a4a stock assessment framework
DRAFT

Ernesto Jardim[1], Colin Millar[1], and Finlay Scott[1]

[1]European Commission, Joint Research Centre, IPSC / Maritime Affairs Unit, 21027 Ispra (VA), Italy
*Corresponding author ernesto.jardim@jrc.ec.europa.eu

August 18, 2014

Contents

1 Running assessments

In the a4a assessment model, the model structure is defined by submodels, which are the different parts of a statistical catch at age model that require structural assumptions.

There are 5 submodels in operation: a model for F-at-age, a model for the initial age structure, a model for recruitment, a (list) of model(s) for abundance indices catchability-at-age, and a list of models for the observation variance of catch-at-age and abundance indices. In practice, we fix the variance models and the initial age structure models, but in theory these can be changed.

The submodels form use linear models. This opens the possibility of using the linear modelling tools available in R: see for example the mgcv gam formulas, or factorial design formulas using `lm()`. In R's linear modelling language, a constant model is coded as ~ 1, while a slope over age would simply be $\sim age$. For example, we can write a traditional year/age separable F model like $\sim factor(age) + factor(year)$.

The 'language' of linear models has been developing within the statistical community for many years, and constitutes an elegant way of defining models without going through the complexity of mathematical representations. This approach makes it also easier to communicate among scientists

- 1965 J. A. Nelder, notation for randomized block design
- 1973 Wilkinson and Rodgers, symbolic description for factorial designs
- 1990 Hastie and Tibshirani, introduced notation for smoothers
- 1991 Chambers and Hastie, further developed for use in S

There are two basic types of assessments available in a4a: the management procedure fit and the full assessment fit. The management procedure fit does not compute estimates of covariances and is therefore quicker to execute, while the full assessment fit returns parameter estimates and their covariances at the expense of longer fitting time.

1.1 Stock assessment model details

The statistical catch at age model is based on the well known Baranov catch equation

$$e^{\log C} = \frac{\mathbf{F}}{\mathbf{F}+M}\left(1 - e^{-\mathbf{F}-M}\right)\mathbf{R}e^{-\sum \mathbf{F}+M}$$

and the survival equation

$$e^{\log I} = \mathbf{QR}e^{-\sum \mathbf{F}+M}$$

where

$$\log C_{ay} = \sigma^{\mathbf{2}}_{\mathbf{ay}} \qquad \log I_{ays} = \tau^{\mathbf{2}}_{\mathbf{ays}}$$

In these equations M is natural mortality, F fishing mortality, R recruitment, Q survey catchability and C catch. All these variables are defined by age groups, although in the formula the indices were removed for better readability.

The quantities $\log F$, $\log Q$, $\log R$, $\log observation\, variances$ and $\log initial\, age\, structure$ (in bold in the equations above), need to be given a form, which is done using linear models. Recruitment is a special case. It is modelled as a fixed variance random effect, using the hard coded models Ricker, Beverton Holt, smooth hockeystick or geometric mean, which can use linear models for their parameters $\log a$ or $\log b$, where relevant. As an alternative the $\log R$ submodel can use a linear model like the other submodels

1.2 Quick and dirty

Here we show a simple example of using the assessment model using plaice in the North Sea. The default settings of the stock assessment model work reasonably well. It's an area of research that will improve with time. Note that because the survey index for plaice has missing values we get a warning saying that we assume these values are missing at random.

```
data(ple4)

## Warning:  data set 'ple4' not found

data(ple4.indices)

## Warning:  data set 'ple4.indices' not found

fit <- sca(ple4, ple4.indices)

## Error:  could not find function "sca"
```

To inspect the stock assessment summary constituted of trends of fishing mortality (harvest), spawning stock biomass (SSB), catch and recruits, the user may add the `a4aFit` object to the original `FLStock` object using the + method and plot the result (Figure **??**).

```
stk <- ple4 + fit

## Error:  object 'ple4' not found

plot(stk)

## Error:  object 'stk' not found
```

In more detail, one can plot a 3D representation of fishing mortality (Figure **??**),

```
persp3D(z=harvest(fit)[drop=TRUE], x=as.numeric(dimnames(harvest(fit))[[1]]), y=as.numeric(dimnames(har

## Error:  could not find function "persp3D"
```

population abundance (Figure **??**) is displaid as a 3D ribbon,

```
ribbon3D( z = stock.n(fit)[drop=TRUE], x= as.numeric(dimnames(stock.n(fit))[[1]]), y= as.numeric(dimnam

## Error:  could not find function "ribbon3D"
```

as well as catch-at-age (Figure **??**).

```
ribbon3D( z = catch.n(fit)[drop=TRUE], x= as.numeric(dimnames(catch.n(fit))[[1]]), y= as.numeric(dimnam

## Error:  could not find function "ribbon3D"
```

1.3 Diagnostics

A set of plots to inspect the fit quality and assumptions are implemented. The most common is to look at standardized log-residuals to check for biased results or large variances. Note that the standardization should produce residuals with variance 1, which means that most residual values should be between ~ -2 and ~ 2. These residuals also allow the user to check for deviances from the log-normal assumption.

The `residuals()` method will compute standardized residuals which can be plotted using a set of packed methods.

```
res <- residuals(fit, ple4, ple4.indices)

## Error:  object 'fit' not found
```

Figure **??** shows a scatterplot of residuals by age and survey, with a smoother to guide (or mis-guide ...) your visual analysis.

```
plot(res)

## Error:  object 'res' not found
```

The common bubble plot by year and age for each survey are shown in Figure **??**. It shows the same information as Figure **??** but in a multivariate perspective.

```
bubbles(res)

## Error:  could not find function "bubbles"
```

Figure **??** shows a quantile-quantile plot to assess how well do the residuals match the normal distribution.

```
qqmath(res)

## Error:  could not find function "qqmath"
```

To have a look at how well is the model predicting catches and abundance, one can use the `plot()` method with the *a4aFit* object and the *FLStock* (Figure **??**) object or the *FLIndex* object (Figure **??**).

```
plot(fit, ple4)

## Error:  object 'fit' not found
```

```
plot(fit, ple4.indices)

## Error:  object 'fit' not found
```

To get information about the likelihood fit the method `fitSumm()` will extract information about likelihood, number of parameters, etc, and the methods `AIC()` and `BIC()` will compute the information criteria.

```
fitSumm(fit)

## Error:  could not find function "fitSumm"

AIC(fit)

## Error:  object 'fit' not found

BIC(fit)

## Error:  object 'fit' not found
```

1.4 Data structures

As mentioned above, the output of the stock assessment method may be simpler, with or without all the information about the parameters of the model like the variance-covariance matrix. In the first case the class of the output object is *a4aFitSA* while in the second case is *a4aFit*.

This section will describe the data structures of these classes and the classes that compose them.

Starting with the basic model output class, *a4aFit*, the slots of this class are shown on the code below and Figure ??.

```
showClass("a4aFit")

## Error:  "a4aFit" is not a defined class
```

```
## Error:  could not find function "plotS4"
```

Fitted values are stored in the `stock.n`, `harvest`, `catch.n` and `index` slots. It also contains information carried over from the stock object used to fit the model, like the the name of the stock in `name`, any description provided in `desc` and the age and year range and mean F range in `range`. There is also a wall clock that has a breakdown of the time taken to run the model in `clock`.

The full assessment fit returns an object of class *a4aFitSA*, the slots of this class are shown on the code below and Figure ??.

```
showClass("a4aFitSA")

## Error:  "a4aFitSA" is not a defined class
```

```
## Error:  could not find function "plotS4"
```

The additional slots in the assessment output are the `fitSumm` and `pars` slots, which are containers for model summaries and the model parameters. The `pars` slot is a class of type *SCAPars* (Figure ??) which is itself composed of sub-classes, designed to contain the information necessary to simulate from the model.

```
showClass("SCAPars")

## Error:  "SCAPars" is not a defined class

showClass("a4aStkParams")

## Error:  "a4aStkParams" is not a defined class

showClass("submodel")

## Error:  "submodel" is not a defined class
```

```
## Error:  could not find function "plotS4"
```

The *SCAPars* is built using objects of class *a4aStkParams* (Figure ??) and *submodel* (Figure ??). These classes have the following slots.

```
## Error:  could not find function "plotS4"
```

```
## Error:  could not find function "plotS4"
```

For example, all the parameters required so simulate a time-series of mean F trends is contained in the `stkmodel` slot, which is a class of type *a4aStkParams*. This class contains the relevant submodels (see later), their parameters `params` and the joint covariance matrix `vcov` for all stock related parameters.

1.5 The statistical catch-at-age stock assessment framework - the `sca` method

The statistical catch at age (`sca()`) method used in the previous section with the default settings, can be parametrized to control other features of the stock assessment framework. The most interesting ones are the submodels for fishing mortality (F), catchability (Q) and recruitment (R).

An important argument for `sca()` is the type of fit, which controls if a full assessment will be performed or a management procedure type of assessment. The argument is called `fit` and can have the values 'assessment' for a full assessemt or 'MP' for a simpler assessment. By default `sca()` uses `fit='MP'`.

We'll start by looking at the submodel for F, then Q and finally R.

Please note that each of these model *forms* have not been tuned to the data. The degrees of freedom of each model can be better tuned to the data by using model selection procedures such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC), etc.

1.5.1 Fishing mortality submodel

We will now take a look at some examples for F models and the forms that we can get. We'll fix the Q and R submodels.

Lets start with a separable model in which age and year effects are modelled as dummy variables, using the `factor` coding provided by R (Figure **??**).

```
qmod <- list(~ factor(age))
fmod <- ~ factor(age) + factor(year)
srmod <- ~ factor(year)
fit <- sca(stock = ple4, indices = ple4.indices[1], fmodel=fmod, qmodel=qmod, srmodel=srmod)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

Next we may make things a bit more interesting by using an (unpenalised) thin plate spline, where we'll borrow the smoothing splines method (`s()`) provided by package mgcv. We're using the North Sea Plaice data again, and since it has 10 ages we will use a simple rule of thumb that the spline should have fewer than $\frac{10}{2} = 5$ degrees of freedom, and so we opt for 4 degrees of freedom. We will also do the same for year and model the change in F through time as a smoother with 20 degrees of freedom. Note that this is still a separable model, it's a smoothed version of the previous model (Figure **??**).

```
fmod <- ~ s(age, k=4) + s(year, k = 20)
fit1 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod) # notice that you can specify the submodels witho

## Error:  could not find function "sca"
## Error:  could not find function "wireframe"
```

A non-separable model, where we consider age and year to interact can be modeled using a smooth interaction term in the F model using a tensor product of cubic splines with the `te` method (Figure ??), again borrowed from mgcv.

```
fmod <- ~ te(age, year, k = c(4,20))
fit3 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

In the last examples the fishing mortalities (Fs') are linked across age and time. What if we want to free up a specific age class because in the residuals we see a consistent pattern. This can happen, for example, if the spatial distribution of juveniles is disconnected to the distribution of adults. The fishery focuses on the adult fish, and therefore the the F on young fish is a function of the distribution of the juveniles and could deserve a specific model. This can be achieved by adding a component for the year effect on age 1 (Figure ??).

```
fmod <- ~ te(age, year, k = c(4,20)) + s(year, k = 5, by = as.numeric(age==1))
fit4 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

1.5.2 Catchability submodel

The catchability submodel is set up the same way as the F submodel and the tools available are the same. The only difference is that the submodel is set up as a list of formulas, where each formula relates with one abundance index.

We'll start by fixing the F and R models and compute the fraction of the year the index relates to, which will allow us to compute catchability at age and year.

```
fmod <- ~ factor(age) + factor(year)
srmod <- ~ factor(year)
```

A first model is simply a dummy effect on age, which means that a coefficient will be estimated for each age. Note that this kind of model considers that levels of the factor are independent (Figure ??).

```
qmod <- list(~ factor(age))
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"
```

To compute the catchability estimated for each index we'll need to compute the abundance at the moment the index was carried out and divide the predicted index by the abundance. More precisely we'll compute abundance in the mid of the index period, which is stored in the `FLIndex` object, in the slot `range`, in fractions of the year. Later we'll see that we can use the method `predict()` to get the same result, but we'll need a *a4aFitSA* object to get the fitted parameters.

```
# compute N for the fraction of the year the survey is carried out
sfrac <- mean(range(ple4.indices[[1]])[c("startf", "endf")])

## Error:  object 'ple4.indices' not found

# fraction of total mortality up to that moment
Z <- (m(ple4) + harvest(fit))*sfrac

## Error:  could not find function "m"

lst <- dimnames(fit@index[[1]])

## Error:  object 'fit' not found

# survivors
lst$x <- stock.n(fit)*exp(-Z)

## Error:  could not find function "stock.n"

stkn <- do.call("trim", lst)

## Error:  object 'lst' not found

qhat <- index(fit)[[1]]/stkn

## Error:  could not find function "index"
```

```
## Error:  could not find function "wireframe"
```

If one considers catchability at a specific age to be dependent on catchability on the other ages, similar to a selectivity modelling approach, one option is to use a smoother at age, and let the data 'speak' regarding the shape (Figure **??**).

```
qmod <- list(~ s(age, k=4))
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

# compute N for the fraction of the year the survey is carried out
Z <- (m(ple4) + harvest(fit))*sfrac

## Error:  could not find function "m"

lst <- dimnames(fit@index[[1]])

## Error:  object 'fit' not found

lst$x <- stock.n(fit)*exp(-Z)

## Error:  could not find function "stock.n"

stkn <- do.call("trim", lst)

## Error:  object 'lst' not found

qhat <- index(fit)[[1]]/stkn

## Error:  could not find function "index"
## Error:  could not find function "wireframe"
```

As in the case of F, one may consider catchability to be a process that evolves with age and year, including an interaction between the two effects. Such model can be modelled using the tensor product of cubic splines, the same way we did for the F model (Figure ??).

```
qmod <- list(~ te(age, year, k = c(3,40)))
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

# compute N for the fraction of the year the survey is carried out
Z <- (m(ple4) + harvest(fit))*sfrac

## Error:  could not find function "m"

lst <- dimnames(fit@index[[1]])

## Error:  object 'fit' not found

lst$x <- stock.n(fit)*exp(-Z)

## Error:  could not find function "stock.n"

stkn <- do.call("trim", lst)

## Error:  object 'lst' not found

qhat <- index(fit)[[1]]/stkn

## Error:  could not find function "index"
```

```
## Error:  could not find function "wireframe"
```

Finally, one may want to investigate a trend in catchability with time, very common in indices built from CPUE data. In the example given here we'll use a linear trend in time, set up by a simple linear model (Figure ??).

```
qmod <- list( ~ s(age, k=4) + year)
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

# compute N for the fraction of the year the survey is carried out
Z <- (m(ple4) + harvest(fit))*sfrac

## Error:  could not find function "m"

lst <- dimnames(fit@index[[1]])

## Error:  object 'fit' not found

lst$x <- stock.n(fit)*exp(-Z)
```

```
## Error:  could not find function "stock.n"

stkn <- do.call("trim", lst)

## Error:  object 'lst' not found

qhat <- index(fit)[[1]]/stkn

## Error:  could not find function "index"
```

```
## Error:  could not find function "wireframe"
```

1.5.3 Catchability submodel for age aggregated indices

The previous section was focused on age disaggregated indices, but age aggregated indices (CPUE, biomass, DEPM, etc) may also be used to tune the total biomass of the population. In these cases the `index` slot of the classFLIndex is a vector and the age dimension must have one single element called "all". Note that in this case the qmodel should be set without age factors, although it can have a "year" component and covariates if needed.

```
# creating an index (note the name of the first dimension element)
dnms <- list(age="all", year=range(ple4)["minyear"]:range(ple4)["maxyear"])

## Error:  object 'ple4' not found

bioidx <- FLIndex(FLQuant(NA, dimnames=dnms))

## Error:  could not find function "FLIndex"

index(bioidx) <- stock(ple4)*0.001

## Error:  could not find function "stock"

index(bioidx) <- index(bioidx)*exp(rnorm(index(bioidx), sd=0.1))

## Error:  could not find function "index"

range(bioidx)[c("startf","endf")] <- c(0,0)

## Error:  object 'bioidx' not found

# note the name of the first dimension element
index(bioidx)

## Error:  could not find function "index"

# fitting the model
fit <- sca(ple4, FLIndices(bioidx), qmodel=list(~1))

## Error:  could not find function "sca"
```

The same methods that are applied to age disaggregated indices apply here, see standardized log residuals in Figure **??**. It's also possible to mix several indices of both types.

```
plot(residuals(fit, ple4, FLIndices(bioidx)))

## Error:  object 'fit' not found
```

1.5.4 Catchability submodel for single age indices

Similar to age aggregated indices one may have an index that relates only to one age, like a recruitment index. In this case the *FLIndex* object must have in the first dimension the age it referes to. The fit is then done relating the index with the proper age in numbers. Note that in this case the qmodel should be set without age factors, although it can have a "year" component and covariates if needed.

```
fit <- sca(ple4, FLIndices(ple4.index[1]), qmodel=list(~1))

## Error:  could not find function "sca"
```

As previously, the same methods apply, see standardized log residuals in Figure **??**.

```
plot(residuals(fit, ple4, FLIndices(ple4.index[1])))

## Error:  object 'fit' not found
```

1.5.5 Stock-recruitment submodel

The S/R submodel is a special case, in the sense that it can be set up with the same linear tools as the F and Q models, but it can also use some hard coded models. The example shows how to set up a simple dummy model with `factor()`, a smooth model with `s()`, a Ricker model (`ricker()`), a Beverton and Holt model (`bevholt()`), a hockey stick model (`hockey()`), and a geometric mean model (`geomean()`). See Figure **??** for results. As mentioned before, the 'structural' models have a fixed variance, which must be set by defining the coefficient of variation. We now fix the F and Q submodels before fiddling around with the S/R model.

```
fmod <- ~ s(age, k=4) + s(year, k = 20)
qmod <- list(~ s(age, k=4))
```

```
srmod <- ~ factor(year)
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

srmod <- ~ s(year, k=20)
fit1 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

srmod <- ~ ricker(CV=0.1)
fit2 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

srmod <- ~ bevholt(CV=0.1)
fit3 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
```

```
## Error:  could not find function "sca"

srmod <- ~ hockey(CV=0.2)
fit4 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

srmod <- ~ geomean(CV=0.1)
fit5 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

## Error:  could not find function "sca"

flqs <- FLQuants(factor=stock.n(fit)[1], smother=stock.n(fit1)[1], ricker=stock.n(fit2)[1], bevertonHol

## Error:  could not find function "FLQuants"
```

```
xyplot(data~year, groups=qname, data=flqs, type="l", auto.key=list(points=FALSE, lines=TRUE, columns=3)

## Error:  could not find function "xyplot"
```

1.6 The major effects - age, year and cohort

All submodels use the same type of specification process, the R formula interface, wich gives lot's of flexibility to explore models and combination of sub-models. As a reference one can consider three major effects that can be modelled the same way, the age affect, year effect and cohort effect. As examples note the following models, in these cases applied to fishing mortality, and all of them as a factor, which means one coefficient will be estimated for each level of the factor, meaning age, year or cohort repectively.

```
# the age effect
ageeffect <- ~ factor(age)

# the year effect
yeareffect <- ~ factor(year)

# the cohort
cohorteffect <- ~ factor(year-age)

# the fits
fit1 <- sca(ple4, ple4.indices, fmodel=yeareffect)

## Error:  could not find function "sca"

fit2 <- sca(ple4, ple4.indices, fmodel=ageeffect)

## Error:  could not find function "sca"

fit3 <- sca(ple4, ple4.indices, fmodel=cohorteffect)

## Error:  could not find function "sca"
```

and the graphical representation of the three models in Figure **??**

```
wireframe(data~year*age, data=harvest(fit1), main='year effect')

## Error:  could not find function "wireframe"

wireframe(data~year*age, data=harvest(fit2), main='age effect')

## Error:  could not find function "wireframe"

wireframe(data~year*age, data=harvest(fit3), main='cohort effect')

## Error:  could not find function "wireframe"
```

1.7 The statistical catch-at-age stock assessment framework advanced features - the a4aSCA method

A more advanced method for stock assessment can be used through the `a4aSCA()` method. This method gives access to the submodels for $N1$, σ^2_{ay} and I_{ays} as well as arguments to get the ADMB files, etc. Check the manual pages with `?a4aSCA` for more information. This method has 'assessment' as the default value for the `fit` argument, which means that the hessian is going to be computed and all the information about the parameters will be returned by default. Note that the default models of each submodel can be accessed with

```
fit <- a4aSCA(ple4, ple4.indices[1])

## Error:  could not find function "a4aSCA"

submodels(fit)

## Error:  could not find function "submodels"
```

1.7.1 N1 model

The submodel for the stock number at age in the first year of the time series is set up with the usual linear tools (Figure **??**), but bare in mind that the year effect does not make sense here.

```
n1mod <- ~s(age, k=4)
fit1 <- a4aSCA(ple4, ple4.indices[1], n1model=n1mod)

## Error:  could not find function "a4aSCA"

flqs <- FLQuants(smo=stock.n(fit1)[,1], fac=stock.n(fit)[,1])

## Error:  could not find function "FLQuants"
```

```
xyplot(data~age, groups=qname, data=flqs, type="l", auto.key=list(points=FALSE, lines=TRUE, columns=2))

## Error:  could not find function "xyplot"
```

1.7.2 Variance model

The variance model allows the user to set up the shape of the observation variances σ^2_{ay} and I_{ays}. This is an important subject related with fisheries data used for input to stock assessment models. It's quite common to have more precision on the most represented ages and less precision on the less frequent ages. This is due to the fact that the last ages do not appear as often at the auction markets, in the fishing operations or on survey samples.

By default the model assumes constant variance over time and ages (1 model) but it can use other models specified by the user. As with the other submodels, R linear model capabilities are used (Figure ??).

```
vmod <- list(~1, ~1)
fit1 <- a4aSCA(ple4, ple4.indices[1], vmodel=vmod)

## Error:  could not find function "a4aSCA"

vmod <- list(~ s(age, k=4), ~1)
fit2 <- a4aSCA(ple4, ple4.indices[1], vmodel=vmod)

## Error:  could not find function "a4aSCA"

flqs <- FLQuants(cts=catch.n(fit1), smo=catch.n(fit2))

## Error:  could not find function "FLQuants"
```

```
xyplot(data~year|age, groups=qname, data=flqs, type="l", scales=list(y=list(relation="free")), auto.key

## Error:  could not find function "xyplot"
```

1.7.3 Working with covariates

In linear model one can use covariates to explain part of the variance observed on the data that the 'core' model does not explain. The same can be done in the a4a framework. The example below uses the North Atlantic Oscillation (NAO) index to model recruitment.

```
nao <- read.table("http://www.cdc.noaa.gov/data/correlation/nao.data", skip=1, nrow=62, na.strings="-99
dnms <- list(quant="nao", year=1948:2009, unit="unique", season=1:12, area="unique")
nao <- FLQuant(unlist(nao[,-1]), dimnames=dnms, units="nao")

## Error:  could not find function "FLQuant"

nao <- seasonMeans(trim(nao, year=dimnames(stock.n(ple4))$year))

## Error:  could not find function "seasonMeans"

nao <- as.numeric(nao)

## Error:  (list) object cannot be coerced to type 'double'
```

First by simply assuming that the index drives recruitment (Figure ??).

```
srmod <- ~ nao
fit2 <- sca(ple4, ple4.indices[1], qmodel=list(~s(age, k=4)), srmodel=srmod)
```

```
## Error:  could not find function "sca"

flqs <- FLQuants(simple=stock.n(fit)[1], covar=stock.n(fit2)[1])

## Error:  could not find function "FLQuants"
```

```
## Error:  could not find function "xyplot"
```

In a second model we're using the NAO index not to model recruitment directly but to model one of the parameters of the S/R function (Figure ??).

```
srmod <- ~ ricker(a=~nao, CV=0.1)
fit3 <- sca(ple4, ple4.indices[1], qmodel=list(~s(age, k=4)), srmodel=srmod)

## Error:  could not find function "sca"

flqs <- FLQuants(simple=stock.n(fit)[1], covar=stock.n(fit3)[1])

## Error:  could not find function "FLQuants"
```

```
## Error:  could not find function "xyplot"
```

Note that covariates can be added to any submodel using the linear model capabilities of R.

1.7.4 Assessing ADMB files

The framework gives access to the files produced to run the ADMB fitting routine through the argument `wkdir`. When set up all the ADMB files will be left in the directory. Note that the ADMB tpl file is distributed with the `FLa4a`. One can get it from your R library, under the folder `myRlib/FLa4a/admb/`.

```
fit1 <- a4aSCA(ple4, ple4.indices, wkdir="mytest")

## Error:  could not find function "a4aSCA"
```

1.8 Predict and simulate

To predict and simulate R uses the methods `predict()` and `simulate()`, which were implemented in `FLa4a` in the same fashion.

```
fit <- sca(ple4, ple4.indices[1], fit="assessment")

## Error:  could not find function "sca"
```

1.8.1 Predict

Predict simply computes the quantities of interest using the estimated coefficients and the design matrix of the model.

```
fit.pred <- predict(fit)

## Error:  object 'fit' not found

lapply(fit.pred, names)

## Error:  object 'fit.pred' not found
```

1.8.2 Simulate

Simulate uses the variance-covariance matrix computed from the Hessian returned by `ADMB` and the fitted parameters, to parametrize a multivariate normal distribution. The simulations are carried out using the method `mvrnorm()` provided by the R package `MASS`. Figure ?? shows a comparison between the estimated values and the medians of the simulation, while Figure ?? presents the stock summary of the simulated and fitted data.

```
fits <- simulate(fit, 100)

## Error:  object 'fit' not found

flqs <- FLQuants(sim=iterMedians(stock.n(fits)), det=stock.n(fit))

## Error:  could not find function "FLQuants"
```

```
## Error:  could not find function "xyplot"
```

```
## Error:  object 'ple4' not found
## Error:  object 'stks' not found
```

1.9 Geeky stuff

A lot more can be done with the a4a framework. The next sections will describe methods that are more technical. What we'd categorize as 'matters for geeks', in the sense that these methods usually will require the users to 'dive' into `R` a bit more.

```
fit <- sca(ple4, ple4.indices[1], fit="assessment")

## Error:  could not find function "sca"
```

1.9.1 External weigthing of likelihood components

By default the likelihood components are weighted using inverse variance. However, the user may change the weights by setting the variance of the input parameters. This is done by adding a variance matrix to the `catch.n` and `index.n` slots of the stock and index objects. These variances will be used to penalize the data during the likelihood computation. The values should be given as coefficients of variation on the log scale, so that variance is $\log(CV^2 + 1)$. Figure ?? shows the results of two fits with distinct likelihood weightings.

```
stk <- ple4

## Error:  object 'ple4' not found

idx <- ple4.indices[1]

## Error:  object 'ple4.indices' not found

# variance of observed catches
varslt <- catch.n(stk)

## Error:  could not find function "catch.n"

varslt[] <- 0.4

## Error:  object 'varslt' not found

catch.n(stk) <- FLQuantDistr(catch.n(stk), varslt)

## Error:  could not find function "FLQuantDistr"

# variance of observed indices
varslt <- index(idx[[1]])

## Error:  could not find function "index"

varslt[] <- 0.1

## Error:  object 'varslt' not found

index.var(idx[[1]]) <- varslt

## Error:  object 'varslt' not found

# run
fit1 <- a4aSCA(stk, idx)

## Error:  could not find function "a4aSCA"

flqs <- FLQuants(nowgt=stock.n(fit), extwgt=stock.n(fit1))

## Error:  could not find function "FLQuants"
```

```
## Error:  could not find function "xyplot"
```

1.9.2 More models

There's a set of methods that allow the user to have more flexibility on applying the models referred before. For example to break the time series in two periods, using the method `breakpts()`, or fixing some parts of the selection pattern by setting F to be the same for a group of ages, using `replace()`.

The example below (Figure **??**) replaces all ages above 5 by age 5, which means that a single coefficient is going to be estimated for age 5-10.

```
fmod <- ~ s(replace(age, age>5, 5), k=4) + s(year, k=20)
fit <- sca(ple4, ple4.indices, fmod)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

Or else one can use a closed form fort the selection pattern. The example below uses a logistic form (Figure ??).

```
fmod <- ~ I(1/(1+exp(-age)))
fit <- sca(ple4, ple4.indices, fmod)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

In the next case we'll use the `breakpts()` to split the time series at 1990, although keeping the same shape in both periods, a thin plate spline with 3 knots (Figure ??).

```
fmod <- ~s(age, k = 3, by = breakpts(year, 1990))
fit <- sca(ple4, ple4.indices, fmod)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

More complicated models can be built with these tools. For example, Figure ?? shows a model where the age effect is modelled as a smoother (the same thin plate spline) throughout years but independent from each other.

```
fmod <- ~ factor(age) + s(year, k=10, by = breakpts(age, c(2:8)))
fit <- sca(ple4, ple4.indices, fmod)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

A quite complex model that implements a cohort effect can be set through the following formula. Figure ?? shows the resulting fishing mortality. Note that in this case we end up with a variable F pattern over time, but rather than using 4 * 10 = 40 parameters, it uses, 4 + 10 + 10 = 24.

```
fmodel <- ~ s(age, k = 4) + s(pmax(year - age, 1957), k = 10) + s(year, k = 10)
fit <- sca(ple4, ple4.indices, fmodel=fmodel)

## Error:  could not find function "sca"
```

```
## Error:  could not find function "wireframe"
```

1.9.3 Propagate natural mortality uncertainty

In this section we give an example of how uncertainty in natural mortality, set up using the `m()` method and the class *a4aM* (see Section **??**), is propagated through the stock assessment. We'll start by fitting the default model to the data.

```
data(ple4)

## Warning:  data set 'ple4' not found

data(ple4.indices)

## Warning:  data set 'ple4.indices' not found

fit <- sca(ple4, ple4.indices)

## Error:  could not find function "sca"
```

Using the a4a methods we'll model natural mortality using a negative exponential model by age, Jensen's estimator for the level and a constant trend with time. We include multivariate normal uncertainty using the `mvrnorm()` method and create 25 iterations.

```
nits <- 25

shape <- FLModelSim(model=~exp(-age-0.5))

## Error:  could not find function "FLModelSim"

level <- FLModelSim(model=~k^0.66*t^0.57, params = FLPar(k=0.4, t=10), vcov=matrix(c(0.002, 0.01,0.01,

## Error:  could not find function "FLModelSim"

trend <- FLModelSim(model=~b, params=FLPar(b=0.5), vcov=matrix(0.02))

## Error:  could not find function "FLModelSim"

m4 <- a4aM(shape=shape, level=level, trend=trend)

## Error:  could not find function "a4aM"

m4 <- mvrnorm(nits, m4)

## Error:  could not find function "mvrnorm"

range(m4)[] <- range(ple4)[]

## Error:  object 'ple4' not found

range(m4)[c("minmbar","maxmbar")]<-c(1,1)

## Error:  object 'm4' not found

flq <- m(m4)[]

## Error:  could not find function "m"
```

```
quant(flq) <- "age"

## Error:  object 'flq' not found

stk <- propagate(ple4, nits)

## Error:  could not find function "propagate"

m(stk) <- flq

## Error:  object 'flq' not found
```

We fit the same model to the new stock object which has uncertainty in the natural mortality. The assessment is performed for each of the 25 iterations.

```
fit1 <- sca(stk, ple4.indices)

## Error:  could not find function "sca"
```

And compare the two results (Figure ??). It's quite easy to run these kind of tests and a large part of our effort is to create the tools to do so.

```
## Error:  could not find function "FLStocks"
```

1.9.4 WCSAM exercise - replicating itself

The World Conference on Stock Assessment Methods (WCSAM) promoted a workshop where a large simulation study was used to test the performance of distinct stock assessment models. The first criteria used was that the models should be able to reproduce itself. The process involved fitting the model, simulating observation error using the same model, and refitting the model to each iteration. The final results should be similar to the fitted results before observation error was added (see Deroba, et.al, 2014 for details). The following analysis runs this analysis and Figure ?? presents the results.

```
# number of iters
nits <- 25
# fit the model
fit <- a4aSCA(ple4, ple4.indices[1])

## Error:  could not find function "a4aSCA"

# update the stock data
stk <- ple4 + fit

## Error:  object 'ple4' not found

# simulate controlling the random seed
fits <- simulate(fit, nits, 1234)

## Error:  object 'fit' not found

# update stock and index data, now with iters
stks <- ple4 + fits

## Error:  object 'ple4' not found
```

```
idxs <- ple4.indices[1]

## Error:  object 'ple4.indices' not found

index(idxs[[1]]) <- index(fits)[[1]]

## Error:  could not find function "index"

# run assessments on each iter
sfit <- a4aSCA(stks, idxs, fit="MP")

## Error:  could not find function "a4aSCA"
```

```
## Error:  could not find function "FLStocks"
```

1.9.5 Parallel computing

This is an example of how to use the `parallel` R package to run assessments. In this example each iteration is a dataset, including surveys, and we'll run one assessment for each iteration. Afterwards the data is pulled back together in an *FLStock* object and plotted (Figure ??). Only 20 iterations are run to avoid taking too long. Also note that we're using 4 cores. This parameter depends on the computer being used. These days almost all computers have at least 2 cores.

Finally, compare this code with the one for replicating WCSAM and note that it's exactly the same, except that we're using `mclapply()` from package `paralell` instead of `lapply()`.

```
data(ple4)

## Warning:  data set 'ple4' not found

data(ple4.indices)

## Warning:  data set 'ple4.indices' not found

nits <- 25
fit <- a4aSCA(ple4, ple4.indices[1])

## Error:  could not find function "a4aSCA"

stk <- ple4 + fit

## Error:  object 'ple4' not found

fits <- simulate(fit, nits, 1234)

## Error:  object 'fit' not found

stks <- ple4 + fits

## Error:  object 'ple4' not found

idxs <- ple4.indices[1]

## Error:  object 'ple4.indices' not found
```

```
index(idxs[[1]]) <- index(fits)[[1]]

## Error:  could not find function "index"

library(parallel)
options(mc.cores=3)
lst <- mclapply(split(1:nits, 1:nits), function(x){
        out <- try(a4aSCA(iter(stks, x), FLIndices(iter(idxs[[1]], x)), fit="MP"))
        if(is(out, "try-error")) NULL else out
})

stks2 <- stks

## Error:  object 'stks' not found

for(i in 1:nits){
        iter(catch.n(stks2), i) <- catch.n(lst[[i]])
        iter(stock.n(stks2), i) <- stock.n(lst[[i]])
        iter(harvest(stks2), i) <- harvest(lst[[i]])
}

## Error:  could not find function "catch.n"

catch(stks2) <- computeCatch(stks2)

## Error:  could not find function "computeCatch"

stock(stks2) <- computeStock(stks2)

## Error:  could not find function "computeStock"

stks3 <- FLStocks(orig=stk, sim=stks, fitsim=stks2)

## Error:  could not find function "FLStocks"
```

```
## Error:  object 'stks3' not found
```

1.10 Model averaging

To merge results from several fits, using distinct models or datasets, we follow Millar, et.al, 2014. The method `ma()` is the wrapper to the distinct methods, although for now only the AIC averaging is implemented. Figures ?? and ?? show the results.

```
data(ple4)

## Warning:  data set 'ple4' not found

data(ple4.indices)

## Warning:  data set 'ple4.indices' not found

f1 <- sca(ple4, ple4.indices, fmodel=~ factor(age) + s(year, k=20), qmodel=list(~ s(age, k = 4), ~ s(ag
## Error:  could not find function "sca"

f2 <- sca(ple4, ple4.indices, fmodel=~ factor(age) + s(year, k=20), qmodel=list(~ s(age, k = 4)+year, ~

## Error:  could not find function "sca"

stock.sim <- ma(a4aFitSAs(list(f1=f1, f2=f2)), ple4, AIC, nsim = 100)

## Error:  could not find function "ma"

stks <- FLStocks(f1=ple4+f1, f2=ple4+f2, ma=stock.sim)

## Error:  could not find function "FLStocks"
```

```
## Error:  object 'ssb' not found
## Error:  object 'iterMedians' not found
## Error:  could not find function "xyplot"
```

```
## Error:  object 'stks' not found
```