Virtual Population analysis using eXtended Survivor Analysis

11 August, 2017

Required packages

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

- FLR: FLCore, FLAssess, FLXSA, ggplotFL
- CRAN: [reshape] You can do so as follows,

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

```
# This chunk loads all necessary packages,
# trims pkg messages
library(FLCore)
library(FLAssess)
library(FLXSA)
```

```
[1] "plot"
```

Introduction

What is VPA

Virtual population analysis (VPA) is a modeling technique commonly used in fisheries science for reconstructing the historical population structure of an age structured fish stock using information on the deaths of individuals in each time step. The time steps are typically, though not necessarily, annual and the deaths are usually partitioned into mortality due to fishing and natural mortality. In some instances natural mortality may be further partitioned into predation mortality and mortality from other causes, such as disease, senesence etc.

VPA is the most commonly used term to refer to cohort reconstruction techniques used in fisheries. It is virtual in the sense that the population size is not observed or measured directly but is inferred or backcalculated to have been a certain size in the past. Several different software implementations of cohort reconstruction for fish populations exist including ADAPT which is often used in Canada and the USA and XSA (**???**) which is commonly used in Europe. The back-calculations in these implementations work the same way but they differ in the statistical methods used for “tuning” to indices of population size. Tuning refers to the use of auxilliary information to determine the terminal fishing mortalities and population numbers. Most tuning approaches involve a regression of fishing mortality against fishing effort to estimate population abundance at age through an iterative convergence to some threshold criterion. Relatively simple techniques, the Laurec-Shepherd method (**???**) for example, have been shown to work well with simulated data but there is little theoretical work to justify or validate these approaches (**???**).

A number of assessment methods are made available in FLR as well as the basic VPA tools to enable you to develop your own assessment methods. In this tutorial we will cover the basic VPA tools, simple methods for tuning a VPA and finally show how to run **FLXSA**.

Stock assessment methods within the FLR package structure

The package **FLAssess** contains the basic class for age and biomass based stock assessments. It provides a standard class, **FLAssess**, for data input,

stock status estimation and diagnostic inspection. The **FLAssess** package has a variety of uses. It can be applied within a stock assessment working group setting or, alternatively, as part of the management procedure in a formal Management Strategy Evaluation (MSE). FLAssess provides a common interface for existing stock assessment methods (e.g. XSA) allowing methods to be used interchangeably. It also includes various methods of general use such as setting up a short-term forecast (`stf`), running VPAs (`VPA` or `SepVPA`) and calculating F from catches. There are several steps to be completed when conducting an assessment. This tutorial considers only the process of running `VPA` and `FLXSA` stock assessment model.

Additional tutorials are available, e.g. A quick introduction to FLR, An overview of the FLCore classes, Loading your data into FLR.

We will start by importing the data sets for the North Sea Plaice stock and the fishery independent abundance indices. We will use these example data sets for all of the examples in this tutorial.

```r
data(ple4)
data(ple4.indices)
```

The North Sea Plaice `FLStock` object already has values estimated for harvest and stock numbers. We should remove these first and replace them with `NA`.

```r
harvest(ple4)[] <- NA
stock.n(ple4)[] <- NA
```

We should note at this point that the example below should not be considered the definitive assessment for the North Sea Plaice. We provide this example merely to show the procedure for conducting assessments using FLR.

The VPA method

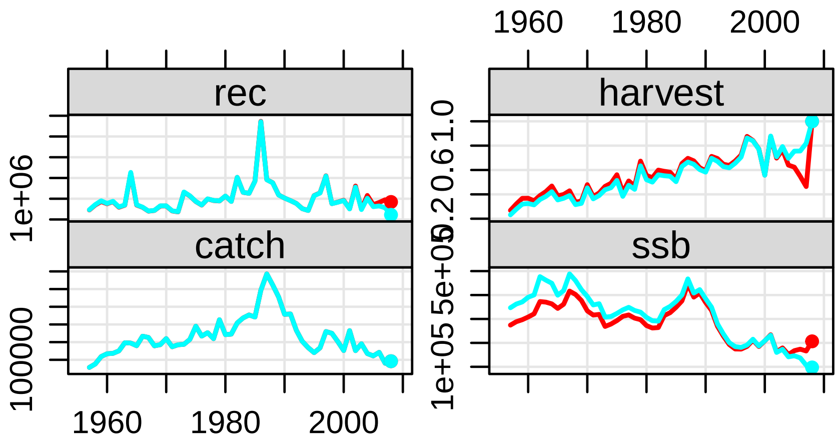
The `VPA` method implements Pope's Virtual Population Analysis (VPA). It is called with the command VPA which returns an object of class `FLVPA` that is itself an extension of the `FLAssess` class. The `VPA` method estimates population numbers and fishing mortalities at age by back-calculating values down each cohort. To do this, the method requires initial values of harvest for the terminal age and terminal year in the `FLStock` object. These terminal values must be specified by the user prior to running the `VPA`. The arguments to the `VPA` method are the `FLStock` object for which values are to be calculated and two optional arguments.

The range method will show details of the age and year range of the `ple4` `FLStock` object. We can use this information to manually specify the terminal values in the harvest slot. In this instance we will set these

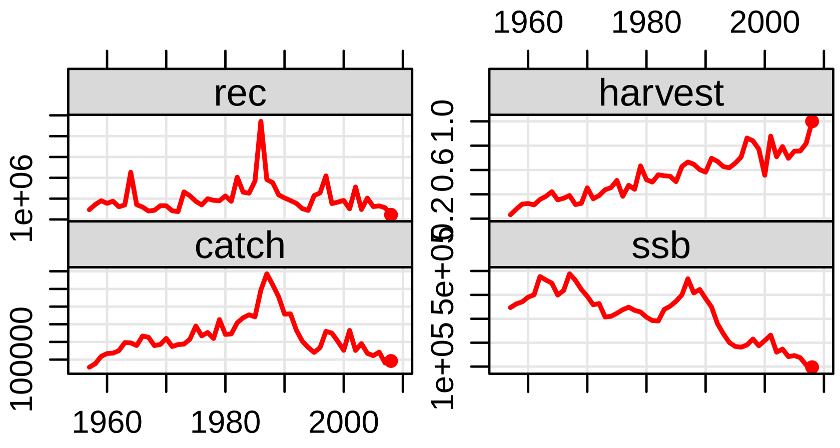
values to 1.0. Remember to convert the values to be of type character when indexing the `FLQuants`.

```
harvest(ple4)[ac(range(ple4)["max"]), ] <- 1
harvest(ple4)[, ac(range(ple4)["maxyear"])] <- 1

ple4.vpa <- VPA(ple4, fratio = 1, fit.plusgroup = T)
ple4.new <- ple4 + ple4.vpa

## Have a look in stock number ##
stock.n(ple4.vpa)[, ac(2005:range(ple4)["maxyear"])]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

    year
age  2005     2006     2007     2008
  1  617107.6 651356.8 562832.6 223891.7
  2  707289.2 466784.2 380163.8 434637.0
  3  151485.3 316039.3 191802.6 132333.9
  4  226211.3  68925.0 118240.4  72891.0
  5   38136.3 107157.4  28441.5  52364.4
  6   28841.9  13983.0  53883.2  10250.9
  7    9605.1  11355.9   6069.8  25778.2
  8    5609.2   3510.6   4868.9   2972.9
  9    4731.0   2602.7   1059.0   1109.7
  10   1304.6   2779.7   2455.4   1477.4

units:  10^3

## Have a look in fishing mortality ##
harvest(ple4.vpa)[, ac(2004:range(ple4)["maxyear"])]

An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

    year
age  2004    2005    2006    2007    2008
  1  0.27236 0.17918 0.43846 0.15847 1.00000
  2  0.78482 0.70557 0.78940 0.95527 1.00000
  3  0.65511 0.68747 0.88315 0.86750 1.00000
  4  0.72831 0.64717 0.78517 0.71449 1.00000
  5  0.54731 0.90333 0.58748 0.92048 1.00000
  6  0.76362 0.83209 0.73452 0.63729 1.00000
  7  0.59593 0.90651 0.74687 0.61378 1.00000
  8  0.64751 0.66787 1.09844 1.37879 1.00000
  9  0.52522 0.67534 0.68483 0.76661 1.00000
  10 0.52522 0.67534 0.68483 0.76661 1.00000

units:  f

## Plot results ##
plot(FLStocks(ple4 = ple4, vpa = ple4.new))
```



```
plot(FLStocks(vpa = ple4.new))
```



The estimated population numbers and fishing mortality values at age from the `VPA` are now available in the returned object. Note that the terminal values for fishing mortalty are the user defined values that were specified prior to running the `VPA`.

A simple method for tuning a VPA

As noted above the VPA method requires user defined terminal estimates of fishing mortality. This dependency limits the usefulness of the method since it is often the most recent, terminal, estimates that are of most concern to fishery managers. Additional catch at age and effort information, derived either from a sub component of the fishery or from a fishery independent source such as a research survey, can be used to 'tune' the assessment, as described above, and thereby obtain better estimates of fishing mortality and stock numbers in the most recent years. Several so-called ad hoc techniques for tuning a VPA have been developed. A relatively simple technique that has been widely used is the Laurec Shepherd method. This method can be easily implemented in FLR using the basic VPA tools that are provided in the **FLAssess** package.

The example shown below is a simple implementation that allows for a single tuning fleet. With a little extra effort it could be easily extended to accomodate multiple tuning fleets. The technical details of the method are not explained here.

```
# Define Laurec-Sheperd function #

lsm <- function(stock, index, fratio = 1, fit.plusgroup = T) {
    harvest(stock)[, ac(range(stock)["maxyear"])] <- 0.5
    diff <- 1
    while (diff > 1e-06) {
        stock <- stock + VPA(stock, fratio = fratio)
        ages <- range(index)["min"]:range(index)["max"]
        yrs <- range(index)["minyear"]:range(index)["maxyear"]
        stk <- trim(stock, year = yrs, age = ages)
        Cp <- catch.n(index)/catch.n(stk)
        q <- sweep(Cp * harvest(stk), 2, effort(index),
            "/")
        gmq <- apply(q, 1, function(x) exp(mean(log(x),
            na.rm = T)))
        mFp <- gmq * c(apply(effort(index), 1,
            mean))
        Fr <- mFp * (apply(Cp, 1, mean, na.rm = T))^-1
        Fnew <- c(Fr, rep(Fr[ac(max(ages)), ],
            2))
        diff <- sum(abs(harvest(stock)[, ac(range(stock)["maxyear"])] -
            Fnew))
        harvest(stock)[, ac(range(stock)["maxyear"])] <- c(Fnew)
    }
    res <- VPA(stock, fratio = fratio, fit.plusgroup = fit.plusgroup)
    index.res(res) <- FLQuants(q)
    return(res)
}
```

The new Laurec-Shepherd function can now be called without having to specify terminal values in the harvest slot. The arguments to the `VPA` method are also formally declared as arguments to our new function. Note that the function returns an object of class `FLVPA` that has been created from a call to the `VPA` method and that the catchability residuals are stored in the `index.res` slot of the returned object.

```
harvest(ple4)[] <- NA
stock.n(ple4)[] <- NA

ple4.LSvpa <- lsm(ple4, ple4.indices[[1]], fratio = 1,
    fit.plusgroup = T)

ple4.new2 <- ple4 + ple4.LSvpa

stock.n(ple4.LSvpa)[, ac(2005:range(ple4)["maxyear"])]
```

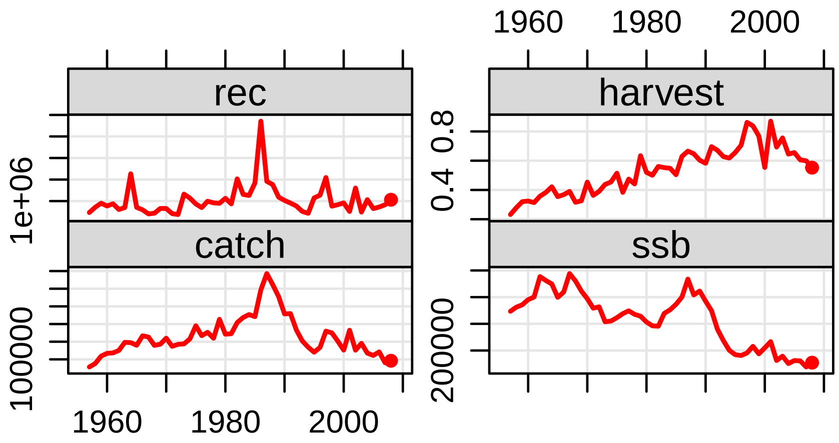
```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique
```

```
    year
age  2005      2006      2007      2008
  1   651630.6  723646.2  830403.7 1062462.8
  2   744226.7  498014.1  445482.0  676711.9
  3   160749.9  349332.2  219926.4  191082.2
  4   260434.0   77277.6  148195.8   98207.7
  5    44423.6  138033.8   35966.1   79384.4
  6    31207.8   19641.1   81763.4   17025.5
  7    10552.0   13485.6   11174.9   50955.4
  8     6662.9    4362.3    6788.9    7585.4
  9     5613.2    3553.2    1824.2    2831.4
  10    1547.9    3794.9    4229.6    3769.6

units:  10^3
```

```
harvest(ple4.LSvpa)[, ac(2004:range(ple4)["maxyear"])]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

    year
age  2004    2005    2006    2007    2008
  1  0.26047 0.16885 0.38515 0.10467 0.14403
  2  0.75311 0.65632 0.71734 0.74645 0.52350
  3  0.59011 0.63245 0.75749 0.70621 0.57850
  4  0.65258 0.53485 0.66483 0.52423 0.63590
  5  0.51494 0.71615 0.42367 0.64787 0.54216
  6  0.71471 0.73904 0.46396 0.37288 0.48158
  7  0.52253 0.78332 0.58633 0.28744 0.38769
  8  0.56992 0.52869 0.77187 0.77453 0.28659
  9  0.45906 0.53499 0.45235 0.37372 0.28659
  10 0.45906 0.53499 0.45235 0.37372 0.28659

units:  f
```

```
# Compare the results with previous fits.
plot(FLStocks(vpa = ple4.new2))
```



FLXSA

The Laurec-Shepherd method above is a relatively simple technique for tuning a VPA. XSA is a

more sophisticated method that uses information on individual cohort sizes to estimate survivors at each age in the terminal population. Although the modelling approach is more involved the method requires the same input of catch numbers at age and indices of catch per unit effort and it retains at its core the basic `VPA` method. The details of the XSA method are too complex to show here, or to code individually as we have for the Laurec-Shepherd approach. Instead the `FLXSA` method has been developed as an additional package to **FLAssess**.

The `FLXSA control` object

The `FLXSA.control` object contains all of the user defined model settings for running an XSA analysis. It can be created in several different ways. The simplest method is to accept all of the default settings by calling the FLXSA.control function without any extra arguments:

```
FLXSA.control()
```

```
tol          1e-09
maxit        30
min.nse          0.3
fse          0.5
rage         0
qage         10
shk.n        TRUE
shk.f        TRUE
shk.yrs          5
shk.ages         5
window       100
tsrange          20
tspower          3
vpa          FALSE
```

Alternatively the default settings can be over-written by specifying values at the point of creation or by overwriting them afterwards.

```
ctrl <- FLXSA.control(maxit = 50, qage = 8)
ctrl <- FLXSA.control()
slot(ctrl, "qage") <- as.integer(8)
slot(ctrl, "maxit") <- as.integer(50)
```

Note that in the example above, when modifying the control object after creation, it is necessary to coerce the values 8 and 50 to type integer. This is because the default type numeric cannot be used in this slot. Such coercion is not necessary when using the `FLXSA.control` function as this check is performed internally by the function. You can use the `getSlots` function to determine the class of object associated with any given slot.

```
xsa.control <- FLXSA.control(maxit = 50, fse = 2.5)
ple4.xsa <- FLXSA(ple4, ple4.indices, xsa.control)
ple4.xsa.t1 <- FLXSA(ple4, ple4.indices[[1]],
    xsa.control)
```

Once the control object has been created, the XSA analysis can be run as a one-line command. The "`FLXSA` function returns an object of class `FLXSA` which extends the `FLAssess` class. The `FLXSA` object contains all of the information in the `FLAssess` class plus additional information specific to the XSA assessment method, such as the survivors estimates and their internal and external standard errors. The control object used for the assessment is also stored in the returned `FLXSA` object to provide a record of what settings were used for that particular run. All of the settings in the returned control object will remain the same except for the `maxit` slot that contains the maximum number of iterations for the analysis. This value will be overwritten with the actual number of iterations taken to reach convergence, if indeed the model had converged before the maximum number initially specified.

XSA Results

Appart from the model diagnostics, the `FLXSA` method returns two important results, namely the estimated values of fishing mortality and population numbers at age. These are returned as `FLQuants` and are stored in the `harvest` and `stock.n` slots, respectively, of the `FLXSA` object. These estimated values can be very easilly read back into an `FLStock` object using the `+` operator. Once the results have been read back into a FLStock object we can look at some of the key information such as SSB, recruitment and mean fishing mortality values. But before concentrating too much on the results of the assessment it is advisable to first investigate some of the model diagnostics.

```
ple4.new <- ple4 + ple4.xsa
ple4.ssb <- ssb(ple4.new)
ple4.rec <- rec(ple4.new)
ple4.fbar <- fbar(ple4.new)
```

XSA Diagnostics

There are many diagnostic checks that one might be interested in conducting to examine the model fit. The first might be to see if the model has reached convergence within the specified number of iterations.

```
slot(slot(ple4.xsa, "control"), "maxit")
```

```
[1] 50
```

Additionally one can check for discrepancies between the internal and external standard errors of the survivors estimates. Very often plots of the catchability residuals are made to inspect for any obvious trends or departures from the assumption of constant catchability over time. Some examples of these plots and details of their creation from FLR objects are provided below but you should also consult the tutorial on lattice plotting and advanced graphics for FLR to see examples of other ways to graphically display your data.

There are several ways to access diagnostic information about your fitted XSA model. The easiest is perhaps to use the `diagnostics` function, which will replicate the diagnostic output produced by the original VPA suite (developed in the early 1990's). Note that this function merely outputs the results to the screen and no object is created by the method. The function was created to allow the user to cut and paste the information from the console to a report. The output can be quite long, particularly if the assessment comprises a large number of ages and many tuning indices. The standard output can be divided roughly into eight sections each providing different information about the model and the fit. These sections comprise the model dimensions; parameter settings; regression weights; the estimated fishing mortalities and population numbers for the last

10 years; the aggregated survivors estimates; the log catchability residuals for each of the tuning indices and finally the individual survivors estimates for each year-class represented in the terminal year.

In order to make this document more readable we will print out only a few sections of the diagnostic output at a time. We can do this by passing a vector of `TRUE` and `FALSE` values to the sections argument of the `diagnostics` method. By default all sections are set to `TRUE` so that all of the information is output to the screen. In order to reduce the quantity of output further

we will run a new XSA for a reduced number of ages and with only one tuning index and will start by outputting only the dimension information and the parameter settings from our diagnostics.

```
ple4.xsa2 <- FLXSA(trim(ple4, age = 1:4), ple4.indices[[3]],
    xsa.control)
diagnostics(ple4.xsa2, sections = c(T, T, rep(F,
    6)))
```

```
FLR XSA Diagnostics 2017-08-11 14:29:26

CPUE data from indices

Catch data for 52 years 1957 to 2008. Ages 1 to 4.

  fleet first age last age first year
1   SNS         1        3       1982
  last year alpha beta
1      2008  <NA> <NA>


 Time series weights :

    Tapered time weighting applied
   Power =   3 over  20 years

 Catchability analysis :

     Catchability independent of size for all ages

     Catchability independent of age for ages >   3

 Terminal population estimation :

     Survivor estimates shrunk towards the mean F
    of the final   5 years or the  5 oldest ages.

    S.E. of the mean to which the estimates are shrunk =   2.5

    Minimum standard error for population
    estimates derived from each fleet =  0.3

    prior weighting not applied
```

Next we can output the regression weights and the fishing mortalities and population numbers for the last 10 years and also the aggregated survivors estimates.

```
diagnostics(ple4.xsa2, sections = c(F, F, T, T,
    T, T, F, F))
```

```
Regression weights
     year
age    1999 2000  2001  2002  2003  2004
  all 0.751 0.82 0.877 0.921 0.954 0.976
     year
age   2005  2006 2007 2008
  all 0.99 0.997    1    1



 Fishing mortalities
   year
age  1999  2000  2001  2002  2003  2004
  1 0.157 0.126 0.078 0.177 0.160 0.257
  2 0.199 0.323 0.783 0.692 0.487 0.766
  3 0.059 0.109 0.778 0.616 0.862 0.320
  4 0.059 0.109 0.778 0.616 0.862 0.320
   year
age  2005  2006  2007  2008
  1 0.180 0.297 0.105 0.188
  2 0.644 0.805 0.491 0.532
  3 0.653 0.729 0.910 0.289
  4 0.653 0.729 0.910 0.289



 XSA population number (Thousand)
      age
year         1       2       3       4
  1999  923227 1263858 7080877 1728304
  2000  943622  713736  937084 2344103
  2001  487694  752940  467320  238728
  2002 2016895  408177  311443  140324
  2003  486915 1529127  184816  124328
  2004 1084190  375601  850173  165566
  2005  614183  759001  158039  224799
  2006  903854  463963  360804   72453
  2007  825289  607783  187650  100907
  2008  834079  671988  336541  184849


 Estimated population abundance at 1st Jan  2009
      age
year   1      2      3      4
  2009 0 627021 358724 229377
```

And finally we can output the catchability residuals and the individual survivors estimates. Note that very little thought went into the parameter settings for this particular model fit so please don't interrogate the output presented here too closely. Also note that we do not normally expect the diagnostics output to be broken up as we have here. We present it in this way purely to make it more presentable in this document. By default all sections are set to TRUE so it is very likely that you won't need to give this argument at all when calling the diagnostics method.

```
diagnostics(ple4.xsa2, sections = c(F, F, F, F,
    F, F, T, T))
```

```


 Fleet:  SNS

 Log catchability residuals.

   year
age   1982   1983  1984   1985   1986   1987
  1  0.161 -0.157 0.129 -0.790 -0.526 -0.769
  2  0.432  0.161 0.276  0.536 -0.495  0.045
  3 -0.273 -1.504 0.056 -0.001 -0.352 -0.718
   year
age   1988   1989   1990   1991  1992   1993
  1 -0.644 -0.139 -0.638  0.544 0.523  0.403
  2 -0.023  0.201 -0.085  0.107 0.701  0.511
  3  0.719  0.357  0.024 -0.074 0.051 -0.349
   year
```

```
age   1994   1995   1996   1997   1998  1999
  1 -0.024 -0.447 -1.170     NA  0.163 0.688
  2  0.531 -0.429  0.268     NA -0.379 0.530
  3 -0.334 -0.378  0.267 -0.055  0.383 0.302
   year
age   2000   2001   2002 2003  2004   2005
  1  0.278  0.219 -0.146   NA 0.004 -0.068
  2 -0.227  0.194 -0.223   NA 0.114 -0.378
  3 -0.380 -0.091 -0.133   NA 0.060 -0.053
   year
age   2006   2007  2008
  1 -0.189 -0.080 0.005
  2  0.082 -0.103 0.060
  3  0.170 -0.058 0.119


 Mean log catchability and standard error of ages with catchability
 independent of year class strength and constant w.r.t. time

                1       2       3
Mean_Logq -3.8398 -5.1320 -6.4314
S.E_Logq   0.4126  0.4126  0.4126


 Terminal year survivor and F summaries:

 ,Age 1 Year class =2007

source
     scaledWts survivors yrcls
SNS      0.971    628413  2007
fshk     0.029    582833  2007

 ,Age 2 Year class =2006

source
     scaledWts survivors yrcls
SNS      0.973    379472  2006
fshk     0.027    278972  2006

 ,Age 3 Year class =2005

source
     scaledWts survivors yrcls
```

```
SNS      0.981    256941  2005
fshk     0.019    175760  2005
```

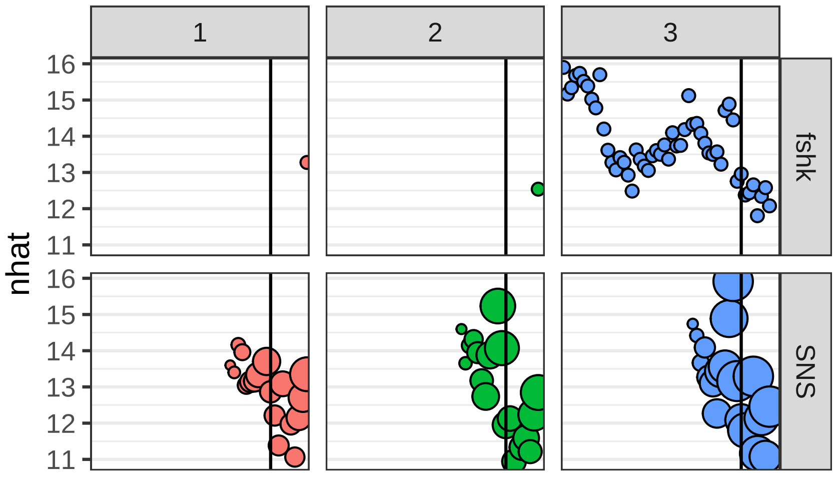
Remember that the diagnostics method will only output text to the console, enabling you to copy and paste the output to a report or other document. If you want to access the diagnostic data you will need to access the specific slots of the returned `FLXSA` object. The information that you will requie is contained in various slots. The individual estimates of population number from each source (ie. tuning series and F shrinkage) and their individual weightings are stored as a dataframe in the `diagnostics` slot of the returned object. Other slots contain the internal and external standard errors; the log catchability residuals. For a more thorough description of the XSA diagnostics you should consult the VPA users manaual.

Plotting Diagnostics

Very often the quickest and simplest way to determine the fit of the model is through visual inspection of the various diagnostic outputs.

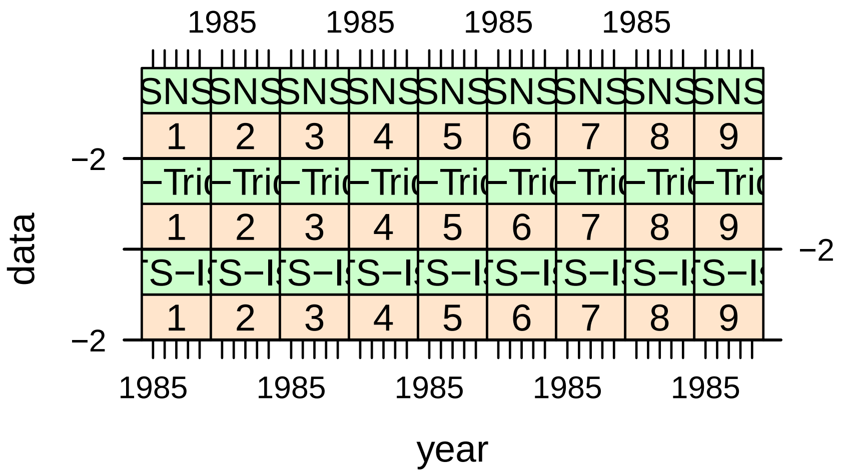
The default plot for '"FLXSA"' class shows the weight given to each of the indices, incluiding the shrinkage, to estimate total numbers at age along ages and years. The size of the bubbles in the plot is proportional to the weight given to the index to estimate the terminal numbers at age. The rows corresponds with the indices used and the columns with age classes. The y axis represent the estimate of numbers at age obtained from each index.

```
plot(ple4.xsa2)
```



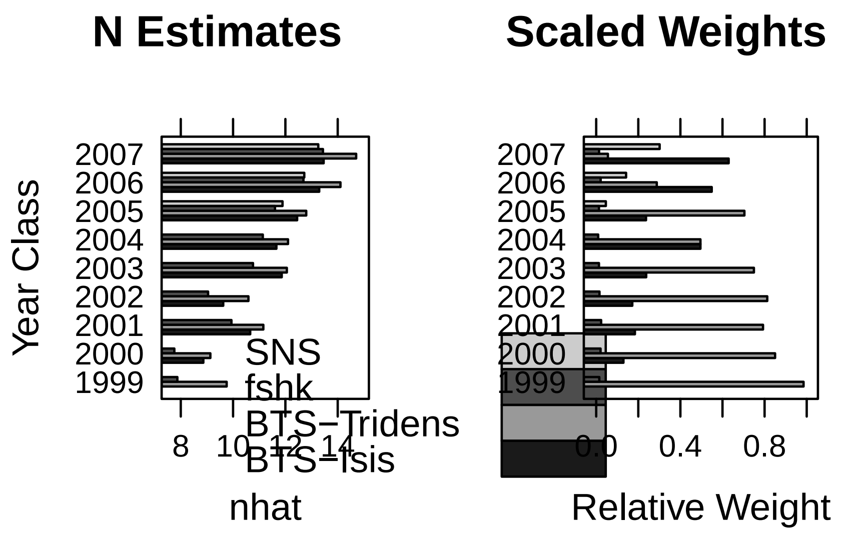
Below we provide examples of how to extract the relevant information from the return `FLXSA` object and to plot it using a variety of lattice functions available to R. We start by plotting the log catchability residuals at age from each of the three tuning series. The data are stored as an FLQuants object in the index.res slot of the FLXSA object. First we need to assign names to each of the FLQuant objects so we know which fleet they represent.

```
names(ple4.xsa@index.res) <- names(ple4.indices)
pfun <- function(x, y, ...) {
    panel.xyplot(x, y, ...)
    panel.loess(x, y, ...)
    panel.abline(h = 0, col = "grey", lty = 2)
}
plot(xyplot(data ~ year | ac(age) + qname, data = index.res(ple4.xsa),
    panel = pfun))
```



A simple comparison of the terminal year survivors estimates can be obtained from the information stored in the diagnostics slot of the FLXSA object. In the following example we first extract the information relevant to the survivors estimates in the final year and store it as a temporary object. The weights values contained in this data set are the raw fleet based weights that have been calculated from the standard errors of the fleet based survivors estimates at each age in the cohort. To aid visualisation and to see the relative contribution of each fleets estimate to the final estimated value of survivors we re-scale the weights to a maximum value of 1 and plot both the fleet based survivors estimates from each fleet and their scaled weight. The results show relatively consistent estimates of survivors from all fleets across most ages. The scaled weights show the some series to have the greatest influence on the terminal estimates at the younger ages whilst others have greater influence at older ages and that throughout all ages F shrinkage recieves very little weighting.

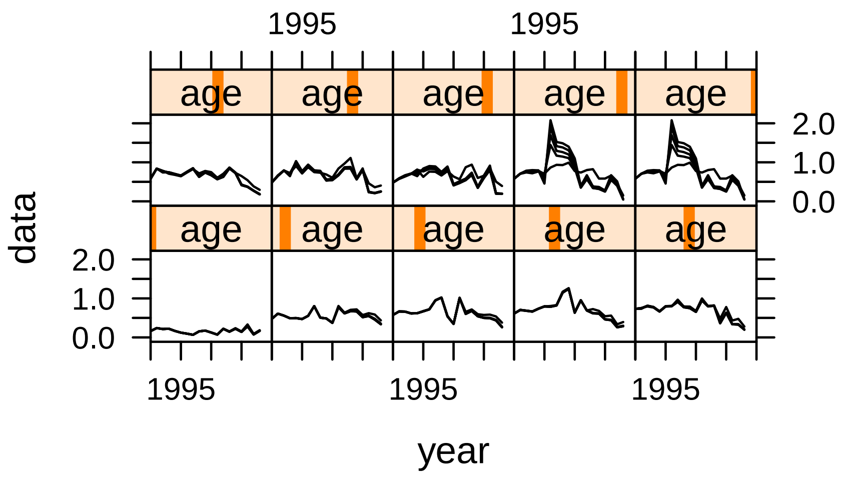
```
diag <- slot(ple4.xsa, "diagnostics")[is.element(slot(ple4.xsa,
    "diagnostics")$year, 2008), ]
diag <- cbind(diag, w.scaled = diag$w/rep(tapply(diag$w,
    diag$yrcls, sum), c(table(diag$yrcls))))
nplot <- barchart(ac(yrcls) ~ nhat, groups = source,
    data = diag, col = grey(c(0.1, 0.6, 0.3, 0.8)),
    main = "N Estimates", ylab = "Year Class",
    key = list(x = 0.3, y = 0.25, text = list(legend = rev(c("BTS-Isis",
        "BTS-Tridens", "fshk", "SNS"))), rectangles = list(col = grey(rev(c(0.1,
        0.6, 0.3, 0.8))))))
wplot <- barchart(ac(yrcls) ~ w.scaled, groups = source,
    data = diag, col = grey(c(0.1, 0.6, 0.3, 0.8)),
    main = "Scaled Weights", ylab = "", xlab = "Relative Weight")
print(nplot, position = c(0, 0, 0.5, 1), more = TRUE)
print(wplot, position = c(0.5, 0, 1, 1))
```



Sensitivity to different model settings.

The simplified calling format of FLXSA makes it very easy to run multiple analyses to investigate model sensitivity to parameter settings. A wide variety of such investigations are possible. In this simple example we will look at the effect that different F shrinkage standard errors have on the terminal estimates of fishing mortality. We start by creating a vector of F shrinkage values to be used in the anlyses and by creating an FLQuant with sufficient dimensions to store the results. To do this we use the propagate function to extend an FLQuant in the 6th dimension by the number of runs that we are going to perform. The estimates of fishing mortality for each XSA run are stored in the FLQuant using the 6th dimension to hold each iteration. The results show little sensitivity to increasing F shrinkage values at values between 1.0 and 2.5 .

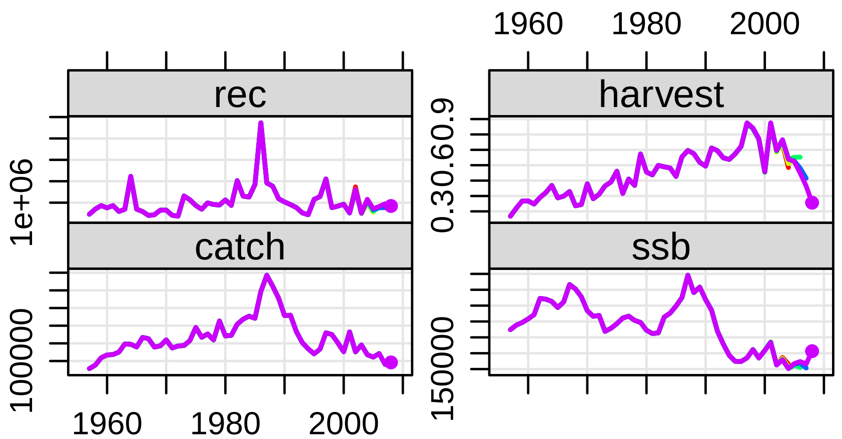
```
fsevals <- seq(0.5, 2.5, by = 0.5)
res <- propagate(harvest(ple4), length(fsevals))
for (i in 1:length(fsevals)) {
    xsa.control <- FLXSA.control(fse = fsevals[i])
    iter(res, i) <- harvest(FLXSA(ple4, ple4.indices,
        xsa.control))
}
plot(xyplot(data ~ year | age, groups = iter,
    data = res, type = "l", col = "black", xlim = c(1990:2010)))
```



Retrospective Analyses

An important diagnostic check is to see how the estimated values vary as the time series of the input data changes. We can make use of existing `R` functions to apply the same assessment model to successively truncated the time series of input data. In this example we are using `window` to truncate the `FLStock` object to the specified year range, the `+` operator to pass the results of the XSA into the `FLStock` object and the `tapply` function to perform this action over the year range `2004:2008`. Note that the resulting object, called `ple4.ret`, is of class `FLStocks` ie. a list of `FLStock` objects, each one having a separate year range.

```
retro.years <- 2004:2008
ple4.retro <- tapply(retro.years, 1:length(retro.years),
    function(x) {
        window(ple4, end = x) + FLXSA(window(ple4,
            end = x), ple4.indices)
    })

# coerce into FLStocks object
ple4.retro <- FLStocks(ple4.retro)
# full retrospective summary plot
ple4.retro@names = ac(c(retro.years))  ###Add years to legend
plot(ple4.retro)
```



More information

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

Software Versions

- R version 3.4.1 (2017-06-30)
- FLCore: 2.6.3.9006
- FLXSA: 2.5.20140808
- FLAssess: 2.6.1
- **Compiled**: Fri Aug 11 14:29:44 2017

License

Author information

Dorleta Garcia AZTI. Marine Reserach Unit. Txatxarramendi Ugartea z/g, 48395, Sukarrieta, Basque Country, Spain.

Alessandro MANNINI. European Commission, DG Joint Research Centre, Directorate D - Sustainable Resources, Unit D.02 Water and Marine Resources, Via E. Fermi 2749, 21027 Ispra VA, Italy. `https://ec.europa.eu/jrc/`

References