Package ‘aMSE’

June 28, 2022

Type Package

Title A Framework for Abalone Management Strategy Evaluation

Version 0.0.7

Date 2022-06-22

Maintainer Malcolm Haddon <`malcolm.haddon@gmail.com`>

Description aMSE provides a formal framework for conducting management strategy evaluation for abalone stocks (although, in principle, such analyses for any sedentary organisms could be conducted). There are functions designed to facilitate the generation of input data file templates, model initiation and conditioning, and a standard template for the generation of new harvest control rules. The vignette Use_aMSE provides details of its operation. Rcpp is included in readiness of speeding various vital dynamics but currently no functions are implemented in Rcpp. The package has now been checked to operate using R4.0.0.

Depends R (>= 3.5.0)

License MIT + file LICENSE

Imports hutils, hplot, captioner, makehtml

RoxygenNote 7.1.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

URL `https://github.com/haddonm/aMSE`

BugReports `https://github.com/haddonm/aMSE/issues`

NeedsCompilation no

Author Malcolm Haddon [aut, cre]

R topics documented:

`addpops` *addpops adds the populations from a single replicate together*

Description

addpops adds the populations from a single replicate together to form zone totals. It can only be used on matureB, exploitB, catch, acatch, and recruits. the sum of acatch is already available as TAC in the projection dynamics object 'zoneDP'

Usage

```
addpops(invar, nyrs, reps)
```

Arguments

`invar`	the summable variable eg catch, recruits, matureB
`nyrs`	the number of years of data
`reps`	the number of replicates

Value

a 2D matrix of yrs x reps

Examples

```
print("wait on data")
```

`addrecvar` *addrecvar adds recruitment variation to end of conditioning step*

Description

addrecvar replicates the historical depletion step in zoneDD reps times and copies the first Nyrs - varyrs of values into each replicate. Then it steps through each replicate adding recruitment variation to the dynamics for the last varyrs years. This is so that when the projections begin each replicate will already have recruitment variability fully developed and there will be no time lag on the recruitment and subsequent dynamics. Take especial note that prior to the projections the proposed harvest strategy needs to be applied to the historical fishery data (in Tasmania this is currently catches and CPUE). This application occurs in lines 60 - 65 in the addrecvar function. This WILL NEED ATTENTION with other jurisdictions.

Usage

```
addrecvar(
  zoneDD,
  zoneC,
  glob,
  condC,
  ctrl,
  varyrs,
  calcpopC,
  sigR = 1e-08,
  sigB = 1e-08,
  lastsigR = 0.3
)
```

Arguments

`zoneDD`	the dynamic zone after historical fishery data has been used to finalize the conditioning #param zoneDDR the empty dynamic zone object ready for expansion and variation
`zoneC`	the zone's constants object for each population
`glob`	the object containing the global constants
`condC`	the object containing the historical fishery data

`ctrl`	the control object
`varyrs`	the number of years at the end of the historical period to which recruitment variation is to be added
`calcpopC`	a function that takes the output from hcrfun and generates the actual catch per population expected in the current year.
`sigR`	the initial recruitment variation default=1e-08
`sigB`	the initial biomass cpuie variation default = 1e-08
`lastsigR`	the recruitment variation to be added to the final varyrs

Value

an initialized dynamics zone object for the projections with the first year populated

Examples

```
print("wait on suitable data-sets")
```

`alldirExists` *alldirExists Checks the existence of both a run and data directory*

Description

alldirExists answers the questions 'do both a rundir and a datadir directory exist?' It uses dir.exists and reports existence if already present and provides a stop warning if either does not exist. Of course it can also be used to determine whether a single directory exists. By default the second directory, indir2 = indir1. This allows for the datadir = rundir within aMSE, but also allows for a separate datadir.

Usage

```
alldirExists(indir1, indir2 = indir1, make = FALSE, verbose = TRUE)
```

Arguments

`indir1`	a character string containing the name of the first directory whose existence is to be checked before it is created if it does not already exist.
`indir2`	a character string containing the name of a second directory whose existence is to be checked, by default this has the same value as indir1
`make`	if the directory does NOT exist should it be created. default = FALSE; if make=FALSE and a directory does not exist a warning will be given to the console.
`verbose`	default=TRUE, prints directory status to the console, If make is set to FALSE and a directory does not exist a warning will always be given.

Value

a message to the screen if the directory exists or is created; if make is TRUE then it also creates the directory as listed in 'indir1'.

Examples

```
indirect <- getwd()
alldirExists(indirect)
```

`aMSE` *aMSE functions for Conditioning and Running an Invertebrate MSE*

Description

The aMSE package provides functions to facilitate the conditioning and running of an invertebrate Management Strategy Evaluation system. The Operating model dynamics are based around using size-based dynamics rather than age-based dynamics (because most invertebrates, like abalone, are difficult to age accurately and consistently). The growth dynamics are described using the inverse logistic curve (see Haddon et al. 2008). Earlier versions of this invertebrate MSE are described on A development version is #useDynLib aMSE, registration = TRUE #importom Rcpp evalCpp available on GitHub at github.com/haddonm/aMSE.

Data sets

constants is conditioning data for 6 populations

ctrl control file for a 2 MSU 6 population MSE run

midg an abalone tagging data-set from the Actaeons

product productivity matrix from a 2 MSU 6 population example

zone1 the constants common to a zone

tasab Abalone maturity data from two sites on the south-west of Tasmania.

taszoneC a zone list made up of 6 equilibrium populations

taszoneD a list of 8 matrices and 2 arrays defining the dynamics of a zone

References

Haddon, M., Mundy, C., and D. Tarbath (2008) Using an inverse-logistic model to describe growth increments of blacklip abalone (*Haliotis rubra*) in Tasmania. *Fishery Bulletin* **106**: 58-71.

Haddon, M., Mayfield, S., Helidoniotis, F., Chick, R. and C. Mundy (2013) *Identification and Evaluation of Performance Indicators for Abalone Fisheries*. FRDC Final Report 2007/020. CSIRO Oceans and Atmosphere and Fisheries Research Development Corporation. 295 p.

Haddon, M. and C. Mundy (2016) *Testing abalone empirical harvest strategies for setting TACs and associated LMLs, which include the use of novel spatially explicit performance measures*. FRDC Final Report 2011/028. CSIRO Oceans and Atmosphere and Fisheries Research Development Corporation. Hobart 182 p

asSAU *asSAU generates an output list of SAU values from the in put zoneDP*

Description

asSAU takes the population based results in zoneDP and converts them into a list based around SAU. saucpue is already calculated during the application of the harvest strategy.

Usage

```
asSAU(projzone, sauindex, saunames, b0, exb0)
```

Arguments

projzone	the population based results list zoneDP plus the saucpue all from doprojection
sauindex	the SAU index of each population
saunames	the names of each SAU
b0	a vector of the B0 for each SAU
exb0	a vector of the ExB0 for each SAU

Value

a list of results based around SAU

Examples

```
print("wait on new data")
```

aszone *aszone sums the various SAU dynamics and fishery properties*

Description

aszone calculates zone wide totals by summing the mature and exploitable biomass across SAU, it also sums the catches and recruitment levels. It calculates zone wide harvest rates by dividing the catches by the available exploitable biomass. It also calculates the depletion levels of both the mature and exploitable biomass by dividing their zone totals through time by the initial zone wide B0 and ExB0. Finally, it calculates a zone-wide cpue by catch-weighting each of the SAU cpue values and then summing each set of SAU for each year and iteration.

Usage

```
aszone(sauzone, zoneCP)
```

Arguments

sauzone	the output from the applyharvest strategy function, a large list of arrays or results
zoneCP	the constant part of the projection zone

Value

a list of 8 nyrs x reps matrices, summarizing the fishery outputs at the geographical scale of the zone

Examples

```
print("wait on data")
```

---

biology_plots   *biology_plots generates a series of stored plots and tables*

Description

biology_plots generates the yield vs spawning biomass, weight-at-lenght, and emergence plots for all populations. In addition, it also tabulates the biological properties of each population and SAU and the total zone

Usage

```
biology_plots(rundir, glb, zoneC, matL = c(30, 210), Lwt = c(80, 210))
```

Arguments

`rundir`	the results directory
`glb`	the globals list
`zoneC`	the zonal constants by population, zoneC
`matL`	a vector of two containing the left and right hand size classes for use in the maturity-at-length plots
`Lwt`	a vector of two containing the left and right hand size classes for use in the wight-atlength plots

Value

invisibly returns the biological properties of the populations

Examples

```
args(biology_plots)
```

blockE13 — *blockE13 is an abalone data-set for testing performance measures*

Description

blockE13 is a fishery data-set for blacklip abalone (*Haliotis rubra*) from block 13 in the eastern zone, this is Tasmania's Block 13. It constitutes three time-series of the same cpue data in different formats (see below). It is for use when testing the performance measures within Tasmania's MCDA, although it could be used for other purposes, such as illustrating the typical linear relationship between catch and CPUE. Note there will only ever be one less PM value than there are cpue data in the time-series.

Format

A data.frame of abalone fishery data

year the year of fishing

coeff the back-transformed coefficients from a standardization

scaled the coefficients scaled to a mean of one

cpue the coefficients scaled to the nominal geometric mean of the time series, to place it on the nominal scale

catch the related catch from the eastern parts of block 13

Subjects

- performance measure estimation
- relationship between catch and CPUE

Source

Mundy, C. and J.M. McAllister (2020) Tasmanian Abalone Assessment 2019. IMAS, University of Tasmania.

Examples

```
data(blockE13)
blockE13
```

calcprojsel — *calcprojsel generates the selectivity and selectivity x weight*

Description

calcprojsel is used to produce each projection years' selectivity and selectivity x weight-at-size. projSel is an Nclass x projyrs matrix, a separate selectivity for each projection year, which allows for changing the selectivity during the projection. The projSelWt is a Nclass x projyrs x numpop array, required because each population's weight-at-size relationship will be slihtly different.

Usage

```
calcprojsel(zoneC, projC, glb)
```

Arguments

`zoneC`	the constants object
`projC`	the projection object from zone1
`glb`	the globals object

Value

a list of projSel and projSelWt

Examples

```
 print("wait on new example data")
```

`calcsau` *calcsau compares an input variable to a constant for each SAU*

Description

calcsau divides an input variable, such as matureB or exploitB by their respective constant, such as B0 or ExB0, from a vector of values for each SAU. This is used to calculate the depletion levels of mature and exploitable biomass

Usage

```
calcsau(invar, saunames, ref0)
```

Arguments

`invar`	either zoneDP$matureB or zoneDP$exploitB
`saunames`	the names of each SAU
`ref0`	either a vector of B0 or ExB0 for each SAU

Value

an array of projyrs x nSAU x reps

Examples

```
print("wait on new data")
```

`catchweightCE` *catchweightCE uses historical catch-by-sau to make weighted zone cpue*

Description

catchweightCE is used when characterizing the historical fisheries data. It uses matching catch-by-sau to generate catch-weighted estimates of zone-wide CPUE through time.

Usage

```
catchweightCE(cedat, cdat, nsau)
```

Arguments

`cedat`	historical cpue from condC
`cdat`	historical catches from condC
`nsau`	the number of SAU, from glb

Value

a vector of catch-weghted cpue for the zone from SAU cpue estimates

Examples

```
print("wait on suitable data sets")
```

`changecolumn` *changecolumn alters a selected column of recdevs in the control file*

Description

changecolumn is designed to be used when conditioning the OM when optimizing ad hoc recruitment deviates by calculating the SSQ between the predicted and observed CPUE for a selected range of years. If the length of new values is not the same as the selected linerange the function will stop with a warning. When a line is changed then all -1 values are changed to +1

Usage

```
changecolumn(rundir, filename, linerange, column, newvect, verbose = FALSE)
```

Arguments

`rundir`	the rundir for the scenario
`filename`	the character name of the control file
`linerange`	the linerange within the control file containing the rows representing the selected years of recruitment deviates. Obtained by using 'getrecdevcolumn
`column`	the sau + 1 to account for the initial year in each row of deviates
`newvect`	the new values to replace those present. If using optim these would be exp(ans$par), if 'nlm' exp(ans$estimate)
`verbose`	should console reports of before and after be made? default = FALSE

Value

nothing although values within the control file will be changed and, if verbose=TRUE, it will write to the console.

Examples

```
print("wait on suitable internal data sets")
```

`changeline` *changeline replaces a given line in a given file with new text*

Description

changeline enables a text file to be changed line by line. One identifies a given line, perhaps by using 'findlinenumber', and this can then be replaced by an input character string. Obviously this is a fine way to mess up a data file so use with care.

Usage

```
changeline(indir, filename, linenumber, newline, verbose = FALSE)
```

Arguments

`indir`	the directory path in which to find the text file. Usually, rundir
`filename`	the full name of the text file in quotations.
`linenumber`	either the line number within the text file to be changed, or, the character name of the variable to be changed, e.g. 'AvRec'
`newline`	the character string with which to replace the line
`verbose`	should confirmation be output to the console. default=FALSE

Value

nothing but it does alter a line in a text file. Optionally it may confirm the action to the console

See Also

findlinenumber, changevar

Examples

```
print("wait on an example")
```

changevar	*changevar can alter the value in a single line in, eg, the control file*

Description

changevar is a DANGEROUS function for lazy people. I say that because it can damage your control files if not used carefully. It speeds changing a single value within, say, the control.csv file. For example, if one wanted to conduct a retrospective analysis on what would have happened should we have introduced the HS sooner, one could use something like the following to change the value to 44 implying to start projections in 2017. changevar(varname="CATCHES",newvalue=44,filename="controlsau.csv", rundir=rundir) This can only change a single value on a single line but can be used to alter the values of the variables listed in 'goodnames'. Currently, this only works for Windows machine. Let me know if others are required.

Usage

```
changevar(filename, rundir, varname, newvalue, prompt = FALSE, verbose = TRUE)
```

Arguments

`filename`	name of the file to be changed. eg 'control.csv'
`rundir`	name of the scenario directory in which the file is to be found
`varname`	the name of the object whose value is to be changed
`newvalue`	the new value of the object
`prompt`	should allowable names first be listed with current values?
`verbose`	should concluding remarks to console be made. default=TRUE

Value

nothing but a file is changed - be careful

See Also

findlinenumber, changeline

Examples

```
print("Wait on some time passing")
```

`checkmsedata` *checkmsedata contains some tests of the niput MSE saudata file*

Description

checkmsedata contains some tests of the niput MSE saudata file

Usage

```
checkmsedata(intxt, rundir, verbose = TRUE)
```

Arguments

`intxt`	the data file from readLines used in readsaudatafile
`rundir`	the rundir for the scenario
`verbose`	should test results be put to the console as well as filed, default=TRUE

Value

nothing but will write a file to rundir and may write to the console

Examples

```
print("wait on example data sets")
```

`checksizecompdata` *Title*

Description

Title

Usage

```
checksizecompdata(rundir, controlfile, verbose = TRUE)
```

Arguments

`rundir`	the directory in which all files relating to a particular run are to be held.
`controlfile`	default="control.csv", the filename of the control file present in rundir containing information regarding the run.
`verbose`	Should progress comments be printed to console, default=TRUE#'

Value

nothing but it does plot a graph

Examples

```
print("wait on data sets")
```

compareCPUE *compareCPUE plots the historical cpue against conditioned cpue*

Description

compareCPUE generates a plot of the historical cpue and compares it with the predicted cpue from the conditioning. The predicted is black and the observed is green. This is primarily there are an aid to conditioning the operating model. It also calculates the simple sum of squared differences between the observed and predicted, again to aid in the conditioning.

Usage

```
compareCPUE(histCE, saucpue, glb, rundir, filen = "", obscol = 2)
```

Arguments

`histCE`	the matrix of historical cpue by SAU
`saucpue`	the predicted cpue from the conditioning on historical data
`glb`	the globals object
`rundir`	the rundir for the given scenario
`filen`	the file name of the plot if it is to be saved
`obscol`	the colour of the observed CPUE on the plots, default=2=red

Value

a vector of length nsau containing the ssq for each SAU

Examples

```
print("wait on suitable internal data sets")
# sauZone=out$condout$sauZone; saucpue=sauZone$cpue; filen="";histCE=out$condC$histCE
# glb=out$glb;obscol=2
```

comparevar *comparevar generates the quantiles for eahc of a set of input scenarios*

Description

comparevar generates quantiles for each of a set of input scenarios. It takes the full timeline of dynamics and outputs just the projections and the quantiles of those projections

Usage

```
comparevar(dyn, glbc, scenes, var = "cpue")
```

Arguments

`dyn`	this is a list of the out$sauout$zonePsau produced by each scenario. This has to be be generated from teh saved RData files from each scenario
`glbc`	a list of the global objects from each scenario being compared
`scenes`	a list of the out$ctrl$runlabel from each scenario
`var`	what variable from the dynamics to summarize, valid names include catch, acatch, cpue, harvestR, desplsB, depleB, recruit, matureB, and exploitB, default = 'cpue'

Value

a list of the quantiles of the input var for all sau, with each scenario being a different component of the quantscen list, and a list of three dimensional arrays of the actual values of var in the varc list.

See Also

plotscene

Examples

```
print("wait on internal datasets")
```

`compzoneN` *compzoneN compares numbers-at-size before/after depletion*

Description

compzoneN generates a plot comparing the unfished numbers-at-size with those for a given level of depletion.

Usage

```
compzoneN(unfN, curN, glb, yr, depl, ssc = 5, LML = 0, rundir = "")
```

Arguments

`unfN`	the unfished numbers-at-size from getzoneprops
`curN`	the current numbers-at-size from getzoneprops
`glb`	the global object
`yr`	the year of the dynamics
`depl`	the depletion level of the current n-a-s
`ssc`	index for starting size class. thus 1 = 2, 2 = 4, 5 = 10, etc. default = 5 for it plots size classes from 10mm up
`LML`	the legal minimum length in the comparison year
`rundir`	the results directory, default = "" leading to no .png file, just a plot to the screen.

Value

invisibly the filename ready for logfilename

Examples

```
print("still to be developed")
# unfN=unfN; curN=depN;glb=glb; yr=1; depl=0.3993; LML=132; rundir=rundir
```

`confirmdir` *confirmdir checks to see if a directory exists and makes it if not*

Description

confirmdir enables one to be sure a selected directory exists. If it has not bee created then confirmdir will create it if it does not already exist. This is useful when defining output directories on the hard drive where large objects may be stored.

Usage

```
confirmdir(x, make = TRUE, verbose = TRUE)
```

Arguments

`x`	the directory to be checked and created if necessary
`make`	should the directory be created if it does not already exist? default=TRUE
`verbose`	should responses be sent to the console? default=TRUE

Value

nothing but it can create a directory

Examples

```
x <- tempdir()
confirmdir(x)
```

`copyto` *copyto copies a vector of files from one scenario directory to annother*

Description

copyto copies a vector of files (see examples) from one scenario's directory to another. If a filename includes the name of the scenario, eg controlM15h75.csv, is found in the scenario M15h75, and it is to be copied to, say, M15h5, then the filename will be changed automatically. Care should be taken when using this function as it writes files and potentially new directories to your storage drives.

Usage

```
copyto(prefixdir, fromdir, todir, filelist, makenew = TRUE, verbose = TRUE)
```

Arguments

`prefixdir`	the directory (path) containing the different scenarios
`fromdir`	just the name of the directory from which to source the files listed in filelist (no path)
`todir`	just the name of the new destination directory (no path)
`filelist`	a vector of filenames to be copied, as character.
`makenew`	if the 'todir' does not exist should it be created using dir.create? default = TRUE
`verbose`	should details be printed to the console, default=TRUE

Value

Nothing but it will transfer files and change their names. This information will be printed to the console if verbose = TRUE

Examples

```
## Not run:
 prefixdir <- "c:/Users/User/DropBox/A_codeUse/aMSEUse/scenarios/"
 fromdir <- "M15h75"
 todir <- "M15h5"
 vectfiles <- c("controlM15h75.csv","saudataM15h75.csv",
                "lf_WZ90-20.csv","run_aMSE_M15h75.R","TasHS1_Tas.R")
 copyto(prefixdir=prefixdir,fromdir=fromdir,todir=todir,filelist=vectfiles)

## End(Not run)
```

`ctrlfiletemplate` *ctrlfiletemplate generates a template input control file*

Description

ctrlfiletemplate generates a standardized control file template. Generate this and then modify the contents to suit the system you are attempting to simulate. Defaults to 100 replicates. There needs to be as many recdevs as there are conditioning catches (in the template = 58 rows. If they are all set to -1, the default, then they will have no effect and recdevs will be taken off the stock-recruitment curve with random deviates defined by withsigR found in the ctrl file. If devrec = 1.0 then the recdevs from 1980 to 2016 will be set to 1.0 which means there will be no random variation and the system is ready to be conditioned on those recdevs to improve the predicted CPUE during the historical conditioning period. If devrec = 0.0, then the recdevs for 1980 - 2016 will be set to values that have already conditioned the model and provides a reasonable fit to the observed CPUE trends.

Usage

```
ctrlfiletemplate(indir, filename = "controlsau.csv", devrec = -1)
```

Arguments

`indir`	directory in which to place the control.csv file
`filename`	the name for the generated ctrlfile, a character string that defaults to control-sau.csv.
`devrec`	what form should the recdevs take? valid values are -1, the default (= random deviates imposed dependent upon withsigR), 0.0, where a set of conditioned values between 1980 and 2016 are imposed that improve the fit to the observed CPUE as inserted by ctrlfiletemplate, or 1.0, which prepares the recdevs for a round of conditioning ready to improve the observed fit to cpue.

Value

invisibly the fill path and name of the control file. More importantly, it write a control file template to that directory.

See Also

datafiletemplate, readsaudatafile, readctrlfile

Examples

```
## Not run:
 yourdir <- tempdir()
 datafiletemplate(nSAU=8,yourdir,"saudata_test.csv")
 ctrlfiletemplate(yourdir,filename="testctrl.csv",devrec=-1)   #
 control <- readctrlfile(yourdir,"testctrl.csv")
 str(control,max.level=1)

## End(Not run)
```

---

datafiletemplate   *datafiletemplate generates a template input datafile akin to M15h75*

Description

datafiletemplate generates a standard input datafile to use as a template, go in and edit it appropriately to suit your own needs. It contains the probability distributions that are sampled to provide the necessary biological constants for each population. It also contains the proportional distribution of recruitment levels determined for Tasmania by examining the GPS data-logger data for the zone and allocating recruitment in proportion to the relative yield by area over the last 8 years.

Usage

```
datafiletemplate(nSAU, indir, filename = "saudata_test.csv")
```

Arguments

`nSAU`	the number of SAU in the zone
`indir`	the directory into which to place the generated data file
`filename`	the name for the generated datafile, a character string, defaults to saudata_test.csv, which is the default within the ctrlfiletemplate function

Value

a standard definition data file, to be read by readdatafile whose name and path is returned invisibly

See Also

ctrlfiletemplate, readsaudatafile, readctrlfile

Examples

```
## Not run:
 yourdir <- tempdir()
 datafiletemplate(nSAU=8,yourdir,"saudata_test.csv")
 constants <- readsaudatafile(yourdir,"saudata_test.csv")
 str(constants,max.level=1)
 print(constants[,1:10])

## End(Not run)
```

---

defineBlock *defineBlock subdivides defined number of populations into blocks*

Description

subdivides the defined number of populations into blocks in the numbers defined in the constant blkpop read into 'constants'. It allocates the populations sequentially to numblk blocks; if numblk does not divide into numpop exactly, any remainder is put into the final block. For block can read SAU = spatial assessment unit.

Usage

```
defineBlock(numblk, blknum, numpop)
```

Arguments

numblk	number of blocks in the simulated zone, from 'constants'
blknum	number of populations in each block, from 'constants'
numpop	the total number of populations in the zone

Value

a vector of length numpop, indexing which block each pop is in.

Examples

```
  nblock <- 4
  blkpop <- c(3,4,8,2)
  blockI <- defineBlock(nblock,blkpop,numpop=17)
  print(blockI)
  cat(length(blockI),sum(blkpop),"\n")
```

`definepops` *definepops makes a parameter vector, popdef, for each population*

Description

definepops makes a parameter vector, popdef, for each population. Each population will have slightly different biological properties sampled from probability density distributions defined in const=condDat$constants. The input can be either the whole of condDat or just condDat$constants, which contains the definitions of the probability density functions used to define each population.

Usage

```
definepops(inSAU, inSAUindex, const, glob)
```

Arguments

`inSAU`	the number of SAUs defined in the zone, a scalar
`inSAUindex`	a vector of the block index for each population
`const`	a matrix with rows containing the PDF parameters for each of the biological parameters. The columns of the matrix reflect values for each populations from readdatafile
`glob`	the globals object from readzonefile

Value

a matrix with a row for each population and whose columns are parameters defining the biological parameters for each population.

Examples

```
## Not run:
  data(zone1)
  glb <- zone1$globals
  data(constants)
  ans <- makezoneC(zone1,constants)
  zoneC <- ans$zoneC
  popdefs <- ans$popdefs
  print(popdefs)

## End(Not run)
```

`depleteSAU` *depleteSAU resets zoneD, approximately to an input depletion level*

Description

depleteSAU resets the depletion level of the whole zone and does this by searching for the harvest rate that leads to the mature biomass in each population is as close as possible to the desired depletion level. This means the individual populations will likely vary around the target depletion, so the depletion across the zone will only be approximately at the target depletion. The depletion is measured relative to the effective B0 as that takes account of any larval dispersal. This function uses the production curve array to search for harvest rates that bound the target depletion and then re-searches across those bounds using len intervals.

Usage

```
depleteSAU(zoneC, zoneD, glob, initdepl, product, len = 15)
```

Arguments

`zoneC`	the constants components of the simulated zone
`zoneD`	the dynamic components of the simulated zone
`glob`	the general global variables
`initdepl`	a vector of target depletion levels for each SAU. This is found in zone1$condC, but could obviously be modified in each run.
`product`	the production curve matrix from doproduction
`len`	the number of intervals in the trial harvest rates

Value

a revised zoneD object

Examples

```
## Not run:
  data(zone)
  glb <- zone$glb
  depl <- rep(0.3,glb$nSAU)
  zoneDD <- depleteSAU(zone$zoneC,zone$zoneD,glb,initdepl=depl,zone$product)
  sum((zoneDD$matureB[1,]/sum(zoneDD$matureB[1,]))*zoneDD$deplsB[1,])
  mean(zoneDD$deplsB[1,])

## End(Not run)  # zoneC=zone$zoneC;zoneD=zone$zoneD;glob=zone$glb;initdepl=origdepl;product=zone$product;le
```

---

diagnosticsproj *diagnosticsproj plots a series of diagnostics to DiagProj*

Description

diagnosticsproj provides a series of plots and results that illustrate the properties of the projections. These include the residuals between SAU actual catches and their predicted catches. But also a series of plots of the projections for only nrep trajectories to illustrate that the dynamic variables are changing through time in a plausible or 'realistic' manner.

Usage

```
diagnosticsproj(zonePsau, glb, rundir, nrep = 3)
```

Arguments

zonePsau	the SAU scale object containing the dynamics results
glb	the global constants object
rundir	the rundir for the scenario
nrep	the number of replicate trajectories to plot; default=3

Value

Nothing but it does add some plots to rundir

Examples

```
print("wait on suitable data sets")
```

---

dodepletion *dodepletion resets zoneD to an input depletion level*

Description

dodepletion resets the depletion level of the whole zone and does this by searching for the harvest rate that leads to the sum of the mature biomass, across populations, divided by the sum of the effective B0 across populations is as close as possible to the desired depletion level. This means the individual populations will likely vary around the target depletion, but across the zone it will be correct. The depletion is measured relative to the effective B0 as that takes account of any larval dispersal. This function uses the production curve matrix to search for harvest rates that bound the target depletion and then re-searches across those bounds using len intervals.

Usage

```
dodepletion(zoneC, zoneD, glob, depl, product, len = 15)
```

Arguments

`zoneC`	the constants components of the simulated zone
`zoneD`	the dynamic components of the simulated zone
`glob`	the general global variables
`depl`	the target depletion proportion for the whole zone
`product`	the production curve matrix from doproduction
`len`	the number of intervals in the trial harvest rates

Value

a revised zoneD object

Examples

```
## Not run:
  data(zone1)
  glb <- zone1$globals
  data(constants)
  ans <- makezoneC(zone1,constants)
  zoneC <- ans$zoneC
  glb <- ans$glb  # now contains the move matrix
  ans <- makezone(glb,zoneC)
  zoneC <- ans$zoneC
  zoneD <- ans$zoneD
  ans2 <- modzoneC(zoneC,zoneD,glb)
  zoneC <- ans2$zoneC  # now has MSY and deplMSY
  product <- ans2$product
  zoneDD <- dodepletion(zoneC,zoneD,glb,depl=0.3,product)
  sum((zoneDD$matureB[1,]/sum(zoneDD$matureB[1,]))*zoneDD$deplsB[1,])
  mean(zoneDD$deplsB[1,])

## End(Not run)
```

`dohistoricC` *dohistoricC imposes the historical catches on an unfished zone*

Description

dohistoricC is used during the conditioning of the zone/region and imposes the historical catches, held in the zone data object obtained using readzonefile, onto an unfished initial zone, and it does this by imposing the time-series of catches to each SAU/block. The operation is through the use of oneyearC, which imposes one year's catches, which are a vector of SAU catches for each year of the series.

Usage

```
dohistoricC(zoneDD, zoneC, glob, condC, calcpopC, sigR = 1e-08, sigB = 1e-08)
```

Arguments

`zoneDD`	The input unfished dynamic zone, zoneD, object
`zoneC`	the zone constants object, zoneC
`glob`	the globals object
`condC`	from the zone1 object, contains historical fisheries data plus
`calcpopC`	a function that takes the output from hcrfun and generates the actual catch per population expected in the current year.
`sigR`	the recruitment variation included
`sigB`	the variability introduced to the catches by population by fishers not knowing the distribution of exploitable biomass exactly. What this value should be is unknown, the default=1e-08, is arbitrary but avoids any effective fisher allocation error between populations.

Value

a zoneD object

See Also

oneyearsauC, oneyearcat, oneyearrec, do_MSE

Examples

```
print("wait on some data sets")
#  zoneD,zoneC,glob=glb,condC,calcpopC=calcpopC,sigR=1e-08,sigB=1e-08
```

`doproduction` *doproduction estimates a production curve for each population*

Description

doproduction estimates a production curve for each population in the simulated zone. It does this by sequentially applying a series of increasing harvest rates and running the populations to equilibrium to discover, empirically, the yield and other details, such as exploitable biomass, mature biomass, the actual harvest rate, the catch or yield, the mature depletion, and the relative cpue. Keep in mind that the time taken to run this function depends on the number of populations but mainly on the length of the harvest rate sequence, which is determined by lowlim, uplim, and inc. The lonjger the sequence the slower the run. However, the estimates of MSY and related statistics are expected to have better accuracy (resolution) the finer the harvest rate increments. So when initiating the zone it is best to have a long sequence of finely incremented harvest rates that take a long time.

Usage

```
doproduction(zoneC, zoneD, glob, lowlim = 0, uplim = 0.35, inc = 0.005)
```

Arguments

`zoneC`	the constants components of the simulated zone
`zoneD`	the dynamic components of the simulated zone
`glob`	the general global variables
`lowlim`	the lower limit of harvest rate applied, default=0.0
`uplim`	the upper limit of harvest rate applied, default=0.35
`inc`	the harvest rate increment at each step, default=0.005

Value

an array of the six productivity variables by numpop by the number of harvest rates applied

Examples

```
print("wait") #  zoneC=zoneC; zoneD=zoneD; glob=glb; lowlim=0.0;uplim=0.4;inc=0.01
```

`doprojections` *doprojections conducts the replicate model runs for Tasmania*

Description

doprojections conducts the replicate model runs for Tasmania using the mcdahcr and the input hsargs. Note the use of a list that takes up both the population and catch numbers-at-size and puts that into getdata. By referencing the iteration in the arrays being passed to getdata the object sizes are being reduced and hence, hopefully, the runtime will be reduced accordingly.

Usage

```
doprojections(
  ctrl,
  zoneDP,
  zoneCP,
  glb,
  hcrfun,
  hsargs,
  sampleCE,
  sampleFIS,
  sampleNaS,
  getdata,
  calcpopC,
  makehcrout,
  verbose = FALSE,
  ...
)
```

Arguments

`ctrl`	the ctrl object from readctrlfile
`zoneDP`	the object used to contain the dynamics from the replicate model runs
`zoneCP`	the object used to contain the constants for each population used in model dynamics
`glb`	the object containing the global constants for the given run
`hcrfun`	the name of the harvest control rule that is used to calculate the multiplier for the previous aspirational catches (possibly for each SAU but possibly the TAC for the whole zone) so as to estimate the aspirational catches or TAC or the following year
`hsargs`	the constants used to define the workings of the hcr
`sampleCE`	a function that generates the CPUE statistics
`sampleFIS`	a function that generates the FIS statistics
`sampleNaS`	a function that generates the Numbers-at-size samples
`getdata`	a function that gathers all the data required by the hcrfun and combines it into an hcrdata object ready for the hcrfun. It is expected to call sampleCE, sampleFIS, and sampleNAS, even if they only return NULL.
`calcpopC`	a function that takes the output from hcrfun and generates the actual catch per population expected in the current year.
`makehcrout`	is a function from HS.R that produces an object that is updated in each iteration by the hcrfun. If no such object is required then have a function that returns NULL.
`verbose`	should the iterations be counted on the console?
`...`	the ellipsis used in case any of the functions hcrfun, sampleCE, sampleFIS, sampleNas, and getdata require extra arguments not included in the default named collection

Value

a list containing the full dynamics across all years zoneDP, and the final output from the HCR as hcrout

See Also

oneyearsauC, make_html, do_MSE

Examples

```
print("wait on suitable internal data sets")
```

dosau — *dosau plots the conditioning history for the dynamics*

Description

dosau plots the deplsB, cpue, matureB, catch, harvestR, and recruits for a given SAU during the years of conditioning. This aims to assist the conditioning process by illustrating the state of the sau during and at the end of conditioning on the fishery. If extra is TRUE then it also plots the exploitable biomass and the depletion of the exploitable biomass. A loess fit is also plotted onto the recruitments plot to give some insight into the recruitment deviates. Unfortunately, the 'true' predicted recruitment levels (without variation) cannot be easily estimated because the implementation of larval movement disturbs how many recruits are present each year in each population and hence each SAU.

Usage

```
dosau(inzone, glb, picksau, histCE, yrnames, recdev)
```

Arguments

`inzone`	the conditioned zone (zoneDD) after it has been converted to sau scale by using getsauzone
`glb`	the global constants object
`picksau`	which sau should be plotted
`histCE`	the historical cpue series from condC
`yrnames`	the years of historical catches eg 1973:2019
`recdev`	the recdevs for a single SAU

Value

nothing but it does generate a plot

Examples

```
print("wait on suitable data-sets")
```

dosauplot — *dosauplot generates the plot of the chosen variable for each sau*

Description

dosauplot takes the results of the prepareproject and the projection function, it combines the trajectories across the years of conditioning and the projection years, for all replicates, and plots all replicates for the years from 'startyr' to the final year of the projections. It includes the median and inner 90 plot, if the historical conditioning CPUE is included in histCE, it also includes the original cpue series. Beware of trying to plot the historical cpue in years where there are no historical cpue data, although there are error captures at work to limit the pot to available.

Usage

```
dosauplot(
  ylabel,
  postrep,
  glb,
  startyr,
  addCI = FALSE,
  CIprobs = c(0.05, 0.5, 0.95),
  histCE = NULL,
  addCE = FALSE
)
```

Arguments

`ylabel`	the variable name to be plotted, used as a y-axis label
`postrep`	the replicate values from the replicated projection zoneDP for the variable selected taken from the output of 'zonetosau'.
`glb`	the global constants object
`startyr`	the index for the first year of the conditioning dynamics to include
`addCI`	should quantile confidence bounds be included on the plots, default=FALSE
`CIprobs`	the quantiles used to generate the confidence intervals. The default = c(0.05,0.5,0.95)
`histCE`	the historical cpue data if included, default = NULL
`addCE`	should historical cpue be included on the plots, default=FALSE

Value

invisibly, a list of CI and median for each SAU

Examples

```
print("wait on suitable built in data sets")
```

`do_condition` *do_condition is a function to assist with conditioning the model*

Description

do_condition is a utility function that can be used to speed the conditioning of the operating model on available fishery data. During the generation of the simulated zone the operation that takes the longest is the estimation of the productivity of each population. The productivity is estimated at equilibrium and getting there is what takes the time. When conditioning the model the objective is to match the predicted fishery response to the historical catches to the observed responses. 'do_condition' generates the simulated zone without running the projections, which simplifies any searches for optimal values of AvRec (average unfished recruitment), and especially for suitable values of recruitment deviates. If any of the initdepl (initial depletion) values, as listed in the control file are less than 1.0 then before applying the historical catches it first depletes each population within each SAU to the initdepl value for each SAU. For this to work doproduct must be TRUE. This will slow down any searches for an optimum set of AvRec and recdevs, so a different strategy for doing that will be needed.

Usage

```
do_condition(
  rundir,
  controlfile,
  calcpopC,
  verbose = FALSE,
  doproduct = FALSE,
  dohistoric = TRUE,
  mincount = 100
)
```

Arguments

`rundir`	the full path to the directory in which all files relating to a particular run are to be held.
`controlfile`	the filename of the control file present in rundir containing information regarding the particular run.
`calcpopC`	a function that takes the output from hcrfun (either the aspirational catch x SAU or TAC x zone) and generates the actual catch per population in each SAU expected in the current year.
`verbose`	Should progress comments be printed to console, default=FALSE
`doproduct`	should production estimates be made. default=FALSE
`dohistoric`	should the historical catches be applied. Default=TRUE
`mincount`	determines the minimum sample size for a size-composition sample to be included in plots and analyses. Default = 100

Value

a large list containing tottime, projtime, starttime, glb, ctrl, zoneDD, zoneDP, projC, condC, sauout, and outzone

See Also

makeequilzone, dohistoricC, prepareprojection, doprojections

Examples

```
print("wait on suitable data sets in data")
# rundir=rundir; controlfile=controlfile;calcpopC=calcexpectpopC
# verbose=TRUE; doproduct=FALSE; dohistoric=TRUE; mincount=100
```

`do_MSE` *do_MSE an encapsulating function to hold the details of a single run*

Description

do_MSE is a form of meta-function that contains the code and function calls that constitute a single scenario run for aMSE. This is the reason the argument list is as long as it is. The code has been encapsulated like this to simplify development, maintenance, and distribution. Already with the number of variant conditioning examples produced as tests of the code-base, errors have arisen through a failure to propagate all changes to the code now in this function to all the separate MSE run files used to manage each scenario. A potential flaw lies with the need to apply the jurisdictionHS functions to any historical fishery data used to condition the operating model, ready for the first year of the projections. Currently, the code used is unique to the Tasmanian case and this obviously still requires generalization. This may require a new function to be produced in each jurisdictionHS.R file, but that is still under consideration. Part of the development of this meta-function will be to further articulate the generation of results, and the inclusion of error traps on the argument list entries. To aid in the conditioning the option of not calculating the productivity has been added. By default it will occur and when projecting it will also always occur. HSstats is a list that is always saved and contains performance statistics for the HS used (currently contains sum5 and sum10 the cumulative catches to 5 and 10 years into the projections).

Usage

```
do_MSE(
  rundir,
  controlfile,
  hsargs,
  hcrfun,
  sampleCE,
  sampleFIS,
  sampleNaS,
  getdata,
  calcpopC,
  makeouthcr,
  varyrs = 7,
  startyr = 42,
  verbose = FALSE,
  ndiagprojs = 3,
  savesauout = FALSE,
  cutcatchN = 56,
  matureL = c(70, 200),
  wtatL = c(80, 200),
  mincount = 100
)
```

Arguments

`rundir`	the full path to the directory in which all files relating to a particular run are to be held.
`controlfile`	the filename of the control file present in rundir containing information regarding the particular run.
`hsargs`	the constants used to define the workings of the hcr
`hcrfun`	the name of the harvest control rule that is used to calculate the aspirational catches by SAU or TAC by zone for the following year
`sampleCE`	a function from jurisdictionHS.R that generates the CPUE statistics from the projected data

`sampleFIS`	a function from jurisdictionHS.R that generates the FIS statistics
`sampleNaS`	a function from jurisdictionHS.R that generates the Numbers-at-size samples
`getdata`	a function that gathers all the data required by the hcrfun and combines it into an hcrdata object ready for the hcrfun
`calcpopC`	a function that takes the output from hcrfun (either the aspirational catch x SAU or TAC x zone) and generates the actual catch per population in each SAU expected in the current year.
`makeouthcr`	Another JurisdictionHS.R function that generates (or not) an object that is continually updated by the hcrfun. To avoid very inefficient code this object (at least for Tasmania) is cycled through the iterations.
`varyrs`	how many years at the end of the conditioning on the fishery, data into zoneDD, to which to add recruitment variation, default = 7, which suits the Tasmanian west coast. Used in prepareprojection
`startyr`	the index for the first year of the conditioning dynamics to include in the plotted cpue trajectories, to give startyr:indexoflastyear. used in sauplots.
`verbose`	Should progress comments be printed to console, default=FALSE
`ndiagprojs`	the number of replicate trajectories to plot in the diagnostics tab to illustrate ndiagprojs trajectories to ensure that such projections appear realistic; default=3
`savesauout`	should the sau dynamics object be saved as an sauoutD.RData file? 100 replicates of 56 populations for 58 years of conditioning and 30 years of projection = about 5.7 Mb. default=FALSE.
`cutcatchN`	to reduce the size of the final array of numbers-at-size in the catch one can remove all the empty cells below a given size class. In the default there are 105 2mm size classes and setting cutcatchN = 56 removes size classes 1:55 2 - 110 mm; only from catchN
`matureL`	is a vector of 2, default = c(70,200), that define the x-axis bounds on the biology maturity-at-Length plots
`wtatL`	is a vector of 2, default = c(80,200), that define the x-axis bounds on the biology weight-at-Length plots
`mincount`	given size-composition data what minimum sample size will be deemed acceptable for inclusion in the plots and conditioning. default=100.

Value

a large list containing tottime, projtime, starttime, glb, ctrl, zoneDD, zoneDP, projC, condC, sauout, and outzone

See Also

makeequilzone, dohistoricC, prepareprojection, doprojections, getprojyrC

Examples

```
print("wait on suitable data sets in data")
```

driftrec *driftrec adjusts the recruitment allowing for larval drift*

Description

driftrec adjusts the recruitment levels in the stock dynamics allowing for larval drift. The input drift levels can be a constant for each population, or a vector, with a value for each population if their rates are assumed to different. If a constant valuer is input it is extended to a vector of the same length as the number of populations. Each population is assumed to connect only to the populations either side of it. The populations on the end only connect into the simulated populations and so only lose half the recp value for that population, which is sent to the population it is beside in the conditioning. If the zone is not conditioned using yield or productivity from spatial data then productivity will be allocated randomly and the recp (recruitment proportion lost) may as well be set to zero.

Usage

```
driftrec(recs, recp)
```

Arguments

`recs`	the vector of recruits calculated internally to oneyear
`recp`	the recruitment proporiton lost. Either a constant (eg 0.02 for 2 percent) or a vector, a value for each population.

Value

a vector of revised recruitment levels accounting for larval drift.

Examples

```
recs <- c(243699,49747,1285137,1492742,923282,273427)
newrec <- driftrec(recs,0.04)
newrec0 <- driftrec(recs,0.0)
recs
newrec
newrec0
sum(recs)
sum(newrec)
sum(newrec0)
```

fillzoneDef *fillzoneDef characterizes simulated Zone; holds zone properties*

Description

fillzoneDef Characterizes the simulated Zone; holds the zone properties. Lists the date and time the zone was made, its size and basic biological properties: MSY, B0, Average bLML, Average SaM, and more details

Usage

```
fillzoneDef(zoneC, zoneD, prod)
```

Arguments

`zoneC`	the constnats for the zone to be characterized.
`zoneD`	the dynamic parts of the rgion being characterized
`prod`	the production object from doproduction

Value

A list of 10 objects containing the properties of the zone; has the class zoneDefinition - for S3 methods

- Production the productivity of each populations separately
- defpop each population's defining parameters
- Struct the zone's hierarchical structure
- nBlock the number of blocks
- numpop the number of populations
- SummaryPop biological properties of populations
- SummaryZone summary of the zone's properties
- BlockProp summary of the block properties
- Date date/time of fillzoneDef activation; if the function is run soon after a zone's creation this can be a proxy for the zone creation.

Examples

```
# needs summaryPop and summaryZone
txt2 <- 'always use examples rather than example'
```

`findF1` *findF1 approximates the F0.1 harvest rate for each population*

Description

findF1 generates estimates of the annual harvest rate that would be equivalent to the F0.1 for each population. It does this by calculating the gradient of the production curve for the selected pop number and selecting the best approximation by searching for the gradient that is cloest to 0.1.

Usage

```
findF1(product)
```

Arguments

`product`	the 3D array from doproduction

Value

either the index within the production array or the vector of gradients

Examples

```
## Not run:
# data files changed, no longer works
data(product) #for pop=1, a 28percent drop from Hmsy leads to a
findF1(product=product) # loss of 3 tonnes, 4 percent of MSY
findmsy(product)  # compare the AnnH, Deplet, and RelCE levels.

## End(Not run)
```

---

findlinenumber *findlinenumber prints out the contents of a text file with line numbers*

Description

findlinenumber solves the problem of finding the line number in a given text file when one wants to change a specific line using the function changeline. After inc lines are printed the function stops and waits for any character to be input (a space will suffice) before printing the next inc lines

Usage

```
findlinenumber(rundir, filename, inc = 20)
```

Arguments

rundir	the directory path in which to find the text file.
filename	the full name of the text file in quotations.
inc	the number of lines on screen

Value

nothing, but it does print the contents of a text file with the respective line numbers on the console.

Examples

```
print("wait on a suitable example")
```

findmsy — *findmsy identifies the closest productivity value to MSY*

Description

findmsy for each population in the zone, identifies the closest productivity value to MSY and returns the vector of productivity values for the selected harvest rate. The Catch in each populaiton = MSY and the other variables relate to the value at MSY

Usage

```
findmsy(product)
```

Arguments

`product`	array of productivity values output from doproduction

Value

a matrix of numpop rows containing the approximate MSY related productivity values and the index of the approx MSY

Examples

```
## Not run:   # takes far too long
  data(zone1)
  glb <- zone1$globals
  data(constants)
  ans <- makezoneC(zone1,constants)
  zoneC <- ans$zoneC
  glb <- ans$glb
  ans <- makezone(glb,zoneC)
  zoneC <- ans$zoneC
  zoneD <- ans$zoneD
  ans2 <- modzoneC(zoneC,zoneD,glb)
  zoneC <- ans2$zoneC  # zone constants
  product <- ans2$product
  approxMSY <- findmsy(product)
  print(approxMSY)

## End(Not run)
```

fishery_plots — *fishery_plots generates a set of plots relating to the fishery properties*

Description

fishery_plots produces a series of plots relating to the fishery and its properties. The selectivity curves reflect the LML through time and so this needs to be represented.

Usage

```
fishery_plots(rundir, glb, select, histyr, projLML, rge = 50:90)
```

Arguments

`rundir`	the rundir for the particular scenario
`glb`	the object contianing the global constants
`select`	the selectivity matrix from zoneCP, with all years of selectivity
`histyr`	the object from condC containing the historical LML
`projLML`	the object containing the LML in the projections
`rge`	the range of the midpts for the size classes so that the view can be limited so that the lower arm set to zero and the upper set to 1 need not all be viewed and to give greater separation to any set of LML, which will usually be close to each other.

Value

invisibly, the matrix of unique selectivities

Examples

```
print("wait on suitable internal data sets")
```

---

getaav *getaav calculates annual absolute variation in catch*

Description

getaav calculates the annual absolute change in catch for an input vector of catches, which could be across a series of years or even across different spatial units for a single year (an unusual use). The equation used is aav = 100 x sum(|Ct - Ct-1|)/(sum(Ct).

Usage

```
getaav(invect)
```

Arguments

`invect`	a vector of catches

Value

a single scalar value the AAV of the input catches

Examples

```
  catch <- c(1,2,3,4,5,4,3,2,1)
  getaav(catch)  # should equal 0.32
```

getavrec — *getavrec pulls out just the AvRec values for each sau for a scenario*

Description

getavrec extracts the AvRec values for each SAU for a given scenario. It does this by reading in the saudata file and using grep to search for the correct line and returning the line as is. Care is required to only take the first record of 'AvRec' so as not to consider the variability in sAvRec.

Usage

```
getavrec(rundir, datafile, nsau)
```

Arguments

`rundir`	the directory in which one finds the saudata file for the given scenario
`datafile`	the exact name of the saudata file
`nsau`	the number of SAU to be found

Value

a numeric vector of the average recruitment for each SAU

Examples

```
## Not run:
  rundir <- "c:/Users/User/DropBox/A_codeUse/aMSEUse/scenarios/MhLML/M1h5/"
  getavrec(rundir,"saudataM1h5.csv",8)

## End(Not run)
```

getConst — *getConst extracts 'nb' numbers from a line of text*

Description

getConst parses a line of text and extracts 'nb' pieces of text as numbers

Usage

```
getConst(inline, nb, index = 2)
```

Arguments

`inline`	text line to be parsed, usually obtained using readLines
`nb`	the number of numbers to extract
`index`	which non-empty object to begin extracting from?

Value

a vector of length 'nb'

Examples

```
  txtline <- "MaxDL , 32,32,32"
  getConst(txtline,nb=3,index=2)
```

---

getCPUEssq *getCPUEssq calculates ssq between historical and conditioned cpue*

Description

getCPUEssq takes in the historical CPUE and the conditioned zone's cpue and calculates the sum-of squared differences between them to assist with the conditioning. This is a greatly simplified version of plotcondCPUE, which plots a graph as well as estimating the ssq. For speed it is better to use getCPUEssq.

Usage

```
getCPUEssq(histCE, saucpue, glb)
```

Arguments

`histCE`	the matrix of historical cpue by SAU
`saucpue`	the predicted cpue from the conditioning on historical data
`glb`	the globals object

Value

a vector of length nsau containing the ssq for each SAU

Examples

```
print("wait on suitable internal data sets")
```

---

getLFdata *getLFdata reads the observed LF-composition data*

Description

getLFdata reads the observed LF-composition data, which needs to be stored in a csv file with the following format. Each SAU must use the same sequence of length-classes as rows, and the same sequence of years as columns, even if there are empty rows and columns. In addition, the first column must list the length-class and the second column lists the SAU. See the 'data-file-description.docx', and the example data files to see examples of the required format.

Usage

```
getLFdata(rundir, filename)
```

Arguments

`rundir`	the directory containing the LF data files
`filename`	the complete filename for the length-composition data file

Value

a list containing the lfs array (lengths x years x sau) and palfs matrix of the count of observations in each year for each sau. This is then contained in the condC object.

See Also

getzoneLF, getnas

Examples

```
print("wait on suitable internal data files")
# rundir=rundir; filename="lf_WZ90-20.csv"
```

---

getLFlogL *getLFlogL calculates the multi-nomial log-likelihood for sizecomp data*

Description

getLFlogL when comparing the observed size-composition data from commercial catches with those rpedicted during the historical fishing period one can quantify any differences using the multinomial log- likelihood. This function calculates this, and the value can then be combined with am ssq value from a comparison of the predicted and observed CPUE ready to be minimized if conditioning the OM on a fisheries real data. As always, the operating model is running at a population level while data arrives at an SAU level, so this is only an approximate approach, but it should improve over using purely the results from the sizemod R package.

Usage

```
getLFlogL(catchN, obsLFs, glb, wtsc, sau)
```

Arguments

`catchN`	the predicted numbers-at-size in the catch for all years
`obsLFs`	the observed NAS, after cleaning by the preparesizecomp function
`glb`	the globals list
`wtsc`	the weighting given to the size-composition data
`sau`	which sau is the subject of analysis

Value

the weighted ssq for the filtered observed size-composition data

Examples

```
print("wait on data sets")
```

getline — *getline extracts a vector of numbers from a txt or csv files*

Description

getline can be used to extract the AvRec and MaxCEpars values for each SAU for a given scenario. It does this by reading in the saudata file and using grep to search for the varname, which gives the first linenumber with that name. One can extract from the saudata file as well as from the control file. The idea is to use values from that line when conditioning the operating model.

Usage

```
getline(rundir, filen, varname, nobs)
```

Arguments

`rundir`	the directory in which one finds the saudata or control file for the given scenario
`filen`	the exact name of the file being examined
`varname`	what variable name is required? Take care with spelling
`nobs`	how many numbers to return, usually nSAU

Value

a numeric vector for each SAU or of length nobs

Examples

```
## Not run:
  rundir <- "c:/Users/User/DropBox/A_codeUse/aMSEUse/scenarios/MhLML/M1h5/"
  getline(rundir,"saudataM1h5.csv","AvRec",8)

## End(Not run)
```

getlistvar — *getlistvar extracts a vector or matrix from zoneC*

Description

getlistvar extracts a vector or matrix from zoneC. If a vector of scalars, the names relate to populations, if a matrix the columns relate to populations. Only Me, R0, B0, effB0, ExB0, effExB0, MSY, MSYDepl, bLML, scalece, SaM, SAU, popdef, LML, Maturity, WtL, Emergent, and MatWt are currently valid choices. The indexvar = popdef would generate a listing of all the constants. If you only want a single constant from popdefs then use indexvar2.

Usage

```
getlistvar(zoneC, indexvar, indexvar2 = "")
```

Arguments

`zoneC`	the constants components of the simulated zone
`indexvar`	the name of the variable to be extracted; character
`indexvar2`	the name of the variable within popdef to extract

Value

either a vector or matrix of values depending on variable

Examples

```
data(zone)
zoneC <- zone$zoneC
getlistvar(zoneC,"MSY")
getlistvar(zoneC,"B0")
getlistvar(zoneC,"popdef","AvRec")
```

`getLogical` *getLogical extracts nb logicals from an input line of text*

Description

getLogical obtains nb logicals from an input line

Usage

```
getLogical(inline, nb)
```

Arguments

`inline`	text line to be parsed, usually obtained using readLines
`nb`	the number of logicals to extract, if nb is longer than the number of logicals within inline the vector will contain NAs

Value

a vector of length nb

Examples

```
txtline <- "Depleted, TRUE"
aMSE:::getLogical(txtline,nb=1)
txtline2 <- "calcthis, TRUE, FALSE"
aMSE:::getLogical(txtline2,nb=2)
```

getmaxCE	*getmaxCE identifies peak CPUE during projections for each SAU*

Description

getmaxCE is one of the HS performance statistics. Because it uses the targetCE from the TasmanianHS it is obviously Tasmanian specific.

Usage

```
getmaxCE(ceCI, glb, targetCE)
```

Arguments

`ceCI`	is the sauout$outCI$cpue object which contains the CI for the replicate projections
`glb`	the globals object
`targetCE`	the target CE from the hsargs

Value

a matrix of nSAU x 5: 5p, 50p, 95p year, and difference between the target and the maximum median value, all for each SAU

Examples

```
## Not run:
  ceCI <- out$sauout$outCI$cpue
  glb <- out$glb
  targCE <- hsargs$maxtarg
  getmaxCE(ceCI=ceCI,glb=glb,targetCE=targCE)

## End(Not run)
```

getnas	*getnas gets the numbers-at-size for all populations and the zone*

Description

getnas extracts numbers-at-size for all populations, SAUs, and the zone. There will therefore be numpop + 1 columns of numbers-at-size fr the particular year given.

Usage

```
getnas(zoneD, yr, glob)
```

Arguments

`zoneD`	the dynamic portion of the zone
`yr`	which yr from the range available should be summarized
`glob`	the global variables object

Value

a matrix of numpop + 1 columns of numbers-at-size for each population and for the zone

See Also

getLFdata, getzoneLF, prepareprojection, doprojections

Examples

```
print("wait on suitable data sets")
```

getprojyrC   *getprojyrC returns cumulative catch from selected projection years*

Description

getprojyrC is used to calculate the cumulative catch taken in each SAU and across the zone for selected years of each replicate. Thus, the output would be a replicates x nSAU matrix of total catches across the first 'period' years of the projection under the given harvest strategy.

Usage

```
getprojyrC(catsau, glb, period = 10)
```

Arguments

`catsau`	the time series of catches summed across populations within each SAU. This array (reps x SAU x Allyears), is found within the 'out' object from the do_MSE function (as in out$catsau).
`glb`	the globals object. Again found in 'out', as in out$glb
`period`	how many years to cumulate; default = 10

Value

a reps x SAU matrix of summed catches

Examples

```
print("wait on suitable internal data sets")
```

getrecdevcolumn — *getrecdevcolumn extracts a column of recdevs from a control file*

Description

getrecdevcolumn is used when conditioning the operating model by modifying the recruitment deviates. getrecdevcolumn extracts a column of years selected from a selected controlfile. It does this to use those values as the starting parameter vector for optim.

Usage

```
getrecdevcolumn(rundir, filename, yearrange, sau, verbose = FALSE)
```

Arguments

`rundir`	the rundir for the scenario
`filename`	the character name of the control file
`yearrange`	the range of years of rec devs to be selected
`sau`	which SAU or block is to be fitted?
`verbose`	should details of the run be made. default=FALSE

Value

a list of the vewctor of recdevs, the vector of the yearrange, and a vector of the linenumbers within the control file that are to be changed

Examples

```
print("wait on suitable internal data-sets")
```

getsauzone — *getsauzone summarizes zoneD into SAU and zone*

Description

getsauzone rowsums the matrices of matureB, exploitB, catch, and recruit from the input zoneD into the separate SAUs and the total into the zone. The harvestR is simply the respective catch divided by the exploitB, and the cpue are the individual population cpue weighted relative to the catch taken from each population in either the SAU or the complete zone.

Usage

```
getsauzone(zoneD, glb, B0, ExB0)
```

Arguments

`zoneD`	the zoneD after nyrs of dynamics
`glb`	the globals object
`B0`	is the sau based B0
`ExB0`	is the sau based ExB0

Value

a list of six matrices of nSAU columns of SAU summaries, and one column for the zone

Examples

```
print("wait on an example")
```

---

getsingleNum *getsingleNum find a line of text and extracts a single number*

Description

getsingleNum uses grep to find an input line. If the variable being searched for fails then NULL is returned

Usage

```
getsingleNum(varname, intxt)
```

Arguments

`varname`	the name of the variable to get from intxt
`intxt`	text to be parsed, usually obtained using readLines

Value

a single number or, if no value is in the data file a NULL

Examples

```
## Not run:
 txtlines <- c("replicates, 100","Some_other_text, 52")
 getsingleNum("replicates",txtlines)
 getsingleNum("eeplicates",txtlines)
 getsingleNum("other",txtlines)

## End(Not run)
```

---

getssqparts *getssqparts calculates the ssq for the cpue and size-comp data*

Description

getssqparts is used during the conditioning of the operating model to improve the fit of the model parameters to the observed SAU scale data on cpue and size-composition. It only uses sum-of squared deviations so an arbitrary weight is given to the size-composition data, and that value should be such that changes in the ssq for the cpue are of the same order of magnitude for the size-composition data.

Usage

```
getssqparts(rundir, controlfile, calcpopC, mincount = 100, wtsc = 0.02)
```

Arguments

`rundir`	the directory in which all the files and outputs are kept
`controlfile`	the name of the controlfile for the MSE run
`calcpopC`	the function from the harvest strategy that is used to calculate the expected catch by population within each say each year. The annual catch by sau is known but in the MSE that needs to be distributed, in this case without error, across each population.
`mincount`	the minimum count used in the size-composition data selection default = 100, this can be a vector of length nsau with a value for each sau if you want to give a different value to each sau.
`wtsc`	the weighting given to the size-composition data ssq, default= 0.02. This can be a vector of length nsau if you want to give each sau a different weighting.

Value

a vector of the total ssq, the cpue ssq, and the size-comp ssq

Examples

```
print("wait on internal data sets")
```

---

getStr *getStr obtains a string from an input text line*

Description

getStr obtains a string from an input text line in which any parts are separated by ','. Then, after ignoring the first component, assumed to be a label, it returns the first nb parts.

Usage

```
getStr(inline, nb)
```

Arguments

`inline`	input text line with components separated by ','
`nb`	number of parts to return

Value

a vector of character string(s)

Examples

```
  txt <- "runlabel, development_run, label for this particular run"
  getStr(txt,1)
```

`getsum` *getsum sums each of the main dynamics within zoneD*

Description

getsum is only used by getsauzone to sum each of the main dynamic variables within zoneD, and hence is not exported.

Usage

```
getsum(inmat, index)
```

Arguments

`inmat`	what matrix within zoneD to sum into SAU and zone
`index`	a vector containing an index of populations within SAU

Value

an nSAU+1 column matrix summarizing each SAU and the zone

Examples

```
print("wait on an example")
```

`gettasdevssq` *gettasdevssq calculates the SSQ between observed and predicted CPUE*

Description

gettasdevssq is used when conditioning the aMSE operating model using ad hoc recruitment deviates after conditioning on AvRec. It focuses on a single SAU at a time. log transformed recdevs are used to avoid the possibility of negative recdevs. NOte this does not test for equilibrium of the initial operating model, it just assumes it reaches equilibrium.

Usage

```
gettasdevssq(
  param,
  rundir,
  ctrlfile,
  calcpopC,
  locyrs,
  sau,
  obsLFs = obsLFs,
  wtsc = 0.1,
  verbose = FALSE,
  outplot = FALSE,
  console = FALSE,
  full = FALSE
)
```

Arguments

`param`	the log transformed sequence of recdevs for the selected years
`rundir`	the rundir for the scenario
`ctrlfile`	the character name of the control file
`calcpopC`	the function used to distribute catches among populations. This is from the external JurisdictionHS.R file that is 'sourced' in.
`locyrs`	the rows within the matrix of recruitment deviates corresponding to the selected years.
`sau`	which sau (in TAS 1 - 8) is to be worked on
`obsLFs`	the observed length-composition of the catches for the sau, usually from condC$compdat$lfs[,,sau]
`wtsc`	what weighting should the total Multi-Nomial ikelihood be multiplied by to scale to the range of change in thw CPUE ssq? Default=0.1
`verbose`	should console reports be made? default = FALSE
`outplot`	should a plot be generated for output to the webpage. default=FALSE
`console`	should the plot go to the console. If FALSE and outplot is TRUE then a plot will go to rundir. If TRUE theplot will go to the console. DEfault=FALSE
`full`	should the ssq components be returned together or separately. default=FALSE

Value

a scalar value which is the SSQ for the selected years of CPUE

Examples

```
print("wait on suitable internal data sets")
```

`getunFished` *getunFished - extracts all data relating to year 1; unfished.*

Description

getunFished - extracts all data relating to year 1; unfished. Of course, this will only work as long as movezoneYear hasn't been called, which would disrupt the contents of the first year. But if applied to zone rather than zone1 etc, this should be fine. The outputs include: R0, B0, B0crypt, popdef, MSY, MSYDepl, LML, ExB0, and Nt

Usage

```
getunFished(zoneC, zoneD, glb)
```

Arguments

`zoneC`	the constant part of the zone
`zoneD`	the dynamic part of the zone
`glb`	contains the global variables used everywhere

Value

a list of multiple components relating to the unfished stock this includes a list of each of the equilibrium populations and the zone as the last element of the list

Examples

```
## Not run:
  data(zone) # would normally use zone <- makeequilzone(rundir,"control.csv")
  unfish <- getunFished(zone$zoneC,zone$zoneD,zone$glb)
  str(unfish,max.level=1)

## End(Not run)
```

getvar — *getvar a replacement for sapply to obtain scalar constants*

Description

getvar is a replacement for sapply to obtain scalar constants from zoneC and is significantly faster. It should be used to obtain things like B0, R0, MSY, scalece, etc. Still need to use sapply to pull out vectors.

Usage

```
getvar(zoneC, invar)
```

Arguments

`zoneC`	the constants object for the zone
`invar`	a character variable eg. "B0" or "R0"

Value

a numpop vector of the invar constants from zoneC

See Also

getlistvar

Examples

```
data(zone)
zoneC <- zone$zoneC
getvar(zoneC,"MSY")
getvar(zoneC,"B0")
```

getvect *getvect extracts invar from the popdef vector in zoneC*

Description

getvect extracts a numpop vector of invar from the popdef vector in zoneC. Still need to use sapply to pull out complete vectors such as popdef or maturity etc.

Usage

```
getvect(zoneC, invar)
```

Arguments

zoneC	the constants object for the zone
invar	a character variable eg. "steeph", "DLMax"

Value

a numpop vector of invar from the numpop popdefs in zoneC

See Also

getlistvar

Examples

```
data(zone)
zoneC <- zone$zoneC
getvect(zoneC,"steeph")
```

getzoneLF *getzoneLF extracts all LF data from a zone across pops and years*

Description

getzoneLF extracts all LF data from a zone across all populations for each year. Thus an Nclass x Nyrs X numpop matrix is compressed into a Nclass x Nyrs matrix

Usage

```
getzoneLF(zoneD, glb)
```

Arguments

zoneD	the dynamic part of the zone
glb	the global constants

Value

an Nclass x Nyrs matrix containing LF data across all populations by year

Examples

```
print("An example needs to be written")
```

---

getzoneprod *getzoneprod zone scale summary of product matrix from doproduction*

Description

getzoneprod takes in the product matrix from doproduction and sums the mature and exploitable biomassand the total catches. Then it recalculates, for the zone, the approximate annual harvest rate, the depletion level, and the relative catch rate. The annual harvest rate and relative catch rate are simply the arithmetic mean of the values for all populations, which the depletion is the column of mature biomass divided by the first value = B0

Usage

```
getzoneprod(product)
```

Arguments

`product`	The productivity array from doproduction containing the range of imposed harvest rates, and the resulting outputs for each population

Value

a matrix containing the approximate productivity matrix for the zone

Examples

```
data(zone)
zoneprod <- getzoneprod(zone$product)
head(zoneprod,20)
```

---

getzoneprops *getzoneprops extracts the depletion level for a given year*

Description

getzoneprops extracts a set of properties for each population and summarizes them for the zone as well. These properties include effB0, matureB, legalmatB, propprot, MSY, effexB0, exploitB, SpBDepl, ExBDepl, legalDepl, MSYDepl, LML, and bLML. See the model documentation for the meaning of each of these.

Usage

```
getzoneprops(zoneC, zoneD, glb, year = 1)
```

Arguments

`zoneC`	the constants for the zone being simulated
`zoneD`	the dynamic part of the zone being simulated
`glb`	the global constants
`year`	which years information is wanted, default=1

Value

a matrix of population properties with a column of totals

Examples

```
data(zone)
str(zone,max.level=1)
# round(getzoneprops(zone$zoneC,zone$zoneD,zone$glb),4)
# zoneC=zoneC; zoneD=zoneD;glb=glb;year=1
```

`historicalplots` *historicalplots a wrapper function to call plots of fishery plots*

Description

historicalplots is a wrapper used by do_MSE and do_condition, to plot out details of the historical fishery data used in the conditioning. It currently plots out the historical catches and CPUE.

Usage

```
historicalplots(rundir, condC, glb, commonscale = FALSE, proportion = FALSE)
```

Arguments

`rundir`	the directory where all results are stored
`condC`	the R object contaiining the historical fishery data generated by the readctrlfile function
`glb`	the global variables object
`commonscale`	should the SAU plots share a common y-axis scale, default = FALSE
`proportion`	should the plot be of proportion relative to the maximum catch through each time-series or as raw tonnes x year, default=FALSE

Value

nothing but it does add plot files to the rundir

Examples

```
print("wait on suitable data sets")
```

imperr *imperr calculates population catches from sau catches with error*

Description

imperr converts aspirational sau catches into population level catches while introducing management implementation error. Here this error is implemented as Log-Normal errors on diver intuitions concerning the relative abundance in each population. The error is imposed separately on the populations in each SAU.

Usage

```
imperr(catchsau, exb, sauindex, sigmab = 1e-08)
```

Arguments

`catchsau`	the predicted or aspirational catch per SAU from the Harvest control rule
`exb`	the exploitable biomass at the end of the previous year. In the first year of the projections this would be the last year of the conditioning.
`sauindex`	the SAU index for each population
`sigmab`	the Log-Normal standard deviation of implementation error. The default value = 1e-08, which effectively means no errors.

Value

a vector of population catches for the year to be imposed after the estimation of exploitable biomass

Examples

```
print("wait on suitable internal data sets")
```

lf10 *lf10 contains three years of length-composition data for block 10*

Description

lf10 is a data.frame of length-composition of commercial catch from block 10 on the west coast of Tasmania in a longdat format as derived from the commlf function makelongdat.This makes it useful for illustrating the use of makewidedat.

Format

A data.frame of abalone size-composition data

year the year of sampling

sau the block or SAU

length the measured size in mm, with 2mm size classes

counts the counts at each length class

propcounts the proportion of each length class of the total

Subjects

- selectivity
- size-based stock assessment model fitting
- size-distributions of commercial catches

Source

Thanks to the Institute of Marine and Antarctic Science, which is part of the University of Tasmania, and especially to Dr Craig Mundy, leader of the Abalone Group, for permission to use this data.

Examples

```
data(lf10)
head(lf10,20)
mids <- seq(138,210,2)
answer <- makewidedat(lf10,mids)
answer[1:20,]
```

`logistic` *logistic a Logistic selectivity function*

Description

logistic a logistic selectivity function. This uses the logistic function 1/(1+exp(-log(19.0)*(lens-inL50)/(delta))), where delta = inL95 - inL50. This explicitly defines the SM50 but uses delta (which is SM95-SM50) as the second. This ensures that when adding variation to parameters, to vary between populations, when SM95 and SM50 are close together it is not possible for SM50 to become larger than SM95. Be careful using the knifeedge option. Strictly knifeedge selectivity would entail the selectivity values being zero up to the knife-edge and then being 1.0. This is not what happens here. Instead the knifeedge option literally sets all values to zero at and below the value of knifeedge but leaves any curve above that value as it is. Hence this is not strict knife-edge selectivity. However, it does provide a selectivity curve that reflects the selectivity of a diver led fleet workingon abalone.

Usage

```
logistic(inL50, delta, lens, knifeedge = 0)
```

Arguments

`inL50`	is the length at 50 percent selection
`delta`	is the difference between the 95 percent selection and the 50 percent selection
`lens`	a vector of lengths for which the logistic maturity value will be calculated
`knifeedge`	defaults to 0. If knifeedge is set to a particular length then the selectivity less than the value of knifeedge is set to zero.

Value

A vector of length(lens) containing the predicted selectivity at length values

Examples

```
## Not run:
inL50 <- 100.0
delta <- 8.0
lens <- seq(2,210,2)
select <- logistic(inL50,delta,lens)
select <- logistic(inL50,delta,lens,knifeedge=105)

## End(Not run)
```

---

makeabpop *makeabpop generates full population structure in an unfished state*

Description

makeabpop generates the full population structure in an unfished state. The structure is pre-determined: Me R0 B0 effB0 ExB0, effExB0 MSY MSYDepl bLML scalece SaM popdef (vector of constants) LML G Maturity WtL Emergent Select SelWt MatWt SAUname. See the AbMSE documentation to see the full definition of a zone, made up of a zoneC and a zoneD. Notice the presence of effB0 and effExB0, these relate to the influence of larval dispersal on each populations productivity. The effective B0 relates to the unfished mature biomass after larval dispersal occurs and the population achieves equilibrium.

Usage

```
makeabpop(popparam, midpts, projLML)
```

Arguments

`popparam`	the vector of biological parameters values that define the specific properties of this population. Obtained from popdefs, which is produced by definepops
`midpts`	the center values of the size classes dervied from the zone data file
`projLML`	the LML expected in the projection years 2 - Nyrs; a vector obtained from the zonedatafile

Value

a list of numpop lists of 19 objects as detailed above.

Examples

```
print("See the code for makezoneC to see usage of makeabpop")
```

makeequilzone *makeequilzone high level function generates equilibrium zone*

Description

makeequilzone is a high level function that merely hides the details of generating the original unfished zone after reading in the data files, estimates the productivity, and sets up the results directory, rundir, ready to receive files.

Usage

```
makeequilzone(
  rundir,
  ctrlfile = "control.csv",
  doproduct = TRUE,
  verbose = TRUE
)
```

Arguments

`rundir`	the directory containing the control csv files. It can/will also act to store results in a manner that will allow them to be displayed using makehtml.
`ctrlfile`	the main file that controls the particular run. It contains the name of the data file that is used to biologically condition the numpop populations
`doproduct`	boolean, should the productivity calculations be made during the conditioning. Set to FALSE conditionOM
`verbose`	Should progress comments be printed to console, default=TRUE

Value

a list of zoneC, zoneD, glb, constants, saudat,product, ctrl, and zone1

Examples

```
print("wait on datafiles")
```

makemove *makemove produces the movement matrix*

Description

makemove produces the movement matrix, which assumes that only adjacent populations contribute to each other. Thus, a central population would lose half its larvae to one side and half to the other side, but would receive half of their larval dispersal each. As larval dispersal is modelled as a proportion those populations that produce more larvae will lose more in absolute terms. Those populations at the edges of the zone only lose half the larval dispersal into the zone and it is assumed that what they lose out of the zone will be matched by what will come into the zone.

Usage

```
makemove(npop, recs, ld, sigmove = 0)
```

Arguments

npop	the number of populations
recs	the original unfished recruitment levels estimated for each population
ld	the larval dispersal rate for all populations
sigmove	currently no variation is assumed to occur. This is here in case it needs to get implemented, but as there is little or no data on actual larval dispersal levels currently this is set to zero and not used.

Value

a npop x npop matrix describing movement among populations

Examples

```
recruits <- c(636146,263878,819189,1112513,285025,671573.9)
makemove(npop=6,recs=recruits,ld=0.04)
```

---

makeoutput *makeoutput take the output from do_MSE and generates the HTML files*

Description

makeoutput simplifies taking the output from do_MSE and producing the HTML files used to display all the results in rundir. Note that 'runnotes' is now a vector of character strings made up using paste0.

Usage

```
makeoutput(
  out,
  rundir,
  postdir,
  controlfile,
  hsfile = NULL,
  doproject = TRUE,
  openfile = TRUE,
  verbose = FALSE
)
```

Arguments

`out`	the output from do_MSE
`rundir`	the full path to the directory holding the all scenario files
`postdir`	the name of the directory holding the results, also used to name the internal webpage
`controlfile`	the controlfile used to run the MSE
`hsfile`	the name of the harvest strategy file, default=NULL
`doproject`	have the projections been run? default = TRUE.
`openfile`	should the website be opened automatically? default=TRUE
`verbose`	should details of producing the HTML files be written to the console? default=FALSE

Value

nothing but it does generate a set of HTML files in rundir. If verbose=TRUE it also writes text to the console

Examples

```
print("wait on internal data-sets")
```

`makewidedat` *makewidedat converts long data to a wide data format*

Description

makewidedat takes the output of the commlf function makelongdat and converts the columns of year x length x count into a matrix of counts with axes of lengths x years, ready for comparisons with predicted length composition of catches from the operating model.

Usage

```
makewidedat(inlong, mids, counts = FALSE)
```

Arguments

`inlong`	the long form data.frame derived for commlf's makelongdat containing at least 'year', 'length', and 'propcounts'
`mids`	the range of lengths to be found within the input data file. These are matched with the size distribution for each year of data and the appropriate matrix cells filled with the propcounts.
`counts`	default = FALSE, which means that the wide format will contain the proportion of counts for each year. If TRUE, then the actual counts by size-class will be output by year

Value

a matrix of proportional counts for each size-class x year

Examples

```
data(lf10)
mids <- seq(138,210,2)
answer <- makewidedat(lf10,mids)
answer[1:20,]
answerC <- makewidedat(lf10,mids,counts=TRUE)
answerC[1:20,]
```

---

makezone *makezone generates the dynamic parts of the simulated zone*

Description

makezone generates the dynamics components of the simulated zone and completes the constant components of the simulated zone. The term 'zone' refers to the upper level of geographical detail that is used. Thus, for example, in Tasmania we might simulate a number of statistical blocks with multiple populations. In combination, we refer to the total as a zone. The matrices are Nyrs x numpop and the two arrays are N x Nyrs x numpop.

Usage

```
makezone(glob, zoneC)
```

Arguments

glob	the global constants defined for the current simulation. These include numpop, nblock, midptsd, Nclass, and Nyrs
zoneC	the zoneC object from makezoneC

Value

a list of the dynamics and the constant components of the simulation

Examples

```
## Not run:
  data(zone1)
  glb <- zone1$globals
  data(constants)
  ans <- makezoneC(zone1,constants)
  zoneC <- ans$zoneC
  glb <- ans$glb
  ans2 <- makezone(glb,zoneC)
  str(ans2,max.level=2)

## End(Not run)
```

makezoneC *makezoneC makes the constant part of the simulation*

Description

makezoneC makes the constant part of the simulated zone. Once defined this does not change throughout the simulation. Once made it still requires makezone to be run to fill in the B0, ExBo, MSY, MSYDepl, and the scalece values, and to produce zoneD, the dynamic part of the new zone

Usage

```
makezoneC(zone, const)
```

Arguments

zone	the object derived from the readzonefile function
const	the object derived from the readdatafile function

Value

a list containing the constant part of the simulated zone

Examples

```
## Not run:
data(zone1)
data(constants)
ans <- makezoneC(zone=zone1,const=constants)
zoneC <- ans$zoneC
popdefs <- ans$popdefs
str(zoneC,max.level=1)
str(zoneC[[1]])  # not complete at this stage
print(popdefs)

## End(Not run)
```

makezoneDP *makezoneDP generates the container for the projection dynamics*

Description

makezoneDP generates an object designed to hold the outputs from each replicate within a set of projections. This is identical to zoneD except it contains a repeat for each iteration and now includes a cesau, which is the population-catch-weighted cpue for each SAU.

Usage

```
makezoneDP(allyr, iter, glb)
```

Arguments

`allyr`	number of years of historical catches plus projection years
`iter`	the number of replicates
`glb`	the object containing the global variables

Value

a large list containing an object ready for the projection dynamics

Examples

```
print("Could add variation to the harvest rates so that when ")
print("prepareprojection was run the range of initial H values would increase ")
```

`maturity` *maturity Logistic maturity curve*

Description

maturity this uses the logistic function: exp(a+bL)/(1+exp(a+bL)), which has the property that the SM50 = -a/b and the interquartile distance is 2.Ln(3)/b.

Usage

```
maturity(ina, inb, lens)
```

Arguments

`ina`	is the intercept of the exponential function
`inb`	is the gradient of the exponential function
`lens`	a vector of lengths for which the logistic maturity value will be calculated

Value

A vector of length(lens) containing the predicted maturity at length values

Examples

```
## Not run:
a <- -14.383
b <- 0.146017
lens <- seq(2,210,2)
Maturity <- maturity(a,b,lens)

## End(Not run)
```

midg *midg is an abalone tagging data-set from the Actaeons*

Description

midg is a tagging data-set for blacklip abalone (*Haliotis rubra*) from the middle ground in the Actaeons in Tasmania's Block 13. All individuals were recaptured in 2003, the site number was 478, at Latitude -43.54 longitude 146.99, and there are 347 observations. All Dt = 1 year.

Format

A data.frame of abalone tagging data

RecapL the length at recapture

Lt the length at tagging

Dt The time interval between tagging and recapture, in this instance they are all listed as 1 year

DL the growth increment in mm

Subjects

- growth curves
- inverse logistic, von Bertalanffy, Gompertz
- Static model fitting

Source

Thanks to the Institute of Marine and Antarctic Science, which is part of the University of Tasmania, and especially to Dr Craig Mundy, leader of the Abalone Group, for permission to use this data collected in 2003.

Examples

```
data(midg)
head(midg,20)
oldpar <- par(no.readonly=TRUE)
plot(midg$Lt,midg$DL,type="p",pch=16,cex=1.0,xlim=c(5,180))
abline(h=0,col=1)
par(oldpar)
```

modprojC *modprojC produces three objects used to condition the zone*

Description

modprojC produces an object used to condition the zone when projecting it following any conditioning. Once the initial conditions for the projection have been attained through an initial depletion of an unfished equilibrium, or more particularly by conditioning on historical data then the projC object will be used to condition the projections. In essence, this puts the selectivity and selectivity x weight-at-size into projC. It can handle changes in LML through the years of projection.

Usage

```
modprojC(zoneC, glob, projC)
```

Arguments

`zoneC`	the constant part of the zone
`glob`	the globals for the simulation
`projC`	the project definition object from zone1 from readctrlfile

Value

an updated projC with projSel and projSelWt completed

Examples

```
print("wait on new example data")
```

`modzoneC` *modzoneC runs the zone 3 x hyrs to equilibrium*

Description

modzoneC runs the doproduction function so it can then appropriately fill in each population's MSY and MSYDepl values. It outputs both the finalized zoneC and the prodution array. Hence we need to push copies of both zoneC and zoneD into modzoneC

Usage

```
modzoneC(zoneC, zoneD, glob, lowlim = 0, uplim = 0.4, inc = 0.005)
```

Arguments

`zoneC`	the constant components of the simulated zone
`zoneD`	the dynamic components of the simulated zone
`glob`	the general global variables
`lowlim`	the lower limit of harvest rate applied, default=0.0
`uplim`	the upper limit of harvest rate applied, default=0.4
`inc`	the harvest rate increment at each step, default=0.005

Value

a list containing the updated zoneC and product

Examples

```
## Not run:   # the doproduction part takes too long to run
  data(zone1)
  glb <- zone1$globals
  data(constants)
  ans <- makezoneC(zone1,constants)
  zoneC <- ans$zoneC
  ans <- makezone(glb,zoneC)
  zoneC <- ans$zoneC
  zoneD <- ans$zoneD
  ans2 <- modzoneC(zoneC,zoneD,glb)
  str(ans2,max.level=2)

## End(Not run)   # zoneC=zoneC; zoneD=zoneD; glob=glb; lowlim=0.0;uplim=0.4;inc=0.01
```

modzoneCSel *modzoneCSel changes the selectivity characteristics in zoneC*

Description

modzoneCSel changes the selectivity characteristics in zoneC which is necessary when making a projection under a different LML. This function assumes a constant LML during any projections.

Usage

```
modzoneCSel(zoneC, sel, selwt, glb)
```

Arguments

zoneC	the constant zone object from setupzone
sel	the new selectivity as a matrix of selectivity by population
selwt	new selectivity x WtL as a matrix of SelWt x Population
glb	the globals object

Value

the zoneC object modified ready for projection

Examples

```
print("wait on suitable data")
```

numbersatsize *numbersatsize plots details of the numbers-at-size*

Description

numbersatsize plots up the initial unfished numbers- at-size distribution, omitting the first four size classes to avoid the recruitment numbers dominating the plot.

Usage

```
numbersatsize(rundir, glb, zoneD, ssc = 5)
```

Arguments

rundir	the results directory, if set to "" then plot is sent to the console instead
glb	the globals list
zoneD	the dynamic part of the zone, zoneD
ssc	index for starting size class. thus 1 = 2, 2 = 4, 5 = 10, etc. default = 5 for it plots size classes from 10mm up

Value

nothing but it does add one plot to the results directory

Examples

```
print("this will be quite long when I get to it")
```

onesau *onesau plots the dynamics for a single SAU*

Description

onesau plots the details of matureB, exploitB, catch, acatch, harvestR, cpue, recruit, deplsB, depleB for a single SAU on one plot

Usage

```
onesau(
  prerep,
  postrep,
  glb,
  startyr,
  picksau,
  addCI = FALSE,
  CIprobs = c(0.05, 0.5, 0.95),
  histCE = NULL
)
```

Arguments

`prerep`	the zoneDsau object that represents the SAU data
`postrep`	the zonePsau object that represents the SAU data
`glb`	the global constants object
`startyr`	the year from which to begin the conditioned dynamics
`picksau`	which sau should be plotted
`addCI`	should the 90 percent CI be added, default=FALSE
`CIprobs`	what CI should be fitted, default=c(0.05,0.5,0.95)
`histCE`	should the historical CPUE be added to the cpue plot

Value

a list of the CI for each variable

Examples

```
print("Wait on suitable data sets")
```

`oneyear` *oneyear one year's harvest dynamics for one abpop*

Description

oneyear do one year's dynamics for one input population. Used to step through populations of a zone. Its dynamics are such that it first undergoes growth, then half natural mortality. This allows an estimate of exploitable biomass before fishing occurs. The remaining dynamics involve the removal of the catch, the application of the last half of natural mortality and the addition of recruits. Which allows the exploitable biomass to be estimated after fishing. The recruitment occurs in oneyearD so that larval dispersal can be accounted for. Thus, oneyear requires oneyearD.

Usage

```
oneyear(MatWt, SelWt, selyr, Me, G, qest, WtL, inNt, inH, lambda, scalece)
```

Arguments

`MatWt`	maturity x Weight-at-size from zoneCC for the population and year
`SelWt`	selectivity x Weight-at-size from zoneCC for the population and year
`selyr`	selectivity from zoneC for the population and year
`Me`	natural morality from zoneC for the population
`G`	growth transtion matrix from zoneC for the population
`qest`	the catchability from zoneCC for the population
`WtL`	the weight-at-alength from zoneC for the population
`inNt`	the numbers at size for the year previous to the year of dynamics. These are projected into the active year.
`inH`	a literal annual harvest rate as a proportion to be removed as catch during the year, a scalar
`lambda`	the non-linearity parameter. if = 1.0 then cpue linearly related to expB, if < 1 then hyper-stability of cpue occurs.
`scalece`	the xoneC'pop'$scalece used to scale al cpue to the sau nominla scale

Value

a list containing ExploitB, MatureB, Catch, Harvest, Nt, ce, CatchN, and midyexpB used to update the given pop in yr + 1

Examples

```
print("need to wait on built in data sets")
#oneyear <- function(MatWt,SelWt,selyr,Me,G,qest,WtL,inNt,inH,lambda,scalece)
```

`oneyearcat` *oneyearcat one year's catch dynamics for one abpop*

Description

oneyear does one year's dynamics for one input population. Where the fishing is based on a given catch per population, which would be determined by any harvest control rule. It is used to step through populations of a zone. Its dynamics are such that each population first undergoes growth, then half natural mortality. This allows an estimate of exploitable biomass before fishing occurs. The remaining dynamics involve the removal of the catch, after estimating the harvest rate from catch/exploitb, the application of the last half of natural mortality and the addition of recruits. This allows the exploitable biomass to be estimated after fishing. The cpue uses the average of the pre-fishing and post-fishing exploitB to smooth over any changes brought about by fishing. The recruitment occurs in oneyearC so that larval dispersal can be accounted for. oneyearcat is not used independently of oneyearC.

Usage

```
oneyearcat(
  MatWt,
  SelWt,
  selyr,
  Me,
  G,
  scalece,
  WtL,
  inNt,
  incat,
  sigce,
  lambda,
  qest
)
```

Arguments

`MatWt`	maturity x Weight-at-size from zoneCC for the population and year
`SelWt`	selectivity x Weight-at-size from zoneCC for the population and year
`selyr`	selectivity from zoneCC for the population and year
`Me`	natural morality from zoneCC for the population
`G`	growth transtion matrix from zoneCC for the population

`scalece`	the xoneC'pop'$scalece used to scale al cpue to the sau nominla scale
`WtL`	the weight-at-alength from zoneCC for the population
`inNt`	the numbers at size for the year previous to the year of dynamics. These are projected into the active year.
`incat`	the literal annual catch from the population. Derived from the harvest control rule for each SAU, then allocated to each population with respect to its relative catching performance.
`sigce`	the process error variation associated with the estimates of cpue from the model. Found in the ctrl object as withsigCE. It is implemented as Log-Normal error on the exploitable biomass value from the model
`lambda`	the hyper-stability term from zoneC
`qest`	the estimated catchability from sizemod from zoneC

Value

a list containing a vector of ExploitB, MatureB, Catch, and ce, and a matrix NaL containing Nt and CatchN used to update a pop in yr + 1

See Also

dohistoricC, oneyearcat, oneyearrec

Examples

```
print("need to wait on built in data sets")
```

`oneyearD` *oneyearD conducts one year's dynamics on zoneD in the MSE*

Description

oneyearD conducts one year's dynamics on zoneD in the MSE returning the revised zoneD, which will have had a single year of activity included in each of its components. This uses zoneC but always within the environment of another function in which zoneC (as zoneC) can be found. Used in runthreeH, (and hence dodepletion and doproduction) and in testequil.

Usage

```
oneyearD(zoneC, zoneD, inHt, year, sigmar, Ncl, npop, movem)
```

Arguments

`zoneC`	the constant portion of the zone with a list of properties for each population
`zoneD`	the dynamics portion of the zone, with matrices and arrays for the dynamic variables of the dynamics of the operating model
`inHt`	a vector of harvest rates taken in the year from each population
`year`	the year of the dynamics, would start in year 2 as year 1 is the year of initiation.
`sigmar`	the variation in recruitment dynamics, set to 1e-08 when searching for equilibria.
`Ncl`	the number of size classes used to describe size
`npop`	the number of populations, the global numpop
`movem`	the larval dispersal movement matrix

Value

a list containing a revised dynamics list

Examples

```
print("wait on revised data")
```

`oneyearrec` *oneyearrec calculates the Beverton-Holt recruitment*

Description

oneyearrec calculates the Beverton-Holt recruitment for the input populations in a single year; parameterized with steepness, R0, and B0. To drop variation to insignificant levels set sigmar-1-08. To allow for inclusion of recruitment deviates, which can be required to explain high catches when stock biomass would otherwise be low, we have introduced the devR argument. As the deviates are all multiplicative and range in value > 0, then the default = -1 so this will be ignored unless a deviate is input.

Usage

```
oneyearrec(steep, R0, B0, Bsp, sigR, devR = -1)
```

Arguments

`steep`	the steepness for the population; scalar
`R0`	the unfished recruitment levels for the population; scalar
`B0`	the unfished spawning biomass; scalar
`Bsp`	the current spawning biomass; scalar
`sigR`	standard deviation of the recruitment residuals; scalar. set this to 1e-08 to avoid recruitment variability
`devR`	if recruitment deviates are available then they should be input here. If negative values then a random epsilon is used, otherwise epsilon is given the value of devR. default=-1, so by default fixed recruitment deviates are off.

Value

an absolute number of recruits from a given spawning biomass

See Also

oneyearsauC, oneyearcat, dohistoricC

Examples

```
## Not run:
steep <- 0.707
R0 <- 319971
B0 <- 313
Bsp <- 147
insigmar <- 0.3
oneyearrec(steep,R0,B0,Bsp,insigmar)

## End(Not run)
# steep=steep;R0=r0;B0=b0;Bsp=dyn["matureb",];sigR=sigmar;devR=rdev
```

---

oneyearsauC *oneyearsauC conducts one year's dynamics using catch not harvest*

Description

oneyearsauC conducts one year's dynamics in the simulation using historical SAU catches rather than harvest rates. The harvest rates are estimated after first estimating the exploitable biomass. returning the revised zoneD, which will have had a single year of activity included in each of its components. uses the function imperr to introduce implementation error to the actual catches.

Usage

```
oneyearsauC(
  zoneCC,
  inN,
  popC,
  year,
  Ncl,
  sauindex,
  movem,
  sigmar = 1e-08,
  sigce = 1e-08,
  r0,
  b0,
  exb0,
  rdev = -1
)
```

Arguments

`zoneCC`	the constant portion of the zone with a list of properties for each population ##param exb the vector of exploitable biomass for each population for the ## previous year
`inN`	the numbers-at-length for each population for the previous year
`popC`	a vector of actual catches to be taken in the year from each SAU
`year`	the year of the dynamics the exb and inN would be from year-1
`Ncl`	the number of size classes used to describe size, global Nclass
`sauindex`	the sau index for each population from glb

`movem`	the larval dispersal movement matrix, global move
`sigmar`	the variation in recruitment dynamics, set to 1e-08 when searching for an equilibrium.
`sigce`	the process error on the relationship between cpue and exploitable biomass, set to 1e-08 when searching for an equilibrium.
`r0`	the unfished R0 level used by oneyearrec
`b0`	the unfished mature biomass B0, used in recruitment and depletion
`exb0`	the unfished exploitable biomass used in depletion
`rdev`	the recruitment deviates for each SAU for the given year

Value

a list containing a revised dynamics list

See Also

dohistoricC, oneyearcat, oneyearrec

Examples

```
print("Wait on new data")
```

`oneyrgrowth` *oneyrgrowth one years' growth for a population and initial size*

Description

oneyrgrowth one years' growth for a given population and initial size. Used to determine the size after two year's growth as a measure of the expected bLML - biological Legal Minimum Length. A reflection of the two year rule in Tasmania. To get this the function should obviously be run twice.

Usage

```
oneyrgrowth(inpop, startsize = 2)
```

Arguments

`inpop`	the abpop to be grown forward
`startsize`	default = 2, but often set to the size at 50 percent maturity or the vector of midpts to determine the growth of Nt

Value

the expected mean size after one year's growth

Examples

```
## Not run:
nblock <- 2
filename <- datafiletemplate(numblock=nblock,filename="block3.csv")
condDat <- readdatafile(filename)
const <- condDat$constants
blkpop <- condDat$blkpop
numpop <- condDat$numpop
popdefs <- definepops(nblock,blockI,const)
pop1 <- makeabpop(popdefs[1,],condDat$midpts,
                  condDat$ProjLML[popdefs[1,20]])
oneyrgrowth(pop1,startsize=70)

## End(Not run)
```

`onezoneplot` *onezoneplot plots out one variable from the zone*

Description

once the projected zone has been summarized by poptozone one can add plots to the results using plotZone. This in turn uses onezoneplot to plot each variable in turn.

Usage

```
onezoneplot(invar, rundir, glb, CIprobs, varname, startyr, addfile = TRUE)
```

Arguments

`invar`	which zone variable to plot
`rundir`	the run directory for the results
`glb`	the global constants object
`CIprobs`	the quantiles to use for the CI.
`varname`	the name of the variable for use in labels
`startyr`	in what year number should the plot begin.
`addfile`	should a png file be added to the results, default = TRUE

Value

it generates a png file of the plot if addfile remains TRUE

Examples

```
print("wait on suitable data")
```

optimizeAvRec — *optimizeAvRec improves OM fit to CPUE by adjustnig AvRec*

Description

optimizeAvRec is used to improve the fit to the time-series of CPUE by adjusting the SAU initial value for AvRec. If one assesses an SAU using the sizemod package, that summarizes across a whole SAU. After population allocation within aMSE, the productivity of an SAU may well have been disturbed away (up or down) from the assessed optimum. Hence, it becomes necessary, still, to optimize both the AvRec and the recdevs during the conditioning.

Usage

```
optimizeAvRec(
  rundir,
  controlfile,
  datafile,
  calcpopC,
  snames,
  lowmult = 0.7,
  highmult = 1.3,
  linenumber = 29,
  verbose = TRUE
)
```

Arguments

`rundir`	the rundir for the scenario
`controlfile`	the name of the control file
`datafile`	the name of the datafile from the controlfile
`calcpopC`	the funciton, from the HS file that allocates catches across each of the populations within each SAU
`snames`	the names given to each SAU
`lowmult`	the multiplier to place a lower bound when search for the optimum AvRec for each SAU. default = 0.7, ie param AvRec * 0.7
`highmult`	the multiplier to place an upper bound when search for the optimum AvRec for each SAU. default = 1.3, ie param AvRec * 1.3
`linenumber`	the linenumber inside the datafile that is to be changed to the optimum AvRec values. Default = 29, whic suits Tasmania.
`verbose`	should updates on progress be sent to the console. default=TRUE

Value

a vector of the final optimized AvRec values. This also modifies the datafile - SO BE CAREFUL.

Examples

```
print("wait on suitable data files")
```

optimizerecdevs *optimizerecdevs improves the fit to the recdevs for a given sau*

Description

optimizerecdevs an important part of optimizing the match between the dynamics observed in the fishery and those of the operating model is to adjust the annual recruitment deviates so that they improve the fit to the observed CPUE and observed numbers-at-size. optimizerecdevs attempts to do that.

Usage

```
optimizerecdevs(
  rundir,
  sau,
  controlfile,
  calcpopC,
  wtsc = 0.02,
  maxiter = 100,
  yearrange = 1980:2016,
  verbose = FALSE,
  mincount = 100,
  plottoconsole = FALSE,
  optimmethod = "Nelder-Mead"
)
```

Arguments

`rundir`	the rundir for the scenario
`sau`	which sau to apply this to
`controlfile`	the name of the control file
`calcpopC`	the funciton, from the HS file that allocates catches across each of the populations within each SAU
`wtsc`	what weight to give to the size-composition data,default=0.1
`maxiter`	The maximum number of iterations to run before stopping. Default = 100, one needs at least 400 to see a real difference
`yearrange`	which recdev years should be 'conditioned'. Default= 1980:2016
`verbose`	text output to console, default=FALSE
`mincount`	the minimum number of iterations in the solver, default=100
`plottoconsole`	should the final plot be sent to the console = TRUE of the rundir = FALSE. default=FALSE
`optimmethod`	which optim method to use? default='Nelder-Mead'

Value

a scalar value which is the total SSQ for the selected years for the CPUE and sizecomps. It also alters the recdevs in the controlfile!

Examples

```
print("wait on data sets")
```

`plotCNt` *plotCNt plots the historical conditioning Nt or catchN for all years*

Description

plotCNt the historical conditioning on the fisheries data leads to a three dimensional array of numbers-at-size x years x numpop. We use 'prepareDDNt' to convert this to a 3D array by SAU. PlotCNt then plots, for each SAU, numbers-at-size for all years of conditioning with the first year emphasized in black and the last year in red.

Usage

```
plotCNt(Nt, glb, vline = NULL, start = 3)
```

Arguments

`Nt`	the 3D array of numbers-at-size x years x SAU from prepareDDNt
`glb`	the global constants object
`vline`	add a vertical dashed line, perhaps at the LML, default=NULL
`start`	from what size class should the maximum Y value be measured. The default = 3 (= 6mm)

Value

nothing but it does plot a graph

Examples

```
 print("wait on suitable built in data-sets")
```

`plotcondCPUE` *plotcondCPUE plots the historical cpue against conditioned cpue*

Description

plotcondCPUE generates a plot of the historical cpue and compares it with the predicted cpue from the conditioning. The predicted is black and the observed is green. This is primarily there are an aid to conditioning the operating model. It also calculates the simple sum of squared differences between the observed and predicted, again to aid in the conditioning.

Usage

```
plotcondCPUE(histCE, saucpue, glb, rundir, filen = "")
```

Arguments

`histCE`	the matrix of historical cpue by SAU
`saucpue`	the predicted cpue from the conditioning on historical data
`glb`	the globals object
`rundir`	the rundir for the given scenario
`filen`	the file name of the plot if it is to be saved

Value

a vector of length nsau containing the ssq for each SAU

Examples

```
print("wait on suitable internal data sets")
```

`plotconditioning` *plotconditioning illustrate the zones dynamics after conditioning*

Description

plotconditioning converts the conditioned dynamic zone object from population scale to sau scale and then plots the various components to depict their state after conditioning. This is designed to place such plots into the rundir to assist in the conditioning process.

Usage

```
plotconditioning(zoneDD, glb, zoneC, histCE, rundir, recdevs)
```

Arguments

`zoneDD`	the conditioned dynamic zone object
`glb`	the global constants object
`zoneC`	the constant zone object
`histCE`	the historical CPUE matrix
`rundir`	the rundir for the given scenario
`recdevs`	the recruitment deviates from the control file, condC

Value

the sau scale zone dynamics, invisibly

Examples

```
print("wait on suitable data sets")
```

`plothistcatch` *plothistcatch generates a plot of the historical catches by SAU*

Description

plothistcatch generates a plot of histircal catches by SAU and by Zone. It can plot can tonnes x year or proportion. All SAU plots can have independent or common y-axis scales.

Usage

```
plothistcatch(
  yrs,
  histC,
  nsau,
  saunames,
  commonscale = FALSE,
  proportion = FALSE,
  filen = ""
)
```

Arguments

`yrs`	a vector of the years of catch history
`histC`	the matrix of catches by years
`nsau`	the number of SAU
`saunames`	the SAU names as a vector
`commonscale`	should the SAU plots share a common y-axis scale, default = FALSE
`proportion`	should the plot be of proportion relative to the maximum catch through each time-series or as raw tonnes x year, default=FALSE
`filen`	the name of the file used to save the plot, default="", which sends the plot to a separate window

Value

nothing but it does generate a plot either to a window or a file

Examples

```
print("wait on suitable data sets")
```

plothistce *plothistce generates a plot of the historical cpue by SAU*

Description

plothistce generates a plot of historical cpue by SAU. It can plot as cpue or proportion of maximum cpue.

Usage

```
plothistce(rundir, condC, glb, proportion = FALSE, filen = "")
```

Arguments

`rundir`	the direcotry where all results are stored
`condC`	the R object contiaining the historical fishery data generated by the readctrlfile function
`glb`	the globol variables object
`proportion`	should the plot be of proportion relative to the maximum catch through each time-series or as raw tonnes x year, default=FALSE
`filen`	the name of the file used to save the plot, default="", which sends the plot to a separate window

Value

nothing but it does generate a plot either to a window or a file

Examples

```
print("wait on suitable data sets")
```

plothsstats *plothsstats plots some HS performance statistics*

Description

plothstats generates plots for the sum of catches for the first 5 and 10 years of the projections. This is really only interesting when compared with the same statistics from a different operating model definition or different harvest strategy. The input data is generated during the running of do_MSE.

Usage

```
plothsstats(rundir, hsstats, glb)
```

Arguments

`rundir`	the scenario's results directory
`hsstats`	the statistics about the harvest strategies performance
`glb`	the global constants object

Value

nothing but it does generate two plots into rundir

Examples

```
print("wait on suitable in ternal data sets")
```

---

plotNt *plotNt plots the size-composition for each SAU*

Description

plotNt accepts size-composition data (either the population Nt, of the catchN), and plots the replicate size-distributions for each SAU. It then puts the first year's size-distribution on top and if the medcol that defines the colour of hte mdeian of the replicates is different from 0 (not plotted) then it plots the median as well. A contrasting colour choice is 4 = Blue.

Usage

```
plotNt(Nt, year, glb, start = 3, medcol = 0)
```

Arguments

`Nt`	the 4 dimensional array of numbers-at-size Nclass,years,nsau,reps.
`year`	which year from dimension 2 should be plotted?
`glb`	the global constants object
`start`	from what size class should the maximum Y value be measured. The default = 3 (= 6mm)
`medcol`	colour of the median line; default=0 = not plotted

Value

invisibly the matrix of median values, Nclass x nsau

Examples

```
print("wait on suitable built in data-sets")
```

plotpopprops *plotpopprops plots out some properties of input populations*

Description

once the depleted zone (zoneCP and zoneDD) has been generated the component populations have their properties summarized in propD. This is saved as propertyDD.csv and output to the zoneDD tab. To provide a visual consideration of these properties this function generated histograms for an array of input array column names.

Usage

```
plotpopprops(x, rundir, glb, varnames, startyr, console = TRUE, bins = 25)
```

Arguments

`x`	the array containing columns to plot
`rundir`	the run directory for the results, can be set to "" if plotting to the console
`glb`	the global constants object
`varnames`	the names of each variable to select the columns and to use in labels
`startyr`	in what year number should the plot refer to?
`console`	should plot go to the console or a png be saved? Default=TRUE
`bins`	the number of breaks to plot in each histogram, default=25

Value

it generates a png file of the plot if console = FALSE, otherwise it is drawn to the console

Examples

```
print("wait on suitable data")
# x=propD;rundir=rundir;glb=glb;varnames=columns;startyr=hyrs;bins=21;console=FALSE
```

plotprod *plotprod graphs a selected productivity variable*

Description

plotprod graphs a selected productivity variable from the choice of ExB exploitable biomass, MatB mature of spawning biomass = Bmsy, AnnH the actual annual harvest rate Hmsy, Catch the yield at Bmsy and Hmsy = MSY, Deplet the mature biomass depletion level, and RelCE the relative cpue at MSY

Usage

```
plotprod(
  product,
  xname = "MatB",
  yname = "Catch",
  xlimit = NA,
  xlab = "Mature Biomass t",
  ylab = "Production t",
  font = 7,
  filename = "",
  devoff = FALSE
)
```

Arguments

`product`	the output from doproduction
`xname`	the name of the productivity variable for the x-axis, defaults to MatB
`yname`	the name of the y-xis variable, default=Catch=Yield
`xlimit`	default=NA, enables the range of the x-axis to be constrained, for example using xlimit=c(0.15,0.4)
`xlab`	the x-axis label, default=""
`ylab`	the y-axis label, default=""
`font`	the type of font used in the plot. default = 7, bold Times, 6 is not bold, 1 is sans-serif, 2 bold sans-serif
`filename`	the complete path and filename of where to save the png plot file. default is empty, meaning no file is produced.
`devoff`	a boolean to allow the plot device to remain open for the user to add more components. Of course, if this is set to FALSE then it is up to the user to use dev.off

Value

invisibly a list of the x and y matrices plotted

Examples

```
data(zone)
product <- zone$product
plotprod(product)
stat <- findmsy(product)
abline(h=stat[,"Catch"],col=1:6,lwd=2)
abline(v=stat[,"MatB"],col=1:6,lwd=2)
print(stat)
```

`plotproductivity` *plotproductivity characterizes each population's yield curve*

Description

plotproductivity characterizes each population's yield curve, it also describes the total productivity of the zone.

Usage

```
plotproductivity(rundir, product, glb)
```

Arguments

`rundir`	the results directory
`product`	the productivity 3-D array
`glb`	the globals list

Value

nothing but it does place five png files into rundir

Examples

```
print("this will be quite long when I get to it")
```

`plotscene` *plotscene literally plots up the output from comparevar*

Description

plotscene takes the output from comparevar and plots the quantiles relative to each other.

Usage

```
plotscene(
  scenquant,
  glbc,
  var = "cpue",
  ymin = 0,
  filen = "",
  legloc = "topleft",
  legplot = 1
)
```

Arguments

scenquant	a list of the quantiles for the given variable from each scenario
glbc	a list of the global objects from each scenario being compared
var	what variable from the dynamics to summarize, valid names include catch, acatch, cpue, harvestR, desplsB, depleB, recruit, matureB, and exploitB, default = 'cpue'
ymin	allows one to set the lower limit to the yaxis, default = 0
filen	a filename for use if saving the output to said file, default = ", which implies the plot goes to the console.
legloc	legend location, default='topleft'
legplot	which plot should contain the legend, default=1

Value

nothing but it does generate a plot with nsau panels

Examples

```
print("wait on internal datasets")
```

---

plotsizecomp *plotsizecomp generates a plot of available size composition data*

Description

plotsizecomp generates a plot of available size composition data with the option of plotting the predicted size-composition of catch on top of it; this latter option is only possible when plotting the distributions as proportions.

Usage

```
plotsizecomp(
  rundir,
  incomp,
  SAU = "",
  lml = NULL,
  catchN = NULL,
  start = NA,
  proportion = TRUE,
  console = TRUE,
  width = 10,
  height = 9,
  fnt = 7,
  tabcategory = "predictedcatchN"
)
```

Arguments

`rundir`	the directory for all ctrl, data, and output files.
`incomp`	filtered size-composition data from the fishery
`SAU`	the name of the SAU used to label the y-axis
`lml`	a vector of the lml in each year represented by incomp
`catchN`	the predicted size-composition of the catch
`start`	which size-class to start from, default=NA, which means all size-classes in the samples will be used.
`proportion`	should the plots be as proportions or counts,default=TRUE
`console`	should the plot be sent ot the console or saved to rundir. default=TRUE
`width`	the width of the plot, default = 10
`height`	the height of the plot, default = 9
`fnt`	what font to use, default font = 7, bold times
`tabcategory`	what name to give to the webage tab when the plots are saved. Default="predictedcatchN"

Value

nothing but it does generate a plot

See Also

preparesizecomp, popNAStosau

Examples

```
print("wait on data sets")
```

`plotZone` *plotZone plots the projected TAC across all replicates*

Description

plotZone plots out the projected TAC for all replicates. The first value is the sum of the final catches in the conditioning period.

Usage

```
plotZone(
  inzone,
  rundir,
  glb,
  startyr,
  CIprobs = c(0.05, 0.5, 0.95),
  addfile = TRUE
)
```

Arguments

`inzone`	the dynamic zone projection object
`rundir`	the rundir for the scenario
`glb`	the global constants object
`startyr`	which year number to begin the plot
`CIprobs`	the quantiles used to generate the confidence intervals. The default = c(0.05,0.5,0.95)
`addfile`	should the plots be added to the rundir =TRUE or only sent to the console = FALSE. Default = TRUE

Value

nothing but it adds a plot to the zonescale tab in the rundir

Examples

```
print("wait on suitable datasets")
# inzone=outzone;rundir=rundir;glb=glb;CIprobs=c(0.05,0.5,0.95);addfile=TRUE; startyr=40
```

`plotzonesau` *plotzonesau generates a plot of the zone scale and sau scale*

Description

plotsonesau enables the zone catch or catch weighted zone cpue (from catchweightCE) to be compared with the sau catch or the sau cpue within a single plot. This has value for gaining insights into the fishery dynamics of a zone and how it responds to fishing pressure. NOTE WELL: Currently, this function is setup for Tasmania's set of 8 SAU. Assuming other jurisdictions have more or less then the layout of the plots would need to be altered. Until these are known this function will likely only work as intended with Tasmanian data!

Usage

```
plotzonesau(
  zonetot,
  saudat,
  saunames,
  label,
  labelsau,
  side = 3,
  sauscale = TRUE
)
```

Arguments

`zonetot`	a vector of either the total catch or zone summary cpue for a zone. Ideally, the vector should be named using the year in which each value was taken.
`saudat`	either the catch or the cpue for each SAU (year x sau), again this matrix should have both row and column names

`saunames`	a vector of nSAU names. Ideally very short ones to ensure they fit on each small plot
`label`	the y-axis label for the zone plot
`labelsau`	the y-axis label for each row of the SAU plots
`side`	should the SAU label be at the top of bottom of each plot. default = 3 (= top of plot). The main alternative would be 1 (the bottom) but 2 (left side) and 4 (right side) also work.
`sauscale`	should all SAU plots have the own y-axis scale = TRUE, the default. If you want each plot to have the same scale sauscale=FALSE.

Value

nothing but this does generate a plot.

See Also

catchweightCE

Examples

```
print("wait on suitable internal dats sets")
```

---

popNAStosau *popNAStosau converts NAS data by population to NAS x sau*

Description

popNAStosau converts numbers-at-size data x population into the same by SAU. This is designed to compare ny observed NAS data from a zone with those predicted by the conditioned model, it could also be used to improve the conditioning code to avoid fitting only to CPUE

Usage

```
popNAStosau(popNAS, glb)
```

Arguments

`popNAS`	either the catchN or the Nt from zoneDD, that is the zone after the equilibrium zone (zoneD) has had the historical catches applied.
`glb`	the globals object. Needed for the sauindex and various array dimnesion labels.

Value

An Nclass x nyrs x nsau array of numbers-at-size data

See Also

preparesizecomp, plotsizecomp

Examples

```
#e.g.  popNAStosau(out$zoneDD$catchN,out$glb)
```

`poptosau` *poptosau converts projected population dynamics to SAU scale results*

Description

poptosau the MSE dynamics are run at the population level but all management decisions are made at the SAU level, hence the results of the projections need to be translated into results at the SAU level. poptosau uses the sauindex to sum the variables that can be summed, which include matureB, exploitB, catch, and recruit, and combines their population results to form their respective SAU results. Thus, for an input array of dimensions [projyrs,numpop,reps] one receives back an array of [projyrs, nSAU,reps]

Usage

```
poptosau(invar, glb)
```

Arguments

`invar`	either zoneDP$ matureB, exploitB, catch, or recruit
`glb`	the global constants object

Value

a results array of dimension [projyrs, nSAU,reps]

Examples

```
print("wait on appropriate internal datasets")
```

`poptosauCE` *poptosauCE combines population cpue into sau as catch weighted sums*

Description

poptosauCE combines cpue from separate populations into their respective sau using a catch-weighted strategy. The sauindex is used to identify which populations to apply the sau total catches to.

Usage

```
poptosauCE(catvect, cpuevect, sauindex)
```

Arguments

`catvect`	the vector of catches x population for a given year
`cpuevect`	the vector of cpue x population for a given year
`sauindex`	the sau indices of each population

Value

a list of saucpue and saucatch

Examples

```
print("wait on appropriate built-in data files")
# catvect=zoneDD$catch[1:finalyr,]; cpuevect=zoneDD$cpue[1:finalyr,]
```

`poptozone` *poptozone translates the zone_pop objects to a single zone object*

Description

poptozone combines the dynamic results for each variable so that results by population become results by zone. matureB, exploitB, catch, recruit, catchN, and Nt are simple summations of the totals for each population into their respective Zone The harvest rate would be end of year or beginning of year estimates derived from dividing the catch x zone by the exploitable biomass x zone. Similarly the deplsB and depleB are the end of year matureB and exploitB divided by their respective unfished estimated by Zone obtained using getvar(zoneC,"B0").

Usage

```
poptozone(inzone, NAS = NULL, glb, B0, ExB0)
```

Arguments

`inzone`	one of the zone dynamics objects containing replicates, made up of populations
`NAS`	the numbers-at-size 4D arrays from doprojection; default=NULL so it can be ignored during conditioning
`glb`	the object containing the global constants
`B0`	the sum of B0 across all populations, use getvar(zoneC,"B0")
`ExB0`	the sum of ExB0 across all populations use getvar(zoneC,"ExB0")

Value

a list of dynamics variables by zone

Examples

```
print("wait on suitable internal data sets ")
```

`prepareDDNt`	*prepareDDNt converts the population based historical Nt to SAU-based*

Description

prepareDDNt processes the zoneDD, following the conditioning on the historical fishery data, so that the population numbers-at-size are converted to SAU-based numbers-at-size. It does this for both Nt and catchN. This is to allow analysis, tabulation, and plotting at the SAU scale. Input data is 3D Nclass x years x numpop

Usage

```
prepareDDNt(inNt, incatchN, glb)
```

Arguments

`inNt`	the 3 dimensional array of population numbers-at-size
`incatchN`	the 3 dimensional array of catch numbers-at-size
`glb`	the global constants object

Value

a list of Nt and catchN x SAU

Examples

```
print("wait on suitable internal data-sets")
```

`prepareprojection`	*prepareprojection high level function that sets up a projection*

Description

prepareprojection is a high level function that restructures zoneC by including the projected selectivity (in case the LML is set to change during the projections), and also selectivity x weight-at-size (for computational speed). It then generates a new zoneD with room for all replicates as well as the aspirational catches from each HS. It does this by converting the arrays of year x pop, to year x pop x replicate. It then uses the conditioned data in zoneDep to predict the first aspirational catches for the projections and conducts the initial replicate, thus starting the application of the HS. Finally, it adds recruitment variation to each of the replicates and keeps the last year of each iteration of the addrecvar function as the start of each replicate projection, with zeros in the catch, cpue, and cesau arrays

Usage

```
prepareprojection(
  projC = projC,
  condC = condC,
  zoneC = zoneC,
  glb = glb,
  zoneDD = zoneDD,
  ctrl = ctrl,
  varyrs = varyrs,
  calcpopC = calcpopC,
  lastsigR = 0.3
)
```

Arguments

projC	the projection object from readctrlfile
condC	historical conditioning data
zoneC	the constant part of the zone
glb	the global variables
zoneDD	the zone after initial depletion through conditioning on the fishery
ctrl	the ctrl object for the scenario run
varyrs	how many years at the end to add recruitment variation
calcpopC	a function that takes the output from hcrfun and gernerates the actual catch per population expected in the current year.
lastsigR	recruitment variation for when it is applied for varyrs

Value

a list of the dynamic zone object as a list of arrays of projyrs x populations x replicates, plus the revised projC and revised zoneC

Examples

```
print("wait on data files")
```

---

preparesizecomp *preparesizecomp strips out empty columns and identifies samples*

Description

preparesizecomp takes in the size composition of catch data and removes empty columns and those with totals of less than the defined mincount. The minimum used depends on sampling routines. In Tasmania, a sample of 100 is the usual minimum taken from individual landings. Single landings may not be representative of much.

Usage

```
preparesizecomp(sizecomp, mincount = 100)
```

Arguments

`sizecomp`	the size-composition of catch data from the dat object made by readLBMdata.
`mincount`	the minimum number of observations within a year for inclusion in the analysis, default = 100.

Value

a cleaned version of the sizecomp matrix

Examples

```
print("wait on data sets")
```

`print.zone` *print.zone S3 method for printing a summary of a given zone*

Description

print.zone S3 method for printing a summary of a given zone

Usage

```
## S3 method for class 'zone'
print(x, ...)
```

Arguments

`x`	a zone object
`...`	in case there are extra parameters

Value

nothing, but does print out a summary of the zone

Examples

```
## Not run:
txt1 <- 'all example code should be able to run'

## End(Not run)
```

print.zoneDefinition *print.zoneDefinition S3 method for printing zonedef summary*

Description

print.zoneDefinition S3 method for printing a summary of the zoneDef.

Usage

```
## S3 method for class 'zoneDefinition'
print(x, ...)
```

Arguments

x	a zonedefinition object
...	in case there are extra parameters

Value

nothing, but does print out a summary of the zeonDef

Examples

```
## Not run:
txt1 <- 'all example code should be able to run'

## End(Not run)
```

printline *printline literally prints a selected line from a text file*

Description

printline is used to check that one has selected the correct line for modification inside a datafile. This is done when conditioning the MSE operating model on the parameter AvRec

Usage

```
printline(rundir, datafile, linenumber = 29)
```

Arguments

rundir	the scenario directory holding all files for a scenario
datafile	the name of the datafile for the scenario, from the ctrl object
linenumber	which linenumber to print, default =29

Value

nothing but it does print a line to the console

Examples

```
print("wait on suitable example")
```

---

putNA *putNA can add NAs to the start and end of a vector*

Description

putNA fulfils a common requirement to expand a vector with NAs to assist a plot by ensuring the x and y vectors are the same length. It can accept both character and numeric vectors. This used to be called addNA, but there is a function in base R with that name that does something completely different.

Usage

```
putNA(x, pre, post)
```

Arguments

`x`	the vector to which NAs are to be added
`pre`	how many NA to be added the front
`post`	how many to be added to the end

Value

a vector of length(x) + pre + post

Examples

```
vect <- rnorm(10,mean=5,sd=1)
putNA(vect,3,5)
```

---

readctrlfile *readctrlfile checks rundir contains the required csv files*

Description

readctrlfile checks rundir contains the required csv files including the named control file, which then contains the names of the region data file, and the population data file. The run stops if any are not present or are misnamed.

Usage

```
readctrlfile(rundir, infile = "control.csv", verbose = TRUE)
```

Arguments

`rundir`	the directory in which all files relating to a particular run are to be held.
`infile`	default="control.csv", the filename of the control file present in rundir containing information regarding the run.
`verbose`	Should progress comments be printed to console, default=TRUE

Value

the control list for the run

Examples

```
## Not run:
# this has sinoce been modified and needs updating
rundir <- tempdir()
ctrlzonetemplate(rundir)
datafiletemplate(6,rundir,filename="zone1sau2pop6.csv")
ctrl <- readctrlfile(rundir)
ctrl

## End(Not run)
```

`readdatafile` *readdatafile reads in a matrix of data defining each population*

Description

readdatafile expects a matrix of probability density function definitions that are used to define the populations used in the simulation. These constitute the definition of popdefs.

Usage

```
readdatafile(numpop, indir, infile)
```

Arguments

`numpop`	the total number of populations across the zone
`indir`	directory in which to find the date file
`infile`	character string with filename of the data file

Value

a matrix of values defining the PDFs used to define the properties of each population. The contents of popdefs

Examples

```
## Not run:
data(zone1)
glb <- zone1$globals
glb
data(constants)
constants
ctrlfile <- "control.csv"
ctrl <- readctrlfile(glb$numpop,rundir,ctrlfile)
reg1 <- readzonefile(rundir,ctrl$zonefile)
popdefs <- readdatafile(reg1$globals,rundir,ctrl$datafile)
print(popdefs)

## End(Not run)
```

`readsaudatafile` *readsaudatafile generates the constants matrix from sau data*

Description

readsaudatafile uses data described at the SAU level to make the constants file, which is then used to generate the popdefs matrix containing the specific values. This change from explicitly defining each population has been implemented to simplify the conditioning of each operating model.

Usage

```
readsaudatafile(rundir, infile, optpar = NULL)
```

Arguments

`rundir`	the directory in which the data file is to be found. This will usually be the rundir for the scenario run
`infile`	the name of the specific datafile used.
`optpar`	the optimum parameters from sizemod used to replace inputs for the main parameters estimated.

Value

a list of the constants matrix with values for each population and the original matrix of sau values from readsaudatafile

Examples

```
print("wait on suitable data sets")
# rundir=rundir; infile=ctrl$datafile;optpar=opar
```

`replaceVar` *replaceVar replaces values of a variable in the input datafile*

Description

replaceVar replaces the values of a variable in the input datafile. TAKE CARE, it overwrites the original file! However, it also saves the original file by adding an '_old' to the filename. Obviously if the function is used multiple times the original 'original' file will be over-written on the second use. Alternatively, one could readin the datafile using readdateFile and then directly alter the condDat, e.g., in the case of a four block zone in which the AvRec vale is changed: condDat$constants["AvRec",] <- c(12.5,12.2,12.4,12.1)

Usage

```
replaceVar(infile, invar, newval)
```

Arguments

`infile`	the name, and path, of the data file to be altered
`invar`	the text name of the variable to be changed
`newval`	the new value with which to replace the current values

Value

The function over-writes the original file but saves the original by adding an '_old' to the end of the original filename.

Examples

```
## Not run:
filename <- datafiletemplate(numblock=1,filename="oneblock.csv")
replace(filename,"AvRec",15.75)
condDat <- readdatafile(filename)
print(round(condDat$constants,4))

## End(Not run)
```

`resetexB0` *resetexB0 resets the unfished exploitable biomass at time zero*

Description

resetexB0 sets the unfished exploitable biomass at time zero to the average of the exploitable biomass levels before and after any fishing (when there are zero catches). Hence growth and natural mortality are included.

Usage

```
resetexB0(zoneC, zoneD)
```

Arguments

`zoneC`	the constant components of the simulated zone
`zoneD`	the dynamic components of the simulated zone

Value

a refreshed zoneC with updated ExB0 values and zoneD$depleB=1

Examples

```
print("wait on data")
```

resetLML — *resetLML sets the LML for a given zone from a given year*

Description

resetLML sets the LML for a given zone from a given year out to the final year. It changes the selectivity, the SelWt values, and the LML value per block.

Usage

```
resetLML(inzone, inLML, inyear, glob)
```

Arguments

`inzone`	a zone object
`inLML`	a vector of LML for each block to be imposed from inyear on.
`inyear`	the start year for the new LML
`glob`	the global variables object

Value

A zone object with the LML changed from inyear on

Examples

```
## Not run:
txt1 <- 'all example code should be able to run'

## End(Not run)
```

restart — *restart transfers final year values of zoneD into the first year*

Description

restart transfers the final year values from the dynamics part of the zone (zoneD), into the first year. This is used, for example, when searching for an equilibrium state if there is larval dispersal > 0.0. Of if one sets the initial depletion to anything other than 1.0. Contains the option of setting every other cell to zero, which is the default.

Usage

```
restart(oldzoneD, hyrs, npop, N, zero = TRUE)
```

Arguments

`oldzoneD`	the old zoneD containing the dynamics as run for Nyrs.
`hyrs`	The number of years of dynamics, the hyrs from glb
`npop`	The number of populations, the global numpop
`N`	the number of size classes, the global Nclass
`zero`	should the arrays be otherwise filled with zeros? The default = TRUE

Value

a list containing a revised dynamics list

Examples

```
print("wait for built in data sets")
```

`rewritecontrolfile` *rewritecontrolfile generates a revised control file*

Description

rewritecontrolfile is used to generate a new control file after the option of reading in parameters from sizemod has been used. The rewritten file contains the revised parameter values.

Usage

```
rewritecontrolfile(indir, zone1, controlfile)
```

Arguments

`indir`	the directory into which the new file should be written. This would usually be the same as rundir, the scenario directory
`zone1`	a large object inside the output object from makeequilzone that contains SAUnames, SAUpop, minc, cw, larvdisp, randomseed, initLML, condC, projC, globals, ctrl, catches, and projyrs. Obviously there is some redundency.
`controlfile`	name of the original control file

Value

nothing but it does write a file called 'oldcontrolfilename_new.csv' into rundir

Examples

```
print("wait on internal data sets")
```

`rewritedatafile` *rewrwitedatafile generates a revised saudatafile*

Description

rewritedatafile is used to generate a new saudata file after the option of reading in parameters from sizemod has been used. The rewritten file contains the revised parameter values.

Usage

```
rewritedatafile(indir, glb, zone1, saudat)
```

Arguments

`indir`	the directory into which the new file should be written. This would usually be the same as rundir, the scenario directory
`glb`	the globals object
`zone1`	a large object inside the output object from makeequilzone that contains SAUnames, SAUpop, minc, cw, larvdisp, randomseed, initLML, condC, projC, globals, ctrl, catches, and projyrs. Obviously there is some redundency.
`saudat`	the main contents of the saudata file

Value

nothing but it does write a file called 'oldsaudatafile_new.csv' into the indir

Examples

```
print("wait on internal data sets")
```

`runthreeH` *runthreeH conducts the dynamics with constant catch 3 times*

Description

runthreeH is used when searching numerically for an equilibrium and it conducts the hyrs dynamics three times, each time through it replaces year 1 with year hyrs. Thus if hyrs is 40 it conducts 3 * 39 years of dynamics (117 years). This is not exported. It uses zoneC but always it does this inside the environment of another function where zoneC can be found Used inside dodepletion and doproduction. maxiter may need to be increased when we introduce a larger movement rate between populations for greenlip, or if the number of conditioning years are fewer than 45.

Usage

```
runthreeH(zoneC, zoneD, inHarv, glob, maxiter = 2)
```

Arguments

`zoneC`	the constants components of the simulated zone
`zoneD`	the dynamics portion of the zone, with matrices and arrays for the dynamic variables of the dynamics of the operating model
`inHarv`	a vector, length numpop, of annual harvest rates to be held constant across all years.
`glob`	the globals variable from readzonefile
`maxiter`	default=3; the number of runs through the equilibrium loop.

Value

a list containing a revised dynamics list, zoneD

See Also

dodepletion, doproduction

Examples

```
print("wait on built in data sets")
# zoneC=zoneC; zoneD=zoneD; glob=glob; inHarv=rep(initH[aH],numpop); maxiter=2
```

`sauavrecssq` *sauavrecssq applies the AvRec and returns the ssq from the cpue*

Description

sauavrecssq applies the AvRec vector while adjusting just one for the picksau, and then compares the predicted CPUE with the observed and returns a sum-of-squared differences. This is used by optim while using the Brent option to find an optimum for a single parameter.

Usage

```
sauavrecssq(
  param,
  rundir,
  controlfile,
  datafile,
  linenum,
  calcpopC,
  extra,
  picksau,
  nsau,
  outplot = FALSE
)
```

Arguments

`param`	is the AvRec for the selected SAU
`rundir`	the rundir for the scenario being considered
`controlfile`	the name of the control file, no path
`datafile`	the name of the data file, no path
`linenum`	the linenumber containing the AvRec vector
`calcpopC`	the HS function that calculates the aspirational catches for each SAU
`extra`	the vector of SAU AvRecs minus the selected SAU value
`picksau`	which SAU is to be worked on as in 1:nsau
`nsau`	the total number of SAU
`outplot`	shoudla plot be generated for output to the webpage. default=FALSE

Value

the SSQ for the selected SAU in a comparison of predicted and observed CPUE

Examples

```
print("Wait on suitable internal data files")
#  param=param;rundir=rundir;controlfile=controlfile;datafile=datafile;linenum=29,
#  calcpopC=calcexpectpopC;extra=extra;picksau=sau;nsau=nsau
```

`sauplots` *sauplots generates the dosauplots for the dynamic variables*

Description

sauplots generates the dosauplots for the dynamic variables including cpue, catch, acatch (aspirational catch), matureB, exploitB, recruit, and harvestR

Usage

```
sauplots(
  zoneDP,
  NAS,
  glb,
  rundir,
  B0,
  ExB0,
  startyr,
  addCI = TRUE,
  histCE = NULL,
  tabcat = "projSAU"
)
```

Arguments

`zoneDP`	the dynamic object produced by the projections
`NAS`	the numbers-at-size 4D arrays from doprojection
`glb`	the object containing the global variables
`rundir`	the results directory
`B0`	the B0 values by population use getvar(zoneC,"B0")
`ExB0`	the ExB0 values by population use getvar(zoneC,"ExB0")
`startyr`	the index for the first year of the conditioning dynamics to include in the trajectory, to give startyr:indexoflastyear,eg startyr=40
`addCI`	should confidence intervals be added to envelope plots, default=TRUE
`histCE`	historical CPUE data used in CPUE plots, default=NULL
`tabcat`	the name of the results website tab for the plots

Value

a list of lists of CI for each SAU and variable as well as the zoneDsau and zonePsau

Examples

```
print("wait on suitable internal data-sets")
```

saurecdevs *saurecdevs plots the recruitment deviates for each SAU*

Description

saurecdevs enables a visual comparison of the 'fitted' recruitment deviates for each SAU.

Usage

```
saurecdevs(recdevs, glb, rundir, filen = "")
```

Arguments

`recdevs`	the matrix of recdevs (out$condC$recdevs)
`glb`	the globals object
`rundir`	the scenario directory for results storage
`filen`	the filename (no path needed) to be used in the rundir

Value

nothing, but it does generate a plot and save a png file if filen is populated

Examples

```
print("wait on suitable data sets")
```

sautopop *sautopop translates a vector of SAU properties into population properties*

Description

sautopop uses the sauindex object to distribute a set of SAU properties (for example recruitment deviates) into a set of population properties.

Usage

```
sautopop(x, sauindex)
```

Arguments

`x`	the vector of length nsau, of properties
`sauindex`	the index of which populations are in which SAU

Value

a vector of length numpop with each population having the property of the respective SAU

Examples

```
sauprop <- c(1,2,3,4,5)  # 5 SAU each with 3 populations
sauind <- c(1,1,1,2,2,2,3,3,3,4,4,4,5,5,5) # the sauindex
sautopop(sauprop,sauind)
```

`saveobject` *saveobject is used to save RData files of particular objects*

Description

saveobject is used to save RData files of particular objects. Currently, after projecting 56 populations for 100 replicates the final 'out' object is over 500Mb, and zoneDP, the projection object, makes up 430Mb of that. Save that if you wish, but if, say, one wanted only to save the outzone object (which summarizes outcomes at the zone level), that is only 22kb. This function facilitates the saving process.

Usage

```
saveobject(obname, object, postfix = "", rundir)
```

Arguments

`obname`	the name of the object to be saved, as a character string
`object`	the parent object from which the object$obname is to extracted
`postfix`	any postfix addition you want for the name default=""
`rundir`	the run directory for the scenario

Value

nothing but it does save a file to the rundir

Examples

```
print("wait on tempdir use")
# obname="outzone"; postfix="test"; object=out; rundir=rundir
```

`save_hsargs` *save_hsargs sends a copy of the hs arguments to rundir*

Description

save_hsargs saves a copy of the HS arguments (held in hsargs) to the rundir directory ready to be printed as text into the HSperf tab of the output webpage. They are stored in a file called hsargs.txt. The function expects hsargs to be a list, which it prints component by component to a txt file.

Usage

```
save_hsargs(rundir, hsargs)
```

Arguments

`rundir`	the data and results direcotry for the scenario
`hsargs`	the harvest strategy arguments object.

Value

it saves a text file into the rundir sub-directory and modifies the resultTable.csv file for the web-page summary of scenario results

Examples

```
print("wait on suitable internal data sets")
```

`scaleto1` *scaleto1 scales an input vector of CPUE to a mean of one x avCE*

Description

scaleto1 scales a vector of CPUE to a mean of one or avCE. The use of a mean of one means that visual comparisons between different time-series becomes visually simplified. The avCE option could be used to scale the CPUE to the average geometric mean - so as to put it on the nominal scale

Usage

```
scaleto1(invect)
```

Arguments

`invect`	a vector of linear scale CPUE

Value

a vector of CPUE re-scaled to a mean of one

Examples

```
 ce <- c(0.4667187,1.2628564,0.8442146,0.9813531, 0.5554076,0.7426321)
 scaleto1(ce)
```

setupzone — *setupzone makes zone's constant, dynamic, and productivity parts*

Description

setupzone makes the zone's constant, dynamic, and productivity parts returning them, along with glb, in a list. The objective of this function is to generate the orignal unfished equilibrium zone of nSAU SAU, and numpop populations

Usage

```
setupzone(
  constants,
  zone1,
  doproduct,
  uplim = 0.4,
  inc = 0.005,
  verbose = TRUE
)
```

Arguments

`constants`	the population constants derived from readdatafile
`zone1`	the zonal object driving the construction
`doproduct`	boolean, should the productivity calculations be made during the conditioning. defined in do_MSE and makeequilzone
`uplim`	the upper limit of harvest rate applied, default=0.4
`inc`	the harvest rate increment at each step, default=0.005
`verbose`	Should progress comments be printed to console, default=TRUE

Value

a list of zoneC, zoneD, product, and glb the main components of the zone

Examples

```
## Not run:
data(constants)
data(zone1)
out <- setupzone(constants,zone1)
zoneC <- out$zoneC
glb <- out$glb
str(zoneC[[1]])
str(glb)

## End(Not run)
```

STM *STM Generates the Size Transition Matrix for Inverse Logistic*

Description

STM With the input of the four parameters inside a vector, and a vector of initial lengths or mid-points of size classes STM generates a square transition matrix with the probabilities of growing from each initial size into the same or larger sizes. Negative growth is disallowed. All columns in the matrix sunm to one.

Usage

```
STM(p, mids)
```

Arguments

`p`	a vector of four parameters in the following order MaxDL the maximum growth increment of the inverse logistic, L50 the initial length at which the inflexion of the growth increment curve occurs, L95 - the initial length that defines the 95th percentile of the growth increments, SigMax - the maximum standard deviaiton of the normal distribution used to describe the spread of each distribution of growth increments
`mids`	a vector of initial lengths which also define the width of each size class thus, mids from 2 - 210 woul dimply 2mm size classes 1 - 3 = 2, 3 - 5 = 4, etc

Value

A square matrix with dimension =the length of the mids vector

References

Haddon, M., Mundy, C., and D. Tarbath (2008) Using an inverse-logistic model to describe growth increments of blackip abalone (Haliotis rubra) in Tasmania. Fisheries Bulletin 106: 58-71

Examples

```
param <- c(25.0,120.0,170.0,4.0)
midpts <- seq(2,210,2)
G <- STM(param,midpts)
print(round(G[1:30,1:8],4))
```

summarizeprod — *summarizeprod generates a summary of the productivity properties*

Description

summarizeprod generates a summary of the productivity properties by examining the productivity matrix for each SAU/population and extracting the Bmsy, annualH, MSY, Depletion, and RelCE at the maximum catch level (which approximates the MSY). It summarizes the total zone by summing all the productivity matrices and search for the largest catch again. It generates estimates of the annualH, depletion and RelCE by using a weighted average of those values from the separate SAU or populations, where the weighting is the proportion of the sum of the MSYs taken in each sau or population. This latter is only an approximation but provides at least an indication.

Usage

```
summarizeprod(product, saunames)
```

Arguments

`product`	The productivity array from doproduction containing the range of imposed harvest rates, and the resulting outputs for each population
`saunames`	the names of the different SAU

Value

a matrix containing the approximate productivity matrix for the zone

Examples

```
## Not run:
data(zone)
product <- zone$product
zoneprod <- summarizeprod(product,saunames=zone$zone1$SAUnames)
round(zoneprod,3)

## End(Not run)
```

sumpop2sau — *sumpop2sau gathers population data into sau data using sauindex*

Description

sumpop2sau gathers population data into sau data using sauindex

Usage

```
sumpop2sau(invect, sauindex)
```

Arguments

`invect`	a vector of population values for a given variable
`sauindex`	the indices of each sau for each population

Value

a vector of length nsau containing the sum of population values for each sau

Examples

```
vect <- c(5.8,6.2,13.2,23.8,3.3,3.7,29.7,26.3,38.9,9.1)
sauind <- c(1,1,2,2,3,3,4,4,5,5)
sumpop2sau(vect,sauind)   # should be 12 37 7 56 48
```

`sumpops` *sumpops takes the zoneDP and sums the populations within each SAU*

Description

sumpops summarizes the matureB, exploitB, catch, recruit variables within the depleted zone that has undergone the replicate application of an HS, and sums the values from each population within each SAU.

Usage

```
sumpops(invar, sauindex, saunames)
```

Arguments

`invar`	either zoneDP$matureB or exploitB, or catch, or recruit
`sauindex`	the SAU index of each population
`saunames`	the names of each SAU

Value

an array of projyrs x nSAU x reps

Examples

```
print("wait on new data")
```

`tasab` *tasab is a matrix of abalone maturity-at-length data*

Description

tasab is a 715 x 4 matrix of maturity-at-length data for blacklip abalone (*Haliotis rubra*) from two sites along the Tasmanian west coast. All data was collected in February 1995, but details, such as site name, accurate location, statistical block, year, month, and other details have been omitted for brevity.

Format

A data.frame of maturity-at-length data

site an identifier for the two different sites sampled

sex I = immature, M = male, F = female

length the shell length in mm

mature was the animal mature = 1 or not = 0

Subjects

- maturity ogives or logistic curves
- Binomial likelihoods

Source

Thanks to the Institute of Marine and Antarctic Science, which is part of the University of Tasmania, and especially to Dr Craig Mundy, leader of the Abalone Group, for permission to use this data collected in February 1995.

Examples

```
data(tasab)
head(tasab,20)
table(tasab$site,tasab$sex)
```

`testequil` *testequil runs a zone for hyrs and determines stability*

Description

testequil runs a given zone for hyrs at the given harvest rate, and then tests that the last values of matureB, exploitB, recruitment, and spawning biomass depletion are the same as the first (to three decimal places). It reports this to the console if verbose=TRUE. This is used with a harvest rate of zero and no variation in recruitment when defining the equilibrium zone under the application of the movement matrix.

Usage

```
testequil(zoneC, zoneD, glb, inH = 0, verbose = TRUE)
```

Arguments

`zoneC`	the constants components of the simulated zone
`zoneD`	the dynamic components of the simulated zone
`glb`	the global variables
`inH`	a vector of numpop harvest rates
`verbose`	should results go to the console, default=TRUE

Value

the dynamics component with hyrs of dynamics

Examples

```
## Not run:   # modzoneC takes too long to run because of doproduction
 data(zone)
 zoneDe <- testequil(zoneC=zone$zoneC,zoneD=zone$zoneD,glb=zone$glb)

## End(Not run)    #zoneC=zoneC; zoneD=zoneD; glb=glb; inH=0.0; verbose=TRUE
```

`WtatLen` *WtatLen Power function to describe weight at length relationship*

Description

WtatLen This can be used to generate any power function.

Usage

```
WtatLen(ina, inb, lens)
```

Arguments

`ina`	is the intercept of the power function
`inb`	is the gradient (or the explonent) of the power function
`lens`	a vector of lengths for which the weight will be calculated

Value

A vector of length(lens) containing the predicted values

Examples

```
## Not run:
a <- 0.0000542856
b <- 3.161415
lens <- seq(2,210,2)
wtL <- WtatLen(a,b,lens)

## End(Not run)
```

zone — *zone the primary object obtained from the function makeequilzone*

Description

zone contains seven objects, including 5 lists, a matrix, and an array. This is the

Format

A list of objects plus a matrix and array that make up the initial equilibrium zone

zoneC a list of the constants for each population

zoneD a list of the dynamic parts of the populations of a zone

glb a list of global constants, containing numpop,nSAU,midpts, Nclass, Nyrs

constants a matrix of biological properties for each population in the zone, derived from the datafile

product the productivity array from doproduction

ctrl the list containing control information for the run, including the datafile for the constants, the reps, the variation to be included when projecting

zone1 a list of objects used in the MSE

Examples

```
data(zone)
str(zone,max.level=1)
```

zonetosau — *zonetosau translates the zonexpop objects to zonexsau objects*

Description

zonetosau combines the dynamic results for each variable so that results by population become results by SAU. matureB, exploitB, catch, recruit, catchN, and Nt are simple summations of the totals for each population into their respective SAU. The harvest rate would be end of year or beginning of year estimates derived from dividing the catchxsau by the exploitable biomass x sau. Similarly the deplsB and depleB are the end of year matureB and exploitB divided by their respective unfished estimated by SAU obtained using getvar(zoneC,"B0").

Usage

```
zonetosau(inzone, NAS = NULL, glb, B0, ExB0)
```

Arguments

`inzone`	the projected zone dynamics objects made up of populations
`NAS`	the numbers-at-size 4D arrays from doprojection; default=NULL so it can be ignored during conditioning
`glb`	the object containing the global constants
`B0`	the estimate B0 for each population use getvar(zoneC,"B0")
`ExB0`	the estimate ExB0 for each population use getvar(zoneC,"ExB0")

Value

a list of dynamics variables by SAU

Examples

```
print("wait on suitable internal data sets ")
```

Index