in weight. The dimension in `quant, unit` and `area` must be equal to 1 and in the rest of the dimensions it must be congruent with general simulation settings.

`catch` : `FLQuant` to store total catch in weight. The dimension in `quant, unit` and `area` must be equal to 1 and in the rest of the dimensions it must be congruent with general simulation settings.

`uncertainty` : `FLQuant` to store the error that is multiplied to the point estimate of growth . The dimension in `quant, unit` and `area` must be equal to 1 and in the rest of the dimensions it must be congruent with general simulation settings. Thus, a different error can be used for each year, season and iteration.

`params` : An `array` to store the parameters of the model. The dimensions of the array are `params, year, season, iter`. The dimension in `year, season` and `iter` must be congruent with general simulation settings. Thus, a different set of parameters can be used for each year, season and iteration.

`covar` : An `FLQuants` object. The elements of the list are used to store covariables' values and it is used to apply growth models with covariables. Its functionality is the same as in `FLSR` object.

A.2 `FLSRSim` class

`FLSRsim` class has been created in order to facilitate the simulation of recruitment in age structured populations. The recruitment dynamics are simulated as follows:

$$R_{y,s} = \Phi(S_{y-tl_0,s-tl_1}, covars_{y-tl_0,s-tl_1}) \cdot \varepsilon_{y,s} \cdot \rho_{y,s} \tag{20}$$

where $R_{y,s}$ is the recruitment in year y and season s, Φ is the stock-recruitment model, tl_0 and tl_1 are the year and season lag between spawning and recruitment, respectively, S_{y-tl_0,tl_1} and $covars_{y-tl_0,s-tl_1}$ are the stock index and covariables in year $y - tl_0$ and season tl_1, $\varepsilon_{y,s}$ is the uncertainty value in year y and season s and $\rho_{y,s}$ is the proportion of recruitment that recruits in year y and season s and is produced by stock index S in year $y - tl_0$ and season tl_1 .

`rec:` An `FLQuant` with dimension $[1, ny, 1, ns, 1, it]$ used to store recruitment.

`ssb:` An `FLQuant` with dimension $[1, ny, 1, ns, 1, it]$ used to store SSB or the stock index used in the stock-recruitment relationship.

`covar:` An `FLQuants` to store the covariables used in the stock-recruitment relationship. For details on the use of this slot look at the description of `FLSR` class.

`uncertainty:` An `FLQuant` with dimension $[1, ny, 1, ns, 1, it]$ used to store the uncertainty related to stock-recruitment process. The content of this slot is multiplied to the point estimate of recruitment. As its effect is multiplicative, then set it equal to 1 for all year, season and iteration if uncertainty is not going to be considered around stock-recruitment curve.

`proportion:` An `FLQuant` with dimension $[1, ny, 1, ns, 1, it]$ used to store the proportion of the recruitment produced by stock index in year $y - \texttt{timelag}[1, s]$ and season $\texttt{timelag}[2, s]$ that recruits in year y and season s. The content of this slot is multiplied to the point estimate of recruitment. As its effect is multiplicative, then set it equal to 1 if all the recruitment produced by certain stock index is recruited at the same time and set it equal to 0 if none of the recruitment produced by certain stock index is recruited in that season.

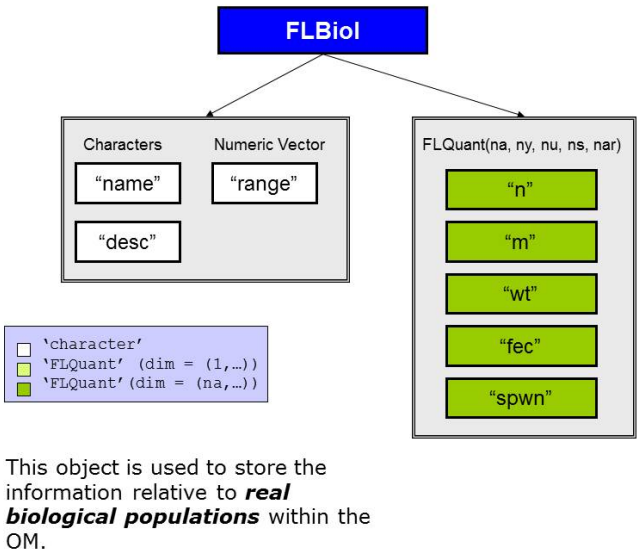
`model:` Character string or formula. If character it specifies the name of the function used to simulate the recruitment. If formula the left hand side of $\sim$ must be equal to `rec` and the elements in right hand side must be among `ssb, covars` and `params`.

`params:` An array with dimension `[nparams, ny, ns, it]`, thus, the parameters may be year, season and iteration dependent. Year dimension in parameters may be useful to model regime shifts.

`timelag:` A matrix with dimension `[2,ns]`. This object indicates the time lag between spawning and recruitment in each season. For each season, the element in the first row indicates the age at recruitment and the element in the second row indicates the season at which the recruitment was spawn.

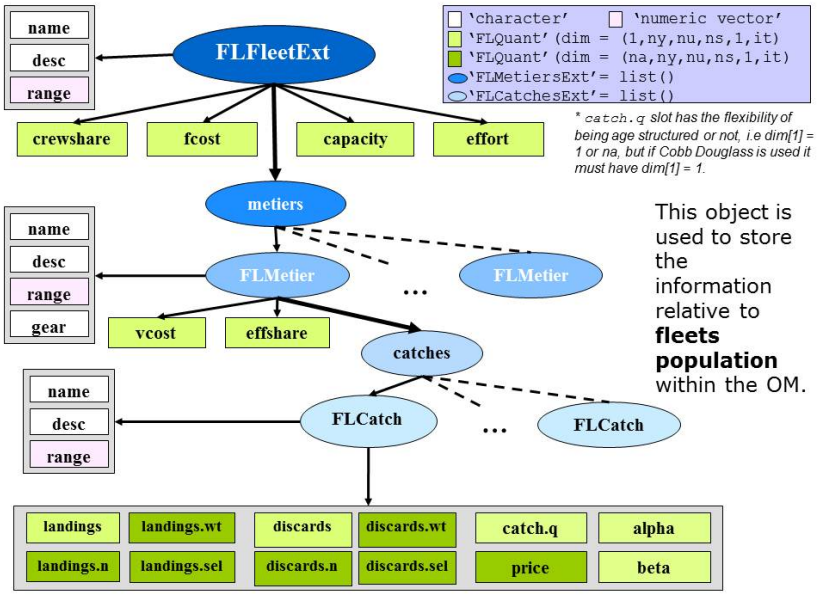
B Graphical representation of FLR Objects

Figure 3: FLBiol object



This object is used to store the information relative to ***real biological populations*** within the OM.

Figure 4: FLFleetExt object



This object is used to store the information relative to **fleets population** within the OM.

Figure 5: FLSRsim object

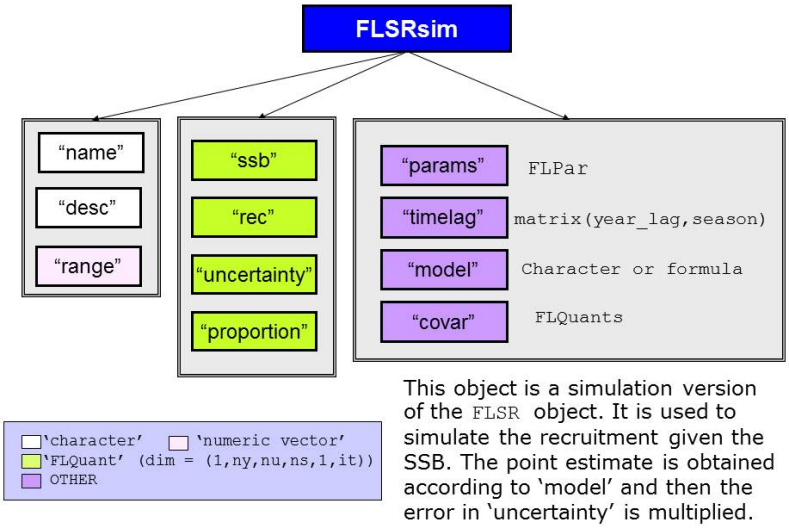


Figure 6: FLBDsim object

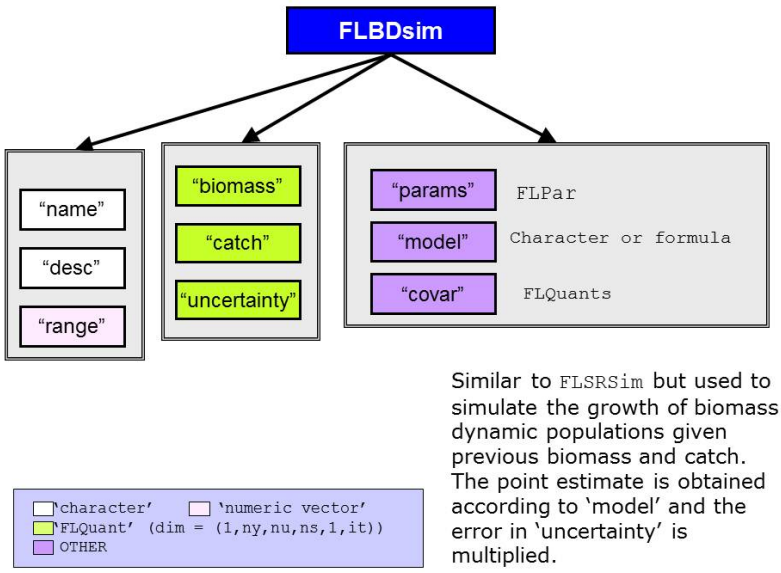


Figure 7: FLIndex object

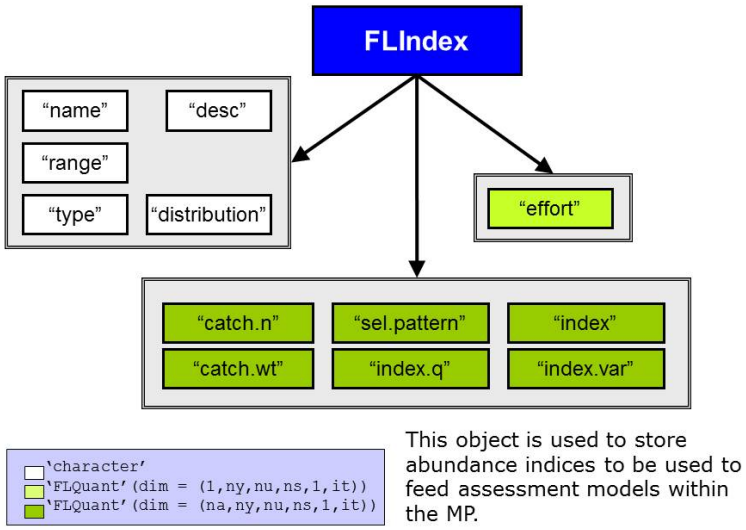
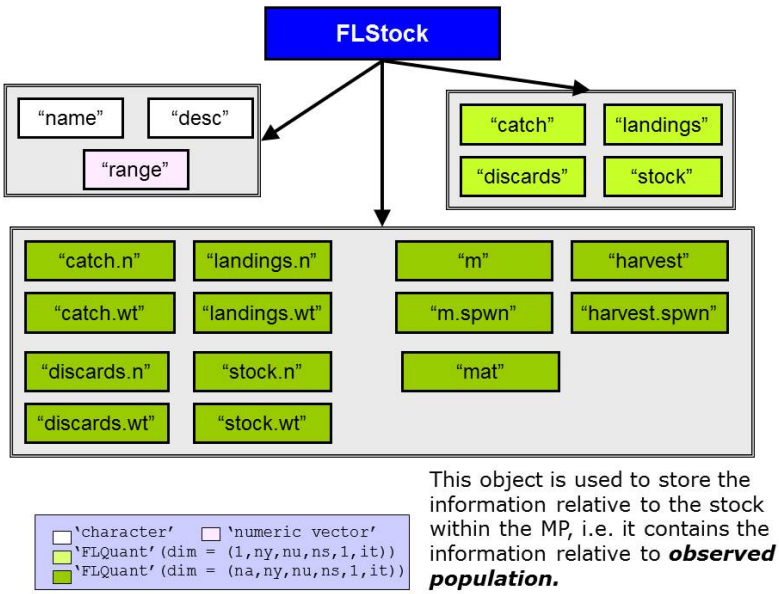



Figure 8: FLStock object