User Manual for IOTC Yellowfin and Bigeye Tuna MSE software

Version 3.0 – July 2021

Paavo Jumppanen (Contact: Paavo.Jumppanen@csiro.au)

Dale Kolody

CSIRO Oceans & Atmosphere  
GPO Box 1538, Hobart, TAS, 7001, Australia

Table of Contents

[1 Introduction 1](#_Toc525132053)

[1.1 Acronyms and definitions used in this document 1](#_Toc525132054)

[1.2 Documentation conventions 3](#_Toc525132055)

[2 Obtaining and Installing the Software 3](#_Toc525132056)

[3 Software Organization 4](#_Toc525132057)

[3.1 OM Conditioning with Stock Synthesis 7](#_Toc525132058)

[3.2 MSE Control and Projection software 9](#_Toc525132059)

[3.3 C++ Projection Code 18](#_Toc525132060)

[3.3.1 General Coding Details Regarding the Mseom C++ Library 19](#_Toc525132061)

[3.3.2 R DLL and Interface Files 21](#_Toc525132062)

[3.3.3 Loading the Library 21](#_Toc525132063)

[3.3.4 Creating and Destroying an Object Instance 22](#_Toc525132064)

[3.3.5 The MSY projection 23](#_Toc525132065)

[3.3.6 The Managed Catch / Effort Forward Projection 24](#_Toc525132066)

[3.4 Some issues to be aware of: 26](#_Toc525132067)

[4 R Classes 27](#_Toc525132068)

[MseDefinition 27](#_Toc525132069)

[Description 27](#_Toc525132070)

[Attributes 27](#_Toc525132071)

[Name 27](#_Toc525132072)

[Label 27](#_Toc525132073)

[Date 27](#_Toc525132074)

[Author 27](#_Toc525132075)

[Notes 27](#_Toc525132076)

[PrimarySource 27](#_Toc525132077)

[CppMethod 27](#_Toc525132078)

[UseCluster 27](#_Toc525132079)

[npop 27](#_Toc525132080)

[nfleets 27](#_Toc525132081)

[SSRootDir 28](#_Toc525132082)

[SBlim 28](#_Toc525132083)

[Flim 28](#_Toc525132084)

[OMList 28](#_Toc525132085)

[nsimPerOMFile 28](#_Toc525132086)

[proyears 28](#_Toc525132087)

[targpop 28](#_Toc525132088)

[seed 28](#_Toc525132089)

[recentPerFirst 28](#_Toc525132090)

[recentPerLast 28](#_Toc525132091)

[seasonCEDist 28](#_Toc525132092)

[nsubyears 28](#_Toc525132093)

[lastSeas 29](#_Toc525132094)

[firstSeas 29](#_Toc525132095)

[firstSSYr 29](#_Toc525132096)

[firstCalendarYr 29](#_Toc525132097)

[lastCalendarYr 29](#_Toc525132098)

[firstMPYr 29](#_Toc525132099)

[MPDataLag 29](#_Toc525132100)

[catchBridge 29](#_Toc525132101)

[catchBridgeCV 29](#_Toc525132102)

[indexFisheries 29](#_Toc525132103)

[ReccvTin 29](#_Toc525132104)

[ReccvRin 30](#_Toc525132105)

[RecACTin 30](#_Toc525132106)

[NInitCV 30](#_Toc525132107)

[NInitCVdecay 30](#_Toc525132108)

[selExpRange 30](#_Toc525132109)

[selAgeRange 30](#_Toc525132110)

[selWLRange 30](#_Toc525132111)

[TACEcv 30](#_Toc525132112)

[Ccv 30](#_Toc525132113)

[Icv 31](#_Toc525132114)

[IACin 31](#_Toc525132115)

[Cbcv 31](#_Toc525132116)

[nCALobs 31](#_Toc525132117)

[Ibeta 31](#_Toc525132118)

[ITrendin 31](#_Toc525132119)

[ImplErrBias 31](#_Toc525132120)

[RecScale 31](#_Toc525132121)

[modelWeight 31](#_Toc525132122)

[totalSims 32](#_Toc525132123)

[Methods 32](#_Toc525132124)

[createMseFramework(.Object, Report=FALSE, UseCluster=NA, UseMSYss=0) 32](#_Toc525132125)

[5 Operating Model Equations 32](#_Toc525132126)

[5.1 Notation 32](#_Toc525132127)

[5.2 Population Dynamics 33](#_Toc525132128)

[5.3 Management Implementation Errors 40](#_Toc525132129)

[5.4 Observation Errors 40](#_Toc525132130)

[6 Candidate Management Procedures 41](#_Toc525132131)

[7 Performance Measures and Reference Points 43](#_Toc525132132)

[8 Acknowledgements 46](#_Toc525132133)

[9 References 47](#_Toc525132134)

# Introduction

This user manual provides a roadmap to the Indian Ocean Tuna Commission (IOTC) yellowfin (YFT) and bigeye (BET) Management Strategy Evaluation (MSE) software, including the mathematical equations for the dynamics, candidate MPs, and R scripts for demonstrating the *niMSE-IO-BET-YFT* software*.*  The initial software development work was jointly funded by the EU (through IOTC/FAO) and CSIRO. The user manual focuses on technical implementation details and options within the software. This manual, which is a living document, is updated from time to time as features and contributions are added.

This project was initiated with an over-arching design objective of keeping the code accessible to IOTC member scientists through the use of freely available software, familiar to the fisheries science community, and ideally re-using and extending established MSE code to the extent possible. Most of the software is R-based (The R Foundation for Statistical Computing), and depends on packages freely available in the CRAN repository. The core of the MSE platform was adapted from the Carruthers et al (2014) application developed for Atlantic Bluefin Tuna (ABT) (and funded by the ICCAT Atlantic-Wide Research Programme on Bluefin Tuna). Whilst the first implementation of this software closely mimicked the structure of the Carruthers code, that approach suffered from significant performance issues when running large sets of models. As such, the software was completely re-designed and restructured with a view to minimising memory use and extending the use of cluster processing to reduce computation time.

The overall package is relatively simple to install and run for anyone familiar with R, though software modifications might not be easy for novice R users. There are two main components that are not implemented in R: (i) Stock Synthesis (SS) assessment software (Methot and Wetzel 2013) is used for Operating Model (OM) conditioning, and (ii) an optional C++ projection component called from R. Additionally, promising new MPs have been developed which require TMB software (that interfaces with R).

The MSE software is freely available from Github (<https://github.com/pjumppanen/niMSE-IO-BET-YFT>), and the archive should include all of the project documents submitted to the IOTC (niMSE-IO-BET-YFT\Report\phase 3 - to Jul2021). The objects and scripts required to reproduce the most recent MP evaluations presented to the IOTC community (as documented in the relevant working papers) should be in this archive (but it is always best to consult with the authors before downloading, as there are inevitable delays in archiving). This document refers to example scripts with a linear sequence of function calls and (hopefully) informative comments. The scripts used for the most recent working papers should have analogous structure, but unfortunately, the exact sequence of function calls may have followed a circuitous path, with inconsistent commenting. SS files produced to condition the OMs are very large and are not archived outside of CSIRO (though could be made available if they were of interest).

## Acronyms and definitions used in this document

ABT Atlantic Bluefin Tuna.

BET Bigeye Tuna.

CCSBT Commission for the Conservation of Southern Bluefin Tuna.

CPUE Catch per Unit Effort – usually assumed to be standardized into a relative abundance index for fish vulnerable to a particular fishery.

HCR Harvest Control Rule – the numerical algorithm for recommending a management action (e.g. providing a TAC given a biomass estimate). In this document, the term is generally not intended to encompass data collection and analysis or fitting a stock assessment model, as these may be considered to be separate components of a complete MP.

ICCAT International Commission for the Conservation of Atlantic Tuna

IOTC Indian Ocean Tuna Commission.

IWC International Whaling Commission.

OM Operating Model – this usually refers to the combination of the generic projection software and suite of model specifications used to simulation test the performance of candidate MPs. We often refer to the OM projection software and OM parameterization separately.

MP Management Procedure – the simulation-tested combination of pre-agreed data collection methods, supporting analysis, and Harvest Control Rule. The term is often used interchangeably with MSE, however the *sensu stricto* MP definition (as used in the IWC and CCSBT) explicitly requires a very high level of pre-specification (i.e. of the data requirements and supporting analyses), to preclude the inherent risk of assessment groups failing to reach consensus during the application of an HCR. MSE is a broader term that does not necessarily imply the same degree of pre-specification.

MSE Management Strategy Evaluation – the process (or final product) of simulation testing a fishery management strategy (see MP).

MSY Maximum Sustainable Yield.

SS Stock Synthesis assessment model software.

TAC Total Allowable Catch – the catch quota set by an MP (it could be fishery-specific or the aggregate across fisheries, depending on context).

TAE Total Allowable Effort – a fishery effort constraint set by an MP. In this context it is manifested as an effort multiplier applied to recent estimates of fishery-specific fishing mortality from an assessment model. For the simulation testing, there is an assumption that effort regulations will translate directly into fishing mortality regulations. In practice, it may be very difficult to define effort is such a way that this can be achieved.

WPM IOTC Working Party on Methods.

WPTT IOTC Working Party on Tropical Tunas

YFT yellowfin tuna.

## Documentation conventions

We use the following style conventions in this document:

filename.ext specific file or directory names or functions using character style FileNames.

Variables variable and parameter names using character style Variables.

Code source code using paragraph style Code.

Menu Commands menu and/or button presses in dialogs and programs shown using Menu character style.

# Obtaining and Installing the Software

The software can be downloaded from Github (https://github.com/pjumppanen/niMSE-IO-BET-YFT). This software was developed under Windows 7 (Service Pack 1), 64 bit Operating System. The code and documentation is publicly accessible (independent developers should fork their own version of the code and propose extensions to be re-integrated into the master via the normal GitHub pull request mechanism). The following steps are required before an MSE can be set-up and run:

1. Install R from <https://cloud.r-project.org/>. We recommend that new R-users should consider an R tutorial, and install an Interactive Development Environment (e.g. Rstudio or TINN-R) to navigate the source files and scripts.
2. Install the following R packages from CRAN:
   * *keep*
   * *stringr*
   * *r4ss*
   * *ggplot2*
   * *reshape2*
   * *parallel*
   * *abind*
   * *data.table*
   * *mseviz*
   * *ggstance*
   * plus any others that are noted to be missing when running the software
3. Go to <https://github.com/pjumppanen/niMSE-IO-BET-YFT> and click on the clone or download button, then press the download zip button.
4. Extract the niMSE-IO-BET-YFT-master.zip file to a drive / directory of your choice.
5. Phase 3 of the project included development of some new MPs that use TMB (Template Model Builder) software to implement numerically-efficient joint process and observation error production models (e.g. Kolody & Jumppanen 2020, 2021A). These appear to be the most promising MPs developed to date, and have additional installation requirements:

* CRAN package RTools
* R package *BuildSys* (in part developed through this project, is available from the authors, and expected to be incorporated into CRAN). *BuildSys* manages the compilation of the TMB DLL, and adds the capacity for efficient IDE debugging.

The R code and control files, scripts, etc. are ASCII files and can simply be edited with standard text editors (or within an R IDE), and submitted to an active R session to update the MSE. There are also R callable C++ routines implementing the Baranov catch equation projection code in a DLL. We provide the source code, including the CSIRO-developed ADT software (Automatic Differentiation with Tapenade) and ancillary libraries, and the fully functional DLL. Any change to the C++ code involving the objective functions will require updated differentials. Although the C++ code is freely modifiable, the step of creating differentials through the use of TAPENADE (INRIA, France) requires a license (as of Sep 2016, an annual license is once again free for academic use).

# Software Organization

Figure 1 shows the class hierarchy in the niMSE-IO-BET-YFT software. The MseDefinition class is a placeholder for all the parameters required to define an instance of the MseFramework class. This includes information naming the Stock Synthesis derived assessment model outputs to use, how to weight them and how many simulations to run, operating model stochasticity, year range of projection and starting year of MP management. The MseFramework class in turn holds one instance of the StockSynthesisModel class per Stock Synthesis assessment result set. The StockSynthesisModel class contains one instance of the StockSynthesisModelData class, which serves as a storage location for the model data used in projections. It exists to allow the data needed to run a projection to be marshalled across process boundaries (when using cluster processing) in a memory efficient manner. The single instance of the ReferenceVars class in StockSynthesisModel contains the MSY projection data needed to produce the normalised reporting statistics for MP projections. The ManagementVars class, on the other hand, contains the performance data from a model run. StockSynthesisModel contains one instance of ManagementVars for the historic data from Stock Synthesis, and one instance per MP projection that contains the projected performance data. The Projection class is a helper class that centralises the code implementing the projection process, and is used to perform the MSY and MP projections for a given Stock Synthesis model.

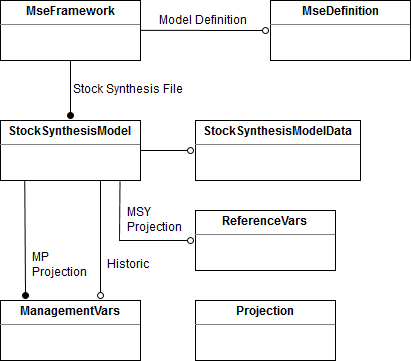


Figure 1. Class diagram for niMSE-IO-BET-YFT software.

Figure 2 shows the sequence of operations in setting up and running a complete MSE using the IOTC yellowfin and bigeye MSE software. Functionally, it is divided into two parts, (1) Conditioning and (2) Model setup and running.

The *“Operating Model Conditioning”* phase involves fitting a suite of Stock Synthesis assessment models to estimate parameters and states that are consistent with the historical data and general understanding of the fishery through the stock assessment process. In this application, conditioning is carried out using Stock Synthesis. The operating model implemented in niMSE-IO-BET-YFT is structurally consistent with the Stock Synthesis implementation (subject to the minor difference in how fishing mortality is implemented as discussed below). This ensures that when the identified model parameters from conditioning are transferred into the niMSE-IO-BET-YFT operating model, the observed dynamics will be consistent with Stock Synthesis.

Simple R functions automate the process of setting up an array of stock synthesis analyses using template SS control and data files. Batch files are produced to automate the running of the SS analyses, and additional R functions are provided for exploring and evaluating the suite of assessment results. These tools form a template to help automate parts of the OM conditioning process, but the end-user must decide which models to retain or reject from the ensemble using their own judgement, and modify the template files accordingly. These functions are ad hoc tools that worked for the specific cases described and are included because they may be useful, but they were never intended to be robust all-purpose tools. Users will need to know enough about R to modify them for their own purposes.

In the case of the *“Setup and run MSE”* phase, R scripts are used to firstly define the MSE, then with that definition, create an MseFramework object instance which then forms the basis for tuning and testing MP’s. The runMse() method is used to invoke both MP tuning and MP testing and a number of other methods are available to query statistics and plot performance data for those runs.

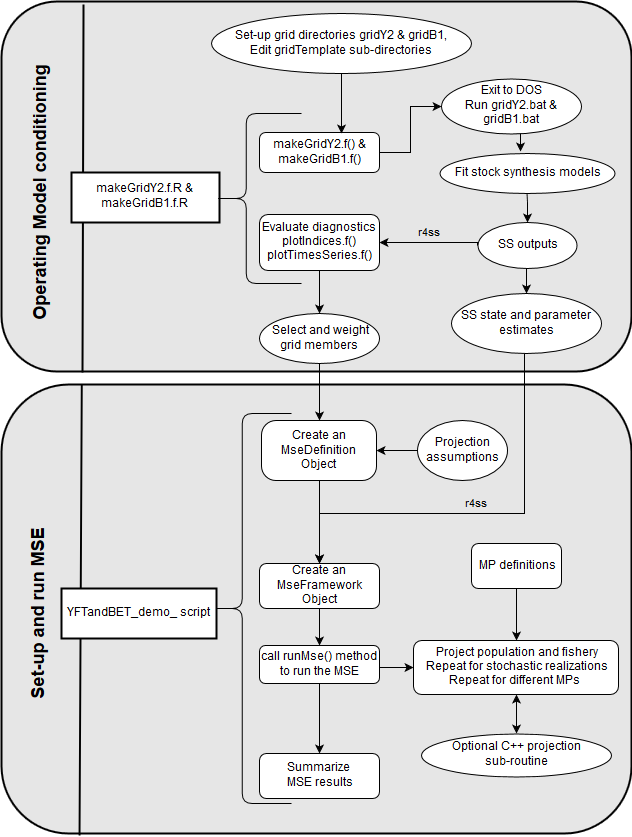


Figure 2. Flowchart illustrating the sequence of processing performed in running the IOTC yellowfin and bigeye MSE. Square boxes represent R-based code.

## OM Conditioning with Stock Synthesis

Conditioning is the process of estimating OM parameters that are consistent with (conditional on) historical data and assumptions about the dynamics of the fishery and population. The approach to conditioning used here involves fitting a suite of Stock Synthesis (SS) assessment models (Methot and Wetzel 2013) that encompass a range of parameter, data and/or structural uncertainties relative to the most recent stock assessment. Note that this step can be skipped if the desired OM has already been created and saved. In that case, one can jump to the next section and simply load() the OM from within R (as indicated in **YFTandBET\_demo\_script.R**)

We refer to the combination of models as a *"grid"*. Originally, this involved a full balanced factorial design of all possible assumption interactions (e.g. 3 levels of steepness, 3 levels of *M* and 3 alternative catch history scenarios would form a grid of 3 X 3 X 3 = 27 models). As the number of interactions increased, fractional factorial design was adopted to keep the grid size tractable. We opted for 50-150 models, with a balanced representation of each factor level. The minimum design would allow all main effects to be estimated in the context of a GLM. All 2 way interactions were recognized as desirable, but seemed unlikely to influence MP selection results in the comparisons undertaken (Kolody & Jumppanen 2019). Though, it appears that in the tests conducted, the R package *planor* always included all 2 way interactions among the 3 level factors. We usually refer to the combination of models comprising an OM as an ensemble, which could be the same as a grid, or include models from multiple grids, or may exclude some models from a grid (e.g. if they are judged to be biologically implausible, or redundant).

Table 1 lists the main scripts and functions used for the conditioning for the demonstration case yellowfin and bigeye OMs.

Table 1. Main scripts and functions used to conduct the SS conditioning for the demonstration case Indian Ocean yellowfin and bigeye operating models.

|  |  |
| --- | --- |
| R Script |  |
| make\_YFTOMGridScript.R | YFT conditioning |
| make\_BETOMGridScript.R | BET conditioning |
| seasAsYrToDecYr.f.R | Functions for converting date formats |
|  |  |
| R Function |  |
| makeGridY2.f() | Make YFT grid |
| makeGridB1.f() | Make BET grid |
| importGrid.f() | Import grid of SS output files into R |
| seasAsYrToMSEYrSeas.f() | Function for converting between SS seasons-defined-as-years and calendar years (in file seasAsYrToDecYr.f.R) |
| plotIndices.f() | Plot some summary SS output statistics partitioned by grid assumptions |
| timeSeriesPlots.f() | Plot time series of some SS output statistics as summary distribution percentiles |
|  |  |

For the demonstration cases, we used SS version 3.24Y (included in the bundled files as ss3.24Y.exe). Earlier versions of SS may fail because of a change to the growth parameterization, or provide incorrect results, due to a bug in the spawning biomass calculations (discussed in the phase 1 project report). The file make\_YFTOMGridScript.R reproduces the yellowfin results from the final report which is the example case discussed below (make\_BETOMGridScript.R is the analogous file for bigeye). Key steps (following software installation described in section 2):

1. Prior to running make\_YFTOM\_script.R, it is necessary to create the root dir gridY3, which must contain a gridTemplate folder and SS executable (see: ...\MSE-IO-BET-YFT\OMconditioning\YFT\gridY3\gridTemplate). gridTemplate includes all the files required to conduct an SS model fitting (e,g, control, data and forecast files). The files templateYFT.dat and/or templateYFT.ctl need to be modified to include all assessment model options required for the grid. This is accomplished with character string option flags. For example, to represent three levels of stock recruit steepness, templateYFT.ctl contains the following lines:

# steepness switches

# xxx h70 0.2 1 0.7 0.7 0 0.2 -1 # SR\_steep

# xxx h80 0.2 1 0.8 0.8 0 0.2 -1 # SR\_steep

# xxx h90 0.2 1 0.9 0.9 0 0.2 -1 # SR\_steep

Each line starts with a character string identifier e.g. # xxx h70, where the first character is the Stock Synthesis comment character # that causes SS to ignore the remainder of the line.

2) The function makeGridY3.f()thencreates a file structure in which a separate SS model is assigned to its own directory, which contains all of the required SS inputs. A model sub-folder is created for every combination of model options and the options are captured in the folder name. That is, folder GridY3 contains 54 model configurations, across 5 dimensions, each given a sub-folder with a name resembling R4MvEst\_h70\_M06\_t00\_q0. In this case, the first dimension R4MvEst refers to spatial and population structure (4 regions, movement estimated) and is actually identical for all models in this grid. The other 4 dimensions have 2-3 levels each (see the phase 1 project report for details), where h70 refers to the stock recruit steepness option (*h* = 0.7), while the other options include h80 and h90 (*h =* 0.8 and 0.9 respectively).

makeGridY3.f()copies over the requisite template files, and modifies them by stripping out the relevant option flags (removing the # xxx h70 comment above activates the first line of the .ctl file above, while the other options remain as inactive comments).

The function also creates a DOS batch file within each new directory to run the individual SS analysis. There are two batch file options, with or without the inverse Hessian calculation. Including the calculation can greatly increase the SS run time, but may provide useful information about parameter estimation uncertainty or convergence problems. A master batch file at the GridY3 root, gridY3.bat is also created, which sequentially calls all of the individual batch files. By default, 4 smaller non-overlapping batch files are also created that can be run instead of the master batch file (in parallel to take advantage of multiple CPUs).

3) After running makeGridY3.f(), pause in the make\_YFTOM\_script.R script, open a DOS command windows and call gridY3.bat. This runs a SS analysis (fits an assessment model) for each element of the grid, and could take many hours, depending on the size of the grid and complexity of the models.

4) Function importGrid.f()imports each SS model output as an individual R object.

5) Function plotIndices.f() plots some simple summary distribution statistics across the grid to rapidly inspect for outlier behaviour in terms of convergence, gross fit between predictions and observations, recruitment trends, general stock status inferences, etc. Each index is disaggregated according to assessment option, and the distribution (boxplot) is plotted with the other options marginalized, (e.g. see the phase 1 final report). Function plotTimeSeries.f() can be used to plot the distribution of some of the standard time series, e.g. *Bt/BMSY* (see phase 1 final report).

The procedure outlined in this R-script does not constitute a comprehensive stock assessment model evaluation, but provides a useful way to set-up and review a substantial number of models. The demonstration grids for both bigeye and yellowfin are very well-behaved, and reasonably consistent with the inferences from the stock assessments that were current at the time. However, a large grid with contrast in many dimensions might include some implausible interactions. Subjective decisions about what options to include in the grid, and how to weight (or reject) individual models are usually required, and the whole process may involve several iterations. Detailed results from some individual models can and should be examined using the r4ss package (version 1.24.0, from CRAN) functions SS\_outputs() and SS\_plots(), as in a traditional SS stock assessment**.** Detailed inspections have typically included stock status “corners” (the highest and lowest depletion, *B/BMSY*, and the highest and lowest productivity, *MSY*) and any strange outliers with respect to the aggregate indices examined. The latest software and scripts do not support the sort of stock assessment model performance diagnostics that are under investigation for more general use in the IOTC (e.g. Matsumoto et al. 2018). Additionally, the DOS batch files that fit the SS grid models have been updated to include repeated minimization from jittered initial parameter values, to reduce the problem of convergence failure and sensitivity to initial conditions.

In this MSE software, differential weighting of SS models within the OM ensemble is achieved by the number of stochastic simulations for each set of SS parameters (i.e. number of simulations is proportional to the weight). Most OM model specifications are adopted directly from the SS outputs, including the initial numbers-at-age for the first year of the projections (with the addition of age-specific noise), key biological parameters (M, stock recruit steepness, size-at-age, etc.) and fishery selectivity. However, a number of additional parameters are required to fully specify an OM as detailed in the MseDefinition and MseFramework objects below.

The full suite of SS output files required to populate the demonstration case OMs in the phase 1 report are not provided on GitHub (because of the large number and size of files). They can be recreated using the scripts above, however, the full demonstration case OMs are supplied as R objects that can be loaded to run the MSE. The SS output files are only provided for a couple of models to demonstrate the OM loading.

## MSE Control and Projection software

Once the conditioned assessment models have run and the grid of YFT and BET model results created, the demo script YFTandBET\_demo\_script.R can be run. To do so, run Rgui (64 bit), then change the working directory to the folder YFT-MSE (which is a subfolder of the niMSE-IO-BET-YFT-master project that you downloaded and extracted from GitHub). Do so by selecting the File/Change Dir… menu item and navigate to, and select the YFT-MSE folder and press OK. The demonstration cases can be run by selecting the File/Source R Code… menu, navigating to the RScripts subfolder, selecting the **YFTandBET\_demo\_script.R** and pressing the OK button. It may be preferable to step through the R script by submitting the code line by line or in small blocks. Using an IDE for R is a preferable way to work with the R script files.

The R script YFTandBET\_demoScript.R demonstrates several MSE applications (lower portion of Figure 2) and assumes that the grid of SS models for the YFT and BET cases have been created. The line of code,

source("Source/MseMain.R")

loads all the source code required for the application into the R session. The code in,

source("RScripts/Build OMyftNEr.R")

defines an MseDefinition object which, in this case, is a demonstration definition with minimal process or observation error, 2 stock synthesis models, and one replicate for each model. From this definition, we create an MseFramework instance with,

OMyftNEr <- createMseFramework(MseDef, UseCluster=0)

The MseFramework at this point is initialised and ready to perform MP evaluation. All historic series are stored in the object instance and the MSY projections carried out. For large grids of stock synthesis models the time required to carry out the MSY projection can be large so it can be useful to save the resulting MseFramework object for later use in MP evaluation and testing, as with,

# Save the MseFramework

save(OMyftNEr,   
 file=paste(getwd(),"/Objects/OMyftNEr.RDA",sep=""))

# Load the previously created MseFramework

load(file=paste(getwd(),"/Objects/OMyftNEr.RDA",sep=""))

To run a management strategy evaluation we call the runMse() method of MseFramework. The method prototype is,

function(.Object,

MPs,

TuningPars=NA,

interval=3,

Report=FALSE,

CppMethod=NA,

UseCluster=NA,

EffortCeiling = as.double(20.0),

TACTime = 0.5,

rULim = 0.5)

MPs is a list of MPs to use in the projection. MPs can be specified in a number of ways, depending on requirements. In the simplest approach, an R collection of MP class names can be used, as in,

MPs <- c("CC200",  
 "CC400",  
 "IT1.50",  
 "IT3.50",  
 "PT41.100.2",  
 "PT41.100.9")

Using this approach, the names are used to identify the MPs in the statistics summaries and graphical output is the MP class names. Alternatively, the MP reporting names can be altered by using a named list, as in,

MPs <- list(MP1="CC200",  
 MP2="CC400",  
 MP3="IT1.50",  
 MP4="IT3.50",  
 MP5="PT41.100.2",  
 MP6="PT41.100.9")

In this case, the statistics will be summarised using the names MP1 through MP6. In addition, with a list, the elements can be of class MP\_Spec rather than simply character strings. This becomes important when wanting to use a tuned MP with a given tuning in an MSE. Typically, instances of this class are not created directly but result from conducting an MP tuning and obtaining the tuning by calling the method getMPs().

Returning to the runMse() method parameters, the Report argument turns detailed reporting of the run on or off. The CppMethod and UseCluster parameters are over-rides of the corresponding class attributes in the MseDefinition class. The TACTime parameter sets the time in the annual cycle that the TAC catch is taken in the Pope approximate for TAC projections. The rULim parameter applies a limit to fishing effort in the Pope approximation for TAC projections. The EffortCeiling parameter applies a limit to maximum fishing effort in the solution to the Baranov catch equation. The parameter interval controls the duration between MP management updates and defaults to every 3 years. Finally, the TuningPars parameter specifies the parameters that control MP tuning. When NA (or not explicitly specified) the MSE is run without MP tuning.

Returning to the example script, the line,

OMyftNEr <- runMse(OMyftNEr,

MPs="CC001",

interval=3,

Report=F,

UseCluster=0)

runs an MSE projection using the MP CC001, which is constant catch of 1 metric ton, or an approximation to a closed fishery. The results of the projection are stored within the returned class instance and in this case, updates the original MseFramework object. However, it could assign to an entirely new MseFramework object, keeping the original intact.

Time series plots can be created using the mseviz R package (<https://github.com/iagomosqueira/mseviz>). The MseFramework methods msevizHistoricTimeSeriesData() and msevizProjectedTimeSeriesData() return the historic and projected time series data in the format required by the plotOMruns() function in the mseviz package. Both these methods have the prototype,

function(.Object, prefix="", Indicators=NULL)

where .Object is the MseFramework object instance, prefix is a string used to prefix the names of MPs in reported results, and Indicators is a list of indicators to report and can be any combination of the indicators in Table 2. If Indicators is NULL then all the indicators in Table 2 are reported.

Table 2. List of indicators supported by the msevizHistoricTimeSeriesData() and msevizProjectedTimeSeriesData() methods.

|  |  |
| --- | --- |
| R Statistic Name | Meaning |
| “CPUE(aggregate)” | Annual catch per unit effort aggregated over all fleets |
| “Recruitment” | Annual recruitment |
| “B/B0” | Annual biomass over initial biomass |
| “B/BMSY” | Annual biomass over biomass at MSY |
| “SSB/SSB0” | Annual spawning stock biomass over initial spawning stock biomass |
| “SSB/SSBMSY” | Annual spawning stock biomass over spawning stock biomass at MSY |
| “F/FMSY” | Annual effort (aggregated over all fleets) over effort at MSY |
| “C” | Annual catch aggregated over all fleets |
| “TAC”  “C/TAC”  “PrGreen” | Aggregate annual catch quota  Ratio of Catch/TAC (indicator of implementation error, which could be intentional or a consequence of a shortage of vulnerable fish)  Annual probability of being in green Kobe |
| “PrRed” | Annual probability of being in red Kobe |
| “PrOrange” | Annual probability of being in orange Kobe |
| “PrYellow” | Annual probability of being in yellow Kobe |
| “Recruitment by Qtr” | Quarterly recruitment keyed by the quarter. Key names are “Recruitment Q1” through to “Recruitment Q4” |
| “CbyRF” | Annual catch by fleet. Key names are “C by Fleet 1”, “C by Fleet 2”,… through to the number of fleets in the model |
| “CPUEbyArea” | Annual catch per unit effort aggregated by area / region |

At the time of development the implementation of plotOMruns() in the mseviz package did not exactly fulfil the requirements of this application so an alternative that was based on the mseviz implementation was developed. plotOMruns2() is that implementation and has the prototype,

plotOMruns2 <- function(om.dt,

runs.dt,

indicator,

limit = missing,

target = missing,

Cref=missing,

probs = c(0.1, 0.25, 0.5, 0.75, 0.9),

ylab = "",

ribCol = "grey",

lastHistYr = 2015,

firstMPYr = 2019,

doWorms = TRUE,

CScale=0.001)

om.dt and runs.dt are the data.table outputs of the msevizHistoricTimeSeriesData()and msevizProjectedTimeSeriesData() methods respectively. Indicator is a string naming the indicator to produce in the output data.tables and can be one of the indicators named in Table 2. Note also, that for the indicator to be plot-able then that indicator must have been named in the calls to the msevizHistoricTimeSeriesData() and msevizProjectedTimeSeriesData(). Furthermore, if worms are to be plotted for the given indicator then the "SSB/SSBMSY" indicator must have also be named in those method calls. If limit is provided then plotOMruns2() will include a red horizontal line with y-intercept equal to limit. Similarly, if target is provided then a green horizontal line with y-intercept equal to target will be drawn. If Cref is provided then a black horizontal line with y-intercept Cref times CScale is drawn. probs specifies the percentile divisions to use in calculating the statistics. ylab gives an additional label to the plot. ribCol sets the colour of the ribbon in the plot. lastHistYr and firstMPYr set the x-intercepts for vertical lines drawn on the plot to indicate the last historical year and first MP year respectively. doWorms enables or disables the plotting of worms. CScale sets the catch scale to apply to the Cref data value.

Returning to the demo script, we generate a series of time series plots with the R code,

# Plot some key time series

histd <- msevizHistoricTimeSeriesData(OMyftNEr)

projd <- msevizProjectedTimeSeriesData(OMyftNEr)

plotOMruns2(histd, projd, "SSB/SSBMSY", doWorms=FALSE)

plotOMruns2(histd, projd, "CPUE(aggregate)", doWorms=FALSE)

plotOMruns2(histd, projd, "C", doWorms=FALSE)

plotOMruns2(histd, projd, "F/FMSY", doWorms=FALSE)

In this particular case doWorms is set to false because the example only has 2 model realisations but doWorms requires a minimum of 3 and so would fail if doWorms=TRUE.

General numerical statistics can be produced for the MSE model projections using the performanceStatistics() method. This method has the function prototype,

function(.Object,

Statistics,

AvgFirstYr,

AvgLastYr=NA,

percentiles=c(0.1,0.25,0.5,0.75,0.8,0.9),

thisMP=NA,

prefix="",

appendTo=NULL)

The Statistics parameter is an R collection naming the statistics to report and can be any of the names shown in Table 3. The AvgFirstYr and AvgLastYr parameters control the year range over which the named statistic is calculated. If both parameters are specified then both should be in calendar years. If AvgLastYr is NA then AvgFirstYr represents the number of years to calculate the statistics over with a starting year equal to the first MP year, as specified by the firstMPYr attribute in the MseDefinition object that created the MseFramework object. The method calculates mean and percentile values (from the distribution) with percentile divisions controlled by the percentile parameter. Normally this method will calculate statistics for all MPs tested in the MseFramework. By naming the MP with the thisMP parameter we can limit the statistics calculation to results for this MP only. The MPs named in the MP column of the results data.table correspond to the MPs in the MSE projection. We can add a prefix to the MPs named by specifying a prefix string with the prefix parameter. We can aggregate the statistics result from this method call to a previous one by providing a performanceStatistics() results set as the appendTo argument in the method call.

Table 3. List of Statistics supported by the performanceStatistics() method.

|  |  |
| --- | --- |
| R Statistic Name | Meaning |
| "SBoSB0" | Mean spawning biomass over initial spawning biomass |
| “minSBoSB0” | Minimum spawning biomass over initial spawning biomass |
| “SBoSBMSY” | Mean spawning biomass over spawning biomass at MSY |
| “FoFMSY” | Mean effort over effort at MSY |
| “GK” | Probability of being in green Kobe |
| “RK” | Probability of being in red Kobe |
| “PrSBgtSBMSY” | Probability of spawning biomass being greater than spawning biomass at MSY |
| “PrSBgt0.2SB0” | Probability of spawning biomass being greater than 20% of initial biomass |
| “PrSBgtSBlim” | Probability of spawning biomass being greater than SBlim where SBlim is 40% of spawning biomass at MSY |
| “Y” | Mean catch mass in metric tons |
| “relCPUE” | Mean catch rates relative to catch rates over four last data years |
| “YoMSY” | Mean catch over catch at MSY |
| “APCY” | Mean catch mass at time t over catch mass at time t-1 |
| “AAVY” | Percentage average absolute change in catch mass |
| “YcvPct” | Mean standard deviation of catch mass over mean catch mass (a normalised measure of catch variability) |
| “PrYlt0.1MSY” | Probability that mean catch is less than 10% of catch at MSY |
| “Yf” | Mean catch mass by fleet |
| “CPUEr” | Catch per unit effort by area / region |

As alluded to earlier, calls to the runMse() method can be used to tune MPs. All that is required is the naming of MPs with the class IO\_MP\_tune and providing a class instance for the TuningPars argument that defines the tuning. An extract from the demo script shown below shows a typical example of this.

TuningPars <- new("TuningParameters")

TuningPars@performanceMeasure <- "SBoSBMSY"

TuningPars@performanceMeasureClass <- "0.5"

TuningPars@performanceMeasureYears <- 20

TuningPars@tuningTarget <- 1.7

TuningPars@tuningTolerance <- 0.01

MPL2 <- c("PT41.100.2", "PT41.100.9", "PT41.tune.9")

# forcing C++ method because it runs faster

print(system.time(OMyft2r108 <- runMse(OMyft2r108, TuningPars=TuningPars, MPs=MPL2, interval=3, Report=F, UseCluster=1, CppMethod=1)))

In the list of MPs to use in runMse() PT41.100.2 and PT41.100.9 are non-tuned MPs (class IO\_MP) and PT41.tune.9 is a tuned MP. If runMse() were called without a TuningPars argument then PT41.tune.9 would simply be run as a fixed MP. By providing the TuningPars argument we instruct it to perform tuning on any tunable MPs named in the MPs argument. Internally, when performing tuning, the performanceStatistics() method is invoked to measure the performance for the sake of tuning. In the TuningParameters class instance above, performanceMeasure and performanceMeasureYears correspond to the Statistics and AvgFirstYr arguments to the call to performanceStatistics(). Hence, in this case we are measuring SBoSBMSY (as defined in Table 3) with a 20 year average. The performanceStatistics() method produces a measure for the mean and all the percentiles (0.1,0.25,0.5,0.75,0.8,0.9) meaning that it is necessary to name which figure to use in tuning. This is the purpose of the performanceMeasureClass and indicates that we will be using the 50th percentile of SBoSBMSY. Functionally, “SBoSBMSY” combines with “0.5” to create the column name that is queried in the performanceStatistics() output. Finally, tuningTarget and tuningTolerance set the tuning target we wish to achieve and the degree of fit require for tuning to complete. In this case, we are determining the required tuning for PT41.tune.9 such that the 50th percentile of SBoSBMSY averaged over 20 years has a value of 1.7 to within 1%. Assuming the tuning converged, the MP tuning can be obtained by calling the getMPs() method. For instance,

getMPs(OMyft2r108)

returns,

[[1]]

An object of class "MP\_Spec"

Slot "MP":

[1] "PT41.100.2"

Slot "MP\_Name":

[1] "PT41.100.2"

Slot "tune":

[1] 1

Slot "tuneError":

[1] 0

[[2]]

An object of class "MP\_Spec"

Slot "MP":

[1] "PT41.100.9"

Slot "MP\_Name":

[1] "PT41.100.9"

Slot "tune":

[1] 1

Slot "tuneError":

[1] 0

[[3]]

An object of class "MP\_Spec"

Slot "MP":

[1] "PT41.tune.9"

Slot "MP\_Name":

[1] "PT41.tune.9"

Slot "tune":

[1] 2.458244

Slot "tuneError":

[1] 0.005882353

It returns a list of objects of class MP\_SPEC, one for each MP. In our example we had three, the first two being fixed MPs and the last being tuned. In the first two MPs the “tune” slot is 1 and the “tuneError” slot is 0 because they are fixed MPs. In the final tuned MP the “tune” slot has a value of 2.260303 which is the parameter value needed to obtain the desired tuning. The “tuneError” parameter has a value of 0.005882353 meaning the tuning is to within 0.59% of the target level. The “tuneError” parameter is useful in determining after the fact, whether a given tuning converged.

As an aid in producing reports there are two additional functions that may be of use : createTable1() and createTable2(). Both these functions create summary tables in a format required by the IOTC in the context of MSE reporting. Both functions have the same prototype of,

function(years, results, MPs, MPs\_short, prefix="")

years is the years over which to average results, results is an R list of MseFramework objects holding the results of runMse() calls. MPs is a list of MPs to report. MPs\_short is a corresponding list of names to substitute for the full MP names in the tabular output. prefix is a string to prefix to the resulting file name. Both functions produce word XML output files that can be opened with Microsoft Word. The files are written to the Report folder and have names that follow the template {prefix}Table\_{1 or 2}\_{years}yr.xml . For instance,

# create Table1 example

createTable1(20, list(OMyft2r108), c("PT41.100.2", "PT41.100.9", "PT41.tune.9"), c("MP1", "MP2", "MP3"))

creates the table,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Performance Measure** | | | | |
| **Management Procedure** | SB/SBMSY | Prob(Green) | Prob(SB>limit) | Mean Catch | Catch Variability |
| MP1 | 1.15 (0.78-1.52) | 0.51 | 0.88 | 210.0 (143.8-256.1) | 7.03 |
| MP2 | 1.95 (1.66-2.16) | 0.85 | 0.99 | 152.6 (24.1-203.2) | 23.24 |
| MP3 | 1.70 (1.16-2.05) | 0.66 | 0.94 | 224.5 (25.1-262.2) | 30.64 |

And,

# create Table2 example

createTable2(20, list(OMyft2r108), c("PT41.100.2", "PT41.100.9", "PT41.tune.9"), c("MP1", "MP2", "MP3"))

creates the table,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Status : maximise stock status** |  | **20 year average** | | |
|  |  | **MP1** | **MP2** | **MP3** |
| Mean spawner biomass relative to pristine | SB/SB0 | 0.40 | 0.68 | 0.59 |
| Minimum spawner biomass relative to pristine | SB/SB0 | 0.16 | 0.21 | 0.15 |
| Mean spawner biomass relative to SBMSY | SB/SBMSY | 1.15 | 1.95 | 1.70 |
| Mean fishing mortality relative to FMSY | F/Ftar | 0.62 | 0.23 | 0.56 |
| Mean fishing mortality relative to target | F/FMSY | 0.62 | 0.23 | 0.56 |
| Probability of being in Kobe green quadrant | SB,F | 0.51 | 0.85 | 0.66 |
| Probability of being in Kobe red quadrant | SB,F | 0.17 | 0.03 | 0.16 |
| **Safety : maximise the probability of remaining above low stock status (i.e. minimise risk)** | | | | |
| Probability of spawner biomass being above 20% of SB0 | SB | 0.76 | 0.96 | 0.86 |
| Probability of spawner biomass being above BLim | SB | 0.88 | 0.99 | 0.94 |
| **Yield : maximise catches across regions and gears** | | | | |
| Mean catch (1000 t) | C | 209.99 | 152.59 | 224.46 |
| Mean relative CPUE (aggregate) | C | 0.52 | 0.38 | 0.55 |
| Mean catch relative to MSY | C/MSY | 1.79 | 3.04 | 2.79 |
| **Stability: maximise stability in catches to reduce commercial uncertainty** | | | | |
| Mean absolute proportional change in catch | C | 7.03 | 23.24 | 30.64 |
| % Catch coefficient of variation | C | 0.25 | 0.84 | 0.90 |
| Probability of shutdown | C | 0.00 | 0.43 | 0.37 |

niMSE-IO-BET-YFT can make use of cluster processing to improve throughput. The use of cluster processing is controlled through the UseCluster parameter in the MseDefinition class and the corresponding argument over-ride in the runMse() method. When TRUE a call to runMse() will result in a cluster of processes (by default as many as there are processors on the host machine, otherwise a user-defined limit) being established and used to farm out the required processing. At present runMse() does parallel processing by stock synthesis model so to fully utilize the cluster requires that the number of stock synthesis models is greater than or equal to the number of processors. In future, the code may be re-implemented to better adapt the problem to the cluster and better utilise the available processing power under low numbers of “stock synthesis models” conditions.

When cluster processing is active a monitoring window (Rterm session) opens and displays the process output from the cluster processors. This then allows the real time monitoring of the cluster and will hopefully allow us to quickly identify and terminate problem runs rather than wasting time waiting for completion to see if there was a problem. Be careful not to exit the Rterm window as it cannot be re-established once closed and will close automatically when no longer required.

## C++ Projection Code

In addition to the computational efficiency gains to be expected by using C++ based projection code for the main projection calculations, there were two other perceived benefits: i) a parallel implementation by a second programmer provided an independent check on the integrity of the R-based projection code, and ii) the C++ code was implemented with the standard Baranov catch equations. The main problem with the Baranov equation is that there is no closed form solution for fishing mortality, given a known catch (e.g. when simulating a TAC extraction). An iterative, computationally intensive algorithm is required to reach a solution (to a user-defined degree of precision). In this case, the Baranov equations are solved over the whole 4 quarter period pertaining to the TAC.

Because of computational efficiency limits, the R-based MSE projection code uses an approximation to the catch equation (described in section 5). While there is some debate about whether the Baranov catch equation is the most appropriate to describe the natural and fishing mortality processes (e.g. Liu and Heino 2014), it is the most commonly used approach in fisheries assessments, including the Stock Synthesis models used for recent yellowfin and bigeye assessments and OM conditioning. The phase 1 final report provides a comparison of the two approaches for YFT and BET demonstration case OMs using several MPs with a range of constant catch objectives, from which it is evident that:

* C++ allows the computationally-demanding Baranov equations to be implemented with about the same run time as the R approximation, such that there is no obvious speed advantage to either.
* When fishing mortality rates were low enough that the prescribed quotas could be attained, the numerical results were essentially equivalent and should not make a noticeable difference to the MP selection process.
* The BET results were also equivalent when quotas were too high to be attained. However, the YFT results can potentially deviate substantially when fishing mortality is very high. The problem appears to arise in YFT because of the spatial structure, when exploitation rates in some areas can be extremely high, while other areas remain rather less exploited. In these cases, the deviation between Baranov and the approximation was lowest when the catch was extracted at the beginning of the time-step (rather than the mid-point), and when the proportional harvest rate constraint was activated at 0.3 rather than 0.5 (see section 5). However, this observation is not the result of a systematic analysis, and may not be true in all circumstances.
* The choice between R and C++ may have implications for choosing between some MPs, but we expect that neither approach would be very accurate under the relevant circumstances, because it is likely that fishery behaviour would change if depletion was that high. Presumably those MPs would not be consistent with the Commission objectives.

The C++ based projection code represents the solution that is the most consistent with the SS conditioning (and most fisheries applications), but it comes with a cost in terms of programmer skill requirements. CSIRO developed an adjunct tool, ADT (Automatic Differentiation with Tapenade), that when combined with TAPENADE (*TRansformations et Outils Informatiques Pour le Calcul Scientifique*), allows the development of C++ code that can make use of forward and reverse mode differentials created through code translation (as opposed to operator overloading) and higher order differentials. TAPENADE (<http://www-sop.inria.fr/tropics/tapenade.html>) was developed by INRIA, National Institute for Research in Computer Science and Control, France). We developed the C++ portion of the MSE tool with the understanding that TAPENADE would be freely available to provide high quality differential codes. Part way through the phase 1 project, INRIA changed their licensing arrangement, in a manner that undermined the intended distribution options for the software. As a consequence, we maintained fully functional R and C++ implementations of the MSE projection code, and made it simple to use either interchangeably. As of Sep 2016, INRIA have reverted to their original license agreement, such that TAPENADE annual licenses are again available free of charge for academic use.

The C++ approach has an additional potential advantage in that the EffortCeiling parameter is defined relative to recent fishing mortality and has a useful interpretation in that it. The default setting (20) essentially allows vulnerable age-classes to be fished to almost zero for most fleets. The yellowfin OM problems in 2020-2021 (Kolody and Jumppanen 2021B) suggest that this ceiling could play a role in evaluating model plausibility. e.g. It might be reasonable to assume that fishing effort in the large industrial fleets simply cannot more than double (or some other credible number) in a two year period. Simulations in which the reported catches cannot be attained (i.e. new observations since the data used in conditioning), could flag OMs that are dubiously pessimistic.

### General Coding Details Regarding the Mseom C++ Library

The niMseom library provides functional support to do MSY projections and managed catch / effort projections. It is implemented in C++ and relies upon ADT to create R callable interface code and differentiated versions of the respective objective functions used in minimisation. The latter part requires the TAPENADE tool. Here we give a brief overview of how the library is structured in an ADT based project but for an in depth discussion refer to the users-manual.pdf in the ADT software distribution, which is available from GitHub (https://github.com/pjumppanen/ADT). Note that the niMseom library source code is bundled with the niMSE-IO-BET-YFT project.

ADT based libraries are built by deriving a class from AdtArrays and adding the functionality required. The AdtArrays class is provided by the ADlib library and provides support for multi-dimensional C style arrays with arbitrary indexing and contiguous memory block allocation. Any code that needs to be interfaced with R, or needs to be differentiable, must be coded in a subset of C++ and use only intrinsic data types and the array types provide by ADlib. This is required because ADT parses and translates the source code, and the complete language definition for C++ is hideously complex. The code generally also requires special comment fields that tell ADT about the sizes of arrays and other details regarding the generation of interface code. An ADT make file tells ADT which files to parse and process. The processing involves parsing the code, performing the prescribed operations on the code and producing new methods and possibly interface code added to a class derived from the parsed one.

In this case, the base class is OperatingModelBase and is declared in the niMseom.hpp header file. A subset of the class definition is shown below.

class OperatingModelBase : public AdtArrays

{

protected:

/\* AD\_LIBNAME Mseom \*/

/\* AD\_ALIAS OmB=D\_OperatingModelBase \*/

/\* AUTOINIT \*/

int npop;

int nages;

int nsubyears;

int nareas;

int nfleets;

ARRAY\_1I Recsubyr/\* nsubyears \*/;

/\* AUTODEC \*/

int SpawnPerYr;

ARRAY\_3D EforYear/\* nfleets, nareas, nsubyears \*/;

The comment AD\_LIBNAME tells ADT the name of the library generated in compilation. It is needed so that ADT can generate the correctly named registration function to register R callable C functions with R.

The AD\_ALIAS comment tells ADT the name of the class it will be generating interface code on and its alias. The long name is the actual class name whereas the alias is used to generate functional prefixes for the R callable interface.

The AUTOINIT comment instructs ADT that any class attributes appearing after this comment should be initialised via the constructor. This remains in force until the AUTODEC comment, which instructs ADT to only generate code to instantiate those attributes but not initialise them from R via constructor arguments. Therefore, in the sample shown the attributes npop, nages, nsubyears, nareas, nfleets and Recsubyr will be initialised via constructor arguments whereas SpawnPerYear and EforYear will not.

Note the two array declarations and the corresponding comments. Arrays are multi-dimensional and currently Adlib provides support for up to 10 dimension arrays. The type name for arrays always begins with an ARRAY\_ prefix followed by the number of dimensions followed by a type specifier. I refers to a signed integer type whereas D refers to a double type. The comment appearing after the attribute names the dimensions for the array, the general format of which is,

/\* {from index:}to index \*/

with the from index being optional. When not specified the from index defaults to 1 in deference to the array indexing in R. The EforYear array therefore has the dimensions nfleets, nareas and nsubyears. Note that in a class declaration context all dimension variables must be declared as class attributes in an AUTOINIT context and in a class method context any dimensions not appearing as class attributes must be passed in the method argument list. It should be noted that arrays in R are organised in column major format whereas the C/C++ language is row major. Therefore, all the array dimensions in the C++ code appear in reverse order to that of R to avoid the need to re-map the memory when using R arrays in C++.

With this knowledge on basic aspects of an ADT built library it should be possible for someone familiar with C/C++ programming to be able to understand the niMseom library source code.

### R DLL and Interface Files

The R DLL and interface files consist of; niMseom.dll, Om\_R\_interface.r, and OmB\_R\_interface.r. The two R files provide human friendly R functions that wrap corresponding .External invocations. Both must be included in the R code to use the library via source() commands.

### Loading the Library

The dynamic library should be loaded into R via the dyn.load() command. Furthermore, the code should check for previous loading of the DLL and only load it once. Failure to do so usually results in R crashing, although the exact cause of this behaviour is unclear. The supplied code assumes the R dll resides in the lib sub-folder off the niMSE-IO-BET-YFT parent folder and is illustrated below. It is generalised to support Linux builds as well as Windows, and includes the corresponding R interface code.

# load niMseom module and R interface code

# Load the library

if (version$os == "mingw32")

{

# Running in Windows

LibName <- "niMseom"

LibExtension <- ".dll"

if (version$arch == "i386")

{

LibFolder <- "win32/"

} else

{

LibFolder <- "x64/"

}

} else

{

# Running in Linux

LibName <- "libniMseom"

LibExtension <- ".so"

LibFolder <- "linux/"

}

# Only load library if not already loaded. Loading more than

# once results in R mis-behaving and crashing

if (is.na(match(LibName, attr(getLoadedDLLs(), "names"))))

{

LibPath <- paste("./lib/",

LibFolder,

LibName,

LibExtension,

sep="")

dyn.load(LibPath)

}

source("./lib/OmB\_R\_interface.r")

source("./lib/Om\_R\_interface.r")

### Creating and Destroying an Object Instance

The callable code within the niMseom library comprise a number of methods in a given class. Therefore, to be able to make use of the code we must first create an object instance in R. We do so with, for example,

Obj <- Om.create(npop,

nages,

nsubyears,

nareas,

nfleets,

OM@Recsubyr)

Internally the class constructor creates a number of working arrays needed for the model code whose dimensions are based on npop, nages, nsubyears, nareas and nfleets. When we no longer need the object instance it should be discarded by calling,

Om.destroy(Obj)

### The MSY projection

The niMseom library provides methods to return both the MSY projection and the managed catch / effort forward projection. To ensure consistency it implements the population dynamics in common code shared between both these tasks. The population dynamics implementation is functionally identical to R implementation although in the case of the R code there is duplication between the different tasks.

The C++ implementation of the MSY projection is functionally identical to the R based implementation as testing has demonstrated. The essential difference in returned results is attributable to the different behaviour of the minimizer used. In the case of the R code the optimize() function is used for minimisation whereas the C++ code uses the equivalent of the lbfgsb() C callable minimizer which is used internally by the optim() function in R when using the "L-BFGS-B" method.

The MSY projection works by maximising the yield of a forward projection in time using a scaled effort based on the reference catch years. The projection is taken forward a fixed number of years specified in the call and needs to be sufficient in length to ensure steady state behaviour is reached. With a valid object instance the MSY level is found by calling the Om.nt.findMSYrefs() function as with,

Om.nt.findMSYrefs(Obj,

as.integer(if (Report) 1 else 0),

.Object@ECurrent,

.Object@q,

.Object@R0,

.Object@M[,,1],

.Object@mat[,,1],

.Object@Idist,

.Object@Len\_age[,,1],

.Object@Wt\_age[,,1],

.Object@sel,

.Object@mov,

.Object@h,

.Object@Recdist,

.Object@SRrel,

N,

NBefore,

SSN,

C,

SSBA,

as.integer(length(MseDef@targpop)),

as.integer(MseDef@targpop),

as.integer(nyears),

MinPar,

MSY,

BMSY,

SSBMSY,

SSBMSY\_B0,

as.integer(1000))

Note that some arguments are output arguments (SSN, C, SSBA, MinPar, MSY, BMSY, SSBMSY, SSBMSY\_B0), some are state variables (N, NBefore) and others input parameters (ECurrent, q, R0, M, mat, Idist, Len\_age, Wt\_age, sel, mov, h, Recdist, SRrel). The output arguments and the state variables are owned by R and must have the correct type and size for the code to function as expected.

### The Managed Catch / Effort Forward Projection

The managed catch / effort forward projection is driven by an overall TAC level and fishery specific effort levels (TAE’s). All fisheries with zero effort levels are assumed to be managed under an overall TAC. This overall TAC is broken down to fleet specific TACs calculated on the basis of the reference catch mass case in CMCurrent (plus implementation error). The projection is carried out over a full year with the fleet specific TACs corresponding to annual figures. Forward projection is solved by minimizing,

where the summation is carried out over all fisheries under TAC management. There is no attempt to impose a penalty on extremely high effort in order to obtain a given TAC, other than a maximum effort boundary constraint controlled by the EffortCeiling parameter in the calling R code. This parameter is the maximum allowable annualised effort for any given TAC controlled fishery. We can represent the total instantaneous mortality in mathematical form as,

where *F* is the fishing mortality, *M* is the natural mortality, and *a*, *r* and *t* are the age, region and time indices respectively. The fishing mortality component can be further broken down to be,

where,

Here *mf* is the fishery based multiplier applied to *Ecurrent* that determines the applied effort in each fishery *f* and region *r,* *S* is the selectivity for age *a* in fishery *f*, and is the parameter being solved for by the optimiser. *Ecurrent* is the historical effort by fishery and region in recent years. Hence the effort in any given fishery within any given region and time can never exceed *EffortCeiling* and the effort ceiling applies to every fishery and quarter independently.

The forward projection takes place in the call to the Om.nt.projection() function call, as in,

Om.nt.projection(Obj,

y,

as.integer(if (Report) 1 else 0),

EffortCeiling,

TAC,

TAEbyF,

TACEError,

ECurrent,

CMCurrent,

q,

R0,

M[,,y],

mat[,,y],

Idist,

Len\_age[,,y],

Wt\_age[,,y],

Wt\_age\_mid[,,y],

selTS,

mov,

h,

Recdist,

Recdevs\_Y,

recSpatialDevs,

ssModelData@SRrel,

N\_Y,

NBefore\_Y,

SSN\_Y,

C\_Y,

SSBA\_Y,

as.integer(100))

As with the MSY projection, some of the arguments represent output and state parameters which need to be setup within the calling R session. Furthermore, the Om.nt.projection() function assumes the N and NBefore state variables are initialised to a known state consistent with the point in time it is projecting from. For this reason it is necessary to run a historic projection for the first year to ensure N and NBefore are in the correct known state. This is handled by the call to OmB.nt.runHistoric(). Prior to running either, it is necessary to call OmB.nt.initialiseParameters() to initialise the initial population and to calculate survivorship. It is also necessary to call Om.nt.beginProjection() prior to forward projection. This function call sets up the initial value of the optimisation parameters, which are saved in between annual projections with subsequent year starting point carrying on from the previous year fit. The rationale behind this behaviour is to (hopefully) reduce the number of iterations required to solve the projection step by starting at a point close to the anticipated solution. The assumption is that the effort multiplier series is likely to be highly correlated with previous years.

## Some issues to be aware of:

* R, R packages and Stock Synthesis are all continuously evolving, and newer or older versions than those described here can be expected to cause software instability, e.g. the phase 1 final report discusses problems with Stock Synthesis version 3.24F, which was used in the Langley (2015) assessment; switching to 3.24Y required some model re-parameterization.
* Year-season configuration. Because of a limitation in the way that SS assigns seasonal tag age-classes, at this time it is preferable for the tropical tuna assessment models to define calendar seasons as model years (e.g. Langley 2015). Some functions are provided for converting time units back and forth in seasAsYrToDecYr.f.R, for plotting and importing to the MSE software below. In the OM software, rate processes are defined in annual units (standard IOTC reporting units). Many SS parameters in a calendar-seasons-as-SS-model-years configuration must be re-scaled (e.g. *MOM = 4MSS*, *MSYOM* = 4*MSYSS* ). The conversion is not necessarily intuitive, e.g. *FOM, quarter =1/4 FSS*, because the calendar-seasons-as-SS-model-years configuration includes a fishery duration definition of 0.25, which is internally converted to an "annual" equivalent for reporting, and which must in turn be externally converted back to a true quarterly *F* because the OM operates on a quarterly time-step. Use of a true year-season SS configuration was not explicitly supported in the OM importing software, and will (probably) cause errors if attempted.
* In spatial models, SS reports spatially-aggregated fishing mortality, and not the region-specific fishing mortality, which is required to estimate the local effective effort for effort-based management. In principle, it should be possible to back-calculate fishing mortality by region from the SS numbers-at-age outputs, but initial attempts to do so suggested that the relevant outputs are not intuitively defined. This is important for calculating MSY-based reference points and using effort-based management, but does not affect the quota-based management projections, which have been favoured by IOTC MSE initiatives to date.
* The OM software has flexibility to accept SS inputs with different structural assumptions (e.g. number of spatial units, fisheries, age-classes, etc.). However, these structures must be constant for an individual OM definition, i,e, At this time, different spatial structures or numbers of fisheries, etc. must be defined as different OMs. The MSE software could be run on these different OMs independently, and the results subsequently merged (though the need has not yet arisen to develop such functions).
* The OM software was designed with the expectation that there would be exactly one relative abundance index per region (standardized longline CPUE), and all MPs tested to date assumed that the quarterly-regional CPUE series could be simply combined into a single annual, spatially-aggregated index (with error characteristics calculated on an OM-specific basis to ensure that the MP did not receive unrealistically informative data). These assumptions may need to be revisited following the 2021 yellowfin stock assessment.
* The software was originally intended to support multiple independent spawning populations, with independent biology (i.e. retaining this feature from the Carruthers et al 2014 ABT application). However, this feature remains untested in the projection code, and the code for importing multi-stock OMs would need modification.

# R Classes

MseDefinition

Description

This class forms a container for all the parameters required to uniquely define an MSE framework. It has no explicit initializer. Normal usage involves using the new operator to create a new but empty object instance and then initializing the attributes explicitly. Some defined attributes are not actively used at this time (and may not be completely implemented).

Attributes

Name

A character string naming the operating model definition. This parameter is for future reference purposes only.

Label

A character string labelling the operating model definition. This parameter is used for result presentation purposes.

Date

A character string naming the date of authorship of the operating model definition. This parameter is for future reference purposes only.

Author

A character string naming the person responsible for the model definition preparation. This parameter is for future reference purposes only.

Notes

A character string giving any additional notes pertinent to the model definition. This parameter is for future reference purposes only.

PrimarySource

A character string parameter describing the source of the conditioned model data this definition is based on. This parameter is for future reference purposes only.

CppMethod

A numeric parameter governing whether to use C++ code for model MSY and management forward projection. 0 = use R code, 1 = use C++ code.

UseCluster

A numeric parameter governing whether to use R cluster processing for model MSY and management forward projection. 0 = no cluster processing, 1 = use cluster processing.

npop

A numeric parameter setting the number of populations in the model definition.

nfleets

A numeric parameter setting the number of fleets in the model definition.

SSRootDir

A character string parameter containing the path to the folder containing the Stock Synthesis output files that the model will be using.

SBlim

A numeric parameter setting the reference line value for SSB / SSBMSY plots.

Flim

A numeric parameter setting the reference line value for F / FMSY plots.

OMList

A list of Stock Synthesis models to include in the definition. Typically, this will be the list returned from a call to one of the grid creation functions such as makeGridY3.f(), or a subset of that list.

nsimPerOMFile

A numeric parameter setting the number of simulations to perform per model in the OMList.

proyears

A numeric parameter setting the number of projection years to carry out in the MSE.

targpop

A numeric parameter governing which population/s to collected the MSE performance data for. This can be vector of more than one population (assuming the model has more than one), in which case performance measures are summed over the named populations (not tested).

seed

A numeric parameter used to seed the R random number generator.

recentPerFirst

A numeric parameter setting the first season to use in calculating the seasonal catch effort pattern to use in forward projection. This is a value in sub-years relative to the final sub-year of assessment data and counted backwards. A value of 0 means the final sub-year.

recentPerLast

A numeric parameter setting the last season to use in calculating the seasonal catch effort pattern to use in forward projection. This is a value in sub-years relative to the final sub-year of assessment data and counted backwards. A value of 0 means the final sub-year. Combined with recentPerFirst, it provides a window of sub-years to base an average seasonality on.

seasonCEDist

A numeric parameter governing whether to use seasonal catch effort distribution. 0 = no seasonality (all sub-year catch and effort are equal), 1 = seasonal catch and effort based on the average obtained over the recentPerFirst and recentPerLast sub-years (must be a multiple of whole years).

nsubyears

A numeric parameter setting the number of subyears in the model definition. For quarterly models this will have a value of 4.

lastSeas

A numeric value representing the last season (subyear) of historic assessment data. This allows assessment data to end mid-year.

firstSeas

A numeric value representing the first season (subyear) of historic assessment data. This allows assessment data to start mid-year.

firstSSYr

A numeric value governing the first year of stock synthesis results to include in the MSE. This value is in *Stock Synthesis* years, which represent quarters in the context of the SS assessment of YFT and BET.

firstCalendarYr

A numeric value representing the first calendar year of historic assessment data.

lastCalendarYr

A numeric value representing the first calendar year of historic assessment data.

firstMPYr

A numeric value representing the calendar year in which MP management begins. The supplied management procedures will be utilised from this year onward.

MPDataLag

A numeric value setting the delay in years between the current year of operation and the last set of assessment data supplied to the management procedure. This parameter accounts for the fact that data collation takes time and assessment results always typically lag the year in which management decisions are to be implemented.

catchBridge

A numeric vector indicating the known (annualised) catch history between the final year of assessment and the beginning of management under MPs. The vector is of length between 0 and firstMPYr-lastCalendarYr-1 . Years not covered by the catch bridge vector are considered unknown and are assigned catch based on the previous catch, combined with a lognormal random deviate with CV specified by catchBridgeCV.

catchBridgeCV

A numeric value setting the CV for the lognormal random deviate used to arrive at the catch for unknown years in the catch bridge. Unknown catch is arrived at by applying the deviate to the last known year up to but not including the year that the management projection starts.

indexFisheries

A numeric vector of the fishery numbers from which (fishery-selected) CPUE-based relative abundance indices are to be calculated. In the initial application, there must be exactly one index per region, and these are combined into an aggregate index for the MP calculations.

ReccvTin

A numeric vector, one entry per population, representing the temporal CV for recruitment aggregated over regions. If a value is negative then the recruitment CV is calculated from the RMSE of the SS outputs.

ReccvRin

A numeric value setting the CV for the lognormal random deviate used to provide variability in recruitment among areas.

RecACTin

A numeric value setting the recruitment temporal auto-correlation for the recruitment process (applied annually, even though recruitment occurs quarterly).

NInitCV

A numeric value setting the CV for the lognormal random deviate applied to the initial age 1 population numbers. The other ages have this CV reduced by an exponential decay factor, NInitCVdecay.

NInitCVdecay

A numeric value governing the relationship between the initial population CV at age 1 (NInitCV) and the corresponding value at age a. The relationship is an exponential one described by exp(NinitCV \*(a-1)).

selExpRange

A numeric value governing the exponent range of temporal variability in selectivity. The exponent variation is parameterised as a sinusoid with angular frequency governed by selWLRange and amplitude selExpRange. The angular frequency is randomly chosen from a uniform distribution whose range is selWLRange.

selAgeRange

A numeric value governing the age range shift of selectivity as a function of time. The age shift in selectivity is parameterised as a sinusoid with angular frequency governed by selWLRange and age shift amplitude selAgeRange. The angular frequency is randomly chosen from a uniform distribution whose range is selWLRange.

selWLRange

A pair of numeric values governing the range of angular frequencies used to parameterise temporal selectivity change. The angular frequency is randomly chosen from a uniform distribution whose range is selWLRange. The angular frequency is related to the period of variation in years by the relationship,

Where is the angular frequency, and *T* is the wave period in years (either time or ages depending on context of use).

TACEcv

A numeric vector of length nfleets specifying the CV for a lognormal random deviate applied to fleet specific TAC and TAE.

Ccv

A numeric vector of length 2 specifying the range of CV for a lognormal random deviate to be applied to fleet specific catch as observation error. The actual CV for each fleet is determined by sampling a uniform distribution whose range is specified by Ccv.

Icv

A numeric vector of length 2 specifying the range of CV for a lognormal random deviate to be applied to fleet specific relative abundance indices. The actual CV for each fleet is determined by sampling a uniform distribution whose range is specified by Icv.

IACin

A numeric value setting the autocorrelation in relative abundance lognormal error deviates.

Cbcv

A numeric value specifying the catch bias CV. The fleet specific catch bias is obtained by sampling a lognormal distribution with mean 1, whose CV is Cbcv.

nCALobs

A numeric vector of length 2 specifying the range of the number of annual catch at length (CAL) observations. The actual number of observations is determined by sampling a uniform distribution whose range is specified by nCALobs.

Ibeta

A numeric vector of length 2 specifying the log range of the hyper-stability parameter. The actual value of the hyper-stability parameter is determined by sampling a uniform distribution whose range is specified by the log of nCAAobs and then taking the inverse log (not tested to date).

ITrendin

A numeric value controlling trending in abundance index estimates. A negative value means the trend is extracted from the Stock Synthesis assessment model filename (file-specific index trends assume the single digit following a single q in the filename defines the trend). A positive value is the multiplier applied to a 1% per-annum compounded trend. For example if we use 3 for ITrendin then the trend in abundance indices is 3% per annum.

ImplErrBias

If specified, ImplErrBias is a vector proyears in length and represents a multiplier on catch or effort at a specific year in the projection. For positive values it represents a multiplier on TAC. For negative values it fixes effort to a scaled level based on the fishing effort prior to fixed effort coming into force. That is, if *F* is the effort prior to *ImplErrBias* becoming negative the effort for the negative *ImplErrBias* is

RecScale

A numeric value or numeric vector of length proyears that scales recruitment. For a single value the projected years of recruitment are multiplied by RecScale. For a vector the recruitment is scaled by the RecScale value for the corresponding year. RecScale defaults to 1.

modelWeight

If specified, modelWeight is a vector of length equal to the length of OMList. Values in this vector represent the weight applied to a particular model in OMList. modelWeight in combination with totalSims determines the number of simulations to apply to a given stock synthesis model during the projection phase using multinomial selection. This is illustrated by the R code responsible (in simplified form) shown below.

nsimPerOMFile<-t(rmultinom(1,

size=totalSims-1,

prob=modelWeight))))

As can be seen, the use of modelWeight and totalSims results in the nsimPerOmFile vector being re-assigned based on the desired weighting and total simulations. If modelWeight is unspecified then the number of simulations for a given model is specified by the original nsimPerOmFile vector values.

totalSims

This attribute along with modelWeight is used to determine the number of simulations for each stock synthesis model (nsimPerOMFile). If modelWeight is not specified then this attribute is ignored. See nsimPerOMFile for more information.

Methods

createMseFramework(.Object, Report=FALSE, UseCluster=NA, UseMSYss=0)

This method creates a new instance of the MseFramework class based upon the specification in .Object. Call this method to create a properly initialized MseFramework object.

# Operating Model Equations

## Notation

We have attempted to maintain consistency with the Carruthers et al. (2014) Atlantic Bluefin MSE naming conventions and presentation style, to facilitate comparison for users involved with both IOTC and ICCAT.

States, parameters and subscripts are summarized in Table 4. For readability, we often omit subscripts where the context should be self-evident (e.g. maturity is not scripted by year, season and region, because it is usually assumed to be invariant, though that could change in future versions). Subscripts denoting multiple stocks have also been omitted as they are not relevant for any of the demonstration cases to date. When seasonality is not explicitly important, we usually substitute the quarterly time subscript *t* to represent the combination of year (*y*) and season (*s*):

.

There is also some redundancy in the specification of fisheries and regions. Because each fishery operates in exactly one region, the subscript *f* implies a unique region *r* (though the reverse is not true, as multiple fisheries can operate in the same region).

We recycle the greek characters *τ* and *ω* to indicate auto-correlated and independent random normal deviates, respectively, and *σ* for variance-related parameters:

1)

2) ,

where *ρ* is the lag(1) correlation co-efficient (seasonal or annual). The OM was only set up for some processes to use auto-correlated errors (and *ρ* = 0 can be specified to remove it).

Table 4. Operating model states, parameters, scripts and superscripts used in this document (but not necessarily in the code).

|  |  |  |
| --- | --- | --- |
| Variables / Parameter | | |
| *N* | *Number in population* |
| *SB* | *Spawning Biomass* |
| *C* | *Catch* |
| *M* | *Natural mortality* |
| *M* | *Maturity (proportion)* |
| *E* | *Fishing effort* |
| *S* | *Fishery selectivity* |
| *F* | *Fishing mortality* |
| *h* | *Beverton-Holt stock-recruit steepness* |
| *P, ϕ* | *Proportion of a distribution* |
| *Τ* | *A lag(1) auto-correlated random normal deviate* |
| *Ω* | *An independent random normal deviate* |
| *Σ* | *Variance-related parameter* |
| *W* | *Mass-at-age* |
| *L* | *Length-at-age* |
| *α, β* | *Mass-length parameters* |
| *Ψ* | *Movement probability* |
| *TAC* | *Related to quota-based management* |
| *TAE* | *Related to effort-based management* |
| *R* | *Recruitment* |
| *R0* | *Virgin Recruitment* |
| *ρ* | *Auto-correlation co-efficient* |
| Subscripts / Superscripts (capitals denote the total/final, i.e. *A* = oldest age class) | | |
| *a, A* | *Age-class (quarters)* |
| *l,L* | *length-class* |
| *y, Y* | *Year* |
| *s, S* | *Season (quarters)* |
| *t, T* | *Time (simplified representation of combined year and season)* |
| *r, R* | *Region* |
| *TAC* | *Related to quota-based management* |
| *TAE* | *Related to effort-based management* |
| *beforeX* | *Before event X* |
| *afterX* | *After event X* |
| *rec* | *Recruitment-related* |
| *recent* | *Relates to a user-defined recent period* |
| *imp* | *Relates to implementation error* |
| *init* | *Relates to initial conditions* |
| *obs* | *Relates to observation error* |
| *SS* | *Relates to Stock Synthesis state/parameter estimates* |
| *vul* | *Vulnerable (fishery-selected)* |
| *mass, numbers* | *Catch units* |

## Population Dynamics

The following equations describe the dynamics for an individual stochastic realization, conditional on a specific SS scenario within an ensemble of models.

The model follows fairly standard fisheries assumptions for the most part: age-structured with years and quarterly seasons, spatially-structured (optional) with multiple fisheries. The option of describing multiple stocks (independent biology including spawning) has been retained from the ABT MSE for potential future implementation, but this option is not fully tested or supported in the Indian Ocean OM conditioning process at this time. The Indian Ocean MSE has simpler dynamics than the ABT application, in that each fishery is assigned to exactly one area and is assumed not to move among areas. Accordingly, there is no attempt to model the fishery effort distribution as a function of the fish population. However, the Indian Ocean model is more complicated in the sense that there is potentially simultaneous catch-based and effort-based management. It is up to each MP to define which fisheries are managed by a catch quota, and which fisheries are managed by effort multipliers. The default assumption is that the annual aggregate catch quota is allocated among the quota-managed fisheries in proportion to the recent observed distribution (defined in the OMs). The fishery-specific quotas are in turn extracted quarterly, in proportion to the recently observed quarterly catch distributions. The phase 3 OM can simulate IUU fishing (with an arbitrary mix of reporting and non-reporting) and effort-creep scenarios, and these are the basis of some of the standard robustness tests.

The initial numbers-at-age, *N*, of the population to be projected forward are extracted from the last year of the SS outputs, with some additional user-defined error:

3) .

The parameter *σinit* controls the magnitude of the error at age 2 (the youngest age imported from SS), and *d* describes how fast the error declines with age estimates[[1]](#footnote-1).

Recruitment, *Rt* (to age1, corresponding to age 0 in SS notation), is calculated for each year and season, assuming a spatially-aggregated spawning population (at the beginning of the time-step) according to a Beverton-Holt stock-recruitment relationship with auto-correlated log-normal recruitment deviates:

4) .

The spawning stock biomass, *SB*, is the aggregate across all regions at the beginning of the time-step:

5) 

where *m* is the maturity schedule and is the mass-at-age at the beginning of the time-step, related to length with the standard relationship:

6) .

Note that SS distinguishes between size-at-age at the beginning of the time-step (for spawning purposes), and at the mid-point of the time-step (for catch calculations). This is an unnecessary complication for populations which are assumed to have continuous spawning, however, it was added to minimize inconsistencies between the conditioning and projection code. The aggregate recruitment is partitioned among regions according to a stationary distribution:

7) 

Age-specific, season-specific movement occurs instantaneously, after recruitment, but before mortality:

8) 

where *ψ* is the probability of an individual of age *a* moving from region *r*, to region *k* in season *s*. (there was no spatial structure or migration in the BET demonstration case, while the YFT demonstration case had constant migration among seasons).

There are two options for the catch equations. The C++ based projection sub-routine uses the standard Baranov equations (9, 10) to apply continuous natural and fishing mortality (*M* and *F*):

9) ,

and

10) .

An iterative algorithm is required to simultaneously solve for the catch of the effort-managed fleets and the fishing mortality of the quota-managed fleets.

The R-based approach is a hybrid of Baranov and something similar to Pope's approximation, which does not require an iterative solution (described in equations 11-13). Natural mortality and effort-based fishery extraction are first applied up to a point Δ within the time-step (0 ≤ Δ ≤ 1; 0.5 by default):

11) 

where *f* ϵ *TAE* refers to all effort-managed fisheries. This is followed by instantaneous catch extraction for the quota-managed fisheries:

12) 

where *f* ϵ *TAC* refers to all fisheries that are managed by catch quotas. Finally there is a second continuous mortality event covering the remainder of the time-step:

13) .

Total catch for this latter approach is the sum of the TAC extraction (from 12) and the catch corresponding to each of 11 and 13 which is equivalent to the closed form of equation 9 (i.e. in 11 and 13, *F* is a known function of effort for the effort-managed fleets). Note that the dynamics of equations 9-10 and 11-13 are identical if all fisheries are effort-managed.

In equations 9, 11, and 13, fishing mortality is a function of purely age-based selectivity and effective effort, i.e.

14) 

where effort is known for the effort-managed fleets. Each fishery is managed by either quota or effort. The MP is required to provide a non-zero TAE (in the form of an effective effort multiplier) for each effort-managed fishery, and an aggregate TAC for the remaining fisheries. The effective effort multiplier assumes that the fishing mortality will increase or decrease relative to a recently observed level *Frecent*, (defined in the main text for the demonstration cases). This implicitly assumes that fishery catchability is stationary relative to the effort units defined in the MP (note that in recent YFT and BET assessments, this is only assumed to be true for standardized longline effort).

Selectivity for each fishery is represented by a vector of length *A* scaled to a maximum of 1.0 for the most vulnerable age(s). There are two options for introducing temporal variability to the selectivity in projections. The first shifts the baseline selectivity function (imported from SS) toward younger or older ages by an integer amount, , as a function of time:

15) 

16) 

Where and  fishery-specific amplitude and angular frequency parameters respectively, and *y* is the projection year starting from 1 (i.e. *y* =1 sets the phase angle so that selectivity is unchanged in the first projection year). An analogous option increases or decreases the degree of targeting in the baseline selectivity:

17) 

18) 

A value of >1 disproportionately increases (<1 decreases) the relative vulnerability of the more highly selected age-classes relative to the other age-classes. Separate and values are used for each type of selectivity process error. The amplitude  for the temporal variability is user-defined and identical among fisheries, but the sign of the amplitude is random for each fishery (i.e the initial trend can be in either direction). The angular frequency is a fishery-specific random sample from a user-defined uniform distribution. The demonstration case parameters in operating models OMyft2 and OMbet1 were selected such that:

* The age-shift amplitude, = 1 results in a maximum selectivity shift of +/- 1 (quarterly) age-class
* The selectivity targeting amplitude exponent, = 0.6, was arbitrarily selected (i.e. from a subjective judgement that the value is big enough to have an effect, but not too big to render the fishery unidentifiable).
* The uniform distribution for the angular frequency parameters was [0.0625,0.5], such that the lower bound results in a monotonic directional change in the selectivity for a projection period of 26 years or less, while the upper bound results in 2 full oscillations of the selectivity vector within a 26 year projection period.

Figure 3 illustrates the temporal variability in selectivity observed for a purse seine and longline fishery from OMbet1. Note that temporal variability in selectivity was added primarily as a means of ensuring that the information content of unbiased size composition data was not unrealistically informative, however, it also introduces time series error structure to the CPUE series. The selectivity process error has not been used in recent iterations of MP evaluations. If any new MPs attempt to use sie/age composition data to extract more information about the stock, the stationary selectivity assumptions should be revisited.

The aggregate TAC is distributed among seasons and TAC-managed fisheries according to the "recent" distribution of effort or catch. In the case of the C++ based Baranov equations (9, 10), this is achieved by removing the total annual TAC assuming the recent effort distribution remains constant:

19)  ,

while *Ff ϵ TAE* is provided by the MP. The function minimizer solves for the effort multiplier, *x*, that satisfies the catch equations for all 4 quarters simultaneously.

In contrast, the R-based Baranov approximation (equation 12) has a closed form solution. For those fisheries that are managed as part of the quota system, it is assumed that the TAC is apportioned according to the recent (user-defined) catch-in-mass distribution by fishery and season:

20) .

The age-specific, region-specific, catch extraction is calculated each season:

21) ,

where *Bvul*, represents the relative biomass that is vulnerable to each fishery via the selectivity function, *S*:

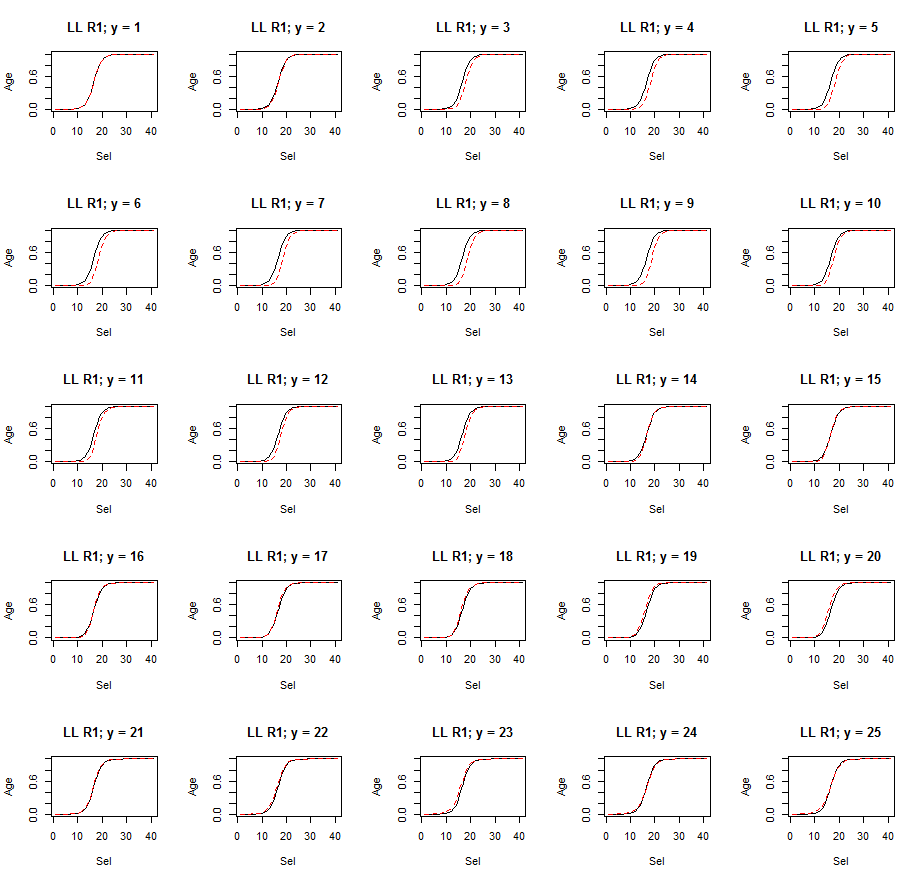
22) ,

The catch in numbers extracted from equation 12 is then:

23) .

Lastly, the age-classes and time-steps of both approaches are incremented from the end of each time-step to the beginning of the next time-step in the standard way (including a plus-group accumulator, in which the characteristics of all fish ≥ age *A* are assumed to be identical):

24) .



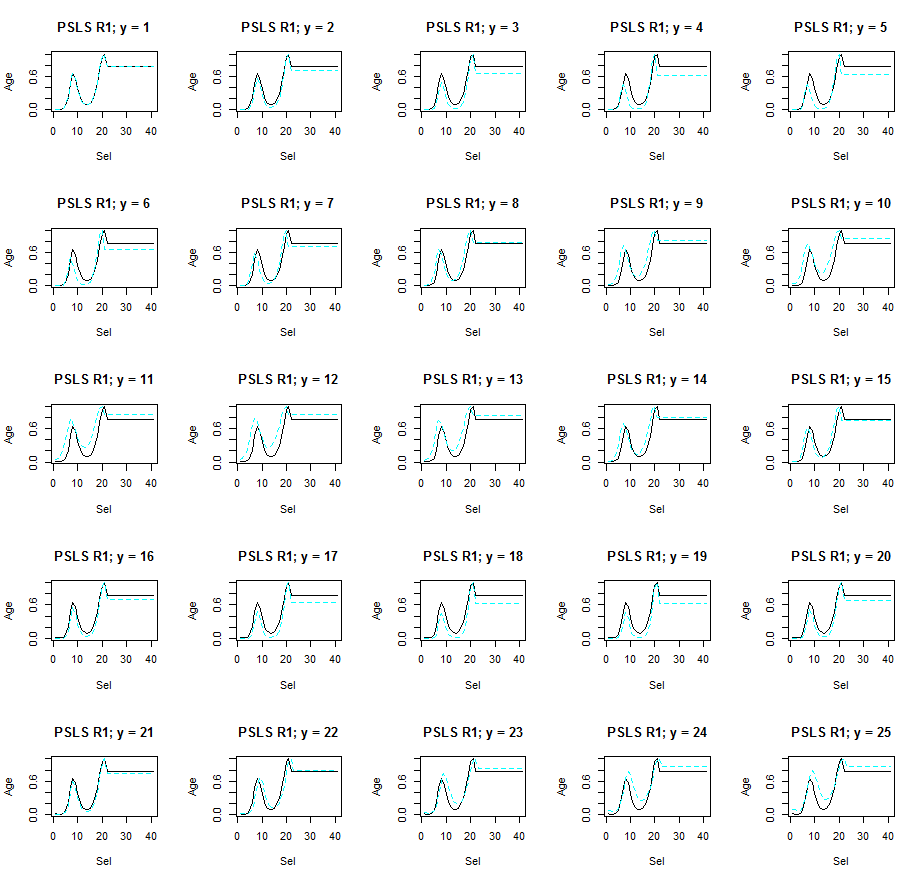


Figure 3. Examples of annual changes in longline selectivity (top panel) and purse seine associated selectivity (bottom panel) over a 25 year projection period for OMbet1. Black line is the stationary selectivity estimated in the SS model, coloured line is the temporally variable selectivity.

## Management Implementation Errors

There is independent implementation error on each fishery-specific quota or effort multiplier (applied equally to all seasons within years):

25) 

26) 

To prevent numerical problems when harvest rates are very high, there is an additional constraint imposed on the approximate catch extraction of equation 12, manifested as a modification to the normal harvest rate:

27) 

28) 

This constraint is required to prevent the catch equation approximation from extracting more than 100% of the population. Harvest rates > *Ulim* continue to increase monotonically toward an asymptote of 1-. It is under these high harvest rate situations, when the TAC cannot be attained, that the projections from the catch approximation can deviate appreciably from the Baranov equations. The final report was completed with values (*Ulim* = 0.5, = 0.12, and TACTime = 0.5), while it appears that the approximation yields results closer the Baranov equation with (*Ulim* = 0.3, = 0.3, and TACTime = 0.01).

## Observation Errors

Currently, the MPs only have access to a subset of relatively aggregated data. Further disaggregation (e.g. all or different data by season and fishery) may eventually prove desirable, but providing these data to the MP without proper consideration of the error structures risks providing unrealistically informative data. e.g. If a relatively high CPUE observation error CV = 0.6 is applied independently each quarter to each of 4 fisheries, the aggregate annual CV ~ 0.15, which is probably unrealistically optimistic for commercial CPUE.

Annual observed catch-in-mass is reported as the product of catch-in-numbers and mass-at-age, summed over ages, seasons and fisheries (fisheries and regions are redundant), with an aggregate log-normal error:

29) 

Observed catch-at-length frequency distributions, *X*, are provided as a multinomial sample from the true catch distributions:

30) ,

where *P* represents the true proportion of fish caught in length class *l*. The sample is unbiased, however, the sample size, *n*, should be set much lower than most real catch-at-length sampling programmes to recognize that samples are not truly random. Furthermore, temporally-structured selectivity can be used to ensure that size data are not unrealistically informative.

An annual aggregate relative abundance index is calculated:

31)

where *rf* ϵ *LL*, indicates the region or fishery (as appropriate) corresponding to the (informative) longline fisheries, *S* is the fishery selectivity, and *q* is the constant of proportionality calculated from the whole of the conditioning assessment model CPUE time period. The CPUE observation consists of:

32)

where βis a hyper-stability / hyper-depletion parameter, and δ is a temporal catchability trend, and the stochastic error and auto-correlation are applied on an annual basis.

# Candidate Management Procedures

The phase 1 final report includes a qualitative description (cartoon schematic) of the IT and PT MPs described below. The parameter values from the main report refer to the MP equations defined in Table 5. In phase 3, new MPs have been explored, based on random effects models implemented in TMB (Kolody & Jumppanen 2020, 2021A).

For the PT MPs, the Pella-Tomlinson model dynamics consist of:

33) ,

where *B* is aggregate biomass, *C* is the total catch in mass, *r* and *K* are the estimated population growth rate and carrying capacity, and *p* is the equilibrium yield "shape" parameter (*p = BMSY/B0*, fixed at 0.33 in the demonstration MPs). The model is fit using the least-squares (observation error) objective function:

34) 

where *I* is the aggregate annual CPUE index, and

35) 

The HCR requires estimates of *BY/B0* and

36) .

Using the R-based fitting algorithm, the model appeared to fit some complicated dynamics very poorly (i.e. presumably due to the inability to describe recruitment variability), but the MP generally provided respectable management performance. We did not explore the frequency with which the base R fitting algorithm, optim(), was actually finding the global minimum.

Table 5. MPs included with the initial IOTC MSE release. Additional control parameters are omitted for the IT and PT MPs, including: 1) MP start year, ii) frequency of MP application, 3) data lag (assumed 0 below).

|  |
| --- |
| CCE (constant catch and/or effort) |
| **TAC/TAE calculation:**  *TAEf =*  *TAC = kTAC*  **Control parameters:**  = constant fishery-specific effort-multiplier; constant for all years,  *kTAC* = aggregate *TAC* for fisheries with  = 0; constant for all years. |
| IT (aim for CPUE target) |
| **TAC/TAE calculation:**  *TAEf* = 0      **Data/analysis:**  *Iy* = stanardized longline CPUE in year *y* (aggregated over areas and seasons)  (weighted mean CPUE of last 3 years to reduce volatility)  *m* = CPUE trend (linear regression slope from *Iy-4* to *Iy*)  **Control parameters:**  *IT*= target CPUE  *k1...k4* = responsiveness (gain) parameters  Δmin, Δmax = maximum change constraints |
| PT (Pella Tomlinson with 40:10 type HCR) |
| **TAC/TAE calculation:**  *TAEf* = 0      **Data/analysis:**  *DPT* = Current depletion estimated from fitting Pella-Tomlinson (Observation error) model to catch and CPUE  *MSYPT* = MSY estimate from production model  **Control parameters:**  *k1...k3* = HCR function modifiers  Δmin, Δmax = maximum change constraints |

# Performance Measures and Reference Points

The software reports a standard set of management performance indicators as proposed by the IOTC Working Party on Methods, endorsed by the Scientific Committee (SC 2015), and subsequently refined by the WPM informal sub-group on MSE. Example plots are shown in the phase 1 report.

The IOTC WPM informal MSE working group (April 2016) proposes:

*Time series will be calculated over projection windows of 1, 5, 10 and 20 years, where year 1 is the first year that a TAC or TAE is applied (i.e. as opposed to the first year of projections which might be based on a known or assumed catch because of data and decision process time lags). While it is recognized that MSE is intended to look at medium to longer term performance, one year is included because it is inevitable that industry stakeholders will want to know what the implications of adopting an MP will be in the first year (and they may be particularly disruptive in a rebuilding situation).*

*The performance indicators described in Table [A1] are calculated for each stochastic realization, and then presented as percentiles (10, 25, 50, 75 and 90%) from the distribution of all realizations*

*It was noted that there is currently an inconsistency in the [Commission] identified management objectives, in that achieving target reference points of FMSY and BMSY with near perfect precision would correspond to roughly equal probabilities of being in the green and red Kobe quadrants, while a high probability of being in the green quadrant implies F < FMSY and B > BMSY.*

It has become standard to include 15 year summary statistics in TCMP reports, because this better represents the stock dynamics up to and including the period used in the tuning objectives, and downplays the importance of MP behaviour beyond the tuning objective window. Standard graphical outputs are described below (and illustrated elsewhere in the user manual and phase 1 project report):

*Time series (quantiles plus some individuals iterations) plots will be used to describe key MSE outputs, including SB/SBMSY (SB/SB0, B/BMSY, B/B0), Catch, F/FMSY, Recruitment and/or CPUE:*

* *Interim reference point reference lines should be included (green target, red limit) when appropriate.*
* *Plots are to indicate the median with a line, and the 25-75th and 10-90th percentiles with shaded ribbons.*
* *When appropriate, 3 individual realizations should be plotted on top, corresponding to the 25, 50, 75th percentiles of SB/SBMSY (or SB/SB0) over the 20 year projection period. These same three individual realizations should then be plotted in all relevant time series plots (i.e. irrespective of which percentile the realization corresponds to in the other plots)*

*Four core trade-off plots, computed for each of 10 and 20 years of projection (i.e. year 1 = first TAC/TAE implementation)*

*1. SB/SBMSY (or SB/SB0 for skipjack) vs. Yield*

*2. Pr(Green Kobe) vs. Yield*

*3. Pr(SB > BLim) vs. Yield*

*4. mean(1 – Cy/Cy-1) vs. Yield*

*Confidence interval plots (double whisker aka udon-soba plots). The summary statistics from [Table xxx] will be summarized graphically by their median, thick confidence interval whiskers for the 25-75th percentiles, and thin whiskers indicating 10-90th percentiles. These plots can compare several MPs for a single performance statistic within a single panel and can pack a lot of information into a small space, but they are less convenient for identifying broad patterns than the other plot types.*

MSY-based reference points are calculated assuming the same "recent" effort distribution (by fishery and season) used for TAC partitioning.

Annual aggregate fishing mortality for reporting purposes is calculated using the SS approach, in which *Z* is first calculated on a spatially aggregated basis, and *F = Z - M* :

37) 

where *s* consists of the 4 seasons within year *y* (and strictly speaking the numerator in the final element of the summation above would be *Ny+1,s=1,a+1,r*).

Table 6. Performance measures (the interim target and limit reference points from Resolution 15/10 are shown for yellowfin and bigeye only; they differ for other IOTC species; Flimit = 1.4FMSY, Ftarget=FMSY, SBlimit=0.4SBMSY, SBtarget=SBMSY)

|  |  |  |
| --- | --- | --- |
| **Status : maximize stock status** | | |
| 1. Mean spawner biomass relative to pristine | *SB*/*SB*0 | Arithmetic mean over years |
| 2. Minimum spawner biomass relative to pristine | *SB*/*SB*0 | Minimum over years |
| 3. Mean spawner biomass relative to *SBMSY* | *SB/SBMSY* | Arithmetic mean over years |
| 4. Mean fishing mortality relative to target | *F*/*Ftar* | Arithmetic mean over years |
| 5. Mean fishing mortality relative to *Fmsy* | *F*/*FMSY* | Arithmetic mean over years |
| 6. Probability of being in Kobe green quadrant | *SB*,*F* | Proportion of years that *SB*≥*SBtar*and *F*≤*Ftar* |
| 7. Probability of being in Kobe red quadrant | *SB*,*F* | Proportion of years that *SB*<S*Btar* and *F*>*Ftar* |
| **Safety : maximize the probability of remaining above low stock status (i.e. minimize risk)** | | |
| 8. Probability of spawner biomass being above 20% of *SB*0 | *SB* | Proportion of years that *SB*>0.2*B0* |
| 9. Probability of spawner biomass being above *B*Lim = 0.4*SBMSY* | *SB* | Proportion of years that *SB*>0.4*SBMSY* |
| **Yield : maximize catches across regions and gears** |  |  |
| 10. Mean catch | *C* | Arithmetic mean over years |
| 11. Mean catch by region and/or gear | *C* | Arithmetic mean over years |
| 12. Mean catch relative to MSY | *C/MSY* | Arithmetic mean over years |
| **Abundance: maximize catch rates to enhance fishery profitability** | | |
| 13. Mean catch rates by region and gear  (for fisheries with meaningful catch-effort relationship) | *I* | Arithmetic mean over years |
| **Stability: maximize stability in catches to reduce commercial uncertainty** | | |
| 14. Mean absolute proportional change in catch | *C* | Arithmetic mean over years of abs(1-*Ct*/*Ct*−1) |
| 15. Variance in catch | *C* | Variance over years |
| 16. Probability of shutdown | *C* | Proportion of years that *C*< 0.1*MSY* |

# Acknowledgements

This work was funded by:

* phase 1 - European Union (through IOC) and CSIRO (administered by IOTC/FAO)
* phase 2 – The Global Environment Facility – Common Oceans Areas Beyond National Jurisdiction project and CSIRO (administered by IOTC/FAO)
* phase 3 – Australian Department of Foreign Affairs and Trade and CSIRO (administered by IOTC/FAO)

Funding agencies do not necessarily agree with the views expressed by the authors.. The R-based MSE code was adapted from the Atlantic Bluefin MSE work developed by Tom Carruthers (and funded by the ICCAT GBYP project). Technical oversight and advice was provided by the IOTC Working Party on Methods, notably Iago Mosqueira, Toshihide Kitakado, Nokome Bentley, Gerald Scott, Gorka Merino, Hilario Murua and Rishi Sharma. Adam Langley and Ian Taylor provided helpful insight on the inner workings of Stock Synthesis. Some of the summary result graphics were adopted from the Fisheries Library in R (FLR) code provided by Iago Mosqueira.

# References

Carruthers, T, Kell, L, Davies, C. 2014. Evaluating management strategies for Atlantic bluefin tuna. Report 1 – contract report to support to BFT assessment (GBYP 02/2014) of the Atlantic-wide research programme on bluefin tuna (ICCAT GBYP– Phase 4).

Kolody & Jumppanen 2019 – Indian Ocean Bigeye Tuna MSE Update March 2019. IOTC–2019–WPM10-INF02

Kolody, D, Jumppanen, P. 2020. [A candidate Management Procedure based on a Joint Process and Observation Error Random Effects Production Model](https://www.iotc.org/documents/WPM/11/13). IOTC–2020–WPM11–13.

Kolody, D, Jumppanen, P. 2021A. IOTC Bigeye Tuna Management Procedure Evaluation Update June 2021. IOTC-2021-TCMP04-08.

Kolody, D, Jumppanen, P. 2021B. IOTC Yellowfin Tuna Management Procedure Evaluation Update June 2021. IOTC-2021-TCMP04-09.Liu, X, Heino, M. 2014. Overlooked biological and economic implications of within-season fishery dynamics. Can. J. Fish. Aquat. Sci. **71**: 181–188 (2014) dx.doi.org/10.1139/cjfas-2013-0029.

Matsumoto, T, Yokoi, H, Satoh, K, Kitakado, T. 2018. Diagnoses for stock synthesis model on yellowfin tuna in the Indian Ocean. IOTC–2018–WPTT20–42\_Rev1. Methot, R.D., Wetzel, C.R., 2013. Stock synthesis: A biological and statistical framework for fish stock assessment and fishery management. Fish. Res. 142: 86– 99.

R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.

1. the estimated precision of partially recruited cohorts is usually artificially low because i) most tuna assessments assume stationary selectivity, and ii) recruitment deviations for recent cohorts are often set to zero to avoid dubious estimates (which can be very influential to medium term dynamics). [↑](#footnote-ref-1)